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

Computer Architecture and Assembly Language Lab

Spring 2016

Lab 1

Machine Language Instructions, warming up with simulator, high level-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.

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*

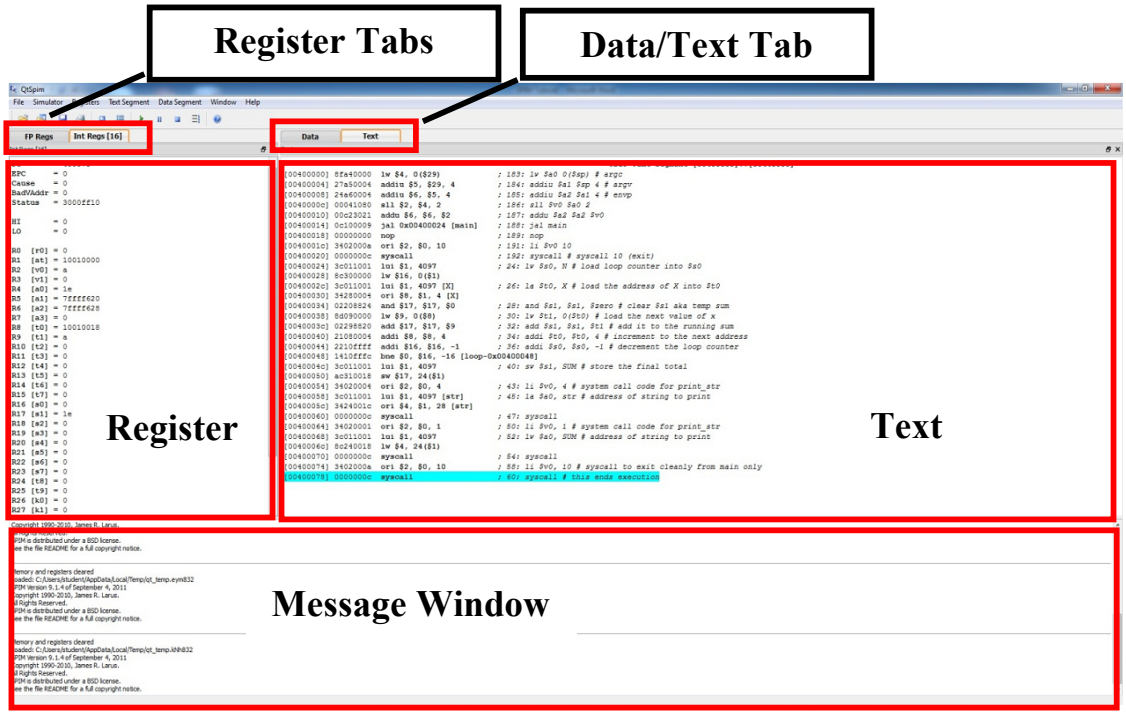
Spim is a simulator that runs MIPS32 programs. It's been around for more than 20 years. *QtSpim* is the newest version of *Spim*. It runs on Microsoft Windows, Mac OS X, and Linux with the same source code and the same user interface. It can do a context and syntax check while loading an assembly program. In addition, it adds in necessary overhead instructions as needed, and updates register and memory content as each instruction is executed. It has been pre-loaded on the computers in EE 103. To acquire this software for your personal computer, you can

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

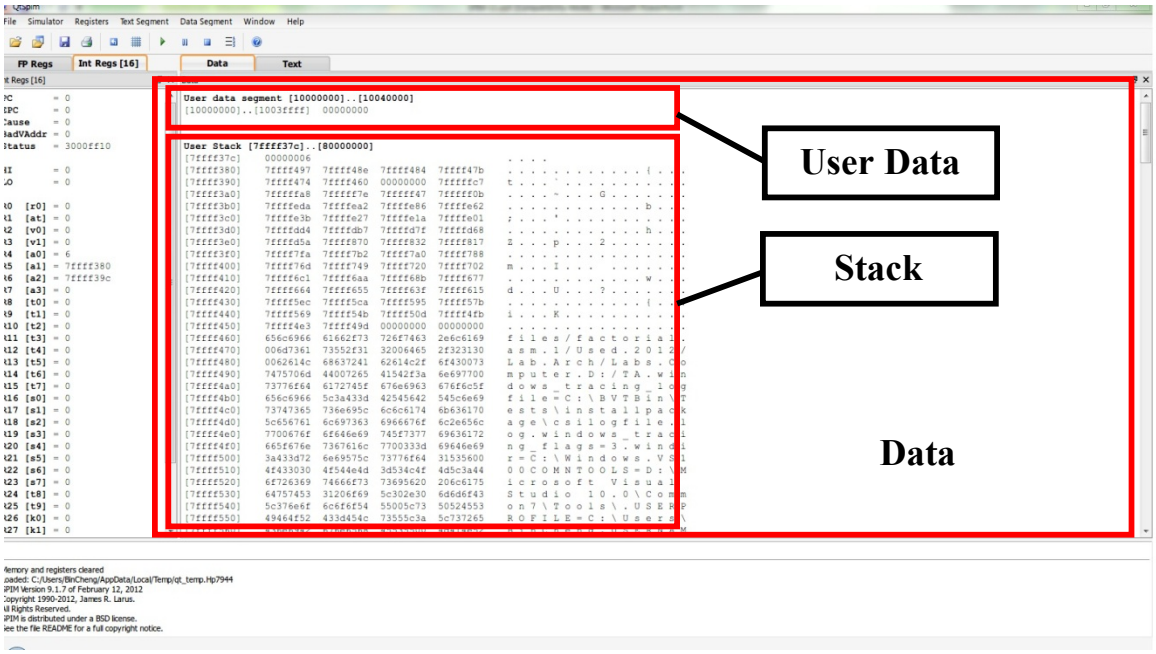
download it from **http://sourceforge.net/projects/spimsimulator/files/** or **http://pages.cs.wisc.edu/~larus/spim.html.**

➢ Get Started

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

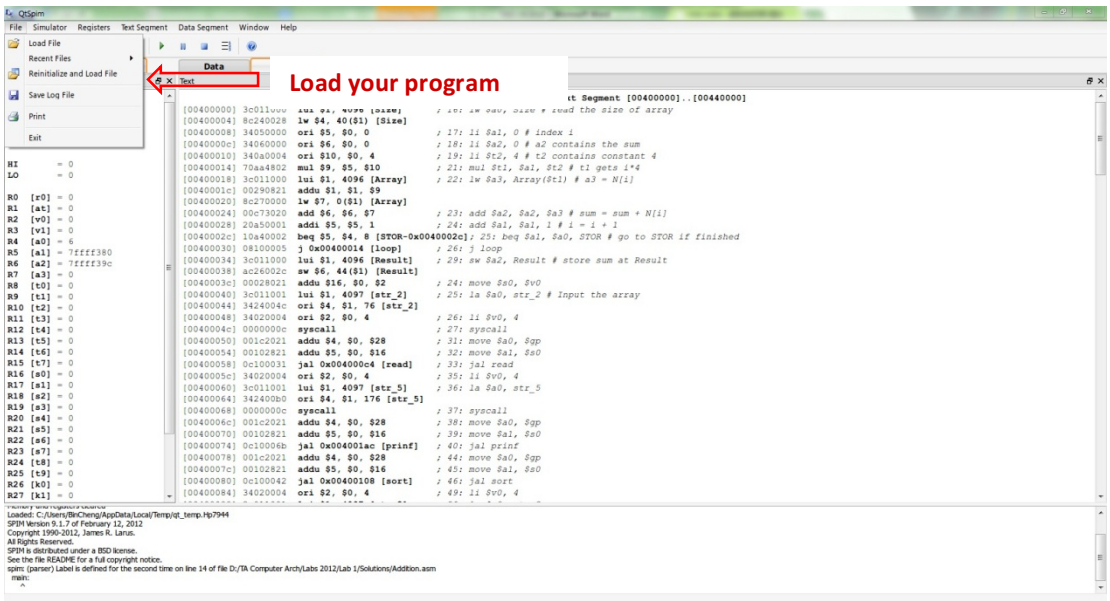


1. The Register tabs display the content of all registers. You can view the registers as binary, hexadecimal or decimal. The processor Program Counter is also shown here.
2. The Text Tab displays the MIPS instructions loaded into memory to be executed. The text window has five columns. From left to right, 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`**".

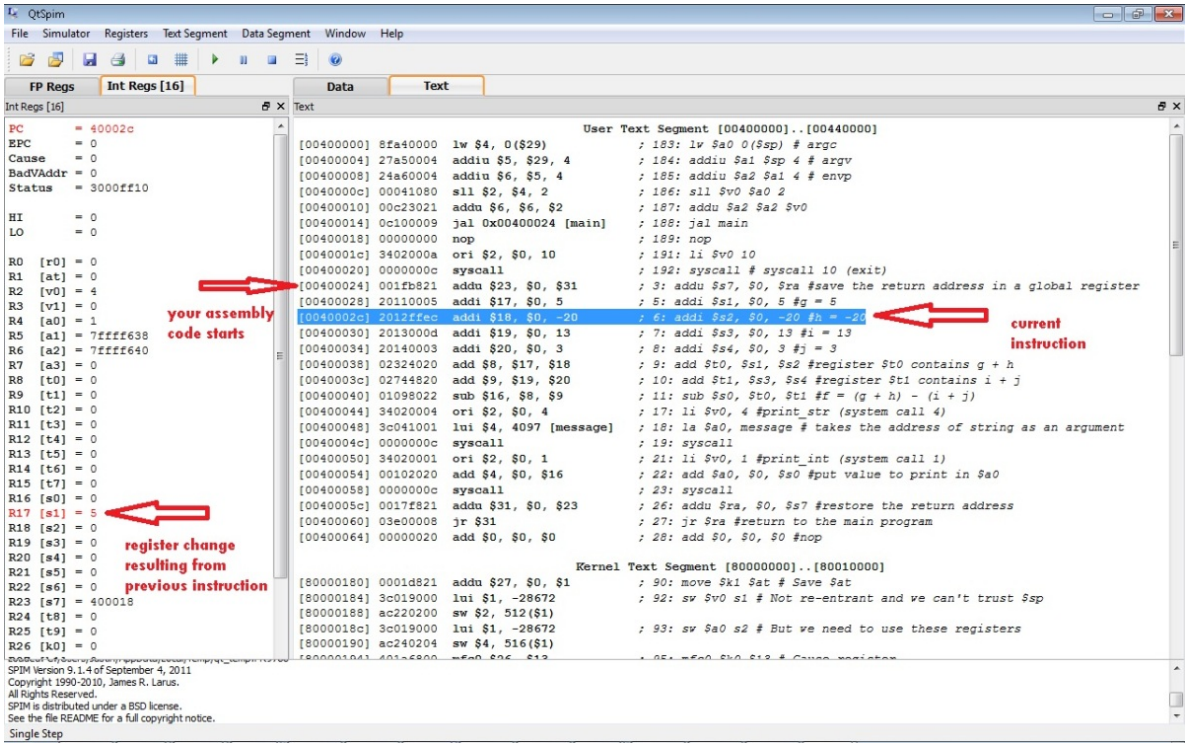


➢ Running a Program in *QtSpim*

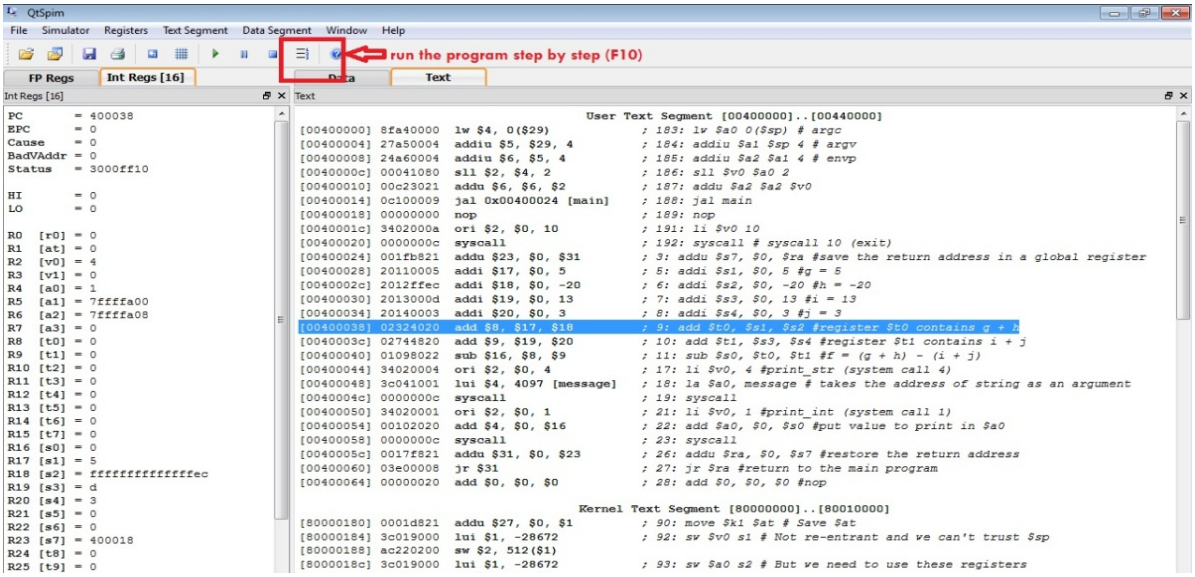
1. Use a text editor to create your program "**`xxx.asm`**" or "**`xxx.s`**".
2. Click on the "**`File`**" button and open your program



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

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. 

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.

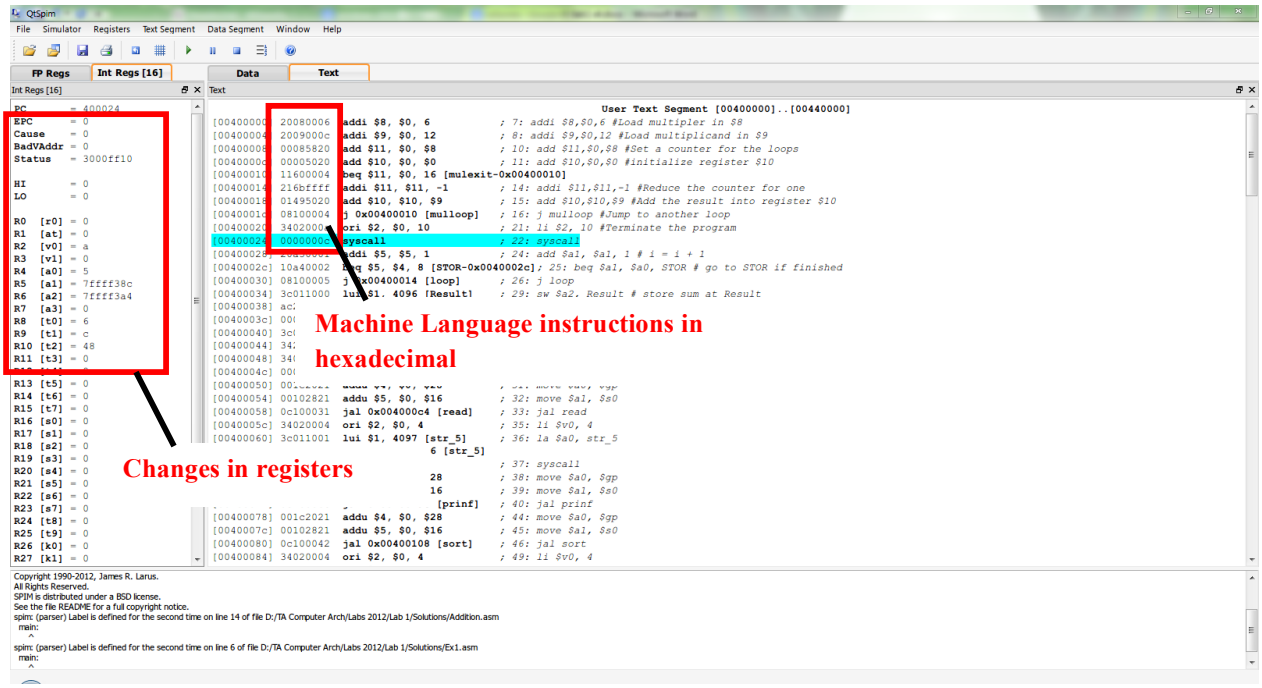
Exercise 1: Practice using the simulator

In order to understand the operation of *QtSpim*, type to a file named "**`Ex1.asm`**" and execute the following MIPS assembly program. This program power operation of two unsigned integers which are placed in two registers and writes the result to another register.

```
# Exercise1 is used in assignments 1 and 2
.data 0x10000000
.text 0x00400000
.globl main
main:
    addi $10, $0, 8
    li $11, 6
    # calculate the $11 -th power of $10
    add $8, $0, $10
    li $12, 2
powerloop:
    bgt $12, $11, powerexit
    sub $11, $11, 1
    mul $10, $10, $8
    j powerloop
powerexit:
    # power operation loop is over
    # Is the result in $10 correct? The result in $10 is hexadecimal
    li $2, 10
    syscall
```

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 add immediate instruction **`addi $10, $0, 80`** load immediate instruction **`li $11, 6`** the fields format are

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. **Hint:** Wikipedia page of 'MIPS instruction set' might be useful.

Instructions	op	rs	rt	rd	shamt	funct
add $8, $0, $10						
	op	**rs**	**rt**	**constant**		
addi $10, $0, 8						

Assignment 2

A well-written program always has comments in front of each instruction. Put the comments to the end of each line after a sharp sign (#), for example,

```
li $11, 6   #Place (load immediate) number 6 into register $11.
```

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`**.

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

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.

```
# Exercise2 is used in assignments 3
.text 0x00400000
.align 2
.globl main
main:
lw $a0, Size              # load the size of array into $a0, using lw
addi $a0, $a0, -1         # init index j with size - 1
li $a1, 0                 # init index i = 0
                          # t2 contains constant 4, init t2
loop:
    mul $t1, $a1, $t2         # t1 gets i*4
                              # t4 gets j*4
    lw $a3, Array1($t1)        # a3 = Array[i]
                              # double the value a3 = a3 * 2
                              # store a3 to Array1[j]
                              # i = i+1
                              # j = j-1
    bge $a1, $a0, END
    j loop
END:
    li $v0, 10                # syscall code 10 is for exit
    syscall                   # make the syscall
.data 0x10000000
.align 2
Array1: .word 2 5 6 7 12 16 25 27
Size: .word 8
```

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

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.

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 j is (size-1) but 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.

```
# 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 sum of the $t1 and $t2.
# $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.
add $t2, $t0, $t1 # compute the sum.

## Print out $t2.
move $a0, $t2 # move the number to print into $a0.
li $v0, 1 # load syscall print_int into $v0.
syscall # make the syscall.
li $v0, 10 # syscall code 10 is for exit.
syscall # make the syscall.
```

Assignment 4

1- Type and execute the code in the example program, record the value of each register.

2- Discuss the importance and value of register $v0 in using **`syscall`**.

3- Write a simple program in assembly language that can read and print a string. **Hint:** Look at **"MIPS System Call in SPIM.htm"** file to see how to work with system call in SPIM.

Assignment 5

Write a program in assembly language and calculate the product of N integers given by user, and N is also given by user.

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`**.
- 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
add $d, $s, $t	**$d = $s + $t; advance_pc (4);**
addi $t, $s, imm	**$t = $s + imm; advance_pc (4);**
sub $d, $s, $t	**$d = $s - $t; advance_pc (4);**
div $s, $t	**$LO = $s / $t; $HI = $s % $t; advance_pc (4);**
mult $s, $t	**`$LO = $s * $t; advance_pc (4);`**

Assignment 6

Let $a[0] = 0$, $a[1] = 1$, $a[2] = 1$ and $a[n] = (a[n-1] + a[n-2] + a[n-3])$ for $n \geq 3$
Write a program to calculate $a[9]$ and provide the output.

Assignment 7

Translate the following C code to assembly. Assume that the variables **`i, j`** and **`k`** are assigned to registers **`$s1, $s2`** and **`$s3`**, respectively. Besides, assume that the base address of arrays **`A`** and **`B`** are in registers **`$s4`** and **`$s5`**, respectively. Assume that the elements of the arrays A and B are 4-byte words.

```
B[k] = (A[i] + A[j]) * A[j]
```

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

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 8

Explain what the main function of **ex5.asm** is. How is this function realized? What is the condition for the input number to give the correct answer?

```
#ex5.asm

.data 0x10000000

    ask: .asciiz "\nEnter a number:"

    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

    addi $t1, $0, 0 # i = 0

    addi $t2, $0, 189    # ans = 189, Starting case (n=0)

    li $t3, 2
```

```
loop:
    beq $t1, $t0, END    # from i = 0 to n-1 (n times)
    addi $t1, $t1, 1     # i = i+1
    div $t2, $t3         # LO = ans / 2, The result is stored on LO
    mflo $t2             # ans = LO, Copies the content of LO to t2
    j loop
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

Lab report

Write a proper report including your codes, results (snapshot of output) and the conclusion of assignments and convert it to **pdf** format. Please also attach the **code** files (*.s,*.asm) to **Sakai** together with the report. Each lab report is **due** before the start of the next lab. Please include your name and Student ID in both report and the code.

References

1. Patterson and Hennessy, "Computer Organization and Design: The Hardware / Software interface", 5th 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/