CS224

Section No.: 2

Spring 2019

Lab No. 1

Name: Hassan Raza

ID No.: 21701811

**Part 1. Preliminary Work / Preliminary Design Report**

**Q. 5: Define following:**

**Symbolic machine instructions:**

Symbolic machine instructions are instructions given to computer using assembly languages, i-e not using high-level programming languages. In this, symbols and words are used for each instruction. For example:

For adding two numbers: add $s0, $s1. $s2 For subtracting two numbers: sub $t0, $t3, $t5

**Machine instruction:**

Machine instruction is an instruction given in machine code. Machine code is the language of computers and is composed of binary numbers. It is read by the computer's central processing unit ([CPU](https://whatis.techtarget.com/definition/processor)). For example:

For adding two numbers: 000000100011001010000000000100000 For subtracting two numbers: 00000001011011010100000000100010

**Assembler directives:**

Assembler directives control the assembly process and supply the data to the program. They are used for assembling data into sections, reserving memory spaces, initializing memory and declaring global variables etc. Examples for assembler directives include:

For aligning the memory: .align n For storing string in memory: .asciiz str

**Pseudo instructions:**

Pseudo instructions are the instructions given to assembler which are not executed directly by the assembler rather they direct the assembler to do some particular operations. These are used to perform operations with single statement and then assembler expands that satement to a set of instructions needed for that operation. Examples for pseudo instructions include:

For moving one variable value into another: move move $t, $s

Implementation: addiu $t, $s, 0

For loading a number or alphabet into variable: li (load immidiate) li $t, C

Implementation: lui $t, C\_hi  
 ori $t, $t, C\_lo

**Q. 4:Generate object code for following:**

All these instructions are made of two instructions, thus we will find hexadecimal code for two.

.space acquires first memory spaces ‘a’ starts from 0x00000014 and ‘b’ starts from 0x00000020

All the instructions are I-type instructions, so we will have opcode(6 bits), rs(5 bits), rt(5bits) and imm(16bits)

**la $t1, a**

pseudo instruction: la $t1, a

real instruction: lui $1, 0x00001001 ori $9, 00000014

Binary code: 001111 00000 00001 0001000000000001 001101 00100 10010 0000000000010100

Hexadecimal: 0x3C011001 0x34290014

**la $t2, b**

pseudo instruction: la $t2, b

real instruction: lui $1, 0x00001001 ori $10, 00000020

Binary code: 001111 00000 00001 0001000000000001 001101 00001 01010 0000000000100000

Hexadecimal: 0x3C011001 0x342A0020

**lw $t2, b**

pseudo instruction: lw $t2, b

real instruction: lui $1, 0x00001001 lw $10, 00000020($1)

Binary code: 001111 00000 00001 0001000000000001 100011 00001 01010 0000000000100000

Hexadecimal: 0x3C011001 0x8C2A0020

**lw $t2, b**

pseudo instruction: lw $t2, b

real instruction: lui $1, 0x00001001 lw $10, 00000020($1)

Binary code: 001111 00000 00001 0001000000000001 100011 00001 01010 0000000000100000

Hexadecimal: 0x3C011001 0x8C2A0020

**Q. 1: MIPS program that creates array:**

.text

main:

#take input for array size

li $v0, 4

la $a0, size

syscall

li $v0, 5

syscall

move $t0, $v0

#declaring array

addi $s0, $0, 0

addi $t2, $0, 0

#taking elements and storing in array

for:

beq $s0, $t0, exit3

li $v0, 4

la $a0, mess

syscall

li $v0, 5

syscall

sw $v0, array+0($t2)

addi $t2, $t2, 4

addi $s0, $s0, 1

j for

exit3

#displaying elements

addi $s0, $0, 0

addi $t2, $0, 0

for2:

beq $s0, $t0, exit

addi $s0, $s0, 1

lw $t1, array($t2)

li $v0, 1

#addi $a0, $t1, 0

move $a0, $t1

syscall

addi $t2, $t2, 4

j for2

exit:

#reversing elements

addi $s0, $0, 0

addi $v0, $zero, 4

la $a0, space

syscall

for3:

beq $s0, $t0, exit2

addi $t2, $t2, -4

lw $t1, array($t2)

li $v0, 1

move $a0, $t1

syscall

addi $s0, $s0, 1

j for3

exit2:

.data

size: .asciiz "enter size"

mess: .asciiz "enter element"

space: .asciiz "\n"

.align 2

array: .space 80

**Q. 2: MIPS program that checks if a string is a palindrome:**

.text

.globl main

main:

#taking string

li $v0, 4

la $a0, enter

syscall

li $v0, 8

la $a0, string

li $a1, 20

syscall

#making counters

add $t0, $zero, $zero

add $t1, $zero, $zero

length:

lb $s0, string($t0)

beq $s0, '\0', NEXTCHECK

add $t0, $t0, 1

add $t1, $t1, 1

j length

NEXTCHECK:

add $t0, $zero, 0

sub $t2, $t1, $t0

sub $t2, $t2, 2

div $t5, $t2, 2

add $t5, $t5, 1

add $t3, $0, 0

palindrom:

lb $s0, string($t0)

lb $s1, string($t2)

add $t0, $t0, 1

sub $t2, $t2, 1

add $t3, $t3, 1

beq $t5, $t0, pand

beq $s0, $s1, palindrom

bne $s0, $s1, ELSE

j palindrom

ELSE:

li $v0, 4

la $a0, notpalind

syscall

j EXIT

pand:

li $v0, 4

la $a0, palind

syscall

EXIT:

li $v0, 10

syscall

.data

string: .space 80

notpalind: .asciiz "not palindromic!"

palind: .asciiz "palindromic!"

enter: .asciiz "enter string: "

**Q. 3: MIPS program that computes expression:**

.text

.globl main

main:

#take input

li $v0, 4

la $a0, enter

syscall

li $v0, 5

syscall

move $t0, $v0

li $v0, 4

la $a0, enter2

syscall

li $v0, 5

syscall

move $t1, $v0

sub $t2, $t0, $t1

andi $t3, $t2, 15

li $v0, 4

la $a0, sol

syscall

li $v0, 1

move $a0, $t3

syscall

.data

enter: .asciiz "enter 1st number"

enter2: .asciiz "enter second number"

sol: .asciiz "solution is:"