[T03] Esercitazione 3

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 (selezione)

Tradurre nel file `E1/e1.s` la seguente funzione C contenuta in `E1/e1.c` che realizza la funzione `min` con tre argomenti:

```
int min(int x, int y, int z) {
    if (x < y) {
        if (x < z) return x;
        else return z;
    } else {
        if (y < z) return y;
        else return z;
    }
}
```

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

Esercizio 2 (loop)

Tradurre nel file `E2/e2.s` la seguente funzione C contenuta in `E2/e2.c`:

```
unsigned int fib(unsigned int n) {
    unsigned int fib1 = 0, fib2 = 1;
    if (n == 0)
        return fib1;
    if (n == 1)
```

```
        return fib2;
    for (; n > 1; n--) {
        fib1 = fib1 + fib2;
        // scambiamo fib1 e fib2 senza usare variabile aggiuntiva
        fib1 = fib1 + fib2;
        fib2 = fib1 - fib2; // ricorda che in IA32 le istruzioni sono "D = D op S"
                            // mentre questa istruzione e' "D = S - D" pero' puo' esser
                            // facilmente spezzata in due passaggi!
        fib1 = fib1 - fib2;
    }
    return fib2;
}
```

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

Esercizio 3 (comparatore per interi)

Tradurre nel file `E3/e3.s` la seguente funzione C contenuta in `E3/e3.c` che realizza un comparatore per tipi `int`:

```
int comp(const void* xv, const void* yv) {
    int* x = (int*)xv; // nota: in IA32 gli indirizzi non hanno tipo
    int* y = (int*)yv; // e sono semplicemente dei numeri senza segno
    return *x - *y;
}
```

Nota: in IA32 i puntatori non hanno tipo e sono semplicemente dei contenitori che denotano indirizzi, cioè interi senza segno. Pertanto il cast `(int*)` è come se non ci fosse nella traduzione IA32.

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` la vostra versione personale della funzione della libreria standard libc strcpy che copia la stringa `src` nel buffer `dest` e restituisce `dest`. Il prototipo della funzione da realizzare è il seguente:

```
char *my_strcpy(char *dest, const char *src);
```

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:

```
int f(int x, int y){
    int z = x + y;
    if (z > 15) return 1;
    return 0;
}
```

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 -g -m32 e5_main.c e5.s -o e5 -m32`, la funzione non calcola il risultato corretto:

```
> ./e5
Test 1: 0 [corretto: 0]
```

```
Test 2: 0 [corretto: 1]
Test 3: 1 [corretto: 0]
Risultato: 1/3
```

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

Domande

Domanda 1) Si vuole tradurre l'istruzione C `if (a <= c) goto L`, dove `a` è una variable `unsigned int` memorizzata nel registro `eax` e `b` è una variable `unsigned int` memorizzata nel registro `ecx`, occorre usare:

- "`cmpl %eax, %ecx`" e "`jle L`"
- "`cmpl %ecx, %eax`" e "`jle L`"
- "`cmpl %eax, %ecx`" e "`jbe L`"
- "`cmpl %ecx, %eax`" e "`jbe L`"

Domanda 2) Si consideri il seguente frammento di codice:

```
1. xorl %eax, %eax
2. movw $1337, %ax
3. movl $2, %ecx
4. imulw %cx, %ax
5. andl $11, %eax
6. ???
7. ret
```

Quale delle seguenti istruzioni può essere inserita a linea 6 per far ritornare alla funzione il valore 20?

- `orl $10, %eax`
- `xorl $15, %eax`
- `notl %eax`
- Nessuna delle precedenti

Domanda 3) Si assuma di operare in una architettura IA32 sul seguente frammento di memoria (`indirizzo: valore`):

```
0x1000: 0xA1
0x1001: 0xB2
0x1002: 0xC3
0x1003: 0xD4
```

Eseguendo le seguenti istruzioni:

```
movw $0xF5E6, 0x1002
movl 0x1000, %eax
```

Cosa conterrà il registro `%eax`?

- `0xF5E6A1B2`
- `0xF5E6B2A1`
- `0xA1B2F5E6`
- `0xE6F5B2A1`

Domanda 4) Si consideri il seguente frammento di programma:

```
int x = 1;
printf("%d\n", *(char*)&x);
```

In quali casi il programma stampa 0?

- In nessun caso
- Solo se la piattaforma è big-endian
- Solo se la piattaforma è little-endian
- Solo se la piattaforma è a 32 bit

Soluzioni

Esercizio 1 (selezione)

C equivalente:

```
int min(int x, int y, int z) {
    int eax = x;
    int ecx = y;
    int edx = z;
    if (eax >= ecx) goto CD;
    if (eax >= edx) goto D;
    // eax = eax;
    goto E;
CD:
    if (ecx >= edx) goto D;
    eax = ecx;
    goto E;
D:
    eax = edx;
    goto E;
E:
    return eax;
}
```

IA32:

```
.globl min

min: # int min(int x, int y, int z) {
    movl 4(%esp), %eax  # int eax = x;
    movl 8(%esp), %ecx  # int ecx = y;
    movl 12(%esp), %edx # int edx = z;
    cmpl %ecx, %eax     # if (eax >= ecx) goto CD;
    jge CD
    cmpl %ecx, %eax     # if (eax >= edx) goto D;
    jge D
                        # // eax = eax;
    jmp E               # goto E;
CD:
    cmpl %edx, %ecx     # if (ecx >= edx) goto D;
    jge D
    movl %ecx, %eax     # eax = ecx;
    jmp E               # goto E;
D:
    movl %edx, %eax     # eax = edx;
    # jmp E             # goto E;
E:
    ret                 # return eax;
```

Esercizio 2 (loop)

C equivalente:

```c
unsigned int fib(unsigned int n) {
    unsigned int edx = 0, eax = 1;
    unsigned int ecx = n;
    if (ecx != 0) goto F1;
    eax = edx;
    goto E;
F1:
    if (ecx != 1) goto F2;
    // eax = eax;
    goto E;
F2:
L:
    if (ecx <= 1) goto E;
    ecx--;
    edx = edx + eax;
    edx = edx + eax;
    eax = -eax;
    eax = edx + eax;
    edx = edx - eax;
    goto L;
E:
    return eax;
}
```

IA32:

```
.globl fib
fib:    # unsigned int fib(unsigned int n) {
    xorl %edx, %edx         # unsigned int edx = 0
    movl $1, %eax           # eax = 1;
    movl 4(%esp), %ecx      # unsigned int ecx = n;
    testl %ecx, %ecx        # if (ecx != 0) goto F1;
    jne F1
    movl %edx, %eax         # eax = edx;
    jmp E                   # goto E;
F1:
    cmpl $1, %ecx           # if (ecx != 1) goto F2;
    jne F2
                            # // eax = eax;
    jmp E                   # goto E;
F2:
L:
    cmpl $1, %ecx           # if (ecx <= 1) goto E;
    jbe E
    decl %ecx               # ecx--;
    addl %eax, %edx         # edx = edx + eax;
    addl %eax, %edx         # edx = edx + eax;
    negl %eax               # eax = -eax;
    addl %edx, %eax         # eax = edx + eax;
    subl %eax, %edx         # edx = edx - eax;
    jmp L                   # goto L;
E:
    ret                 # return eax;
```

Esercizio 3 (comparatore per interi)

IA32:

```
.globl comp

comp:
        movl 4(%esp), %ecx
        movl 8(%esp), %edx
        movl (%ecx), %eax
        subl (%edx), %eax
        ret
```

Esercizio 4 (Palestra C)

```
char *my_strcpy(char *dest, const char *src) {
    char* aux = dest;
    while(*src) *dest++ = *src++;
    *dest = '\0';
    return aux;
}
```

Esercizio 5 (Debugging)

```
.globl f

f:                                  # int f(int x, int y){
    movl 4(%esp), %eax
    movl 8(%esp), %ecx
    movl %eax, %edx                 # int z = x;
    addl %ecx, %edx  # errore 1     # z += y;
    cmpl $15, %edx                  # if (z > 15) goto E;
    jg E             # errore 2
    movl $0, %eax                   # return 0;
    ret

E:  movl $1, %eax                   # return 1;
    ret
```

Domande

Domanda 1)

- “`cmpl %ecx, %eax`” e “`jbe L`” [“`cmpl %ecx, %eax`” perchè è “`D = D op S`” e “`jbe L`” perchè il confronto è fra variabili `unsigned`]

Domanda 2)

- Nessuna delle precedenti [`((1337 * 2) & 11) = 2`, considerando la A viene `2 | 10 = 10`, considerando la B viene `2 ^ 15 = 13`, considerando la C viene `~2 = 4294967293`]

Domanda 3)

- `0xF5E6B2A1` [`C3` è sostituito da `E6`, `D4` da `F5`, e poi i byte vengono letti in ordine inverso (da `0x1003` a `0x1000`) a causa dell’endianness]

Domanda 4)

- Solo se la piattaforma è big-endian [In memoria, vengono scritti in ordine i byte {`0x0`, `0x0`, `0x0`, `0x1`} e `(char*)&x` punta al prima byte]