INDIAN INSTITUTE OF TECHNOLOGY GOA

COMPUTER ARCHITECTURE LAB (CS 211)

Faculty: Dr. Sharad Sinha
Teaching Assistant: Prachi Kashikar
1st Contributor: Pavitra Bhade

LAB 01

In this course, you will write assembly language programs and simulate them using the MIPS32 simulator QtSpim. QtSpim is a self-contained simulator that runs MIPS32 assembly programs. It reads and executes assembly language programs written for this processor.
We will be splitting this lab exercise in to two parts:

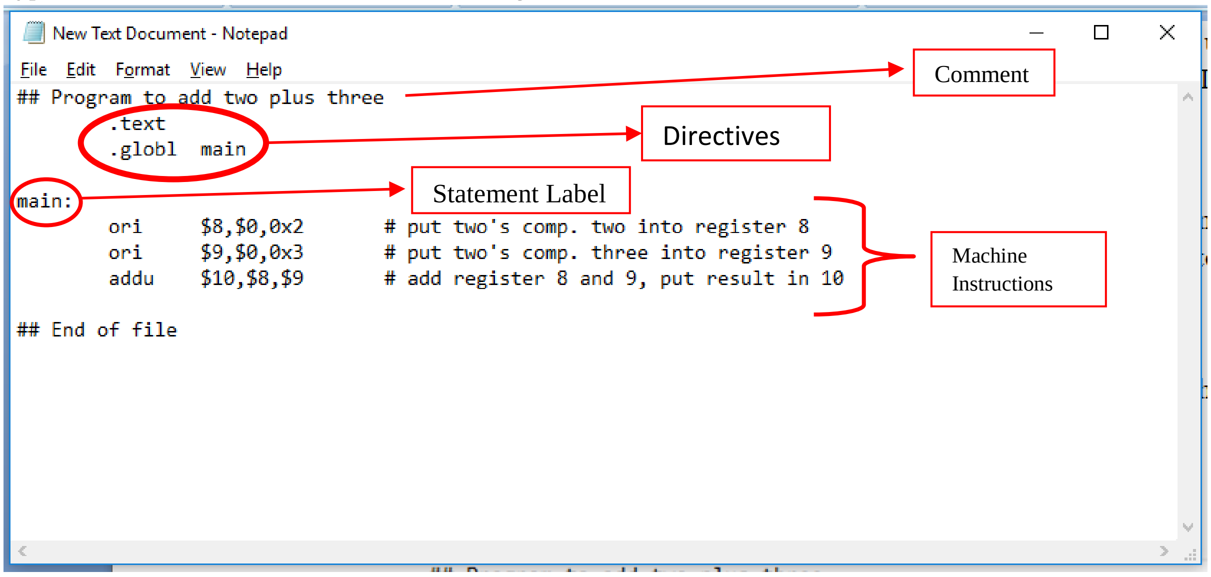
PART A:
In this part, you will become familiar with QtSpim simulator by working with a simple MIPS assembly language program to add two numbers.

a) You will first write the assembly program to add two numbers as detailed in the steps listed below.
b) You will then simulate the program in QtSpim and examine how the values in registers and in program counter change.

STEPS:

1. To write the assembly code, we need to open a text editor. It can be the Notepad on Windows or the Text Editor on Linux.

2. Type the demo code in the editor as shown in Fig. 1



```
## Program to add two plus three
        .text
        .globl  main
main:
        ori     $8,$0,0x2       # put two's comp. two into register 8
        ori     $9,$0,0x3       # put two's comp. three into register 9
        addu    $10,$8,$9       # add register 8 and 9, put result in 10

## End of file
```

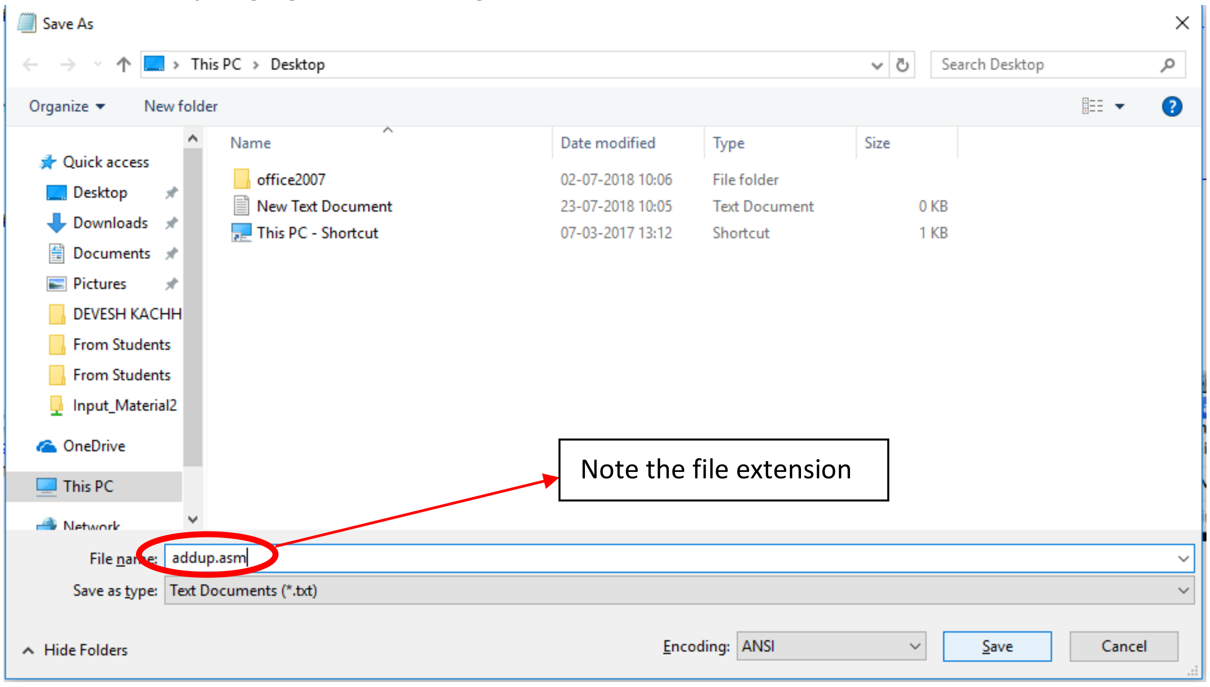
Fig.1
The character '#' starts a comment. Everything on the line from '#' to the right is ignored.

Directives tell the assembler what the programmer wants.

.text directive tells the assembler, that the following lines are '.text' – source code for the program. .globl main tells the assembler that the identifier *main* will be used outside of this source file (that is, used "globally") as the label of a particular location in main memory.

main is a statement label, or a symbolic address, which is actually a source code name for a location in memory. This means, here, main stands for the address of the first machine instruction. The programmer refers to memory locations by name, and lets the assembler figure out the numeric address.

3. Once this program is typed, this file needs to be saved with '.asm' extension since this program is written in assembly language as shown in Fig. 2.



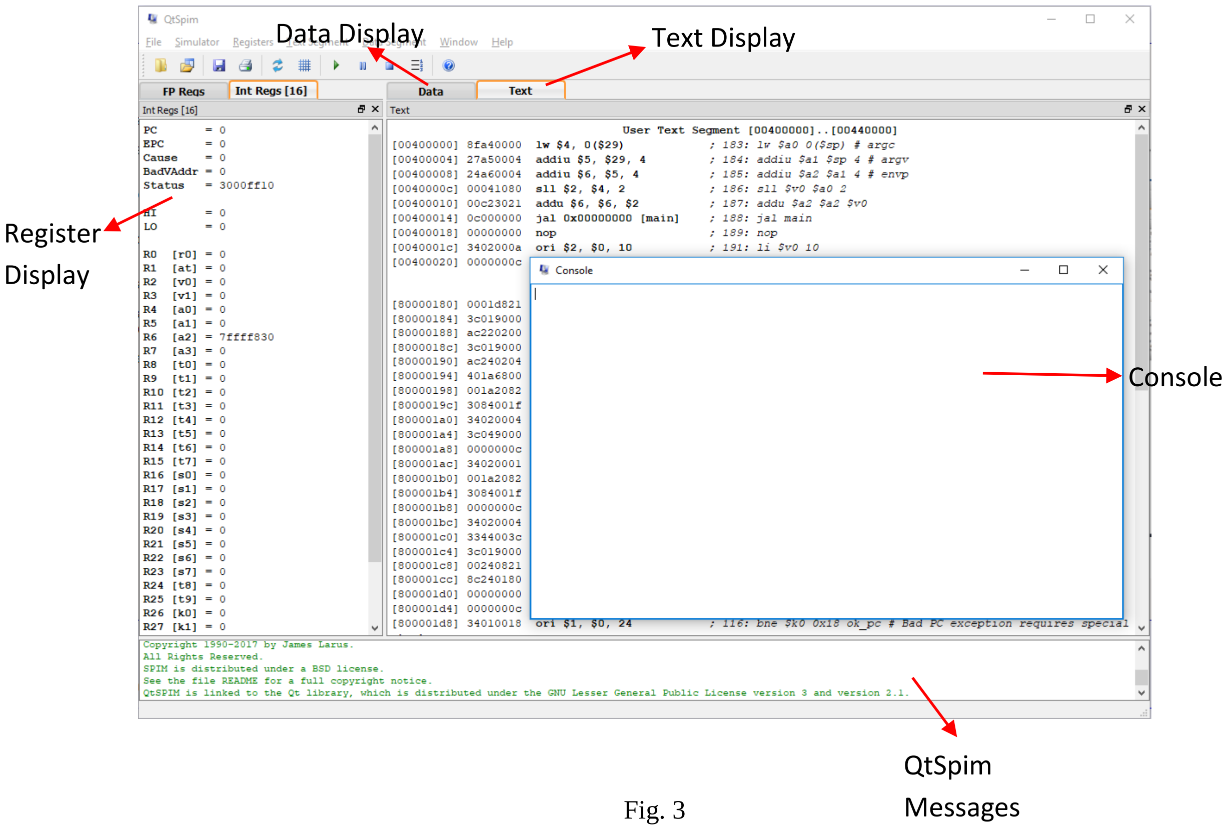
Name	Date modified	Type	Size
office2007	02-07-2018 10:06	File folder	
New Text Document	23-07-2018 10:05	Text Document	0 KB
This PC - Shortcut	07-03-2017 13:12	Shortcut	1 KB

File name: addup.asm
Save as type: Text Documents (*.txt)
Encoding: ANSI

Fig. 2

With this we finish with writing the assembly code. Now we need to simulate it in the QtSpim simulator.

4. Open the QtSpim tool by double clicking on the icon present on the Desktop. On Linux, you can type QtSpim in the search bar. Two windows open as shown in Fig. 3. One is the simulator and the other is the console.
The simulator is divided into 4 regions.



```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffff830
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
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                         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


[80000180] 0001d821
[80000184] 3c019000
[80000188] ac220200
[8000018c] 3c019000
[80000190] ac240204
[80000194] 401a6800
[80000198] 001a2082
[8000019c] 3084001f
[800001a0] 34020004
[800001a4] 3c049000
[800001a8] 0000000c
[800001ac] 34020001
[800001b0] 001a2082
[800001b4] 3084001f
[800001b8] 0000000c
[800001bc] 34020004
[800001c0] 3344003c
[800001c4] 3c019000
[800001c8] 00240821
[800001cc] 8c240180
[800001d0] 00000000
[800001d4] 0000000c
[800001d8] 34010018  ori $1, $0, 24         ; 116: bne $k0 0x18 ok_pc # Bad PC exception requires special
```

```
Copyright 1990-2017 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.
```

Fig. 3

Register Display- Shows the contents of all 32 general purpose registers (in binary, hex or decimal) , the floating point registers and few others.

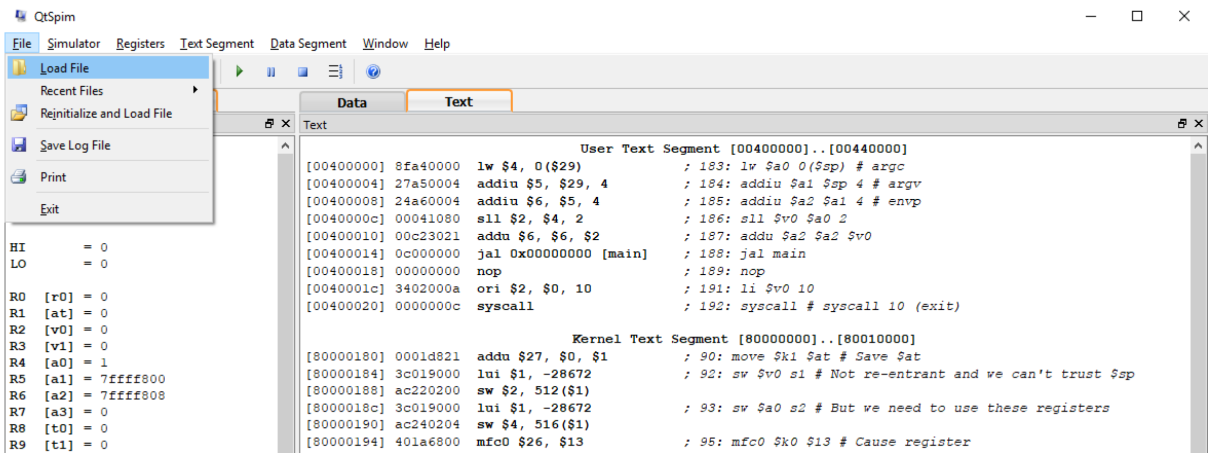
Text Display- This region is split into 4 columns. They correspond to the address of the machine instruction, the machine instruction code (in hex), the corresponding assembly language code and the explaination of the code.

Data and Stack: This shows the sections of MIPS memory, that hold ordinary data and also the data pushed on Stack.

SPIM Messages: This shows messages from the simulator(often error messages)

Console: Character output from the simulated computer appears here.

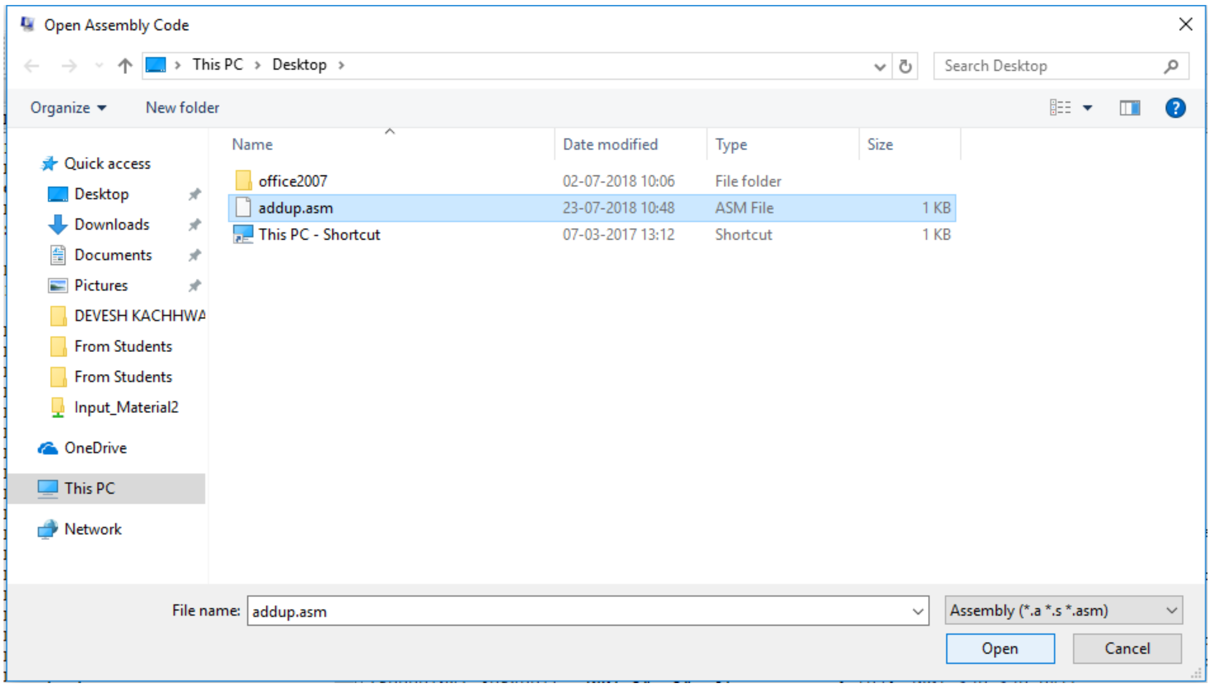
5. Now you need to load your source file for simulation as shown in Fig. 4
   File ->Load File and Select the assembly source code file stored with .asm extension as shown in Fig. 5


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
Load File
Recent Files
Reinitialize and Load File
Save Log File
Print
Exit
Data
Text

```
HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff800
R6  [a2] = 7ffff808
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
```

```
                         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
```

Fig. 4



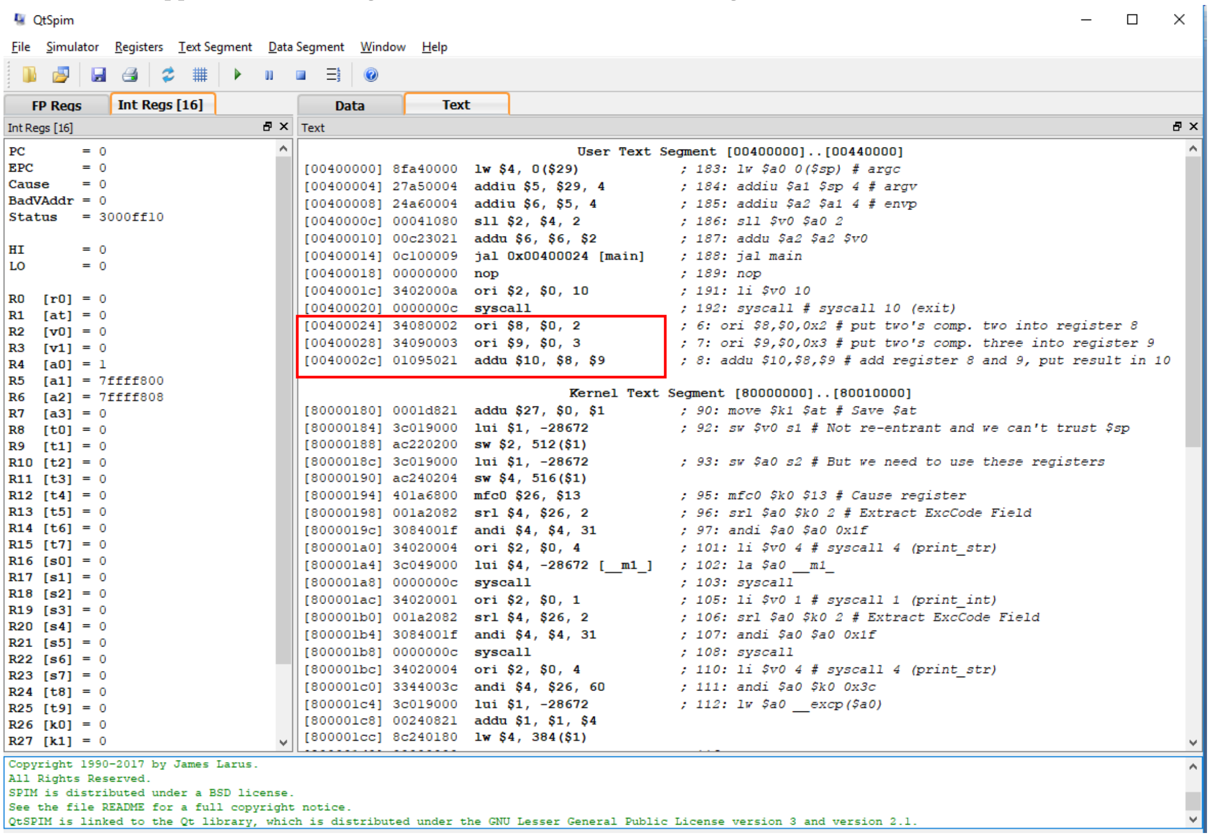
Name	Date modified	Type	Size
office2007	02-07-2018 10:06	File folder	
addup.asm	23-07-2018 10:48	ASM File	1 KB
This PC - Shortcut	07-03-2017 13:12	Shortcut	1 KB

File name: addup.asm

Assembly (*.a *.s *.asm)

Fig. 5

6. The code will appear in the text region of the simulator as seen in Fig. 6



```
Int Regs [16]
PC        = 0
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff800
R6  [a2] = 7ffff808
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
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 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] 0c100009  jal 0x00400024 [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)
[00400024] 34080002  ori $8, $0, 2            ; 6: ori $8,$0,0x2 # put two's comp. two into register 8
[00400028] 34090003  ori $9, $0, 3            ; 7: ori $9,$0,0x3 # put two's comp. three into register 9
[0040002c] 01095021  addu $10, $8, $9         ; 8: addu $10,$8,$9 # add register 8 and 9, put result in 10

                       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)
```

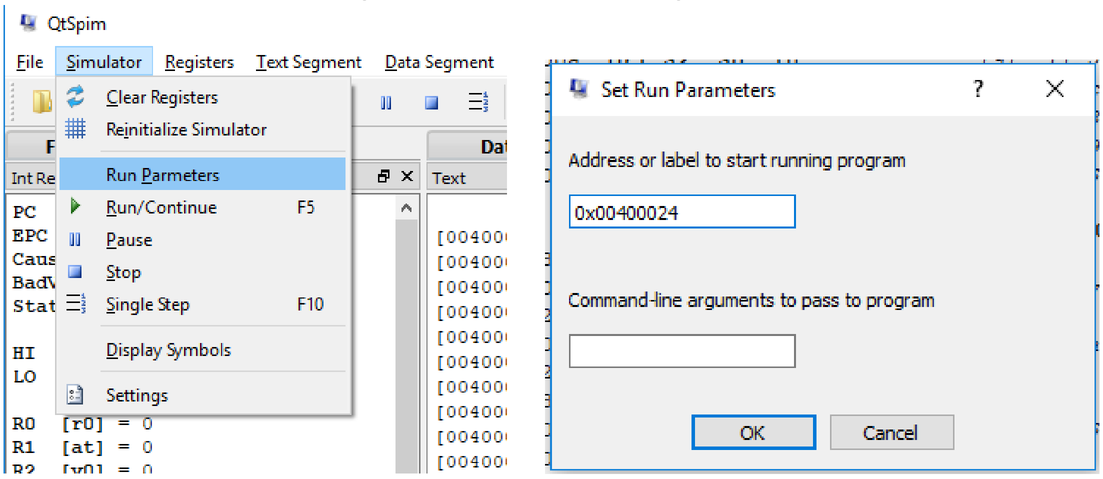
```
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.
```

Fig. 6

7. Now to simulate this code, you need to initialise the Program Counter Register, to point to the memory location that refers to the start of your code.
As seen in Fig. 6 in the red box, the start of your code points to location [00400024], this has to be stored in PC.

Simulator -> Run Parameters

Add the address 00400024 in the given text box as shown in Fig. 7



```
PC
EPC
Caus
BadV
Stat

HI
LO

R0   [r0]  = 0
R1   [at]  = 0
R2   [v0]  = 0
```

```
[00400
[00400
[00400
[00400
[00400
[00400
[00400
[00400
[00400
[00400
```

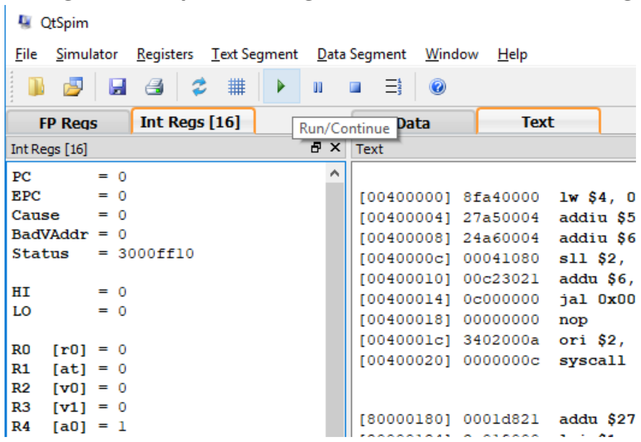
Address or label to start running program: 0x00400024

Command-line arguments to pass to program:

Fig. 7

8. This completes the initialisation process , now you have to simulate the program. There are two ways in which this can be done : either simulate the entire program or simulate the program stepwise.
When the entire program is simulated, only the final output is seen in the respective registers.
But when the program is simulated step wise, the changes in the registers during the intermediate steps can be seen, which helps to understand the flow of the code and also to troubleshoot and debug errors.

9. Complete Simulation:
This can be done by pressing the F5 key or clicking the Run icon as shown in Fig. 8..

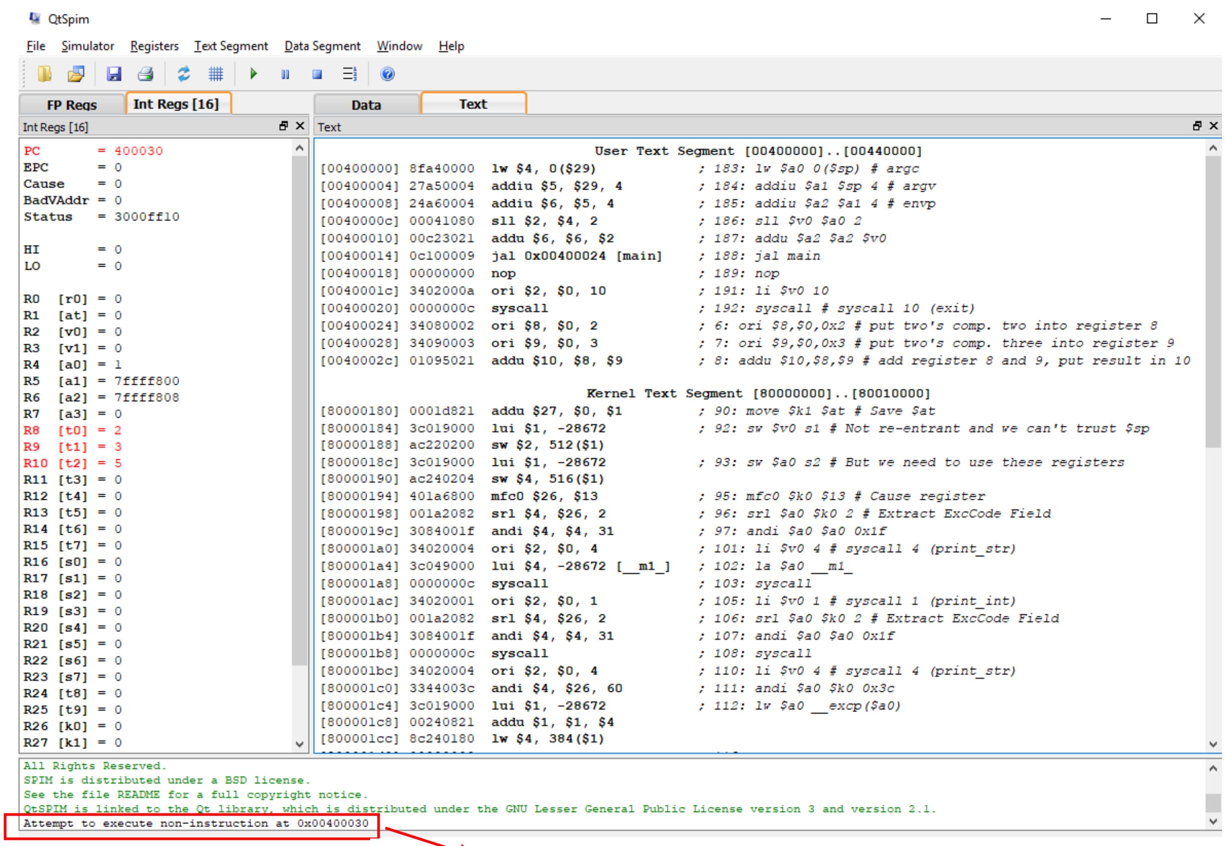


```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
```

```
[00400000] 8fa40000  lw $4, 0
[00400004] 27a50004  addiu $5
[00400008] 24a60004  addiu $6
[0040000c] 00041080  sll $2,
[00400010] 00c23021  addu $6,
[00400014] 0c000000  jal 0x00
[00400018] 00000000  nop
[0040001c] 3402000a  ori $2,
[00400020] 0000000c  syscall


[80000180] 0001d821  addu $27
```

Fig. 8

Once the program is simulated, you can check the values of PC and the registers that store the output (R8,R9 and R10 in this case). PC points to the address of the next machine instruction. Last instruction that was executed was stored at 0040002c , hence PC points to the next location which is 00400030 as you can see in Fig. 9.


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text

```
Int Regs [16]
PC       = 400030
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff800
R6  [a2] = 7ffff808
R7  [a3] = 0
R8  [t0] = 2
R9  [t1] = 3
R10 [t2] = 5
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
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

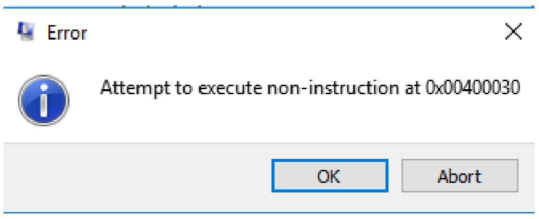
```
Text
                        User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4        ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4         ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [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)
[00400024] 34080002  ori $8, $0, 2           ; 6: ori $8,$0,0x2 # put two's comp. two into register 8
[00400028] 34090003  ori $9, $0, 3           ; 7: ori $9,$0,0x3 # put two's comp. three into register 9
[0040002c] 01095021  addu $10, $8, $9        ; 8: addu $10,$8,$9 # add register 8 and 9, put result in 10

                       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)
```

```
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.
Attempt to execute non-instruction at 0x00400030
```

Shows the end of the program as no machine code at 00400030

Fig. 9

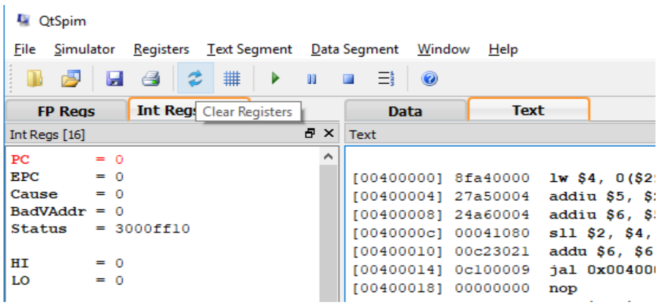
The simulator tries to execute the instruction at the next address location after executing the last machine instruction, but since our code has ended, it notifies in the SPIM messages region of the simulator as shown in Fig. 9 in the red box or as an error message in the error dialog box in Fig. 10.



Attempt to execute non-instruction at 0x00400030

Fig. 10.

10. Now to try simulating it step by step, you need to reinitialise the registers. This can be done by clicking the icon to clear registers as shown in Fig. 11.


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Reg
Clear Registers
Data
Text

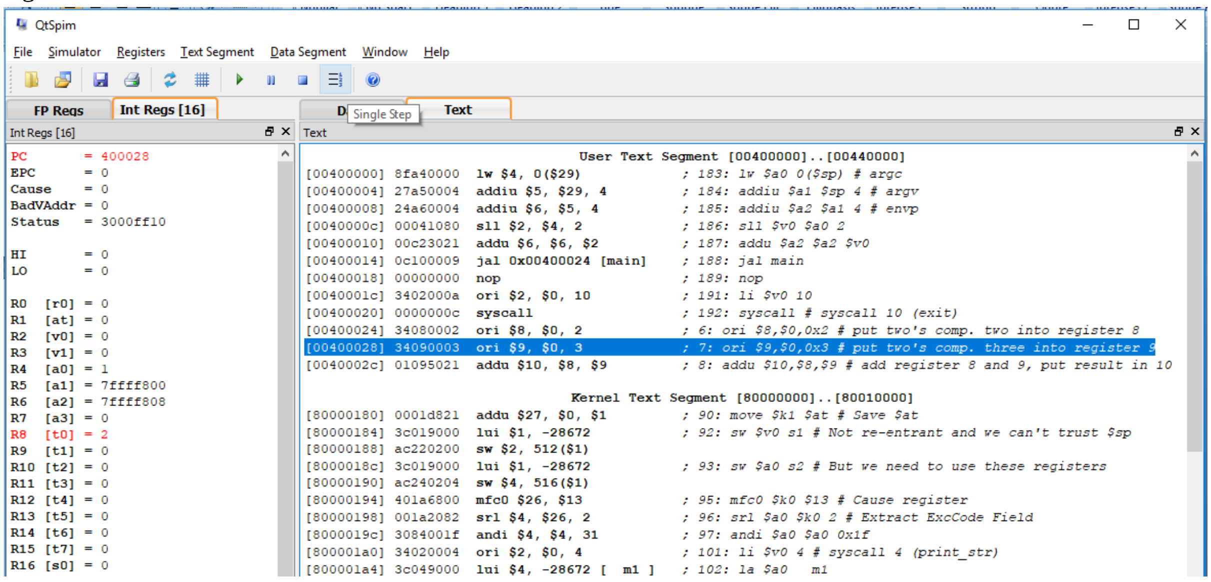
```
Int Regs [16]
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0
```

```
Text

[00400000] 8fa40000  lw $4, 0($2
[00400004] 27a50004  addiu $5, $
[00400008] 24a60004  addiu $6, $
[0040000c] 00041080  sll $2, $4,
[00400010] 00c23021  addu $6, $6
[00400014] 0c100009  jal 0x00400
[00400018] 00000000  nop
```

Fig. 11

Once you click on this icon,you can see that PC and other registers that held the output previously, are cleared to zero.(Note: PC still holds the first machine instruction address as you had hardcoded it initially)

To start simulation step by step, you can press the F10 key, or click the single step icon as shown in Fig. 12.


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
D
Single Step
Text
Int Regs [16]
Text

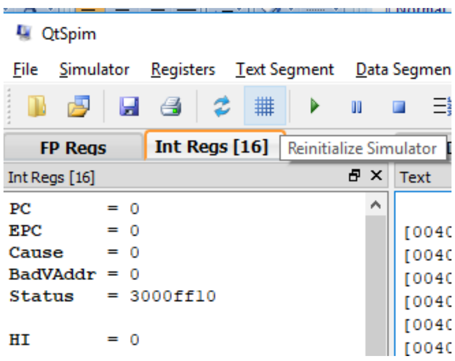
```
PC       = 400028
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff800
R6  [a2] = 7ffff808
R7  [a3] = 0
R8  [t0] = 2
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
```

```
                    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] 0c100009  jal 0x00400024 [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)
[00400024] 34080002  ori $8, $0, 2            ; 6: ori $8,$0,0x2 # put two's comp. two into register 8
[00400028] 34090003  ori $9, $0, 3            ; 7: ori $9,$0,0x3 # put two's comp. three into register 9
[0040002c] 01095021  addu $10, $8, $9         ; 8: addu $10,$8,$9 # add register 8 and 9, put result in 10

                    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
```

Fig. 12

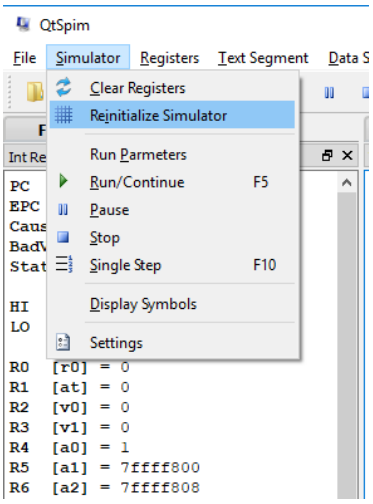
On each click, the changes in PC and other registers will be reflected in the Registers area.
At the end, when the PC points to the location after the last machine instruction i.e. the next instruction, the simulation will stop.

11. Now to reinitialise the simulator, that is, to load a new assembly program file, we need to first remove the already loaded file and also clear all the registers. This can be done by clicking the reinitialise icon or navigating to:
Simulator->Reinitialise as shown in Fig. 13.



```
PC       = 0
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
```

OR



```
PC
EPC
Caus
BadV
Stat

HI
LO

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff800
R6  [a2] = 7ffff808
```

Fig. 13

PART B

In this part, you will write MIPS assembly language programs to do the following exercises:

a) Add numbers 100 and -82 and store the result in register \$10.(Hint: Find 2's complement of 82 and add it to 100)
b) Store FFFF in \$8. Left shift it 2 times and store the result in memory location 0x10000000.
c) Evaluate the expression $(2x+3)^2$ where x is the content in register \$10 based on exercise (a). Store the result in register \$13.

You should check whether the results of the above exercises reflect correctly in the appropriate registers and the memory locations.