CS224
Section No.: 01
Spring 2018
Lab No.: 2
Deniz Yüksel / 21600880

PRELIMINARY DESIGN REPORT

**Part 1-a**. **(10 points) convertHexToDec**:

```
        # $a0 = address of the first char in hexNo string.
        # $s1 = use for traversing the string.
        # $s2 = string length.
        # $s3 = exponent for taking power.
        # $s7 =
        # $t2 = compared with s2 for other traversals.
        # $t7 = backup address of $a0.



        .text
        .globl __start
__start:

        la $a0, hexNo
        li $s4, 0
        li $s2, 0
        addi $s3, $0, 1
        jal convertHexToDec
        add $s6, $0, $v0        # result comes in $v0
        addi $v0,$0,1
        add $a0,$0,$s6
        syscall
        li $v0,10               # Terminate
    syscall
```

```
convertHexToDec:


        add $s7, $ra, $0
        sub $sp, $sp, 4
        sw $s7, 0($sp)          # Store the initial value of $s7.
        add $t7, $0 , $a0       # t7 for reserving a0.
        jal traverse
        add $s6, $0, $v1        # store v1 into s6 for backup.
        sub $s6, $s6, 2         # subtract 2 from s6, last char was null char and the pointer is now
pointing the right of the null char.
        sub $s2, $s2, 1         # subtract 1 from the length for correcting it.
        addi $t2, $0, 0         # same job with $s2, counter. Now $s2 has the correct length.
        jal subtractAscii
        lw $s7, 0($sp)          # Load back the initial value of $s7.
        add $ra, $0, $s7        # Recover $ra.
        jr $ra


traverse:


        lb $s1, 0($t7)          # Load s1 with the first element of t7, which is the first char.
        addi $s2, $s2, 1 # increment s2, which is the count.
        addi $t7, $t7, 1 # skip to the next char.
        bnez $s1, traverse      # if s1 is not the zero char, stay in the loop.
        add $v1, $0, $t7        # result will come in v1.
        jr $ra                  # go back to line 30.
subtractAscii:
        lb $s1, 0($s6)
        bltu $s1, 48, error
        bleu $s1, 57, doNumbers
        bltu $s1, 65, error
        bleu $s1, 70, doLetters
```

```
        bge  $s1, 70, error
doNumbers:
        addi $s1, $s1, -48
        b continue
doLetters:
        addi $s1, $s1, -55

continue:
        bgt $s1, 15, error
        mul $s1, $s1, $s3       # hexNo[2] * 1, hexNo[1] * 16, hexNo[0] * 64...
        addi $t2, $t2, 1 # add to count.
        add $s4, $s4, $s1       # add to sum.
        addi $s6, $s6, -1       # Decrement the pointer.
        sll $s3, $s3, 4         # shift left 4 times. 1 -> 16 -> 64
        bne $t2, $s2, subtractAscii
        addi $v0 , $s4, 0
        jr $ra
error:
        li $v0, 4
        la $a0, strNotProper
        syscall
        addi $v0, $0, -1
        jr $ra

        .data
strNotProper: .asciiz "ERROR! Result is "
hexNo: .asciiz "1A"
```

**Part 1-b**. **(10 points) interactWithUser**:

```
        # $a0 = address of the first char in hexNo string.
        # $s1 = use for traversing the string.
        # $s2 = string length.
```

```
	# $s3 = exponent for taking power.
	# $s7 = For remembering $ra.
	# $t2 = compared with s2 for other traversals.
	# $t7 = backup address of $a0.



	.text
	.globl __start
__start:


interactWithUser:
	# Get a string from the user.


	li $v0, 4
	la $a0, request
	syscall


	li $v0, 8
	la $a0, strBuffer
	li $a1, 20		# This number -1 will be the length of the array. In this case, if user
#enters 19 elements, the program will continue without hitting enter button.
	syscall
	li $s4, 0
	li $s2, 0
	addi $s3, $0, 1
	jal convertHexToDec
	add $s6, $0, $v0	# result comes in $v0
	addi $v0,$0,1
	add $a0,$0,$s6
	syscall
	li $v0,10		# Terminate
```

```
    syscall
convertHexToDec:
        add $s7, $ra, $0
        sub $sp, $sp, 4
        sw $s7, 0($sp)          # Store the initial value of $s7.
        add $t7, $0 , $a0       # t7 for reserving a0.
        jal traverse
        add $s6, $0, $v1        # store v1 into s6 for backup.
        sub $s6, $s6, 3         # subtract 3 from s6, when the user inputs a string, last two chars are
10 and NULL char.
        sub $s2, $s2, 2         # subtract 2 from the length for correcting it.
        addi $t2, $0, 0         # same job with $s2, counter. Now $s2 has the correct length.
        jal subtractAscii
        lw $s7, 0($sp)          # Load back the initial value of $s7.
        add $ra, $0, $s7        # Recover $ra.
        jr $ra
traverse:
        lb $s1, 0($t7)          # Load s1 with the first element of t7, which is the first char.
        addi $s2, $s2, 1 # increment s2, which is the count.
        addi $t7, $t7, 1 # skip to the next char.
        bnez $s1, traverse      # if s1 is not the zero char, stay in the loop.
        add $v1, $0, $t7        # result will come in v1.
        jr $ra                  # go back to line 30.
subtractAscii:
        lb $s1, 0($s6)
        bltu $s1, 48, error
        bleu $s1, 57, doNumbers
        bltu $s1, 65, error
        bleu $s1, 70, doLetters
        bge  $s1, 70, error
doNumbers:
```

```
        addi $s1, $s1, -48
        b continue
doLetters:
        addi $s1, $s1, -55
continue:
        bgt $s1, 15, error
        mul $s1, $s1, $s3       # hexNo[2] * 1, hexNo[1] * 16, hexNo[0] * 64...
        addi $t2, $t2, 1 # add to count.
        add $s4, $s4, $s1       # add to sum.
        addi $s6, $s6, -1       # Decrement the pointer.
        sll $s3, $s3, 4         # shift left 4 times. 1 -> 16 -> 64
        bne $t2, $s2, subtractAscii
        addi $v0 , $s4, 0
        jr $ra
error:
        li $v0, 4
        la $a0, strNotProper
        syscall
        j interactWithUser
        .data
strNotProper:   .asciiz "ERROR! \n "
request:        .asciiz " Please enter an hexadecimal number: \n "
strBuffer:      .space 100
```

**Part 2 (10 points)**:

Object code for the first jump ( j next):

Jump is a J type of instruction so its opcode is 00010. The remaining 26 bits is the address of “next” label. We calculated the address of next from the code, each instruction in this code are 4 bytes. So, the address of “next” is 1001038 if we assume the address of “previous” is 100101C. Then we write the address of next as 4 bits for each bit in hexadecimal form, which translates into ( with the opcode);

000010 0001 0000 0000 0001 0000 0011 1000.

Then, we group the bits four by four and acquire 0840040E, with 00 at the end standing as two bits. We leave the last two zeros and acquire this code.

Code for j previous: It is generated in the same way I explained above. This time, we evaluate j previous and will use the address of “previous” label. Opcode of j in hex is 2. The address of previous is 100101C. Thus, the 32 bit code of this instruction is,

000010 0001 0000 0000 0001 0000 0001 1100.

We group four by four to acquire the machine instruction: 08400407.