
IL RISKIATUTTO
GIOCO CONDOTTO DA
Giorgio Delzanno
NUOVISSIMO
10 PARTITE AL TABELLONE
600 DOMANDE
RISC-V assembly
main:
li a0, 1
la a1, msg
call printf
j main
msg:
.asciz "Riskiatutto!\n"
CPU
MEMORIA
ALU
BUS
ESP32
PC
SP
RA
t0
t1
t2
t3
t4
t5
t6
Cache
RAM
ROM
lw t0, 0(sp)
addi t0, t0, 1
sw t0, 0(sp)
beq t0, zero, end
j loop
end: nop
RISC-V
RIDUZIONE
E ADATTAMENTO
DI MOGI.GIANINA
ITADINFO
CONVEGNO ITALIANO
SULLA DIDATTICA DELL'INFORMATICA

Un po’ di storia

- ***Rischiatutto*** è stato un programma televisivo italiano di genere telequiz, trasmesso dalla Rai dal 1970 al 1974 per cinque edizioni. Condotto da Mike Bongiorno, andò in onda sul Secondo Programma dal 5 febbraio 1970 al 25 maggio 1974.
- L'idea originale fu presa dal format americano del programma *Jeopardy!*, ma invece del peculiare meccanismo secondo cui a una risposta bisognava trovare una domanda, la RAI optò per il classico meccanismo domanda-risposta.
- La trasmissione, inizialmente, doveva chiamarsi *A repentaglio!*, ricalcando il titolo originale del programma; il nome venne cambiato durante la fase di preparazione.





Ma è del mestiere questo?



Obiettivi, materiale e metodologia

- Introduzione ai concetti dell'architettura degli elaboratori e della programmazione
- Materiale e tool che si potrebbero usare in classe
  - Guide, cheatsheet, simulatori per RISC V
  - Useremo un simulatore «giocoso» fatto per questo laboratorio per impare una versione semplificata del linguaggio assembly RISC V
- Metodologia:
  - partendo da linguaggio assembly RISC V si costruiscono le astrazioni più comuni usate nei linguaggi ad alto livello: iterazione, array, alternativa, chiamata di sottoprogrammi e procedure
- Labo per la scuola:
  - Microprocessori come ESP32 e loro simulatori per possibili laboratori da maker

Architettura dei Calcolatori

- Computer Architecture =

  insieme delle istruzioni (linguaggio, ISA) +

  organizzazione interna (registri, memorie…)

- Un’architettura non definisce l’implementazione HW, ne descrive le funzionalità.

RISC e CISC

- Complex Instruction Set Computing (CISC) e Reduced Instruction Set Computing (RISC) sono le due principali classi di architetture
- RISC ha un insieme semplificato di operazioni che mirano a raggiungere obiettivi semplici in pochi cicli.
  - La CPU non ha bisogno di circuiti complessi per eseguire queste istruzioni, quindi potenzialmente sono più economiche e semplici da implementare.
- CISC è l'esatto opposto
  - In passato i programmatori scrivevano in Assembler
- I processori moderni AMD ed INTEL incorporano le filosofie di progettazione di entrambe le architetture.
  - L'architettura x86 è principalmente CISC ma ha un microcodice per convertire istruzioni complesse in semplici istruzioni ridotte di tipo RISC.

RISC-V

- Il progetto nasce nel 2010 dall'Università della California, Berkeley
- Standard APERTO per un'ISA basata su RISC. I principi:

1. La regolarità favorisce la semplicità
   Ad esempio il numero degli operandi
2. Il caso più frequente dev'essere anche il più veloce
3. Più piccolo, più veloce
   Pensiamo all'UTF-8 rispetto all'Unicode.
   Qui pochi registri e poche istruzioni semplici
4. Un buon progetto richiede compromessi astuti
   Tutte le istruzioni hanno la stessa lunghezza, 32 bit.
   Ma lo standard prevede possibili estensioni

https://riscv.org/

Materiale e tool

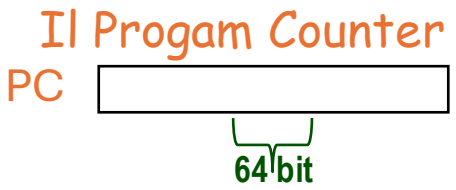
https://github.com/riscv/learn

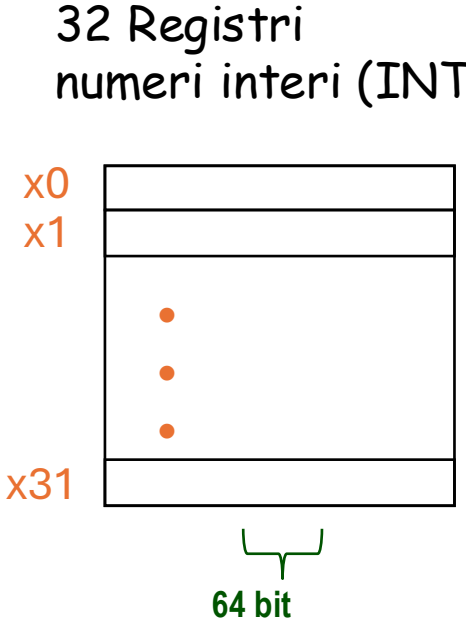
https://github.com/dvoytik/riscv-cheats

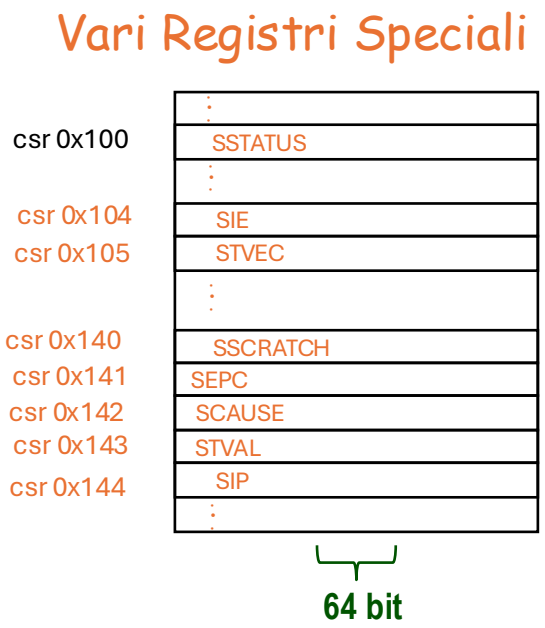
https://github.com/thethirdone/rars

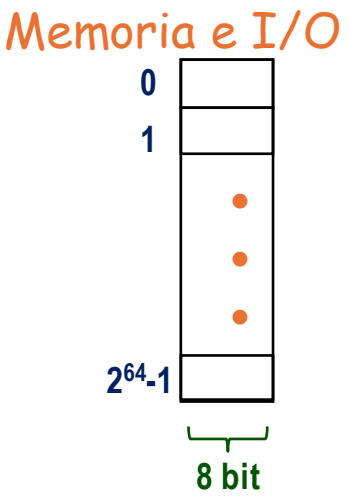
“Instruction Set Architecture”
(es. RISC-V RV64IMDF)

Registri e memoria





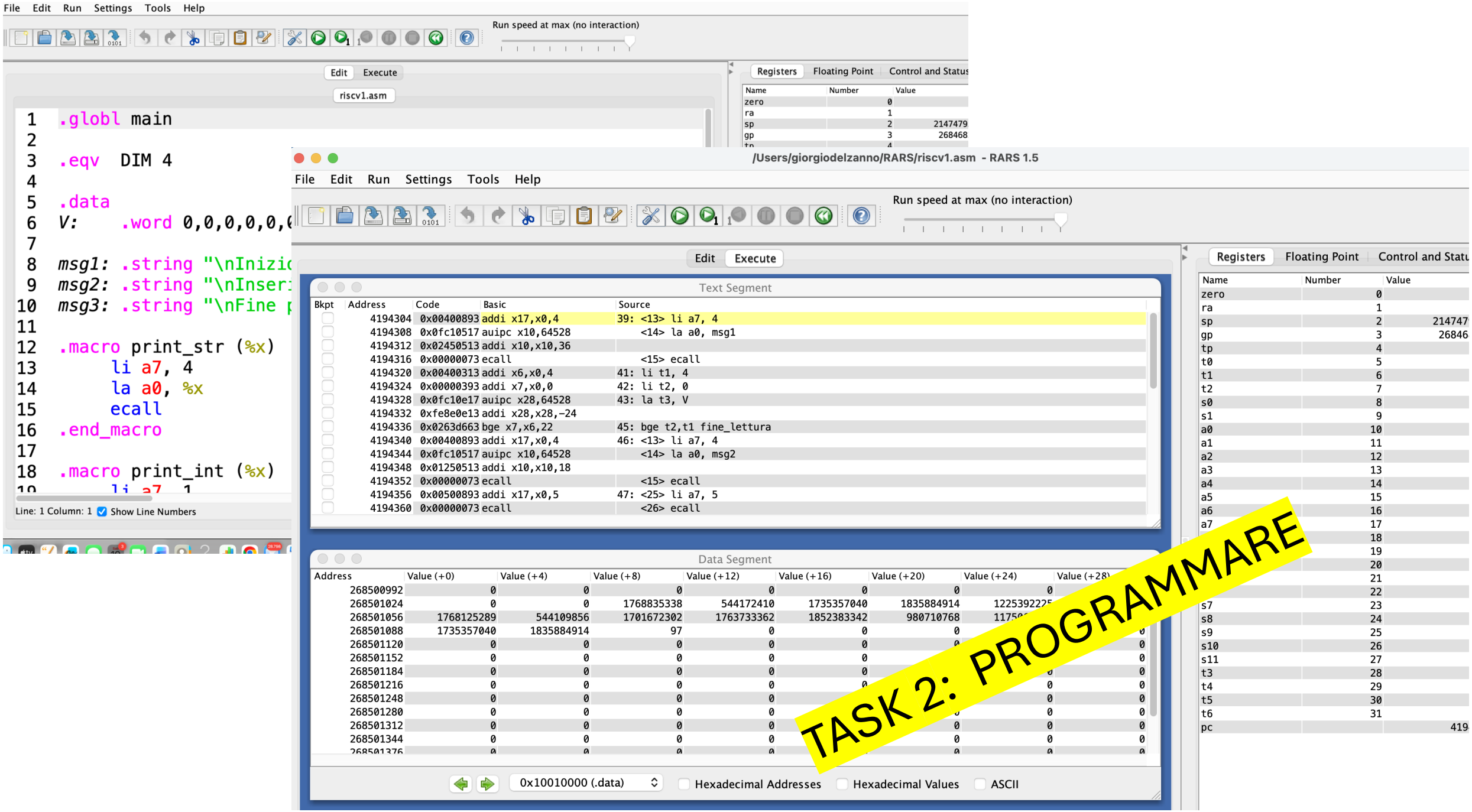




“Instruction Set Architecture”

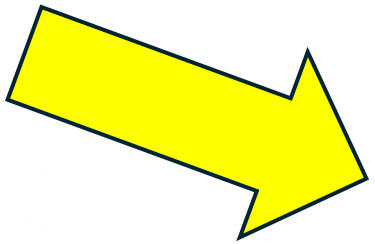
- Categorie di Istruzioni
  - Aritmetiche
  - Load/Store
  - Jump/Branch
  - Istruzioni speciali

TASK 1: SPIEGARE SEMANTICA


File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
riscv1.asm
1 .globl main
2
3 .eqv DIM 4
4
5 .data
6 V: .word 0,0,0,0,0,0
7
8 msg1: .string "\nInizio
9 msg2: .string "\nInseri
10 msg3: .string "\nFine p
11
12 .macro print_str (%x)
13 li a7, 4
14 la a0, %x
15 ecall
16 .end_macro
17
18 .macro print_int (%x)
Line: 1 Column: 1 Show Line Numbers
Registers Floating Point Control and Status
Name Number Value
zero 0
ra 1
sp 2 2147479
gp 3 268468
/Users/giorgiodelzanno/RARS/riscv1.asm - RARS 1.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic Source
4194304 0x00400893 addi x17,x0,4 39: <13> li a7, 4
4194308 0x0fc10517 auipc x10,64528 <14> la a0, msg1
4194312 0x02450513 addi x10,x10,36
4194316 0x00000073 ecall <15> ecall
4194320 0x00400313 addi x6,x0,4 41: li t1, 4
4194324 0x00000393 addi x7,x0,0 42: li t2, 0
4194328 0x0fc10e17 auipc x28,64528 43: la t3, V
4194332 0xfe8e0e13 addi x28,x28,-24
4194336 0x0263d663 bge x7,x6,22 45: bge t2,t1 fine_lettura
4194340 0x00400893 addi x17,x0,4 46: <13> li a7, 4
4194344 0x0fc10517 auipc x10,64528 <14> la a0, msg2
4194348 0x01250513 addi x10,x10,18
4194352 0x00000073 ecall <15> ecall
4194356 0x00500893 addi x17,x0,5 47: <25> li a7, 5
4194360 0x00000073 ecall <26> ecall
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+12) Value (+16) Value (+20) Value (+24) Value (+28)
268500992 0 0 0 0 0 0 0 0
268501024 0 0 1768835338 544172410 1735357040 1835884914 1225392225
268501056 1768125289 544109856 1701672302 1763733362 1852383342 980710768 1175
268501088 1735357040 1835884914 97 0 0 0 0
268501120 0 0 0 0 0 0 0
268501152 0 0 0 0 0 0 0
268501184 0 0 0 0 0 0 0
268501216 0 0 0 0 0 0 0
268501248 0 0 0 0 0 0 0
268501280 0 0 0 0 0 0 0
268501312 0 0 0 0 0 0 0
268501344 0 0 0 0 0 0 0
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Registers Floating Point Control and Statu
Name Number Value
zero 0
ra 1
sp 2 214747
gp 3 26846
tp 4
t0 5
t1 6
t2 7
s0 8
s1 9
a0 10
a1 11
a2 12
a3 13
a4 14
a5 15
a6 16
a7 17
18
19
20
21
22
s7 23
s8 24
s9 25
s10 26
s11 27
t3 28
t4 29
t5 30
t6 31
pc 419
TASK 2: PROGRAMMARE

```cpp
#include<iostream>
using namespace std;

int fattoriale(int n){
  int f;
  if(n==1) return 1;
  f=fattoriale(n-1)*n;
  return f;
}
main(){
  cout<<fattoriale(5);
}
```



```
.globl main
.data
 m1: .string "! = "
 m2: .string " \n"

.macro exit (%x)
     li a7, 93
     li a0, %x
     ecall
.end_macro

 .macro print_str (%x)
     li a7, 4
     la a0, %x
     ecall
.end_macro

.macro print_int (%x)
     li a7, 1
     mv a0, %x
     ecall
.end_macro
```

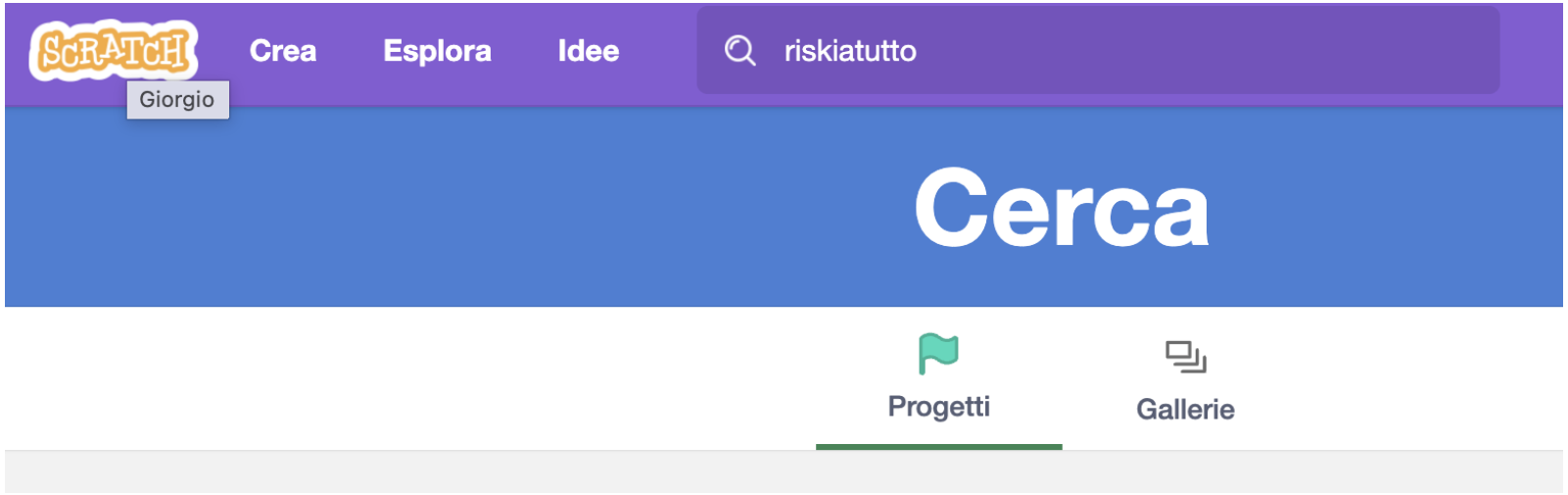
```
.text
main:
  li t3, -1 # per cancellare contenuto stack
  li t4, 1  # usato per controllo caso base
  li s0, 5  # salvo argomento
  mv a0,s0  # preparo argomento della chiamata
  jal fattoriale # chiamo la funzione
  mv s1,a0  # salvo il risultato
  print_int(s0) #stampo !s0=s1
  print_str(m1)
  print_int(s1)
  print_str(m2)
  exit(0)
```

```
fattoriale:
 addi sp, sp, -8 # alloco call-frame
 sw   a0, 4(sp)  # salvo argomento x sullo stack (servirà per x*fatt(x-1)
 sw   ra, 8(sp)  # salvo indirizzo di ritorno sullo stack
test:
 beq  a0, t4, base # se a0 == 1 salto a "base"
passo:
 addi a0, a0, -1  # preparo parametro per
 jal  fattoriale  # chiamo fattoriale(
 mv   t1, a0      # salvo risulta
 lw   a0, 4(sp)   # ripristi
 lw   ra, 8(sp)   # ripri                ritorni in ra
 sw   t3, 4(sp)   #                      me
 sw   t3, 8(sp)
 mul  a0,a0
                  att(x-1)
```

TASK 3: ASSEMBLARE

Riskiatutto: tutto in uno con … Scratch



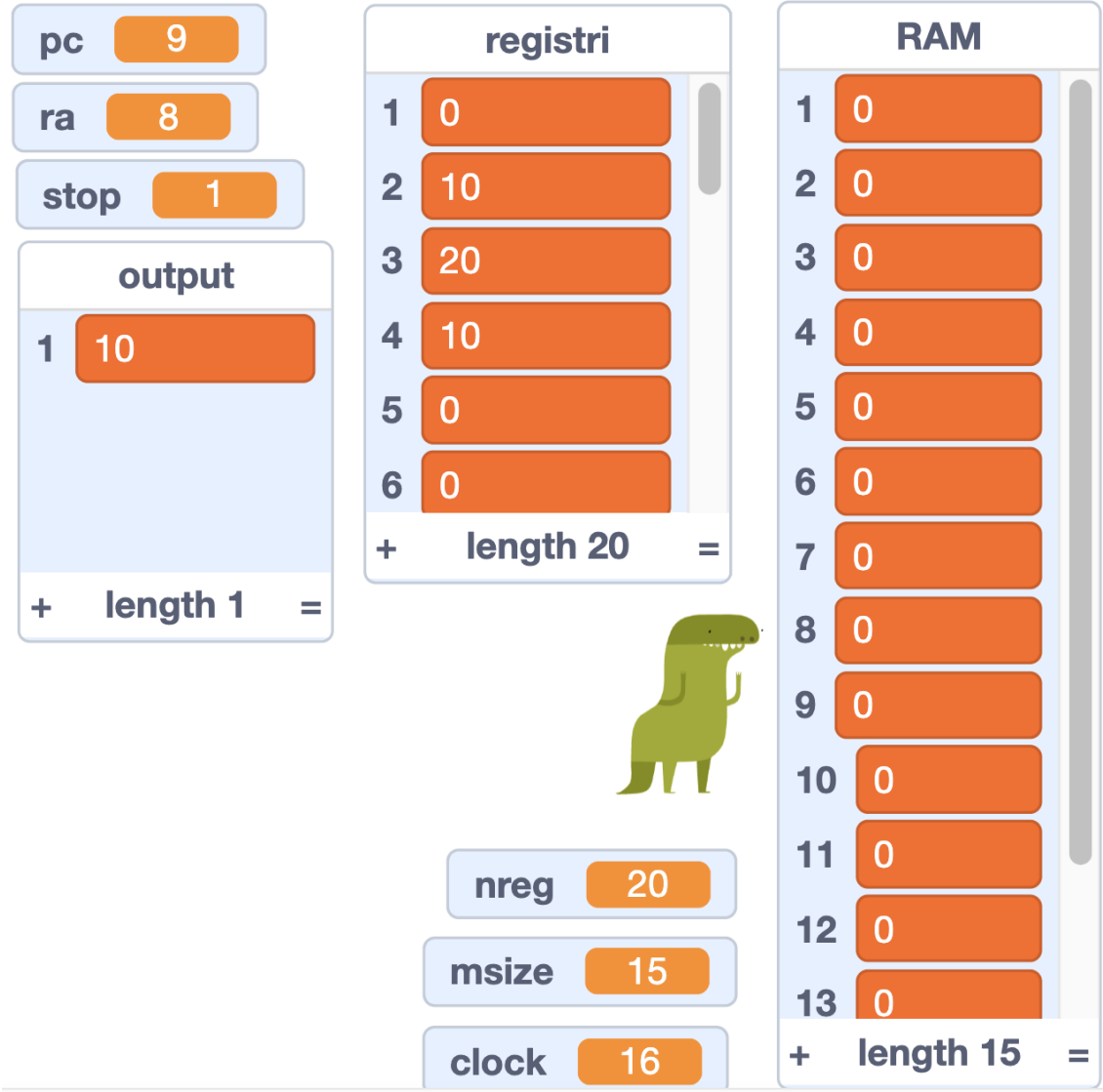


https://scratch.mit.edu/projects/1389293932/

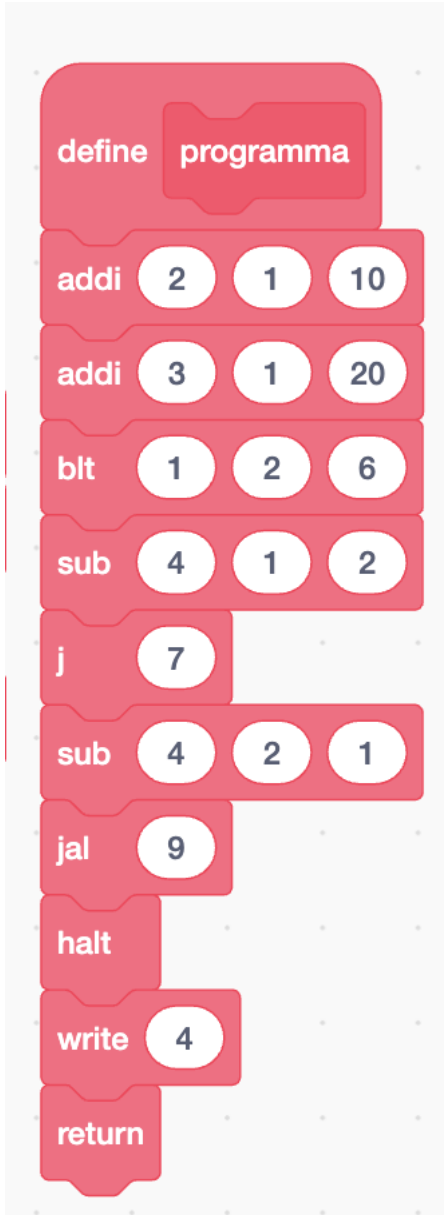
RISC V in Scratch

Interpretazione dei comandi come operazioni sulle seguenti strutture dati

- Registri = array di numeri
- Memoria RAM = array di numeri
- PC e RA = variabili

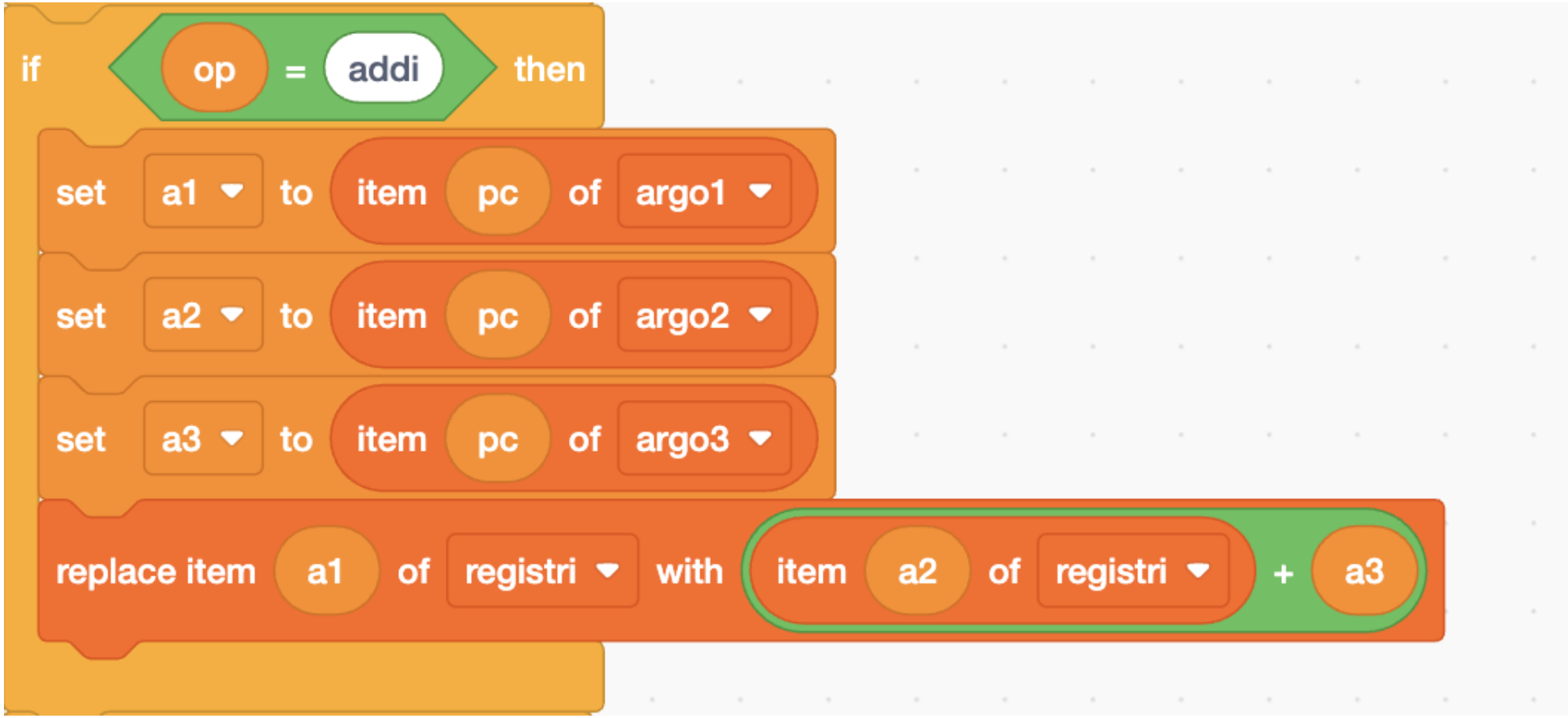


Programma generato con blocchi



	opcode	argo1	argo2	argo3
1	addi	2	1	10
2	addi	3	1	20
3	blt	1	2	6
4	sub	4	1	2
5	j	7	0	0
6	sub	4	2	1
7	jal	9	0	0

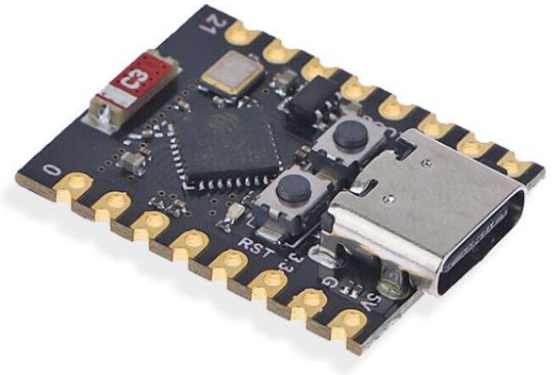
Semantica delle istruzioni = codice dell'interprete



Giochiamo!



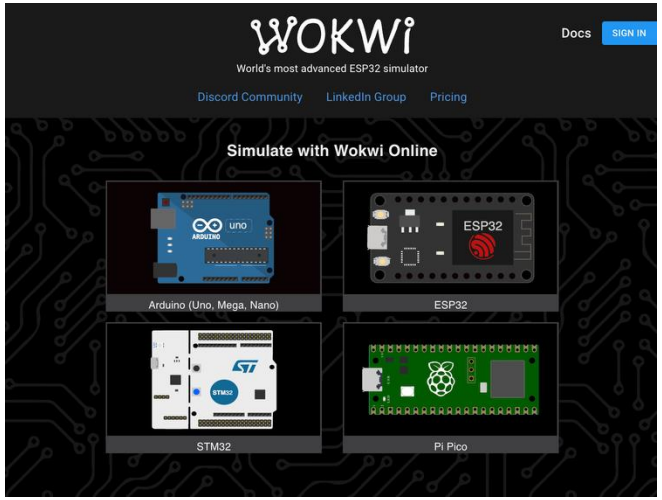
ESP32 e simulatori



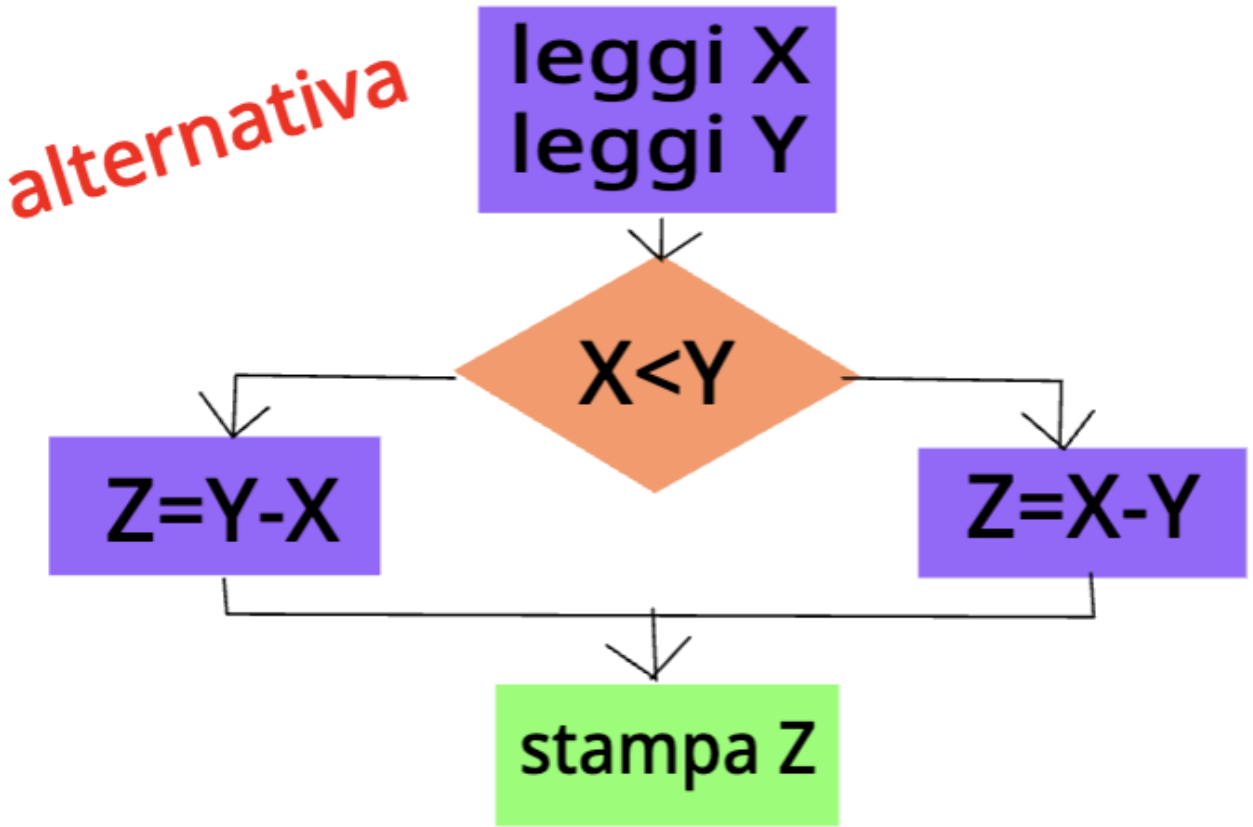
https://www.espressif.com/en/products/socs/esp32-c3

https://www.espressif.com/en/products/socs/esp32-c6

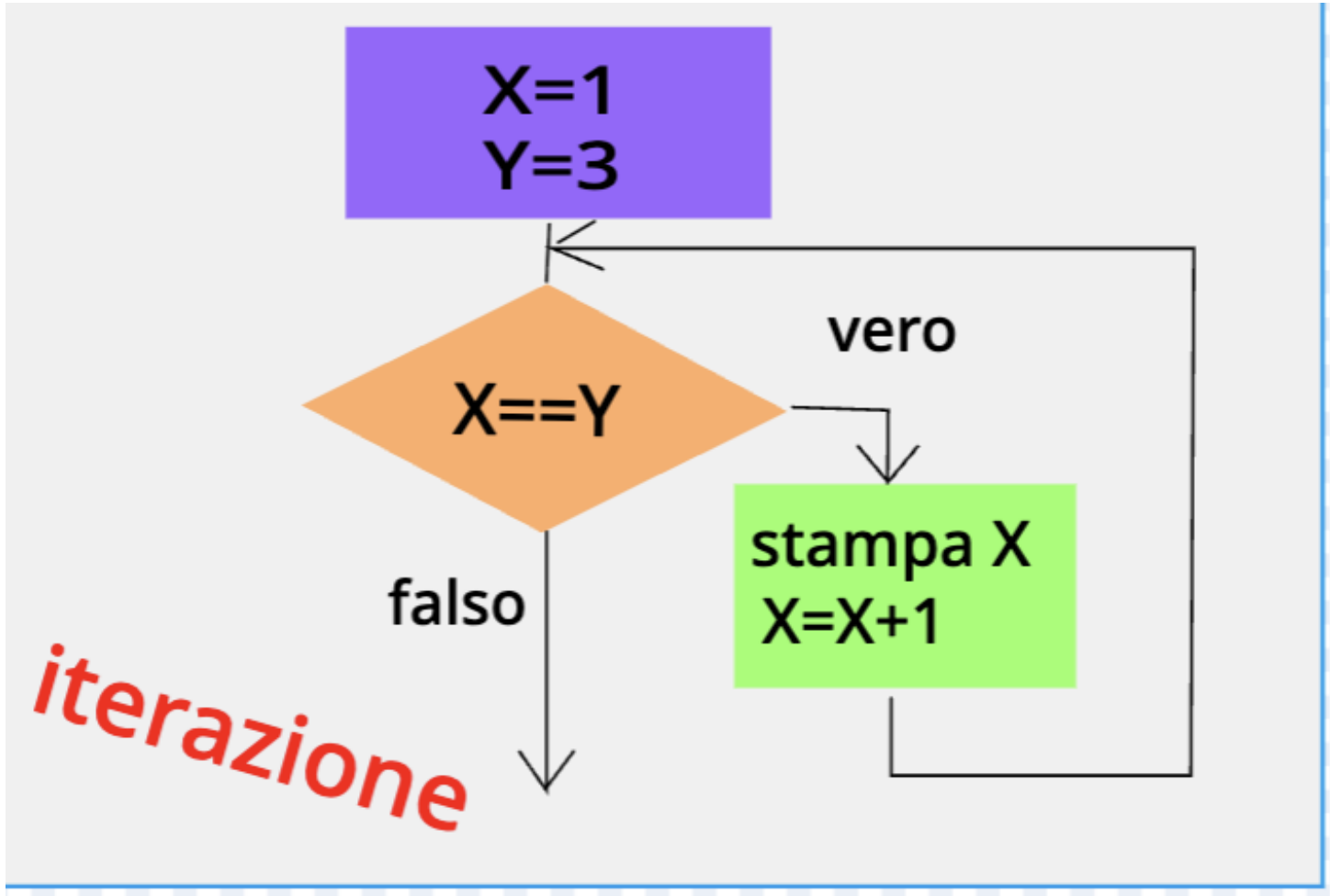
https://wokwi.com/projects/458731845768602625



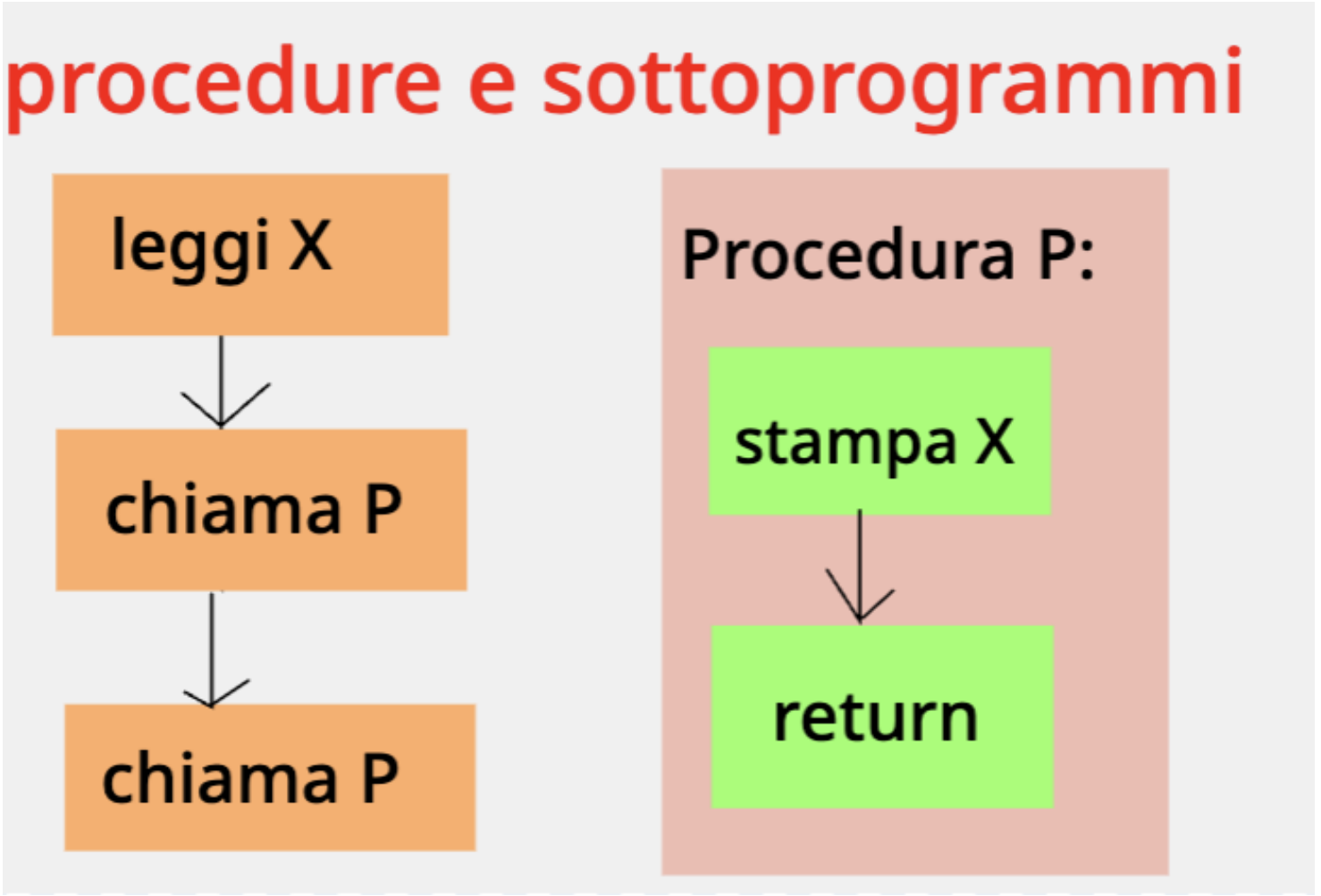
Esercizio 1



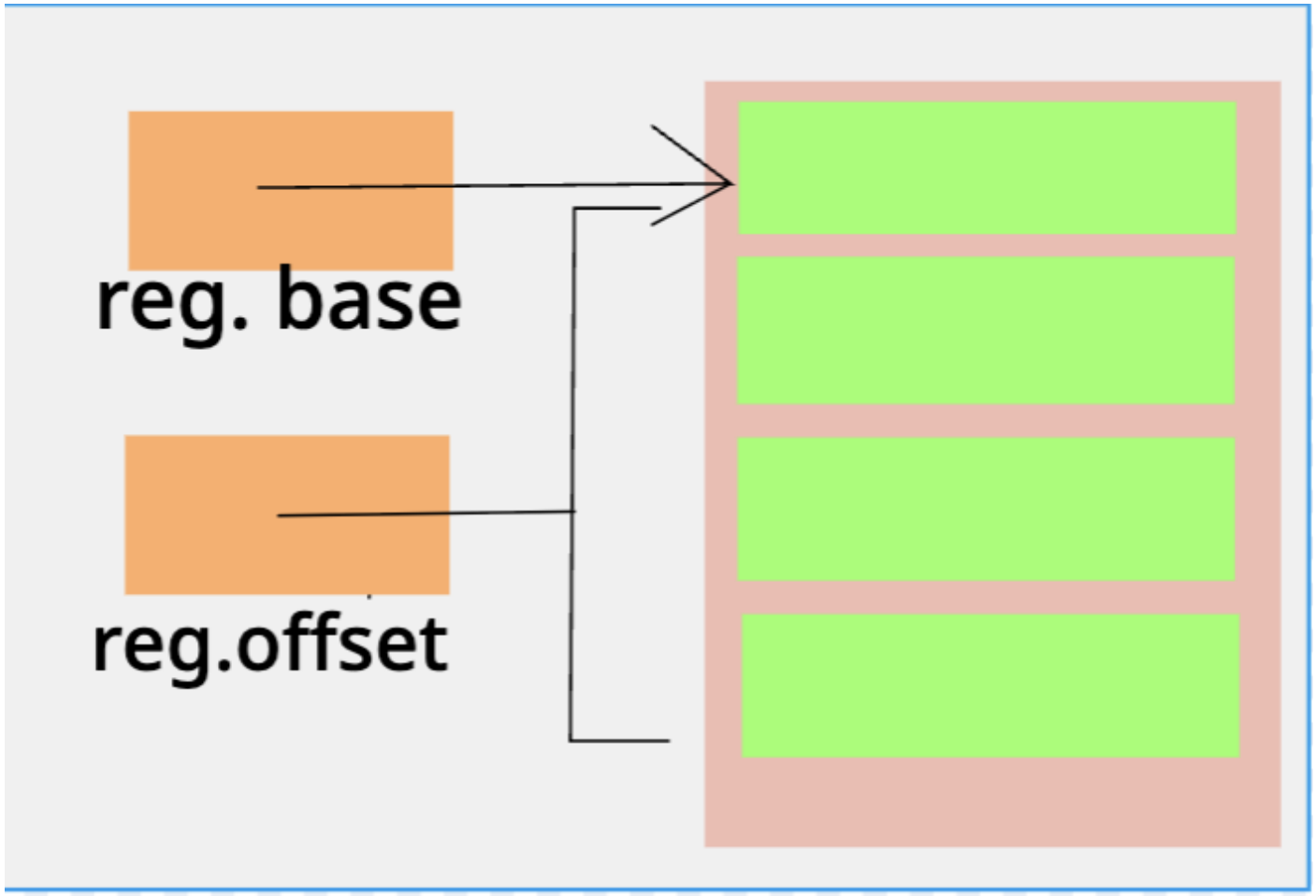
Esercizio 2



Esercizio 3

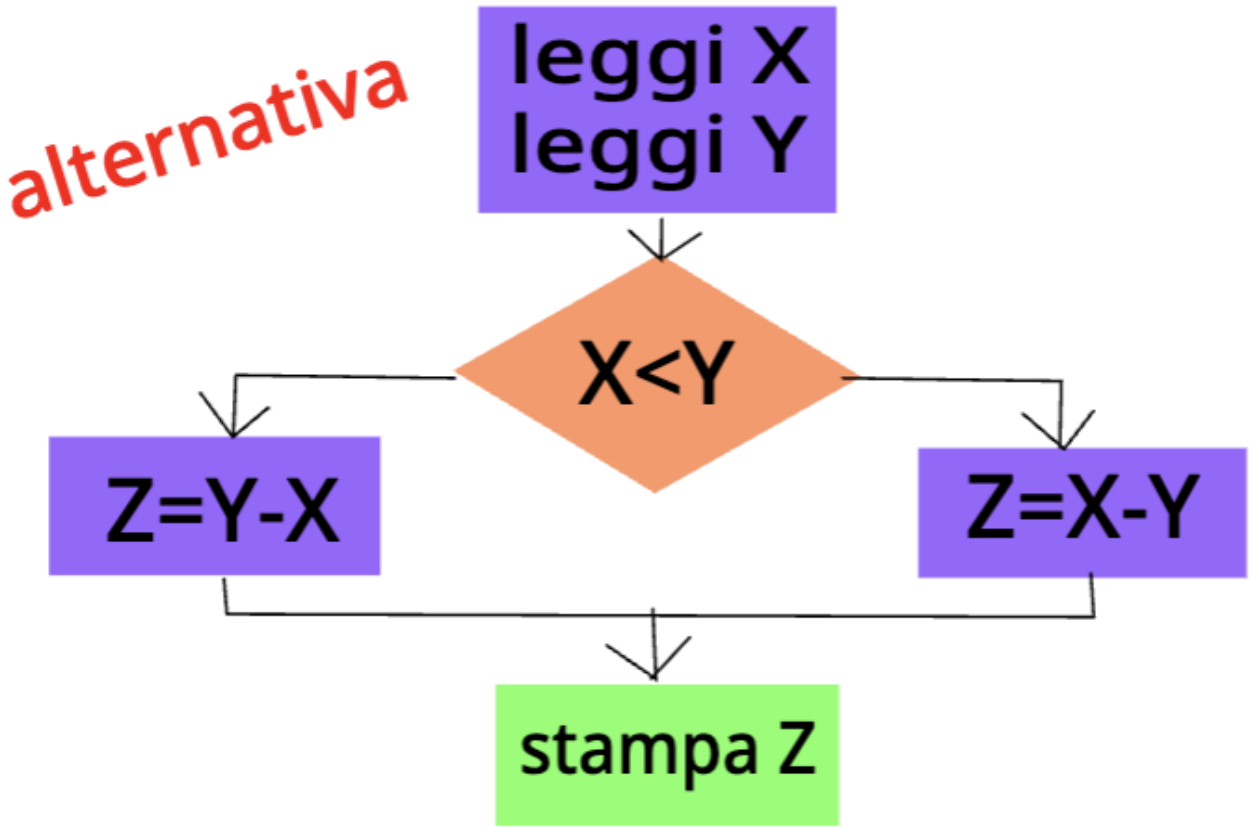

procedure e sottoprogrammi
leggi X
chiama P
chiama P
Procedura P:
stampa X
return

Esercizio 4



Esercizio 5

Esercizio 6



WOKWI SAVE SHARE Docs SIGN IN

main.c | diagram.json | blink.S | CMakeLists.txt | Library Manager

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"

void app_main(void) {
    // Configura GPIO2 come output
    gpio_set_direction(GPIO_NUM_2, GPIO_MODE_OUTPUT);

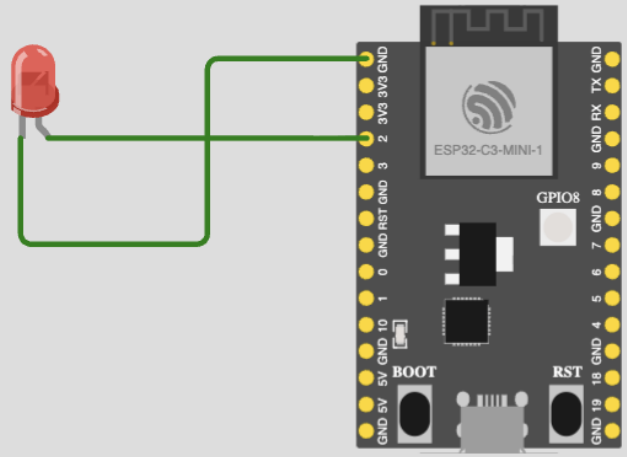
    while (1) {
        // LED HIGH
        asm volatile(
            "li t0, 0x60004000\n\t"
            "li t1, 0x08\n\t"
            "add t1, t0, t1\n\t"
            "li t3, 0x4\n\t"   // GPIO2 mask
            "sw t3, 0(t1)\n\t"
        );

        vTaskDelay(500 / portTICK_PERIOD_MS);

        // LED LOW
        asm volatile(
            "li t0, 0x60004000\n\t"
            "li t2, 0x0C\n\t"
            "add t2, t0, t2\n\t"
            "li t3, 0x4\n\t"
            "sw t3, 0(t2)\n\t"
        );

        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
}
```

Simulation



main.c | diagram.json | **blink.S** | CMakeLists.txt | Library Manager

```
.section .text
.global blink_asm

blink_asm:
    li t0, 0x60004000       # GPIO base
    li t1, 0x08             # GPIO_OUT_W1TS
    add t1, t0, t1
    li t2, 0x0C             # GPIO_OUT_W1TC
    add t2, t0, t2
    li t3, (1 << 2)         # GPIO2 mask
loop:
    sw t3, 0(t1)            # SET HIGH
    nop
    nop
    nop
    nop
    sw t3, 0(t2)            # SET LOW
    nop
    nop
    nop
    nop
    j loop
```

WOKWi SAVE SHARE

main.c | diagram.json | blink.S | **CMakeLists.txt** | Library Manager

```
df_component_register(
    SRCS "main.c" "blink.S"
    INCLUDE_DIRS "."
)
```