

# Função de Raiz Quadrada usando o Método Newton-Raphson Recursivo

Organização e Arquitetura de Processadores

Eduarda Patricio (23111258-2) • Giovanna Castro (23111285-2)  
Naiumy dos Reis (23111738-3) • Yasmin Aguirre (23111329-1)

## 1. Algoritmo em alto nível

O programa abaixo contém a implementação do método Newton-Raphson de forma recursiva na linguagem Java.

```
import java.util.*;
public class altoNivel {
    // Método para calcular a raiz quadrada usando o método Newton-Raphson recursivamente
    public static int sqrt_nr(int x, int i) {
        int result = 0;
        // Caso base: se i for 0, retorna 1 e faz o primeiro cálculo do método
        if (i == 0) {
            result = 1;
        }
        // Caso contrário, faz o cálculo i-1 do método recursivamente
        if (i > 0) {
            result = (sqrt_nr(x, i - 1) + (x / sqrt_nr(x, i - 1))) / 2;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int i = 0;
        int x = 0;
        System.out.println("\nPrograma de Raiz Quadrada - Newton-Raphson\r\n" +
            "Desenvolvedoras: Eduarda Patricio, Giovanna Castro, Naiumy dos Reis e Yasmin Aguirre");

        while (true) {
            System.out.println("\nDigite os parâmetros x e i para calcular sqrt_nr (x, i) ou -1 para abortar a execução");
            System.out.println("Digite o parâmetro x:");
            x = in.nextInt();

            if (x < 0) {
                break;
            }
            System.out.println("Digite o parâmetro i:");
            i = in.nextInt();
        }
    }
}
```

```

    if (i < 0) {
        break;
    }
    // Chama o método sqrt_nr e exibe o resultado
    int a = sqrt_nr(x, i);
    System.out.println("sqrt(" + x + ", " + i + ") = " + a);
}
in.close();
}
}

```

## 2. Algoritmo em baixo nível

O programa abaixo contém a implementação do método Newton-Raphson de forma recursiva em Assembly do MIPS.

<b>.macro prtStr(%string)</b>	# Macro para printar strings
addi \$sp, \$sp, -8	# Abre dois espacos na pilha
sw \$v0, 0(\$sp)	# Guarda o valor que esta no registrador \$v0 no topo da pilha
sw \$a0, 4(\$sp)	# Guarda o valor que esta no registrador \$a0 na segunda posicao da pilha
la \$a0, %string	# Carrega o endereco da string em \$a0
li \$v0, 4	# Carrega a instrucao 4 (print_string) em \$v0
syscall	# Chama o sistema para executar a instrucao
lw \$a0, 4(\$sp)	# Recupera o valor que esta na segunda posicao da pilha de volta ao \$a0
lw \$v0, 0(\$sp)	# Recupera o valor que esta no topo da pilha de volta ao \$v0
addi \$sp, \$sp, 8	# Apaga o espaco aberto na pilha
<b>.end_macro</b>	

<b>.macro leInt(%int)</b>	# Macro para ler um valor inteiro
addi \$sp, \$sp, -4	# Abre espaco na pilha para um int
sw \$v0, 0(\$sp)	# Coloca o que esta no registrador \$v0 no topo da pilha
li \$v0, 5	# Carrega a instrucao 5 (read_int) em \$v0
syscall	# Chama o sistema para executar a instrucao
move %int, \$v0	# Move o valor de \$v0 para o destino %int
lw \$v0, 0(\$sp)	# Recupera o valor que esta no topo da pilha de volta ao \$v0
addi \$sp, \$sp, 4	# Apaga o espaco aberto na pilha
<b>.end_macro</b>	

<b>.macro prtInt(%inteiro)</b>	# Macro para printar um valor inteiro
addi \$sp, \$sp, -8	# Abre dois espacos na pilha
sw \$v0, 0(\$sp)	# Guarda o valor que esta no registrador \$v0 no topo da pilha
sw \$a0, 4(\$sp)	# Guarda o valor que esta no registrador \$a0 na segunda posicao da pilha
lw \$a0, %inteiro	# Carrega o endereco do inteiro em \$a0
li \$v0, 1	# Carrega a instrucao 1 (print_int) em \$v0
syscall	# Chama o sistema para executar a instrucao
lw \$a0, 4(\$sp)	# Recupera o valor que esta na segunda posicao da pilha de volta ao \$a0
lw \$v0, 0(\$sp)	# Recupera o valor que esta no topo da pilha de volta ao \$v0

```

addi    $sp, $sp, 8          # Apaga o espaco aberto na pilha
.end_macro

.macro callRaiz()            # Macro para chamar a funcao raiz
addi    $sp, $sp, -12        # Abre tres espacos na pilha
sw      $a0, 8($sp)          # Guarda o valor que esta no registrador $a0 (x) na terceira posicao da pilha
sw      $a1, 4($sp)          # Guarda o valor que esta no registrador $a1 (i) na segunda posicao da pilha
sw      $ra, 0($sp)          # Guarda o valor que esta no registrador $r0 no topo da pilha
jal     raiz                 # Vai para raiz
lw      $ra, 0($sp)          # Recupera o valor que esta no topo da pilha de volta ao $ra
lw      $a1, 4($sp)          # Recupera o valor que esta na segunda posicao da pilha de volta ao $a1
lw      $a0, 8($sp)          # Recupera o valor que esta na terceira posicao da pilha de volta ao $a0
addi    $sp, $sp, 12         # Apaga o espaco aberto na pilha
.end_macro

.macro resultado()          # Macro para printar o resultado
sw      $v0, result          # Salva em result o valor lido em $v0
prtStr(strRaizI)             # Print("sqrt("
prtInt(x)                    # Print("sqrt(" + x
prtStr(strVirg)              # Print("sqrt(" + x + ", "
prtInt(i)                    # Print("sqrt(" + x + ", " + i
prtStr(strRaizF)             # Print("sqrt(" + x + ", " + i + ") = "
prtInt(result)               # Print("sqrt(" + x + ", " + i + ") = " + result);
.end_macro

.macro exit()               # Macro para sair do programa
li      $v0, 10              # Carrega a instrucao 10 (exit) em $v0
syscall                                # Chama o sistema para executar a instrucao
.end_macro

.data
x: .space 4
i: .space 4
result: .space 4
strProgm: .asciiz "\nPrograma de Raiz - Newton-Raphson\nDesenvolvedoras: Eduarda Patricio, Giovanna Castro,
Naiumy dos Reis e Yasmin Aguirre"
strDigite: .asciiz "\n\nDigite os parametros x e i para calcular sqrt_nr(x, i) ou -1 para abortar a execucao\nDigite o
parametro x: "
strPrml: .asciiz "Digite o parametro i: "
strRaizI: .asciiz "sqrt("
strVirg: .asciiz ", "
strRaizF: .asciiz ") = "

.text
.globl main
main:
prtStr(strProgm)            # Usa o macro prtStr para printar a string de strProgm

```

<b>loop:</b>	
<b>prtStr(strDigite)</b>	# Usa o macro prtStr para printar a string de strDigite
<b>leInt(\$a0)</b>	# Usa o macro leInt para ler o valor inteiro de \$a0
<b>sw           \$a0, x</b>	# Salva em x o valor lido em \$a0
<b>blt           \$a0, \$zero, fim</b>	# Verifica se a entrada e negativa (x < 0)
<b>prtStr(strPrml)</b>	# Usa o macro prtStr para printar a string de strPrml
<b>leInt(\$a1)</b>	# Usa o macro leInt para ler o valor inteiro de \$a1
<b>sw           \$a1, i</b>	# Salva em i o valor lido em \$a1
<b>blt           \$a1, \$zero, fim</b>	# Verifica se a entrada e negativa (i < 0)
<b>callRaiz()</b>	# Chama o macro que faz a funcao de raiz quadrada
<b>resultado()</b>	# Chama o macro que imprime o resultado
<b>jal loop</b>	# Repete o programa enquanto a entrada for positiva
<b>fim:</b>	
<b>exit()</b>	# Chama o macro que encerra o programa
 <b>raiz:</b>	
<b>lw           \$a1, 4(\$sp)</b>	# Carrega o valor de \$a1 em i
<b>lw           \$a0, 8(\$sp)</b>	# Carrega o valor de \$a0 em x
<b>bgt           \$a1, \$zero, recurs</b>	# Se (i > 0), vai para "recurs"
<b>li           \$v0, 1</b>	# Define o resultado como 1 se (i == 0)
<b>jr           \$ra</b>	# Volta para o macro callRaiz()
<b>recurs:</b>	
<b>addiu       \$a1, \$a1, -1</b>	# Decrementa i (i--)
<b>callRaiz()</b>	# Chama o macro que faz a funcao de raiz quadrada de novo
<b>addiu       \$a1, \$a1, 1</b>	# Restaura o valor de i (i++)
<b>div          \$t0, \$a0, \$v0</b>	# $x / \text{sqrt\_nr}(x, i - 1)$
<b>add          \$v0, \$v0, \$t0</b>	# $\text{sqrt\_nr}(x, i - 1) + (x / \text{sqrt\_nr}(x, i - 1))$
<b>srl          \$v0, \$v0, 1</b>	# $(\text{sqrt\_nr}(x, i - 1) + (x / \text{sqrt\_nr}(x, i - 1))) / 2$
<b>jr          \$ra</b>	# Volta para raiz

3. Capturas de tela do MARS

Screenshots do simulador MARS:

a) Área de código compilada:

Text Segment

Bkpt	Address	Code	Basic	Source
	4194716	0x23bdfbf9	addi \$29,\$29,-8	<2> addi \$sp,\$sp,-8 # Abre dois espacos na pilha
	4194720	0xafaf200004	sw \$2,0(\$29)	<3> sw \$v0, 0(\$sp) # Guarda o valor que esta no registrador \$v0 no topo da
	4194724	0xafaf400004	sw \$4,4(\$29)	<4> sw \$a0, 4(\$sp) # Guarda o valor que esta no registrador \$a0 na segunda
	4194728	0x3c011001	lui \$1,4097	<5> la \$a0, strRaizF # Carrega o endereco da string em \$a0
	4194732	0x34240112	ori \$4,\$1,274	
	4194736	0x24020004	addiu \$2,\$0,4	<6> li \$v0, 4 # Carrega a instrucao 4 (print string) em \$v0
	4194740	0x0000000c	syscall	<7> syscall # Chama o sistema para executar a instrucao
	4194744	0x8fa40004	lw \$4,4(\$29)	<8> lw \$a0, 4(\$sp) # Recupera o valor que esta na segunda posicao da pilha
	4194748	0x8fa20000	lw \$2,0(\$29)	<9> lw \$v0, 0(\$sp) # Recupera o valor que esta no topo da pilha de volta ad
	4194752	0x23bdfbf9	addi \$29,\$29,8	<10> addi \$sp,\$sp,8 # Apaga o espaco aberto na pilha
	4194756	0x23bdfbf9	addi \$29,\$29,-8	<24> addi \$sp,\$sp,-8 # Abre dois espacos na pilha
	4194760	0xafaf200004	sw \$2,0(\$29)	<25> sw \$v0, 0(\$sp) # Guarda o valor que esta no registrador \$v0 no topo da
	4194764	0xafaf400004	sw \$4,4(\$29)	<26> sw \$a0, 4(\$sp) # Guarda o valor que esta no registrador \$a0 na segunda
	4194768	0x3c011001	lui \$1,4097	<27> lw \$a0, result # Carrega o endereco do inteiro em \$a0
	4194772	0x8c240008	lw \$4,8(\$1)	
	4194776	0x24020001	addiu \$2,\$0,1	<28> li \$v0, 1 # Carrega a instrucao 1 (print int) em \$v0
	4194780	0x0000000c	syscall	<29> syscall # Chama o sistema para executar a instrucao
	4194784	0x8fa40004	lw \$4,4(\$29)	<30> lw \$a0, 4(\$sp) # Recupera o valor que esta na segunda posicao da pilha
	4194788	0x8fa20000	lw \$2,0(\$29)	<31> lw \$v0, 0(\$sp) # Recupera o valor que esta no topo da pilha de volta ad
	4194792	0x23bdfbf9	addi \$29,\$29,8	<32> addi \$sp,\$sp,8 # Apaga o espaco aberto na pilha
	4194796	0x0c10000a	jal 4194344	88: jal loop # Repete o programa enquanto a entrada for posit
	4194800	0x2402000a	addiu \$2,\$0,10	90: <58> li \$v0, 10 # Carrega a instrucao 10 (exit) em \$v0
	4194804	0x0000000c	syscall	<59> syscall # Chama o sistema para executar a instrucao

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0
\$at	1	1
\$v0	2	10
\$v1	3	0
\$a0	4	-1
\$a1	5	8
\$a2	6	0
\$a3	7	0
\$t0	8	22
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194800

Mars Messages Run I/O

Clear -- program is finished running --

b) Estado dos registradores ao final da execução:

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$v0	2	10
\$v1	3	0
\$a0	4	-1
\$a1	5	8
\$a2	6	0
\$a3	7	0
\$t0	8	22
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$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	4194800
pc		4194808
hi		16
lo		22

c) Área de pilha utilizada para a recursividade:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7ffffefe0	500	4194532	7	500	0	22	22	0
0x7fffff000	0	0	0	0	0	0	0	0
0x7fffff020	0	0	0	0	0	0	0	0
0x7fffff040	0	0	0	0	0	0	0	0
0x7fffff060	0	0	0	0	0	0	0	0
0x7fffff080	0	0	0	0	0	0	0	0
0x7fffff0a0	0	0	0	0	0	0	0	0
0x7fffff0c0	0	0	0	0	0	0	0	0
0x7fffff0e0	0	0	0	0	0	0	0	0
0x7fffff100	0	0	0	0	0	0	0	0
0x7fffff120	0	0	0	0	0	0	0	0
0x7fffff140	0	0	0	0	0	0	0	0
0x7fffff160	0	0	0	0	0	0	0	0
0x7fffff180	0	0	0	0	0	0	0	0
0x7fffff1a0	0	0	0	0	0	0	0	0
0x7fffff1c0	0	0	0	0	0	0	0	0

d) Exemplo de execução do programa:

Mars MessagesRun I/O

Programa de Raiz - Newton-Raphson

Desenvolvedoras: Eduarda Patricio, Giovanna Castro, Naiumy dos Reis e Yasmin Aguirre

Digite os parametros x e i para calcular sqrt\_nr(x, i) ou -1 para abortar a execucao

Digite o parametro x: 500

Digite o parametro i: 8

sqrt(500, 8) = 22

Digite os parametros x e i para calcular sqrt\_nr(x, i) ou -1 para abortar a execucao

Digite o parametro x: -1

-- program is finished running --

Clear