Relatório do Laboratório 03

Alunos: Vinicius Henrique Ribeiro (23200351) e Lucas Furlanetto Pascoali (23204339)
Professor: Marcelo Daniel Berejuck
Disciplina: Organização de Computadores I

Questão 1

A primeira questão solicita a implementação de um procedimento que realizasse o método iterativo de Newton, a fim de encontrar, de forma aproximada, a raiz quadrada de um número x, utilizando-se da seguinte fórmula, seja a estimativa inicial 1:

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

O programa deveria cumprir os seguintes requisitos:

- Possuir um procedimento chamado raiz_quadrada, que recebe um parâmetro x de precisão dupla, calcula e retorna o valor aproximado da raiz quadrada de x;
- Escreva um loop que se repita n vezes e calcule n valores de estimativa e, em seguida, retorne a estimativa final após n iterações. O valor de n deve ser informado pelo usuário;
- O número a ser calculado deve ser fornecido pelo teclado;
- O programa principal deverá chamar o procedimento raiz_quadrada para realizar o cálculo da raiz;
- Todos os cálculos devem ser feitos usando instruções e registradores de ponto flutuante de precisão dupla;
- Compare o resultado da instrução sqrt.d com o resultado do seu procedimento raiz_quadrada.

Na seção *.data* do programa temos a declaração do valor de n, que o usuário pode modificar para aumentar ou diminuir a precisão do resultado. O valor de x que vai ser o parâmetro do procedimento raiz_quadrada. A estimativa que tem valor inicial de 1 e uma constante chamada de dois, que possui o valor inicial de 2.0 e serve para auxiliar o cálculo da estimativa.

```
.data
        n:              .word 20
        x:              .double 0.0
        dois:           .double 2.0
        estimativa:     .double 1.0
```

Já na seção *.text* temos os procedimentos MAIN, raiz_quadrada, raiz_loop, end_loop e EXIT. No MAIN, inicialmente realizamos a leitura do teclado, para obter o valor de X fornecido pelo usuário e salvamos esse valor na memória. Depois pulamos para o procedimento raiz_quadrada, após isso carregamos o valor de X para o registrador $f4 e calculamos a raiz quadrade de X através da instrução *sqrt.d*, para compararmos com o valor calculado através do método de Newton.

```
MAIN:
        li      $v0, 7
        syscall
        s.d     $f0, x

        jal     raiz_quadrada

        l.d     $f4, x
        sqrt.d  $f4, $f4

        j       EXIT

```

Partindo para o procedimento raiz_quadrada, começamos carregando os valores da memória para os registradores. O registrador $t0 recebe n, $f2 recebe X, $f4 recebe a estimativa e $f8 recebe dois. Depois entramos no *label* raiz_loop, no qual iniciamos comparando se o valor de n é igual a zero, caso seja, "pulamos" para o *label* end_loop. Senão, prosseguimos para realizar a divisão de X pela estimativa, após isso somamos a estimativa a esse valor, por fim dividimos tudo por dois, subtraímos 1 de $t0 (n) e reiniciamos o loop.

```
raiz_quadrada:
        lw      $t0, n
        l.d     $f2, x
        l.d     $f4, estimativa
        l.d     $f8, dois
raiz_loop:
        beq     $t0, $zero, end_loop

        div.d   $f6, $f2, $f4
        add.d   $f4, $f6, $f4
        div.d   $f4, $f4, $f8
        sub     $t0, $t0, 1
        j       raiz_loop
```

Já o *label* end_loop apenas salva o resultado do cálculo no registrador $f0 e retorna para o main. Enquanto o *label* EXIT apenas finaliza o programa.

```
end_loop:
        mov.d   $f0, $f4
        jr $ra

EXIT:   nop
```

Ao executar o programa com diferentes valores de n e de x, é perceptível que quanto maior o número de n, mais próximo o valor do método iterativo de newton chegará do valor calculado pela instrução *sqrt.d*, quando executamos com 20 iterações, chegamos a conseguir o mesmo valor.

Questão 2

A segunda questão solicita a implementação de um procedimento que realizasse uma aproximação da função *seno* utilizando séries de taylor até o vigésimo termo. A fórmula é:

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

O programa deveria seguir apenas dois requisitos:

- Limite seu cálculo aos primeiros 20 termos da série;
- Mostre que o resultado obtido com o seu procedimento está convergindo para o resultado esperado.

Na seção *.data* temos alguns valores de x, um array, usados para testar o nosso programa, *sinal_monomio* que foi utilizado para ficar alternando entre soma e subtração dos termos da série, *numero_1,* um quebra linha e *resultado* que contém os valores 1 e 0.

```
.data
        x:              .double -5.5 -4 -1 0, 3.141592, 4, 5, 10, 15, 17, 18, 19
        sinal_monomio:  .double 1
        numero_1:       .double 1
        resultado:      .double 0
        s:              .asciiz "\n"
```

Seguindo temos a função *main* que fica responsável por testar todos os valores do array *x.* Nela, começamos passando o endereço base do array *x* para $s0 e 0 para $s1, que será o nosso contador do loop *loop_testes*. Dentro do loop, percorremos *x*, e para cada iteração chamamos a função *seno_x*, que faz a aproximação. Temos também duas chamadas de sistemas para imprimir o resultado.

```
.text
main:
        la      $s0, x
        li      $s1, 0
loop_testes:
        bge     $s1, 12, loop_testes_exit
        addi    $s1, $s1, 1
        l.d     $f0, 0($s0)
        addi    $s0, $s0, 8
        jal     seno_x
        li      $v0, 3
        mov.d   $f12, $f2
        syscall
        li      $v0, 4
        la      $a0, s
        syscall
        j loop_testes
loop_testes_exit:

        j       exit_programa
```

A função seno faz a chamada de duas funções: *potencia_n* e *fatorial*. Para fazer as chamadas primeiramente salvamos o valor de $ra na pilha. Feito isso, podemos inicializar os registradores $f4 com o parâmetro passado por $f0, $f6 com o sinal da operação do monômio de uma determinada iteração, $f8 para armazenar o acumulado do resultado e $t0 o expoente.

O *loop_seno* é responsável por fazer as chamadas e o cálculo de cada termo. De início temos a instrução *bgt* para controlar o número de iterações, nela foi colocado um máximo de 41 para o expoente, tendo em vista que incrementamos o expoente de dois em dois. O primeiro bloco constroi os argumentos de *potencia_n*, que retorna o valor no registrador $f2. O próximo bloco faz a chamada do *fatorial*, que retorna o valor em $f10. A parte final do loop fica encarregada de fazer o cálculo da divisão ($f2 / $f10), inverter o sinal para próxima operação do monômio, acumular em $f8 os valores obtidos pela divisão e incrementar o expoente.

O fim da função recupera o valor do $ra da pilha para retornamos ao *main.*

```
# recebe o argumento em f0 e retorna em f2
seno_x:
        addi    $sp, $sp, -4            # incrementa a pilha
        sw      $ra, 4($sp)             # salva o endereço de retorno

        mov.d   $f4, $f0                # salva x em f4
        l.d     $f6, sinal_monomio      # configura uma variável para alterar o sinal da operação dos monômios
        l.d     $f8, resultado                  # acumulador do resultado
        li      $t0, 1                  # inicializa t0 (expoente e fatorial)
loop_seno:
        bgt     $t0, 41, exit_loop_seno         # condição para o loop

        #move   $a0, $s0        # x é passado como base no argumento de potencia_n
        move    $a0, $t0        # a0 é o expoente
        mov.d   $f0, $f4        # coloco o valor de x na base
        jal     potencia_n      # chamada da função potencia_n
        # retorno de potencia_n está em f2

        move    $a0, $t0        # passa o argumento para fatorial
        jal     fatorial        # o resultado do fatorial está f10

        div.d   $f2, $f2, $f10  # pega o retorno e divide t1, guardando no próprio t1

        mul.d   $f2, $f2, $f6   # configura o sinal do monômio
        neg.d   $f6, $f6        # inverte o sinal para o próximo monômio

        add.d   $f8, $f8, $f2   # soma o monômio no acumulador

        addi    $t0, $t0, 2     # incrementa o expoente e fatorial para a próxima iteração
        j       loop_seno
exit_loop_seno:
        mov.d   $f2, $f8
        addi    $sp, $sp, 4
        lw      $ra, 0($sp)
        jr      $ra
```

A função *potencia_n* recebe o argumento do expoente e da base em $a0 e $f0, respectivamente. Seu funcionamento é simples: O expoente é decrementado em 1 a cada iteração, enquanto o valor em $f2 (que é inicializado em 1) é multiplicado pela base, acumulando seu valor. Usamos $f2 como retorno.

```
# procedimento para calcular a n-ésima potência (f0, a0) = (base, expoente) e retorna em f2 também
# utiliza f0 e f2
potencia_n:
        l.d     $f2, numero_1           # configura o retorno
loop_pot:
        beqz    $a0, exit_loop_pot
        addi    $a0, $a0, -1    # decrementa o expoente
        mul.d   $f2, $f2, $f0   # acrescente no acumulador a parcela da potência
        j       loop_pot
exit_loop_pot:
        jr      $ra     # fim do procedimento potencia_n
```

No *fatorial*, recebemos um número em $a0 como argumento e retornamos em $f10, que está sendo inicializado com o número 1. Usamos $a0 como condição de parada e multiplicador no loop, sendo decrementado a cada iteração, passamos seu valor em formato .*word* para .*double* utilizando as instruções *mtc1.d* e *cvt.d.w* e multiplicamos com o acumulado. Com o valor guardado em $f12, multiplicamos $f10 e guardamos na mesma variável.

Ao fim da imagem podemos ver um *label* para o fim do programa.

```
fatorial:
        l.d     $f10, numero_1          # configura o retorno
loop_fat:
        ble     $a0, 1, exit_loop_fat   # a0 vai decrementar até chegar a 1
        mtc1.d  $a0, $f12               # passa o inteiro para f12
        cvt.d.w $f12, $f12              # converte para double
        mul.d   $f10, $f10, $f12        # acumulador para as multiplicações
        addi    $a0, $a0, -1            # decrementa a0 para achar o próximo multiplicador
        j       loop_fat
exit_loop_fat:
        jr      $ra     # fim do procedimento do fatorial



exit_programa:  # label para quando a main terminar (fim do programa)
```

As saídas para cada valor do array *x*:

```
-- program is finished running (dropped off bottom) --

0.7055403255703919
0.7568024953079275
-0.8414709848078965
0.0
6.535897926662 35E-7
-0.7568024953079275
-0.9589242746631357
-0.5440211107317372
0.6558295384920879
0.2091102514134828 3
12.707730031657817
135.54935999952806
```

Os valores aproximados da função seno:

```
# sen(-5.5) ≈ 0.7055
# sen(-4) ≈ 0.7568
# sen(-1) ≈ -0.8415
# sen(0) = 0
# sen(3.141592) ≈ 0 (pois π ≈ 3.141592, e sen(π) = 0)
# sen(4) ≈ -0.7568
# sen(5) ≈ -0.9589
# sen(10) ≈ -0.5440
# sen(15) ≈ 0.6503
# sen(17) ≈ -0.9614
# sen(18) ≈ -0.7509
# sen(19) ≈ 0.1499
```

Podemos perceber que para valores próximos de 0, que é onde o nosso polinômio de taylor foi centralizado, temos valores muito próximos do real. A partir do valor 15 em radianos nós perdemos precisão, o que é esperado devido ao grau do polinômio não ser tão grande.