[T05] Esercitazione 5

Istruzioni per l'esercitazione:

- Aprite il form di consegna in un browser e loggatevi con le vostre credenziali `uniroma1`.
- Scaricate e decomprimete sulla scrivania il codice dell'esercitazione. Vi sarà una sotto-directory separata per ciascun esercizio di programmazione. Non modificate in alcun modo i programmi di test `*_main.c`.
- Rinominare la directory chiamandola `cognome.nome`. Sulle postazioni del laboratorio sarà `/home/studente/Desktop/cognome.nome/`.
- È possibile consultare appunti/libri e il materiale didattico online.
- Rispondete alle domande online sul modulo di consegna.
- **Finiti gli esercizi**, e non più tardi della fine della lezione:
  - **zippate la directory di lavoro** in `cognome.nome.zip` (`zip -r cognome.nome.zip cognome.nome/`).
- **Per consegnare**:
  - inserite nel form di consegna come autovalutazione il punteggio di ciascuno dei test forniti (inserite zero se l'esercizio non è stato svolto, non compila, o dà errore di esecuzione).
  - fate **upload** del file `cognome.nome.zip`.
  - **importante**: verificate di aver ricevuto mail di conferma per la sottomissione del form
- Se siete in laboratorio, prima di uscire:
  - **importante**: fate logout dal vostro account Google!
  - eliminate dal desktop la directory creata (`rm -rf cognome.nome`).
  - rimettete a posto eventuali **sedie** prese all'ingresso dell'aula!

Per maggiori informazioni fate riferimento al regolamento delle esercitazioni.

Esercizio 1 (Numeri di Fibonacci)

Tradurre nel file `E1/e1.s` la seguente funzione C contenuta in `E1/e1.c` che calcola i numeri di Fibonacci:

```
int fib(int n) {
    if (n<2) return 1;
    return fib(n-1)+fib(n-2);
}
```

Usare il main di prova nella directory di lavoro `E1` compilando con `gcc -m32 e1_main.c e1.s -o e1`.

Esercizio 2 (Conto numero elementi uguali)

Tradurre nel file `E2/e2.s` la seguente funzione C contenuta in `E2/e2.c` che, dati due array di `short`, conta il numero di indici per cui gli array hanno lo stesso valore:

```
int counteq(short* v1, short* v2, int n) {
    int i, cnt = 0;
    for (i=0; i<n; ++i) cnt += (v1[i]==v2[i]);
    return cnt;
}
```

Suggerimento: usare le istruzioni `SETcc` e `MOVZ`/`MOVS`.

Usare il main di prova nella directory di lavoro `E2` compilando con `gcc -m32 e2_main.c e2.s -o e2`.

Esercizio 3 (Clonazione buffer di memoria)

Tradurre nel file `E3/e3.s` la seguente funzione C contenuta in `E3/e3.c` che clona un blocco di memoria di `n` byte all'indirizzo `src`. Il nuovo blocco deve essere allocato con `malloc` e deve avere lo stesso contenuto del blocco `src`. Sarà compito del chiamante di `clone` deallocare il blocco di memoria allocato da `clone`.

```c
#include <stdlib.h>
#include <string.h>

void* clone(const void* src, int n) {
    void* des = malloc(n);
    if (des==0) return 0;
    memcpy(des, src, n);
    return des;
}
```

Suggerimento: per copiare i dati `src` al nuovo blocco si suggerisce di usare la funzione `memcpy`. Si noti che è possibile chiamare funzioni di libreria C con call come se fossero normali funzioni scritte dall'utente.

Usare il main di prova nella directory di lavoro E3 compilando con `gcc -m32 e3_main.c e3.s -o e3`.

NOTA: in caso di errore in fase di linking su `memcpy` per via di PIE, usare `gcc -m32 e3_main.c e3.s -o e3 -no-pie`

Esercizio 4 (Palestra C)

Scrivere nel file `E4/e4.c` una funzione C `drop_spaces` che, data una stringa `text`, elimina tutti gli spazi.

```c
void drop_spaces(char* text);
```

Usare il main di prova nella directory di lavoro E4 compilando con `gcc e4_main.c e4.c -o e4`.

Esercizio 5 (Debugging)

Data la funzione C:

```c
#include "e5.h"

int f(int x, int y) {
    int b = x;
    int di = y;
    int a = g(b, di);
    int si = a;
    a = g(di, b);
    a = a + si;
    return a;
}
```

Uno studente ha tradotto la funzione in ASM nel file `E5/e5.s`. Purtroppo la traduzione presenta alcuni errori. Infatti, generando il binario con `gcc -m32 e5_main.c e5.s -o e5 g.s -g`, la funzione genera un errore (e non calcola il risultato corretto):

```
> ./e5
Segmentation fault (core dumped)
```

Usare Valgrind e GDB per analizzare step by step l'esecuzione ed identificare gli errori. Infine, correggere gli errori.

NOTA: ignorare il codice di `g.s`.

Domande

1. Se una funzione `foo` ha un prologo in cui vengono salvati due registri callee-save e vengono riservati 12 byte per ospitare variabili locali, argomenti ed eventuale padding, quale di questi operandi permette di accedere al secondo argomento di `foo`?
   - A. `(%esp)`
   - B. `4(%esp)`
   - C. `8(%esp)`
   - D. `12(%esp)`
   - E. `20(%esp)`
   - F. `24(%esp)`
   - G. `28(%esp)`
   - H. `32(%esp)`
2. Assumendo `%al = 5`, eseguire `movsbl %al, %eax` porta allo stesso risultato in `%eax` rispetto eseguire `movzbl %al, %eax`?
   - A. Sì
   - B. No
3. Assumendo di avere una funzione `foo` che chiama una funzione `baz`. Quale tra le seguenti affermazioni risulta essere vera:
   - A. `foo` non può utilizzare il registro `%eax`
   - B. `foo` non può utilizzare il registro `%ebx`
   - C. `baz` non può utilizzare il registro `%eax`
   - D. `baz` non può utilizzare il registro `%ebx`
   - E. Nessuna delle precedenti affermazioni è vera
4. Se una funzione `baz` viene chiamata da una funzione `foo`, quale delle seguenti affermazioni risulta essere **falsa**:
   - A. `baz` prima di poter utilizzare `%edi` deve salvare il suo contenuto e ripristinarlo prima di effettuare la `ret`
   - B. `baz` può utilizzare `%edx` senza dover preservare il suo contenuto iniziale
   - C. `foo` deve salvare il contenuto di `%ecx` se vuole preservarne il contenuto prima di chiamare `baz`
   - D. `foo` deve salvare il contenuto di `%esi` se vuole presevarne il valore prima di chiamare `baz`
5. Quale fra le seguenti istruzioni risulta essere valida:
   - A. `setl %eax`
   - B. `setba %al`
   - C. `leal (%eax, %edx, 6), %ecx`
   - D. `movl (%eax), 4(%esp)`
   - E. `leal -1(%ecx), %eax`
   - F. `addl %eax, $4`
6. Assumendo che `%eax=0x0000BEEF` e `%ecx=0`, quanto vale `%ecx` dopo aver eseguito l'istruzione `movsbw %al, %cx`?
   - A. `0x000000EF`
   - B. `0xFFFFFFEF`
   - C. `0x0000FFEF`
   - D. `0x0000EFEF`
7. Se `%ecx=0`, qual è il valore di `%al` dopo le istruzioni `testl %ecx, %ecx` e `setge %al`
   - A. 0

- B. 1
- C. nessuna delle precedenti

Soluzioni

Esercizio 1 (Numeri di Fibonacci)

Video esplicativo della soluzione

e1_eq.c

```
int fib(int n) {
    int di = n;
    int a = 1;
    if (di<2) goto E;
    di--;
    a = fib(di);
    int b = a;
    di--;
    a = fib(di);
    a = a + b;
E:  return a;
}
```

e1.s

```
 .globl fib

 fib:                       # int fib(int n)
     pushl %edi             # prologo
     pushl %ebx
     subl $4, %esp

     movl 16(%esp), %edi    # int di = n;
     movl $1, %eax          # int a = 1;
     cmpl $2, %edi          # if (di<2)
     jl E                   #     goto E;
     decl %edi              # di--;
     movl %edi, (%esp)      #         di
     call fib               # a = fib( | )
     movl %eax, %ebx        # int b = a;
     decl %edi              # di--;
     movl %edi, (%esp)      #         di
     call fib               # a = fib( | )
     addl %ebx, %eax        # a = a + b;

 E:  addl $4, %esp          # epilogo
     popl %ebx
     popl %edi
     ret                    # return a;
```

Esercizio 2 (Conto numero elementi uguali)

e2_eq.c

```
int counteq(short* v1, short* v2, int n) {

    short* di = v1;
    short* si = v2;
```

```
    int d = n;
    int a = 0;
    d--;

L:  if (d<0) goto E;
    short c = di[d];
    short b = si[d];
    char cl = c == b;
    int cc = (int)cl;
    a = a + cc;
    d--;
    goto L;
E:  return a;
}
```

e2.s

```
.globl counteq
counteq:  # int counteq(short* v1, short* v2, int n)

    pushl %edi                                  # prologo
    pushl %esi
    pushl %ebx

    movl 16(%esp), %edi                         # short* di = v1;
    movl 20(%esp), %esi                         # short* si = v2;
    movl 24(%esp), %edx                         # int d = n;
    xorl %eax, %eax                             # int a = 0;
    decl %edx                                   # d--;

L:  testl %edx, %edx                            # if (d<0)
    jl E                                        #    goto E;
    movw (%edi, %edx, 2), %cx                   # short c = di[d];
    movw (%esi, %edx, 2), %bx                   # short b = si[d];
    cmpw %cx, %bx                               # char cl =
    sete %cl                                    #    c == b;
    movsbl %cl, %ecx                            # int cc = (int)cl;
    addl %ecx, %eax                             # a = a + cl;
    decl %edx                                   # d--;
    jmp L                                       # goto L;
E:
    popl %ebx                                   # epilogo
    popl %esi
    popl %edi

    ret                                         # return a;
```

Esercizio 3 (Clonazione buffer di memoria)

e3_eq.c

```
#include <stdlib.h>
#include <string.h>

void* clone(const void* src, int n) {
    const void* si = src;
    int b = n;
    void* a = malloc(n);
    void* di = a;
    if (di==0) goto N;
    memcpy(di, si, b);
    a = di;
```

```
    return a;
N:  a = 0;
    return a;
}
```

e3.s

```
.global clone

clone:  # void* clone(const void* src, int n) {

    pushl %esi                          # prologo
    pushl %edi
    pushl %ebx
    subl $12, %esp

    movl 28(%esp),%esi                  # const void* si = src;
    movl 32(%esp),%ebx                  # int b = n;
    movl %ebx, (%esp)                   #                   b;
    call malloc                         # void* a = malloc(|)
    movl %eax, %edi                     # void* di = a
    xorl %eax, %eax                     # a = 0;
    testl %edi, %edi                    # if (di==0)
    je E                                #    goto E;
    movl %edi, (%esp)                   #         di
    movl %esi, 4(%esp)                  #         |   si
    movl %ebx, 8(%esp)                  #         |   |   b
    call memcpy                         # memcpy( | , |  ,| );
    movl %edi, %eax                     # a = di;
E:
    addl $12, %esp                      # epilogo
    popl %ebx
    popl %edi
    popl %esi
    ret                                 # return a;
```

Esercizio 4 (Palestra C)

e4.c

```
void drop_spaces(char* text) {
        char* s = text;
        while(*text)
                if (*text == ' ') text++;
                else *s++ = *text++;
        *s = '\0';
}
```

Esercizio 5 (Debugging)

e5.s

```
.globl f

f:  # int f(int x, int y) {
    pushl %ebx
    pushl %edi
    pushl %esi
    subl $8, %esp
```

```
    # ERRORE!!!!
    # offset di spiazzamento da ESP calcolari in modo errato
    # prima parametro si trova a: 4 + ESP + 12 (tre push) + 8 (due parametri)
    movl 24(%esp), %ebx         # int b = x;
    movl 28(%esp), %edi         # int di = y;
    movl %ebx, (%esp)
    movl %edi, 4(%esp)
    call g                      # int a = g(b, di);
    movl %eax, %esi             # int si = a;
    movl %edi, (%esp)
    movl %ebx, 4(%esp)
    call g                      # a = g(di, b);
    addl %esi, %eax             # a = a + si;
    # ERRORE!!!!
    # occorre prima deallocare lo spazio per i parametri e poi fare le pop
    addl $8, %esp
    # ERRORE!!!!
    # l'ordine delle pop deve essere inverso a quello delle push!
    popl %esi
    popl %edi
    popl %ebx
    ret             # return a;
```

Domande

Risposte corrette ai quiz:

1) G. `28(%esp)`

2) A. Sì

3) E. Nessuna delle precedenti affermazioni è vera

4) D. `foo` deve salvare il contenuto di `%esi se vuole presevarne il valore prima di chiamare baz`

5) E. `leal -1(%ecx), %eax`

6) C. `0x0000FFEF`

7) B. 1