Name(1)	Jeffrey Ye
Name(2)	Sai Sao Kham
Name(3)	Vivian Luo
Name(4)	Mitchell Schmidt
Instructor	Prof.Shambhu Upadhyaya
Assignment:	CSE341 Project 1
Due Date:	03/07/2016

Problem 1

In the sample factorial program(nrfact.mips) that was given to us, we observed that when we gave the program a valid input, the factorial of that number was printed out on the console. However, when we gave the program an input out of the range of 0 to 10, nothing was printed in the console. Also if a decimal between 0 to 10 was given, error messages were produced in the console. When we remove the syscalls in the program, we saw that no messages were printed out on the console. A runtime exception error occurred when a variable was used as an input. The code would still run even if you were to delete the .globl main line from the program, but if you were to remove the main: line the program wouldn't execute. Also removing any labeled section within the code will cause an error when branching.

Problem 2

2a) Li means "Load immediate". La means "Load address".

#from wiki https://en.wikibooks.org/wiki/MIPS_Assembly/Pseudoinstructions

Li (load immediate) → lui (load upper immediate) and then ori (logical or immediate)

La (load address) → lui (load upper immediate) and then ori (logical or immediate)

Commented Code

```
#factorial
.text
.globl main
main:
subu $sp,$sp,32      #allocates space for stack, stack frame 32 bits long
sw $ra,20,($sp)      #returning address is stored in stack
sw $fp,16,($sp)        #frame pointer(stores the address of sp before it moved), restores
frame
                      pointer
addu $fp,$sp,32      #sets up frame pointer

li $a0,10            #puts 10 in a0
la $t0,fact          #puts address of fact into $t0
jalr $t0                     #jumps to fact function and stores returning address in $31

lw $ra,20($sp)               #loads contents from stack into returning address
lw $fp,16($sp)               #loads content from stack into frame pointer
j $rra               #jump to returning address($ra stores return address of jal calls)
fact:
subu $sp,$sp,32      #allocates space for stack
sw $ra,20($sp)       #stores returning address of stack
sw $fp,16($sp)       #stores previous frame pointer
addu $fp,$sp,32      #sets up frame pointer

sw $a0,0($fp)                #stores argument to frame pointer

lw $2,0($fp)         #loads x=10
move $2,$a0          #copies x=10 to argument

bgtz $2,$L2          #branch to x > 0
li $2,1              #puts 1 in reg2
j $L1                #go to L1

$L2:
lw $3,0($fp)         #loads 10
subu $2,$3,1         #10-1
#subu $2, $2,1
move $a0,$2          #copies content in reg2 to argument
```

```
jal fact            #jump to fact function and store returning address

lw $3,0($fp)        #loads 10
mul $2,$2,$3        #10*fact(10-1), answer in reg2

$L1:
lw $31,20($sp)      #restores returning address
lw $fp,16($sp)              #restores frame pointer
addu $sp,$sp,32     #pop stack
j $31               #go to returning address
```

2b) The starting address of the main routine in ‘rfacts.asm’ is 0x00400018. The operation that stored this address in $31 was the jal (jump and link) operation. The point of doing this is to allow a program to jump to another label while maintaining knowledge of its position in the section of code where jal was called.

Problem 3

```
.data
        prompt1: .asciiz "Enter int one: "
        prompt2: .asciiz "Enter int two: "
        prompt3: .asciiz "Enter int three: "
        message: .ascii "\nThe two smaller ints added together = "
        x: .word 0
        y: .word 0
        z: .word 0

.text
.globl main

        main:
```

```
#Promt the user for x
li $v0, 4
la $a0, prompt1
syscall

#Get x
li $v0, 5
syscall

#Store the result in x
lw $t0, x
add $t0, $t0, $v0
sw $t0, x

#Promt the user for y
li $v0, 4
la $a0, prompt2
syscall

#Get y
li $v0, 5
syscall

#Store the result in y
lw $t1, y
add $t1, $t1, $v0
sw $t1, y

#Prompt the user for z
li $v0, 4
la $a0, prompt3
syscall

#Get z
li $v0, 5
syscall

#Store the result in z
```

```
lw $t2, z
add $t2, $t2, $v0
sw $t2, z

#Compare ints
bge $t0,$t1, .xbigger           # x >= y
        j .ybigger        # x < y

.xbigger:
        bge $t0,$t2, .xxbigger          # x >= z
                j .zbigger        # x < z
.ybigger:
        bge $t1,$t2, .yybigger          # y >= z
                j .zbigger        # y < z
.xxbigger:      #get y and z
        lw $t3, y
        lw $t4, z
        j .donehere
.yybigger:      #get x and z
        lw $t3, x
        lw $t4, z
        j .donehere
.zbigger:       #get x and y
        lw $t3, x
        lw $t4, y
        j .donehere
.donehere:
        add $t5, $t3, $t4

#Display message
li $v0, 4
la $a0, message
syscall

#print or show int in $t5
li $v0, 1
move $a0, $t5
syscall
```

```
#Exit
li $v0, 10
syscall
```

Testing

- During testing, when we entered the values 5, 6, 7; the value resulted was 11.

- When the integers -2, -5, 0; were inputted the resulting value was -7.

- When the values 11, 11, 11; were inputted the sum outputted was 22.

Problem 4

```
# Sources
#http://www.cs.nott.ac.uk/~psztxa/g51csa/l10-hand.pdf
#https://www.youtube.com/watch?v=B0GAXDjfdbQ
#https://www.youtube.com/watch?v=B6ky4Weahm4
```

Recursive Code:

```
.data
	prompt:		.asciiz "Enter the nummber you want in the fibonacci sequence: "
	message:	.ascii "\The fibonacci number is: "
	theNumber:	.word 0
	theAnswer:	.word 0

.text
.globl main
	main:
	#Get the number to be found in fibo sequence
	li $v0, 4
	la $a0, prompt
	syscall
```

```
	li $v0, 5
	syscall

	sw $v0, theNumber

	#Display Results
	li $v0,4
	la $a0, message
	syscall

	#Call fibo
	lw $a0, theNumber
	jal fibo
	sw $v0, theAnswer

	li $v0, 1
	lw $a0, theAnswer
	syscall

	#Exit
	li $v0, 10
	syscall


.globl fibo
	fibo:
	addi $sp, $sp, -8		#Allocate space for sp
	sw $ra, 0($sp)
	move $v0, $a0

	ble $a0, 1, fiboBranchDone		#if n <= 1 return

	sw $a0, 4($sp)
	addi $a0, $a0, -1		#n-1
	jal fibo			#calls f(n-1)

	lw $a0, 4($sp)
	sw $v0, 4($sp)
```

```
addi $a0, $a0, -2           #n-2
jal fibo                    #calls f(n-2)

lw $v1, 4($sp)
add $v0, $v0, $v1

fiboBranchDone:
lw $ra, 0($sp)
addi $sp, $sp, 8
jr $ra
```

Testing

- During testing, when we entered the value 8, the value resulted was 21.
- When the integer 9 was inputted, the resulting value was 34.
- When the value 10 was inputted, the output was 55.

Non-Recursive Code:

```
.data
    message: .asciiz "\nFibonacci of 9 is "

.text
.globl main

    main:

    li $s0, 0       #f(n)
    li $s1, 1       #f(n+1)
    li $s2, 2       #count
    li $s3, 0       #answer

    .fibo:
        bge $s2, 9, .done
        add $s4, $s0, $s1
        move $s0 $s1
        move $s1, $s4
```

```
        addi $s2, $s2, 1
        j .fibo

    .done:
    add $s3, $s0, $s1

    #Display message
    li $v0, 4
    la $a0, message
    syscall

    #print answer
    li $v0, 1
    move $a0, $s3
    syscall

    #Exit
    li $v0, 10
    syscall
```

Testing

The instruction count for the Non-Recursive Fibonacci method was 29, while the instructions executed for the Non-Recursive method was 55. The instruction count for the Recursive Fibonacci method was 48 and the number of instructions executed was about 8209. Thus clearly showing that the Non Recursive Fibonacci is faster than the Recursive Fibonacci method, due to the Recursive Fibonacci needing to execute more instructions than the Non Recursive method.