CS224

Section No.: 5

Spring 2017

Lab No. 2

Your Full Name/Bilkent ID: Berat Biçer 2150350

**1**. **(20 points)** Write MIPS assembly language programs as described below.

**a**. **(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 "17", and returns its decimal (178= 1510) equivalent in register $v0. If the number stored in the input string is not a proper octal number it returns a negative value in $v0. A sketch of this convertToDec program is as follows.

\_\_start:

...

la $a0, octalNo

jal convertToDec

# result comes in $v0

...

convertToDec:

.data

octalNo: .asciiz "17"

**b**. **(10 points)** **interactWithUser**: Write a subprogram, called interactWithUser, that asks the user enter an octal number in the form of a string, makes sure that it is a proper octal number if not generates an error message and ensures that a proper input is received. It passes this address to the subprogram defined above convertToDec, if necessary modify it, and gets the result from it and returns the decimal value back to the caller, i.e. the main program. The main program is the one that contains the label \_\_start . How to read a string: See MIPS system calls on the web to understand how to read a string. Invoke interactWithUser from the main program that contain the label \_\_start.

**Use the $s registers during the implementation of the above subprograms**. Using $s registers means that you have to save them to stack and restore them back from stack. (Question: Why do we use the $s registers? Answer: To have practice about the use of stack in MIPS.) Make sure that you also save/restore any other register that need to be saved.

**2**.  **(10 points)** Generating machine instructions   
(**Branch instructions 3 points each, jump instruction 4 points**) Give the object code in hexadecimal for the following branch (be, bne) and jump (j) instructions.

... # some other instructions

again: add ... # there is an instruction here and meaning is insignificant

add ... # likewise for the other similar cases

beq $t0, $t1, next

bne $t0, $t1, again

add ...

add ...

add ...

next: j again

Assume that the label again is located at memory location 10 01 00 2016. If you think that you do not have enough information to generate the code explain. See slide number 117 in the new Chap 6 slides of the textbook for the MIPS memory map (available at our unilica web site, Documents folder).

**Answers:**

**1-a) # IO**

**# Get a string as an octal number, convert it to base 10**

**.data**

**octal: .asciiz "457"**

**.text**

**la $a0, octal # input**

**addi $a1, $0, 3 # input size**

**jal convertToDec**

**addi $t0, $v0, 0**

**# Output**

**li $v0, 1**

**la $a0, ($t0)**

**syscall**

**# Exit**

**li $v0, 10**

**syscall**

**#----------------------------------------------------------------------------------------**

**convertToDec:**

**# Arguments: a0: string base address, a1: string size**

**addi $sp, $sp, -12**

**sw $a0, ($sp)**

**sw $a1, 4($sp)**

**sw $ra, 8($sp)**

**addi $s0, $a1, -1 # power count**

**interpret:**

**lb $s1, ($a0) # save char value into**

**beq $s1, 48, prelabel0 # ascii 0**

**beq $s1, 49, prelabel1 # ascii 1**

**beq $s1, 50, prelabel2 # ascii 2**

**beq $s1, 51, prelabel3 # ascii 3**

**beq $s1, 52, prelabel4 # ascii 4**

**beq $s1, 53, prelabel5 # ascii 5**

**beq $s1, 54, prelabel6 # ascii 6**

**beq $s1, 55, prelabel7 # ascii 7**

**beq $s1, 56, prelabel8 # ascii 8**

**beq $s1, 57, prelabel9 # ascii 9**

**j finish**

**prelabel0:**

**addi $s1, $0, 0**

**j label**

**prelabel1:**

**addi $s1, $0, 1**

**j label**

**prelabel2:**

**addi $s1, $0, 2**

**j label**

**prelabel3:**

**addi $s1, $0, 3**

**j label**

**prelabel4:**

**addi $s1, $0, 4**

**j label**

**prelabel5:**

**addi $s1, $0, 5**

**j label**

**prelabel6:**

**addi $s1, $0, 6**

**j label**

**prelabel7:**

**addi $s1, $0, 7**

**j label**

**prelabel8:**

**addi $s1, $0, 8**

**j label**

**prelabel9:**

**addi $s1, $0, 9**

**label: # Character in s1 and it's valid**

**addi $sp, $sp, -8**

**sw $a0, ($sp)**

**sw $a1, 4($sp)**

**addi $a0, $s0, 0 # power count**

**addi $a1, $s1, 0 # char in a1**

**addi $sp, $sp, -4**

**sw $v0, ($sp)**

**jal power**

**lw $s2, ($sp) # v0**

**addi $sp, $sp, 4**

**add $v0, $v0, $s2 # sum**

**lw $a0, ($sp)**

**lw $a1, 4($sp)**

**addi $sp, $sp, 8**

**addi $a0, $a0, 1 # base address**

**addi $a1, $a1, -1 # size**

**addi $s0, $s0, -1 # power**

**bne $a1, 0, interpret**

**finish :**

**lw $ra, 8($sp)**

**addi $sp, $sp, 12**

**jr $ra**

**# end of convertToDec**

**power: # a0 = x, a1 = num, v0 = result**

**beq $a0, 0, zero**

**loop: # calculate 8^x \* num**

**addi $t0, $0, 8**

**mul $a1, $t0, $a1**

**addi $a0, $a0, -1**

**bne $a0, 0, loop**

**addi $v0, $a1, 0 # load result into v0**

**j end**

**zero:**

**addi $v0, $a1, 0**

**end:**

**jr $ra**

**# End of power**

**1-B)**

**# asks the user enter an octal number in the form of a string, makes sure that it is a proper octal number**

**# if not generates an error message and ensures that a proper input is received.**

**# It passes this address to the subprogram defined above convertToDec,**

**# if necessary modify it, and gets the result from it**

**# and returns the decimal value back to the caller, i.e. the main program**

**.data**

**cmd: .asciiz "\nEnter an integer as octal number's digit count: "**

**cmd2: .asciiz "\nEnter an octal number with the size you specified before: "**

**cmd3: .asciiz "\n"**

**.text**

**start:**

**# Instruction**

**li $v0, 4 # print string**

**la $a0, cmd**

**syscall**

**# Dynamic Memory Allocation**

**li $v0, 5 # read int**

**syscall**

**addi $sp, $sp, -4**

**sw $v0, ($sp) # save string size**

**lw $t1, ($sp) # size -----> t1**

**li $v0, 9 # dynamic memory request**

**addi $t1, $t1, 1 # space for null character**

**la $a0, ($t1) # memory amount**

**syscall**

**addi $sp, $sp, -4**

**sw $v0, ($sp) # save base address**

**# Get input**

**li $v0, 4 # print string**

**la $a0, cmd2**

**syscall**

**li $v0, 8 # read string**

**lw $a0, ($sp)**

**la $a1, ($t1)**

**syscall # a0 has the string's base address, a1 has its size**

**lw $a1, 4($sp)**

**addi $a1, $a1, 0 # null character is irrelevant for method**

**addi $v0, $0, 0 # reset v0**

**jal convertToDec # a0: base address, a1: input size**

**move $s0, $v0 # save return value**

**# Output**

**li $v0, 4 # print string**

**la $a0, cmd3**

**syscall**

**li $v0, 1**

**la $a0, ($s0)**

**syscall**

**# Exit**

**addi $sp, $sp, 8 # restore sp**

**li $v0, 10**

**syscall**

**#----------------------------------------------------------------------------------------**

**convertToDec:**

**# Arguments: a0: string base address, a1: string size**

**addi $sp, $sp, -12**

**sw $a0, ($sp)**

**sw $a1, 4($sp)**

**sw $ra, 8($sp)**

**addi $s0, $a1, -1 # power count**

**interpret:**

**lb $s1, ($a0) # save char value into**

**beq $s1, 48, prelabel0 # ascii 0**

**beq $s1, 49, prelabel1 # ascii 1**

**beq $s1, 50, prelabel2 # ascii 2**

**beq $s1, 51, prelabel3 # ascii 3**

**beq $s1, 52, prelabel4 # ascii 4**

**beq $s1, 53, prelabel5 # ascii 5**

**beq $s1, 54, prelabel6 # ascii 6**

**beq $s1, 55, prelabel7 # ascii 7**

**beq $s1, 56, prelabel8 # ascii 8**

**beq $s1, 57, prelabel9 # ascii 9**

**j finish**

**prelabel0:**

**addi $s1, $0, 0**

**j label**

**prelabel1:**

**addi $s1, $0, 1**

**j label**

**prelabel2:**

**addi $s1, $0, 2**

**j label**

**prelabel3:**

**addi $s1, $0, 3**

**j label**

**prelabel4:**

**addi $s1, $0, 4**

**j label**

**prelabel5:**

**addi $s1, $0, 5**

**j label**

**prelabel6:**

**addi $s1, $0, 6**

**j label**

**prelabel7:**

**addi $s1, $0, 7**

**j label**

**prelabel8:**

**addi $s1, $0, 8**

**j label**

**prelabel9:**

**addi $s1, $0, 9**

**label: # Character in s1 and it's valid**

**addi $sp, $sp, -8**

**sw $a0, ($sp)**

**sw $a1, 4($sp)**

**addi $a0, $s0, 0 # power count**

**addi $a1, $s1, 0 # char in a1**

**addi $sp, $sp, -4**

**sw $v0, ($sp)**

**jal power**

**lw $s2, ($sp) # v0**

**addi $sp, $sp, 4**

**add $v0, $v0, $s2 # sum**

**lw $a0, ($sp)**

**lw $a1, 4($sp)**

**addi $sp, $sp, 8**

**addi $a0, $a0, 1 # base address**

**addi $a1, $a1, -1 # size**

**addi $s0, $s0, -1 # power**

**bne $a1, 0, interpret**

**finish :**

**lw $ra, 8($sp)**

**addi $sp, $sp, 12**

**jr $ra**

**# end of convertToDec**

**power: # a0 = x, a1 = num, v0 = result**

**beq $a0, 0, zero**

**loop: # calculate 8^x \* num**

**addi $t0, $0, 8**

**mul $a1, $t0, $a1**

**addi $a0, $a0, -1**

**bne $a0, 0, loop**

**addi $v0, $a1, 0 # load result into v0**

**j end**

**zero:**

**addi $v0, $a1, 0**

**end:**

**jr $ra**

**# End of power**

**2) BEQ: 0X11090004, BNE: 0x1509FFFC, J: 0X08004008**