CS224
Section No.: 1
Fall 2019
Lab No. 1
Berdan Akyürek/21600904

PRELIMINARY REPORT

1. Input an Array:

```
#create array
.data
	prompt: .asciiz "Enter no of elements: "
	promptarr: .asciiz "Enter input for array: "
	space: .asciiz " "
	newline: .asciiz "\n"
	arr: .space 80
	arr2: .space 80
.text

# array base address = $s0
# index = $t0
# number of elements = $s1
# current element = $t1



# print enter numbers

li $v0, 4
la $a0, prompt
syscall

# take input from user

li $v0, 5
syscall

# store input in $s0

move $s1, $v0

addi $t0, $zero, 0

# $s1 times take an input for array with loop and store results in array

loop:
beq $t0, $s1, exit

li $v0, 4
la $a0, promptarr
```

```
syscall

li $v0, 5
syscall

move $t1, $v0

sb $t1 arr($t0)

addi $t0, $t0, 1

j loop

exit:

# print array using loop

addi $t0, $zero, 0

loop2:

beq $t0, $s1, exit2

lb $t1, arr($t0)

li $v0, 1
move $a0, $t1
syscall

li $v0, 4
la $a0, space
syscall

addi $t0, $t0, 1

j loop2

exit2:

li $v0, 4
la $a0, newline
syscall

# reverse array

# $t0 = index for arr
# $t2 = temporary element while turning
# $t3 = index for arr2

addi $t0, $zero, 0
add $t3, $zero, $s1
addi $t3, $t3, -1
```

```
loop3:

beq $t0, $s1, exit3

lb $t1, arr($t0)
sb $t1, arr2($t3)

addi $t0, $t0, 1
addi $t3, $t3, -1
j loop3

exit3:



# print arr2 (reversed array)

addi $t0, $zero, 0

loop4:

beq $t0, $s1, exit4

lb $t1, arr2($t0)

li $v0, 1
move $a0, $t1
syscall

li $v0, 4
la $a0, space
syscall

addi $t0, $t0, 1

j loop4

exit4:
```

2. Palindrome:

```
.data
        prompt: .asciiz "Enter no of elements: "
        promptarr: .asciiz "Enter input for array: "
        space: .asciiz " "
        newline: .asciiz "\n"
        pal: .asciiz "This array is a palindrom."
        notpall: .asciiz "This array is NOT a palindrom."
        arr: .space 80
        arr2: .space 80
.text

# print enter numbers

li $v0, 4
la $a0, prompt
syscall

# take input from user

li $v0, 5
syscall

# store input in $s0

move $s1, $v0

addi $t0, $zero, 0

# $s1 times take an input for array with loop and store results in array

loop:
beq $t0, $s1, exit

li $v0, 4
la $a0, promptarr
syscall

li $v0, 5
syscall

move $t1, $v0

sb $t1 arr($t0)

addi $t0, $t0, 1

j loop

exit:

#reverse array and store in arr2

# reverse array

# $t0 = index for arr
# $t2 = temporary element while turning
# $t3 = index for arr2

addi $t0, $zero, 0.data
```

```
        prompt: .asciiz "Enter no of elements: "
        promptarr: .asciiz "Enter input for array: "
        space: .asciiz " "
        newline: .asciiz "\n"
        pal: .asciiz "This array is a palindrom."
        notpall: .asciiz "This array is NOT a palindrom."

        arr: .space 80
        arr2: .space 80
.text

# print enter numbers

li $v0, 4
la $a0, prompt
syscall

# take input from user

li $v0, 5
syscall

# store input in $s0

move $s1, $v0

addi $t0, $zero, 0

# $s1 times take an input for array with loop and store results in array

loop:
beq $t0, $s1, exit

li $v0, 4
la $a0, promptarr
syscall

li $v0, 5
syscall

move $t1, $v0

sb $t1 arr($t0)

addi $t0, $t0, 1

j loop

exit:

#reverse array and store in arr2
```

```
# reverse array

add $t3, $zero, $s1
addi $t3, $t3, -1

loop2:

beq $t0, $s1, exit2

lb $t1, arr($t0)
sb $t1, arr2($t3)

addi $t0, $t0, 1
addi $t3, $t3, -1
j loop2

exit2:

# check reversed and normal array

addi $t0, $zero, 0

lb $t1, arr($t0)
lb $t4, arr2($t0)

loop3:

beq $t0, $s1, exit3

bne $t1, $t4, notpal

addi $t0, $t0, 1
j loop3

exit3:

#print palindrom

li $v0, 4
la $a0, pal
syscall

j end

notpal:
#print NOT palindrom

li $v0, 4
la $a0, notpall
syscall

end:
```

3. Perform Division Without Division Instruction:

```
.data
        askdivident: .asciiz "Enter divident: "
        askdivisor: .asciiz "Enter divisor: "
        quot: .asciiz "Quotient: "
        rema: .asciiz "Remainder: "
        newline: .asciiz "\n"

.text

# $s0 = divident
# $s1 = divisor
# $s2 = quotient
# $s3 = remainder

# ask for divident

li $v0, 4
la $a0, askdivident
syscall

# store divident in $s0

li $v0, 5

syscall
move $s0, $v0

# ask for divisor

li $v0, 4
la $a0, askdivisor
syscall

# store divisor in $s1

li $v0, 5

syscall
move $s1, $v0

# copy divident to $t0 to change

addi $t0, $s0, 0

# set quotient to 0
```

```
addi $s2, $zero, 0

# division

loop:

slt $t1, $t0, $s1 # if divident > divisor, $t1 = 0
beq $t1, 1, exit  # if divident < divisor, exit

addi $s2, $s2, 1
sub $t0, $t0, $s1

j loop

exit:

addi $s3, $t0, 0


# print the quotient

li $v0, 4
la $a0, quot
syscall

li $v0, 1
move $a0, $s2
syscall

# print new line

li $v0, 4
la $a0, newline
syscall

# print the remainder

li $v0, 4
la $a0, rema
syscall

li $v0, 1
move $a0, $s3
syscall
```

4. Object Code Generation:

add $t0, $t1, $t2
Binary: 00000001001010100100000000100000
Hex: 0x012A4020

addi $s0, $s3, 15
Binary: 00100010011100000000000000001111
Hex: 0x2270000F

mult $a0, $a1
Binary: 00000000100001010000000000011000
Hex: 0x00850018

sw $t1, 8($t2)
Binary: 10101101010010010000000000001000
Hex: 0xAD490008

lw $t2, 8($t1)
Binary: 10001101001010100000000000001000
Hex: 0x8D2A0008

5. Define Terms:

Symbolic machine instruction: An instruction that is understandable by human and convertible to machine code easily.

Examples:
add $t0, $t1, $t2
sub $t0, $t1, $t2

Machine instruction: An instruction that consists of 1’s and 0’s. They can be processed by computer but hard to understand by human.

Examples:
00000000100001010000000000011000 = mult $a0, $a1
10101101010010010000000000001000 = sw $t1, 8($t2)

Assembler directive: A directive that is used by assembler to process the program right way.

Examples:
.data
.text

Pseudo instruction: An instruction that cannot be directly translated to machine instruction but can be first translated to Symbolic instructions, then to machine instructions.

Examples:
clear $t0 = add $t0, $0, $0
move $s1, $s2 = add $s2, $s1, $0