**Assignment 5**

ルビス　ルスファン　アンシャー

Lubis, Luthfan Anshar

15B15953

1. Factorial (factorial.s)

The program accepts an integer input from user, which factorial is then calculated by the following code:

*Accepts an integer input from user, then copied into $t0 register.*

main:

li $v0, 5

syscall

move $t0, $v0

*Creates stack, then assigns address 0 for the return address, and address 4 for the initial input which factorial will be calculated.*

addi $sp, $sp, -8

sw $ra, 0($sp)#save return address

sw $t0, 4($sp)#save initial n

jal fact#jump to subroutine fact

*The initial input is then put into temporary register $t1 and $t3 for multiplication later. Register $t3 will be used for displaying the output. First, the value of the input ($t1) is checked to see whether it is 0 or not. In the case of 0, the program jumps to zero.*

fact:

lw $t1, 4($sp)

lw $t3, 4($sp)

beq $t1, $zero, zero

*In the case of 0, it simply returns the value 1.*

zero:

li $t3, 1

jr $ra

*In the case of not 0, the program continues to notzero, where the factorial calculation is executed with loop until the value of $t1 becomes 0. The code then jumps to end*

notzero:

addi $t1, $t1, -1

beq $t1, $zero, end

mul $t3, $t3, $t1

j notzero

*end sends the process back to the upper part of the code.*

end:

jr $ra

*The output $t3 is copied into $a0 and then printed. The return address is also returned back from stack, and the stack is then restored.*

li $v0, 1

move $a0, $t3

syscall

lw $ra, 0($sp)

addi $sp, $sp, 8

jr $ra

Examples:

1. input = 0; output = 1
2. input = 1; output = 1
3. input = 12; output = 479001600
4. Binary Conversion with Callee-saves (callee.s)

Below is the result of changing the code written in the study materials into one using callee-saves:

.text

main:

addi $sp, $sp, -12

sw $ra, 0($sp) # ra is callee-save

sw $s0, 4($sp) # s0 is callee-save

sw $s1, 8($sp) # s1 is callee-save

li $s0, 0

li $s1, 0

loop:

# the program basically loops the same way as the program’s example

jal read1bit

sll $s0, $s0, 1

add $s0, $s0, $v0

addi $s1, $s1, 1

blt $s1, 8, loop

move $a0, $s0

end:

# because callee-saves were used, they have to be restored

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

addi $sp, $sp, 12

li $v0, 1

syscall

jr $ra

read1bit:

li $v0, 5

syscall

jr $ra

Examples:

1. input = 00000000; output = 0
2. input = 11111111; output = 255
3. input = 10110011; output = 179

1. Array and Sum (array.s)

The program first accepts integer input from user to determine the length of the array, then accepts every single element of the array from the user. The program then prints the sum of all the elements of the array.

*Creates stack for return address and later uses.*

main:

addi $sp, $sp, -8

sw $ra, 0($spa)

li $t3, 0

*Accepts input to determine the length of array which value is then copied into register $a0. Then checks if the length is equal to 0 then jump to end.*

li $v0, 5

syscall

move $t0, $v0

move $a0, $t0

beq $a0, $zero, end

*If the length of the array is not equal to 0, then proceeds into create\_array.*

sw $t0, 4($sp) # store length of array

jal create\_array

*create\_array is a subroutine that assigns the address of the array and store the address.*

create\_array:

# the size of memory necessary to store every element of the array is (length of array \* 4)

sll $t0, $t0, 2

# assigns the array head’s address

li $v0, 9

move $a0, $t0

syscall

move $t0, $v0 # copy the new address to $t0

sw $t0, 8($sp) # save the address to stack

lw $a1, 4($sp) # load length of array

*The program moves into read\_int where it loops to accept elements of array.*

read\_int:

li $v0, 5

syscall # accepts integer

move $a0, $v0

# inserts the value to current address of array

sw $a0, 0($t0)

# decreases the remaining length of array by 1

addi $a1, $a1, -1

# checks whether the remaining length of array is 0 or not, if 0, jumps to calc\_sum, if not, continues.

beq $a1, $zero, calc\_sum

# add the current address with 4, for the next integer

addi $t0, $t0, 4

j read\_int

*If there is no longer any integer to insert into the array, then the program jumps to calc\_sum. calc\_sum and sum calculates the sum of all elements of the array.*

calc\_sum:

# loads the address of the first element of the array

lw $t0, 8($sp)

lw $a1, 4($sp) # loads the length of array

sum:

# loads the element from the specified address

lw $t1, 0($t0)

# execute the addition

add $t3, $t3, $t1

# check whether there is still remaining element to add

addi $a1, $a1, -1

beq $a1, $zero, end

# add the address by 4 to load the next element in the array

addi $t0, $t0, 4

j sum

*The program finalizes in end, where it displays the sum and restore the stack.*

end:

# display the sum

li $v0, 1

move $a0, $t3

syscall

# restore stack

lw $ra, 0($sp)

addi $sp, $sp, 8

jr $ra

Examples:

1. input = 5, 3, 3, 3, 3, 3; output = 15
2. input = 7, 23, 23, 1, 23, 54, 4, 4; output = 132
3. Exception (myexceptions.s)

If this exception handler responded to the program:

1. If the exception code is not equal with 13 and the system call number is not equal with 100, the value in $a0 will be multiplied by 10
2. If the exception code is equal with 13 or the system call number is equal with 100, the value 100 will be returned.

.ktext 0x80000180

# obtaining the exception code from Cause register

mfc0 $k0, $13

srl $k0, $k0, 2

andi $k0, $k0, 0x1f

# checking whether the exception code is not equal with 13

li $k1, 13

bne $k0, $k1, done

# checking whether the system call number is equal with 100

li $k1, 100

beq $v0, $k1, done

# multiplies the value in $a0 by 10

li $k1, 10

mul $a0, $a0, $k1

move $v0, $a0

done:

# returns to the program

mfc0 $k0, $14

addi $k0, $k0, 4

mtc0 $k0, $14

eret

.text

.globl \_start

\_start:

jal main

li $v0, 10

syscall