Trabalho - Assembly do Processador MIPS (T1) - Organização e Arquitetura de Processadores (OAP)

Caio Madeira (24280006)

April 24, 2025

1 Introdução

Este trabalho apresenta a resolução do problema referente a implementação do algorítimo relacionado ao método numérico de Netwon-Raphson, de forma recursiva. Para a descrição em alto nível foi escolhida a linguagem C, devido a familiaridade com ela. Ademais, o trabalho procura satisfazer os pontos solicitados para a solução na especificação técnica.

2 Implementação

2.1 Implementação em Alto Nível (C)

Para a descrição do algorítimo em alto nível foi usada a linguagem de programação C.

Listing 1: Implementação em C

```c
/*
Square root with Recursive Newton-Raphson Method by Caio Madeira
*/

#include<stdio.h>
#include<math.h>

int sqrt_nr(int x, int i);

int sqrt_nr(int x, int i) {
    if (i == 0) return 1; // 1,^i=0. Caso base necessario, caso contrario:
        sinal SIGFPE (Floating Point Exception)
    return (sqrt_nr(x, i-1)+(x/sqrt_nr(x,i-1)))/2;
}

// Fun  o auxiliar apenas pra fins de valida  o do resultado usando a math
    .h de C
int valide_result(int x, int r) {
    if (sqrt(x) == r)
        return 1;
    return 0;
}

int main(void)
{
    // Caso bom
    int x = 100;
    int i = 5;

    // // caso ruim
    // int x = 100;
```

```c
    // int i = 3;

    int r = sqrt_nr(x, i);

    // O resultado deve ser 10, pois raiz_quadrada de 100 = 10
    printf("sqrt_nr(%d, %d) = %d\n", x, i, r);
    if (valide_result(x, r) == 1)
        printf("resultado convergiu adequadamente.\n");
    else
        printf("resultado NAO convergiu adequadamente.\n");

    return 0;
}
```

2.2 Implementação em MIPS Assembly

Para a implementação em MIPS Assembly, foi seguido estritamente as especificações, tendo as duas funções e duas macros. Além do mais, as dicas e recomendações de mensagens foram seguidas também.

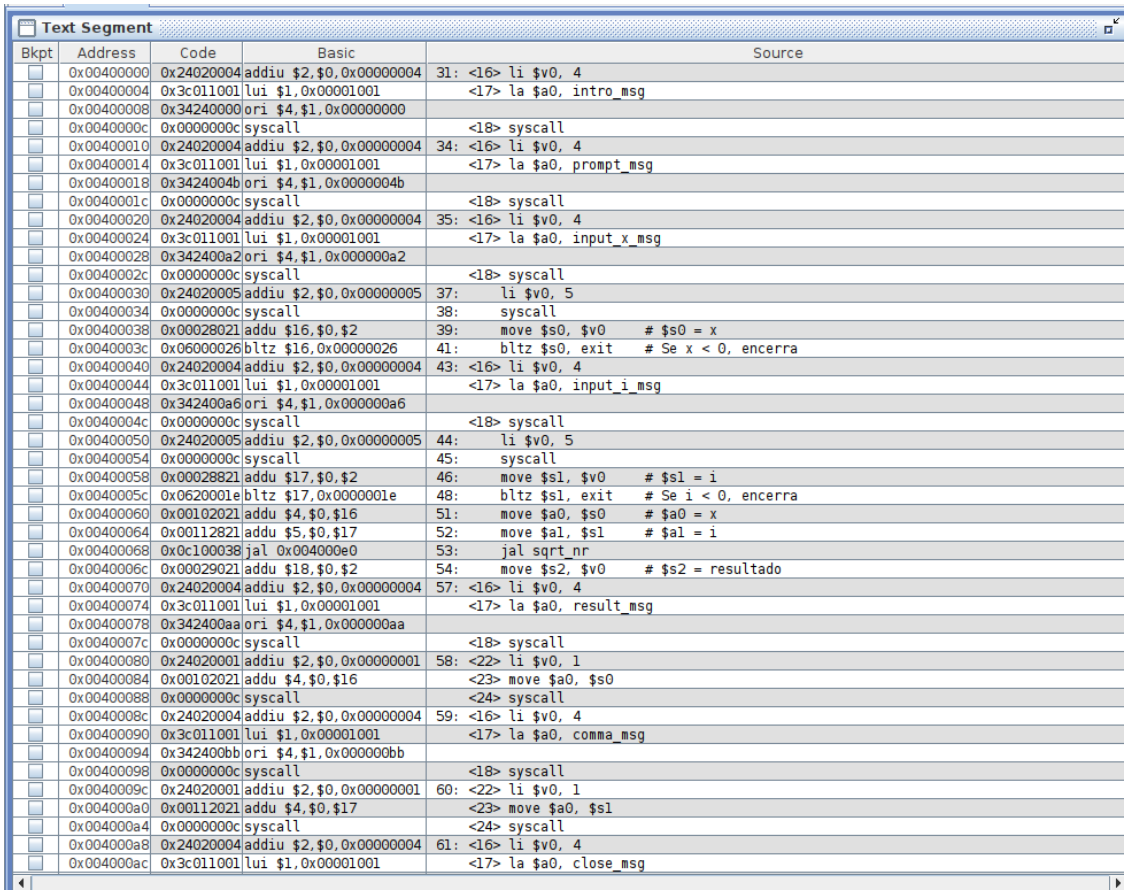
Listing 2: Implementação em MIPS Assembly

```
# Programa MIPS - Newton-Raphson recursivo para raiz quadrada
# Caio Madeira | 24280006

.data
intro_msg:      .asciiz "\nPrograma de Raiz Quadrada     Newton-Raphson\
    nDesenvolvedores: Caio Madeira\n"
prompt_msg:     .asciiz "\Digite os parametros x e i para calcular sqrt_nr(x,
    i) ou -1 para abortar a execucao:\n"
input_x_msg:    .asciiz "x: "
input_i_msg:    .asciiz "i: "
result_msg:     .asciiz "Resultado: sqrt("
comma_msg:      .asciiz ", "
close_msg:      .asciiz ") = "
space_mg:       .asciiz "\n"


# ------------------------------
# Macros
# ------------------------------
.macro print_str (%str)
    li $v0, 4
    la $a0, %str
    syscall
.end_macro

.macro print_int (%reg)
    li $v0, 1
    move $a0, %reg
    syscall
.end_macro

.text
.globl main

main:
    print_str(intro_msg)

loop:
    print_str(prompt_msg)
    print_str(input_x_msg)
```

```
    li $v0, 5
    syscall
    move $s0, $v0     # $s0 = x
    bltz $s0, exit    # se x < 0, encerra

    print_str(input_i_msg)
    li $v0, 5
    syscall
    move $s1, $v0     # $s1 = i
    bltz $s1, exit    # se i < 0, encerra

    # chama de sqrt_nr(x, i)
    move $a0, $s0     # $a0 = x
    move $a1, $s1     # $a1 = i
    jal sqrt_nr
    move $s2, $v0     # $s2 = resultado

    # print do resultado
    print_str(result_msg)
    print_int($s0)
    print_str(comma_msg)
    print_int($s1)
    print_str(close_msg)
    print_int($s2)
    print_str(space_mg)
    j loop

exit:
    li $v0, 10
    syscall

# -------------------------------
# Fun   o recursiva sqrt_nr(x, i)
# -------------------------------
sqrt_nr:
    beqz $a1, base_case  # se i == 0, retorna 1

    # salvando contexto de ra, a0, a1
    addi $sp, $sp, -12
    sw $ra, 0($sp)
    sw $a0, 4($sp)
    sw $a1, 8($sp)

    # recursividade sqrt_nr(x, i - 1)
    addi $a1, $a1, -1
    jal sqrt_nr

    # recupera o contexto
    lw $a0, 4($sp)    # x
    lw $t0, 8($sp)    # i (opcional, n o usado depois)
    move $t1, $v0     # t1 = r = sqrt_nr(x, i-1)

    # Calcula x / r
    move $t2, $a0     # x
    move $t3, $t1     # r
    div $t2, $t3
    mflo $t4          # t4 = x / r

    add $t5, $t1, $t4 # t5 = r + (x / r)

    # divide por 2 com sra (shift)
```

```
    sra $v0, $t5, 1    # v0 = (r + x/r) / 2

    # restaura registrador de retorno e desempilha
    lw $ra, 0($sp)
    addi $sp, $sp, 12
    jr $ra

base_case:
    li $v0, 1
    jr $ra
```

3 Evidências



Text Segment

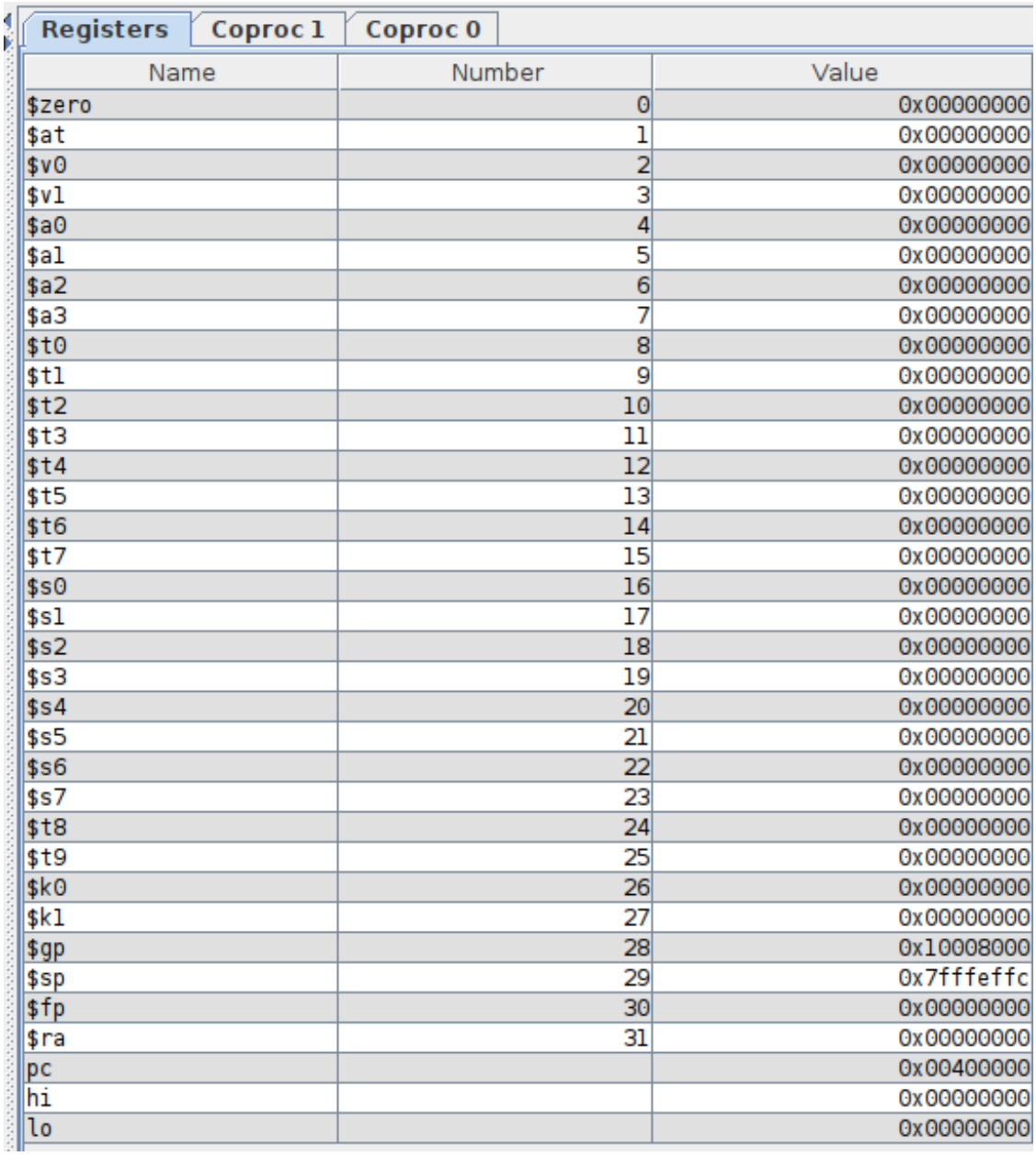
Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu $2,$0,0x00000004	31: <16> li $v0, 4
	0x00400004	0x3c011001	lui $1,0x00001001	<17> la $a0, intro_msg
	0x00400008	0x34240000	ori $4,$1,0x00000000	
	0x0040000c	0x0000000c	syscall	<18> syscall
	0x00400010	0x24020004	addiu $2,$0,0x00000004	34: <16> li $v0, 4
	0x00400014	0x3c011001	lui $1,0x00001001	<17> la $a0, prompt_msg
	0x00400018	0x3424004b	ori $4,$1,0x0000004b	
	0x0040001c	0x0000000c	syscall	<18> syscall
	0x00400020	0x24020004	addiu $2,$0,0x00000004	35: <16> li $v0, 4
	0x00400024	0x3c011001	lui $1,0x00001001	<17> la $a0, input_x_msg
	0x00400028	0x342400a2	ori $4,$1,0x000000a2	
	0x0040002c	0x0000000c	syscall	<18> syscall
	0x00400030	0x24020005	addiu $2,$0,0x00000005	37: li $v0, 5
	0x00400034	0x0000000c	syscall	38: syscall
	0x00400038	0x00028021	addu $16,$0,$2	39: move $s0, $v0 # $s0 = x
	0x0040003c	0x06000026	bltz $16,0x00000026	41: bltz $s0, exit # Se x < 0, encerra
	0x00400040	0x24020004	addiu $2,$0,0x00000004	43: <16> li $v0, 4
	0x00400044	0x3c011001	lui $1,0x00001001	<17> la $a0, input_i_msg
	0x00400048	0x342400a6	ori $4,$1,0x000000a6	
	0x0040004c	0x0000000c	syscall	<18> syscall
	0x00400050	0x24020005	addiu $2,$0,0x00000005	44: li $v0, 5
	0x00400054	0x0000000c	syscall	45: syscall
	0x00400058	0x00028821	addu $17,$0,$2	46: move $s1, $v0 # $s1 = i
	0x0040005c	0x0620001e	bltz $17,0x0000001e	48: bltz $s1, exit # Se i < 0, encerra
	0x00400060	0x00102021	addu $4,$0,$16	51: move $a0, $s0 # $a0 = x
	0x00400064	0x00112821	addu $5,$0,$17	52: move $a1, $s1 # $a1 = i
	0x00400068	0x0c100038	jal 0x004000e0	53: jal sqrt_nr
	0x0040006c	0x00029021	addu $18,$0,$2	54: move $s2, $v0 # $s2 = resultado
	0x00400070	0x24020004	addiu $2,$0,0x00000004	57: <16> li $v0, 4
	0x00400074	0x3c011001	lui $1,0x00001001	<17> la $a0, result_msg
	0x00400078	0x342400aa	ori $4,$1,0x000000aa	
	0x0040007c	0x0000000c	syscall	<18> syscall
	0x00400080	0x24020001	addiu $2,$0,0x00000001	58: <22> li $v0, 1
	0x00400084	0x00102021	addu $4,$0,$16	<23> move $a0, $s0
	0x00400088	0x0000000c	syscall	<24> syscall
	0x0040008c	0x24020004	addiu $2,$0,0x00000004	59: <16> li $v0, 4
	0x00400090	0x3c011001	lui $1,0x00001001	<17> la $a0, comma_msg
	0x00400094	0x342400bb	ori $4,$1,0x000000bb	
	0x00400098	0x0000000c	syscall	<18> syscall
	0x0040009c	0x24020001	addiu $2,$0,0x00000001	60: <22> li $v0, 1
	0x004000a0	0x00112021	addu $4,$0,$17	<23> move $a0, $s1
	0x004000a4	0x0000000c	syscall	<24> syscall
	0x004000a8	0x24020004	addiu $2,$0,0x00000004	61: <16> li $v0, 4
	0x004000ac	0x3c011001	lui $1,0x00001001	<17> la $a0, close_msg

Figure 1: A área de código montada.

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400084	0x00102021	addu $4,$0,$16	<23> move $a0, $s0
	0x00400088	0x0000000c	syscall	<24> syscall
	0x0040008c	0x24020004	addiu $2,$0,0x00000004	59: <16> li $v0, 4
	0x00400090	0x3c011001	lui $1,0x00001001	<17> la $a0, comma_msg
	0x00400094	0x342400bb	ori $4,$1,0x000000bb	
	0x00400098	0x0000000c	syscall	<18> syscall
	0x0040009c	0x24020001	addiu $2,$0,0x00000001	60: <22> li $v0, 1
	0x004000a0	0x00112021	addu $4,$0,$17	<23> move $a0, $s1
	0x004000a4	0x0000000c	syscall	<24> syscall
	0x004000a8	0x24020004	addiu $2,$0,0x00000004	61: <16> li $v0, 4
	0x004000ac	0x3c011001	lui $1,0x00001001	<17> la $a0, close_msg
	0x004000b0	0x342400be	ori $4,$1,0x000000be	
	0x004000b4	0x0000000c	syscall	<18> syscall
	0x004000b8	0x24020001	addiu $2,$0,0x00000001	62: <22> li $v0, 1
	0x004000bc	0x00122021	addu $4,$0,$18	<23> move $a0, $s2
	0x004000c0	0x0000000c	syscall	<24> syscall
	0x004000c4	0x24020004	addiu $2,$0,0x00000004	63: <16> li $v0, 4
	0x004000c8	0x3c011001	lui $1,0x00001001	<17> la $a0, space_mg
	0x004000cc	0x342400c3	ori $4,$1,0x000000c3	
	0x004000d0	0x0000000c	syscall	<18> syscall
	0x004000d4	0x08100004	j 0x00400010	64: j loop
	0x004000d8	0x2402000a	addiu $2,$0,0x0000000a	67: li $v0, 10
	0x004000dc	0x0000000c	syscall	68: syscall
	0x004000e0	0x10a00012	beq $5,$0,0x00000012	74: beqz $a1, base_case # Se i == 0, retorna 1
	0x004000e4	0x23bdfff4	addi $29,$29,0xffff...	77: addi $sp, $sp, -12
	0x004000e8	0xafbf0000	sw $31,0x00000000($29)	78: sw $ra, 0($sp)
	0x004000ec	0xafa40004	sw $4,0x00000004($29)	79: sw $a0, 4($sp)
	0x004000f0	0xafa50008	sw $5,0x00000008($29)	80: sw $a1, 8($sp)
	0x004000f4	0x20a5ffff	addi $5,$5,0xffffffff	83: addi $a1, $a1, -1
	0x004000f8	0x0c100038	jal 0x004000e0	84: jal sqrt_nr
	0x004000fc	0x8fa40004	lw $4,0x00000004($29)	87: lw $a0, 4($sp) # x
	0x00400100	0x8fa80008	lw $8,0x00000008($29)	88: lw $t0, 8($sp) # i (opcional, não usado depois)
	0x00400104	0x00024821	addu $9,$0,$2	89: move $t1, $v0 # t1 = r = sqrt_nr(x, i-1)
	0x00400108	0x00045021	addu $10,$0,$4	92: move $t2, $a0 # x
	0x0040010c	0x00095821	addu $11,$0,$9	93: move $t3, $t1 # r
	0x00400110	0x014b001a	div $10,$11	94: div $t2, $t3
	0x00400114	0x00006012	mflo $12	95: mflo $t4 # t4 = x / r
	0x00400118	0x012c6820	add $13,$9,$12	97: add $t5, $t1, $t4 # t5 = r + (x / r)
	0x0040011c	0x000d1043	sra $2,$13,0x00000001	100: sra $v0, $t5, 1 # v0 = (r + x/r) / 2
	0x00400120	0x8fbf0000	lw $31,0x00000000($29)	103: lw $ra, 0($sp)
	0x00400124	0x23bd000c	addi $29,$29,0x0000...	104: addi $sp, $sp, 12
	0x00400128	0x03e00008	jr $31	105: jr $ra
	0x0040012c	0x24020001	addiu $2,$0,0x00000001	108: li $v0, 1
	0x00400130	0x03e00008	jr $31	109: jr $ra

Figure 2: A área de código montada.



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Figure 3: O estado dos registradores ao término de uma execução.

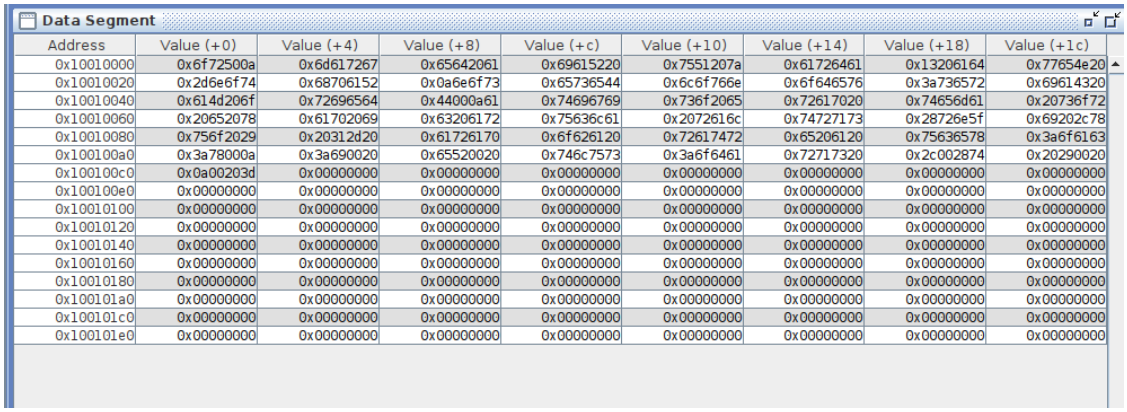


Figure 4: A área de pilha utilizada para a recursividade.

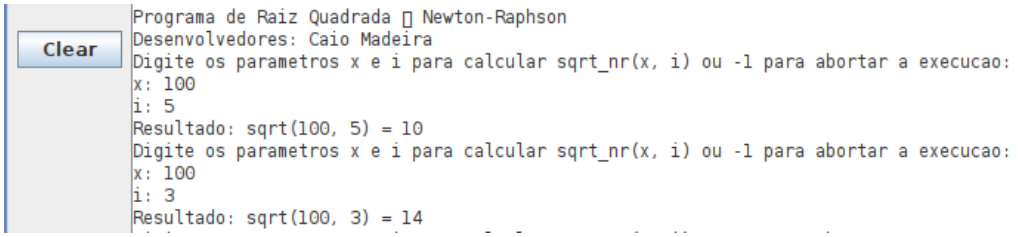


Figure 5: Um exemplo de execução do programa.