[T07] Esercitazione 7

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 (Palestra IA32)

Tradurre nel file `E1/e1.s` la seguente funzione C contenuta in `E1/e1.c` che implementa il classico algoritmo ricerca binaria (o dicotomica) restituento 1 se `x` appartiene all'array `v` di `n` interi `int` e 0 altrimenti. Usare il file `E1/e1_eq.c` per sviluppare la versione C equivalente.

```c
#include "e1.h"

int binsearch(int *v, int n, int x) {
    int i=0, j=n;
    while (i<j) {
        int m = (i+j)/2;
        if (v[m]==x) return 1;
        if (v[m]>x) j=m;
        else i=m+1;
    }
    return 0;
}
```

Suggerimento: usare lo shift per dividere per 2.

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

Esercizio 2 (Palestra C)

Un tipico formato per rappresentare immagini digitali è la matrice `row-major`, dove le righe sono disposte consecutivamente in memoria. Per matrici a toni di grigio a 8 bit, ogni cella dell'array è compresa tra 0 (nero) e 255 (bianco). In totale, per un'immagine di altezza h e ampiezza w, l'array contiene `w*h` celle. La cella (0,0) è collocata nell'angolo superiore sinistro dell'immagine. Il pixel di coordinate `(i,j)`, dove `i` è la coordinata verticale e `j` quella orizzontale, risiede nella cella di indice `v[i*w+j]` dell'array.

Scrivere una funzione C `blur5` con il seguente prototipo:

```
void blur5(unsigned char* in, unsigned char* out, int w, int h);
```

che applica a un'immagine di input a 256 toni di grigio (`unsigned char`) un classico filtro che consente di sfocare l'immagine. I parametri sono:

- in: array della matrice di input da sfocare
- out: array della matrice di output sfocata
- w: ampiezza delle immagini di input e di output (indici j in [0,w])
- h: altezza delle immagini di input e di output (indici i in [0,h])

Il filtro da applicare è un semplice procedimento di convoluzione: ogni pixel di coordinate (i,j) dell'output sarà calcolato come la media aritmetica dei 25 valori in input nella finestra 5x5 centrata in (i,j). I pixel di output vicino ai bordi, la cui finestra 5x5 uscirebbe dai bordi dell'immagine di input, prendono semplicemente il valore del corrispondente pixel di input.

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

Domande

Rispondi alle seguenti domande, tenendo conto che una risposta corretta vale 1 punto, mentre una risposta errata vale 0 punti.

Domanda 1 Dato il seguente codice in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata:

```c
#define FACTOR 3

void f(int *v, int n) {
    int i, x = 10;
    for (i = 0; i < n; i++) {
        v[i] += x * FACTOR;
    }
}
```

```
f:
    movl    4(%esp), %ecx
    movl    8(%esp), %edx
    testl   %edx, %edx
    jle L1
    movl    %ecx, %eax
    leal    (%ecx,%edx,4), %edx
L3:
    addl    $30, (%eax)
    addl    $4, %eax
    cmpl    %edx, %eax
    jne L3
L1:
    ret
```

- A. Solo constant propagation
- B. Solo constant folding
- C. Constant folding e loop invariant code motion
- D. Constant folding e constant propagation
- E. Nessuna delle precedenti

Domanda 2 Data la seguente funzione f in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata:

```c
int f(int x) {
    int s = 0;
    while (x > 0) s += x--;
    return s;
}
```

```
.globl  f
f:
    subl    $16, %esp
    movl    $0, 12(%esp)
    jmp L2
L3:
    movl    20(%esp), %eax
    leal    -1(%eax), %edx
    movl    %edx, 20(%esp)
    addl    %eax, 12(%esp)
L2:
    cmpl    $0, 20(%esp)
    jg  L3
    movl    12(%esp), %eax
    addl    $16, %esp
    ret
```

- A. Loop unrolling
- B. Register allocation
- C. Common subexpression elimination
- D. Loop invariant code motion
- E. Nessuna delle precedenti

Domanda 3 Dato il seguente codice quale delle seguenti affermazioni risulta vera?

```c
int c = 0;

int g(int x) {
    c++;
    return x*x;
}

int f(int *v, int n, int x) {
    int i, a = 0;
    for (i = 0; i < n; i++)
        a += i + g(x);
    return a;
}
```

- A. gcc non è in grado di applicare la tecnica del loop invariant code motion in questo codice
- B. gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se g è definita nello stesso file sorgente di f
- C. gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se si attiva almeno il livello di ottimizzazione 3 (gcc -O3)
- D. gcc è in grado di applicare la tecnica del loop invariant code motion, ma solo se g viene dichiarata “static”

Domanda 4 Dato il seguente codice in linguaggio C e la sua traduzione in linguaggio assembly, dire quale tra le seguenti tecniche di ottimizzazione è stata applicata:

```c
int hash(int x, int y) {
    return 31*(31 + x) + y;
}
```

```
hash:
    movl    4(%esp), %eax
    leal    31(%eax), %edx
    movl    %edx, %eax
    sall    $5, %eax
    subl    %edx, %eax
    addl    8(%esp), %eax
    ret
```

- A. Common subexpression elimination
- B. Constant propagation
- C. Constant folding
- D. Dead code elimination
- E. Strength reduction
- F. Nessuna delle precedenti

Domanda 5 Dato un programma P, supponendo di ottimizzare una porzione di codice che impegna il 40% del tempo totale di esecuzione di P, ottenendo su tale porzione di codice uno speedup di 2x, qual è lo speedup complessivo su P?

- A. circa 1.67x
- B. 1.25x
- C. 0.8x
- D. 2x
- E. Nessuna delle precedenti

Domanda 6 Dato un programma P, supponendo di ottimizzare una porzione di codice che impegna il 75% del tempo totale di P, qual è lo speedup *massimo teorico* che possiamo ottenere su P?

- A. Non è possibile determinare alcun limite allo speedup ottenibile su P, senza conoscere lo speedup ottenuto sulla porzione di codice ottimizzata
- B. 4/3 x
- C. 4x
- D. 0.75x
- E. Nessuna delle precedenti

Domanda 7 Data la seguente porzione del report di gprof per un dato programma, quale funzione rappresenta il maggior collo di bottiglia prestazionale (bottleneck)?

```
             Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.20% of 4.89 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    4.89                 main [1]
                0.00    4.56    1000/1000        A [2]
                0.02    0.31    1000/1000        B [5]
                0.00    0.00       1/1           init [6]
-----------------------------------------------
                0.00    4.56    1000/1000        main [1]
[2]     93.3    0.00    4.56    1000         A [2]
                4.26    0.31    1000/1000        C [3]
-----------------------------------------------
                4.26    0.31    1000/1000        A [2]
[3]     93.3    4.26    0.31    1000         C [3]
                0.31    0.00   10000/20000       D [4]
-----------------------------------------------
                0.31    0.00   10000/20000       B [5]
                0.31    0.00   10000/20000       C [3]
[4]     12.6    0.61    0.00   20000         D [4]
-----------------------------------------------
                0.02    0.31    1000/1000        main [1]
[5]      6.7    0.02    0.31    1000         B [5]
                0.31    0.00   10000/20000       D [4]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         init [6]
-----------------------------------------------
```

- A. A
- B. B
- C. C
- D. D
- E. main
- F. init

Soluzioni

Esercizio 1 (Palestra IA32)

e1_eq.c

```
int binsearch(int *v, int n, int x) {
    int i=0;
    int j=n;
    int a=1;
L:  if (i>=j) goto E0;
    int m = (i+j) >> 1;
    if (v[m]==x) goto E1;
    if (v[m]<=x) goto F;
    j=m;
    goto L;
F:  i=m+1;
    goto L;
E0: a=0;
```

```
E1: return a;
}
```

e1.s

```
.globl binsearch

binsearch:  # int binsearch(int *v, int n, int x) {

    # Register allocation: i:esi, j:edi, a=eax, m=ecx, v:edx, n:ebx, x:ebp

    pushl %esi
    pushl %edi
    pushl %ebx
    pushl %ebp

    movl 20(%esp), %edx
    movl 24(%esp), %ebx
    movl 28(%esp), %ebp

    xorl %esi, %esi             # int i=0;
    movl %ebx, %edi             # int j=n;
    movl $1, %eax               # int a=1;
L:  cmpl %edi, %esi             # if (i>=j)
    jge E0                      # goto E0;
    leal (%esi,%edi), %ecx      # m = (i+j)
    sarl %ecx                   #     >> 1;
    cmpl %ebp, (%edx, %ecx, 4)  # if (v[m]==x)
    je E1                       #   goto E1;
    cmpl %ebp, (%edx, %ecx, 4)  # if (v[m]<=x)
    jle F                       # goto F;
    movl %ecx, %edi             # j=m;
    jmp L                       # goto L;
F:  leal 1(%ecx), %esi          # i=m+1;
    jmp L                       # goto L;
E0: xorl %eax, %eax             # a=0;
E1: popl %ebp
    popl %ebx
    popl %edi
    popl %esi
    ret                         # return a;
```

Esercizio 2 (Palestra C)

e2.c

```
#define IDX(i,j,w) ((i)*(w)+(j))

void blur5(unsigned char* in, unsigned char* out, int w, int h){
    int i,j,u,v;
    for (i=0; i<h; ++i)
        for (j=0; j<w; ++j)
            if (i<2 || i>h-3 || j<2 || j>w-3)
                out[IDX(i,j,w)] = in[IDX(i,j,w)];
            else {
                unsigned somma = 0;
                for (u=-2; u<=2; ++u)
                    for (v=-2; v<=2; ++v)
                        somma += in[IDX(i+u,j+v,w)];
                out[IDX(i,j,w)] = somma/25;
```

```
            }
}
```

Domande

1. D - Constant folding e constant propagation
2. E - Nessuna delle precedenti
3. A - gcc non è in grado di applicare la tecnica del loop invariant code motion in questo codice
4. F - Strength reduction
5. B - 1.25x
6. C - 4x
7. C - C