Relatório 04 | Organização de computadores I - INE5411

Alunos: Pedro Henrique Gimenez - 23102766 e Victória Rodrigues Veloso - 23100460

1. Exercício 1

Para o primeiro exercício, o método iterativo de newton foi implementado para encontrar a raiz quadrada de um número utilizando a fórmula de estimativa da figura 1

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

Figura 1. Estimativa da raiz quadrada através do método de newton

1.1 Implementação

1.1.1 Script python

Para implementar o código em assembly, inicialmente foi criado um script em Python. Isso foi feito com o objetivo de fornecer uma visão mais clara dos passos a seguir. O algoritmo em linguagem de alto nível foi elaborado para se assemelhar o máximo possível ao que seria feito em assembly, com a distinção entre labels e a declaração de variáveis de forma clara e organizada.

```python
def raiz_quadrada(x, n):
    # Estimativa inicial
    estimativa = 1.0

    # Loop para calcular n valores de estimativa
    for _ in range(n):
        # Calcula a nova estimativa usando o método de Newton
        estimativa = (( x / estimativa)+estimativa) / 2.0

    return estimativa

def main():
    # Solicita o número e a quantidade de iterações ao usuário
    x = float(input("Digite o número para calcular a raiz quadrada: "))
    n = int(input("Digite o número de iterações desejadas: "))

    # Calcula a estimativa da raiz quadrada usando o método de Newton
    estimativa_final = raiz_quadrada(x, n)
```

```python
    # Exibe o resultado
    print(f"A estimativa da raiz quadrada de {x} após {n} iterações é:
{estimativa_final}")

if __name__ == "__main__":
    main()
```

Figura 2.Script python para o método de newton

1.1.2 Implementação assembly

Iniciamos o código alocando as variáveis nos registradores e, em seguida, realizamos chamadas no sistema para captar os valores de x (valor para o qual queremos descobrir a raiz quadrada) e n (número de iterações). Após isso, o procedimento 'raiz_quadrada' é invocado. Dentro do procedimento, um registrador chamado 'estimativa' é iniciado com o valor 1.0, enquanto outro registrador é inicializado com a constante a ser usada para a divisão. Em seguida, um loop é executado até que o número de iterações 'n' alcance zero. Durante cada iteração do loop, um registrador temporário, como $f2, é utilizado para armazenar os cálculos parciais.

```
raiz_quadrada:


      l.d $f4, 0($s2)             #inicia o registrador estimativa com 1.0
      li $t0, 2             #inicia a constante 2
      mtc1 $t0, $f6               #manda para f6
      cvt.d.w $f6,$f6             #converte para double

    loop:
      addi $a0, $a0,-1        #decrementa o valor de n (iterações)
      div.d $f2, $f0, $f4         #registrador auxiliar(f2) =
(x/estimativa)
      add.d $f2,$f2, $f4          #registrador auxiliar(f2) =
(x/estimativa) + estimativa
      div.d $f2, $f2, $f6         #registrador auxiliar(f2) =
(x/estimativa) + estimativa/2
      mov.d $f4, $f2              #atualiza a estimativa
      bnez $a0, loop          #se n != 0, vai para o loop e repete novamente,
caso contrário volta para a main

      jr $ra                      #volta para a main
Após o final dos cálculos (quando o número de iterações é igual a 0).
Voltamos para a "main" e o resultado é armazenado na memória.
```

1.1.3 Erro absoluto

Após armazenar o resultado, a raiz quadrada é encontrada através da instrução "sqrt.d", a fim de comparar os resultados obtidos.

Para n = 5

$$Erro1 = abs(1.414213562373095 - 1.4142135623730951)/1.4142135623730951 * 100$$
$$Erro1 = 1.57009245868377E - 14$$

Para n = 20

$$Erro2 = abs(1.41213562373095 - 1.412135623730951)/1.412135623730951 * 100$$
$$Erro2 = Erro = 7.86201414345654E - 14$$

Para n = 50

$$Erro3 = abs(1.41213562373095 - 1.412135623730951)/1.412135623730951 * 100$$
$$Erro3 = 7.86201414345654E - 14$$

Conforme o número de iterações aumenta, o erro absoluto diminui, conforme esperado. Também é possível observar que o algoritmo converge antes de 20 iterações.

1.2 Execução do programa

Conforme a figura 3, iremos simular a execução do programa encontrando a raiz quadrada de 2 com 10 iterações

```
Insira o valor de X: 2

Insira o valor de N: 10
```

Figura 3. Valores de entrada do teclado fornecidos pelo usuário

Conforme figura 4, após receber a entrada do teclado, o programa armazena a entrada no registrador $f0 e inicia os registradores estimativa $f4 com 1 e $f6 com a constante 2 (esta última será utilizada para realizar as divisões da estimativa.

Name	Float	Double
$f0	0.0	2.0
$f1	2.0	
$f2	0.0	0.0
$f3	0.0	
$f4	0.0	1.0
$f5	1.875	
$f6	0.0	2.0

Figura 4.Armazenamento de valores nos registradores

Figura 5.Resultado após 1 iteração

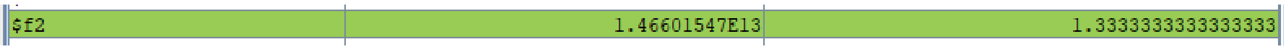

Figura 6.Resultado após 2 iterações

Figura 7.Resultado após 3 iterações

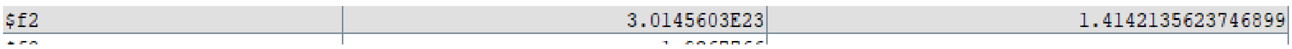

Figura 8. Resultado após 10 iterações

2. Exercício 2

2.1 Implementação

2.1.1 Script python

Assim como no exercício 1, antes de iniciar a aplicação em assembly, um script python foi montado.

```
def fatorial(x):
    if (x == 0):
        return 1
    else:
        return x * fatorial(x-1)

def potencia(x, n):
    valor = 1
    for _ in range(n):
        valor *= x

    return valor

def seno(x):
    # somatoria de     (((-1)**n) / (2*n + 1)! ) *  x**(2*n + 1)
    # vou chamar de:   ( (sinal)  /   (fat)!   ) *   (x**fat)   )
    # vou chamar de:   (       (fator)           *      (x2)     ) = interno
    #                  seno += interno

    seno = 0
```

```python
    for n in range(20):
        sinal = potencia(-1, n) # sinal

        # calculando fat = (2*n + 1)
        fat = n + n
        fat += 1

        # fator = sinal / (2*n + 1)!
        fator = fatorial(fat)
        fator = sinal / fator

        # x2 = x ** (2*n + 1)
        x2 = potencia(x, fat)

        # seno += a iteracao interna
        interno = fator * x2
        seno += interno

    return seno


def main():
    x = float(input("Digite o número para calcular o seno (em radianos): "))
    resultado_aproximado = seno(x)
    print(f"O valor aproximado do seno de {x} radianos é:
{resultado_aproximado}")

if __name__ == "__main__":
    main()
```

Figura 9. Script python para a aproximação do seno

2.1.2 Implementação em assembly

O código foi dividido em três seções principais: fatorial, potenciação e seno, além da seção principal, onde o usuário interage com o programa.
A função fatorial calcula o fatorial de um número inteiro usando recursão. Ela recebe o parâmetro $a0 contendo o número a ser calculado e retorna o resultado em $f4 pois o resultado precisa estar em double para números muito altos. O algoritmo utiliza a técnica de recursão, decrementando o valor de entrada até atingir zero, multiplicando-o pelo resultado da chamada recursiva.
A função potência calcula a potência de um número em ponto flutuante. Ela recebe dois parâmetros: $f2 como base e $a0 como expoente, retornando o resultado em $f8. O algoritmo realiza a multiplicação sucessiva da base até alcançar o expoente desejado.

Já a função seno implementa a fórmula do seno através de uma série de operações. Ela utiliza um loop para iterar 20 vezes, calculando cada termo da série e acumulando o resultado em $f12. Cada termo é calculado usando as funções de potenciação e fatorial, juntamente com operações matemáticas.
Na seção principal, o programa interage com o usuário, solicitando o número para o cálculo do seno em radianos. Em seguida, chama a função seno para calcular a aproximação do seno e exibe o resultado na tela.

2.2 Execução do programa

Iremos iniciar a execução do programa para calcular o valor do sen de $\frac{\pi}{2} \approx 1.57$. Como pode ser visto pela figura 10.

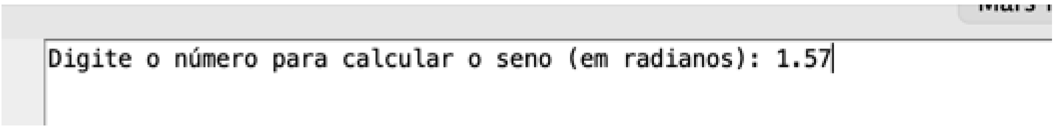

Figura 10. Console do MARS com o input solicitado pelo programa e o valor de 1.57 inserido pelo usuário

Em seguida, o programa irá armazenar o valor em $f0 e começa chamando a função seno. Na função, serão realizadas 20 iterações, nas quais são realizadas as somas da equação `(((-1)**n) / (2*n + 1)! ) *  x**(2*n + 1)` em que n é a iteração atual e x o valor dado pelo usuário.

Para ficar mais fácil de entender vamos nomear partes da equação como:
- sinal = (-1) ** n
- fat = (2*n) + 1
- fator = sinal / (fat)!
- x2 = x ** fat
- interno = fator * x2

Enquanto o valor da iteração ($t1) não chegar em 20, a função irá realizar (as figuras mostram a segunda iteração com x = 1.57 e n = 1) :

- Cálculo do sinal em que armazena (-1) ** n em $f8, utilizando a função potência

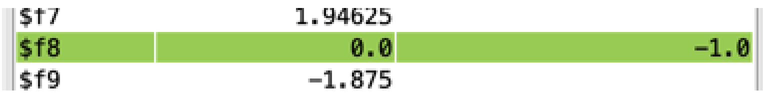

Figura 11. Registrador $f8 armazena (-1) ** 1 para calcular o sinal

- Em seguida, calcula fat = 2*n + 1 que está armazenado no registrador $t2

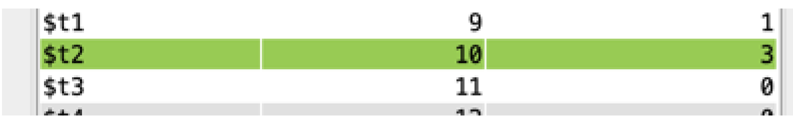

Figura 12. Registrador $t2 armazena 2*1 + 1 = 3 para calcular o fat

- É calculado o fator = sinal / (fat)!, para isso é necessário realizar o fatorial do valor armazenado em $t2. A função fatorial recebe a word e retorna um double armazenado em $f4 para armazenar valores muito grandes (Figura 13). Em seguida é realizada a divisão do sinal pelo fatorial calculado e é armazenada em $f4 também (Figura 14).

$f3	-1.875	
$f4	0.0	6.0
$f5	2.375	
$f6	1.2644384E11	1.57

Figura 13. Registrador $f4 após função fatorial armazenando 3! = 6

$f2	0.0	-1.0
$f3	-1.875	
$f4	1.4660155E13	-0.16666666666666666
$f5	-1.5416666	
$f6	1.2644384E11	1.57

Figura 14. Registrador $f4 após calculator sinal/fat = -⅙ = -0.1666…

- Calcula-se em seguida x2 = x ** fat utilizando novamente a função potencia e o resultado é armazenado no registrador $f8.

$f7	1.94625	
$f8	7.309508E32	3.8698930000000002
$f9	2.2337365	
$f10	0.0	0.0

Figura 15. Registrador $f8 armazena 1.57**3 = 3.869893

- Por fim, é calculado interno = fator * x2 e armazenado em $f6 (Figura 16) e depois é somado o resultado em seno ($f12) (Figura 17).

$f5	-1.5416666	
$f6	-3.0090892E-17	-0.6449821666666666
$f7	-1.7862455	

Figura 16. Registrador $f6 armazena -0.1666 * 3.869893 = -0.644982

$f11	0.0	
$f12	NaN	0.9250178333333334
$f13	1.8562543	

Figura 17. Registrador $f12 armazena 1.57+(-0.644982) = 0.92501783

Após realizar as 20 iterações, $f12 estará com o valor final da aproximação do seno (Figura 18) e depois o valor final será impresso no console (Figura 19).

$f11	0.0	
$f12	2.7279325E13	0.9999996829318349
$f13	1.8749999	

Figura 18. Registrador $f12 armazenando o resultado final após a 20ª iteração

```
Digite o número para calcular o seno (em radianos): 1.57
Digite o número para calcular o seno (em radianos): 0.9999996829318349
-- program is finished running --
```

Figura 19. Console do MARS após o programa ser finalizado

É possível analisar que o programa foi concluído com sucesso e obteve o resultado esperado, pois 1.57 é a aproximação de pi/2 cujo seno é 1.