Relatório do Laboratório 04 - Organização de Computadores

Grupo:

- Henrique Mateus Teodoro - 23100472
- Rodrigo Schwartz - 23100471

Exercício 1

O exercício 1 consiste em criar um programa em Assembly MIPS, que utiliza o método iterativo de Newton, para encontrar, de forma aproximada, a raiz quadrada de um número x qualquer. O número x deve ser um número de precisão dupla, e tanto o número x quanto o número de iterações que o método deve realizar devem ser passados pelo usuário via console. A estimativa é calculada da seguinte forma:

$$Estimativa = \left(\frac{\left(\frac{x}{Estimativa} \right) + Estimativa}{2} \right)$$

Inicialmente, no segmento de dados, são instanciadas algumas variáveis, que serão utilizadas no decorrer do programa:

```
.data
        string_input: .asciiz "Insira um valor x: "
        string_iteracoes: .asciiz "Insira um número de iterações para o método: "
        string_result: .asciiz "A raiz de x é: "
        string_esperado: .asciiz "\nResultado esperado: "
        string_erro_abs: .asciiz "\nErro absoluto: "
        estimativa: .double 1
        num2: .double 2.0
```

Todas as variáveis que contém a palavra string na composição de seu nome serão utilizadas para imprimir uma mensagem no console, para que o programa fique mais interativo e compreensível. Já as últimas duas variáveis foram declaradas para serem utilizadas no cálculo da raiz de x. Houve essa necessidade de declarar e ler da memória dois números double, visto que não há uma instrução que carregue imediatamente um registrador do tipo **$f** com um valor de ponto flutuante de dupla precisão qualquer.

Então inicia-se o segmento de texto, realizando as operações de input do usuário:

```
.text
        li $v0, 4                       # seta a operação de impressão de string
        la $a0, string_input            # passa um ponteiro que aponta para string_input para $a0
        syscall                         # faz a impressão da string

        li $v0, 7                       # seta a operação de leitura de double
        syscall                         # faz a leitura do double, e salva em $f0

        li $v0, 4                       # seta a operação de impressão de string
        la $a0, string_iteracoes        # passa um ponteiro que aponta para string_iteracoes para $a0
        syscall                         # faz a impressão da string

        li $v0, 5                       # seta a operação de leitura de int
        syscall                         # faz a leitura de int que será o nº de iterações
        move $t1, $v0                   # move o valor lido que está em $v0 para $t0
```

Nesse primeiro bloco de código do segmento .text, é feita a impressão da string **string_input** no console, e em sequência é feita a leitura do número x em ponto flutuante de dupla precisão, cuja raíz de x será calculada. Após isso, imprimisse também a string **string_iteracoes**, e em sequência é feita a leitura de um número n inteiro, que será utilizado para determinar quantas iterações serão feitas utilizando o método de Newton. O valor de x é armazenado no registrador **$f0** e o número de iterações n é armazenado no registrador **$t1**.

```
la $s0, estimativa              # passa o endereço de estimativa para $s0
l.d $f2, 0($s0)                 # passa o conteúdo que está em $s0 para $f2

la $s1, num2                    # passa o endereço de num2 para $s1
l.d $f6, 0($s1)                 # passa o conteúdo que está em $s1 para $f6 ($f6 = 2)
```

São então lidos da memória, e passados para um registrador do tipo **$f,** o valor da estimativa (instanciada no segmento de dados como 1) e o valor num2 (instanciado no segmento de dados como 2). A estimativa é utilizada como valor inicial da estimativa da raiz, e será utilizada para calcular então a nova estimativa da raiz de x. Já o **num2**, será utilizado para realizar a divisão por 2 no cálculo da nova estimativa. A estimativa foi armazenada no registrador **$f2** e o **num2** foi armazenado no registrador **$f6**.

É então feita a chamada da função **raiz_quadrada**, utilizando a instrução **jal**:

```
jal raiz_quadrada               # faz a chamada da função raiz_quadrada, e desvia para a label
j exit                          # quando a função terminar, da um jump para a label exit
```

Ao realizar essa chamada, o registrador **$ra** recebe **PC+4**, ou seja, armazena o endereço de retorno da função (aponta para a instrução seguinte, **j exit**), e o registrador **$pc** aponta para o endereço da função **raiz_quadrada**, fazendo com

que o programa desvie a sua execução, e passe a iniciar a execução da função **raiz_quadrada**.

```
raiz_quadrada:
        beq $t0, $t1, fim_funcao        # compara se o $t0 == $t1 (i == nº de iterações), se sim, desvia para a label fim_funcao

        div.d $f4, $f0, $f2             # divide o valor de x (que está em $f0) pela estimativa $f2 e salva em $f4
        add.d $f4, $f4, $f2             # soma em $f4 o valor de $f4 com $f2
        div.d $f4, $f4, $f6             # faz a divisão de $f4 por 2 e salva em $f4
        mov.d $f2, $f4                  # passa o valor em $f4 (resultado do cálculo da estimativa) para $f2 (estimativa)

        addi $t0, $t0, 1                # incrementa o comparador do laço ($t0) em 1
        j raiz_quadrada

fim_funcao:
        jr $ra                          # faz o retorno para seguir a execução correta após o término da função
```

Inicialmente, dentro de **raiz_quadrada**, é feito uma comparação utilizando a instrução **beq**, para saber se a iteração atual (**$t0** armazena essa informação) é igual a iteração máxima n (**$t1** armazena essa informação). Se essa condição for satisfeita, significa que o método já realizou todas as n iterações, e portanto, deve sair do loop e encerrar a sua execução, desviando para a label **fim_funcao**:

```
fim_funcao:
        jr $ra                          # faz o retorno para seguir a execução correta após o término da função
```

Na label **fim_funcao**, é feita a chamada de retorno utilizando a instrução **jr $ra**, que faz com que o registrador **$pc** receba o endereço armazenado por **$ra**, que contém o endereço de retorno da função.

Caso o **beq** não seja satisfeito, significa que o programa deve seguir realizando o cálculo da estimativa da raiz de x, e então, o seguinte trecho de código é executado:

```
        div.d $f4, $f0, $f2             # divide o valor de x (que está em $f0) pela estimativa $f2 e salva em $f4
        add.d $f4, $f4, $f2             # soma em $f4 o valor de $f4 com $f2
        div.d $f4, $f4, $f6             # faz a divisão de $f4 por 2 e salva em $f4
        mov.d $f2, $f4                  # passa o valor em $f4 (resultado do cálculo da estimativa) para $f2 (estimativa)

        addi $t0, $t0, 1                # incrementa o comparador do laço ($t0) em 1
        j raiz_quadrada
```

Inicialmente, divide-se o valor de x pela estimativa (**div.d $f4, $f0, $f2**) e então armazena-se esse resultado em **$f4**. É feita a soma da estimativa no resultado da divisão calculada anteriormente (**add.d $f4, $f4, $f2**), e então divide-se tudo isso por 2 (**div.d $f4, $f4, $f6**). Com o valor da nova estimativa calculada e armazenada em **$f4**, transfere-se essa nova estimativa para o registrador **$f2**, para que na próxima iteração (se houver), o valor da estimativa já esteja atualizado. Faz-se então o incremento do registrador **$t0** (**addi $t0, $t0, 1**), para atualizar a iteração atual, e verificar no próximo loop, se deve ocorrer uma nova iteração do

método ou não. Após o término do cálculo, retorna-se para o loop utilizando a instrução **j raiz_quadrada**.

```
jal raiz_quadrada               # faz a chamada da função raiz_quadrada, e desvia para a label
j exit                          # quando a função terminar, da um jump para a label exit
```

Após o término do cálculo, a função retorna sua execução executando a instrução **j exit**, ou seja, desviando sua execução para a label **exit**, que mostrará o resultado obtido no cálculo, e também, o valor esperado para a raiz do número de x:

```
exit:
    li $v0, 4                   # seta a operação de impressão de string
    la $a0, string_result       # passa um ponteiro que aponta para string_input para $a0
    syscall                     # faz a impressão da string

    li $v0, 3                   # seta a operação de impressão de double
    mov.d $f12, $f2             # move o valor do cálculo da raiz_quadrada usando o método iterativo de Newton que está em $f2 para $f12, para imprimir o double no console
    syscall                     # faz a impressão do double
```

Nesse primeiro bloco de código, é feita a impressão da string **string_result** no console, e então imprime-se em sequência o valor da estimativa obtido através do método iterativo de Newton.

```
    sqrt.d $f8, $f0             # faz a operação de raiz usando a instrução em $f8

    li $v0, 4                   # seta a operação de impressão de string
    la $a0, string_esperado     # passa um ponteiro que aponta para string_input para $a0
    syscall                     # faz a impressão da string

    li $v0, 3                   # seta a operação de impressão de double
    mov.d $f12, $f8             # move o valor do cálculo da raiz_quadrada  usando a instrução sqrt.d que está em $f8 para $f12, para imprimir o double no console
    syscall                     # faz a impressão do double
```

Depois disso, utilizando a instrução **sqrt.d**, calcula-se o valor correto da raiz de x passado via console no início do programa, e armazena-se esse valor em **$f8**, para que seja feito uma comparação entre o valor obtido (estimativa que está em **$f2**) com o valor esperado (que está em **$f8**). É feita então a impressão da string **string_esperado** e do valor esperado.

```
    sub.d $f12, $f8, $f2        # faz o cálculo do erro absoluto ($f8 resultado esperado - $f2 valor obtido) e salva em $f12
    abs.d $f12, $f12            # faz o módulo do erro absoluto

    li $v0, 4                   # seta a operação de impressão de string
    la $a0, string_erro_abs     # passa um ponteiro que aponta para string_input para $a0
    syscall                     # faz a impressão da string

    li $v0, 3                   # seta a operação de impressão de double
    syscall                     # faz a impressão do double
```

Por fim, calcula-se o erro absoluto entre o resultado esperado e o valor obtido, afim de saber se o método está aproximando bem ou não o valor da raiz de x. Esse erro absoluto é calculado basicamente realizando a diferença entre o resultado esperado e o resultado obtido (**sub.d $f12, $f8, $f2**), armazenando então o erro em **$f12**. Por fim, realiza-se a operação de módulo, para que o erro seja um

valor positivo (**abs.d $f12, $f12**), e então, é feita a impressão da string **string_erro_abs** e a impresão do double erro absoluto.

Vamos então realizar alguns testes no programa, para verificar se ele está funcionando adequadamente ou não. Vamos iniciar um teste para calcular a raiz de 9, que é 3, utilizando 5 iterações:

```
Insira um valor x: 9
Insira um número de iterações para o método: 5
A raiz de x é: 3.000000001396984
Resultado esperado: 3.0
Erro absoluto: 1.3969838619232178E-9
-- program is finished running (dropped off bottom) --
```

Fazendo os inputs corretos no console, obtém-se um valor muito próximo ao valor esperado 3. Percebe-se isso devido ao fato do erro absoluto ser um valor na casa de 10^-9, ou seja, um erro muito pequeno. Por tanto, houve uma aproximação excelente do resultado esperado.

Vamos realizar um segundo teste, calculando a raiz aproximada de um número quebrado, como 156.987, utilizando 12 iterações:

```
Insira um valor x: 156.987
Insira um número de iterações para o método: 12
A raiz de x é: 12.52944531892773
Resultado esperado: 12.52944531892773
Erro absoluto: 0.0
-- program is finished running (dropped off bottom) --
```

Utilizando-se 12 iterações, foi possível chegar no valor esperado, com um erro absoluto de 0. Isso ocorre porque quanto mais iterações são utilizadas no cálculo, mais precisão terá o resultado obtido. Com isso, é possível confirmar novamente, o adequado funcionamento do programa.

Exercício 2

O exercício 2 consiste em criar um programa em Assembly MIPS que utiliza uma série para aproximar o valor do seno para um número x qualquer. A série utilizada é a seguinte:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

Inicialmente, no segmento de dados, são instanciadas algumas variáveis, que serão utilizadas no decorrer do programa, como mensagens de exibição ou valores em double que vão ser necessários no cálculo:

```
.data
        string_input_x: .asciiz "insira o valor de x: "
        string_input_n: .asciiz "insira o número de termos: "
        string_output: .asciiz "\n o seno do número é: "
        numMenos1: .double -1.0
        num1: .double 1
```

No segmento .text, as variáveis necessárias para o cálculo são lidas pelo teclado. O primeiro parâmetro a ser digitado é o valor x, o qual calcularemos o seu seno, que ficará salvo em $f0. O segundo parâmetro é o número n de termos que deve ser calculado, ficando salvo em $t1.

```
.text
        li $v0, 4                   # seta a operação de impressão de string
        la $a0, string_input_x      # passa um ponteiro que aponta para string_input_x para $a0
        syscall

        li $v0, 7                   # seta a operação de leitura de double
        syscall                     # salva por padrão em $f0

        li $v0, 4                   # seta a operação de impressão de string
        la $a0, string_input_n      # passa um ponteiro que aponta para string_input_n para $a0
        syscall

        li $v0, 5                   # seta a operação de leitura de inteiro
        syscall
        move $t1, $v0
```

O endereço do double -1 é salvo em $s1 e o valor efetivo “-1” é salvo no registrador $f6, sendo utilizado posteriormente nos cálculos. Depois de ler os valores necessários para o cálculo, passamos N, que é o número de termos para ser calculado (salvo em $t1), para o registrador $a1 e chamamos a função seno. Após o retorno da função seno, o código pula para o label exit, com o jump.

```
la $s1, numMenos1
l.d $f6, 0($s1)              # número -1

move $a1, $t1
jal  seno
j exit
```

Antes de explicar sobre a função seno em si, faz-se necessário falar sobre as outras duas funções que são auxiliares para o cálculo principal: potenciação e fatorial. A função de potenciação calcula o valor de uma base double elevada a um expoente inteiro. Para utilizá-la é necessário passar para $t1 o expoente e $f2 a base, o retorno da função será salvo em $f4. Inicialmente, na função, o registrador $f4 é inicializado em 1 e, através de um laço de repetição controlado pelo contador $t0 e pelo expoente $t1, ele é utilizado para acumular a multiplicação do que ele já tem vezes a base. Ou seja, multiplicamos o valor 1, salvo no início em $f4, pela base ($f2) e salvamos o resultado novamente em $f4. Isso é repetido tantas vezes conforme o expoente passado, finalizando o cálculo da potenciação.

```
#------------------------------------  Potenciação ---------------------------------
potenciacao:
        # $t1 = expoente, $f2 = base, $f4 = retorno
        la $t9, num1
        l.d $f4, 0($t9)

        li $t0, 0 # contador
repeticao_potenciacao:
        bge $t0, $t1, fim_funcao_potenciacao
        mul.d $f4, $f4, $f2
        addi $t0, $t0, 1
        j repeticao_potenciacao
fim_funcao_potenciacao:
        jr $ra
```

Para utilizar a função fatorial é necessário passar o valor a ser calculado o fatorial para o registrador $f2, o retorno será dado em $f4. O valor 1 inicialmente é salvo em $f4 e convertemos o número a ser calculado o fatorial ($f2) para um inteiro que será salvo em $t8 (ele será utilizado no comparador do loop para saber quando parar). O contador do loop, $t1, é inicializado em 1. Um laço de repetição é iniciado e só é possível sair dele quando o contador ($t1) for maior do que o número a ser calculado o fatorial (salvo como inteiro em $t8). Dentro do laço, convertemos o valor do contador $t1 para um double salvo em $f16. Após isso, é feita a multiplicação do contador em double ($f16) vezes o acumulador ($f4). O resultado da multiplicação é salvo também em $f4. O contador $t1 (inteiro) é incrementado em um e é realizado um jump indo para o label loop, realizando a próxima iteração. No final das iterações, o fatorial é calculado. A imagem a seguir mostra a função:

```
fatorial:
        # $f2 = número, $f4 = retorno

        # Inicializa o fatorial como 1.0
        la $t9, num1
        l.d $f4, 0($t9)

        cvt.w.d $f16, $f2        # Converte o valor de double para inteiro
        mfc1 $t8, $f16           # Move o valor inteiro de $f16 para $t8 (número para calcular o fatorial)

        li $t1, 1                # Inicializa o contador do loop (i = 1)
loop:
        bgt $t1, $t8, fim_fatorial

        mtc1 $t1, $f16           # Move o valor inteiro de $t1 para $f16
        cvt.d.w $f16, $f16       # Converte o valor de inteiro para double

        mul.d $f4, $f4, $f16     # Multiplica o fatorial acumulado por f16
        addi $t1, $t1, 1         # Incrementa o contador do loop

        j loop                   # Volta para o início do loop

fim_fatorial:
        jr $ra
```

Com esses auxiliares a função seno finalmente pode ser calculada. Vale lembrar que o valor x que deve ser calculado o seno está salvo em $f0, o número de termos está salvo em $a1 e o retorno será dado em $f14. Inicialmente aumentamos a pilha e salvamos nela o endereço de retorno $ra, pois como há chamada de outras funções no meio desse código, esse valor poderia ser perdido. O contador que será utilizado, salvo no registrador $t3, é inicializado em zero (o contador é o valor n da fórmula). A partir daqui, o processo de cálculo será executado algumas vezes, de acordo com o número de termos.

```
#------------------------------------  Seno ----------------------------------

# $f0 = x; $a1, = numero_termos, $f14 = resultado do seno
seno:
        addi $sp, $sp, -4
        sw $ra, 0($sp)
        li $t3, 0 # contador
```

Inicialmente é feita a verificação se o contador é maior ou igual ao número de termos n (salvo em $a1), para sair do loop. Depois disso, o valor 2n + 1, que será utilizado algumas vezes no cálculo, é armazenado no registrador $t4.

```
repeticao_seno:
        bge $t3, $a1, fim_funcao_seno

        mul $t4, $t3, 2
        addi $t4, $t4, 1
```

Seguindo, o seguinte valor é calculado:

$$(-1)^n$$

Passamos para $f2 (base) o valor de -1, salvo em $f6, e para $t1 (expoente) o valor de n (que é o contador), salvo em $t3. Chamamos a função de potenciação e salvamos o retorno em $f8.

```
mov.d $f2, $f6
move $t1, $t3
jal potenciacao
mov.d $f8, $f4
```

O próximo valor a ser calculado é:

$$x^{2n+1}$$

Passamos para $f2 (base) o valor de x, salvo em $f0, e para $t1 (expoente) o valor de 2n + 1, salvo em $t4. Chamamos a função de potenciação e seu resultado fica salvo em $f4.

```
mov.d $f2, $f0
move $t1, $t4
jal potenciacao
```

O passo seguinte é realizar a seguinte multiplicação:

$$\frac{(-1)^n}{\quad} x^{2n+1}$$

Como o primeiro valor já foi calculado (salvo em $f8) e o segundo foi calculado agora (salvo em $f4), basta multiplicá-los.

```
mul.d $f8, $f8, $f4
```

O próximo valor a ser calculado é:

$$(2n+1)!$$

Primeiro, convertemos o valor de 2n + 1, salvo em $t4, para um double, salvo em $f2 (o registrador $f2 é o lugar onde deve ser salvo o valor a ser calculado o fatorial). Depois, basta chamar a função fatorial e passar o valor de retorno ($f4) para o registrador $f10.

```
mtc1 $t4, $f2
cvt.d.w $f2, $f2
jal fatorial
mov.d $f10, $f4
```

Após isso, precisamos fazer a divisão dos valores calculados anteriormente pelo último cálculo:

$$\frac{(-1)^n}{(2n+1)!}x^{2n+1}$$

Como a multiplicação da parte de cima da fração já foi realizada e salva em $f8, basta realizar a divisão pelo registrador $f10 (registrador em que o denominador da fração está salvo). O resultado é novamente armazenado em $f8.

```
div.d $f8, $f8, $f10
```

Por fim, o registrador $f14 é responsável por armazenar o somatório do valor calculado em cada loop. Logo, basta agora apenas fazer a soma do conteúdo de $f14 com o de $f8 (onde está salvo o cálculo anterior) e armazenar o resultado novamente em $f14. No fim, o contador n é incrementado e com o jump voltamos para o início do laço.

```
add.d $f14, $f14, $f8

addi $t3, $t3, 1
j repeticao_seno
```

Quando o laço termina, resgatamos o endereço de retorno da função, que estava salvo na pilha, e utilizamos a instrução jr para retornar.

```
fim_funcao_seno:
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra
```

Por fim, apenas é exibido na tela o resultado do cálculo do seno:

```
# ------------------------------- Print Resultado ---------------------------------------
exit:
        li $v0, 4                   # seta a operação de impressão de string
        la $a0, string_output       # passa um ponteiro que aponta para string_result para $a0
        syscall

        li $v0, 3                   # seta a operação de impressão de double
        mov.d $f12, $f14
        syscall
```

A seguir, segue um exemplo do bom funcionamento do cálculo. Vale a pena lembrar que o valor x que deseja-se calcular o seno deve ser informado em radianos.

```
insira o valor de x: 1.57079633
insira o número de termos: 20

 o seno do número é: 1.0
-- program is finished running (dropped off bottom) --
```

O valor de x utilizado nesse exemplo corresponde, aproximadamente, ao ângulo de 90° (pi/2). O número de iterações utilizado foi 20. Analisando o resultado, nota-se que o cálculo convergiu corretamente para o seno de 90º (1). Diminuindo o número de iterações para 10, nota-se que o valor converge corretamente, mas não é tão exato:

```
insira o valor de x: 1.57079633
insira o número de termos: 10

 o seno do número é: 0.9999999999999998
-- program is finished running (dropped off bottom) --
```