Lab 4

Istruzioni per Prendere Decisioni

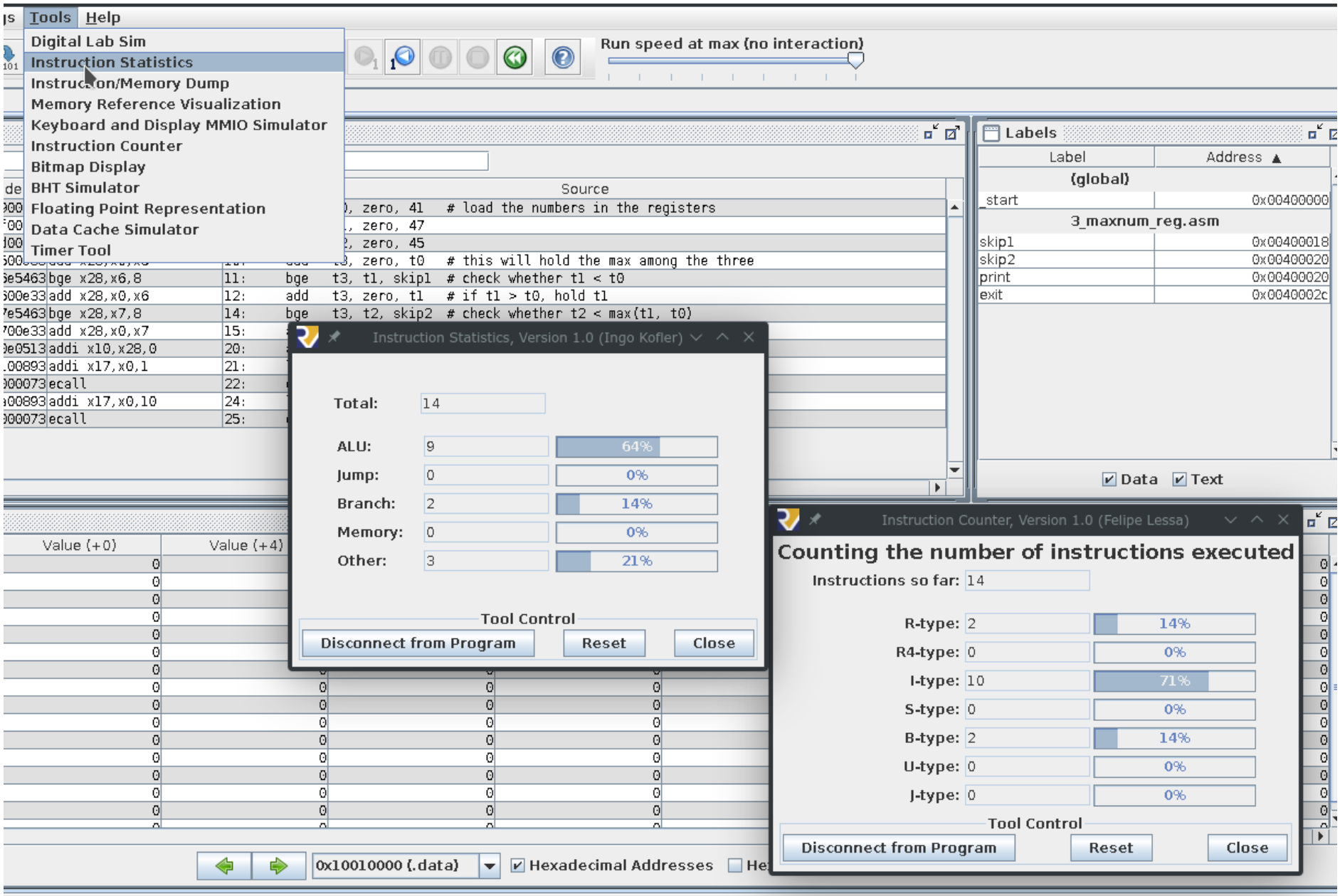
Obiettivi

- Confronto fra un codice C/Java e la sua traduzione 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
bge x28,x6,8
bge t3, t1, skip1 # check whether t1 < t0
add x28,x0,x6
add t3, zero, t1 # if t1 > t0, hold t1
bge x28,x7,8
bge t3, t2, skip2 # check whether t2 < max(t1, t0)
add x28,x0,x7
addi x10,x28,0
addi x17,x0,1
ecall
addi x17,x0,10
ecall
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.

decision making in RISCV

- In C (o in Java) abbiamo il controllo di flusso
  - l'output di test/confronti determina quali blocchi di codice vengono eseguiti
- in RISCV non possiamo definire blocchi di codice: **abbiamo solo le LABELS**
  - definite da testo seguito dai due punti (per esempio `_start:`, o `loop1:`)
- il controllo di flusso è realizzato tramite salti alle labels

istruzioni

- `beq, bne, bge, blt`
- hanno due registri come operandi, e
- richiedono di specificare un indirizzo
  - codifica della label

istruzioni

`beq, bne, bge, blt`

- costrutti tipicamente usati per l'iterazione (`if-else, while, for`)
- i loop sono tipicamente di dimensione ridotta (< 50 istruzioni)
- le istruzioni risiedono in un'area localizzata di memoria (`.text`)
  - salti più ampi sono limitati dalla dimensione del codice
  - indirizzo dell'istruzione corrente risiede nel program counter (PC)

indirizzamento relativo al PC

- usa il campo immediate come offset (=scartamento) in complemento a 2 al PC
  - possiamo quindi specificare $\pm 2^{11}$ indirizzi dal PC
- le **istruzioni sono word-aligned**: significa che l'indirizzo è sempre un multiplo di 4 (in byte)
- invece di specificare $\pm 2^{11}$ byte dal PC, possiamo quindi indirizzare labels **nell'intervallo $\pm 2^{11}$ words = $\pm 2^{13}$ byte**

calcolo dell'offset

- `beq x19,x10,End # if(x19 == x10) salta a End`
  - salta alla prima istruzione che si trova sotto la label `End`

- **se non facciamo il salto (se fallisce il test)**

  `PC = PC+4 = prossima istruzione`

- **se facciamo il salto**

  `PC = PC+ (immediate*4)`

  - dove `immediate` è il numero di istruzioni fra l'istruzione corrente e l'etichetta, sia in avanti (+), sia indietro (-)

esempio

si inizia a contare
dall'istruzione successiva

```
Loop: beq x19,x10,End
  add x18,x18,x10
  addi x19,x19,-1
  j    Loop
End:
```

1
2
3
4

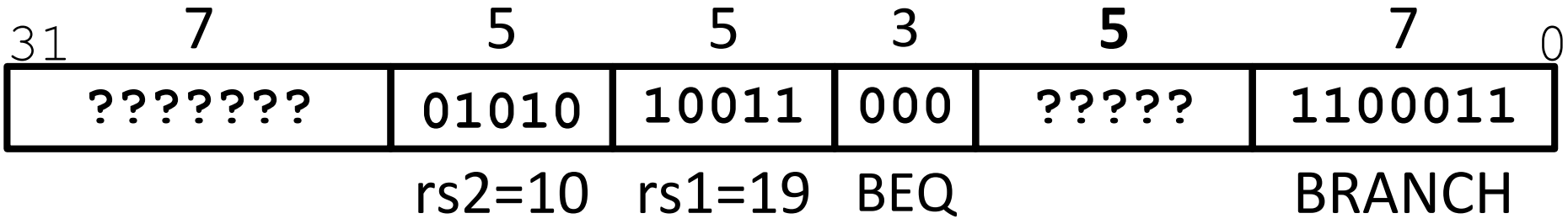
l'offset sarà quindi 4*32bit = 16 bytes

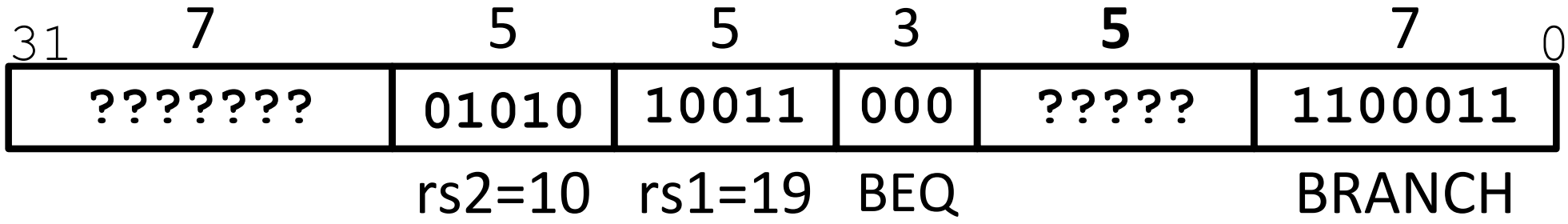
esempio

si inizia a contare
dall'istruzione successiva

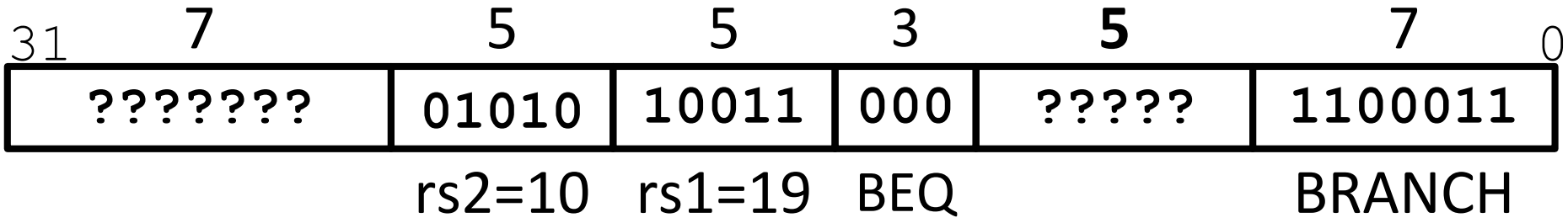
```
Loop: beq x19,x10,End   <-- 0
  add x18,x18,x10       1
  addi x19,x19,-1       2
  j    Loop             3
End:                    4
```

l'offset sarà quindi 4*32bit = 16 bytes



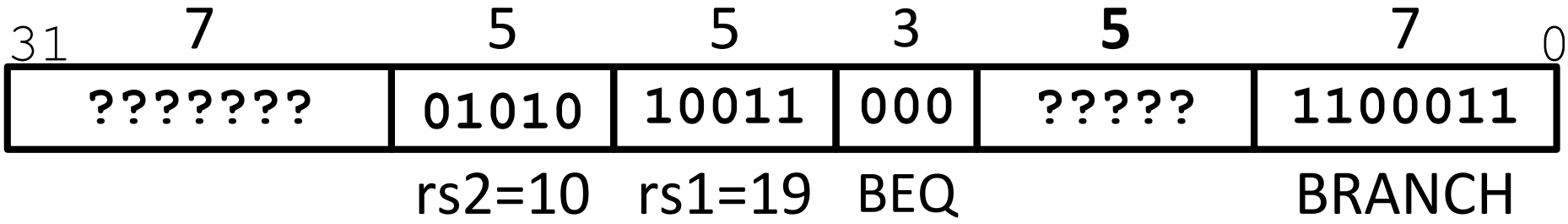


- il B-format contiene 2 registri source (`rs1` e `rs2`) e un `immediate` a 12 bit.
- l'immediate rappresenta valori da -2^{12} a $+2^{12}-2$ con incrementi di passo 2-byte
- i 12 bit dell'immediate codificano offset anche a 13-bit perché il bit più basso dell'offset è sempre a zero, quindi non è necessario memorizzarlo



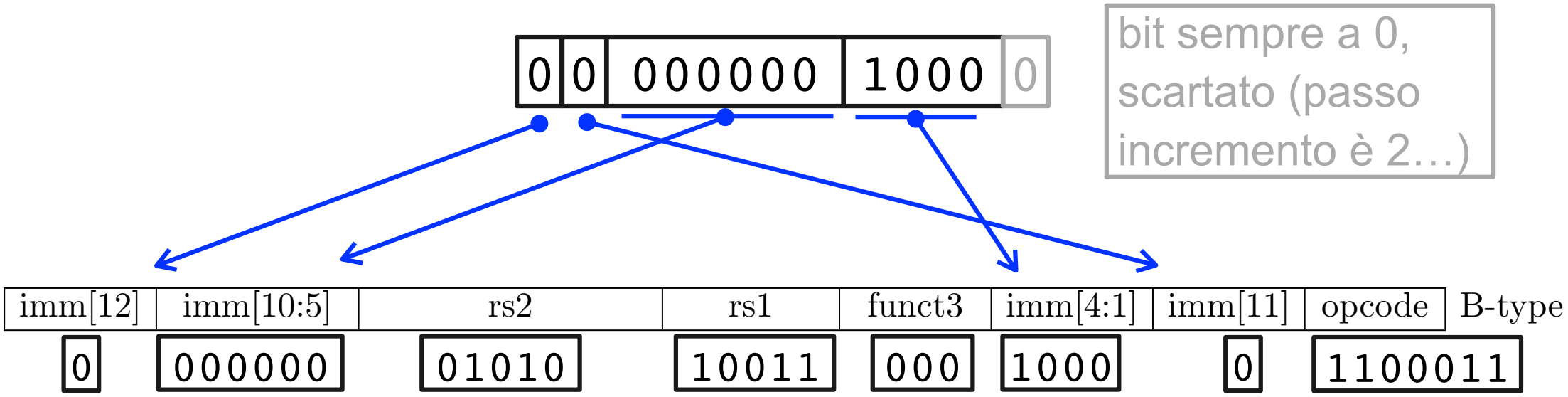
`beq x19,x10,offset = 16 bytes`

- immediate a 13 bit, `imm[12:0]` contenente il valore 16



`beq x19,x10,offset = 16 bytes`

- immediate a 13 bit, `imm[12:0]` contenente il valore 16



istruzioni rilevanti per gli esercizi

- branch if equal (`beq`)
  - `beq, reg1, reg2, label`
  - se il valore in `reg1` = valore in `reg2`, si salta a `label`
- branch if not equal (`bne`)
  - `bne, reg1, reg2, label`
  - se il valore in `reg1` != da quello in `reg2`, si salta a `label`
- jump (`j`)
  - `j label`
  - salto incondizionato alla label `label`.

if/else

codice C

```
if(i==j){
  a = b // then
} else {
  a = -b // else
}
```

in italiano:

se TRUE, allora esegui blocco

se FALSE, allora esegui blocco

else

if/else

codice C

```
if(i==j){
  a = b // then
} else {
  a = -b // else
}
```

in italiano:

se TRUE, allora esegui blocco

se FALSE, allora esegui blocco else

RISCV (bne)

```
# i -> s0, j -> s1
# a -> s2, b -> s3

bne s0,s1,???
???
add s2,s3,x0
j end
else:
sub s2,x0,s3
end:
```

altri test…

- branch less than (`blt`)
  - `blt reg1,reg2,label`
  - se il valore in `reg1` < del valore in `reg2`, vai a `label`
- branch greater than or equal (`bge`)
  - `bge reg1, reg2, label`
- branch equal (`beq`)
  - `beq reg1, reg2, label`

costrutti per iterazione

- costrutti in linguaggi di alto livello: `while`, `do… while`, `for`
- IDEA: il controllo del flusso è realizzato fondamentalmente attraverso i test condizionali

esercizi

Lab 4 - 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
```

Lab 4 - Esercizio 1 - if … then

Frammento 1

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

```
        bge    t0, zero, end1
end1:
```

```
# x = x - y
# if !(x < 0) jump
#    x = 0
# fi
# y = y - 1
```

Lab 4 - 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
```

Lab 4 - 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
```

Lab 4 - 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
```

Lab 4 - 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**.

Lab 4 - 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?

Lab 4 - 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?

Lab 4 - 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;
}
```

Lab 4 - 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 (100101_2), 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?

Lab 4 - 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)
```

Lab 4 - Esercizio 6 - Cicli FOR Annidati

Tradurre il seguente frammento di codice C in codice assembler 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 5 e 3, quante istruzioni RISC-V verranno eseguite per completare il ciclo?