Tutorato Architettura degli Elaboratori 08

Alberto Paparella[1]

27 Maggio 2025

[1]Dipartimento di Matematica e Informatica, Università degli studi di Ferrara

Esercizi di laboratorio

Istruzioni

Ogni esercizio è descritto tramite un programma in linguaggio C.

Se la realizzazione di un I/O non è richiesta, le variabili possono essere inizializzate con istruzioni assembler o con direttive (nel caso degli array). Si raccomanda di seguire le convenzioni del linguaggio assembler specie per ciò che riguarda le funzioni.

Si raccomanda di utilizzare i commenti per indicare la corrispondenza fra variabili del C e registri.

I commenti sono anche utili per descrivere cosa volete fare nel codice.

Esercizio 1

Programma che calcola il prodotto di due numeri naturali in una CPU a 32 bit priva di moltiplicatore. L'esercizio deve essere risolto senza utilizzare macro (bge ...). Si ignorino eventuali problemi di overflow.

```c
int main() {
    unsigned int x, y; /* moltiplicando e moltiplicatore */
    unsigned int p; /* prodotto */
    int i, tmp;     /* variabili */
    x = 16; y = 18; p = 0;
    i = 0;
    while (i < 32) {
    tmp = y & 1;    /* i-th bit of y */
        if (tmp != 0)
            p = p + x;
        y = y >> 1;
        x = x << 1; /* x=x*2 */
        i = i + 1;
    }
}
```

Listing 1: Codice C per il calcolo del prodotto fra due interi.

Esercizio 2

Verifica dell'ordinamento di un vettore. Anche in questo caso non si devono utilizzare macro. Il codice non é ottimizzato, ma questo non riguarda l'esercizio.

```c
int main() {
    int array[8]={0,1,4,2,7,8,4,6};
    int i;
    /* ordinamento crescente e str. crescente */
    int ord_c, ord_sc;
    i = 0;
    ord_c = ord_sc = 1; /* true */
    while (i < 7) {
        if (array[i] >= array[i+1])
            ord_sc = 0;
        if (array[i] > array[i+1])
            ord_c = 0;
        i = i + 1;
    }
}
```

Listing 2: Codice C per la verifica dell'ordinamento di un vettore.

Esercizio 3

Funzione che calcola un esempio di espressione. Si faccia l'ipotesi di dover preservare tutti i registri utilizzati dall'espressione (compreso i $t).

```c
int main() {
    int a,b,c,d;
    int v;
    a = 7; b = 4; c =4 ; d = 2;
    ....
    v = dist(a,b,c,d);
}

int dist(int a, int b, int c, int d) {
    int result;
    result = (a + b) >> (c - d) + b << d;
    /* << e >> sono prioritari rispetto alla somma */
    return result;
}
```

Listing 3: Codice C per il calcolo di un'espressione.

Esercizio 4

Programma inteso a verificare le differenze fra & ed && in C.

```c
int main() {
    int x, y, w;
    x = 9; y = 6;
    w = 0;
    if (x & y) /* bitwise and */
        w = 1;
    else
        if (x && y) /* logical and */
            w = 2;
}
```

Listing 4: Codice C per per verificare le differenza fra & ed &&.

Soluzioni

Esercizio 1

```
1  .text
2      addi $s0, $zero, 1024    # x
3      addi $s1, $zero, 16      # y
4      addi $s2, $zero, 0       # p
5
6      addi $t0, $zero, 0
7      addi $t1, $zero, 32
8
9  loop:
10     beq $t0, $t1, endloop
11     andi $t2, $s1, 1
12     beq $t2, $zero, label
13     add $s2, $s2, $s0
14 label:
15     srl $s1, $s1, 1
16     sll $s0, $s0, 1
17     addi $t0, $t0, 1
18     j loop
19 endloop:
```

Listing 5: Codice assembly MIPS per il calcolo del prodotto fra due interi.

Esercizio 2 (1)

```
1  .data
2      array: .word 0, 1, 2, 4, 6, 7, 9, 0
3
4  .text
5      addi $s0, $zero, 1 # ord_c
6      addi $s1, $zero, 1 # ord_sc
7      addi $s2, $zero, 0 # i
8      addi $t0, $zero, 7
9      addi $t1, $zero, 0
10
11     # loop non ottimizzato
12     lw $t2, array($t1)
```

Listing 6: Codice assembly MIPS per la verifica dell'ordinamento di un vettore (1).

Esercizio 2 (2)

```
1  loop:
2      beq $s2, $t0, endloop
3      addi $s2, $s2, 1
4      sll $t1, $s2, 2      # addr=4*i
5      lw $t3, array($t1)
6      slt $t4, $t2, $t3    # array[i+1]<array[i]
7      bne $t4, $zero, label0
8      addi $s1, $zero, 0
9  label0:
10     slt $t4, $t3, $t2    # array[i]>array[i+1]
11     beq $t4, $zero, label1
12     addi $s0, $zero, 0
13 label1:
14     addi $t2, $t3, 0     # evita una lw per ciclo
15     j loop
16 endloop:
```

Listing 7: Codice assembly MIPS per la verifica dell'ordinamento di un vettore (2).

Esercizio 3 (1)

```
1  .text
2  main:
3      addi $s0, $zero, 7 # a
4      addi $s1, $zero, 4 # b
5      addi $s2, $zero, 4 # c
6      addi $s3, $zero, 2 # d
7
8      # possible operations on $s0..3 and other registers
9      addi $a0, $s0, 0 # argument 1
10     addi $a1, $s1, 0 # argument 2
11     addi $a2, $s2, 0 # argument 3
12     addi $a3, $s3, 0 # argument 4
13     jal dist # call Function
14     addi $s4, $v0, 0 # returned value
```

Listing 8: Codice assembly MIPS per il calcolo di un'espressione (1).

Esercizio 3 (2)

```
1  dist:
2      addi $sp, $sp, -12 # make space on stack to
3      # store three registers
4      sw $s0, 0($sp) # save $s0 on stack
5      sw $s1, 4($sp) # save $s1 on stack
6      sw $s2, 8($sp) # save $s2 on stack
7
8      add $s0, $a1, $a0 # a+b
9      sub $s1, $a2, $a3 # c-d
10     sllv $s2, $a1, $a3 # b<<d
11     srlv $s0, $s0, $s1 # >>
12     add $v0, $s0, $s2
13
14     lw $s0, 0($sp) # restore $s0 from stack
15     lw $s1, 4($sp) # restore $t0 from stack
16     lw $s2, 8($sp) # restore $t0 from stack
17     addi $sp, $sp, 12 # deallocate stack space
18
19     jr $ra # return to caller
```

Listing 9: Codice assembly MIPS per il calcolo di un'espressione (2).

Esercizio 4

```
.text
    addi $s0, $zero, 9
    addi $s1, $zero, 7
    addi $s2, $zero, 0
    # nota: in base 2 ho 1001 e 0110 sono entrambi != 0
    # e quindi "veri" per && ma il loro and bit a bit da 0
    # ovvero falso
    and $t0, $s0, $s1
    beq $t0, $zero, label
    addi $s2, $zero, 1
    j join
label:
    beq $s0, $zero, join
    beq $s1, $zero, join
    addi $s2, $zero, 2
join:
```

Listing 10: Codice assembly MIPS per verificare le differenza fra & ed &&.