Resolução da Função de Ackermann

George S. Rother

Matheus Faccio Poletto

Escola Politécnica – PUCRS

10/05/2023

Introdução

O objetivo deste trabalho, trata-se da criação de um algoritmo capaz de calcular a fórmula de Ackermann dentro da linguagem MIPS.

$$A(m,n) = \left\{ egin{array}{ll} n+1 & ext{se } m=0 \ A(m-1,1) & ext{se } m>0 ext{ e } n=0 \ A(m-1,A(m,n-1)) & ext{se } m>0 ext{ e } n>0 \end{array}
ight.$$

Representação em Alto nível do algoritmo trabalhado(Linguagem utilizada: Java-17Lts).

```
J app.java
     import java.util.Scanner;
     public class app{
         public static void main(String[] args){
             Scanner in = new Scanner(System.in);
             System.out.println("Programa de Ackermann");
             System.out.print("Insira o valor de m: ");
             int m = in.nextInt();
                 System.out.println("Valor de m inválido");
                 System.exit(0);
             System.out.print("Insira o valor de n: ");
13
              int n = in.nextInt();
              if (n < \theta){
                 System.out.println("Valor de n inválido");
                 System.exit(0);
              System.out.println("A(" + m + ", " + n + ") = " + Ackermann(m, n));
              in.close();
         public static int Ackermann(int m, int n){
             if(m == 0){
              else if(n == 0){
                 return Ackermann(m - 1, 1);
                 return Ackermann(m - 1, Ackermann(m, n - 1));
```

1. - Funcionamento do programa

A resolução deste problema foi operada dentro do compilador MARS, o qual utiliza a linguagem MIPS.

O código inicia-se solicitando, um de cada vez, os números os quais serão calculados dentro da função:

```
Programa de Ackermann
Insira o valor de m: 2
Insira o valor de n: 2
```

Após coletados os dados, o programa inicia o tratamento e resolução do problema, gerando o output do valor referente:

```
A(2, 2) = 7

-- program is finished running --
```

Os valores finais dos registradores referentes ao caso exemplo "(2,2)", são os seguintes:

Registers	Coproc 1 Cop	roc 0		
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x10010000		
\$v0	2	0x00000000a		
\$vl	3	0x00000000		
\$a0	4	0x00000000a		
\$al	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0x00000007		
\$t1	9	0x10010068		
\$t2	10	0x00000000		
\$t3	11	0x10010064		
\$t4	12	0x10010072		
\$t5	13	0x00000007		
\$t6	14	0x00000002		
\$t7	15	0x00000002		
\$s0	16	0x00000000		
\$sl	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		0x00400150		
hi		0x00000000		
lo		0x00000000		

Data Segme	nt							□ □
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x676f7250	0x616d6172	0x20656420	0x656b6341	0x6e616d72	0x000a206e	0x69736e49	0x6f206172 -
0x10010020	0x6c617620	0x6420726f	0x3a6d2065	0x6e490020	0x61726973	0x76206f20	0x726f6c61	0x20656420
0x10010040	0x00203a6e	0x6f6c6156	0x656e2072	0x69746167	0x6e206f76	0x61206fe3	0x74696563	0x0000006f =
0x10010060	0x00000002	0x00000002	0x00000007	0x2c002841	0x20290020	0x0000203d	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000 🐷

Exemplo de output para entrada negativa em N:

```
Programa de Ackermann
Insira o valor de m: 2
Insira o valor de n: -1
Valor negativo não aceito
-- program is finished running --
```

Valores finais nos registradores:

Registers	(Coproc 1	Сор	roc 0	
Name		Numb	er		Value
\$zero			0		0x00000000
\$at			1		0x10010000
\$v0			2		0x00000000a
\$vl			3		0x00000000
\$a0			4		0x10010044
\$al			5		0x00000000
\$a2			6		0x00000000
\$a3			7		0x00000000
\$t0			8		0x000000002
\$tl			9		Oxffffffff
\$t2			10		0x00000000
\$t3			11		0x10010060
\$t4			12		0x00000000
\$t5			13		0x00000000
\$t6			14		0x00000000
\$t7			15		0x00000000
\$s0			16		0x00000000
\$sl			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					0x004000b8
hi					0x00000000
lo					0x00000000

	Data Segme	ent							ר בי
Ė	Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
	0x10010000	0x676f7250	0x616d6172	0x20656420	0x656b6341	0x6e616d72	0x000a206e	0x69736e49	0x6f206172 -
	0x10010020	0x6c617620	0x6420726f	0x3a6d2065	0x6e490020	0x61726973	0x76206f20	0x726f6c61	0x20656420
	0x10010040	0x00203a6e	0x6f6c6156	0x656e2072	0x69746167	0x6e206f76	0x61206fe3	0x74696563	0x0000006f =
	0x10010060	0x00000002	0x00000000	0x00000000	0x2c002841	0x20290020	0x0000203d	0x00000000	0x00000000
	0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
	0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
	0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
	0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000 🐷

Exemplo de output com entrada negativa em M:

Programa de Ackermann Insira o valor de m: -1 Valor negativo não aceito -- program is finished running --

Estado final dos registradores:

Registers	Coproc 1 Cop	roc 0			
Name	Number	Value			
\$zero	0	0x00000000			
\$at	1	0x10010000			
\$v0	2	0x0000000a			
\$vl	3	0x00000000			
\$a0	4	0x10010044			
\$al	5	0x00000000			
\$a2	6	0x00000000			
\$a3	7	0x00000000			
\$t0	8	0xfffffff			
\$t1	9	0x00000000			
\$t2	10	0x00000000			
\$t3	11	0x00000000			
\$t4	12	0x00000000			
\$t5	13	0x00000000			
\$t6	14	0x00000000			
\$t7	15	0x00000000			
\$s0	16	0x00000000			
\$sl	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			
рс		0x004000b8			
hi		0x00000000			
lo		0x00000000			

Data Segme	nt							<u> </u>
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x676f7250	0x616d6172	0x20656420	0x656b6341	0x6e616d72	0x000a206e	0x69736e49	0x6f206172
0x10010020	0x6c617620	0x6420726f	0x3a6d2065	0x6e490020	0x61726973	0x76206f20	0x726f6c6l	0x20656420
0x10010040	0x00203a6e	0x6f6c6156	0x656e2072	0x69746167	0x6e206f76	0x61206fe3	0x74696563	0x0000006f
0x10010060	0x00000000	0x00000000	0x00000000	0x2c002841	0x20290020	0x0000203d	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

2 - Algoritmo do programa

O programa possui declarações no escopo ".data" referentes aos textos utilizados e para os armazenamentos das entradas as quais o usuário irá realizar e uma para armazenar o valor final.

Aqui são declarados e inicializados dados estáticos que poderão ser utilizadas em todo programa:

```
entrada: .asciiz "Programa de Ackermann \n"
  entrada2: .asciiz "Insira o valor de m: "
  entrada3: .asciiz "Insira o valor de n: "
  negativo: .asciiz "Valor negativo não aceito"
  m: .word 0
  n: .word 0
  result: .word 0
  printResult1: .asciiz "A("
  printResult2: .asciiz ", "
  printResult3: .asciiz ") = "
```

A "main" do programa empilha as Strings que serão escritas na tela e chama o sistema para que elas sejam printadas, solicita o primeiro valor e com esse valor recebido o testa primeiramente se é igual a zero e após isso se é negativo. Caso o primeiro teste retorne falso, o programa passa para o segundo que verificará se o valor é negativo, se for, chama o método "encerra", o qual irá dar fim ao programa. Caso passe no primeiro ou no segundo teste o programa continuará:

```
globl main
main:
    la $a0, entrada # pega o endereço da mensagem
    li $v0, 4  # passo o valor 4 para o v0 que significa impressão de string syscall # chama o sistema operacional para imprimir a mensagem
    la $a0, entrada2 # pega o endereço da mensagem
    li $v0, 4  # passo o valor 4 para o v0 que significa impressão de string syscall # chama o sistema operacional para imprimir a mensagem
    syscall
    li $v0, 5
                      # passo o valor 5 para o v0 que significa leitura de inteiro
    syscall # chama o sistema operacional para ler o valor de m
move $t0, $v0 # move o valor de m para o registrador t0
    beq $t0, $zero, continuaM # compara se m for igual a 0, se for, continua
    blez $t0, encerra # se m for negativo, encerra o programa
continuaM:
continuaM:
   la $t3, m
                      # pega o endereÃ$o da variÃ;vel m
    sw $t0, 0($t3) # salva o valor de m na variã; vel m
    la $aO, entrada3 # pega o endereÃ$o da mensagem
    li $v0, 4  # passo o valor 4 para o v0 que significa impressÃfo de string
    syscall
                       # chama o sistema operacional para imprimir a mensagem
   1i $v0.5
                      # passo o valor 5 para o v0 que significa leitura de inteiro
                     # chama o sistema operacional para ler o valor de n
    move \ \$t1, \ \$v0 \qquad \  \  \# \ move \ o \ valor \ de \ n \ para \ o \ registrador \ t1
    beq $t1, $zero, continuaN # se n for igual a 0, continua
    blez $t1, encerra # se n for negativo, encerra o programa
```

Após solicitado o valor de N, irá fazer os mesmos testes realizados com o valor M em N. Caso esteja tudo ok irá continuar o programa(cai no escopo "continuaN").

```
continuaN:
   la $t3, n
                   # pega o endereço da variÃ;vel n
   sw $t1, 0($t3) # salva o valor de n na variã;vel n
   addi $sp, $sp, -12 # aloca espaã$o para os parã¢metros e para o retorno
   sw $t0, 8($sp)
                   # salva o valor de m na pilha
   sw $t1, 4($sp)
                   # salva o valor de n na pilha
                   # salva o valor de retorno na pilha
   sw $ra, O($sp)
   jal Ackermann # chama a funã§ãto Ackermann
   lw $ra, O($sp) # recupera o valor de estado da pilha
   lw $t0, 8($sp) # recupera o valor de retorno na pilha
   addi $sp, $sp, 12 # desaloca espaã$o para os parã¢metros e para o retorno
   la $t1, result # pega o endereÃ$o da variÃ;vel result
   sw $t0, 0($t1) # salva o valor de retorno na vari\tilde{A}; vel result
                   # vai terminar o programa
```

Caso o valor de M ou N seja negativo, o programa irá pular para o método "encerra", que irá finalizar o programa antecipadamente e imprimirá a frase, "Valor negativo não aceito".

```
encerra:

la $aO, negativo

li $vO, 4  # passo o valor 4 para o v0 que significa impressão de string

syscall

li $vO, 10  # passo o valor 10 para o v0 que significa encerrar o programa

syscall
```

Continuando o programa, daremos inicio a função de Ackermann. Sempre que iremos realizar uma chamada na função, independente de ser a primeira ou chamadas recursivas, precisamos reservar um espaço na memória ram para guardar os valores que serão de utilidade para função e retorno na mesma.

O espaço de reserva na memória é feito imitando uma pilha. Ou seja, quando subtraímos 12 bytes do nosso stackPointer estamos reservando aqueles 12 bytes para guardar valores. Nesses 12 bytes, por padrão de código, sempre era armazenado nos espaços de 12 a 8 o valor de M(primeiro parâmetro da func.) na chamada da função Ackermann, e esse espaço também era utilizado para guardar o valor de retorno das chamadas recursivas. No espaço entre 8 a 4 é armazenado o valor de N(segundo parâmetro da func.) na chamada da função de Ackermann. Já no espaço restante, 4 a 0 bytes, é armazenado o endereço ao qual a função deve retornar quando acabar seu processo.

É possível observar em certos casos que foram utilizadas (reservado na memória ram) pilhas de 16 bytes. Isso ocorreu pois precisávamos guardar o valor de N quando iniciou a função pois o mesmo seria utilizado em chamadas futuras da função. O padrão da pilha se mantém o mesmo dos outros, a única diferença é que do da posição 16 a 12 é encontrado o valor de N.

Sempre ao retornar da chamada recursiva, para não perdermos a nossa localização na memória ram e ficarmos com uso extra de memória, sempre liberamos aquele espaço que estava sendo ocupado. Sendo esse espaço de 12 bytes ou 16 bytes.

O método Ackermann está descrito abaixo:

```
Ackermann:
   lw $t0, 8($sp) # recupera o valor de m da pilha
   lw $t1, 4($sp) # recupera o valor de n da pilha
   beqz $t0, else1 # se m for igual a 0, vai para o else1
   beqz $t1, else2 # se n for igual a 0, vai para o else2
   addi $t1, $t1, -1 # decrementa o valor de n
   addi $sp, $sp, -16 # aloca espaço para os parâmetros e para o retorno
   sw $t0, 12($sp) # salva o valor de m na pilha
   sw $t0, 8($sp)
                   # salva o valor de m na pilha
   sw $t1, 4($sp)
                   # salva o valor de n na pilha
   sw $ra, O($sp) # salva o valor de retorno na pilha
   jal Ackermann # chama a função Ackermann para A(m, n - 1)
   lw $ra, O($sp)  # recupera o valor de estado da pilha
   lw $t1, 8($sp)
                   # recupera o valor que deu na chamada recursiva
                   # recupera o valor de m da pilha
   lw $t0, 12($sp)
   addi $sp, $sp, 16 # desaloca espaço para os parâmetros e para o retorno
   addi $t0, $t0, -1 # decrementa o valor de m
   addi $sp, $sp, -12 # aloca espaço para os parâmetros e para o retorno
   sw $t0, 8($sp)
                   # salva o valor de m - 1 na pilha
   sw $t1, 4($sp) # salva o valor de retorna da chamada recursiva A(m, n - 1) na pilha
   sw $ra, O($sp) # salva o valor de retorno na pilha
   jal Ackermann # chama a função Ackermann para A(m - 1, A(m, n - 1))
   lw $ra, O($sp)
                   # recupera o valor de retorno da pilha
   lw $t0, 8($sp) # recupera o valor de retorna da chamada recursiva A(m - 1, A(m, n - 1)) na pilha
   addi $sp, $sp, 12 # desaloca espaço para os parâmetros e para o retorno
   sw $t0, 8($sp) # salva o valor de retorna da chamada recursiva A(m-1, A(m, n-1)) na pilha
                   # retorna para o endereco de retorno
```

O método *Ackermann* trabalha em conjunto com outras duas chamadas, "else1" e "else2", sendo que, a chamada "else1" trabalha em cima do valor de N e "else2" com os dois valores, M e N, realizando operações de incremento, decremento e chamada para o método *Ackermann* dando continuidade à função.

```
addi $t1, $t1, 1 # incrementa o valor de n
   sw $t1, 8($sp) # salva o valor de n na pilha
                   # retorna para o endereÃ$o de retorno
   jr $ra
else2:
   addi $t0, $t0, -1 # decrementa o valor de m
                  # coloca o valor 1 em n
   addi $sp, $sp, -16 # aloca espaã$o para os parā¢metros e para o retorno
   sw $t1, 12($sp) # salva o valor de n na pilha
   sw $t0, 8($sp) # salva o valor de m - 1 na pilha
   sw $t4, 4($sp) # salva o valor de 1 na pilha
   sw $ra, O($sp) # salva o valor de estado na pilha
   jal Ackermann # chama a funã§ãto Ackermann para A(m - 1, 1)
   lw $ra, O($sp) # recupera o valor de estado da pilha
   lw $t0, 8($sp) # recupera o valor de retorna da chamada recursiva A(m - 1, 1) na pilha
   lw $t1, 12($sp) # recupera o valor de n na pilha
   addi $sp, $sp, 16 # desaloca espaã$o para os parã¢metros e para o retorno
   sw $t0, 8($sp) # salva o valor de retorna da chamada recursiva A(m - 1, 1) na pilha
   ir $ra
                   # retorna para o endereã$o de retorno
```

Através de chamadas recursivas o programa terá seu andamento e assim que finalizadas todas as operações, o programa pula para a função fim, que irá recuperar os valores originais de M e N para realizar a impressão dos números os quais foram utilizados para calcular a função, buscará o resultado final que estará na .word result, e assim iniciará a impressão de "A(X, Y) = Z".

```
fim:
   la $t5, result # pega o endereÃ$o da variÃ;vel result
   lw $t5, O($t5) # pega o valor da variÃ;vel result
   la $t6 , m # pega o endereÃ$o da variÃ;vel m
   lw $t6, O($t6) # pega o valor da variÃ; vel m
   la $t7 , n
                  # pega o endereÃ$o da variÃ;vel n
   lw $t7, O($t7) # pega o valor da variÃ; vel n
   la $t4, printResult1 # pega o endereço da string printResult1
   li $v0, 4  # passo o valor 4 para o v0 que significa impressãfo de string
   move $aO, $t4  # passo o endereÃ$o da string printResult1 para o a0
   syscall
   li $v0, 1
                  # passo o valor 1 para o v0 que significa impressÃto de inteiro
   move $aO, $t6  # passo o valor de m para o aO
   svscall
   la $t4, printResult2 # pega o endereÃ$o da string printResult2
               # passo o valor 4 para o v0 que significa impressÃto de string
   li $v0, 4
                   # passo o endereÃ$o da string printResult2 para o a0
   move $a0, $t4
    syscall
   1; $v0. 1
                  # passo o valor 1 para o v0 que significa impressÃto de inteiro
   move $aO, $t7 # passo o valor de n para o aO
   syscall
   la $t4, printResult3 # pega o endereÃ$o da string printResult3
                  # passo o valor 4 para o v0 que significa impressÃfo de string
                  # passo o endereço da string printResult3 para o a0
   move $a0, $t4
   syscall
   li $v0, 1
                  # passo o valor 1 para o v0 que significa impressãto de inteiro
   move $aO, $t5 # passo o valor de result para o aO
   li $v0. 11
                 # passo o valor 11 para o v0 que significa impressÃfo de caractere
   li $aO, '\n' # passo o valor de \n para o a0
   syscall
   li $v0, 10
                   # passo o valor 10 para o v0 que significa encerrar o programa
    syscall
```

Ao final de tudo, é passado com um li (load integer) o valor 10 para \$v0 (registrador do sistema) e é executado um syscall que encerrará o programa.