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

Solution of the Question 1

```
#                   * text segment *
#  Includes part (a), part(b), part(c)

        .globl __start
        .text
__start:
        # Print an intro message
        li $v0, 4
        la $a0, intro
        syscall

        # Function call
        jal interactWithUser

        # exit the program
        li $v0, 10
        syscall

# Function for intereacting with the user
interactWithUser:
        mallocMenu:  # allocate the stack space
                subi $sp, $sp, 20
                sw $s0, 16($sp)
                sw $s1, 12($sp)
                sw $s2, 8($sp)
                sw $s3, 4($sp)
                sw $ra, 0($sp)
        menu:
                # display the menu
                jal display
                #user option selection
                li $v0, 12 # v0 contains char
                syscall
                # load options
                li $s0, 'a'
                li $s1, 'b'
                li $s2, 'c'
                # make calls to the subprogram labels according to the which option chosen
                beq $v0, $s0, option1
                beq $v0, $s1, option2
                beq $v0, $s2, exitOption
                # invalid character print an error message
                li $v0, 4
```

```
        la $a0, errorOptionsMsg
        syscall
        j menu
        # make function calls to the subprograms
        option1:
                # Request the string
                la $a0, promptOctal
                li $v0, 4
                syscall
                la $a0, octalAddress
                li $a1, 21 # max 21 bytes to read
                li $v0, 8 # read octal number as string
                sw $a0, octalAddress
                syscall
                # load the address
                la $a0, octalAddress
                jal convertToDec
                # print the decimal value
                move $a0, $v0
                li $v0, 1
                syscall
                # continue
                j continue
        option2:
                # read user input
                la $a0, promptReverse
                li $v0, 4
                syscall
                li $v0, 5
                syscall
                move $a0, $v0
                jal reverseNumber
                move $a0, $v0 # display the hex reversed
                li $v0, 34
                syscall
                j continue
        continue:
                # Display menu again
                j menu
        exitOption:
                # deallocate space
callocMenu:     # deallocate the stack space
        lw $ra, 0($sp)
        lw $s3, 4($sp)
        lw $s2, 8($sp)
        lw $s1, 12($sp)
        lw $s0, 16($sp)
        addi $sp, $sp, 20
        jr $ra # return main


## Function for displaying the options
```

```
display:
	li $v0, 4
	la $a0, menuMsg
	syscall
	la $a0, convertToDecMsg
	syscall
	la $a0, reverseNumberMsg
	syscall
	la $a0, exitMsg
	syscall
	jr $ra # back to main

## Function converts octave number to decimal equivalent
convertToDec:
	malloc:# allocate the stack space
		subi $sp, $sp, 20
		sw $s0, 16($sp)
		sw $s1, 12($sp)
		sw $s2, 8($sp)
		sw $s3, 4($sp)
		sw $ra, 0($sp)
	# find end of the string
	# s0 contains address of the last char after stringEnd
	stringEnd:
		move $s0, $a0 # s0: address of the string
		nextChar:
			lb $s1, 0($s0) # s1: current char
			blt $s1, 10, foundEnd # "enter" (ASCII : 10)"
			addi $s0, $s0, 1 # goto next char
			j nextChar
		foundEnd:
				subi $s0, $s0, 2 # excluding enter (ASCII: 10)
	li $s2, 1 # go from lsd(least sig digit) to msd(most sig digit)
	li $s3, 8
	li $v0, 0 # will contain the result
	calcDec:
		# check beginning of string reached or not
		blt $s0, $a0, finish
		lb $s1, 0($s0) # load the octal character
		bgt $s1, 56, notValidOctal # digit > 7 (not valid octal value)
		blt $s1, 48, notValidOctal # digit < 0 (not valid octal value)
		asciiToDec:
			subi $s1, $s1, 48 # get the decimal value
			mul $s1, $s2, $s1 # adjust
		# add to the result
		add $v0, $v0, $s1
		# decrement the address go to the next char in string
		subi $s0, $s0, 1
		mul $s2, $s2, $s3 # 8**(digit)
		j calcDec
	notValidOctal:
		# raise an error
```

```
        li $v0, -1
    finish:
        # obtained the result in v0
    calloc:  # deallocate the stack space
        lw $ra, 0($sp)
        lw $s3, 4($sp)
        lw $s2, 8($sp)
        lw $s1, 12($sp)
        lw $s0, 16($sp)
        addi $sp, $sp, 20
    jr $ra # return main

# Reverse the decimal value  obtained from the user
reverseNumber:
    # allocate stack
    addi $sp, $sp, -16
    sw $s0, 12($sp)
    sw $s1, 8($sp)
    sw $s2, 4($sp)
    sw $s3, 0($sp)
    # Load array with decimal value
    move $s1, $a0
    li      $s0, 0   # temporary
    li $s3, 0 # contains the reversed val
    li      $s2, 15 # $ Mask
    # Reverse the number
    next:
        beq     $s1, $zero      end # check whether any value left
        and     $s0, $s1, $s2 # obtain the current integer
        or $s3, $s3, $s0 # write to reversed
        srl $s1, $s1, 4 #obtain the next decimal val
        sll $s3, $s3, 4 #open place for new dec value
        li $s0, 0 # reset temp
        j next
    end:
        srl $s3, $s3, 4 # reverse the extra shift
        move $v0, $s3 # return the result
    #restore registers back
    lw $s3, 0($sp)
    lw $s2, 4($sp)
    lw $s1, 8($sp)
    lw $s0, 12($sp)
    addi $sp, $sp, 16
    jr $ra # return main
```

```
#                    * data segment*

            .data
octalAddress: .space 8
intro: .asciiz "\nPROGRAM: Interactive menu for performing number conversions\n"
menuMsg: .asciiz "\n*** MENU ***\n"
convertToDecMsg: .asciiz "a) Convert user input(octal) to decimal\n"
reverseNumberMsg: .asciiz "b) Reverse the user input(hex)\n"
promptOctal: .asciiz "\nPlease enter the octal value(7 digits max): "
decResultMsg: .asciiz "\nThe decimal equivalent: "
exitMsg: .asciiz "c) Exit menu\n"
errorOptionsMsg: .asciiz "\nERROR: Please choose a valid menu option!!!\n"
promptReverse: .asciiz "\nEnter the integer you want to reverse: “
```

Solution of the Question 2

Address of the label *again* starts from 10 01 00 40_{16}.
There are 9 instructions until *next* label
Address of the *next* label 0x10 01 00 40 + 4 * 9 = 10 01 00 76_{16}

There are 4 *add* instructions until the first conditional branch *beq*. Each add instruction is 4 words.
Beq uses pc-relative addressing like other conditional branch instructions. It increments the program counter by immediate value + 4
The immediate of *beq* can calculated as following: PC = PC + 4 + immediate
Label *next* is 4 instructions past instruction *beq* so immediate = 4
Address of beq: 0x10010056
Beq is an I-type instruction; opcode: 4_{hex} | rs: 8 = $t0| rt: 9 = $t1| immediate: 4
Generating object code:
000100 01000 01001 0000 0000 0000 0100
0001 0001 0000 1001 0000 0000 0000 0100
0x11090004

Bne uses pc relative address as well.
Address of bne: 0x10010060
Thus, in order to go to the label *again* immediate value should be “-7” calculated according to PC = PC + 4 + immediate.
Bne is an I-type instruction; opcode: 5_{hex} | rs: 10 = $t2| rt: 11 = $t3| immediate: -7
Generating object code:
000101 00010 00011 1111 1111 1111 1001
0001 0100 0100 0011 1111 1111 1111 1001
0x1443FFF9

Jump instruction *j* uses direct addressing. Least sig 2 bits are 0 since addresses are word aligned.
Most sig 4 bits taken from PC’s most sig bits. Thus, 32-bit address obtained.
j instruction is J-type with an opcode 2_{hex}. Stores 26-bit immediate value
Generating object code:
000010 0000 0000 0001 0000 0000 0100 00
0000 1000 0000 0000 0100 0000 0001 0000
0x8004010

jr is a R-type instruction. Jumps to the 32-bit address held at register *rs*. Since we don't know address where this function called from we can't generate the object code. The register *ra* supposed to contain address of the next instruction.