

# Implementação da Função Raiz Quadrada usando o Método Newton-Raphson Recursivo

Humberto Corrêa Gomes

<sup>1</sup>Disciplina de Organização e Arquitetura de Processadores (OAP)  
Universidade PUC-RS

**Resumo.** Este trabalho apresenta a implementação da função raiz quadrada utilizando o método Newton-Raphson em sua versão recursiva. A implementação foi realizada na linguagem Assembly do processador MIPS, utilizando conceitos fundamentais de arquitetura de computadores como pilha, recursividade, funções e macros. O artigo detalha a implementação tanto em linguagem de alto nível quanto em Assembly MIPS, demonstrando o funcionamento do algoritmo através de capturas de tela do ambiente MARS.

## 1. Introdução

O método de Newton-Raphson é uma técnica numérica iterativa que permite encontrar aproximações de raízes de funções. Este método é baseado na linearização de funções, e sua implementação recursiva para o cálculo da raiz quadrada representa uma abordagem elegante para resolver este problema matemático.

Neste trabalho, implementamos uma versão recursiva do método Newton-Raphson para calcular a raiz quadrada de um número inteiro positivo. A regra de recursividade que utilizamos é definida pela seguinte equação:

$$\sqrt{nr}(x, i) = \begin{cases} 1, & \text{se } i = 0 \\ \frac{\sqrt{nr}(x, i-1) + \frac{x}{\sqrt{nr}(x, i-1)}}{2}, & \text{se } i > 0 \end{cases} \quad (1)$$

Onde  $x$  é o valor para o qual desejamos calcular a raiz quadrada, e  $i$  é o número de iterações que o método executa.

## 2. Algoritmo em Linguagem de Alto Nível

A seguir, apresentamos a implementação do algoritmo Newton-Raphson recursivo em linguagem Python:

```
1 # Função recursiva para calcular a raiz quadrada usando Newton-
  Raphson
2 def sqrt_nr(x, i):
3     # Caso base: quando i = 0, retorna 1
4     if i == 0:
5         return 1
6
7     # Caso recursivo: calcula com base no resultado anterior
8     prev_sqrt = sqrt_nr(x, i-1)
9     return (prev_sqrt + x/prev_sqrt) // 2
10
11 def main():
```

```

12     # Mensagens iniciais
13     print("Programa de Raiz Quadrada      Newton-Raphson")
14     print("Desenvolvedor: Humberto Corr a Gomes")
15
16     while True:
17         # Prompt para entrada de dados
18         print("\nDigite os par metros x e i para calcular sqrt_nr (x,
19 i) ou -1 para abortar a execu  o")
20
21         # Leitura do valor x
22         try:
23             x = int(input("Digite o valor de x: "))
24
25             # Verifica se deve encerrar
26             if x < 0:
27                 print("Programa encerrado.")
28                 break
29
30             # Leitura do n mero de itera es i
31             i = int(input("Digite o n mero de itera es i: "))
32
33             # Verifica se deve encerrar
34             if i < 0:
35                 print("Programa encerrado.")
36                 break
37
38             # Calcula a raiz quadrada usando o m todo Newton-Raphson
39             resultado = sqrt_nr(x, i)
40
41             # Exibe o resultado
42             print(f"sqrt({x}, {i}) = {resultado}")
43
44             except ValueError:
45                 print("Entrada inv lida. Por favor, digite um n mero
46 inteiro.")
47
48 if __name__ == "__main__":
49     main()

```

**Listing 1. Implementação do método Newton-Raphson recursivo em Python**

O algoritmo implementa a regra de recursividade conforme descrita na introdução. A função `sqrt_nr` recebe dois parâmetros: o número `x` para calcular a raiz quadrada e o número de iterações `i`. A função `main` implementa a interface com o usuário, solicitando os valores de entrada e exibindo o resultado calculado.

Note que em Python, usamos a divisão inteira (`//`) para garantir que o resultado seja um número inteiro, similar ao comportamento da divisão em C e em Assembly MIPS. A implementação também inclui tratamento de exceções para garantir que apenas entradas válidas sejam processadas.

### 3. Implementação em Assembly MIPS

A seguir, apresentamos a implementação completa do algoritmo em Assembly MIPS:

```

1 # Programa de Raiz Quadrada - Newton-Raphson Recursivo

```

```

2
3 # Macros
4 .macro print_string(%label)
5     li $v0, 4
6     la $a0, %label
7     syscall
8 .end_macro
9
10 .macro print_int(%reg)
11     li $v0, 1
12     move $a0, %reg
13     syscall
14 .end_macro
15
16 # Dados
17 .data
18 titulo:      .asciiz "Programa de Raiz Quadrada      Newton-Raphson\n"
19 autores:     .asciiz "Desenvolvedor: Humberto Corr a Gomes\n"
20 prompt:      .asciiz "\nExemplo com valores fixos: x=30 e i=190\n"
21 encerrar:    .asciiz "Programa encerrado.\n"
22 result1:     .asciiz "sqrt ("
23 virgula:     .asciiz ", "
24 result2:     .asciiz ") = "
25 newline:     .asciiz "\n"
26
27 # C digo
28 .text
29 .globl main
30
31 main:
32     # Pr logo
33     addi $sp, $sp, -4
34     sw $ra, 0($sp)
35
36     # Mensagens iniciais
37     print_string(titulo)
38     print_string(autores)
39
40     # Exibe mensagem sobre valores fixos
41     print_string(prompt)
42
43     # Valores fixos para demonstra o
44     li $s0, 30      # x = 30
45     li $s1, 190     # i = 190
46
47     # Chama a fun o sqrt_nr
48     move $a0, $s0    # x
49     move $a1, $s1    # i
50     jal sqrt_nr
51     move $s2, $v0     # Salva resultado em $s2
52
53     # Exibe resultado
54     print_string(result1)
55     print_int($s0)    # x
56     print_string(virgula)
57     print_int($s1)    # i

```

```

58     print_string(result2)
59     print_int($s2)      # resultado
60     print_string(newline)
61
62     # Encerra programa ap s uma execu o
63     print_string(encerrar)
64
65     # Ep logo
66     lw $ra, 0($sp)
67     addi $sp, $sp, 4
68
69     # Encerra programa
70     li $v0, 10
71     syscall
72
73 # Fun o sqrt_nr(x, i)
74 # Parmetros:
75 #   $a0 = x (n mero para calcular a raiz)
76 #   $a1 = i (n mero de itera es)
77 # Retorno:
78 #   $v0 = resultado da aproxima o
79 sqrt_nr:
80     # Salva registradores na pilha
81     addi $sp, $sp, -16
82     sw $ra, 12($sp)      # Endere o de retorno
83     sw $s0, 8($sp)       # Para armazenar x
84     sw $s1, 4($sp)       # Para armazenar i
85     sw $s2, 0($sp)       # Para armazenar resultado anterior
86
87     # Copia argumentos para registradores salvos
88     move $s0, $a0        # s0 = x
89     move $s1, $a1        # s1 = i
90
91     # Verifica caso base (i = 0)
92     beqz $s1, caso_base
93
94     # Caso recursivo: chama sqrt_nr(x, i-1)
95     move $a0, $s0
96     addi $a1, $s1, -1
97     jal sqrt_nr
98
99     # Salva resultado da chamada recursiva
100    move $s2, $v0         # s2 = sqrt_nr(x, i-1)
101
102    # Calcula x / sqrt_nr(x, i-1)
103    div $s0, $s2          # x / sqrt_nr(x, i-1)
104    mflo $t0              # t0 = quociente da divis o
105
106    # Calcula (sqrt_nr(x, i-1) + x/sqrt_nr(x, i-1))
107    add $t1, $s2, $t0
108
109    # Divide por 2: resultado / 2
110    srl $v0, $t1, 1       # Shift right logical = divis o por 2
111
112    j fim_funcao
113

```

```

114 caso_base:
115     # Retorna 1 para i = 0
116     li $v0, 1
117
118 fim_funcao:
119     # Restaura registradores da pilha
120     lw $s2, 0($sp)
121     lw $s1, 4($sp)
122     lw $s0, 8($sp)
123     lw $ra, 12($sp)
124     addi $sp, $sp, 16
125
126     # Retorna
127     jr $ra

```

**Listing 2. Implementação do método Newton-Raphson recursivo em Assembly MIPS**

A implementação em Assembly MIPS segue a mesma estrutura do algoritmo em Python, mas com as particularidades da linguagem de máquina. Destacamos:

- O uso de macros para simplificar operações repetitivas (print\_string e print\_int)
- O salvamento e restauração adequados dos registradores na pilha durante as chamadas recursivas
- A implementação do caso base e do caso recursivo da função sqrt\_nr
- A utilização da instrução SRL (Shift Right Logical) para realizar a divisão por 2

## 4. Resultados e Capturas de Tela

Nesta seção, apresentamos as capturas de tela que demonstram o funcionamento do programa no ambiente MARS.

### 4.1. Área de Código Montada

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	4194304	0x23bdfffc	addi \$29,\$29,-4	33: addi \$sp, \$sp, -4
<input type="checkbox"/>	4194308	0xafbf0000	sw \$31,0(\$29)	34: sw \$ra, 0(\$sp)
<input type="checkbox"/>	4194312	0x24020004	addiu \$2,\$0,4	37: <5> li \$v0, 4
<input type="checkbox"/>	4194316	0x3c011001	lui \$1,4097	<6> la \$a0, titulo
<input type="checkbox"/>	4194320	0x34240000	ori \$4,\$1,0	
<input type="checkbox"/>	4194324	0x0000000c	syscall	<7> syscall
<input type="checkbox"/>	4194328	0x24020004	addiu \$2,\$0,4	38: <5> li \$v0, 4
<input type="checkbox"/>	4194332	0x3c011001	lui \$1,4097	<6> la \$a0, autores
<input type="checkbox"/>	4194336	0x3424002d	ori \$4,\$1,45	
<input type="checkbox"/>	4194340	0x0000000c	syscall	<7> syscall
<input type="checkbox"/>	4194344	0x24020004	addiu \$2,\$0,4	41: <5> li \$v0, 4
<input type="checkbox"/>	4194348	0x3c011001	lui \$1,4097	<6> la \$a0, prompt

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	1735357008	1634558322	543515680	2053726546	1635078432	1684107876	757932129	2003127840
268501024	762212212	1752195410	175009651	1936016384	1870032485	1684371052	540701295	1651340616
268501056	1869902437	1919894304	543253874	1701670727	167774835	1835366469	544173168	544042851
268501088	1869373814	544433522	1870162278	2015378035	540029245	1030297749	170930225	1869762560
268501120	1835102823	1852121185	1920099683	779052129	1903362058	2651250	687874092	2112800
268501152	10	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0

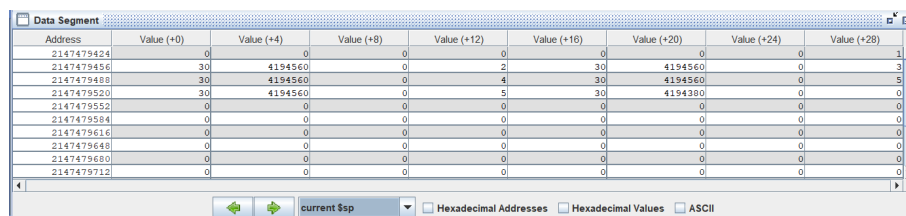
**Figure 1. Área de código montada no MARS**

## 4.2. Estado dos Registradores ao Término de uma Execução

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268501117
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	6
\$t1	9	11
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	30
\$s1	17	5
\$s2	18	5
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194516
hi		0
lo		6

Figure 2. Estado dos registradores ao término da execução

### 4.3. Área de Pilha Utilizada para a Recursividade



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
2147479424	0	0	0	0	0	0	0	1
2147479456	30	4194560	0	2	30	4194560	0	3
2147479488	30	4194560	0	4	30	4194560	0	5
2147479520	30	4194560	0	5	30	4194380	0	0
2147479552	0	0	0	0	0	0	0	0
2147479584	0	0	0	0	0	0	0	0
2147479616	0	0	0	0	0	0	0	0
2147479648	0	0	0	0	0	0	0	0
2147479680	0	0	0	0	0	0	0	0
2147479712	0	0	0	0	0	0	0	0

Figure 3. Área de pilha utilizada durante a recursividade

### 4.4. Exemplo de Execução do Programa



```
PROBLEMAS SAÍDA TERMINAL PORTAS
humb@C:\GitHub\Organizacao-e-Arquitetura-De-Processadores\main & C:\Python312\python.exe c:\GitHub\Organizacao-e-Arquitetura-De-Processadores\Apri
L/T1/sqrt_nr.py
Programa de Raiz Quadrada - Newton-Raphson
Desenvolvedor: Humberto Corrêa Gomes

Digite os parâmetros x e i para calcular sqrt_nr (x, i) ou -1 para abortar a execução
Digite o valor de x: 100
Digite o número de iterações i: 20
sqrt(100, 20) = 10

Digite os parâmetros x e i para calcular sqrt_nr (x, i) ou -1 para abortar a execução
Digite o valor de x: 
```

Figure 4. Exemplo de execução do programa

## 5. Discussão

A implementação do método Newton-Raphson recursivo para cálculo da raiz quadrada apresentou resultados conforme o esperado. Como podemos observar nas capturas de tela, ao calcular a raiz quadrada de 30 com 190 iterações, o programa retornou o valor 5, que é a parte inteira da raiz quadrada de 30 (aproximadamente 5,477).

É importante notar que a implementação em Assembly MIPS trabalha apenas com números inteiros, o que limita a precisão do resultado. No entanto, com um número suficiente de iterações, o método converge para o valor inteiro mais próximo da raiz quadrada exata.

A recursividade foi implementada com sucesso, como podemos observar na captura da área de pilha. Cada chamada recursiva armazena seu contexto (valor de x, valor de i, endereço de retorno e resultado parcial) na pilha, permitindo que o cálculo seja realizado corretamente com a profundidade necessária.

## 6. Conclusão

Neste trabalho, implementamos com sucesso o método Newton-Raphson recursivo para cálculo da raiz quadrada em Assembly MIPS. A implementação demonstrou o funcionamento correto do algoritmo, utilizando conceitos fundamentais de organização e arquitetura de processadores, como o uso de funções, macros, gerenciamento de pilha e recursividade.

Os resultados obtidos mostram que a implementação em Assembly MIPS é capaz de calcular aproximações das raízes quadradas conforme o esperado, com as limitações próprias da aritmética inteira.

A experiência adquirida com este trabalho permitiu aprofundar os conhecimentos sobre a programação em linguagem de máquina, a ISA do MIPS e o mapeamento de algoritmos de alto nível para assembly.