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

Computer Architecture and Assembly Language Lab

Spring 2017

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*

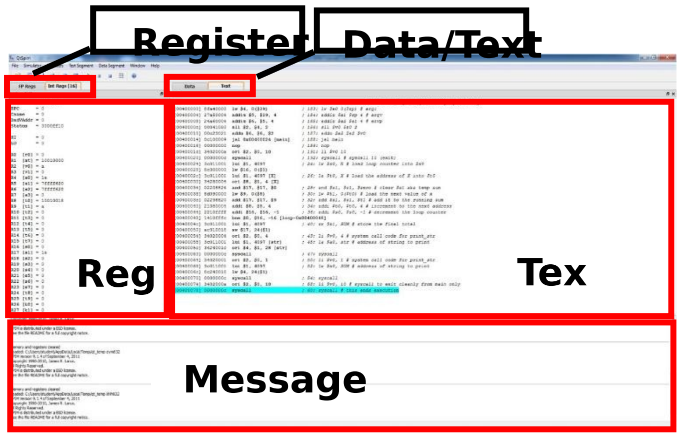
Spim is a simulator that runs MIPS32 programs. It's been around for more than 20 years. *QtSpim* is the newest version of *Spim*. It runs on Microsoft Windows, Mac OS X, and Linux with the same source code and the same user interface. It can do a context and syntax check while loading an assembly program. 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

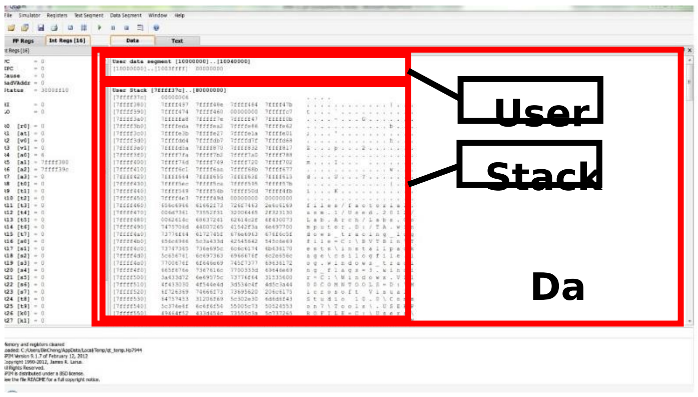
QtSpim is installed on the computers in EE 103. To acquire this software for your personal computer, you can download it from **http://sourceforge.net/projects/spimsimulator/files/**

Get Started

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

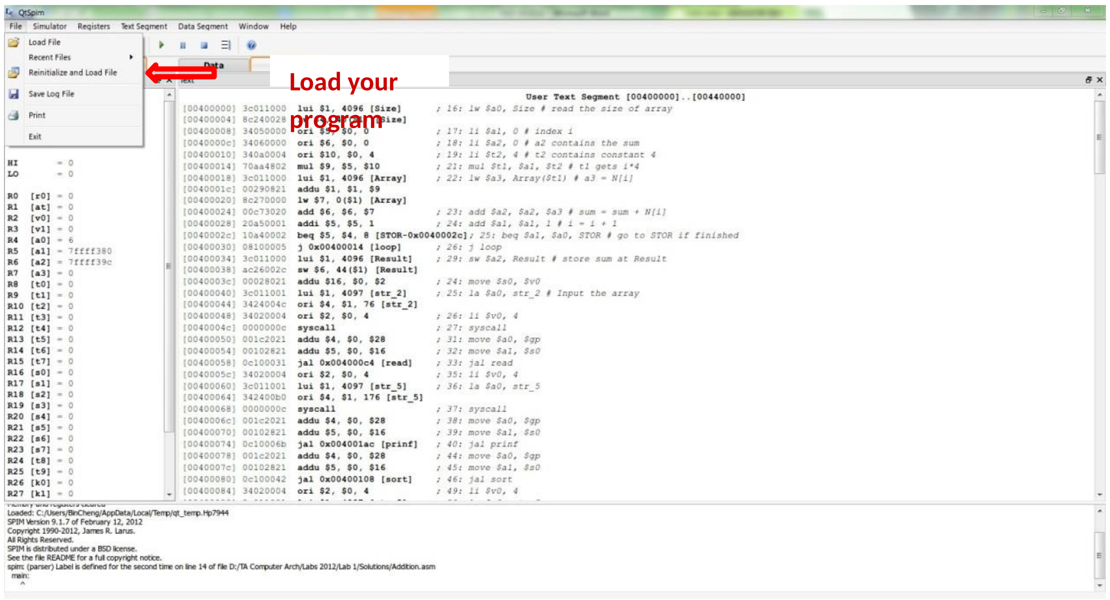


1. The Register tabs display the content of all registers. You can view the registers as binary, hexadecimal or decimal. The processor Program Counter is also shown here.
2. The Text Tab displays the MIPS instructions loaded into memory to be executed. The text window has five columns. From left to right,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



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



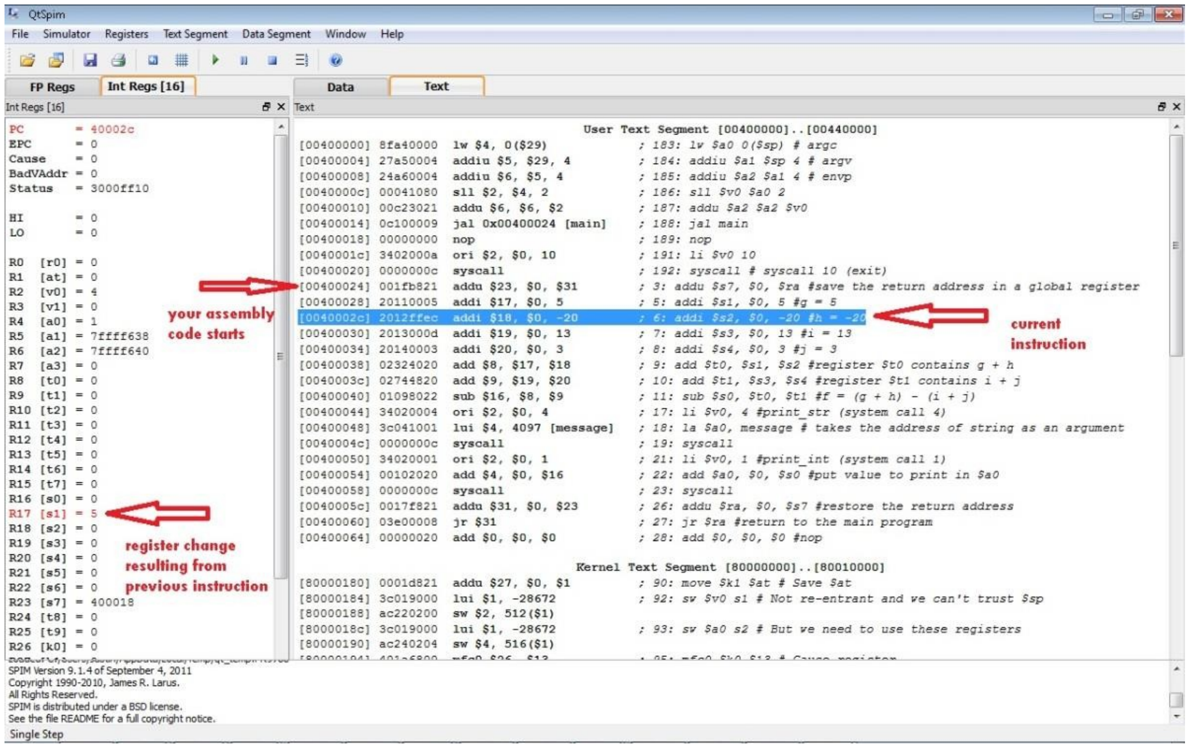
```
HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 6
R5  [a1] = 7ffff380
R6  [a2] = 7ffff39c
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 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                                  User Text Segment [00400000]..[00440000]
[00400000] 3c011000  lui $1, 4096 [Size]       ; 16: lw $a0, Size # read the size of array
[00400004] 8c240028  [illegible] [Size]
[00400008] 34050000  ori $5, $0, 0             ; 17: li $a1, 0 # index i
[0040000c] 34060000  ori $6, $0, 0             ; 18: li $a2, 0 # a2 contains the sum
[00400010] 340a0004  ori $10, $0, 4            ; 19: li $t2, 4 # t2 contains constant 4
[00400014] 70aa4802  mul $9, $5, $10           ; 21: mul $t1, $a1, $t2 # t1 gets i*4
[00400018] 3c011000  lui $1, 4096 [Array]      ; 22: lw $a3, Array($t1) # a3 = N[i]
[0040001c] 00290821  addu $1, $1, $9
[00400020] 8c270000  lw $7, 0($1) [Array]
[00400024] 00c73020  add $6, $6, $7            ; 23: add $a2, $a2, $a3 # sum = sum + N[i]
[00400028] 20a50001  addi $5, $5, 1            ; 24: add $a1, $a1, 1 # i = i + 1
[0040002c] 10a40002  beq $5, $4, 8 [STOR-0x0040002c]; 25: beq $a1, $a0, STOR # go to STOR if finished
[00400030] 08100005  j 0x00400014 [loop]       ; 26: j loop
[00400034] 3c011000  lui $1, 4096 [Result]     ; 29: sw $a2, Result # store sum at Result
[00400038] ac26002c  sw $6, 44($1) [Result]
[0040003c] 00028021  addu $16, $0, $2          ; 24: move $s0, $v0
[00400040] 3c011001  lui $1, 4097 [str_2]      ; 25: la $a0, str_2 # Input the array
[00400044] 3424004c  ori $4, $1, 76 [str_2]
[00400048] 34020004  ori $2, $0, 4             ; 26: li $v0, 4
[0040004c] 0000000c  syscall                   ; 27: syscall
[00400050] 001c2021  addu $4, $0, $28          ; 31: move $a0, $gp
[00400054] 00102821  addu $5, $0, $16          ; 32: move $a1, $s0
[00400058] 0c100031  jal 0x004000c4 [read]     ; 33: jal read
[0040005c] 34020004  ori $2, $0, 4             ; 35: li $v0, 4
[00400060] 3c011001  lui $1, 4097 [str_5]      ; 36: la $a0, str_5
[00400064] 342400b0  ori $4, $1, 176 [str_5]
[00400068] 0000000c  syscall                   ; 37: syscall
[0040006c] 001c2021  addu $4, $0, $28          ; 38: move $a0, $gp
[00400070] 00102821  addu $5, $0, $16          ; 39: move $a1, $s0
[00400074] 0c10006b  jal 0x004001ac [prinf]    ; 40: jal prinf
[00400078] 001c2021  addu $4, $0, $28          ; 44: move $a0, $gp
[0040007c] 00102821  addu $5, $0, $16          ; 45: move $a1, $s0
[00400080] 0c100042  jal 0x00400108 [sort]     ; 46: jal sort
[00400084] 34020004  ori $2, $0, 4             ; 49: li $v0, 4
```

```
Memory and registers cleared
Loaded: C:/Users/BinCheng/AppData/Local/Temp/qt_temp.Hp7944
SPIM Version 9.1.7 of February 12, 2012
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
spim: (parser) Label is defined for the second time on line 14 of file D:/TA Computer Arch/Labs 2012/Lab 1/Solutions/Addition.asm
  main:
    ^
```

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

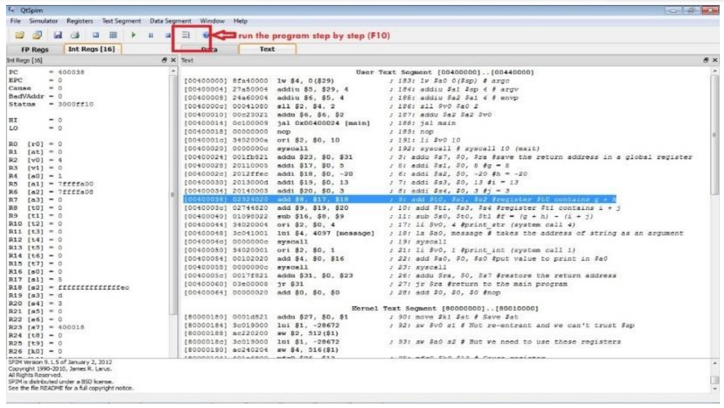

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
Text
your assembly code starts
current instruction
register change resulting from previous instruction
Single Step

```
PC       = 40002c
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff638
R6  [a2] = 7ffff640
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 [s0] = 0
R17 [s1] = 5
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 400018
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 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] 00041080  sll $2, $4, 2             ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2           ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [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)
[00400024] 001fb821  addu $23, $0, $31         ; 3: addu $s7, $0, $ra #save the return address in a global register
[00400028] 20110005  addi $17, $0, 5           ; 5: addi $s1, $0, 5 #g = 5
[0040002c] 2012ffec  addi $18, $0, -20         ; 6: addi $s2, $0, -20 #h = -20
[00400030] 2013000d  addi $19, $0, 13          ; 7: addi $s3, $0, 13 #i = 13
[00400034] 20140003  addi $20, $0, 3           ; 8: addi $s4, $0, 3 #j = 3
[00400038] 02324020  add $8, $17, $18          ; 9: add $t0, $s1, $s2 #register $t0 contains g + h
[0040003c] 02744820  add $9, $19, $20          ; 10: add $t1, $s3, $s4 #register $t1 contains i + j
[00400040] 01098022  sub $16, $8, $9           ; 11: sub $s0, $t0, $t1 #f = (g + h) - (i + j)
[00400044] 34020004  ori $2, $0, 4             ; 17: li $v0, 4 #print_str (system call 4)
[00400048] 3c041001  lui $4, 4097 [message]    ; 18: la $a0, message # takes the address of string as an argument
[0040004c] 0000000c  syscall                   ; 19: syscall
[00400050] 34020001  ori $2, $0, 1             ; 21: li $v0, 1 #print_int (system call 1)
[00400054] 00102020  add $4, $0, $16           ; 22: add $a0, $0, $s0 #put value to print in $a0
[00400058] 0000000c  syscall                   ; 23: syscall
[0040005c] 0017f821  addu $31, $0, $23         ; 26: addu $ra, $0, $s7 #restore the return address
[00400060] 03e00008  jr $31                    ; 27: jr $ra #return to the main program
[00400064] 00000020  add $0, $0, $0            ; 28: add $0, $0, $0 #nop

                              Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1          ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672            ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672            ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
```

```
SPIM Version 9.1.4 of September 4, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
```

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.`**


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
run the program step by step (F10)
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
Text

```
PC       = 400038
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffffa00
R6  [a2] = 7ffffa08
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 [s0] = 0
R17 [s1] = 5
R18 [s2] = ffffffffffffffec
R19 [s3] = d
R20 [s4] = 3
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 400018
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 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] 00041080  sll $2, $4, 2             ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2           ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [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)
[00400024] 001fb821  addu $23, $0, $31         ; 3: addu $s7, $0, $ra #save the return address in a global register
[00400028] 20110005  addi $17, $0, 5           ; 5: addi $s1, $0, 5 #g = 5
[0040002c] 2012ffec  addi $18, $0, -20         ; 6: addi $s2, $0, -20 #h = -20
[00400030] 2013000d  addi $19, $0, 13          ; 7: addi $s3, $0, 13 #i = 13
[00400034] 20140003  addi $20, $0, 3           ; 8: addi $s4, $0, 3 #j = 3
[00400038] 02324020  add $8, $17, $18          ; 9: add $t0, $s1, $s2 #register $t0 contains g + h
[0040003c] 02744820  add $9, $19, $20          ; 10: add $t1, $s3, $s4 #register $t1 contains i + j
[00400040] 01098022  sub $16, $8, $9           ; 11: sub $s0, $t0, $t1 #f = (g + h) - (i + j)
[00400044] 34020004  ori $2, $0, 4             ; 17: li $v0, 4 #print_str (system call 4)
[00400048] 3c041001  lui $4, 4097 [message]    ; 18: la $a0, message # takes the address of string as an argument
[0040004c] 0000000c  syscall                   ; 19: syscall
[00400050] 34020001  ori $2, $0, 1             ; 21: li $v0, 1 #print_int (system call 1)
[00400054] 00102020  add $4, $0, $16           ; 22: add $a0, $0, $s0 #put value to print in $a0
[00400058] 0000000c  syscall                   ; 23: syscall
[0040005c] 0017f821  addu $31, $0, $23         ; 26: addu $ra, $0, $s7 #restore the return address
[00400060] 03e00008  jr $31                    ; 27: jr $ra #return to the main program
[00400064] 00000020  add $0, $0, $0            ; 28: add $0, $0, $0 #nop

                              Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1          ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672            ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672            ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
```

```
SPIM Version 9.1.5 of January 2, 2012
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
```

Exercises

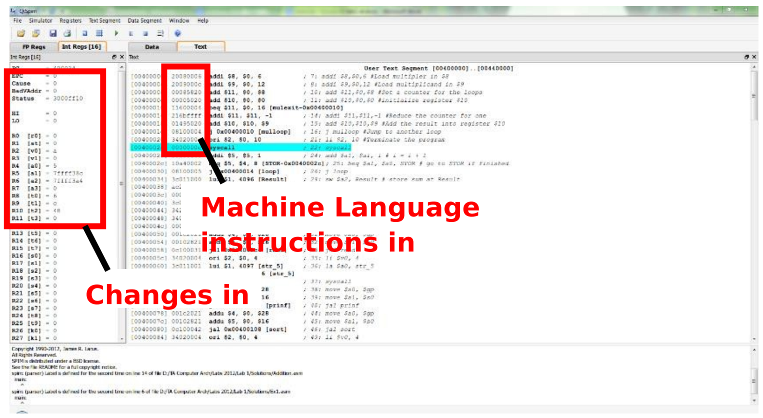
In these exercises, you will use *QtSpim* to practice some basic assembly language operations, such as memory-transfer, reading and printing, and simple arithmetic operations.

Part 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 sum of the numbers less or equal than a number that is stored in memory and the result is stored in a register.

```
 # Exercise1 is used in assignments 1 and 2
.text 0x00400000
       .globl main


main:


      lw $10, Number1($0)
      ori $11, $0, 1
      ori $9, $0, 1
loop:
      bge $11, $10, exit
      add $9, $10, $9
      addi $10, $10, -1
      j loop
exit:
#the computation 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
```

After executing this program, you can see:


QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Machine Language instructions in
Changes in

```
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 5
R5  [a1] = 7ffff38c
R6  [a2] = 7ffff3a4
R7  [a3] = 0
R8  [t0] = 6
R9  [t1] = c
R10 [t2] = 48
R11 [t3] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                              User Text Segment [00400000]..[00440000]
[0040000  20080006 addi $8, $0, 6         ; 7: addi $8,$0,6 #Load multipler in $8
[0040000  2009000c addi $9, $0, 12        ; 8: addi $9,$0,12 #Load multiplicand in $9
[0040000  00085820 add $11, $0, $8        ; 10: add $11,$0,$8 #Set a counter for the loops
[0040000  00005020 add $10, $0, $0        ; 11: add $10,$0,$0 #initialize register $10
[0040001  11600004 beq $11, $0, 16 [mulexit-0x00400010]
[0040001  216bffff addi $11, $11, -1      ; 14: addi $11,$11,-1 #Reduce the counter for one
[0040001  01495020 add $10, $10, $9       ; 15: add $10,$10,$9 #Add the result into register $10
[0040001  08100004 j 0x00400010 [mulloop] ; 16: j mulloop #Jump to another loop
[0040002  3402000  ori $2, $0, 10         ; 21: li $2, 10 #Terminate the program
[0040002  0000000c syscall                ; 22: syscall
[0040002           addi $5, $5, 1         ; 24: add $a1, $a1, 1 # i = i + 1
[0040002c] 10a40002 beq $5, $4, 8 [STOR-0x0040002c]; 25: beq $a1, $a0, STOR # go to STOR if finished
[00400030] 08100005 j 0x00400014 [loop]   ; 26: j loop
[00400034] 3c011000 lui $1, 4096 [Result] ; 29: sw $a2, Result # store sum at Result
[0040005c] 34020004 ori $2, $0, 4         ; 35: li $v0, 4
[00400060] 3c011001 lui $1, 4097 [str_5]  ; 36: la $a0, str_5
[00400078] 001c2021 addu $4, $0, $28      ; 38: move $a0, $gp
[0040007c] 00102821 addu $5, $0, $16      ; 39: move $a1, $s0
[00400080] 0c100042 jal 0x00400108 [sort] ; 40: jal sort
[00400084] 34020004 ori $2, $0, 4         ; 43: li $v0, 4
```

```
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
spim: (parser) Label is defined for the second time on line 14 of file
  main:
    ^
spim: (parser) Label is defined for the second time on line 6 of file
  main:
    ^
```

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

Instructions	**op**	**rs**	**rt**	**rd**	**shamt**	**funct**
add $9,$10,$9						
	op	**rs**	**rt**	**constant**		
addi $10,$10,-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.

Part 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.

```
#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 zero
loop:
mul $t1, $a1, $t2           # t1 gets i*4
                            # a3 = Array[i]
                            # a4 = Array2[i]
                            # result = Array2[i] - 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
Array2: .word 5 8 9 10 15 19
Size: .word 6
```

In order to understand the operation of memory transfer instructions (`lw` and `sw`) you will have to complete Assignment 3 which is based on the above assembly program, whose name is “**`Ex2.asm`**”. Type and execute the program as described in Part 1.

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. Record the registers at the end of the program.
3. Record the data segment of this program and check the output.

Part 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`**.

3- Write a program in mips assembly that will prompt the user to enter his/her name and then print it out.

```
# 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.


sub $t2, $t0, $t1 # compute the difference.

## 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.
# end of ex2.asm.
```

Part 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 5

Write a program in Mips assembly that gets four integers from the user and calculates the Sum of differences of all pairs created with these numbers.

Part 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 6

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 7

Change the main function of **`Ex5.asm`** so that it provides the power of 3 that corresponds to the user's input.

Lab report

Write a proper report that includes your codes, results and the conclusion of assignments. The program files must be included in the report AND in separate .s files.
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 AND follow the given Template for the cover page. ALL reports must be in pdf format

```
#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 = $vo, Move the user input to t0
            addi $t1, $0, 0      # i = 0
            addi $t2, $0, 1   # ans = 1, Starting case (n=0) is 1
            li $t3, 3                   #we store 3 in $t3
loop:
   beq $t0, 1, END  #i
            div $t0, $t3                #divide $t0 with $t3
            mfhi $t1             #store the reminder to $t1
            beq $t1, $0, LABEL_1   #if the reminder is 0 then we go to LABEL_1
            addi $t0, $t0, -1    #we reduce $t0
            j loop                              # go back to loop

LABEL_1:  mult $t2, $t0
                        mflo $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
```

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/