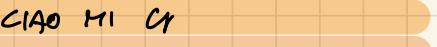


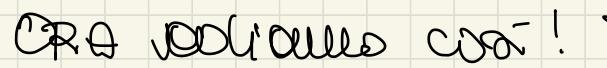
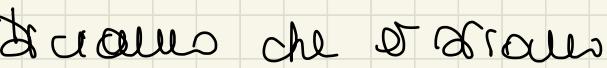
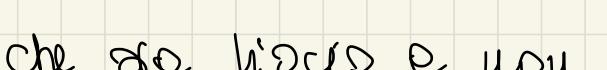
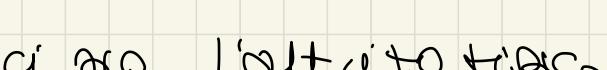


Sistemi Di Calcoli

CIAO   
NEL MONDO REALE →    
NEL   
  


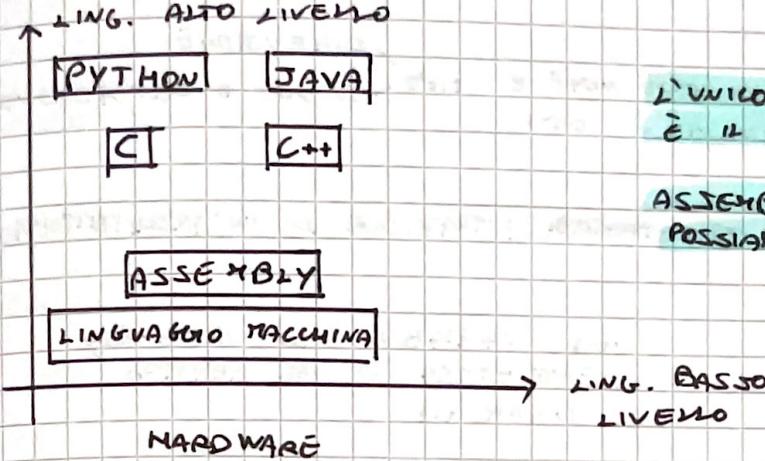
CIAO MI CHIAMA VERONICO  
e sono un gattino ciccione !  
Sto cercando un amico con cui  
giocare, ho fatto proprio l'odore perfetto  
per me.  
VUOI ESSERE TU IL MIO AMICO ?  
Ci direttissimo un mondo, te lo prometto !



CRA VOLGARE CO' !    
che sto lì da e non ci sto l'ultimo tipo  
delle carte ...    
questione di obbligare. Quindi BRAVO,  
AUGURI !! 

Probabilmente ecco cosa mi piacerebbe di più perché la parola è pur  
occasione. Forse il mio preludere appunto così. Bevi CIAO !

## PROGRAMMAZIONE A BASSO LIVELLO



L'UNICO LINGUAGGIO CHE PARLA LA CPU  
È IL LINGUAGGIO MACCHINA

ASSEMBLY È DI BASSO LIVELLO E  
POSSIAMO COMPRENDERELO

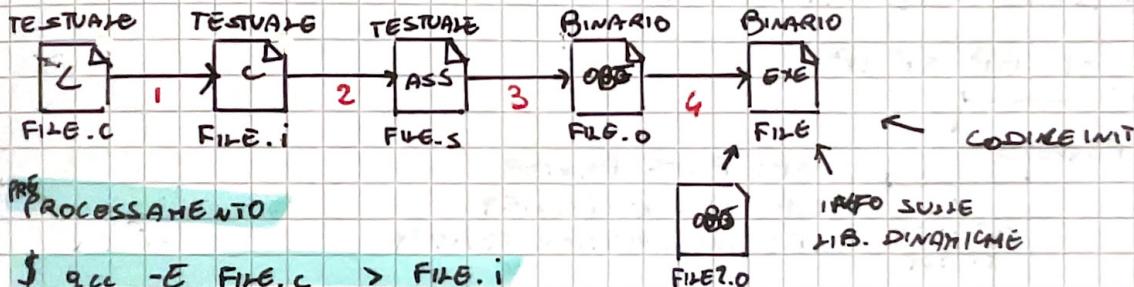
## PIPELINE DI COMPILAZIONE

COME PASSARE DA C A LINGUAGGIO MACCHINA?

C è TESTUALE:  
HUMAN FRIENDLY



BINARIO: SEQUENZA DI BYTES  
LO CAPISCE SOLO LA MACCHINA



### 1 PRE PROCESSAMENTO

```
$ gcc -E FILE.c > FILE.i
```

```
#include <stdio.h>
#define PI 3.14...
```

### 2 COMPILAZIONE

```
$ gcc -S FILE.i > FILE.c
```

#### TRADUZIONE 1:n

↳ NON È UNIVOCÀ, CI SONO VARIE TRADUZIONI MA OGNI TRADUZIONE DÈVE  
ESSERE SEMANTICAMENTE EQUIVALENTE AL CODICE C ORIGINALE

TRADUZIONE INVERSA → DECOMPILAZIONE

### 3 ASSEMBLAGGIO

#### TRADUZIONE 1:1

↳ IMPLEMENTATA DAL PRODUTTORE

TRADUZIONE INVERSA → DISASSEMBLAGGIO

```
$ objdump -d FILE.O
```

```
$ gcc -c FILE.S > FILE.O > FILE.i
```

### 4 LINKING

```
$ gcc -o FILE FILE.O -lm
FILE.O
: SERVE SE HAI
LIB. MATEMATICHE
```

## PIATTAFORMA x86 (IA32)

INT X;  $\Rightarrow$  DICHIARAZIONE VAR CON UN CERTO NOME E TIPO - SEGNO o SENZA SEGNO  
X = 10;  $\Rightarrow$  ASSEGNAZIONE  $\rightarrow$  SPOSTAMENTO DI DATI

- INSTRUCTION-SET-ARCHITECTURE (ISA)  $\rightarrow$  ~~PERIZOMA~~ ASTRAZIONE DI UN'ARCHITETTURA

- STATO CPU  $\begin{cases} \text{REGISTRI} \\ \text{MEMORIA (RAX)} \end{cases}$
- FORMATO DELLE ISTRUZIONI
  - 1 SPOSTAMENTO DATI
  - 2 ARITMETICO / LOGICHE
  - 3 CONTROLLO DEL FLUSSO
- EFFETTO ISTRUZIONI SULLO STATO

UNA VARIABILE C IN ASSEMBLY VIENE MESSA O NEL REGISTRO O IN MEMORIA

## REGISTRO

REGISTRI IA32 (x86)  $\rightarrow$  32 bit; 4 byte

A, B, C, D, DI, SI, BP, SP, IP

GENERAL PURPOSE

↓  
GESTIONE STACK

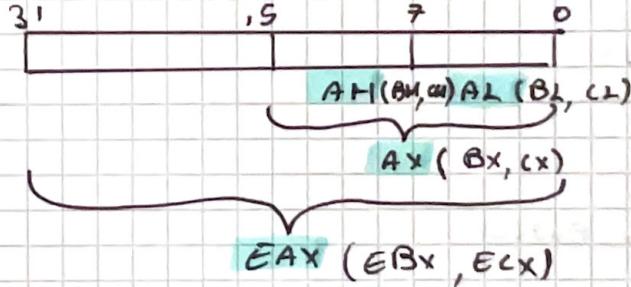
BP: BASE POINTERS

SP: STACK POINTER

IP: INSTRUCTION POINTER

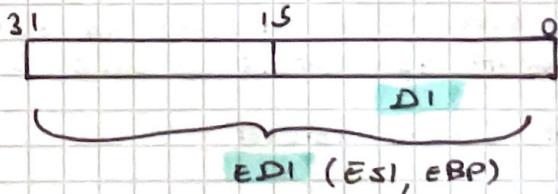
REGISTRO A (B, C, D)

32 bit



REGISTRO DI (SI, BP)

32 bit



INT X  $\leftrightarrow$  EAX, EBX, ECX, EDX (32 bit)

SHORTY  $\leftrightarrow$  AX, BX, DI, SI (16 bit)

CHAR B  $\leftrightarrow$  AL, AH, CL, CH (8 bit)

A, C, D SONO I REGISTRI PREFERITI

AL REGISTRO NON INTERESSA SE È CON o SENZA SEGNO

# SINTASSI

LINGUAGGIO  
ASSEMBLY

INTEL

ATT (GNU ASSEMBLY O GAS)

## ISTRUZIONI SPOSTAMENTO DATI

### ISTRUZIONE MOV

- MOV S, D → DESTINAZIONE | D ← S  
 ↓  
 SORGENTE

S ← IMMEDIATO (\$)  
 REGISTRO (%)  
 MEMORIA

D ← REGISTRO  
 MEMORIA

- OP D UNARIA | D ← OP D

- OP S,D BINARIA | D ← D OP S

- OP S<sub>1</sub>, S<sub>2</sub>, D TERNARIA | D ← D OP

IN SINTASSI ATT:

OP [ ] S,D  
 ↓  
 SUFFISSO  
 ↓  
 INDICA BYTE MANIPOLATI  
 DA OP

OP	SUFFISSO	DIENSIONE	TIPO C
	b	1 BYTE	CHAR, UNSIGNED CHAR
	w	2 BYTE	(UNSIGNED) SHORT
	l	4 BYTE	(UNSIGNED) INT (UNSIGNED) LONG

VINCOLO: SORGENTE E DESTINAZIONE NON POSSONO ESSERE  
 ENTRAMBI OPERANDI MEMORIA

# X=10 # COMMENTO

movl \$10, %eax  
 ↓ | MOVE REGISTRO 32 bit  
 IMMEDIATO REGISTRO

# SHORT Y=700;  
 movw \$700, %cx # UNSIGNED CHAR Z=1000;  
 movb \$1000, %dl

ESE:

int f() {  
 int x;  
 x=10;  
 RETURN x; → } → OK

SPETTA A NOI

① ASSEGNA  
VALORE DI  
RITORNO  
② Torna  
a chi  
chiama

ABI → APPLICATION-BINARY-INTERFACE ("CONTRATTO" TRA I PROGRAMMATATORI DI UNA PIATTAFORMA)  
 ↳ ABI LINUX IA32

- VALORE DI RITORNO

REGOLA # 1: IL VALORE DI RITORNO VIENE MEMORIZZATO NEL REGISTRO A

4 BYTE  $\rightarrow$  EAX      2 BYTE  $\rightarrow$  AX

ES:

.GLOBAL f (CHIAMATA FUNZIONE)

f:

```
#X  $\leftrightarrow$  EAX
movl $10, %eax
ret
```

```
# INT f()
# INT X;
# X=10;
# RETURN X;
```

PER COMPILARE:

gcc -m32 f.c f.s -o f      IMPORTANTE!

### ISTRUZIONI ARITM. / LOGICHE

- ADD	S, D	$D \leftarrow D + S$
- SUB	S, D	$D \leftarrow D - S$
- IMUL	S, D	$D \leftarrow D * S$ (SEGUO)
- MUL	S, D	$D \leftarrow D * S$ (NO SEGUO)
- NEG	D	$D \leftarrow -D$
- INC	D	$D \leftarrow D + 1$
- DEC	D	$D \leftarrow D - 1$
- AND	S, D	$D \leftarrow D \& S$
- OR	S, D	$D \leftarrow D   S$
- XOR	S, D	$D \leftarrow D ^ S$
- NOT	D	$D \leftarrow \sim D$

BIT WISE  $\rightarrow$  CONFRONTO  
2 BIT ALLA VOLTA

ES:

.GLOBAL f

f:

```
movl $5, %eax # INT a=5;
addl $7, %eax # a = a + 7;
ret # RETURN a;
```

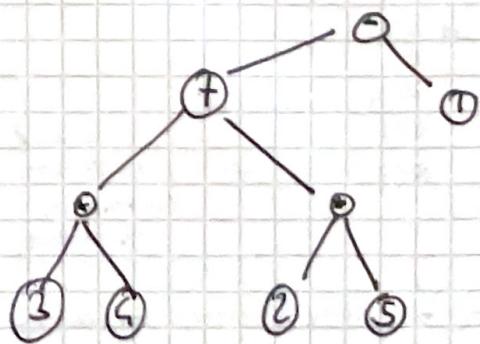
ES:

.GLOBAL f

f:

```
movl $3, %eax # INT f();
imull $1, %eax # WT a=3;
movl $2, %ecx # INT c=2;
imull $5, %ecx # c=c*S;
addl %ecx, %eax # a=a+c;
decl %eax # a=a-1;
ret # RETURN a;
```

```
# INT f()
# WT a=3;
# a = a * 1;
# INT c=2;
# c = c * 5;
# a = a + c;
# a = a - 1;
# RETURN a;
```



SERVE UNO SPAZIO SOTTO

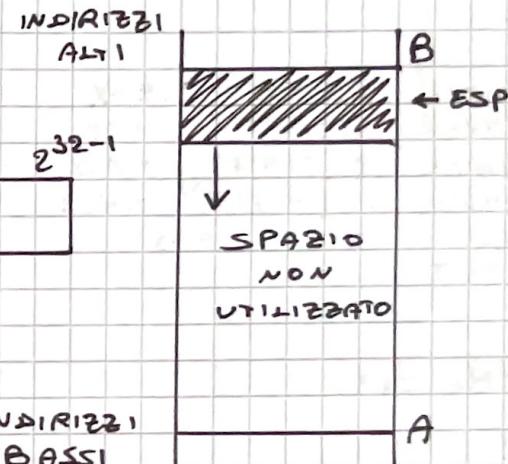
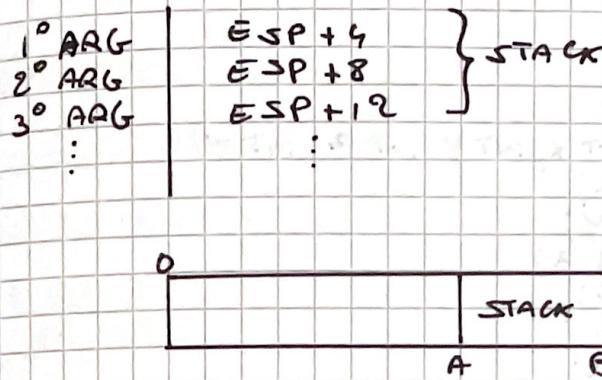
ES:

```
INT f (INT x) {
    RETURN x + 1;
}
```

GLOBAL f  
⇒ f:

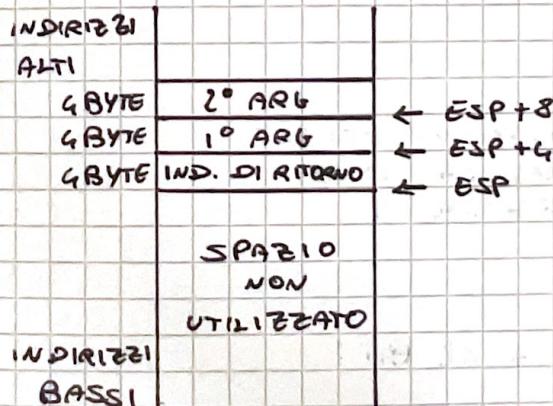
```
movl ?, %eax
incl %eax
RET
```

ARGOMENTO DELLA FUNZIONE?



ESP = ESP - 32 ← RISERVARE  
ESP = ESP + 32 ← DEALLOCARE

PARTE ESECUZIONE DI f:



OPERANDO MEMORIA CON SPAZIAMENTO COSTANTE (1° TIPO)

ESP  
↓ ATT  
g(%ESP)  
|  
BASE  
SPAZIAMENTO COSTANTE

- 1 LEGGE ESP
- 2 SOMMA 4
- 3 LEGGE IN MEMORIA ALL'INDIRIZZO CALCULATO IN 2

movl g(%ESP), %eax  
incl %eax  
RET

RISULTATO FUNZIONE DI SOPRA

ES:

.GLOBAL SOMMA

```
SOMMA : # INT SOMMA(INT X, INT Y)
    movl 4(Y.ESP), Y.EAX # INT a = x;
    movl 8(Y.ESP), Y.ECX # INT c = y;
    ADDL Y.ECX, Y.EAX # a = a + c;
    RET # RETURN a;
```

ES:

.GLOBAL f

f:

```
    movl 4(Y.ESP), Y.EAX # INT f(INT X, INT Y, INT Z, INT W)
    IMULL 8(Y.ESP)', Y.EAX # INT a = x;
    # a = a * y;
    movl 12(Y.ESP)', Y.ECX # INT c = z;
    IMULL 16(Y.ESP)', Y.ECX # c = c * w;
    ADDL Y.ECX, Y.EAX # a = a + c;
    RET # RETURN a;
```

ES:

.GLOBAL SQR

SQR:

```
    movl 4(Y.ESP), Y.EAX # INT SQR(INT X)
    IMUL Y.EAX, Y.EAX # a = a * a;
    RET # RETURN a;
```

### ISTRUZIONE DIV (UNSIGNED)

- DIV S | A  $\leftarrow$  D : A / S      CONCATENAZIONE DEI BIT  
      ↓                                    |  
 SORGENTE | D  $\leftarrow$  D : A V. S      DEL REGISTRO D E  
   DEL REGISTRO A

IL NUMERATORE NON LO DECIDIAMO NOI

SE IO MO S (DENOMINATORE), IL RISULTATO A 32 BIT:

<u>DIV ECX</u>	EAX $\leftarrow$ EDX : EAX / ECX	32 BIT
	EDX $\leftarrow$ EDX : EAX V. ECX	
<u>DIV CX</u>	AX $\leftarrow$ DX : AX / CX	16 BIT
	DX $\leftarrow$ DX : AX V. CX	
<u>DIV CL</u>	AL $\leftarrow$ AX / CL	8 BIT
	AH $\leftarrow$ AX V. CL	

ES:

INT X = 10;  
INT Z = 2, 15;  
↓

\$0 PERCHÉ A INTERESSA  
SOLO A, QUINDI D  
DEVE ESSERE 0

```
    movl $10, Y.EAX
    movl $5, Y.ECX
    DIVL Y.ECX
    movl $0, Y.EDX    oppure XORL Y.EDX, Y.EDX
```

ES:

.GROBL f

R:

```

movl 4(Y.ESP), Y.EAX
movl 8(Y.ESP), Y.EDX
XORL Y.EDX, Y.EDX
DIVL Y.EDX
RET

```

```

# UNSIGNED f(UNSIGNED x, UNSIGNED y)
# INT a = x;
# INT c = y;
# INT d = 0;
# a = a/c;
# RETURN a;

```

## ISTRUZIONE IF/ELSE

MECCANISMO DI TEST:

1 OPERAZIONE ARITMETICO/LOGICO

2 VIENE ESEGUITO UN SALTO CONDIZIONATO IN BASE AD UNA PROPRIETÀ (CONDITION CODE) SUL RISULTATO R

CONDITION CODE	PROPRIETÀ
e (z)	$R == 0$
ne	$R \neq 0$
ge	$R > 0$
le	$R \geq 0$
a	$R < 0$
ae	$R \leq 0$
b	$R > 0$
be	$R \geq 0$
	$R < 0$
	$R \leq 0$

DA SAPERE!

ISTRUZIONE SALTO CONDIZIONATO  $\rightarrow$  jjcc L  
T  
CONDITIONAL  
CODE LABEL

SI SALTA A L SE LA PROPRIETÀ INDICATA DA CC È VERA

ES:

Subl Y.ECX, Y.CAX

$$R = EAX - ECX$$

jjg L

:

L:

:

jjg : SALTA SE  $R > 0$ , VA A (L:)

## ISTRUZIONE CMP (COMPARE)

CMP S,D |  $R \leftarrow D-S$

SUB S,D |  $R \leftarrow D-S$ ,  $D \leftarrow R$

## ASSEMBLY

```

CMP S,D
jcc L
    1STR1;

```

```

L:
    1STR2;

```

C	C EQUIVALENTE
IF (TEST) 1STR1; 1STR2;	IF (!TEST) GOTO L; 1STR1;
L: 1STR2;	

ES:

~~INT~~ F (INT a) {

IF (a < 0)      IF (a >= 0) GOTO L;  
  a = -a;      a = -a;

RETURN a;

}

L:  
  RETURN a;

ASM

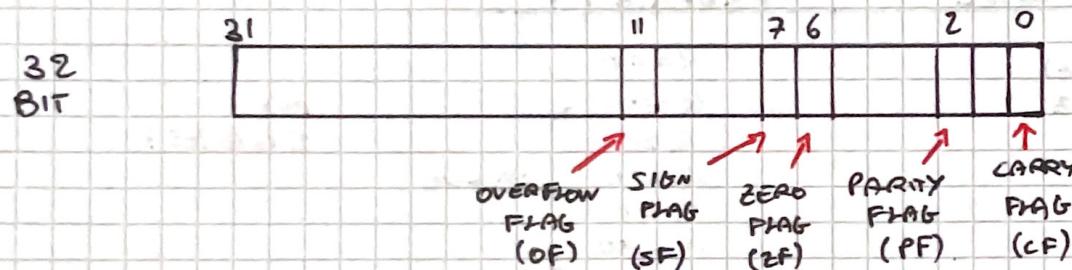
movl 4(%ESP), %EAX  
cmpl \$0, %EAX  
jge L  
negl %EAX

a >= 0  
↓  
a - 0 = 0      >= 0  
    ↓  
    cc

L:

RET

### REGISTRO EFLAGS



IN BASE AL CONDITION CODE I FLAG VENGONO AGGIORNATI

### ISTRUZIONE ELSE (JUMP)

C

C EQUIVALENTE

ASSEMBLY

IF (TEST) ISTR 1;

IF (!TEST) GOTO L;

CMP \$0, %D

ELSE

ISTR 2;

ISTR 1;

JCC L

ISTR 3;

GOTO L2;

LSTR 1

,

L:

JMP L2

ISTR 2;

L:

L2:

ISTR 2

ISTR 3;

L2:

JMP L SALTA A L

ES:  
INT MAX(INT X, INT Y) {

INT a = x;  
INT b = y;

IF (a <= b) GOTO L;  
GOTO L2;

L:

a = b;

L2:

RETURN a;

}

.GOBL MAX

MAX:

MOVl 4(Y.ESP), Y.EAX  
MOVl 8(Y.ESP), Y.EBX  
CMPL Y.EBX, Y.EAX  
JLE L  
JMP L2

L:

MOVl Y.EBX, Y.EAX

L2:

RET

### ISTRUZIONE TEST

TEST S,D	R = D & S	SIMILE A AND S,D	R = D & S
			D ← R

MOLTE VOLTE INVECE DI TROVARE "CMP \$0, D" ABBIANO "TEST D,D"

ES:

INT BETWEEN(INT X, INT Y, INT Z) {  
IF (X ≥ Y) GOTO F;  
IF (Y ≥ Z) GOTO F;  
RETURN 1;

F:  
RETURN 0;

INT a; INT d = y; INT a = z;  
→ IF (X >= Y) GOTO F;  
IF (Y >= Z) GOTO F;  
a = 1;  
GOTO E;

F:  
a = 0;

E:  
RETURN;

.GOBL BETWEEN

→

BETWEEN:

MOVl 4(Y.ESP), Y.ELX  
MOVl 8(Y.ESP), Y.EDX  
MOVl 12(Y.ESP), Y.EAX  
CMPL Y.EDX, Y.ELX  
JGE F  
CMPL Y.EAX, Y.EDX  
JGE F  
MOVl \$1, Y.EAX  
JMP E

F:

MOVl \$0, Y.EAX // XORL Y.EAX, Y.EAX

E:

RET

## CICLO WHILE

C

C EQUIVALENTE

ASSEMBLY

WHILE (TEST) ISTR 1;  
ISTR 2;

L:  
IF (!TEST) GOTO E;  
ISTR 1;  
GOTO L;  
  
E:  
ISTR 2;

L:  
CMP S,D  
JCC E  
ISTR 1  
JMP L  
  
E:  
ISTR 2

ES:

```
INT SUM (INT n) {  
    INT c=0;  
    INT a=0;  
    WHILE (n > 0) {  
        a += n;  
        n --;  
    }  
    RETURN a;  
}
```

```
INT SUM (INT n) {  
    INT c=n;  
    INT a=0;  
L:  
    IF (c <= 0) GOTO E;  
    a += c;  
    c -= 1;  
    GOTO L;  
  
E:  
    RETURN a;
```

.GLOBAL SUM  
  
SUM:  
 MOVl 4(%ESP), %ECX  
 XORl %EAX, %EAX  
L: TESTl %ECX, %ECX  
 Jle E  
 ADDl %ECA, %EAX  
 DECl %ECX  
 SMPL L  
  
E: RET

## CICLO FOR

C

C EQUIVALENTE

ASSEMBLY

FOR (INIT; TEST; UPDATE) {  
 ISTR 1;  
}  
ISTR 2;

INIT;  
L:  
IF (!TEST) GOTO E;  
ISTR 1;  
UPDATE;  
GOTO L;  
  
E:  
ISTR 2;

INIT  
  
L:  
CMP S,D  
JCC E  
ISTR 1  
UPDATE  
JMP L  
  
E:  
ISTR 2

ES:

```
UNSIGNED COUNT1 (UNSIGNED n) {  
    UNSIGNED a=0;  
    UNSIGNED c=n;
```

```
L: IF (c <= 0) GOTO E;  
    INT d = c & 1;  
    IF (d == 0) GOTO F;  
    a++;  
F: c = c / 2;  
    GOTO L;  
E: RETURN a;
```

.GLOBAL COUNT1

COUNT1:

XORl %EAX, %EAX F:  
MOVl 4(%ESP), %ECX  
L: CMPL \$0, %ECX  
Jbe E  
TESTl %ECX, %ECX  
Jc F \$1, %ECX  
INCL %EAX  
E: RET

ES:  
 UNSIGNED FACT (UNSIGNED n) {  
 UNSIGNED a=1;  
 UNSIGNED c=2;  
 UNSIGNED d=n;

L: IF (c > d) GOTO E;  
 a\*=c;  
 ++c;  
 GOTO L;

E:  
 RETURN a;  
 }

.620B1 FACT

FACT:

MOV \$1, Y.EAX  
 MOVE \$2, Y. ECX  
 MOVL Y.ECX, Y. EDX  
~~MOVL Y. ECX, Y. EDX~~

~~MOVPL Y. ECX, Y. EDX~~

L:  
 CMPL 4(Y.ESP), Y. ECX  
 JA E  
 IMUL Y. ECX, Y. EDX  
 INC Y. ECX  
 JMP L

E: RET

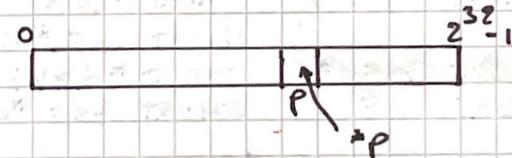
## PROTEZIONE IN TERA

BIT  
 {8, 16} → {32}  
 SHORT  
 CHAR

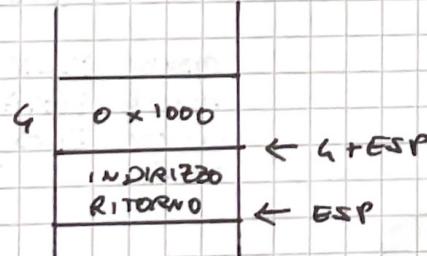
VOID F (CHAR x) ⇒ IL CHIAMANTE DEVE PROTEGGERE IL PARAMETRO A 32 BIT

## PUNTATORI

INT \* x;  
 SHORT \* y;  
 CHAR \* z; } T \* → 32 bit →  
 IA32  
 INDIRIZZO DI MEMORIA



WT F (CHAR \* p)



movl G(Y.ESP), Y.EAX ①  
 movb (Y.EAX), Y.CL ②

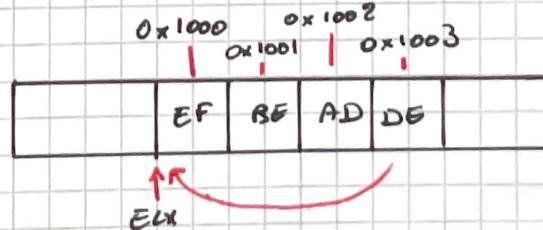
- ① OTTIENI PUNTATORE
- ② OTTIENI OGGETTO PUNTATO

ENDIANNESS  
 BIG ENDIAN  
 LITTLE ENDIAN

# ECX = 0x1000

movl \$0x DEADBEEF, Y.EAX  
 move Y.EAX, (Y. ECX)  
 movl (Y. ECX), Y. EDX

# EDX = 0x DGADEBEEF



movN (Y. ECX), Y.DX  
 # DX = 0x BEEF

ES:

```
VOID TIMES2 (SHORT* p){  
    SHORT *a = p;     $\rightarrow$  INDIRIZZO  
    SHORT c = *a;     $\rightarrow$  LEGGO 10 SHORT  
    c = c * 2;  
    *a = c;  
}
```

.GLOBL TIMES2

TIMES2:

```
    movl 4(%ESP), %EAX  
    movw (%EAX), %CX  
    imulw $2, %CX  
    movw %CX, (%EAX)  
    RET
```

ES:

```
VOID SWAP (CHAR* x, CHAR* y){  
    CHAR* c = x;  
    CHAR* d = y;  
  
    CHAR AL = *c;  
    CHAR AH = *d;  
  
    *c = AH;  
    *d = AL;  
}
```

.GLOBL SWAP

SWAP:

```
    movl 4(%ESP), %ECX  
    movl 8(%ESP), %EDX  
  
    movb (%ECX), %AL  
    movb (%EDX), %AH  
  
    movb %AH, (%ECX)  
    movb %AL, (%EDX)  
    RET
```

ES:

```
VOID SWAP (SHORT* x, SHORT* y){  
    SHORT *c = x;  
    SHORT cx = *c;  
    SHORT *d = y;     $\rightarrow$   
    SHORT dx = *d;  
  
    *d = cx;  
    c = x;  
    *c = dx;  
}
```

.GLOBL SWAP

SWAP:

```
    movl 4(%ESP), %ECX  
    movw (%ECX), %CX  
  
    movl 8(%ESP), %EDX  
    movw (%EDX), %DX  
  
    movw %CX, (%EDX)  
    movl 4(%ESP), %EAX  
    movw (%EAX), %CX  
    movw %DX, (%EAX)  
    RET
```

ES:

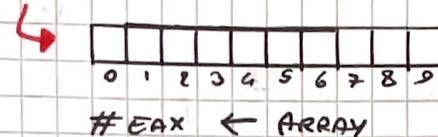
```
VOID R (UNSIGNED SHORT n, UNSIGNED SHORT* SPAR,  
        UNSIGNED SHORT* SDIP){  
    UNSIGNED SHORT CFO;
```

# ARRAY / STRINGHE IN ASM

```
INT i = 5;
INT ARRAY [10];
INT e = ARRAY [i];
```

Ogni cella equivale a 4 byte

ARRAY → BASE



4 byte

```
movl (Y.EAX), Y.ECX # ARRAY[0]
movl 4(Y.EAX), Y.ECX # ARRAY[1]
movl 8(Y.EAX), Y.ECX # ARRAY[5]
```

OPERANDO MEMORIA → IN SINTASSI ATT

BASE INDICE SCALA

$\Rightarrow \text{ADDR} = Y.\text{Reg1} + Y.\text{Reg2} \cdot \text{SCALA}$

$+ d$

BASE + INDICE · SCALA + COSTANTE

/      |      |      |      |      |      |  
 d (Y.Reg1, Y.Reg2, SCALA)      {1, 2, 4, 8}  
 |      |      |      |      |      |  
 COSTANTE      BASE      INDICE      64 bit      EAX      EDX      C  
 (SI PUÒ OMETTERE, d=0)      (i)      COSTANTE

È POSSIBILE OMETTERE LA SCALA

$(Y.\text{Reg1}, Y.\text{Reg2}) = (Y.\text{Reg1}, Y.\text{Reg2}, 1)$

ES:

```
SHORT GET (SMORT * v, UNSIGNED n, UNSIGNED i){
  IF (i >= n)
    RETURN -1;
  RETURN v[i];
}
```

```
SHORT GET (SMORT * v, UNSIGNED n, UNSIGNED i){
  SHORT * c = v;
  UNSIGNED d = n;
  UNSIGNED a = i;
  SHORT a2;

  IF (i < n) GOTO F;
  a2 = -1;
  GOTO E;

F: a2 = c[a];
E: RETURN a2;
```

```
}
```

.GLOBAL GET

GET:

```
movl 4(Y.ESP), Y.ECX
movl 8(Y.ESP), Y.EDX
movl 12(Y.ESP), Y.EAX
andl Y.EDX, Y.EAX
jb F
movw $-1, Y.AX
jmp E
```

F:

movw (Y.ECX, Y.EAX, 2), Y.AX

E:

RET

C EQUIVALENTE

ASSEMBLY

EJ:

```

SMORT SUM (SMORT *v, INT n) {
    WT i = 0;
    SMORT a = 0;
    SMORT *d = v;
    L:
        IF (i >= n) GOTO E;
        a = a + d[i];
        i++;
        GOTO L;
    E:
        RETURN a;
}

```

.GLOBL SUM  
SUM:

```

XORL Y.ECX, Y.ECX
XORLW Y.AX, Y.AX
MOVL 4(Y.ESP), Y.EDX

```



```

L: Cmpl 8(Y.ESP), Y.ECX
jge E
ADDW (Y.EDX, Y.ECX, 2), Y.AX
INCL Y.ECX
JMP L
E:
RET

```

EJ:

```

UNSIGNED MY_STRLEN (CONST CHAR *s) {
    CONST CHAR *c = s;
    UNSIGNED a = 0;
    L:
        *s++ == 0
        IF (*s == 0) GOTO E; // *s: LEGGO CARATTERE PUNTATO
        a++;
        GOTO L; // S++: INCREMENTO PUNTATORE
    E:
        RETURN a;
}

```

.GLOBL MY\_STRLEN

MY\_STRLEN:

```

MOVL 4(Y.ESP), Y.ECX
XORL Y.EAX, Y.EAX
L:
MOVB (Y.ECX), Y.DL
INCL Y.ECX
TESTB Y.DL, Y.DL // CMPB $0, Y.DL
JBE E
INCL Y.EAX
JMP L
E:
RET

```

EJ:

```

CHAR * STRCAT (CHAR * DEST, CONST CHAR * SRC) {
    CHAR * c = DEST;
    CONST CHAR * d = SRC;
    CHAR * s = DEST;
    L: IF (*c == 0) GOTO E;
    c++;
    GOTO L;
    E:;
    CHAR AL = *d;
    *c = AL;
    IF (AL == 0) GOTO E2;
    c++;
    d++;
    GOTO E;
    E2: RETURN s;
}

```

.GLOBL STRCAT  
STRCAT:



```

MOVL 4(Y.ESP), Y.ECX
MOVL 8(Y.ESP), Y.EDX
L: TESTB CRPB $0, (Y.ECX) je E
INCL Y.ECX
JMP L
E: MOVB (Y.EDX), Y.AL
MOVB Y.AL, (Y.ECX)
TESTB Y.AL, Y.AL
JBE E2
INCL Y.ECX
INCL Y.EDX
JMP E
E2: MOVL 4(Y.ESP), Y.EAX
RET

```

## ARITMETICA DEI PUNTATORI

SE  $p$  È UN PUNTATORE NEL REGISTRO EAX:

```
CHAR* ADDL $1, Y.EAX
SHORT* ADDL $2, Y.EAX
INT* ADDL $4, Y.EAX
```

IN C ABBIAMO UN ARITMETICA DEI PUNTATORI AUTOMATICA.  
IN ASM È ESPlicita, DOBBIAMO FARLA NOI

ES:

```
VOID ARRAY_SET(SHORT*v, INT n, SHORT c){  
    SHORT *p = v + n - 1;  
    WHILE (*p >= v){  
        *p-- = c;  
    }  
}
```

↓ C\_EQ

```
VOID ARRAY_SET(SHORT*v, INT n, SHORT*x){  
    SHORT c = x;  
    INT d = n;  
    SHORT* a = v;  
    a = a + n;  
    a = a - 1;  
L: IF (a < v) GOTO E;  
    *a = x;  
    a = a - 1;  
    GOTO L;  
E: RETURN
```

→ ASM

.GLOBAL ARRAY\_SET

ARRAY\_SET:

```
movw 12(Y.ESP), Y.CX  
movl 8(Y.ESP), Y.EDX  
movl 4(Y.ESP), Y.EAX  
ADDL Y.EDX, Y.EAX  
ADDL Y.EDX, Y.EAX  
SUBL $2, Y.EAX  
L: CXPL 4(Y.ESP), Y.EAX  
Jb E  
movw Y.CX, (Y.EAX)  
SUBL $2, Y.EAX  
JmpL  
E: RET
```

ES DI ESSERE:

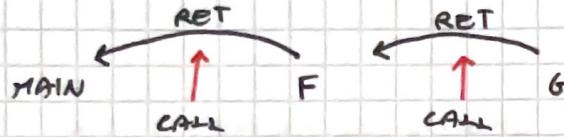
```
# INT IS_PREFIX (CONST CHAR*s, CONST CHAR*x){  
    CONST CHAR *c = s;  
    CONST CHAR *d = x;  
L: ;  
    CHAR AL = *c;  
    IF (AL == 0) GOTO E;  
    IF (*d == 0) GOTO E;  
    IF (AL != *d) GOTO E;  
    C++;  
    d++;  
    GOTO L;  
E: ;  
    INT a  
    IF (a != 0) GOTO F;  
    a = 1;  
    GOTO E?;  
F:  
    a = 0;  
E?:  
    RETURN a;
```

→ ASM

.GLOBAL IS\_PREFIX  
IS\_PREFIX:

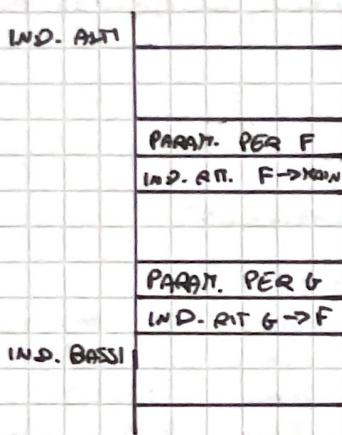
```
movl 4(Y.ESP), Y.ECX  
movl 8(Y.ESP), Y.EDX  
L:  
    movb (Y.ECX), Y.AL  
    TESTB Y.AL, Y.AL  
    JZ E $0  
    CMPB X, (Y.EDX)  
    JZ E  
    CXPL Y.AL, (Y.EDX)  
    INCL Y.ECX  
    INCL Y.EDX  
    JMP L  
E:  
    CMPB $0, (EECX)  
    JNZ F  
    movl $1, Y.EAX  
    JMP E?  
F:  
    XORL Y.EAX, Y.EAX  
E?:  
    RET
```

## CHIAMATA A FUNZIONE



MAIN: CHIAMANTE → CALLER  
F: CHIAMATA → CALLEE

F: CHIAMANTE  
G: CHIAMATA



NUOVO STACK FRAME DI UNA FUNZIONE:

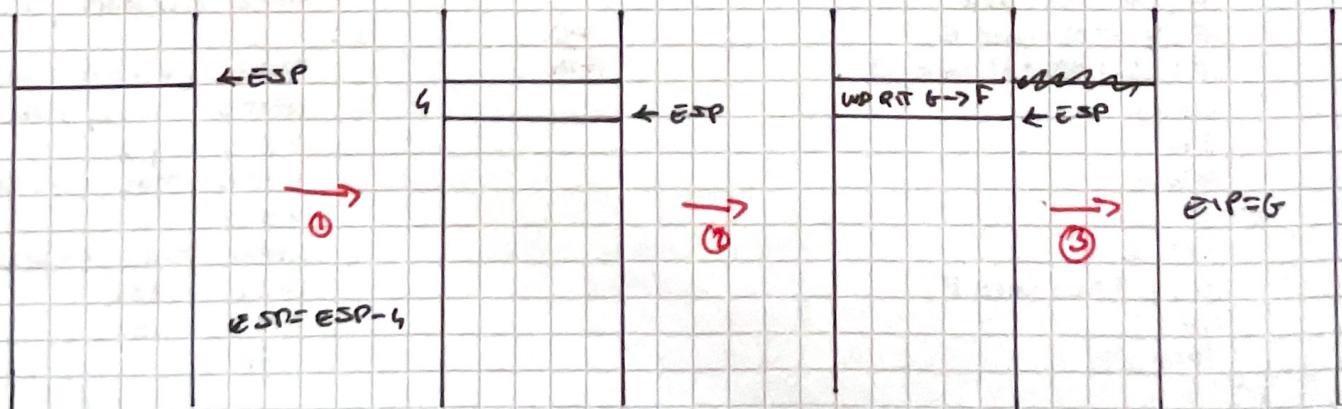
- 1) COPIA DEI REGISTRI { EBX, EDI, ESI, EBP }
- 2) VARIABILI LOCALI
- 3) PARAMETRI PER LA FUNZIONE CHIAMATA
- 4) INDIRIZZO DI RITORNO DELLA FUNZIONE CHIAMATA

INT f() {  
 INT x = g();  
 x = x + 1;  
 RETURN x;  
}

CALL g  
INCL %EAX  
RET

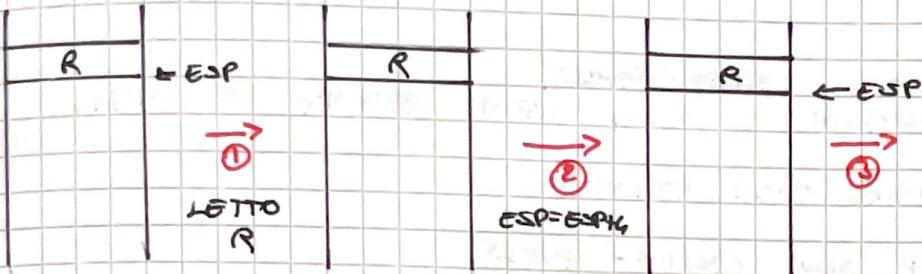
## ISTRUZIONE CALL

- CALL FN
- ① RISERVA 4 BYTE NELLA STACK (SUB \$4, %ESP)
  - ② SERVE INDIRIZZO DI RITORNO PER FN NELLO SPAZIO RISERVATO PRIMA
  - ③ EIP = FN

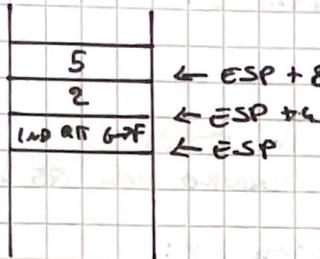


## ISTRUZIONE RET

- RET
- ① LEGGE UN INDIRIZZO (4 byte) (R a ESP)
  - ② DEALLOCATE 4 byte dello stack (ADDL \$4, %ESP)
  - ③ EIP = R



WT f() {  
  INT x = g(2,5);  
  x = x + 1;  
  RETURN x;  
}

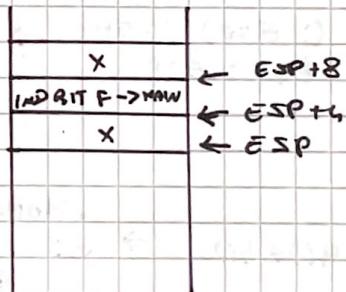


.GOBL f  
f:  
  SUBL \$8, Y.ESP PROLOGO  
  movl \$2, (Y.ESP)  
  movl \$5, S(Y.ESP)  
  CALL g  
  INCL Y.EAX  
  ADDL \$8, Y.ESP EPILOGO  
  RET PRIMA DELA RET

ES:  
INT f(WT x){  
  INT n = g(x);  
  n = n + 1;  
  RETURN n;  
}

LEGGE ARG. DI  
F DALLA STACK  
COPIA QUANTO  
LETTTO NEL PUNTO  
CHE DEVE CONTENERE  
PARAMETRO DI G

.GOBL f  
f:  
  SUBL \$4, Y.ESP  
  movl 8(Y.ESP), Y.EAX  
  movl Y.EAX, (Y.ESP)  
  CALL g  
  INCL Y.EAX  
  ADDL \$4, Y.ESP  
  RET



ES:  
INT f() {  
  RETURN g() + h();

WT f() {  
  INT a = g();  
  ~~INT b = h();~~  
  RETURN a;  
  
  INT c = a;  
  a = h();  
  a = c;  
  RETURN a;

.GOBL f  
f:  
  CALL g  
  movl Y.EAX, Y.ECX  
  CALL h  
  ADDL Y.ECX, Y.EAX  
  RET

NON FUNZIONA!

## REGISTRI ABI IA32

1) I REGISTRI A, C, D SONO CALLE - SAVED  
 ↳ LA FUNZIONE CHIAMANTE SI SALVA QUESTI REGISTRI SE CHIAMA UNA FUNZIONE.  
 CHIAMATA PUÒ SPORCARIE QUESTI REGISTRI

2) I REGISTRI B, DI, SI, BP SONO CALLEE - SAVED,  
 ↳ CHIAMATA NON PUÒ SPORCARLI  
 CHIAMATA SE VUOLE UTMIZZARLI DEVE PRESERVARLI

PRESERVARE REGISTRO:

- A SALVA VALORE DEL REGISTRO IN STACK
- B LEGGI VALORE DALLA STACK E SCRIVI NEL REGISTRO

STRADA 1:

A `subl $4, %ESP`  
`movl %EBX, (%ESP)`

⇒ PUSH REG

$$ESP = ESP - 4$$

COPIO VALORE DI REG NELLO SPAZIO AMMOSATO PRIMA

B `movl (%ESP), %EBX`  
`ADDR $4, %ESP`

⇒ POP REG

LEGGO DA STACK UN VALORE E LO SCRIVO IN REG  
 $ESP = ESP + 4$

STRADA 2:

ES:

INT f() {  
 RETURN g() + h(); } → .GLOBAL f

CALL g  
 PUSHL %EBX  
 movl %EAX, %EBX  
 CALL h  
 ADDL %EBX, %EAX  
 POPL %EBX  
 RET

ES:

INT R(INT x) {

INT y=1; INT z=2;  
 INT n = g(x+1);  
 INT a = n+y+z  
 RETURN a;

.GLOBAL R

R:

PUSHL %EDI  
 PUSHL %EBP  
 SUBL \$4, %ESP

} PROLOGO

movl \$1, %EDI  
 movl \$2, %EBP  
 movl 16(%ESP), %EAX  
 WCL %EAX

movl %EAX, (%ESP)

CALL g

ADDL %EDI, %EAX

ADDL %EBX, %EAX

ADDL \$4, %ESP

POPL %EBP

POPL %EDI

RET

PUSH E POP  
 VANNNO GESTITI IN  
 ORDINE INVERSO

X	
IND. RET.	← ESP + 16
EDI	← ESP + 12
EBP	← ESP + 8
EBP X+1	← ESP + 4
IND. RET	← ESP

HO UNA FUNZIONE f CHE → HO A, C, D  
NON CHIAMA ALTRE FUNZIONI → SOLO SE NON BASTANO USO B, D, S, BP  
(MA LE DEVO PRESERVARE)

HO UNA FUNZIONE f CHE → B, D, S, BP + A, C, D + A, C, D  
FA CHIAMATE (PRESERVANDOLE) PER VALORI TEMPORANEE PER VARIABILI NON TEMP.  
(PRESERVANDOLE)

**ES:**

```
INT f(INT x){  
    RETURN g(x) + h(x); } → INT f(INT x){  
    INT c = x;  
    INT a = g(c);  
    INT b = a;  
    c = x;  
    INT a = h(c);  
    a = b + a;  
    RETURN a; }
```

Abbiamo 2 chiamate allo stesso param.

.globl f  
⇒ f:  
 pushl y.EBX / } PROLOGO  
 subl \$4, y.ESP  
 movl 12(y.ESP), y. ECX  
 movl y. ECX, (y.ESP)  
 call g  
 movl y.EAX, y.EBX  
 movl 12(y.ESP), y. ECX  
 movl y. ECX, (y.ESP)  
 call h  
 addl y.EBX, y.EAX  
 addl \$4, y.ESP } EPILOGO  
 popl y.EBX  
 ret

USIAMO 12(y.ESP) POICHÉ PUSML E SUBL \$4 SPOSTANO TUTTO DI G  
GASANO, QUINDI IL PARAMETRO X  
SARÀ A 12(y.ESP) INVECE DI 4(y.ESP)

**ES**

```
UNSIGNED f(INT *v, UNSIGNED n){  
    UNSIGNED i, cnt = 0;  
    FOR (i = 0; i < n; i++)  
        cnt += IS_NEGATIVE(v[i]);  
    RETURN cnt; }
```

.globl f  
f:  
 pushl y.EBX  
 pushl y. EDI  
 pushl y. ESI  
 pushl y. EBP } PROLOGO  
 subr \$4, y.ESP  
 movl 24(y.ESP), y. EBP  
 movl 28(y.ESP), y. EDI  
 xorl y. ESI, y. ESI  
 xorl y. EBP, y. EBP  
 L: cmpl y. EDI, y. ESI  
 jae E  
 movl (y. EBP, y. ESI, 4), y. ECX  
 movl y. ECX, (y.ESP) ————— SPOSTATO ECX IN ESP  
 call IS\_NEGATIVE  
 addl y. EAX, y. EBP  
 incl y. ESI  
 jmp L  
E: movl y. EBP, y. EAX  
 addl \$4, y.ESP } EPILOGO  
 poshl y. EBP  
 poshl y. ESI  
 poshl y. EDI  
 poshl y. EBP  
 ret

```
UNSIGNED f(INT *v, UNSIGNED n){  
    INT *EBP = v;  
    UNSIGNED EDI = n;  
    UNSIGNED ESI = 0; // i  
    UNSIGNED EBX = 0; // cnt  
    L: IF (ESI >= EDI) GOTO E;  
    INT c = EBP[ESI];  
    INT a2 = IS_NEGATIVE(c);  
    EBX += a2;  
    ++ESI;  
    GOTO L;  
E: ;  
    a = EBX;  
    RETURN;
```

ES:

```

UNSIGNED p (INT* v, UNSIGNED n) {
    UNSIGNED b = 1; // in
    UNSIGNED d = n;
    INT* s1 = v;
L: IF (b >= d) GOTO E;
    INT c = s1[b-1];
    INT d = s1[b];
    INT a = ORDINATI (c, d); =>
    IF (a != 0) GOTO F;
    a = 0;
    GOTO E2;
F: D++;
    GOTO L;
E: a = 1;
E2: RETURN;

```

PERCHÉ USIAMO	4 IND RIT
	4 EBX
	4 EDI
movl %eax, (%esp)	4 ESI
? 4	4 EDX
	4 ECX ← ESP

L ↴

GLOBL p  
R:  
PUSHL %EBX  
PUSHL %EDI  
PUSHL %ESI  
SUBL \$8, %ESP  
movl \$1, %EBX  
movl 28(%ESP), %ESI  
movl 24(%ESP), %ESI  
L: CMPL %EDI, %EBX  
JAE E,  
**NOTE:**  
movl -4(%ESI, %EBX, 4), %ECX  
movl (%ESI, %EBX, 4), %EDX  
movl %ECX, (%ESP)  
movl %EDX, 4(%ESP)  
CALL ORDINATI  
CMPL \$0, %EAX  
jne F  
XORL %EAX, %EAX  
JMP E2  
F: INCL %EBX  
JMP L  
E: MOVW \$1, %EAX  
E2: RET

## CONVERSTIONE DI TIPO

### • SCENARIO UPCAST (PROTOSIOME)

CHAR c = 5; } OGGETTO 8 bit "DIVENTA" A 32 bit → movs  
INT a = c;

UNSIGNED CHAR q = 5; } OGGETTO 8 bit "DIVENTA" A 32 bit → movz  
UNSIGNED INT a = c;

IL VALORE DI QUESTO A 32 bit DEVE RITENERE UGUALE A QUESTO PIÙ 8 bit  
QUINDI NON C'È PERDITA DI INFORMAZIONE

### • SCENARIO DOWNCAST (TRONCAMENTO)

INT c = 0xCAFE;  
CHAR q = c; } 32 bit → 8 bit

UNSIGNED c = 0xCAFE; } 32 bit → 8 bit  
UNSIGNED char q = c;

ABBIATO UNA PERDITA DI INFORMAZIONE (IL BIT MENO SIGNIFICATIVO)

## ISTRUZIONI DI CONVERSIONE

### • UP CAST (8 → 32)

MOV S, P	MOVS (b,l) S,D
MOVZ S,D	MOVZ (b,e) S,D

### • DOWN CAST (32 → 8)

USIAMO LA SOLITA movb

## CONVERSIONE CON SEGNO

Q=L	CHAR C	SHORT C	INT C
CHAR a	movb	movb	movb
SHORT a	movsbw	movw	movw
INT a	movsbl	movwl	movl

- LA SORGENTE NON PUÒ ESSERE IMMEDIATO
- DESTINAZIONE PUÒ ESSERE SOLO REGISTRO

movS GUARDA IL MSB (MOST SIGNIFICANT BIT)  
DELLA SORGENTE E REPLICÀ IL MSB SU  
NUOVI BIT DA GENERARE

## CONVERSIONE SENZA SEGNO

Q=L	UNSIGNED CHAR C	UNSIGNED SHORT C	UNSIGNED INT C
UNS. CHAR a	movb	movb	movb
UNS. SHORT a	movzbw	movw	movw
UNS. INT a	movzbl	movwl	movl

movz AGGIUNGE BIT NUOVI SEMPRE A 0

ES:

```
INT f (CHAR *p, CHAR *q){
    INT *c = p;
    INT *d = q;
    CHAR cl = *c;
    CHAR dl = *d;
    INT c2 = cl;
    INT d2 = dl;
    INT a = d2;
    a = a + c2;
    RETURN;
}
```

} PROMOZIONE ⇒  
OPERAZIONI ARITMETICHE  
SI FANNO A 32 BIT

GROBL R  
R:

```
movl 4(Y.ESP), Y.ECX
movl 8(Y.ESP), Y.EDX
movb (Y.ECX), Y.CL
movb (Y.EDX), Y.DL
movsbl Y.CL, Y.ECX
movsbl Y.DL, Y.EDX
movl Y.ECX, Y.EAX
ADDL Y.EDX, Y.EAX
RET
```

ES:

```

INT num_vowels(CHAR *s){
    INT CNT=0;
    WHILE (*s)
        CNT += IS_VOWEL (*s++);
    RETURN CNT;
}

```



```

INT num_vowels(CHAR *s){
    CHAR *S1 = s;
    INT b=0;
L: IF (*S1 != 0) GOTO E;
    CHAR CL = *S1;
    INT C = CL;
    S1++;
    INT A = IS_VOWEL(C);
    b+=A;
    GOTO L;
E: a=b
RETURN a

```

.GLOBL NUM\_VOWELS  
NUM\_VOWEL:  
PUSHL Y.EBX  
PUSHL Y.ESI  
SUBL \$4, Y.ESP  
MOVL 16(Y.ESP), Y.ESI  
XORL Y.EBX, Y.EBX

⇒ L:

CMPB \$0, (Y.ESI)  
JE E  
MOVBL (Y.ESI), Y.CL  
MOVSBBL Y.CL, Y.ECX  
INCL Y.ESI  
MOVL Y.ECX, (Y.ESP)  
CALL IS\_VOWEL  
ADDL Y.EAX, Y.EBX  
JMP L

E:

MOVL Y.EBX, Y.EAX  
ADDL \$4, Y.ESP  
POP L Y.ESI  
POP L Y.EBX  
RET

## VARIABILI LOCALI IN STACK

CI SONO ALMENO 3 CASI IN CUI UNA VARIABILE C VIENE MESSA NELLO STACK

1 HO PIÙ VARIABILI COME REGISTRI

2 VIENE UTILIZZATO OPERATORE "ADDRESS\_OF & INT X"

X NON PUÒ STARE IN UN REGISTRO, PER AVERE INDIRIZZO DEVO METTERE X IN MEMORIA

3 SE HO DUE ARRAY O DELLE STACK

ES:

```

INT f(){
    INT x;
    LOAD(&x);
    RETURN x;
}

```

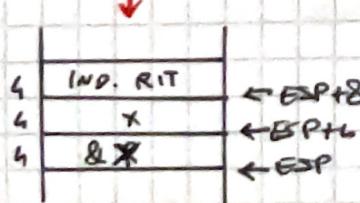
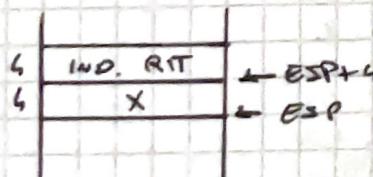
⇒

.GLOBL F  
F:

SUBL \$8, Y.ESP ]①  
MOVL Y.ESP, Y.EAX ]②  
ADDL \$4, Y.EAX  
MOVL Y.EAX, (Y.ESP)  
CALL LOAD  
MOVL 4(Y.ESP), Y.EAX  
ADDL \$8, Y.ESP  
RET

① RISERVO SPAZIO PER:  
-VARIABILE loc. X (8BYTE)  
-PARAMETRO DI LOAD

② CALCOLO INDIRIZZO DI X (&x)  
SENZA ACCEDERE IN  
MEMORIA, LEGGERE, E  
METTERE IN EAX



## ISTRUZIONE LEA (LOAD EFFECTIVE ADDRESS)

EQUIVALE A

```
movl y.ESP, y.EAX
addl $4, y.EAX
```

**LEAL** 4(y.ESP), y.EAX

- A CALLODO INDIRIZZO ESP+4
- B SCRIVE A IN EAX

**VS**

**movl 4(y.ESP), y.EAX**

- A CALLODO INDIRIZZO ESP+4
- B LEGGE INDIRIZZO A
- C SCRIVE B IN EAX

L'ISTRUZIONE LEA VIENE SPESSO "ABUSATA" PER CALCOLARE ESPRESSIONI ARITMETICHE

**LEAL d (y.REG1, y.REG2, SCALA), y.REG3**

$E = d + REG1 + REG2 \cdot SCALA$

ES:

$E = 7 + 20 + 15 \cdot 4 \rightarrow$

```
movl $15, y.ECX
movl $20, y.EDX
LEAL 7(y.EDX, y.ECX, 4), y.EAX
```

ES:

```
INT f (INT x, INT y) {
    INT q, r;
    GET_DIV_Rem (&q, &r, x, y);
    RETURN q;
}
```

VOID GET\_DIV\_Rem (INT\*d, INT\*r, INT\*x, INT\*y);

$\downarrow$

```
INT f (INT x, INT y) {
    INT q;
    INT r;
    INT c = x;
    INT d = y;
    GET_DIV_Rem (&q, &r, x, y);
    INT a = q;
    RETURN a;
}
```

$\Rightarrow$

.glob f  
f:  
 subl \$24, y.ESP
 movl 28(y.ESP), y.ECX
 movl 32(y.ESP), y.EDX
 leal 16(y.ESP), y.EAX
 movl y.EAX, (y.ESP)
 leal 20(y.ESP), y.EBX
 movl y.EAX, 4(y.ESP)
 movl 8(y.ECY), 8(y.ESP)
 movl 12(y.EDX), 12(y.ESP)
 call GET\_DIV\_Rem
 movl 16(y.ESP), y.EAX
 addl \$24, y.ESP
 ret

ES:

```
VOID FRONT_CONCAT (CONST CHAR* SL, INT n) {
    CHAR BUF[SL];
    CHAR *b = BUF;
    CONST CHAR* SL = S;
    INT DI = 0;
    *b = 0;
    INT BP = n;
    L: IF (DI >= BP) GOTO E;
    CONST CHAR* CL = SL[DI];
    STRCAT(b, CL); DI++;
    GOTO L;
E:
    PUTS(b);
}
```

$\Rightarrow$

.glob f  
f:  
 pushl y.ECX
 pushl y.ESI
 pushl y.EDI
 pushl y.EBP
 subl \$520, y.ESP
 leal 8(y.ESP), y.EBX
 movl 540(y.ESP), y.ESI
 xorl y.EDI, y.EDI
 movl 544(y.ESP), y.EBP
 movb \$0, (y.EBX)
 L: cmpb y.EBP, y.EDI
 jge E
 movl (y.ESI, y.EDI, 4), y.ECX
 movl y.EBX, (y.ESP)
 movl y.ECX, (y.ESP)
 ret

movl y.ECX, (y.ESP)
 call STRCAT
 incl y.EDI
 jmp E

E:
 movl y.EBX, (y.ESP)
 call PUTS
 addl \$520, y.ESP
 popl y.EBP
 popl y.EDI
 popl y.ESI
 popl y.EBX

## ESPRESSIONE BOOLEANA

ES:

```
INT f( INT x, INT y ) {  
    RETURN x < y;  
}  
INT a = x;  
INT b = y;  
IF(a >= c) GOTO F;  
a > 1  
GOTO E;  
F: a = 0;  
E: RETURN a;
```

ESISTE UNA VERSIONE  
PIÙ COMPATTA

SET cc D

D ← 0 SE cc=0  
D ← 1 SE cc=1

cc = CONDITION CODE

D → 8bit

R == 0 → ZF = 1  
R != 0 → ZF = 0

SETZ Y.AL → 0 = AL SE ZF == 0  
SETNE Y.AL → 1 = AL SE ZF == 1

ES:

```
INT f( INT x, INT y ) {  
    RETURN x < y;  
}  
GLOBAL f
```

R:

```
MOVBL 4(Y.ESP), Y.ECX  
MOVBL 8(Y.ESP), Y.EDX  
CMPBL Y.ECX, Y.EDX  
SETBL Y.AL ← ! SE x < y  
MOVSBBL Y.AL, Y.ECX  
RET
```

! SE x < y  
o ATRIMENTI

ES:

```
INT BETWEEN( SHORT x, SHORT y, SHORT z ) {  
    SHORT c = x;  
    SHORT d = y;  
    CHAR AL = &x <= &y;  
    C = B;  
    CHAR AM = &x <= &z;  
    AL = AL & AM;  
    INT a = AL;  
    RETURN a;  
}
```

.GLOBAL BETWEEN  
BETWEEN:

```
MOVW 4(Y.ESP), Y.CX  
MOVW 8(Y.ESP), Y.DX  
CMPW Y.CX, Y.DX  
SETBL Y.AL  
MOVW 12(Y.ESP), Y.CX  
CMPW Y.CX, Y.DX  
SETBL Y.AL  
ANDBL Y.AM, Y.AL  
MOVSBBL Y.AL, Y.ECX  
RET
```

ES:

INT IS\_SPACE( CHAR \* s ) {

RETURN A S != NULL & B \*s == " "

## REGOLA DEL CORTOCIRCUITO

- IN CASO DI AND:  $A \& B$

SE A È FALSA, NON DEVI VALUTARE B

- IN CASO DI OR:  $A \text{ || } B$

SE A È VERA, NON DEVI VALUTARE B

SE NON RISPETTO QUESTA  
REGOLA SCOPPIA TUTTO,  
VALE NEL MONDO REALE

## REGOLE DI ALLINEAMENTO E STRUCT

REGOLA DI BASE DI ALLINEAMENTO ESP → DEVE SEMPRE ESSERE UN MULTIPLO DI 4  
(NEL CORSO)

$$\text{ESP \% 4} == 0$$

NEL MONDO REALE → ESP DEVE ESSERE UN MULTIPLO DI 16  
 $\text{ESP \% 16} == 0$

PUSH/POP → OK PER L'INVIARANTE

ALLOCAZIONE SPAZIO →   
PARMETRI OK  
VARIABILI LOCALI BISOGNA SOTTRARRE SEMPRE 4

## STRUCT

```
STRUCT S {  
    CHAR x;  
    SHORT y;  
    INT z;  
}
```

D1: SIZEOF (STRUCT S) ?  
D2: OFFSETOF(y) ?

## REGOLE:

1 UN OGGETTO DI DIMENSIONE X DEVE TROVARSI ALLINEATO AD X  
→ DEVE "INIZIARE" IN MEMORIA AD UN INDIRIZZO MULTIPLO DI X

CHAR x → ALLINEATO A 1

SHORT y → ALLINEATO A 2

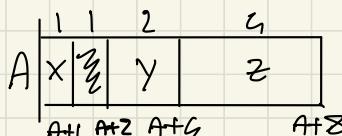
INT z → ALLINEATO A 4

2 UNA STRUCT HA UN ALLINEAMENTO CHE DERIVA DALLA DIMENSIONE DEL CAMPO PIÙ GRANDE → STRUCT S DEVE ESSERE ALLINEATO A 4

3 DIMENSIONE COMPLESSIVA DI UNA STRUCT DEVE ESSERE MUL TIPLA DELLA DIMENSIONE DEL CAMPO PIÙ GRANDE

$$\text{SIZEOF (STRUCT S)} \% 4 == 0$$

4 IL COMPILATORE NON PUÒ RIORDINARE I CAMPI DI UNA STRUCT



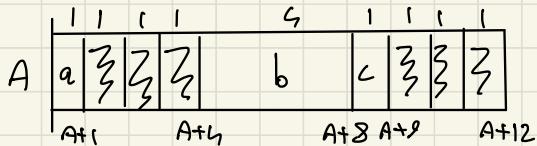
IL SECONDO SPAZIO  
LO RIEMPIATO CON DEL  
PADDING

$$\begin{aligned}\text{SIZEOF(STRUCT S)} &== 8 \\ \text{OFFSETOF(y)} &== 2\end{aligned}$$

ES: STRUCT S {

    CMAR a;  
    INT b;  
    CMAR c;

}



SIZE = 12

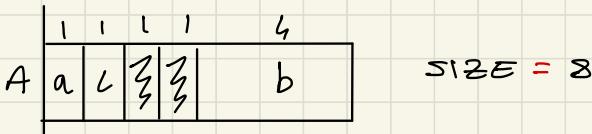
PER LA REGOLA 3 AGGIUNGONO 3 byte DI PADDING

MA POSSIAMO RISPARMIARE SPAZIO RIDAGANIZZANDO I CAMPI:

STRUCT S {

    CMAR a;  
    CMAR c;  
    INT b;

}

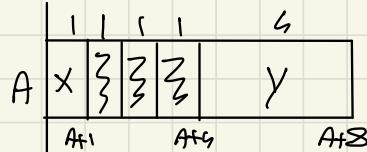


SIZE = 8

ES: STAVG S {

    CMAR X;  
    INT Y;

}



void f(STRUCT S \*p, CMAR x, INT y){

    p → x=x;

    p → y=y;

}



```

movl 4(%ESP), %EAX
movb 3(%ESP), %CL
movl 12(%ESP), %EDX
movb %CL, (%EAX)
movl %EDX, 4(%ESP)
RET

```

ES: TYPEDEF STRUCT NODO NODO;

STRUCT NODO {

    INT ELEM; //

    NODO \*NEXT; //

}

INT SUM (NODO \*p){

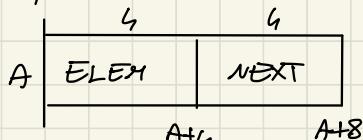
    INT CNT=0;

    FOR ( ; p; p=p→NEXT)

        CNT += p→ELEM;

    RETURN CNT;

}



NODO \*d=p;

INT a=0;

→ L: IF (d==NULL) GOTO E; →

INT c = d→ELEM;

a=a+c;

d=d→NEXT;

GOTO L;

E: RETURN a;

.GLOBAL SUM

SUM:

```

movl 4(%ESP), %EDX
xorl %EAX, %EAX
li %EDX, %EDX
je E
movl (%EDX), %EDX
addl %EDX, %EDX
movl 4(%EDX), %EDX
jmp L
E: RET

```

```

ES: TYPEDEF STRUCT { //BASE
    CHAR GENDER; //OFFSET
    LCHAR* NAME; //OFFSET
    LCHAR AGE; //OFFSET
} PERSON_T; //SIZEOF

```

	1	1	1	1	S	1	1	1	1
A	g	z	z	z	NAME	A	z	z	z
	A+1		A+5			A+8	A+9		A+12

```

PERSON_T *FWD_YOUNGEST(PERSON_T *v, INT n) {
    PERSON_T *a = NULL;
    PERSON_T *c = v;
    INT d=0;
    L: IF (d>=n) GOTO E;
    IF (a==NULL) GOTO A;
    CHAR b = c[d].AGE;
    IF (b >= a->AGE) GOTO F;
    A: a = &c[d];
    F: d++;
    GOTO L;
    E: RETURN a;
}

```

.GLOBAL FWD\_YOUNGEST  
FWD\_YOUNGEST:

```

PUSHL 1/EBX
XORL 1/EAX, 1/EAX
MOVL 8(-1/ESP), 1/ECX
XORL 1/EDX, 1/EDX
L: CMPL 12(-1/ESP), 1/EDX
JGE E
TESTL 1/EAX, 1/EAX
JNE A
MOVBL 3(1/ECX), 1/BL
CMPB 8(1/EAX), 1/BL
JGE F
A: MOVL 1/ECX, 1/EAX
F: INCL 1/EDX
ADDL $12, 1/ECX
JMP L
E: POOL 1/EBX
RET

```

ES:

```

TYPEDEF STRUCT {
    CHAR PRIVILEGI;
    SHORT USER_ID;
} STUD_T;

```

	1	1	2
A	P	z	USER

SIZE = 4

```

VOID GET_UTENTE (CHAR* PRIVILEGI, SHORT* USER_ID){
    STUD_T UTENTE;
    STUD_T *a = &UTENTE;
    CURR_USER(a);
    CHAR *c = PRIVILEGI;
    LCHAR d = UTENTE.PRIVILEGI;
    *c = d;
    SHORT *c2 = USER_ID;
    SHORT d2 = UTENTE.USER_ID;
    *c2 = d2;
}

```

.GLOBAL GET\_UTENTE

GET\_UTENTE:

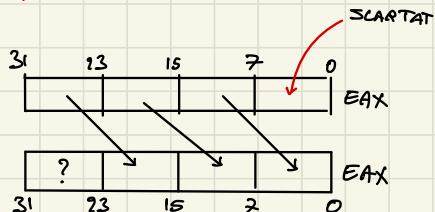
```

SUBR $8, 1/ESP
LEAL 4(-1/ESP), 1/EAX
MOVL 1/ESP, (-1/ESP)
CALL CURR_USER
MOVL 12(-1/ESP), 1/ECX
MOVBL 4(-1/ESP), 1/DL
MOVBL 1/ECX, (-1/ECX)
MOVL 16(-1/ESP), 1/EDX
MOVW 2(-1/ECX), 1/DX
MOVW 1/DX, (-1/ECX)
ADDL $8, 1/ESP
RET

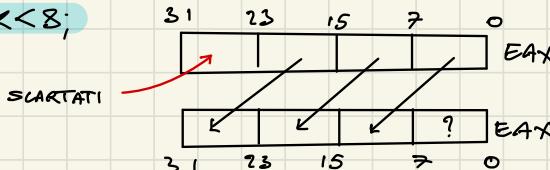
```

## ISTRUZIONE SHIFT

INT a = ..;  
a = a >> 8;



a = a << 8;



## SECONDO $\Rightarrow$ SHIFT ARITMETICO

SAL	S, D	D	$\leftarrow$	D << S	SAR/L = SHIFT ARITHMETIC LEFT/RIGHT
SAR	S, D	D	$\leftarrow$	D >> S	

## SENZA SECONDO $\Rightarrow$ SHIFT LOGICO

SHL	S, D	D	$\leftarrow$	D << S
SMR	S, D	D	$\leftarrow$	D >> S

I NUOVI BIT SONO MESSI A ZERO

ES: INT c = 0xABCDCAFE;  
INT d = 0xAB<sub>3</sub>DCAFE<sub>5</sub>;

c = c << 8;  $\rightarrow$  SAL \$8, %ECX  
d = d << 8;  $\rightarrow$  SHLL \$8, %EDX

c = c >> 8;  $\rightarrow$  SARL \$8, %ECX  
d = d >> 8;  $\rightarrow$  SMRL \$8, %EDX

ECX = 0x ABCDCAFE00  
EDX = 0x ABCDCAFE00 } SHIFT A SINISTRA

ECX = 0xFFABCDC0A  
EDX = 0x00ABCDC0A } SHIFT A DESTRA

a >> x  $\rightarrow$  a / 2<sup>x</sup>

a << x  $\rightarrow$  a \* 2<sup>x</sup>

ES: INT f(INT x, INT y) {  
INT a = x;  
INT c = y;  
INT d = a;  
d = d >> 31;  
a = d : a/c;  
RETURN a;  
}

.GLOBAL f  
f:  
movl 4(%ESP), %EAX  
movl 8(%ESP), %ECX  
movl %EAX, %EDX  
SARL \$31, %EDX  
IDIVL %ECX  
RET

ES: `UNSWIGNED SWAP_ENDIANNESS (UNSIGNED X) {`

```
UNSWIGNED d=x;
UNSWIGNED a=d;
a=a & 0x000000FF;
a=a<<24;
UNSWIGNED l=x;
l=l & 0x0000FF00;
c=l<<8;
a=a|c;
c=x;
c=c & 0x00FF0000;
c=c>>8;
a=a|c;
l=d;
c=c&FF000000;
c=c>>24;
a=a|c;
RETURN a;
```

}

→

. GLOBL SWAP\_ENDIANNESS

SWAP\_ENDIANNESS:

```
movl 4(%ESP), %EAX
movl %EDX, %EAX
andl $0x000000FF, %EAX
jml $94, %EAX
movl %ECX, %ECX
andl $0x0000FF00, %ECX
smll $8, %ECX
orl %ECX, %EAX
movl %EDX, %ECX
andl $0x00FF0000, %ECX
shrl $8, %ECX
orl %EDX, %EAX
andl $0xFF000000, %EAX
shrl $24, %ECX
orl %EAX, %EAX
RET
```

## ISTRUZIONE MOV CONIZIONALE

`CMOVCC S,D | D ← { S SE CC=1  
D SE CC=0 }`

S NON PUÒ ESSERE IMMEDIATO

D DEVE ESSERE UN REGISTRO  
16 bit / 32 bit

ES: `IF(a>b) y=x; → CMPL B,A  
CMOVL X,Y`

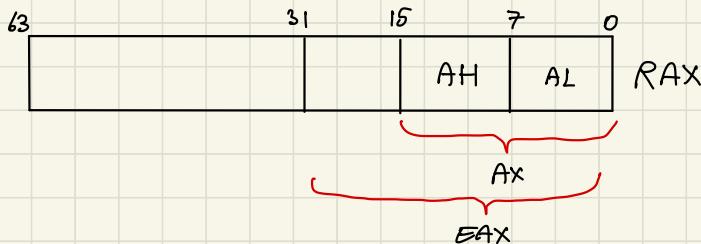
ES: `INT MAX( INT X, INT Y){  
 RETURN (X>Y) ? X:Y;  
}` → `INT MAX( INT X, INT Y){  
 INT a=x;  
 INT c=y;  
 IF (a <= c) a=c;  
 RETURN a;  
}`

. GLOBL MAX

MAX: `movl 4(%ESP), %EAX  
movl 8(%ESP), %ECX  
cmpl %ECX, %EAX  
cmovel %ECX, %EAX  
RET`

## ESTENSIONE 64 BIT

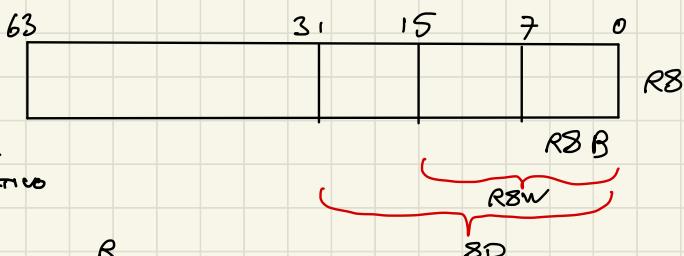
AMD 64  $\Rightarrow$  x86-64



### I REGISTRI SONO:

RBX, RCX, RBP, RSP, RDI, RSI, RIP (SOLITI)

R8, R9, R10 - R15 (NUOVI)



PIATTAFORMA: Linux x86-64 (ABI) SYSTEM V AMD64

SUFFISSI:

- VALORE DI RITORNO  $\rightarrow$  REGISTRO A
- PASSAGGIO DEI PARAMETRI:

1° ARG  $\rightarrow$  RDI  
 2° ARG  $\rightarrow$  RSI IL RESTO  
 3° ARG  $\rightarrow$  RDX IN  
 4° ARG  $\rightarrow$  RCX STACK  
 5° ARG  $\rightarrow$  R8  
 6° ARG  $\rightarrow$  R9

- CALLEE-SAVED  $\rightarrow$  B, BP, R12-R15
- CALLER-SAVE  $\rightarrow$  TUTTI GLI ALTRI

b	8 bit	CCHAR
w	16 bit	SHORT
l	32 bit	INT
q	64 bit	LONG

ES:

UNSGNED f ( INT \*v, UNSIGNED n ) {

  UNSIGNED EBx = 0; //i  
  UNSIGNED n1d = 0; //CNT

  UNSIGNED n13d = n;

  INT \*n16 = v;

L: IF (EBX >= n13d) GOTO E;

  INT EDI = n16[EBX];

  INT EAX = IS\_NEGATIVE(EDI);

  n12d += EAX;

  ++EBX;

  GOTO L;

E: a = n12d;  
  RETURN a;

}

.GLOBAL f

```

f: PUSH q Y. RBX
  PUSH q Y. n16
  PUSH q Y. n13
  PUSH q Y. n12
  XORQ Y. RBX, Y. RBX
  XORL Y. n2d, Y. n12d
  MOVQ Y. RS1, n13
  MOVQ Y. n12, Y. n16
  LEAQ Y. n13, Y. RBX
  JAE E
  MOVL (Y. n16, Y. RBX, 4), Y. EDI
  CALL IS_NEGATIVE
  ADDR Y. EAX, Y. n2d
  INCL Y. RBX
  SJP L
E: MOVL Y. n2d, Y. EAX
  POPQ Y. n12
  POPQ Y. n13
  POPQ Y. n14
  POPQ Y. RBX
  RET
  
```

## OTTIMIZZAZIONE DI CODICE

QUANDO SERVE OTTIMIZZARE UN PROGRAMMA?

- 1) IDENTIFICARE I COLLI DI BOTTEGLIA (HOT SPOT) IN CUI SI CONCENTRA IL GROSSO DEL LAVORO DI UN PROGRAMMA
- 2) STUDIARE L'IMPATTO CHE L'OTTIMIZZAZIONE DI UNA PARTE HA SULLE PRESTAZIONI DELL'INTERO PROGRAMMA

### METRICHE DI PRESTAZIONE:

- LATENZA: TEMPO RICHIESTO PER IL COMPLETAMENTO DI UN'OPERAZIONE
- THROUGHPUT: OPS COMPLETATE NELL'UNITÀ DI TEMPO (OPS = n OPERAZIONI)
- SPAZIO: MEMORIA RICHIESTA
- ENERGIA: CONSUMO ENERGETICO

EVENTO	LATENZA	LATENZA SCALATA
CICLO DI CLOCK / ACCESSO REGISTRO	0.6 ns	1 s
ACCESSO IN MEMORIA (CACHE L1)	0.9 ns	2 s
CACHE L2	2.8 ns	3.5 s
CACHE L3	28 ns	1 min
DRAM	100 ns	4 min
DISCO SSD	50/100 μs	1.5 - 4 giorni
DISCO HDD	1 - 10 ms	1-9 mesi
INVIO PACCHETTO USA	65-180 ms	5 - 11 anni

movl \$10, (%eax)



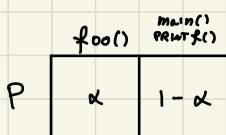
SE VA MALE IN L1  
VADO AVANTI  
FACENDO L'ACCESSO  
IN L2

### SPEEDUP: IMPATTO DI OTTIMIZZAZIONE

$$\text{SPEEDUP} = \frac{\text{TEMPO PROGRAMMA NON OTTIMIZZATO}}{\text{TEMPO PROGRAMMA OTTIMIZZATO}} = \frac{T}{T'}$$

$$\text{SPEEDUP} = 2x$$

$$s = \frac{100}{50} = 2x$$



$\alpha$  = TEMPO RICHIESTO DALLA PORZIONE DI CODICE  
 $1 - \alpha$  = TEMPO SUL RESTO

### LEGGE DI Amdahl

$$\text{SPEEDUP} = \frac{1}{\frac{\alpha}{K} + 1 - \alpha}$$

$$K = \frac{s \cdot \alpha}{s(\alpha-1) + 1}$$

$$\alpha = \frac{K(s-1)}{(K-1)s}$$

d. TEMPO SPESO SU UNA PORZIONE CHE VOGLIAMO OTTIMIZZARE  
k: SPEEDUP OBTENUTO (DOPO L'OTTIMIZZAZIONE) SU QUESUE PORZIONI DI CODICE

### DIMOSTRAZIONE FORMULA:

$$s = \frac{T}{T'} = \frac{\alpha}{\frac{\alpha}{K} + (1-\alpha)} = \frac{1}{\frac{\alpha}{K} + 1 - \alpha}$$

$$\alpha \xrightarrow{\text{OTTIMIZZO}} \alpha' \Rightarrow \frac{\alpha}{\alpha'} = K$$

$$\alpha = 0.5 \quad K = \infty$$

$$S_{TOTALE} = \frac{1}{\frac{0.5}{\infty} + 1 - 0.5} = 2$$

**ES:**  $\alpha = 0.5$   $K = 2x$

$$S = \frac{1}{\frac{0.5}{2} + 1 - 0.5} = 1.33x$$

$\alpha = 0.9$   $K = 6x$

$$S = \frac{1}{\frac{0.9}{6} + 1 - 0.9} = 6x$$

SE  $\alpha$  PESA POCO OTTENGO  
COMUNQUE UNO SPEEDUP POCO ELEVATO  
SPEED UP MAX OTTENIBILE

## PROFILAZIONE DELLE PRESTAZIONI

- SERVE AD IDENTIFICARE I COSTI DI BOTTLIGLIA

METODI

**MANUALE:** INSTRUMENTO (AGGIUNGO CODICE) IL PROGRAMMA PER MISURARE IL "COSTO" DI UNA PORZIONE DI CODICE

**AUTOMATICO:** USANDO DEI TOOL (PROFILER) CALCOLA UN MODO AUTOMATICO DI COME IL TEMPO VENNE SPESO NELLE FUNZIONI

**MANUALE:**

ABBIAMO DIVERSI MODI, DIPENDE DA QD CHE VOGLIAMO MISURARE

USER TIME (TEMPO DEL CODICE SCRITTO DA ME)

TEMPO SPESO SUL MIO PROGRAMMA

SYSTEM TIME (TEMPO SPESO PER ESEGUIRE IL PROGRAMMA DAL SISTEMA)

**AUTOMATICO:**

PERFORMANCE PROFILING AUTOMATICO

→ GPROF

- 1) COMPILARE IL PROGRAMMA CON IL FLAG -pg
- 2) ESEGUIRE IL PROGRAMMA (VIENE CREATO SUL DISCO UN REPORT gmon.out)
- 3) ANALIZZARE IL REPORT CON GPROF ( $\$ gprof ./e$ )

## PROBLEMATICA MISURAZIONE TEMPO

**1) LATENZA:** TEMPO RICHIESTO DALLA FUNZIONE DI MISURAZIONE DEL TEMPO  
↳ SCELGO LA PRIMITIVA GIUSTA IN BASE ALLA GRANDEZZA RICHIESTA

**2) RISOLUZIONE:** MINIMO INTERVALLO MISURABILE  
↳ SCELGO LA PRIMITIVA GIUSTA IN BASE AL TEMPO DELLA FUNZIONE DA MISURARE

PER MISURARE UNA FUNZIONE LA SI CHIAMA TANTE VOLTE E SI FA UNA "MEDIA"

## OTTIMIZZAZIONI

C SONO 2 TIPI:

**ALGORITMO:** RISULTATI PIÙ IMPORTANTI → BUBBLE SORT  $O(n^2)$   
MERGE SORT  $O(n \log n)$   
QUICK SORT  $O(n \log n)$  } SCELTE DEL PROGRAMMATORE

**CODICE:** - RIDURRE LE ISTRUZIONI } SVOLTE SIA DAL PROGRAMMATORE  
- RIDURRE IL COSTO DELLE ISTRUZIONI } CHE DAL COMPILATORE

SE SERVE UN'OTTIMIZZAZIONE CHE PUÒ FARLE IL COMPILATORE  
NON PERDiamo TEMPO E LO FACCIAMO FARLE A LUI

# UN PROGRAMMATORE PROFESSIONISTA PRIMA DELL'EFFICIENZA Pensa A:

- CORRETTEZZA
- PORTABILITÀ
- MANUTENIBILITÀ
- MODULARITÀ → PAGO UN COSTO
- ROBUSTEZZA
- EFFICIENZA → "MONETA" PER PAGARE IL COSTO DI ALTRE QUALITÀ

OSSERVATO 2 MACRO CATEGORIE DI OTTIMIZZAZIONI:



FLAG -Ox:

- O0: DEFAULT, DEBUG
- O1: PRIMO LIVELLO DI OTTIMIZZAZIONE (PIÙ LENTO DI O0)
- O2: SECONDO LIVELLO DI OTTIMIZZAZIONE  
(PIÙ AGGRESSIVO DI O1 MA PIÙ LENTO, NON DOVREBBE AUMENTARE LA DIMENSIONE BINARIO)
- O3: TERZO LIVELLO DI OTTIMIZZAZIONE  
(IL PIÙ AGGRESSIVO, PUÒ FAR CRESCERE NOTEVOLMENTE LA DIMENSIONE DEL BINARIO)
- Os: OTTIMIZZARE PRESTAZIONI MUOVENDO LA DIMENSIONE DEL BINARIO

OTTIMIZZAZIONI DEL WORK:

- CONSTANT FOLDING → ESPRESSIONE COSTANTE VIENE SOSTITUITA DAL SUO VALORE
- CONSTANT PROPAGATION → PROPAGARE NEL CODICE LE COSTANTI
- COMMON SUBEXPRESSION ELIMINATION (CSE) → ELIMINO ESPRESSIONI COMUNI (LO FA IL COMPILATORE)
- LOOP INVARIANT CODE MOTION → C'È UN INVARIANTE DENTRO IL LOOP CHE (NON SEMPRE DA COMPILATORE)
- LOOP UNROLLING →

```
INT f(INT x){  
    INT a=x;  
    for (INT i=0; i<n; i++) →  
        a = a+x;  
    RETURN a;  
}  
  
INT f(INT x){  
    INT a=x;  
    {  
        a = a+x; }  
    a = a+x; }  
    :  
    RETURN a;
```

MI RISPARMIO i, LA JUMP E i++

- FUNCTION IN-LINING → DOVE POSSIBILE SOSTituIRE LA CHIAMATA A FUNZIONE CON IL SUO CORPO

```
INT f(INT x){  
    RETURN x*x;  
}  
  
VOID foo(INT *v, INT n){  
    INT i;  
    FOR(i=0; i<n, i++) →  
        v[i] = f[i];  
}  
  
VOID foo(INT *v, INT n){  
    INT i;  
    FOR(i=0; i<n, i++)  
        v[i] = i*i;
```

IL COMPILATORE LO FA A VOLTE, DIPENDE DA QUANTO È LUNGA LA FUNZIONE O SE VIENE CHIAMATA IN MOLTI PUNTI

OTTIMIZZAZIONE COSTO DELLE ISTRUZIONI:

- REGISTER ALLOCATION → IL COMPILATORE CON -O1, 2, 3 CERCA DI TENERE LE VARIABILI PIÙ USATE NEI REGISTRI PER RIDURRE GLI ACCESSI A MEMORIA (CHE SONO COSTOSI) (AVVIENE AUTOMATICAMENTE)
- OPERATOR STRENGTH REDUCTION →  $x/8 \rightarrow x >> 3$     $x*64 \rightarrow x \ll 6$     $x*2 \rightarrow x \ll 1$     $x*15 \rightarrow (x \ll 4) - x$    LO FA IN AUTOMATICO IL COMPILATORE  
 $10^{12} \rightarrow 2^3$     $2^6$     $x+x$
- DATA ALIGNMENT → COME QUELLA DELLE STRUCT E LA FACCIAMO NOI

## GESTIONE ERRORE

- CODICE DI TERMINAZIONE** → CODICE (INTERO) IDENTIFICA LO STATO DI TERMINAZIONE DI UN PROCESSO
  - 0: NON C'È STATO ERRORE
  - #0: C'È STATO ERRORE

IL CODICE DI TERMINAZIONE VIENE DEFINITO:

- RETURN DAL MAIN
- EXIT / \_EXIT / ABORT

## ERRORE:

OPERAZIONE INVALIDA	RECUPERABILI → CONVENZIONE	VALORE DI RITORNO	IMPORRE LA VARIABILE GLOBALE
- NON RECUPERABILI     EXIT   ABORT	- RECUPERABILI → CONVENZIONE	<ul style="list-style-type: none"> <li>→ PUNTATORE → NULL</li> <li>→ INTERO → 0 SE OK</li> <li>→ ALTRIMENTI</li> </ul>	ERRNO

## GESTIONE DI FILE

DUE MODI:

1 **SYSTEM CALL**: OPEN, CLOSE, READ, WRITE, LSEEK → LAVORANO SU FILE DESCRIPTOR (FD), A BASSO LIVELLO (WRAPPER POSIX)

2 **FUNZIONI LIBC**: fopen, fclose, fread, fwrite, fseek, ftell, fprintf, fgets, etc... → LAVORANO SU STRUCT FILE, AD ALTO LIVELLO

POSIX DEFINISCE PER OGNI PROCESSO TRE CANALI DI COMUNICAZIONE:



	FILE DESCRIPTOR	FILE*
STANDARD INPUT	0	STDIN
STANDARD OUTPUT	1	STDOUT
STANDARD ERROR	2	STDERR

FILE DESCRIPTOR	FILE*	DESCRIZIONE
OPEN	fopen	APRIRE FILE
CLOSE	fclose	CHIUDERE FILE
READ	fread	LEGGE DAI BYTES
WRITE	fwrite	SIRVE DAI BYTES
LSEEK	fseek / ftell	SPOSTARE NELL'FILE
	fprintf	PRINTF IN FILE
	fgets	LEGGI RIGA FILE
	fscanf	SCANF IN FILE

ES:  
INT (\*c)(int);  
      ^ 1 2  
      5 6 7 8

- 1 PARTO DAL DICHIARATORE CHE È LA VARIABILE/TIPO CHE VOGLIO DICHIARARE
- 2 VADO A DX QUANDO "POSSO"
- 3 VADO A SX QUANDO "DEVO"

4 È UN PUNTATORE A FUNZIONE CHE PRENDE UN INTERO E CHE RITORNA UN INTERO

ES:  
INT (\*compar)(const void\*, const void\*);  
      ^ 1 2 3 4  
      5 6 7 8

compar è un puntatore a funzione che prende un puntatore const void e un puntatore a const void che restituisce un intero

STRDUP(STRING S) → ALLOCA DINAMICAMENTE E COPIA DIRETTAMENTE IN POSIZIONE

## PROCESSI

È UN ESECUZIONE DI UN PROGRAMMA

- L'IMMAGINE DI MEMORIA → CODICE + DATI
- LO STATO DELLA CPU
- RISORSE UTILIZZATE
- METADATI → PID E IL PPID

PID = PROCESS IDENTIFIER  
PPID = PARENT PID

COME VENGONO LANCIATI I PROCESSI? ↘

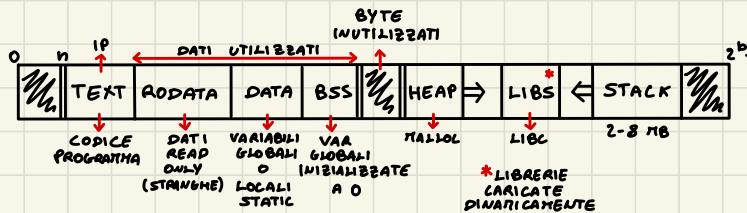
1) RICHIESTA ESPlicita DA UTENTE

ES: DOPPIO CLICK SU ICONA

2) PROCESSO → PROCESSO

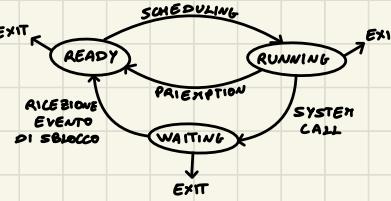
3) RISPOSTA AGLI EVENTI: TRAMITE TIMER CHE ATTIVA SEGNALE

## IMMAGINE DI MEMORIA DI UN PROCESSO



$$b = 32 \text{ bit o } 64 \text{ bit}$$

## STATI DI UN PROCESSO



## ALGORITMO DI SCHEDULING ROUND-ROBIN



PRIORITÀ:



## COMANDO WAIT

PERCHÉ MIO FIGLIO TERMINA?

- ① FIGLIO CHIAMA LA EXIT CON CODICE DI TERMINAZIONE: `exit(35)` → LA SEMANTICA GHELA DIANO NOI
- ② IL KERNEL LO UCCIDE (ES: TANTITE SEGUOLE)



## FLUSSO DI CONTROLLO ELEZIONALE

IL FLUSSO DI CONTROLLO ELEZIONALE VIOLA LE REGOLE DEL FLUSSO NORMALE, METTENDO NELL'IP UN VALORE CHE SPOSTA IL CONTROLLO IN ZONE DI CODICE (TEXT) NON CONFORMI CON IL FLUSSO DI CONTROLLO NORMALE

1. INTERRUPT O INTERRUZIONI
2. SEGNALE

### INTERRUZIONI:

È UNA NOTIFICA INVIATA ALLA CPU VIA HARDWARE VIA SOFTWARE

AD Ogni NOTIFICA LA CPU INVIA IL KERNEL DEL S.O. → INVOCÀ UN GESTORE PER L'INTERRUPT RICEVUTO

⇒ INTERRUPT VECTOR TABLE (INTERRUPT DESCRIPTOR TABLE)

↳ INDIG DEGLI INTERRUPT, PUNTATORI AL CODICE DI GESTIONE

ES: EVENTI CHE GENERANO INTERRUPT:

- CLICK DEL MOUSE
- PRESSIONE TASTO
- TERMINAZIONE DI OPERAZIONI I/O DA SU DISCO
- DIVISIONE PER ZERO
- ACCESO ERRATO A MEMORIA

### 1° DISTINZIONE:

- RECUPERABILI: POSSIBILITÀ DI CONTINUARE IL PROGRAMMA INTERROTTO
- NON RECUPERABILI: ALTREMENTI, IL PROGRAMMA SI INTERROMPE

### 2° DISTINZIONE

- INTENZIONALI: IL SOFTWARE DECIDE DI MANDARE UN INTERRUPT
- NON INTENZIONALI: INTERRUPT VIA HARDWARE

INTERRUPT ASINCRONI: NON SO GUARDO MI ARRIVA LA NOTIFICA DI INTERRUPT  
SONO GENERATI DALL'HARDWARE INVIANDO UN SEGNALE ELETTRICO ALLA CPU MEDIANTE UNO DEI PIEDINI METALLICI CHE SI TROVANO SULLA SCHEDA MADRE  
RISPONDO IN QUESTA CATEGORIA LE INTERRUZIONI GENERATE DA DISPOSITIVI ESTERNI

ESISTONO 3 SOTTO CATEGORIE SINCRONE:

1. TRAP (INTERRUZIONI SOFTWARE)
2. FAULT
3. ABORT

1. TRAP: SONO GENERATE INTENZIONALMENTE DA UN PROGRAMMA E SONO DIREZIONATI VERSO LA CPU  
↳ INT \$0x0 → IDENTIFICATIVO PER LANCiare LE SYSTEM CALL (32 bit) → (64 bit SYSCALL(1))  
(118)<sub>10</sub>

2. FAULT: SONO INTERRUZIONI NON INTENZIONALI GENERATE DALLA CPU E SONO GENERALMENTE RECUPERABILI  
↳ TENTATIVO DI LETTURA/SCRITTURA AD UNA ZONA DI MEMORIA A WI IL PROGRAMMA NON È CONSENTITO ACCEDERE → PAGE FAULT (SIGSEGV)  
↳ DIVISIONE PER ZERO → SIGFPE

3. ABORT: SONO INTERRUZIONI NON INTENZIONALI GENERATI DALLA CPU E SONO NON RECUPERABILI  
↳ SCHERMATA BLU WINDOWS O KERNEL PANIC

### CASI DI USO DI INTERRUPT

1) PREEMPTION E CONTEXT SWITCH TRA PROCESSI

2) INVOCARE LE SYSTEM CALL CON INT 0x80

3) PASSARE DALLO STATO WAITING A READY

1. I SISTEMI MULTITASKING CON TIME SHARING SI BASANO SU INTERRUPT GENERATI DA UN TIMER PER REALIZZARE IL MECANISMO DI PREEMPTION



→ CONTEXT SWITCH O CAMBIO DI CONTESTO

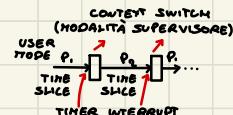
CONTEXT SWITCH:

i. LO STATO DEL PROCESSO INTERROTTO VIENE SALVATO NEL PROCESS CONTROL BLOCK

ii. IL PCB DEL PROCESSO INTERROTTO VIENE INSERITO DAL KERNEL NELLA CODA DEI PCB DEI PROCESSI READY

iii. LO STATO DEL PROCESSO READY DA PORTARE IN ESECUZIONE VIENE IMPOSTATO A RUNNING → AGGIORNATO IL PCB DEL PROCESSO APPENA SCHEDULATO

→ IMMAGINE DI MEMORIA  
→ STATO CPU  
→ PID  
→ PPID  
→ METADATI



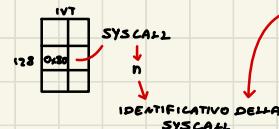
2. SYSTEM CALL E IL SOLO INVOCARNE CON 0x80

INTERRUPT SINCRONI BASATI SU TRAP

↳ CAMBIO MODALITÀ DI ESECUZIONE PASSANDO DA UTENTE A SUPERVISORE, CONSENTENDO DI ESEGUIRE PORZIONI DI CODICE DEL KERNEL IN MODO SICURO E CONTROLLATO

↳ CALL FUNCTION USER MODE (CHIAMATA A FUNZIONE)

↳ INT 0x80



eax	CODICE ID SYSTEM CALL
ebx	1° PARAMETRO
ecx	2° PARAMETRO
...	MAX 6 PARAMETRI

### 3. PASSAGGIO DI STATO DI UN PROCESSO DA WAITING A READY



#### SEGNALI:

UN SEGNALE È UNA NOTIFICA CHE VIENE FATTA DA UN PROCESSO (GENERALMENTE), OPPURE DAL KERNEL ED È INDIRIZZATA AD UN PROCESSO (NO CPU)

	DESTINAZIONE	GESTIONE	MITTENTE
INTERRUPT	CPU PROCESSO	MW/SW SW	DEVICE/CPU/PROCESSO PROCESSO/KERNEL
SEGNALE			

COM'È DEFINITO UN SEGNALE?

1) NUMERO CHE LO IDENTIFICA

2) PID A WI VOGLIO MODIFICARE UN SEGNALE

UN PROCESSO PUÒ INSTALLARE UN GESTORE DI SEGNALI

ESISTONO DEI COMPORTAMENTI DI DEFAULT DI UN SEGNALE (GESTORE NON INSTALLATO)

1. TERM: IL PROCESSO CHE RICEVE "TERM" TERMINA

2. CORE: IL PROCESSO CHE RICEVE "CORE" TERMINA, MA IN PIÙ VIENE GENERATO UN CORE DUMP

3. IGN: IL PROCESSO CHE RICEVE "IGN" IGNORA IL SEGNALE

UN REPORT CHE MI FOTOGRAFIA IL SISTEMA NEL MOMENTO DELL'ERRORE

TIPO SEGNALE	ID	DESCRIZIONE	IGNORABILE	GESTIBILE	DEFAULT
SIGTERM	15	SOFT KILL	SI	SI	TERM
SIGINT	2	CTRL + C	SI	SI	TERM
SIGQUIT	3	CTRL + \	SI	SI	CORE
SIGKILL	9	HARDKILL	NO	NO	TERM
SIGCHLD	17	MUORE FIGLIO	SI	SI	IGN
SIGSEGV	11	ACC. ERRORE MM.	SI	SI	TERM
SIGFPE	8	ERRORE ARITH.	SI	SI	CORE
SIGNALM	16	TIMER	SI	SI	CORE

# INCLUDE <signal.h>

INT KILL (PID, E PID, INT Sig)

↓  
PID DEL PROCESSO A  
CUI IL SEGNALE DEVE  
ESSERE INVIAUTO

RIS: 0 IN CASO DI SUCESSO  
-1 IN CASO DI ERRORE  
ERRORE DESCRITTO  
IN ERNO

ERNO: EPERM  
PERMISSION  
DENIED

ALLA SYSTEM CALL CORRISPONDE L'ONOMINA DA TERMINALE

↪ KILL - SEGNALE PID ES: KILL -SIGTERM 9872  
ID OPPURE ID COSTANTE ASSOCATA  
-16 SIGKILL  
-3 SIGTERM 9872  
-5 SIGPOLL 9872

INSTALLAZIONE DI UN GESTORE A SEGNALE

# INCLUDE <signal.h>

INT SIGACTION (INT SIG, CONST STRUCT SIGACTION \*ACT, STRUCT SIGACTION \*OACT)

NULL OLD

STRUCT SIGACTION:

VOID (\*SA\_HANDLER)(INT SIG) → GESTORE DEL SEGNALE CHE PUÒ ASSUMERE 3 COMPORTAMENTI DIVERSI:

1. CODICE UTENTE DALLA FORMA  
VOID HANDLER (INT SIG)
2. SIG\_IGN: IL SEGNALE VA IGNORATO
3. SIG\_DFL: VIENE RIPARISTINATO IL GESTORE DI DEFAULT

→ RITORNA 0 IN CASO DI SUCESSO  
RITORNA -1 ALTRIMENTI

## LA MEMORIA

È VISTA GENERALMENTE DA UN PROCESSO COME UNO SPAZIO DI INDIRIZZI CONTIGUI  
CONTENTE CODICE, TEXT, VARIABILI, BLOCCHI ALLOCATI DINAMICAMENTE NELL'HEAP ECC...

↳ IL PROBLEMA È CAPIRE QUALE ZONE DELLA MEMORIA FISICA DESTINARE AI VARI PROCESSI

ABBIANO DUE NOMENCLATURE:

POTENZE DI 2-BINARIO (BAN)			POTENZE DI 10-DEGNALE (DISCHI)		
Ki.B	Kibi Byte o Kebi Byte	$2^{10}B = 1024$	KB	KiloByte	$10^3B$
Mi.B	Mibi Byte o Mebi Byte	$2^{20}B = 1.048.576$	MB	MegaByte	$10^6B = 1.000.000$
Gi.B	Gibi Byte o Gebi Byte	$2^{30}B$	GB	GigaByte	$10^9B = 1 \text{ MILIARDO}$

### ALLOCAZIONE DINAMICA DELLA MEMORIA

UN ALLOCATORE DI MEMORIA SUPPORTA 3 PRIMITIVE CHE SONO UTILIZZATE DAI PROGRAMMI PER RICHIEDERE BLOCCHI DI MEMORIA:

- $\text{P = ALLOCA}(n) \rightarrow$  RESTITUISCE L'INDIRIZZO INIZIALE DI UN BLOCCO DI ALMENO  $n$  BYTES CONTIGUI
- $\text{P = RIDIMENSIONA}(p, n) \rightarrow$  RIDIMENSIONA IL BLOCCO  $p$  ALLA NUOVA DIM  $n$
- $\text{DEALLOCA}(p) \rightarrow$  RI LASCA IL BLOCCO  $p$

TUTTI GLI ALLOCATORI FUNZIONANO ATTRAVERSO QUESTE 3 PRIMITIVE

$\text{ALLOCA}(n) \rightarrow \text{MALLOC}(n)$   
 $\text{RIDIMENSIONA}(p, n) \rightarrow \text{REALLOC}(p, n)$   
 $\text{DEALLOCA}(p) \rightarrow \text{FREE}(p)$       MEMORY LEAK SE NON DEALLOCATE  $\rightarrow$  È L'ACCUMULARSI DI BLOCCHI ALLOCATI PRECEDENTEMENTE

### ALLOCAZIONE IN CASCATA

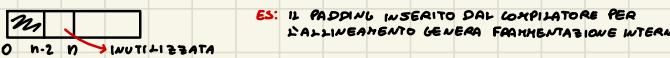
ABBIANO 3 LIVELLI:

1. ALLOCATORE DI SISTEMA: LA MEMORIA FISICA (RAM) DEVE ESSERE ALLOCATA AI PROCESSI IN MODO CHE CIASCUNO ABBIA SPAZIO PER LA PROPRIA IMMAGINE DI MEMORIA (TEXT, DATA, APPDATA, BSS, HEAP, STACK)
2. MALLOC/REALLOC: A SUA VOLTA I PROCESSI DEVONO POTER GESTIRE LO SPAZIO IN HEAP. QUINDI SI GESTISCE LO SPAZIO ATTRAVERSO LE CHIAMATE MALLOC/REALLOC O FREE
3. ALLOCATORE CUSTOM: PARTIZIONA ALCUNI DEI BLOCCHI RESTITUITI DA MALLOC IN BLOCCHI PIÙ PICCOLI PER CONSEGNARE UN USO OTTIMIZZATO DELLO SPAZIO

### FRAGMENTAZIONE INTERNA ED ESTERNA

È UN FENOMENO INDESIDERABILE TALE PER CIÒ CHE ESISTE DELLO SPAZIO LIBERO IN MEMORIA FISICA MA INUTILIZZABILE:

- INTERNA: SI HA QUANDO VI È SPAZIO INUTILIZZATO ALL'INTERNO DI UN BLOCCO PRECEDENTEMENTE ALLOCATO



- ESTERNA: SI MANIFESTA SE UNA RICHIESTA DI ALLOCAZIONE NON PUÒ ESSERE SODDISFATTA CON LO SPAZIO LIBERO SUFFICIENTE POICHÉ NON CONTIGUO. ESISTE SOLO NEI SISTEMI EMBEDDED CHE USANO SOLO MEMORIA FISICA (NON VIRTUALE)



### QUALITÀ DI UN ALLOCATORE

LA BONTÀ DI UN ALLOCATORE SI MISURA SOTTO DUE ASPETTI PRINCIPALI:

1. TEMPO: UN ALLOCATORE DEVE SUPPORTARE IL PIÙ VELOCE POSSIBILE LE OPERAZIONI DI ALLOCAZIONE, RIDUT. E DEALLOCAZIONE
2. SPAZIO: UN ALLOCATORE DEVE RIUSCIRE IL PIÙ POSSIBILE VO SPAZIO PRECEDENTEMENTE DEALLOCATO

FRAGMENTAZIONE  $\rightarrow$  SUDDIVISIONE DELLO SPAZIO LOGICO DI UN ESTERNA  $\rightarrow$  SUDDIVISIONE DELLO SPAZIO LOGICO DI UN PROCESSO CON IL SUO SPAZIO FISICO

I SISTEMI EMBEDDED SOFFRIVANO DI 2 PROBLEMATICHE:

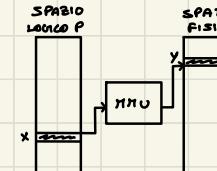
1. MANCANZA DI PROTEZIONE: TUTTI I PROCESSI VEDEVANO TUTTA LA MEMORIA FISICA.

UN PROCESSO POTEVA ACCEDERE AD UNA REGIONE NON SUA

2. FRAGMENTAZIONE ESTERNA

SOLUZIONE

SUDDIVIDERE LO SPAZIO LOGICO DI UN PROCESSO COL SUO SPAZIO FISICO



I PUNTATORI SONO IN P IL PROGRAMMA CHE ESEGUE L'ISTRUZIONE NON HA ALUNA VISIBILITÀ DELL'INDIRIZZO FISICO CORRESPONDENTE  
 MMU: MEMORY MANAGEMENT UNIT





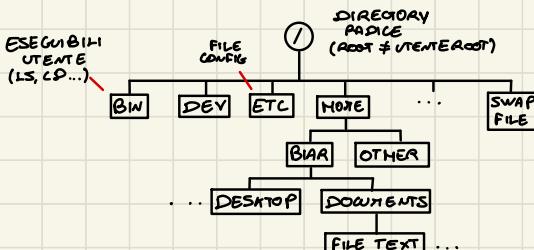
## FILE SYSTEM

È UN'ASTRAZIONE CHE È DEFINITA A LIVELLO DI SISTEMA OPERATIVO E SI OCCUPA DELLA GESTIONE DEI FILES E PEGLI DIRECTORY IN WI RISIEDONO

IL FILE SYSTEM SI BASA SU 2 CONCETTI:

1. **FILE** → ARCHIVIO CHE CONTIENE I BYTE. QUESTI ULTIMI PROVENGONO DA UNA SORGENTE (DISCO)
2. **DIRECTORY** → È UN CONTENITORE DI FILES O DI ALTRE DIRECTORY

IN UNIX LO STANDARD PER APPRESENTARE FILES E DIRECTORY È NOTO COME **FILE SYSTEM HIERARCHY STANDARD**:



/HOME/BIAR/DOCUMENTS/FILETEXT

Ogni file o directory è identificato da un percorso (path) che ne identifica la pos all'interno dell'albero:

- **PERCORSO ASSOLUTO** → ELENCA TUTTE LE DIRECTORY CHE BISOGNA ATTRAVERSARE PER ARRIVARE AL PUNTO DESIDERATO, A PARTIRE DALLA DIRECTORY RADICE
- **PERCORSO RELATIVO** → DESCRIVE LA POS RELATIVA DI UNA DIRECTORY RISPETTO AD UN'ALTRA
  - .. (DOPPIO PUNTO): PERCORSO RELATIVO CHE DENOTA LA DIRECTORY GENITOR
  - . (PUNTO): PERCORSO RELATIVO CHE DENOTA LA DIRECTORY STESSA

## ORGANIZZAZIONE FILE SYSTEM

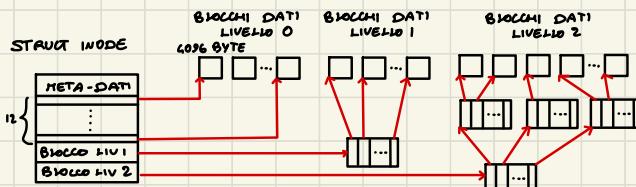
In sistemi Unix-like ad ogni file o directory è associata una struttura dati chiamata **inode**.

**inode** →

- METADATI (PROPRIETÀ DEL FILE)
- INDIG A BLOCCHI DI DATI (INFO RELATIVE SALVATE SU DISCO)

È COMPOSTA DA:

- NUMERO DELL'INODE = INDICE TABELLA DELL'INODE
- TIPO DI FILE = REGOLARE, DIRECTORY
- PERMESSI DEL FILE = LETTURA, SCRITTURA, ESEGIBILE
- NUMERO HARDLINK AL FILE = LNK SIMBOLICO AL FILE
- USER ID / GROUP ID = UTENTE / GRUPPO PROPRIETARIO DEL FILE (es BIAR / STUD)
- DIMENSIONE FILE IN BYTES
- TEMPO ULTIMO ACCESSO / MODIFICA AL FILE
- TEMPO ULTIMA MODIFICA ALL'INODE = SI AGIORNA QUANDO UNA PROPRIETÀ DI SU VIENE MODIFICATA
- RIFERIMENTI diretti ai blocchi = INDIG DEI PRIMI BLOCCHI DEL FILE
- RIFERIMENTO INDIRETTO SINUOSO / DOPPIO A BLOCCHI = INDIG DI BLOCCO CHE CONTIENE INDIG A BLOCCHI DATI / INDIG DI BLOCCO CHE CONTIENE INDIG A BLOCCHI CHE CONTENGONO A SUO VANTO INDIG A BLOCCHI DATI



$$[(12 + 1024 + 1024) \cdot 4096] = 4 \text{ GB} \quad (\text{INAMMISIBILE} \rightarrow \text{SCHEMA AD INDIREZIONE TRIPLO} = 16 \text{ TB})$$

ES:

- UN FILE DI 256 BYTE OCCUPA 1 BLOCCO → SPRECO DI 4096-256 BYTE
- UN FILE DI 16000 BYTE OCCUPA 4 BLOCCI
- UN FILE DI 4 MiB OCCUPA 1024 BLOCCHI SUDDIVISI IN 12 BLOCCI DI LIV 0, 1012 BLOCCI DI LIV 1 E 1 BLOCCO DI INDICE

## PERMESSI

A ELEMENTO FIGLIO DELLA DIRECTORY CORRENTE VENGONO MOSTRATE LE SEGUENTI INFO:

### 1) TIPO DI FILE

- d: DIRECTORY
- : FILE QUALSIASI
- l: LINK SIMBOLICO

### 2) PERMESSI (ULTIMI 9 CARATTERI PRIMA COLONNA)

- 3) NUMERO HARD LINK VERSO UN FILE (SECONDA COLONNA)
- 4) UTENTE PROPRIETARIO DEL FILE (TERZA COLONNA)
- 5) GRUPPO PROPRIETARIO DEL FILE (QUARTA COLONNA)
- 6) DIMENSIONE DEL FILE (QUINTA COLONNA)
- 7) DATA ULTIMA MODIFICA DEL FILE (SESTA E SETTIMA COLONNA)
- 8) DATA ULTIMA MODIFICA DEL FILE (OTTAVA COLONNA)
- 9) NOME FILE O LINK AL FILE (NONA COLONNA)

1) PERMESSI SONO MOSTRATI COME UNA STRANIA DI 9 CARATTERI CHE SPECIFICA I PERMESSI PER 3 ENTITÀ:

- 1) USER = UTENTE PROPRIETARIO (ES BIAR)
- 2) GROUP = GRUPPO PROPRIETARIO (ES STUD)
- 3) OTHER = ALTRI UTENTI

A ENTITÀ SONO POSSIBILI I SEGUENTI PERMESSI:

- 1) r: READ → LETTURA CONTENUTO FILE. SE DIRECTORY, PERMESSO DI ELENCHARE UN ELEMENTO CONTENUTO
- 2) w: WRITE → SCRITTURA DI UN CONTENUTO SU FILE. SE DIRECTORY È AMMESSO RWOMINARE/ELIMINARE/RINUOGLIERE
- 3) x: EXECUTE → ESECUZIONE FILE. SE DIRECTORY ACCESSO CONSENTITO

I PERMESSI SONO SEMPRE VISUALIZZATI NELL'ORDINE **RWX** ED UN SIMBOLO - INDICA L'ASSERVAZIONE DEL PERMESSO IN TALE POS → NOTAZIONE SIMbolica = ESPRIMO ESPLICATIVAMENTE I PERMESSI RWX A ENTITÀ (UGO)

NOTAZIONE OTTale = È POSSIBILE ESPRIMERE I PERMESSI IN UNA FORMA PIÙ COMPATTA.

I PERMESSI RWX POSSONO ESSERE MANTENUTI UTILIZZANDO 3 BIT b<sub>2</sub>, b<sub>1</sub>, b<sub>0</sub> DOVE:

- 1) b<sub>2</sub> SE SETTATO AD 1 INDICA IL PERMESSO DI LETTURA (r)
- 2) b<sub>1</sub> SE SETTATO AD 1 INDICA IL PERMESSO DI SCRITTURA (w)
- 3) b<sub>0</sub> SE SETTATO AD 1 INDICA IL PERMESSO DI ESECUZIONE (x)

SI PUÒ ESPRIMERE LA COMBINAZIONE DEI PERMESSI ASSEGNAZI AD UN'ENTITÀ,  
CALCOLANDO IL NUMERO OTTALI:

ES

b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	
0	0	0	→ 0 NESSUN PERMESSO
0	0	1	→ 1 SOLO X
0	1	0	→ 2 SOLO W
1	0	0	→ 4 SOLO R
0	1	1	→ 3 WX
1	0	1	→ 5 RX
1	1	0	→ 6 RW
1	1	1	→ 7 RWX

PER ESPRIMERE I PERMESSI DI TUTTE E 3 LE ENTITÀ (UGO) AVRÒ BISOGNO SOLAMENTE DI 3 CIFRE OTTALI!

ES

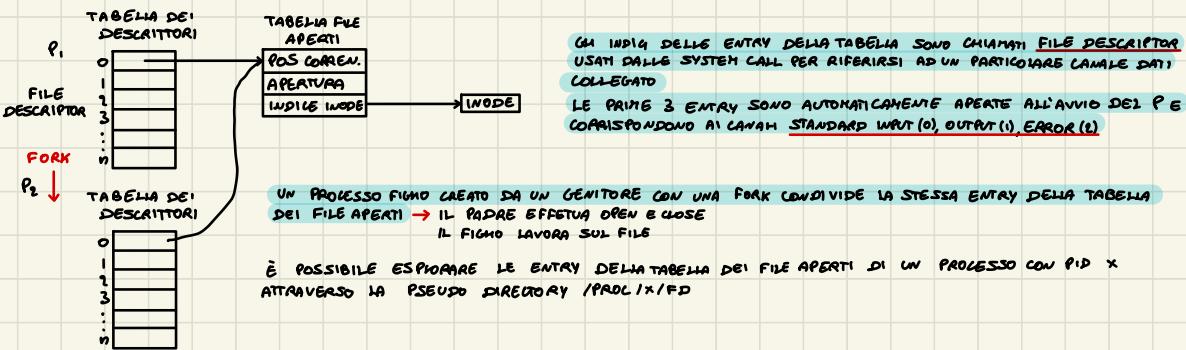
UGO 775 : HO TUTTI I PERMESSI | IL GROUP HA PER BIAR | OTHER HA PERMESSI RX | PERMESSI RWX

PER IMPORTE DEI PERMESSI SI UTILIZZA IL COMANDO CMDHOP

## TABELLA DEI DESCRITTORI DI FILE DI UN PROCESSO

Ogni processo ha una propria tabella chiamata TABELLA DEI DESCRITTORI DI FILE, le cui entry sono puntatori a un'altra tabella di file aperti che contengono varie informazioni, fra cui:

- 1) LA POS CORRENTE NEL FILE
- 2) LA MODALITÀ DI APERTURA
- 3) L'INDICE DELL'INODE DEL FILE



## MOUNT E UNMOUNT DI FILE SYSTEM

I FILE SYSTEM SONO VISTI COME DISPOSITIVI DENTRO LA DIRECTORY /DEV → /DEV/SDA1  
PER VEDERE L'ELENCO DEI FILE SYSTEM MONTATI E SU QUALI DIRECTORY SI USA IL COMANDO DF

## OPERAZIONI SU FILE E DIRECTORY

- OPEN	- OPENDIR
- CLOSE	- CLOSEDIR
- WRITE	LAVORANO SU FILE
- /READ	LAVORANO SU DIRECTORY
- LSEEK*	- READDIR

- \* 1) SEEK\_SET = INIZIO FILE
- 2) SEEK\_CUR = POS CORRENTE
- 3) SEEK\_END = FINE FILE

## LINK A FILE

È POSSIBILE CREARE DEGLI ALIAS CHIAMATI LINK A FILE.

UN LINK È UTILE PER FAR APPARIRE UN FILE IN DIRECTORY DIVERSE SENZA DOVER DUPLICARE I DATI

LE MODIFICHE AL FILE FATTE USANDO IL LINK SONO VISIBILI SIA ACCEDENDO IL LINK CHE IL FILE ORIGINALE

ABBIANO 2 TIPI DI LINK

- SYMBOLIC LINK → UN FILE CHE CONTIENE IL PERCORSO DEL FILE ORIGINALE.

ESSO PUÒ RISIDERE ANCHE SU UN FILE SYSTEM DIVERSO.

SE IL FILE ORIGINALE VIENE SPOSTATO → IL SYMBOLIC LINK DIVENE INDEFINITO

CANCELLARE UN SYMBOLIC LINK NON CANCELLA IL FILE ORIGINALE

UN SYMBOLIC LINK HA UN INODE DIVERSO DAL FILE ORIGINALE

Ln -s PATH FILE ORIGINALE LINK (-s SOFT)

- HARD LINK → MIRROR DEL FILE ORIGINALE, MANO STESSO INODE NUMBER.

HARD LINK E FILE LINKATO DEVONO RISIDERE NELLO STESSO FILE SYSTEM DELL'ORIGINALE  
IL FILE VIENE CANCELLATO QUANDO TUTTI GLI HARD LINK A QUEL FILE SONO STATI ELIMINATI

USIAMO LINK E UNLINK PER LINKARE E ELIMINARE UN HARD LINK

USIAMO SYMLINK PER LINKARE UN SYMBOLIC LINK IN C

2 VIE  
4 LINIEN  
64 BYTE  
LRU



62 413 6200 42 916 400 520

0 6 96 0 14 6 8

0	1	2	3
0 0(0)			
6 0(1)	6(0)		
96 96(0)	6(1)		
0 96(1)	0(0)		
14 14(0)	0(1)		
6 14(1)	6(0)		
8 8(0)	6(1)		

---

$[0, 2^{24} - 1]$

$$\alpha = \frac{K(s-1)}{(k-1)s} = \frac{1.5(1.2-1)}{(1.5-1)1.2} = 0.5 \quad 50\%$$

U G O

U  
R W X

G  
R X

O  
R

U            G            O  
1 1 1        1 0 1        1 0 0

U            G            O  
7            5            4

---

2<sup>4</sup> bit

---

F D E N W

F D E N W

+ + + + +

+ + + + \*

+ + + + +

F D E N W

F D E N W

---

2 VIE      1446      422      908      409      665      345      991

4 LINEE  
128 BYTE

11      3      7      3      5      2      7

0      1      2      3

11		11(0)
3		11(1) 3(0)
7		7(0) 3(1)
3		7(1) 3(0)
5		5(0) 3(1)
2	2(0)	5(1) 3(2)
7	2(1)	6(2) 7(0)

$$\alpha = \frac{K(s-1)}{S(k-1)} = \frac{3.5(1.4-1)}{1.4(3.5-1)} = \frac{3.5 \cdot 0.9}{1.4 \cdot 2.5} = \frac{1.4}{3.5} = 0.4 \quad 40\%$$

2	VIE	105	782	386	1356	11	394	265
4	LINEE	3	29	12	42	0	10	8
32	BYTE							

	0	1	2	3
3			3(0)	
29	29(0)		3(1)	
12	29(1)	12(0)	3(2)	
42	42(0)	12(1)	3(3)	
0	42(1)	0(0)	3(4)	
10	10(0)	0(1)	3(5)	
8	10(1)	8(0)	3(6)	-

40 usec

8 msec

$$27. \quad \left| \begin{array}{l} \\ 60 : x = 2 \cdot 100 \end{array} \right.$$

$$\begin{aligned} \frac{5000}{2} &= 2000 - 40 \\ &= 1960 \\ &1.96 \text{ msec} \end{aligned}$$

SPAZIO LOGICO  $2^{20}$

$2^8 / 2^2 = 2^6$  ENTRY

$$v-d = 9$$

$$d = 20 - 9 = 11 \rightarrow 2^8 = 2048 = 2KB$$

$$V = 32$$

$$2KB = 2^11 \text{ BYTE} \quad d = 11$$

$$V-d = 32-11 = 21 \quad 2^{21} = 2MB$$

$$\text{ENTRY DA } 32 \text{ bit} = 4 \text{ BYTE} \rightarrow 4 \cdot 2MB = 8MB$$


---

$$\begin{array}{ll} \text{DIM PAGINA} & 4KB = 2^{12} \\ \text{DIM SPAZIO} & \text{LOGICO} = [0, 2^{16}-1] \end{array}$$


---

2 VIE	230	56	243	67	361	182	295
4 LINEE	7	1	7	2	11	6	9
32 BYTE							

	0	1	2	3
7			7(0)	
1			7(1)	1(0)
7			7(0)	1(1)
2	2(0)		7(1)	1(2)
11	2(1)		7(2)	11(0)
5	2(2)		5(0)	11(1)
9	2(3)	-	5(1)	9(0)

U60	U (rwx)	U(111)
	G (r x)	G(101)
	O (x)	O(001)

$\rightarrow 0751$

F D E H W  
 F D E H W  
 + + + +  
 + + + +  
 F D E H W  
 F D E H W

---

COMPL ASS	6200	413	62	42	916	400	520
4 LINEE	96	6	0	0	14	6	8
64 BYTE							

	0	1	2	3
96	96(0)			
6	96(1)	6(0)		
0	96(2)	6(1)	0(0)	
0	96(3)	6(2)	0(0)	
14	96(4)	6(3)	0(1)	14(0)
6	96(5)	6(0)	0(2)	14(1)
8	8(0)	6(1)	0(3)	14(2)

---

$$\alpha = \frac{k(s-1)}{(k-1)s} = \frac{2 \cdot 0.5}{1 \cdot 1.5} = \frac{1}{1.5} = 0,666\ldots = 67\%$$

COMPL ASS 62 413 6200 42 916 400 520  
 4 LINEE  
 32 BYTE 1 12 193 1 28 12 16

	0	1	2	3	
1	1(0)				CN
12	1(1)	12(0)			CN
193	1(2)	12(1)	193(0)		CN
1	1(0)	12(2)	193(1)		CN
28	1(1)	12(3)	193(2)	28(0)	CN
12	1(2)	12(0)	193(3)	28(1)	CN
16	1(3)	12(1)	16(0)	28(2)	CAPACITY MISS

# PAGINE = DIM TABELLA / DIM. ELEMENTO

$$\hookrightarrow 2^{20} / 2^2 = 2^{18} \text{ PAGINE} \quad d=18 \quad v=32$$

DIM PAGINA = MEMORIA TOTALE / # PAGINE

$$2^{32} / 2^{18} = 2^14 \text{ BYTES} = 16KB$$

32 bit  $2^{32}$  SPAZIO LOGICO

$$2KB = 2048 \text{ bit} = 2^{11}$$

$$32 \text{ bit} = 4 \text{ BYTE} = 2^2$$

$$2^{11} / 2^2 = 2^9$$

$$v-d=9 \rightarrow d=32-9=23 \rightarrow 2^{23}$$

$$\rightarrow 8388608 = 8MB$$

UGO      U(rwx) → U(1 11)  
 G(rw) → G(110) → 0764  
 O(r) → O(100)

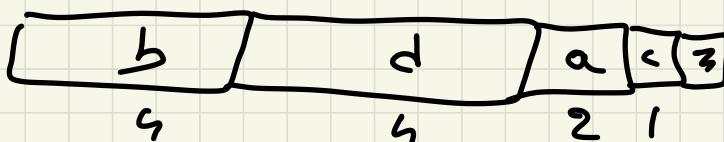
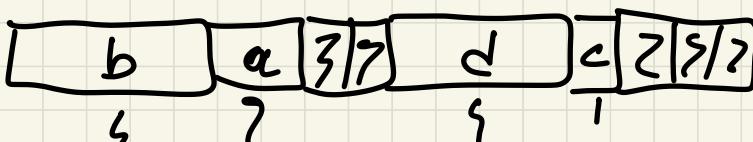
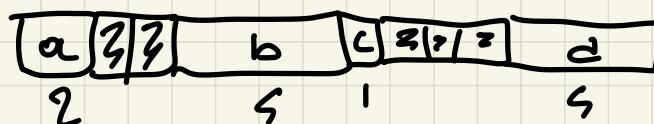
compl ass    37    432    258    793    773    935    268  
 4 LINEE  
 32 BYTE    1    13    8    29    25    29    8

	0	1	2	3	
1	1(0)				CW
13	1(1)	13(0)			CW
8	1(2)	13(1)	8(0)		CW
29	1(3)	13(2)	8(1)	29(0)	CW
25	1(4)	13(3)	8(2)	25(0)	CW
29	29(0)	13(4)	8(3)	24(1)	CW
8	29(1)	13(5)	8(0)	25(2)	CW

$$\alpha = 0.4$$

$$S = 1.25 \times$$

$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1} = \frac{0.5}{0.25} = \frac{1}{2} \rightarrow 50\%$$



$$2^{24} \\ 1MB = 2^{20}$$

$$2^{24} / 2^{20} = 2^4$$

0x34AE20  
0x442AAD  
0x2A9B80

$$0532 \rightarrow \begin{matrix} U(101) \\ r \quad x \end{matrix} \begin{matrix} G(011) \\ w \quad x \end{matrix} \begin{matrix} O(010) \\ w \end{matrix}$$

2 VIE	4200	210	928	583	657	1367	398
4 LINEE	65	3	14	9	10	21	6
64 BYTE							

	0	1	2	3
65			65(0)	
3			65(1)	3(0)
14	14(0)		65(2)	3(1)
9	14(1)		9(0)	3(2)
10	14(2)	10(0)	9(1)	3(3)
21	14(3)	10(1)	9(2)	21(0)
6	6(0)	10(2)	9(3)	21(1)

$$\alpha = 0.6 \\ S = 1.9\%$$

$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1} = \frac{1.14}{0.25} = 4,56$$

$$\frac{1}{4,56} = 0.21 \rightarrow 80\%$$

0643	U(110)	rw
	G(100)	r
	O(011)	wx

$$U_60 \quad U(011) \\ U(001) \rightarrow 0315 \\ U(100)$$


---

$$\alpha = 0.6 \\ S = 9.04$$

$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1} = \frac{1,294}{0,184} = 6,65$$

$$\frac{1}{6,65} = 0.15 \rightarrow 85\%$$


---

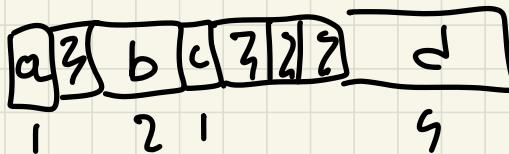
$$\alpha_1 = 0.3 \\ \alpha_2 = 0.4$$

$$\frac{1}{(0.6 + (0.4 \cdot 0.7))} = 1.14x$$


---

$$EAx = 0 \\ Ax = 16 \\ Cx = 0 \\ CX = 2 \\ Ax = 16 < Cx = 16 \times 2^2 = 16 \times 4 = 64 \\ Ax = 64 / 1 = 64$$


---



$$S = \frac{1}{\frac{\alpha}{K} + 1 - \alpha} \quad A 20\% \rightarrow K = 0.8 \rightarrow S = 1.1x \\ B 40\% \rightarrow K = 0.6 \rightarrow S = 1.20x \\ C \\ D$$

0423 → U (1 00) n  
 G (0 10) w  
 O (0 11) wx

---

$$S = 1.33x$$

$$\alpha = 0.5$$

$$S = \frac{1}{\frac{\alpha}{K} + 1 - \alpha}$$

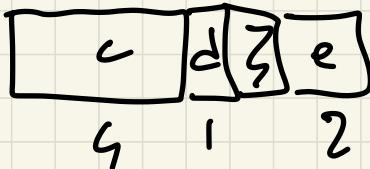
$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1}$$

- A 40% → K = 0.6 → 0.75x  
 B 60% → K = 0.4 → 0.57x  
 C 50% → K = 0.5 → 0.66x  
 D 70% → K = 0.3 → 0.46
- 

$$K = \frac{1}{1.98} = 0.5 \text{ 50\%}$$

0123    U (001) x  
 G (010) w  
 O (011) wx

---



$$\alpha = 0.8$$

$$S = 1.66$$

$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1} = \frac{1.328}{0.663} = 1.98$$

$$\frac{1}{1.99} = 0.50 \text{ 50\%}$$


---

0735    U (111) rnwx  
 G (011) wx  
 O (101) r x

$$\alpha = 0.6$$

$$S = 2.6 \times$$

$$K = \frac{S \cdot \alpha}{S(\alpha-1)+1} = \frac{1.25}{-0.25} = -5 \quad \text{NESSUNA}$$

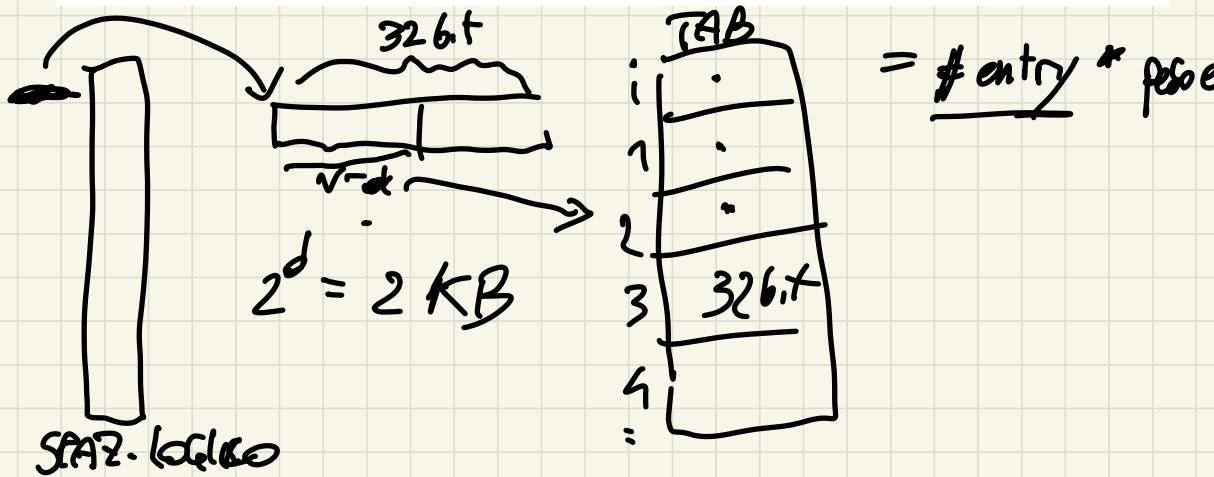
0125       $V(001)$       x  
               $G(010)$       w  
               $O(101)$       r x

### Domanda 2 (paginazione)

Si consideri un sistema di calcolo con spazio logico dei processi a 32 bit. Quanto occupa la tabella delle pagine se ogni pagina è di 2 KB? Si assuma che le entry della tabella delle pagine siano grandi ciascuna 32 bit.

$$2^{11}/2^2 = 2^9 \quad V-d=9 \quad d=32-9 \Rightarrow 2^{23}$$

A	32 MB	B	64 MB
C	16 MB	D	8 MB



Con  $V-d$  bit posso scrivere  $2^{V-d}$  numeri diversi

$$2^9 = 2 \text{ KB} : 1 \text{ KB} = 2^9 \rightarrow d = 11$$

$$V-d = 32-d = 32-11 = 21 \rightarrow 2^{21} : \# \text{entry}$$

$$\text{peso Tabella} = 2^{21} \cdot (32 \text{ bit}) = 2^{21} \cdot (2^2 \text{ byte}) = \frac{2^3}{2^3} \text{ GB}$$

$v[12]$  16 BYTE

Blocco 0  $v[0]$   $v[1]$   $v[2]$   $v[3]$

Blocco 1  $v[4]$   $v[5]$   $v[6]$   $v[7]$

Blocco 2  $v[8]$   $v[9]$   $v[10]$   $v[11]$

$C_M \rightarrow C_H \rightarrow C_H \rightarrow C_M \rightarrow C_H \rightarrow C_M$

1      2

NUOVI



U	rwX	7	
G	r x	5	0756
O	r	4	

---

SPAZIO LOGICO  $2^{16}$  v=16

$$d = 4 \text{ KB} = 4096 \text{ bit} = 2^{12}$$

I d BIT MENO SIGNIFICATIVI SONO L'OFFSET  
 I PRIMI v-d BIT PIÙ SIGNIFICATIVI SONO  
 L'INDICE DELLA TABELLA DELLE PAGINE

---

FDEM W

FDEMR W

```

* * * *
* * * *
* * * *
* * * *
F D E M W
F D E R W

```

---

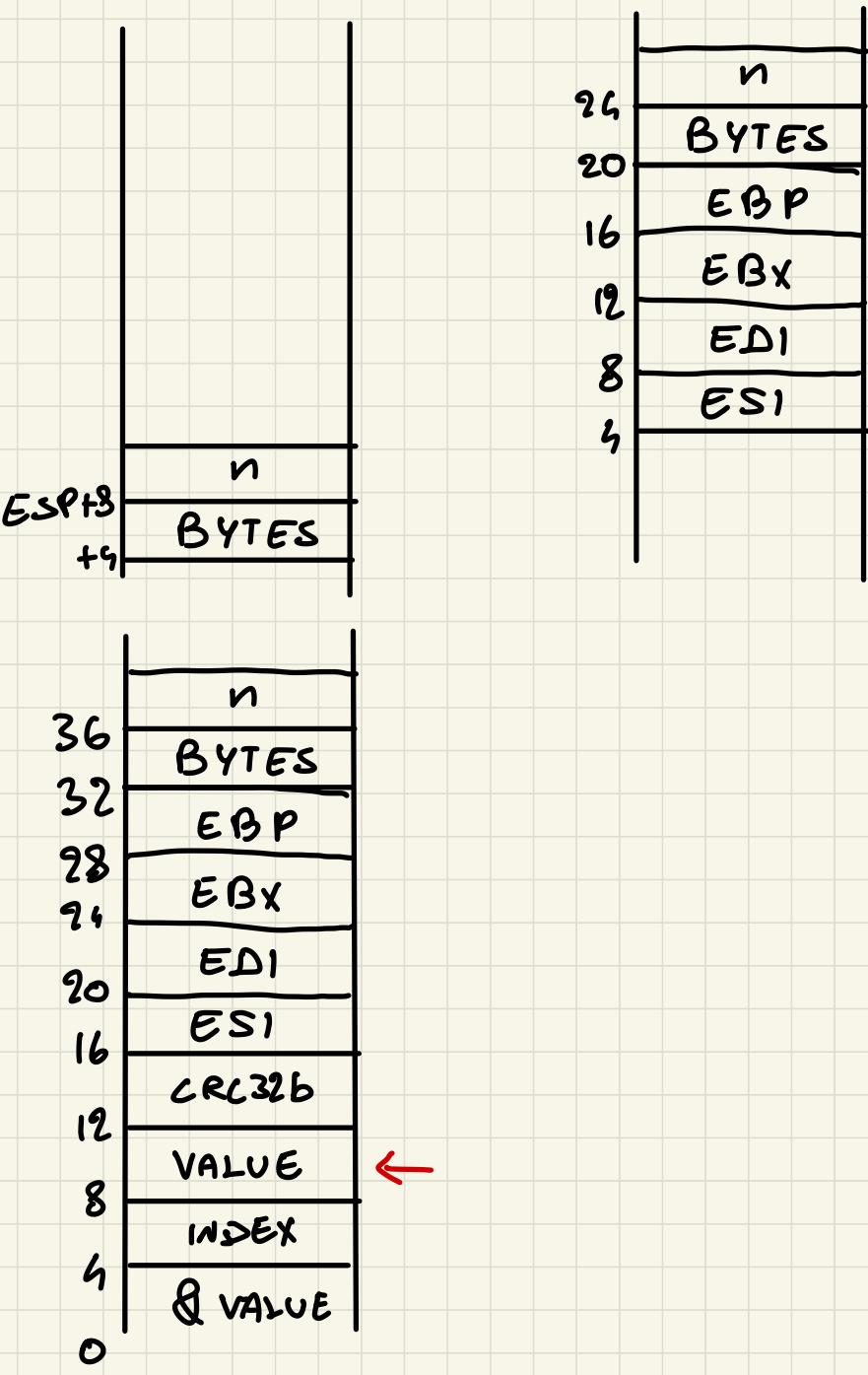
2 VIE

4 LINEE

128 BYTE

0 1 2 3

11		11(0)	
3		11(1)	3(0)
7		7(0)	3(1)
3		7(1)	3(0)
5		5(0)	3(1)
2	2(0)	5(1)	3(2)
7	2(1)	-	5(9) 7(0)



2 VIE  
64 BYTE

3 200 670 258 189 5439 1915 956

	0	1	2	3
50	50(0)			
10	50(1)	10(0)		
3	50(2)	10(1)	3(0)	
2	2(0)	10(2)	3(1)	
84	2(1)	84(0)	3(2)	
29	2(2)	84(1)	3(3)	29(0)
14	14	84	3	29

FDEπW

FDEM W

+ ++ +  
++ + + +

F D E π W

F D E M W

F D E π W

$$\alpha = 0.3$$

$$S = 1.21$$

$$K = \frac{S \cdot \alpha}{S(\alpha - 1) + 1} = \frac{0.363}{0.153} = 2,37$$

$$\frac{1}{2,37} = 0.42 \quad 60\%$$



$$\text{OFFSET}(w) = 12$$

1 MB =  $2^{20}$

V - d = 4 bit più significativi diversi

C

C

D

B