Calcolatori Elettronici
Esercitazione 8

M. Sonza Reorda – M. Monetti
M. Rebaudengo – R. Ferrero
L. Sterpone – E. Vacca

Politecnico di Torino
Dipartimento di Automatica e Informatica

Obiettivi

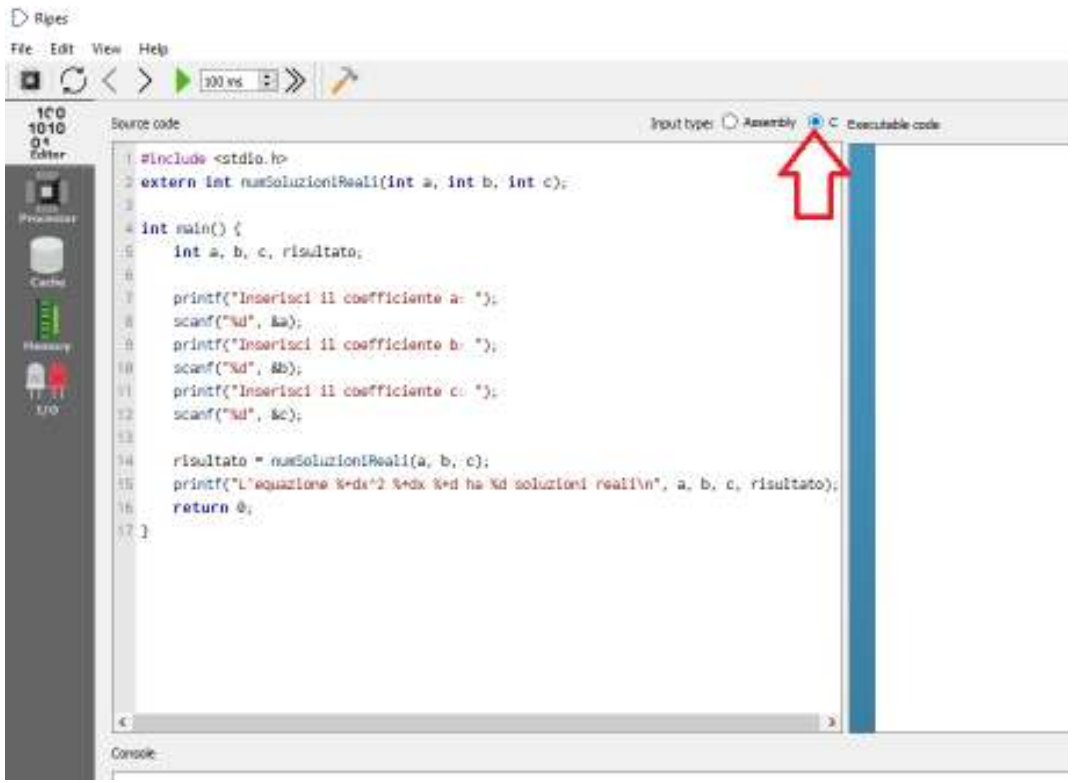
Codice C e assembly

Compilare codice C in Ripes

- Scaricare la toolchain per RISC-V contentente:
  - cross-compiler: riscv64-unknown-elf-gcc
  - assembler
  - linker
  - Debugger
  - https://github.com/sifive/freedom-tools/releases/tag/v2020.04.0-Toolchain.Only
- Il cross-compiler:
  - è un compilatore che gira su una certa architettura (Windows, Linux, Apple)
  - ma genera codice per un'architettura diversa
  - nel nostro caso il codice compilato sarà eseguito su una piattaforma RISC-V 64 bit embedded, non dotata di sistema operativo (ELF = Executable and Linkable Format per sistemi bare-metal).

Compilare codice C in Ripes

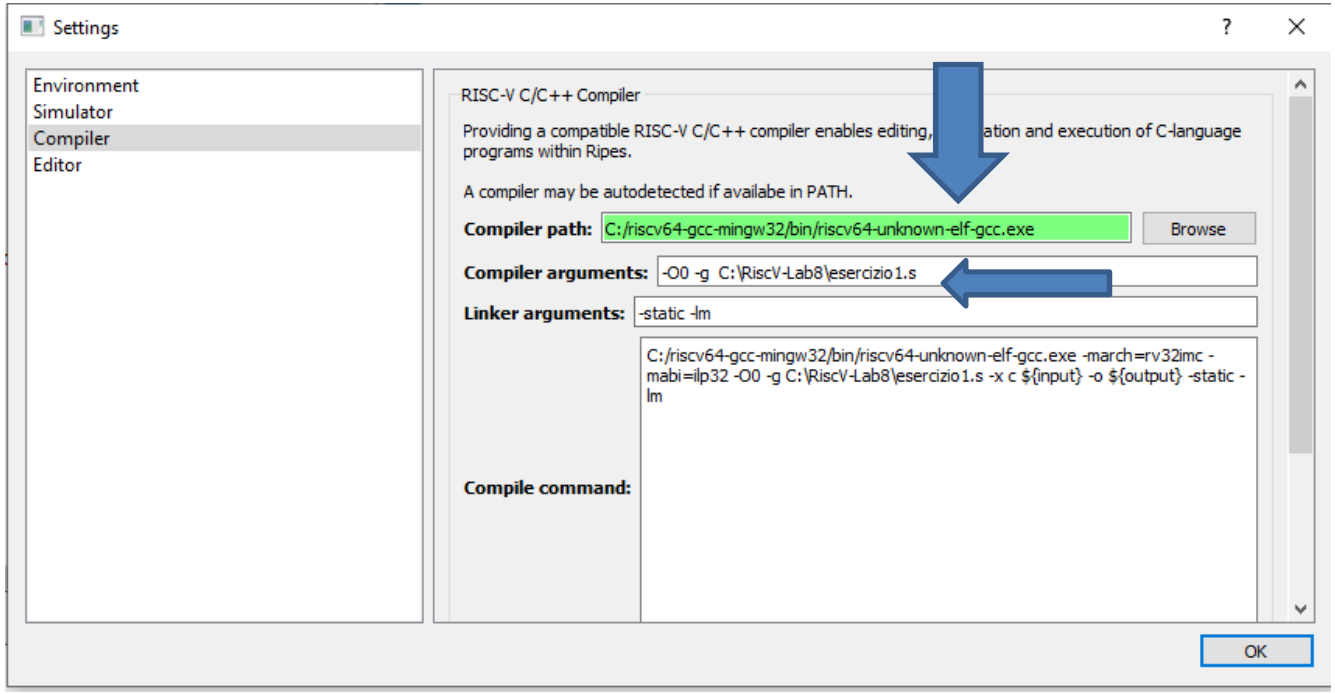
- Utilizzare simulatore Ripes.me versione desktop
  - https://github.com/mortbopet/ripes/releases



Selezione Input Type **C**

Compilare codice C in Ripes

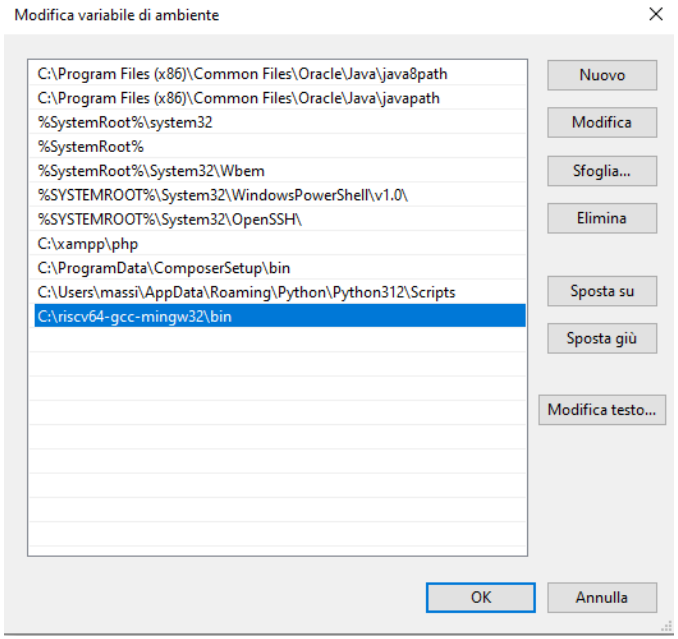
- Edit > Settings > Compiler



-O0 -g C:\RiscV-Lab8\esercizio1.s

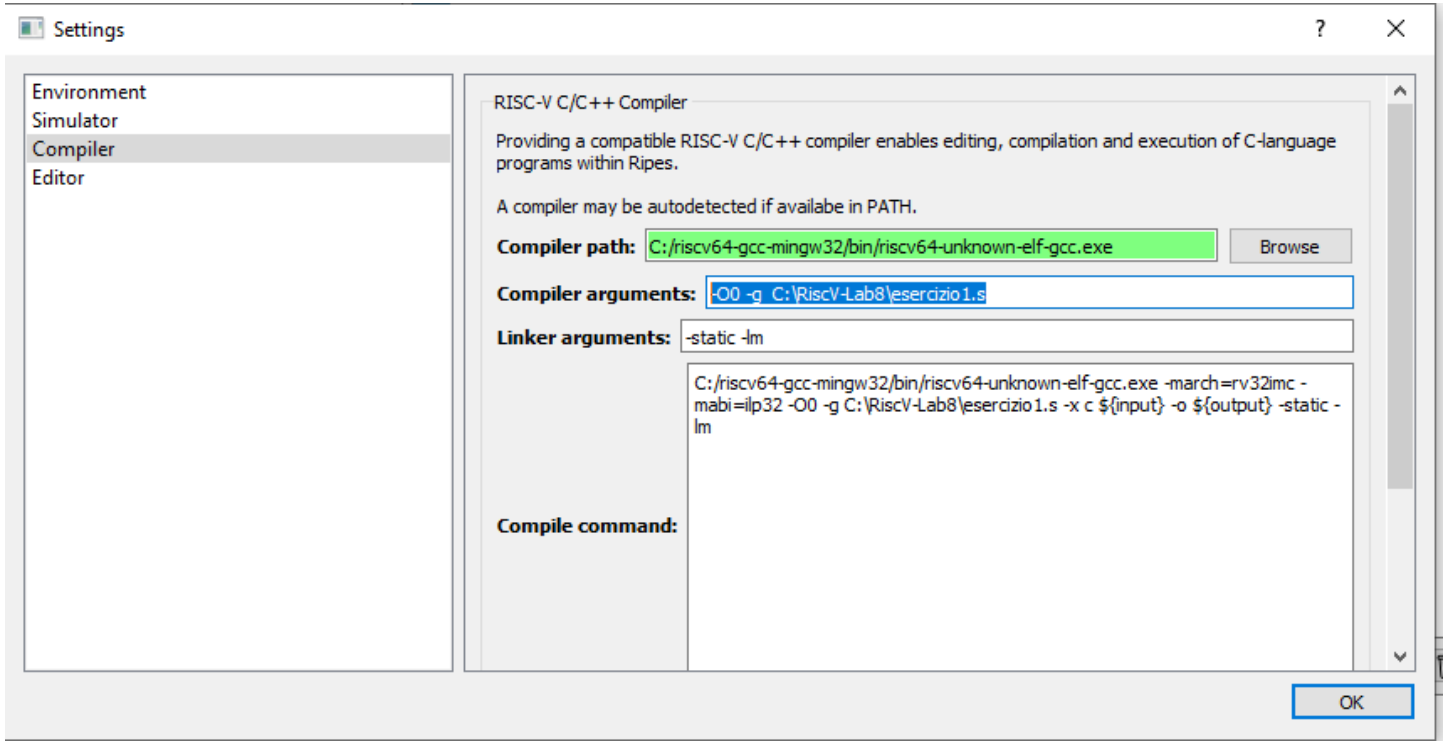
Compilare codice C in Ripes

- 1
  - Path della Toolchain per RISC-V *C:/riscv64-gcc-mingw32/bin/riscv64-unknown-elf-gcc.exe*
  - può essere ovunque, ma *C:/riscv64-gcc-mingw32/bin/riscv64-unknown-elf-gcc.exe* va inserita nelle variabili d'ambiente



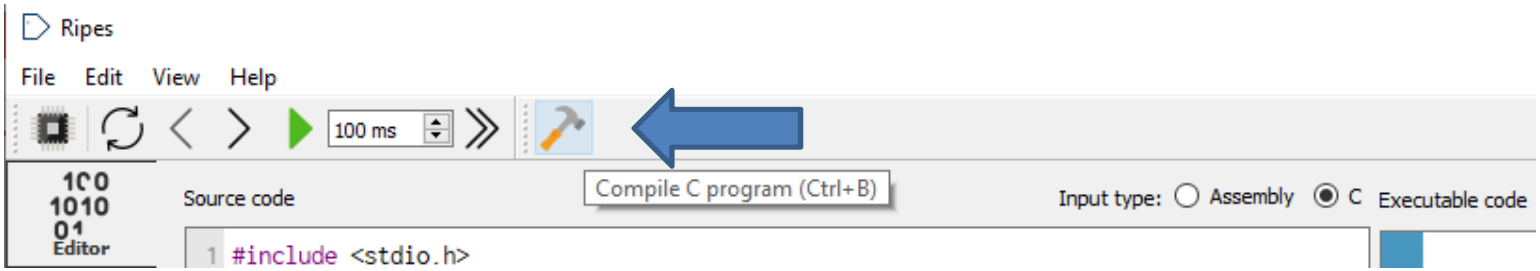
Compilare codice C in Ripes

- 2
  - Compiler arguments **-O0 -g C:\RiscV-Lab8\esercizio1.s**

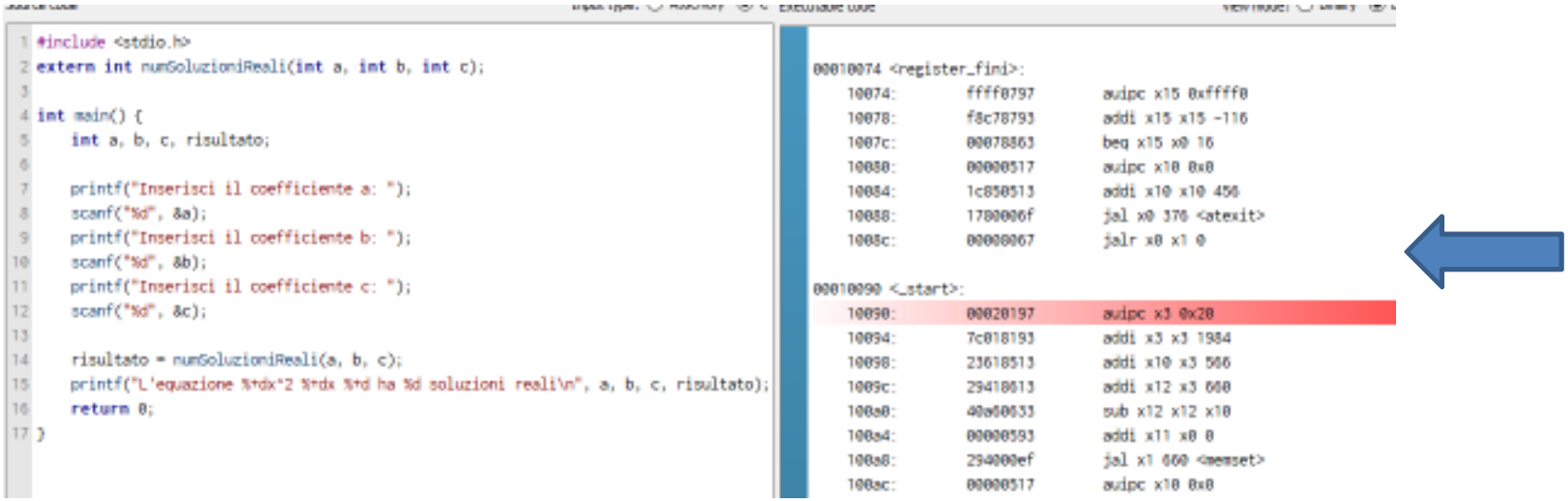


Compilare codice C in Ripes

- Verifica compilazione



- Se tutto ok, appare il disassemblato sulla dx



Nel file C, il prototipo della funzione deve essere preceduto da *extern*

- ***extern int numSoluzioniReali(int a, int b, int c);***

La procedura RISC-V deve essere globale

.globl numSoluzioniReali

Esercizio 1

- Si scriva la **procedura ASM *`numSoluzioniReali`*** che restituisce il numero di soluzioni reali di un'equazione di secondo grado nella forma

$$ax^2 + bx + c = 0$$

- a, b e c sono interi con segno passati come parametro alla procedura
- Si assuma che i calcoli non diano overflow.
- Per i salti condizionati, si utilizzino soltanto le istruzioni `slt`, `beq` e bne.

Esercizio 1

- Si scriva un **programma C** che acquisisca dall'utente 3 numeri interi a, b e c
- Successivamente, il programma chiama la procedura **`numSoluzioniReali`** passando i 3 valori
- Infine, il programma visualizza a video il valore restituito dalla procedura.

Esercizio 1.c [chiamante-C]

```
#include <stdio.h>
extern int numSoluzioniReali(int a, int b, int c);

int main() {
        int a, b, c, risultato;

        printf("Inserisci il coefficiente a: ");
        scanf("%d", &a);
        printf("Inserisci il coefficiente b: ");
        scanf("%d", &b);
        printf("Inserisci il coefficiente c: ");
        scanf("%d", &c);

        risultato = numSoluzioniReali(a, b, c);

        printf("L'equazione %+dx^2 %+dx %+d ha %d soluzioni reali\n", a, b, c,
risultato);
        return 0;
}
```

Esercizio 1.s [chiamato-ASM]

```
.data
.text
# parametri in input: a0 = a, a1 = b, a2 = c
# valore resistuito in a0: numero di soluzioni reali

.globl numSoluzioniReali

numSoluzioniReali:
         mul t0, a1, a1              # t0 = b^2
         mul t1, a0, a2              # t1 = a * c
         slli t1, t1, 2              # t1 = 4 * a * c
         sub t0, t0, t1              # t0 = discriminante
         beq t0, zero, sol_coinc
         slt t1, t0, zero
         bne t1, zero, no_sol
         li a0, 2
         jr ra
sol_coinc: li a0, 1
         jr ra
no_sol: li a0, 0
         jr ra
```

Esercizio 2

- Si scriva una procedura che riceve in input*:*
  - una matrice quadrata di word memorizzata per righe
  - la dimensione (numero di righe) delle matrice
- La procedura restituisce*:*
  - 2 se la matrice è diagonale
  - 1 se la matrice è simmetrica
  - 0 se la matrice non è simmetrica.
- Testare la procedura richiamandola*:*
  - da codice RISC-V (nello stesso file .s)
  - da codice C (in un file .c)

Esercizio 2

- Si ricorda che in una matrice diagonale solamente i valori della diagonale principale possono essere diversi da 0, mentre una matrice simmetrica ha la proprietà di essere la trasposta di se stessa

- Esempio di matrice diagonale:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

- Esempio di matrice simmetrica:

$$\begin{bmatrix} 1 & 4 & 5 & 6 & 7 \\ 4 & 2 & 8 & 6 & 4 \\ 5 & 8 & 3 & 2 & 9 \\ 6 & 6 & 2 & 4 & 4 \\ 7 & 4 & 9 & 4 & 5 \end{bmatrix}$$

Esercizio 2.c [chiamante-C]

```
#include <stdio.h>
extern int diagonale_o_simmetrica(int *matrice, int dim);
int main() {
        int matrice[] = {
                1, 0, 0, 0, 0,
                0, 2, 0, 1, 0,
                0, 0, 3, 0, 0,
                0, 1, 0, 4, 0,
                0, 0, 0, 0, 5
        };
        int risultato;
        risultato = diagonale_o_simmetrica(matrice, 5);
        switch (risultato) {
                case 0:  printf("La matrice non e' simmetrica");
                         break;
                case 1:  printf("La matrice e' simmetrica");
                         break;
                case 2:  printf("La matrice e' diagonale");
                         break; }
        return 0;
        }
```

Esercizio 2.s [chiamato-ASM]

```
.data
matrix: .word  1, 0, 0, 0, 0
        .word  0, 2, 0, 1, 0
        .word  0, 0, 3, 0, 0
        .word  0, 1, 0, 4, 0
        .word  0, 0, 0, 0, 5

         .text
main:    la a0, matrix
         li a1, 5
         jal diagonale_o_simmetrica

         # visualizza il risultato
         li a7, 1
         ecall

         li a7, 10
         ecall
```

Esercizio 2.s [chiamato-ASM]

```
.globl diagonale_o_simmetrica

diagonale_o_simmetrica:
         # salvo s0 nello stack per preservarlo
         addi sp, sp, -4
         sw s0, 0(sp)
         li t0, 2           # t0 conterrà il risultato (ipotesi iniziale: diagonale)
         slli t1, a1, 2     # t1: offset per passare alla riga successiva della matrice
         addi t2, t1, 4     # t2: offset tra gli elementi lungo la diagonale
         addi a1, a1, -1    # a1: contatore ciclo esterno
ciclo1:  mv t3, a1          # t2: contatore ciclo interno
         mv t4, a0          # t3: puntatore a elementi su riga
         mv t5, a0          # t4: puntatore a elementi su colonna
ciclo2: addi t4, t4, 4
         add t5, t5, t1
         lw t6, 0(t4)
         beq t6, zero, next
        li t0, 1            # non è diagonale
next:    lw s0, 0(t5)
         bne t6, s0, no_simm         # se non è simmetrica (né diagonale), esco dal ciclo
         addi t3, t3, -1
         bne t3, zero, ciclo2
```

Esercizio 2.s [chiamato-ASM]

```
          add a0, a0, t2
          addi a1, a1, -1
          bne a1, zero, ciclo1
          j fine

no_simm: li t0, 0
fine:     mv a0, t0
          lw s0, 0(sp)                # ripristino s0
          addi sp, sp, 4
          jr ra
```