



Testando o Teorema de De Morgan

Pedro Garcia

<http://www.sawp.com.br>

04 de Fevereiro de 2010

Assembly Working Party

Teorema de De Morgan

- Teorema de lógica, que obteve vasta aplicabilidade em aplicações da lógica booleana.
- Muito utilizado em computação e eletrônica para simplificação de expressões booleanas, implicando em circuitos mais simples.
- Base fundamental para produção de portas lógicas compostas por combinação de uma porta única (NAND or NOR).

Primeiro teorema de De Morgan.

- Enunciado:

Definition

“O complemento do produto é a soma dos complementos”.

- Lembrando que na aritmética booleana, o produto têm equivalência lógica “e”, enquanto que a soma equivale logicamente à operação “ou”.
- Este teorema pode ser visualizado pela tabela abaixo.

A	B	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Primeiro teorema de De Morgan.

$$\overline{(A \cdot B \cdot C \cdots N)} = \bar{A} + \bar{B} + \cdots + \bar{N}$$

Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

1 Do 1º teorema, sabemos que $\overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$



Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

- 1 Do 1º teorema, sabemos que $\overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$
- 2 Negando a expressão acima, temos que $\overline{\overline{(\overline{A} \cdot \overline{B})}} = \overline{\overline{\overline{A}} + \overline{\overline{B}}} = A \cdot B$



Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

- 1 Do 1º teorema, sabemos que $\overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$
- 2 Negando a expressão acima, temos que $\overline{\overline{(\overline{A} \cdot \overline{B})}} = \overline{\overline{\overline{A}} + \overline{\overline{B}}} = A \cdot B$
- 3 Tomando $A = \overline{X}$ e $B = \overline{Y}$



Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

- 1 Do 1º teorema, sabemos que $\overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$
- 2 Negando a expressão acima, temos que $\overline{\overline{(\overline{A} \cdot \overline{B})}} = \overline{\overline{\overline{A}} + \overline{\overline{B}}} = A \cdot B$
- 3 Tomando $A = \overline{X}$ e $B = \overline{Y}$
- 4 Obtemos que: $A \cdot B = \overline{X} \cdot \overline{Y}$



Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

- 1 Do 1º teorema, sabemos que $\overline{A \cdot B} = \overline{A} + \overline{B}$
- 2 Negando a expressão acima, temos que $\overline{\overline{A \cdot B}} = \overline{\overline{A} + \overline{B}} = A \cdot B$
- 3 Tomando $A = \overline{X}$ e $B = \overline{Y}$
- 4 Obtemos que: $A \cdot B = \overline{X} \cdot \overline{Y}$
- 5 $\overline{\overline{A} + \overline{B}} = \overline{\overline{X} + \overline{Y}}$



Segundo teorema de De Morgan.

Definition

“O complemento da soma é igual ao produto dos complementos.”

Demonstração.

- 1 Do 1º teorema, sabemos que $\overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$
- 2 Negando a expressão acima, temos que $\overline{\overline{(\overline{A} \cdot \overline{B})}} = \overline{\overline{\overline{A}} + \overline{\overline{B}}} = A \cdot B$
- 3 Tomando $A = \overline{X}$ e $B = \overline{Y}$
- 4 Obtemos que: $A \cdot B = \overline{X} \cdot \overline{Y}$
- 5 $\overline{\overline{A}} + \overline{\overline{B}} = \overline{\overline{X}} + \overline{\overline{Y}}$
- 6 Portanto, $\overline{X} \cdot \overline{Y} = \overline{(\overline{\overline{X}} + \overline{\overline{Y}})}$



Segundo teorema de De Morgan.

$$\overline{(A + B + C + \dots + N)} = \bar{A} \cdot \bar{B} \dots \bar{N}$$

C

Ilustrando *De Morgan* em C

```
1 int main(void) {
2     int a, b, n_op_ab, n_op_a_b, n_a, n_b;
3
4     /* primeiro teorema de De Morgan */
5     for (a = 0; a <= 1; a++)
6         for (b = 0; b <= 1; b++) {
7             n_op_ab = !(a & b);
8             n_a = !a;
9             n_b = !b;
10            n_op_a_b = n_a | n_b;
11            n_op_ab == n_op_a_b; /* should tb TRUE, 1 */
12        }
13
14    /* segundo teorema de De Morgan */
15    for (a = 0; a <= 1; a++)
16        for (b = 0; b <= 1; b++) {
17            n_op_ab = !(a | b);
18            n_a = !a;
19            n_b = !b;
20            n_op_a_b = n_a & n_b;
21            n_op_ab == n_op_a_b; /* should tb TRUE, 1 */
22        }
23 }
```

Implementação em ASM

Compile com

```
gcc -g demorgan.s -o demorgan
```

demorgan.s – Programa principal.

```
1 main:
2     pushl   %ebp
3     movl   %esp, %ebp
4
5     # stt: show trivial things
6     call  _preamble
7
8     # 1th De Morgan Theorem
9     call  _first
10
11    # 2th De Morgan Theorem
12    call  _second
13
14    leave
15    ret
```

preamble

_preamble

```
1 _preamble:  
2     pushl   %ebp  
3     movl   %esp, %ebp  
4     subl   $8, %esp  
5  
6     movl   $head, (%esp)  
7     call   puts  
8  
9     leave  
10    ret
```

\$head

```
1 head:  
2     .asciz "\x1b[H\x1b[2JTrue Tables from De Morgan Theorem.\n  
3     \nAuthor: Pedro Garcia."
```

_first

Cabeçalho da função

```

1 _first:
2     pushl   %ebp
3     movl   %esp, %ebp
4     subl   $8, %esp
5
6     movl   $first_demorgan, (%esp)
7     call   puts
8
9     movl   $first_true_table_head, (%esp)
10    call   printf

```

\$first_demorgan

```

1 first_demorgan:
2     .asciz "\n1th Thorem: !(A & B) == !A | !B\n"

```

\$first_true_table_head

```

1 first_true_table_head:
2     .asciz "| A      B | !(A.B) | !A + !B |\n"

```

_first

primeiro teorema de De Morgan

```
1   movl   false, %eax
2   movl   false, %ebx
3   call  _first_make_table
4
5   movl   false, %eax
6   movl   true, %ebx
7   call  _first_make_table
8
9   movl   true, %eax
10  movl   false, %ebx
11  call  _first_make_table
12
13  movl   true, %eax
14  movl   true, %ebx
15  call  _first_make_table
```

Equivalente em C

```
1   /* primeiro teorema de De Morgan */
2   for (a = 0; a <= 1; a++)
3       for (b = 0; b <= 1; b++)
4           first_make_table(a, b);
```


_first_make_table

Inserindo a função na pilha

```
1 _first_make_table:  
2     pushl   %ebp  
3     movl   %esp, %ebp  
4     subl   $64, %esp  
5  
6     movl   %eax, 4(%esp)  
7     movl   %ebx, 8(%esp)
```

_first_make_table

Processando o Primeiro Teorema de De Morgan

```
1  movl    %eax, %ecx
2  not     %ecx
3  movl    %ebx, %edx
4  not     %edx
5
6  # !A + !B
7  or      %ecx, %edx
8  movl    %edx, 16(%esp)
9
10 # !(A.B)
11 and     %ebx, %eax
12 notl    %eax
13 movl    %eax, 12(%esp)
```

Equivalente em C

```
1  n_op_ab = !(a & b);
2  n_a = !a;
3  n_b = !b;
4  n_op_a_b = n_a | n_b;
```

_first_make_table

Outputing

```
1  # print
2  movl  $true_table_fmt, (%esp)
3  call  printf
4
5  leave
6  ret
```

\$true_table_fmt

```
1 true_table_fmt:
2  .asciz "| %2d  %2d |  %2d  |  %2d  |\n"
```

_second

Cabeçalho da função

```
1 _second:  
2     pushl   %ebp  
3     movl   %esp, %ebp  
4     subl   $8, %esp  
5  
6     movl   $second_demorgan, (%esp)  
7     call   puts  
8  
9     movl   $second_true_table_head, (%esp)  
10    call   printf
```

\$second_demorgan

```
1 second_demorgan:  
2     .asciz "\n2th Theorem: !(A | B) == !A & !B\n"
```

\$second_true_table_head

```
1 second_true_table_head:  
2     .asciz "| A      B | !(A+B) | !A . !B |\n"
```

_second

segundo teorema de De Morgan

```
1   movl   false, %eax
2   movl   false, %ebx
3   call  _second_make_table
4
5   movl   false, %eax
6   movl   true, %ebx
7   call  _second_make_table
8
9   movl   true, %eax
10  movl   false, %ebx
11  call  _second_make_table
12
13  movl   true, %eax
14  movl   true, %ebx
15  call  _second_make_table
```

Equivalente em C

```
1   /* segundo teorema de De Morgan */
2   for (a = 0; a <= 1; a++)
3       for (b = 0; b <= 1; b++)
4           second_make_table(a, b);
```

__second_make_table

Inserindo a função na pilha

```
1 __second_make_table:  
2     pushl   %ebp  
3     movl   %esp, %ebp  
4     subl   $64, %esp  
5  
6     movl   %eax, 4(%esp)  
7     movl   %ebx, 8(%esp)
```

_second_make_table

Processando o Segundo Teorema de De Morgan

```
1  movl    %eax, %ecx
2  not     %ecx
3  movl    %ebx, %edx
4  not     %edx
5
6  # !A . !B
7  and     %ecx, %edx
8  movl    %edx, 16(%esp)
9
10 # !(A+B)
11 or      %ebx, %eax
12 notl   %eax
13 movl    %eax, 12(%esp)
```

Equivalente em C

```
1  n_op_ab = !(a | b);
2  n_a = !a;
3  n_b = !b;
4  n_op_a_b = n_a & n_b;
```

_second_make_table

Outputing

```
1  # print
2  movl  $true_table_fmt, (%esp)
3  call  printf
4
5  leave
6  ret
```

\$true_table_fmt

```
1 true_table_fmt:
2  .asciz "| %2d  %2d |  %2d  |  %2d  |\n"
```


Resultados

```
$ ./demorgan
```

```
True Tables from De Morgan Theorem.\n
Author: Pedro Garcia.
```

```
1th Thorem: !(A & B) == !A | !B
```

A	B	!(A.B)	!A + !B
0	0	-1	-1
0	-1	-1	-1
-1	0	-1	-1
-1	-1	0	0

```
2th Thorem: !(A | B) == !A & !B
```

A	B	!(A+B)	!A . !B
0	0	-1	-1
0	-1	0	0
-1	0	0	0
-1	-1	0	0

Observações, Considerações e Reclamações

Porque o atributo FALSE possui valor 0 e TRUE é -1 (ao invés de 1)?

demorgan.s

```

1     false:
2         .int 0
3
4     true:
5         .int -1

```

\$./demorgan

1th Theorem: $!(A \& B) == !A \mid !B$

A	B	!(A.B)	!A + !B
0	0	-1	-1
0	-1	-1	-1
-1	0	-1	-1
-1	-1	0	0

2th Theorem: $!(A \mid B) == !A \& !B$

A	B	!(A B)	!A . !B
0	0	-1	-1
0	-1	0	0
-1	0	0	0
-1	-1	0	0

Observações, Considerações e Reclamações

Porque o atributo FALSE possui valor 0 e TRUE é -1 (ao invés de 1)?

Complemento de 2: o sinal negativo é representado como “1” em binário. O resultado impresso pelo programa está em “decimal”.

Uma alternativa para remover o sinal negativo da saída seria utilizar os registradores de *flags* na hora da comparação. Na verdade, a melhor forma de fazermos comparação com resultado booleanos é utilizando estes registradores.

Exemplo

```
1    andl %edx, %eax
2    testl %eax, %eax
3    sete %al
4    movzbl %al, %eax
```

Observações, Considerações e Reclamações

Stacking

```
_first:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    ...

_second_make_table:
    pushl %ebp
    movl  %esp, %ebp
    subl  $64, %esp
    ...
```

Por que cada função começa com operações com o registrador %esp?

A chamada de funções é estruturada como uma pilha, o que permite indicar ponto de chamada e de retorno.

Observações, Considerações e Reclamações

Stacking

```
_first:
  pushl %ebp
  movl  %esp, %ebp
  subl  $8, %esp
  ...

_second_make_table:
  pushl %ebp
  movl  %esp, %ebp
  subl  $64, %esp
  ...
```

Por que os valores em cada função varia? (subl \$64, %esp, subl \$8, %esp)

Podemos imaginar que para cada função chamada, devemos alocar um espaço em memória para tal.

Diferentes funções neste programa estão usando quantidades diferentes de recursos em memória (em geral, para chamar as funções **puts** e **printf**). Enquanto uma precisa imprimir apenas uma variável, outras precisam imprimir até cinco, necessitando colocá-las na pilha para chamada da função de saída.

Dúvidas,
sugestões,
reclamações?

Referências

- IDOETA, I.V. *Elementos de Eletrônica Digital*. 35 ed. Érica. São Paulo, 1998. **ISBN 85-7194-019-3**.
- BLUM, R. *Professional Assembly Language*. Wiley Publishing. Indianápolis (EUA), 2005. **ISBN 0-7645-7901-0**.