UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Introduzione a QtSpim

Prof.ssa Giulia Cisotto

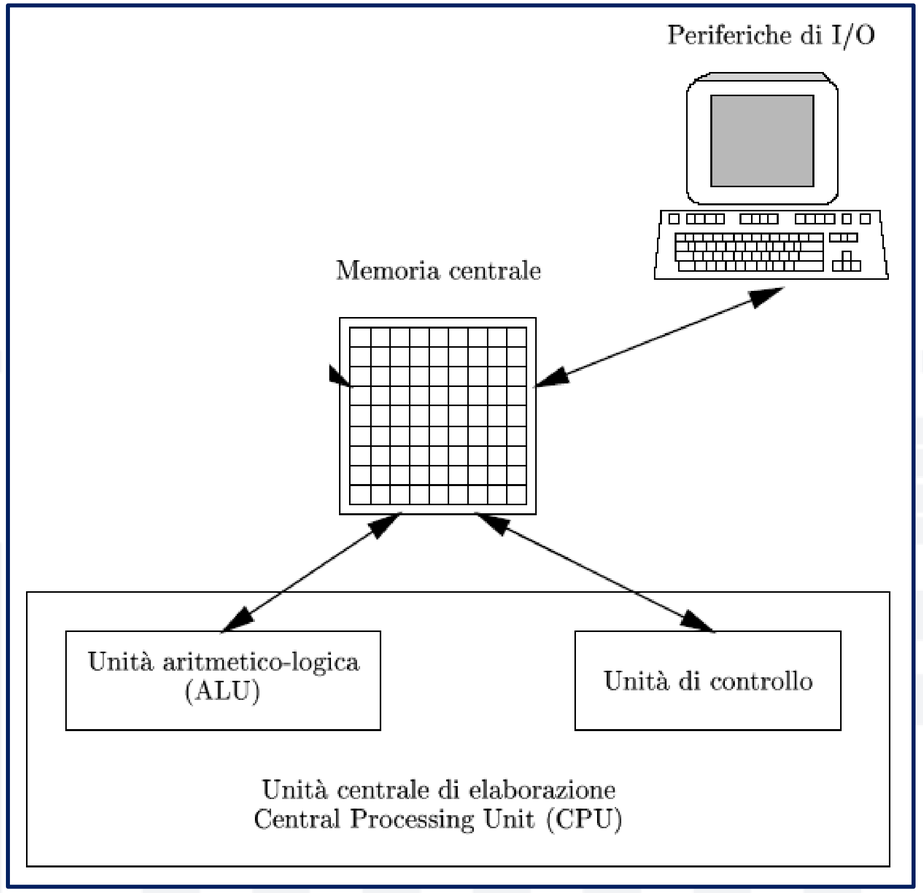
giulia.cisotto@units.it

Trieste, 3 marzo 2025

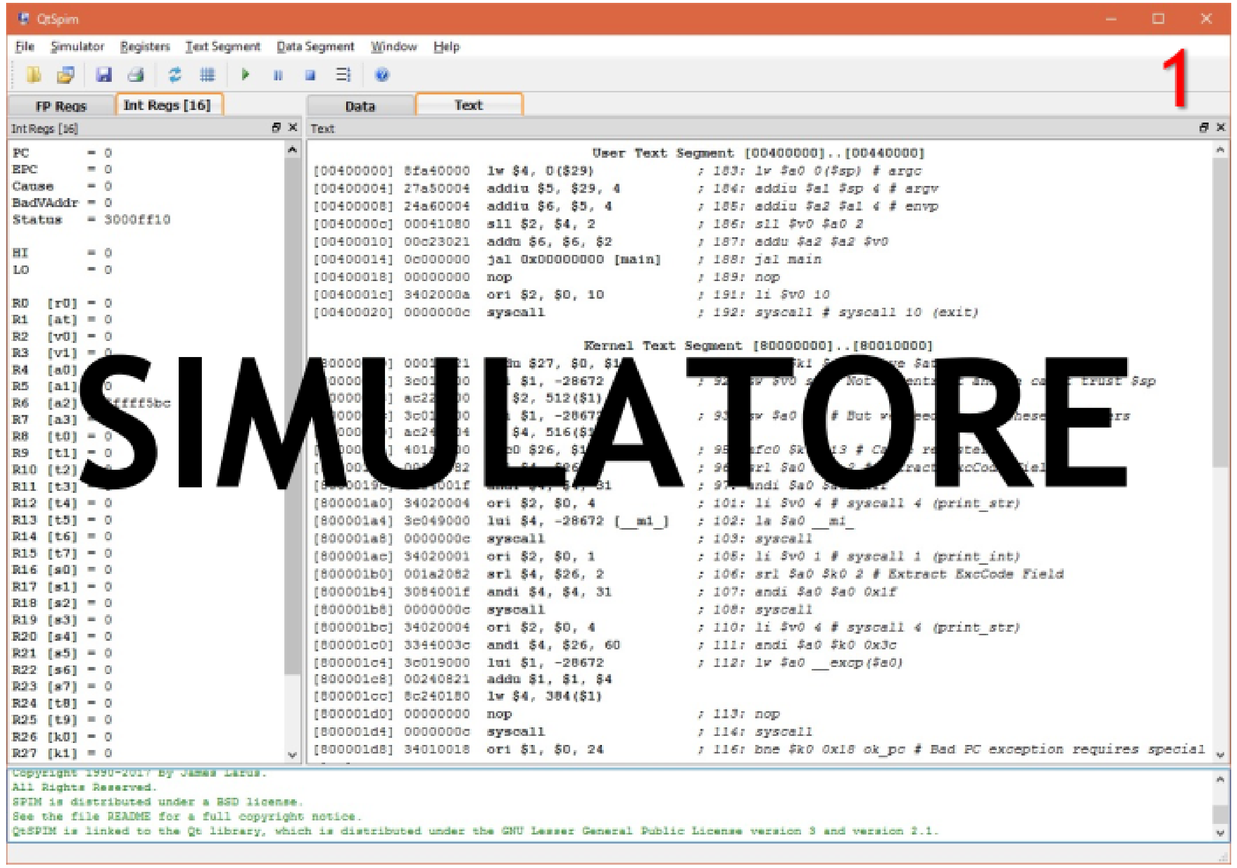
QtSpim

https://spimsimulator.sourceforge.net/

COSA SIMULA?



L'interfaccia del simulatore QtSpim è composta da due parti principali



REGISTRI e MEMORIA

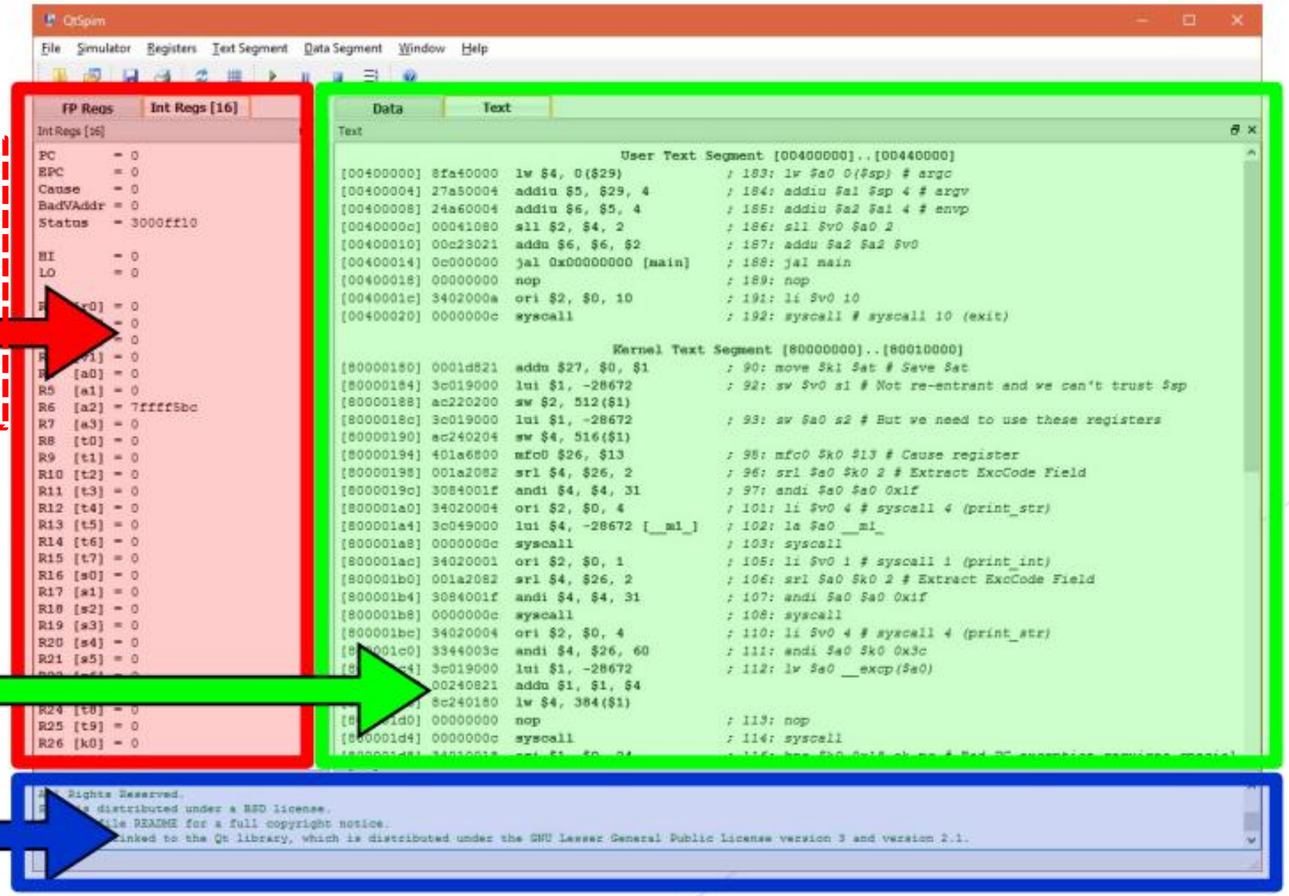
Abbiamo a disposizione **32 registri INTERI e 32 registri FLOATING POINT** (codifica IEEE 754)

LA FINESTRA PRINCIPALE È SUDDIVISA IN TRE PARTI

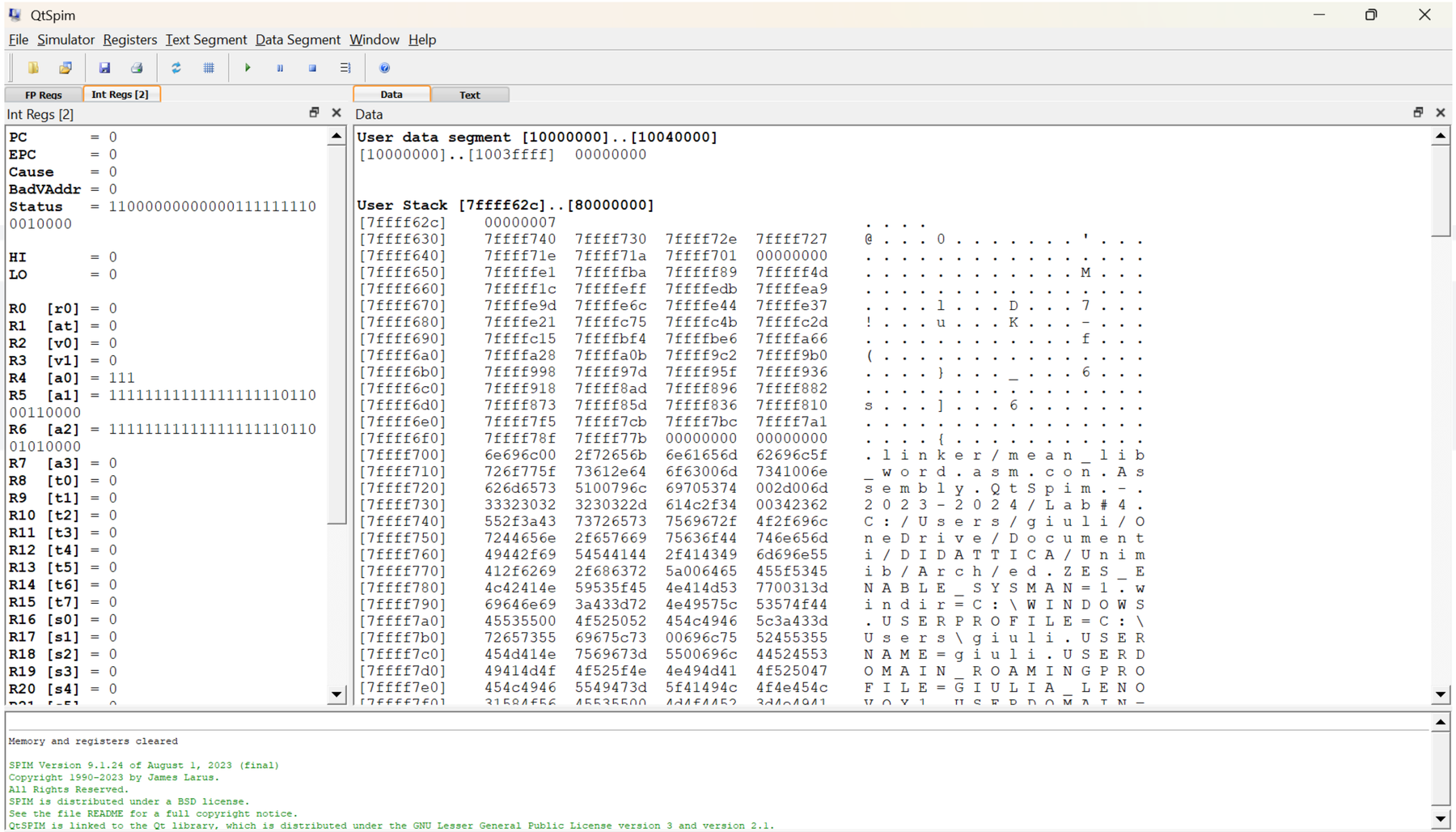
IL PANNELLO SULLA **SINISTRA** VISUALIZZA I REGISTRI FLOATING POINT oppure I REGISTRI INTERI, A SECONDA DELLA SCHEDA IN PRIMO PIANO

IL PANNELLO SULLA **DESTRA** VISUALIZZA IL TEXT SEGMENT (CONTIENE LE ISTRUZIONI MACCHINA) ED IL DATA SEGMENT (CONTIENE I DATI IN MEMORIA), A SECONDA DELLA SCHEDA IN PRIMO PIANO

IL PANNELLO INFERIORE CONTIENE I MESSAGGI DEL SIMULATORE



ESEMPIO DI INTERFACCIA

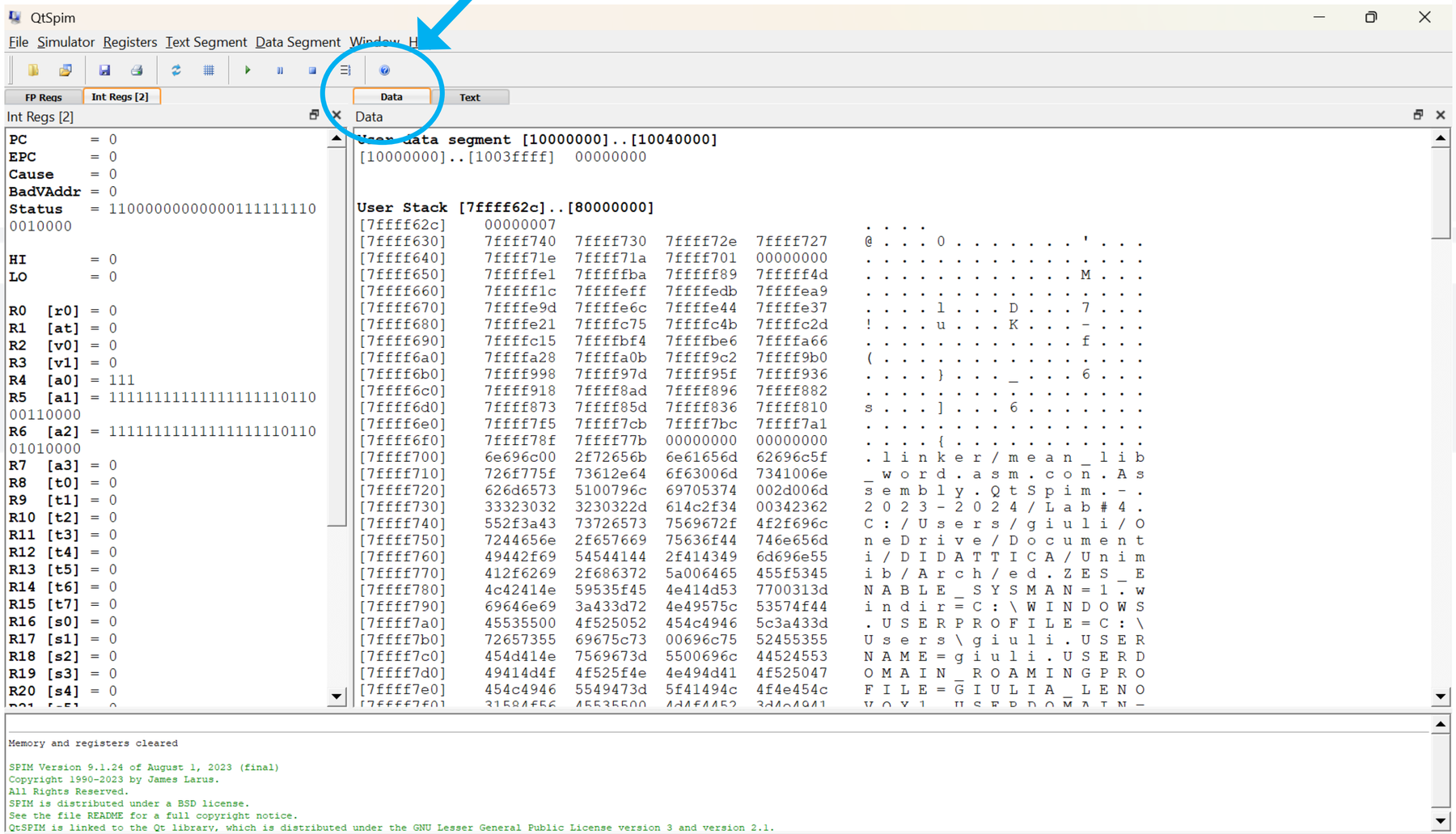


UNIVERSITÀ DEGLI STUDI DI TRIESTE

QtSpim

In questa sezione della memoria verranno memorizzate **i dati dell'utente** (che si carica, all'avvio di QtSpim, per poterlo eseguire)

QtSpim



UNIVERSITÀ DEGLI STUDI DI TRIESTE

QtSpim

In questa sezione della memoria verranno memorizzate le **istruzioni associate al programma dell'utente** (che si carica, all'avvio di QtSpim, per poterlo eseguire)

QtSpim

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs | Int Regs [2] | Data | Text

Int Regs [2]

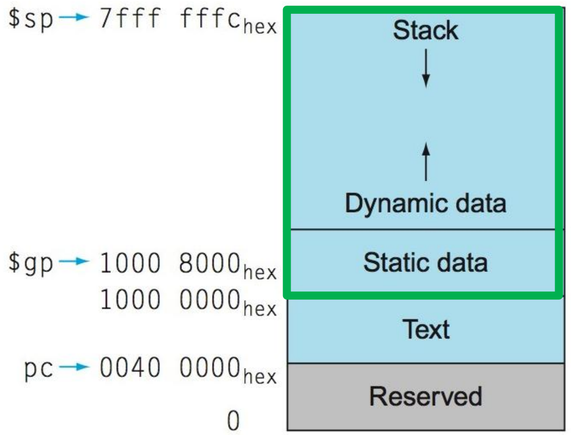
```
PC          = 0
EPC         = 0
Cause       = 0
BadVAddr    = 0
Status      = 11000000000000111111110
0010000

HI          = 0
LO          = 0

R0   [r0]   = 0
R1   [at]   = 0
R2   [v0]   = 0
R3   [v1]   = 0
R4   [a0]   = 111
R5   [a1]   = 11111111111111111110110
00110000
R6   [a2]   = 11111111111111111110110
01010000
R7   [a3]   = 0
R8   [t0]   = 0
R9   [t1]   = 0
R10  [t2]   = 0
R11  [t3]   = 0
R12  [t4]   = 0
R13  [t5]   = 0
R14  [t6]   = 0
R15  [t7]   = 0
R16  [s0]   = 0
R17  [s1]   = 0
R18  [s2]   = 0
R19  [s3]   = 0
R20  [s4]   = 0
```

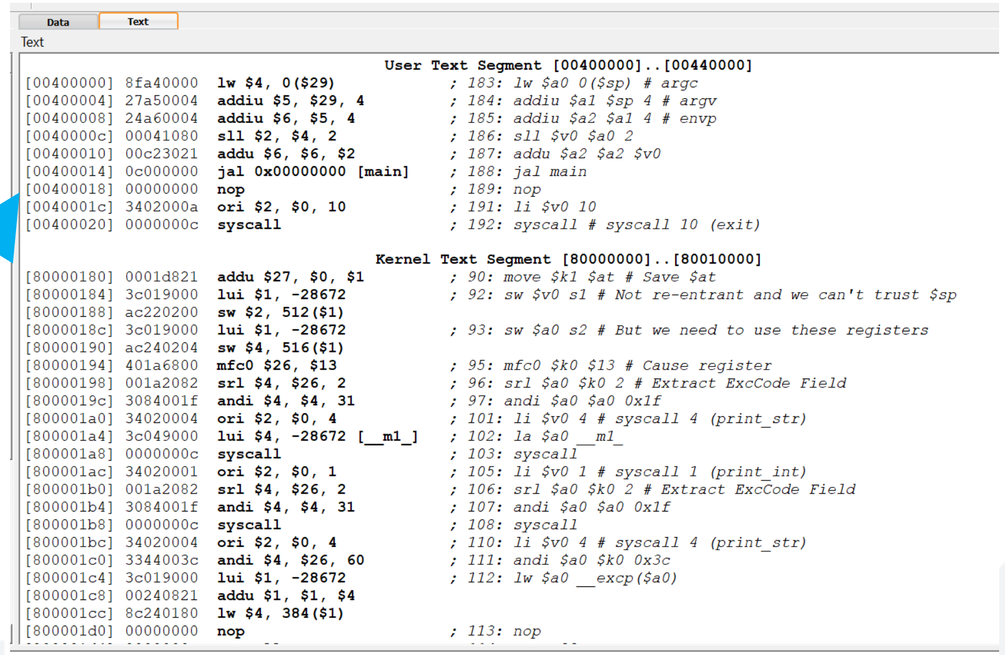
Text

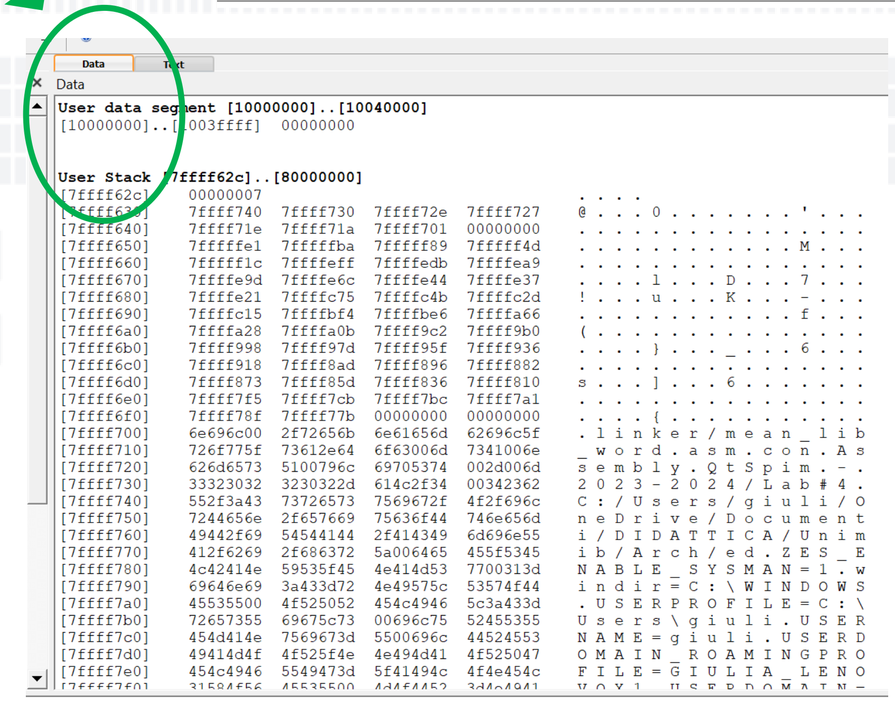
```
                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]   ; 188: jal main
[00400018] 00000000  nop                     ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10          ; 191: li $v0 10
[00400020] 0000000c  syscall                 ; 192: syscall # syscall 10 (exit)

                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1        ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672          ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672          ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13           ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2          ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31         ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4           ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]  ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                 ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1           ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2          ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31         ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                 ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4           ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60        ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672          ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
[800001cc] 8c240180  lw $4, 384($1)
[800001d0] 00000000  nop                     ; 113: nop
```

```
Memory and registers cleared

SPIM Version 9.1.24 of August 1, 2023 (final)
Copyright 1990-2023 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

QtSpim





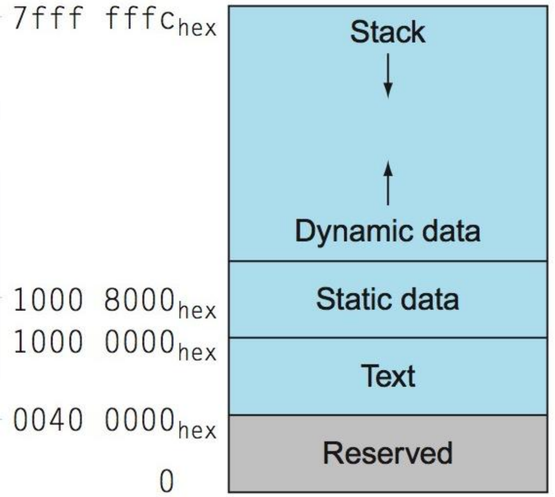


In questa porzione di memoria verranno memorizzate i **dati dell'utente** (caricati in memoria centrale), **le variabili temporanee** che servono durante l'esecuzione di **procedure e altro (si faccia riferimento alle prossime lezioni)**.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

STACK POINTER ($sp)

Lo **Stack Pointer** è un registro (in MIPS è $sp) che indica l'**indirizzo della cima dello stack in memoria**.



Lo stack è un'area di memoria utilizzata per memorizzare temporaneamente dati (ad esempio variabili locali, indirizzi di ritorno di funzioni, parametri di funzioni, ecc.).

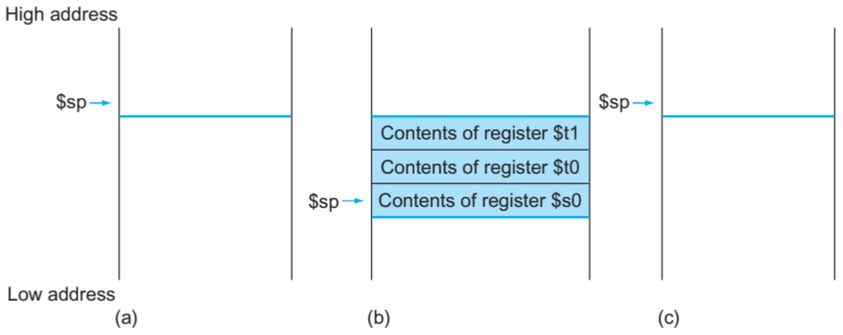
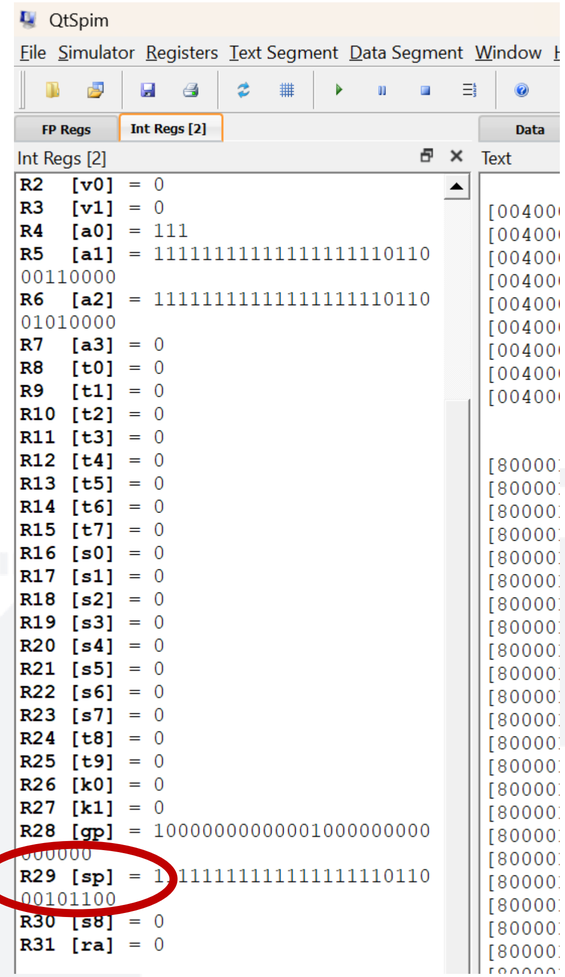


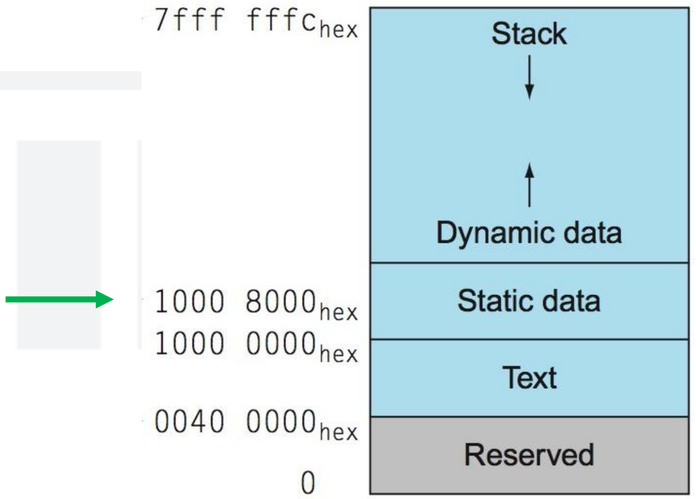
FIGURE 2.10 The values of the stack pointer and the stack (a) before, (b) during, and (c) after the procedure call. The stack pointer always points to the "top" of the stack, or the last word in the stack in this drawing.



Cresce verso indirizzi più bassi: ogni volta si aggiunge qualcosa allo stack, $sp viene decrementato.

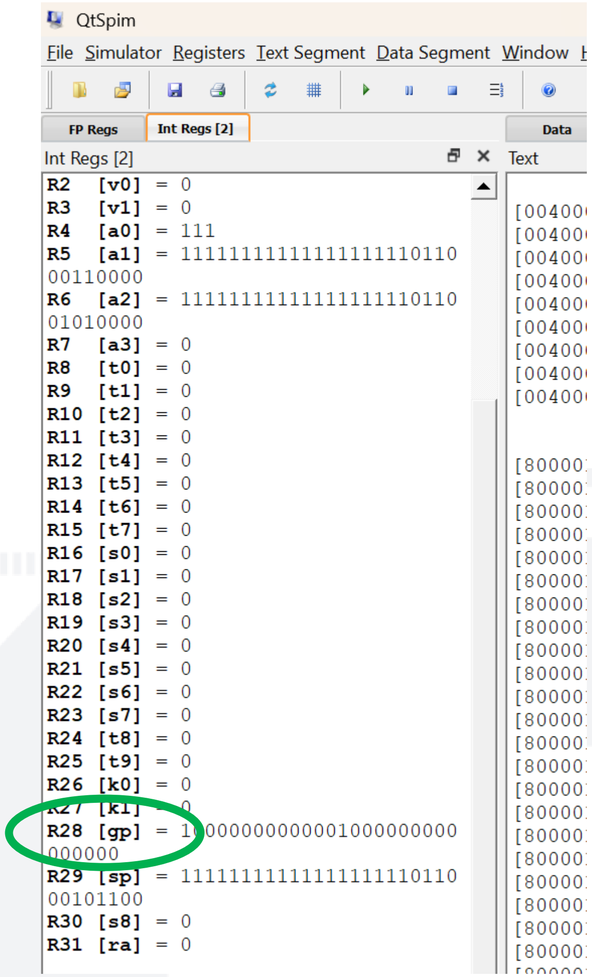
GLOBAL POINTER ($gp)

È un registro speciale in MIPS32 che contiene un **indirizzo di riferimento** per accedere rapidamente alle variabili globali in memoria.



È utile **per accedere a dati statici/globali con indirizzamento breve e veloce**.

Solitamente è inizializzato a un indirizzo centrale nel segmento dati, permettendo di accedere facilmente (con un offset piccolo) alle variabili globali più comuni.

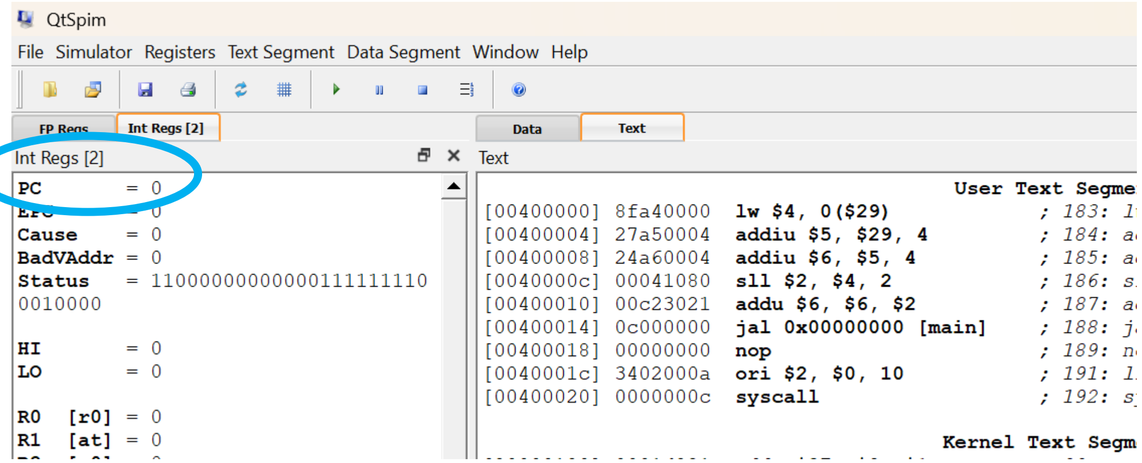


Accesso alle variabili globali più efficiente rispetto all'indirizzamento assoluto.

PROGRAM COUNTER (PC)

Il **Program Counter (PC)** in MIPS32 è un registro speciale della CPU che ha il compito di tenere traccia dell'**indirizzo della prossima istruzione** da eseguire.

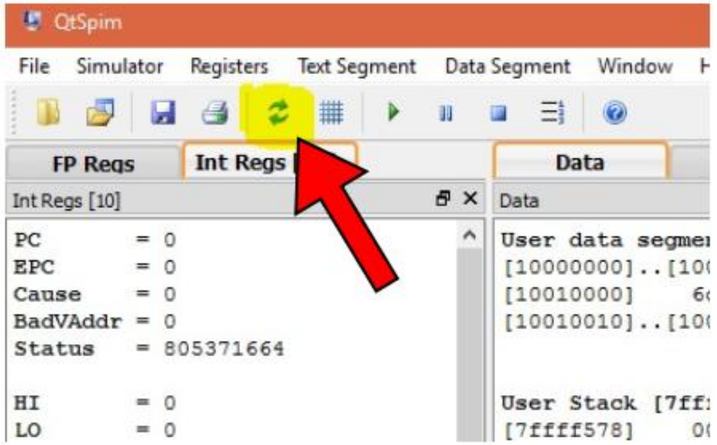
- È un registro a **32 bit**.
- L'indirizzo in esso contenuto è sempre **multiplo di 4**, dato che ogni istruzione MIPS32 occupa **4 byte**.
- Dopo che un'istruzione viene letta (*fetch*), il PC viene automaticamente incrementato di **4**, passando all'istruzione successiva.



Istruzioni sequenziali (la maggior parte): PC = PC + 4

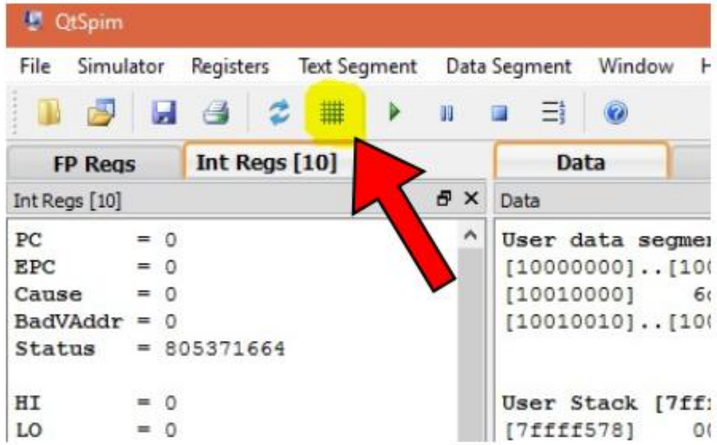
Istruzioni speciali (branch, jump, chiamate funzioni) modificano il PC diversamente.

QtSpim



PULSANTE
«CLEAR REGISTER»

CANCELLA IL CONTENUTO DEI REGISTRI, MA NON LA MEMORIA (AREA DATI)



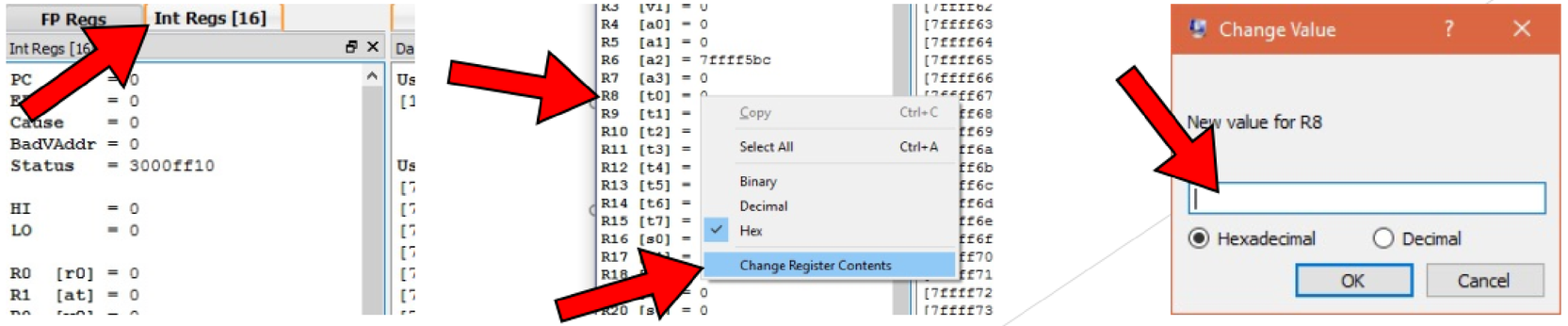
PULSANTE
«REINITIALIZE SIMULATOR»

REIMPOSTA LA MEMORIA (AREA DATI), E I REGISTRI

QtSpim

COME CAMBIARE IL VALORE DI UN REGISTRO (e.g. R8)?

- SELEZIONARE LA SCHEDA DEI REGISTRI DEGLI INTERI «INT REGS [16]» ED INDIVIDUARE IL REGISTRO R8 (Register8)
- FARE CLICK CON IL PULSANTE DESTRO IN CORRISPONDENZA DEL REGISTRO R8
- SELEZIONARE «CHANGE REGISTER CONTENTS»
- INSERIRE IL VALORE (IN FORMATO HEX oppure DEC)



QtSpim

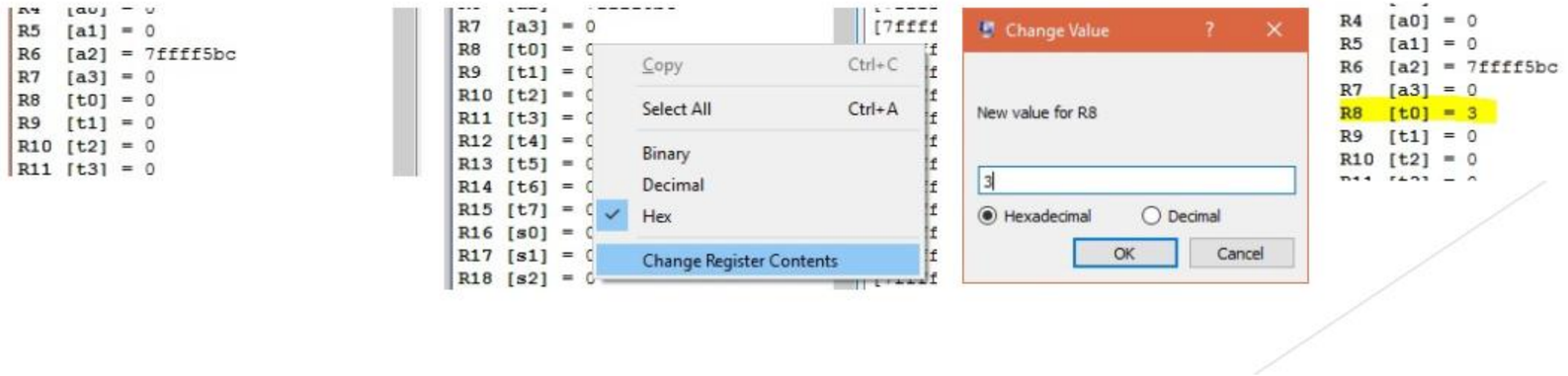
ESEMPIO: IMPOSTARE R8 CON IL VALORE 0x3

1
INDIVIDUO R8

2
CLICK DESTRO +
«CHANGE REGISTER...»

3
SCRIVO IL NUMERO

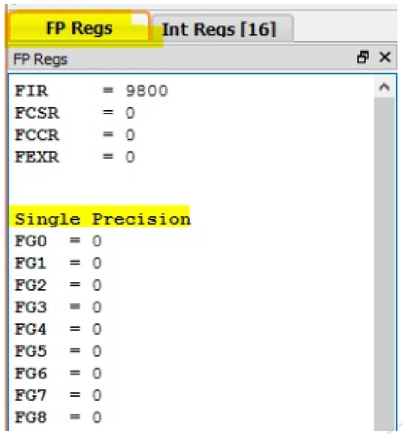
4
RISULTATO



QtSpim

REGISTRI FLOATING POINT SINGLE PRECISION (VIRGOLA MOBILE, SINGOLA PRECISIONE)

- SI VISUALIZZANO ABILITANDO LA SCHEDA «FP Regs»
- SI MODIFICANO ALLO STESSO MODO DEI REGISTRI INTERI
- LA RAPPRESENTAZIONE DEI NUMERI SEGUE LO **STANDARD IEEE754**



N.B. il contenuto del registro è visualizzato a partire dalla prima cifra diversa da 0