



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

## Computer Architecture and Assembly Language Lab

Fall 2016

---

### Lab 1

## Machine Language Instructions, warming up with SPIM simulator, high-to-low level language conversion

---

### Goal

In this laboratory exercise, you will practice basic operations of assembly language by using the simulation tool *QtSpim* [5]. After this lab, you should understand how a computer executes assembly and machine language instructions and be able to write a simple assembly program.

---

### Preparation

Please read the SPIM instructions in Appendix A in the textbook. This pre-knowledge is required in this lab. We will use *QtSpim*, the software that will help you simulate the execution of MIPS assembly programs. In order to know how to use *QtSpim*, you should also look at “*Tutorial on Using QtSpim*”. This material will be supplied on Sakai/Resources.

---

### Introduction to *QtSpim*

*Spim* is a simulator that runs MIPS32 programs. It's been around for more than 20 years. *QtSpim* is the newest version of *Spim*. It runs on Microsoft Windows, Mac OS X, and Linux with the same source code and the same user interface. It can do a context and syntax check while loading an assembly program. In addition, it adds in necessary overhead instructions as needed, and updates register and memory content as each instruction is executed.



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

*QtSpim* is installed on the computers in EE 103. To acquire this software for your personal computer, you can download it from <http://sourceforge.net/projects/spimsimulator/files/>

## ➤ Get Started

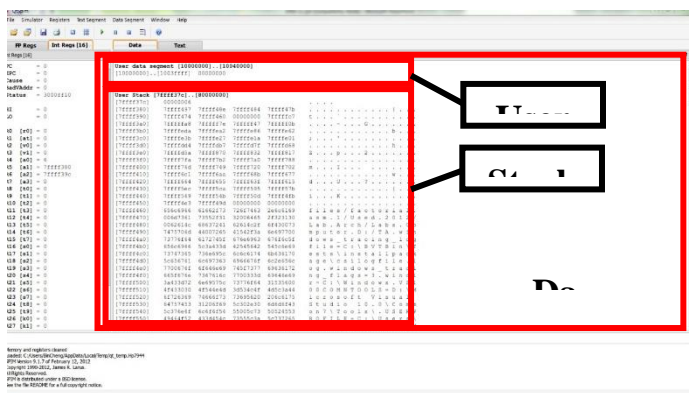
When opening *QtSpim*, you will see the windows below



1. The Register tabs display the content of all registers. You can view the registers as binary, hexadecimal or decimal. The processor Program Counter is also shown here.
2. The Text Tab displays the MIPS instructions loaded into memory to be executed. The text window has five columns. From left to right, they are:
  - The memory address of an instruction,
  - The contents of the address in hexadecimal,
  - The actual MIPS instructions where register numbers are used,
  - The MIPS assembly that you wrote,
  - Any comments you made in your code are displayed.
3. The Data tab displays memory addresses and their values in the data and stack segments of the memory.
4. The Information Console lists the actions performed by the simulator. To activate the Console windows, you need to click on the “**Window**” button, and select “**Console**”.

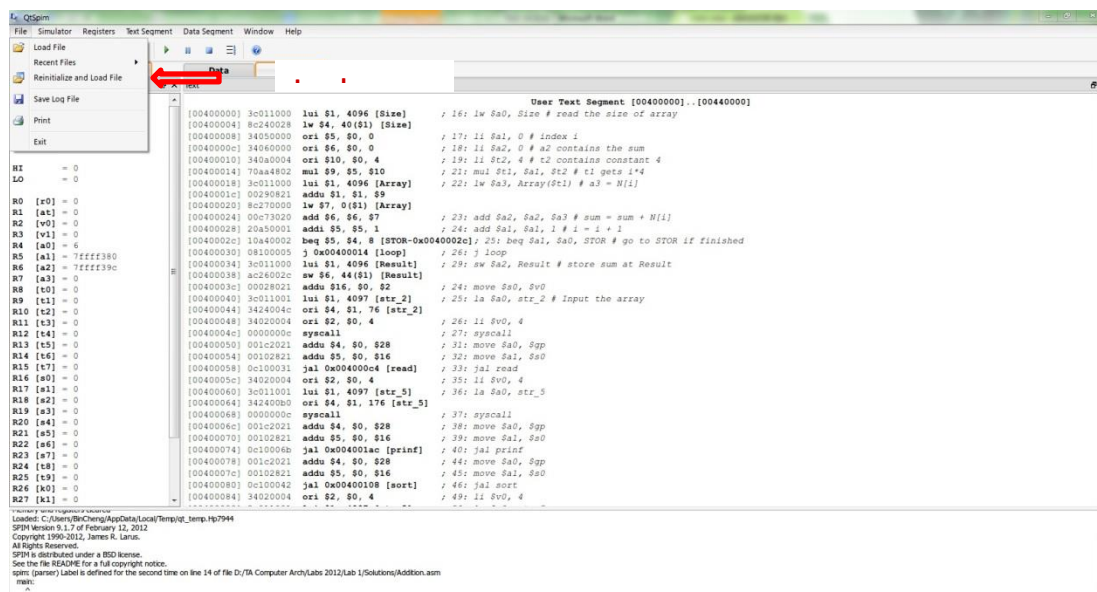


Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey



➤ **Running a Program in QtSpim**

1. Use a text editor to create your program “**xxx.asm**” or “**xxx.s**”.
2. Click on the “**File**” button and open your program



3. You can then run the program by simply pressing the “**run**” button – all instructions will be executed at once, and the final contents of memory and the register file will be reflected in the *QtSpim* window.
- 4.



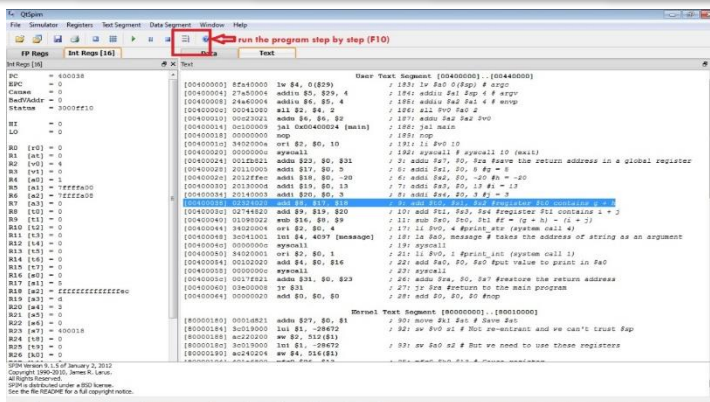
Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

your assembly code starts

current instruction

register change resulting from previous instruction

5. You can also run the program in single-stepping, which allows you to run your program one instruction at a time. The single stepping icon can be found in the toolbar. You can also do it by pressing the function key **F10**.



## Exercises

In these exercises, you will use *QtSpim* to practice some basic assembly language operations, such as memory-transfer, reading and printing, and simple arithmetic operations.



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

## Exercise 1: Practice using the simulator

In order to understand the operation of *QtSpim*, type to a file named "**Ex1.s**" and execute the following MIPS assembly program. This program computes the factorial of a number that is stored in memory and the result is stored in a register..

```
# Exercisel is used in assignments 1 and 2
.text 0x00400000

    .globl main

main:

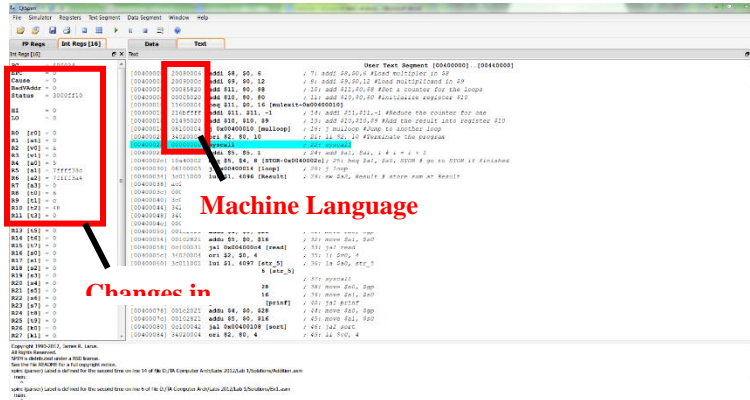
    lw $10, Number1($0)
    ori $11, $0, 1
    ori $9, $0, 1
    #compute the factorial of Number ($10)!
factloop:
    bge $11, $10, factexit
    mul $9, $10, $9
    sub $10, $10, 1
    j factloop
factexit:
    #the computation of the factorial is over
    #Is the result in $9 correct? The result in $9 is in
    hexadecimal form
    li $2, 10
    syscall

    .data 0x10000000
    .align 2
Number1: .word 18
```



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

After executing this program, you can see:



The machine language instructions are divided into fields.

- **op**: Basic operation of the instruction
- **rs**: The first register source operand
- **rt**: The second register source operand
- **rd**: The destination register operand. It gets the result of the operation
- **shamt**: Shift amount
- **funct**: function. This field, often called the *function code*, selects the specific variant of the operation in the op field

For OR immediate instruction `ori $10, $0, 8` the fields format is

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

For simple **add** instruction `add $8, $0, $10`, the fields format are

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

## Assignment 1

For the instructions below, indicate what the corresponding Machine Language Instructions in Binary codes are in the table below





Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

	op	rs	rt	constant
ori \$9,\$0,1				

## Assignment 2

A well-written assembly program always has comments after each instruction and the start of the file.

- a) Put the comments to the end of each line after a sharp sign (#), for example,

```
li $2, 10    #Place (load immediate) number 10 into register $2.
```

Each comment should explain briefly (3 - 4 words max) what the instruction does.

- b) Include a preamble with comments explaining what the program does. Do not write more than 3 to 4 lines.

## Exercise 2: Memory transfer instructions

A data transfer from main memory to a register is possible using the memory reference instruction load word (**lw**),

**lw \$destination, offset(\$base)**

which loads **\$destination** with the data word starting from memory address **offset + \$base**. For example, **lw \$20, 10000 (\$0)** means that 32-bit data in the memory location at address **10000+\$0** is stored into register **\$20**.

A data transfer from a register in the register file to a memory location is done using a store word (**sw**) instruction,

**sw \$source, offset(\$base)**

which stores the **\$source** register into the memory to the address starting from **offset+\$base**. For example, **sw \$10, 20000 (\$2)** stores the contents of register **\$10** into the memory location at address **20000+\$2**. Both in **lw** and **sw** instructions, **offset** is a 16-bit signed integer.



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

```
#Exercise is used in assignment 3
.text 0x00400000
.align 2
.globl main
main:
lw $a0, Size           # Load the size of array into $a0, using lw
li $a1, 0              # initialize index i
                        # t2 contains constant 4, initialize t2
                        # Initialize result to one

loop:
mul $t1, $a1, $t2      # t1 gets i*4
                        # a3 = Array[i]
                        # result = result * Array[i]
                        # store result in the Array2 in location i
                        # i = i + 1

blt $a0, $a1, END      # go to END if finished
j loop
END:
    li $v0, 10
    syscall
.data 0x10000000
.align 2
Array: .word 2 5 6 7 12 16
Size: .word 6
```

In order to understand the operation of memory transfer instructions (**lw** and **sw**), type and execute the following assembly program, whose name is “**Ex2.asm**”. The operation procedure is similar to Ex1.





Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

### Assignment 3

1. Fill out the missing instructions according to the comments.
2. Run the program step by step by pressing the function key **f10**. Observe the change of each register and each memory place at each step.
3. Record the data segment of this program and check the output.
4. Explain why index *i* reaches *size+1* and not *size*?

### Exercise 3: Reading and Printing

SPIM provides a small set of operating system–like services through the system call, named **syscall** instruction. To request a service, a program loads the system call code (it can be found in textbook B.9.1) into register **\$v0** and argument into register file registers **\$a0–\$a3** (or **\$f12** for floating-point values). System calls that return values put their results in the register **\$v0** (or **\$f0** for floating-point results) [1]. Reading and printing operations can be done with **syscall**. In order to become familiar with **syscall**, read the following example first.

In this example, the system call is first used to read input. The system code for this operation is **5**. Then we use system code **1** to print this input. Notice that we also use code **10** to terminate the program.

### Assignment 4

- 1- Type and execute the code in following program, record the value of each register.
- 2- Discuss the importance and value of register **\$v0** in using **syscall**.



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

3- Change the following program to print a message asking for the user to give as input the two numbers.

```
# Used in assignment 4
# Registers used: $t0 - used to hold the first number.
# $t1 - used to hold the second number.
# $t2 - used to hold the difference of the $t1 and $t0.
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter.

.text
main:
## Get first number from user, put into $t0.
li $v0, 5          # load syscall read_int into $v0.
syscall            # make the syscall.
move $t0, $v0      # move the number read into $t0.

## Get second number from user, put into $t1.
li $v0, 5 # load syscall read_int into $v0.
syscall # make the syscall.
move $t1, $v0 # move the number read into $t1.
```

4- Extend the program to include a message that will be provided from the user and printed along with the addition result.

**For example:**

### Exercise 4: Programming Exercise

Addition and subtraction are performed on 32 bit numbers held in the general purpose registers (32 bit-wide each) inside the register file. The result is a 32 bit number itself. Two kinds of instructions are included in the instruction set to do integer addition and subtraction [5]:

- Instructions for signed arithmetic: the 32 bit numbers are considered to be represented in 2's complement. The execution of the instruction (addition or subtraction) may generate an overflow, such as for **add**, **addi**, or an underflow for **sub**,



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

- Instructions for unsigned arithmetic: the 32 bit numbers are considered to be in standard binary representation. Executing the instruction will never generate an overflow error even if there is an actual overflow (the result cannot be represented with only 32 bits), such as, **addiu**, **addu**, **subu**. Some basic arithmetic operations include:

Instructions	Operation
<b>add \$d, \$s, \$t</b>	<b>\$d = \$s + \$t; advance_pc (4);</b>
<b>addi \$t, \$s, imm</b>	<b>\$t = \$s + imm; advance_pc (4);</b>
<b>sub \$d, \$s, \$t</b>	<b>\$d = \$s - \$t; advance_pc (4);</b>
<b>div \$s, \$t</b>	<b>\$LO = \$s / \$t; \$HI = \$s % \$t; advance_pc (4);</b>
<b>mult \$s, \$t</b>	<b>\$LO = \$s * \$t; advance_pc (4);</b>

### Assignment 5

Write a program that gets two pairs of numbers ( $x_1, y_1$ ) , ( $x_2, y_2$ ) and calculates their Manhattan Distance ([https://en.wikipedia.org/wiki/Taxicab\\_geometry](https://en.wikipedia.org/wiki/Taxicab_geometry)) . The equation for the Manhattan Distance is  $d = x_1 - x_2 + y_1 - y_2$ . ( Assume that  $x_1 > x_2$  and  $y_1 > y_2$  )

### Assignment 6

The following program is trying to copy words from the memory point where \$a0 points to the point where \$a1 points. In addition the program counts the words that are copied and stores the counter value to the register \$v0. The copying stops when the program finds a word equal to 0. This last word is copied but should not be counted.

```
Loop: lw $v1, 0($a0)      #read the next word
      Add $v0, $v0, 1     #increase the counter
```



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

```
sw $v1, 0($a1)    #store the word to the destination
add $a0, $a0, 1    #move the pointer to the next word
add $a1, $a1, 1    #move the pointer to the next word
bne $v1, $zero, Loop #if the word is not 0 repeat the loop
```

There are a lot of syntax errors in this program. Fix them by giving a bug free version of this program. [In this particular exercise you do not have to keep the elements of \$v1, \$a0, \$a1.]

### Exercise 5: Get familiar with more complex programs

Load and run the program below, **ex5.asm**. Read the code and comments to familiarize yourself with a more complex sample of assembly language code. Observe the changes of the different registers as you step through the program.

#### Assignment 7

Explain what the main function of **ex5.asm** is. How is this function realized? What would happen if the user was allowed to give a much larger number? Why is that?

#### Assignment 8

Change the main function of **ex5.asm** so that it calculates the sum of the user's input and all the odd numbers that are less or equal than the user's input.

---

### Lab report

Write a proper report using MS Word that includes your codes, results and the conclusion of assignments. The program files must be printed and note that each lab report is due before the start of the next lab. Please put your name and Student ID at the start of the code.

---



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

```
#ex5.asm
.data 0x10000000
    ask: .asciiz "\nEnter a number between 0 and 50000: "
    ans: .asciiz "Answer: "

.text 0x00400000
.globl main
main:
    li $v0,4
    la $a0, ask          # Loads the ask string
    syscall             # Display the ask string
    li $v0, 5           # Read the input
    syscall
    move $t0, $v0        # n = $v0, Move the user input to t0
    addi $t1, $0, 0      # i = 0
    addi $t2, $0, 0      # ans = 1, Starting case (n=0) is 1
    li $t3, 2            # we store two in $t3
loop:
    beq $t0, 0, END      # if we reach 0 we stop
    div $t0, $t3          # divide $t0 with 2
    mfhi $t1             # store the remainder to $t1
    beq $t1, $0, ADD     # if the remainder is 0 then we go to ADD
    sub $t0, $t0, 1      # we reduce $t0
    j loop               # go back to loop

ADD: add $t2, $t2, $t0    # add the even number to the $t2
    sub $t0, $t0, 1      # reduce $t0
    j loop               # go back
END:
    li $v0,4
    la $a0, ans          # Loads the ans string
    syscall
    move $a0, $t2        # Loads the answer
    li $v0, 1
    syscall
    li $v0, 10
    syscall
```



Department of Electrical and Computer Engineering  
Rutgers, The State University of New Jersey

## References

1. Patterson and Hennessy, “Computer Organization and Design: The Hardware / Software interface”, 5<sup>th</sup> Edition.
2. Daniel J. Ellard, “MIPS Assembly Language Programming: CS50 Discussion and Project Book”, September 1994.
3. “Get Started with SPIM”,  
<http://www.cs.washington.edu/education/courses/cse378/05sp/handouts/spimwin.pdf>
4. “Arithmetic in MIPS”, <http://www.cs.iit.edu/~virgil/cs470/Labs/Lab6.pdf>
5. “SPIM: A MIPS32 Simulator”, <http://spimsimulator.sourceforge.net/>