Faculdade de Computação Arquitetura e Organização de Computadores 1

Lab 3

Integrantes:
Arthur Resende Santos - 12011BCC020
Davi de Pontes Pasquini - 12011BCC036
Henrique Braga Alves Pereira - 12011BCC017

P1. Escreva um programa em MIPS assembly, que implemente a função ***isdigit(c)***, que recebe como parâmetro de entrada **($a0)** um char e verifica se o caractere passado é um dígito numérico decimal ou não. A função deve retornar no registrador **$v0** o valor inteiro **1** se for um dígito, caso contrário, ele retorna **0**.
A função ***isdigit(c)*** deve ser declarada no programa MIPS assembly, verificando se o caractere inserido é um caractere numérico [0 – 9] ou não.
O caractere de teste deve ser obtido pela função ***getchar()*** definida abaixo, que emprega a técnica E/S programada com dispositivo mapeado em memória (MMIO) emulado no simulador MARS.

```
getchar: # retorna  char em $v0
     lw   $v0, 0xffff0000
     andi $v0, $v0, 0x01
     beq  $v0, $zero, getchar
     lw   $v0, 0xffff0004
     jr   $ra
```

.text

```
main:
   jal getchar # vai para getchar
   move $a0, $v0 # coloca o char inserido em $a0
   jal isdigit # vai para isdigit
   li $v0, 10 #return 0
   syscall

isdigit:
   sge $t1, $a0, 48 # se for maior ou igual a 48 (ASCII) $t1 = 1 else =0
   sle $t2, $a0, 57 # se for menor ou igual a 57 (ASCII) $t2 = 1 else = 0
   and $v0, $t1, $t2 #$v0 = 1 se $t1 e $t2 forem 1 else $v0 = 0
   move $s0, $v0
   jr $ra

getchar: # retorna char em $v0
   lw $v0, 0xffff0000
   andi $v0, $v0, 0x01
   beq $v0, $zero, getchar
   lw $v0, 0xffff0004
   jr $ra
```

P2. Escreva a função ***atoi*** *(ASCII to int)*, em linguagem de montagem do MIPS, que converta uma *string* decimal em ASCII para um número inteiro armazenado em registrador. Por convenção, o endereço da string numérica é passado via registrador **$a0**, o *string* é uma cadeia de dígitos numéricos [0 a 9]. A função deve transformar o *string* de char para a representação binária de 32 bits, colocando o valor de retorno no registrador **$v0**. Seu programa não deve se preocupar com números negativos. Se for identificado no *string* um caractere que não represente um dígito, seu programa deve retornar o valor -1 no registrador **$v0**. Por exemplo, se o registrador **$a0** aponta para uma sequência de char: "12345", então, quando a função ***atoi()*** terminar, o registrador **$v0** deve ter a representação binária do valor (00000000000000000011000000111001).

```
.data
        askString: .asciiz "Insira uma string de numeros decimais:\n"
        errorString: .asciiz "Caracter invalido!\n"

        inputString: .space 100
.text
        # insercao de string
        li $v0, 4
        la $a0, askString
        syscall

        # leitura da string
        li $v0, 8
        la $a0, inputString
        la $a1, 100
        syscall

        # armazenamento do valor
        li $s0, 0
        li $s2, 0

        # calculo do tamanho da string
        jal strLen # Value in s1

        # zera o contador
        li $s0, 0

        # agora calculamos o valor dos caracteres
        j atoiMain

        # calculo potencia
        pow: # armazena em s3
                # inicia o contador
                li $t0, 1

                # copia a2 para t1
                add $t1, $a2, 0
```

```
        beq $a3, 0, powZeroReturn

        j powLoop

powLoop:
        bge $t0, $a3, powReturn

        # a2 = a2 * t1
        mul $a2, $a2, $t1

        # modifica contador
        addi $t0, $t0, 1

        j powLoop

powReturn:
        # o resultado esta em s3
        move $s3, $a2

        # resume o programa
        jr $ra

powZeroReturn:
        li $s3, 1

        # Resume o programa
        jr $ra

# encontra a posicao do valor decimal e multiplica pelo char
atoiMain: # s4 armazena resultado
        # carrega o endereco da string
        la $a1, inputString
        addu $a1, $a1, $s0
        lbu $a0, ($a1)

        # confere se eh ‘/n’
        beq, $a0, '\n', atoiReturn

        # confere se eh o ultimo algarismo
        beq $a0, $zero, atoiReturn

        # confere se o valor nao extrapola o parametro 0-9
        blt $a0, '0', atoiCharError
        bgt $a0, '9', atoiCharError

        # como o valor eh char, subtraímos 48 para conseguirmos o valor em int
        subi $s7, $a0, 48

        #calcula posicao
        li $a2, 10
        subi $a3, $s1, 1 # a3 = s1 - 1 (position)
        jal pow


        subi $s1, $s1, 1
```

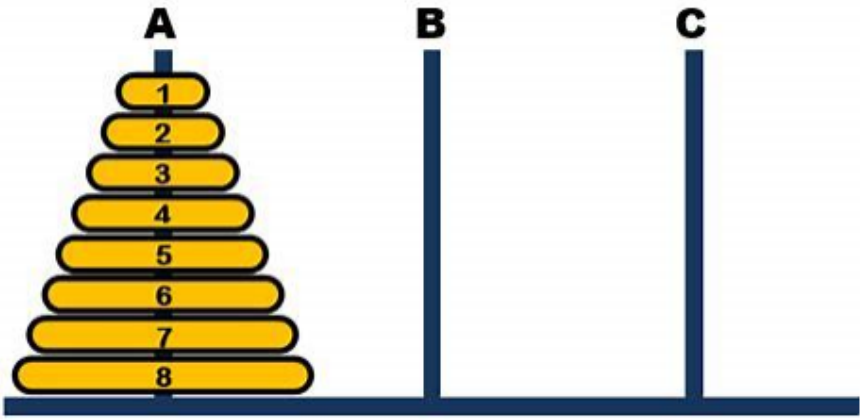
```
            # multiplica os numeros por posicao
            mul $s2, $s3, $s7 # s2 = s3 * a0

# armazena em s4
        add $s4, $s4, $s2 # s4 = s4 + s2


        addi $s0, $s0, 1

        # repete operacoes
        j atoiMain

atoiReturn:
            li $v0, 4
            la $a0, newLine
            syscall

            # imprime resultado
            li $v0, 1
            move $a0, $s4
            syscall

            # saida do programa
            li $v0, 10
            syscall

# para NaN
atoiCharError:
            # imprime o caracter erro
            li $v0, 11
            syscall

            li $v0, 1
            li $a0, -1
            syscall



            li $v0, 4
            la $a0, newLine
            syscall

            # mensagem de erro
            li $v0, 4
            la $a0, errorString
            syscall

            # encerra o programa
            li $v0, 10
            syscall

strLen:
            # carrega o endereco da string
            la $a1, inputString
```

```
        addu $a1, $a1, $s0
        lbu $a0, ($a1)

        # confere se eh o ultimo algarismo
        beq $a0, $zero, strLenReturn

        # altera contador
        addi $s0, $s0, 1
        j strLen

strLenReturn:
        # altera contador
        subi $s1, $s0, 1

        # resume o programa
        jr $ra
```

A ***Torre de Hanói*** é um dos mais famosos jogos de raciocínio matemático inventado pelo matemático francês Édouard Lucas (1842-1891). O jogo consiste em uma base contendo três pinos (hastes), em um dos quais está disposta uma torre formada por alguns discos colocados uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O número de discos pode variar. Observe a disposição inicial de um jogo com seis discos.
Esse jogo tem como objetivo deslocar todos os discos de um pilar para outro qualquer, obedecendo a duas regras:

1) Mover apenas um disco por vez.
2) Um disco com diâmetro maior nunca pode ficar sobre um disco com diâmetro menor.

Implemente em MIPS Assembly uma função recursiva que imprime todos os passos da sequência de movimentos que solucionam o problema, veja o código ilustrativo em linguagem C.




```c
#include <stdio.h>

void move(int n, char de, char via, char para)
{
  if (n > 1) {
    move(n - 1, de, para, via);
    printf("Move disco do pino %c para o pino %c\n", de, para);
    move(n - 1, via, de, para);
  } else {
    printf("Move disco do pino %c para o pino %c\n", de, para);
  }
}
int main()
{
     int n;
     printf("Numero de discos? ");
     scanf("%d",&n);
     move(n, 'A','B','C');
     return 0;
}
```

A lógica que tentamos usar nessa 3 foi de tentar traduzir diretamente o código dado do professor para o MIPS, porém mantendo em mente as 3 regras básicas para realizar uma função recursiva:

(1) - Caso base (n = 1) funcionar

(2) - Assumir que o caso (n - 1) funciona
(3) - Mostrar que o caso (n) funciona usando o caso (n - 1)

```
.data
NumDiscos: .asciiz "Numero de discos? "
MoveDisco: .asciiz "\nMove o disco "
MoveDisco2: .asciiz " do pino "
MoveDisco3: .asciiz " para o pino "

.text
.globl main
main:
   li $v0,  4          # print string
   la $a0,  NumDiscos
   syscall
   li $v0,  5          # le int
   syscall

   # parametros
   add $a0, $v0, $zero # move to $a0
   li $a1, 'A'
   li $a2, 'B'
   li $a3, 'C'

   jal hanoi           # chama hanoi

   li $v0, 10          # exit
   syscall

hanoi:

   #preenche o monte
```

```
addi $sp, $sp, -20
sw   $ra, 0($sp)
sw   $s0, 4($sp)
sw   $s1, 8($sp)
sw   $s2, 12($sp)
sw   $s3, 16($sp)

add $s0, $a0, $zero
add $s1, $a1, $zero
add $s2, $a2, $zero
add $s3, $a3, $zero

addi $t1, $zero, 1
beq $s0, $t1, output

recursao1:

    addi $a0, $s0, -1
    add $a1, $s1, $zero
    add $a2, $s3, $zero
    add $a3, $s2, $zero
    jal hanoi

    j output

recursao2:

    addi $a0, $s0, -1
    add $a1, $s3, $zero
    add $a2, $s2, $zero
    add $a3, $s1, $zero
    jal hanoi
```

```
sair:

    lw   $ra, 0($sp)        # registros do monte
    lw   $s0, 4($sp)
    lw   $s1, 8($sp)
    lw   $s2, 12($sp)
    lw   $s3, 16($sp)

    addi $sp, $sp, 20       # ponteiro do monte

    jr $ra

output:

    li $v0,  4              # print string
    la $a0,  MoveDisco
    syscall
    li $v0,  1              # print int
    add $a0, $s0, $zero
    syscall
    li $v0,  4              # print string
    la $a0,  MoveDisco2
    syscall
    li $v0,  11             # print char
    add $a0, $s1, $zero
    syscall
    li $v0,  4              # print string
    la $a0,  MoveDisco3
    syscall
    li $v0,  11             # print char
    add $a0, $s2, $zero
```

```
syscall


beq $s0, $t1, sair
j recursao2
```