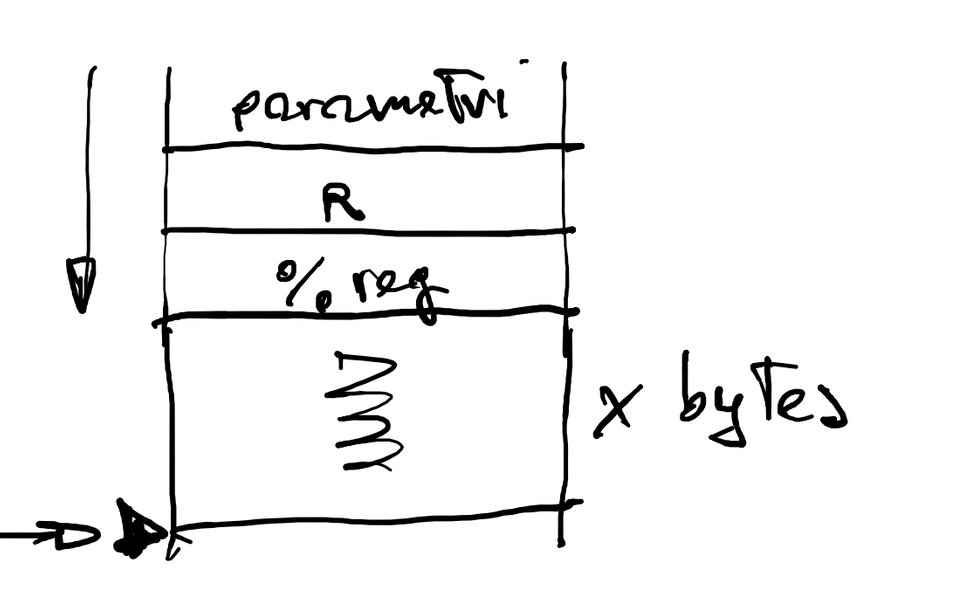
ALLINEAMENTO DELLA STACK

Several key points about the stack frame deserve mention.

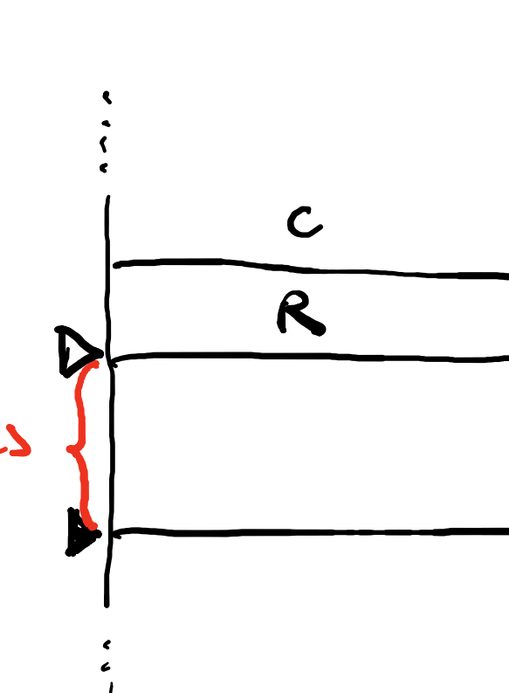
- The stack is word aligned. Although the architecture does not require any alignment of the stack, software convention and the operating system requires that the stack be aligned on a word boundary.

```
pushl %reg
subl $X , %esp
```

$X: multiplo di 4




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

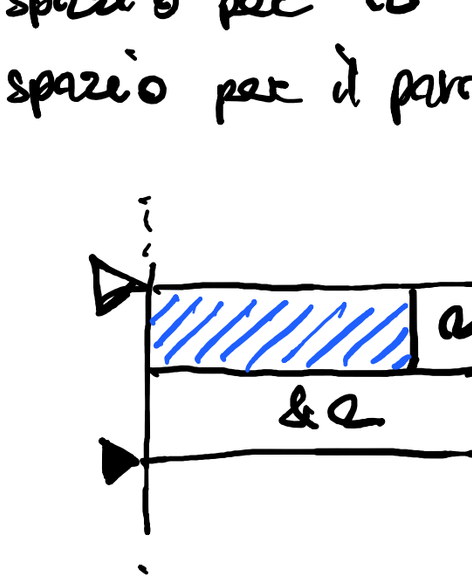


sbagliato:

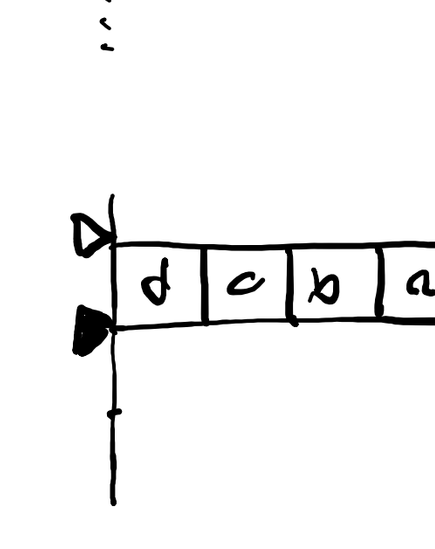
```
subl $1, %esp
subl $4, %esp
⋮
leal ...
call f
```

CODICE CORRETTO

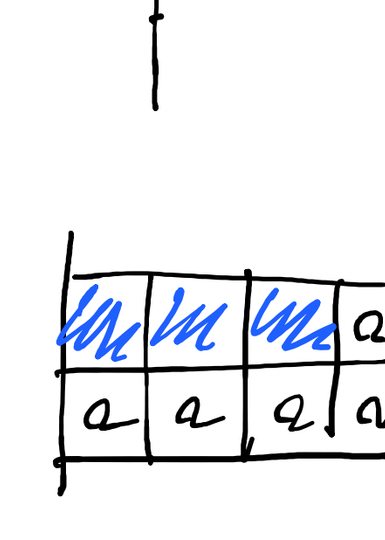
```
subl $4, %esp    # spazio per a
subl $4, %esp    # spazio per il parametro di f
⋮
leal ...
call f
```



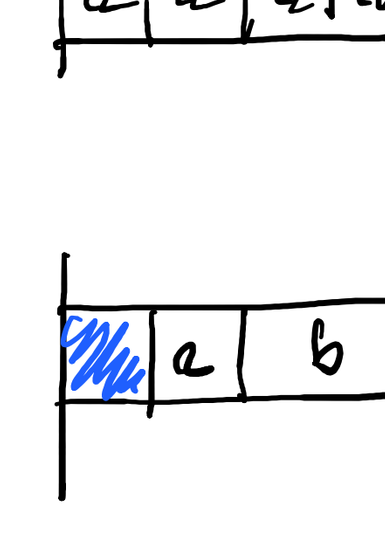
char a, b, c, d;



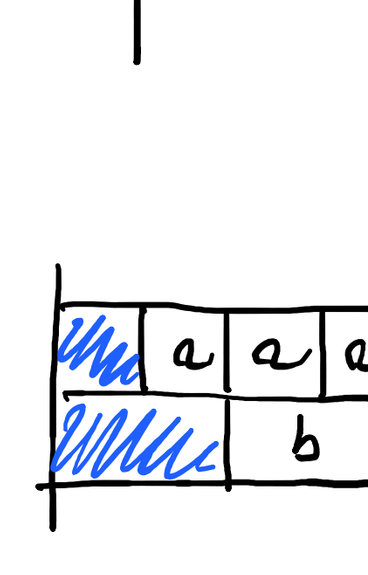
char [5] a;
↓
subl $8 , %esp



char a;
short b;
↓
subl $4, %esp

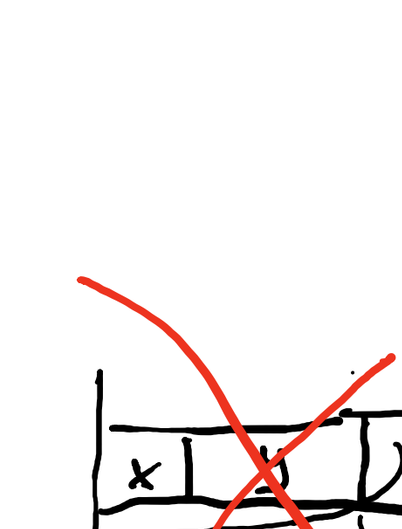


char [3] a;
short b;
↓
subl $8, %esp



ALLINEAMENTO DELLE STRUCT

```
struct s {
    char x;   → 1 byte
    short y;  → 2 byte
    int z;    → 4 byte
}
```

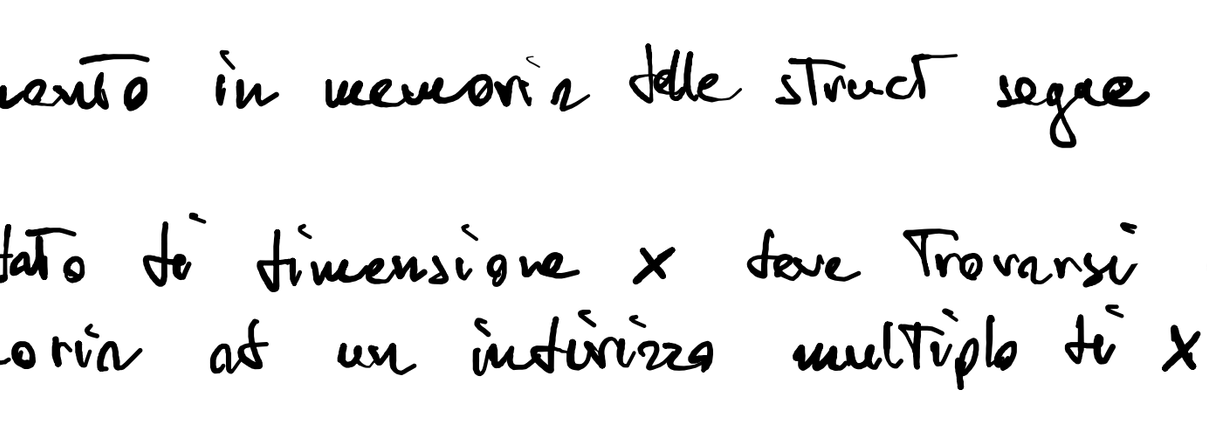


(D1) Dove inizia in memoria ogni campo della struct?

(D2) Quanto spazio occupa in memoria la struct?

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



2. La struct ha un allineamento che dipende dalla dimensione del suo campo più grande
3. La dimensione complessiva di una struct è multiplo della dimensione del suo campo più grande
4. Il compilatore NON RIORDINA MAI i campi di una struct

ESEMPI:

```
struct s {          base
    char x;         +0    |x 0 0 0|
    int y;          +4    |x x x x|
}                   sizeof 8

struct s {          base
    char x;         +0    |x 0 0 0|
    int y;          +4    |x x x x|
    char z;         +8    |x 0 0 0|
}                   sizeof 12

struct s {          base
    int y;          +0    |x x x x|
    char x;         +4    |x
    char z;         +5        x 0 0|
}                   sizeof 8
```

ESEMPIO

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

```
.globl f
f:
    movl 4(%esp), %eax     # p
    movb 8(%esp), %cl      # a
    movl 12(%esp), %edx    # b
    movb %cl , (%eax)
    movl %edx , 4(%eax)
    ret
```

ESEMPIO

```
typedef struct nodo nodo
struct nodo {           base
    int elem;           +0    |x x x x|
    nodo * next;        +4    |x x x x|
}                       sizeof 8
```

```
int sum (nodo * p) {
    int count = 0;
    for ( ; p ; p = p -> next)
        count += p -> elem;
    return count;
}
```

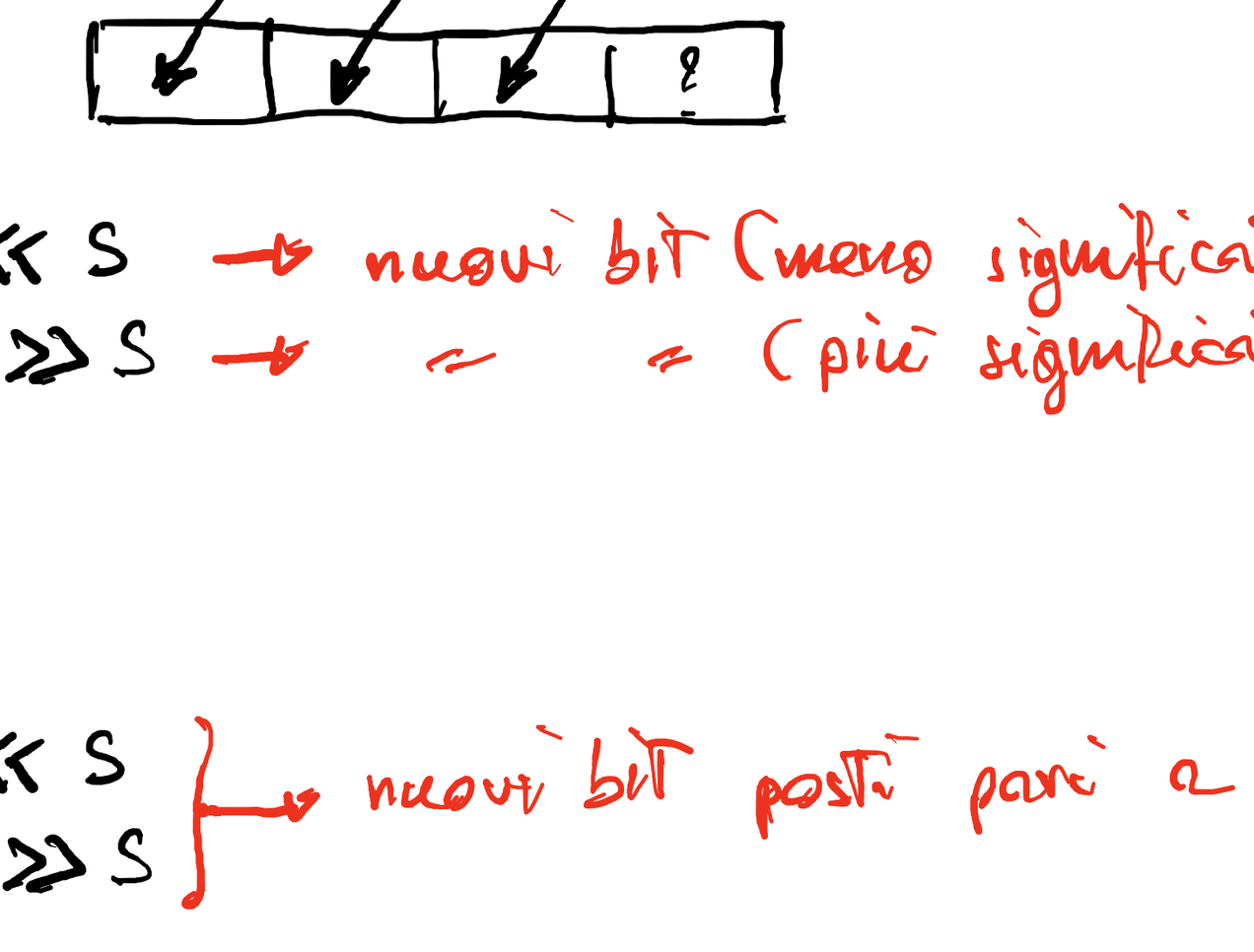
```
int sum (nodo * p){
    nodo * edx = p;
    int eax = 0;
L:
    if (edx == NULL) goto E;
    int ecx = edx -> elem;
    eax = eax + ecx
    edx = edx -> next;
    goto L
E:
    return eax;
```

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

ISTRUZIONI SHIFT

int a = ? ;
a = a >> 8;

a = a << 8;



SHIFT CON SEGNO

SAL S, D | D ⟵ D << S → nuovi bit (meno significativi) posti a 0.
SAR S, D | D ⟵ D >> S → " (più significativi) posti uguale al bit più significativo di D

SHIFT SENZA SEGNO

SHL S, D | D ⟵ D << S
SHR S, D | D ⟵ D >> S
→ nuovi bit posti pari a 0

int c = 0x ABADCAFE; → ecx
unsigned d = 0x ABADCAFE; → edx

c = c << 8; sall $8 , %ecx
d = d << 8; shll $8 , %edx

ecx → ADCAFE00
edx → ADCAFE00

c = c >> 8; sarl $8, %ecx
d = d >> 8; shrl $8, %edx

ecx → FFABADCA   0xAB = (1)011
edx → 00ABADCA

int a;
a = 8;
a = a >> 2;   ......1000 = 8, ...0010 = 2 = $8/2^2$ (2: bit shift)

$a >> x \Longrightarrow a/2^x$

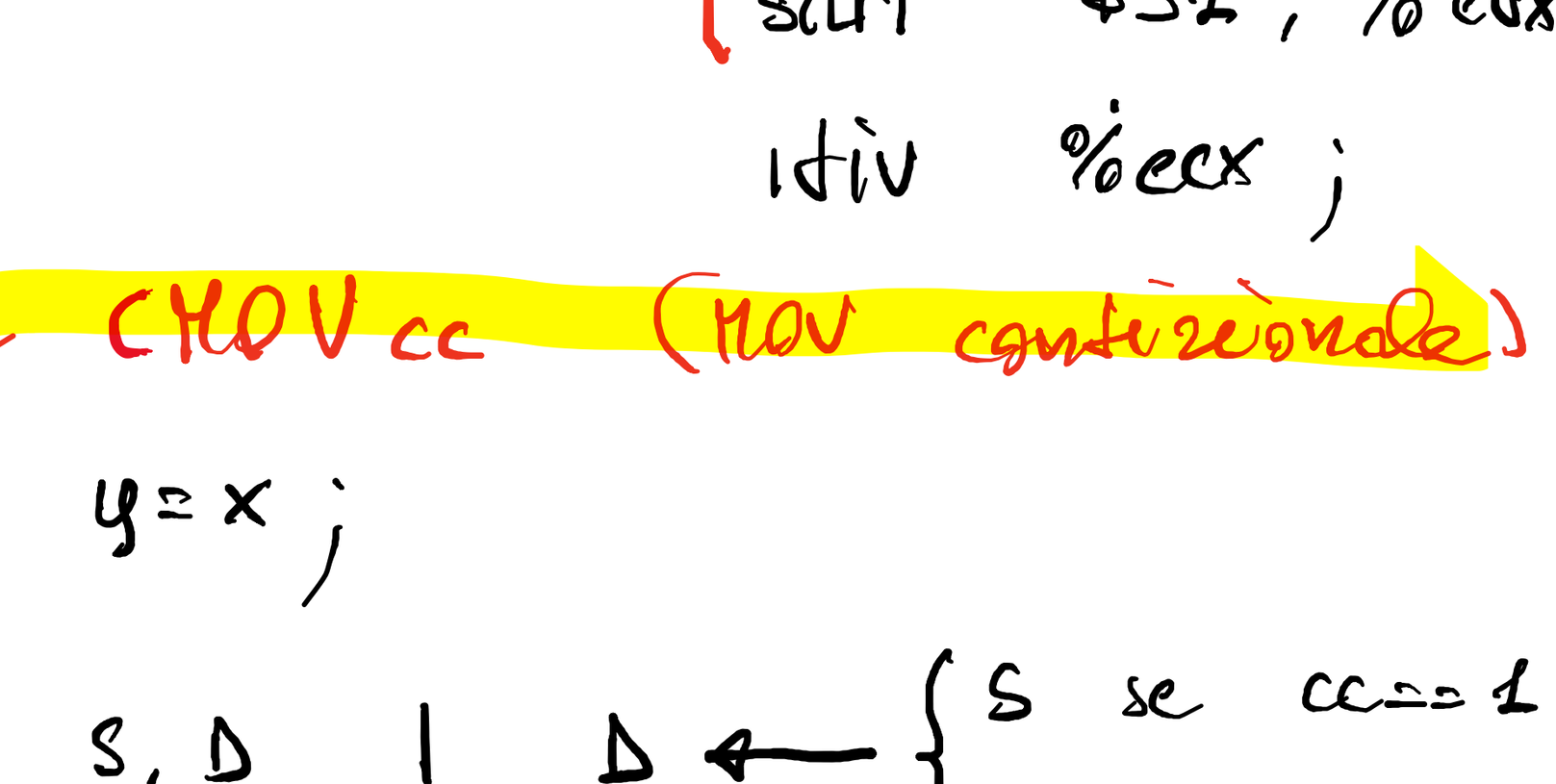
int a;
a = 8
a = a << 2;   ......1000 = 8, ...100000 = 32 = $8 * 2^2$

$a << x \Longrightarrow a * 2^x$

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 %eax, %edx
sarl $31, %edx
idiv %ecx;
```

ISTRUZIONE CMOVcc (MOV condizionale)

if (a > b) y = x;

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

S: non può essere un valore imm
D: deve essere un registro 16/32 bit

```
cmpl B, A
cmovgl X, Y
```