

Lab Sheets 1 & 2 for CS F342 Computer Architecture  
Semester 1 : 2020-21

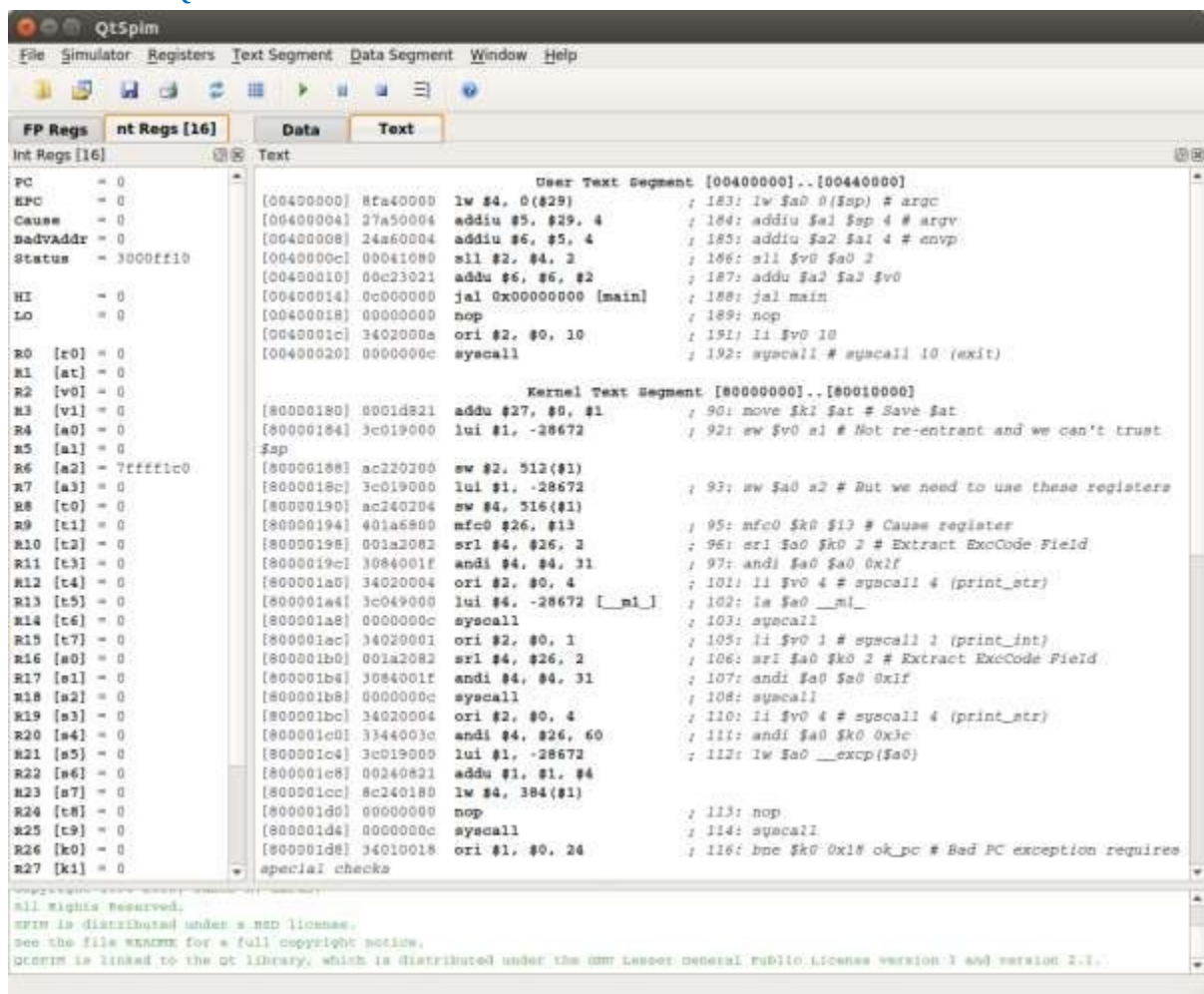
## Goals for Lab 1 & 2:

**Goal 1:** Installing and launching QTSPIM – this tool will be used for major part of the lab work.

### Installing and launching QTSPIM

Open a terminal to run the following commands.

- A. Check the OS and the processor support on your Linux machine – verify whether it is 32 bit or 64 bit. [Try commands like `uname -p` ; `uname -a` ; `man uname`]
- B. Download the appropriate QTSPIM binary from Internet (<https://sourceforge.net/projects/spimsimulator/files/>)
- C. Install using dpkg tool. Do not forget to get “super user privilege” using sudo.  
`$ sudo dpkg -i <qtspim package>`
- D. Launch QTSPIM



```
QtSpim
File Simulator Registers Text Segment Data Segment Window Help

FP Regs  nt Regs [16]  Data  Text
Int Regs [16]  Text

PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000fff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 0
R5 [a1] = 0
R6 [a2] = 7ffffff0
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 [a0] = 0
R17 [a1] = 0
R18 [a2] = 0
R19 [a3] = 0
R20 [a4] = 0
R21 [a5] = 0
R22 [a6] = 0
R23 [a7] = 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] 00041000 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] 8001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 a1 # Not re-entrant and we can't trust
$sp
[80000188] ac220200 sw $2, 512($1) ; 93: sw $a0 a2 # But we need to use these registers
[8000018c] 3c019000 lui $1, -28672 ; 94: sw $a0 a2 # But we need to use these registers
[80000190] ac240204 sw $4, 516($1) ; 95: mfc0 $k0 $13 # Cause register
[80000194] 401a6900 mfc0 $26, $13 ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[80000198] 001a2082 srl $4, $26, 2 ; 97: andi $a0 $a0 0x1f
[8000019c] 3084001f andi $4, $4, 31 ; 101: li $v0 4 # syscall 4 (print_str)
[800001a0] 34020004 ori $2, $0, 4 ; 102: la $a0 __mi_
[800001a4] 3c049000 lui $4, -28672 [__mi_] ; 103: syscall
[800001a8] 0000000c syscall ; 105: li $v0 1 # syscall 1 (print_int)
[800001ac] 34020001 ori $2, $0, 1 ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b0] 001a2082 srl $4, $26, 2 ; 107: andi $a0 $a0 0x1f
[800001b4] 3084001f andi $4, $4, 31 ; 108: syscall
[800001b8] 0000000c syscall ; 110: li $v0 4 # syscall 4 (print_str)
[800001bc] 34020004 ori $2, $0, 4 ; 111: andi $a0 $k0 0x1c
[800001c0] 3344003c andi $4, $26, 60 ; 112: lw $a0 __excp($a0)
[800001c4] 3c019000 lui $1, -28672 ; 113: nop
[800001c8] 00240821 addu $1, $1, $4 ; 114: syscall
[800001cc] 8c240180 lw $4, 384($1) ; 116: bne $k0 0x18 ok_pc # Bad PC exception requires
[800001d0] 00000000 nop ; 117: nop
[800001d4] 0000000c syscall ; 118: syscall
[800001d8] 34010018 ori $1, $0, 24 ; 119: bne $k0 0x18 ok_pc # Bad PC exception requires

special checks

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.
```

Note: Please refer to the following: (i) HP\_AppA.pdf (ii) Chapter 2 Hennessey & Patterson Book (iii) **MIPS Reference Data Card (“Green sheet”)**

**Goal 2:** To get introduced to QTSPIM and implement some code related to - System Calls and User Input. Further, we will do basic integer Add/Sub/And/Or and their immediate flavours (e.g. ori).

Reference for MIPS assembly – refer to the **MIPS Reference Data Card (“Green sheet”)** uploaded in CMS. Further, some of the QTSPIM assembly instructions are beyond this data card (e.g. the pseudo instruction **la**).

Additionally use Appendix A (HP\_AppA.pdf) from Patterson and Hennessey “Assemblers, Linkers and the SPIM Simulator” for gaining background knowledge of SPIM.

In this lab we focus on reversing only integer based instructions (add, or, subi etc.).

### Reference for Registers:

0 zero constant 0	16 s0 callee saves
1 at reserved for assembler	...
2 v0 results from callee	23 s7
3 v1 returned to caller	24 t8 temporary (cont'd)
4 a0 arguments to callee	25 t9
5 a1 from caller: caller saves	26 k0 reserved for OS kernel
6 a2	27 k1
7 a3	28 gp pointer to global area
8 t0 temporary	29 sp stack pointer
...	30 fp frame pointer
15 t7	31 ra return Address
	caller saves

System calls as well as functions (in later part of the semester) should take care of using the registers in proper sequence. Especially take note of V0, V1 [R2, R3 in QTSPIM] and a0-a3 [R4-R7 in QTSPIM] registers.

### Reference for System Calls:

Service	Code (put in \$v0)	Arguments	Result
print_int	1	\$a0=integer	
print_float	2	\$f12=float	
print_double	3	\$f12=double	
print_string	4	\$a0=addr. of string	
read_int	5		int in \$v0
read_float	6		float in \$f0
read_double	7		double in \$f0
read_string	8	\$a0=buffer, \$a1=length	
sbrk	9	\$a0=amount	addr in \$v0
exit	10		

### Reference for Data directives:

#### **.word w1, ..., wn**

-store n 32-bit quantities in successive memory words

#### **.half h1, ..., hn**

-store n 16-bit quantities in successive memory half words

#### **.byte b1, ..., bn**

-store n 8-bit quantities in successive memory bytes

#### **.ascii str**

-store the string in memory but do not null-terminate it

-strings are represented in double-quotes "str"

-special characters, eg. \n, \t, follow C convention

#### **.asciiz str**

-store the string in memory and null-terminate it

#### **.float f1, ..., fn**

-store n floating point single precision numbers in successive memory locations

#### **.double d1, ..., dn**

-store n floating point double precision numbers in successive memory locations

#### **.space n**

-reserves n successive bytes of space

**Layout of Code in QTSPIM:** Typical code layout (\*.asm file edited externally)

# objective of the program

.data #variable declaration follows this line

.text #instructions follow this line

main: # the starting block label

...

xxx

yyy

zzz

.....

li \$v0,10 #System call- 10 => Exit;

syscall # Tells QTSPIM to properly terminate the run

#end of program

**Exercise 0:** Understanding Pseudo instruction.

Not all instructions used in the lab will directly map to MIPS assembly instructions. Pseudo-instructions are instructions not implemented in hardware. E.g. using \$0 or \$r0 we can load constants or move values across registers using add instruction.

**E.g. `li $v0, 10` actually gets implemented by assembler as `ori $v0, $r0, 10`**

In subsequent exercises, identify the pseudo instruction by looking at the actual code used by QTSPIM.

**Exercise 1:** Integer input and output and stepping through the code. Invoking

system calls to output (print) strings and and input (read) integers.

**The following code snippet prints the number 10 on console. Modify it to read any number and print it back.**

Hint: To copy it from \$v0 to \$a0, you can use add or addi with 0 or similar options.

Edit the code in your editor of choice and then load it in QtSpim. Single Step [  ] through the code and look at the register values as you execute various instructions.

The demo code is given below:

**# demo code to print the integer value 10**

```
.data #variable declaration follow this line  
# sample string variable declaration - not used in first exercise.  
myMsg: .asciiz "Hello Enter a number." # string declaration  
        # .asciiz directive makes string null terminated
```

```
.text #instructions follow this line  
main:  
li $a0,10  
li $v0,1  
syscall
```

```
li $v0,10 #System call - Exit - QTSPIM to properly terminate the run  
syscall  
#end of program
```

**Exercise2:** Modify the above code to output “myMsg” along with the input integer. You will use load address MIPS instruction (`la $a0, myMsg`)

**Exercise 3:** Take 2 integers as input, perform addition and subtraction between them and display the outputs. The result of addition is to be displayed as "The sum is =" and that of subtraction is to be displayed as "The difference is =". Check if negative integers can be handled.

*Observations: List all the pseudoinstructions used in this exercise and discuss.*

**Exercise 4:** Disassemble the binary / hex code to MIPS assembly code. Note that pseudoinstructions cannot be identified using this. For your information, a brief discussion for a sample instruction follows below.

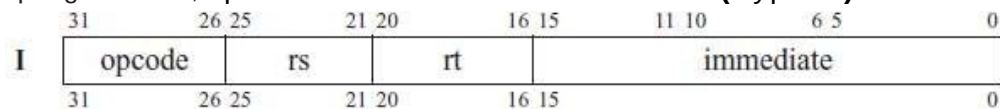
FP Regs	nt Regs [16]	Data	Text
Int Regs [16]			Text
PC = 0			User Text Segment [00400000]..[00440000]
EPC = 0			[00400000] 8fa40000 lw \$4, 0(\$29) ; 183: lw \$a0 0(\$sp) # argc
Cause = 0			[00400004] 27a50004 addiu \$5, \$29, 4 ; 184: addiu \$a1 \$sp 4 # argv
BadVAddr = 0			[00400008] 24a60004 addiu \$6, \$5, 4 ; 185: addiu \$a2 \$a1 4 # envp
Status = 3000fff10			[0040000c] 00041080 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2
			[00400010] 00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0
HI = 0			[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
LO = 0			[00400018] 00000000 nop ; 189: nop
			[0040001c] 3402000a ori \$2, \$0, 10 ; 191: li \$v0 10
R0 [r0] = 0			[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
R1 [at] = 0			[00400024] 3404000a ori \$4, \$0, 10 ; 6: li \$a0, 10
R2 [v0] = 0			[00400028] 34020001 ori \$2, \$0, 1 ; 7: li \$v0, 1
R3 [v1] = 0			[0040002c] 0000000c syscall ; 8: syscall
R4 [a0] = 1			[00400030] 34020005 ori \$2, \$0, 5 ; 10: li \$v0, 5
R5 [a1] = 7ffff1ac			[00400034] 0000000c syscall ; 11: syscall
R6 [a2] = 7ffff1b4			[00400038] 20480000 addi \$8, \$2, 0 ; 12: addi \$t0, \$v0, 0
R7 [a3] = 0			[0040003c] 3c041001 lui \$4, 4097 [str] ; 14: la \$a0, str
R8 [t0] = 0			[00400040] 34020004 ori \$2, \$0, 4 ; 15: li \$v0, 4
R9 [t1] = 0			[00400044] 0000000c syscall ; 16: syscall
R10 [t2] = 0			[00400048] 21040000 addi \$4, \$8, 0 ; 18: addi \$a0,\$t0, 0
R11 [t3] = 0			[0040004c] 34020001 ori \$2, \$0, 1 ; 19: li \$v0, 1

For code at address 0x0040 0038 – which is having a value of 0x2048 0000 when we break into opcode etc. we get:

Binary representation: 0010 0000 0100 1000 0000 0000 0000 0000

OpCode value is: 0010 00 (8 decimal)

As per green card, OpCode 8 decimal is for **addi** (type I)



rs value is: 00 010 => 2 decimal- register v0

rt value is: 01 000 => 8 decimal – register t0

immediate value is: 0000 0000 0000 0000 => 0

Hence the instruction is **addi \$t0, \$v0, 0**

In groups, write different assembly instructions and ask your group members to reverse from hex-notation.

Also as an exercise reverse the following three values:

- 00a64020
- 00a64822
- 34020005