## ISTRUZIONI

MOV S,D assegnazione
MOVS S,D signed } CAST
MOVZ S,D unsigned }
lea S,D leal 16(%esp),%eax
calcola l'indirizzo di 16(%esp) e lo mette in %eax

cmp S,D calcola D-S
test S,D calcola D&S
SETCC D
CMOVCC S,D

## ISTRUZIONI ARITMETICO-LOGICHE

| istruzione | effetto |
|---|---|
| INC D | D = D+1 |
| DEC D | D = D-1 |
| NEG D | D = -D |
| NOT D | D = ~D complemento a 1 |
| ADD S,D | D = D+S |
| SUB S,D | D = D-S |
| IMUL S,D | D = D*S |
| XOR S,D | D = D^S OR ESCLUSIVO |
| OR S,D | D = D \| S |
| AND S,D | D = D&S |

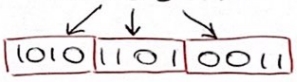
## ESADECIMALI

| | |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |
| A | 1010 |
| B | 1011 |
| C | 1100 |
| D | 1101 |
| E | 1110 |
| F | 1111 |

## CONVERSIONE

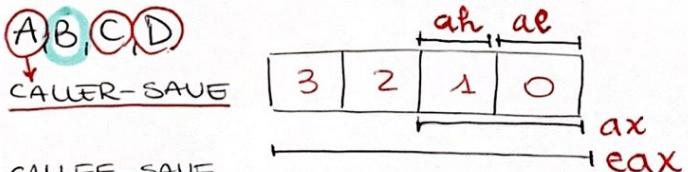
0x : esadecimale
0 : ottale

0xAD3...

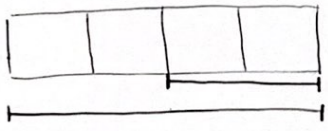



## REGISTRI

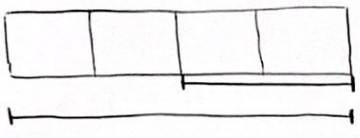
tutti da 32 bit (4 byte)
1 byte = 8 bit



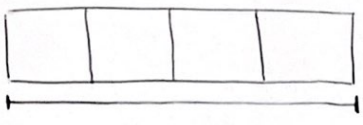

CALLEE-SAVE






SP
STACK POINTER

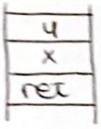



## VARIABILI

- input e output sono SEMPRE a 32 bit
- la funzione di ritorno RET restituisce SEMPRE eax
- i parametri di input di una funzione hanno indirizzo immediatamente precedente a quello di ritorno

l = int → 4 byte
b = char → 1 byte
w = short → 2 byte
puntatore → 4 byte

## FUNZIONE SOMMA




```
movl 4(%esp), %eax   } USO REGISTRI A e C
movl 8(%esp), %ecx   }
addl %ecx, %eax
ret
```

## IF - ELSE

in c:
```
int f(int x){
    int eax = x
    if eax == 0
    return 0
    else
    return 1
```

invece poi:
```
int f(int x) {
int ecx = x
if x!= 0 goto L
movl $0, %eax
ret
L: movl $1, %eax
   ret
```

in assembly:
```
movl 4(%esp), %ecx
testl %ecx,
jne L
movl $0, %eax
ret
L: movl $1, %eax
   ret
```

→ jCC SALTO CONDIZIONATO (CONDIZIONI A PAG. 51)

## CALLER - SAVE e CALLEE - SAVE

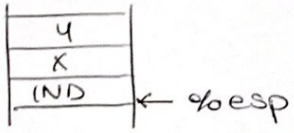
caller - save: A, C, D → usali solo se NON chiami funzioni

callee - save: B, SI, BP, DI → usali se chiami funzioni così le variabili in questi registri rimangono pulite

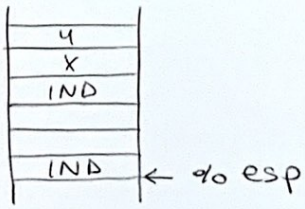
se usi registri callee - save (B, SI, BP, DI) usa push/pop:

all' inizio:
```
pushl %esi
pushl %ebp
```
} a specchio

alla fine:
```
popl %ebp
popl %esi
```




→ push →



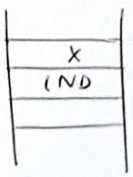

quindi x è in 16(%esp)

## VARIABILI E ALTRE FUNZIONI

se usi altre funzioni (es. call g) devi riservare altro spazio per i parametri di queste.

usa sub/add

```
int f(int x) {
    g(x)
    ret x
    }
```




→ qui allochi ie parametro x necessario a g

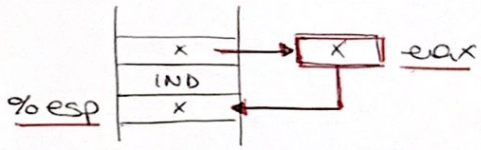
**ESEMPIO**
```
pushl %esi
subl $8, (%esp)
...
addl $8, (%esp)
popl %esi
```
SPAZIO X DUE!

usa **subl $4, (%esp)** e poi **addl $4, (%esp)**

se sono presenti delle push, le sub vanno messe subito dopo. Altrimenti per prime

**ESEMPIO**

```
int f(x) {                          int f(x) {  subl $4, (%esp)
    return g(x) + 1        →                    movl 8(%esp), %eax
    }                                           movl %eax, (%esp)
                                                call g
                                                incl %eax
                                                add $4, (%esp)
                                                ret
                                            }
```




se avessimo avuto bisogno di 2 altre variabili:

sub l $ 8 → 2 spazi ;)
movl 12
movl
movl 16
movl
addl $ 8

**WHILE**

```
while (x > 0)          →   E: if (X<=0)
    ISTRUZIONI                   goto L
                                 ISTRUZIONI
                                 jumpE → jump SALTO INCONDIZIONATO
                           L: ret
```

in assembly: movl $4 (?) X, %ecx

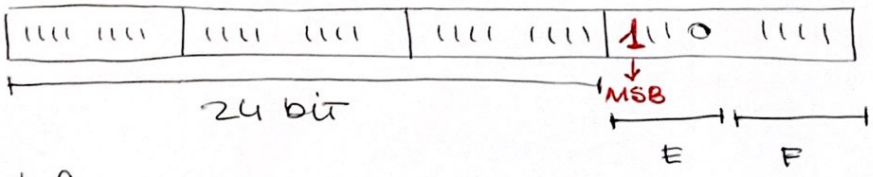
```
E: cmpl 0, X
   jle L  → SALTO CONDIZIONATO
   istruzioni del while
   jump E
L: ret
```

**MOVS e MOVZ** NON NECESSITA DI FORMATTAZIONE CON l, b, w!

- **movs** b, l → cambia da b (char) a l (int)
  - ↳ SIGNED

  "allunga" con ie MOST SIGNIFICANT BIT (MSB) che è l'ultimo a SINISTRA!

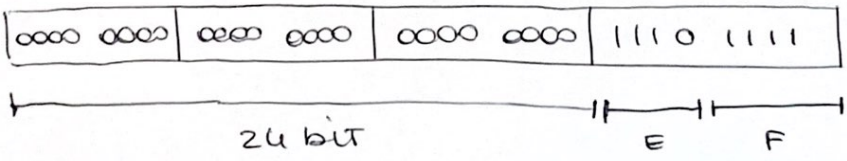
  es. EF = 1110 1111 cioè vado da 1 byte a 4




- **movz** b, l
  - ↳ UNSIGNED

  "allunga" con una serie di zeri

  es. EF = 1110 1111



## cmovCC S,D

usata per le condizioni

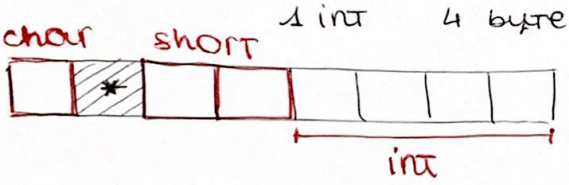
es. cmp $0, %ecx
cmovle l %eax, %ecx
(<=0)

cioè se il risultato della cmp è <=0 mette eax in ecx

## STRUCT

① Gli elementi della struct devono iniziare in posizione di indice multiplo della loro size.

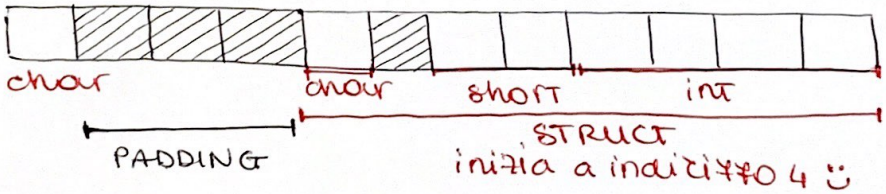
es. struct con 1 char 1 byte
1 short 2 byte
1 int 4 byte




* è detto byte di PADDING

② L'intera struct deve iniziare in posizione di ~~indice~~ indice multiplo del suo elemento più grande

es. char
struct { 1 char, 1 short, 1 int }




③ L'intera struct deve finire in posizione di indice multiplo del suo elemento più grande (cioè la sua size è un multiplo della dimensione dell'elemento più grande)

④ Data una struct non ne puoi riordinare gli elementi

## SHIFT

| | | | |
|---|---|---|---|
| LOGICO | SHL e S,D | (LEFT) | AGGIUNGONO ZERO |
| | SHR e S,D | (RIGHT) | |
| ARITMETICO | SAL e S,D | (LEFT) | |
| | SAR e S,D | (RIGHT) | AGGIUNGE MSB |

es. ECX = 0X ABADCAFE

shl e $8, %ecx : LEFT 0X ABADCAFE → 0X ABCAFE 00
shr e $8, %ecx : RIGHT 0X ABADCAFE → 0X 00ABADCA
sal e $8, %ecx : LEFT 0X ABADCAFE → 0X ADCAFE 00
sar e $8, %ecx : RIGHT 0XABADCAFE → 0X FFABADCA

$8 = 8 byte = 2 int (1 int = 4 byte)

poiché il MSB di A (A = 1010) è 1 e 8×1 = 2F

# VARIO

## • PERMESSI DI UN FILE

| UTENTE | GRUPPO | ALTRI | | | | |
|---|---|---|---|---|---|---|
| r w - | - w - | - w x | → | 6 | 2 | 3 |

con r = lettura, w = scrittura, x = esecuzione

| DECIMALE | BINARIO | STRINGA |
|---|---|---|
| 0 | 000 | - - - |
| 1 | 001 | - - x |
| 2 | 010 | - w - |
| 3 | 011 | - w x |
| 4 | 100 | r - - |
| 5 | 101 | r - x |
| 6 | 110 | rw - |
| 7 | 111 | rwx |

## • CACHE

DATI: - una cache associativa a 2 vie con 4 linee da 128 byte ciascuna e politica di rimpiazzo LRU e preferenza per linee di indice più basso
- un processo che accede in sequenza ai seguenti indirizzi di memoria scrivi interruzioni:
  1446 422 908 409 665 345 991
- alla fine della sequenza di accessi quali sono gli indici dei blocchi contenuti nelle 4 linee di cache?

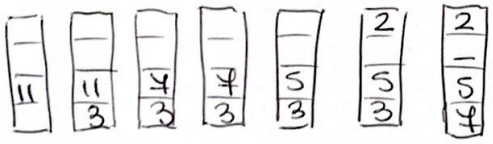
④ DIVIDI

| 1446 | 422 | 908 | 409 | 665 | 345 | 991 |
|---|---|---|---|---|---|---|
| 128 | 128 | 128 | 128 | 128 | 128 | 128 |
| indice: 11 | 3 | 7 | 3 | 5 | 2 | 7 |

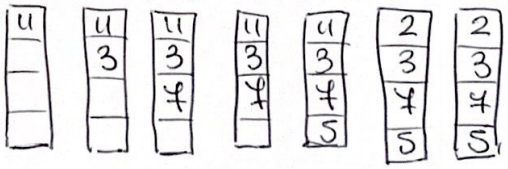
② DISEGNA


③ Metti i pari in 0,1
i dispari in 2,3

Seguendo i criteri
1 - se è vuoto mettilo in casella con indice <
2 - se è pieno sostituisci a valore che è lì da più tempo



③ Se fosse stata cache solo (FULLY) associativa:
Metti i numeri scritta distinzione nelle caselle a partire da quelle con indice <. Se sono tutte piene, sostituisci a quello che è lì da più tempo



④ CONTA i CACHE MISS (-) e i CACHEHIT (cioè le volte che hai fatto SOSTITUZIONI)

NB SE UN INDIRIZZO SI RIPETE NON VIENE AGGIUNTO DUE VOLTE MA AGGIORNA L'"ETÀ" DELLA CASELLA

- **SPEEDUP**

Qual è lo speedup MASSIMO ottenibile per un programma se ottimizziamo una sua porzione che richiede il 30% del tempo di esecuzione?

$$\lim_{K \to +\infty} \frac{1}{\frac{0,3}{K} + 1 - 0,3} = 1,43 \quad \text{poiché} \quad \frac{1}{0 + 1 - 0,3} = 1,428..$$

- **CONFRONTI TRA SPEEDUP**

Conviene rendere 2 volte più veloce una funzione che richiede il 10% del tempo di esecuzione oppure velocizzare del 10% una funzione che richiede il 90% del tempo di esecuzione? (CALCOLA LE DUE SPEED UP E SCEGLI IL + GRANDE)

CASO 1: VELOCIZZARE DI 2 VOLTE UNA F CHE RICHIEDE IL 10% DEL TEMPO →

$$\frac{1}{\frac{0,1}{2} + 1 - 0,1} = 1,05$$

CASO 2: VELOCIZZARE DEL 10% UNA F CHE RICHIEDE IL 90% DEL TEMPO →

$$\frac{1}{\frac{0,9}{1,1} + 1 - 0,9} = 1,08$$

GIUSTA!