CENTRO TECNOLÓGICO
Universidade Federal de Santa Catarina

Laboratório 3

Alunas: Jéssica Regina dos Santos e Myllena da Conceicao Correa
Matrícula: 22100626 e 22104061
Disciplina: Organização de Computadores I
Professor: Marcelo Daniel Berejuck
Entrega: 30/04/25

Objetivo: O objetivo deste relatório é descrever a implementação e análise de dois procedimentos numéricos em Assembly: a aproximação da raiz quadrada utilizando o método de Newton-Raphson e a aproximação da função seno com a Série de Taylor. Busca-se avaliar a precisão dos resultados e consolidar o uso de operações em ponto flutuante na linguagem Assembly.

Questao 1 - **Método Iterativo de Newton-Raphson**

O método iterativo de Newton-Raphson é uma técnica utilizada para encontrar raízes de funções. No caso da raiz quadrada de um número x, a ideia é começar com uma estimativa inicial e, em seguida, aplicar repetidamente uma fórmula iterativa para obter uma estimativa cada vez melhor.
A fórmula iterativa que usamos para calcular a raiz quadrada de x é:

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

Figura 1. Fórmula da Estimativa utilizada nas iterações

Ou:

```
Estimativa(n+1) = 1/2 * (Estimativa(n) + (x/Estimativa(n)))
```

Onde:

- Estimativa(n) é a estimativa atual da raiz quadrada de x após a n-ésima iteração;
- Estimativa(n+1) é a estimativa atualizada da raiz quadrada de x após a (n+1)-ésima iteração.

O procedimento `raiz_quadrada` implementa este método em Assembly. Ele recebe um parâmetro x de precisão dupla e um número de iterações n. Dentro de um loop, são calculados n valores de estimativa usando a fórmula iterativa mencionada acima. Em seguida, é retornada a estimativa final após as n iterações.
Detalhes importantes: a estimativa inicial é 1, o valor de n é informado pelo usuário e os cálculos são feitos usando instruções adequadas e registradores de ponto flutuante de precisão dupla, como na descrição do laboratório.
Para comparar o resultado do procedimento `raiz_quadrada` com a função ‘sqrt.d’, que calcula a raiz quadrada exata, calcula-se o erro absoluto entre os dois resultados. O erro absoluto é simplesmente o módulo da diferença entre os dois valores.

O código feito em assembly para implementar essa função foi feito dessa maneira

1) Bloco de declaração de variáveis relacionadas ao procedimento; X: representa o número do qual se pretende achar a raiz quadrada (neste caso, 3792); Estimativa: estimativa inicial (igual a 1);
Div_2: usado para dividir o double por 2;

```
# VARIÃ‚VEIS RELACIONADAS AO PROCEDIMENTO:

x: .double 3792
estimativa: .double 1
div_2: .double 2
```

2) Bloco de strings a serem impressas no console;

```
pede_n: .asciiz "Digite a quantidade de iterações desejada: "
mostra_raiz: .asciiz "Raiz esperada: "
mostra_estimativa: .asciiz "Raiz estimada: "
mostra_erro: .asciiz "Erro absoluto: "
nova_linha: .asciiz "\n" # símbolo para imprimir nova linha (importante para após imprimir o número necessário)
```

3) Blocos para o carregamento dos endereços das variáveis nos registradores; $s0 recebe endereço de x, $s1 recebe endereço de estimativa e $s2 recebe endereço de div_2; $f0 <= x, $f2 <= estimativa, $f4 <= div_2;

```
la $s0, x
la $s1, estimativa
la $s2, div_2

l.d $f0, 0($s0)
l.d $f2, 0($s1)
l.d $f4, 0($s2)
```

4) Blocos para o recebimento de input de usuário (syscall);
De 30 até 32, é carregado em $v0 o comando para imprimir string, depois carregado o endereço da string que pede pro usuário digitar o valor de n, enfim, acontece a chama de sistema.
Em seguida, o usuário pode digitar a quantidade de iterações (variável n). O valor n é movido para o registrador de argumento $a para ser utilizado no procedimento.

```
li $v0, 4
la $a0, pede_n
syscall

li $v0, 5
syscall
move $a0, $v0
```

5) Chamada do procedimento `raiz_quadrada;`

```
jal raiz_quadrada
```

6) Bloco de outputs do programa;
Ocorre o carregamento de cada string, com os conformes resultados após o procedimento ser realizado. Equivalente a figura abaixo, no terminal de saída:

```
li $v0, 4
la $a0, mostra_estimativa
syscall

li $v0, 3
movf.d $f12, $f2
syscall


li $v0, 4
la $a0, nova_linha
syscall
```

```
65 li $v0, 3
66 sqrt.d $f12, $f0
67 syscall
68
69 # imprime uma nova linha
70 li $v0, 4
71 la $a0, nova_linha
72 syscall
73
74 # Output do erro absoluto
75
76 li $v0, 4
77 la $a0, mostra_erro
78 syscall
79
80 li $v0, 3
81 sub.d $f12, $f12, $f2
82 abs.d $f12, $f12
83 syscall
84
85 j exit
```

Demonstração teste com n loops. Stings + valores de variáveis atribuídos às mesmas.
Importante citar que nas linhas 81 e 82 ocorre o cálculo do valor absoluto (abs|raiz - estimativa|) a partir das instruções `sub.d e abs.d`, com os $f apropriados.

```
Digite a quantidade de iterações desejada: 10
Raiz estimada: 61.5792172733306
Raiz esperada: 61.5792172733261
Erro absoluto: 4.476419235288631E-13
```

Figura 2.

7) Bloco do funcionamento do procedimento (com estrutura de dados pilha);

```
addi, $sp, $sp, -4
sw $ra, 0($sp)

beqz $a0, fim_raiz
div.d $f6, $f0, $f2
add.d $f6, $f6, $f2
div.d $f2, $f6, $f4

addi $a0, $a0, -1
jal raiz_quadrada

fim_raiz:
# retirar dados na pilha
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

Estrutura de dados:

```
#salva dados na pilha
addi, $sp, $sp, -4 # aumenta o tamanho da pilha em 4
sw $ra, 0($sp) # adiciona endereço de retorno na pilha

# retirar dados na pilha
lw $ra, 0($sp) # remove endereço de retorno na pilha
addi $sp, $sp, 4 # diminui o tamanho da pilha em 4
jr $ra
```

"Lógica matemática" do procedimento:

```
beqz $a0, fim_raiz # se n = 0, sai das recursões
div.d $f6, $f0, $f2 # $f6 = x/estimativa
add.d $f6, $f6, $f2 # $f6 = (x/estimativa) + estimativa div.d
$f2, $f6, $f4 # estimativa = ((x/estimativa) + estimativa)/2

addi $a0, $a0, -1 # n = n -1
jal raiz_quadrada # chama o procedimento para n-1 e nova
estimativa (recursão)
```

Saídas de programa considerando loops variando entre 5 e 50 para avaliar o valor do erro absoluto:

n_loops = 2

```
Digite a quantidade de iteraÃ§Ãµes desejada: 2
Raiz estimada: 949.2497363564461
Raiz esperada: 61.5792172733626l
Erro absoluto: 887.6705190830835
```

n_loops = 5

```
Digite a quantidade de iteraÃ§Ãµes desejada: 5
Raiz estimada: 128.9699173580611
Raiz esperada: 61.5792172733626l
Erro absoluto: 67.3907000846985
```

n_loops = 8

```
Digite a quantidade de iteraÃ§Ãµes desejada: 8
Raiz estimada: 61.6093691515417
Raiz esperada: 61.5792172733626l
Erro absoluto: 0.030151878179111
```

n_loops = 11

```
Digite a quantidade de iteraÃ§Ãµes desejada: 11
Raiz estimada: 61.5792172733626l
Raiz esperada: 61.5792172733626l
Erro absoluto: 0.0
```

11 iterações já foram suficientes para o erro absoluto ser zerado (raiz estimada = raiz esperada).

<u>Questão 2 - **Aproximação da Função Seno utilizando a Série de Taylor**</u>

A série de Taylor para a função seno é uma representação infinita que expressa o seno de um ângulo em termos de suas derivadas. A série de Taylor para a função seno é dada por:

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

Figura 3. Série de Taylor para a função seno

O procedimento implementado calcula o seno de um parâmetro x, limitado aos primeiros vinte termos da série.

Para números maiores, apenas 20 termos na série começa a se demonstrar insuficiente.

Diferenças com a calculadora foram percebidas a partir de 9 radianos.

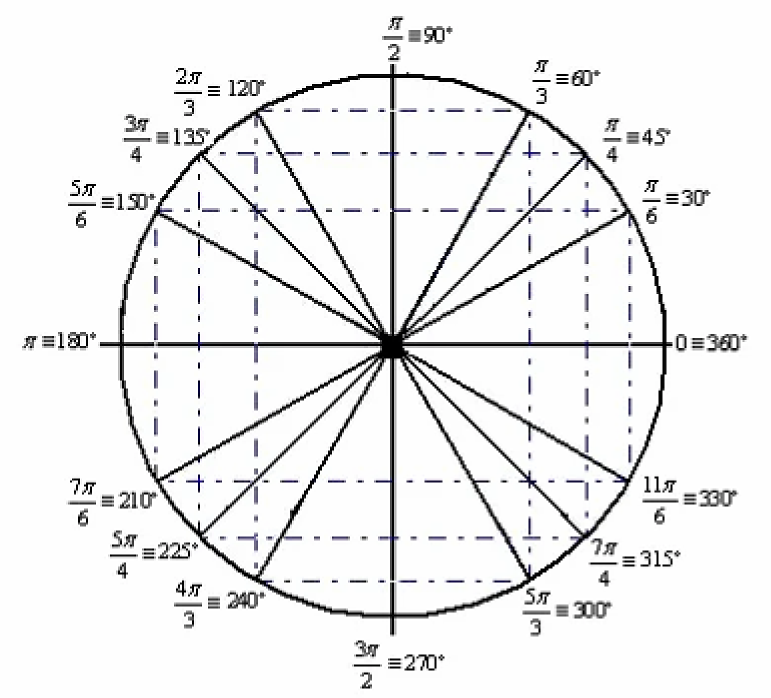

Figura 4. Círculo Trigonométrico, sen(x)

1) Strings para solicitar entrada de x e mostrar o resultado do seno; L5 - L6. Variáveis para o registrador começar com o valor 1 (começando com o valor 0, por exemplo, a multiplicações sucessivas podem acabar nulas, e não queremos isso) e realizar (-1)^n.

```
1  .data
2
3  pede_x: .asciiz "Digite o valor de x: "
4  mostra_seno: .asciiz "Sen(x) = "
5  menos_um: .double -1
6  um: .double 1
```

2) Impressão e pedido de valor ao usuário a partir de “Digite o valor de x:”, no terminal do MARS;
Recebe x: carrega 7 para receber em double e faz o syscall para receber o valor de x, armazenando-o em $f0;
Faz um li para carregar o valor de n usado para calcular a série de Taylor, até o vigésimo termo (n = 19), então chama o procedimento que calcula o seno pela série de Taylor.

```
8  .text
9
10 main:
11 # RECEBE INPUT
12
13 #impressÃ£o da string
14 li $v0, 4
15 la $a0, pede_x
16 syscall
17
18 #recebe x
19 li $v0, 7
20 syscall # $f0 recebe x
21
22 li $a0, 0
23 jal seno
```

3) Bloco de instruções para as impressões de saída no terminal, mostrando a string e o valor do seno de x (input do usuário), aproximado após “passar” pela série.

```
25  # IMPRIME OUTPUT
26
27  #imprime string
28  li $v0, 4
29  la $a0, mostra_seno
30  syscall
31
32  #imprime seno
33  li $v0, 3 #comando de ler seno
34  mov.d $f12, $f6 # $f12 recebe serie de taylor
35  syscall #imprime sen(x)
36
37  j fim # encerra o programa
```

Exemplo, apenas demonstrativo:

```
Digite o valor de x: 1
Sen(x) = 0.8414709848078965
```

A partir da linha 39 do código:

- Como funcionará o procedimento da aproximação da Função Seno utilizando a Série de Taylor?
  Esse bloco do código é uma implementação em Assembly de um algoritmo para calcular o seno de um valor x (input de usuário) usando a Série de Taylor. A série de Taylor é uma maneira de representar funções matemáticas como uma soma infinita de termos. Para calcular o seno usando a Série de Taylor, utiliza-se a seguinte fórmula:

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

O código, a partir daqui, é organizado em três principais procedimentos: `seno`, `fatorial` e `potência`.

No procedimento ‘seno’, são realizados os passos principais do cálculo do seno. Primeiramente, são armazenados os valores de retorno e dos argumentos na pilha para preservar o estado dos registradores. Em seguida, a série de Taylor é calculada recursivamente.
Para cada termo da série, é calculado x elevado a potência de 2n+1, o fatorial de 2n+1, e -1 elevado a potência de n, onde n é o índice do termo.
Os resultados são acumulados em ‘$f6’, que armazena o resultado final do cálculo do seno.
Ao final, os dados são restaurados da pilha e o procedimento retorna.

O procedimento ‘fatorial’ calcula o fatorial de um número n recursivamente. Ele também armazena e restaura o endereço de retorno na pilha. O cálculo do fatorial é feito multiplicando-se o valor atual de $f4 pelo próximo número $f8, e incrementando $f8. Isso é repetido até que n seja igual a zero.

O procedimento ‘potência’ calcula a potência de um número x elevado a um expoente n. Ele segue uma abordagem similar ao fatorial, multiplicando repetidamente o valor atual de $f2 por x até que n seja igual a zero.

Esses procedimentos são interligados para calcular o valor do seno de x usando a Série de Taylor de forma eficiente e precisa. Comentários linha a linha deste bloco são apresentados no arquivo Lab04_Ex2.asm.

Testes Finais com x = 0 e x ≈ π/2, 3π/2, 2π, respectivamente:

```
Digite o valor de x: 0
Sen(x) = 0.0
-- program is finished running (dropped off bottom) --
```

```
Digite o valor de x: 1.5707963267948966192313216916398
Sen(x) = 1.0000000000000002
```

```
Digite o valor de x: 4.712388980384698576939650749193
Sen(x) = -1.0000000000000002
```

```
Digite o valor de x: 6.283185307179586476925286766559
Sen(x) = -1.0925506183112172E-14
```