CS224

Section No.: **5**

Spring 2017

Lab No.: **1**

Your Full Name/Bilkent ID: **Berat Biçer/21503050**

1. Write a MIPS program that

Creates an array of maximum size of **20** elements that asks the user first the number of elements and then enters the elements one by one, displays array contents, reverses the array contents and displays the array (for example 1, 2, 3 becomes 3, 2, 1).

2. Write a MIPS program that

Gets a input string and checks if it is a **palindrome**. (Study load byte, store byte and some other instructions if need).

3. Write a MIPS program that

Implements the following expression without using **div** instruction. If necessary use instructions other that we have seen in the class.

x= (c - d) % 2

4. Generate the object code (in hex) for the following la instructions. Show your work for the intermediate steps (please remember that la is a pseudo instruction and implemented by two instructions).

la $t1, a (lui $t1, a &&ori $t2, $t2, b)

la $t2, b

.....

.data

.space 40

a: .word 1, 2, 3, 4

b: .word 1

5. Define the following terms by providing two examples for each of them.

a. Symbolic machine instruction

b. Machine instruction

c. Assembler directive

d. Pseudo instruction

Answers

1-)

**.data**

**array: .space 80 # Create an array of size 20**

**prompt: .asciiz " Enter an integer: "**

**space: .asciiz " "**

**.text**

**la $s0, array # load base address to s0**

**# Get input count from user**

**# Prompt message**

**li $v0, 4**

**la $a0, prompt**

**syscall**

**# Get input**

**li $v0, 5**

**syscall**

**addi $s1, $v0, 0 # store the input count in s1**

**# calculate last location of the array**

**mul $s2, $s1, 4**

**add $s2, $s2, $s0**

**addi $t1, $s0, 0 # temporarily store s0 inside t1**

**jal getdata**

**addi $t1, $s0, 0 # temporarily store s0 inside t1 to be editted**

**jal display**

**addi $t1, $s0, 0 # temporarily store s0 inside t1 to be editted**

**jal reverse**

**# Prompt message**

**li $v0, 4**

**la $a0, prompt**

**syscall**

**addi $t1, $s0, 0 # temporarily store s0 inside t1 to be editted**

**jal display**

**# Exit**

**li $v0, 10**

**syscall**

**getdata:**

**# loop to get input of amount in s1**

**# Get input**

**li $v0, 5**

**syscall**

**sw $v0, 0($t1) # move data to loc t1+0**

**addi $t1, $t1, 4 # update base address**

**bne $t1, $s2, getdata**

**jr $ra**

**# end of getdata**

**display:**

**# Function that displays array elements on the console**

**lw $t2, 0($t1) # load into t1**

**# Prompt message**

**li $v0, 4**

**la $a0, space**

**syscall**

**# Print data**

**li $v0, 1**

**la $a0, 0($t2)**

**syscall**

**addi $t1, $t1, 4 # increment base address by 4**

**bne $t1, $s2, display**

**jr $ra**

**# end of display**

**reverse:**

**# Function that reverses the order of the array**

**addi $t0, $s1, -1 # t0 = input count-1;**

**sll $t0, $t0, 2 # t0 = t0 \* 4;**

**add $t0, $t1, $t0 # t0 = (input count-1)\*4 + base address, address of the last element**

**jump: # $t1 = base address, $t0 = last address**

**lw $t2, 0($t1) # store an element into t2**

**lw $t3, 0($t0) # store its pair into t3**

**sw $t3, 0($t1) # store pair into the element's place**

**sw $t2, 0($t0) # store the element into its pair's place**

**addi $t1, $t1, 4 # increment base by 4**

**addi $t0, $t0, -4 # decrement base by 4**

**slt $t2, $t0, $t1 # t2 = 1 if t0=last address < t1=base address, else t2 = 0**

**beq $t2, $0, jump**

**jr $ra**

**# end of reverse**

2-)

**.data**

**prompt: .asciiz "Enter a string: "**

**promptYes: .asciiz "\nYES"**

**promptNo: .asciiz "\nNO"**

**.text**

**# Get string input from user**

**# Prompt instruction**

**li $v0, 4**

**la $a0, prompt**

**syscall**

**# Get input**

**li $v0, 8**

**syscall**

**# Store data in registers**

**sw $a0, 0($s0) # buffer**

**sw $a1, 0($s1) # length**

**# Base case, data size = 0 or 1**

**addi $t0, $0, 1**

**beq $t0, $s1, finish # size 1**

**beq $0, $s1, finish # size 0**

**# General case**

**sll $s2, $s1, 1**

**addi $s2, $s2, 1 # find element count untill middle point**

**# Loop for storing bytes in stack**

**move $0, $t0 # loop index**

**addi $s3, $s3, 0 # base address of the array**

**addi $t1, $s3, 0**

**jal loop**

**loop:**

**# Store bytes until midpoint**

**sb $s0, 100($t1) # store byte in address**

**addi $t0, $t0, 1 # increment loop counter by 1**

**addi $t1, $t1, 4 # increment base address by 4**

**bne $t0, $s2, loop**

**compare:**

**# Compare byte equality**

**addi $t2, $t1, 4 # 2 pointers to mid point, one for incrementing and one for decrementing**

**# $t1: decreasing pointer, $t2: increasing pointer**

**bne $t1, $t2, no # if data in t1 != t2, jump to no**

**addi $t1, $t1, -4 # decrease address in t1 by 4 to get previous item in array**

**bne $t1, $s3, compare # if the base address is not reached, jump to compare**

**j yes # comparing is finished, prompt yes**

**no: li $v0, 4**

**la $a0, promptNo**

**syscall**

**yes: li $v0, 4**

**la $a0, promptYes**

**syscall**

**finish:**

**# quit**

3-)

**# assuming $s0 = c, $s1 = d**

**sub $s0, $s0, $s1 # x-y -> x**

**loop:**

**sll $s0, $s0, 1**

**beq $s0, $0, quit**

**addi $t0, $0, 1**

**beq $s0, $t0, quit**

**j loop**

**quit:**

4-)

Generate the object code (in hex) for the following la instructions. Show your work for the intermediate steps (please remember that la is a pseudo instruction and implemented by two instructions).

la $t1, a

la $t2, b

.....

.data

.space 40

a: .word 1, 2, 3, 4

b: .word 1

**.space 40 means open place for 40 data in memory, and t1 has the base address of an array consisting 1, 2, 3 and 4. Then .word 1 for b suggests that b is the base address of the first of an array containing 1 element that is 1.**

**la $t1, a == lui $t1, a: 0x 3D433200 la $t2, b == lui $t2, b: 0x3B74A100**

**ori $t1, a: 0x 35432124 ori $t2, b: 0x34220011**

5-)

**Symbolic Machine Instruction: Another name for Assembly Language**

**Machine Instruction: A bit-based instruction written from Assembly instructions that machines can understand and execute.**

**Assembler Directive: A command that tells the Assembler what it needs to know to be able to execute an Assembly instruction ie, location of the program in memory etc.**

**Pseudo Instruction: An Assembly instruction that needs more than more than one actual Assembly instruction to be executed ie, fake instructions to make coding&reading codes easy. For example la is a pseudo instruction that corresponds to 2 actual Assembly instructions which are (lui, ori).**