



# Relatório Laboratório 03

Maria Eduarda Teixeira Costa e Sofia Gazolla da Costa Silva

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina

Organização de Computadores I

Professor Marcelo Daniel Berejuck

Setembro 2025

## 1 Introdução

Neste terceiro laboratório, a atividade proposta consistiu na implementação de dois exercícios em Assembly MIPS, utilizando o simulador MARS. O objetivo principal foi explorar o uso de registradores e instruções de ponto flutuante de precisão dupla, bem como compreender métodos numéricos para cálculo aproximado. O primeiro exercício teve como foco a implementação de um procedimento denominado `raiz_quadrada`, baseado no método iterativo de Newton e o segundo exercício consistiu na implementação de um procedimento para calcular o valor do seno de um parâmetro  $x$ , utilizando a série de Taylor

## 2 Objetivo

Para que se possa entender melhor como os códigos foram desenvolvidos, é importante entender como funcionam as fórmulas implementadas em cada um:

### Atividade 1:

Para encontrar a raiz quadrada de um número seguindo o método iterativo de Newton, primeiro, devemos escolher um número, do qual iremos descobrir a raiz quadrada e que chamamos de  $X$ . Então, escolhe-se um número que possivelmente seja a sua raiz. No caso do programa, ele sempre será 1. Agora, aplica-se a fórmula. A primeira coisa que será feita vai ser dividir  $X$  pela possível raiz, que chamamos de estimativa. O resultado disso será somado à estimativa e, por fim, é feita a média entre esses dois números. Essa média será a nova estimativa, pois ela significa que a raiz está entre esses dois números.

Isso se repete até que a estimativa fique suficientemente próxima do valor real da raiz quadrada de  $X$  (no caso do nosso programa, se repete o número de vezes que o usuário escolher). A cada iteração, o valor obtido vai corrigindo o anterior, porque se a estimativa for muito pequena,  $X$  dividido por ela será um valor muito grande, e se ela for muito grande, o resultado dessa divisão será muito pequena. Fazendo a média entre esses dois extremos, inevitavelmente nos aproximamos da raiz exata.

Esse processo é tão eficiente que, em poucas repetições, já é possível obter um resultado muito preciso. É por isso que o método de Newton é considerado um dos mais rápidos para calcular raízes quadradas: cada passo praticamente dobra a quantidade de casas decimais corretas da estimativa.

## Atividade 2:

O objetivo desta segunda atividade é implementar em Assembly um procedimento que calcule o seno de um parâmetro real  $x$  empregando a série de Taylor.

Explicação da fórmula:

A função seno pode ser aproximada por sua série de Taylor em torno de zero:

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

Nesta expressão, o termo  $(-1)^n$  faz com que os sinais se alternem (positivo e negativo), enquanto o fatorial no denominador  $(2n+1)!$  faz os termos ficarem cada vez menores. Assim, os primeiros termos são:

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

e a soma parcial desses termos converge para o valor de  $\sin(x)$ .

### 3 Desenvolvimento

#### 1. Atividade 1:

A primeira etapa no desenvolvimento do programa foi desenvolver a seção `.data`. Nela, criamos todas as strings que seriam utilizadas como prompts ou mensagens de saída, uma variável `X` para armazenar o valor inserido pelo usuário, como ele é utilizado mais de uma vez ao longo do código e variáveis contendo os dois valores utilizados para o cálculo (1 e 0.5) em ponto flutuante de precisão dupla, já que todo o cálculo é realizado com esse tipo de número.

Após isso, o primeiro prompt é impresso no terminal, solicitando que o usuário insira um `double` para que a sua raiz quadrada seja calculada. Esse número é armazenado na memória.

Depois, solicita-se, com o segundo prompt, que o número de iterações do loop seja inserido, esse sendo um inteiro. Então, o procedimento `raiz_quadrada` é chamado com um `jal`.

Nessa função, os valores 1 e 0.5 são carregados da memória para dois registradores e uma cópia do contador de iterações é feita, para que o número original não seja alterado. Em seguida, o programa entra em um loop. Ele inicia verificando se o contador já é igual a zero, pois, se ele for, o número de iterações desejado já ocorreu e o programa deve ser encerrado. Então, a fórmula em si é aplicada. O valor de `X` é dividido pela estimativa atual de qual será a raiz quadrada (que inicia como 1) e isso é somado à própria estimativa atual. Então, o resultado é multiplicado por 0.5, resultando na nova estimativa.

Antes de voltar ao início do loop, o contador é decrementado, indicando que uma iteração do laço já foi feita. Quando todas as iterações tiverem ocorrido, o programa usa o `jr $ra` para voltar ao ponto onde parou no `main`.

Com o cálculo da raiz aproximada finalizado, esse resultado é impresso, após a sua respectiva mensagem de saída. Então, como solicitado no enunciado do exercício, a raiz

quadrada do mesmo número é calculada utilizando o comando `sqrt.d`. O resultado disso também é impresso na tela, também com sua mensagem de saída.

Então, há o cálculo do erro absoluto, que é a diferença entre as duas raízes, para que saibamos o quão aproximada a raiz calculada pelo método de Newton realmente foi. Para isso, a raiz exata é subtraída da aproximada e é tirado o absoluto desse valor, fazendo uso do comando `abs.d`. Finalmente, o valor do erro absoluto é impresso na tela, junto com sua mensagem de saída e, enfim, o programa é encerrado. Outro detalhe é que há um procedimento chamado `imprimir_newline`, responsável por realizar quebras de linha entre as frases impressas no terminal. Criamos esse procedimento pois a quebra de linha é utilizada mais de uma vez e quisemos simplificar isso.

## 2. Atividade 2:

Durante o desenvolvimento da segunda atividade a fórmula matemática utilizada foi:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Essa fórmula envolve três partes principais: a potência  $x^{2n+1}$ , o fatorial  $(2n+1)!$ , e o sinal alternado  $(-1)^n$ . Cada um deles foi tratado separadamente no código, com o uso de sub-rotinas.

O programa inicia com a seção `.data`, onde são armazenadas constantes em ponto flutuante como `1.0`, `-1.0`, `0.0`, e o fator de conversão de graus para radianos ( $\pi/180$ ). Nela também são definidas mensagens de entrada e saída para interação com o usuário.

No procedimento `main`, o programa solicita ao usuário um valor de ângulo em graus, utilizando o `syscall 7` para leitura de ponto flutuante. Esse valor é armazenado em `$f0`. Em seguida, é convertido para radianos com a multiplicação `x * ( $\pi/180$ )`. Essa operação é realizada com `mul.d $f12, $f0, $f14`, onde: `$f0` contém o valor lido do usuário (em graus), `$f14` contém o fator de conversão, `$f12` recebe o valor convertido para radianos. A variável `n` (contador de termos) é iniciada em `$t0`, e o limite de termos (20) é carregado em `$t1`. A soma acumulada da série é armazenada em `$f2`, iniciada

com zero.

Dentro do loop principal, o programa calcula o expoente com: `mul $t2, $t0, 2` e `addi $t2, $t2, 1`. Esse valor é passado para duas sub-rotinas: `funcao_potencia`, que retorna o resultado em `$f4` e `fatorial`, que retorna o resultado em `$f6`. O sinal alternado é determinado a partir de: se `$t3 == 0`, o termo é positivo (`$f8 = 1.0`); caso contrário, é negativo (`$f8 = -1.0`). O termo da série é então calculado com: `mul.d $f10, $f4, $f8` e `div.d $f10, $f10, $f6`. Esse valor é somado à variável acumuladora `$f2` e o contador `n` é incrementado. Após 20 iterações, o programa imprime o resultado final da aproximação de  $\sin(x)$ , utilizando o `syscall 3` para saída de ponto flutuante.

As sub-rotinas de potência e fatorial foram implementadas com loops: a `funcao_potencia` multiplica o resultado (inicializado em 1, pois  $1! = 1$ ) `$f4` pela base `$f12` até atingir o expoente `$a0`, já `fatorial` converte o contador inteiro para ponto flutuante de dupla precisão (`mtc1` e `cvt.d.w`) e acumula o produto em `$f6`.

### 3. Quantidade de linhas

Assim como nos últimos laboratório, verificamos a quantidade de linhas em `basic` e `source`, recebendo os seguintes valores:

Exercício	Linhas Basic	Linhas Source
01	66	10
02	68	13

Table 1: Quantidade de Linhas por Programa

Percebemos que ainda há mais linhas em `basic` do que em `source`, o que indica que pseudo-instruções ainda estão sendo utilizadas.

## 4 Materiais e métodos

Para o desenvolvimento dessa atividade foi utilizada a linguagem **Assembly** e a IDE **MARS**, na qual foi realizada toda a escrita, execução e depuração de todos os programas

implementados.

## 5 Resultados

Os resultados obtidos mostraram o funcionamento correto de todas as etapas propostas.

### 1. Atividade 1:

Os resultados obtidos foram condizentes com os da fórmula, mostrando que ela foi implementada corretamente. Sobre os resultados de raiz quadrada fornecidos pela fórmula, pode-se perceber que eles dependem principalmente do quão grande é o número que a raiz será calculada e, principalmente, do número de iterações do laço. Para a maior parte dos casos testados, os valores aproximados foram muito próximos dos exatos, com o erro absoluto sendo até mesmo colocado em forma de notação E.

Foram testados valores entre 0 e 1 e todos os resultados foram muito bons, demonstrando que o código também funciona para números fracionários bem pequenos.

Com 5 iterações, para números até aproximadamente 43, os resultados seguem uma precisão de pelo menos duas casas decimais. Após isso, os valores começam a ficar menos precisos, mas seguem muito próximos. Quanto maior o número, mais iterações são necessárias para uma alta precisão. Testamos vários números com 20 iterações, como sugerido no enunciado do exercício e, para a maior parte dos números, a aproximação foi exata. Para números muito grandes a aproximação não foi exata, mas teve uma precisão impressionantemente alta.

```
Digite um double: 0.73
Digite o numero de iteracoes: 5

Raiz quadrada aproximada: 0.8544003745317531
Raiz quadrada exata: 0.8544003745317531
Erro absoluto: 0.0
-- program is finished running --

Digite um double: 436281936.15
Digite o numero de iteracoes: 20

Raiz quadrada aproximada: 20887.36307315981
Raiz quadrada exata: 20887.36307315981
Erro absoluto: 0.0
-- program is finished running --
```

```
Digite um double: 10.46
Digite o numero de iteracoes: 5

Raiz quadrada aproximada: 3.234192334192497
Raiz quadrada exata: 3.234192325759246
Erro absoluto: 8.43325098642822E-9
-- program is finished running --

Digite um double: 57839199.23
Digite o numero de iteracoes: 20

Raiz quadrada aproximada: 7605.2086907592475
Raiz quadrada exata: 7605.208690759248
Erro absoluto: 9.094947017729282E-13
-- program is finished running --
```

Figure 1: Demonstração do funcionamento com diferentes valores

Além de testar com diversos valores apenas para comprovar o funcionamento do programa, também realizamos alguns testes do mesmo valor com diferentes números de iterações, para mostrar como o algoritmo converge para o valor exato rapidamente.

```
Digite um double: 43.21
Digite o numero de iteracoes: 5

Raiz quadrada aproximada: 6.5741514916126516
Raiz quadrada exata: 6.573431371817918
Erro absoluto: 7.201197947335203E-4
-- program is finished running --
```

```
Digite um double: 43.21
Digite o numero de iteracoes: 7

Raiz quadrada aproximada: 6.573431371817918
Raiz quadrada exata: 6.573431371817918
Erro absoluto: 0.0
-- program is finished running --
```

Figure 2: Demonstração com 43.21

```
Digite um double: 1029.35
Digite o numero de iteracoes: 5

Raiz quadrada aproximada: 42.17984717955139
Raiz quadrada exata: 32.08348484812708
Erro absoluto: 10.09636233142431
-- program is finished running --
```

```
Digite um double: 1029.35
Digite o numero de iteracoes: 10

Raiz quadrada aproximada: 32.08348484812708
Raiz quadrada exata: 32.08348484812708
Erro absoluto: 0.0
-- program is finished running --
```

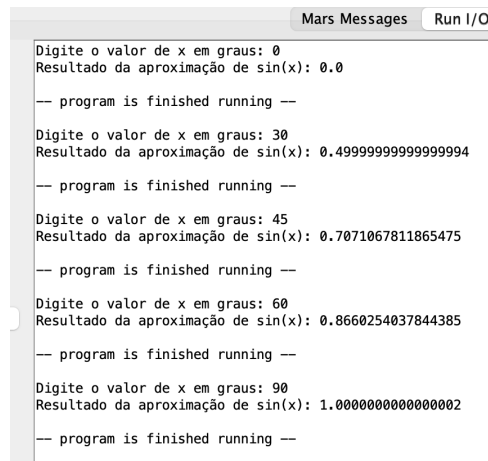
Figure 3: Demonstração com 1029.35

## 2. Atividade 2:

Os resultados obtidos mostraram que a aproximação da função seno por meio da série de Taylor apresentou excelente precisão. Foram realizados testes com ângulos notáveis, como  $0^\circ$ ,  $30^\circ$ ,  $45^\circ$ ,  $60^\circ$  e  $90^\circ$ , e os valores calculados pelo programa mostraram-se praticamente idênticos aos esperados. A diferença entre o valor teórico e o valor obtido foi inferior a  $10^{-15}$  em todos os casos, o que indica uma taxa de erro extremamente baixa. Essa precisão é atribuída ao uso de ponto flutuante em dupla precisão e ao número adequado de termos na série (20), garantindo confiabilidade nos resultados apresentados.

Table 2: Valores do seno para ângulos notáveis

Ângulo (graus)	Ângulo (radianos)	sen(x)
0°	0	0
30°	$\frac{\pi}{6}$	$\frac{1}{2}$
45°	$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$
60°	$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$
90°	$\frac{\pi}{2}$	1



```

Mars Messages Run I/O
Digite o valor de x em graus: 0
Resultado da aproximação de sin(x): 0.0
-- program is finished running --

Digite o valor de x em graus: 30
Resultado da aproximação de sin(x): 0.49999999999999994
-- program is finished running --

Digite o valor de x em graus: 45
Resultado da aproximação de sin(x): 0.7071067811865475
-- program is finished running --

Digite o valor de x em graus: 60
Resultado da aproximação de sin(x): 0.8660254037844385
-- program is finished running --

Digite o valor de x em graus: 90
Resultado da aproximação de sin(x): 1.0000000000000002
-- program is finished running --

```

Figure 4: Resultados Obtidos

## 6 Dificuldades

Durante a realização deste laboratório, enfrentamos alguns desafios, a maioria por se tratar do nosso primeiro contato com operações e registradores em ponto flutuante. O uso do coprocessador 1, responsável pelas instruções de ponto flutuante, exigiu uma compreensão dos registradores `$f` e das instruções específicas como `l.d`, `mul.d`, `div.d`, entre outras.

Outra dificuldade enfrentada foi compreender corretamente a parte matemática em si dos algoritmos propostos nos dois exercícios. Entretanto, após entender como elas



funcionavam, sua implementação foi relativamente simples.

Essas dificuldades contribuíram significativamente para o nosso aprendizado, ampliando a compreensão sobre o funcionamento interno de cálculos matemáticos em baixo nível, além de proporcionar nosso primeiro contato prático com operações em ponto flutuante.

## **7 Conclusão**

Esse foi o primeiro laboratório em que trabalhamos com operações em ponto flutuante e registradores desse tipo, o que foi uma diferença muito importante em relação aos laboratórios anteriores. Ainda assim, a implementação da atividade foi mais simples, pois as fórmulas matemáticas utilizadas nos permitiam seguir uma série de passos no desenvolvimento do nosso código. A estrutura bem definida dos algoritmos contribuiu para uma implementação mais fluída que nas atividades propostas anteriormente.

Além disso, foi uma oportunidade muito boa para desenvolvermos maior familiaridade com registradores de ponto flutuante, suas instruções e alguns cuidados específicos que devemos ter quando operando com eles.