Architettura degli Elaboratori
Corso A
Lab 3

Operandi Allocati in Memoria (load/store)

Esercizio 0 - Media interi

Si scriva un programma in linguaggio RISC-V che carichi 4 numeri interi presente nella memoria in word contigue e calcoli il valore intero della loro media aritmetica (arrotondamento per difetto). Il valore calcolato va salvato in un’ulteriore posizione della memoria contigua a quelle usate per il calcolo.

- **In questo esercizio, utilizzare soltanto il set delle istruzioni "intere di base rv64i".**

Esercizio - Media interi

```
.globl _start
.data
    v1: .word 1
    v2: .word 2
    v3: .word 3
    v4: .word 4
    v5: .word 0
.text
_start:
    la   t1, v1          # get the v1 address
    lw   t2, 0(t1)       # copy the first word to t2
    lw   t3, 4(t1)       # copy the second word to t3
    add  t4, t2, t3      # sum the words and save it on register t4
    lw   t3, 8(t1)       # copy the third word to t3
    add  t4, t4, t3      # sum it and save on register t4
    lw   t3, 12(t1)      # copy the fourth word to t3
    add  t4, t4, t3      # sum it and save on register t4
    srli t4, t4, 2       # shift to the right by 2 positions
    sw   t4, 16(t1)      # store the sum in the 5th word
```

Istruzioni per Prendere Decisioni

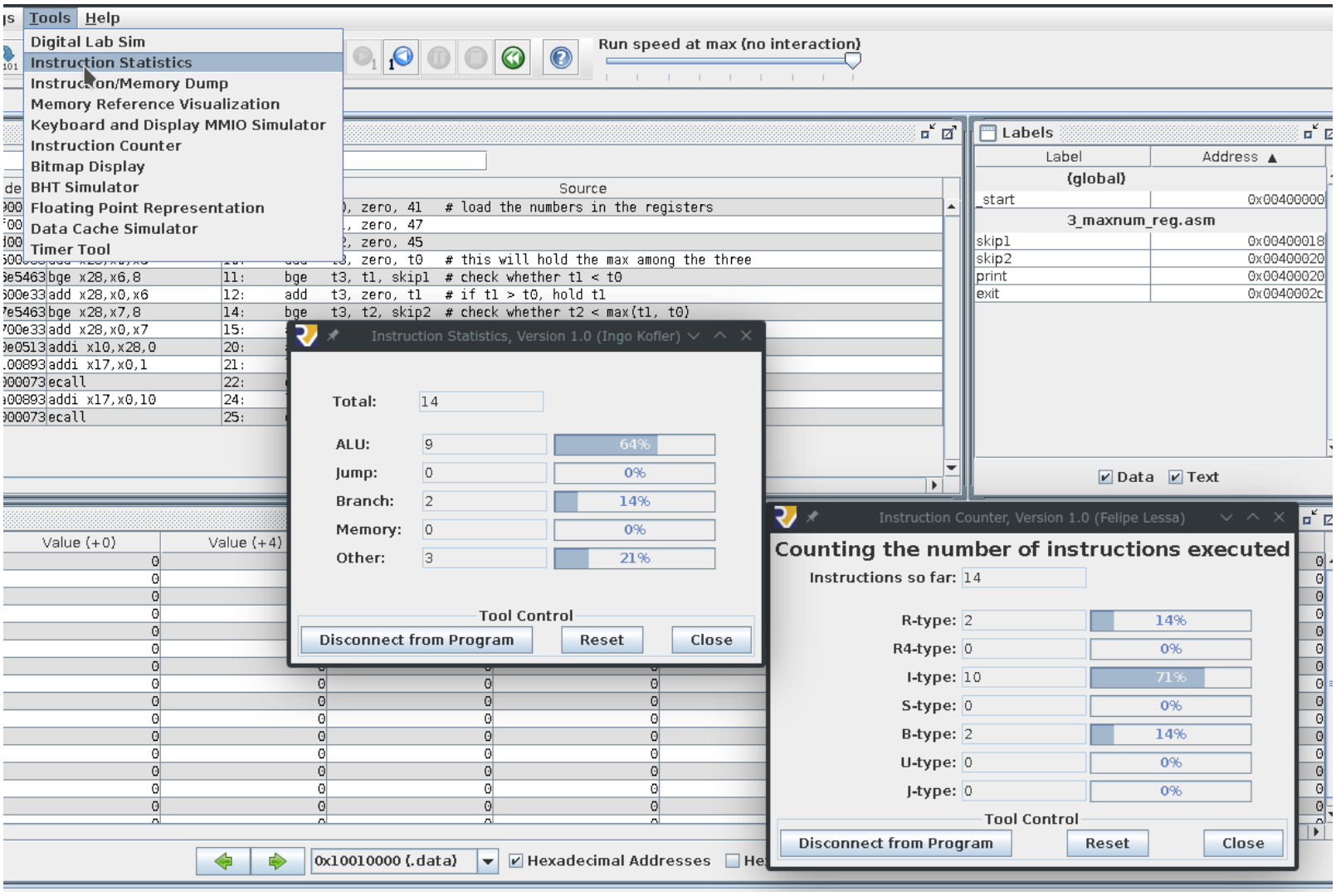
Obiettivi

- Imparare come un codice C/Java viene tradotto in RISC-V
- Tradurre le istruzioni per prendere decisioni
  - if () then else
  - while ()
  - for()
  - do...while()
- Utilizzare il simulatore per misurare il numero di istruzioni RISC-V eseguite per completare ogni esercizio

RISC-V - I registri

Useremo solo i nomi da adesso in poi

Registro	Nome	Utilizzo
x0	zero	La costante 0
x1	ra	Indirizzo di ritorno
x2	sp	Puntatore a stack
x3	gp	Puntatore globale
x4	tp	Puntatore a thread
x5-x7	t0-t2	Temporanei
x8	s0_/_fp	Salvato/puntatore a frame
x9	s1	Salvato
x10-x11	a0-a1	Argomenti di funzione/valori restituiti
x12-x17	a2-a7	Argomenti di funzione
x18-x27	s2-s11	Registri salvati
x28-x31	t3-t6	Temporanei


Tools
Help
Digital Lab Sim
Instruction Statistics
Instruction/Memory Dump
Memory Reference Visualization
Keyboard and Display MMIO Simulator
Instruction Counter
Bitmap Display
BHT Simulator
Floating Point Representation
Data Cache Simulator
Timer Tool
Run speed at max (no interaction)
Source
load the numbers in the registers
this will hold the max among the three
check whether t1 < t0
if t1 > t0, hold t1
check whether t2 < max(t1, t0)
Labels
Label
Address
(global)
_start
0x00400000
3_maxnum_reg.asm
skip1
0x00400018
skip2
0x00400020
print
0x00400020
exit
0x0040002c
Data
Text
Instruction Statistics, Version 1.0 (Ingo Kofler)
Total:
14
ALU:
9
64%
Jump:
0
0%
Branch:
2
14%
Memory:
0
0%
Other:
3
21%
Tool Control
Disconnect from Program
Reset
Close
Instruction Counter, Version 1.0 (Felipe Lessa)
Counting the number of instructions executed
Instructions so far:
14
R-type:
2
14%
R4-type:
0
0%
I-type:
10
71%
S-type:
0
0%
B-type:
2
14%
U-type:
0
0%
J-type:
0
0%
Tool Control
Disconnect from Program
Reset
Close
Value (+0)
Value (+4)
0x10010000 (.data)
Hexadecimal Addresses

Tipo delle istruzioni

Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
`add`	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
`sub` (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
Istruzione (I)	**immediato**		**rs1**	**funz3**	**rd**	**codop**	**Esempio**
`addi` (addizione immediata)	001111101000		00010	000	00001	0010011	addi x1,x2,1000
`ld` (caricamento di parola doppia)	001111101000		00010	011	00001	0000011	ld x1, 1000 (x2)
Istruzione (S)	**Immediato**	**rs2**	**rs1**	**funz3**	**immediato**	**codop**	**Esempio**
`sd` (memorizzazione di parola doppia)	0011111	00001	00010	011	01000	0100011	sd x1, 1000 (x2)

Tipo delle istruzioni

Tipo S	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
	sd	0100011	111	n.a.
Tipo SB	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.

Esercizio 1 - if … then

Scrivere le sequenze di istruzioni RISC-V corrispondente ai seguenti frammenti di pseudocodice. Si supponga che le variabili **x**, **y** siano contenute rispettivamente nei registri **t0**, **t1**.

Frammento 1

```
x = x - y
if (x < 0)
      x = 0
y = y - 1
```

Frammento 2

```
x = (x - 2) + y
if (x < y)
      x = x + 1
else
      y = y + 1
```

Esercizio 1 - if … then

Frammento 1

```
x = x - y
if (x < 0)
      x = 0
y = y - 1
```

```
        bge   t0, zero, end1          # if !(x < 0) jump
                                      #
end1:                                 # fi
                                      #
```

Esercizio 1 - if … then

Frammento 1

```
x = x - y
if (x < 0)
        x = 0
y = y - 1
```

```
.text
_start:
        li    t0, 2
        li    t1, 1

        sub   t0, t0, t1            # x = x - y
        bge   t0, zero, end1        # if !(x < 0) jump
        li    t0, 0                 #   x = 0
end1:                               # fi
        addi  t1, t1, -1            # y = y - 1
```

Esercizio 1 - if … then

Frammento 2

```
x = (x - 2) + y
if (x < y)
        x = x + 1
else
        y = y + 1
```

```
        bge     t0, t1, else2           # if !(x < y) jump
                                        #
        j       end2                    # end
else2:
                                        #
end2:                                   # fi
```

Esercizio 1 - if … then

Frammento 2

```
x = (x - 2) + y
if (x < y)
        x = x + 1
else
        y = y + 1
```

```
.text
_start:
        li    t0, 1
        li    t1, 2

        addi  t0, t0, -2          # x = (x - 2)
        add   t0, t0, t1          # x = x + y

        bge   t0, t1, else2       # if !(x < y) jump
        addi  t0, t0, 1           #         x = x + 1
        j     end2                # end
else2:
        addi  t1, t1, 1           # y = y + 1
end2:                             # fi
```

Esercizio 2 - Max

Si scriva un programma in linguaggio RISC-V che carichi tre numeri interi su **t0, t1 e t2**, e poi inserisca il valore massimo tra i tre nel registro **t3**.

Esercizio 2 - Max

```
# max of three numbers present in registers
.text
    addi  t0, zero, 41   # load the numbers in the registers
    addi  t1, zero, 47
    addi  t2, zero, 45

    add   t3, zero, t0   # this register will hold the max
    bge   t3, t1, end1
    add   t3, zero, t1   # if t1 > t0, hold t1
end1:
    bge   t3, t2, end2
    add   t3, zero, t2   # if t2 > max(t1, t0), hold t2
end2:
    # t3 has the max

print:
    addi  a0, t3, 0
    li    a7, 1
    ecall
…
```

```
if !(t3 >= t1) {
    …
}
```

environment call.
per adesso:
a0 -> intero da stampare ad schermo

Esercizio 3 - Fibonacci Iterativo

Considerando il seguente frammento di codice che ritorna l'N-esimo numero della sequenza di Fibonacci - Fib(n) - scrivere l'equivalente in RISC-V. Assumere che la variabile N sia memorizzata nel registro t0. Il risultato finale (variabile R) va lasciato nel registro t1. Si utilizzino altri registri temporanei per le variabili A e B, e il minor numero possibile di istruzioni.

```
int N = 8;
int R = 1;
int A = 0; int B = 1;
while (N > 0) {
      R = A + B;
      A = B;
      B = R;
      N = N - 1;
}
```

- Quante istruzioni RISC-V sono necessarie per realizzare il frammento di codice C?
- Quante istruzioni RISC-V verranno eseguite per completare il ciclo quando N=8?

Esercizio 3 - Fibonacci Iterativo

```
loop1:
        ble t0, zero, end1

        j loop1
end1:
```

```
while (N > 0) {

}
```

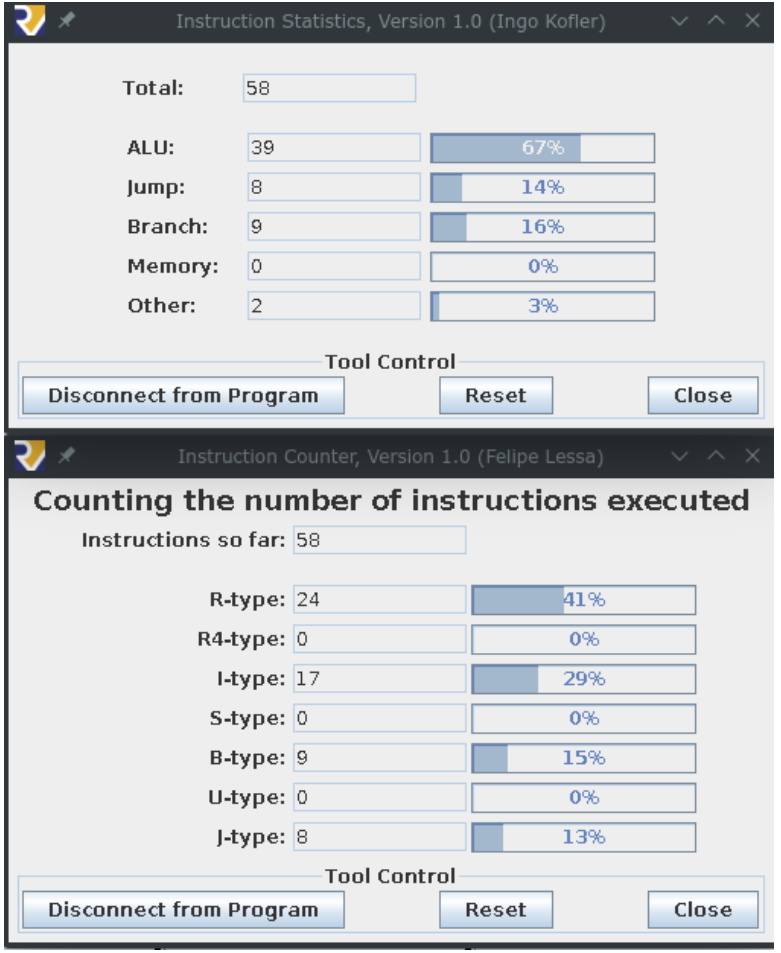
Esercizio 3 - Fibonacci Iterativo

```
.text
_start:
        li t0, 8
        li t1, 1
        li t2, 0
        li t3, 1
loop1:
        ble t0, zero, end1

        add  t1, t2, t3
        add  t2, t3, zero
        add  t3, t1, zero
        addi t0, t0, -1
        j loop1
end1:
```

```
int N = 8;
int R = 1;
int A = 0;
int B = 1;
while (N > 0) {

        R = A + B;
        A = B;
        B = R;
        N = N - 1;
}
```

Esercizio 3 - Fibonacci Iterativo

```
.text
_start:
        li t0, 8
        li t1, 1
        li t2, 0
        li t3, 1
loop1:
        ble t0, zero, end1

        add  t1, t2, t3
        add  t2, t3, zero
        add  t3, t1, zero
        addi t0, t0, -1
        j loop1
end1:
```



Esercizio 4 - Quadrati perfetti

Si scriva un programma RISC-V che calcoli la somma dei primi **N quadrati perfetti**. Il programma deve assumere che N sia nel registro **t1** e stampare a schermo la somma ottenuta.

- Quante istruzioni RISC-V sono necessarie?
- Quante istruzioni RISC-V verranno eseguite quando N=10?

Esercizio 4 - Quadrati perfetti

Soluzione possible in C

```
int N=10;
int S=0;
int i;
for (i=1; i<=N; ++i){
  S = S + i*i;
}
```

Esercizio 4 - Quadrati perfetti

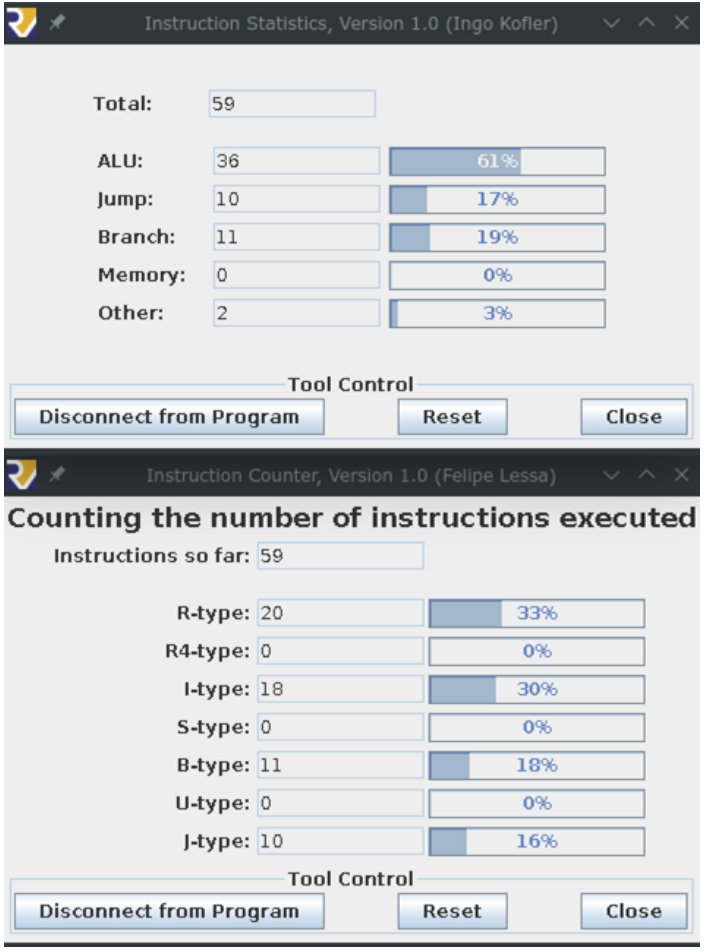
```
        li    t2, 1           # i=1
loop1:
        bgt   t2, t0, end1    # if (i > N) jump




        addi  t2, t2, 1       # i++
        j     loop1           # jump for
end1:
```

Esercizio 4 - Quadrati perfetti

```
.text
_start:
        li    t0, 10          # N
        li    t1, 0           # S

        li    t2, 1           # i=1
loop1:
        bgt   t2, t0, end1    # if (i > N) jump

        mul   t3, t2, t2      # t3 = i*i
        add   t1, t1, t3      # S = S + t3;

        addi  t2, t2, 1       # i++
        j     loop1           # jump for
end1:
```

Esercizio 4 - Quadrati perfetti

```
.text
_start:
        li    t0, 10          # N
        li    t1, 0           # S

        li    t2, 1           # i=1
loop1:
        bgt   t2, t0, end1    # if (i > N) jump

        mul   t3, t2, t2      # t3 = i*i
        add   t1, t1, t3      # S = S + t3;

        addi  t2, t2, 1       # i++
        j     loop1           # jump for
end1:
```



Esercizio 5 - contauno

Scrivere il codice RISC-V che restituisce il numero di bit uguali a 1 contenuti nel valore binario presente nel registro **t0**. Per esempio, se **t0** ha il valore binario equivalente al numero intero 37, il risultato atteso è 3.

Suggerimento: usare opportunamente le istruzioni logiche **and**, **srl** ...

- Quante istruzioni RISC-V sono necessarie?
- Quante istruzioni RISC-V verranno eseguite quando t0=37?

Esercizio 5 - contauno

```
N  = 37
M  = 1
R  = 0
i  = 64
do {
    R = R + N&M
    N = N >> 1
    i = i - 1
} while (i > 0)
```

Esercizio 5 - contauno

```
i  = 64
do {


    i = i - 1
} while (i > 0)
```

```
      li    t3, 64              # i
loop1:


      addi  t3, t3, -1          # i = i - 1
      ble   t3, zero, end1      # if (i<=0) end
      j     loop1                   # else continue
end1:
```

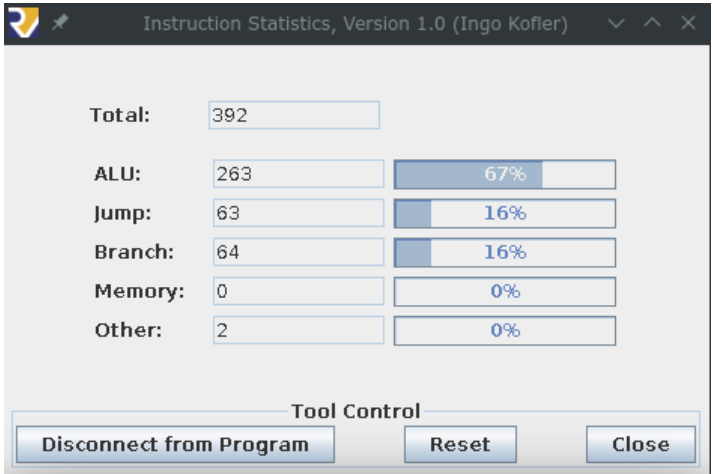
Esercizio 5 - contauno

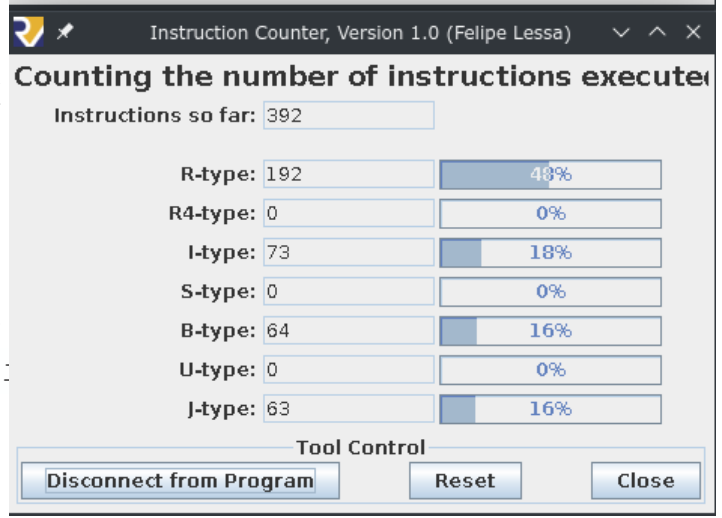
```
N  = 37
M  = 1
R  = 0
i  = 64
do {
    R = R + N&M
    N = N >> 1
    i = i - 1
} while (i > 0)
```

```
.text
_start:
      li    t0, 37          # N
      li    t1, 1           # M
      li    t2, 0           # R
      li    t3, 64          # i
loop1:
      and   t5, t0, t1      # N & M
      add   t2, t2, t5      # R = R + N & M
      srli  t0, t0, 1       # N = N >> 1

      addi  t3, t3, -1      # i = i - 1
      ble   t3, zero, end1  # if (i<=0) end
      j     loop1               # else continue
end1:
```

Esercizio 5 - contauno

```
.text
_start:
        li      t0, 37              # N
        li      t1, 1               # M
        li      t2, 0               # R
        li      t3, 64              # i
loop1:
        and     t5, t0, t1          # N & M
        add     t2, t2, t5          # R = R + N & M
        srli    t0, t0, 1           # N = N >> 1

        addi    t3, t3, -1          # i = i - 1
        ble     t3, zero, end1      # if (i<=0) end
        j       loop1                  # else conti
end1:
```





Esercizio 6 - Cicli FOR Annidati

Tradurre il seguente frammento di codice C in codice assembly RISC-V. Si utilizzi il minor numero possibile di istruzioni. Si supponga che le variabili a, b e R siano contenute rispettivamente nei registri t0, t1, t2

```
for (i=0; i<a; i++) {
     for (j=0; j<b; j++) {
          R = 2*R + i + j;
   }
}
```

- Quante istruzioni RISC-V sono necessarie per realizzare il frammento di codice?
- Supponendo che le variabili a e b vengono inizializzate a 10 e 5, quante istruzioni RISC-V verranno eseguite per completare il ciclo?

Esercizio 6 - Cicli FOR Annidati

```
for (i=0; i<a; i++) {
  for (j=0; j<b; j++) {
    R = 2*R + i + j;
  }
}
```

```
        li   t4, 0            # j
LOOPJ:
        bge  t4, t1, ENDJ     # when (j>=b) jump



        addi t4, t4, 1        # j++
        j    LOOPJ
ENDJ:
```

Esercizio 6 - Cicli FOR Annidati

```
for (i=0; i<a; i++) {
  for (j=0; j<b; j++) {
    R = 2*R + i + j;
  }
}
```

```
        li   t3, 0            # i=0
LOOPI:
        bge  t3, t0, ENDI     # when (i>=a) jump
        li   t4, 0            # j
LOOPJ:
        bge  t4, t1, ENDJ     # when (j>=b) jump




        addi t4, t4, 1        # j++
        j    LOOPJ
ENDJ:
        addi t3, t3, 1        # i++
        j    LOOPI
ENDI:
```

Esercizio 6 - Cicli FOR Annidati

```
for (i=0; i<a; i++) {
  for (j=0; j<b; j++) {
    R = 2*R + i + j;
  }
}
```

```
        li   t3, 0            # i=0
LOOPI:
        bge  t3, t0, ENDI     # when (i>=a) jump
        li   t4, 0            # j
LOOPJ:
        bge  t4, t1, ENDJ     # when (j>=b) jump
        add  t2, t2, t2       # R=2R
        add  t5, t3, t4       # i+j
        add  t2, t2, t5       # R+=i+j
        addi t4, t4, 1        # j++
        j    LOOPJ
ENDJ:
        addi t3, t3, 1        # i++
        j    LOOPI
ENDI:
```

Esercizio 6 - Cicli FOR Annidati

```
        li   t3, 0            # i=0
LOOPI:
        bge  t3, t0, ENDI     # when (i>=a) jump
        li   t4, 0            # j
LOOPJ:
        bge  t4, t1, ENDJ     # when (j>=b) jump
        add  t2, t2, t2       # R=2R
        add  t5, t3, t4       # i+j
        add  t2, t2, t5       # R+=i+j
        addi t4, t4, 1        # j++
        j    LOOPJ
ENDJ:
        addi t3, t3, 1        # i++
        j    LOOPI
ENDI:
```