

# **L'Assembler 8086**

## **Concetti Generali**

M. Rebaudengo - M. Sonza Reorda

Politecnico di Torino  
Dip. di Automatica e Informatica

1

M. Rebaudengo, M. Sonza Reorda

## **Sommario**

- **Introduzione**
- **Pseudo-Istruzioni**
- **Operatori**
- **Modi di Indirizzamento**
- **Istruzioni**

2

M. Rebaudengo, M. Sonza Reorda

# Introduzione

I programmi Assembler sono composti di:

- *istruzioni*: generano un'istruzione macchina
- *direttive* o *pseudo-istruzioni*: sono comandi per l'assemblatore.

## Esempio

Istruzione:	ADD	AX, 5
Direttiva:	VAR1	DB ?

3

M. Rebaudengo, M. Sonza Reorda

# Formato delle Istruzioni nel Codice Sorgente

```
label: mnemonico operando, operando ;commento
```

dove

label:        identifica l'indirizzo di partenza di una istruzione

operando:    in numero variabile da 0 a 2

mnemonico:   identifica una istruzione

commento:    qualsiasi sequenza di caratteri terminata da un fine-riga

## Esempio

```
lab1: mov ax, 5 ; carico ax
```

4

M. Rebaudengo, M. Sonza Reorda

## Formato delle Istruzioni nel Codice Oggetto

Ogni istruzione a livello di codice oggetto corrisponde ad una sequenza di byte in numero variabile tra 1 e 6.

I bit che compongono una istruzione possono essere suddivisi in due gruppi:

- il *codice operativo*, che specifica l'operazione che deve essere svolta
- gli *operandi*, in numero variabile tra 0 e 2; il modo in cui si specifica ove risiede ciascun operando è noto come *modo di indirizzamento*.

5

M. Rebaudengo, M. Sonza Reorda

## Variabili

Sono utilizzate per identificare in modo simbolico una zona di memoria contenente dati.

All'atto della definizione della variabile, si definiscono

- il nome
- la dimensione
- il tipo di dato contenuto (eventualmente)
- il valore di inizializzazione (eventualmente).

6

M. Rebaudengo, M. Sonza Reorda

## Identificatori

Sono i nomi che possono essere assegnati a istruzioni, variabili, procedure, costanti, segmenti.

Sono così composti:

- il primo carattere può essere una lettera (a-z, A-Z), oppure uno dei 4 caratteri @ \_ \$ ?
- gli altri caratteri possono essere una lettera, un numero, o uno dei 4 caratteri sopra.

Le *parole chiave* non possono fungere da identificatori.

Nel MASM 6.0 la lunghezza massima di un identificatore è 247.

## Costanti

- binarie: 001101B
- ottali: 15O, 15Q
- esadecimale: 72H, 0DH, 0BEACH (devono iniziare con un numero)
- decimali: 13, 13D
- stringhe: 'S', 'Ciao'
- reali in base 10: 2.345925, 715E-3

## MASM 6.xx

- Non è case sensitive
- Non è sensibile agli spazi
- Ogni istruzione deve occupare una riga: altrimenti la riga precedente deve terminare con \.

9

M. Rebaudengo, M. Sonza Reorda

## Pseudo-Istruzioni

Sono direttive per l'Assemblatore, che non corrispondono a istruzioni macchina nel codice generato.

Le categorie principali di pseudo-istruzioni sono:

- pseudo-istruzioni per la definizione di variabili
- pseudo-istruzioni per la definizione di costanti
- pseudo-istruzioni per la gestione dei segmenti.

10

M. Rebaudengo, M. Sonza Reorda

# Pseudo-istruzioni per la Definizione di Variabili

## Formato:

*[nome] direttiva valore*

- *direttive*: BYTE (DB), WORD (DW), DWORD (DD), QWORD (DQ)
- *valore* può essere:
  - un valore numerico
  - una stringa tra apici
  - il carattere ?
  - il costrutto `num DUP (val)` che replica `num` volte il valore `val`.

11

M. Rebaudengo, M. Sonza Reorda

## Direttiva DB

La direttiva DB permette di definire strutture dati costituite da byte.

È utilizzata per la memorizzazione di:

- caratteri
- stringhe
- numeri interi.

### Esempi:

```
CITTA      DB      "T","o","r","i","n","o"  
CITTA2     DB      "Torino"  
STR2       DB      'Programma perfetto !'  
TAB        DB      1, 123, 84, 5 DUP (?), -10
```

12

M. Rebaudengo, M. Sonza Reorda

## Direttiva DW

La direttiva DW permette di definire strutture dati costituite da word (2 byte).

È utilizzata per la memorizzazione di:

- 1 o 2 caratteri
- numeri interi
- indirizzi di offset.

La direttiva DW NON può essere utilizzata per memorizzare una sequenza di caratteri.

Esempi:

```
TAB      DW    1, 350, -4000, 1024
LISTA    DB    100 DUP (?)
LIST_OFF DW    LISTA
```

13

M. Rebaudengo, M. Sonza Reorda

## Direttiva DW per la memorizzazione di offset

La direttiva DW può essere utilizzata per allocare la memoria necessaria per memorizzare un offset.

Esempio

```
LISTA          DB    100 DUP (?)
LISTA_OFFSET   DW    LISTA
```

14

M. Rebaudengo, M. Sonza Reorda

## Direttiva DD

La direttiva DD permette di definire strutture dati costituite da *doubleword* (4 byte).

È utilizzata per la memorizzazione di:

- 1 o 2 caratteri
- numeri interi
- indirizzi interi (registro di segmento e offset).

La direttiva DD NON può essere utilizzata per memorizzare una sequenza di caratteri.

### Esempi:

```
TAB          DD    125000
LISTA        DB    100 DUP (?)
LIST_ADD     DD    LISTA
```

15

M. Rebaudengo, M. Sonza Reorda

## Direttiva DD per la memorizzazione di indirizzi

La direttiva DD può essere utilizzata per allocare la memoria necessaria per memorizzare un indirizzo (offset+segmento).

### Esempio

```
LISTA        DB    100 DUP (?)
LISTADDR     DD    LISTA
```

La memorizzazione avviene con l'offset nella word avente indirizzo minore.

16

M. Rebaudengo, M. Sonza Reorda

## Direttive DQ e DT

Le direttive DQ e DT permettono di definire dati costituiti da *quadword* (8 byte) e *tenbyte* (10 byte).

Quadword e tenbyte sono utilizzati per memorizzare numeri di grande precisione per il coprocessore matematico.

Una variabile tenbyte può memorizzare un dato di tipo *decimale impaccato* secondo il seguente formato:

- 2 numeri decimali per ogni byte
- il primo byte memorizza il segno
- gli altri 9 possono memorizzare fino a 18 cifre decimali.

17

M. Rebaudengo, M. Senza Reorda

## Definizione di Costanti

### Formato:

*simbolo EQU espressione*

*simbolo = espressione*

Definiscono costanti simboliche durante l'assemblaggio.

Le costanti definite con = possono essere cambiate di valore nel corso del programma, a differenza di quelle definite con EQU.

*espressione* può essere un'espressione intera, una stringa di 1 o 2 caratteri, o un indirizzo.

### Esempi

```
column EQU 80
row     EQU 25
screen EQU colum*row
```

18

M. Rebaudengo, M. Senza Reorda

# Direttive per la Gestione dei Segmenti

Permettono la definizione e la gestione dei segmenti.

A partire dalla versione 5.0 sono state introdotte alcune pseudo-istruzioni (`.MODEL`, `.DATA`, `.CODE`, `.STACK`) che semplificano il problema.

19

M. Rebaudengo, M. Sonza Reorda

## Direttiva `.MODEL`

La direttiva `.MODEL` definisce gli attributi di base relativi all'intero modulo sorgente:

- modello di memoria
- convenzione dei nomi
- sistema operativo (DOS o WINDOWS)
- tipo di stack.

20

M. Rebaudengo, M. Sonza Reorda

## Direttiva `.MODEL`

### Formato

```
.MODEL    modello [opzioni, ...]
```

### Uso

Il `modello` è obbligatorio e definisce le dimensioni dei segmenti di codice e di dato.

Le opzioni possibili sono relative a:

- convenzioni dei nomi e delle chiamate per procedure e simboli pubblici
- tipo di *stack*.

## Modelli di memoria

Il MASM supporta i modelli di memoria standard usati dai linguaggi di alto livello Microsoft.

I possibili modelli di memoria sono i seguenti:

	Default	Default	Data e
	Code	Data	Code Combinati
TINY	Near	Near	Sì
SMALL	Near	Near	No
MEDIUM	Far	Near	No
COMPACT	Near	Far	No
LARGE	Far	Far	No
HUGE	Far	Far	No

## Tipi di stack

Nel comando `.MODEL` è possibile specificare il tipo di stack:

- **NEARSTACK** (default): il segmento di stack ed il segmento di dato sono all'interno dello stesso segmento fisico (DS ed SS coincidono).
- **FARSTACK**: i segmenti di dato e di stack sono distinti.

## Linguaggio

All'interno della direttiva `.MODEL` è possibile specificare un'opzione che garantisce la compatibilità con linguaggi di alto livello.

Le possibili opzioni sono:

- **PASCAL**
- **BASIC**
- **C**
- **FORTRAN.**

## Creazione dei segmenti di Dato

I programmi possono contenere sia dati di tipo NEAR sia dati di tipo FAR.

In generale i dati più importanti e più frequentemente usati sono memorizzati in un'area dati di tipo NEAR, dove l'accesso è più veloce, mentre i dati meno frequentemente usati sono memorizzati in segmenti di dato di tipo FAR.

La direttiva `.DATA` crea un segmento di dato di tipo NEAR.

La direttiva `.FARDATA` crea un segmento di dato di tipo FAR. La direttiva `.STARTUP` in un modello di memoria che prevede più segmenti di dato NON inizializza i registri DS ed ES. Tale operazione deve essere fatta dal programmatore.

25

M. Rebaudengo, M. Sonza Reorda

## Esempio

```
.MODEL          compact
.STACK          2048
.FARDATA        segm1
vett1          DB 100 DUP (?)
               .FARDATA    segm2
vett2          DB 100 DUP (?)
               .CODE
ASSUME DS:segm1, ES:segm2
               .STARTUP
MOV    AX, segm1
MOV    DS, AX
MOV    AX, segm2
MOV    ES, AX
...
.EXIT
END
```

26

M. Rebaudengo, M. Sonza Reorda

## Creazione dei segmenti di Codice

Nei modelli di memoria `tiny`, `small` e `compact` il segmento di codice è di tipo `NEAR`: più segmenti di codice su più moduli sono mappati su uno stesso segmento fisico di codice.

Nei modelli di memoria `medium` e `large` il segmento di codice è di tipo `FAR`: ogni direttiva `.CODE` corrisponde ad un segmento di codice fisico diverso.

In uno stesso modulo è possibile creare più segmenti di codice utilizzando diverse direttive `.CODE` e assegnando un nome ad ogni singolo segmento.

27

M. Rebaudengo, M. Sonza Reorda

## SEGMENT e ENDS

Definiscono l'inizio e la fine di un segmento, specificandone anche alcuni attributi.

### Formato

```
nome SEGMENT {align} {READONLY} {combine} {'class'}  
nome ENDS
```

### Esempio

```
CODE SEGMENT    WORD PUBLIC 'CODE'  
...  
CODE ENDS
```

28

M. Rebaudengo, M. Sonza Reorda

## SEGMENT e ENDS: campo align

Definisce il tipo di allineamento con cui inizia il segmento.

Può assumere i valori seguenti:

- **BYTE**
- **WORD**
- **DWORD**
- **PARA** (default)
- **PAGE** (la pagina occupa 256 byte).

## SEGMENT e ENDS: campo READONLY

Specificando l'attributo **READONLY**, l'assemblatore esegue una serie di controlli che producono un messaggio di errore quando si tenta di modificare il contenuto del segmento tramite un indirizzamento diretto.

## SEGMENT e ENDS: campo combine

Definisce come il linker deve combinare i segmenti che hanno lo stesso nome, ma che compaiono in file diversi.

Può assumere i seguenti valori:

- **PRIVATE:** (default) non combina i segmenti di moduli diversi
- **PUBLIC** o **MEMORY:** concatena tutti i segmenti con lo stesso nome
- **STACK:** concatena tutti i segmenti di stack con lo stesso nome
- **COMMON:** sovrappone i segmenti con lo stesso nome
- **AT exp:** forza l'inizio del segmento all'indirizzo **exp**.

31

M. Rebaudengo, M. Sonza Reorda

## SEGMENT e ENDS: campo class

Permette una ulteriore distinzione tra segmenti, facendo sì che segmenti con lo stesso nome non siano combinati se il loro campo **class** assume valori diversi.

32

M. Rebaudengo, M. Sonza Reorda

## Direttiva ASSUME

### Formato

```
ASSUME segm_reg:segm_name {, segm_reg:segm_name}
```

### Significato

Associa il registro di segmento `segm_reg` al segmento di nome `segm_name`.

L'informazione fornita da `ASSUME` viene utilizzata dall'assemblatore per determinare il registro di segmento da utilizzare per il calcolo degli indirizzi fisici di variabili.

La direttiva `ASSUME` NON provvede al caricamento dell'indirizzo del segmento nel registro relativo.

33

M. Rebaudengo, M. Sonza Reorda

## Esempio

```
DSEG      SEGMENT  PARA PUBLIC 'DATA'
VAR1      DB      0
DSEG      ENDS
ESEG      SEGMENT  PARA PUBLIC 'DATA'
VAR2      DB      0
ESEG      ENDS
CSEG      SEGMENT  PARA PUBLIC 'CODE'
          ASSUME   CS:CSEG,SS:STACK
BEGIN:    MOV     AX,DSEG
          MOV     DS,AX
          ASSUME   DS:DSEG      ;associa al registro
                                ;DS il segmento DSEG
          MOV     AX, ESEG
          MOV     ES, AX
```

34

M. Rebaudengo, M. Sonza Reorda

```
                ASSUME    ES:ESEG    ;associa al registro
                                ;ES il segmento ESEG
MOV    AL, VAR1    ;la variabile VAR1 è nel
                                ;segmento DSEG
MOV    VAR2, AL    ;la variabile VAR2 è nel
                                ;segmento ESEG
                CSEG ENDS
STACK      SEGMENT  PARA STACK 'STACK'
                DB    64 DUP("STACK  ")
STACK      ENDS
                END    BEGIN
```

35

M. Rebaudengo, M. Sonza Reorda

## Direttiva END

### Formato

```
END {etichetta}
```

### Significato

Conclude un modulo di programma.

Se il modulo contiene la prima istruzione del programma, la relativa etichetta deve essere specificata come operando.

In tal modo l'assemblatore, il linker ed il loader possono comunicare al processore l'istruzione da cui iniziare l'esecuzione del programma.

36

M. Rebaudengo, M. Sonza Reorda

# Operatori

Si distinguono in

- operatori per il calcolo degli attributi di una variabile  
(TYPE, LENGTH, SIZE, SEG, OFFSET)
- operatori aritmetici, logici e relazionali  
(+, -, \*, /, MOD, AND, OR, NOT, XOR, EQ, GT, etc.)
- operatori che modificano il tipo di una variabile  
(PTR).

37

M. Rebaudengo, M. Sonza Reorda

## Operatore OFFSET

**Formato:**

*OFFSET variabile*

**Funzionamento:**

L'operatore OFFSET restituisce il valore dell'offset di una variabile.

**Applicazione:**

L'operatore OFFSET può essere usato in alternativa all'istruzione LEA.

**Esempio:**

Le due seguenti istruzioni sono equivalenti:

```
MOV      AX, OFFSET VAR
LEA      AX, VAR
```

38

M. Rebaudengo, M. Sonza Reorda

## Limiti dell'operatore OFFSET

L'operatore `OFFSET` può essere applicato solo ad operandi indirizzati direttamente attraverso un nome di variabile e non ad operandi indirizzati indirettamente.

### Esempio

```
MOV AX, OFFSET VAR[SI] ;Errore !!
```

Si possono utilizzare le seguenti istruzioni:

```
MOV AX, OFFSET VAR
```

```
ADD AX, SI
```

oppure:

```
LEA AX, VAR[SI]
```

## Operatori TYPE, LENGTH e SIZE

### Formato:

*TYPE*            *variabile*

*LENGTH*        *variabile*

*SIZE*            *variabile*

### Funzionamento:

L'operatore `TYPE` restituisce il numero di byte dell'operando.

L'operatore `LENGTH` restituisce il numero di unità allocate per l'operando.

L'operatore `SIZE` restituisce lo spazio di memoria utilizzato dall'operando.

`SIZE = LENGTH * TYPE`

## Operatore LENGTH

L'operatore LENGTH ha senso solo per variabili allocate attraverso l'operatore DUP. In tutti gli altri casi restituisce infatti il valore 1.

### Esempio

```

EXP      DW      5 DUP (?)
EXP2     DW      1, 2, 3, 4, 5
         MOV     AX, LENGTH EXP
         MOV     BX, TYPE EXP
         MOV     CX, SIZE EXP
         MOV     DX, LENGTH EXP2

```

Dopo queste istruzioni in AX c'è 5, in BX c'è 2, in CX c'è 10 ed in DX c'è 1.

41

M. Rebaudengo, M. Sonza Reorda

## Operatore SEG

### Formato:

*SEG*            *variabile*

### Funzionamento:

L'operatore SEG restituisce l'indirizzo di inizio del segmento a cui appartiene l'operando *variabile*.

### Esempio:

```

LEA     SI, STR
MOV     AX, SEG STR
MOV     DS, AX

```

è equivalente a:

```

LDS     SI, STR

```

42

M. Rebaudengo, M. Sonza Reorda

# Operatori Aritmetici, Logici e Relazionali

- aritmetici: +, -, \*, /, MOD
- logici: AND, OR, NOT, XOR
- relazionali: EQ, NE, LT, LE, GT, GE

Gli operatori possono comparire esclusivamente in espressioni valutabili al tempo di assemblaggio.

43

M. Rebaudengo, M. Sonza Reorda

## Operatore PTR

### Formato:

*tipo PTR nome*

### Funzionamento:

L'operatore PTR forza l'assemblatore a modificare per l'istruzione corrente il *tipo* del dato avente come identificatore *nome*.

### Esempio:

```
.DATA
TOT      DW      ?
          .CODE
MOV      BH, BYTE PTR TOT
MOV      CH, BYTE PTR TOT+1
```

44

M. Rebaudengo, M. Sonza Reorda

## Operatore PTR

### Esempio

(segue)

```
.DATA
COPPIA DB 2 DUP (?)
.CODE
MOV AX, COPPIA ; ERRORE!
MOV AX, WORD PTR COPPIA
```

### Esempio

```
INC [BX]
```

La cella da incrementare corrisponde ad una word o ad un byte ?

L'assemblatore non può saperlo e genera errore.

Soluzione:

```
INC BYTE PTR [BX]
```

45

M. Rebaudengo, M. Sonza Reorda

## Modi di Indirizzamento

Il Modo di Indirizzamento di una istruzione definisce lo spazio indirizzabile ed il metodo da usarsi per il calcolo dell'indirizzo stesso.

Gli operandi possono essere contenuti:

- in registri
- nell'istruzione stessa
- in memoria
- su una porta di I/O.

46

M. Rebaudengo, M. Sonza Reorda

## Modi di Indirizzamento (II)

I Modi di Indirizzamento sono i seguenti:

- Register
- Immediate
- Direct
- Register Indirect
- Base Relative
- Direct Indexed
- Base Indexed.

47

M. Rebaudengo, M. Sonza Reorda

## Register Addressing

Nell'istruzione è specificato il registro da utilizzare come operando.

### Esempio

```
MOV BX, AX
```

AX	4F02
----	------

BX	xxxx
----	------

AX	4F02
----	------

BX	4F02
----	------

48

M. Rebaudengo, M. Sonza Reorda

## Immediate Addressing

Nell'istruzione stessa è indicato il dato da usare come operando.

### Esempio

BH 

xxxx
------

MOV BH, 07h

BH 

07
----

49

M. Rebaudengo, M. Sonza Reorda

## Note

- L'operando può anche essere un simbolo definito con una pseudo-istruzione EQU.

### Esempio

K EQU 1024

:

MOV CX, K

- il dato indicato nell'istruzione viene trasformato dall'assemblatore in formato binario su 8 o 16 bit, e scritto nel campo opportuno dell'istruzione macchina.

50

M. Rebaudengo, M. Sonza Reorda

## Direct Addressing

Nell'istruzione è contenuto l'*identificatore* di una variabile, corrispondente all'Effective Address della parola di memoria da utilizzare come operando.

Alla variabile può essere sommato o sottratto un *displacement* attraverso gli operatori + e -.

Una notazione equivalente prevede l'utilizzo delle parentesi quadre per racchiudere il displacement o l'identificatore della variabile.

L'indirizzo fisico è ottenuto sommando l'EA con il contenuto del Data Segment Register DS.

51

M. Rebaudengo, M. Sonza Reorda

## Direct Addressing

(segue)

L'indirizzo fisico è ottenuto sommando l'EA con il contenuto del Data Segment Register DS.

**Formalismo:**

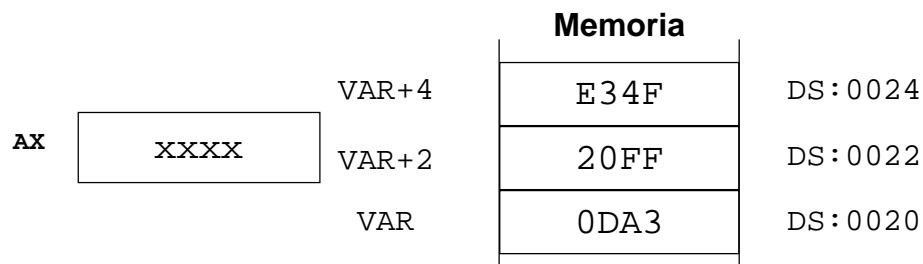
```
nome  
[ nome ]  
nome+displacement  
nome[displacement]  
nome-displacement  
nome[-displacement]
```

**Esempi:**

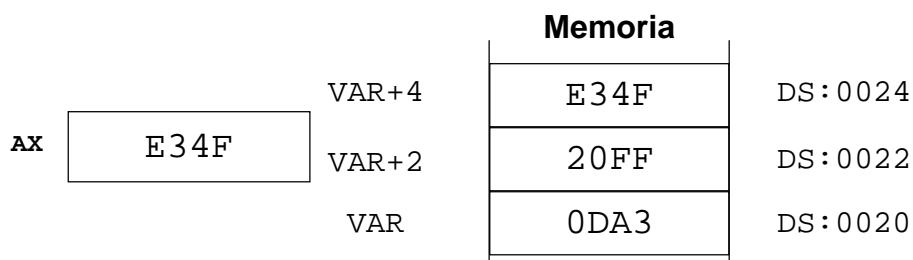
```
MOV AX, TABLE  
MOV AX, TABLE+2  
MOV AX, TABLE[2]
```

52

M. Rebaudengo, M. Sonza Reorda



MOV AX, VAR[4]



53

M. Rebaudengo, M. Sonza Reorda

## Segment Override

Nel modo di indirizzamento diretto il registro di segmento di default è il registro DS.

L'operatore di segment override (:) permette di utilizzare per il calcolo dell'indirizzo fisico un registro di segmento diverso da quello di *default*.

### Esempi

MOV AX, ES:VAR2

54

M. Rebaudengo, M. Sonza Reorda

## Register Indirect Addressing

L'Effective Address dell'operando è contenuto in uno dei seguenti registri:

- Base (BX)
- Index Register (DI oppure SI)
- Base Pointer (BP).

Viene detto *Indirect* perchè nell'istruzione è indicato dove trovare l'indirizzo dell'operando.

### Esempio

```
MOV AX, [BX]
```

55

M. Rebaudengo, M. Sonza Reorda

## Registri di segmento

Ciascun registro è abbinato ad un particolare registro di segmento:

DS ⇒ BX

DS ⇒ SI

DS ⇒ DI

(tranne per le istruzioni di manipolazione di stringhe in cui il registro di segmento è ES).

SS ⇒ BP.

56

M. Rebaudengo, M. Sonza Reorda

## Esempio

Codice per il trasferimento di un vettore in un altro usando il Register Indirect Addressing.

```

:
MOV SI, OFFSET SOURCE_VECT
MOV DI, OFFSET DEST_VECT
MOV CX, LENGTH SOURCE_VECT
QUI: MOV AX, [SI]
MOV [DI], AX
ADD SI, 2
ADD DI, 2
LOOP QUI

```

57

:

M. Rebaudengo, M. Sonza Reorda

## Base Relative Addressing

L'Effective Address dell'operando è calcolato sommando il contenuto di uno dei Base Register (BX o BP) ad un *displacement* rappresentato da una costante presente nell'istruzione stessa.

Formato assembler:

```
[ <register> ] + <constant>
```

### Esempio

```
MOV AX, [BX]+4
```

### Nota

```
MOV AX, [BP]+4
```

```
MOV AX, 4[BP]
```

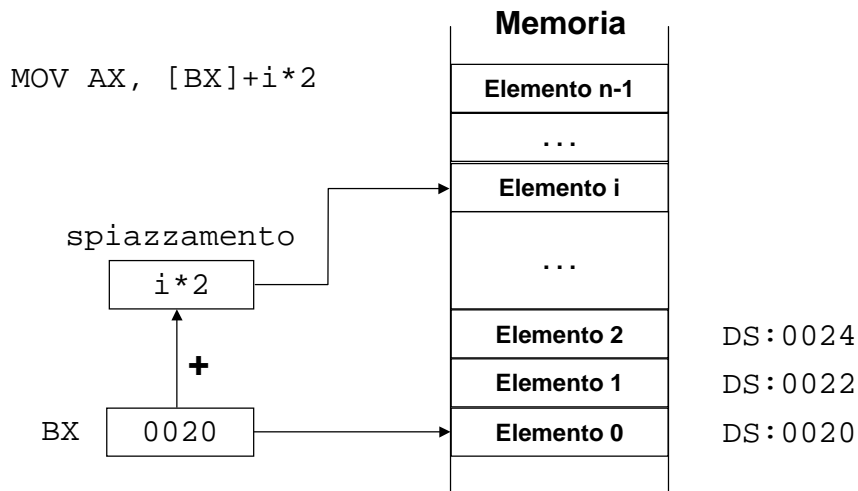
```
MOV AX, [BP+4]
```

sono equivalenti.

58

M. Rebaudengo, M. Sonza Reorda

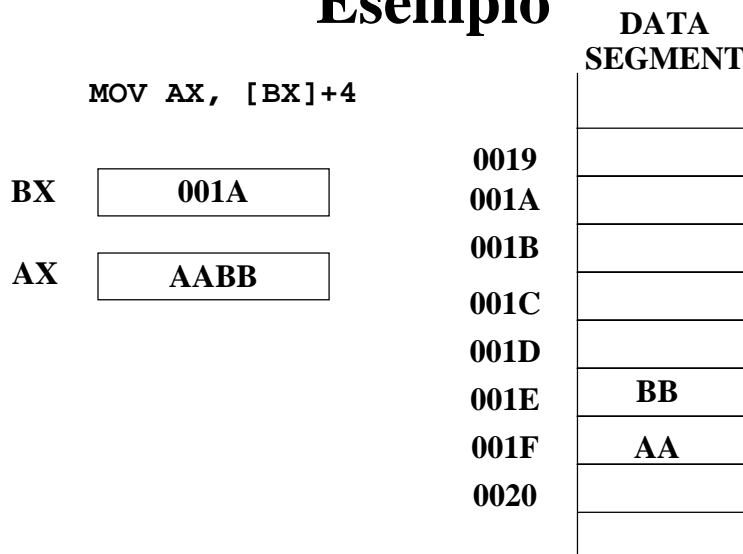
## Calcolo dell'EA per un indirizzamento *Base Relative*



59

M. Rebaudengo, M. Sonza Reorda

## Esempio



60

M. Rebaudengo, M. Sonza Reorda

## Direct Indexed Addressing

L'Effective Addressing dell'operando è calcolato sommando il valore di un *offset* contenuto in una variabile al contenuto di un *displacement* contenuto in uno degli Index Register (SI o DI).

### Formato

<variable> [SI]

<variable> [DI]

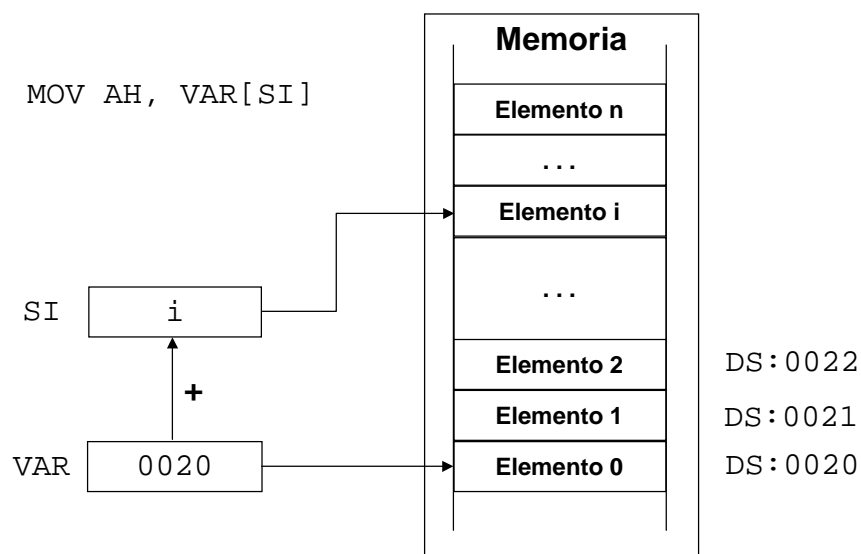
### Esempio

```
MOV AX, TABLE[DI]
```

61

M. Rebaudengo, M. Sonza Reorda

## Calcolo dell'EA per un indirizzamento *Direct Indexed*



62

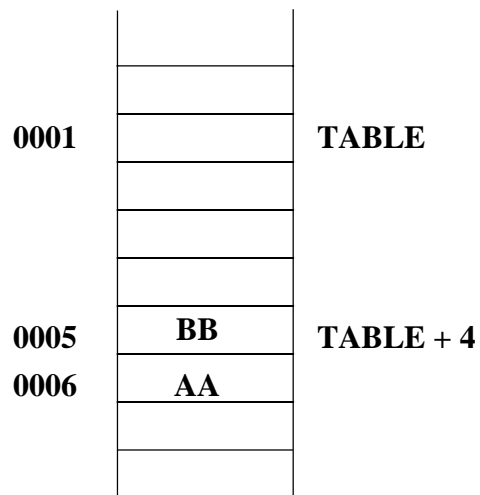
M. Rebaudengo, M. Sonza Reorda

## Esempio

```
MOV AX, TABLE[DI]
```

DI     **0004**

AX     **AABB**



63

M. Rebaudengo, M. Sonza Reorda

## Esempio

Codice per il trasferimento di un vettore in un altro usando il Direct Indexed Addressing.

```

:
MOV SI, 0
MOV CX, LENGTH SOURCE_VECT
QUI: MOV AX, SOURCE_VECT [SI]
      MOV DEST_VECT [SI], AX
      ADD SI, 2
      LOOP QUI
:

```

64

M. Rebaudengo, M. Sonza Reorda

## Base Indexed Addressing

L'Effective Address dell'operando è calcolato come somma dei seguenti termini:

- contenuto di uno dei *Base Register* (BX o BP)
- contenuto di uno degli *Index Register* (SI o DI)
- un campo opzionale *displacement* che può essere un identificatore di variabile oppure una costante numerica.

65

M. Rebaudengo, M. Sonza Reorda

## Base Indexed Addressing

(segue)

### Formato

```
<variable> [BX] + [SI]
<variable> [BX] + [DI]
<variable> [BP] + [SI]
<variable> [BP] + [DI]
```

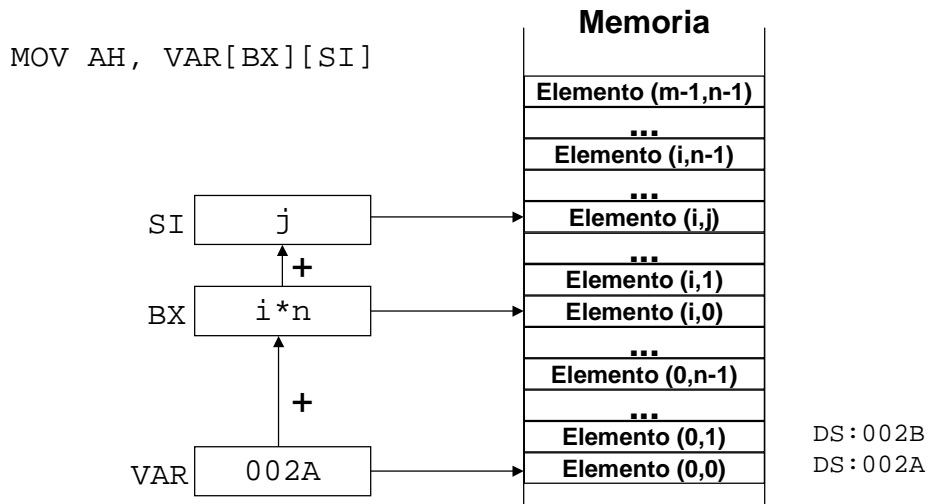
### Esempio

```
MOV AX, TAB[BX][DI]+6
```

66

M. Rebaudengo, M. Sonza Reorda

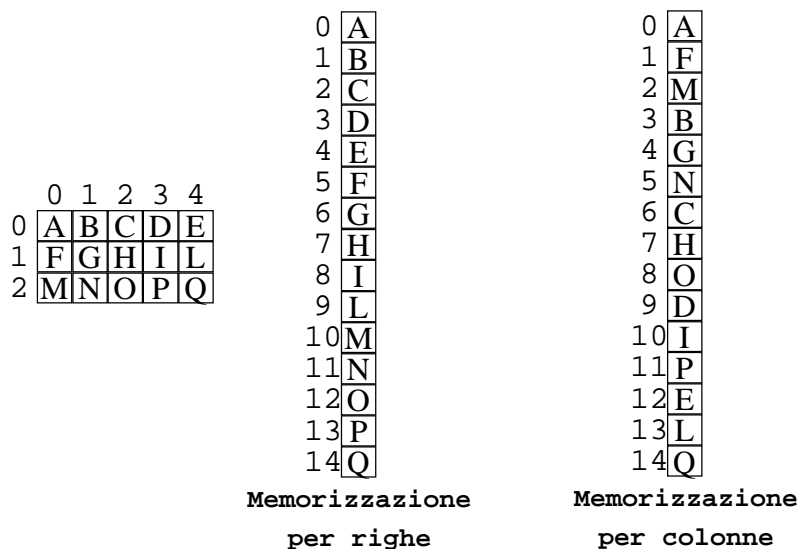
## Calcolo dell'EA per un indirizzamento *Base Indexed*



67

M. Rebaudengo, M. Sonza Reorda

## Memorizzazione di una matrice



68

M. Rebaudengo, M. Sonza Reorda

## Copia di una riga in una matrice di dati

### Specifiche:

Date due matrici SORG e DEST di dimensione 4x5, si deve copiare la quarta riga da SORG a DEST.

```
main()
{
int i;
int sorg[4][5], dest[4][5];
...
for (i=0 ; i < 5 ; i++)
    dest[3][i] = sorg[3][i];
...
}
```

69

M. Rebaudengo, M. Sonza Reorda

## Soluzione Assembler

```
RIGHE      EQU      4
COLONNE    EQU      5
.MODEL     small
.STACK
.DATA
SORG       DW      RIGHE*COLONNE DUP (?) ; matrice sorgente
DEST       DW      RIGHE*COLONNE DUP (?) ; matrice destinazione
.CODE
MOV        BX, COLONNE*3*2 ; caricamento in BX del primo
                        ; elemento della quarta riga
MOV        SI, 0 ; inizializzazione del registro SI
MOV        CX, 5 ; in CX del numero di colonne
ciclo:     MOV        AX, SORG[BX][SI]
MOV        DEST[BX][SI], AX
ADD        SI, 2 ; scansione dell'indice
LOOP       ciclo ; fine? No => va a ciclo
...        ; Sì
```

70

M. Rebaudengo, M. Sonza Reorda

## **Istruzioni**

**L'Assembler 8086 rende disponibili 92 tipi di istruzioni, raggruppabili nelle seguenti classi:**

- **Trasferimento Dati**
- **Aritmetiche**
- **Manipolazione di Bit**
- **Control Transfer**
- **Manipolazione di Stringhe**
- **Interrupt Handling**
- **Process Control.**