Lab Sheet 2 for CS F342 Computer Architecture
Semester 1 : 2023-24

Explanation: the basic program in MIPS (initializing variable, take input from user, addition), basic binary analysis of instructions used in MIPS architecture.

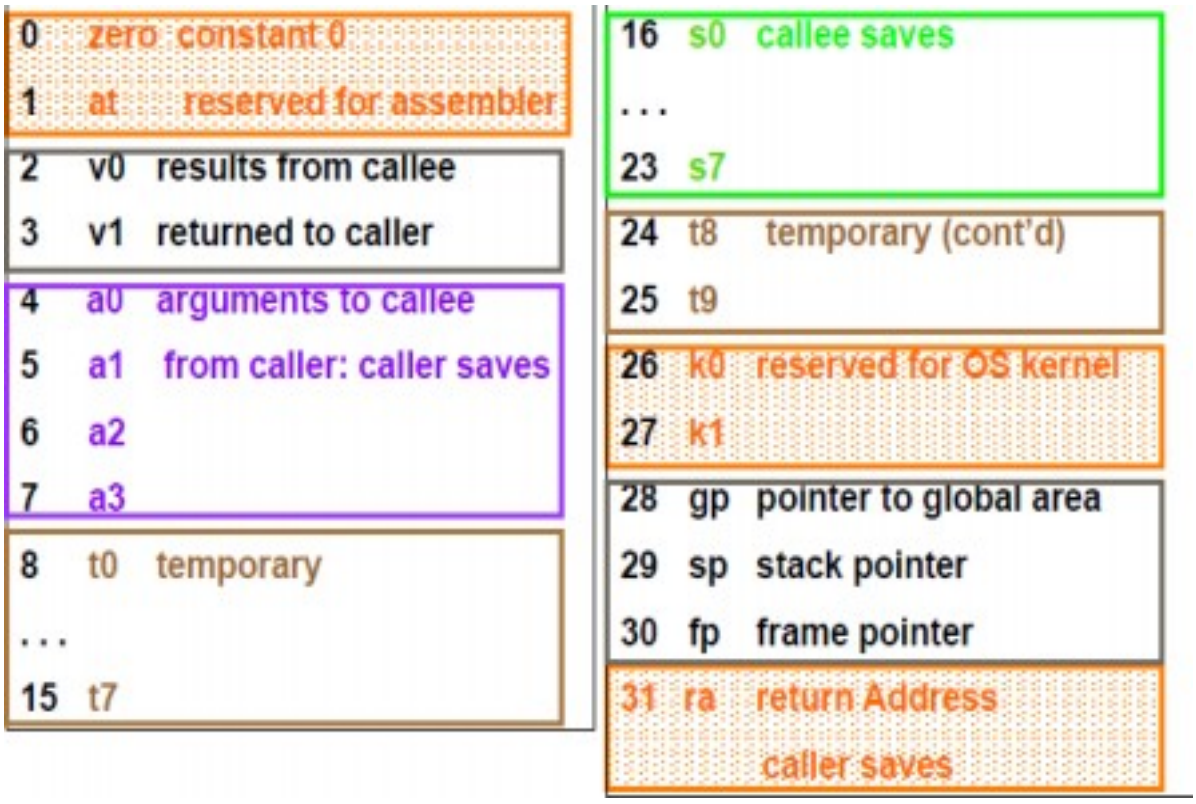
Goal : To get introduced to QTSPIM and implement some code related to - System Calls and User Input. Further, we will do basic integer Add/Sub/And/Or and their immediate flavours (e.g. ori).

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:



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

Class Exercises

Exercise 1: Modify the above code to output “myMsg” along with the input integer.

Hint: 1) You will use load address MIPS instruction (la $a0, myMsg)

2) For printing string use (li $v0 4)

Exercise 2: 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.

Hint: To read integer input from user use (li $v0 5)

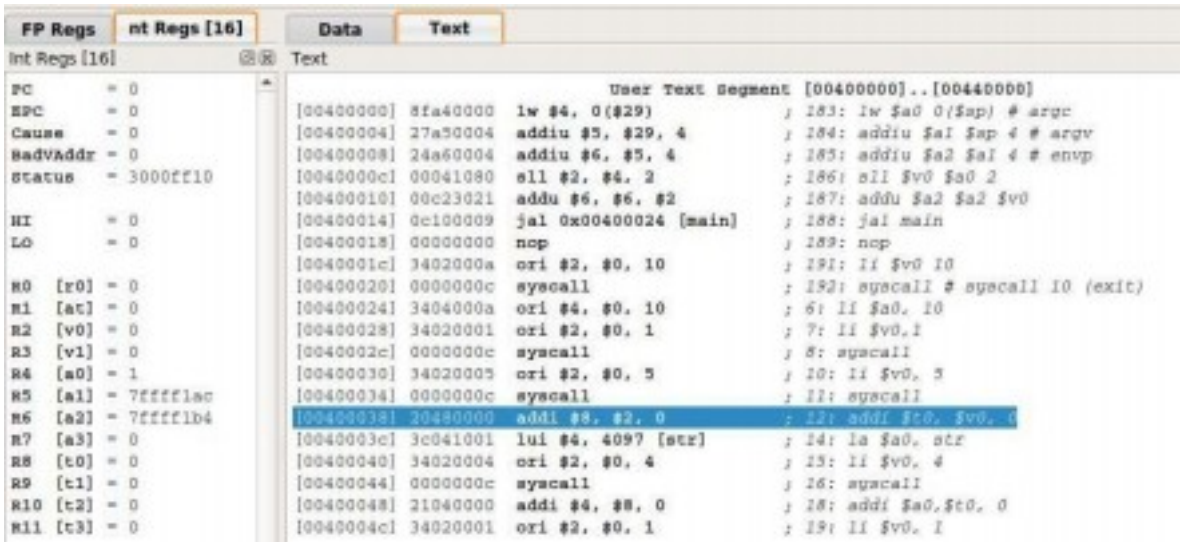
2) use store word sw or load word lw instruction to read input variables and store in memory.

3) Finally perform addition and store in $a0 and print using (li $v0 1)

4) you can also use move ins to move between registers instead of mem location.

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

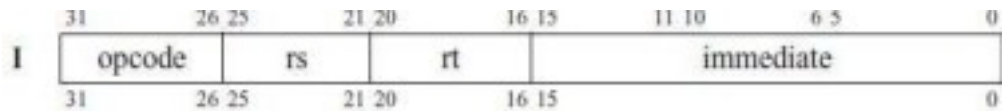
Exercise 3: Disassemble the binary/hex code to MIPS assembly code. Note that pseudoinstructions cannot be identified using this. For your information, a brief discussion of a sample instruction follows below.



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 as an exercise reverse the following three values:

1. 22940004
2. 00c23021
3. 34020005