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

Computer Architecture and Assembly Language Lab

Fall 2016

Lab 2

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

Goal

In this Laboratory, you will learn how to use flow control, inputs and outputs to your processor and how to make decisions in the assembly language. Before, you have learned in Laboratory 1 how to read the input variables and print them as an output. After Laboratory 2 you will have the knowledge of how to use 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 tutorials under "resources" for Laboratory 1 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 .

Introduction [1, 2]

As you have seen in the first Laboratory, ***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

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

Qtspim window that you can run your program on and the other is a console window that displays the results to you (Figure 1).

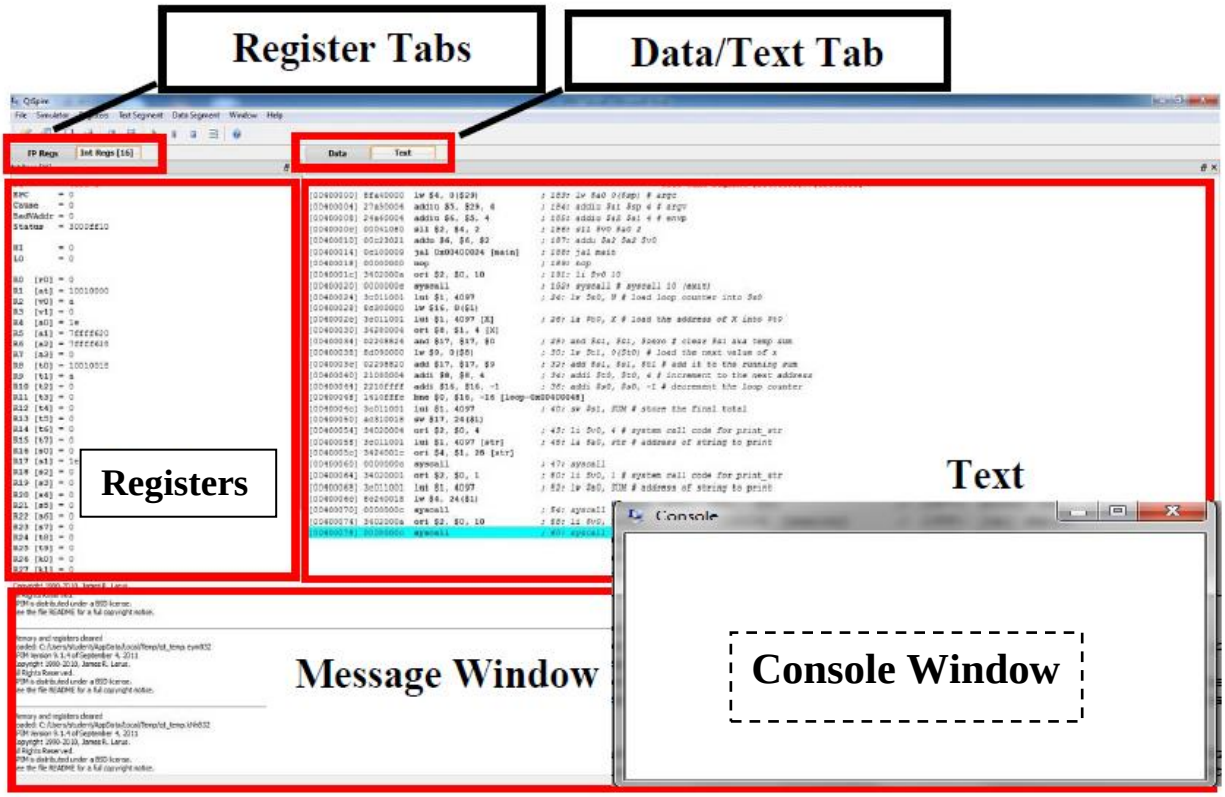

Figure 1: The ***Qtspim***'s windows

To run your program, you have to do the following:

1. Create a new text file, write your code 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.** To execute 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

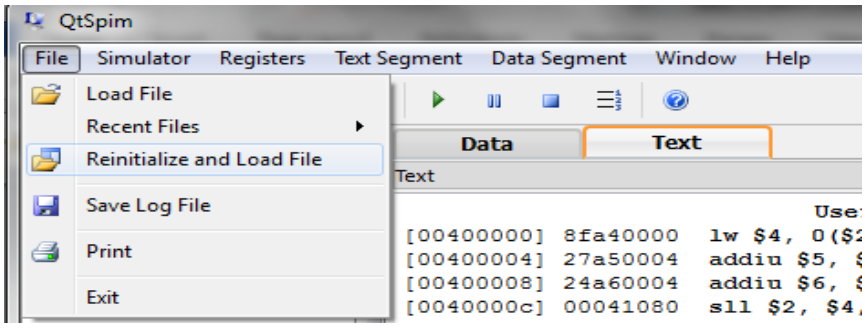

Figure 2: Run the program

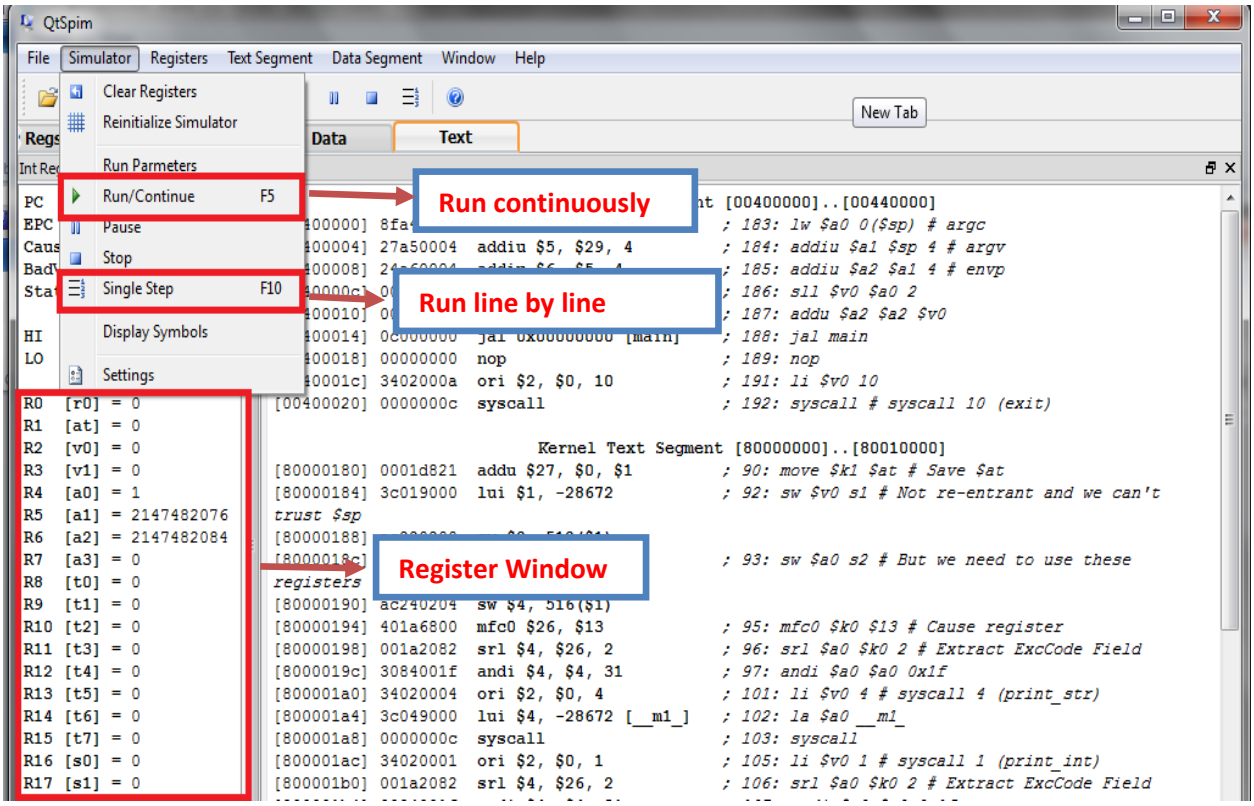

Figure 3: Register window after executing the program

Before doing Laboratory 2 you must be familiar with the process of reading and printing an integer in SPIM. In SPIM, execution of a program starts from the very top instruction, which we always label 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

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

is a simple program that computes the sum of 1 + 2. Copy the code below to a text file name as **Ex1**, **reinitialize** it and run.

```
# add.asm-- A program that computes the sum of 1 plus 2,
# writing the result in register $t0.
# Registers used:
# t0 – temporary register used to hold the result.
# t1 - used to hold the constant 1.
main:
li   $t1, 1                  # load immediate 1 into $t1.
addi  $t0, $t1, 2            # immediate addition $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 (OS). Copy the code below 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 immediate 1 into $t1.

addi   $t0,  $t1, 2 # compute the immediate sum of $t1 and 2, store
the result in $t0.


li    $v0, 10         # Load immediate 10 into register $vo

syscall               # syscall code 10 is for exit.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 read and print 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 code below 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:
```

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

```
# $t0 - used to hold the first number.

# $t1 - used to hold the second number.

# $t2 - used to hold the difference of $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 difference and place result in $t2.


## 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 process 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 argument 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.

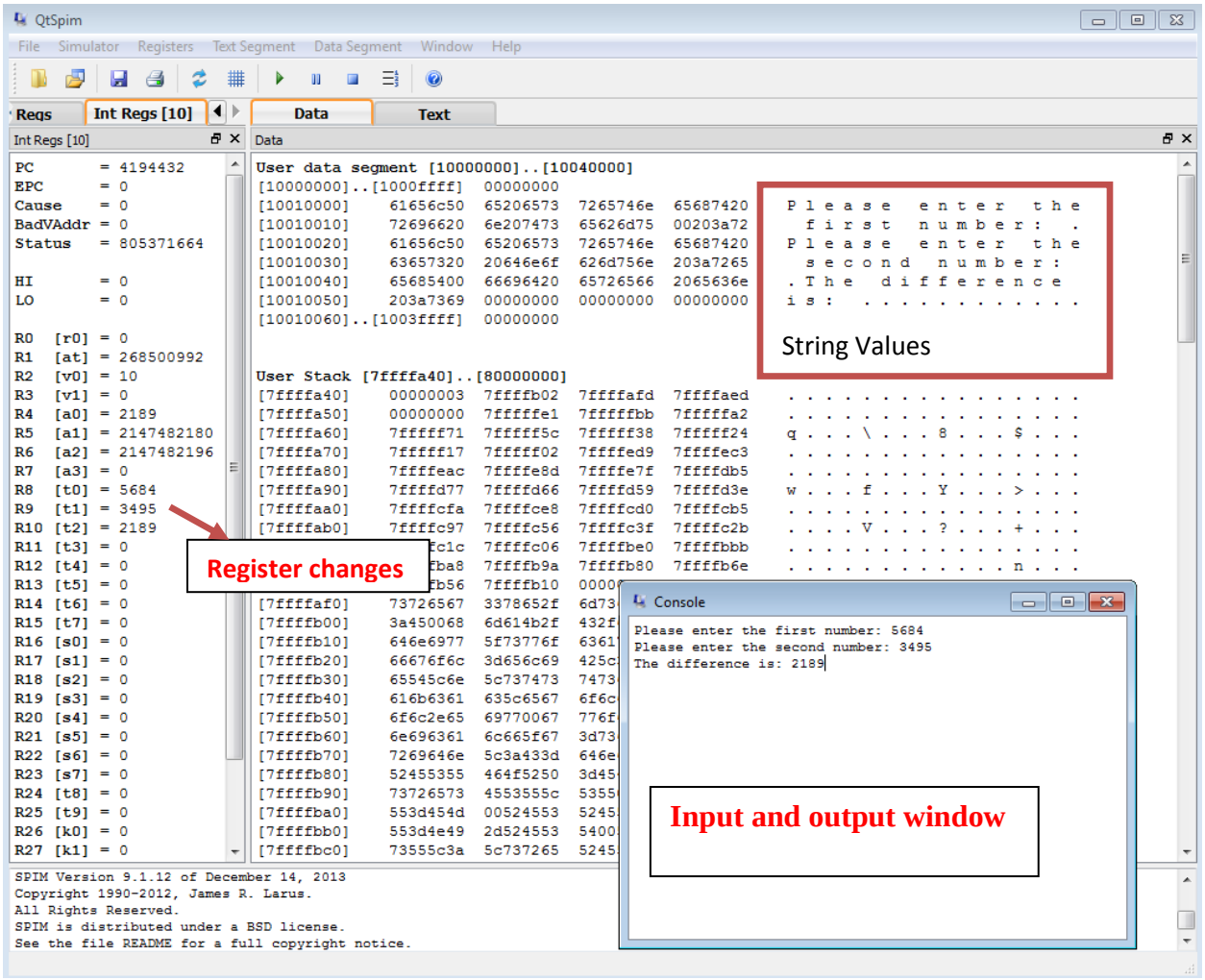

Figure 4: Program for subtracting two integer numbers and printing the result with the proper messages.

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 if a set condition is met. The basic concept of a loop is based on 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

```
beq $t0, $zero, endloop
```

and it means that if register **t0** is equal to zero then go to the address marked by the label **endloop** label. To show how loops and branches work in assembly see the following example. This example is a simple one only for introducing loops and branches and does not do anything important.

```
1 # branch.asm– Loop and Branches program.
2 # Registers used:
3 # $t0 – used to hold the number of iteration
4 # $t1 – used to hold the counter value in each iteration
5 # $v0 - syscall parameter and return value.
6 # $a0 - syscall parameter-- the string to print.
7  .text
8  main:
9  li   $t0, 5          # load the address of hello_msg into $a0.
10 li   $t1, 0          # initialize the counter of the loop to 0
11 loop:
12   beq $t0, $t1, endloop  # if $t0==$t1 then go to endloop
13   addi $t1, $t1, 1        # increment the counter, i.e. $t1++
14 j 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.

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

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.

Exercise: Try to interpret the above code for yourself and guess how many iterations it will do. If we swap the instructions in lines 12 and 13, does it affect the number of iterations or not? Explain the reasons.

Assignments

Assignment 1

Write a program in MIPS that reads a non-negative integer **n**, sums all the values less than or equal to **n**, and displays the result with a message indicates the result. The program should reject any negative number and print an appropriate message. Comment your code appropriately.

Assignment 2

Write a program in MIPS that reads a positive integer **n** that is less than or equal to 100 from the user. Find the closest prime number less than or equal to **n**. The program should reject any number that is out of this range with an appropriate message. Print out the prime number with a message indicating the result. Comment your code appropriately.

First, write the pseudo-code of this program in C.

Assignment 3

Write a program in MIPS that prompts user for a positive integer input **n** that is less than or equal to 1000. Calculate two sums: one is the sum of all even numbers, the other is the sum of all odd numbers less than or equal to **n**. Display the results with appropriate messages. The program should reject any invalid input with an appropriate message. Comment your code appropriately.

Assignment 4

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

Write a program in MIPS that prompt user for two input strings. Compare the second string to the first string. If the two strings match, terminate the program. If the two strings do not match, display the mismatch characters from the second string (including the excessive characters). Prompt the user for another input of the second string to fix the mismatch. Re-check again to see if the strings are matched. Display ap

propriate messages and terminate the program. Comment your code appropriately.

Assignment 5

Write a MIPS program that asks users to setup a password (length between 8 and 12; with only upper and lower case letters). Prompt the user to enter the password string and check if it meets the requirements. If the password string does not meet the requirements, print out an error message and prompt the user for a second try. In the case the user does not enter the proper password string again, display an error message and terminate the program. Otherwise, prompt the user for re-entering the password string. If the entered string is correct, display an appropriate message and terminate the program. If not, give the user two more chances before terminating the program with an error message.

Example of program output on console:

```
Set a password: Lab
Failed. Please enter a password with the size of 8 to 12!
Set a password: Lablablab
Re-enter the password: Lab
Incorrect, you have 2 more chance! Please re-enter the password:
Lablablab
Password is setup.
```

Assignment 6

Write the program in MIPS that declares an array of positive integers. The size of the array is not fixed. The final element of the array should have the value of 0xF, which is not used in calculations, and should be used to calculate the size of the array. The program can prompt the user for an integer input. Compare the input with the array. Display the index of the input in the array if found. If the number does not match any element in the array, insert the number in the array in sorted order. The program is terminated when input a negative integer. Print the new array with appropriate message on the screen. For example assume an array as follows:

```
(array:    .word 4, 5, 23, 5, 8, 3, 15, 67, 8, 9, 0xF)

Sorted array: 3, 4, 5, 5, 8, 8, 9, 15, 23, 67
Enter a number to search: 20
Number not found. Added to array.
Sorted array: 3, 4, 5, 5, 8, 8, 9, 15, 20, 23, 67
Enter a number to search: 5
Number at index 2
```

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

```
Enter a number to search: -1
Program terminated.
```

Lab report

Write a report that includes the codes, results and conclusions of the assignments. This report must include the pseudo-codes and the codes in MIPS assembly language. It is better to add some screen shots from ***QtSpim*** showing intermediate results as the programs run.

Your reports along with the codes must be uploaded on Sakai and the report must be printed and delivered before the start of the next lab. Please put your name and Student ID at the start of 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.