# Architettura degli Elaboratori 1

linguaggio macchina

Alessandro Memo Gennaio '03

## processore x86 base

| | | | |
|---|---|---|---|
| AX | AH | AL | accumulator |
| BX | BH | BL | base |
| CX | CH | CL | counter |
| DX | DH | DL | data |

4 registri base a 16 bit (AX, BX, CX e DX) ad utilizzo generale e preferenziale

Ciascun registro è composto da due unità ad 8 bit contraddistinte dalla lettera H (High) ed L (Low) 8

## processore x86 base

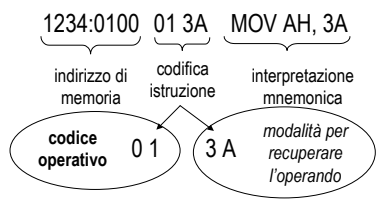
| | |
|---|---|
| SP | stack pointer |
| BP | base pointer |
| SI | source index |
| DI | destination index |

IP — instruction pointer

PSW — process status word

| | |
|---|---|
| code segment | CS |
| data segment | DS |
| stack segment | SS |
| extra segment | ES |

## x86 Instruction Set Architecture




A1 xx yy MOV AX,[yyxx]
03 CB ADD CX,BX

**tipi di istruzioni**

RET *nessun operando*

INC AL *un operando*

MOV AL,3B *due operandi*

ADD AL,BL *tre operandi*
*AL ⇐ AL+BL*

## x86 metodi di indirizzamento

**metodi di indirizzamento** (utilizzati per specificare come recuperare un operando necessario per l'esecuzione dell'istruzione):

- register addressing — MOV AX,BX
- immediate addressing — MOV AL,23
- direct addressing — MOV AL,[0123]
- indexed addressing — MOV DL,[SI]
- based indexed addressing — MOV CH,[BX+SI]
- based displaced indexed addressing — MOV AL,[0123+BX+SI]
- ed altri

## Instruction Set Architecture

**MOV *op1,op2*** copia il secondo operando nel primo

MOV AL,BL — MOV [0123],BL

MOV AL,12 — MOV SI,AX

**INC *op1*** aggiunge 1 all'operando

**DEC *op1*** sottrae 1 all'operando

INC AL — INC [0123]

DEC BL — DEC [0123]

## Instruction Set Architecture

**ADD *op1,op2*** addizione intera tra il primo ed il secondo operando, e salva il risultato nel primo

ADD AL,BL ADD [0123],BL

ADD AL,12 ADD SI,AX

**ADC op1,op2** somma con carry i due operandi

**SUB op1,op2** sottrai ad *op1* il valore di *op2*, e salva in *op1*

**CMP op1,op2** confronta op1 con op2

## Instruction Set Architecture

**AND op1,op2** AND logico

**NOT *op1*** negazione in complemento a 1

**OR *op1*** OR inclusivo logico

**XOR *op1,op2*** OR esclusivo

**STC** imposta a 1 il bit di carry

**CLC** azzera il bit di carry

## Instruction Set Architecture

**RCL *op1* ,1** Rotate Left con carry di 1 bit

**RCR *op1*,1** Rotate Right con carry di 1 bit

**ROL *op1* ,1** Rotate Left di 1 bit

**ROR *op1* ,1** Rotate Right di 1 bit

**SAL *op1* ,1** Shift Left di 1 bit [0 → dati → carry]

**SAR *op1* ,1** Shift Right di 1 bit [carry ← dati ← 0]

## DEBUG del DOS

move **M** range address
name **N** [pathname] [arglist]
output **O** port byte
proceed **P** [=address] [number]
quit **Q**
register **R** [register]
search **S** range list
trace **T** [=address] [number]
unassemble **U** [range]
write **W** [address] [drive] [firstsector] [number]

assemble **A** [address]
compare **C** range address
dump **D** [range]
enter **E** address [list]
fill **F** range list
go **G** [=address] [addresses]
hex **H** value1 value2
input **I** port
load **L** [address] [drive] [firstsector] [number]

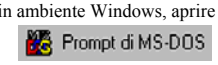
*address 100 DS:0123 1234:5678* *list 33 44 C5 "Pippo"*
*range 100 123* *number solo in HEX*

**SOMMA ESADECIMALE CON INDIRIZZAMENTO DIRETTO**

```
0100 jmp 0105 ←——— salta l'area dati
0102 db 23 ← dato_1   }
0103 db 0A ← dato_2   } area dati
0104 db 00 ← risultato}
0105 MOV  AL,[0102] ← carica in AL il valore di dato_1
0108 ADD  AL,[0103] ← somma ad AL il valore di dato_2
010C MOV  [0104],AL ← salva la somma in risultato
010F MOV  AH,4C   }
0111 INT  21      } termine regolare del programma
```

## Uso del DEBUG

1. Se siamo in ambiente Windows, aprire una finestra DOS




2. Lanciare l'utility **debug**

```
C:\>debug
```

3. Entrare in modalità Assembla digitando [**a**] :

*il valore del segmento di codice è quasi sicuramente diverso da quello visualizzato, ma per i nostri esempi è ininfluente*

```
-a
1F71:0100
```

## Uso del DEBUG

4. Inserire le istruzioni desiderate

```
0100 jmp 0100
0102 db 23
0103 db 0A
0104 db 00
0105 MOV  AL,[0102]
0108 ADD  AL,[0103]
010C MOV  [0104],AL
010F MOV  AH,4C
0111 INT  21
```

inizialmente non so quanto varrà

adesso so quanto vale, e quindi scrivo il valore corretto (**0105**)

## Uso del DEBUG

5. Uscire dalla modalità Assembla premendo [**Invio**] all'inizio di una riga vuota

```
1F71:0102 int 21
1F71:0104
-
```

6. Verificarne la correttezza con la modalità Disassembla, digitando [**u**]

```
-u
1F71 :0100 EB2A          JMP     012C
1F71 :0102 3412          XOR     AL,12
1F71 :0104 7856          JS      015C
1F71 :0106 BC9AF0        MOV     SP,F09A
```

## Uso del DEBUG

7. Leggere i valori iniziali dei registri, digitando [**r**]

```
-r
AX=0000  BX=0000  CX=0050  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=169D  ES=169D  SS=169D  CS=169D  IP=0100   NV UP EI PL NZ NA PO NC
169D:0100 EB2A          JMP     012C
-
```

8. Leggere il contenuto di un'area di memoria, digitando [**d**] seguito eventualmente dall'indirizzo di inizio dell'area desiderata ed eventualmente anche la sua lunghezza

```
-d 169D:0102
169D:0100        34 12 78 56 BC 9A-F0 DE 11 11 22 22 33 33     4.xV...
169D:0110  44 44 55 55 66 66 77 77-88 88 99 99 AA AA BB BB   DDUUffww.
169D:0120  CC CC DD DD EE EE FF FF-5E C3 50 56 BA 00 00 BE   ........^
```

## Uso del DEBUG

9. Mandare in esecuzione le prime istruzioni e verificarne l'efficacia

```
-t
AX=0000  BX=0000  CX=0050  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=169D  ES=169D  SS=169D  CS=169D  IP=012C   NV UP EI PL NZ NA PO NC
169D:012C BA0000        MOV     DX,0000
-t
AX=0000  BX=0000  CX=0050  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=169D  ES=169D  SS=169D  CS=169D  IP=012F   NV UP EI PL NZ NA PO NC
169D:012F BE0000        MOV     SI,0000
```

10. Salvare il file (aggiustare la lunghezza con BX:CX) …

```
-r bx
BX 0000
:0
-
```

```
-r cx
CX 0000
:13
-
```

## Uso del DEBUG

11. Salvare il file (stabilirne il nome con **N**) …

```
-n somma.com
-
```

12. Salvare il file (con il comando **W**)

```
-w
Scrittura di 00013 byte in corso
-
```

13. Ritorno al DOS

```
-q

C:\WINDOWS>
```

**SOMMA ESADECIMALE CON INDIRIZZAMENTO INDICIZZATO CON SPIAZZAMENTO**

```
0100 jmp 012C

 0102   34 12 78 56 BC 9A
 0108   F0 DE 11 11 22 22
 010E   33 33 44 44 55 55
 0114   66 66 77 77 88 88
 011A   99 99 AA AA BB BB
 0120   CC CC DD DD EE EE
 0128   FF FF ?? ?? ?? ??
```

```
012C MOV  DX,0
012F MOV  SI,0
0132 MOV  AX,[SI+0102]
0136 MOV  CX,12
0139 ADD  SI,2
013C ADD  AX,[SI+0102]
0140 ADC  DX,0
0143 LOOP 0139
0145 MOV  [0128],AX
0148 MOV  [012A],DX
014C MOV  AH,4C
014E INT  21
```

**INVERTIRE GLI ELEMENTI DI UN VETTORE DI 16 ELEMENTI DA 8 BIT**

```
0100 jmp 0112

0102   00 11 22 33 44 55
0108   66 77 88 99 AA BB
010E   CC DD EE FF

0112 MOV  SI,0102
0115 MOV  DI,0111
0118 MOV  AL,[SI]
011A MOV  AH,[DI]
011C MOV  [SI],AH
011E MOV  [DI],AL
0120 INC  SI
0121 DEC  DI
0122 CMP  SI,DI
0124 JBE  0118
0126 MOV  AH,4C
0128 INT  21
```

## esercizi proposti

Invertire gli elementi di un vettore di 16 elementi interi a 16 bit.

Calcolare il numero di elementi pari presenti in un vettore di 16 elementi interi ad 8 bit.

Ordinare in ordine crescente gli elementi di un vettore di 16 elementi interi ad 8 bit.