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

Computer Architecture and Assembly Language Lab

Spring 2018

Lab 1

Machine Language Instructions, introduction to SPIM 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 and you will be able to write a simple assembly program.

Preparation

1. Please read the SPIM instructions in your lab resources folder. This knowledge is required in this lab. We will use *QtSpim*, a 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 is also in Sakai/Resources.

Introduction to *QtSpim*

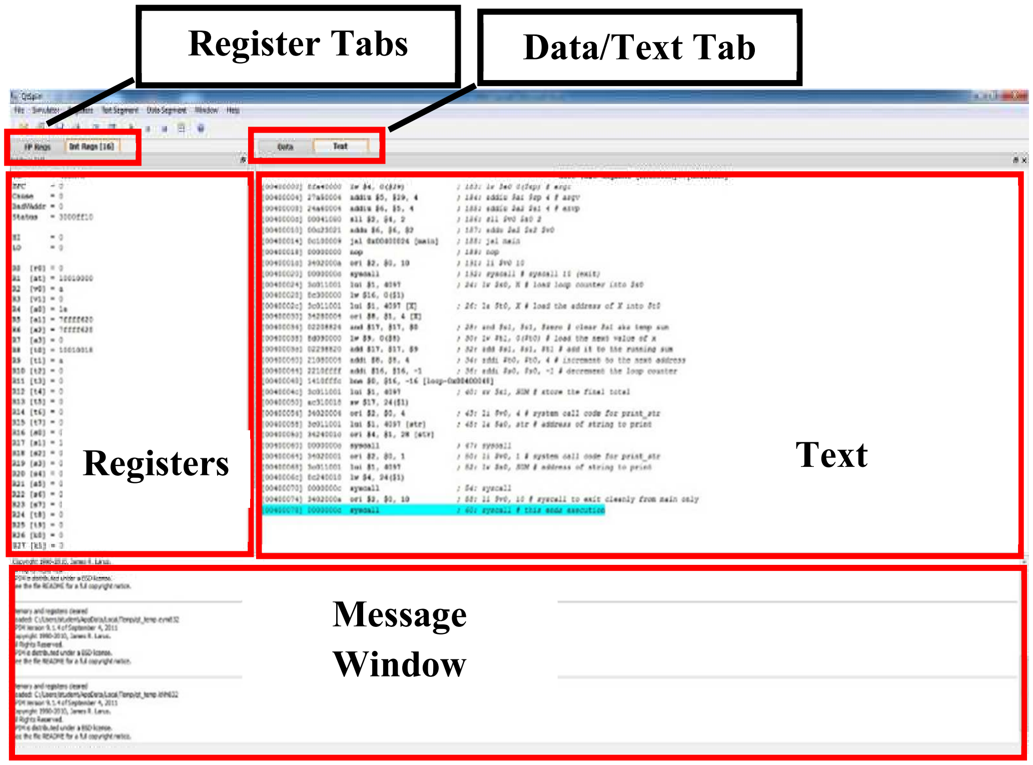
Spim is a simulator that runs MIPS32 programs. It's been around for more than 20 years. *QtSpim* is the newest version of *Spim*. It runs on Microsoft Windows, Mac OS X, and Linux with the same source code and the same user interface. It can do a context and syntax check while loading an assembly program. In addition, *QtSpim* 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 has been installed on the new computers in EE 103 before your first lab started. To acquire this software for your personal computer, you should download it from http://sourceforge.net/projects/spimsimulator/files/

➢ Get Started

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



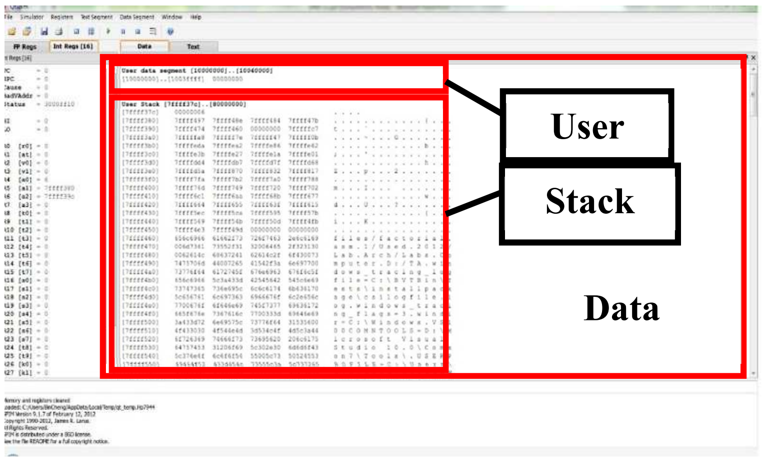
```
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 10010000
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 1a
R5  [a1] = 7ffff620
R6  [a2] = 7ffff628
R7  [a3] = 0
R8  [t0] = 10010018
R9  [t1] = a
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 1
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
```

```
[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] 3c011001  lui $1, 4097             ; 24: lw $s0, N # load loop counter into $s0
[00400028] 8c300000  lw $16, 0($1)
[0040002c] 3c011001  lui $1, 4097 [X]         ; 26: la $t0, X # load the address of X into $t0
[00400030] 34280004  ori $8, $1, 4 [X]
[00400034] 02208824  and $17, $17, $0         ; 28: and $s1, $s1, $zero # clear $s1 aka temp sum
[00400038] 8d090000  lw $9, 0($8)             ; 30: lw $t1, 0($t0) # load the next value of x
[0040003c] 02298820  add $17, $17, $9         ; 32: add $s1, $s1, $t1 # add it to the running sum
[00400040] 21080004  addi $8, $8, 4           ; 34: addi $t0, $t0, 4 # increment to the next address
[00400044] 2210ffff  addi $16, $16, -1        ; 36: addi $s0, $s0, -1 # decrement the loop counter
[00400048] 1410fffc  bne $0, $16, -16 [loop-0x00400048]
[0040004c] 3c011001  lui $1, 4097             ; 40: sw $s1, SUM # store the final total
[00400050] ac310018  sw $17, 24($1)
[00400054] 34020004  ori $2, $0, 4            ; 43: li $v0, 4 # system call code for print_str
[00400058] 3c011001  lui $1, 4097 [str]       ; 45: la $a0, str # address of string to print
[0040005c] 34240010  ori $4, $1, 28 [str]
[00400060] 0000000c  syscall                  ; 47: syscall
[00400064] 34020001  ori $2, $0, 1            ; 50: li $v0, 1 # system call code for print_str
[00400068] 3c011001  lui $1, 4097             ; 52: lw $a0, SUM # address of string to print
[0040006c] 8c240018  lw $4, 24($1)
[00400070] 0000000c  syscall                  ; 54: syscall
[00400074] 3402000a  ori $2, $0, 10           ; 58: li $v0, 10 # syscall to exit cleanly from main only
[00400078] 0000000c  syscall                  ; 60: syscall # this ends execution
```

```
Copyright 1990-2010, James R. Larus.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
-------------------------------------------------
Memory and registers cleared
Loaded: C:\Users\student\AppData\Local\Temp\qt_temp.[illegible]
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.
-------------------------------------------------
Memory and registers cleared
Loaded: C:\Users\student\AppData\Local\Temp\qt_temp.[illegible]
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.
```

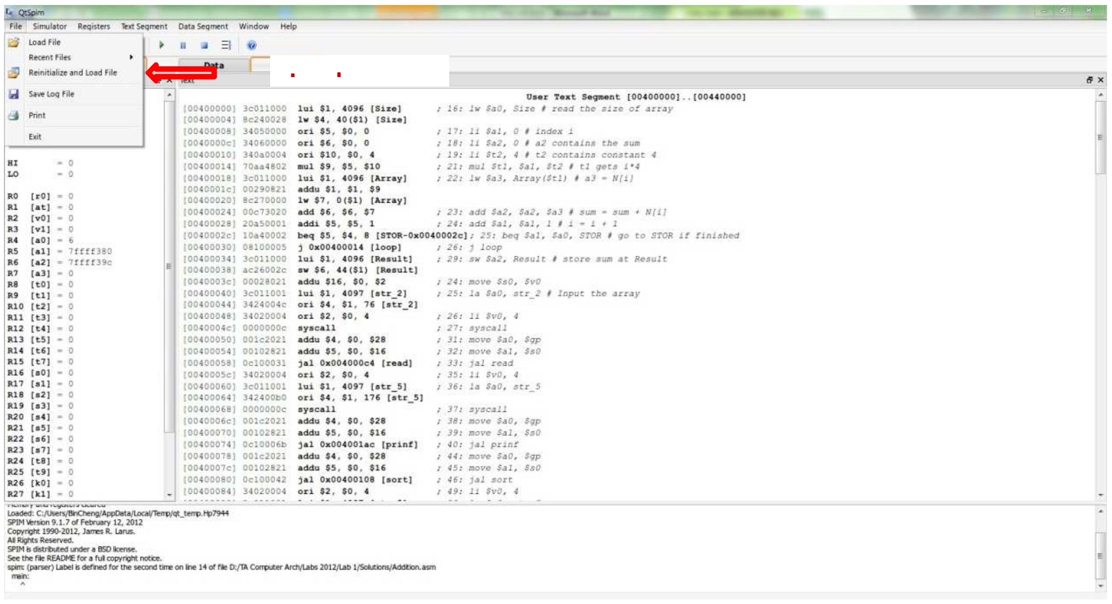
1. The Register tabs display the content of all registers. You can view the registers as binary, hexadecimal or decimal. The processor Program Counter is also shown here.
2. The Text Tab displays the MIPS instructions loaded into memory waiting to be executed. The text window has five columns. From left 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 code that you wrote,
   - Any comments you made in your code are also displayed.

3. The Data tab displays memory addresses and their values in the data and stack segments of the main 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

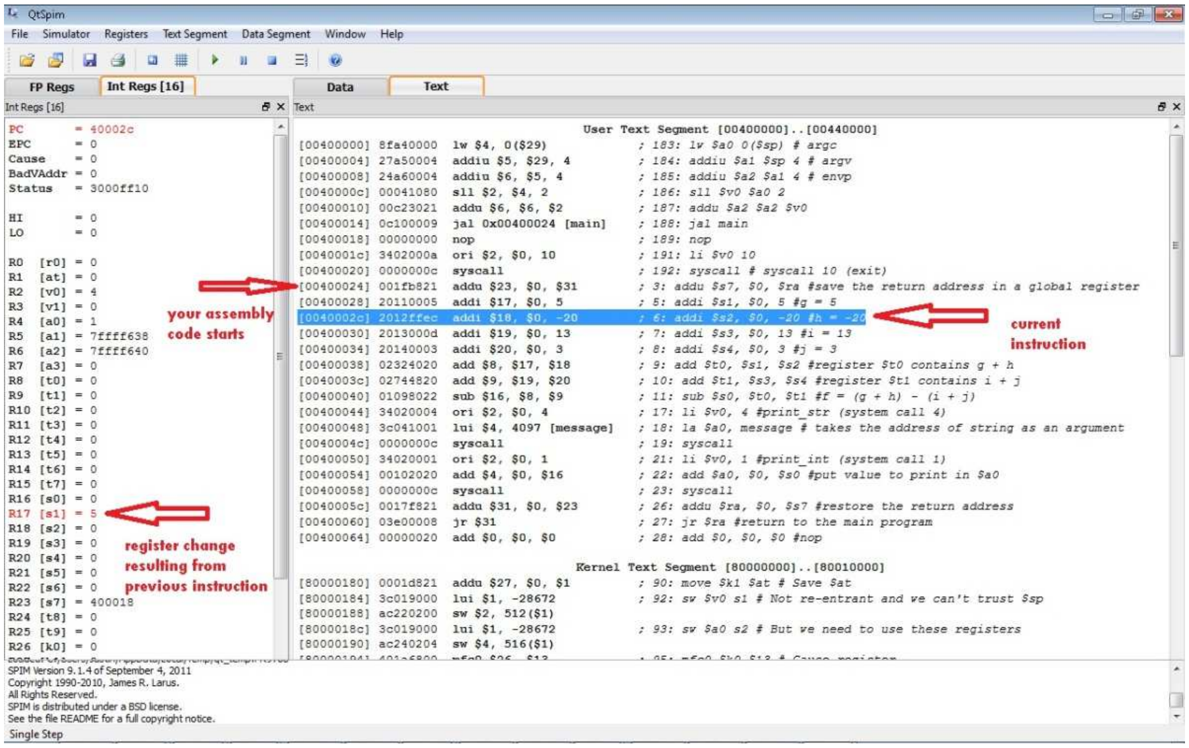


```
HI         = 0
LO         = 0

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  lw $4, 40($1) [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
```

```
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 (a small memory unit in the processor where intermediate results are stored) will be reflected in the *QtSpim* window.
4.

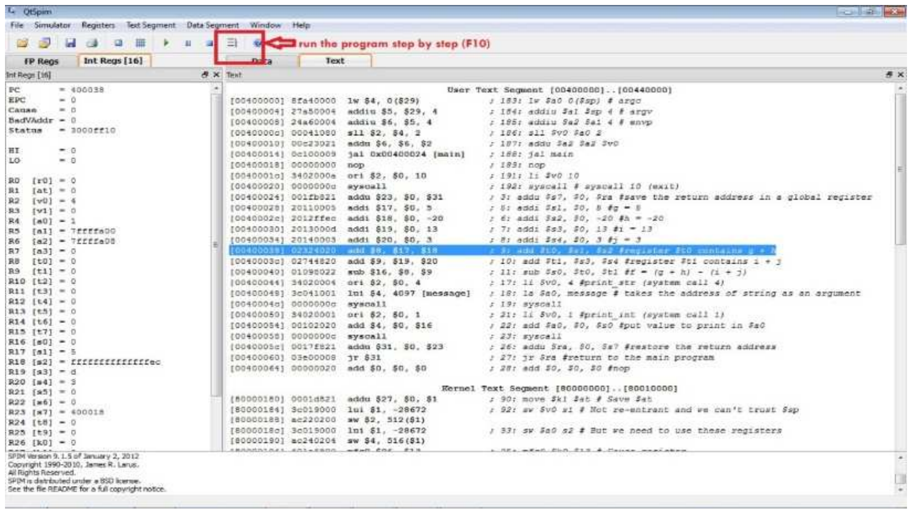

QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
your assembly code starts
current instruction
register change resulting from previous instruction
Single Step

```
Int Regs [16]
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
```

```
Text
                          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]
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
```

```
Text
                          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

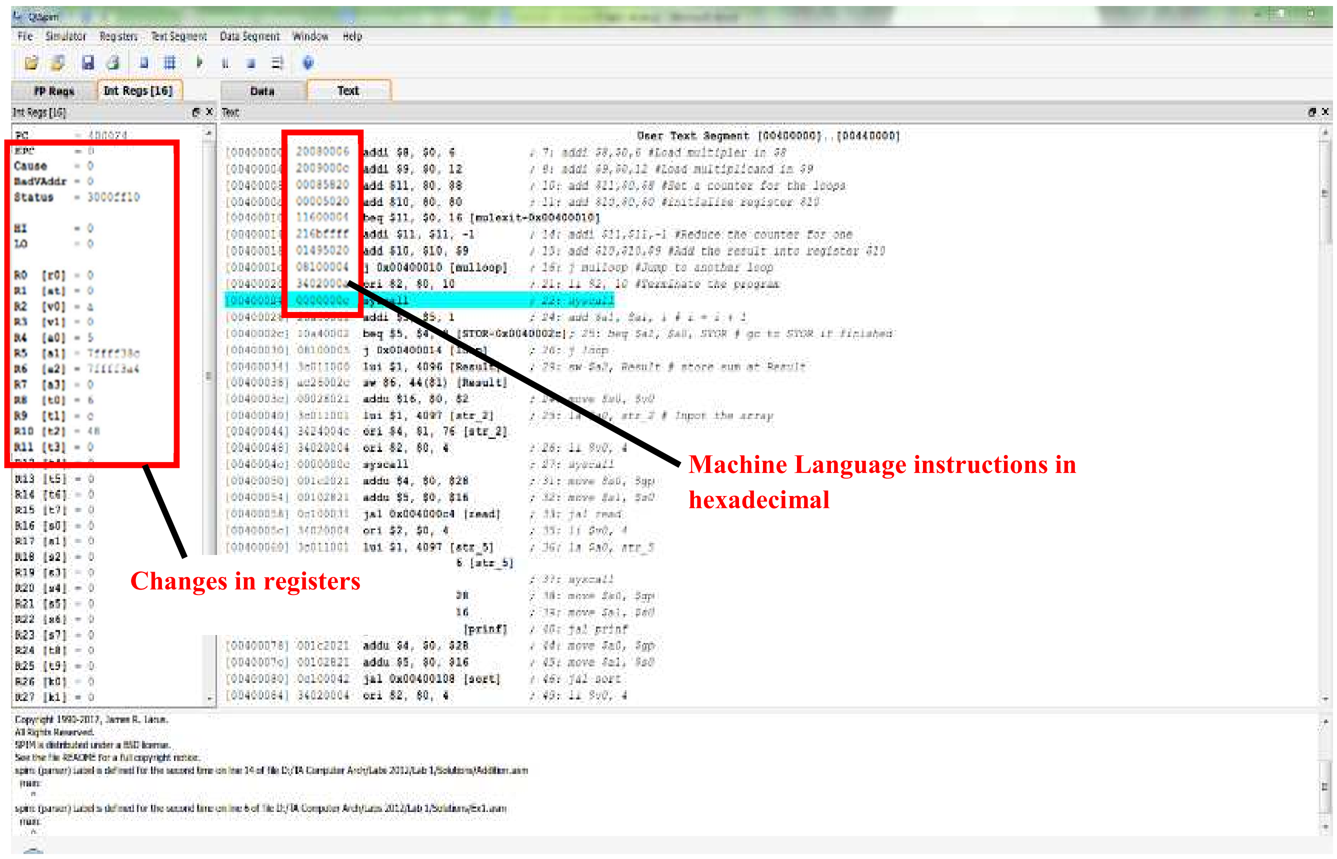
In these exercises, you will use *QtSpim* to practice some basic assembly language operations, such as memory-transfer, reading and printing, and simple arithmetic operations.

Practice using the simulator

In order to understand the operation of *QtSpim*, create a file named "**program1.s**" and execute

```
 # program1 is used in assignments 1 and 2
.text 0x00400000

       .globl main


main:


      lw $10, Number1($0) # this instruction loads a word from RAM

      ori $11, $0, 1 #OR immediate instruction

      ori $9, $0, 1

#compute the factorial of Number ($10)!

factloop:

      bge $11, $10, factexit #branch grater or equal instruction

      mul $9, $10, $9 #multiply contents of registers $10 and $9

      sub $10, $10, 1 #subtract 1 from contents of register $10 and place
result in reg $10

      j factloop #jump to label factloop

factexit:

#the computation of the factorial number has finished at this point

#Is the result in register $9 correct? The result in $9 is in hex form

  li $2, 10 #load immediate number 10 in register $2

  syscall


    .data 0x10000000

    .align 2

Number1: .word 18
```

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.


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

```
Int Regs [16]
PC      = 400024
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
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
```

```
Text
                                        User Text Segment [00400000]..[00440000]
[00400000] 20080006  addi $8, $0, 6          ; 7: addi $8,$0,6 #Load multiplier in $8
[00400004] 2009000c  addi $9, $0, 12         ; 8: addi $9,$0,12 #Load multiplicand in $9
[00400008] 00085820  add $11, $0, $8         ; 10: add $11,$0,$8 #Set a counter for the loops
[0040000c] 00005020  add $10, $0, $0         ; 11: add $10,$0,$0 #initialize register $10
[00400010] 11600004  beq $11, $0, 16 [mulexit-0x00400010]
[00400014] 216bffff  addi $11, $11, -1       ; 14: addi $11,$11,-1 #Reduce the counter for one
[00400018] 01495020  add $10, $10, $9        ; 15: add $10,$10,$9 #Add the result into register $10
[0040001c] 08100004  j 0x00400010 [mulloop]  ; 16: j mulloop #Jump to another loop
[00400020] 3402000a  ori $2, $0, 10          ; 21: li $2, 10 #Terminate the program
[00400024] 0000000c  syscall                 ; 22: syscall
[00400028] [illegible]  addi $5, $5, 1       ; 24: add $a1, $a1, 1 # i = i + 1
[0040002c] 10a40003  beq $5, $4, 4 [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
                                  6 [str_5]
                                             ; 37: syscall
                                  28         ; 38: move $a0, $gp
                                  16         ; 39: move $a1, $s0
                                   [prinf]   ; 40: jal print
[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
```

```
Copyright 1990-2017, 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:
     ^
spim: (parser) Label is defined for the second time on line 6 of file D:/TA Computer Arch/Labs 2012/Lab 1/Solutions/Ex1.asm
  main:
     ^
```

After executing this program, you can see:
The machine language instructions are divided into fields.

- **`op`**: Basic operation of the instruction (indicated the type of instruction R-type, I-type, J-type etc.)
- **`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 type of operation specified in the op field (example op field indicated R-type operation, while **`funct`** field indicates whether it is an addition **`add`** or a subtraction **`sub`**

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

Note that all instructions are 32 bits wide in MIPS so the sum of bit fields always adds up to 32.

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

You can find description for every instruction in the MIPS Instruction reference document in the Useful Resources Folder in you lab Sakai page.

Assignment 1

For the instructions below, indicate what the corresponding Machine Language Instructions in Binary codes are in the table below (hint – you need to convert 8 to its binary equivalent using 16 bits, of which the actual bits for 8 are the least significant ones and the rest are 0s)

	op	**rs**	**rt**	**constant**
`addi $3, $0, 15`				

	op	**rs**	**rt**	**rd**	**shamt**	**funct**
`mul $8,$8,$12`						

(Hint Register file addresses are 5 bits; **`op`** field for all R-type instructions is 0**)**

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 **`$destination`** with the data word located in RAM at memory address **`offset +$base`**. For example, **`lw $20, 10000 ($0)`** means that 32-bit data in RAM memory at address **`10000+$0`** is stored into register **`$20`**.

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

```
sw $source, offset($base)
```

which stores the contents of **`$source`** register into the RAM memory address **`offset+$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. It represents the number of rows the address given by $2 is incremented to find the location where the word stored in register $10 will be stored at in RAM. This is because RAM in organized as a stack of rows, each raw containing one word.

Assignment 2

Translate the following C code to MIPS. Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3 and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively. Assume that the size of the elements of the arrays A and B are:

a. **1-byte**
b. **4-byte**
   A[8*i] = A[2*i] - B[2*i] ;

Assignment 3

Translate the following MIPS code to C. Assume that the variables f, g, h, i and j are assigned to registers $s0, $s1, $s2, $s3 and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
addi $t0, $s6, 8
add $t1, $s3, $0  #register $0 always holds 32 0s
sw $t1, 12($s7)
lw $t0, 12($s6)
add $s0, $t1, $t0
```

Reading and Printing

SPIM provides a small set of operating system–like services through the system call instruction, named **`syscall.`** To request a service, a program loads the system call code (it can be found in MIPS System calls uploaded in the Useful Resources folder in Sakai) into register **`$v0`** and arguments into register file registers **`$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.

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

.text
main:
## Get first number from the user, store it 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 the 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 register $t1.
```

In this example, the **`syscall`** instruction 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.

Arithmetics

Addition and subtraction are performed on 32 bit integer 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 MIPS 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**
addi $t, $s, imm	**$t = $s + imm**
sub $d, $s, $t	**$d = $s - $t**
div $s, $t	**$LO = $s / $t; $HI = $s % $t**
mult $s, $t	**`$LO = $s * $t`**

Assignment 4

1. **Discuss the importance and value of register $v0 when using syscall**
2. **Write a program in MIPS assembly language that implements the following**
    a. **Asks the user to input an integer between 1 and 20.**
    b. **Report to the user that it will calculate the volume of a cube with side the user input. The message should look like "Calculating cube's volume with side <number> inches"**
    c. **Calculate the cube's volume**
    d. **Terminate with reporting the volume of the cube with an appropriate message.**

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

Programming Exercises

Assignment 5

Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i and j are stored in registers $s0, $s1, $t0 and $t1, respectively. Also assume that register $s2 holds the base address of the array D.

```
for (i=0; i<a; i++)
        for (j=0;j<b;j++)
                D[j] = i**2 + j**2 ;
```

Assignment 6

Write a program that requests three integers , a1, n, and d from the user and then calculates the nth term of an arithmetic progression with initial state a1 and common difference between terms d.

Lab report

Write a proper report using the ECE Lab report template that includes your codes, results and the conclusion of assignments. The report must be uploaded in Sakai in pdf format and the code files must be in .s or .asm format. Please put your name and Student ID at the start of the code.

References

1. Daniel J. Ellard, "MIPS Assembly Language Programming: CS50 Discussion and Project Book", September 1994.

2. "Get Started with SPIM", http://www.cs.washington.edu/education/courses/cse378/05sp/handouts/spimwin.pdf
3. "Arithmetic in MIPS", http://www.cs.iit.edu/~virgil/cs470/Labs/Lab6.pdf
4. "SPIM: A MIPS32 Simulator", http://spimsimulator.sourceforge.net/