CSS 224
Section No.: 2
Spring 2019
Lab No. 1
Zeynep Cankara/21703381

Solution of the Question 1

```
## Text Section ##
        .globl __start
        .text
__start:
        # prompt the task on screen
        li $v0, 4
        la $a0, task
        syscall
        # prompt user to enter size of the array
        li $v0, 4
        la $a0, arrayMsg
        syscall
        # read the user input
        li $v0, 5
        syscall
        sw $v0, size
        jal createArray
        jal displayItems
        # $t0 points to the address of the last element, load as an argument to reverseArray
        la $a0, 0($t0)
        jal reverseArray
        jal displayItems
        # exit the program
        li $v0, 10
        syscall

# Function for instantiating the array with the user input
createArray:
        lw $t1, size # load the size into the t0
        la $t0, array # load the base address into the t1
        addi $t2, $0, 0 # i = 0 (index)

        # prompt user to enter the values
        la $a0, readItemsMsg
        li $v0, 4
        syscall
        readInput:
                li $v0, 5          # user input (int)
                syscall
                # write data to the current index
                sw $v0, 0($t0)
                # increment array index by one position (4 bytes)
```

```
                addi $t0, $t0, 4          # array[i]
                # increment the index
                addi $t2, $t2, 1          # i += 1
                # continue if i < size
                blt $t2, $t1, readInput
        jr $ra # go to the next instruction

# Function displays the array content
displayItems:
        lw $t1, size # load the size into the t0
        la $t0, array # load the base address into the t1
        addi $t2, $0, 0 # i = 0 (index)

        # display message
        li $v0, 4
        la $a0, displayMsg
        syscall
        display:
                # read data from the memory
                lw $a0, 0($t0)
                # increment array index by one position (4 bytes)
                addi $t0, $t0, 4          # array[i]
                # print item
                li $v0, 1
                syscall
                # if not the last element add seperator in between
                addi $t4, $t2, 1

                bgt $t1, $t4, seperate # if (index + 1) < size seperate
                ble $t1, $t4, done        # else done
                seperate:
                        la $a0, comma
                        li $v0, 4
                        syscall
                done:
                # increment the index
                addi $t2, $t2, 1          # i += 1
                # continue if i < size
                blt $t2, $t1, display
        # set $t0 adress to the last element
        subi $t0, $t0, 4
        jr $ra # go to the next instruction

## Function reverses array items
reverseArray:
        # $t0 already have the address of the last element in the array
        la $t1, array # load the base address into the t1
        # swap elements if  ($t1 > $t0)
        swapItems:
                # load items into the registers
                lw $t5, 0($t0)
                lw $t6, 0($t1)
```

```
                # swap and store items
                sw $t5, 0($t1)
                sw $t6, 0($t0)
                # increment address of $t0 by 1 word (4 byte)
                addi $t1, $t1, 4
                # decrement address of $t1 by 1 word (4 byte)
                subi $t0, $t0, 4
                # check condition
                bgt $t0, $t1, swapItems
        jr $ra # go to the next instruction

## Data Section ##
        .data
array: .space  80 # word (4 bytes) => 20 words (80 bytes), array of 80 bytes allocated
size: .word  0 # int user input (4 bytes)
task: .asciiz "Program creates an array of max size 20 initializes the array in accordance with the
user input. \n"
arrayMsg: .asciiz "Enter the size of the array: (1 <= size <= 20) \n"
readItemsMsg: .asciiz "Please enter the array items: \n"
comma: .asciiz ", "
displayMsg: .asciiz "\n Displaying the array items \n"
```

Solution of the Question 2

```
# Zeynep Cankara -- 20/02/2019
# isPalindrome.asm -- Takes a string as user input checks isPalindrome
# Registers used:

## Text Section ##
        .globl __start
        .text

__start:
        # prompt the task on screen
        li $v0, 4
        la $a0, task
        syscall
        # ask user enter the string
        li $v0, 4
        la $a0, prompt
        syscall
        # read the string
        li $v0,8  # read str
        la $a0, strBuffer #load byte space into address
        li $a1, 20 # allot the byte space for strings
        sw $a0, strBuffer # store string in the buffer
        syscall

        # $t0 points to the address of the string
        la $a0, strBuffer
        jal size # obtain size of the string
```

```
        move $a0, $v0 # save the last character adress
        jal isPalindrome

        # test ispalindrome function
        move $s0, $v0

        # return result isPalindrome
        li $v0, 4
        la $a0, isPalindromeMsg
        syscall
        # test ispalindrome function
        move $a0, $s0
        li $v0, 1
        syscall

    li $v0,10 # exit program
    syscall

## Function for counting number of characters in the string
## reg used: v0: returns the address of the last char in the string
size:
        # $a0 have the address of the string
        move $t0, $a0  # store string in $t0 reg
        addi $t2, $0, 0 # cnt = 0, counter for characters in the string
        startCnt:
                lbu $t3, 0($t0)   # load current character in t3
                beq $t3, $0, done # finish when encounter a NULL character
                addi $t2, $t2, 1 # cnt += 1
                addi $t0, $t0, 1 # go to next char
                j startCnt
                done:
                        subi $t2, $t2, 1 # exclude enter character
                        sw $t2, strSize # store the size of the string
                        subi $t0, $t0, 2 # go back 2 chars (1 for ENTER 1 for 00(end of string))
                        move $v0, $t0 # return the sadress of the last char
        jr $ra # goto the next instruction

## Function Takes the last char address of the string (a0) loads adress of the first char from the
buffer (strBuffer)
## Compares and return 1 if the string is palindrome 0 otherwise
## reg used: a0,t0, t1, v0
isPalindrome:
        la $t0, strBuffer # address of the first char
        move, $t1, $a0 # adress of the last char
        next:
                lb $t3, 0($t0)    # load the first character
                lb $t4, 0($t1)    # load the last character
                bne $t3, $t4, false       # characters not equal return 0
                ble $t1, $t0, true        # address of last char <= adress of first char return 1
                addi $t0, $t0, 1
                subi $t1, $t1, 1
                j next
```

```
        false:
                li $v0, 0           # return 0
                j finish
        true:
                li $v0, 1 # return 1
        finish:
    jr $ra # goto the next instruction

## Data Section ##
    .data
strBuffer: .space 20 # 1 char (1 byte), string can have max 20 letters
task: .asciiz "Program checks whether string is palindrome or not \n"
prompt: .asciiz "Please enter the string: \n"
strSize: .word  0 # number of characters in string
isPalindromeMsg: .asciiz "String is palindrome if result is 1, 0 otherwise \n"
```

Solution of the Question 3

```
# prelimReportQuestion3.asm -- Implements the arithmetic expression (c - d) % 16 without using div

## Text Section ##
    .globl __start
    .text
__start:
    # prompt the program task on screen
    li $v0, 4
    la $a0, task
    syscall

    # prompt user to enter c
    li $v0, 4
    la $a0, cMsg
    syscall
    # read the user input
    li $v0, 5
    syscall
    sw $v0, c

    # prompt user to enter d
    li $v0, 4
    la $a0, dMsg
    syscall
    # read the user input
    li $v0, 5
    syscall
    sw $v0, d

    # call the function to evaluate expression
    lw $a0, c
    lw $a1, d
```

```
        jal evaluate

        # print x
        li $v0, 4
        la $a0, xMsg
        syscall
        # print remainder
        li $v0, 1
        lw $a0, x
        syscall


        # exit the program
        li $v0, 10
        syscall

## Function evaluates expression x = (c - d) % 16 without using command div
## Inputs: [$a0: c; $a1: d}
## Outputs: {$v0: x}
evaluate:
        move $t0, $a0 # t0 holds c
        move $t1, $a1 # t1 holds d
        sub $t2, $t0, $t1 # t2 holds (c-d)
        blt $t2, $0, makePositive
        bge $t2, $0, eval
        makePositive:
                addi $t2,$t2, 16 # consecutively add 16 to make (c-d) positive
                blt $t2, 0, makePositive
        eval:
                addi $t3, $0, 1 # int i = 1 (index)
                addi $t4, $0, 0 # int product = 0
                while:
                        mul $t4, $t3, 16 # product = 16 * i
                        addi $t3, $t3, 1 # i += 1
                        ble $t4, $t2, while
                # evaluate x
                # x  = (c-d) - (product - 16)
                subi $t4, $t4, 16  # (product - 16)
                sub $v0, $t2, $t4 # x  = (c-d) - (product - 16)
                sw $v0, x
        jr $ra

## Data Section ##
        .data
task: .asciiz "Program implements the arithmetic expression x = (c - d) % 16 without using div. \n"
cMsg: .asciiz "Please enter c value: "
dMsg: .asciiz "\nPlease enter d value: "
xMsg: .asciiz "\nValue of x is "
c: .word  0 # c value type(int) size(4 bytes)
d: .word  0 # d value type(int) size(4 bytes)
x: .word  0 # x value type(int) size(4 bytes)
```

Solution of the Question 4

Memory Allocation Explained

.data # starts from address: 0x10010000
.space 20 # allocates 20 bytes of space starting from 0x10010000, ending 0x10010020
a # array of 3 words. 1 word (4 bytes). a is an array containing 12 (3x4) bytes. Words allocated consecutively within 20 bytes of space allocated previously. Have base address 0x10010014
b # single word allocated after 20 bytes of space, have a base address of 0x10010020. Takes up 4 bytes of space.

Machine Instructions Explained

la $t1, a # breaks down to 2 instructions as follows:
lui $at, 0x1001 (1.)
ori $t1, $at, 20 (2.)

la $t2, b # breaks down to 2 instructions as follows:
lui $at, 0x1001 (3.)
ori $t1, $at, 32 (4.)

lw $t2, b # breaks down to 2 instructions as follows:
# lui $1, 0x1001 (5.)
# lw $10, 32 (6.)

1.) lui: I type instruction. Opcode: 0xf. R[rt]: $at. R[rs]: 0. imm: 0x1001
Machine Instruction: 001111 00000 00001 0001 0000 0000 0001
Hex format: 0x3c011001

2.) ori: I type instruction. Opcode: 0xd. R[rt]: $t1. R[rs]: $at.imm: 20
Machine Instruction: 001101 00001 01001 0000 0000 0001 0010
Hex format: 0x34290014

3.) lui: I type instruction. Opcode: 0xf . R[rt]: $at. R[rs]: 0. imm: 0x1001
Machine Instruction: 001111 00000 00001 0001 0000 0000 0001
Hex format: 0x3c011001

4.) ori: I type instruction. Opcode: 0xd. R[rt]: $t2. R[rs]: $at.imm: 32
Machine Instruction: 001101 00001 01010 0000 0000 0010 0000
Hex format: 0x342a0020

5.) lui: I type instruction. Opcode: 0xf. R[rt]: $at. R[rs]: 0. imm: 0x1001
Machine Instruction: 001111 00000 00001 0001 0000 0000 0001
Hex format: 0x3c011001
6.) lw: I type instruction. Opcode: 0x23. R[rs]: $1 = $at. R[rt]: $10 = $t1. Imm: 32
Machine Instruction: 100011 00001 01010 0000 0000 0010 0000
Hex format: 0x8c2a0020

Solution of the Question 5

a. Symbolic machine instruction:

Symbolic format for machine language instruction. It makes machine instructions more readable to humans.

Examples:

subi $t0, 3
add $t1, $t0

b. Machine instruction:

Machine readable instructions made from bits (1's and 0's). They are necessary for the machine to carry out specific processes.

Examples:

0010 0001 0010 1001 1111 1111 1111 1101
1010 1111 1010 1011 1111 1111 1110 1100

c. Assembler directive:

Assembler directive is a message to the assembler which is necessary for the assembler to carry out some assembly process. They do not executed by the program.

Examples:

.data
.space

d. Pseudo instruction:

Complex instructions which have simple human readable form in order to write complex tasks more easily. Simple form assembly language commands which do not have direct machine equivalent. Performed using simpler instructions.

Examples:

la $t2, $t1
move $t1, $t0,