Lab Sheet 2 for CS F342 Computer Architecture

Semester 1 – 2017-18

Version 1.0

Goals for the Lab: We will get introduced to QtSpim and implement some code related to - System Calls and User Input. Furthermore we will do basic integer Add/Sub/And/Or and their immediate flavours (e.g. ori).

<u>Reference for MIPS assembly</u> – refer to the **MIPS Reference Data Card ("Green Card")** uploaded in CMS. Note that it's not green in PDF. Furthermore some of the QtSpim assembly instructions are beyond this data card (e.g. the pseudo instruction ***<u>la</u>***).

Additionally use Appendix from of Patterson and Hennessey (Appendix B in 4th Edition) "Assemblers, Linkers and the SPIM Simulator" for background of SPIM.

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

<u>Reference for Registers:</u>

0	zero	constant 0
1	at	reserved for assembler
2	v0	results from callee
3	v1	returned to caller
4	a0	arguments to callee
5	a1	from caller: caller saves
6	a2	
7	a3	
8	t0	temporary
. . .		
15	t7	

16	s0	callee saves
. . .		
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	pointer to global area
29	sp	stack pointer
30	fp	frame pointer
31	ra	return Address caller saves

System calls as well as functions (in later part of 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.

<u>Reference for System Calls:</u>

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

<u>Layout of Code in QtSPIM:</u> 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
```

<u>Exercise 0:</u> 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`

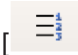
In subsequent exercises, identify the pseudo instruction by looking at the actual code used by QtSpim.

<u>Exercise 1:</u> Integer input and output and stepping through the code.

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

Following code snipped prints number 10 on console. Modify it to read a 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.

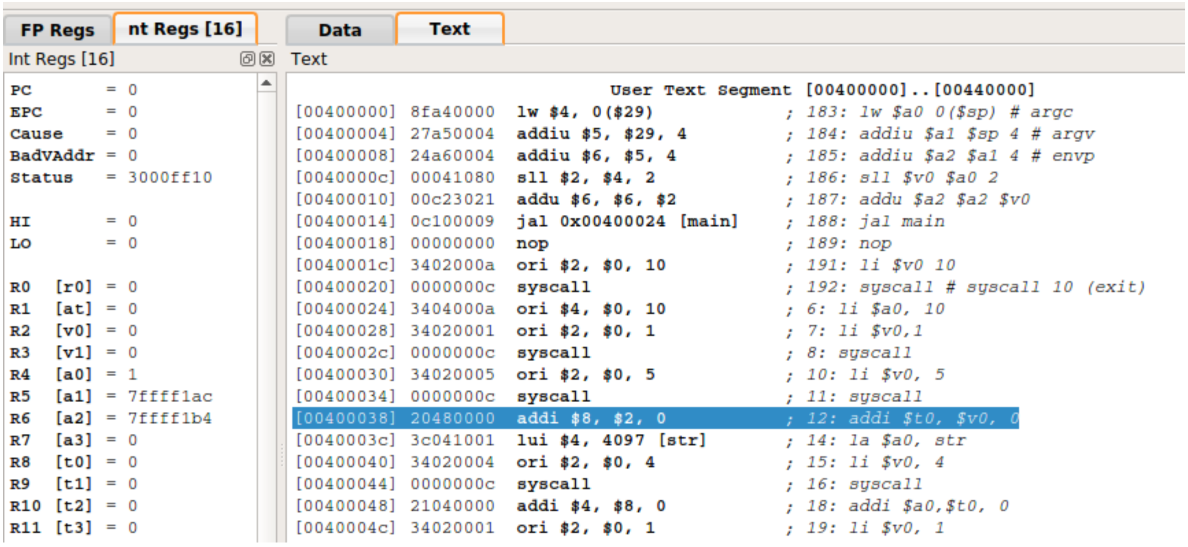
```
# demo code to print the integer value 10

.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.

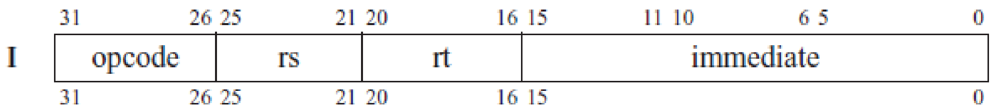
Observations: List all the pseudoinstructions used in this exercise and discuss.

Exercise 4: Extend Exercise 3 to 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.



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 reverse the following three values:

1. `00a64020`
2. `00a64822`
3. `34020005`

Next Week (Lab 3): Multiply / Divide + Simple Floating point instructions.