

# I processori

Matteo Sonza Reorda

Politecnico di Torino  
Dip. di Automatica e Informatica

1

## Introduzione

**Un processore è un dispositivo che compie 2 tipi di operazioni:**

- **esegue istruzioni (contenute in memoria)**
- **interagisce con il mondo esterno (attraverso opportune interfacce).**

2

# Gestione di eventi esterni

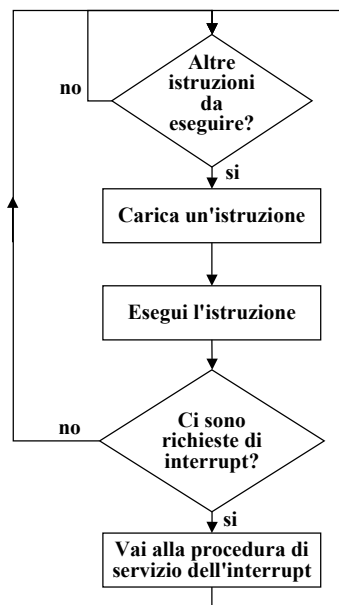
Il processore partecipa normalmente a tutte le operazioni che si svolgono sul bus del sistema:

- ad alcune partecipa direttamente (ad esempio facendo accesso alla memoria o ai dispositivi periferici)
- ad altre partecipa semplicemente prendendo parte al meccanismo di arbitraggio del bus.

Particolare importanza ha il meccanismo dell'*interrupt*, attraverso il quale un dispositivo esterno richiede l'esecuzione di una procedura di servizio.

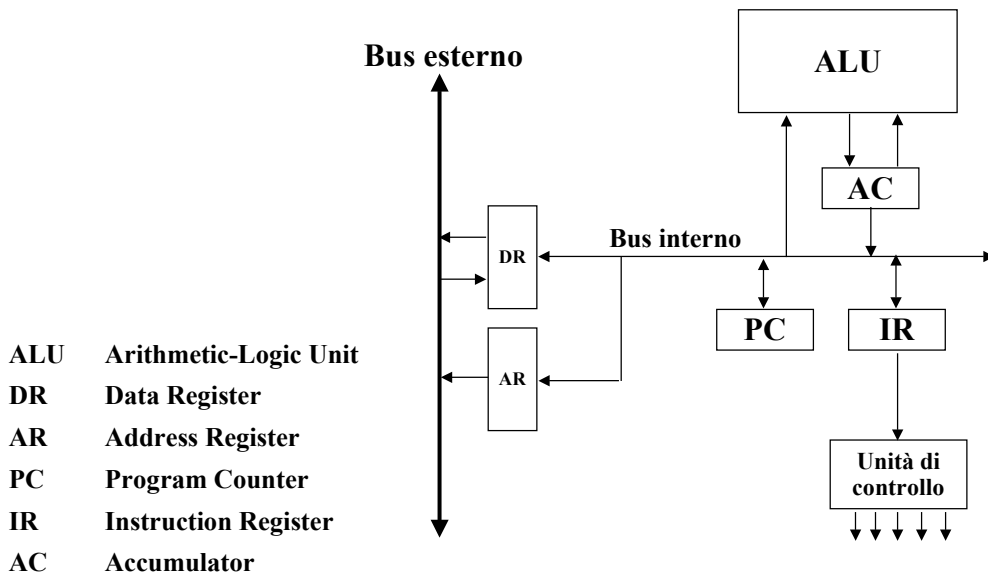
3

## Comportamento di un processore



4

# Architettura minima di una CPU



5

## Istruzioni

L'esecuzione di ciascuna istruzione macchina si compone di 2 fasi:

- *fetch*: il codice dell'istruzione viene letto dalla memoria
- *execute*: il codice viene prima decodificato, poi eseguito. Questo comporta normalmente l'accesso ad uno o più operandi, l'esecuzione di una operazione su di essi, la scrittura del risultato.

La combinazione delle due fasi si dice *ciclo di istruzione*.

6

# Registri

**Il tempo per accedere alla memoria è normalmente superiore al tempo necessario alla CPU per processare i dati.**

**L'accesso alla memoria rappresenta quindi un *collo di bottiglia* per le prestazioni delle CPU.**

**Per questa ragione all'interno della CPU sono presenti alcune celle di memoria particolarmente veloci, note come *registri*.**

**Ove possibile le operazioni vengono svolte utilizzando i registri per contenere operandi e risultato.**

7

## Esempio

**Tutti i processori Intel (a partire dall'8086) contengono i seguenti registri (aventi parallelismo 16 bit):**

- **AX, BX, CX, DX**
- **SI, DI**
- **DS, ES, CS, SS**
- **SP, BP.**

8

# Microistruzioni

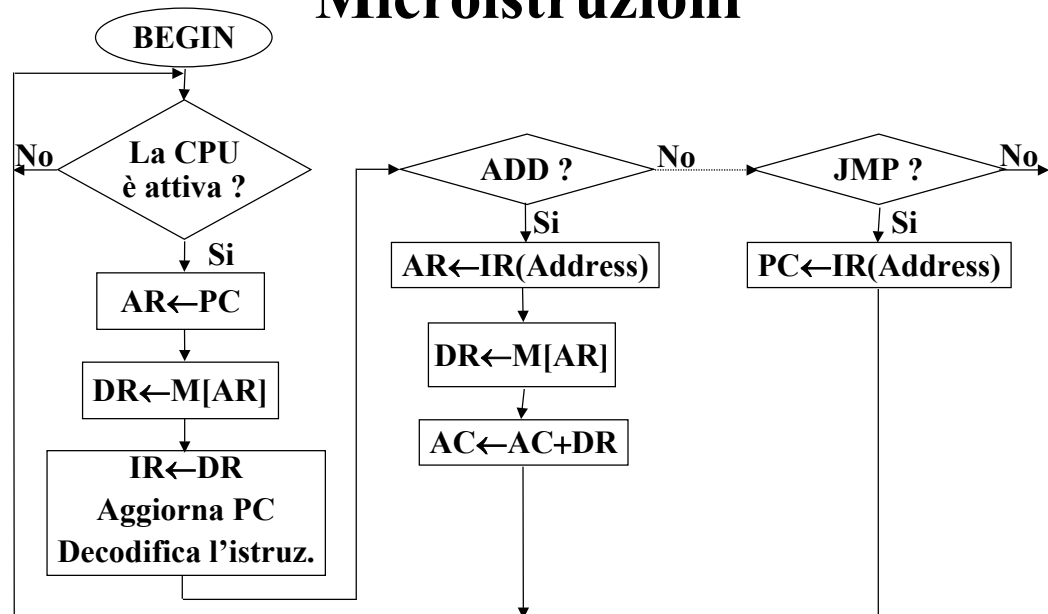
L'esecuzione delle 2 fasi di un'istruzione corrisponde all'attivazione di un certo numero di *microistruzioni* (corrispondenti ad esempio al trasferimento di un valore da un registro ad un altro).

Il tempo per l'esecuzione di una microistruzione è definito come *CPU cycle time* ( $t_{CPU}$ ) e spesso coincide con il ciclo di clock del processore.

Nei processori tradizionali ciascuna istruzione richiede quindi un tempo pari ad un certo numero di colpi di clock.

9

## Microistruzioni



10

# Input/Output

L'accesso ai dispositivi di I/O viene realizzato in maniera simile a quanto fatto per l'accesso alla memoria: ad ogni dispositivo di I/O è associato un *dispositivo di interfaccia*, che contiene un certo numero di registri (anche detti *porte*) visibili alla CPU attraverso il bus.

Ad ogni registro di interfaccia corrisponde un indirizzo, sul quale la CPU può eseguire opportune operazioni di lettura o scrittura.

11

## Esempio

Si consideri la connessione di una *stampante*, realizzata normalmente attraverso un'interfaccia parallela.

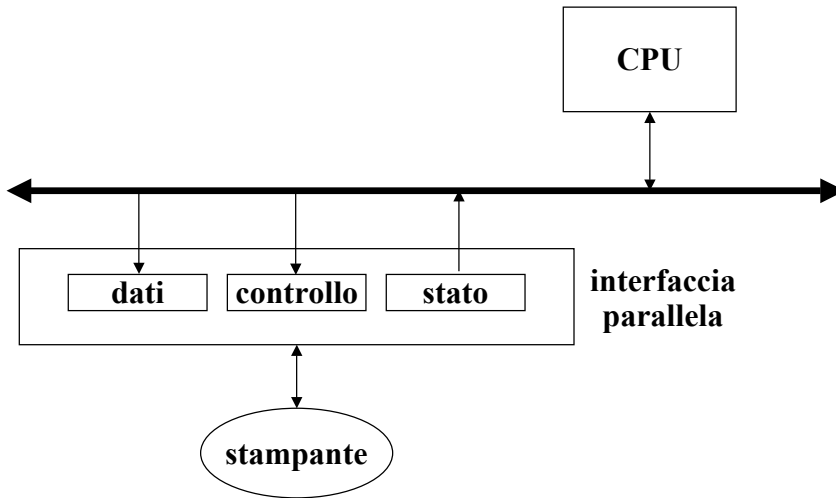
La stampante sarà vista dalla CPU come 3 registri corrispondenti ad altrettanti indirizzi:

- un registro per i *dati* (output)
- un registro per le informazioni di *controllo* (output)
- un registro per le informazioni di *stato* (input).

Per far eseguire delle operazioni alla stampante la CPU eseguirà delle scritture o letture sui 3 registri.

12

# Esempio



13

# Estensioni

Rispetto allo schema di base di un processore sono normalmente adottate alcune estensioni relative ai seguenti punti:

- registri
- unità aritmetiche
- status register
- stack.

14

# Registri

Il numero dei registri influenza fortemente le prestazioni di un processore. Quando sono molti si parla di *register-file* o *scratch-pad memory*.

È possibile che per ogni registro venga imposto un particolare uso; si possono avere

- registri dati
- registri indice
- registri contatore.

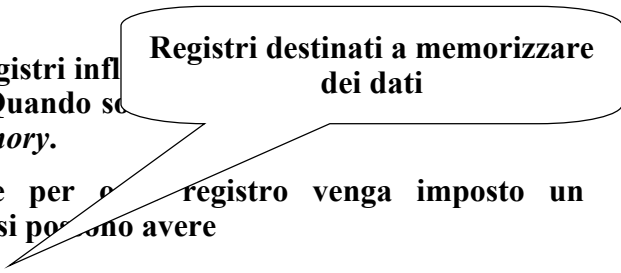
15

# Registri

Il numero dei registri influenza le prestazioni di un processore. Quando sono molti si parla di *scratch-pad memory*.

È possibile che per ogni registro venga imposto un particolare uso; si possono avere

- registri dati
- registri indice
- registri contatore.



Registri destinati a memorizzare dei dati

16



# Registri

Il numero dei registri influenza fortemente le prestazioni di un processore. Quando sono molti si parla di *register-file* o *scratch-pad memory*.

È possibile che per ogni particolare uso; si possono avere

- registri dati
- registri indice
- registri contatore.

Registri destinati a memorizzare degli indirizzi in memoria, corrispondenti a celle in cui si trovano gli operandi

17

# Registri

Il numero dei registri influenza fortemente le prestazioni di un processore. Quando sono molti si parla di *register-file* o *scratch-pad memory*.

È possibile che per ogni registro venga imposto un particolare uso; si possono avere

- registri dati
- registri indice
- registri contatore.

Registri destinati ad eseguire operazioni di conteggio

18

# Unità aritmetiche

Possono essere più o meno potenti, a seconda delle operazioni che implementano:

- somma e sottrazione in fixed-point
- moltiplicazione e divisione in fixed-point
- operazioni in floating-point.

19

## Registro di stato

È un registro particolare contenente due gruppi di bit (o *flag*) denominati

- Bit di stato
- Bit di controllo.

20

# Bit di stato

I bit di stato sono

- forzati dal verificarsi di particolari condizioni (*carry*, *overflow*, risultato positivo o negativo, risultato nullo, parità) a seguito dell'esecuzione delle istruzioni
- testati dalle istruzioni di salto condizionato (e da eventuali altre).

In questo modo si implementano a livello di codice assembler i salti condizionati.

21

## Esempio

Codice C

```
if(a == b)
    a = 1;
```

Codice assembler 80x86

```
MOV    AX, a
MOV    BX, b
SUB    AX, BX
JNE    dopo
MOV    AX, 1
dopo:  MOV    a, AX
```

22

## Esempio

Codice C

```
if (a ==  
    a = 1;
```

Forza a 0 o 1 il flag  
di zero, a seconda  
che il risultato sia  
nullo o meno

Codice assembler 80x86

```
MOV  AX, a  
MOV  BX, b  
SUB  AX, BX  
JNE  dopo  
MOV  AX, 1  
dopo:MOV  a, AX
```

Salta se il flag di  
zero vale 0

23

## Bit di controllo

Controllano il comportamento del processore rispetto ad un determinato aspetto.

Ad esempio

- Il *bit di interrupt* controlla la sensibilità agli interrupt
- Il *bit di trap* fa funzionare o meno il processore in modalità debug.

Tali bit sono forzati a 0 o 1 da apposite istruzioni.

24

# Stack

**È una struttura LIFO allocata in memoria.**

**Molti processori possiedono:**

- **un registro special-purpose, denominato *Stack Pointer* (SP) che punta all'elemento top**
- **due istruzioni (PUSH e POP) che manipolano implicitamente lo SP.**

25

## Stack: usi

**Lo stack viene usato:**

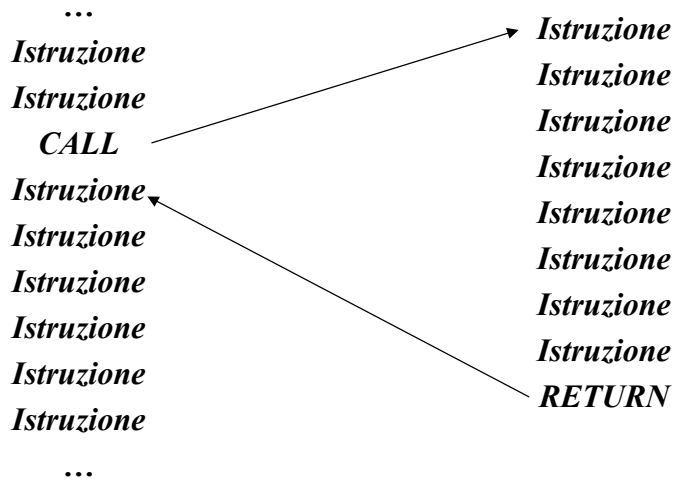
- **per le chiamate alle procedure (normali o di servizio dell'interrupt)**
- **per il passaggio di parametri alle procedure**
- **per la memorizzazione di variabili locali.**

26

# Chiamate a procedura

Progr. princ.

Procedura

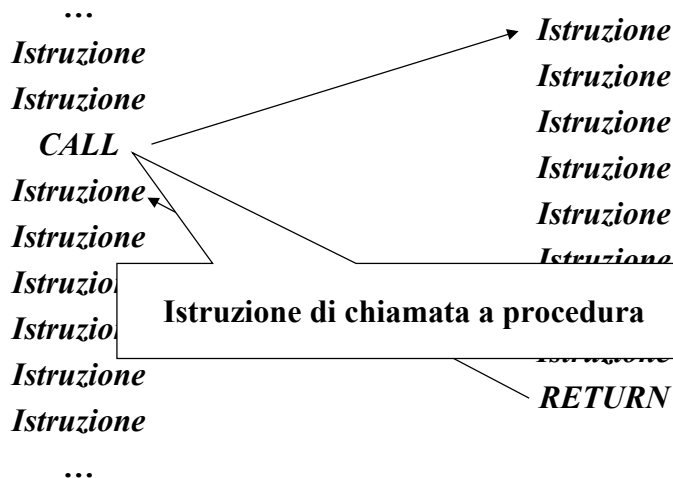


27

# Chiamate a procedura

Progr. princ.

Procedura

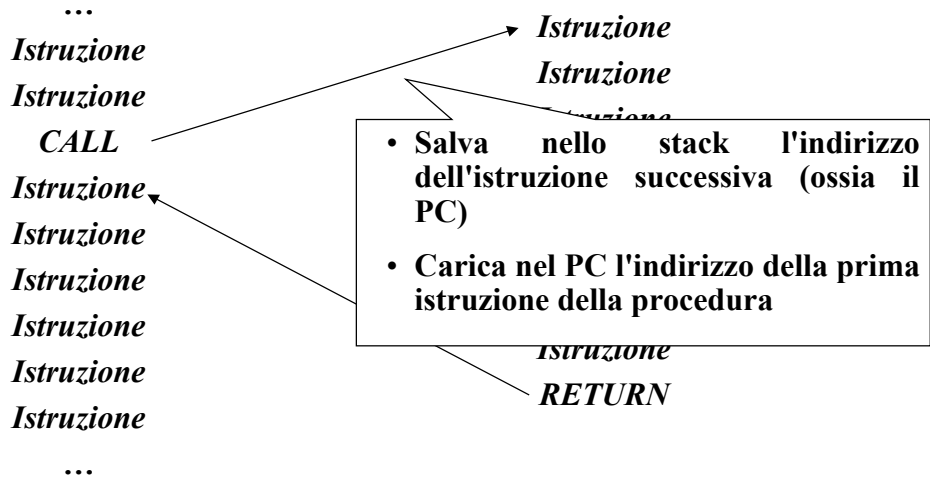


28

# Chiamate a procedura

Progr. princ.

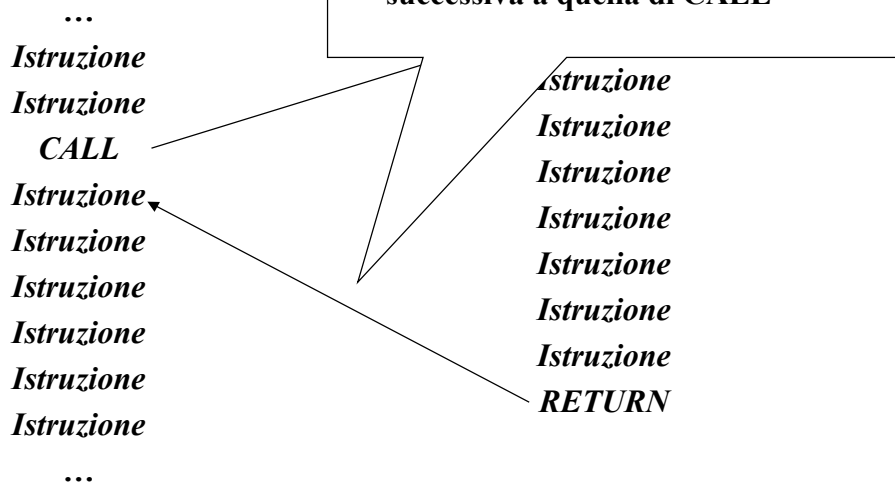
Procedura



29

# Chiamate a procedura

Progr. princ.



30

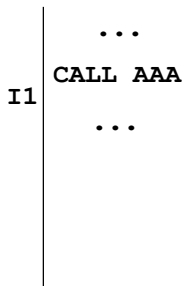
# Vantaggi

L'uso dello stack nella gestione delle procedure presenta il vantaggio di permettere l'*annidamento* delle procedure.

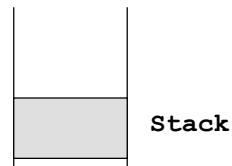
In tal caso ciascuna istruzione **RETURN** accede all'indirizzo di ritorno salvato nello stack dall'ultima istruzione **CALL** eseguita.

31

## Esempio



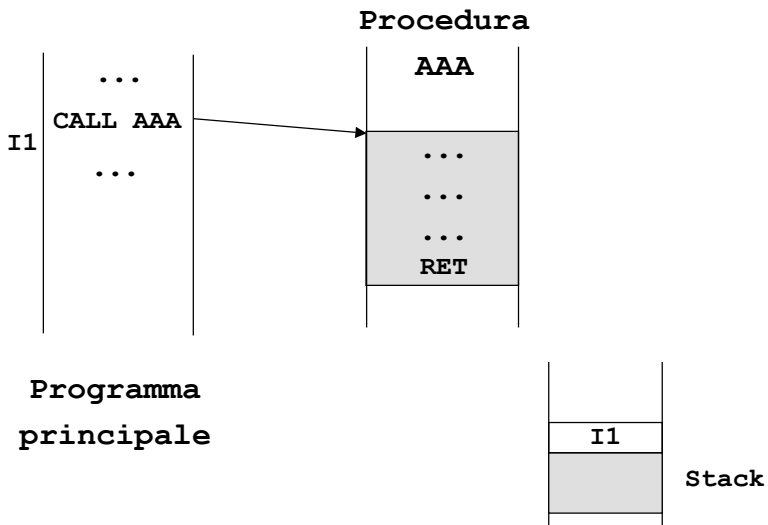
Programma  
principale



32

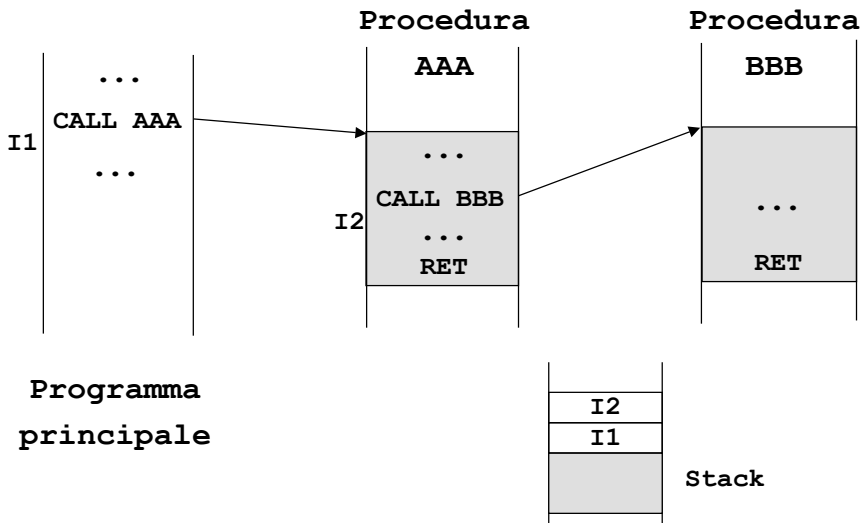


# Esempio



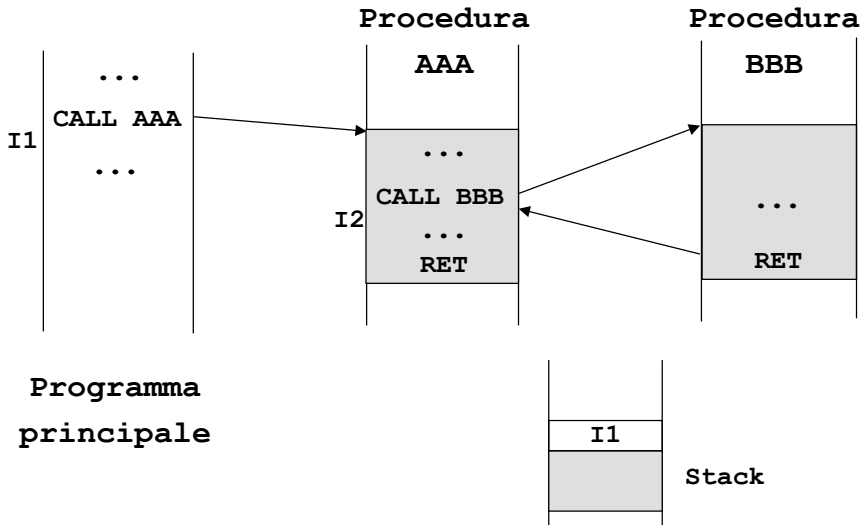
33

# Esempio



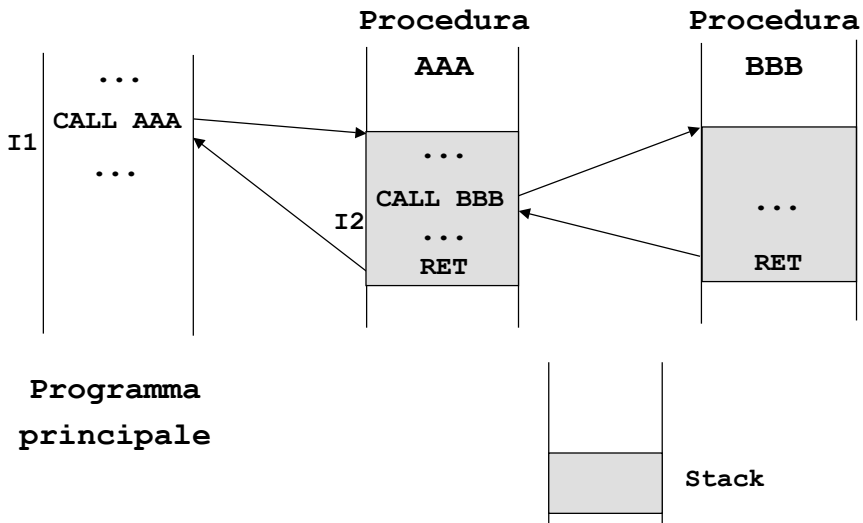
34

# Esempio



35

# Esempio



36

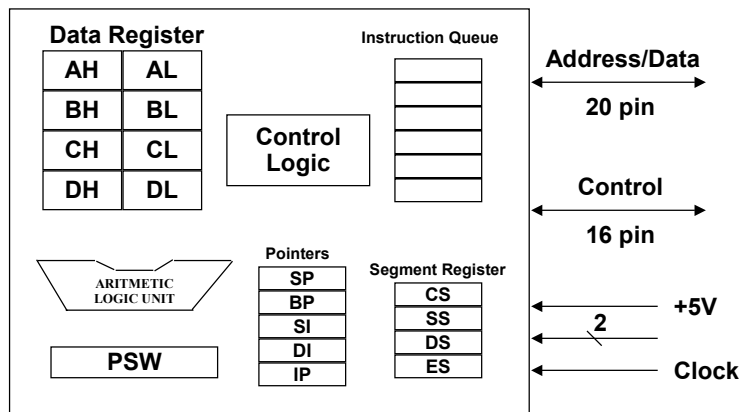
# Soluzioni alternative

In taluni casi (ad esempio nei sistemi *IBM 360-370*) il salvataggio dell'indirizzo di ritorno veniva effettuato utilizzando una particolare locazione di memoria.

In altri casi (ad esempio nell'architettura *PowerPC*) si usa un particolare registro.

37

## Architettura 8086



38

# Calcolo degli indirizzi

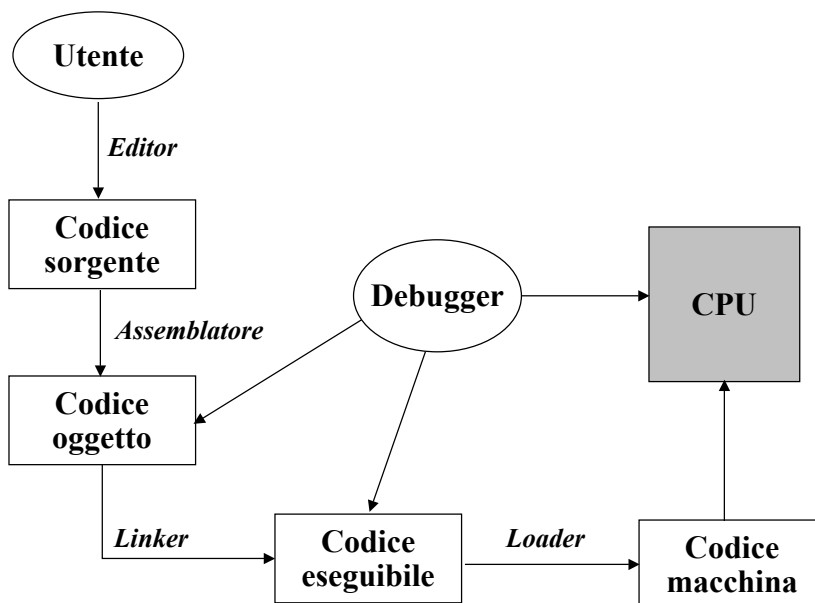
Ogni volta che l'8086 deve generare un indirizzo da mettere sull'A-bus (*physical address*), esso esegue una operazione di somma tra il contenuto di un registro puntatore oppure di BX (*effective address* o *offset*) ed il contenuto di un registro di segmento (*segment address*).

La somma avviene dopo aver moltiplicato per 16 (shift di 4 posizioni) il contenuto del registro di segmento:

$$\begin{array}{rcl} \boxed{16 \text{ bit}} & & \text{Effective Address} \\ + & & \\ \boxed{16 \text{ bit}} \quad \boxed{0000} & & \text{Segment Address} * 16 \\ = & & \\ \boxed{20 \text{ bit}} & & \text{Physical Address} \end{array}$$

39

# Ciclo di vita di un programma



40

# Assemblaggio

**Il file sorgente viene trasformato in file oggetto tramite:**

- **rimozione dei commenti**
- **associazione di ciascuna variabile simbolica ad una adeguata locazione di memoria**
- **traduzione in codice macchina di ciascuna istruzione.**

**Alcune operazioni non possono ancora essere completate, ad esempio quelle che riguardano:**

- **le variabili definite in altri moduli**
- **le procedure definite in altri moduli.**

41

## Link

**Procede alla creazione del file eseguibile a partire da più file oggetto.**

**Provvede a verificare la correttezza dei richiami a variabili e procedure definite in altri moduli e a generare gli indirizzi opportuni.**

**Il file prodotto non è di solito immediatamente eseguibile in quanto**

- **risiede su disco anziché in memoria**
- **deve essere indipendente dalla posizione in memoria in cui verrà caricato.**

42

# Loader

**Fa parte del Sistema Operativo.**

**Provvede a**

- **reperire il file eseguibile su disco**
- **caricarlo in memoria, eseguendo le eventuali modifiche rese necessarie dopo che è stata decisa la sua posizione in memoria**
- **fare in modo che il processore inizi l'esecuzione del programma.**

43

# Debugger

**Interagisce con i processi di assemblaggio, link ed esecuzione, permettendo al programmatore di disporre di funzionalità quali:**

- **breakpoint**
- **esecuzione passo passo**
- **accesso a variabili e registri (in lettura e scrittura)**
- **...**

44

# Linguaggi assembler

**Ciascun processore ha il proprio linguaggio assembler (in termini di codice macchina).**

**Processori diversi appartenenti ad una stessa famiglia possono riconoscere lo stesso codice macchina.**

45

## Caratteristiche di un linguaggio assembler

- Operazioni permesse
- Tipi di dato
- Formato delle istruzioni
- Registri utilizzabili
- Modi di indirizzamento
- Facilità d'uso
- ...

46

# Formato delle istruzioni

A livello di codice macchina, le istruzioni sono formate da 2 parti:

- il codice dell'operazione da svolgere (*opcode*)
- le informazioni sugli operandi.

A seconda del numero tipico di operandi ammesso in ogni istruzione si possono avere processori a 0, 1, 2 o 3 operandi.

opcode	operando 1	operando 2
--------	------------	------------

47

## Modi di indirizzamento

Rappresentano i modi attraverso i quali è specificato il tipo ed il valore o la posizione di ciascun operando.

I più frequenti sono:

- indirizzamento tramite registro
- indirizzamento immediato
- indirizzamento con accesso in memoria.

48



# Accesso in memoria

Laddove compare come operando un indirizzo, questo può essere specificato in 2 modi:

- indirizzamento diretto
- indirizzamento indiretto

Ciascun indirizzo può essere espresso in due modi:

- assoluto
- relativo.

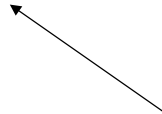
49

## Indirizzamento *tramite registro*

L'operando è contenuto in un registro.

### Esempio

MOV AX, 99



Indirizzamento  
tramite  
registro

50

# Indirizzamento *immediato*

Il valore dell'operando è specificato direttamente nell'istruzione.

## Esempio

MOV AX, 99

Indirizzamento  
immediato

Il valore dell'operando è memorizzato nel codice macchina dell'istruzione.

51

# Indirizzamento *diretto*

Nell'istruzione è specificato l'indirizzo della cella di memoria ove si trova l'operando.

## Esempio

MOV xxxx, 57

xxxx

57

Indirizzamento  
diretto

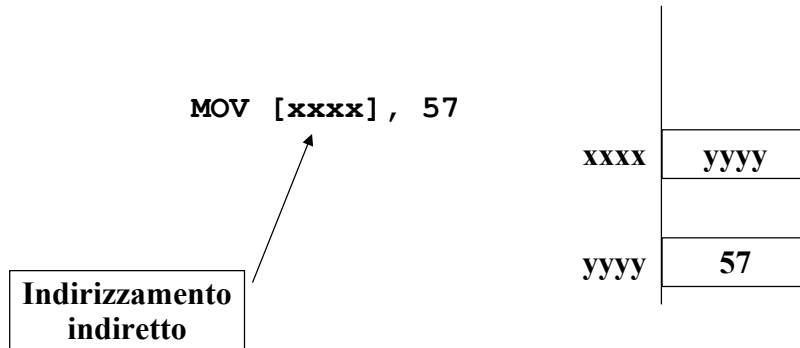
Il valore dell'indirizzo è memorizzato nel codice macchina dell'istruzione.

52

# Indirizzamento *indiretto*

Nell'istruzione è specificato l'indirizzo di una cella di memoria, in cui è scritto l'indirizzo dell'operando.

## Esempio

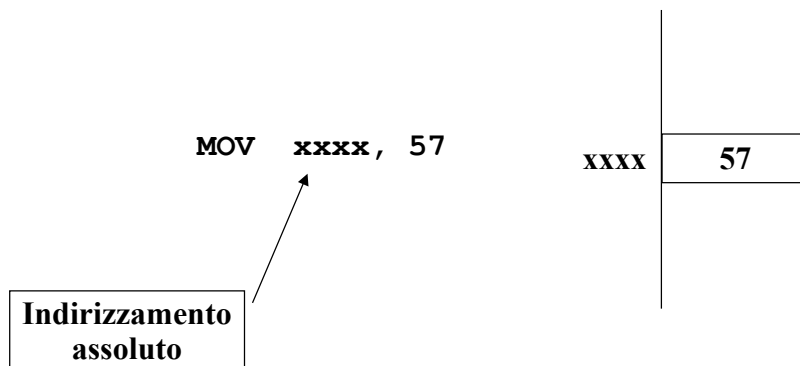


53

# Indirizzamento *assoluto*

L'indirizzo dell'operando è specificato per esteso nell'istruzione.

## Esempio

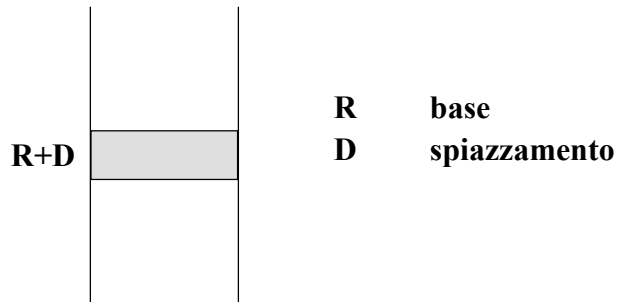


54

# Indirizzamento *relativo*

L'istruzione specifica dove si trova un valore, che corrisponde ad uno *spiazzamento* rispetto ad un certo indirizzo di riferimento (*base*).

## Esempio



Il valore dello spiazzamento è memorizzato nel codice macchina dell'istruzione.

55

## Esempio

L'assembler 80x86 include l'indirizzamento *base-indexed*:

```
MOV [BX][SI], 57
```

Tale modo di indirizzamento si presta molto bene a scandire vettori.

56

# Indirizzamento relativo (II)

## Vantaggi:

- minor lunghezza dell'istruzione
- rilocabilità
- facilità di scansione di vettori.

## Svantaggi:

- complessità dell'hardware
- maggiore tempo di esecuzione.

57

# Ortogonalità

È una caratteristica di alcuni linguaggi Assembler, tale per cui ogni operando di ogni istruzione può essere espresso attraverso uno qualsiasi dei modi di indirizzamento.

L'ortogonalità in generale riduce i costi di programmazione, ma aumenta quelli dell'hardware.

58

# Tipi di istruzioni

Ogni processore ha il suo linguaggio Assembler.

Esistono tuttavia forti somiglianze tra i linguaggi Assembler dei vari processori.

Le caratteristiche di un buon linguaggio sono:

- *completezza*
- *efficienza*
- *regolarità* (ortogonalità)
- *compatibilità* rispetto a precedenti versioni.

59

## Classi di istruzioni

- Istruzioni di trasferimento dati
- Istruzioni aritmetiche
- Istruzioni logiche
- Istruzioni per il controllo del programma
- Istruzioni di Input/Output.

60

# **Il linguaggio assembler Intel**

**Verranno elencate le principali istruzioni che fanno parte del linguaggio assembler supportato dai processori Intel, a partire dall'8086.**

**61**

## **Istruzioni di trasferimento dati**

**MOV**

**XCHG**

**PUSH**

**POP**

**62**

# Istruzioni aritmetiche

ADD

SUB

MUL

DIV

INC

DEC

NEG

63

# Istruzioni logiche

AND

OR

XOR

NOT

SHL, SHR

RCL, RCR

64



# Istruzioni per il controllo del programma

**JMP**

**Jxx**

**CALL**

**RET**

**CMP**

65

# Istruzioni per il controllo del programma

**JMP**

**Jxx**

**CALL**

**RET**

**CMP**

**Ad esempio:**

**JE**

**JNE**

**JG**

**JGE**

**JL**

**JLE**

**JC**

**JNC**

66

# Istruzioni di I/O

IN  
OUT

67

## Istruzioni per il controllo del processore

STI, CLI

HALT

WAIT

NOP

68

# Esempio

**Frammento di codice che esegue la somma degli elementi di un vettore di 100 interi:**

```
...
MOV     SI, 0
MOV     AX, 0
MOV     BX, vett
MOV     CX, 100
ciclo:  ADD     AX, [BX][SI]
        INC     SI
        INC     SI
        DEC     CX
        CMP     CX, 0
        JNE     ciclo
        MOV     somma, AX
```

69

## 8086: formato delle istruzioni macchina

**Non esistono regole per la traduzione delle istruzioni dal linguaggio sorgente al codice macchina.**

**Per ogni istruzione si hanno regole specifiche.**

**È tuttavia possibile fare alcune considerazioni generali.**

# Numero di byte

Le istruzioni macchina dell'8086 hanno una dimensione che varia da 1 a 6 byte.

Il formato prevede:

- 1 o 2 byte per specificare il codice operativo ed il modo di indirizzamento
- da 0 a 4 byte aggiuntivi, contenenti eventuali indirizzi in memoria o dati immediati.

71

## Primo byte

Oltre al Codice Operativo, il primo byte può contenere alcuni bit con un significato particolare:

- **W**: se vale 0 l'istruzione lavora sui byte, se vale 1 lavora sulle word
- **D**: nelle istruzioni con 2 operandi uno di questi deve normalmente essere un registro; a seconda del valore di D, il registro corrisponde all'operando sorgente (D=0) o destinazione (D=1)
- **S**: compare nelle istruzioni che prevedono un operando immediato; se questo è su una word (W=1) ma il MSB è nullo, è possibile rappresentare il solo LSB ponendo S=1.

72

# Secondo byte

In alcune istruzioni il secondo byte è ancora destinato a specificare il codice operativo ed il modo di indirizzamento.

Tale byte può assumere 2 formati; il primo viene utilizzato per istruzioni con un solo operando, il secondo per istruzioni con due operandi; in tal caso uno dei due corrisponde al registro specificato dal campo REG:



73

## REG

<i>Codice</i>	<i>Registro</i>	
	W=1	W=0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

74

# MOD e R/M

Il sottocampo MOD specifica il *modo di indirizzamento* e se nell'istruzione è presente un *displacement*.

Il sottocampo R/M specifica se si usa un registro oppure una locazione in memoria.

75

## MOD

MOD	<u>funzione</u>
00	nessun displacement
01	displacement su 8 bit
10	displacement su 16 bit
11	R/M è un registro

76

# REG e R/M

Il sottocampo R/M (quando MOD = 11) specifica un registro:

	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

77

## R/M

Se il campo MOD contiene uno dei valori 00, 01 o 10, il sottocampo R/M assume un significato diverso, ossia specifica il tipo di indirizzamento:

000	DS: [BX+SI]
001	DS: [BX+DI]
010	SS: [BP+SI]
011	SS: [BP+DI]
100	DS: [SI]
101	DS: [DI]
110	SS: [BP]
111	DS: [BX]

78

# Esempi

Verranno considerati 7 casi esemplificativi di istruzioni, aventi formato tra loro diverso.

79

## Istruzioni su 1 byte (I)

Questo formato è tipico delle istruzioni senza operandi.

Formato:



Esempio:

L'istruzione NOP è codificata come

**1001 0000**

80



## Istruzioni su 1 byte (II)

Questo formato è tipico delle istruzioni con un solo operando, corrispondente ad un registro.

Formato:



Esempio:

L'istruzione PUSH, qualora lavori su un registro, è codificata come

01 010 reg

81

## Istruzioni su 2 byte (I)

Questo formato è tipico delle istruzioni con due operandi, corrispondenti entrambi ad un registro.

Formato:



Esempio:

L'istruzione MOV AX, BX è codificata come

10 00 10 1 1      11 000 011

82

## Istruzioni su 2 byte (II)

Questo formato è tipico delle istruzioni con due operandi, di cui è un registro, e l'altro risiede in memoria, e ad esso si accede tramite indirizzamento indiretto con registro.

**Formato:**



**Esempio:**

L'istruzione **MOV AX, [BX]** è codificata come

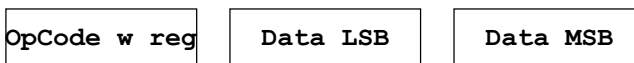
10 00 10 1 1          00 000 111

83

## Istruzioni su 3 byte

Questo formato è tipico delle istruzioni con due operandi, di cui è un registro, e l'altro è un valore immediato.

**Formato:**



**Esempio:**

L'istruzione **MOV AX, imm.** è codificata su 3 byte.

**Nota**

Qualora i bit *s* e *w* valgano 11, il dato può essere rappresentato su un solo byte ed esteso poi a 2 byte in fase di esecuzione.

84

## Istruzioni su 4 byte (I)

Questo formato è tipico delle istruzioni con due operandi, di cui è un registro, e l'altro risiede in memoria, e ad esso si accede specificando un offset.

**Formato:**



**Esempio:**

L'istruzione **MOV AX, var** è codificata come

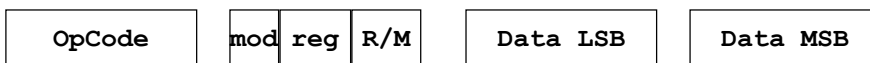
**10 00 10 11    00 000 110    offset LSB    offset MSB**

85

## Istruzioni su 4 byte (II)

Questo formato è tipico delle istruzioni con due operandi, di cui è un registro, e l'altro è un operando immediato.

**Formato:**



**Esempio:**

L'istruzione **MOV [BX], imm.** è codificata su 4 byte.

86

# Istruzioni su 6 byte

Questo formato è tipico delle istruzioni con due operandi, di cui uno risiede in memoria, e l'altro è un operando immediato.

**Formato:**



**Esempio:**

L'istruzione **MOV var, imm.** è codificata su 6 byte.

87

## Esempio

opcode D WMOD REG R/M

100010 1 1 11 101 100

Opcode = MOV

D = trasferimento a REG

W = word

MOD = R/M è un registro

REG = BP

R/M = SP

***MOV BP, SP***

88

# Esempio

```
opcode  W  MOD REG R/M LOW DISP HIGH DISP
1100011 1  10 000 111  00000000 00001000
```

LOW DATA HIGH DATA

00110100 00010010

Opcode = MOV immediata

W = Word

MOD = 16 bit displacement

REG = 000 (non usata nell'indirizzamento immediato)

R/M = DS:[BX]

Displacement = 1000H

Data = 1234H

89

*MOV WORD PTR [BX+1000H], 1234H*

## Tempi di esecuzione

Il tempo di esecuzione di un'istruzione dipende dalla frequenza di clock, dal tipo dell'istruzione, dalla posizione degli operandi (in un registro, immediato, in memoria), dall'allineamento degli operandi.

Il tempo richiesto può essere così scomposto:

- tempo per il calcolo dell'EA dell'eventuale operando in memoria
- tempo per l'accesso a tale operando
- tempo per l'esecuzione.

# Clock

I manuali forniscono per ciascuna istruzione il numero di colpi di clock necessari. Per ottenere il tempo, tale numero va moltiplicato per il periodo del clock.

91

## Calcolo dell'EA

<i>Indirizzamento</i>	<i># clock</i>
Diretto	6
Register Indirect	5
Register Relative	9
Based Indexed	
(BP)+(DI)	7
(BX)+(SI)	7
(BP)+(SI)	8
(BX)+(DI)	8
Based Indexed Relative	
(BP)+(DI)+disp	11
(BX)+(SI)+disp	11
(BP)+(SI)+disp	12
(BX)+(DI)+disp	12

92

# Tempo per l'accesso all'operando

Se l'operando in memoria è una word posta ad un indirizzo dispari, l'8086 richiede 4 colpi di clock in più per ogni accesso in memoria.

Se quindi l'operando in memoria coincide con il risultato, si dovranno aggiungere 8 colpi di clock.

93

## Esempio 1

**ADD AX, BX**

**# colpi di clock richiesti: 3**

**frequenza di clock    5 MHz**

**tempo richiesto        600 nsec**

94

## Esempio 2

**ADD AX, [BX+SI]+4**

# colpi di clock richiesti per l'esecuzione	9
# colpi di clock richiesti per il calcolo dell'EA	11
# colpi di clock aggiuntivi se l'operando è ad un indirizzo dispari	4
frequenza di clock	5 MHz
tempo richiesto	$(9+11+4)*200\text{nsec}=4.8\text{ }\mu\text{sec}$

95

## Esempio 3

**ADD [BX+SI]+4, AX**

# colpi di clock richiesti per l'esecuzione	9
# colpi di clock richiesti per il calcolo dell'EA	11
# colpi di clock aggiuntivi se l'operando è ad un indirizzo dispari	4
frequenza di clock	5 MHz
tempo richiesto	$(9+11+4+4)*200\text{nsec}=5.6\text{ }\mu\text{sec}$

96



# Esempi di programmi in assembler 80x86

- Scrittura di un valore in memoria
- Somma di due valori
- Somma degli elementi di un vettore
- Lettura e visualizzazione di un vettore di caratteri
- Ricerca del carattere minimo

97

## Scrittura di un valore in memoria

```
.MODEL small
.STACK
.DATA
VAR    DW    ?
.CODE
.STARTUP
MOV    VAR, 0
.EXIT
END
```

98

# Somma di due valori

```
.MODEL small
.STACK
.DATA
OPD1    DW    10
OPD2    DW    24
RESULT  DW    ?
.CODE
.STARTUP
MOV  AX, OPD1
ADD  AX, OPD2
MOV  RESULT, AX
.EXIT
END
```

99

# Somma degli elementi di un vettore (I)

```
.MODEL SMALL
.STACK
.DATA
VETT    DW    5, 7, 3, 4, 3
RESULT  DW    ?
.CODE
.STARTUP
MOV  AX, 0
ADD  AX, VETT
ADD  AX, VETT+2
ADD  AX, VETT+4
ADD  AX, VETT+6
ADD  AX, VETT+8
MOV  RESULT, AX
.EXIT
END
```

100

# Somma degli elementi di un vettore (II)

```
DIM      EQU 15
          .MODEL      small
          .STACK
          .DATA
VETT     DW 2, 5, 16, 12, 34, 7, 20, 11, 31, 44, 70, 69, 2, 4, 23
RESULT   DW  ?
          .CODE
          .STARTUP
MOV AX, 0          ; azzera il registro AX
MOV CX, DIM        ; carica in CX la dimensione
                  ; del vettore
MOV DI, 0          ; azzera il registro DI
```

101

```
lab:     ADD AX, VETT[DI]      ; somma ad AX l'i-esimo elemento
                                      ; di VETT
          ADD DI, 2            ; passa all'elemento successivo
          DEC CX               ; decrementa il contatore
          CMP CX, 0            ; confronta il contatore con 0
          JNZ lab              ; se diverso da 0 salta
          MOV RESULT, AX       ; altrimenti scrivi il risultato
          .EXIT
          END
```

102

# Lettura e visualizzazione di un vettore di caratteri

```
DIM      EQU 20
.MODEL   small
.STACK
.DATA
VETT     DB  DIM DUP(?)
.CODE
.STARTUP
MOV  CX, DIM          ; carica in CX la dimensione
                        ; del vettore
MOV  DI, 0            ; azzerà il registro DI
MOV  AH, 1            ; predisposizione del registro AH
```

103

```
lab1:  INT  21H          ; lettura di un carattere
        MOV  VETT[DI], AL ; memorizzaz. del carattere letto
        INC  DI          ; passa all'elemento successivo
        DEC  CX          ; decrementa il contatore
        CMP  CX, 0       ; confronta il contatore con 0
        JNZ  lab1        ; se diverso da 0 salta
        MOV  CX, DIM
        MOV  AH, 2       ; predisposizione del registro AH
lab2:  DEC  DI          ; passa all'elemento precedente
        MOV  DL, VETT[DI]
        INT  21H          ; visualizzazione di un carattere
        DEC  CX          ; decrementa il contatore
        CMP  CX, 0       ; confronta il contatore con 0
        JNZ  lab2        ; se diverso da 0 salta
        .EXIT
END
```

104

# Ricerca del carattere minimo

```
.MODEL      small
.STACK
DIM EQU 20
.DATA
TABLE DB DIM DUP(?)
.CODE
.STARTUP
MOV CX, DIM
LEA DI, TABLE
MOV AH, 1          ; lettura
lab1: INT 21H
MOV [DI], AL
INC DI
LOOP lab1          ; ripeti per 20 volte
MOV CL, 0FFH       ; inizializzazione di CL
MOV DI, 0
```

105

```
ciclo: CMP CL, TABLE[DI]    ; confronta con il minimo attuale
JB dopo
MOV CL, TABLE[DI]          ; memorizza il nuovo minimo
dopo: INC DI
CMP DI, DIM
JB ciclo
output: MOV DL, CL
MOV AH, 2
INT 21H                     ; visualizzazione
.EXIT
END
```

106