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

Computer Architecture and Assembly Language Lab

Spring 2016

Lab 2

Control Flow (including loops and branches), Inputs & Outputs, Making decisions

Goal

In this Laboratory, you will learn how to use flow control, inputs and outputs and how to make decisions in the assembly language. It will include branch and loops. Before, you have learned in lab 1 how to read the input variables and print them as an output. After this lab you will have the knowledge for using loop and branch instructions with MIPS processors.

Preparation

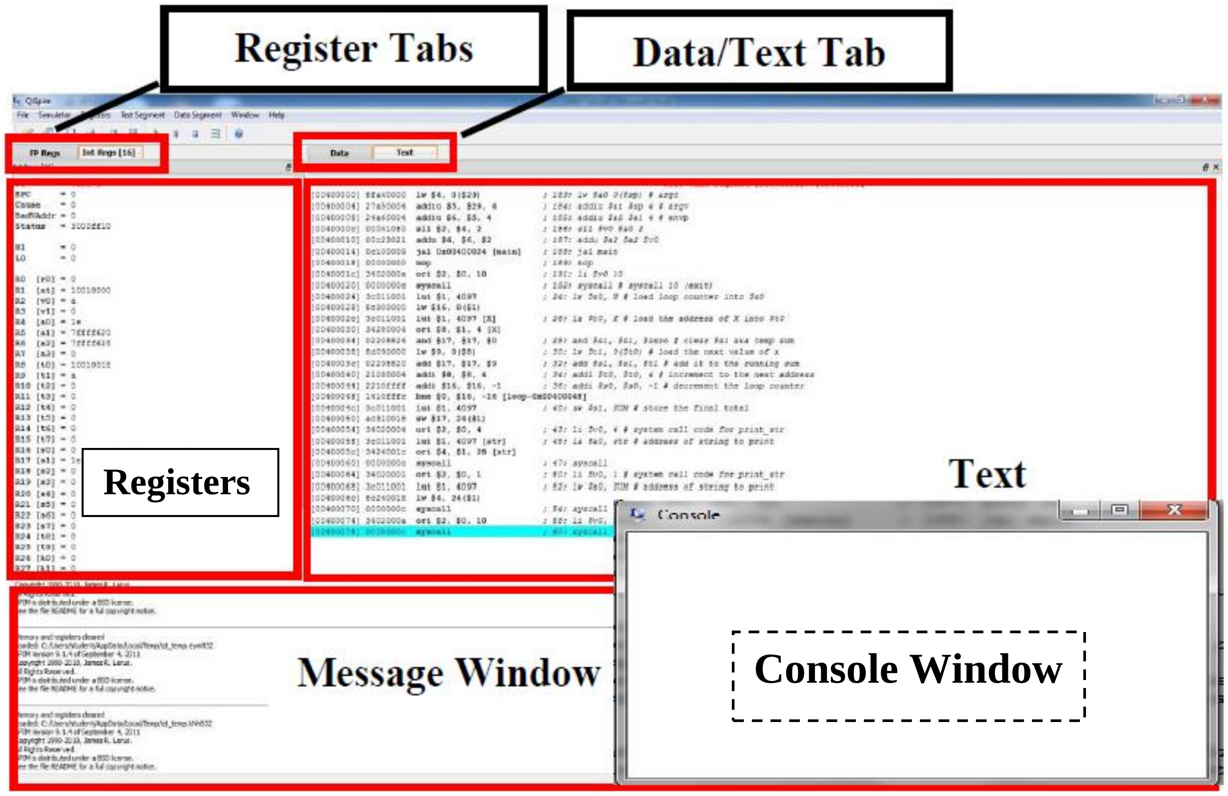
Please carefully read the second chapter of the textbook (Patterson and Hennessy 5th edition) and also the SPIM instructions in Appendix A.10 in the textbook before attending the lab.

You should also look at the tutorial under "resources" on Sakai. Any additional material will be uploaded on the Sakai "resources". The ***QtSpim*** program needed to do the lab exercises has been installed on the ECE 103 computers. To acquire this software for your personal computer, you can download it from http://sourceforge.net/projects/spimsimulator/files .

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

Introduction [1, 2]

As you have seen in the first Lab, ***Qtspim*** is a simulator that runs MIPS 32 programs. By clicking on the icon Qtspim.exe on the screen, two windows will be appear, the first one is ***Qtspim*** window that you can run your program on it and the other one is a console window that displays the result to you (Figure 1).



```
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] = 1e
R5  [a1] = 7ffff620
R6  [a2] = 7ffff618
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] = 1e
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] 1610fffc  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] 3424001c  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,
[00400078] 0000000c  syscall                  ; 60: syscall
```

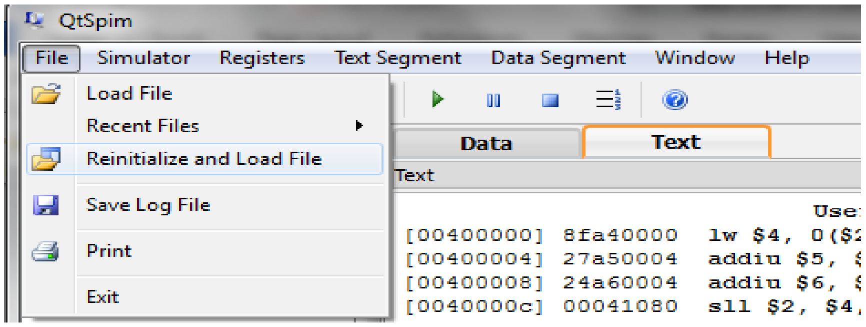
```
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
PIM is distributed under a BSD license.
ee the file README for a full copyright notice.

lemory and registers cleared
oaded: C:/Users/student/AppData/Local/Temp/qt_temp.eym832
PIM Version 9.1.4 of September 4, 2011
opyright 1990-2010, James R. Larus.
ll Rights Reserved.
PIM is distributed under a BSD license.
ee the file README for a full copyright notice.

lemory and registers cleared
oaded: C:/Users/student/AppData/Local/Temp/qt_temp.xhh832
PIM Version 9.1.4 of September 4, 2011
opyright 1990-2010, James R. Larus.
ll Rights Reserved.
PIM is distributed under a BSD license.
ee the file README for a full copyright notice.
```

Figure 1: The ***Qtspim***'s windows

For running your program, you have to do the following:

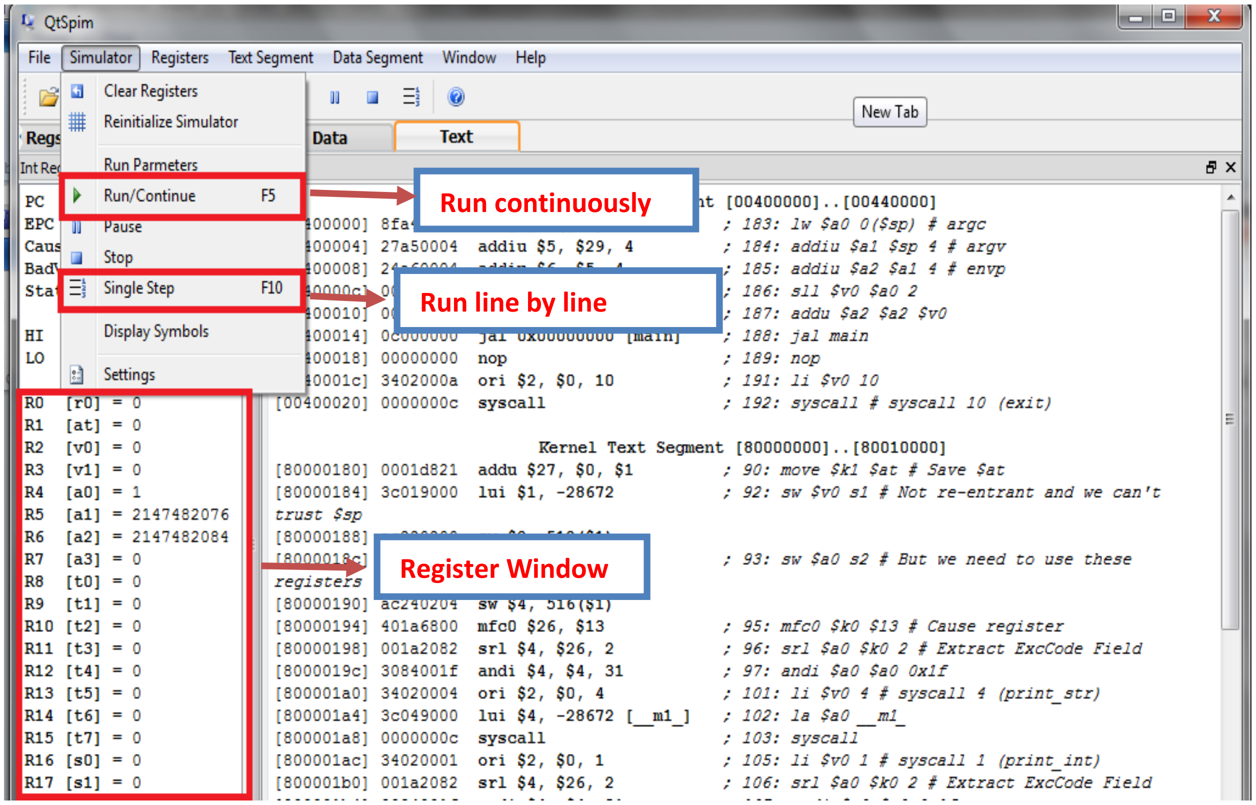
1. Create a new text file, write your codes and save it as **x.asm** or **x.s.**
2. Click on the **File** button and choose **Reinitialize and Load File** and load your program (Figure 2)
3. You can run your program at once using the **Run** icon from **Simulator** on the menu bar or using the function key **F5.** For running your program line by line, you can use the F10 function key**.**
4. You can follow the results by looking at the registers in the left window.(Figure 3)

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



```
Text
                                          Use
[00400000] 8fa40000  lw $4, 0($2
[00400004] 27a50004  addiu $5, $
[00400008] 24a60004  addiu $6, $
[0040000c] 00041080  sll $2, $4,
```

Figure 2: Run the program



```
PC
EPC
Caus
Badv
Sta

HI
LO

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 2147482076
R6  [a2] = 2147482084
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
```

```
                                   nt [00400000]..[00440000]
400000] 8fa                        ; 183: lw $a0 0($sp) # argc
400004] 27a50004  addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
400008] 24                         ; 185: addiu $a2 $a1 4 # envp
40000c] 00                         ; 186: sll $v0 $a0 2
400010] 00                         ; 187: addu $a2 $a2 $v0
400014] 0c000000  jal 0x00000000 [main] ; 188: jal main
400018] 00000000  nop              ; 189: nop
40001c] 3402000a  ori $2, $0, 10   ; 191: li $v0 10
[00400020] 0000000c  syscall       ; 192: syscall # syscall 10 (exit)

            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]
[8000018c]                               ; 93: sw $a0 s2 # But we need to use these
registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13      ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2     ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31    ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4      ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall            ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1      ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2     ; 106: srl $a0 $k0 2 # Extract ExcCode Field
```

Figure 3: Run the program

Before doing Laboratory 2 you must be familiar with the process of reading and printing the integer in SPIM. In SPIM, execution of a program starts from the very top instruction, which we always labeled it as **`main`** for convenience. A label is a symbolic name for an address in memory. In MIPS assembly, a label is a symbol name, followed by a colon. Labels must be the first item on the line. Below, you can see the way we are using labels to show the beginning of the program. It is a simple program that computes the sum of 1 and 2. Copy the below codes to a text file name as **Ex1** and **reinitialize** it and run.

```
# add.asm-- A program that computes the sum of 1 and 2,

# leaving the result in register $t0.

# Registers used:

# t0 - used to hold the result.

# t1 - used to hold the constant 1.


main:

li   $t1, 1                   # load 1 into $t1.

addi  $t0, $t1, 2              # $t0 = $t1 + 2.


# end of add.asm
```

The end of a program is defined in a very different way. Similar to C, where the exit function can be called in order to halt the execution of a program, one way to halt a MIPS program is with something analogous to calling **`exit`** in C. Unlike C, however, if you forget to "call exit" your program will not gracefully exit when it reaches the end of the main function. The way to tell SPIM that it should stop executing your program, and also to do a number of other useful things, is with a special instruction called a **`syscall`**. The **`syscall`** instruction suspends the execution of your program and transfers control to the operating system. Copy the below code into the text file **Ex2** and run to see how you can exit from the program.

```
# add.asm-- A program that computes the sum of 1 and 2,

# leaving the result in register $t0.

# Registers used:

# t0 - used to hold the result.

# t1 - used to hold the constant 1.

# v0 - syscall parameter.


main:                # SPIM starts execution at main.

li    $t1,  1        # load 1 into $t1.

addi   $t0,  $t1, 2   # compute the sum of $t1 and 2, and  put it into
$t0.


li    $v0, 10        # syscall code 10 is for exit.

syscall              # make the syscall.


# end of add.asm
```

We see that the exit process is done by placing a 10 (the number for the exit **`syscall`**) into register **`$v0`** before executing the **`syscall`** instruction.

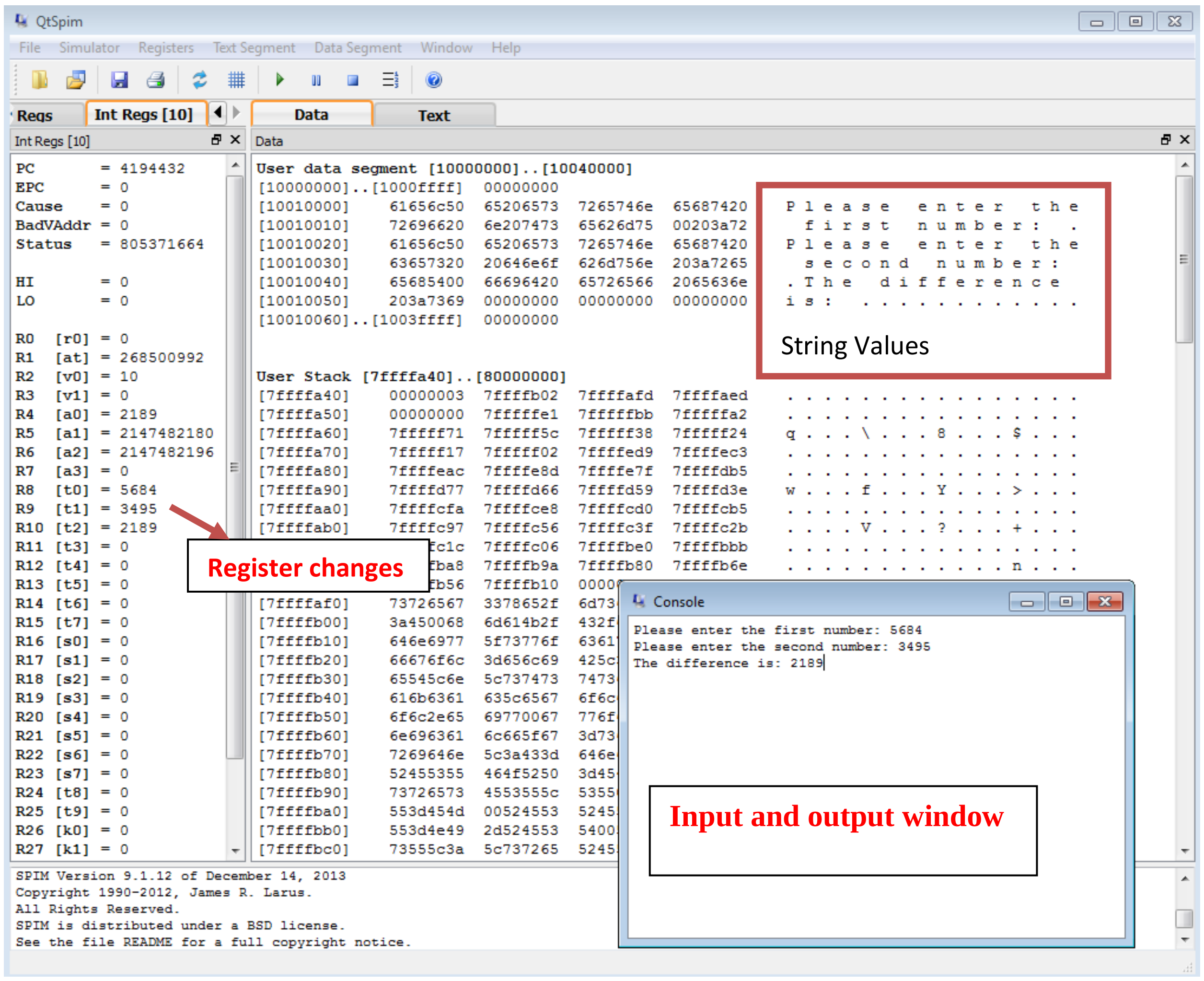
Input and Output

Now we will see how we can do reading and printing integers in MIPS. Both of these operations can be done using **`syscall`** instructions with the proper instruction code (see Appendix at the end of this manual). Use the below codes as Ex3 and run the program. This program shows you how to read two integer numbers and print the difference of the two numbers in the console window.

```
# sub2.asm-- A program that computes and prints the difference

# of two numbers specified at runtime by the user.

# 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 $t2.

# $v0 - syscall parameter and return value.

# $a0 - syscall parameter.


main:

## Get first number from user, put into $t0.

li $v0, 4            # load syscall print string

la $a0, str1         # load address of str1 to register a0

syscall              # make the syscall.


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, 4            # load syscall print string
```

```
la $a0, str2         # load address of str2 to register a0
syscall              # make the syscall.


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.
li $v0, 4            # load syscall print string
la $a0, str3         # load address of str3 to register a0
syscall              #make the call
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.


.data
str1: .asciiz "Please enter the first number: "
str2: .asciiz "Please enter the second number: "
str3: .asciiz "The difference is equal to: "
# end of sub2.asm.
```

As the example above indicates, for reading the integers you must use **`syscall`** instruction in combination with the **`$v0`** register containing a value of 5. When it's done, the OS waits for keyboard input until user presses **`Enter`**. Afterwards, the input value will be stored in the **`$v0`** register. The story is the same for printing an integer to the output. In this case we use **`syscall`** and loads 1 into the **`$v0`** register, which says to the OS that we want to print out an integer. Also, for printing an integer, its value must be first loaded in the argument register **`$a0`** before calling the **`syscall`** instruction. Also, for printing a string, its address needs to be put into register **`$a0`** and the constant 4 into **`$v0`** before using **`syscall`**. As an example see the three strings in sub2.asm above. You can also see the contents in the Data tab of the simulator.



```
PC       = 4194432
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 268500992
R2  [v0] = 10
R3  [v1] = 0
R4  [a0] = 2189
R5  [a1] = 2147482180
R6  [a2] = 2147482196
R7  [a3] = 0
R8  [t0] = 5684
R9  [t1] = 3495
R10 [t2] = 2189
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 data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    61656c50  65206573  7265746e  65687420   P l e a s e   e n t e r   t h e
[10010010]    72696620  6e207473  65626d75  00203a72     f i r s t   n u m b e r :   .
[10010020]    61656c50  65206573  7265746e  65687420   P l e a s e   e n t e r   t h e
[10010030]    63657320  20646e6f  626d756e  203a7265     s e c o n d   n u m b e r :
[10010040]    65685400  66696420  65726566  2065636e   . T h e   d i f f e r e n c e
[10010050]    203a7369  00000000  00000000  00000000   i s :   . . . . . . . . . . . .
[10010060]..[1003ffff]  00000000


User Stack [7ffffa40]..[80000000]
[7ffffa40]    00000003  7ffffb02  7ffffafd  7ffffaed   . . . . . . . . . . . . . . . .
[7ffffa50]    00000000  7fffffe1  7fffffbb  7fffffa2   . . . . . . . . . . . . . . . .
[7ffffa60]    7fffff71  7fffff5c  7fffff38  7fffff24   q . . . \ . . . 8 . . . $ . . .
[7ffffa70]    7fffff17  7fffff02  7ffffed9  7ffffec3   . . . . . . . . . . . . . . . .
[7ffffa80]    7ffffeac  7ffffe8d  7ffffe7f  7ffffdb5   . . . . . . . . . . . . . . . .
[7ffffa90]    7ffffd77  7ffffd66  7ffffd59  7ffffd3e   w . . . f . . . Y . . . > . . .
[7ffffaa0]    7ffffcfa  7ffffce8  7ffffcd0  7ffffcb5   . . . . . . . . . . . . . . . .
[7ffffab0]    7ffffc97  7ffffc56  7ffffc3f  7ffffc2b   . . . . V . . . ? . . . + . . .
                   fc1c  7ffffc06  7ffffbe0  7ffffbbb   . . . . . . . . . . . . . . . .
                   fba8  7ffffb9a  7ffffb80  7ffffb6e   . . . . . . . . . . . . n . . .
                   fb56  7ffffb10  0000
[7ffffaf0]    73726567  3378652f  6d73
[7ffffb00]    3a450068  6d614b2f  432f
[7ffffb10]    646e6977  5f73776f  6361
[7ffffb20]    66676f6c  3d656c69  425c
[7ffffb30]    65545c6e  5c737473  7473
[7ffffb40]    616b6361  635c6567  6f6c
[7ffffb50]    6f6c2e65  69770067  776f
[7ffffb60]    6e696361  6c665f67  3d73
[7ffffb70]    7269646e  5c3a433d  646e
[7ffffb80]    52455355  464f5250  3d45
[7ffffb90]    73726573  4553555c  5355
[7ffffba0]    553d454d  00524553  5245
[7ffffbb0]    553d4e49  2d524553  5400
[7ffffbc0]    73555c3a  5c737265  5245
```

```
Please enter the first number: 5684
Please enter the second number: 3495
The difference is: 2189
```

```
SPIM Version 9.1.12 of December 14, 2013
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.
```

Figure 4: Program for substracting two numbers and print the result with the proper messages.

Loops and Branches

If you need to write a program that does some repetitive jobs, you may need to use loops and branches in your code. You can use branch instruction to jump from one part of the code to another. The basic concept of a loop hides in using branch instructions, carefully. For example, you can use the label **`Loop`** and after that some instructions. When you want another iteration to be executed you must use **`b`** command by doing **`b Loop`**. Also, there must be a condition for breaking from the loop. For example you can use **`beqz $t0, endloop`** and it means that if register **`t0`** is equal to zero then go to **`endloop`** label. To show how loops and branches work in assembly see the following example. This example is a simple one only for introducing loop and branches and does not do anything important.

```
# branch.asm— Loop and Branches program.
# Registers used:
# $t0 – used to hold the number of iteration
# $t1 – used to hold the counter value in each iteration
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter-- the string to print.

 .text
 main:
 li   $t0, 5              # init the number of loop.
 li   $t1, 0                 # init the counter of the loop

 loop:
   beq $t0, $t1, endloop  # if $t0==$t1 then go to endloop
   addi $t1, $t1, 1        # increment the counter, i.e. $t1++
14 b loop                    # branch to loop label


15 endloop:

16 li   $v0, 10              # 10 is the exit syscall.

17 syscall                   # do the syscall.
```

As you can see in this example, a part of the code is labeled as **`loop`**. The **`b`** instruction in this code does a branch to where the **`loop`** label is placed. But every loop needs a condition to stop. By using the branch if equal (**`beq`**) instruction the program is checking if the condition $t1 is equal to $t0 is satisfied or not. If the condition is satisfied it branches to the label specified in this instruction (here is the **`endloop`** label), otherwise another iteration of the loop is executed.

Assignments

The ***QtSpim*** program needed to do these exercises has been installed on the ECE 103 computers.

Assignment 1

Try to interpret the above code for yourself and guess how many iterations it will do. If we replace the line 12 with **blt $t0, $t1, endloop** or **bge $t0, $t1, endloop** or **bnez $t1, endloop**, how does it affect the number of iterations? Explain the reasons.

Assignment 2

Write a program in MIPS to print out the largest prime number which is smaller than 300.
(Please make sure to write your pseudo-code first and include it in the report.)

Assignment 3

Write a program in MIPS to ask user to input two integers between 10 to 500. If the input number is out of the boundary or one input number is divisible by the other, an error message should be printed. Otherwise, calculate the sum of all even numbers between the two input numbers (excluding the two). (Please make sure to write your pseudo-code first and include it in the report.)

Assignment 4

Write a program to print out a diamond of letters '*' as the following example. The number of lines **n** are inputted from user. (Please make sure to write your pseudo-code first and include it in the report.)

Example:

```
The number of lines? 5

  *

 ***

*****

 ***

  *
```

Assignment 5

Write a MIPS program that asks users to setup a password (with length between 6 and 10; both upper and lower case letters should be included). First let the user enter a string and check if it meets all the requirements. If not, print out an error message and ask the user to enter another string. In the case the user does not enter a proper string again, the program terminates with an appropriate message. Otherwise, after storing the password, again ask the user to re-enter the password. If it is correct, print an appropriate message. Otherwise, give 2 more chances to the user to enter the password. If the user cannot reenter the same password, display an appropriate message on the screen.

Example program output on console:

```
Set a password: Lab

Failed. Please enter a password with the size of 6 to 10!

Set a password: Lablab

Re-enter the password: Lab

Incorrect, you have 2 more chance! Please re-enter the password:
Lablab

Password is setup.
```

Assignment 6
Write a MIPS program that converts a hex string into its binary representation. The hex string should only consist of digit characters **'0', '1', …, '9'** and characters **'A', 'B', 'C', 'D', 'E', 'F'**. Your program should output an error if the input hex string is invalid.
Example program output on console:

```
Hex string: 1A
Its binary value: 00011010
```

```
Hex string: 3R
Invalid hex string
```

Assignment 7
Write a program that takes an array (input from user and can be arbitrary size). It returns the sorted array using bubble sort, and print out the median, the number of positive integers, negative integers, and zeros in an array of integers. The last element of the array should have the value 0xF, which is not used for the calculations and not counted for the size. Display those numbers with appropriate message on the console output. For example, assume an array as follows:

```
(Array1: .word 12, 2, -4, 16, 5, -20, 0, 10, 0xF)
```

```
Sorted Array: -20  -4  0  2  5  10  12  16
The median is: 3

The number of positives is: 5
The number of negatives is: 2
The number of zeroes is: 1
```

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", 5h Edition, 2014.
2. Daniel J. Ellard, "MIPS Assembly Language Programming: CS50 Discussion and Project Book", September 1994.

Appendix

You can see here some useful MIPS assembly language instructions and their descriptions.

Table 1: Basic MIPS Instructions

Instruction	Operand	Description
li	des, const	Load the constant const into des.
lw	des, addr	Load the word at addr into des.
add	des, src1, src2	des gets src1 + src2.
sub	des, src1, src2	des gets src1 - src2.
move	des, src1	Copy the contents of src1 to des.

div	src1, reg2	Divide src1 by reg2, leaving the quotient in register lo and the remainder in register hi.
div	des, src1, src2	des gets src1 / src2.
mfhi	des	Copy the contents of the hi register to des.
mflo	des	Copy the contents of the lo register to des.
mthi	src1	Copy the contents of the src1 to hi.
mtlo	src1	Copy the contents of the src1 to lo.
mul	des, src1, src2	des gets src1 _ src2
mult	src1, reg2	Multiply src1 and reg2, leaving the low-order word in register lo and the high-order word in register hi.
rem	des, src1, src2	des gets the remainder of dividing src1 by src2.
syscall		Makes a system call(refer to table 2 for more information)

Table 2: SPIM syscall

Service	Code	Arguments
print_int	1	$a0
print_float	2	$f12
print_double	3	$f12
print_string	4	$a0
read_int	5	none
read_float	6	none
read_double	7	none
read_string	8	$a0 (address), $a1 (length)
sbrk	9	$a0(length)
exit	10	none

Table 3: Branch Instructions

Instruction	Operand	Description
b	lab	Unconditional branch to lab.
beq	src1, src2, lab	Branch to lab if src1 = src2.
bne	src1, src2, lab	Branch to lab if src1 ≠ src2.
bge	src1, src2, lab	Branch to lab if src1 ≥ src2.
bgt	src1, src2, lab	Branch to lab if src1 > src2.
ble	src1, src2, lab	Branch to lab if src1 ≤ src2.
blt	src1, src2, lab	Branch to lab if src1 < src2.
beqz	src1, lab	Branch to lab if src1 = 0.

`bnez`	`src1, lab`	`Branch to lab if src1 ≠ 0.`
`bgez`	`src1, lab`	`Branch to lab if src1 ≥0.`
`bgtz`	`src1, lab`	`Branch to lab if src1 > 0.`
`blez`	`src1, lab`	`Branch to lab if src1 _ 0.`
`bltz`	`src1, lab`	`Branch to lab if src1 < 0.`
`bgezal`	`src1, lab`	`If src1 _ 0, then put the address of the next instruction into $ra and branch to lab.`
`bgtzal`	`src1, lab`	`If src1 > 0, then put the address of the next instruction into $ra and branch to lab.`
`bltzal`	`src1, lab`	`If src1 < 0, then put the address of the next instruction into $ra and branch to lab.`