UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
INE5411 - ORGANIZAÇÃO DE COMPUTADORES I

PEDRO TAGLIALENHA (22203674)
VITOR PRAXEDES CALEGARI (22200379)

RELATÓRIO LABORATÓRIO 3

FLORIANÓPOLIS
2023

1- Introdução

No laboratório 3 da disciplina de Organização de Computadores, realizamos uma atividade prática utilizando a linguagem Assembly para o processador MIPS no simulador MARS. O objetivo foi aplicar os conceitos teóricos aprendidos sobre loops, instruções de desvios, procedimentos, pilha entre outros para a solução de problemas de cálculo numérico em assembly, realizando iterações para chegar na aproximação da resposta.

Na primeira atividade, implementamos um procedimento chamado **raiz_quadrada** em Assembly que recebe um parâmetro x de precisão dupla e calcula uma estimativa aproximada da raiz quadrada de x. Um loop é utilizado para iterar n vezes, onde n é informado pelo usuário. Durante cada iteração, a estimativa é atualizada de acordo com o método iterativo de Newton. Foram utilizadas instruções e registradores de ponto flutuante de precisão dupla para realizar todos os cálculos.

Além disso, comparamos o resultado obtido com o procedimento **raiz_quadrada** com o resultado da função **sqrt.d** para calcular o erro absoluto em diferentes iterações, testamos a variação da aproximação com n = 1 até n = 50.

Na segunda atividade, implementamos um procedimento para calcular o seno de um parâmetro x usando uma série de Taylor. Limitamos o cálculo aos primeiros 20 termos da série. Todos os cálculos foram realizados com instruções e registradores de ponto flutuante de precisão dupla.Para a implementação dessa solução, foi necessário a criação de diversos procedimentos que calculavam parte da série que o MIPS não consegue por default como, por exemplo, potências e fatorial.

2- Instruções Assembly Utilizadas

- **addi** (Add Immediate): Esta instrução é usada para adicionar um valor imediato a um registrador. Por exemplo, addi $sp, $sp, -12 é usado para diminuir o valor de $sp (ponteiro de pilha) em 12 unidades.
- **sw** (Store Word): Esta instrução é usada para armazenar um valor de registrador na memória. Por exemplo, sw $s0, 4($sp) armazena o valor de $s0 na memória, deslocando o endereço em $sp em 4 unidades.
- **lw** (Load Word): Esta instrução é usada para carregar um valor da memória para um registrador. Por exemplo, lw $s0, 4($sp) carrega o valor da memória (no endereço deslocado de $sp em 4 unidades) para $s0.
- **jal** (Jump and Link): Esta instrução é usada para fazer uma chamada de procedimento (função). Ela salta para o endereço do procedimento e armazena o endereço de retorno em $ra (registrador de retorno).
- **jr** (Jump Register): Esta instrução é usada para retornar de um procedimento. Ela salta para o endereço armazenado em um registrador (geralmente $ra, que contém o endereço de retorno).
- **beq** (Branch if Equal) e **bne** (Branch if Not Equal): Essas instruções são usadas para realizar saltos condicionais com base na igualdade ou diferença de valores em registradores.
- **beqz** (Branch if Equal to Zero) : A instrução beqz é semelhante à beq, mas especificamente verifica se um registrador é igual a zero. Se o valor do registrador for zero, ela realiza um salto condicional.

- **add** (Add) e mul (Multiply): Essas instruções realizam operações aritméticas de adição e multiplicação entre registradores. Por exemplo, add $t0, $zero, $s0 copia o valor de $s0 para $t0.
- **cvt.d.w** (Convert Double to Word): Esta instrução converte um valor de ponto flutuante de precisão dupla para um inteiro de 32 bits.
- **li** (Load Immediate): Esta instrução é usada para carregar um valor imediato em um registrador. Por exemplo, li $v0, 10 carrega o valor 10 em $v0.
- **mtc1.d** (Move to Coprocessor 1 - Double): Esta instrução é usada para mover um valor de um registrador de uso geral para um registrador de coprocessador 1 (registrador de ponto flutuante de precisão dupla). Por exemplo, mtc1.d $t0, $f0 move o valor de $t0 para $f0.
- **cvt.d.w** (Convert Word to Double): Esta instrução converte um valor inteiro de 32 bits para um valor de ponto flutuante de precisão dupla.
- **ldc1** (Load Double to Coprocessor 1): Esta instrução é usada para carregar um valor de ponto flutuante de precisão dupla da memória para um registrador de coprocessador 1.
- **mov.d** (Move Double): Esta instrução move um valor de ponto flutuante de precisão dupla de um registrador para outro. Por exemplo, mov.d $f6, $f30 copia o valor de $f30 para $f6.
- **div.d** (Divide Double): Esta instrução realiza a divisão entre dois valores de ponto flutuante de precisão dupla. Por exemplo, div.d $f6, $f6, $f10 divide o valor em $f6 por $f10 e armazena o resultado em $f6.
- **add.d** (Add Double): Esta instrução realiza uma adição entre dois valores de ponto flutuante de precisão dupla. Por exemplo, add.d $f26, $f26, $f6 adiciona o valor em $f6 ao valor em $f26 e armazena o resultado em $f26.
- **mul.d** (Multiply Double): Essa instrução realiza uma multiplicação entre dois valores de ponto flutuante de precisão dupla. Por exemplo, mul.d $f30, $f30, $f0 multiplica o valor em $f30 pelo valor em $f0 e armazena o resultado em $f30.
- **move (Move):** A instrução `move` é usada para copiar o valor de um registrador para outro registrador.

3- Códigos desenvolvidos

3.1- Questão 1

Código 1: Código para aproximação de raiz quadrada

```
.data
    Estimativa: .double 1
    dois: .double 2
    X: .double 64

.text
main:
    li    $v0, 5                #  Le o numero de iteracoes fornecido pelo usuario
    syscall                     #
    move    $s0, $v0     #

    la    $t0, Estimativa       # Carrega o valor da estimativa inicial da memória
    l.d    $f0, 0($t0)          # no registrador $s1

    la    $t0, X                # Carrega o valor de X da memória no registrador $s2
    l.d    $f2, 0($t0)          #

    la    $t0, dois      # Carrega o valor de dois da memória no registrador $s2
    l.d    $f6, 0($t0)          #
```

```
    sqrt.d    $f8, $f2

loop:

    jal    raiz_quadrada          # Chama o procedimento "raiz_quadrada"

    addi     $s0, $s0, -1         # Contador do loop
    bne    $s0, $zero, loop    #

    li    $v0, 10                 # Termina o programa por meio de uma
    syscall                       # chamada de sistema

raiz_quadrada:

    div.d    $f4, $f2, $f0        # X / Estimativa

    add.d    $f4, $f4, $f0        # (X / Estimativa) + Estimativa

    div.d    $f0, $f4, $f6        # ((X / Estimativa) + Estimativa) / 2

    jr    $ra            # Retorna a main
```

Fonte: Elaborado por autores(2023)

Para a questão 1 implementamos um programa que calcula uma estimativa da raiz quadrada de um número x usando o método de Newton. O programa solicita ao usuário o número de iterações a serem realizadas. Em seguida, ele itera por um loop, chamando o procedimento raiz_quadrada a cada iteração para refinar a estimativa da raiz.

Dentro do procedimento são realizados cálculos para melhorar a estimativa da raiz quadrada com base no método de Newton. O processo envolve dividir X pela estimativa atual, adicionar essa estimativa à divisão e, em seguida, dividir o resultado por 2. Isso atualiza a estimativa da raiz. Após o número especificado de iterações, o programa é encerrado por meio de uma chamada de sistema.

3.2- Questão 2

Código 2: Código para aproximação de Seno

```
.data
    qtd_termos: .word 20
    x: .double 0.1

.text
main:
    la    $t0, x                  # Coloca o registrador $f0 com o valor de X
    ldc1    $f0, 0($t0)           #

    la    $t0, qtd_termos         # Coloca o registrador $s1 com o valor de qtd_termos
    lw    $s1, 0($t0)     #

    move    $a0, $s1      # Passa a quantidade de termos da aproximacao como
                          # parametro para o procedimento seno


    jal     seno                  # Chama o procedimento seno

    li    $v0, 10                 # Termina o programa por meio de uma
    syscall                       # chamada de sistema
```

```
#=============================================================================================
seno:
    #=============================================
    # Calcula aproximacao de seno de x com n termos
    #=============================================

    # Recebe em $f0 o numero base(X) e em $a0 a
    # quantidade de termos da aproximacao

    # $s0 = indice de controle do laco
    # $f0 = base da aproximacao
    # $f26 = registrador de retorno

    addi     $sp, $sp, -12              # Aumenta a pilha e coloca registradores
    sw    $s0, 4($sp)             # utilizados nessa funcao para a memoria
    sw    $t0, 8($sp)             #
    sw    $t1, 12($sp)                    #

    li    $s0, 0                          # Inicia indice de controle do laco

loop_seno:
    # Calcula (-1)^n e armazena em $f6 ------------------------------------------------
    addi    $sp, $sp, -20                 # Aumenta a pilha para salvar $f0, $a0 e $ra
    sdc1    $f0, 4($sp)                   #
    sw    $a0, 12($sp)                    #
    sw    $ra, 16($sp)                    #

    addi     $t0, $zero, -1               # Carrega os registradores $f30 e $f31 com -1
    add    $t1, $zero, $zero      #
    mtc1.d  $t0, $f0              #
    cvt.d.w $f0, $f0              #

    move     $a0, $s0             # Atualiza $a0 com o indice do laco atual

    jal    potencia               # Chama o procedimento para calcular a potencia

    ldc1    $f0, 4($sp)                   # Retorna pilha e valor de $f0, $a0 e $ra
    lw    $a0, 12($sp)                    #
    lw    $ra, 16($sp)                    #
    addi    $sp, $sp, 20                  #

    mov.d     $f6, $f30                   # Move o resultado para $f6
    # ---------------------------------------------------------------------------------

    # Calcula x^(2.n+1) e armazena em $f8 ---------------------------------------------
    addi    $sp, $sp, -12                 # Aumenta a pilha para salvar $a0 e $ra
    sw    $a0, 4($sp)             #
    sw    $ra, 8($sp)             #

    li    $t0, 2                          # Calcula (2.n + 1) e armazena em $a0
    mul     $t0, $t0, $s0                 #
    addi     $t0, $t0, 1                  #
    move    $a0, $t0              #

    jal    potencia               # Chama o procedimento para calcular a potencia

    lw    $a0, 4($sp)             # Retorna pilha e valor de $a0 e $ra
    lw    $ra, 8($sp)             #
    addi    $sp, $sp, 12                  #

    mov.d    $f8, $f30                    # Move o resultado para $f8
    # ---------------------------------------------------------------------------------
```

```
    # Calcula (2.n + 1)! e armazena em $f10 ------------------------------------------------
    addi    $sp, $sp, -20                # Aumenta a pilha para salvar $a0
    sdc1    $f0, 4($sp)                  #
    sw    $ra, 12($sp)                   #

    li    $t0, 2                         # Calcula (2.n + 1) e armazena em $a0
    mul     $t0, $t0, $s0                #
    addi     $t0, $t0, 1                 #

    li    $t1, 0                         # Carrega o registrador $t1 com zero

    mtc1.d     $t0, $f0                  # Move (2.n + 1) para $f0
    cvt.d.w $f0, $f0             #

    jal    fatorial              # Chama o procedimento para calcular o fatorial

    ldc1     $f0, 4($sp)                 # Retorna pilha e valor de $a0
    lw    $ra, 12($sp)                   #
    addi    $sp, $sp, 20                 #

    mov.d    $f10, $f28                  # Move o resultado para $f10
    # ------------------------------------------------------------------------------------

    mul.d    $f6, $f6, $f8               # $f6 = (-1)^n * x^(2.n+1)
    div.d    $f6, $f6, $f10              # $f6 = $f6 / (2.n+1)!
    add.d    $f26, $f26, $f6             # Adiciona o resultado de uma iteracao completa
                                         # ao somatorio


    addi    $s0, $s0, 1
    bne    $s0, $a0, loop_seno

    lw    $s0, 4($sp)             # Retorna antigos valores dos registradores
    lw    $t0, 8($sp)             # para seu correto local e diminui a pilha
    lw    $t1, 12($sp)                   #
    addi     $sp, $sp, 12                #

    jr    $ra                    # Retorna ao procedimento em que foi chamado


#============================================================================================
potencia:

    #=========================================
    #   Calcula potencia X^N
    #=========================================

    # Recebe em $f0 o numero base(X) e em $a0 o expoente (N)
    # e retorna o valor em $f30

    # $s0 = indice de controle do laco
    # $f0 = base da potencia
    # $f30 = registrador de retorno

    addi     $sp, $sp, -12               # Aumenta a pilha e coloca registradores
    sw    $s0, 4($sp)             # utilizados nessa funcao para a memoria
    sw    $t0, 8($sp)             #
    sw    $t1, 12($sp)                   #

    addi     $t0, $zero, 1               # Carrega os registradores $f30 e $f31 com 1
    add    $t1, $zero, $zero      #
    mtc1.d  $t0, $f30             #
    cvt.d.w $f30, $f30                   #
```

```
    beq    $a0, $zero, return_pot         # Caso o expoente seja zero retorna 1 como resultado

    move    $s0, $a0              # Inicializa $s0 como indice de controle para loop_pot

loop_pot:

    # for (int i = N; i > 0; i--) {resultado = resultado * x}

    mul.d    $f30, $f30, $f0              # Faz a multiplicacao:          resultado * X
                                          #
    addi    $s0, $s0, -1                  # Controle do loop vezes que   necessario
    bne    $s0, $zero, loop_pot           # fazer a multiplicacao

return_pot:
    lw    $s0, 4($sp)             # Retorna antigos valores dos registradores
    lw    $t0, 8($sp)             # para seu correto local e diminui a pilha
    lw    $t1, 12($sp)                    #
    addi     $sp, $sp, 12                 #

    jr    $ra                     # Retorna ao procedimento em que foi chamado

#=============================================================================================

fatorial:

    #=========================================
    #           Calcula fatorial X!
    #=========================================

    # Recebe em $f0 o numero base(X)

    # s0 = indice de controle do laco
    # f0 = base do fatorial
    # f28 = registrador de retorno

    addi     $sp, $sp, -12                # Aumenta a pilha e coloca registradores
    sw    $s0, 4($sp)             # utilizados nessa funcao para a memoria
    sw    $t0, 8($sp)             #
    sw    $t1, 12($sp)                    #

    cvt.w.d $f2, $f0              # Move o valor da base do fatorial para
    mfc1.d  $t0, $f2              # checar se   1 ou 0

    beqz    $t0, return_one               # Caso seja 1 ou 0 pular para return_one
    beq    $t0, 1, return_one     #

    move    $s0, $t0              # Inicial o indice de controle do laco

loop_fat:

    addi    $s0, $s0, -1

    add     $t0, $zero, $s0               # Carrega os registradores $f2 e $f3 com $s0
    add    $t1, $zero, $zero      #
    mtc1.d  $t0, $f2              #
    cvt.d.w $f2, $f2              #

    mul.d    $f0, $f0, $f2                # Faz a multiplicacao:          resultado * X

    bne    $s0, 1 , loop_fat      # Controle do loop vezes que   necessario
                                          # fazer a multiplicacao

return_fat:
```

```
    mov.d    $f28, $f0                  # Move o resultado para o registrador de retorno

    lw    $s0, 4($sp)             # Retorna antigos valores dos registradores
    lw    $t0, 8($sp)             # para seu correto local e diminui a pilha
    lw    $t1, 12($sp)                    #
    addi     $sp, $sp, 12                 #

    jr    $ra                     # Retorna ao procedimento em que foi chamado

return_one:
    addi     $t0, $zero, 1                # Carrega os registradores $f28 e $f29 com 1
    add    $t1, $zero, $zero      #
    mtc1.d  $t0, $f28             #
    cvt.d.w $f28, $f28                   #

    lw    $s0, 4($sp)             # Retorna antigos valores dos registradores
    lw    $t0, 8($sp)             # para seu correto local e diminui a pilha
    lw    $t1, 12($sp)                    #
    addi     $sp, $sp, 12                 #

    jr    $ra                     # Retorna ao procedimento em que foi chamado

#=============================================================================================
```

Fonte: Elaborado por autores(2023)

Para a questão 2, desenvolvemos um programa em assembly MIPS que calcula uma aproximação do valor do seno de um número X usando uma série trigonométrica. O programa solicita ao usuário a quantidade de termos da série a serem usados na aproximação.

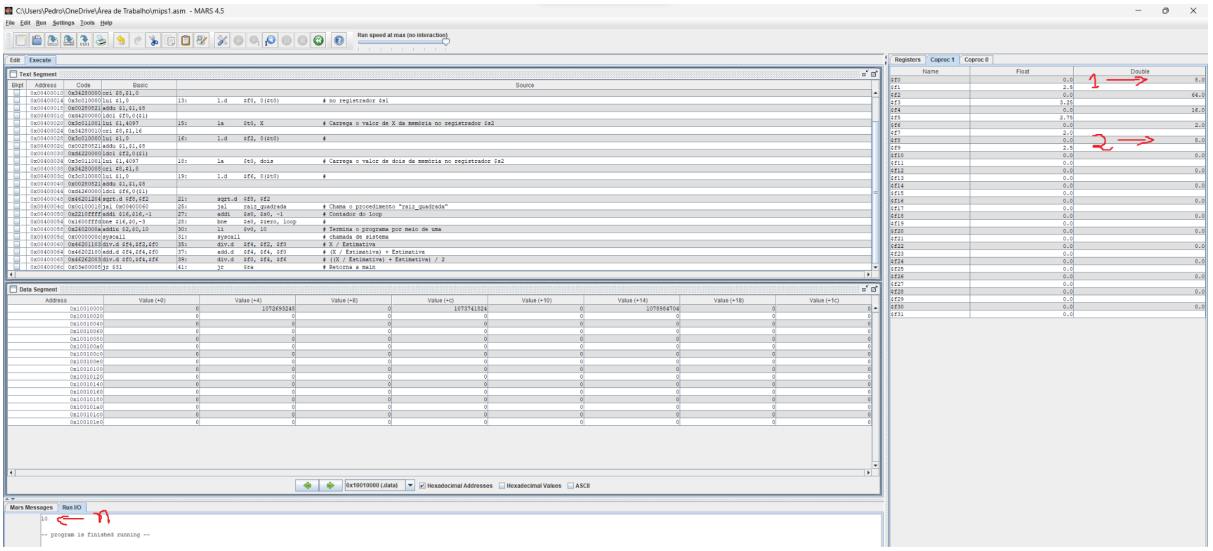
O fluxo do programa começa na função main, onde a quantidade de termos e o valor de X são carregados da memória para os registradores. Em seguida, o programa chama a função seno, passando o número de termos e X como parâmetros.

Dentro da função seno, ocorre o cálculo da aproximação do seno usando a série de Taylor. O programa itera por um loop controlado pelo índice $s0, onde cada iteração calcula um termo da série e o adiciona ao resultado acumulado $f26.

O código também usa procedimentos auxiliares de potência e fatorial para calcular potências e fatoriais, que são parte do cálculo da série de Taylor. Essas funções usam instruções semelhantes para controlar loops e realizar operações matemáticas. O resultado final é a aproximação do seno de X após o número especificado de termos da série.

4- Saídas dos códigos:

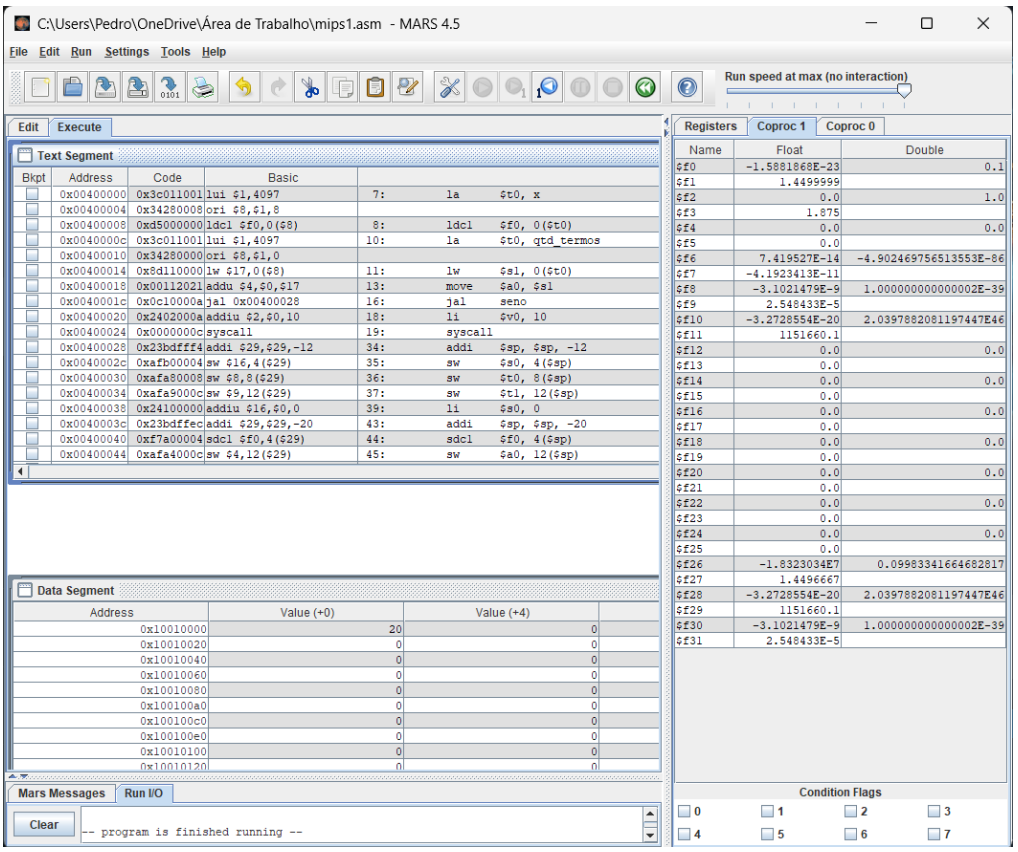
Figura 3: Saída do código 1



Fonte: Elaborado por autores(2023)

Na figura 3, a seta vermelha 1 aponta o valor da aproximação atingida pelo código, a seta 2 aponta o valor exato atingido pela instrução **sqrt.d.** A seta n aponta a entrada das n iterações a serem realizadas.

Figura 4: Saída do código 2



Fonte: Elaborado por autores(2023)

5- Conclusão

Ao longo deste laboratório, alcançamos sucesso no desenvolvimento e resolução das questões propostas, 1 e 2 . Através da execução das tarefas, fomos capazes de explorar e aprofundar nosso entendimento nos conteúdos ministrados durante as aulas teóricas.