

Trabalho de Organização e Arquitetura de Processadores (OAP)

Benjamin Pantoja Mattar, Bernardo Luiz Padilha Zamin

Setembro de 2023

1 Introdução

Este trabalho é parte da disciplina de Organização e Arquitetura de Processadores (OAP). O trabalho consiste na compreensão de um problema algorítmico, no qual deve ser implementada a função recursiva de Wilhelm Ackermann, mais conhecida como a função de Ackermann. Deve ser implementada em alto nível, para o que foi decidido usar Java, e principalmente a programação em assembly, considerando todo o ISA (*Instruction Set Architecture*) do MIPS.

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 *"ackermann"*, que é responsável por toda a lógica do programa. A representação em [Figura 1](#).

3 Descrição em Assembly

A implementação do algoritmo em Assembly foi mais complexa do que em alto nível, como era esperado. Para facilitar a compreensão, será usada a representação por imagens, além de uma breve explicação do funcionamento da função Ackermann de forma recursiva. A seção de dados está sendo representada na [Figura 2](#). A *label* principal *"main"* está sendo representada na [Figura 3](#). A *label* *"loop"*, responsável pela repetição do programa, está sendo representada na [Figura 4](#) e na [Figura 5](#). A *label* *"Ackermann"* está sendo representada na [Figura 6](#). As *labels* *"caso1"* e *"caso2"* estão sendo representadas na [Figura 7](#). A *label* *"fim"* está sendo representada na [Figura 8](#). Por fim, a *label* *"end.loop"*, responsável pelo término do programa, está sendo representada na [Figura 9](#). A área de execução do programa está sendo representada na [Figura 10](#). Após a

execução, dado o fim do programa, o estado dos registradores está representado na *Figura 11*. As áreas *"Text Segment"* e *"Data Segment"* estão sendo representadas na *Figura 12*.

Para o desenvolvimento da função Ackermann na linguagem Assembly MIPS, iniciamos configurando o quadro de pilha (stack frame) para a função Ackermann. Isso é necessário para preservar o estado dos registradores ao chamar a função recursivamente. Depois, movemos os argumentos da função ("m" e "n") para registradores temporários para facilitar o acesso. Assim, verificamos se "m" é igual a 0. Se "m" for igual a 0, pula-se para a label "caso1". Caso contrário, continua executando o código. Depois, verificamos se "m" é menor ou igual a 0. Se for menor ou igual a 0, ele verifica se "n" é igual a 0. Se ambas as condições forem atendidas, chama-se a função Ackermann com "m" - 1 e 1 como argumentos, armazenando o resultado em "s2". Para finalizar, lida-se com o caso em que "m" e "n" não são 0. Primeiro, verifica-se se ambos são maiores que 0. Em seguida, chama-se a função Ackermann duas vezes: uma vez com "m" e "n" - 1 como argumentos e outra vez com "m" - 1 e o resultado anterior como argumentos. O resultado final é armazenado em "s2" e, em seguida, retornado em "v0", restaurando os registradores salvos no quadro de pilha ao final.

4 Conclusão

A implementação da função de Ackermann 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.

Figure 1: Descrição em Java

```

1  import java.util.Scanner;
2
3  public class App {
4      public static void main(String[] args) throws Exception {
5          Scanner in = new Scanner(System.in);
6
7          System.out.println(x:"Programa Ackermann");
8          System.out.println(x:"=====");
9          System.out.println(x:"Componentes: <Benjamin Mattar e Bernardo Zamin>");
10         System.out.println(x:"=====");
11         int m = 0;
12         int n = 0;
13         while (m >= 0 && n >= 0) {
14             System.out.println(x:"Digite os parametros m e n para calcular A(m, n) ou -1 para abortar a execucao");
15             m = in.nextInt();
16             if (m < 0) {
17                 break;
18             }
19             n = in.nextInt();
20             if (n < 0) {
21                 break;
22             }
23             int resp = ackermann(m, n);
24             System.out.printf(format:"A(%d, %d) = %d\n", m, n, resp);
25         }
26         System.out.println(x:"Valor negativo digitado! Programa encerrado!");
27     }
28
29     public static int ackermann(int m, int n) {
30         if (m == 0) {
31             return n + 1;
32         } else if (m > 0 && n == 0) {
33             return ackermann(m - 1, n:1);
34         } else if (m > 0 && n > 0) {
35             return ackermann(m - 1, ackermann(m, n - 1));
36         }
37         return -1;
38     }
39 }
40

```

Figure 2: Seção de dados

```

TRABALHO10APPINAL.asm
1  .data
2
3  ack: .asciiz "          Programa Ackermann"
4
5  nomes: .asciiz "Componentes: <Benjamin Mattar e Bernardo Zamin>"
6
7  espaco: .asciiz "===== "
8
9  mnString: .asciiz "Digite os parametros m e n para calcular A(m, n) ou -1 para abortar a execucao "
10
11 newline: .asciiz "\n"      # String de nova linha
12
13 resposta1: .asciiz "A("
14 resposta2: .asciiz ", "
15 resposta3: .asciiz ")" = "
16
17
18 impFim: .asciiz "Valor negativo digitado! Programa encerrado!"
19 #A(1, 2) = 4
20 .text

```

Figure 3: 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 #m
    move $t1, $zero #n
    li $t5, 1
```

Figure 4: Label Loop

```
TRABALHO10AFINAL.asm
45 loop:
46     # Verificar se m >= 0 exemplo: m=-1 dai t2 =1
47     #alti $t2, $t0, 0 # $t2 = 1 se $t0 < 0, senão $t2 = 0
48
49     # Verificar se n >= 0 exemplo: n=-1 dai t3=1
50     #alti $t3, $t1, 0 # $t3 = 1 se $t1 < 0, senão $t3 = 0
51
52     # Verificar se pelo menos um dos m >= 0 ou n >= 0
53     #se qualquer um dos dois for negativo, $t4 = 1
54     #or $t4, $t2, $t3 # $t4 = 1 se $t2 = 1 ou $t3 = 1, senão $t4 = 0
55
56     # sair do loop se a condição for falsa
57     #beq $t4, $t5, end_loop # Pular para o fim do loop se $t4 = 0
58
59     li $v0, 4
60     la $a0, mnString
61     syscall
62     li $v0, 5
63     syscall
64
65     move $t0, $v0 #m
66     blt $t0, $zero, end_loop
67
68     move $t1, $v0
69
70     li $v0, 5
71     syscall
72     move $t1, $v0 #n
73     blt $t1, $zero, end_loop
74
75     move $s2, $v0
76
```

Figure 5: Label Loop

```

TRABALHO1OAPFINAL.asm
76
77     move    $a0, $s1
78     move    $a1, $s2
79
80     jal     Ackermann
81     move    $s0, $v0
82
83     li      $v0, 4
84     la      $a0, resposta1
85     syscall
86     move    $a0, $t0
87     li      $v0, 1
88     syscall
89     li      $v0, 4
90     la      $a0, resposta2
91     syscall
92     move    $a0, $t1
93     li      $v0, 1
94     syscall
95     li      $v0, 4
96     la      $a0, resposta3
97     syscall
98     li      $v0, 1
99     move    $a0, $s0
100    syscall
101
102    li      $v0, 4
103    la      $a0, newline
104    syscall
105    j       loop
106
107 Ackermann:

```

Figure 6: Label Ackermann

```

TRABALHO1OAPFINAL.asm
107 Ackermann:
108     addi    $sp, $sp, -24
109     sw      $ra, 20($sp)
110     sw      $s0, 16($sp)
111     sw      $s1, 12($sp)
112     sw      $s2, 8($sp)
113     sw      $s3, 4($sp)
114     sw      $s4, 0($sp)
115
116     move    $s0, $a0
117     move    $s1, $a1
118     move    $s2, $0
119
120     bne     $s0, 0, caso1
121     addi    $s2, $s1, 1

```

Figure 7: Labels caso1 e caso2

TRABALHO1OAPFINAL.asm	
122	<i>caso1:</i>
123	ble \$s0, 0, caso2
124	bne \$s1, 0, caso2
125	addi \$a0, \$s0, -1
126	addi \$a1, \$0, 1
127	jal Ackermann
128	move \$s2, \$v0
129	<i>caso2:</i>
130	ble \$s0, 0, fim
131	ble \$s1, 0, fim
132	move \$a0, \$s0
133	addi \$a1, \$s1, -1
134	jal Ackermann
135	move \$a1, \$v0
136	addi \$a0, \$s0, -1
137	jal Ackermann
138	move \$s2, \$v0

Figure 8: Label fim

139	<i>fim:</i>
140	move \$v0, \$s2
141	lw \$ra, 20(\$sp)
142	lw \$s0, 16(\$sp)
143	lw \$s1, 12(\$sp)
144	lw \$s2, 8(\$sp)
145	lw \$s3, 4(\$sp)
146	lw \$s4, 0(\$sp)
147	addi \$sp, \$sp, 24
148	jr \$ra
149	

Figure 9: Label end_loop

```
150  end_loop:
151      li $v0, 4
152      la $a0, newline
153      syscall
154      la $a0, impFim
155      syscall
156      li $v0, 10
157      syscall
```

Figure 10: Área de execução do algoritmo

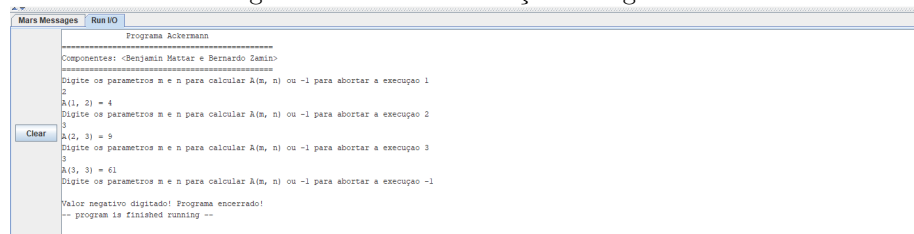




Figure 11: Área de registradores

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	10
\$v1	3	0
\$a0	4	268501213
\$a1	5	60
\$a2	6	0
\$a3	7	0
\$t0	8	-1
\$t1	9	3
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	1
\$t6	14	0
\$t7	15	0
\$s0	16	61
\$s1	17	3
\$s2	18	3
\$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	4194500
pc		4194820
hi		0
lo		0

Figure 12: Áreas Text Segment e Data Segment

Text Segment								
Dist	Address	Code	Basic	Source				
<input type="checkbox"/>	4194304	0x34200004	addis r2,r0,4	23:	li r0, 4			
<input type="checkbox"/>	4194308	0x34210013	lsl r1, r0, #7	24:	la r0, acb			
<input type="checkbox"/>	4194312	0x34240000	ori r4, r1, #0					
<input type="checkbox"/>	4194316	0x0000000c	syscall	26:	syscall			
<input type="checkbox"/>	4194320	0x34210013	lsl r1, r0, #0	26:	la r0, newline			
<input type="checkbox"/>	4194324	0x34240000	ori r4, r1, #0					
<input type="checkbox"/>	4194328	0x0000000c	syscall	27:	syscall			
<input type="checkbox"/>	4194332	0x34210013	lsl r1, r0, #7	28:	la r0, espacio			
<input type="checkbox"/>	4194336	0x34240000	ori r4, r1, #0					
<input type="checkbox"/>	4194340	0x0000000c	syscall	28:	syscall			
<input type="checkbox"/>	4194344	0x34210013	lsl r1, r0, #0	30:	la r0, newline			
<input type="checkbox"/>	4194348	0x34240000	ori r4, r1, #0					
<input type="checkbox"/>	4194352	0x0000000c	syscall	31:	syscall			
<input type="checkbox"/>	4194356	0x34200004	addis r2,r0,4	32:	li r0, 4			
<input type="checkbox"/>	4194360	0x34210013	lsl r1, r0, #7	33:	la r0, nome			
<input type="checkbox"/>	4194364	0x34240000	ori r4, r1, #0					
<input type="checkbox"/>	4194368	0x0000000c	syscall	34:	syscall			
Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
26550092	535976208	535976208	535976208	1917853728	494887832	1092641136	1919249280	1852727664
26550104	1336005192	1701736304	1936037712	1111336664	4341650208	84410864	195378464	169623549
26550106	1919236688	1685217648	163296496	1047423344	1027423488	1027423549	1027423549	1027423549
26550108	1027423549	1027423549	1027423549	1027423549	1027423549	1027423549	1027423549	4013373
26550110	1768384936	1964394100	1618730288	1704673196	159663636	169662304	195117856	54357135
26550112	1668047203	1918987381	161354656	69583052	544567072	188187933	543257185	1919902305
26550114	548366968	2019593345	-811737248	2125665	675548496	2105416	540576841	181831836
<div><div> </div><div><div><input checked="" type="checkbox"/> 0x10000000 (data)</div><div><input type="checkbox"/> Hexadecimal Addresses</div><div><input type="checkbox"/> Hexadecimal Values</div><div><input type="checkbox"/> ASCII</div></div></div>								