Il linguaggio Assembly della CPU 8086

Il linguaggio Macchina

- Fortemente orientato alla macchina
  - Direttamente eseguibile
  - Praticamente illeggibile
- Utilizzato negli anni '50 – '70 per sviluppare programmi
  - Attualmente in disuso
- Ogni famiglia di CPU ha un proprio linguaggio macchina
  - Lo stesso programma in linguaggio macchina non può non essere eseguito su CPU di famiglie diverse

Il linguaggio Assembly

- È un linguaggio di programmazione di basso livello (orientato alla macchina), in quanto permette semplicemente la scrittura mnemonica delle istruzioni in linguaggio macchina
  - Maggiormente leggibile rispetto al linguaggio macchina
  - Deve essere tradotto in linguaggio macchina per essere eseguito
    - Traduzione molto semplice
  - Dipendenza dall'HW
    - per scrivere programmi in Assembly è necessario conoscere l'architettura (livello ISA) della macchina a cui il linguaggio si riferisce

3

Sviluppo in Assembly

- Utilizzato negli anni '70 – '80 per sviluppare programmi
- Attualmente utilizzato solo nello sviluppo di particolari applicazioni
  - Piccoli moduli altamente performanti (es. nei sistemi operativi)
    - Sviluppo integrato: all'interno di un programma scritto in linguaggio di alto livello (es. C, C++), si inseriscono blocchi di istruzioni Assembly
  - Sistemi real-time
    - Sviluppo Stand-alone: Intero programma scritto in Assembly
      - Adatto a programmi molto piccoli

4

Programmazione di basso livello

- Vantaggi
  - Programmi veloci
    - Possibilità di raggiungere performance elevatissime
      - Uso nei sistemi real-time
  - Sfruttamento ottimale delle caratteristiche dell'elaboratore
- Svantaggi
  - Programmi difficili da scrivere e leggere
    - È facile commettere errori
    - È praticamente impossibile scrivere grandi programmi
  - Programmi non portabili
- Scopo didattico
  - conoscere l'Assembly per migliorare la conoscenza dell'architettura dell'elaboratore

5

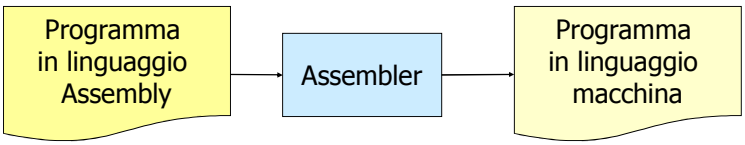
Linguaggi di programmazione di basso livello

- Linguaggio macchina
  - Una istruzione è una stringa di bit che viene interpretata ed eseguita dalla CPU
  - Esempio di istruzione: `0100100100101000`
- Linguaggio Assembly
  - Una istruzione è una stringa alfanumerica che viene tradotta in una corrispondente istruzione in linguaggio macchina
  - Esempio di istruzione: `MOV AX,12`

6

Assembler

- L' assembler (o assemblatore) è un programma che traduce in linguaggio macchina i programmi scritti in linguaggio Assembly
  - legge un file contenente il programma scritto in linguaggio Assembly (codice sorgente)
  - produce un file contenente istruzioni in linguaggio macchina (codice oggetto)



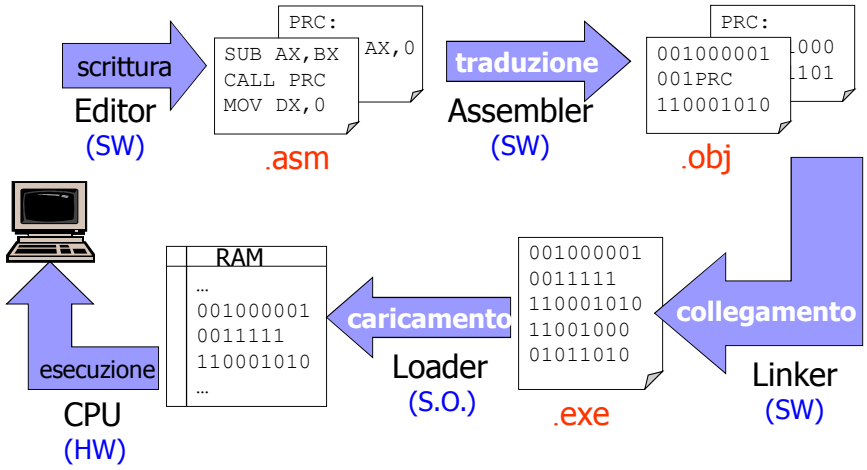
7

Programmi in Assembly

- Un programma in Assembly è una sequenza di istruzioni in linguaggio **Assembly**
  - Detto anche "Codice Sorgente"
  - Estensione dei file: .ASM (usualmente)
- Un programma in Assembly viene tradotto ("assemblato") da un programma, detto **Assembler**
  - Il risultato della traduzione è detto "Codice Oggetto"
  - Estensione del file: .OBJ o .O (usualmente)
- Diversi codici oggetto possono essere collegati fra loro attraverso un **linker**
  - Il risultato del linking è detto "Codice Eseguibile"
  - Estensione del file: .EXE

8

Generazione di un programma eseguibile



9

Elementi di base di un programma assembly

- Un programma scritto in Assembly è composto da statement
- Ogni statement generalmente occupa una riga di codice e comprende:
  - Istruzioni
    - comandi Assembly direttamente traducibili in linguaggio macchina contenenti nomi mnemonici (abbreviazioni per denotare istruzioni, registri, ecc) e nomi simbolici (label per variabili, costanti, etichette, ecc)
  - Pseudo-istruzioni (o direttive)
    - non si traducono in istruzioni in linguaggio macchina
    - permettono di aumentare la leggibilità dei programmi e forniscono all'Assembler alcune direttive sulla traduzione in linguaggio macchina
  - Commenti
    - stringhe di testo precedute dal carattere ";"
    - usati per documentare il codice

10

Elementi di base di un programma assembly

- Le componenti di un programma Assembly sono diversamente gestite dall'assemblatore
  - I commenti vengono ignorati
    - tutti i caratteri in uno statement che seguono un '**;**' sono trascurati
  - Le direttive non sono tradotte in istruzioni in linguaggio macchina
    - forniscono all'assemblatore direttive sulla traduzione
      - Es: le dichiarazioni di variabili e di costanti sono direttive
  - Le istruzioni rappresentano comandi Assembly direttamente traducibili in linguaggio macchina
    - i nomi mnemonici vengono tradotti nel corrispondente opcode (codice operativo)

11

Identificatori

- Gli identificatori sono usati come nomi assegnati ad entità definite dal programmatore (variabili, costanti, label,…)
- Non è possibile usare identificatori uguali alle parole chiave del linguaggio (nomi delle istruzioni, nomi dei registri, nomi di operatori, direttive, …)
  - Esempi: `main, ciclo, END, A1_1, MUL, b1234r, AND, ...`

12

Identificatori

- Un identificatore consiste di una sequenza di caratteri alfanumerici
  - caratteri (a-z, A-Z)
  - cifre da 0 a 9
  - uno dei 4 caratteri speciali @ _ $?
- Un identificatore non può iniziare con una cifra
- Caratteri maiuscoli e minuscoli sono equivalenti
  - l'assembler 8086 è *case insensitive*
  - Es. `VAR1` e `var1` rappresentano lo stesso identificatore

13

Variabili

- Le variabili sono nomi dati a indirizzi di locazioni di memoria
  - Una variabile ha un **nome (indirizzo)** e un **valore**
- In Assembly l'accesso ad un dato può avvenire
  - direttamente attraverso l'indirizzo di memoria
    - Esempio: `MOV AX, [0100h]`
  - denotando l'indirizzo con un identificatore (maggiore leggibilità dei programmi)
    - Esempio: `MOV AX, VAR`

identificatore di variabile

14

Definizione di variabili (scalari)

- Sintassi:

  `<nome> <tipo> <val_iniziale>`

dove

`<nome>` è un identificatore

`<tipo>` indica la dimensione e può essere

- **DB**: Byte (8 bit)
- **DW**: Word (16 bit)
- **DD**: Double Word (32 bit) (non usato nell'8086)

`<val_iniziale>` è il valore di inizializzazione e può essere

- un valore numerico
- una stringa di caratteri racchiusa tra apici
- il carattere `?` (indica nessun valore)

15

Definizione di variabili scalari

- Le variabili possono essere dichiarate ovunque nel programma
  - Normalmente all'inizio del programma o alla fine
- Esempi

```
VALORE DW ?       ;una word non inizializzata
Number1 DB 0      ; un byte inizializzato a 0
Number2 DW 1      ;una word inizializzata a 1
a DB 12           ; un byte inizializzato a 12
max DW 0FFh       ;una word inizializzata a 255
CONFERMA DB 'Y'   ; un byte inizializzato a 089
ANNULLA DB 'N'    ; un byte inizializzato a 078
at DB '@'         ;un byte inizializzato a 064
```

16

Le variabili (array)

- Gli array sono sequenze di dati di tipo omogeneo
  - Es.: vettori numerici (1,2,5,3)
  - Es.: stringhe di caratteri ('prova')
- Le variabili array si dichiarano similmente alle variabili scalari:

```
a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h ; array di 6 byte
b DB 'Hello'                       ; b uguale ad a
c DB 5 DUP(9)                      ; come c DB 9,9,9,9,9
d DW 10 DUP(?)   ; array di 10 word non inizializzate
```

17

Uso delle variabili

- Per leggere/scrivere il contenuto
  - **MOV** <Registro>, <Variabile>
  - **MOV** <Variabile>, <Registro>
- Per leggere l'indirizzo
  - **LEA** <Registro>, <Variabile>
    Esempio:
    ```
    LEA BX, VAR1 ;scrive in BX l'offset di VAR1
    ```
  - **MOV** <Registro>, **OFFSET** <Variabile>
    Esempio:
    ```
    MOV BX, OFFSET VAR1 ;scrive in BX l'offset di VAR1
    ```

18

Uso delle variabili array

- Per leggere/scrivere il contenuto di un elemento
  - **MOV** <Registro>, <Variabile>[indice]
  - **MOV** <Variabile>[indice], <Registro>
- Esempio

```
MOV AL, a[3]
;copia in AL l'elemento dell'array a di indice 3
```

- E' possibile usare i registri BX, SI, DI, BP per contenere l'indice:

```
MOV SI, 3
MOV AL, a[SI]
; copia in AL l'elemento dell'array a di indice 3
```

19

Costanti

- Le costanti sono nomi dati a valori
  - Non hanno indirizzo
    - non compaiono nel codice oggetto ed eseguibile
  - Il valore di una costante non può essere modificato
- Durante la traduzione, l'assemblatore sostituisce ogni occorrenza del nome di una costante con il valore corrispondente
  - Esempi:

```
MAX EQU 10h
_AT_ EQU '@'
MOV AL,_AT_ ;diventa MOV AL,40h
MOV AH,MAX ;diventa MOV AH,10h
```

20

Definizione di costanti

- Sintassi:

  `<nome>` **`EQU`** `<valore>`

dove
`<nome>` è un identificatore
`<valore>` può essere
  - un valore numerico
    - binario: `001101B`
    - ottale: `15O, 15Q`
    - esadecimale: `0Dh, 0BE8Ch` (deve iniziare con una cifra)
    - decimale: `13, 13d`
    - reale in base 10: `2.345678, 112E-3`
  - una stringa di caratteri tra apici
  - una espressione

Espressioni

- In una espressione si possono utilizzare i seguenti operatori:
  - Aritmetici (+,-,*,/ ,`MOD`, `SHL`, `SHR`)
  - Logici (`AND`, `OR`, `XOR`, `NOT`)
  - Relazionali (`EQ`, `NE`, `LT`, `GT`, `LE`, `GE`)

Formato delle istruzioni

```
label: opcode operandi ;commento
```

- etichetta (o label): è un identificatore associato all'istruzione
  - l'assemblatore la sostituisce con l'indirizzo dell'istruzione
- Operation Code: è lo mnemonico dell'istruzione
  - specifica l'operazione che deve essere eseguita dalla CPU
- operandi: riferimento a uno o più operandi
- commento : una frase che chiarisce il significato dell'istruzione

- Esempio

```
START:      MOV    AX, BX        ; copia BX in AX
            CMP    AX, 12        ; confronta AX con 12
```

23

Istruzioni

- L'Assembly 8086 rende disponibili i seguenti tipi di istruzioni:
  - Trasferimento Dati
  - Aritmetiche
  - Manipolazione di Bit
  - Trasferimento di Controllo
  - Manipolazione di Stringhe
  - Manipolazione di Interruzioni

24

Istruzioni di Trasferimento dati

General purpose	MOV POP PUSH XCHG	Move (Byte or Word) Pop a Word from the Stack Push Word onto Stack Exchange Registers
Input-Output	IN OUT	IN Input Byte or Word from input port Output Byte or Word to output Port
Trasferim. Indirizzi	LEA	Load Effective Address
Trasferimento Flag	LAHF SAHF POPF PUSHF	Load AH from 8 low bits of Flags Store AH into 8 low bits of Flags Pop Flags from the Stack Push Flags onto Stack

25

Istruzioni di Trasferimento

- MOV: copia il valore di una variabile/ registro in un registro/variabile
  - Sintassi: **MOV** `<dest>,<source>`
- Esempi

```
MOV AX,10 ; copia il valore 10 in AX
MOV BX,CX ; copia il valore da CX in BX
MOV DX,Number ; copia il valore di Number in DX
```

26

Trasferimenti non ammessi da MOV

- *Memoria ← memoria* `MOV NUM1,NUM2`
  - Si deve passare attraverso un registro generale
  - esempio:

```
MOV AX, NUM1
MOV NUM2, AX
```

- *Registro segmento ← immediato* `MOV DS,10`
  - Si deve passare attraverso un registro generale
  - esempio:

```
MOV AX, 10
MOV DS, AX
```

27

Trasferimenti non ammessi da MOV

- *Reg. segmento ← Reg. segmento* `MOV DS,ES`
  - Si deve passare attraverso un registro generale

```
MOV AX,ES
MOV DS,AX
```

  - oppure attraverso lo stack

```
PUSH ES
POP DS
```

- Qualsiasi trasferimento che utilizzi il registro CS come destinazione `MOV CS,AX`

28

Istruzioni di trasferimento con uso dello stack

- PUSH: Impilamento di un dato nello stack
  - Sintassi: **PUSH** <source>

```
PUSH AX
```

Registro a 16 bit

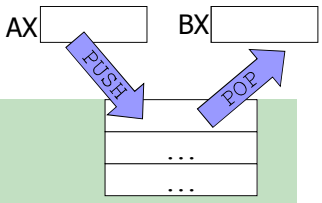
- POP: Estrazione di un dato dallo stack
  - Sintassi: **POP** <dest>

```
POP BX
POP DATO
```

Registro o variabile

29

Esempi




```
MOV AX,13
MOV BX,0
PUSH AX
POP BX
```

Equivale all'istruzione `MOV BX,AX`

Dopo l'esecuzione sia AX che BX contengono il valore 13

```
PUSH CX
PUSH AX
POP CX
POP AX
```

Equivale all'istruzione `XCHG AX,CX`

30

Input/Output

- A ciascun dispositivo è assegnato un numero a 16 bit o a 8 bit, detto *porta.*
- IN (INput byte or word from port)

  Sintassi: **IN** accumulatore,porta

```
IN al, 110 ; legge un byte dalla porta 110 in AL
IN ax, 110 ; legge una word dalla porta 110 in AX
```

- OUT (OUTput byte or word to port)

  Sintassi: **OUT** porta, accumulatore

```
mov ax, 1234
out 199, ax
; visualizza il numero 1234 sul display con porta 199
```

31

Istruzioni aritmetiche

Addizione	ADC ADD INC	Add with Carry Addition Increment
Sottrazione	SUB SBB DEC CMP NEG	Subtract Subtract with Borrow Decrement Compare Negative
Moltiplicazione	IMUL MUL	Integer Multiply, Signed Multiply, Unsigned
Divisione	DIV IDIV	Divide, Unsigned Integer Divide, Signed

32

Addizione

- ADD (ADDition)
  Sintassi: **ADD** <dest>,<source>

```
ADD AX,10
ADD TOTALE,10
ADD AX,CX
ADD AX,TOTALE
ADD TOTALE,AX
```

- ADC (ADd with Carry)
  Sintassi: **ADC** <dest>,<source>
  - somma anche il valore del flag di carry

33

Sottrazione

- SUB (SUBtract)
  Sintassi: **SUB** <dest>,<source>

```
SUB AX,10
SUB TOTALE,10
SUB AX,CX
SUB BX,TOTALE
SUB TOTALE,BX
```

- SBB (SuBtract with Borrow)
  Sintassi: **SBB** <dest>,<source>
  - sottrae anche il valore del flag di carry

34

Operazioni su 32 bit

Esempi

- Sommare i 32 bit memorizzati in BX:AX con DX:CX, con risultato in BX:AX

```
ADD AX,CX  ;somma i 16 bit meno significativi
ADC BX,DX  ;somma i 16 bit più significativi
```

- Sottrarre i 32 bit memorizzati in BX:AX a DX:CX, con risultato in BX:AX

```
SUB AX,CX  ;sottrae i 16 bit meno significativi
SBB BX,DX  ;sottrae i 16 bit più significativi
```

35

Negazione e Confronto

- NEG (two's complemet NEGation)
  Sintassi: **NEG <dest>**

```
NEG BX
NEG VAR
```

- CMP (CoMPare two operands)
  Sintassi: **CMP <dest>,<source>**

```
CMP AX,10
CMP AX,BX
CMP valore,0
CMP AX,valore
CMP valore,DX
```

36

Incremento/Decremento

- Incremento: INC (INCrement by 1)
  Sintassi: **INC** <dest>

```
INC BX
INC VAR
```

- Decremento: DEC (DECrement by 1)
  Sintassi: **DEC** <dest>

```
DEC BX
DEC VAR
```

37

Moltiplicazione

- Senza segno: MUL (MULtiply, unsigned)
  Sintassi: **MUL** <source>

```
MOV  AL,NUMERO1   ; operando a 8 bit
MUL  NUMERO2      ; risultato in AX (16 bit)

MOVE AX,VALORE1   ; operando a 16 bit
MUL  VALORE2      ; risultato in DX:AX (32 bit)
```

- Con segno: IMUL (Integer MULtiply)
  - Stessa sintassi di MUL

38

Divisione

- Senza segno: DIV (DIVision, unsigned)
  Sintassi: **`DIV`** `<source>`
- Con segno: IDIV (Integer DIVision)
  - Stessa sintassi di DIV

39

Divisione

- **Divisione di un byte per un altro byte**

```
MOV  AL,NUM_BTE  ; dividendo a 8 bit
DIV  DIVSR_BTE   ; quoziente in AL e
                 ; resto in AH
```

- **Divisione di una word per un byte**

```
MOV  AX,NUM_WRD  ; dividendo a 16 bit
DIV  DIVSR_BTE   ; quoziente in AL e
                 ; resto in AH
```

- **Divisione di una doubleword per una word**

```
MOV DX,NUM_HSW   ; dividendo
MOV AX,NUM_LSW   ; a 32 bit
DIV  DIVSR_WRD   ; quoziente in AX e
                 ; resto in DX
```

40

Istruzioni di Manipolazione di bit

Logiche	AND OR XOR NOT TEST	Logical AND Logical OR OR Exclusive OR Logical NOT Test
Di Traslazione	SAL SAR SHL SHR	Shift Arithmetic Left (=SHL) Shift Arithmetic Right Shift Logical Left (=SAL) Shift Logical Right
Di Rotazione	ROL ROR RCL RCR	Rotate Left Rotate Right Rotate through Carry Left Rotate through Carry Right

41

Istruzioni logiche

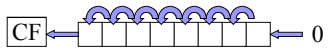
- Congiunzione **AND**
  Sintassi: **AND** `<dst>, <src>`
  - `AND AX,BX`
  - `AND AX,23h`
  - `AND valore, 12h`
  - `AND AX, valore`
  - `AND valore,BX`
- Disgiunzione OR/Disgiunzione esclusiva XOR
  Sintassi: **OR/XOR** `<dst>, <src>`
- Negazione NOT
  Sintassi **NOT** `<dst>`
  - `NOT AX`
  - `NOT valore`

42

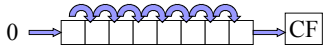
Shift

- Logico a sinistra: SHL (SHift Left) = SAL
  Sintassi: **SAL** <dest>, <posizioni>

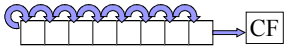
```
SHL AX,3
SHL AX,CL
```



  - Moltiplicazione veloce per 2^n
- Logico a destra: SHR (SHift Right)
  Sintassi: **SHR** <dest>, <posizioni>
  - Divisione veloce per 2^n



- Aritmetico a destra SAR (SHift Arithmetic Right)
  - **SAR** <dest>, <posizioni>



43

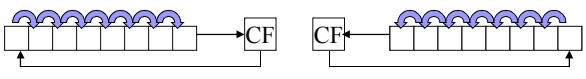
Rotazione

- Senza carry: ROL/ROR (Rotate Left/Right)
  Sintassi: **ROL/ROR** <dest>, <posizioni>

```
ROL AX,3
ROL AX,CL
```

- Con carry: RCL/RCR (Rotate through Carry Left/Right)
  - **RCL/RCR** <dest>,<posizioni>

```
RCL AX,3
RCL AX,CL
```



44

Esercizio

Quale sarà la stringa di bit memorizzata nel registro AX dopo l'esecuzione del seguente stralcio di programma?

AX =(00010110 00110000)$_2$
=(5680)$_{10}$

```
NUM1 DB 35d
NUM2 DB 64d
NUM3 DB 100d
NUM4 DB 80d
…
MOV AX,0
MOV AL,NUM1
ADD AL,NUM3
SUB AL,NUM2
MUL NUM4
```

45

Controllo del flusso

- Istruzioni che modificano la sequenza di esecuzione di un programma
  - Salti
    - Incondizionati
    - Condizionati
  - Chiamata e ritorno da procedure
  - Interruzioni

46

Salto incondizionato

- JMP (JuMP unconditionally)
  Sintassi: **JMP** <label>
  - Salta all'istruzione identificata dalla label specificata
  - Esempio

```
            MOV AL,5
            JMP next
            MOV BL, 10 ; codice non eseguito
            …
next:       MOV BH, 5
```

47

Salti condizionati

- JZ (Jump if Zero)
  Sintassi: **JZ** <label>
  - Se il flag Zero è 1, salta all'istruzione indicata dalla label

```
      MOV AX, 130
      SUB BL,DL
      JZ DIVZERO
      DIV BL
      JMP FINE
DIVZERO:   ; messaggio di errore
...
FINE:      ; altre operazioni
...
```

- JNZ (Jump if Not Zero)
  Sintassi: **JNZ** <label>
  - Se il flag Zero è 0, salta a <label>

48

Salti condizionati

- Salti condizionati, in funzione del risultato della `CMP`
- Operandi senza segno
  - JE/JNE: salta se dest =/≠ source
    Sintassi: **`JE/JNE`** `<label>`
  - JB/JBE: salta se dest </≤ source
    Sintassi: **`JB/JBE`** `<label>`
  - JA/JAE: salta se dest >/≥ source
    Sintassi: **`JA/JAE`** `<label>`

49

Salti condizionati

- Salti condizionati, in funzione del risultato della `CMP`
- Operandi con segno
  - JE/JNE: salta se dest =/≠ source
    Sintassi: **`JE/JNE`** `<label>`
  - JL/JLE: salta se dest </≤ source
    Sintassi: **`JL/JLE`** `<label>`
  - JG/JGE: salta se dest >/≥ source
    Sintassi: **`JG/JGE`** `<label>`

50

Esempio

```
MOV AH,A
MOV AL,B
CMP AH,AL ; confronto tra operandi senza segno
JB A_minore_di_B

; A maggiore o uguale di B
MOV A,AL
MOV B,AH
JMP fine

A_minore_di_B:
MOV A,AH
MOV B,AL

fine: RET

; definizione delle variabili
A DB 42 ; intero senza segno
B DB 12
```

Date due variabili di tipo byte A e B contenenti interi senza segno, memorizzare in A il valore minore e in B il valore maggiore

Esempio

```
MOV AH,A
MOV AL,B
CMP AH,AL ; confronto tra operandi con segno
JL A_minore_di_B

; A maggiore o uguale di B
MOV A,AL
MOV b,AH
JMP fine

A_minore_di_B:
MOV A,AH
MOV B,AL

fine: HLT

; definizione di variabili
A DB -2 ; intero con segno
B DB 12
```

Date due variabili di tipo byte A e B contenenti interi con segno, memorizzare in A il valore minore e in B il valore maggiore

IF…THEN…ELSE

IF (a=b)
THEN SonoUguali
ELSE SonoDiversi;
a = a + b;

```
MOV AX,a
MOV BX,b
CMP AX,BX
JE SonoUguali
JMP SonoDiversi
fineIfThen:
  ADD AX,BX
MOV a,AX
JMP fine
SonoUguali:
   … ; altre operazioni
JMP fineIfThen
SonoDiversi:
   … ; altre operazioni
JMP fineIfThen
fine:
   … ; altre operazioni
```

Iterazione

- LOOP
  Sintassi: **LOOP** <label>
- Esempio

```
MSG DB 10 DUP(?) ; array di 10 byte non inizializzati
N EQU 10

      MOV AL,'A'
      MOV SI,0
      MOV CX, N
next: MOV MSG[SI], AL
      INC SI
      LOOP next
```

54

Esempio

Dato un array di 5 elementi contenente valori interi, calcolare la somma dei suoi elementi e memorizzare il risultato in una variabile

```
      MOV CX, 5       ; inizializza il contatore
      MOV AL, 0       ; AL conterrà la somma
      MOV SI, 0       ; SI è l'indice
next:ADD AL, V[SI]
      INC SI
      LOOP next       ; loop until CX=0
      MOV Res, AL
      RET
; definizione di variabili
V DB 4, 3, 2, 1, 0 ; definizione dell'array
Res DB 0
```

55

Esempio

Dato un array di 5 elementi contenente valori interi, copiare l'array in un altro array

```
      MOV SI, OFFSET source
      MOV DI, OFFSET dest
      MOV CX, 5
 next:MOV AL, [SI]
      MOV [DI], AL
      INC SI
      INC DI
      LOOP next
      RET

 ; definizione degli array
 source DB 4, 3, 2, 1, 0
 dest DB 5 DUP (?)
```

56

Definizione di procedure

- La definizione di una procedura inizia con la direttiva **PROC** e termina con la direttiva **ENDP**
- Sintassi:

```
<nomeProc> PROC
... ; corpo della procedura
...
<nomeProc> ENDP
```

- Esempio:

```
sommaVettori PROC ;inizio procedura
    … ; istruzioni della procedura
    …
sommaVettori ENDP; fine procedura
```

57

Chiamata e ritorno da procedure

- Chiamata a procedura: **CALL**
  Sintassi: **CALL** <nomeProc>
- Ritorno da procedura: **RET** (RETurn from procedure)
  Sintassi: **RET**

```
CALL sommaVettori     ; chiamata a procedura
    …
    …
sommaVettori PROC     ; inizio procedura
    …                 ; corpo della procedura
    …
    RET               ; ritorno da procedura
sommaVettori ENDP     ; fine procedura
```

58

Macro

- Una macro è un nome simbolico che il programmatore dà ad una sequenza di istruzioni
  - Aumenta la leggibilità del codice: si evita di ripetere lo stesso pezzo di codice più volte nel programma
- La definizione di una macro inizia con la direttiva **MACRO** e termina con la direttiva **ENDM**
- Sintassi:

```
<nomeMacro> MACRO [parametro1, parametro2, …]
… ; corpo della macro
…
ENDM
```

- Esempio

```
somma MACRO p1, p2, p3; definizione della macro
      MOV AL, p1
      MOV AH, p2
      ADD AL, AH
      MOV p3, AL
ENDM
…
…
somma 10, 20, BL; chiamata della macro
```

59

Macro e procedure

- Una macro è simile ad una procedura
  - Si differenzia da una procedura per l'effetto della chiamata
    - La chiamata di una procedura comporta un salto al codice della procedura (effettuata in fase di esecuzione)
    - La chiamata di una macro comporta una replica di tutto il codice della macro ad ogni occorrenza del nome della macro (in fase di compilazione)
- Rispetto alle procedure, l'uso delle macro implica
  - una maggiore efficienza in termini di tempo di esecuzione del codice
  - una minore efficienza in termini di compattezza del codice eseguibile
    - Le macro sono adatte per piccoli moduli

60

Espansione di macro

```
myMacro MACRO p1, p2, p3
      MOV AX, p1
      MOV BX, p2
      MOV CX, p3
ENDM

MyMacro 1,2,3
MyMacro 4,5,DX
```

```
MOV AX, 1
MOV BX, 2
MOV CX, 3
MOV AX, 4
MOV BX, 5
MOV CX, DX
```

61

Differenze tra macro e procedure

Procedure

- Direttive per la definizione
  - `PROC` e `ENDP`
- Chiamata: `call nomeProc`
- Il codice di una procedura è presente una sola volta nel codice eseguibile del programma
- Passaggio di parametri tramite stack o registri
- Una procedura termina con `RET`

Macro

- Direttive per la definizione
  - `MACRO` e `ENDM`
- Chiamata: `nomeMacro`
- Il codice di una macro è ripetuto tante volte nel codice eseguibile del programma
- Passaggio di parametri diretto
- Una macro non termina con `RET`

62

Interruzioni e funzioni del BIOS

- 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
  - operazioni di I/O (da tastiera, schermo, stampante)
  - lettura e scrittura su disco
  - …
- 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, detto BIOS (Basic Input Output System)
  - Il BIOS permette la gestione a basso livello di: video, tastiera, mouse, stampante, dischi, porte seriali e parallele

63

Interruzioni

- INT (INTerrupt): genera a livello software una chiamata alla routine di servizio (ISR) relativa ad un certo Interrupt

  Sintassi: **`INT`** `<numero_interruzione>`
  - Il numero di interruzione va da 0 a 255

  Esempio

  ```
  INT 21h
  ```

- IRET (Interrupt RETurn): restituisce il controllo al programma dopo un interrupt
  - Ogni ISR deve terminare con con un'istruzione IRET, al fine di rispristinare correttamente la situazione presente al momento in cui si è verificata l'interruzione.

  Sintassi: **`IRET`**

64

INT 10h: Funzioni del BIOS

AH	Servizio
00h	Modalità video (testo)
01h	Tipo del cursore
02h	Fornisce la posizione del cursore
03h	Lettura posizione del cursore
04h	Selezione della pagina video attiva
…	…
0Eh	Scrive un carattere sul terminale in modalità teletype

65

INT 10h: Funzioni del BIOS

- **INT 10h / AH = 00h**
  - Imposta la modalità video

AL = modalità video desiderata

Possibili valori:
00h – Modalità testo 40x25, 16 colori, 8 pagine
03h - Modalità testo 80x25, 16 colori, 8 pagine

66

INT 10h: Funzioni del BIOS

- **INT 10h** / **AH = 02h**
  - imposta la posizione del cursore

DH = numero di riga
DL = numero di colonna
BH = numero di pagina video (0..7)

67

INT 10h: Funzioni del BIOS

- **INT 10h** / **AH = 0Eh**
  - teletype output

Visualizza un carattere sullo schermo, facendo avanzare il cursore e scorrendo lo schermo se necessario.
La visualizzazione è fatta nella pagina video attiva

AL = codice ASCII del carattere da visualizzare

68

Esempio

Scrive la stringa "Ciao!" sul terminale in modalità teletype

```
   MOV AH, 00h ; servizio imposta dimensione video
   MOV AL, 00h ; prima dimensione: pagina video 40x25
   INT 10h     ; richiede il servizio

   MOV AH, 02h ; servizio imposta posizione cursore
   MOV DL, 10  ; colonna
   MOV DH, 5   ; riga
   MOV BH, 0   ; numero pagina video (0,..,7)
   INT 10h

; visualizza la stringa 'Ciao!' con INT 10h
   MOV    AH, 0Eh    ; imposta il servizio teletype output

   MOV    AL, 'C'
   INT    10h
   MOV    AL, 'i'
   INT    10h
   MOV    AL, 'a'
   INT    10h
   MOV    AL, 'o'
   INT    10h
   MOV    AL, '!'
   INT    10h
```

INT 21h: Interfacciamento con il sistema operativo

AH	**Servizio**
01h	Legge un carattere dall'input standard
02h	Scrive un carattere sull'output standard
09h	Visualizza una stringa di testo sull'output standard
0A	Legge una stringa dall'input standard
4C	Termina il programma restituendo il controllo al S.O.
…	…

70

INT 21h: Interfacciamento con il sistema operativo

- **INT 21h** / **AH=09h** – visualizza la stringa il cui indirizzo è memorizzato in DS:DX

DX=offset della variabile stringa

71

Esempio

Visualizza la stringa "Ciao!"

```
; Definizione della variabile stringa
   msg DB 'Ciao!'

; visualizza la stringa con INT 21h/ AH=9

; Carica in DX l'offset della stringa da visualizzare
   LEA DX, msg
; o equivalentemente MOV DX,OFFSET msg

; Imposta il servizio di visualizzazione
  MOV AH, 9

; richiede il servizio mediante interrupt
   INT 21h

; Termina il programma restituendo il controllo
; al sistema operativo
        MOV AH, 4Ch
        INT 21h
```

72

Esempio con procedura

```
; Definizione della variabile stringa
msg DB 'Ciao!$'

call Scrivi_ciao ; Chiama la procedura

mov ax,4Ch   ; termina il programma
int 21h

; dichiarazione della procedura
Scrivi_ciao PROC
mov dx,OFFSET msg
mov ah,9
int 21h
ret
Scrivi_ciao ENDP ; fine della procedura
```

73

Esempio con macro

```
; definizione di una macro con parametri
; che visualizza un carattere con INT 21h/ AH=2
ScriviCarattere MACRO carattere
   MOV AH, 2
   MOV DL, carattere
   INT 21h
ENDM

; Uso della macro
ScriviCarattere 'C'
ScriviCarattere 'i'
ScriviCarattere 'a'
ScriviCarattere 'o'

; Termina il programma
MOV AH, 4Ch
INT 21h
```

74

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 definizione di procedure
  - pseudo-istruzioni per la definizione di macro
  - pseudo-istruzioni per la definizione dei segmenti

75

Direttive di segmento

- Le direttive di segmento permettono di definire un segmento
  - La definizione di ogni segmento deve iniziare con la direttiva **`SEGMENT`** e deve terminare con la direttiva **`ENDS`**
- Sintassi:

```
<nomeSegment> SEGMENT
contenuto del segmento
<nomeSegment> ENDS
```

76

Segmenti di un Programma Assembly

- segmento contenente i dati
  - contiene le pseudo-istruzioni per l'allocazione delle variabili e le definizioni delle costanti
- segmento contenente lo stack
  - contiene una sola pseudo-istruzione per l'allocazione dello spazio sufficiente per lo stack del programma;
- segmento contenente il codice
  - Contiene le istruzioni del programma

77

Altre direttive importanti

- **#MAKE_COM#**
  - è una direttiva che informa l'assemblatore di produrre direttamente un file .COM (senza linking)
    - in un file .COM viene utilizzato un solo segmento che condivide codice e dati

78

Altre direttive importanti

- **ORG 100h**
  - Questa direttiva informa il traduttore che il file eseguibile (.com) del programma sarà caricato all'offset 100h (=256)
  - Saranno lasciati liberi i primi 256 (100h) byte di memoria prima di inserire il codice macchina del programma
    - Perché il sistema operativo occupa i primi 256 byte del code segment per memorizzare alcuni parametri
- Grazie a questa direttiva l'assemblatore può calcolare l'indirizzo corretto per tutte le variabili quando sostituisce i loro nomi (indirizzi simbolici) con i corrispondenti offset (indirizzi relativi numerici).
- **Questo vale solo per i file .com**.
  - I file .exe, cioè quelli generati dopo il linking, sono caricati all'offset **0000**

79

Struttura generale di un programma Assembly

Per ottenere eseguibili .com

```
; direttive iniziali
#MAKE_COM#
ORG 100h

; corpo del programma

…


; istruzioni per chiudere il programma
MOV AH, 4Ch
INT 21h

; definizione di variabili e costanti
…
```

80