ESPRESSIONI BOOLEANE

```
int f(int x, int y) {
    return x<y;
}
```

```
int a = 1;
if (x<y) goto E;
a = 0;
E:
return a;
```

- istruzione SETcc D
  - cond. code
  - deve essere un registro 8 bit

```
.global f
f:
    movl 4(%esp), %ecx      # x
    movl 8(%esp), %edx      # y
    cmpl %edx, %ecx
    setl %cl
    movzbl %cl, %ecx
    ret
```

REGOLA DEL CORTOCIRCUITO

```
int is_space (char *s) {
    return s != NULL && *s == ' ';
}
```

viene valutata se e solo se l'espressione a sinistra è vera

ALLINEAMENTO DELLA STACK

```
f:
    push %%ecx
    subl $4, %esp
```

```
void f(char c);
void main () {
    char a;
    f(&a);
}
```

subl $1, %esp

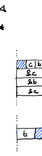
IMPORTANTE: IO GARANTISCO MOMENTO DELL'ESECUZIONE DI QUALSIASI FUNZIONE, ESP DEVE ESSERE UN MULTIPLO DI 4.

LO STACK POINTER DEVE ESSERE ALLINEATO A MULTIPLI DI 4

```
main:
    subl $4, %esp       # spazio per a
    subl $4, %esp       # spazio per il parametro di f
    ...
    leal
    call f
```

```
char a,b,c;
f(&a, &b, &c);
```

> nel prologo subl $16, %esp




```
char a;
short b;
```

> subl $4, %esp

```
char [3] a;
short b;
```

> subl $8, %esp

ALLINEAMENTO IN MEMORIA DI STRUCT

```
struct s {
    char x;
    short y;
    int z;
}
```

(1) Dove inizia in memoria ciascun campo della struct

(2) Quanto spazio occupa complessivamente la struct in memoria.

L'allineamento in memoria delle struct segue 4 regole:

(1) Un dato di dimensione X deve trovarsi allineato in memoria ad un indirizzo multiplo di X

```
char a;
short b;
```

(2) La struct ha un allineamento complessivo in memoria che dipende dalla dimensione del suo campo più grande.

(3) La dimensione complessiva di una struct è multipla della dimensione del suo campo più grande.

(4) Il compilatore NON riordina mai i campi di una struct

Esempi

```
struct s {
    char x;
    int y;
}
```

base	
+0	|x * * *|
+4	|y y y y|
sizeof: 8	

```
struct s {
    char x;
    int y;
    char z;
}
```

base	
+0	|x * * *|
+4	|y y y y|
+8	|x * * *|
sizeof: 12	

```
struct s {
    int y;
    char x;
    char z;
}
```

base	
+0	|y y y y|
+4	|x |
+5	| x * * |
sizeof: 8	

Esempio:

```
struct s {
    char x;     +0
    int y;      +4
}
void f(struct s *p, char a, int b){
    p->x = a;
    p->y = b;
}
```

```
.global f
f:
    movl 4(%esp), %eax
    movl 12(%esp), %edx
    movb 8(%esp), %cl
    movl 16(%esp), %edx
    movb %cl, (%eax)
    movl %edx, 4(%eax)
    ret
```

Esempio:

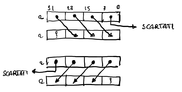
```
typedef struct node node;
struct node {
    int elem;       +0  |y y y y|
    node * next;    +8  |x x x x|
}                       sizeof: 8
int sum (node * p) {
    int count = 0;
    for (; p != 0; p = p->next)
        count += p->elem;
    return count;
}
```

```
int sum (node * p){
    node * edx = p;
    int ecx = 0;
L:  if (edx == NULL)
        goto E;
    int eax = edx->elem;
    ecx += eax;
    edx = edx->next;
    goto L;
E:  return ecx;
```

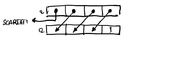
```
.global f
f:
    movl 4(%esp), %edx
    xorl %ecx, %ecx
L:  testl %edx, %edx
    je E
    movl (%edx), %eax
    addl %eax, %ecx
    movl 4(%edx), %edx
    jmp L
E:  ret
```

ISTRUZIONI SHIFT

```
int a = 9;
a = a >> 8;
```




```
a = a << 8;
```



SHIFT CON SEGNO

SAL s, D | D ← D << S → nuovi bit (meno significativi) sono posti uguali a 0

SAR s, D | D ← D >> S → nuovi bit (più significativi) sono posti uguali al valore del bit più significativo di D

SHIFT SENZA SEGNO

SHL s, D | D ← D << S
SHR s, D | D ← D >> S
} → nuovi bit posti uguali a 0

Esempio:

```
int c = 0xABADCAFE;
unsigned d = 0xABADCAFE;
```

c = c << 8;	sall $8, %ecx	ecx → ADCAFE00
d = d << 8;	shll $8, %edx	edx → ADCAFE00
c = c >> 8;	sarl $8, %ecx	ecx → FFADADCA
d = d >> 8;	shrl $8, %edx	edx → 00ADADCA

0x AB = 1 0 1 0

Esempio:

```
int a = 8;
a = a >> 2;
```

···· 00 1000 → 8
···· 000010 → 2 = 8/2^2

bit shift

a >> x ⇒ a / 2^x

```
int a = 8;
a = a << 2;
```

···· 001000 → 8
···· 100000 → 32 = 8 * 2^2

a << x ⇒ a * 2^x

ISTRUZIONE IDIV

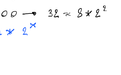
IDIV S | A ← D:A/S quoziente
| D ← D:A % S resto

```
int a = 20;
int c = 3;
a = a/c;
```

```
movl $20, %eax
movl $3, %ecx

movl %eax, %edx
sarl $32, %edx

idivl %ecx
```



ISTRUZIONE CMOVcc (CMOV CONDIZIONALE)

```
if (a > b)  y = x;
```

CMOVcc S, D | D ← { S se cc == 1; D se cc == 0 }

- non può essere un valore immed.
- DEVE essere un registro a 16/32 bit

```
cmpl B, A
cmovg X, Y
```