CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

CS 224 – Spring 2019 – Lab #2

MIPS Assembly Language Programming Using Subprograms

Preliminary Design Report

Berrak Taşkınsu / 21602054

Section No.: 2

CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

Part 1. Preliminary Work / Preliminary Design Report

1. Write MIPS assembly language programs as described below.
The main program calls the subprogram interactWithUser and then stops.

a. (2 points) interactWithUser: Write a subprogram, called interactWithUser, that provides three menu options.

b. (10 points) convertToDec: Write a subprogram, called convertToDec, that receives the beginning address of an asciiz string that contains a octal number in the form of a string, for example, like "14", and returns its decimal ($14_8 = 12_{10}$) equivalent in register $v0.

c. (10 points) reverseNumber: Write a subprogram, called reverseNumber, that receives a decimal number (in $a0) and reverses its bytes and returns as its result (in $v0). For example, if the number received is AABBCCDD in hex it returns DDCCBBAA. For hex display see the related syscall. In the implementation of reverseNumber use *shift* and logical bit manipulation instructions such as *and* etc. as needed.

```
	.text
main:
	jal interactWithUser
	li $v0, 10
	syscall

interactWithUser:
	addi $sp, $sp, -12
	sw $s1, 8($sp)
	sw $s0, 4($sp)
	sw $ra, 0($sp)
	printMenu:
		li $v0, 4
		la $a0, menu
		syscall
		li $v0, 5
		syscall
		move $s0, $v0
		beq $s0, 1, sub1
		beq $s0, 2, sub2
		quit:
			lw $s1, 8($sp)
			lw $s0, 4($sp)
			lw $ra, 0($sp)
			addi $sp, $sp, 12
			jr $ra
		sub1:
			jal convertToDec
			b printMenu
		sub2:
```

CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

```
                li $v0, 4
                la $a0, prom2
                syscall
                li $v0, 5
                syscall
                move $s1, $v0
                move $a0, $s1
                li $v0, 34
                syscall
                li $v0, 4
                la $a0, nline
                syscall
                move $a0, $s1
                jal reverseNumber
                move $a0, $v0
                li $v0, 34
                syscall
                li $v0, 4
                la $a0, nline
                syscall
                b printMenu

convertToDec:
	addi $sp, $sp, -24
	sw $s1, 20($sp)
	sw $s2, 16($sp)
	sw $s3, 12($sp)
	sw $s4, 8($sp)
	sw $s5, 4($sp)
	sw $ra, 0($sp)

	# Conversion from octal to decimal
	li $v0, 4
	la $a0, prom1
	syscall
	li $v0, 8
	la $a0, str
	li $a1, 20
	syscall
	move $s0, $a0
	# s1: string length
	loop:
		lb $s2, 0($s0)
		beqz $s2, stop
		addi $s1, $s1, 1
		addi $s0, $s0, 1
		b loop
	stop:
	addi $s0, $s0, -2
	addi $s1, $s1, -1
	loop2:
```

CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

```
            beq $s3, $s1, end
            lb $s2, 0($s0)
            addi $s2, $s2, -48
            move $s4, $zero
            loop3:
                  beq $s4, $s3, next
                  addi $s4, $s4, 1
                  mul $s2, $s2, 8
                  b loop3
            next:
            add $s5, $s5, $s2
            addi $s3, $s3, 1
            addi $s0, $s0, -1
            b loop2
      end:
      li $v0, 1
      move $a0, $s5
      syscall
      li $v0, 4
      la $a0, nline
      syscall
      #la $s0, str
      #add $s0, $s0, $s1
      lw $s1, 20($sp)
      lw $s2, 16($sp)
      lw $s3, 12($sp)
      lw $s4, 8($sp)
      lw $s5, 4($sp)
      lw $ra, 0($sp)
      addi $sp, $sp, 24
      jr $ra

reverseNumber:
      addi $sp, $sp, -20
      sw $s1, 16($sp)
      sw $s2, 12($sp)
      sw $s3, 8($sp)
      sw $s4, 4($sp)
      sw $ra, 0($sp)

      move $s4, $a0
      # $s4: number

      rem $s1, $s4, 256 # 1st byte
      srl $s4, $s4, 8
      rem $s2, $s4, 256 # 2nd byte
      srl $s4, $s4, 8
      rem $s3, $s4, 256 # 3rd byte
      srl $s4, $s4, 8   # 4th byte

      sll $s1, $s1, 24
```

CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

```
      sll $s2, $s2, 16
      sll $s3, $s3, 8

      or $s1, $s1, $s2
      or $s1, $s1, $s3
      or $s1, $s1, $s4
      move $v0, $s1

      lw $s1, 16($sp)
      lw $s2, 12($sp)
      lw $s3, 8($sp)
      lw $s4, 4($sp)
      lw $ra, 0($sp)
      addi $sp, $sp, 20
      jr $ra

      .data
str:  .space 20
menu: .asciiz "1) Convert an octal number to decimal.\n2) Reverse an hexadecimal number.\n3) Quit.\nEnter your choice: "
prom1:     .asciiz "Please enter an octal number: "
prom2:     .asciiz "Please enter a number: "
nline:     .asciiz "\n"
```

2. (8 points) Generating machine instructions. Give the object code in hexadecimal for the be, bne, j, and jr instructions of the following code segment. Briefly show your work. Assume that the label again is located at memory location 10 01 00 40_{16}. If you think that you do not have enough information to generate the code, explain why.

```
            ... # some other instructions
again:
            add ... # there is an instruction here and meaning is insignificant
            add ... # likewise for the other similar cases
            add ...
            add ...
            beq   $t0, $t1, next
            bne   $t2, $t3, again
            add ...
            add ...
            jr    $ra
next:
            j     again
```

- beq (4hex) and bne (5hex) are both I-type instructions.

CS 224
Section No.: 2
Spring 2019
Lab 02
Berrak Taşkınsu / 21602054

The label next is located 5 instructions below this instruction. But, pc register already points to the next instruction, therefore the immediate value will be 4. This means pc = pc + 4 + 4(4).

beq $t0, $t1, next : 000100 | 01000 | 01001 | 0000 0000 0000 0100
The object code in hexadecimal is 0x11090004

The label again is located 5 instructions above this instruction. But, pc register already points to the next instruction, therefore the immediate value will be -6. This means pc = pc + 4 + 4(-6).

bne $t2, $t3, again : 000101 | 01010 | 01011 | 1111 1111 1111 1010
The objcet code in hexadecimal is 0x154BFFFA

- j (2hex) is a J-type instruction.

The again label is located at memory location 0x10010040. This corresponds to 0001 0000 0000 0001 0000 0000 0100 0000 in binary. This address is 32 bits, yet J-type instructions have 28 bits space for address. We eliminate the first 4 bits and the last two bits of the address and obtain 0000 0000 0001 0000 0000 0100 00.

j again : 000010 | 0000 0000 0001 0000 0000 0100 00
The object code in hexadecimal is 0x08004010

- jr (0/08hex) is an R_type instruction.

jr $ra : 000000 | 11111 | 00000 | 00000 | 00000 | 001000
The object code in hexadecimal is 0x03E00008