CS224

SECTION 03

LAB01 PRELIMINARY REPORT

AYBÜKE CEREN DURAN/21302686

**1. Input an Array:** 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 display the array (for example 1, 2, 3 becomes 3, 2, 1).

**ANSWER: (EXECUTED ON MARS) It outputs 3,2,1 for the array 1,2,3.**

.data

userArray: .space 80

revArray: .space 80 #this array is allocated for user's reversed version array

message: .asciiz "\nHello there! Please enter the size of the array: "

elements: .asciiz "\nNow, enter the elements for your array:\n"

warning: .asciiz "The size you have entered is invalid. You must enter max. 20 \n"

display: .asciiz "Your array elements: "

reverseArray: .asciiz "This is the reversed version of your array: "

done: .asciiz "\n Bye stranger :) "

thespace:.asciiz " "

dot: .asciiz "-) "

.text

welcome:

#t1 holds the max. size which is 20

li $t1, 20

li $s6, 0

#display the message

li $v0, 4

la $a0, message

syscall

#user will enter the size

li $v0, 5

syscall

j validateSize

validateSize:

sgt $s1, $v0, $t1 #this instruction decides whether if the user array size exceeds max.size

bnez $s1, warnTheUser

beqz $s1, enterElementsMessage

warnTheUser:

#display the message

li $v0, 4

la $a0, warning

syscall

b welcome #branch welcome now

enterElementsMessage:

# $t2 will hold the user array size for further use

# $t3 will hold the address of the array

la $a1, userArray

la $s7, revArray #it will hold the reverse array address for further use.

move $t3, $a1

move $t2, $v0

move $t6, $v0

move $s3, $v0

move $t5, $a1 #for further use, we have copied

addi $s6, $s6, 1

#now the user can enter the array elements

li $v0, 4

la $a0, elements

syscall

enterElements:

li $v0, 1

add $a0, $s6, $zero

syscall

li $v0, 4

la $a0, dot

syscall

li $v0, 5

syscall

addi $s6, $s6, 1

sw $v0, 0($t3)#store the element user entered

seq $t7, $t2, 1

beq $t7, 1, enteringOver

addi $t3, $t3,4 #its is time for next element

subi $t2, $t2, 1 #subtract the size by 1 now

j enterElements

enteringOver:

li $v0,4

la $a0, display

syscall

j displayElements

displayElements:

lw $a0, 0($a1)

li $v0, 1

syscall

li $v0, 4

la $a0, thespace

syscall

beq $t6, 1, reverseElementsMessage# display completed, it is time for reverse version

addi $a1, $a1, 4 #trace next element

subi $t6, $t6, 1 #subtract the size now

j displayElements

reverseElementsMessage:

li $v0,4

la $a0, reverseArray

syscall

reverseNow:

lw $t4, 0($a1) #copy the last element of the user's array.

sw $t4, 0($s7) #copy from user's array element into reversed array.

lw $a0, 0($s7)

li $v0, 1

syscall

li $v0, 4

la $a0, thespace

syscall

beq $s3, 1, finish

add $s7, $s7, 4

sub $a1, $a1, 4

subi $s3, $s3, 1 #subtract the size now

j reverseNow

finish:

li $v0,4

la $a0, done

syscall

li $v0, 10

syscall

**2. Palindrome:** Write a MIPS program that

Read an integer array as defined above and check if it is a palindrome. (For example, an array with four elements like 1, 4, 4, 1 is a palindrome: the order of the elements from the beginning to the end and from the end to the beginning are the same.

**ANSWER:** **(EXECUTED ON MARS)** **It outputs YES for the array 3 3. It outputs YES for the array 1 4 5 4 1. It outputs NO for the array 1 3. It outputs YES for the array 2 . It outputs YES for the array with no elements.**

.data

userArray: .space 80

message: .asciiz "\nHello there! Please enter the size of the array: "

elements: .asciiz "\nNow, enter the elements for your array:\n"

warning: .asciiz "The size you have entered is invalid. You must enter max. 20 \n"

display: .asciiz "Your array elements: "

done: .asciiz "\n Bye stranger :) "

thespace:.asciiz " "

dot: .asciiz "-) "

yes: "\n YES, IT IS A PALINDROME \n"

no: "\n NO, IT IS NOT A PALINDROME \n"

end:"\n THE END :)\n"

.text

welcome:

#t1 holds the max. size which is 20

li $t1, 20

li $s6, 0

#display the message

li $v0, 4

la $a0, message

syscall

#user will enter the size

li $v0, 5

syscall

j validateSize

validateSize:

sgt $s1, $v0, $t1 #this instruction decides whether if the user array size exceeds max. size

bnez $s1, warnTheUser

beqz $s1, enterElementsMessage

warnTheUser:

#display the message

li $v0, 4

la $a0, warning

syscall

b welcome #branch welcome now

enterElementsMessage:

# $t2 will hold the user array size for further use

# $t3 will hold the address of the array

beqz $v0, yesPalindrome #if the size is 0, the array is palindrome

beq $v0, 1, yesPalindrome #if the size is 1, the array is palindrome

la $a1, userArray

move $t3, $a1

move $t2, $v0

move $s7, $v0

move $t6, $v0

move $t5, $a1 #for further use, we have copied

addi $s6, $s6, 1

#now the user can enter the array elements

li $v0, 4

la $a0, elements

syscall

enterElements:

li $v0, 1

add $a0, $s6, $zero

syscall

li $v0, 4

la $a0, dot

syscall

li $v0, 5

syscall

addi $s6, $s6, 1

sw $v0, 0($t3)#store the element user entered

seq $t7, $t2, 1

beq $t7, 1, enteringOver

addi $t3, $t3,4 #its is time for next element

subi $t2, $t2, 1 #subtract the size by 1 now

j enterElements

enteringOver:

li $v0,4

la $a0, display

syscall

j displayElements

displayElements:

lw $a0, 0($a1)

li $v0, 1

syscall

li $v0, 4

la $a0, thespace

syscall

beq $t6, 1, computeMiddle# display completed, it is time for reverse version

addi $a1, $a1, 4 #trace next element

subi $t6, $t6, 1 #subtract the size now

j displayElements

computeMiddle:

#now we will hold the address of the middle element

#add $t4, $t5, $a1

#srl $t4, $t4, 1 #now $t4 holds the address of the middle element of the array

sra $t4, $s7, 1

j palindromeDecision

palindromeDecision:

#compare the elements from begin and end

lw $s4, 0($t5) #load from begin

lw $s5, 0($a1) #load from end

bne $s4, $s5, noPalindrome

addi $t5, $t5, 4 #update left side

subi $a1, $a1, 4 #update right side

addi $t4,$t4,-1 #decrement the midpoint to trace the comparisons

beq $t4, 0, yesPalindrome

#beq $t4, $t5, checkMiddleRight

j palindromeDecision

checkMiddleRight:

beq $a1, $t4, yesPalindrome

yesPalindrome:

li $v0,4

la $a0, yes

syscall

j endingScene

noPalindrome:

li $v0,4

la $a0, no

syscall

j endingScene

endingScene:

li $v0,4

la $a0,end

syscall

li $v0,10

syscall

**3. Perform Division Without Division Instruction:** Write a MIPS program that

Inputs two integer numbers n1 and n2 and performs integer division n1/n2 without using division instruction and display both the division and the remainder of the division.

**ANSWER: (EXECUTED ON MARS) It outputs remainder 0 and division 3 for 12 and 4.**

**“ “ “ 1 “ “ 3 for 13 and 4.**

**“ “ “ -3 “ “ -3 for -15 and 6.**

**“ “ “ 0 “ “ 0 for -18 and -9.**

.data

input1: .asciiz "Please enter your first input as integer: \n"

input2: .asciiz "Please enter your second input as integer: \n"

welcome: .asciiz "Hello there! \n"

results: .asciiz "Here are the results: \n"

division: .asciiz "Here is the division: "

blank: .asciiz "\n"

remainder: .asciiz "\nHere is the remainder: "

bye: .asciiz "\nBye :) "

.text

entering:

li $v0, 4

la $a0, welcome

syscall

li $v0, 4

la $a0, input1

syscall

li $v0,5

syscall

move $t0, $v0#input1 is copied

move $t2, $v0#input1 is copied for further use in the loop

li $t3, 0#this is for holding the result, for quotient (case: all inputs are positive)

li $t4, 0#this is for holding remainder

li $t5, 0#this is for holding the result, for quotient (case: all inputs are positive)

li $s0, 0#for checking sign of inputs for further use

li $s1, 0#for checking sign of inputs for further use

li $s2, 0# this is for OR operation for further use

li $s3, 0#this is for AND operation for further use

li $s4, 0#this is for XOR operation for further use

li $v0, 4

la $a0, input2

syscall

li $v0,5

syscall

move $t1, $v0 #input2 is copied

checkSignOfInputs:

slt $s0, $t0, $zero #check the sign of input1

slt $s1, $t1, $zero #check the sign of input2

or $s2, $s1, $s0 #s2=0 go to allPositiveLoop

and $s3, $s1, $s0

xor $s4, $s1, $s0 #if inputs have different sign

beq $s2, 0, allPositiveLoop

beq $s3, 1, allNegativeLoop

beq $s4, 1, difSignLoop

difSignLoop:

bgez $t0, difSignLoop1

blez $t0, difSignLoop2

difSignLoop1:

blez $t2, findRemainder

add $t2, $t2, $t1

subi $t3, $t3, 1

j difSignLoop1

difSignLoop2:

bgez $t2, findRemainder

add $t2, $t2, $t1

sub $t3, $t3, 1

j difSignLoop2

#if all the inputs are positive, go to allPositiveLoop

allPositiveLoop:

blt $t2, $t1, findRemainder

sub $t2, $t2, $t1

addi $t3, $t3, 1

j allPositiveLoop

#if all the inputs are negative, go to allNegativeLoop

allNegativeLoop:

sge $t7, $t2, $zero

beq $t7, 1, findRemainder

sub $t2, $t2, $t1

addi $t5, $t5, 1

j allNegativeLoop

findRemainder:

rem $t4, $t0, $t1

printResults:

li $v0,4

la $a0, results

syscall

li $v0, 4

la $a0, division

syscall

beq $s2, 0, printPosRes

beq $s3, 1, printNegRes

printPosRes:

li $v0, 1

move $a0, $t3

syscall

li $v0, 4

la $a0, remainder

syscall

li $v0,1

move $a0, $t4

syscall

j done

printNegRes:

li $v0, 1

move $a0, $t5

syscall

li $v0, 4

la $a0, remainder

syscall

li $v0,1

move $a0, $t4

syscall

j done

done:

li $v0,4

la $a0, bye

syscall

li $v0, 10

syscall

**4. Object Code Generation:** Generate the object code for the following instructions first in binary then give it in hex.

add $t0, $t1, $t2

addi $s0, $s3, 15

mult $a0, $a1

sw $t1, 8($t2)

lw $t2, 8($t1)

**ANSWER:**

**Binary form of the instructions:**

**0000 0001 0010 1010 0100 0000 0010 0000 add $t0, $t1, $t2**

**0010 0010 0111 0000 0000 0000 0000 1111 addi $s0, $s3, 15**

**0000 0000 1000 0101 0000 0000 0001 1000 mult $a0, $a1**

**1010 1101 0100 1001 0000 0000 0000 1000 sw $t1, 8($t2)**

**1000 1101 0010 1010 0000 0000 0000 1000 lw $t2, 8($t1)**

**Hex form of the instructions:**

**0x012A4020 add $t0, $t1, $t2**

**0x2270000F addi $s0, $s3, 15**

**0x00850018 mult $a0, $a1**

**0xAD490008 sw $t1, 8($t2)**

**0x8D2A0008 lw $t2, 8($t1)**

**5. Define Terms:** Define the following terms and provide an answer etc as described.

a. Symbolic machine instruction: give two examples

b. Machine instruction: give two examples and write their symbolic equivalents

c. Assembler directive: give two examples.

d. Pseudo instruction: give two examples and provide its implementation using real instructions.   
 Note in your answer only use symbolic machine instructions.

**ANSWER:**

**a-) Symbolic machine instruction:**  it refers to another name for the instruction in assembly language. It is text-based representation form of a binary code.

Example:

* x=y+z; **its symbolic machine code** is:

#$s0=x , $s1=y, $s2=z

add $s0, $s1, $s2

* c = c+5;

b=c – 15;

**Symbolic machine codes** of high level codes are below:

#$s0=c, $s1=b

addi $s0, $s0, 5

addi $s1, $s0, -15

**b-) Machine instruction:** It is numeric (hex) version of an operation code (op code) followed by one or more operands.

Example:

lw $t2, 32($0) corresponds to (its machine instruction on Figure-02)

op rs rt imm op rs rt imm

35 0 10 32

100011 00000 01010 0000 0000 0010 0000

Figure-01: field values of lw $t2, 32($0) Figure-02: machine code of lw $t2, 32($0)

sub $t0, $t3, $t5 corresponds to (its machine instruction on Figure-04)

op rs rt rd shamt fucnt op rs rt rd shamt funct

0 11 13 8 0 34

000000 01011 01101 01000 00000 100010

Figure-03: field values of sub $t0, $t3, $t5 Figure-04: machine code of sub $t0, $t3, $t5

**c-) 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.

This directive is executed by the registers in MIPS.

Example:

**.space** **n**: reserves n bytes of memory space

**.word w1, ……., wn**: reserves n words

**d-) Pseudo instruction:** An instruction, similar to a computer instruction in form, that is used to control a compiler or assembler rather than being directly executed as an instruction by hardware.

Example: li is pseudo instruction

li $t0, 0x1234AA55 corresponds to lui $t0, 0x1234

ori $t0, 0Xaa55

move is pseudo instruction

move $s3, $s4 corresponds to add $s3, $s4, $zero