Course Name: Computer Architecture Lab

Course Number and Section: **14:332:333:04**

Lab: Lab # 1 – Warming up with SPIM simulator

Lab Instructor: Christos Mitropoulos

Date Performed: September 15th, 2016

Date Submitted: September 29th, 2016

Submitted by: Pawel Derkacz (ID: 151-00-5994)

----------For Lab Instructor Use ONLY----------

GRADE: _____

COMMENTS:

Electrical and Computer Engineering Department
School of Engineering
Rutgers University, Piscataway, NJ 08854

Code from Exercise 1:

```
# Exercise1 is used in assignments 1 and 2
 .data 0x10000000
        Number1:
        .align 2
        .word 18 #Start data at designated hex location. Number is the number to undergo factorial.
 .text 0x00400000
 .globl main

main:
        lw $10, Number1($0) # Load ‘number1’ into register 10
        ori $11, $0, 1
        ori $9, $0, 1  # both ORIs load "1" into register 11 and 9, respectively
#compute the factorial of Number ($10)!

factloop:
        bge $11, $10, factexit #Once reg10 hits 1 (the value in reg11), leave the loop
        mul $9, $10, $9 #Reg 9 holds the factorial by use of multiplication
        sub $10, $10, 1 #Decrement 10 by 1, to proceed with the factorial
        j factloop # Keep looping until bge condition met

factexit:
        #the computation of the factorial is over
        #Is the result in $9 correct? The result in $9 is in hexadecimal form

        li $2, 10 #Loads "10" into reg 2 in order to give the syscall command "exit"
        syscall
```

- Assignment 1)

ori $9, $0, 1 in binary code is: (op->) 001101 (rs->) 00000 (rt->) 01001 (const->) 0000000000000001 or just 00110100000010010000000000000001

- Assignment 2)

See bold-ed comments above.

Code from Exercise 2:

```
#Exercise is used in assignment 3
 .data 0x10000000
 .align 2
        Array: .word 2 5 6 7 12 16
        Size: .word 6
 .text 0x00400000
 .globl main
main:
        lw $a0, Size    # Load the size of array into $a0, using lw
        li $a1, 0       # initialize index i
        li $t2, 4       # t2 contains constant 4, initialize t2
        li $t0, 1       # Initialize result to one
loop:
        mul $t1, $a1, $t2       # t1 gets i*4
        lw $a3, Array($t1)      # a3 = Array[i]
        mul $t0, $t0, $a3       # result *= Array[i]
        sw $t0, Array($t1)      # store result in the Array2 in location i (???)
        addi $a1, $a1, 1        # i++
        blt $a0, $a1, END       # go to END if finished
j loop

END:
        li $v0, 10
        syscall

# Index i reaches 7 due to a0 < a1, not a0 <= a1. Therefore, a1 must surpass a0.
```

- Assignment 3)

    Part 1 – See above code

    Part 3 – Data segment included after Part 4. (Part 2 skipped – simply asked to observe)

    Part 4 – Index reaches size + 1 due to the condition a0 < a1, meaning a1 must be larger.

```
 User data segment [10000000]..[10040000]
[10000000]    00000002  0000000a  0000003c  000001a4    . . . . . < . . . . . .
[10000010]    000013b0  00013b00  00076200  00000000    . . . . . ; . . . b . . . . . .
[10000020]..[1003ffff]  00000000
```

Code from Exercise 3:

```
# Used in assignment 4
# 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 $t0.
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter.

 .text

main:
        ## Get first number from user, put 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 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 $t1.
```

- Assignment 4)

Part 1:

```
PC       = 4194364
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 3
R3  [v1] = 0
R4  [a0] = 3
R5  [a1] = 2147481180
R6  [a2] = 2147481196
R7  [a3] = 0
R8  [t0] = 8
R9  [t1] = 3
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
R28 [gp] = 268468224
R29 [sp] = 2147481176
R30 [s8] = 0
R31 [ra] = 4194328
```

Part 2 – Syscall DEPENDS on the value found in $v0. This dependency, in turn, causes $v0 to be highly important, seeing as syscall can not function without it. The value within $v0 tells syscall what function it should execute.

Part 3 and 4 (in one program):

```
# Used in assignment 4
# 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 $t0.
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter.

 .data
        choice1:
        .asciiz "\n Please enter first number you wish to calculate the difference of. #1 - #2\n"
        choice2:
        .asciiz "\n Please enter the second number.\n"
        custom:
        .asciiz "\n Please enter the sentence you wish trail the answer. Max 20 characters!\n"
        string: .space 20
.text

main:
        ## Printing initial message
        li $v0, 4          # system call code for writing is 4
        la $a0, choice1# load address of the message into a0
        syscall            # print message

        ## Get first number from user, put into $t0.
        li $v0, 5          # load syscall read_int into $v0.
        syscall # make the syscall.
        move $t0, $v0 # move the number read into $t0.

        ## Printing the second message
        li $v0, 4          # system call code for writing is 4
        la $a0, choice2# load address of the message into a0
        syscall            # print message

        ## Get second number from 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 $t1.

        ## Printing the third message
        li $v0, 4          # system call code for writing is 4
        la $a0, custom # load address of the message into a0
        syscall            # print message
```

```
	## Read string from input
	li $v0, 8	# prep for string reading
	la $a0, string	# give start address to a0 for string
	li $a1, 20	# Specify how long the string will be, stored in a1
	syscall		# Get string

	## Computing the difference and printing
	sub $t2, $t0, $t1	# Computing
	move $a0, $t2		# Move the difference into a0
	li $v0, 1		# Prep for printing int
	syscall			# Print the difference

	li $v0, 4		# Prep for printing string
	la $a0, string		# Give address for string
	syscall			# Print string

	li $v0, 10		# Load code to exit program
	syscall			# exit
```

Code from Exercise 4:

```
# Used in Assignment 5
# Registers used:
# $t0 - used to hold the x coordinate of the first pair.
# $t1 - used to hold the y coordinate of the first pair.
# $t2 and $t5 - used to help compute and hold the distance between the coordinates.
# $t3 - used to hold the x coordinate of the second pair.
# $t4 - used to hold the y coordinate of the second pair.
# $t6 - used to hold -1 in order to flip negative distances.
# $v0 - syscall parameter and return value.
# $a0 - syscall parameter.

.data
	initial:
	.asciiz "\n Initializing Manhattan Distance calculator.\n Please enter the first X-coordinate.
NOTE: Integers only!\n"

	init2:
	.asciiz "\n Please enter the Y-coordinate of the first pair.\n"

	second:
	.asciiz "\n Please enter the X-coordinate of the second pair.\n"

	second2:
	.asciiz "\n Please enter the Y-coordinate of the second pair.\n"
.text
```

```
main:
        li $t6, -1          # Load -1 into $t6 for future use to get abs. value

        ## Printing initial message
        li $v0, 4           # system call code for writing is 4
        la $a0, initial     # load address of the message into a0
        syscall             # print message

        ## Get x-coord from user, put into $t0.
        li $v0, 5           # load syscall read_int into $v0.
        syscall # make the syscall.
        move $t0, $v0 # move the number read into $t0.

        ## Printing second message
        li $v0, 4           # system call code for writing is 4
        la $a0, init2       # load address of the message into a0
        syscall             # print message

        ## Get y-coord from user, put into $t0.
        li $v0, 5           # load syscall read_int into $v0.
        syscall # make the syscall.
        move $t1, $v0 # move the number read into $t1.

        ## Printing third message
        li $v0, 4           # system call code for writing is 4
        la $a0, second # load address of the message into a0
        syscall             # print message

        ## Get second x-coord from user, put into $t0.
        li $v0, 5           # load syscall read_int into $v0.
        syscall # make the syscall.
        move $t3, $v0 # move the number read into $t3.

        ## Printing fourth message
        li $v0, 4           # system call code for writing is 4
        la $a0, second2         # load address of the message into a0
        syscall             # print message

        ## Get second y-coord from user, put into $t0.
        li $v0, 5           # load syscall read_int into $v0.
        syscall # make the syscall.
        move $t4, $v0 # move the number read into $t4.

        ## Compute the distances for x and y
        sub $t2, $t0, $t3       # Compute the difference between X coords and put in $t2
        sub $t5, $t1, $t4       # Compute the difference between Y coords and put in $t5
```

```
        ## Making sure both values are positive

check1:
        bgez $t2, check2        # If $t2 >= 0, continue the program
        mul $t2, $t2, $t6       # Else, multiply $t2 by -1
check2:
        bgez $t5, final         # If $t5 >= 0, continue
        mul $t5, $t5, $t6       # Else, multiply $t5 by -1
final:
        ## Should be dealing with only positive distances now - Compute and Print answer
        add $t2, $t2, $t5       # Get final distance
        move $a0, $t2           # Store the distance into a0 for printing
        li $v0, 1               # Inserted printing code for integers
        syscall                 # Print the integer distance

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

- Assignment 5)

See above code.

- Assignment 6)

```
.text
main:
        li $t0, 0       # Initialize counter to 0
        li $t1, 0       # Holds offset from address
        li $t2, 4       # Holds size of a Word
loop:
        add $t3, $t1, $a0       # $t3 holds the current address to access
        ulw $v1, 0($t3)                 # Go to address in $t3 and store in $v1
        beqz $v1, return        # If the word in v1 is equal to zero, jump to the end

        add $t3, $t1, $a1       # Update address for storing
        usw $v1, 0($t3)                 # Store the word in $v1 at the address in $t3

        addi $t0, $t0, 1# Increment counter by 1
        mul $t1, $t0, $t2       # Offset = counter * size of a Word

        j loop
return:
        li $v0, 10              # Load exit code 10 into v0
        syscall                 # Exit

# NOTE: Address in a0 is NOT guaranteed to be divisible by 4, and thus will cause an exception when trying to read and write.
```

Code from Exercise 5:

```
#ex5.asm
 .data 0x10000000
        ask: .asciiz "\nEnter a number between 0 and 50000: "
        ans: .asciiz "\nAnswer: "

 .text 0x00400000
 .globl main
main:
        li $v0, 4
        la $a0, ask      # Loads the ask string
        syscall          # Display the ask string
        li $v0, 5        # Read the input
        syscall
        move $t0, $v0 # n = $vo, Move the user input to t0
        addi $t1, $0, 0 # i = 0
        addi $t2, $0, 0 # ans = 1, Starting case (n=0) is 1
        li $t3, 2        # we store two in $t3
loop:
  beq $t0, 0, END              # if we reach 0 we stop
        div $t0, $t3           # divide $t0 with 2
        mfhi $t1               # store the remainder to $t1
  beq $t1, $0, ADD             # if the reminder is 0 then we go to ADD
        sub $t0, $t0, 1 # we reduce $t0
  j loop                       # go back to loop
ADD:
        add $t2, $t2, $t0      # add the even number to the $t2
        sub $t0, $t0, 1 # reduce $t0
  j loop                       # go back
END:
        li $v0, 4
        la $a0, ans      # Loads the ans string
        syscall
        move $a0, $t2 # Loads the answer
        li $v0, 1
        syscall
        li $v0, 10       # Exiting
        syscall
```

- Assignment 7)

Ex5's function is to add all the even numbers from 0 to N where N is specified by the user. This function is made by taking N and dividing it by 2. If the remainder is 1, then the number was odd and the loop continues with N-1, else N is added to a register containing the sum. If too large a number is entered, overflow has a possibility of occurring due to the constant addition. The register can only handle a number up to 0xFFFFFFFF.

- Assignment 8)

```
#ex5b.s
 .data 0x10000000
        ask: .asciiz "\nEnter a number between 0 and 50000: "
        ans: .asciiz "\nAnswer: "

 .text 0x00400000
 .globl main
main:
        li $v0, 4
        la $a0, ask      # Loads the ask string
        syscall          # Display the ask string
        li $v0, 5        # Read the input
        syscall
        move $t0, $v0 # n = $vo, Move the user input to t0
        addi $t1, $0, 0 # i = 0
        addi $t2, $0, 0 # ans = 0, Starting case (n=0) is 0
        li $t3, 2        # we store two in $t3
loop:
  beq $t0, 0, END              # if we reach 0 we stop
        div $t0, $t3           # divide $t0 with 2
        mfhi $t1               # store the remainder to $t1
  bne $t1, $0, ADD             # if the reminder is not 0 then we go to ADD
        sub $t0, $t0, 1 # we reduce $t0
  j loop                       # go back to loop
ADD:
        add $t2, $t2, $t0      # add the odd number to the $t2
        sub $t0, $t0, 1 # reduce $t0
  j loop                       # go back
END:
        li $v0, 4
        la $a0, ans      # Loads the ans string
        syscall
        move $a0, $t2 # Loads the answer
        li $v0, 1
        syscall
        li $v0, 10       # Exiting
        syscall
```