

# Architettura degli Elaboratori

## Corso A - turni di laboratorio T1 e T2

**Maurizio Lucenteforte**

**Ricevimento:** inviare una mail a **maurizio.lucenteforte@unito.it**

**Quale turno** di laboratorio seguire:

- procurarsi il **numero di matricola**
- leggere la **cifra meno significativa**:
  - se la cifra letta è **dispari** allora si appartiene al **turno T1**
  - se la cifra letta è **pari** allora si appartiene al **turno T2**

**Quando:**

- Turno T1: lunedì, ore 14:00-16:00, Lab. Turing
- Turno T2: giovedì, ore 09:00-11:00, Lab. Turing

# Lab 1

Le istruzioni RISC-V e il simulatore RARS

# Obiettivi

- Scrivere i nostri primi programmi in linguaggio RISC-V
- Conoscere il simulatore RISC-V RARS
- Simulare l'esecuzione dei programmi RISC-V usando RARS

# Relazione gerarchica tra hardware e software



- Relazione gerarchica tra **hardware** e **software** rappresentati come cerchi concentrici.
- L'hardware è rappresentato dal cerchio più interno e il software applicativo da quello più esterno.
- In applicazioni complesse ci sono spesso **più livelli di applicazioni software**.
- Per esempio, un **server Web (Apache)** che usa un **database (mySQL)**. Entrambi sono eseguiti al di sopra del **software di sistema (e.g., Linux)**

# Compilatore & Assemblatore

- Il **programma** (ad esempio in C) è compilato in **linguaggio assembly**
- In questo corso usiamo **RISC-V**
- Il programma in RISC-V è **quindi tradotto (assemblato) in linguaggio macchina**.
- Inizialmente, useremo il **simulatore RARS** (alternativa: Ripes)
- Alla fine del corso potremmo usare anche **hardware emulato**, SO (Linux) e un compilatore (gcc)

Programma  
in linguaggio  
ad alto livello  
(in C)

```
scambia(size_t v[], size_t k)
{
    size_t temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

↓

Compilatore

↓

Programma  
in linguaggio  
assembler  
(per RISC-V)

```
scambia:
    slli x6, x11, 3
    add x6, x10, x6
    ld x5, 0(x6)
    ld x7, 8(x6)
    sd x7, 0(x6)
    sd x5, 8(x6)
    jalr x0, 0(x1)
```

↓

Assemblatore

↓

Programma  
in linguaggio  
macchina binario  
(per RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
0000000000000001000000001100111
```

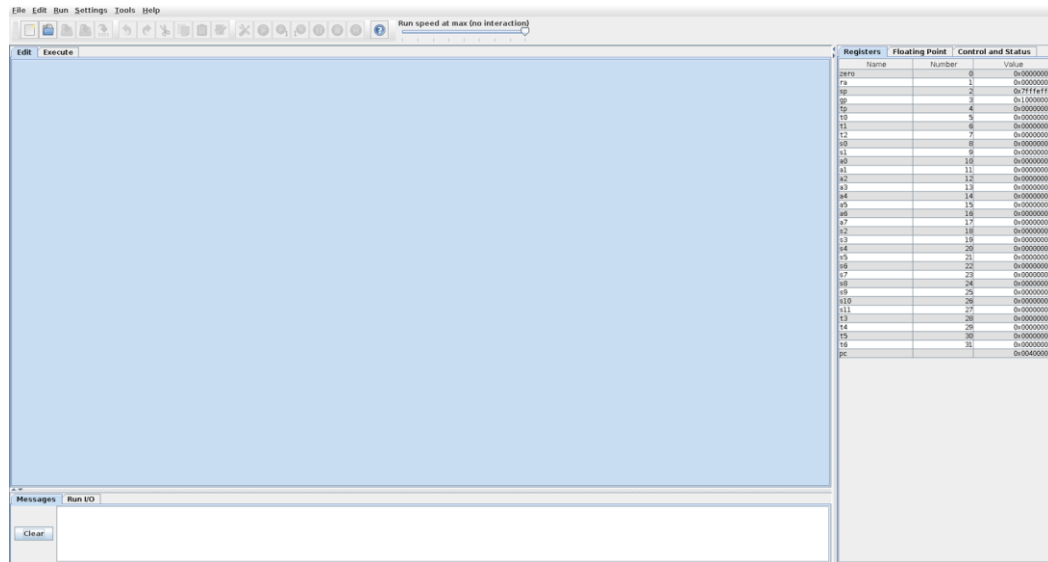
# Simulatore RARS

<https://github.com/TheThirdOne/rars/wiki>

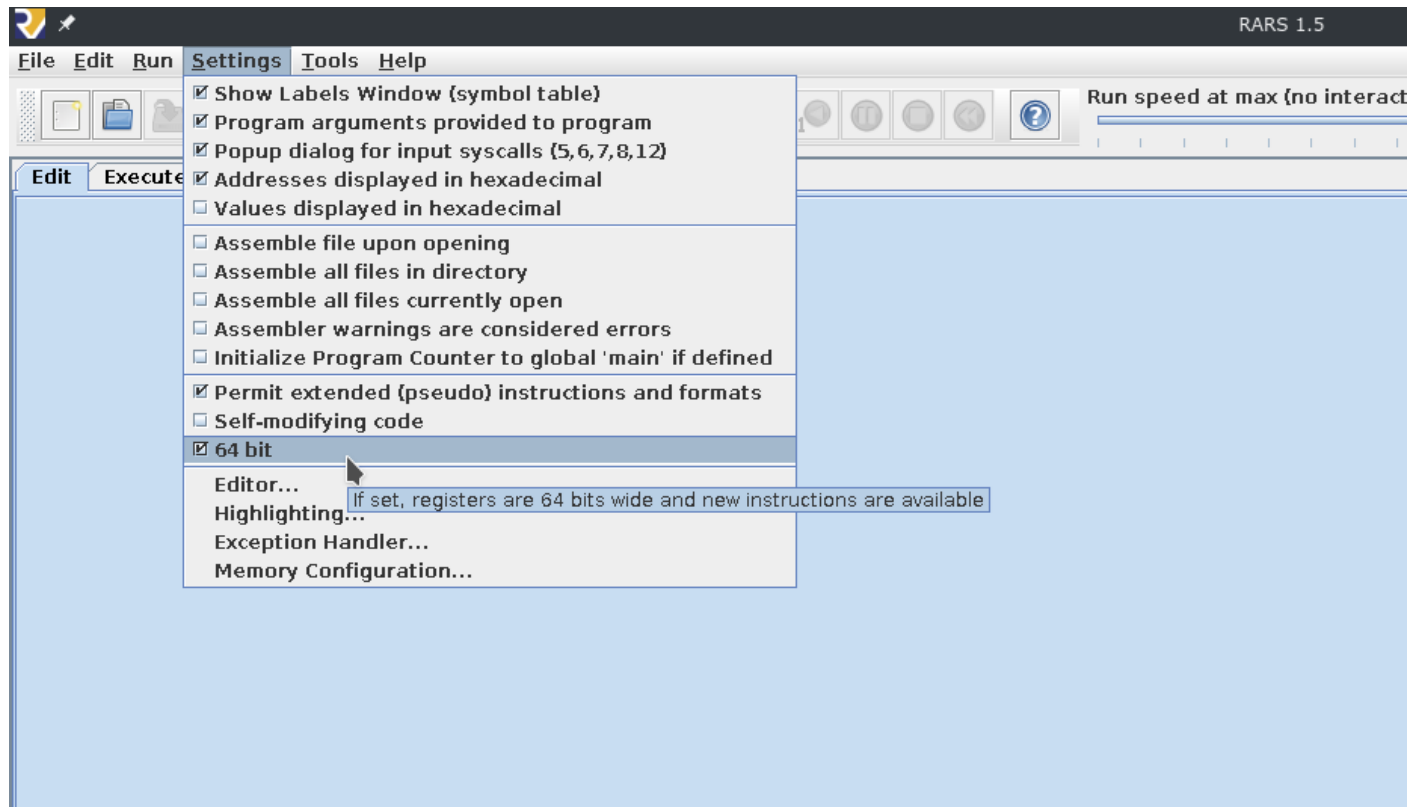
- Download: <https://github.com/TheThirdOne/rars>
- RISC-V Assembler, Runtime and Simulator
- Download JAR
- Necessario JVM 8 per eseguirlo

## Funzionalità principali

- RISC-V (**riscv32** and **riscv64**)
- Supporta **debugging** e **breakpoints**
- **pseudo-instruction** vs **instructions** vs **machine code**
- Statistiche della esecuzione dei programmi, e.g.,
  - a. numero di istruzioni eseguite
  - b. accessi alla memoria



# Cambiare impostazione a 64 bit



File Edit Run Settings Tools Help



Run speed at max (no interaction)

Controlli

Edit Execute

0 template.asm

```
1 .globl _start
2
3 .text
4
5 _start:
6
7     #<assembler code here>
8
9
10 exit:
11     addi x17, x0, 10 # call number 10 = exit
12     ecall
13
```

Editor

Line: 13 Column: 1 ☒ Show Line Numbers

Messages

Run I/O

Clear

I/O

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194304

Registri



# Registri

ABI Name  
(ABI = Application Binary Interface)

"Proper names" (e.g., x1, ... x31)

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

Registers			Floating Point	Control and Status
Name	Number	Value		
zero	0	0		
ra	1	0		
sp	2	2147479548		
gp	3	268468224		
tp	4	0		
t0	5	0		
t1	6	0		
t2	7	0		
s0	8	0		
s1	9	0		
s2	10	0		
a1	11	0		
a2	12	0		
a3	13	0		
a4	14	0		
a5	15	0		
a6	16	0		
a7	17	0		
s2	18	0		
s3	19	0		
s4	20	0		
s5	21	0		
s6	22	0		
s7	23	0		
s8	24	0		
s9	25	0		
s10	26	0		
s11	27	0		
t3	28	0		
t4	29	0		
t5	30	0		
t6	31	0		
pc		4194304		

# Allocazione della memoria per programmi e dati nel RISC-V

- Sarà visto nel dettaglio durante la teoria
- Gli **indirizzi** indicati sono frutto solo di convenzioni software
- Lo **stack pointer** decresce verso il basso verso il segmento dati
- Il **segmento di testo** contiene il **codice del programma**
- I **dati statici e dinamici** sono "sopra" (ad indirizzi maggiori) il segmento testo

SP → 0000 003f ffff fff0<sub>esa</sub>

0000 0000 1000 0000<sub>esa</sub>

PC → 0000 0000 0040 0000<sub>esa</sub>

0



File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Program Arguments:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00a00893	addi x17,x0,10	11: addi x17, x0, 10 # call numbe...
	0x00400004	0x00000073	ecall	12: ecall

Segmento Testo  
(codice tradotto)

Labels

Label	Address
(global)	
start	0x00400000
0_template.asm	
exit	0x00400000

Etichette

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0

Segmento Dati

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	0
s1	9	0
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194304

Registri

Messages Run I/O

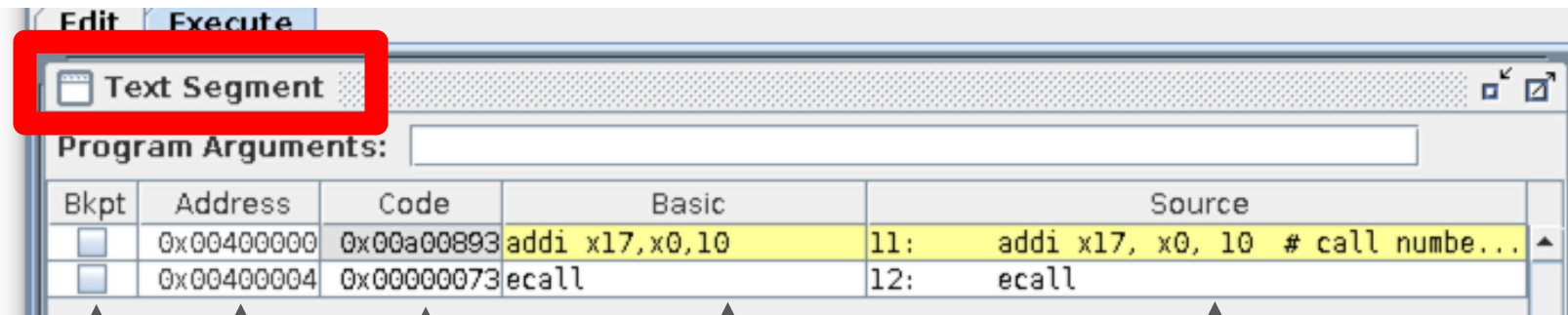
Assemble: assembling /home/idrago/academic/teaching/2022/arch\_i/labs/lab2/0\_template.asm

Assemble: operation completed successfully.

Clear

I/O

# Segmento Testo



breakpoint

indirizzi delle  
istruzioni  
(e.g., per il PC)

codifica dell'istruzione  
(linguaggio di macchina)

RISC-V "basico",  
tradotto durante assemblaggio

sorgente, include per esempio le  
"pseudoistruzioni"

# Segmento Dati

Visualizzazione della memoria, non solo dei "dati statici"

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x10010000	0	0	0	0	0	0	
0x10010020	0	0	0	0	0	0	
0x10010040	0	0	0	0	0	0	
0x10010060	0	0	0	0	0	0	
0x10010080	0	0	0	0	0	0	
0x100100a0	0	0	0	0	0	0	
0x100100c0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0

Navigation: **0x10010000 (.data)** ☒ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Messages | Run I/O

Assembly: assembly.asm  
Assembly: operation.asm

Clear

Stack

↓

↑

Dati dinamici

Dati statici

Testo

Riservato

0x10000000 (.extern)  
0x10010000 (.data)  
0x10040000 (heap)  
current gp  
current sp  
0x00400000 (.text)  
0xffff0000 (MMIO)

## Text Segment

Program Arguments:

Bkp	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x00a00893	addi x17,x0,10	11: addi x17, x0, 10 # call numbe...
<input type="checkbox"/>	0x00400004	0x00000073	ecall	12: ecall

## Labels

Label	Address ▲
<b>(global)</b>	
start	0x00400000
<b>0_template.asm</b>	
exit	0x00400000

☒ Data ☒ Text

## Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x00a00893	0x00000073	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00400000 (.text)

Hexadecimal Address ☒ Hexadecimal Values ☐ ASCII

# RISC-V - Le basi

- Commenti
- Etichette (Labels)
- Direttive assembler (Assembler Directives)
- Istruzioni
  - Pseudo istruzioni (extended)
  - Operandi
  - Registri
  - Dati statici
- Environment Calls (ecall)

- Tutorial  
<https://github.com/TheThirdOne/rars/wiki>
- Direttive assembler  
<https://github.com/TheThirdOne/rars/wiki/Assembler-Directives>
- Istruzioni  
<https://github.com/TheThirdOne/rars/wiki/Supported-Instructions>
- Environment calls (viste dopo)  
<https://github.com/TheThirdOne/rars/wiki/Environment-Calls>

# RISC-V - Commenti

```
# this is a comment
.globl _start

    addi x1, x0, 1    # this is another comment
```

- "#" marca l'inizio dei commenti
- Valido sia all'inizio della riga di codice sia dopo un'istruzione



# RISC-V - Etichette

```
mylabel:
```

```
    # Instructions here are reached using the 'mylabel' label
```

```
mydata:
```

```
    # Data declared here can be referenced using the 'mydata' label
```

- Un'etichetta deve essere scritta come:

```
nome:
```

- Le **etichette** identificano offsets (scostamenti) nella memoria, che ci permettono di recuperare l'indirizzo al quale reperire **codici o dati**
- **Sono usate per le istruzioni di salto, nelle procedure, funzioni ecc.**
- Alcune etichette sono particolari:
  - **\_start:** marca l'indirizzo della prima istruzione del nostro programma (punto di ingresso usato da Linux, il nostro "main")

# RISC-V – Direttive assembler

Una componente importante dei programmi assembly sono le "**direttive**" che indicano come l'assemblatore interpreterà il codice. Le direttive **iniziano con un punto (.)** seguito da una parola chiave.



Direttiva	Funzione
.globl <label>	Dichiara un'etichetta come globale
.text	Inizio del programma, i.e., zona nella memoria dove ci sarà il codice assembly
.data	Zona nella memoria dove ci saranno i dati
.word <data>	Indica che il valore in "data" deve essere rappresentato con 32-bit (word)
.eqv <NAME> <VALUE>	Definisce una <b>costante</b> , ad esempio, <code>.eqv NUMBER 4</code>

# RISC-V - Le istruzioni

Tipo di istruzioni	Istruzioni	Esempio	Significato	Commenti
Aritmetiche	Somma	<code>add x5, x6, x7</code>	$x5 = x6 + x7$	Operandi in tre registri
	Sottrazione	<code>sub x5, x6, x7</code>	$x5 = x6 - x7$	Operandi in tre registri
	Somma immediata	<code>addi x5, x6, 20</code>	$x5 = x6 + 20$	Utilizzata per sommare delle costanti
Trasferimento dati	Lettura parola doppia	<code>ld x5, 40(x6)</code>	$x5 = \text{Memoria}[x6 + 40]$	Spostamento di una parola doppia da memoria a registro
	Memorizzazione parola doppia	<code>sd x5, 40(x6)</code>	$\text{Memoria}[x6 + 40] = x5$	Spostamento di una parola doppia da registro a memoria
	Lettura parola	<code>lw x5, 40(x6)</code>	$x5 = \text{Memoria}[x6 + 40]$	Spostamento di una parola da memoria a registro
	Lettura parola senza segno	<code>lwu x5, 40(x6)</code>	$x5 = \text{Memoria}[x6+40]$	Spostamento di una parola senza segno da memoria a registro
	Memorizzazione parola	<code>sw x5, 40(x6)</code>	$\text{Memoria}[x6+40] = x5$	Spostamento di una parola da registro a memoria
	Lettura mezza parola	<code>lh x5, 40(x6)</code>	$x5 = \text{Memoria}[x6+40]$	Spostamento di una mezza parola da memoria a registro
	Lettura mezza parola, senza segno	<code>lhu x5, 40(x6)</code>	$x5 = \text{Memoria}[x6+40]$	Spostamento di una mezza parola senza segno da memoria a registro
	Memorizzazione mezza parola	<code>sh x5, 40(x6)</code>	$\text{Memoria}[x6+40] = x5$	Spostamento di una mezza parola da registro a memoria
	Memorizzazione mezza parola, senza segno	<code>shw x5, 40(x6)</code>	$\text{Memoria}[x6+40] = x5$	Spostamento di una mezza parola da registro a memoria

# RISC-V - I registri

**Libro** →

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

**I nomi sono più informativi**

# Nostro primo programma RISC-V

```
# marking where execution starts (main)
.globl _start

.text
# assembly code comes in the '.text' section

# the label provided to .globl above is declared here
_start:

    # << your program here!! >>

# env call to terminate the program (explained later)
exit:
    addi x17, x0, 10 # call number 10 = exit
    ecall
```

<b>#</b>	→ commenti
<b>.globl</b>	→ simboli globali
<b>.text</b>	→ segmento testo
<b>.data</b>	→ segmento dati
<b>exit:</b>	→ etichetta
<b>_start:</b>	→ etichetta ("main")

# Lab 1 - Esercizio 1 - Somma numeri

- Scrivere un programma per caricare le costanti 41, 43, 47 nei registri **x5**, **x6** e **x7** e calcolare la loro somma. Il risultato va scritto nel registro **x28**.
- Eseguire il programma nel simulatore RARS. Eseguire un'istruzione alla volta e verificare i valori dei registri.

# Lab 1 - Esercizio 1 - Somma numeri

# sum 3 numbers loaded in x5, x6, x7 and put the result in x28

**.globl \_start**

**.text**

**\_start:**

**addi** x5, x0, 41       # load the numbers

**addi** x6, x0, 43

**addi** x7, x0, 47

**add** x28, x5, x6       # perform the sum

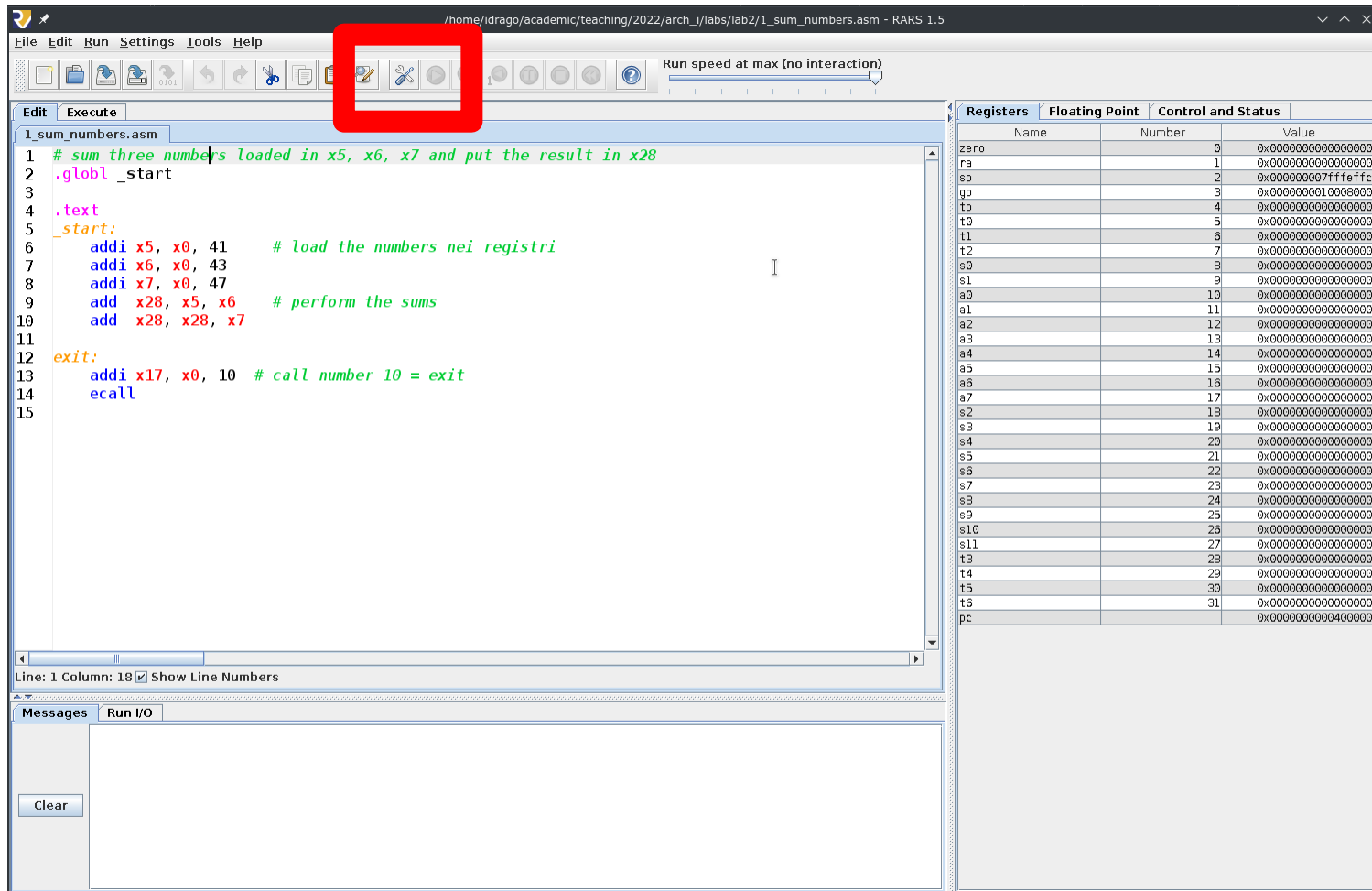
**add** x28, x28, x7

**exit:**

**addi** x17, x0, 10   # call number 10 = exit

**ecall**

# Debugging



The screenshot displays the RARS MIPS simulator interface. The main window shows an assembly file named `1_sum_numbers.asm` with the following code:

```
1 # sum three numbers loaded in x5, x6, x7 and put the result in x28
2 .globl _start
3
4 .text
5 _start:
6     addi x5, x0, 41    # load the numbers nei registri
7     addi x6, x0, 43
8     addi x7, x0, 47
9     add  x28, x5, x6    # perform the sums
10    add  x28, x28, x7
11
12 exit:
13    addi x17, x0, 10    # call number 10 = exit
14    ecall
15
```

A red box highlights the **Run** button in the toolbar. The right panel shows the **Registers** tab, displaying a list of registers and their values:

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007ffffeffc
gp	3	0x0000000010000000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000004000000

The bottom panel shows the **Messages** and **Run I/O** tabs. The **Messages** tab is currently selected, and the **Run I/O** tab is also visible. A **Clear** button is located below the **Messages** tab.



# Debugging

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Run one step at a time

**Text Segment**

Program Arguments:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x02900293	addi x5,x0,0x00000029	6: addi x5, x0, 41 # load the numbers nei...
<input type="checkbox"/>	0x00400004	0x02b00313	addi x6,x0,0x0000002b	7: addi x6, x0, 43
<input type="checkbox"/>	0x00400008	0x02f00393	addi x7,x0,0x0000002f	8: addi x7, x0, 47
<input type="checkbox"/>	0x0040000c	0x00620e33	add x28,x5,x6	9: add x28, x5, x6 # perform the sums
<input type="checkbox"/>	0x00400010	0x007e0e33	add x28,x28,x7	10: add x28, x28, x7
<input type="checkbox"/>	0x00400014	0x00a00893	addi x17,x0,10	13: addi x17, x0, 10 # call number 10 = exit
<input type="checkbox"/>	0x00400018	0x00000073	ecall	14: ecall

**Labels**

Label	Address
(global)	
start	0x00400000
1_sum_numbers.asm	
exit	0x00400014

☒ Data ☒ Text

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 {data} ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

**Registers** **Floating Point** **Control and Status**

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007ffffeffc
gp	3	0x0000000010000000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000000000

**Messages** **Run I/O**

Assembly: /home/ldrago/academic/teaching/2022/arch\_i/labs/lab2/1\_sum\_numbers.asm

Assembly: operation completed successfully.

Clear

# Debugging

Debugger window showing assembly code and registers.

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Execute

Text Segment

Program Arguments:

Address	Code	Basic	Source
0x00400000	0x02900293	addi x5,x0,0x00000029	6: addi x5, x0, 41 # load the numbers
0x00400004	0x02b00313	addi x6,x0,0x0000002b	7: addi x6, x0, 43
0x00400008	0x02f00393	addi x7,x0,0x0000002f	8: addi x7, x0, 47
0x0040000c	0x00620e33	add x28,x5,x6	9: add x28, x5, x6 # perform the sums
0x00400014	0x00a00893	addi x17,x0,10	13: addi x17, x0, 10 # call number 10 = exit
0x00400018	0x00000073	ecall	14: ecall

Labels

Label	Address
(global)	
start	0x00400000
1_sum_numbers.asm	
exit	0x00400014

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp		
gp	3	0x0000000010000000
tp	4	0x0000000000000000
t0	5	0x0000000000000029
t1	6	0x0000000000000000
t2		
s0		
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400004

Messages Run I/O

Clear

# Debugging

Debugger window showing assembly code and registers.

**Assembly Code (Text Segment):**

Address	Code	Basic	Source
0x00400000	0x02900293	addi x5,x0,0x00000029	6: addi x5, x0, 41 # load the numbers
0x00400004	0x02b00313	addi x6,x0,0x0000002b	7: addi x6, x0, 43
0x00400008	0x02f00393	addi x7,x0,0x0000002f	8: addi x7, x0, 47
0x0040000c	0x007e0e33	add x28,x28,x7	10: add x28, x28, x7
0x00400014	0x00a00893	addi x17,x0,10	13: addi x17, x0, 10 # call number 10 = exit
0x00400018	0x00000073	ecall	14: ecall

**Registers:**

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007ffffbfc
gp	3	0x0000000000000000
tp	4	0x0000000000000000
t0	5	0x0000000000000029
<b>t1</b>	<b>6</b>	<b>0x000000000000002b</b>
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
s2	18	0x0000000000000000
s3	19	0x0000000000000000
s4	20	0x0000000000000000
s5	21	0x0000000000000000
s6	22	0x0000000000000000
s7	23	0x0000000000000000
s8	24	0x0000000000000000
s9	25	0x0000000000000000
s10	26	0x0000000000000000
s11	27	0x0000000000000000
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
<b>t6</b>	<b>31</b>	<b>0x0000000000000000</b>
pc		0x0000000000000008

**Data Segment:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

**Run I/O:**

0x10010000 [.data] Hexadecimal Addresses Hexadecimal Values ASCII

# Lab 1 - Esercizio 2 - Overflow

li - Carica immediato  
R[rd] = costante

Supponiamo che i registri x5 e x6 contengano i numeri 0x8000000000000000 e 0xD000000000000000. Usare il simulatore RARS per rispondere alle domande.

- Quale sarà il contenuto di x30 dopo l'esecuzione di questa istruzione

`add x30, x5, x6`

Il contenuto di x30 è corretto, o si è verificato un overflow?

- Quale sarà il contenuto di x30 dopo l'esecuzione di questa istruzione:

`sub x30, x5, x6`

Il contenuto di x30 è corretto o si è verificato un overflow?

- Quale sarà il contenuto di x30 dopo l'esecuzione di queste due istruzioni:

`add x30, x5, x6`

`add x30, x30, x5`

Il contenuto di x30 è corretto o si è verificato un overflow?

# Lab 1 - Esercizio 2 - Overflow

li - Carica immediato  
R[rd] = costante

Supponiamo che i registri x5 e x6 contengano i numeri 0x8000000000000000 e 0xD000000000000000. Usare il simulatore RARS per rispondere alle domande.

- Quale sarà il contenuto di x30 dopo l'esecuzione di questa istruzione:

```
add x30, x5, x6
```

**x30 = 0x5000000000000000 - overflow**

- Quale sarà il contenuto di x30 dopo l'esecuzione di questa istruzione:

```
sub x30, x5, x6
```

**x30 = 0xB000000000000000 - senza overflow**

- Quale sarà il contenuto di x30 dopo l'esecuzione di queste due istruzioni:

```
add x30, x5, x6
```

```
add x30, x30, x5
```

**x30 = 0xD000000000000000 - overflow**

# Attenzione: Istruzione vs Pseudoistruzione

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x800002b7	lui x5,0xffff80000	6: li x5, 0x8000000000000000 # check how this is translated!
<input type="checkbox"/>	0x00400004	0x0002829b	addiw x5,x5,0	
<input type="checkbox"/>	0x00400008	0x00b29293	slli x5,x5,11	
<input type="checkbox"/>	0x0040000c	0x00028293	addi x5,x5,0	
<input type="checkbox"/>	0x00400010	0x00b29293	slli x5,x5,11	
<input type="checkbox"/>	0x00400014	0x00028293	addi x5,x5,0	
<input type="checkbox"/>	0x00400018	0x00a29293	slli x5,x5,10	
<input type="checkbox"/>	0x0040001c	0x00028293	addi x5,x5,0	

- li - "Load immediate", carica una double word, quindi 64 bit
- Però le istruzioni "immediate" (e.g., addi) hanno un parametro da 12 bit

# Lab 1 - Esercizio 2 - Overflow

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x0000000000000000	
ra	1	0x0000000000000000	
sp	2	0x000000007ffefffc	
gp	3	0x0000000010008000	
tp	4	0x0000000000000000	
t0	5	0x8000000000000000	
t1	6	0xd000000000000000	
t2	7	0x0000000000000000	
s0	8	0x0000000000000000	
s1	9	0x0000000000000000	
a0	10	0x0000000000000000	
a1	11	0x0000000000000000	
a2	12	0x0000000000000000	
a3	13	0x0000000000000000	
a4	14	0x0000000000000000	
a5	15	0x0000000000000000	
a6	16	0x0000000000000000	
a7	17	0x0000000000000000	
s2	18	0x0000000000000000	
s3	19	0x0000000000000000	
s4	20	0x0000000000000000	
s5	21	0x0000000000000000	
s6	22	0x0000000000000000	
s7	23	0x0000000000000000	
s8	24	0x0000000000000000	
s9	25	0x0000000000000000	
s10	26	0x0000000000000000	
s11	27	0x0000000000000000	
t3	28	0x0000000000000000	
t4	29	0x0000000000000000	
t5	30	0x5000000000000000	
t6	31	0x0000000000000000	
pc		0x000000000400044	

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0		C
ra	1		C
sp	2	214747954E	
gp	3	268468224	
tp	4		C
t0	5	-922337203685477580E	
t1	6	-345876451382054002E	
t2	7		C
s0	8		C
s1	9		C
a0	10		C
a1	11		C
a2	12		C
a3	13		C
a4	14		C
a5	15		C
a6	16		C
a7	17		C
s2	18		C
s3	19		C
s4	20		C
s5	21		C
s6	22		C
s7	23		C
s8	24		C
s9	25		C
s10	26		C
s11	27		C
t3	28		C
t4	29		C
t5	30	576460752303423488C	
t6	31		C
pc		4194372	