Relatório de atividade

Trabalho Prático

Organização e Arquitetura de Processadores

Componentes: Eduarda Fuchs (23106283), Fernanda Franceschini (23102302), Fernando Neto (23102305) e Luana Sostisso (23106539).

26 de abril de 2024.

1. INTRODUÇÃO

Este trabalho faz parte da disciplina de Organização e Arquitetura de Processadores (OAP). O presente relatório descreve a especificação técnica e a implementação de uma versão recursiva do método numérico de Newton-Raphson, com o propósito específico de calcular a raiz quadrada de um número inteiro e positivo, denominado x.

A abordagem recursiva desta técnica, em contraste com a implementação iterativa convencional, utiliza chamadas de função autônoma com novos parâmetros, reduzindo a complexidade do código e oferecendo uma solução conceitualmente simples para o cálculo da raiz quadrada.

Neste contexto, este trabalho visa explorar a implementação recursiva do método de Newton-Raphson, destacando sua aplicabilidade na resolução de problemas comuns de cálculo numérico. A seguir, demonstraremos a regra de recursividade que organiza a implementação do cálculo da raiz quadrada utilizando este método, seguida pela descrição detalhada da implementação e suas considerações técnicas.

2. DESCRIÇÃO EM JAVA

O algoritmo em Java teve uma implementação relativamente simples, onde foi utilizada apenas uma classe chamada ”App”. Na classe ”App”, foi implementada a função ”main”, a função principal para o funcionamento do programa, e a função ”sqr_int”, que é responsável por toda a lógica do programa.

Na função main programa solicita ao usuário os parâmetros x e i, onde x é o número para o qual queremos calcular a raiz quadrada e i é o número de iterações do método. Ele repete esse processo até que o usuário digite um número negativo ou interrompa a execução.

A cada novos valores de entrada, é realizada uma chamada para o método ”sqr_int”. Ele recebe dois valores como parâmetros, que representam o número a ser calculado e o número de iterações. A função vai chamar a si própria repetidas vezes até i decrementar até 0, calculando a raiz quadrada.

App.java

```java
import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        System.out.println("Programa Raiz Quadrada");
        System.out.println("=======================");
        System.out.println("Componentes: <Luana Sostisso, Eduarda Fuchs, Fernanda Franschescini, Fernando Neto>");
        System.out.println("=======================");
        int x = 0;
        int i = 0;
        while (x >= 0 && i>=0) {
            System.out.println("Digite os parametros x e i para calcular A(x, i) ou -1 para abortar a execuçao");
            x = in.nextInt();
            if(x<0){
                break;
            }
            i = in.nextInt();
            if(i<0){
                break;
            }
            int resp = sqrt_nr(x,i);
            System.out.printf("A(%d, %d) = %d\n", x,i,resp);
        }
        System.out.println("Valor negativo digitado! Programa encerrado!");
    }


    public static int sqrt_nr(int x, int i){
        if(i==0){
            return 1;
        }
        else if (i>0) {
            return (sqrt_nr(x, i-1) + (x/sqrt_nr(x, i-1))) /2;
        }
        return -1;
    }
}
```

3. IMPLEMENTAÇÃO DO CÓDIGO EM ASSEMBLY MIPS

A implementação do algoritmo em Assembly foi mais complexa do que em alto nível, como o esperado. Para facilitar a compreensão, será utilizada a representação por imagens, além de uma breve explicação do funcionamento da função sqrt_nr.

O programa inicia com uma seção .data, que define constantes que representam mensagens de texto que serão exibidas ao usuário durante a execução do programa.

Na seção *.text*, o programa define o ponto de entrada main. Aqui, ele exibe as mensagens iniciais e inicia um loop que solicita ao usuário que insira os parâmetros necessários para calcular a raiz quadrada. Os parâmetros são o número do qual a raiz quadrada será calculada e o número de iterações desejadas para a aproximação. O programa então chama a função *sqrt_nr* para realizar o cálculo da raiz quadrada com base nos parâmetros fornecidos.

A função *sqrt_nr* é responsável por aplicar o método Newton-Raphson para encontrar aproximações às raízes quadradas. Recebe os parâmetros x (o número cuja raiz quadrada será calculada) e i (o número de iterações). Nesta função, é verificado se o número de iterações é maior que zero. Nesse caso, o algoritmo de Newton-Raphson é aplicado iterativamente até o número de iterações seja alcançado. Se o número de iterações for menor ou igual a 0, a função retornará -1.

Ao final de cada iteração, os resultados do cálculo são exibidos ao usuário. O programa continua solicitando novos parâmetros de entrada até que o usuário decida encerrar a execução do programa.

3.1 UTILIZAÇÃO DA PILHA

Ao iniciar a função *sqrt_nr*, a instrução addi $sp,$sp,-16 aloca espaço na pilha para armazenar os valores salvos nos registradores $ra, $s0, $s1 e $s2. Este procedimento é importante para preservar o conteúdo original desses registradores durante a execução da função, pois eles podem ser utilizados para outros fins dentro da função. Posteriormente, as instruções sw $ra, 12($sp), sw $s0, 8($sp), sw $s1, 4($sp) e sw $s2, 0($sp) escrevem o conteúdo desses registradores em o novo endereço específico na pilha alocada.

Posteriormente, as instruções sw $ra, 12($sp), sw $s0, 8($sp), sw $s1, 4($sp), e sw $s2, 0($sp) gravam o conteúdo desses registradores em endereços específicos na pilha recém-alocada.
Ao final da função, as instruções lw $ra, 12($sp), lw $s0, 8($sp), lw $s1, 4($sp), e lw $s2, 0($sp) recuperam os valores salvos dos registradores antes de retornar do *jal sqrt_nr.*

3.2 CÓDIGO ASSEMBLY MIPS

Seção de Dados

```
.data

ack: .asciiz   "Programa de Raiz Quadrada - Newton-Raphson"

nomes: .asciiz "Desenvolvedores: <Luana Sostisso, Eduarda Fuchs, Fernanda Franschescini, Fernando Neto>"

espaco: .asciiz "============================================="

mnString: .asciiz "Digite os parametros x e i para calcular sqrt_nr(x, i) ou -1 para abortar a execuçao \n"

newline: .asciiz "\n"

resposta1: .asciiz "sqrt("
resposta2: .asciiz ", "
resposta3: .asciiz ") = "


impFim: .asciiz "Valor negativo digitado! Programa encerrado!"
```

Label Main

```
main:
        li $v0, 4
        la $a0, ack
        syscall
        la $a0, newline
        syscall
        la $a0, espaco
        syscall
        la $a0, newline
        syscall
        li $v0, 4
        la $a0, nomes
        syscall
        la $a0, newline
        syscall
        li $v0, 4
        la $a0, espaco
        syscall
        la $a0, newline
        syscall
        move $t0, $zero  #x
        move $t1, $zero  #i
        li $t5, 1
```

Label Loop

```
loop:
        li $v0, 4
        la $a0, mnString
        syscall

        li $v0, 5
        syscall

        move $t0, $v0  #x
        blt $t0, $zero, end_loop
        move    $s1, $v0

        li $v0, 5
        syscall

        move $t1, $v0 #i

        blt $t1, $zero, end_loop

        move    $s2, $v0

        move    $a0, $s1
        move    $a1, $s2

        jal     sqrt_nr
        move    $s0, $v0

        li      $v0, 4
        la      $a0, resposta1
        syscall
        move $a0, $t0
        li      $v0, 1
        syscall
        li      $v0, 4
        la      $a0, resposta2
        syscall
        move $a0, $t1
        li      $v0, 1
        syscall
        li      $v0, 4
        la      $a0, resposta3
        syscall
        li      $v0, 1
        move    $a0, $s0
        syscall

        li      $v0, 4
        la $a0, newline
        syscall

        j loop
```

Label sqr_nr

```
sqrt_nr:
        addi    $sp, $sp, -16
        sw      $ra, 12($sp)
        sw      $s0, 8($sp)
        sw      $s1, 4($sp)
        sw      $s2, 0($sp)

        move    $s0, $a0 #$s0 -> x
        move    $s1, $a1 #$s1 -> i
        move    $s2, $0 #$s2 -> retorno

        bne     $s1, 0, caso1 #if($s0(m)!=0)-> caso1
        addi    $s2, $zero, 1   #return 1
        j       fim
```

Caso 1 e Caso 2

```
caso1: # else if(i>0){
        blt     $s1, 0, caso2 #if($s0<=0) -> caso2
        move    $a0, $s0 # x
        addi    $a1, $s1, -1 # i-1
        jal     sqrt_nr # sqrt_nr(x,i-1)
        move    $a3, $v0 # salva o retorno em $a3

        div     $s5, $a0, $a3 # x / retorno da chamada

        add     $s6, $s5, $a3 #soma (sqrt_nr(x, i-1) e (x/sqrt_nr(x, i-1)))
        div     $s5, $s6, 2 # $s6/2

        add     $s2, $zero, $s5
        j       fim


caso2:
        addi    $s2, $zero, -1 #return -1
        j       fim
```

Label fim

```
fim:
        move    $v0, $s2
        lw      $ra, 12($sp)
        lw      $s0, 8($sp)
        lw      $s1, 4($sp)
        lw      $s2, 0($sp)

        addi    $sp, $sp, 16
        jr      $ra
```

Label end_loop

```
end_loop:
        li $v0, 4
        la $a0, newline
        syscall
        la $a0, impFim
        syscall
        li $v0, 10
        syscall
```

4. <u>CAPTURAS DE TELA DO MARS:</u>

Área do código compilada:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu $2,$0,0x00000004	23: li $v0, 4
	0x00400004	0x3c011001	lui $1,0x00001001	24: la $a0, ack
	0x00400008	0x34240000	ori $4,$1,0x00000000	
	0x0040000c	0x0000000c	syscall	25: syscall
	0x00400010	0x3c011001	lui $1,0x00001001	26: la $a0, newline
	0x00400014	0x34240109	ori $4,$1,0x00000109	
	0x00400018	0x0000000c	syscall	27: syscall
	0x0040001c	0x3c011001	lui $1,0x00001001	28: la $a0, espaco
	0x00400020	0x34240083	ori $4,$1,0x00000083	
	0x00400024	0x0000000c	syscall	29: syscall
	0x00400028	0x3c011001	lui $1,0x00001001	30: la $a0, newline
	0x0040002c	0x34240109	ori $4,$1,0x00000109	
	0x00400030	0x0000000c	syscall	31: syscall
	0x00400034	0x24020004	addiu $2,$0,0x00000004	32: li $v0, 4
	0x00400038	0x3c011001	lui $1,0x00001001	33: la $a0, nomes
	0x0040003c	0x3424002b	ori $4,$1,0x0000002b	
	0x00400040	0x0000000c	syscall	34: syscall
	0x00400044	0x3c011001	lui $1,0x00001001	35: la $a0, newline
	0x00400048	0x34240109	ori $4,$1,0x00000109	
	0x0040004c	0x0000000c	syscall	36: syscall
	0x00400050	0x24020004	addiu $2,$0,0x00000004	37: li $v0, 4
	0x00400054	0x3c011001	lui $1,0x00001001	38: la $a0, espaco
	0x00400058	0x34240083	ori $4,$1,0x00000083	
	0x0040005c	0x0000000c	syscall	39: syscall
	0x00400060	0x3c011001	lui $1,0x00001001	40: la $a0, newline
	0x00400064	0x34240109	ori $4,$1,0x00000109	
	0x00400068	0x0000000c	syscall	41: syscall
	0x0040006c	0x00004021	addu $8,$0,$0	42: move $t0, $zero #x

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x676f7250	0x616d6172	0x20656420	0x7a696152	0x61755120	0x64617264	0x202d2061	0x7477654e
0x10010020	0x522d6e6f	0x73687061	0x44006e6f	0x6e657365	0x766c6f76	0x726f6465	0x203a7365	0x61754c3c
0x10010040	0x5320616e	0x6974736f	0x2c6f7373	0x75644520	0x61647261	0x63754620	0x202c7368	0x6e726546
0x10010060	0x61646e61	0x61724620	0x6863736e	0x69637365	0x202c696e	0x6e726546	0x6f646e61	0x74654e20
0x10010080	0x3d003e6f	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d
0x100100a0	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x3d3d3d3d	0x6944003d	0x65746967	0x20736f20	0x61726170
0x100100c0	0x7274656d	0x7820736f	0x69206520	0x72617020	0x61632061	0x6c75636c	0x73207261	0x5f747271
0x100100e0	0x7828726e	0x2969202c	0x20756f20	0x7020312d	0x20617261	0x726f6261	0x20726174	0x78652061
0x10010100	0xe7756365	0x0a206f61	0x73000a00	0x28747271	0x00202c00	0x203d2029	0x6c615600	0x6e20726f
0x10010120	0x74616765	0x206f7669	0x69676964	0x6f646174	0x72502021	0x6172676f	0x6520616d	0x7265636e
0x10010140	0x6f646172	0x00000021	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

O estado dos registradores ao término de uma execução:

Registers | Coproc 1 | Coproc 0

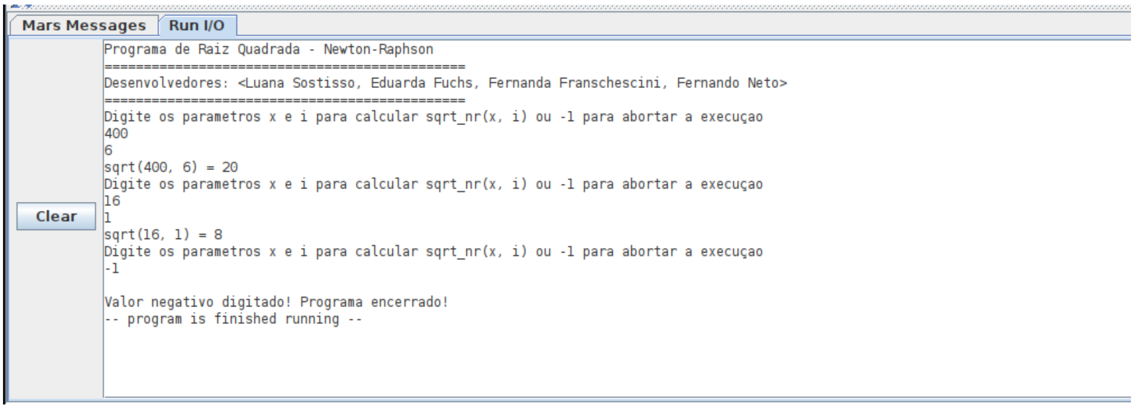
Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010119
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000001
$t0	8	0xffffffff
$t1	9	0x00000001
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000001
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000008
$s1	17	0x00000010
$s2	18	0x00000001
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000008
$s6	22	0x00000011
$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	0x004000c4
pc		0x004001e8
hi		0x00000001
lo		0x00000008

A área de pilha utilizada para a recursividade:

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0
$12 (status)	12	65299
$13 (cause)	13	32
$14 (epc)	14	4194444

Um exemplo de execução do programa:



5. CONCLUSÃO

A implementação da função de Sqr_int em Java e em Assembly foi uma tarefa muito interessante, pois permitiu explorar o mesmo desafio, porém com dificuldades diferentes. No Java, a implementação foi criada de uma maneira relativamente simples e clara, o que facilita a compreensão do problema proposto pelo trabalho. Entretanto, a implementação em Assembly exigiu mais tempo e dedicação, pois a arquitetura do MIPS é mais complexa em relação à programação de um algoritmo em alto nível. A formatação foi feita de forma clara e objetiva, o que facilitou o processo de codificação do algoritmo em linguagem de máquina. Conclui-se que a programação em Assembly, embora mais desafiadora, fornece um conhecimento maior sobre a relação entre arquitetura e organização, além de uma compreensão mais profunda do funcionamento interno de um processador MIPS.