Universidade Federal de São João del-Rei
Arquitetura e Organização de Computadores I
Trabalho Prático 1

Felipe Francisco Rios de Melo
Thales Mrad Leijoto

1. Introdução

O trabalho a seguir foi feito com o intuito de resolver dois problemas utilizando Assembly MIPS. Os problemas que foram resolvidos são: calcular o valor do π, com o usuário digitando o número de termos que ele quer usar para calcular, e calcular o MDC (máximo divisor comum) entre dois números.
A documentação está dividida entre questão 1 e questão 2, que retratam os problemas citados acima. Os algoritmos apresentados no trabalho foram codificados e testados em um software emulador de MIPS, o Mars v4.5.

2. Questão 1

2.1. Modelagem do Cálculo de Aproximação de π

A resolução desse problema consiste em calcular o valor de π através da série infinita de *Gregory-Leibniz*. Todos os termos dessa série tem numerador igual a 4 e o numerador começando de 1 crescendo de 2 em 2, ou seja, somente números ímpares. Para se aproximar do valor de πi, são somados os termos ímpares da série e subtraídos os termos pares.

π = (4/1) - (4/3) + (4/5) - (4/7) + (4/9) - (4/11) + (4/13) - (4/15) …

No algoritmo desenvolvido, o usuário digita no terminal o número de termos que ele quer que sejam utilizados para o cálculo. Nota-se que quanto mais termos, mais próximo do número π chegamos.

2.2. Implementação

2.2.1. Código fonte

2.2.2. Funcionamento do programa

```
.data

msg1: .asciiz "\nDigite quantas termos para calcular o PI deseja: "
msg2: .asciiz "\nO valor de PI é: "
    .text
    .globl main

main:
    li $v0, 4        #imprime a mensagem 1
    la $a0, msg1
    syscall

    li $v0, 5        #le um inteiro
    syscall
    add $a0, $v0, $zero  #guarda valor lido no $a0 (parametro do procedimento)

    jal CalculaPi        #chama procedimento que calcula o pi

    li $v0, 4
    la $a0, msg2     #imprime mensagem 2
    syscall
    li $v0, 2        #imprime valor aproximado de pi ($f12)
    syscall
    j exit           #jump para o fim do programa

CalculaPi:

   addi $sp, $sp, -16  #abre espaço na pilha para os registradores
    sw $s1, 0($sp)
    sw $s2, 4($sp)    #salva na pilha os registradores que serao utilizados
    sw $s3, 8($sp)
    sw $s4, 12($sp)
    li $s1, 1        #contador de termos
    li $s2, 1        #denominador das divisões
    li $s3, 4        #constante do numerador
    li $s4, 2        #constante para descobrir se o s1 é par ou impar

loop:
    slt $t0, $a0, $s1    # quantidade de termos < contador ?
    bne $t0, $zero, return    # se 1, desvia para o retorno da função
    rem $t0, $s1, $s4    # resto da divisão para verificar se o termo é par ou impar
    mtc1 $s3,$f1     #converte inteiro para float e salva em f1
    mtc1 $s2,$f2     #converte inteiro para float e salva em f2
    div.s $f1, $f1, $f2  #divide a constante 4 por um numero impar
    beq $t0, $zero, par  #se s1 for par, pula pra label par
    add.s $f12,$f12,$f1  #soma ao valor de pi o resultado da divisão ($f12 é o argumento syscall para impressão de float)
    j incremento

par:

    sub.s $f12,$f12,$f1  #subtrai ao valor de pi o resultado da divisão

 incremento:
    add $s1, $s1, 1      #incrementa 1 ao contador
    add $s2, $s2, 2      #incrementa 2 ao denominador das diviões (é sempre impar)
    j loop

return:
    lw $s1, 0($sp)
    lw $s2, 4($sp)      #recupera os registradores que foram alterados
    lw $s3, 8($sp)
    lw $s4, 12($sp)
    addi $sp, $sp, -16  # libera espaço da pilha
    jr $ra           #retorna o controle pra main

exit:
    li $v0, 10       #termino de execução
    syscall
```

Primeiramente, é lido do terminal a quantidade de termos para o cálculo de π, o valor é guardado no registrador $a0, que é utilizado como parâmetro do procedimento *CalculaPi*, em seguida chama-se a função.

São declaradas algumas variáveis necessárias para o cálculo, como o contador de termos, o denominador da fração que será sempre ímpar, o numerador que, de acordo com a fórmula, é sempre 4, além da constante 2 que é usada para realizar a divisão que dirá se o termo é par ou ímpar.

Em seguida, entra-se no *label loop*, no qual é checado se o registrador $a0 (quantidade de termos) é menor que o registrador $s1 (contador de iterações do loop), se sim, o programa é direcionado para *label return* (retorno da função), pois o cálculo já está terminado. Divide-se o contador por 2, e o resto desta divisão concluirá se a iteração é par ou ímpar, visto que, se o resto for igual a zero, o contador é par, caso contrário é ímpar. Converte-se de inteiro para número fracionário o denominador e o numerador de uma das frações mostradas na fórmula (seção 2.1) para que a divisão também seja um número fracionário. Se for par, o funcionamento do programa é direcionado para a *label par*, se não, continua o funcionamento normal linha-a-linha. Supondo que o resto da divisão deu diferente de zero (o contador é ímpar). Realizado a divisão, soma o resultado dela ao registrador $f12 (registrador de ponto flutuante) que guarda o valor aproximado de π. Por fim, incrementa-se 1 ao $s1 (contador) e 2 ao $s2 (denominador das divisões) pois deve-se ser sempre ímpar, respeitando a fórmula de Leibniz. Esses passos são repetidos até que seja satisfeita a condição no início da label loop em que a quantidade de termos deve ser igual ao contador.

Para explicar uma iteração do loop, foi suposto que o valor do contador na iteração era ímpar, mas deve-se deixar claro que eles alteram a cada iteração:

```
i = 0 -> par
i = 1 -> ímpar
i = 2 -> par
i = 3 -> ímpar
E assim por diante…
```

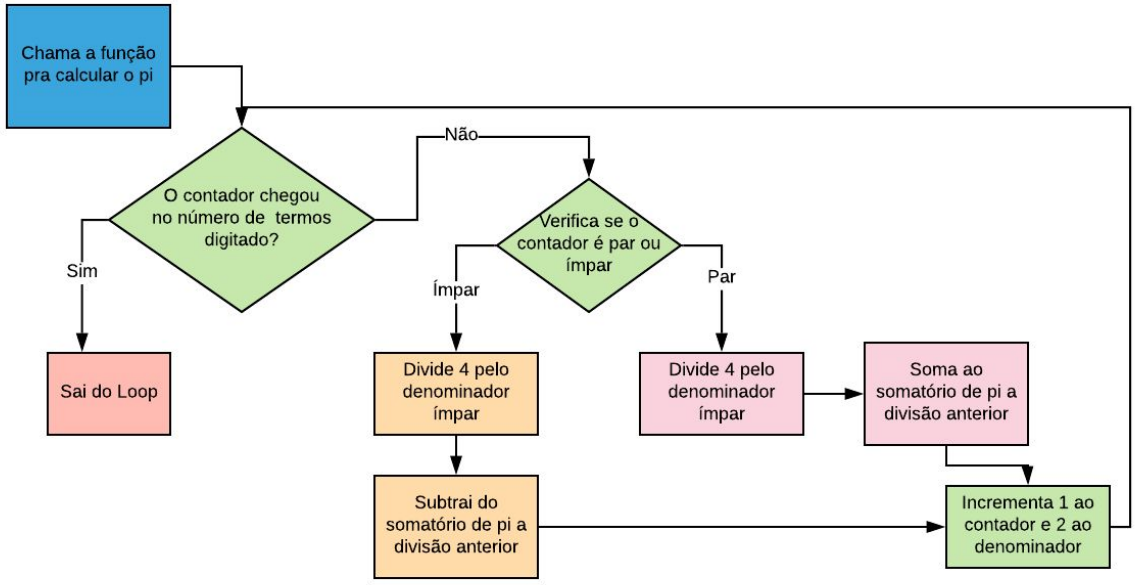
Quando o loop está em uma iteração par segue-se os mesmos procedimentos para quando é ímpar, a diferença é que quando é par, ao invés de somar o resultado da divisão ao registrador $f12, o valor é subtraído do registrador.

Como já explicado, o loop termina quando o valor do contador é igual a quantidade de termos inserida pelo usuário. Quando esta condição é satisfeita, entra-se na l*abel return*, que retorna o controle da função para o programa principal, no ponto em que o procedimento foi chamado.

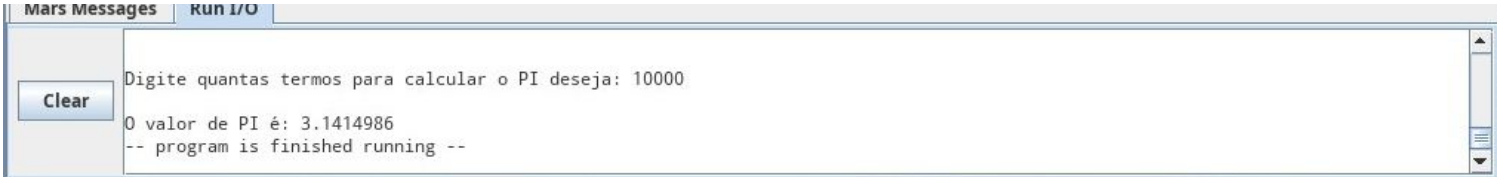
Finalmente, imprime-se o conteúdo de $f12, que é o valor aproximado de π que foi possível alcançar a partir da quantidade de termos dada por linha de comando.

A seguir está o fluxograma de execução da questão 1:

FLUXOGRAMA 1. Execução do procedimento que calcula o valor de π.



2.3 Screenshot console Mars



3. Questão 2

3.1. Modelagem do Cálculo de Máximo Divisor Comum

A resolução desta questão consiste no cálculo do MDC entre dois números inteiros através de uma função recursiva, que representa a seguinte equação:

$$
\begin{cases}
Mdc(a, b) = Mdc(b, r) \\
Mdc(c, 0) = c
\end{cases}
$$

Onde r é o resto da divisão de a por b. O usuário deverá digitar os valores de a e b.

3.2. Implementação

3.2.1. Código Fonte

```
.data

    msg1: .asciiz "\nDigite o primeiro número: "
    msg2: .asciiz "Digite o segundo número: "
    msg3: .asciiz "\nO MDC é igual a: "

.text
.globl main

main:
    li $v0, 4        #printa a mensagem 1
    la $a0, msg1
    syscall

    li $v0, 5        #le o primeiro inteiro
    syscall
    add $a1, $v0, $zero  #guarda valor lido no $a0 (parametro do procedimento)

    li $v0, 4        #printa a mensagem 2
    la $a0, msg2
    syscall

    li $v0, 5        #le o segundo inteiro
    syscall
    add $a2, $v0, $zero  #guarda valor lido no $a1 (parametro do procedimento)

    jal mdc

    add $t0, $v0, $zero  #guarda o retorno da função em uma reigstrador temporario (p/ q o v0 possa ser usado para printar a string)
    li, $v0, 4
    la $a0, msg3     #printa a mensagem 3
    syscall

    add $a0, $t0, $zero  #a0 recebe o valor retornado da funão para que ele possa ser printado na tela
    li $v0, 1
    syscall
    j exit

 mdc:
    addi $sp, $sp, -12   #aloca memoria para variavel recebida da main
    sw $a1, 0($sp)       #carrega o espaço de memoria alocado
    sw $a2, 4($sp)
    sw $ra, 8($sp)       # Guarda endereço de retorno

    bnez $a2, else       # a2 != 0 ?
    add $v0, $a1, $zero  # se a2 == 0, coloca-se o valor de a2 em v0 para que possa ser retornado para a main
     addi $sp, $sp, 12     #liberando memoria
     jr $ra            #desvia o programa para o endereço de retorno do programa principal

else:
     add $t0, $a2, $zero  #aux = $a2
     rem $a2, $a1, $a2    #resto da divisão de $a1 por $a2
     add $a1, $t0, $zero  # $a1 = aux

     jal mdc          #chamada recursiva

     lw $a1, 0($sp)       #recuperando $a1
     lw $a2, 4($sp)       #recuperando $a2
     lw $ra 8($sp)        #recuperando o endereço de retorno
     addi $sp, $sp, 12    #libera o espaço usado
     jr $ra           #desvia o programa para o endereço de retorno do programa principal

exit:
     li $v0, 10       #termino de execução
     syscall
```

3.2.2. Funcionamento do programa

Os dois inteiros são lidos pelo terminal, e armazenados respectivamentes nos registradores, $a1, $a2 (não se iniciou do $a0 pois ele foi usado para outras chamadas do *syscall*). Ocorre um *jump* para o procedimento *mdc.*

Na função *mdc*, o primeiro conjunto de instruções é responsável pela alocação de memória para empilhar os registradores usados no procedimento, que são o $a1 e o $a2 (os dois números lidos pelo terminal, que chamaremos de *a* e *b* daqui pra frente) e o $ra que guarda o endereço de retorno da função que o chamou.

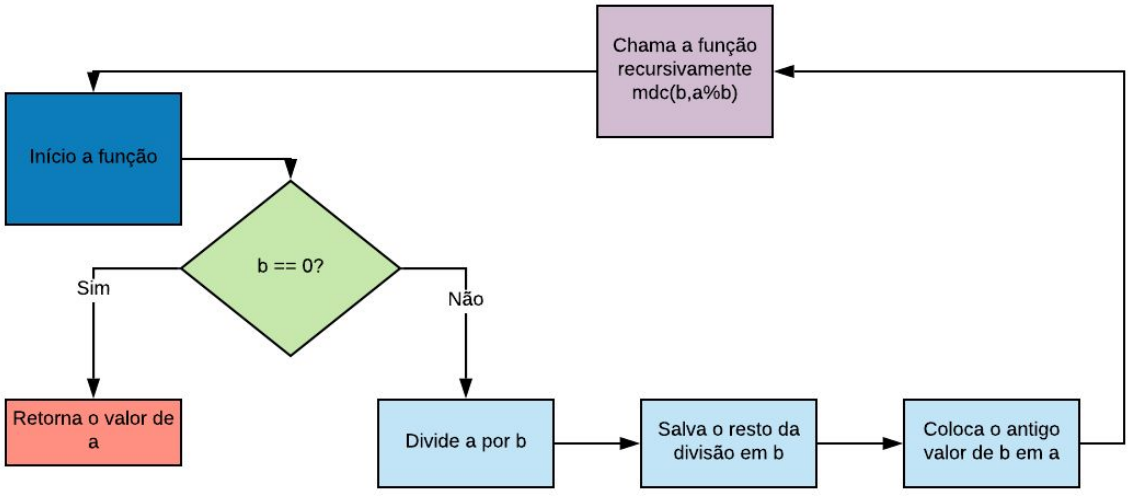
O segundo conjunto, se trata da lógica de cálculo do mdc usando recursividade. Primeiramente, se deve checar o registrador *b* é igual a zero, se sim, já retorna-se o valor de *a* para a *main*, e libera o espaço de memória utilizado no procedimento. Caso contrário, o procedimento é redirecionado para a *label else, na qual* devemos chamar recursivamente novamente a função mdc, isto é feito da seguinte forma: em um registrador auxiliar $t0, armazenamos o valor de *b*, pois no registrador *b*, irá ser armazenado o resto da divisão de *a* por *b* e no registrador *a* será guardado o valor antigo do registrador *b* que estava armazenado temporariamente em $t0. Então, chama-se recursivamente a função *mdc*, com os parâmetros $a1 (b) e $a2 (b % a). Estes passos, seguem especificamente o que mostra o modelo matemático apresentado na seção 2.1.

Na volta de cada chamada recursiva, recupera-se da memória os registradores $a1, $a2 e $ra e desvia-se para o conteúdo do registrador $ra, que contém o endereço de retorno para a função que o chamou.

Na *main* após voltar do procedimento, o mdc que veio da função armazenado em $a0, é atribuído a um registrador temporário $t0, pois o registrador $a0 é alterado para que seja realizado a impressão de uma string no console, após a impressão o $a0 recebe de $t0 o resultado do cálculo de MDC e o imprime na tela.

FLUXOGRAMA 2. Execução do procedimento que calcula o MDC entre dois números.

Esse fluxograma representa o que a função faz a cada vez que é chamada recursivamente.



3.3. Screenshot console Mars