[T04] Esercitazione 4

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 (debugging)

Data la funzione C:

```c
int count(const char *s1){
    int a=0;
    while(*s1){
        a++;
        s1++;
    }
    return a;
}
```

Uno studente ha tradotto la funzione in ASM nel file `E1/e1.s`. Purtroppo la traduzione presenta alcuni errori. Infatti, generando il binario con `gcc -m32 e1_main.c e1.s -o e1 -g`, la funzione non calcola il risultato corretto:

```
> ./e1
Test 1: 0 [corretto: 0]
Test 2: 0 [corretto: 3]
Test 3: 0 [corretto: 24]
Risultato: 1/3
```

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

Esercizio 2 (ricerca in un array)

Tradurre nel file `E2/e2.s` la seguente funzione C contenuta in `E2/e2.c` che cerca un intero in un array:

```
int find(int* v, int n, int x) {
    int i;
    for (i=0; i<n; ++i)
        if (v[i] == x) return 1;
    return 0;
}
```

Suggerimento: provare a riformulare il codice C in una forma equivalente in modo che usi solo tre variabili.

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

Esercizio 3 (strcmp alla vaccinara)

Tradurre nel file `E3/e3.s` la seguente funzione C contenuta in `E3/e3.c` che realizza la funzione di confronto di stringhe C `strcmp` come specificato dallo standard POSIX:

```
char my_strcmp(const char* s1, const char* s2) {
    while (*s1 && *s1 == *s2) {
        s1++;
        s2++;
    }
    return *s1 - *s2;
}
```

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

Esercizio 4 (Palestra C)

Scrivere nel file `E4/e4.c` una funzione dal seguente prototipo che, data una stringa `s` e una stringa `sub`, calcola il numero di posizioni distinte in `s` in cui `sub` appare come sottostringa:

```
int count_substrings(const char* s, const char* sub);
```

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

Domande

Rispondere ai quiz riportati nel form di consegna.

Soluzioni

Esercizio 1 (debugging)

IA32:

```
.globl count

count:                                  # int count(const char *s1){
```

```
    xorl %eax, %eax                     # int a=0;
    movl 4(%esp), %ecx                  # const char* c = s1;
A:  cmpb $0, (%ecx)    # errore 1       # if(!*c) goto E;
    je E               # errore 2
    incl %eax                           # a++;
    incl %ecx                           # c++;
    jmp A                               # goto A;

E:  ret
```

Esercizio 2 (ricerca in un array)

C equivalente:

```
int find(int* v, int n, int x) {
    int* c = v;
    int  d = n;
    int  a = x;
    d--;
L:  if (d < 0)     goto R0;
    if (c[d] == a) goto R1;
    d--;
    goto L;
R0: a = 0;
    return a;
R1: a = 1;
    return a;
}
```

IA32:

```
.global find

find: # int find(int* v, int n, int x) {
    movl 4(%esp), %ecx          # int* c = v;
    movl 8(%esp), %edx          # int  d = n;
    movl 12(%esp), %eax         # int  a = x;
    decl %edx                   # d--;
L:  testl %edx, %edx            # if (d < 0)
    jl R0                       # goto R0;
    cmpl %eax, (%ecx,%edx,4)    # if (c[d] == a)
    je R1                       # goto R1;
    decl %edx                   # d--;
    jmp L                       # goto L;
R0: xorl %eax, %eax             # a = 0;
    ret                         # return a;
R1: movl $1, %eax               # a = 1;
    ret                         # return a;
```

Esercizio 3 (strcmp alla vaccinara)

C equivalente:

```
char my_strcmp(const char* s1, const char* s2) {
    const char* c = s1;
    const char* d = s2;
L:; char a = *c;
    if (a == 0)  goto E;
    if (a != *d) goto E;
```

```
    c++;
    d++;
    goto L;
E:  a = a - *d;
    return a;
}
```

IA32:

```
my_strcmp: # char my_strcmp(const char* s1, const char* s2) {
    movl 4(%esp), %ecx          # const char* c = s1;
    movl 8(%esp), %edx          # const char* d = s2;
L:  movb (%ecx), %al            # ; char a = *c;
    testb %al, %al              # if (a == 0)
    je E                        # goto E;
    cmpb (%edx), %al            # if (a != *d)
    jne E                       # goto E;
    incl %ecx                   # c++;
    incl %edx                   # d++;
    jmp L                       # goto L;
E:  subb (%edx), %al            # a = a - *d;
    ret                         # return a;
```

Esercizio 4 (Palestra C)

```
static int is_prefix(const char* sub, const char* s) {
    while (*sub == *s && *sub!=0 && *s!=0) {
        sub++;
        s++;
    }
    return *sub == 0;
}

int count_substrings(const char* s, const char* sub) {
    int cnt = 0;
    do cnt += is_prefix(sub,s); while(*s++);
    return cnt;
}
```

Domande

Domanda 1) L'operando `(%eax, %ecx, 5)` è valido?

- No [la scala può essere solo {1, 2, 4}]

Domanda 2) Si consideri la variabile `int* p` e si assuma che venga tenuta nel registro `%eax`. A quale istruzione assembly corrisponde l'istruzione C p++?

- `addl $4,%eax` [p è un puntatore ad int: l'indirizzo in `%eax` va incrementato di 4 byte]

Domanda 3) Quali delle seguenti operazioni IA32 permette di azzerare il registro %eax?

- qualunque delle precedenti

Domanda 4) Quali dei seguenti predicati C permette di verificare se la variabile `int x` contiene un valore pari?

- `x & 1 == 0` [se il bit meno significativo è zero allora il numero è pari]

Domanda 5) Come tradurresti in IA32 l'assegnamento `v[5] = 7`, assumendo che `v` sia `int*` e sia tenuto nel registro `%eax`?

- `movl $7, 20(%eax)` [l'elemento `v[5]` si trova a 20 byte da v in quanto è preceduto da 4 elementi di taglia 4 byte]

Domanda 6) Si consideri la riga di comando `gcc main.c prova.s -o prova` su una piattaforma a 64 bit. Dove `prova.s` è un codice IA32. Che esito probabile ti aspetteresti?

- Segmentation Fault [codice IA32 userà `%esp` (32bit) invece che `%rsp` (64bit) andando ad operare con indirizzi non validi]