Função de Ackermann

Dhruv Babani, João Pedro Sostruznik Sotero da Cunha, Miguel de Oliveira Farias Warken e Vítor Aguirre Caús

Escola Politécnica – PUCRS

28 de abril de 2023

**Etapa 1: Especificação do Algoritmo em Alto Nível**

O algoritmo em alto nível para realização da recursão da Função de Ackermann foi desenvolvido em Java, como apresentado no código abaixo:

public static int ackermann (int m, int n){

return ackermannRec(m, n);

}

private static int ackermannRec(int m, int n){

if(m == 0){

return n+1;

} else if(m > 0 e n == 0){

return ackermannRec(m-1, 1);

} else{

return ackermannRec(m-1, ackermannRec(m, n-1));

}

}

O segmento apresentado acima realiza os cálculos da recursão, assumindo que os parâmetros estejam corretamente inseridos pelo usuário. Para assegurar isto, definimos algumas condições na nossa função “*main*”, encerrando o programa caso *‘m’* seja negativo:

public static void main(String[] args){

Scanner scan = new Scanner(System.in);

System.out.println("Programa Ackermann\n Digite os parâmetros m e n para calcular A(m, n) ou -1 para abortar a execução");

int m = scanner.nextInt();

if (!(m<0)){ //se M **não** for negativo

n = scanner.nextInt();

resultado = ackermann(m,n);

System.out.println(resultado);

} else{

System.out.println(“Execução encerrada”);

}

scan.close();

}

Note que para realizar o processo de finalizar o programa, temos que chamar o sistema com a função desejada no registrador “$v0”, e para apresentar o resultado como desejado no enunciado, algumas operações extras devem ser construídas.

**Etapa 2: Especificação do Algoritmo em Baixo Nível (Linguagem de Máquina)**

.data

strInicio: .asciiz "Programa Ackermann\nDigite os parametros m e n para calcular A(m, n) ou -1 para abortar a execucao\n"

strResAbre: .asciiz "A("

strResVirg: .asciiz ","

strResFec: .asciiz ") = "

strEncerra: .asciiz "\n\nExecucao encerrada"

valorM: .word 0

valorN: .word 0

resultado: .word 0

Acima, é mostrada a área de dados do código, com valores asciiz representando as Strings impressas no início e fim da execução, além de words para armazenar os valores iniciais de *m* e *n* e o resultado. Em seguida, mostra-se a função *main.*

.text

.globl main # void main()

main:

la $s0, valorM # carrega endereco valorM

la $s1, valorN # carrega endereco valorN

la $s2, resultado # carrega endereco resultado

addiu $s3, $s3, 1 # $s3 = 1 (auxiliar para o caso m > 0 && n == 0)

la $a0, strInicio # $a0 <- &strIicio

li $v0, 4 # $v0 <- printString

syscall # chama SO

li $v0, 5 # $v0 <- ler int

syscall # chama SO para ler m

move $t0, $v0 # $t0 <- $v0 (valorM)

blt $t0, $zero, fimExecucao # if (m < 0) fim execucao

sw $t0, 0($s0) # 0($s0) <-$t0 (valorM)

li $v0, 5 # $v0 <- ler int

syscall # chama SO para ler N

move $t1, $v0 # $t1 <- $v0 (valorN)

sw $t1, 0($s1) # 0($s1) <- $t1 (valorN)

addiu $sp, $sp, -12 # libera espaco na pilha

sw $t0, 8($sp) # 8($sp) <- $t0 (valorM)

sw $t1, 4($sp) # 4($sp) <- $t1 (valorN)

sw $ra, 0($sp) # 0($sp) <- $ra

jal Ackermann # vai para funcao

#apos retorno

lw $ra, 0($sp) # recupera $ra

lw $t1, 4($sp) # recupera ValorN

lw $t0, 8($sp) # recupera ValorM

addiu $sp, $sp, 12 # recupera espaco da pilha

j printResultado # imprime resultado

A função *main* apresentada acima, inicialmente, imprime na tela a mensagem de início da execução do programa, seguida pela coleta dos valores *m* e *n*. Após isso, realiza-se o salvamento de contexto na área de pilha $sp, seguido de um salto incondicional para a função recursiva. Após o retorno do resultado, os dados do contexto são recuperados e salta-se para a impressão do valor final na tela. A seguir, é apresentada a função recursiva.

# Verifica se valorM == 0

Ackermann:

lw $a0, 8($sp) # $a0 = valorM

bne $a0, $zero, AckermannRec # if(m != 0) proximo label

lw $t0, 4($sp) # $t0 = valorN

addiu $a0, $t0, 1 # $t0 = valorN +1

sw $a0, 0($s2) # resultado = valorN +1

jr $ra # retorna

# Verifica se valorN == 0

AckermannRec:

lw $a0, 4($sp) # $a0 = valorN armazenado

bne $a0, $zero, AckermannDuplo # if(n != 0)proximo label

lw $t0, 8($sp) # $t0 <- valorM armazenado

addiu $sp, $sp, -12 # libera espaco na pilha

addiu $t0, $t0, -1 # $t0 = $t0 -1

sw $t0, 8($sp) # 8($sp) <- $t0 (valorM -1)

sw $s3, 4($sp) # 4($sp) <- 1

sw $ra, 0($sp) # 0($sp) <- $ra

jal Ackermann # vai para funcao

lw $ra, 0($sp) # recupera $ra

lw $t1, 4($sp) # recupera valorN

lw $t0, 8($sp) # recupera valorM

addiu $sp, $sp, 12 # recupera espaco da pilha

jr $ra # retorna

# valorM > 0 e valorN > 0, (realiza recursao dentro da recursao)

AckermannDuplo:

lw $t0, 8($sp) # $t0 = valorM armazenado

lw $t1, 4($sp) # $t1 = valorN armazenado

addiu $sp, $sp, -12 # libera espaco na pilha

addiu $t1, $t1, -1 # $t1 = $t1 - 1 (valorN -1)

sw $t0, 8($sp) # 8($sp) <- $t0 (valorM)

sw $t1, 4($sp) # 4($sp) <- $t1 (valorN -1)

sw $ra, 0($sp) # 0($sp) <- $ra

jal Ackermann # vai para funcao

# apos retorno

lw $ra, 0($sp) # recupera $ra

lw $t1, 4($sp) # recupera valorN -1

lw $t0, 8($sp) # recupera valorM

addiu $sp, $sp, 12 # recupera espaco da pilha

addiu $t0, $t0, -1 # $t0 = valorM -1

lw $t1, 0($s2) # $t1 = resultado atual

addiu $sp, $sp, -12 # libera espaco na pilha

sw $t0, 8($sp) # 8($sp) <- $t0 (valorM -1)

sw $t1, 4($sp) # 4($sp) <- $t1 (resultado)

sw $ra, 0($sp) # 0($sp) <- $ra

jal Ackermann # vai para funcao

lw $ra, 0($sp) # recupera $ra

lw $t1, 4($sp) # recupera resultado anterior

lw $t0, 8($sp) # recupera valorM -1

addiu $sp, $sp, 12 # recupera espaco da pilha

jr $ra # retorna

A função desenvolvida é dividida em três *labels*: o primeiro, ***Ackermann***, verifica se *m* = 0 (operação equivalente ao **if (m == 0)** da implementação Java) e retorna ***n + 1***.

Em ***AckermannRec***, é considerado o teste **if (m > 0 && n == 0)** do código Java, que realiza a recursão em ***Ackermann*** com os valores **m-1** e **n =** **1** (este último armazenado no registrador **$s3**). Já em ***AckermannDuplo***, a etapa **if (m > 0 && n > 0)** do Java é executada, onde realizasse uma recursão em ***Ackermann*** com **m-1** e **n = Ackermann (m, n-1)**.Por fim, após o retorno do resultado, o programa salta incondicionalmente para a impressão do resultado.

printResultado:

la $a0, strResAbre # $a0 <- &strResAbre

li $v0, 4 # $v0 <- 4 (print string)

syscall # chama SO

lw $a0, 0($s0) # $a0 <- &valorM

li $v0, 1 # $v0 <- 1 (print int)

syscall # chama SO

la $a0, strResVirg # $a0 <- &strResVirg

li $v0, 4 # $v0 <- 4 (print string)

syscall # chama SO

lw $a0, 0($s1) # $a0 <- &valorN

li $v0, 1 # $v0 <- 1 (print int)

syscall # chama SO

la $a0, strResFec # $a0 <- &strResFec

li $v0, 4 # $v0 <- 4 (print string)

syscall # chama SO

lw $a0, 0($s2) # $a0 <- &resultado

li $v0, 1 # $v0 <- 1 (print int)

syscall # chama SO

fimExecucao:

la $a0, strEncerra # $a0 <- &strEncerra

li $v0, 4 # $v0 <- 4 (print string)

syscall # chama SO

li $v0, 10 # $vo <-10 (encerra execucao)

syscall # chama SO

Nesta última seção da implementação *Aseembly*, é impresso o resultado da função na formatação **A(m, n) = resultado**. Em seguida, o programa tem sua execução encerrada.

**Etapa 3: Capturas de Tela do Funcionamento do Algoritmo (m = 3, n = 4):**

1. **Área de Código Compilada:**

Tabela

Descrição gerada automaticamente

**Tabela

Descrição gerada automaticamente**

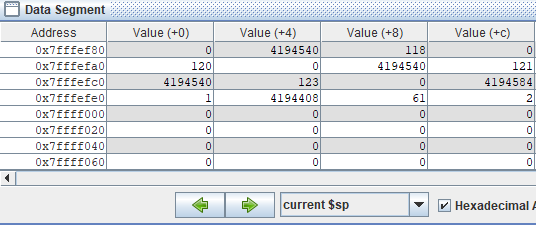
1. **Estado dos registradores após uma execução (m = 3, n = 4):**

**Tabela

Descrição gerada automaticamente com confiança média**

Note que os registradores destacados em vermelho armazenam o resultado ($a0), e os registradores temporários $t0 e $t1 armazenam os valores de m e n, respectivamente. Além disso, os indicados em azul mostram os endereços de memória do valor de *m* ($s0), valor de *n* ($s1), resultado ($s2), e um número “1” auxiliar ($s3).

1. **Área de Pilha Utilizada Para Recursividade:**

****

Tabela

Descrição gerada automaticamente