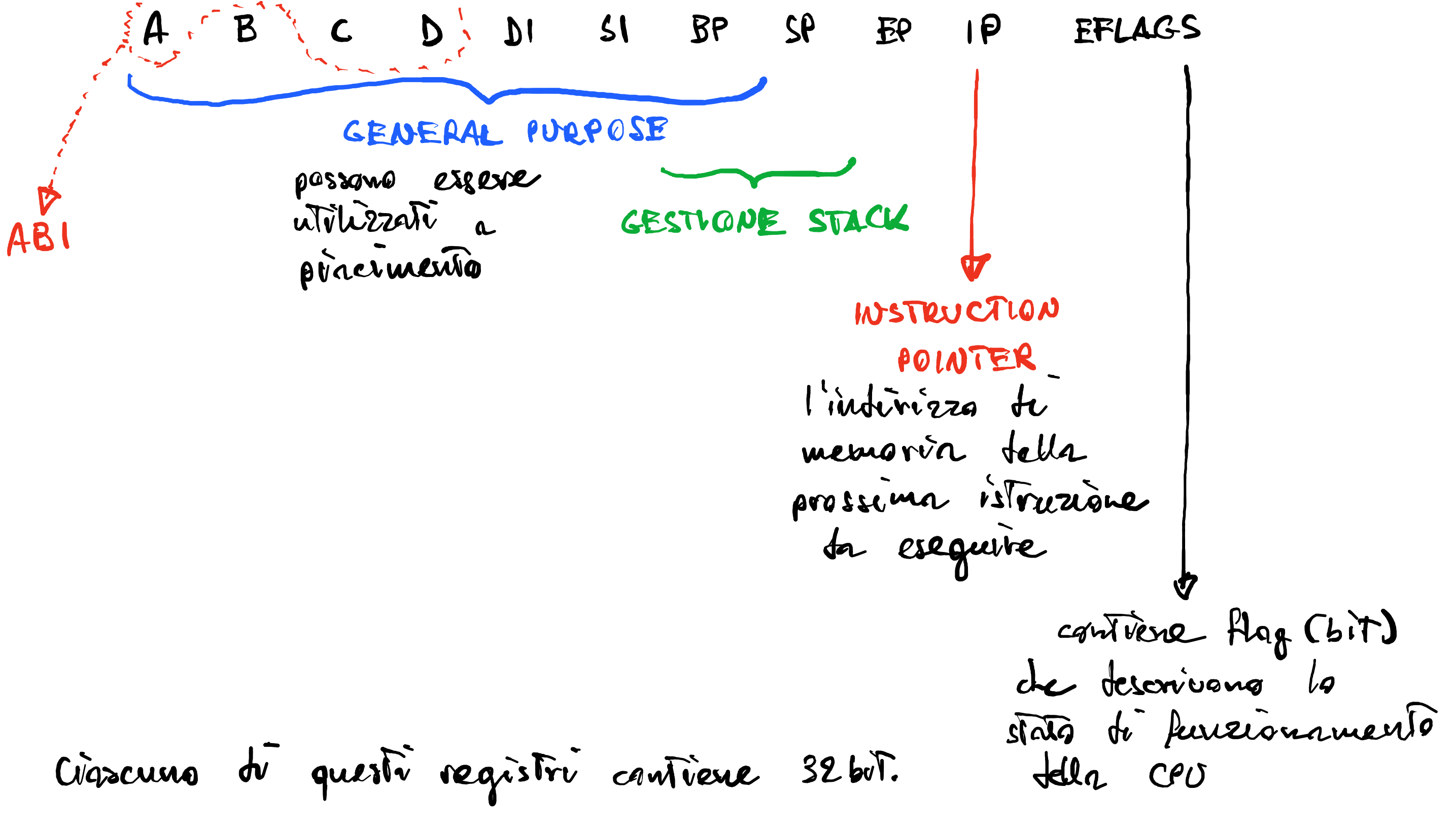
SC - 5/3/2025

mercoledì 5 marzo 2025 10:11

REGISTRI

Registri IA32 :

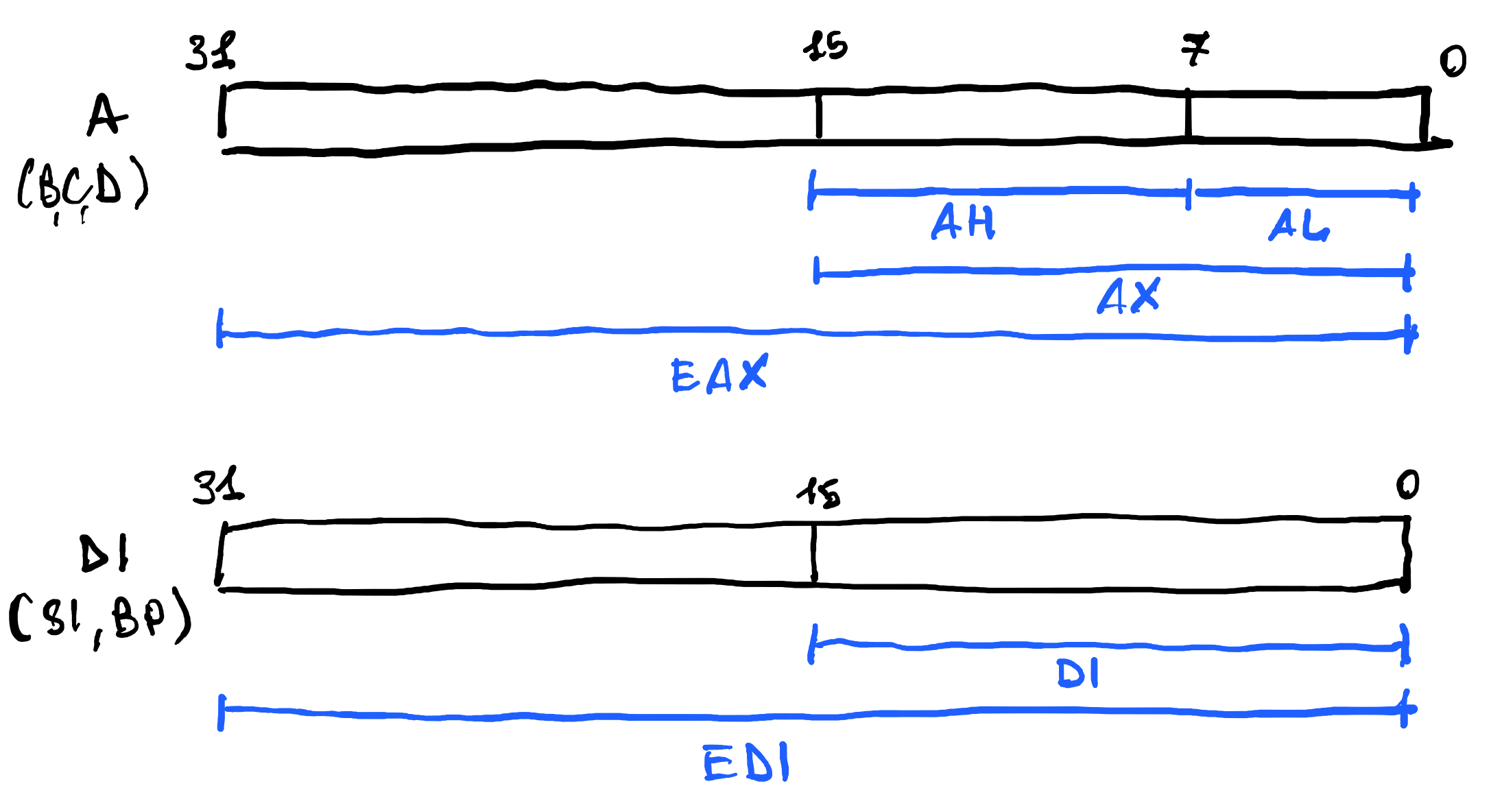


possono essere utilizzati a piacimento

l'indirizzo di memoria della prossima istruzione da eseguire

contiene flag (bit) che descrivono lo stato di funzionamento della CPU

Ciascuno di questi registri contiene 32 bit.



RAPPRESENTAZIONE DEI NUMERI

BIG-ENDIAN : Il byte più significativo viene posto all'indirizzo più basso

LITTLE-ENDIAN : Il byte meno significativo " " " "

Es. Voglio rappresentare 0x 12345678

BE	0x12	0x34	0x56	0x78
	0	1	2	3

LE	0x78	0x56	0x34	0x12
	0	1	2	3

L'architettura IA32 è LE

DEFINIZIONE FUNZIONI IN ASM

Per definire una funzione (Es. "main") utilizziamo una etichetta (o label)

```
nome_etichetta :
```

Per definire la funzione main()

```
main:
```

Per rendere la funzione visibile da altri file oggetto usiamo la direttiva ".globl"

```
.globl main
main:
```

ESEMPIO

```
                                    .globl f
int f() {                           f:
    int x;     dichiarazione        # x -> eax
    x = 10;    assegnazione
    return x;
}
```

FORMATO ISTRUZIONI ASM

Ogni istruzione è caratterizzata da:

- nome
- un certo numero di operandi
- semantica

Riguardo agli operandi:

Ⓐ Istruzioni SENZA operandi (Es.: nop, ret, ...)

Ⓑ Istruzioni UNARIE

OP D | D ← OP D

(D: operando destinazione)

Ⓒ Istruzioni BINARIE

OPt S, D | D ← D OP S → notazione AT&T (in alternativa a notazione Intel)

(S: operando sorgente; D: operando destinazione)

Ⓓ Istruzioni TERNARIE

ISTRUZIONI PER LO SPOSTAMENTO DEI DATI

MOV S, D | D ← S

operando sorgente
- → immediato $ (es.: $10)
- → registro % (es.: %eax)
- → memoria

operando destinazione
- → registro
- → memoria

> Regola : non si usano valori immediati per l'operando D

> Regola : non è possibile usare l'indirizzamento a memoria per entrambi gli operandi

Dobbiamo indicare la dimensione dei dati su cui operare aggiungendo un suffisso a OP

b	1 byte (8 bit)	es.: per char
w	2 byte (16 bit)	es.: short
l	4 byte (32 bit)	es.: int , T*

ESEMPIO

```
                                    .globl f
int f() {                           f:
    int x;     dichiarazione        # x -> eax
    x = 10;    assegnazione         movl $10, %eax
    return x;  -> salvare il valore # non devo fare nulla
                  di ritorno          perché il valore è già in EAX!
               -> Terminare la
                  funzione          ret
}
```

```
int x = 10;  ------------> # x -> eax
             |-----------> movl $10, %eax
```

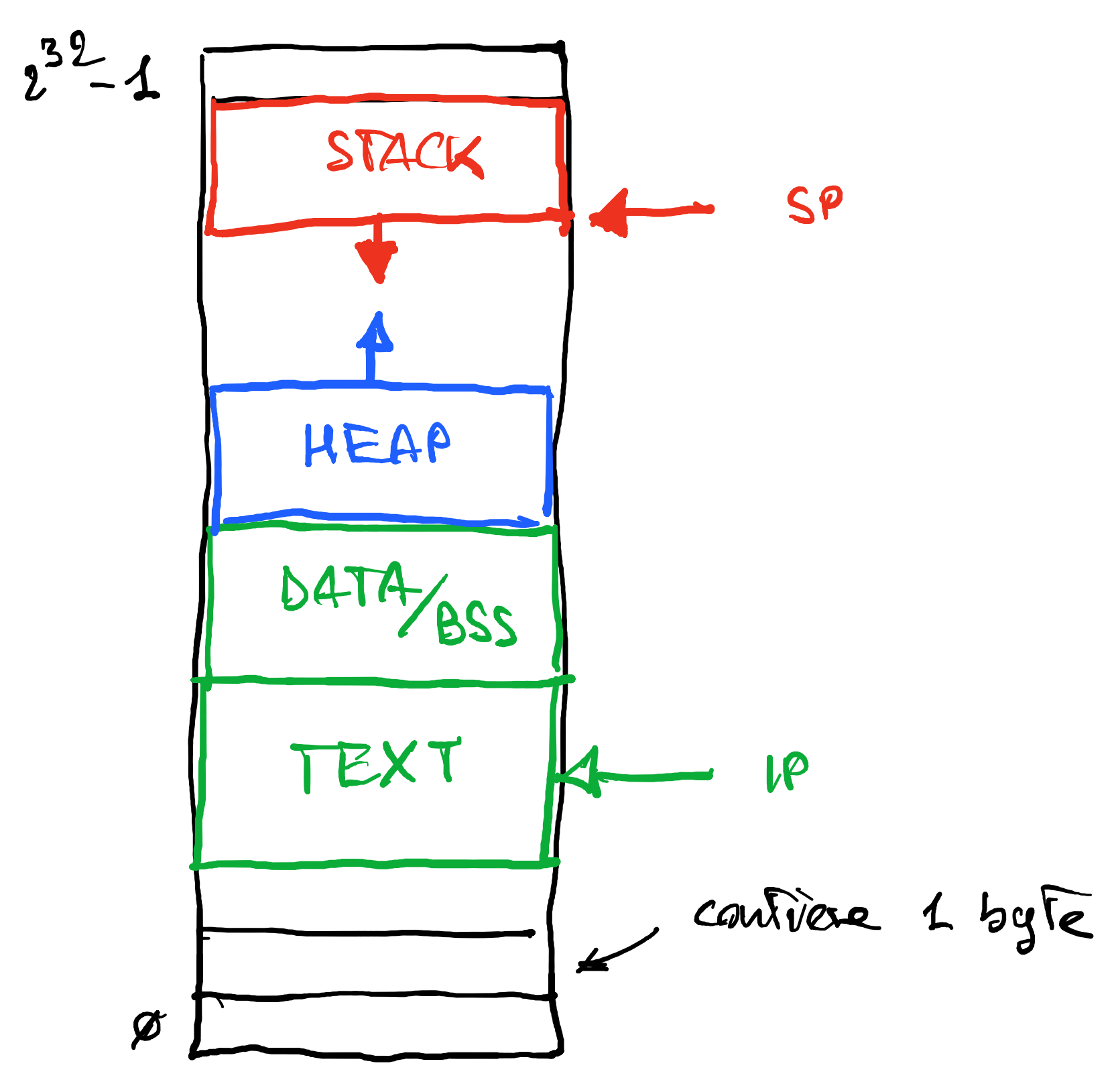
OPERAZIONI ARITMETICHE / LOGICHE IN ASM

ADD	S, D	D ← D+S	
SUB	S, D	D ← D−S	
~~MUL~~	~~S, D~~	~~D ← D*S~~	~~senza segno~~
IMUL	S, D	D ← D*S	con segno
AND	S, D	D ← D & S	bit a bit
OR	S, D	D ← D | S	"
XOR	S, D	D ← D ^ S	"
NEG	D	D ← −D	
NOT	D	D ← ~D	(!D)
INC	D	D ← D+1	
DEC	D	D ← D−1	

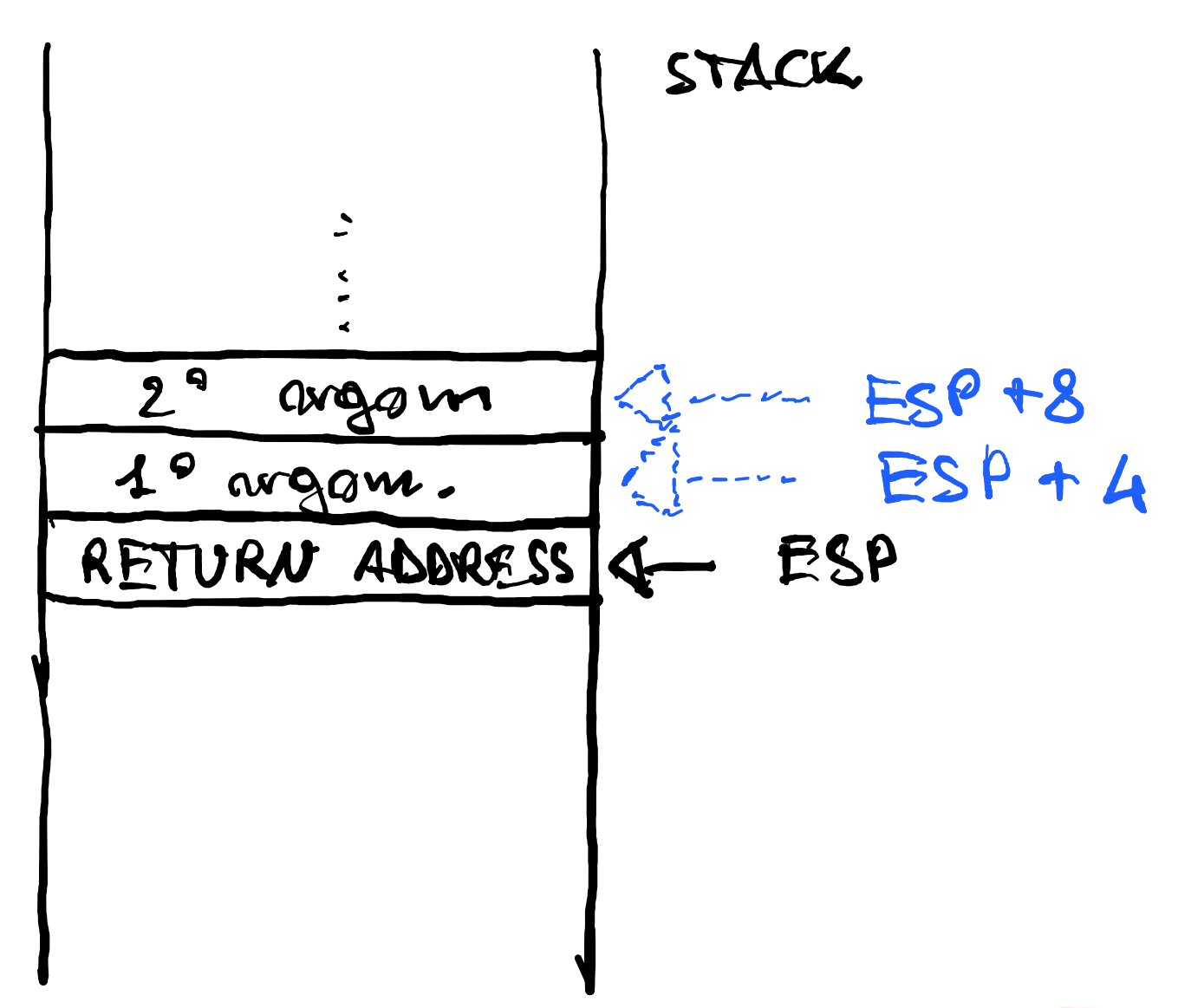
ACCESSO AI PARAMETRI

```
int f(int x) {          int f(int x) {          .globl f
    return x+1              int a;              f:
}                           a = x;                  # a -> eax
                            a++;                    movl ?, %eax
                            return a;               incl %eax
                        }                           ret
```

STRUTTURA DEI DATI IN MEMORIA



Cosa accade quando viene chiamata la funzione f



> Regola : Tutti i parametri attuali (argomenti) occupano 4 byte nello stack

ACCESSO IN MEMORIA CON SPIAZZAMENTO

d(addr)
- addr → puntatore (indirizzo di memoria)
- d → spiazzamento

ESP + 4 → 4(%esp)