CS224
Section No.: 2
Spring 2018
Lab No.: 1
EFE ACER / 21602217

1. Code for the first question:

```
#Preliminary Work - Question 1 - EFE ACER
#Initializes an array of a specified size, reads the items from the user, prints the
#array, reverses the array contents and prints the array again.

        .text
        .globl   __start
__start:
        #Printing an intro message
        la $a0, intro
        li $v0, 4
        syscall
        #Printing a prompt to read arraySize
        la $a0, prompt1
        li $v0, 4
        syscall
        #Reading the value of arraySize
        li $v0, 5
        syscall
        sw $v0, arraySize
        #Calling the readItems function
        jal readItems
        #Calling the printItems function
        jal printItems
        #Calling the reverseArrayContents function and printing the array again
        la $a0, 0($t0) #Passing the address of $t0 as an argument for reverseArrayContents
        jal reverseArrayContents
        la $a0, result2 #printing a message to inform the user that the contents are reversed
        li $v0, 4
        syscall
        jal printItems
        #Stopping the execution
        li $v0, 10
        syscall

#Subprogram to read arraySize items to the array
readItems:
        la $t0, array #$t0 points to the base address of the array
        lw $t1, arraySize #$t1 is set to arraySize (counter for the repeat structure)
        la $a0, prompt2 #printing a prompt to read items one by one
```

```
        li $v0, 4
        syscall
        readItem:
                li $v0, 5 #reading a single item to the current indexed position
                syscall
                sw $v0, 0($t0)
                addi $t0, $t0, 4 #increments the pointer to the next address
                subi $t1, $t1, 1 #decrementing the counter
                bgt $t1, $zero, readItem #repeat until $t1(counter) > 0
        jr $ra #returning to the main program

#Subprogram to print array items
printItems:
        la $t0, array #$t0 points to the base address of the array
        lw $t1, arraySize #$t1 is set to arraySize (counter for the repeat structure)
        la $a0, result1 #printing a message to inform the user
        li $v0, 4
        syscall
        printItem:
                beq $t1, 1, printWithoutSeparator #if counter == 1 skip if block
                lw $a0, 0($t0) #printing the current indexed item
                li $v0, 1
                syscall
                la $a0, separator #printing the separator after the item (if block)
                li $v0, 4
                syscall
                j continue #skipping the else block
                printWithoutSeparator: #printing the item without a separator (else block)
                        lw $a0, 0($t0) #printing the current indexed item
                        li $v0, 1
                        syscall
                continue:
                        addi $t0, $t0, 4 #increments the pointer to the next address
                        subi $t1, $t1, 1 #decrementing the counter
                        bgt $t1, $zero, printItem #repeat until $t1(counter) > 0
        jr $ra #returning to the main program

#Subprogram to reverse array contents
reverseArrayContents:
        subi $t0, $a0, 4 #$t0 points to the last item's address in the array
        la $t1, array #$t1 points to the base address of the array
        reverseItems:
                #Swapping two items pointed by $t0 and $t1
                lw $t2, 0($t1) #base adress to $t2
                lw $t3, 0($t0) #last adress to $t3
                sw $t2, 0($t0) #word in $t2 to $t0
                sw $t3, 0($t1) #word in $t3 to $t1
```

```
		#Computing the difference between the pointers and modifying them
		sub $t2, $t0, $t1
		subi $t0, $t0, 4
		addi $t1, $t1, 4
		bge $t2, 4, reverseItems #repeat until $t0 - $t1 < 4
	jr $ra #returning to the main program

#The data segment
		.data
array:		.space	80 #80/4 = 20 words (a word is 4 bytes)
arraySize:	.word	0
intro:		.asciiz	"The program initializes, reverses and prints an array created by the user.\n"
prompt1:	.asciiz	"Please enter the array size (must be between 1 and 20):\n"
prompt2:	.asciiz	"Please enter the items one by one:\n"
result1:	.asciiz	"The array contents:\n"
result2:	.asciiz	"\nArray contents are reversed.\n"
separator:	.asciiz	", "
```

2. Code for the second question:

```
#Preliminary Work - Question 2 - EFE ACER
#Evaluates the expression x = (c - d) % 2 without using div instruction

	.text
	.globl	__start
__start:
	#Printing an intro message
	la $a0, intro
	li $v0, 4
	syscall
	#Printing a prompt to read c
	la $a0, prompt1
	li $v0, 4
	syscall
	#Reading the value of c
	li $v0, 5
	syscall
	sw $v0, c
	#Printing a prompt to read d
	la $a0, prompt2
	li $v0, 4
	syscall
	#Reading the value of d
	li $v0, 5
	syscall
	sw $v0, d
```

```
        #Calling the calculate function with arguments c and d
        lw $a0, c
        lw $a1, d
        jal calculate
        sw $v0, x
        #Printing a message to display x
        la $a0, result
        li $v0, 4
        syscall
        #Printing the result value x
        lw $a0, x
        li $v0 1
        syscall
        #Stopping the execution
        li $v0, 10
        syscall

#Subprogram to evaluate x = (c - d) % 2 without using div
calculate:
        sub $t0, $a0, $a1
        bge $t0, $zero, else #if (c - d) >= 0 branch to else
        modNegative: #process to find the modulo of a negative number
                addi $t0, $t0, 2 #repeatedly add 2 to $t0
                blt $t0, $zero, modNegative #repeat until $t0 becomes positive
        j return #skip the else block
        else:
                modPositive: #process to find the modulo of a positive number
                        subi $t0, $t0, 2 #repeatedly subtract 2 from $t0
                        bge $t0, 2, modPositive #repeat until $t0 < 2

        return:
                move $v0, $t0 #set $v0 to the remainder
                jr $ra #returning to the main program

#The data segment
            .data
intro:      .asciiz   "The program evaluates the expression x = (c - d) % 2.\n"
prompt1:    .asciiz   "Please enter the value of c:\n"
prompt2:    .asciiz   "Please enter the value of d:\n"
result:     .asciiz   "The value of x is: "
x:          .word    0
c:          .word    0
d:          .word    0
```

3. Object codes in hex:

Explanation for memory allocation:

.data starts from the address: 0x10010000
asciiz "\nHello\n" occupies 8 bytes of space, since \n is 1 byte and null character (asciiz means null terminated string) is 1 byte.
Thus, base address of a is 0x10010000 + 0x00000008 = 0x10010008.
a is an array of 4 words, so it occupies 4 x 4 (a word is 4 bytes) = 16 bytes of space.
Thus, address of b is 0x10010008 + 16 = 0x10010018

Explanation for machine instruction:

la $t1, a -> is indeed:
lui $at, 0x1001 -> 1
ori $t1, $at, 8 -> 2

> 1 is an I-type instruction with:
> opcode of lui = 0xF, R[rs] = 0, $at = $1 = R[rt] = 1, imm = 0x1001
> Thus, machine instruction for 1 is:
> 001111 00000 00001 0001 0000 0000 0001 -> 0x3C011001
>
> 2 is an I-type instruction with:
> opcode of ori = 0xD, R[rs] = $at = $1 = 1, R[rt] = $t1 = $9 = 9, imm = 8
> Thus, machine instruction for 1 is:
> 001101 00001 01001 0000 0000 0000 1000 -> 0x34290008

la $t2, b -> is indeed:
lui $at, 0x1001 -> 3
ori $t2, $at, 18 -> 4

> 3 is an I-type instruction with:
> opcode of lui = 0xF, R[rs] = 0, $at = $1 = R[rt] = 1, imm = 0x1001
> Thus, machine instruction for 1 is:
> 001111 00000 00001 0001 0000 0000 0001 -> 0x3C011001
>
> 4 is an I-type instruction with:
> opcode of ori = 0xD, R[rs] = $at = $1 = 1, R[rt] = $t2 = $10 = 10, imm = 24
> Thus, machine instruction for 1 is:
> 001101 00001 01010 0000 0000 0001 1000 -> 0x342A0018

Hence, the object code is:
0x3C011001
0x34290008
0x3C011001
0x342A0018

4. Definitions and examples for:

a. Symbolic machine instruction:

Is the symbolic format corresponding to a certain machine instruction. Since, reading machine instructions is a tedious process for humans, this symbolic format is preferred to define an executable machine instruction.
Examples:
andi $t3, $t2, 38
add $t0, $s4, $s5

b. Machine instruction:

Are patterns of bits (only 1's and 0's) that the digital system (computer hardware) can translate to certain encoded operations.
Examples:
0011 0001 0100 1011 0000 0000 0010 0110 -> 0x314B0026 (andi $t3, $t2, 38)
0000 0010 1001 0101 0100 0000 0001 0100 -> 0x02954014 (add $t0, $s4, $s5)

c. Assembler directive:

Are statements that are not executed but tell the assembler what to with the symbolic machine instructions. They do not correspond to any machine code or do not contribute to the program size. They basically help the assembler to perform specific tasks in the assembly phase.
Examples:
.text
.data

d. Pseudo instruction:

Are instructions to do more complicated tasks that the instruction set is incapable of. Those instructions are performed by the simpler instructions in the instruction set.
Examples:
la $t1, ($t2) (uses lei and ori)
abs $t1, $t2 (uses an algorithm from Hacker's Delight to set $t1 to absolute value of $t2)