SAPIENZA
UNIVERSITÀ DI ROMA

Sistemi di calcolo

Capitolo 3: Parte II
Come viene tradotto in assembly un programma C?

Corso di Laurea in Ingegneria Informatica e Automatica

Istruzioni di selezione if e if ... else

Meccanismo test: 1) Operazione aritmetico-logica
2) salto se risultato soddisfa determinata proprietà

Proprietà specificata da un "condition code" cc:

cc	proprietà	
e	== 0	
ne	!= 0	
g	> 0	con segno
ge	>= 0	con segno
l	< 0	con segno
le	<= 0	con segno

cc	proprietà	
a	> 0	senza segno
ae	>= 0	senza segno
b	< 0	senza segno
be	<= 0	senza segno

Sub S, D
(risultato è D-S)

jcc L
(istruzione salto condizionato da cc)

Istruzioni di selezione if e if … else

Esempio:

```
    subl %eax, %ecx
    je L      → salta a L se ecx - eax == 0
    ...             cioé ecx == eax
L:  ...
```

```
    subl %eax, %ecx
    jle L     → salta a L se ecx - eax ≤ 0
    ...             cioé ecx ≤ eax
L:  ...
```

Esiste una variante di sub che non modifica la destinazione, **cmp**, usata per i test.

Istruzione di selezione if

Schema istruzione if

C	C equivalente	asm

`if (test) istr1`
`istr2`
	`if (!test) goto L;`
	`istr1`
	`L: istr2`
		`cmp ...`
		`jcc L`
		`istr1`
		`L: istr2`

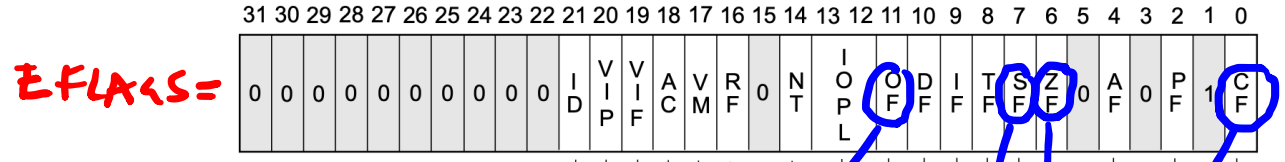
Es:

`if (a<0) a=-a;`
`return a;`
calcolo valore assoluto di a
	`if (a>=0) goto L;`
	`a=-a;`
	`L: return a;`
		`cmp $0,%eax`
		`jge L`
		`negl %eax`
		`L: ret`

Istruzione di selezione if

DEMO 3.3-if-else

Selezione: come funziona?



overflow

riporto

risultato zero

risultato negativo

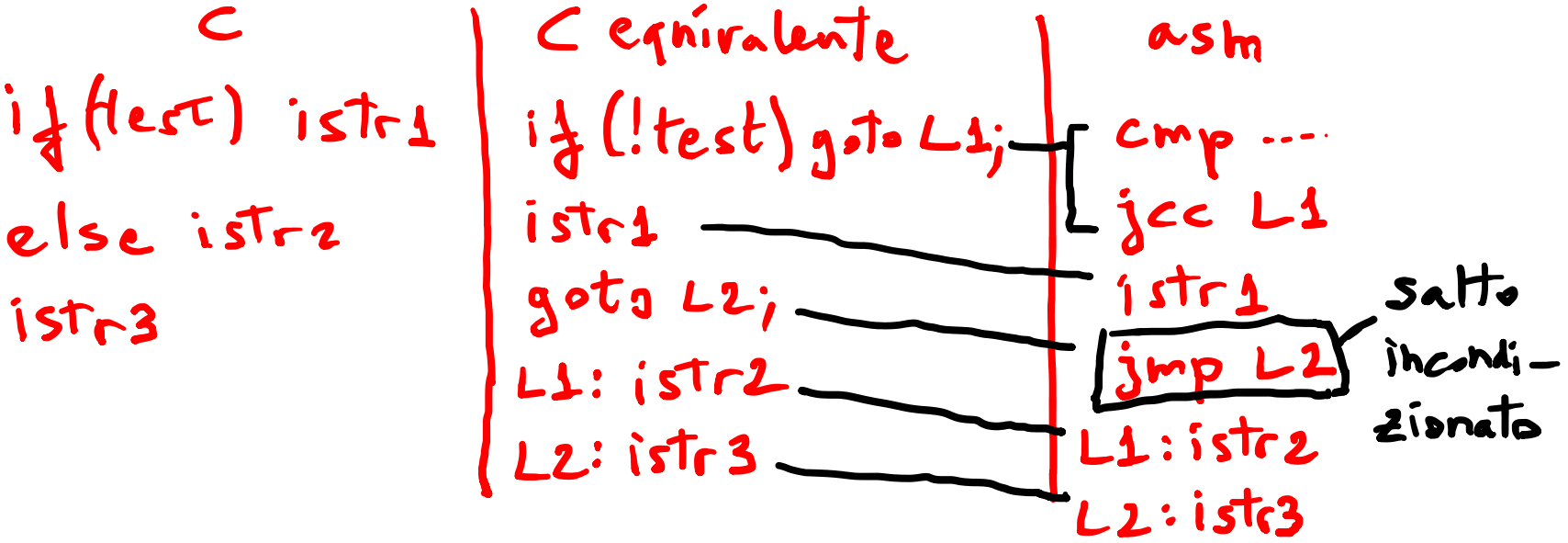
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check / Access Control (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

Es je salta ⇔ ZF=1

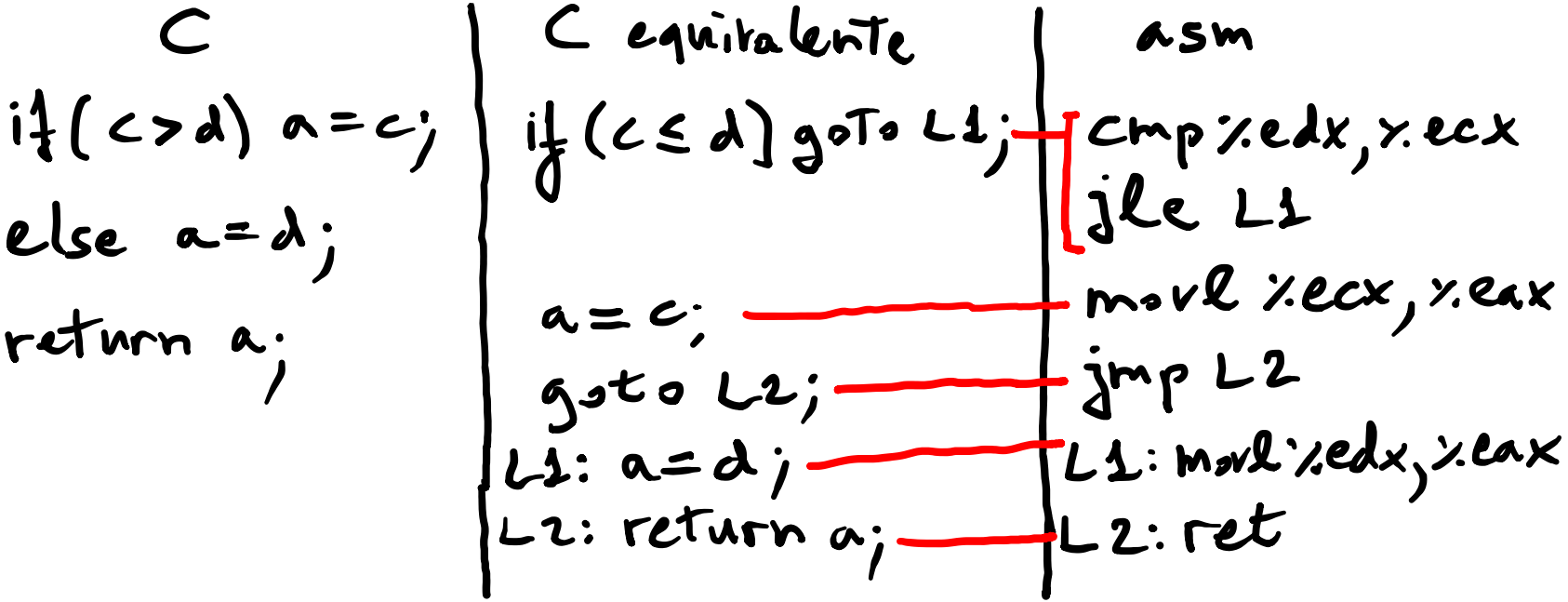
I processori x86 hanno un registro i cui bit vengono settati come effetto collaterale da istruzioni come quelle aritmetico-logiche. Le istr. di salto condizion. si basano su questi bit.

Istruzione di selezione if … else



Istruzione di selezione if … else

Calcola in a il massimo di c e d:



Istruzione di selezione if…else

DEMO 3.3-if-else

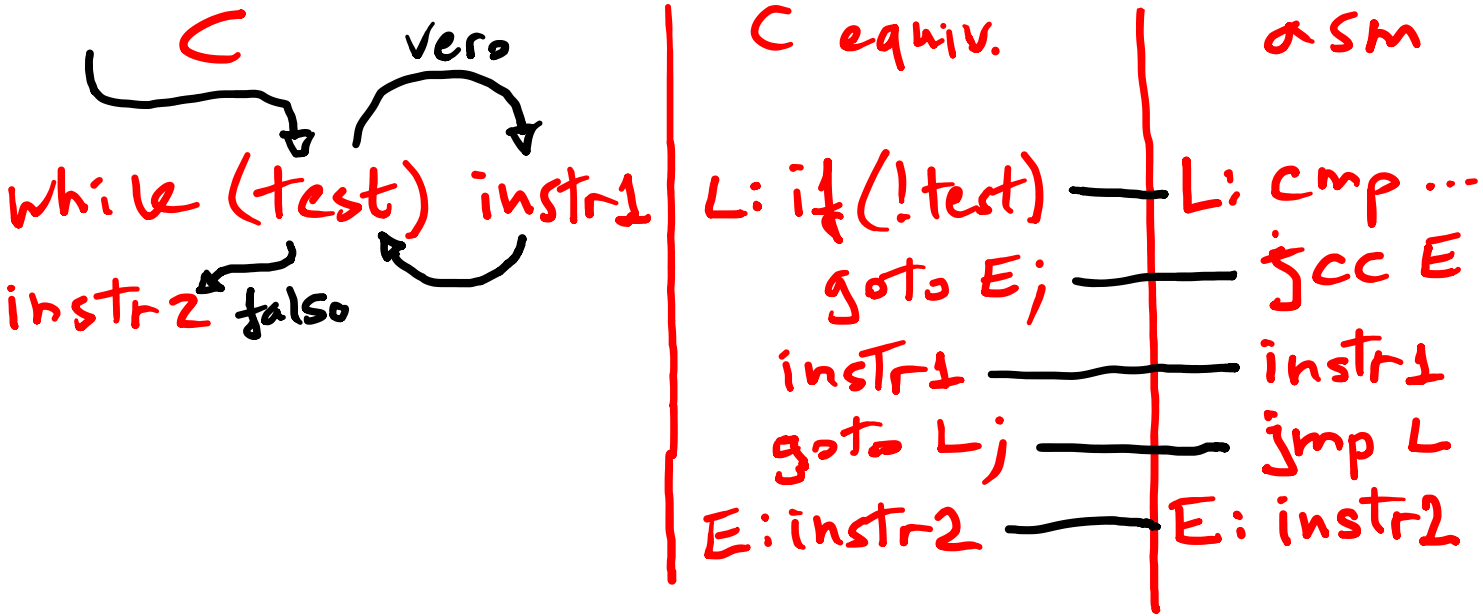
Confronto con zero

Come alternativa a `cmp $0, D` è possibile usare `test D, D` che è equivalente ad `and D, D`, tranne che non modifica la destinaz.

Es.

C	C equivalente	asm
`if (a<0) a=-a;`	`if (a>=0) goto L;`	`testl %eax, %eax`
		`jge L`
`return a;`	`a=-a;`	`negl %eax`
	`L: return a;`	`L: ret`

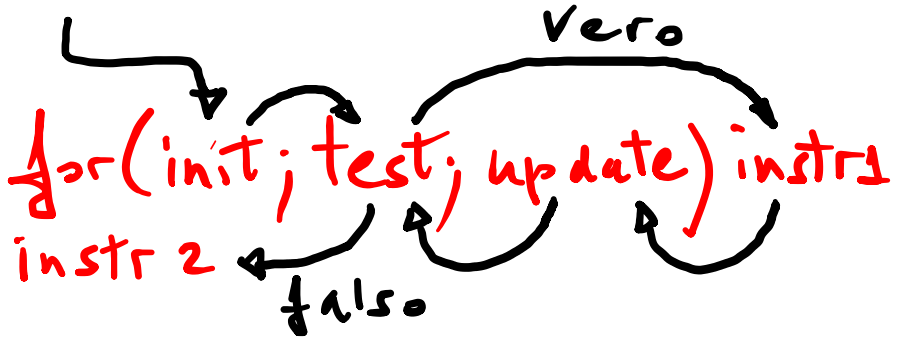
Istruzione di ciclo while

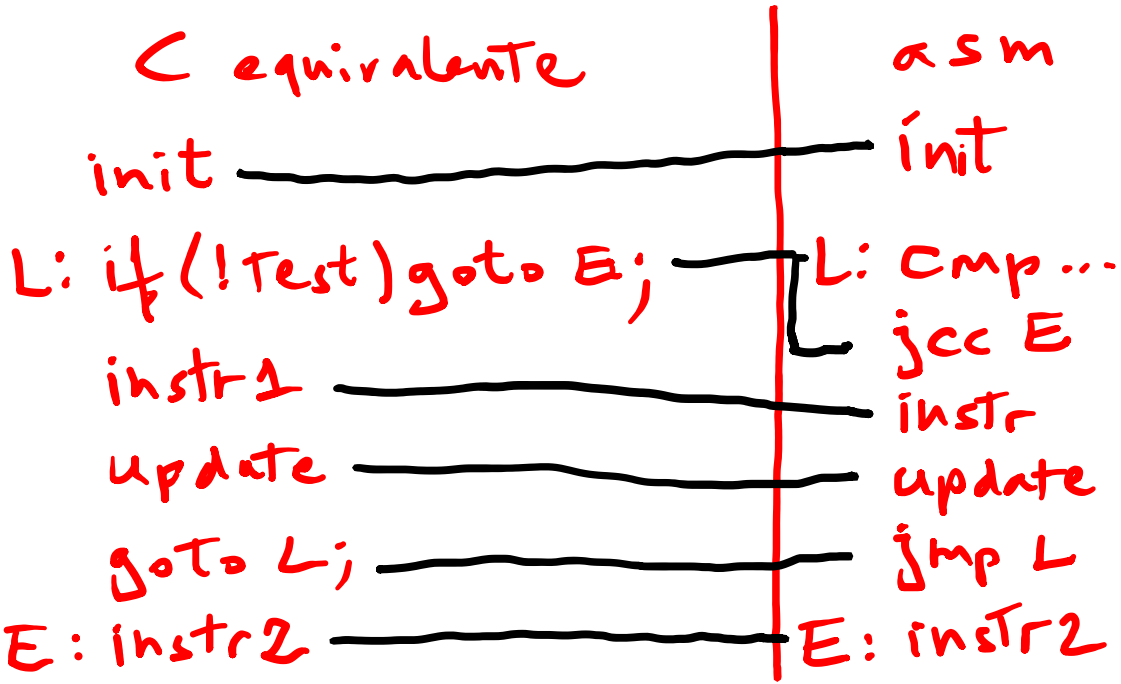


Istruzione di ciclo while

DEMO 3.4-loop

Istruzione di ciclo for





Istruzione di ciclo for

DEMO 3.4-loop/E3-fact

Puntatori

Se un registro reg contiene un indirizzo é possíbile accedere all'oggetto puntato in memoria tramite un **operando indiretto a memoria**:

(reg) → oggetto puntato da reg

Puntatori

C

```
void times2(short * p){
    *p= *p * 2;
}
```

C equivalente

```
void times2(short* p){
    short* a=p;
    short c=*a;
    c=c*2;
    *a=c;
    return;
}
```

asm

```
times2:
    movl 4(%esp), %eax
    movw (%eax), %cx
    imulw $2, %cx
    movw %cx, (%eax)
    ret
```

Puntatori

DEMO 3.5-ptr

Array

Se un registro reg contiene l'indirizzo di un array di oggetti di 1,2,4,8 byte ciascuno, é possibile accedere all'oggetto di indice reg2, dove reg2 é un registro che contiene l'indice, usando un operando a memoria con base, indice e scala:

(reg, reg2, scala), dove scala = 1,2,4,8

base indice

dimensione celle array

Array

Altri modi di indirizzamento a memoria:

(reg, reg2, 1) è equivalente a (reg, reg2)

Array

Es.
```
short get(short * v, int i){
    return V[i];
}
```

C equivalente

```
short get(short* v, int i){
    short* c=v;
    int d=i;
    short a= c[d];
    return a;
}
```

asm

```
get:
    movl 4(%esp),%ecx
    movl 8(%esp),%edx
    movw (%ecx,%edx,2),%ax
    ret
```

(%ecx,%edx,2): %ecx = base, %edx = indice, 2 = scala

Array

DEMO 3.6-array

Chiamate a funzione e passaggio dei parametri

La chiamata a funzione si realizza con l'istruzione call nome-funzione. I parametri vengono passati sulla stack.

Es.

```
int f(){
  return g(2,5);
}
```

```
f: subl $8, %esp      # fa spazio per i parametri attuali
   movl $2, (%esp)    } passaggio parametri
   movl $5, 4(%esp)   }
   call g             # chiamata a funzione
   addl $8, %esp      # dealloca spazio parametri
   ret                # valore di ritorno in eax
```

Chiamate a funzione e passaggio dei parametri

Es.

```
int f(int x) {
  return g(x)+1;
}
```

C equivalente:

```
int f(int x) {
  int a=x;
  a=g(a);
  a++;
  return a;
}
```

asm

```
f: subl $4, %esp
   movl 8(%esp), %eax
   movl %eax, (%esp)
   call g
   incl %eax
   addl $4, %esp
   ret
```

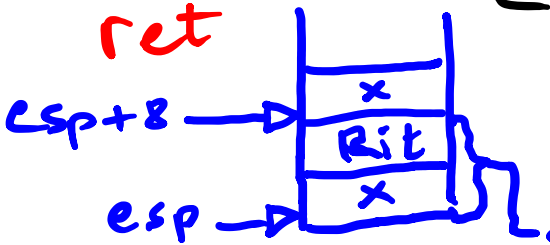
spazio per parametro di g

prende in eax param. x di f

chiama g

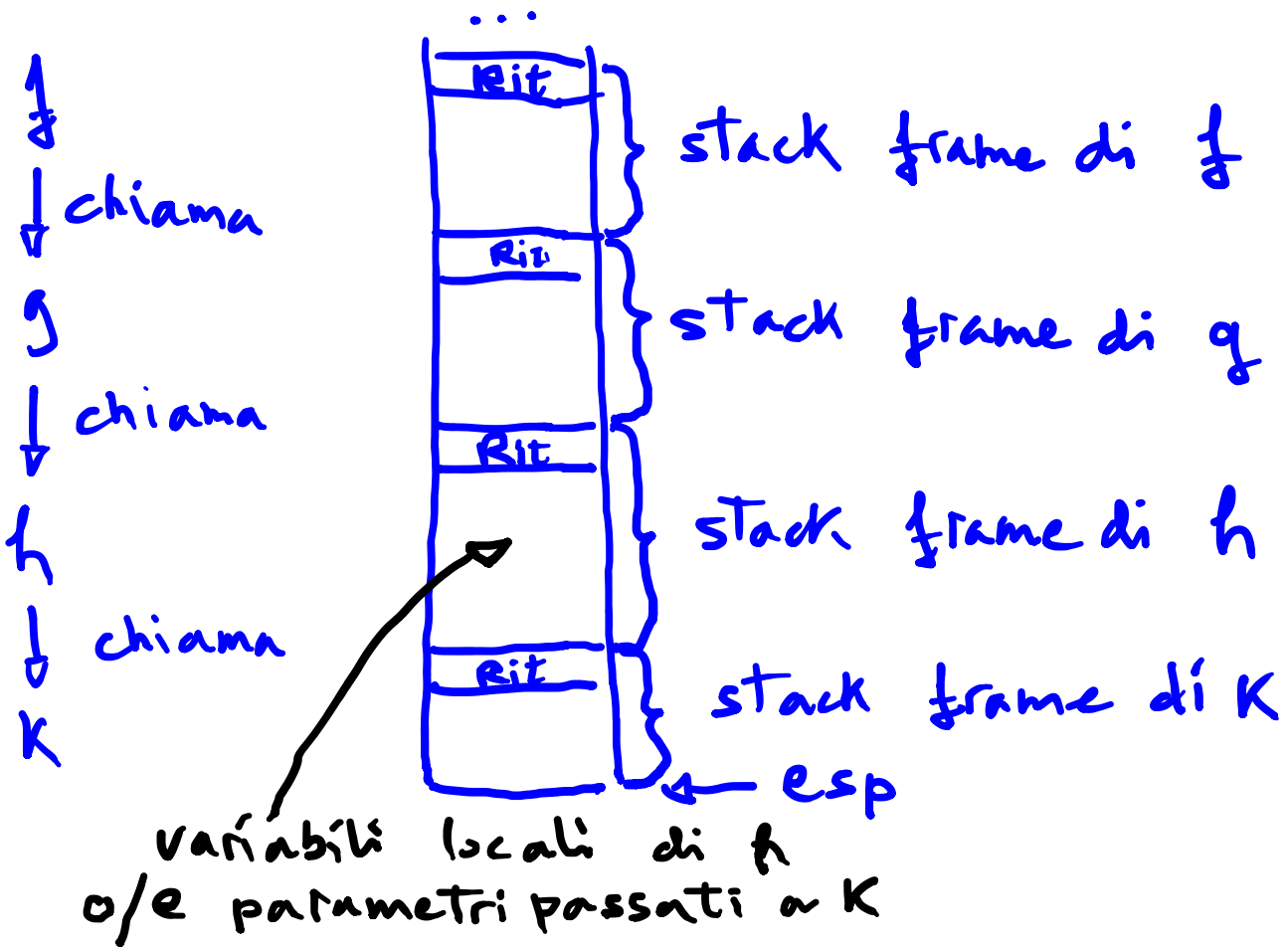
passa eax a g

dealloca spazio per param. di g dalla stack



stack frame di f

Stack frame delle chiamate



Chiamate a funzione e passaggio dei parametri

Es. C

```
int f() {
  return g()+h();
}
```

C equivalente

```
int f() {
  int a=g();
  int c=a;
  a=h();
  a=a+c;
  return a;
}
```

asm

```
f:
  call g
  movl %eax, %ecx
  call h
  addl %ecx, %eax
  ret
```

errato, ecx potrebbe essere sovrascritto dalla chiamata a h

come risolvere?

Convenzioni sull'uso dei registri nelle chiamate

Per risolvere problemi come quello visto, ci si appella a un'Application Binary Interface, un insieme di convenzioni che includono l'uso corretto dei registri a fronte di chiamate. La System V 386 ABI prescrive che:

1) A, C e D sono registri caller-save, cioé devono essere salvati dal chiamante se se ne vuole preservare il contenuto a fronte di una chiamata
2) B, DI, SI, BP sono callee-save, cioé salvati dal chiamato

Convenzioni sull'uso dei registri nelle chiamate

Soluzione 1): registri caller-save

```
f: call g
   pushl %eax
   call h
   popl %ecx
   addl %ecx, %eax
   ret
```

mette sulla cima della stack il valore di eax prodotto da g

riprende dalla cima della stack il valore prodotto da g e lo mette in ecx

Convenzioni sull'uso dei registri nelle chiamate

Soluzione 2): registri callee-save (preferibile)

```
f: pushl %ebx
   call g
   movl %eax,%ebx
   call h
   addl %ebx,%eax
   popl %ebx
   ret
```

prologo: salvataggio in stack dei registri callee-save che si vuole usare

ebx non viene sovrascritto da h per convenzione ABI

epilogo: ripristino dei valori che i registri callee-save avevano prima della chiamata

Convenzioni sull'uso dei registri nelle chiamate

Suggerimenti:

1) Usare registri caller-save nelle funzioni che non effettuano chiamate

2) Usare registri callee-save altrimenti

Chiamate a funzione e passaggio dei parametri

DEMO calls