Testando o Teorema de De Morgan

Pedro Garcia

http://www.sawp.com.br

04 de Fevereiro de 2010

Assembly Working Party

Laboratório de Cálculos Científicos, Instituto de Física, Universidade de Brasília, Brasil

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.

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

Segundo teorema de De Morgan.

Definition

"O complemento da soma é igual ao produto dos complementos."

Demonstração.

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

$\square$

Segundo teorema de De Morgan.

Definition

"O complemento da soma é igual ao produto dos complementos."

Demonstração.

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

$\square$

Segundo teorema de De Morgan.

Definition

"O complemento da soma é igual ao produto dos complementos."

Demonstração.

1. Do 1° 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 $B = \overline{Y}$

□

Segundo teorema de De Morgan.

Definition

"O complemento da soma é igual ao produto dos complementos."

Demonstração.

1. Do 1° 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: $A \cdot B = \overline{X} \cdot \overline{Y}$

□

Segundo teorema de De Morgan.

Definition

"O complemento da soma é igual ao produto dos complementos."

Demonstração.

1. Do 1° 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. $\overline{\overline{A} + \overline{B}} = \overline{X + Y}$

□

Segundo teorema de De Morgan.

Definition

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

Demonstração.

1. Do 1° 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. $\overline{\overline{A} + \overline{B}} = \overline{X + Y}$
6. Portanto, $\overline{X} \cdot \overline{Y} = (\overline{X + Y})$

$\square$

Segundo teorema de De Morgan.

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

C

Ilustrando *De Morgan* em C

```
int main(void) {
    int a, b, n_op_ab, n_op_a_b, n_a, n_b;

    /* primeiro teorema de De Morgan */
    for (a = 0; a <= 1; a++)
      for (b = 0; b <= 1; b++) {
        n_op_ab = !(a & b);
        n_a = !a;
        n_b = !b;
        n_op_a_b = n_a | n_b;
        n_op_ab == n_op_a_b; /* should tb TRUE, 1 */
      }

    /* segundo teorema de De Morgan */
    for (a = 0; a <= 1; a++)
      for (b = 0; b <= 1; b++) {
        n_op_ab = !(a | b);
        n_a = !a;
        n_b = !b;
        n_op_a_b = n_a & n_b;
        n_op_ab == n_op_a_b; /* should tb TRUE, 1 */
    }
}
```

Implementação em ASM

Compile com

gcc -g demorgan.s -o demorgan

demorgan.s – Programa principal.

```
main:
    pushl   %ebp
    movl    %esp, %ebp

    # stt: show trivial things
    call  _preamble

    # 1th De Morgan Theorem
    call  _first

    # 2th De Morgan Theorem
    call  _second

    leave
    ret
```

preamble

_preamble

```
_preamble:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp

    movl    $head, (%esp)
    call    puts

    leave
    ret
```

$head

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

_first

Cabeçalho da função

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

    movl    $first_demorgan, (%esp)
    call    puts

    movl    $first_true_table_head, (%esp)
    call    printf
```

$first_demorgan

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

$first_true_table_head

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

_first

primeiro teorema de De Morgan

```
    movl    false, %eax
    movl    false, %ebx
    call _first_make_table

    movl    false, %eax
    movl    true, %ebx
    call    _first_make_table

    movl    true, %eax
    movl    false, %ebx
    call    _first_make_table

    movl    true, %eax
    movl    true, %ebx
    call    _first_make_table
```

Equivalente em C

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

_first_make_table

Inserindo a função na pilha

```
_first_make_table:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $64, %esp

    movl    %eax, 4(%esp)
    movl    %ebx, 8(%esp)
```

_first_make_table

Processando o Primeiro Teorema de De Morgan

```
movl    %eax, %ecx
not     %ecx
movl    %ebx, %edx
not     %edx

# !A + !B
or      %ecx, %edx
movl    %edx, 16(%esp)

# !(A.B)
and     %ebx, %eax
notl    %eax
movl    %eax, 12(%esp)
```

Equivalente em C

```
n_op_ab = !(a & b);
n_a = !a;
n_b = !b;
n_op_a_b = n_a | n_b;
```

_first_make_table

Outputing

```
    # print
    movl    $true_table_fmt, (%esp)
    call    printf

    leave
    ret
```

$true_table_fmt

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

_second

Cabeçalho da função

```
_second:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp

    movl    $second_demorgan, (%esp)
    call    puts

    movl    $second_true_table_head, (%esp)
    call    printf
```

$second_demorgan

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

$second_true_table_head

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

_second

segundo teorema de De Morgan

```
movl    false, %eax
movl    false, %ebx
call _second_make_table

movl    false, %eax
movl    true, %ebx
call    _second_make_table

movl    true, %eax
movl    false, %ebx
call    _second_make_table

movl    true, %eax
movl    true, %ebx
call    _second_make_table
```

Equivalente em C

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

_second_make_table

Inserindo a função na pilha

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

    movl    %eax, 4(%esp)
    movl    %ebx, 8(%esp)
```

_second_make_table

Processando o Segundo Teorema de De Morgan

```
movl    %eax, %ecx
not     %ecx
movl    %ebx, %edx
not     %edx

# !A . !B
and     %ecx, %edx
movl    %edx, 16(%esp)

# !(A+B)
or      %ebx, %eax
notl    %eax
movl    %eax, 12(%esp)
```

Equivalente em C

```
n_op_ab = !(a | b);
n_a = !a;
n_b = !b;
n_op_a_b = n_a & n_b;
```

_second_make_table

Outputing

```
    # print
    movl    $true_table_fmt, (%esp)
    call    printf

    leave
    ret
```

$true_table_fmt

```
true_table_fmt:
    .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 Theorem: !(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

```
        false:
            .int 0

        true:
            .int -1
```

$ ./demorgan

```
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 Theorem: !(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)?

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

```
	andl   %edx, %eax
	testl  %eax, %eax
	sete   %al
	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**.