

UNIOESTE
Ciência da Computação

Sistemas Digitais
Circuitos Combinacionais

Prof. Jorge Habib El Khouri
Prof. Antonio Marcos Hachisuca

2020/2021

Referências Bibliográficas

1. *Digital Fundamentals*, Thomas L. Floyd; Editora: Pearson; Edição: 11; Ano: 2015;
2. *Sistemas Digitais Princípios e Aplicações*, Ronald J. Tocci; Editora: Pearson; Edição: 11; Ano: 2011;
3. *Computer Organization and Design*, David A. Patterson; Editora: Elsevier; Edição: 1; Ano: 2017
4. *Digital Design: Principles and Practices*, John F. Wakerly; Editora: Pearson; Edição: 5; Ano: 2018;
5. *Guide to Assembly Language Programming in Linux*, Sivarama P. Dandamudi; Editora: Springer; Edição: 1; Ano: 2005.

Sumário

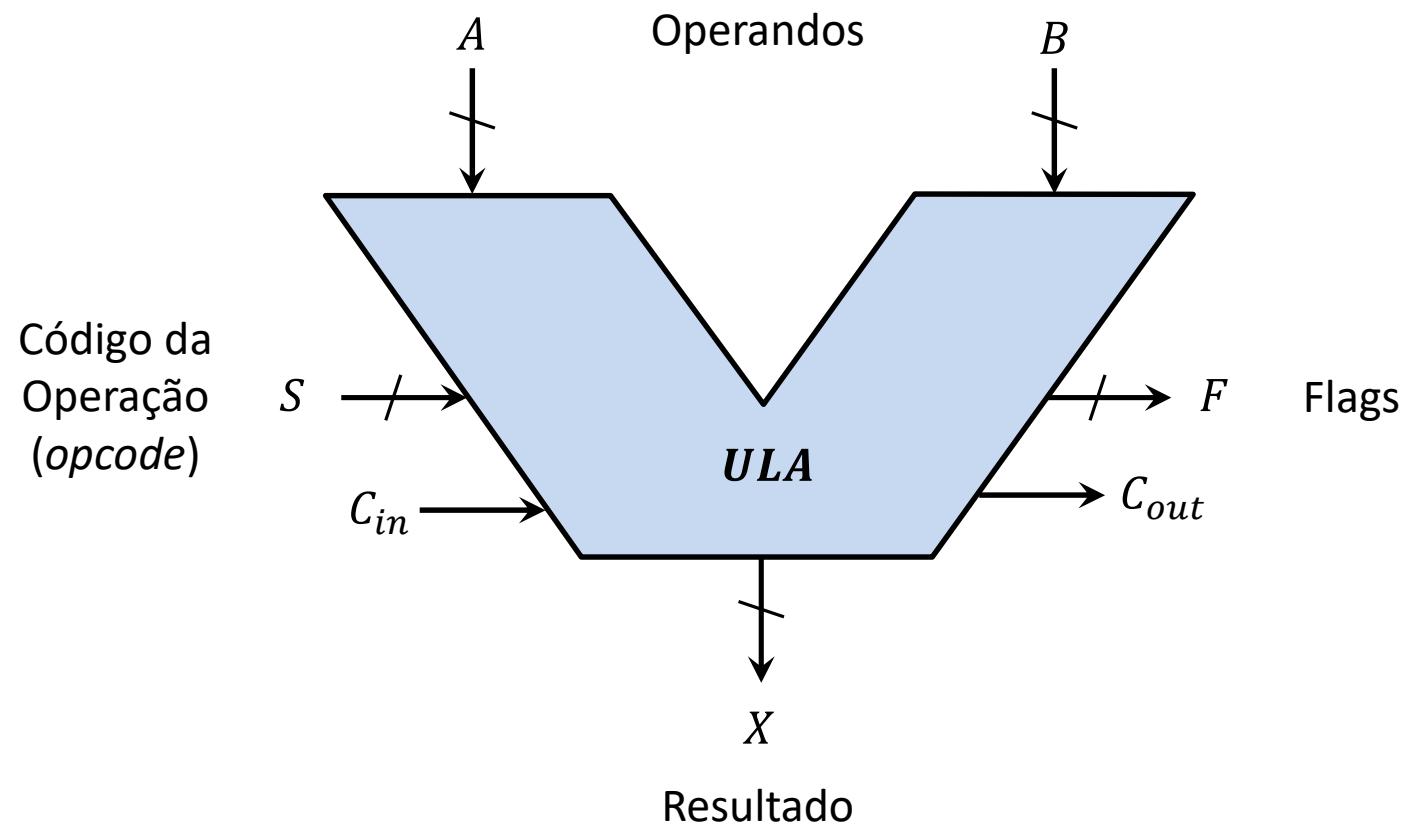
- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

Unidade de Lógica e Aritmética

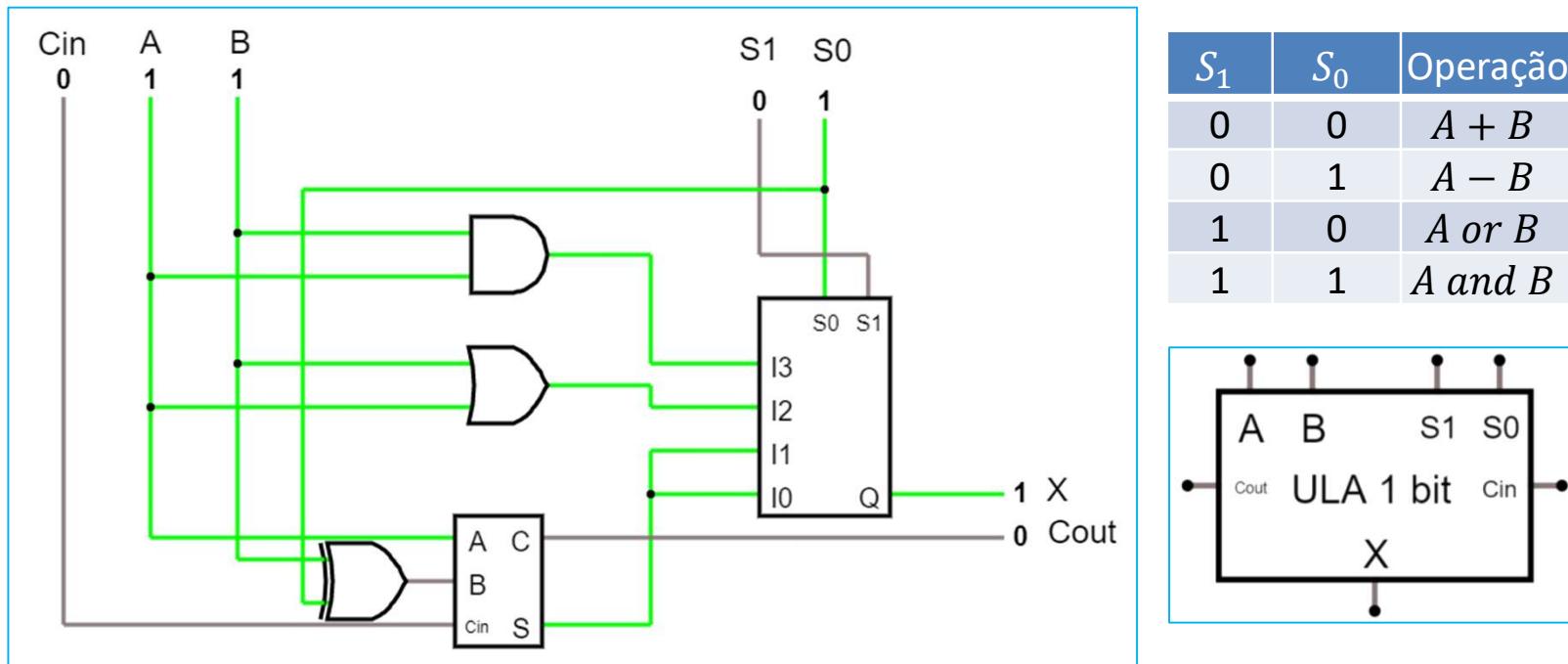
- A *ULA – Unidade de Lógica e Aritmética* é componente central de um processador;
- É onde concentram os circuitos que realizam as operações aritméticas e lógicas básicas com inteiros, tais como: *adição, subtração, and, or* e outras;
- A *adição* é a operação mais importante de uma ULA, pois dela podem ser derivadas a *subtração, multiplicação* e *divisão*;
- Em função da representação em *Complemento de 2*, a soma e a subtração podem compartilhar da mesma lógica, com pequenas adaptações nos bits de entrada;
- Como estratégia geral, as operações são feitas simultaneamente, e um *MUX* seleciona a saída desejada;

Unidade de Lógica e Aritmética



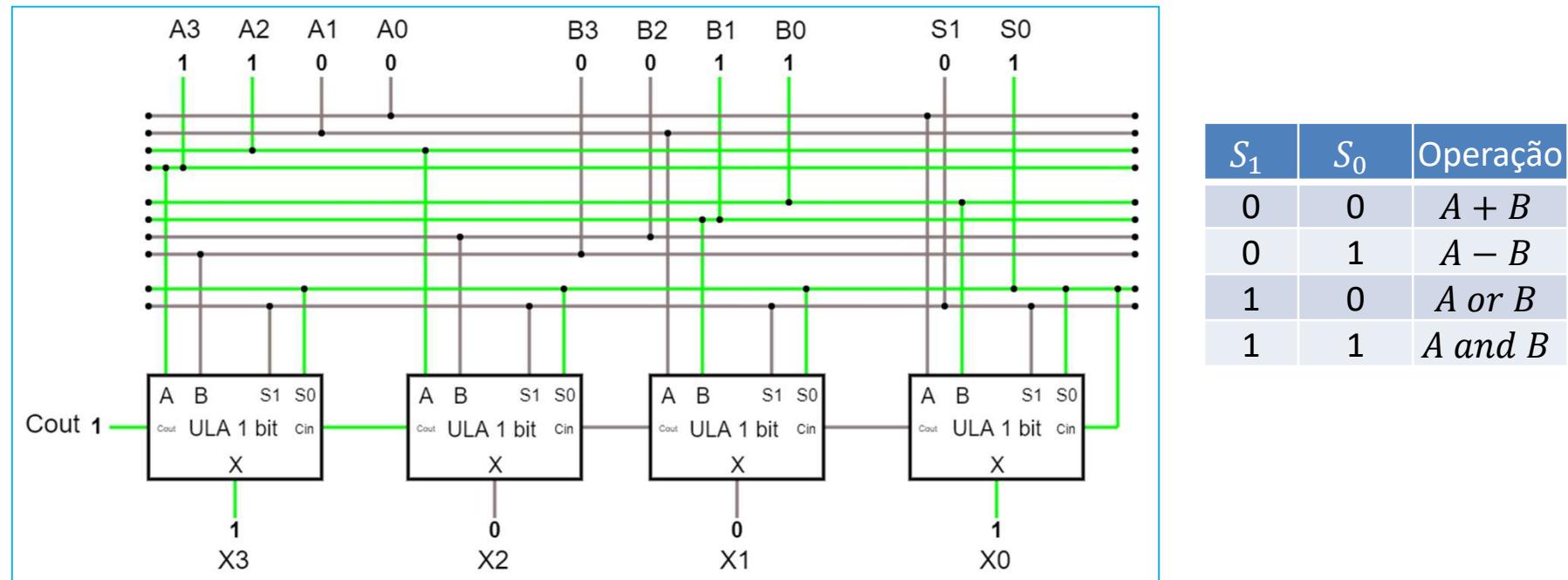
Unidade de Lógica e Aritmética

- A seguinte *ULA* tem capacidade de 1 bit para as operações de *adição*, *subtração*, *and* e *or*:

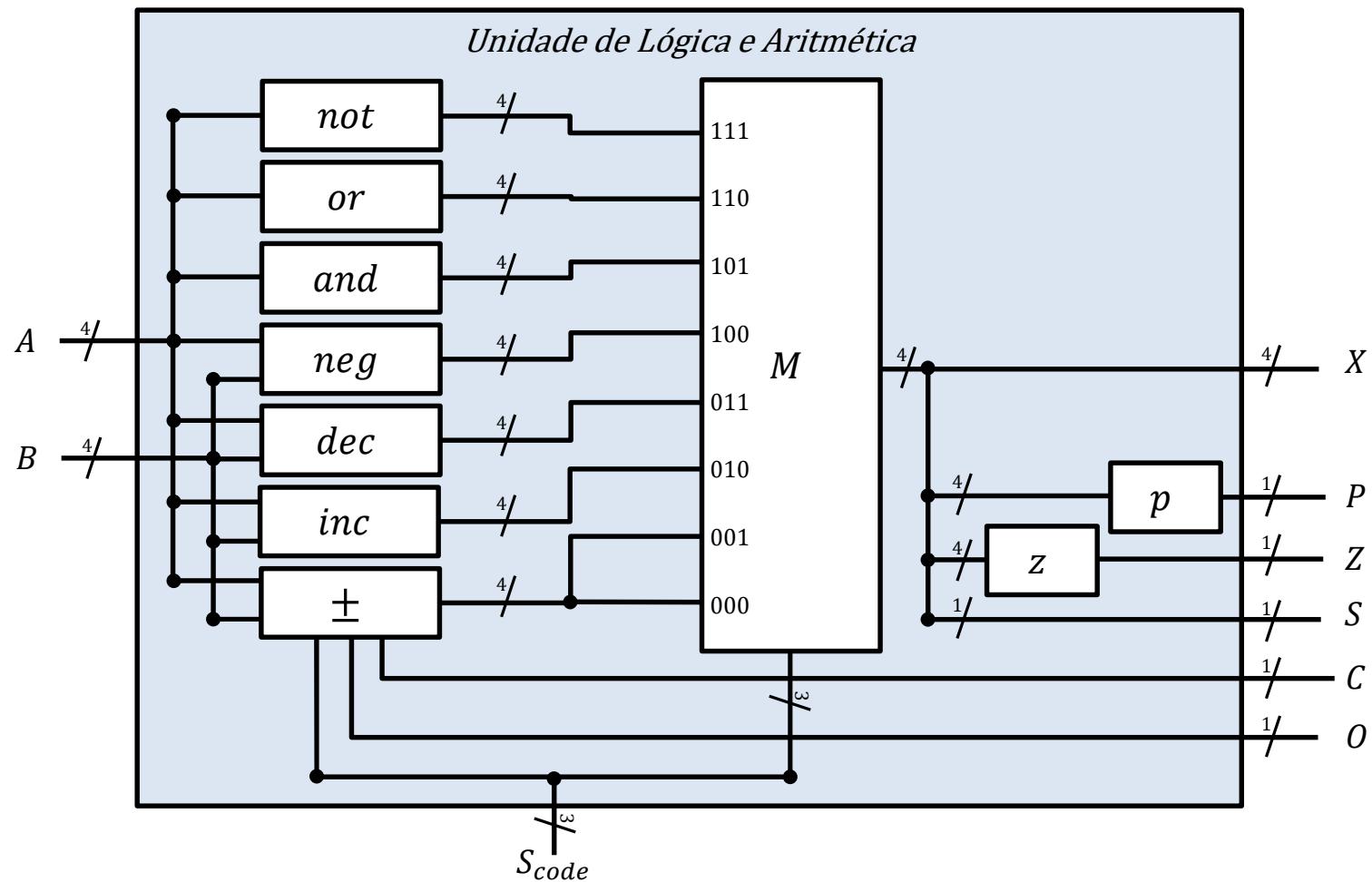


Unidade de Lógica e Aritmética

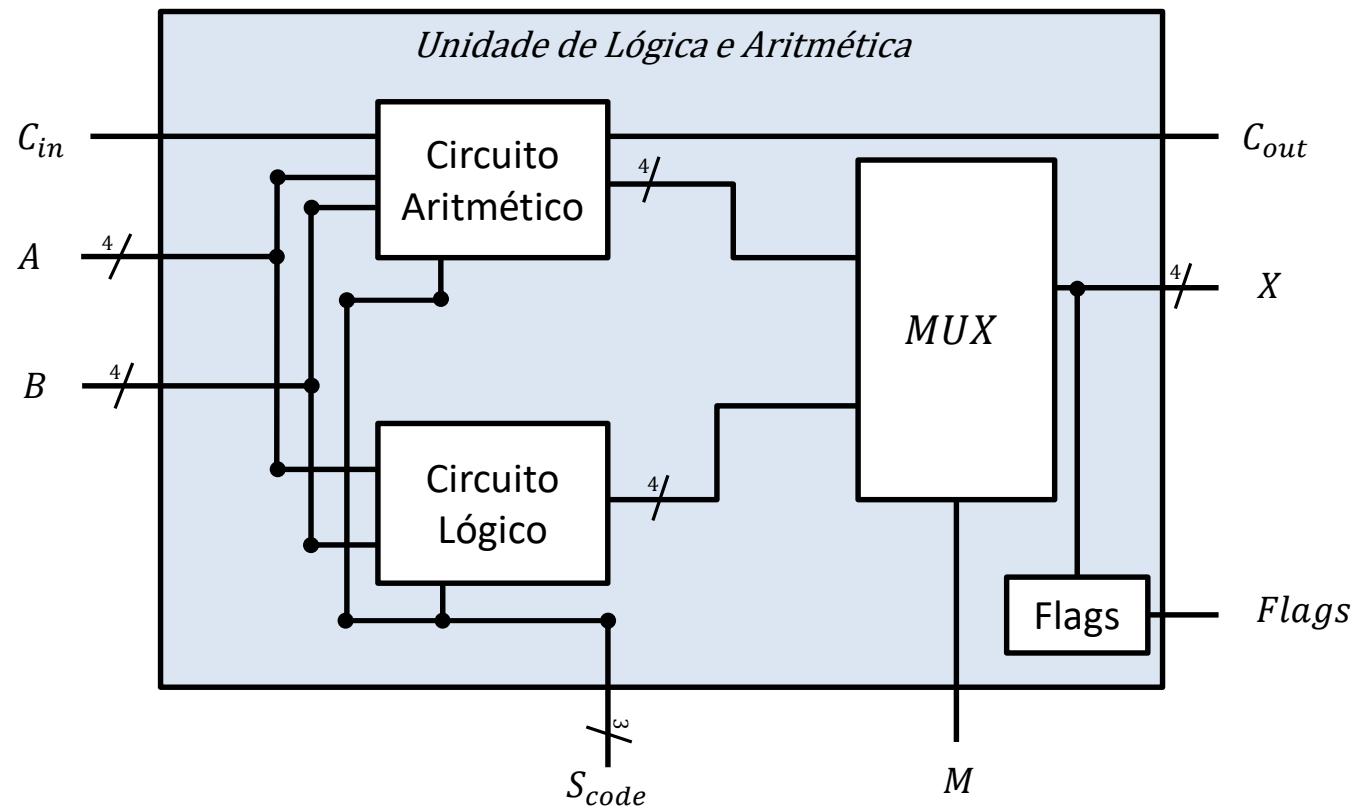
- Uma *ULA* de n bits pode ser formada pela combinação de n *ULAs* de 1 bit:



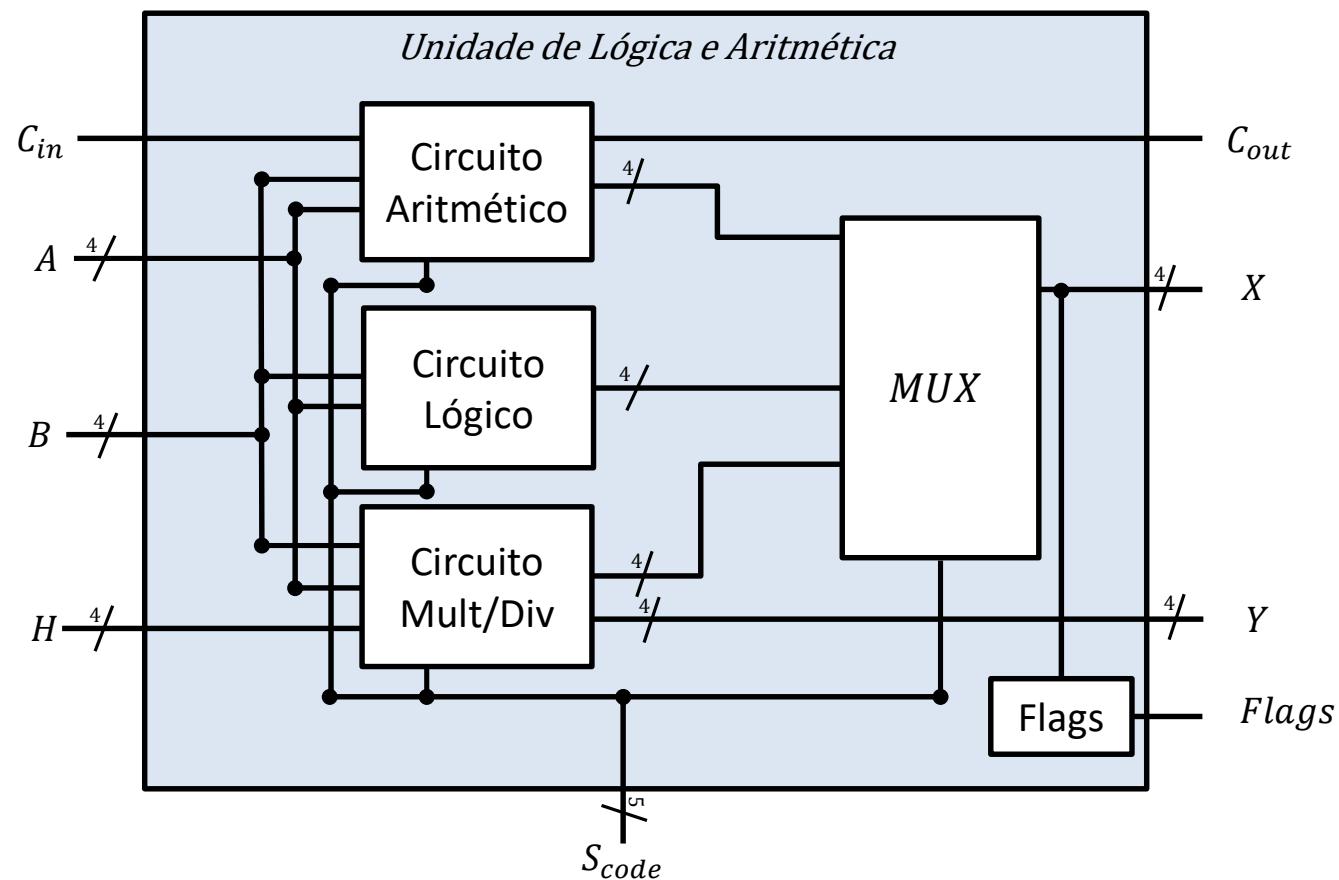
Unidade de Lógica e Aritmética



Unidade de Lógica e Aritmética



Unidade de Lógica e Aritmética



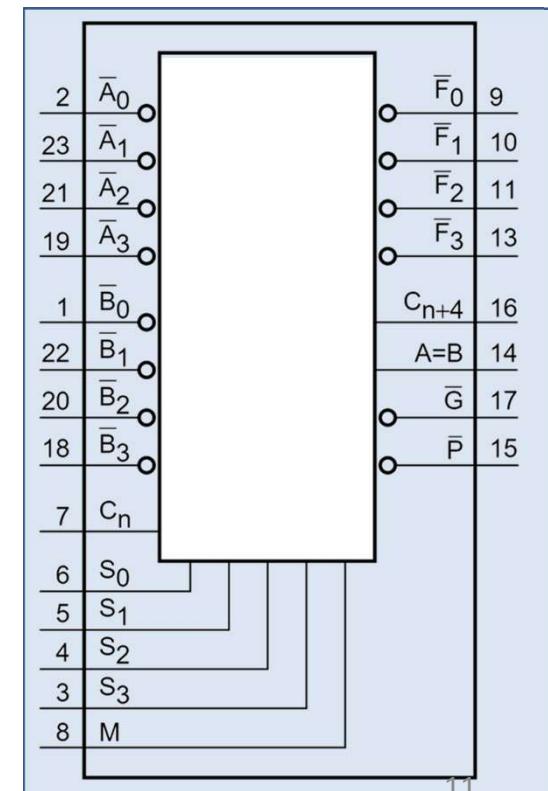
$$\begin{aligned} YX &= A \times B \\ X &= HA/B \\ Y &= HA\%B \end{aligned}$$

Unidade de Lógica e Aritmética

- 4-bit Arithmetic Logic Unit 74HC181:

- O circuito integrado 74HC181 é uma *ULA* de 4 bits com capacidade para 16 operações aritméticas e 16 operações lógicas;
- É da família 7400 de *CIs*;
- Representou um importante passo na construção de computadores nos anos 60;
- Primeira *ULA* em um único circuito integrado;
- Utilizou tecnologia de portas lógicas;
- O circuito tem o equivalente a aproximadamente 75 portas lógicas;
- Na sequência a tabela com as operações e o circuito lógico equivalente.

<https://datasheetspdf.com/pdf-file/491578/Philips/74HC181/1>



Unidade de Lógica e Aritmética

- 4-bit Arithmetic Logic Unit 74HC181

https://datasheetspdf.com/pdf-file/491578/Philips/74HC181_1

| FUNCTION TABLES | | | | | |
|--------------------|-------|-------|-------|--------------------------------|---|
| MODE SELECT INPUTS | | | | ACTIVE HIGH INPUTS AND OUTPUTS | |
| S_3 | S_2 | S_1 | S_0 | LOGIC (M=H) | ARITHMETIC ⁽²⁾ (M=L; $C_n=H$) |
| L | L | L | L | \bar{A} | A |
| L | L | L | H | $A + B$ | $A + B$ |
| L | L | H | L | $\bar{A}B$ | $A + \bar{B}$ |
| L | L | H | H | logical 0 | minus 1 |
| L | H | L | L | $\bar{A}\bar{B}$ | $A + \bar{A}\bar{B}$ |
| L | H | L | H | \bar{B} | $(A + B) + \bar{A}\bar{B}$ |
| L | H | H | L | $A \oplus B$ | $A - B - 1$ |
| L | H | H | H | $\bar{A}\bar{B}$ | $\bar{A}\bar{B} - 1$ |
| H | L | L | L | $\bar{A} + B$ | $A + AB$ |
| H | L | L | H | $\bar{A} \oplus \bar{B}$ | $A + B$ |
| H | L | H | L | B | $(A + \bar{B}) + AB$ |
| H | L | H | H | AB | $AB - 1$ |
| H | H | L | L | logical 1 | $A + A^{(1)}$ |
| H | H | L | H | $A + \bar{B}$ | $(A + B) + A$ |
| H | H | H | L | $A + B$ | $(A + \bar{B}) + A$ |
| H | H | H | H | A | $A - 1$ |

| FUNCTION TABLES | | | | | |
|--------------------|-------|-------|-------|-------------------------------|---|
| MODE SELECT INPUTS | | | | ACTIVE LOW INPUTS AND OUTPUTS | |
| S_3 | S_2 | S_1 | S_0 | LOGIC (M=H) | ARITHMETIC ⁽²⁾ (M=L; $C_n=L$) |
| L | L | L | L | \bar{A} | $A - 1$ |
| L | L | L | H | $\bar{A}\bar{B}$ | $AB - 1$ |
| L | L | H | L | $\bar{A} + B$ | $\bar{A}\bar{B} - 1$ |
| L | L | H | H | logical 1 | minus 1 |
| L | H | L | L | $\bar{A} + \bar{B}$ | $A + (A + \bar{B})$ |
| L | H | L | H | \bar{B} | $AB + (A + \bar{B})$ |
| L | H | H | L | $\bar{A} \oplus \bar{B}$ | $A - B - 1$ |
| L | H | H | H | $A + \bar{B}$ | $A + \bar{B}$ |
| H | L | L | L | $\bar{A}\bar{B}$ | $A + (A + B)$ |
| H | L | L | H | $A \oplus B$ | $A + B$ |
| H | L | H | L | B | $\bar{A}\bar{B} + (A + B)$ |
| H | L | H | H | $A + B$ | $A + B$ |
| H | H | L | L | logical 0 | $A + A^{(1)}$ |
| H | H | L | H | $\bar{A}B$ | $AB + A$ |
| H | H | H | L | $\bar{A}B$ | $\bar{A}\bar{B} + A$ |
| H | H | H | H | A | A |

Notes to the function tables

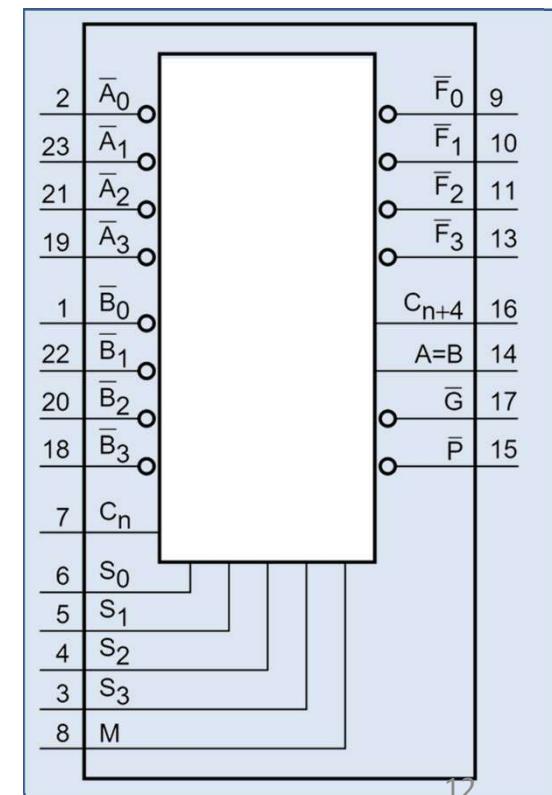
- Each bit is shifted to the next more significant position.
- Arithmetic operations expressed in 2s complement notation.

H = HIGH voltage level
L = LOW voltage level

Notes to the function tables

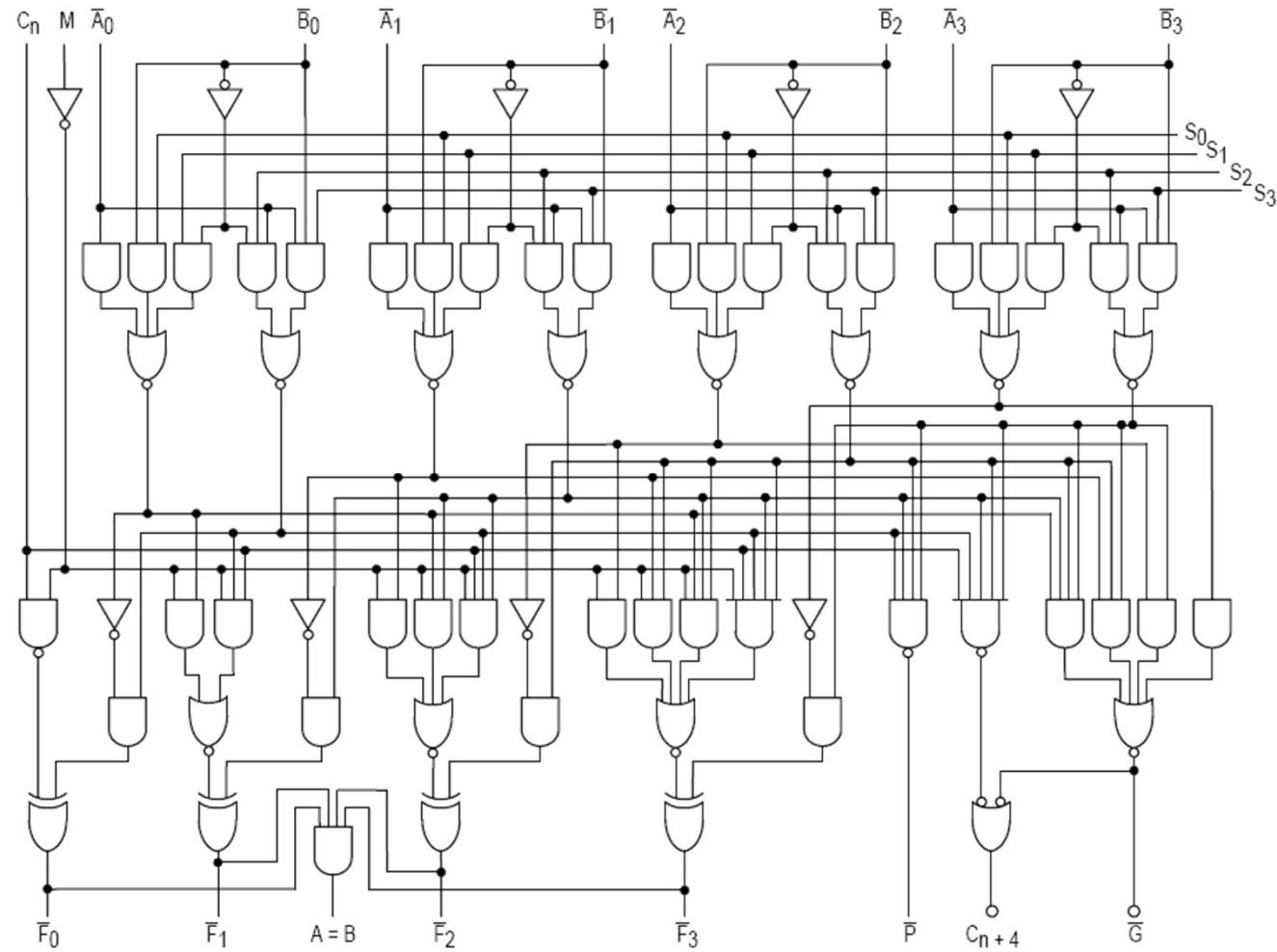
- Each bit is shifted to the next more significant position.
- Arithmetic operations expressed in 2s complement notation.

H = HIGH voltage level
L = LOW voltage level



12

Unidade de Lógica e Aritmética



<https://commons.wikimedia.org/wiki/File:74181aluschematic.png>

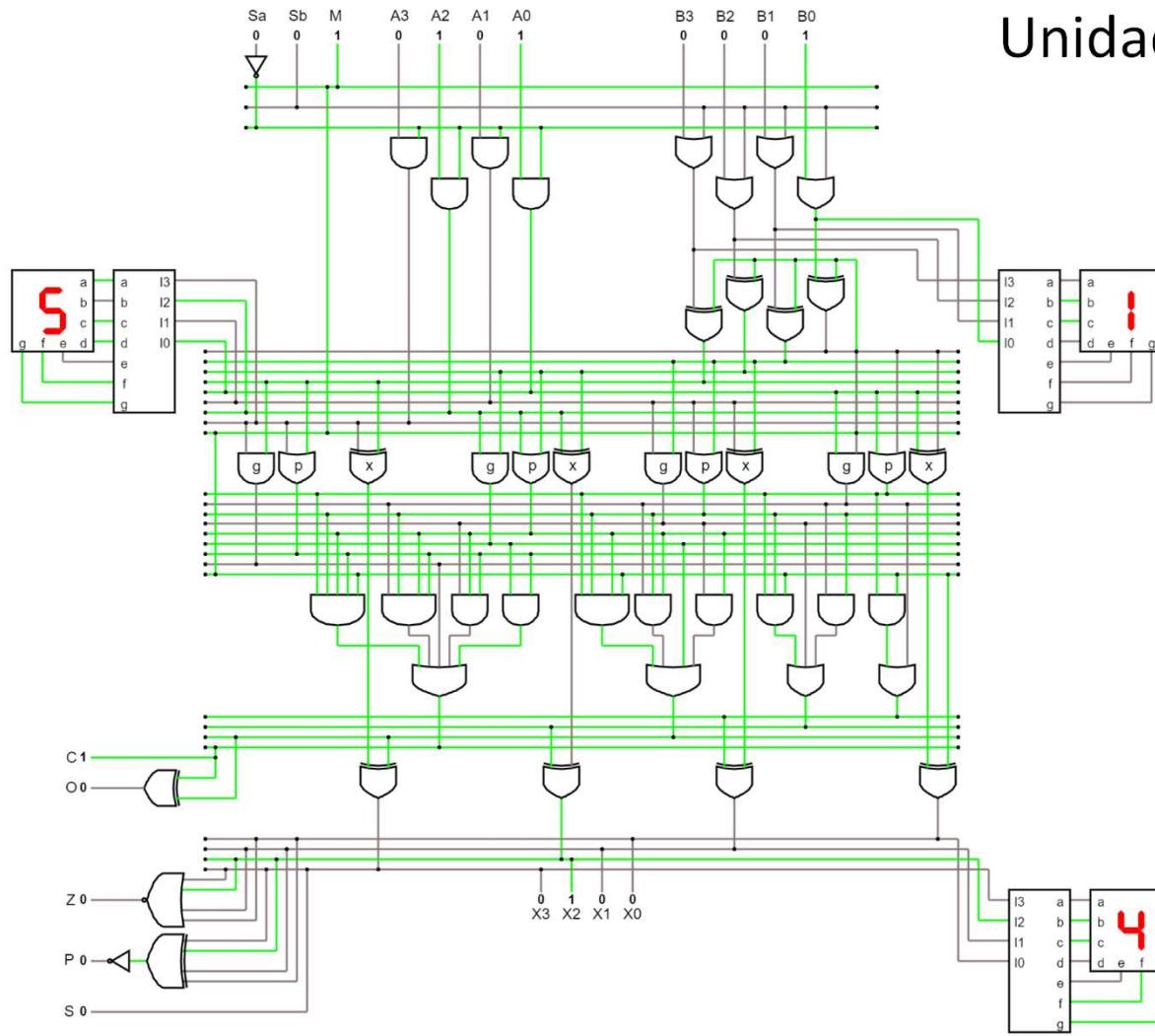
Unidade de Lógica e Aritmética

add como add/sub/neg/inc/dec

- O *Círculo Aritmético* pode fazer uso do poder da operação de *adição*, bem como sua flexibilidade em fornecer diversas funcionalidades;
- Esta tarefa central demanda um circuito de adição otimizado, tal como o uso da estratégia *Look Ahead Carry*;
- Seguem as operações que podem ser facilmente derivadas da adição:

| | | |
|------------|---|----------------|
| <i>add</i> |  | $X = A + B$ |
| <i>sub</i> |  | $X = A + (-B)$ |
| <i>dec</i> |  | $X = A + (-1)$ |
| <i>inc</i> |  | $X = A + 1$ |
| <i>neg</i> |  | $X = 0 + (-B)$ |

$X = \begin{bmatrix} A \\ 0 \end{bmatrix} + \begin{bmatrix} B \\ -B \\ 1 \\ -1 \end{bmatrix}$



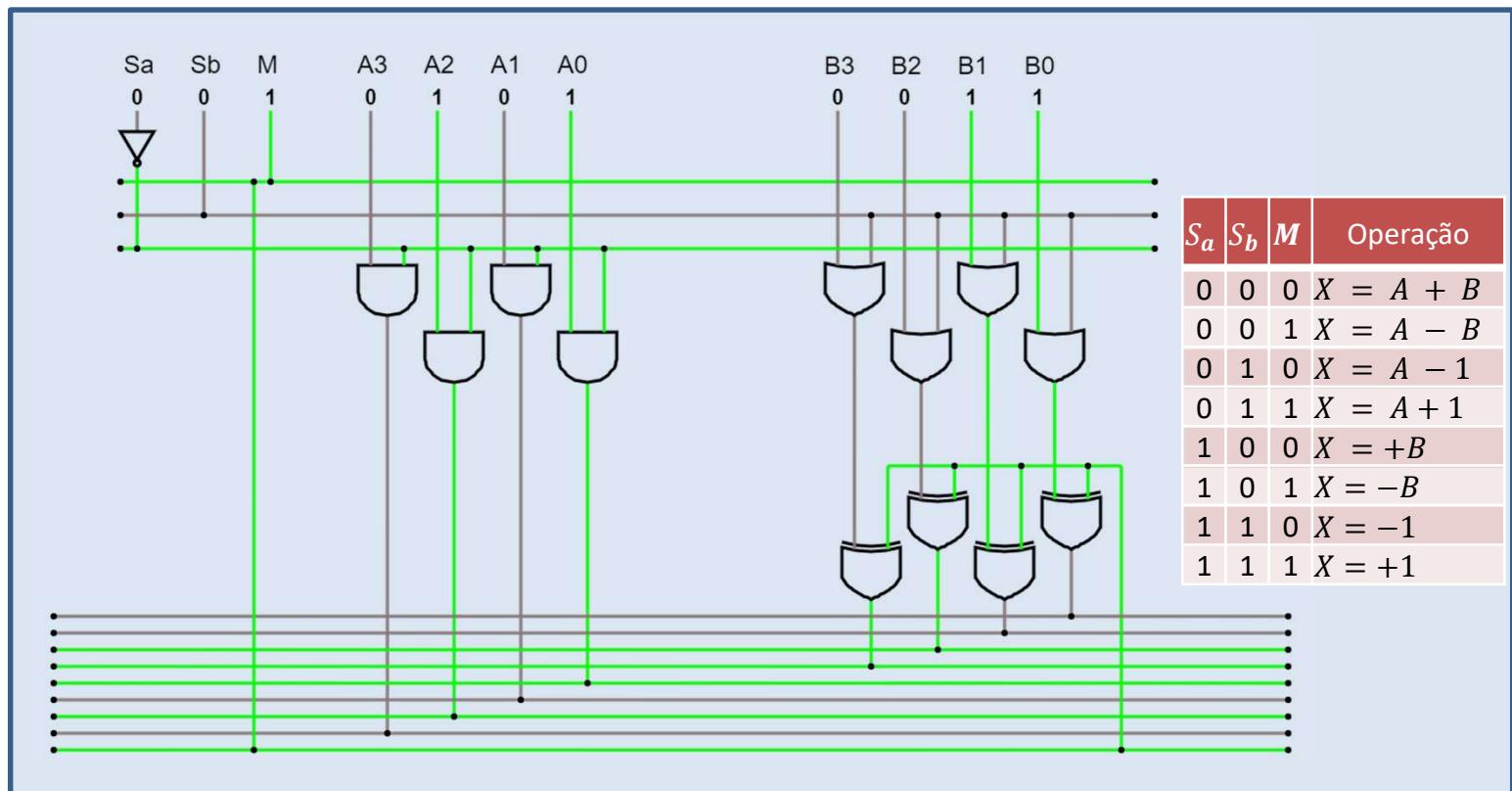
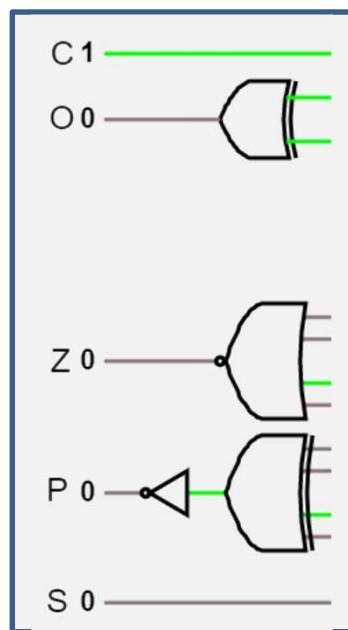
Unidade de Lógica e Aritmética

add como add/sub/neg/inc/dec

| S_a | S_b | M | Operação |
|-------|-------|-----|-------------|
| 0 | 0 | 0 | $X = A + B$ |
| 0 | 0 | 1 | $X = A - B$ |
| 0 | 1 | 0 | $X = A - 1$ |
| 0 | 1 | 1 | $X = A + 1$ |
| 1 | 0 | 0 | $X = +B$ |
| 1 | 0 | 1 | $X = -B$ |
| 1 | 1 | 0 | $X = -1$ |
| 1 | 1 | 1 | $X = 1$ |

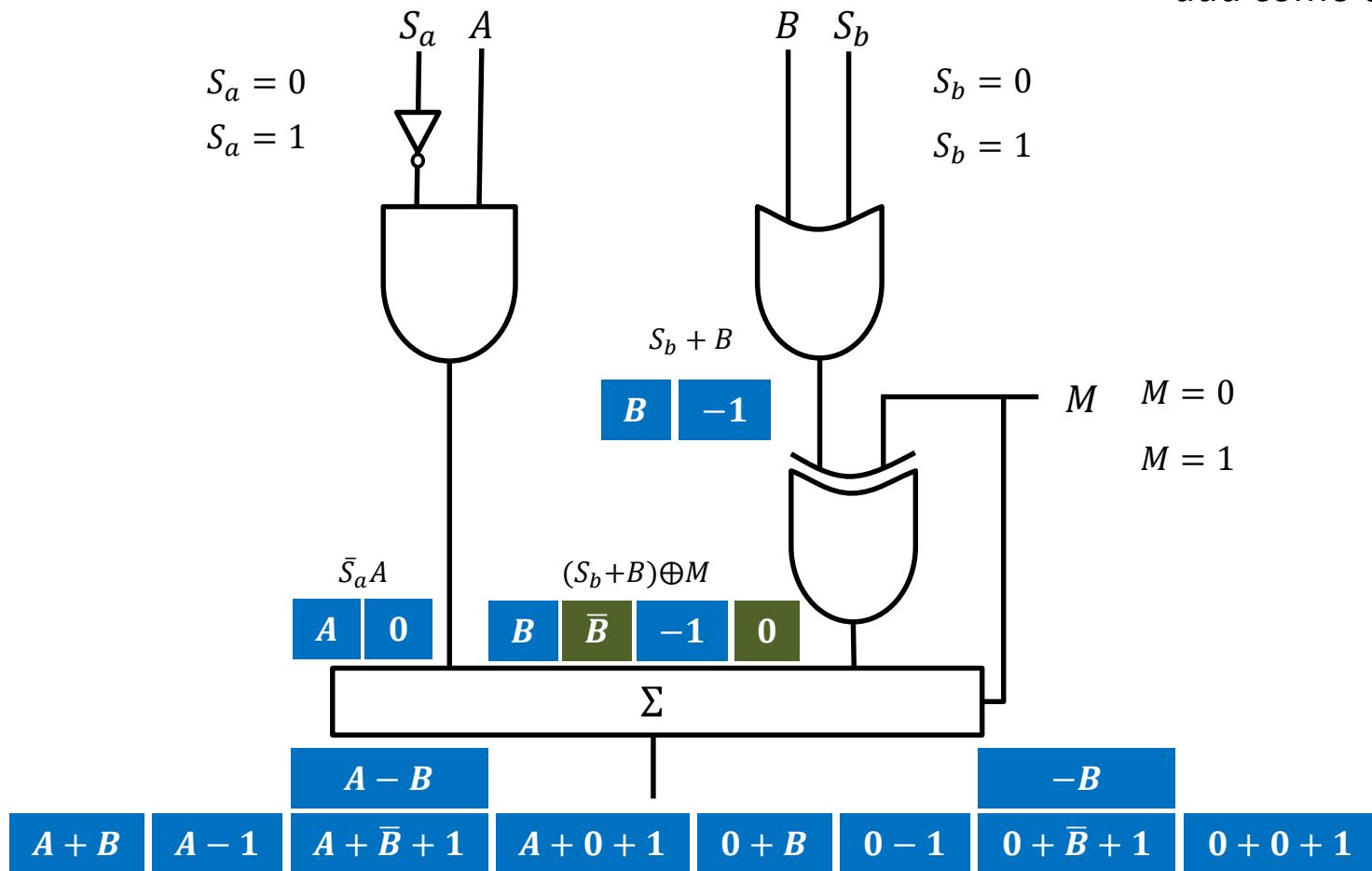
Unidade de Lógica e Aritmética

add como add/sub/neg/inc/dec



Unidade de Lógica e Aritmética

add como add/sub/neg/inc/dec



| S_a | S_b | M | Operação |
|-------|-------|-----|-------------|
| 0 | 0 | 0 | $X = A + B$ |
| 0 | 0 | 1 | $X = A - B$ |
| 0 | 1 | 0 | $X = A - 1$ |
| 0 | 1 | 1 | $X = A + 1$ |
| 1 | 0 | 0 | $X = +B$ |
| 1 | 0 | 1 | $X = -B$ |
| 1 | 1 | 0 | $X = -1$ |
| 1 | 1 | 1 | $X = +1$ |

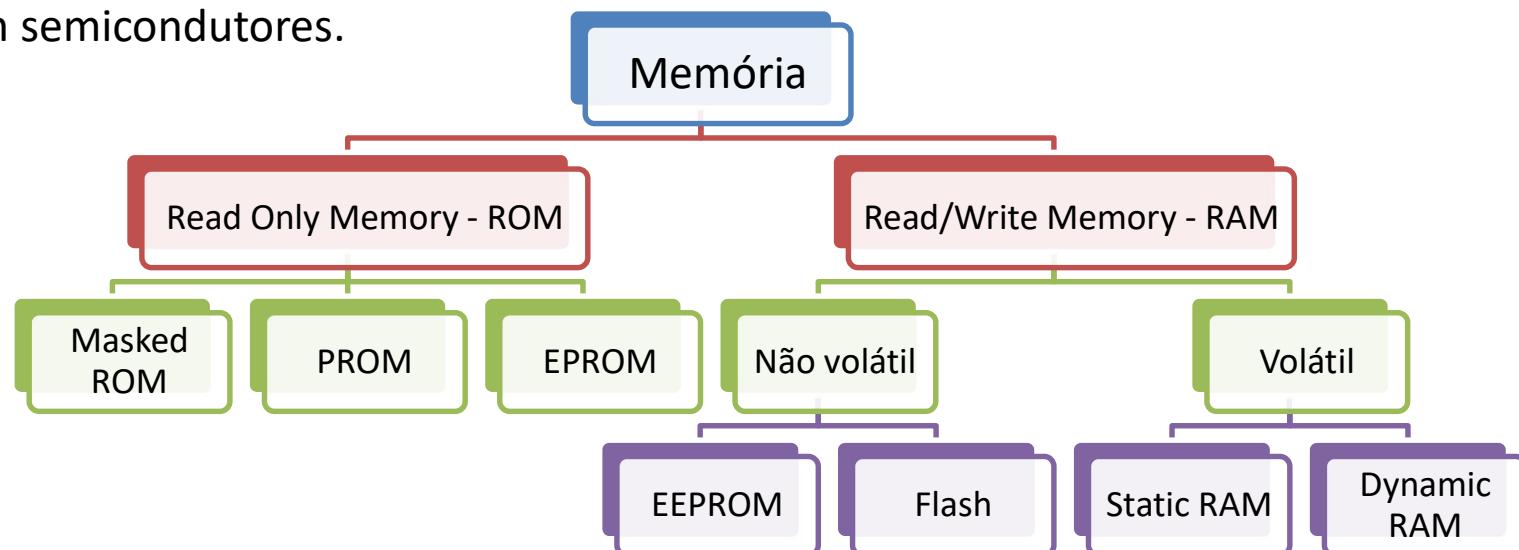
Sumário

- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

Read-Only Memory *ROM*

- A memória é componente fundamental de um sistema de processamento;
- É um dispositivo de armazenamento e recuperação de informações;
- O seguinte diagrama apresenta de forma simplificada os tipos de memória baseados em semicondutores.

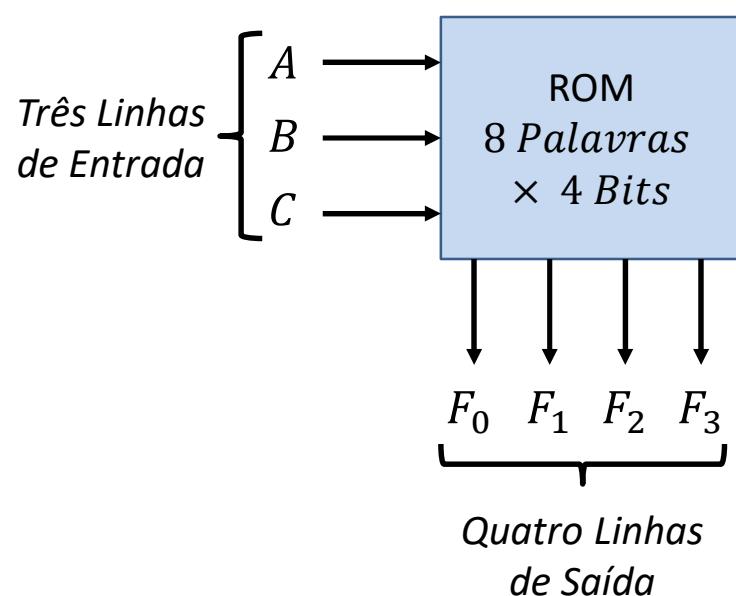


Adaptado de https://www.researchgate.net/figure/Classification-of-semiconductor-memories-Devices-suitable-for-the-realization-of-a_fig4_29528840

Read-Only Memory *ROM*

- Uma *ROM* – *Read-Only Memory* consiste em uma matriz de dispositivos semicondutores que são interconectados para armazenar um array de dados binários.
- Uma vez com os dados binários armazenados na *ROM*, eles podem ser lidos sempre que necessário, mas não podem ser alterados em condições normais de operação;
- A seguinte figura mostra uma *ROM* com três linhas de entrada e quatro linhas de saída.
- Uma tabela verdade pode relacionar as entradas e saídas da *ROM*;

Read-Only Memory ROM



| A | B | C | F ₀ | F ₁ | F ₂ | F ₃ |
|---|---|---|----------------|----------------|----------------|----------------|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |

*Dados típicos armazenados na ROM
 2^3 palavras x 4 bits*

Read-Only Memory *ROM*

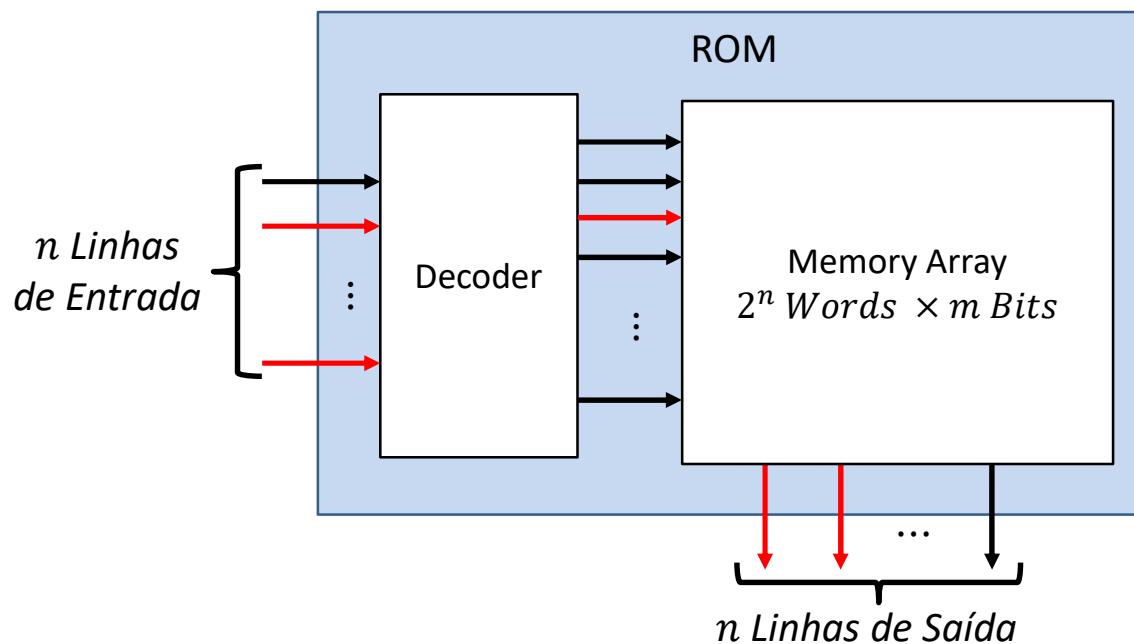
- Para cada combinação de valores informados nas três linhas de entrada, um padrão correspondente de 0 e 1 saem nas linhas de saída da *ROM*;
- Por exemplo, se a combinação $ABC = 010$ for aplicada às linhas de entrada, o padrão $F_0F_1F_2F_3 = 0111$ aparece nas linhas de saída;
- Cada um dos padrões de saída armazenados na *ROM* é chamado de *palavra*;
- Como a *ROM* possui três linhas de entrada, temos $2^3 = 8$ combinações diferentes de valores de entrada;
- Cada combinação de entrada representa um endereço, habilitando a seleção de uma das 8 palavras armazenadas na memória;

Read-Only Memory *ROM*

- As quatro linhas de saída definem uma palavra de quatro bits, sendo o tamanho desta *ROM* de $8 \text{ palavras} \times 4 \text{ bits}$;
- Uma *ROM* com n linhas de entrada e m linhas de saída é formada por uma matriz de 2^n palavras, sendo cada palavra com m bits de comprimento;
- Os tamanhos típicos de *ROMs* disponíveis comercialmente variam de $32 \text{ palavras} \times 4 \text{ bits}$ a $512K \text{ palavras} \times 8 \text{ bits}$ ou superiores;

Read-Only Memory *ROM*

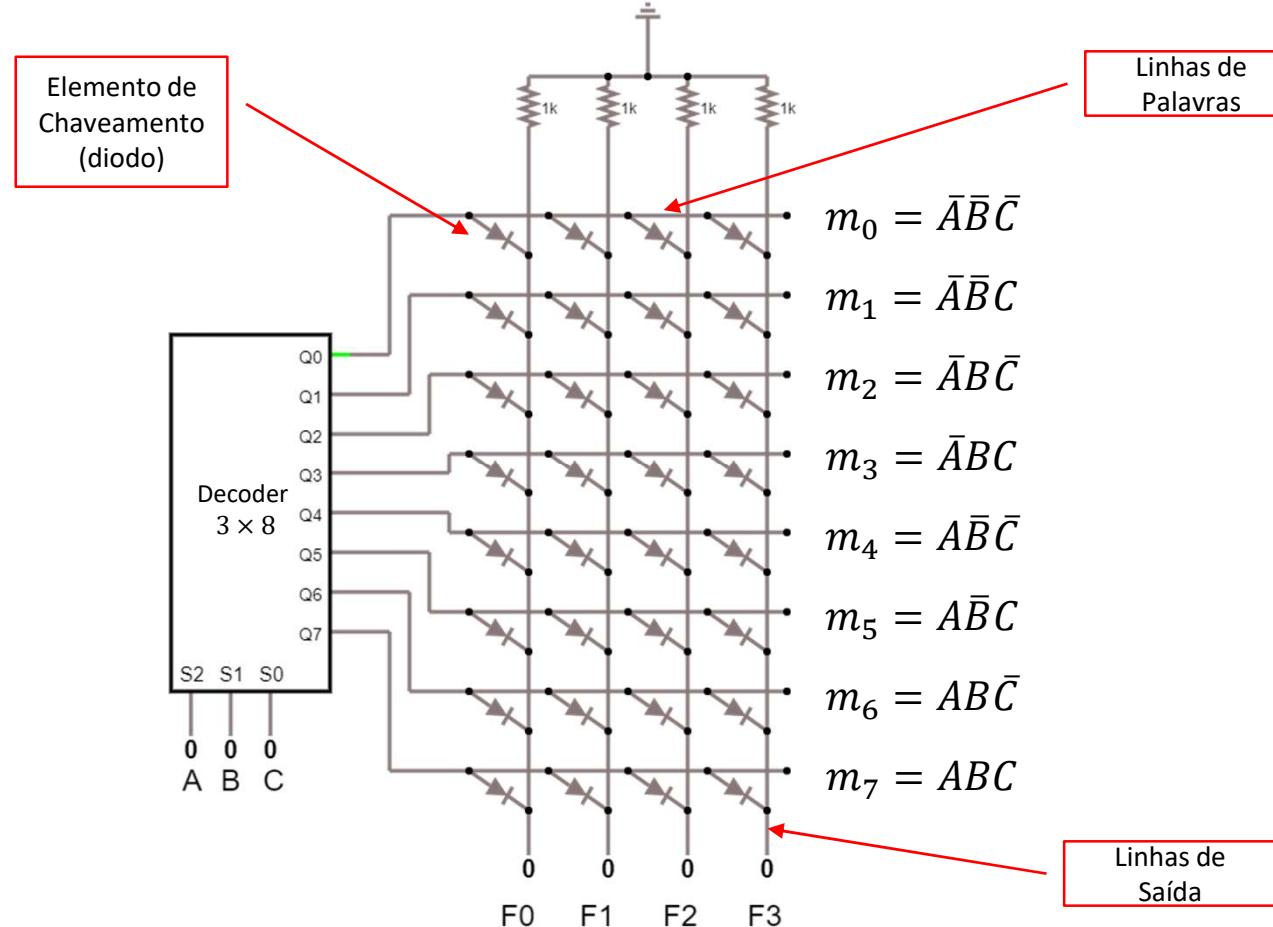
- Uma *ROM* consiste basicamente de um *decodificador* e um *array de memória*;



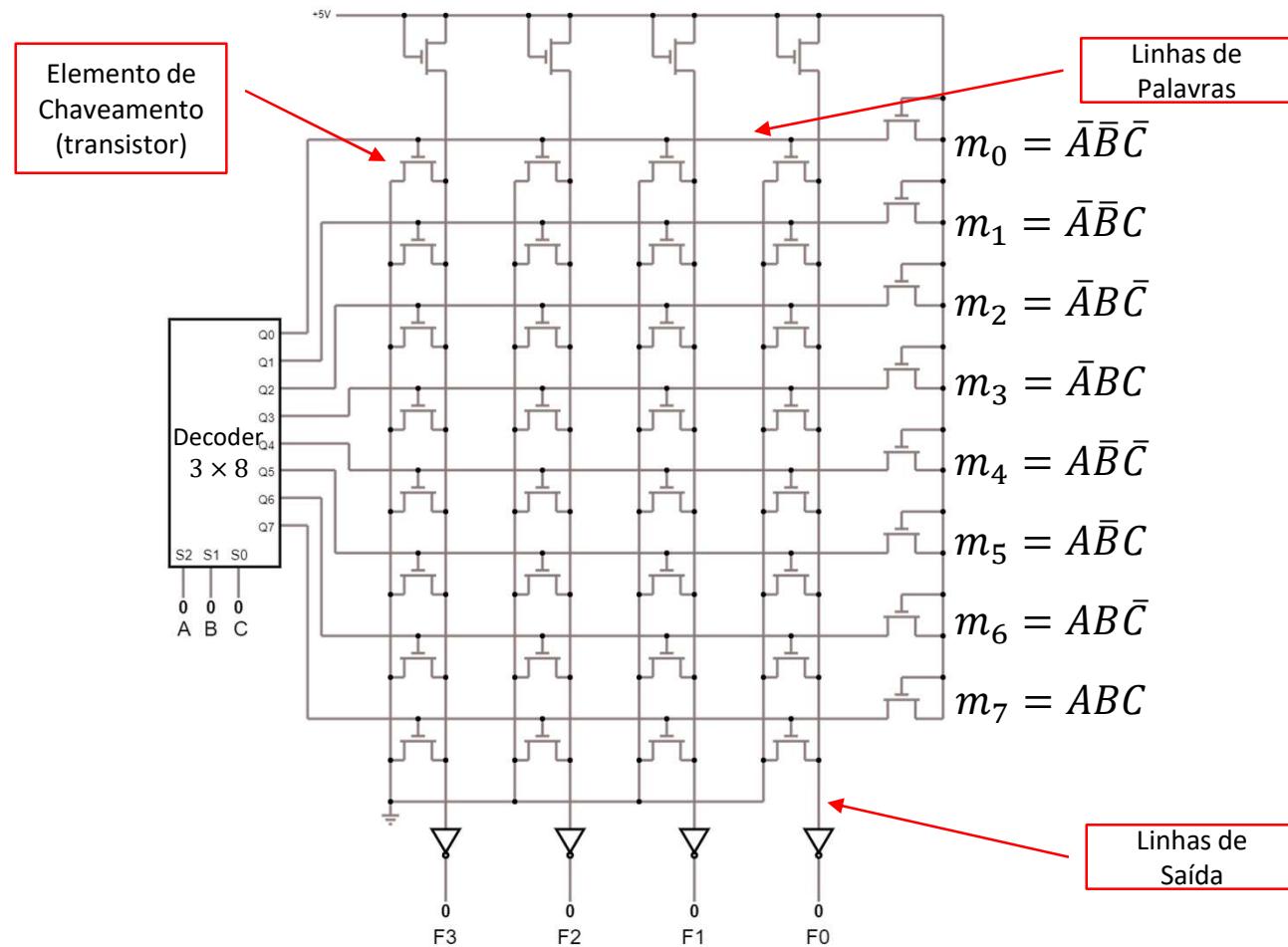
Read-Only Memory *ROM*

- Quando um padrão de n bits é aplicado na entrada do decodificador, uma das 2^n saídas do decodificador será 1.
- A linha de saída do decodificador seleciona uma das palavras do *array* de memória, e o padrão de bits armazenado nesta palavra é transferido para as linhas de saída;
- Seguem algumas figuras ilustrando uma possível estrutura interna de uma *ROM* de $8 \text{ words} \times 4 \text{ bits}$ completa:

Read-Only Memory ROM

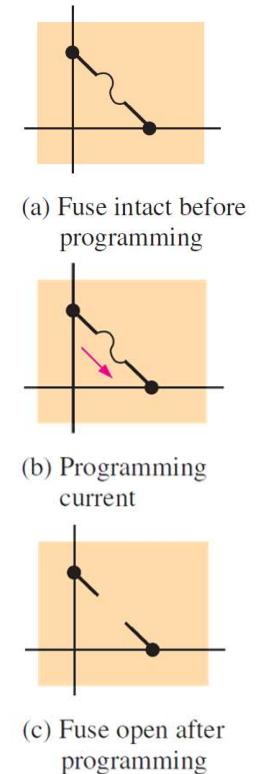


Read-Only Memory ROM



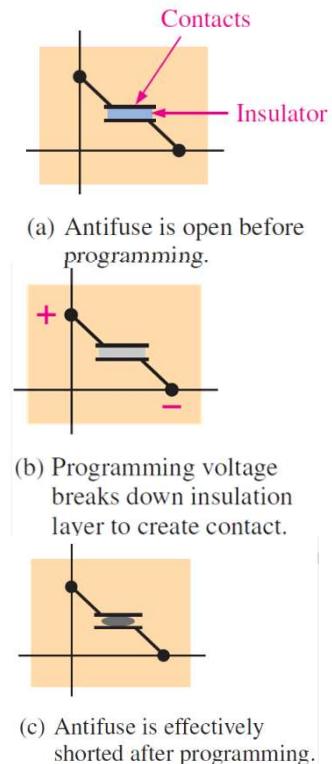
Read-Only Memory *ROM*

- O link programável pode ser de diferentes estratégias ou tecnologias:
 - ✓ Na **Tecnologia Fusível** um dispositivo de programação submete o link a uma corrente que provoca a abertura do circuito de forma permanente. Posição inicial é de link fechado.



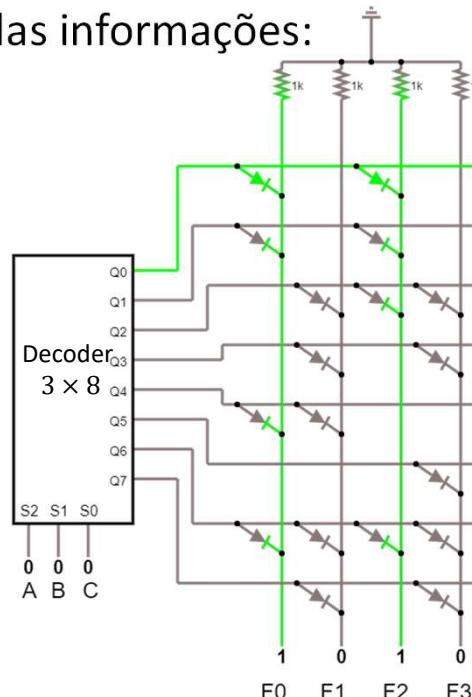
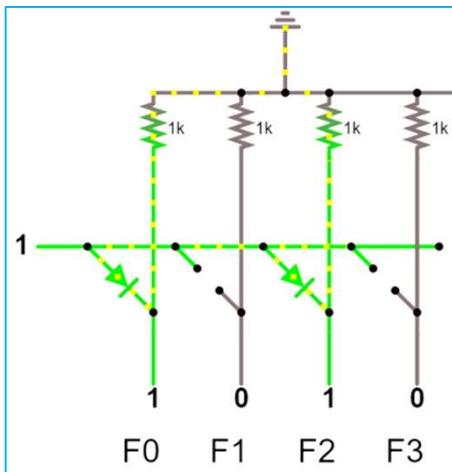
Read-Only Memory *ROM*

- O link programável pode ser de diferentes estratégias ou tecnologias:
 - ✓ Na **Tecnologia Fusível** um dispositivo de programação submete o link a uma corrente que provoca a abertura do circuito de forma permanente. Posição inicial é de link fechado;
 - ✓ A **Tecnologia Anti-fusível** submete o link a um efeito que estabelece a ligação, através do rompimento de um isolamento. Posição inicial é de link aberto.



Read-Only Memory ROM

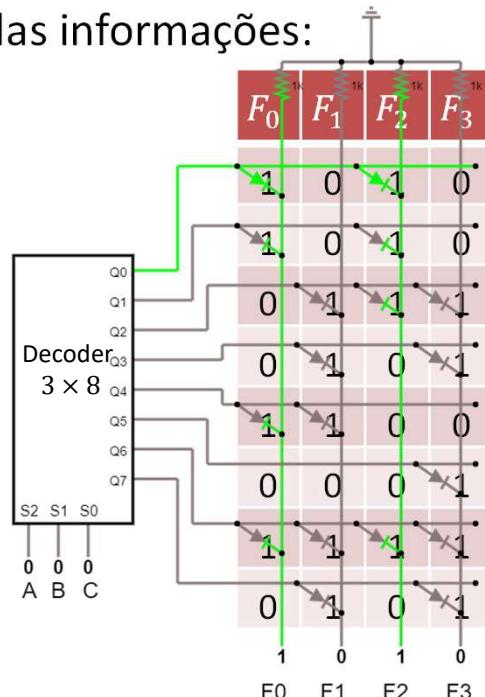
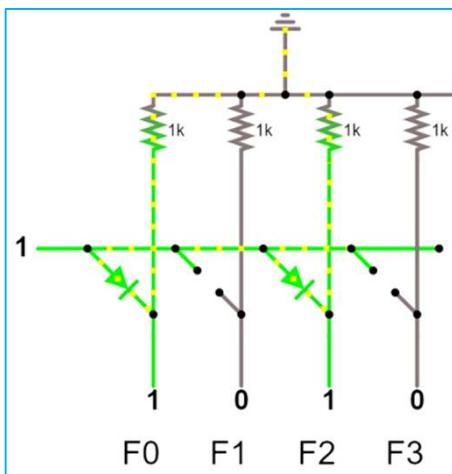
- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:



- ✓ A presença ou ausência do elemento de chaveamento leva a gravação da informação 0 ou 1;
- ✓ Neste diagrama, o diodo faz o papel do elemento de chaveamento;
- ✓ Quando presente, ele permite que a corrente flua da saída do *decoder* para o Resistor Pull-Down (ligado a terra), transferindo a tensão para a linha de saída;
- ✓ Apenas uma linha de palavra é habilitada, de acordo com o endereço selecionado por ABC;

Read-Only Memory ROM

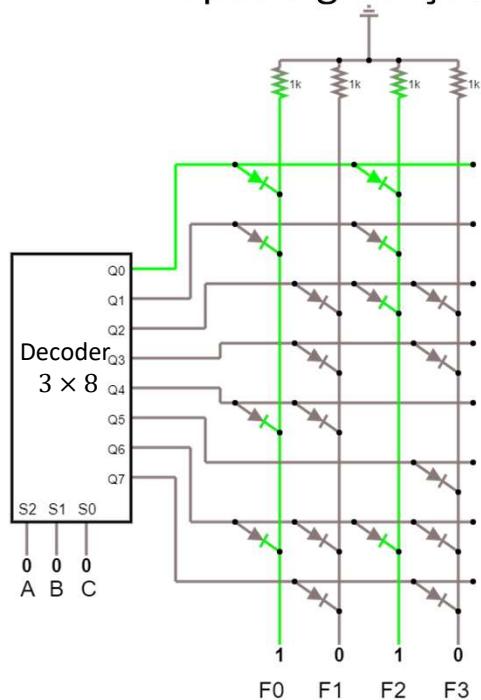
- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:



- ✓ A presença ou ausência do elemento de chaveamento leva a gravação da informação 0 ou 1;
- ✓ Neste diagrama, o diodo faz o papel do elemento de chaveamento;
- ✓ Quando presente, ele permite que a corrente flua da saída do *decoder* para o Resistor Pull-Down (ligado a terra), transferindo a tensão para a linha de saída;
- ✓ Apenas uma linha de palavra é habilitada, de acordo com o endereço selecionado por ABC;

Read-Only Memory ROM

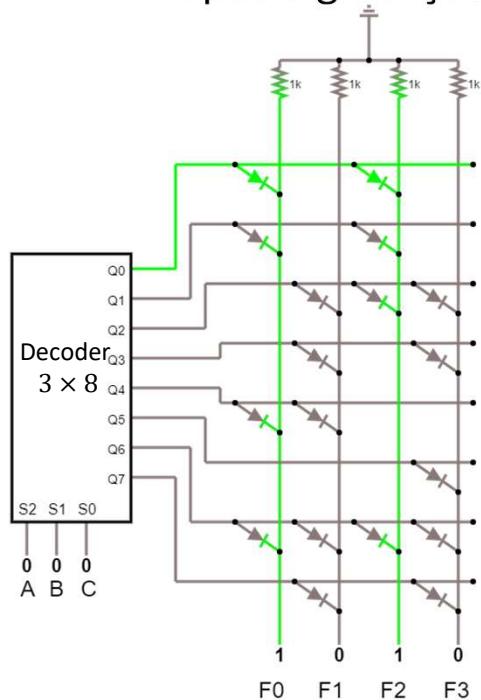
- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:



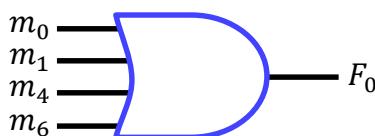
- ✓ O decodificador gera os 8 *mintermos* das três variáveis de entrada;
- ✓ O array de memória forma as quatro funções de saída, combinando os *mintermos* selecionados;
- ✓ Um elemento de chaveamento é colocado na intersecção de uma linha de palavra e uma linha de saída, caso o *mintermo* correspondente integre a função de saída;
- ✓ Caso contrário, o elemento de chaveamento é omitido (ou não conectado);
- ✓ Se o elemento de chaveamento conectar uma linha de saída à uma linha de palavra com valor 1, então a saída também será 1;

Read-Only Memory *ROM*

- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:



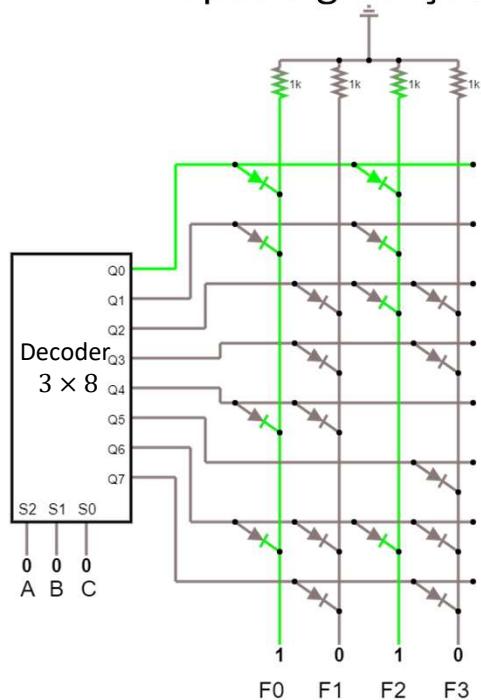
- ✓ Caso contrário, os resistores *pull-down* no topo do circuito fazem com que a linha de saída seja 0;
- ✓ Assim, os elementos de chaveamento organizados de tal forma no array de memória equivalem efetivamente a uma porta *OR* para cada uma das funções de saída;
- ✓ Por exemplo: m_0, m_1, m_4 e m_6 são combinados para formar F_0 ;
- ✓ Este circuito pode ser representado com porta *OR* equivalente.



Roth Jr, Charles H; Kinney, Larry L; Fundamentals of Logic Design, Seventh Edition. Cengage Learning, 2013 .

Read-Only Memory *ROM*

- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:

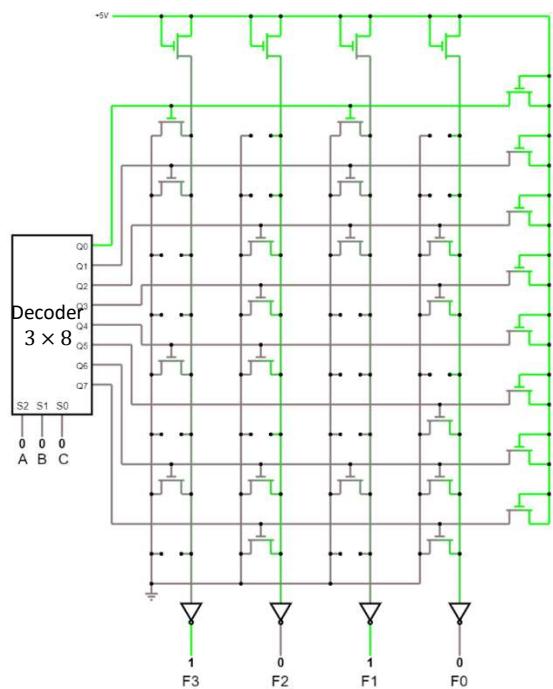


- ✓ A *ROM* desta figura gera as seguintes funções:
 - $F_0 = \Sigma m(0, 1, 4, 6) = \bar{A}\bar{B} + A\bar{C}$
 - $F_1 = \Sigma m(2, 3, 4, 6, 7) = B + A\bar{C}$
 - $F_2 = \Sigma m(0, 1, 2, 6) = \bar{A}\bar{B} + B\bar{C}$
 - $F_3 = \Sigma m(2, 3, 5, 6, 7) = AC + B$
- ✓ Assim, a *ROM* pode ser utilizada para implementar circuitos combinacionais com múltiplas saídas;

Roth Jr, Charles H; Kinney, Larry L; Fundamentals of Logic Design, Seventh Edition. Cengage Learning, 2013 .

Read-Only Memory *ROM*

- A seguinte figura ilustra uma possível estrutura interna de uma *ROM* de 8 words × 4 bits após a gravação das informações:



- ✓ A *ROM* desta figura gera as seguintes funções:
 - $F_0 = \Sigma m(0, 1, 4, 6) = \bar{A}\bar{B} + A\bar{C}$
 - $F_1 = \Sigma m(2, 3, 4, 6, 7) = B + A\bar{C}$
 - $F_2 = \Sigma m(0, 1, 2, 6) = \bar{A}\bar{B} + B\bar{C}$
 - $F_3 = \Sigma m(2, 3, 5, 6, 7) = AC + B$
- ✓ Assim, a *ROM* pode ser utilizada para implementar circuitos combinacionais com múltiplas saídas;

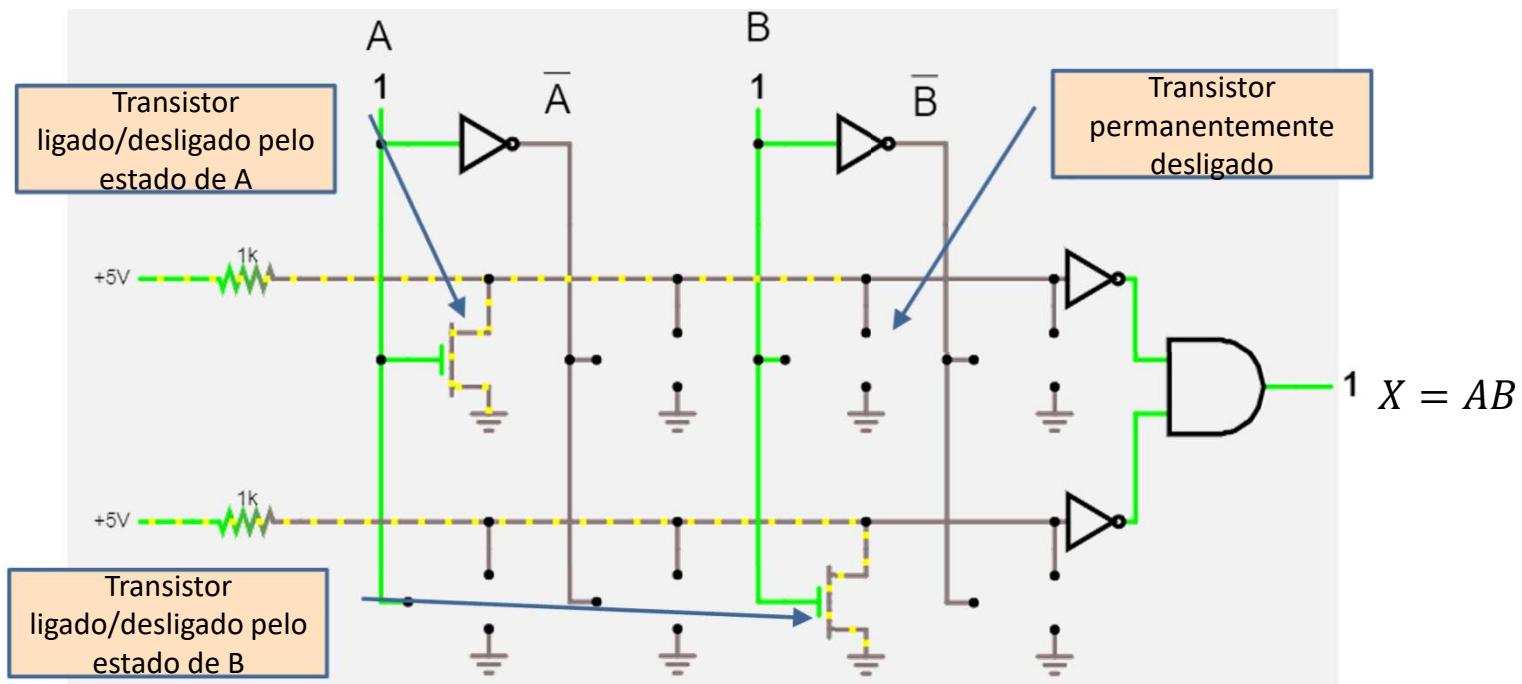
Read-Only Memory *ROM*

- Os três tipos comuns de *ROMs* são:
 - ✓ *Masked ROM*;
 - ✓ *ROMs* programáveis (*PROMs*); e
 - ✓ *Electrically Erasable Programmable ROMs (EEPROMs)*.
- A *Masked ROM* programável é gravada no momento da fabricação, onde a matriz de dados é armazenada permanentemente.
- Neste processo, os elementos de chaveamento são seletivamente inseridos ou omitidos nos pontos de interseções das linhas e colunas da matriz de memória.
- Isso requer a preparação de uma máscara especial, que é usada durante a fabricação do circuito integrado.

Read-Only Memory *ROM*

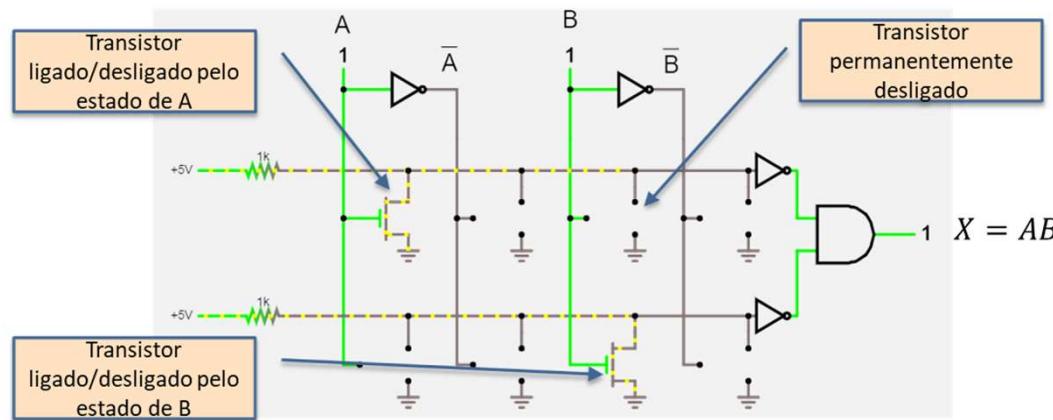
- A preparação desta máscara é cara, o que torna o uso de *Masked ROMs* programáveis viável apenas para grandes quantidade (normalmente vários milhares ou mais) com a mesma matriz de dados;
- A **PROM** – *ROM Programável* pode ser gravada com um dispositivo específico, não podendo mais ser alterada ou apagada;
- A tecnologia *PROM* também é chamada de *OTP - Programável Uma única Vez*;
- Na forma mais comum, é utilizada uma tensão alta para desfazer ou estabelecer uma ligação interna (fusível ou anti-fusível)

Read-Only Memory *ROM*



Read-Only Memory ROM

- Na **Tecnologia EPROM** (*Electrically Programmable Read-Only Memories*) um transistor também faz o papel do link;
- No entanto, *CIs* com janelas podem ser apagados com luz ultravioleta e reprogramadas.
- A tecnologia *EPROM* utiliza um tipo especial de transistor *MOS*, conhecido como *floating gate transistor*, como link programável.



Read-Only Memory *ROM*

- A modificação dos dados armazenados em uma *ROM* é normalmente necessária na fase de desenvolvimento de um sistema digital;
- Isto leva a adoção de *EEPROMs* em detrimento das *Masked ROMs* programáveis;
- As *EEPROMs* utilizam um mecanismo especial de armazenamento de cargas para habilitar ou desabilitar os elementos de chaveamento da matriz de memória.
- Um gravador de *PROM* é utilizado para fornecer pulsos de tensões apropriados para armazenar cargas nas posições de memória da matriz.
- Os dados armazenados nesta técnica são permanentes até serem submetidos a um processo de descarga que apaga a informação;

Read-Only Memory *ROM*

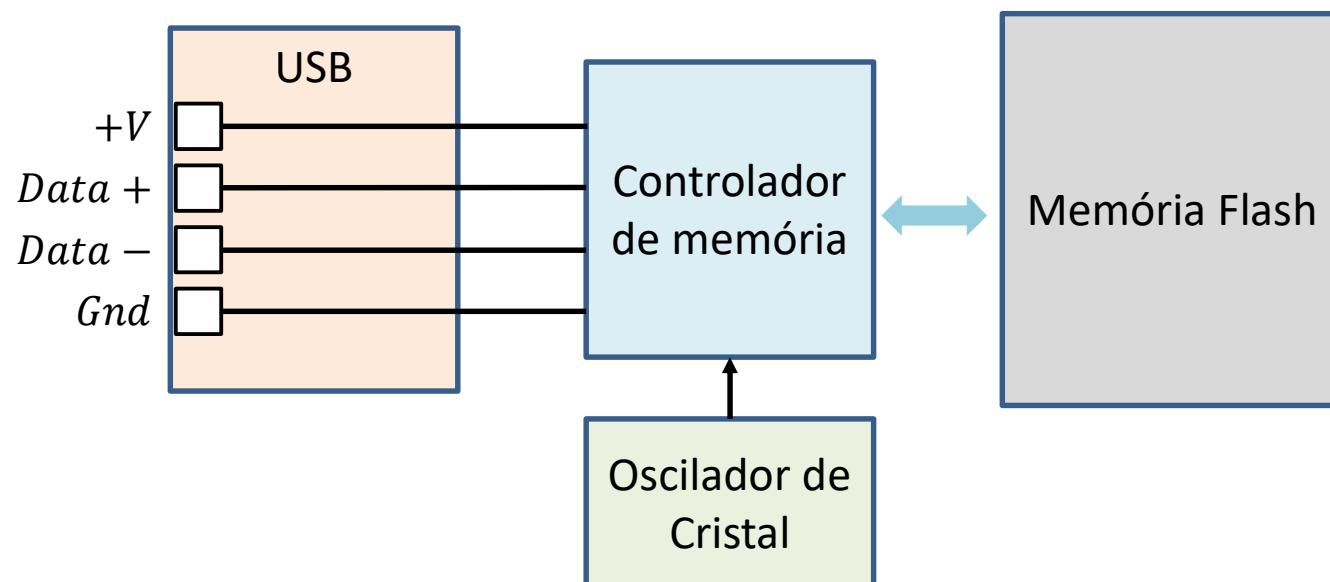
- A **Tecnologia EEPROM** (*Erasable Electrically Programmable Read-Only Memories*) é semelhante à *EPROM* pois também utiliza um *transistor de porta flutuante*;
- A diferença é que a *EEPROM* pode ser apagada e reprogramada eletricamente sem a necessidade de luz *UV*.
- Uma vez que os dados tenham sido apagados, um novo conjunto pode ser armazenado na *EEPROM*;
- Uma *EEPROM* pode ser apagada e reprogramada um número limitado de vezes, normalmente de 100 a 1000 vezes;

Read-Only Memory *ROM*

- As **memórias flash** são semelhantes às *EEPROMs*, exceto pelo material utilizado na construção dos transistores;
- O que proporciona uma memória mais rápida e disponíveis em dispositivos de maior densidade;
- A **Tecnologia Flash** é baseada também em link de transistor, sendo não volátil e reprogramável.
- As memórias *Flash* podem ser apagadas e reprogramadas por um dispositivo integrado a própria memória:
 - permitindo assim que novos dados possam ser gravados mesmo ela estando conectada ao sistema; e
 - sem a necessidade de um equipamento separado.

Read-Only Memory *ROM*

- No caso do disco *USB* de memória *Flash*, tem-se o seguinte diagrama básico:



Read-Only Memory *ROM*

- As tecnologias de fusível, anti-fusível, *EPROM*, *EEPROM* e *Flash* são não-voláteis:
 - retêm sua programação quando a energia for interrompida.
- Um fusível fica permanentemente aberto, um anti-fusível fica permanentemente fechado e os transistores de porta flutuante usados em matrizes baseadas em *EPROM* e *EEPROM* podem reter seu estado ligado ou desligado indefinidamente.

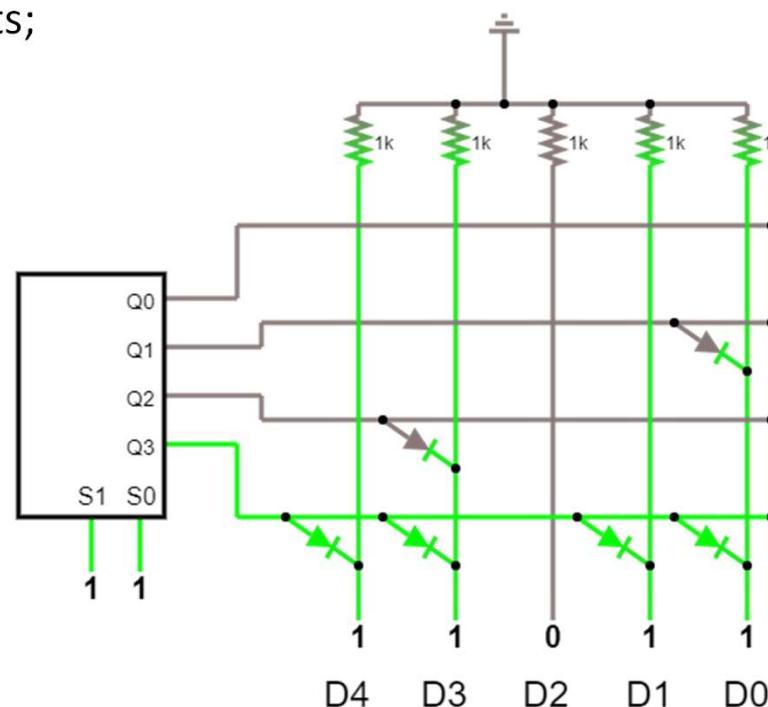
Read-Only Memory

Exemplo

- i. Implementar, utilizando memória *ROM*, a função $f(x) = x^3$, para uma entrada de 2 bits;
- ii. Implementar, utilizando memória *ROM*, a função $f(x) = x^2$, para uma entrada de 3 bits;

Read-Only Memory *Exemplo*

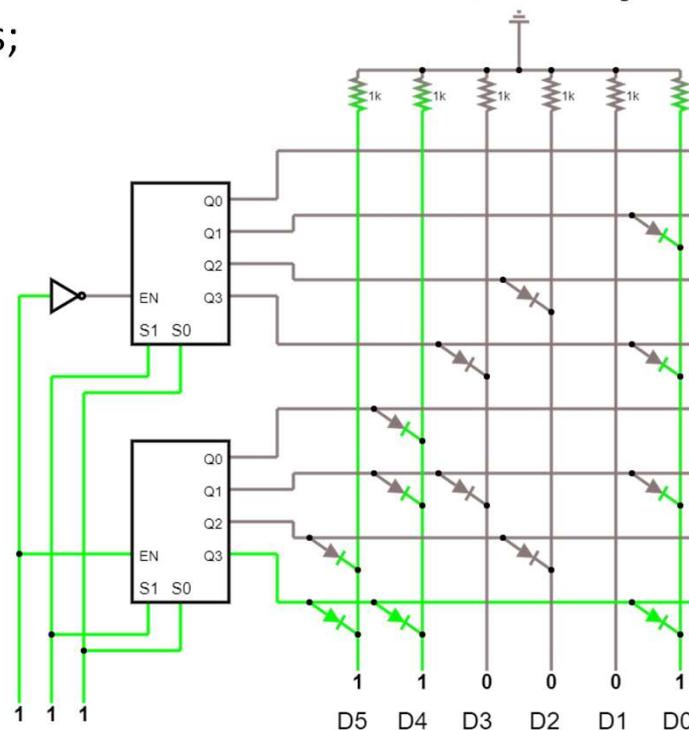
- i. Implementar, utilizando memória *ROM*, a função $f(x) = x^3$, para uma entrada de 2 bits;



<https://tinyurl.com/yecfdawc>

Read-Only Memory *Exemplo*

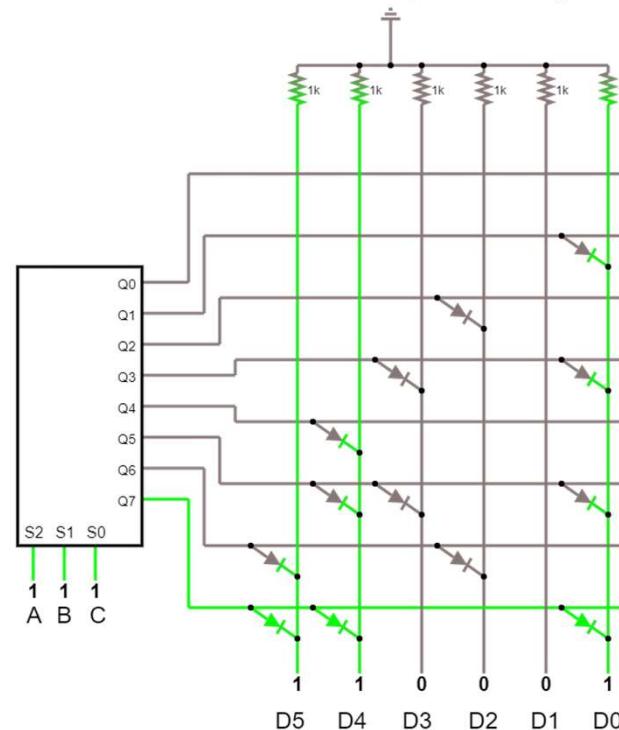
- ii. Implementar, utilizando memória *ROM*, a função $f(x) = x^2$, para uma entrada de 3 bits;



<https://tinyurl.com/ygepxq9z>

Read-Only Memory *Exemplo*

- ii. Implementar, utilizando memória *ROM*, a função $f(x) = x^2$, para uma entrada de 3 bits;



<https://tinyurl.com/yhqwa8gv>

Read-Only Memory

Exercício

- i. Implementar, utilizando memória *ROM*, um somador completo de 1 bit;
- ii. Implementar, utilizando memória *ROM*, um multiplicador de 2×2 bits, fornecendo um resultado de 4 bits.

Sumário

- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

PLD

Programmable Logic Device

- Um *Dispositivo Lógico Programável (PLD)* é o nome dado a um circuito integrado capaz de ser programado para fornecer diversas funções lógicas;
- Os *PLDs* podem ser obtidos com circuitos combinacionais ou sequenciais;
- Os *PLDs* combinacionais simples são capazes de realizar de 2 a 10 funções com 4 a 16 variáveis em um único circuito integrado;
- *PLDs* mais complexos podem conter milhares de portas e *flip-flops*;
- Assim, um único *PLD* pode substituir diversos circuitos integrados, o que pode representar projetos de menor custo.

PLD
Programmable Logic Device

- Quando um sistema digital é projetado usando um *PLD*, as modificações no projeto podem ser feitas facilmente alterando a programação do *PLD* sem a necessidade de refazer a fiação do sistema;
- *PLA* (*Programmable Logic Array*) e *PAL* (*Programmable Array Logic*) são tipos de *PLD*;

Sumário

- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

PLA

Programmable Logic Array

- *Programmable Logic Array* ou Matriz Lógica Programável são circuitos com arranjo lógico pré-formado;
- Disponibiliza um mecanismo de conexão das lógicas;
- É uma alternativa rápida para a criação de circuitos;
- As funções são implementadas no formato *Soma de Produtos*;
- As funções podem combinar variáveis e seus complementos;

PLA

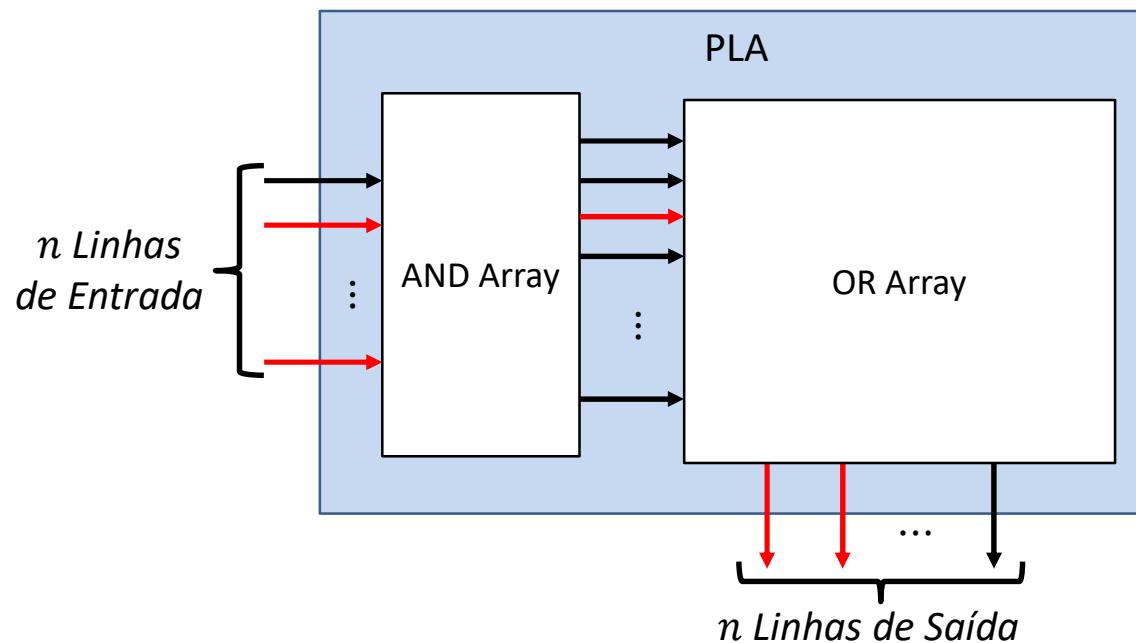
Programmable Logic Array

- Um *PLA* executa a mesma função básica de uma *ROM*;
- Um *PLA* com n entradas e m saídas pode realizar m funções de n variáveis.
- A organização interna do *PLA* é diferente da *ROM*;
- O decodificador é substituído por uma matriz *AND*, que forma um determinado produto de termos com as variáveis de entrada;
- O array *OR* reúne os produtos de termos necessários para formar as funções de saída;
- Então, um *PLA* implementa uma expressão de soma de produtos, enquanto uma *ROM* implementa diretamente uma tabela verdade.

PLA

Programmable Logic Array

- Um *PLA* consiste basicamente de um array *AND* e um array *OR*;



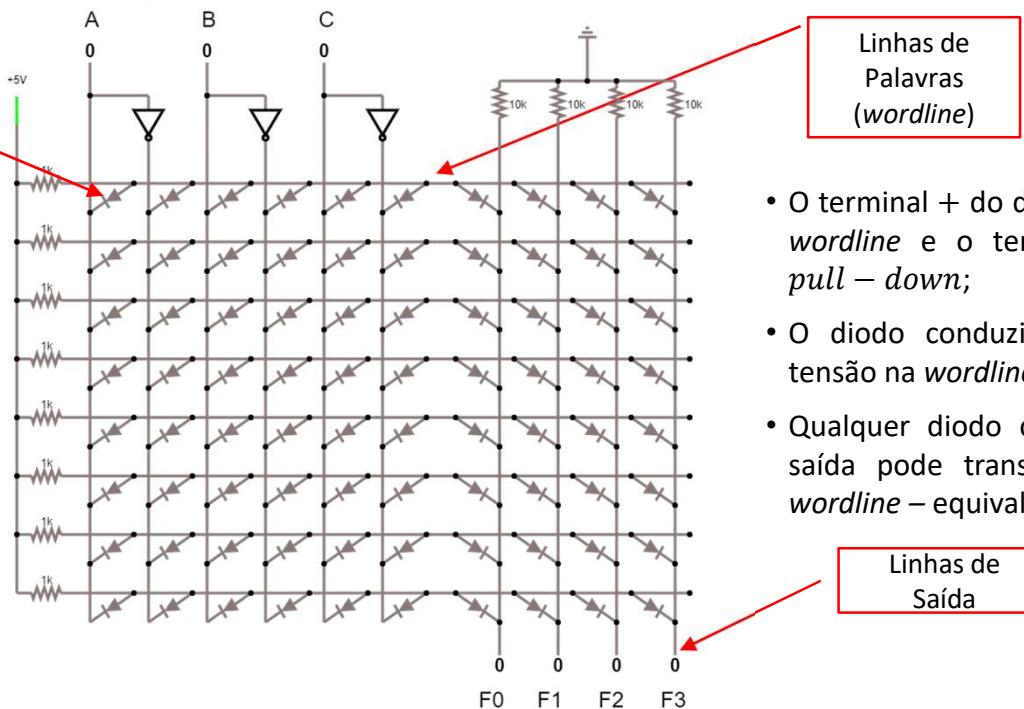
PLA

Programmable Logic Array

- A seguinte figura ilustra uma possível estrutura interna de uma *PLA* de 8 entradas e 4 saídas:

Elemento de Chaveamento (Link Programável)

- O diodo irá conduzir quando a diferença de tensão entre seus terminais for superior a barreira de potencial ($\approx 0,5V$);
- O terminal + está conectado ao resistor *pull-up* e o terminal - à linha de entrada;
- Quando a entrada for 0 (0V), o diodo conduzirá ($V_+ > V_-$), drenando a corrente da *wordline*;
- A *wordline* será 1 quando todos os diodos ligados a ela não drenarem a corrente – equivale ao *AND*;



Linhas de Palavras (*wordline*)

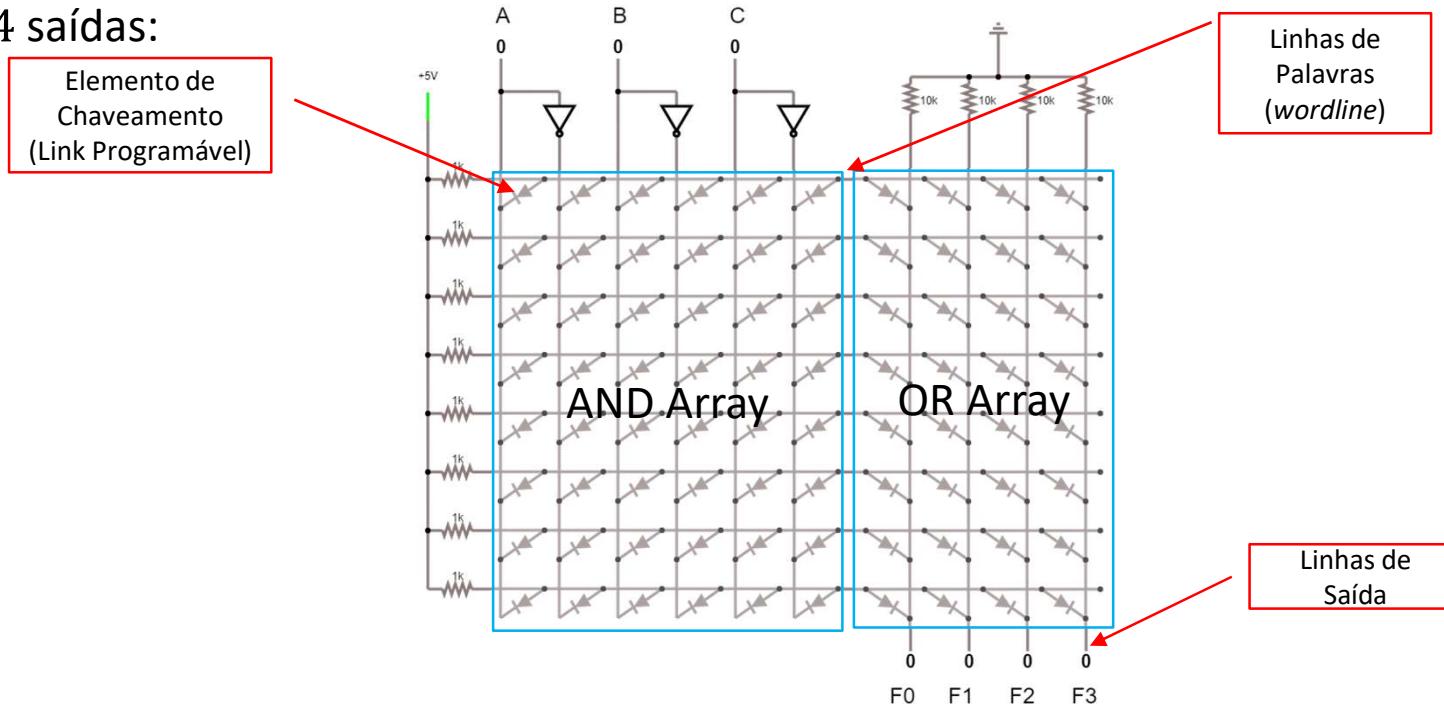
- O terminal + do diodo está conectado à *wordline* e o terminal - ao resistor *pull-down*;
- O diodo conduzirá sempre que tiver tensão na *wordline*;
- Qualquer diodo conectado a linha de saída pode transferir o sinal da sua *wordline* – equivale ao *OR*;

Linhas de Saída

PLA

Programmable Logic Array

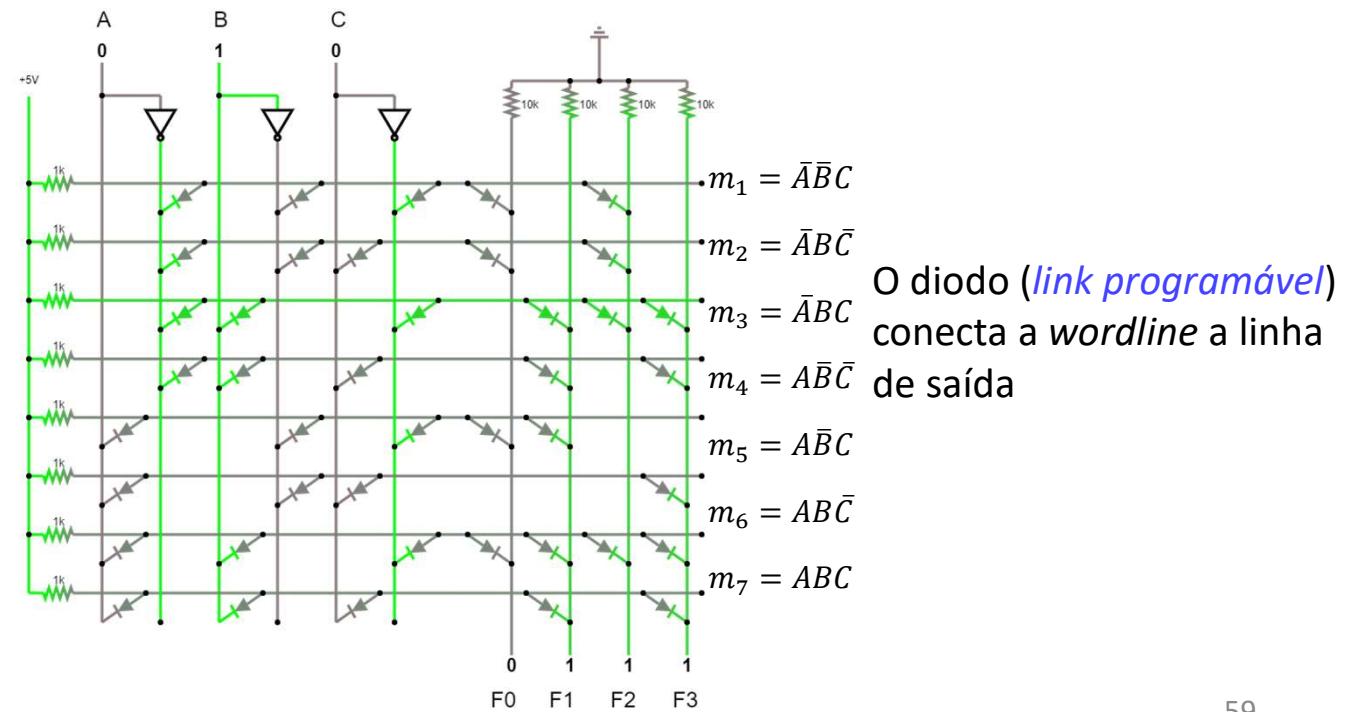
- A seguinte figura ilustra uma possível estrutura interna de uma *PLA* de 8 entradas e 4 saídas:



PLA

Programmable Logic Array

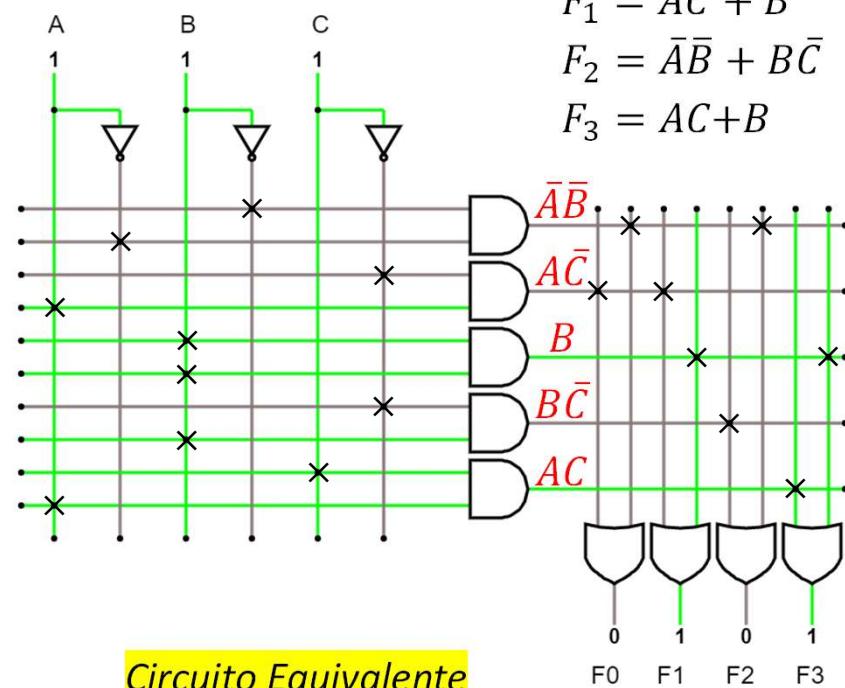
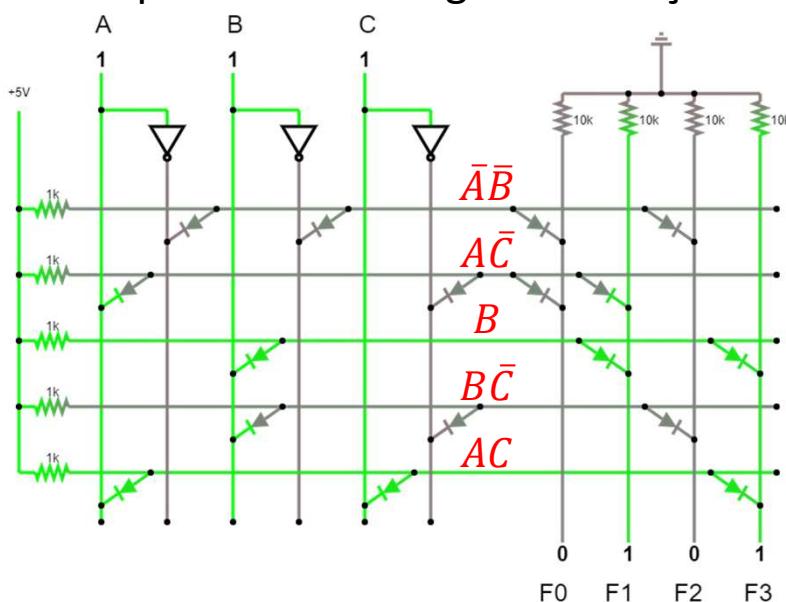
- Implementação de uma *ROM* utilizando um *PLA* de 8 entradas e 4 saídas:



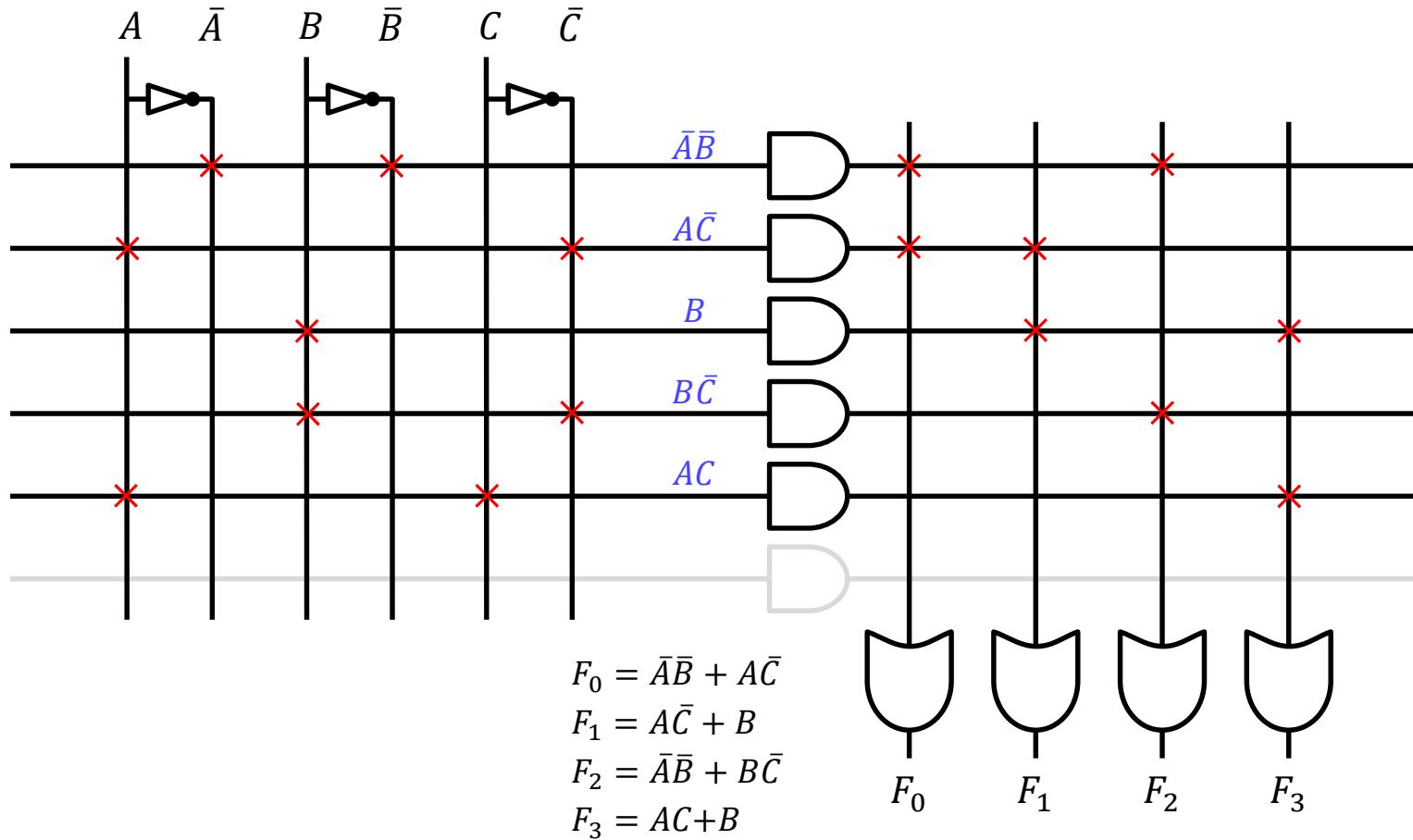
PLA

Programmable Logic Array

- Implementar as seguintes funções lógicas utilizando PLA:



PLA
Programmable Logic Array

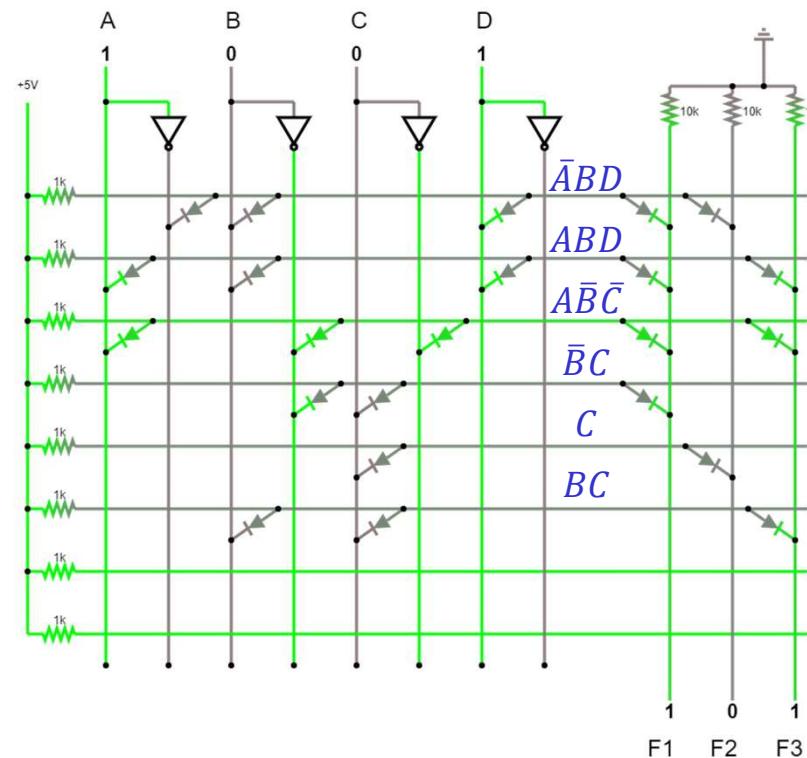


PLA

Programmable Logic Array

- Implementar a seguinte tabela verdade utilizando *PLA*:

| A | B | C | D | F ₁ | F ₂ | F ₃ |
|---|---|---|---|----------------|----------------|----------------|
| 0 | 1 | - | 1 | 1 | 1 | 0 |
| 1 | 1 | - | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | - | 1 | 0 | 1 |
| - | 0 | 1 | - | 1 | 0 | 0 |
| - | - | 1 | - | 0 | 1 | 0 |
| - | 1 | 1 | - | 0 | 0 | 1 |

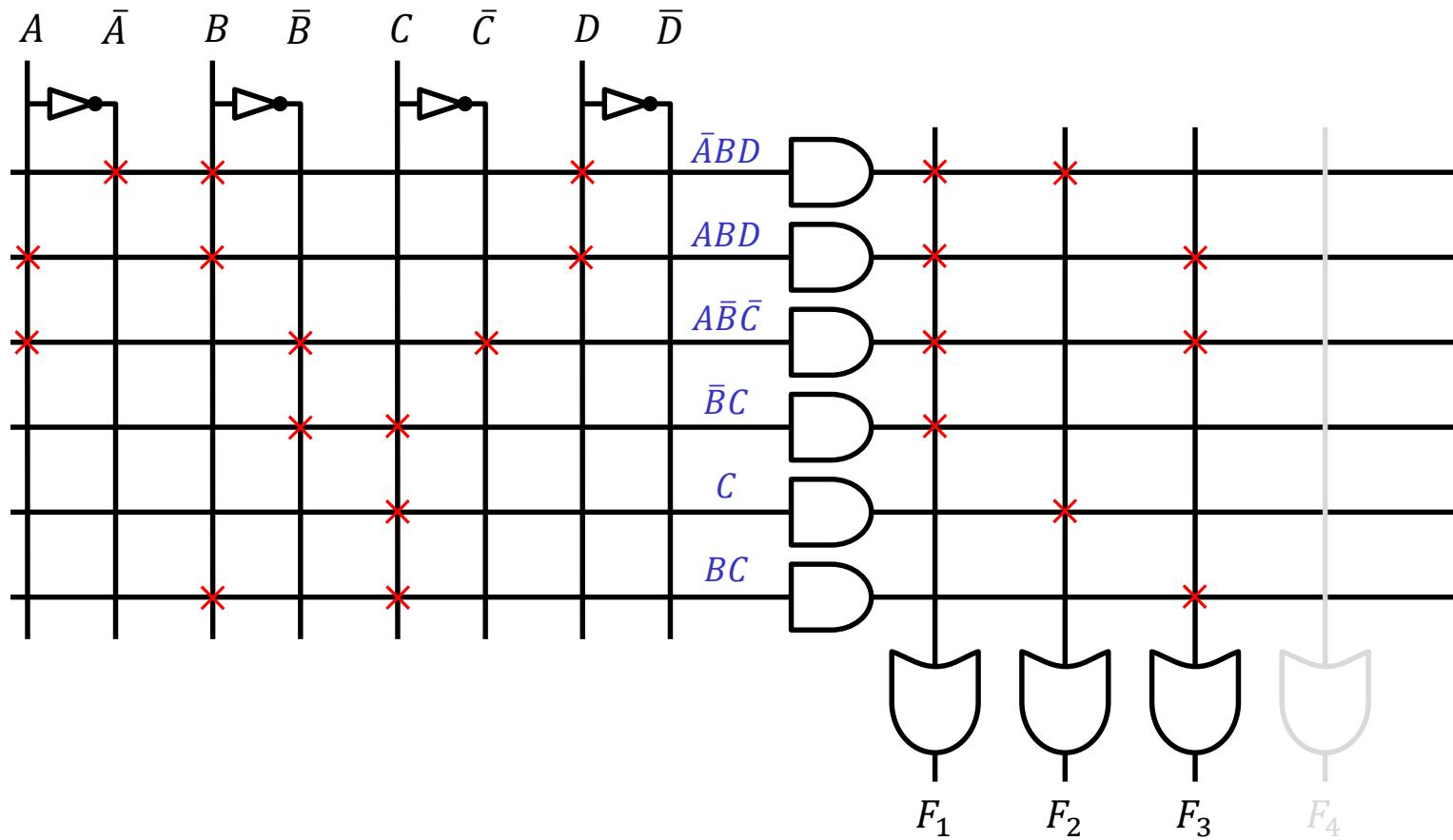


$$0001 \Rightarrow [] \Rightarrow 000$$

$$1001 \Rightarrow [L_3] \Rightarrow 101$$

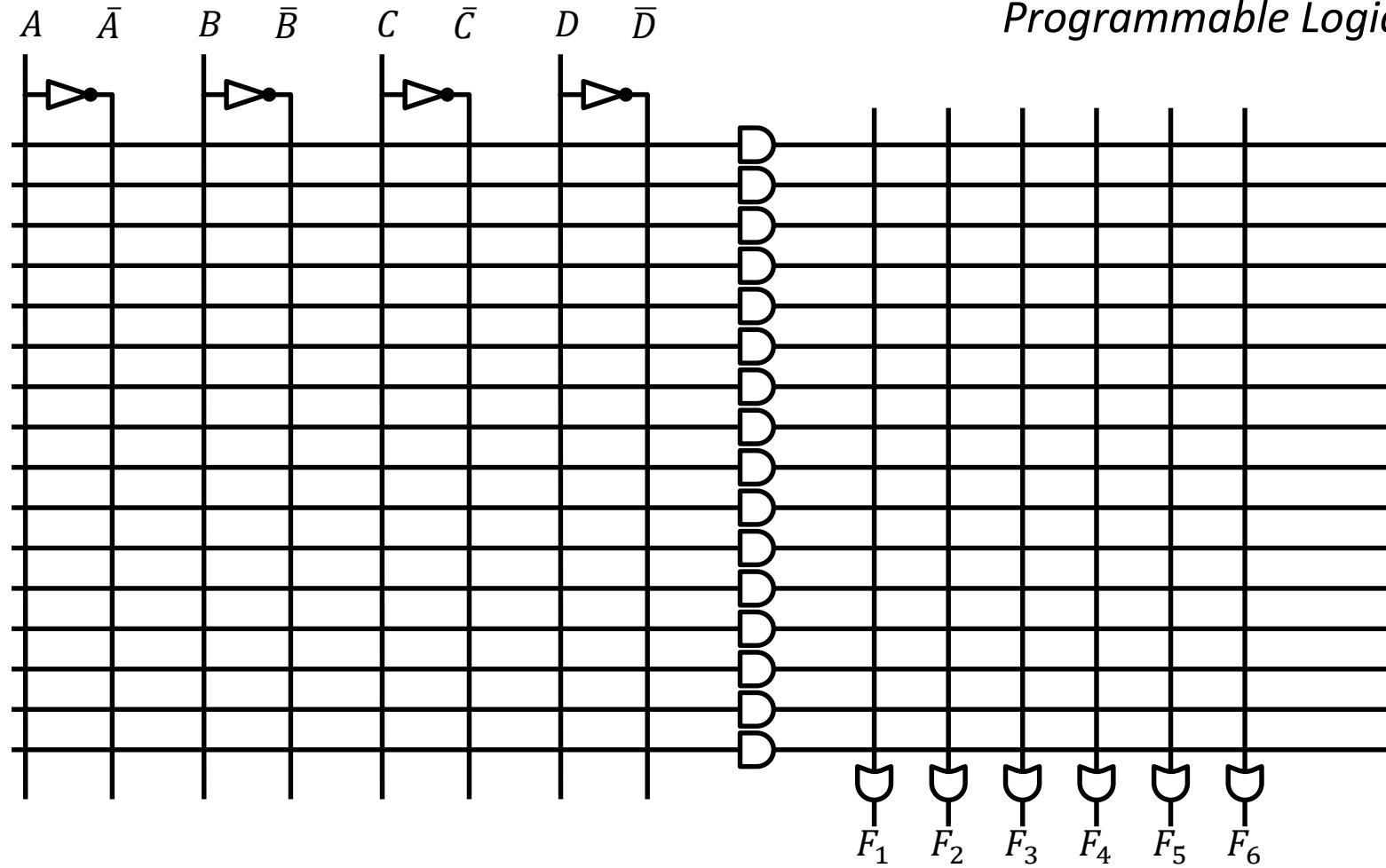
$$0111 \Rightarrow [L_1, L_5, L_6] \Rightarrow 111$$

PLA
Programmable Logic Array



PLA

Programmable Logic Array



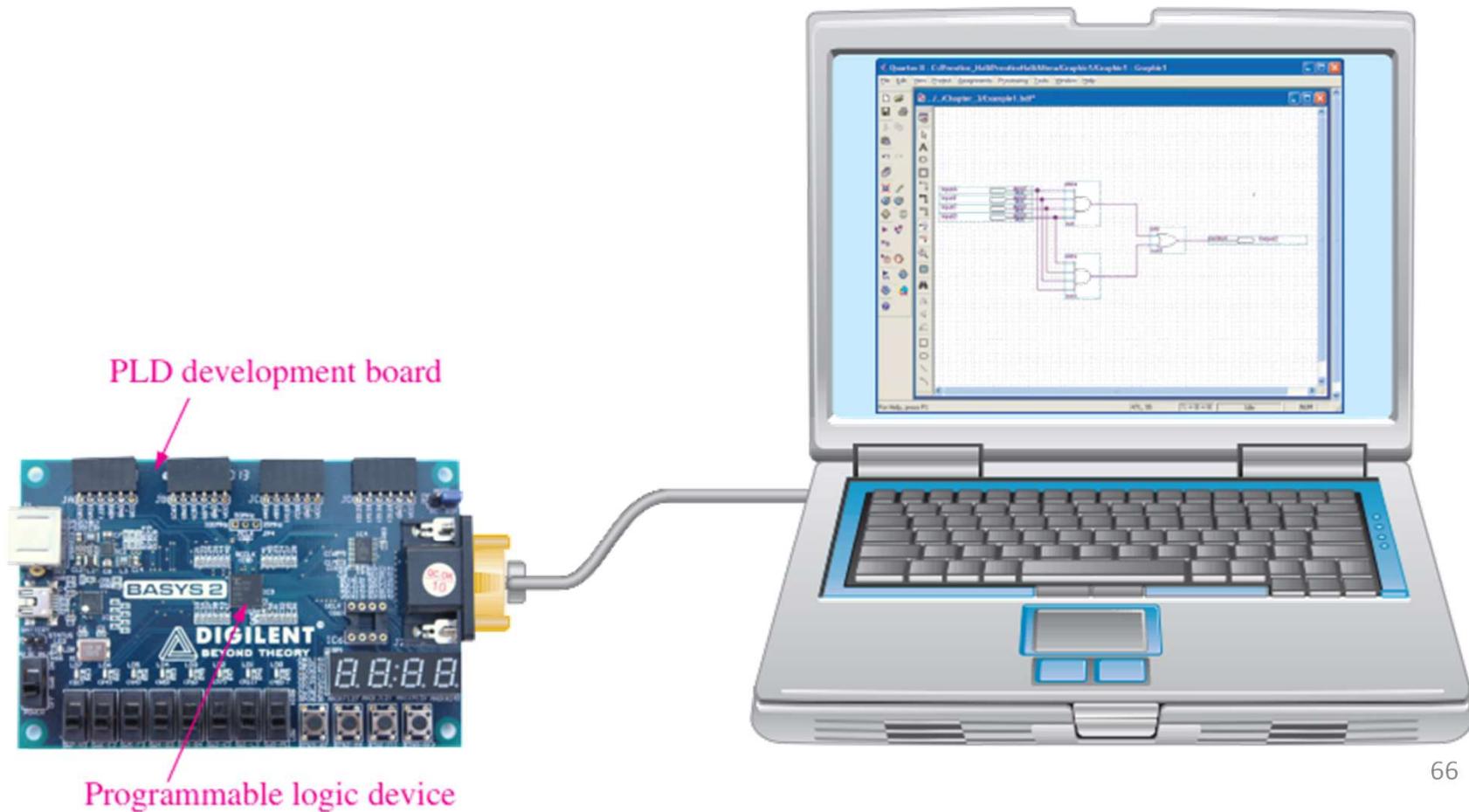
PLA

Programmable Logic Array

- A tabela de um *PLA* é significativamente diferente de uma tabela verdade de uma *ROM*;
- Na tabela verdade da *ROM*, cada uma das linhas representa um *mintermo*;
- Assim, somente uma linha será selecionada para cada combinação dos valores de entrada;
- Por outro lado, cada linha em uma tabela *PLA* representa um termo geral do produto;
- Desta forma, serão selecionadas *zero, uma ou mais* linhas para cada combinação dos valores de entrada.

PLD

Programmable Logic Device



Sumário

- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

PAL

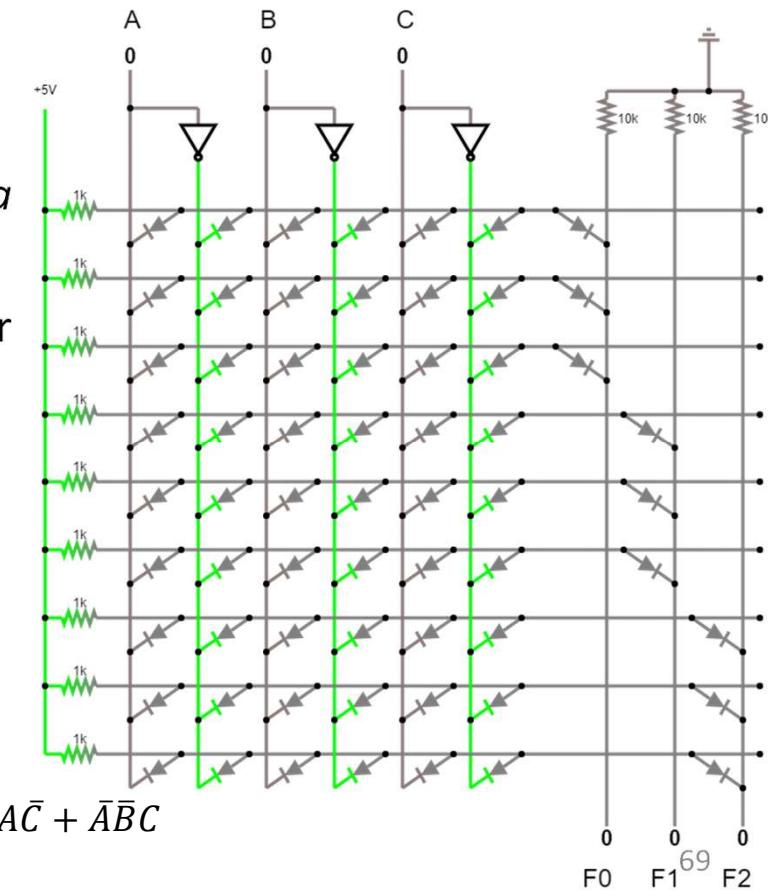
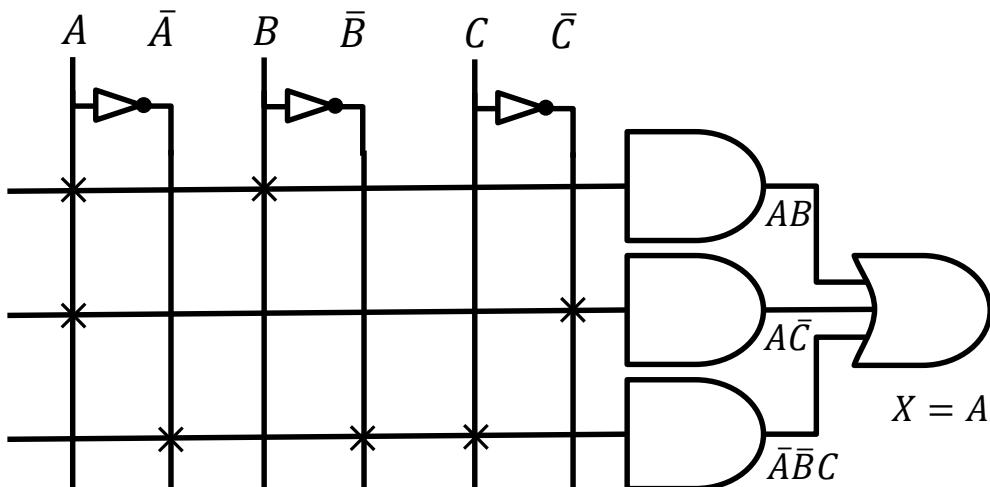
Programmable Array Logic

- O *PAL* (*Programmable Array Logic*) é um tipo particular de array lógico programável, onde o array *AND* é programável e o array *OR* é fixo;
- A estrutura básica do *PAL* é a mesma do *PLA*;
- Pelo fato de apenas a matriz *AND* ser programável, o *PAL* acaba sendo mais econômico do que o *PLA* genérico, e ao mesmo tempo mais fácil de programar;
- Por esse motivo, os projetistas de sistemas digitais frequentemente usam *PALs* para substituir portas lógicas individuais quando várias funções lógicas devem ser implementadas;

PAL

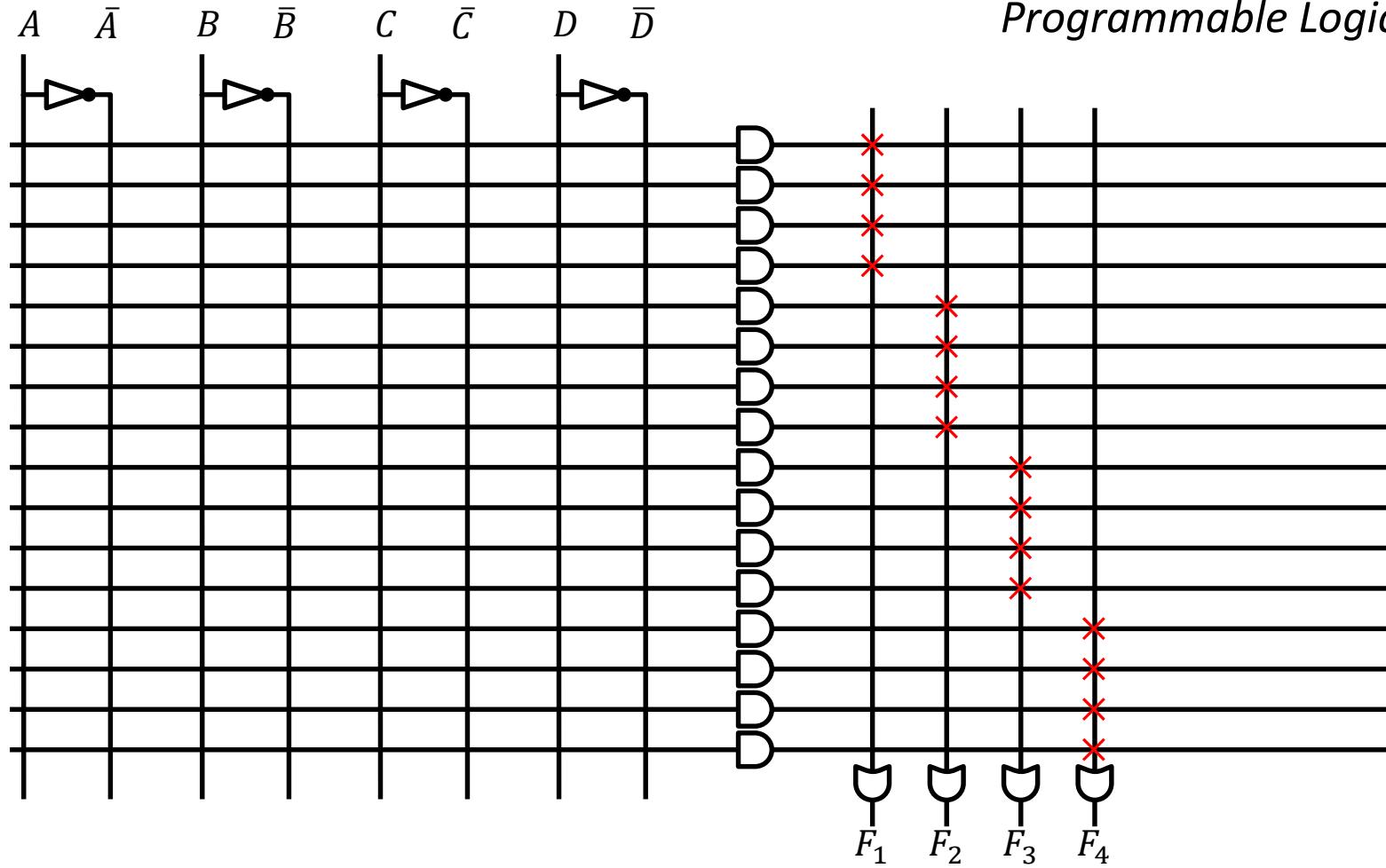
Programmable Array Logic

- Exemplo de uma *PAL* de 3 entradas e 3 saídas;
- O array *AND* é programável através da *queima* dos links;
- O array *OR* é fixo e está programado para aceitar até três entradas em cada linha de saída;



PAL

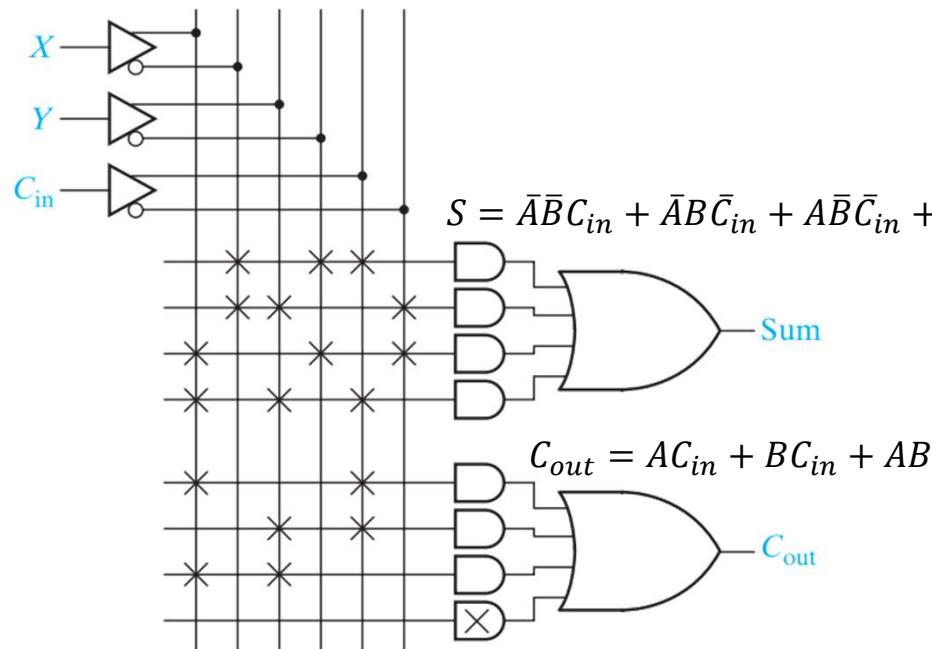
Programmable Logic Array



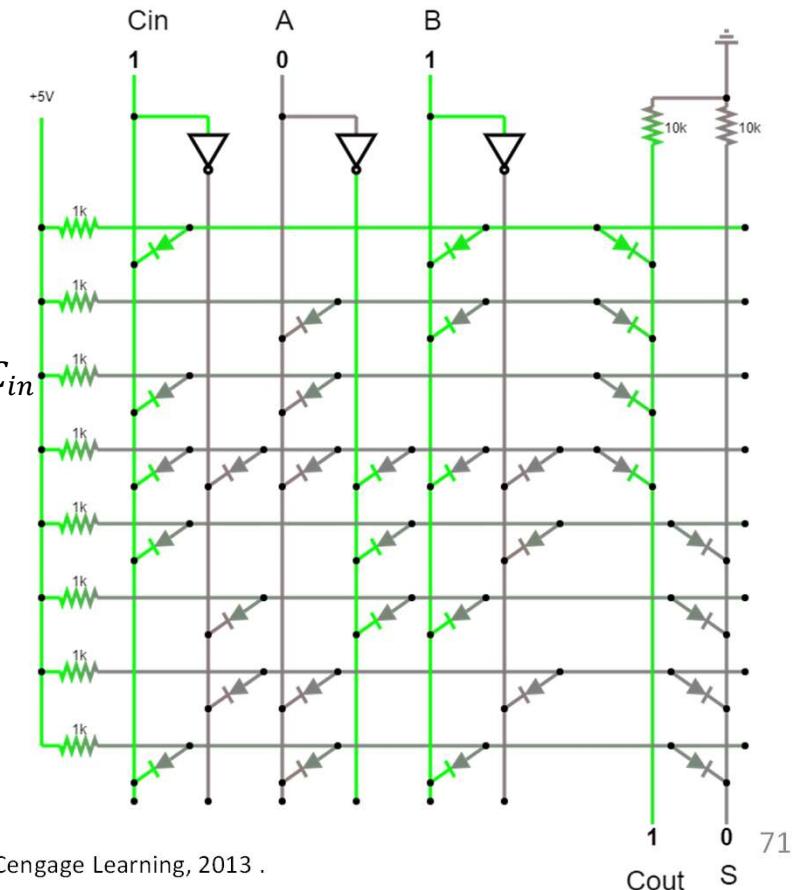
PAL

Programmable Array Logic

- Somador Completo de 1 Bit:



Roth Jr, Charles H; Kinney, Larry L; Fundamentals of Logic Design, Seventh Edition. Cengage Learning, 2013 .



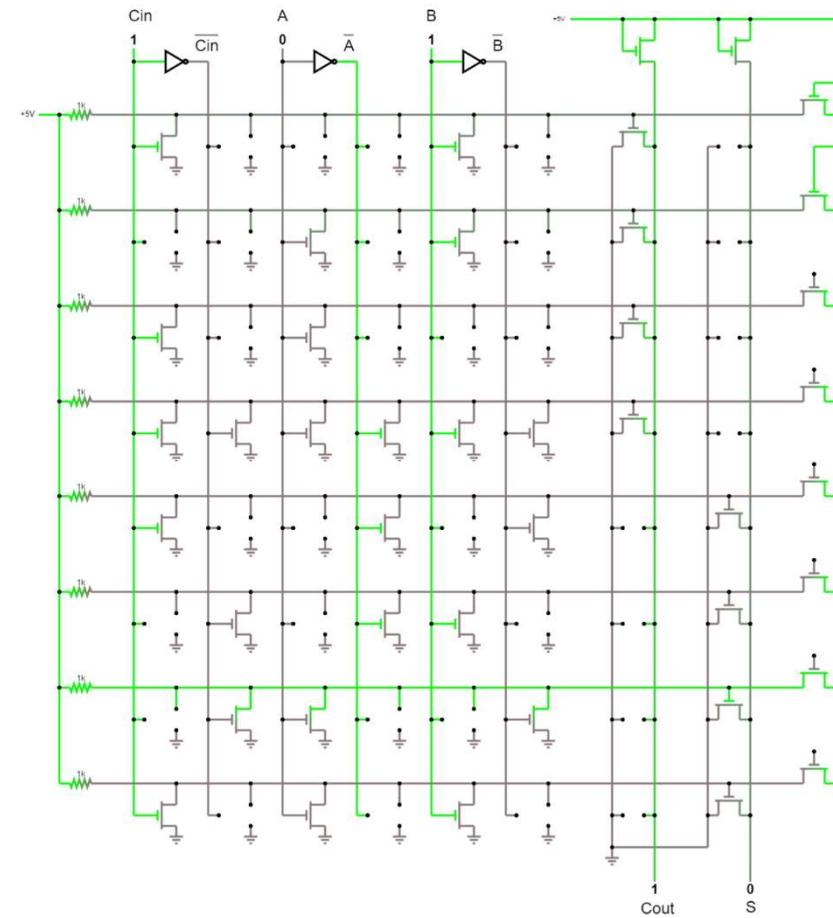
PAL

Programmable Array Logic

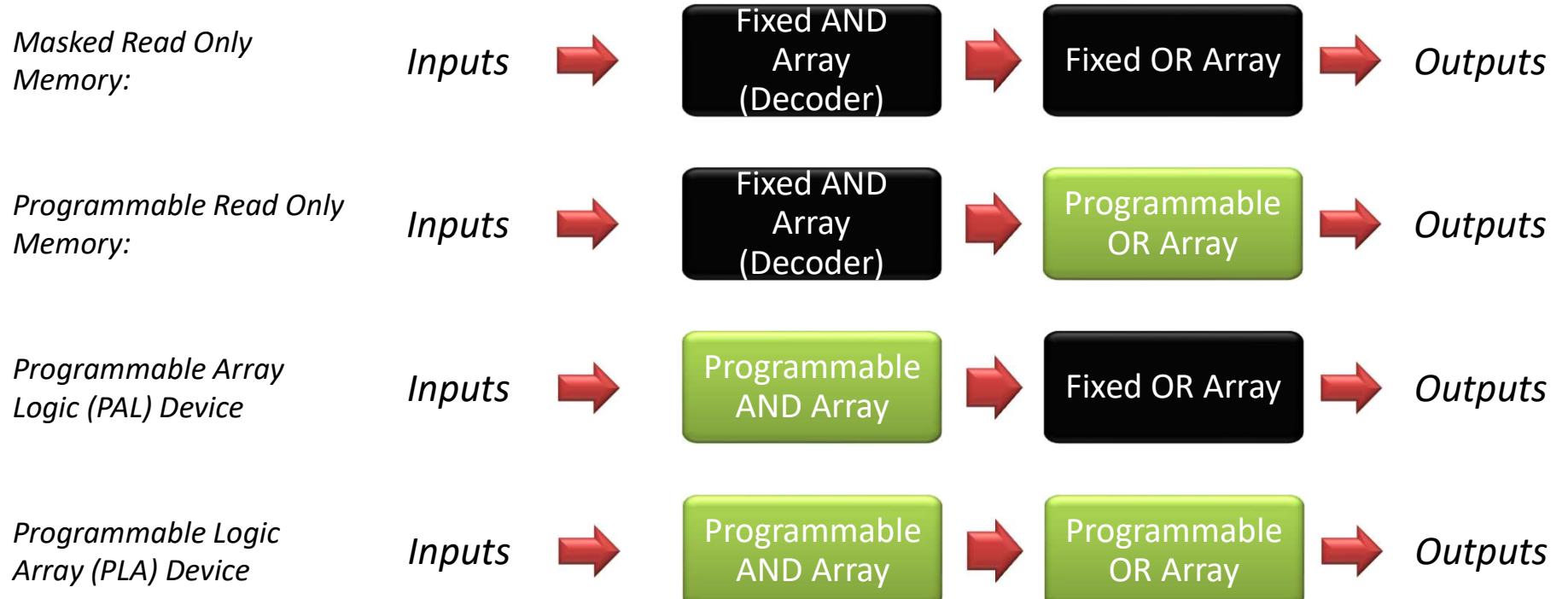
- Somador Completo de 1 Bit:

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$C_{out} = AC_{in} + BC_{in} + AB$$



Disponíveis Programáveis *Tipos*



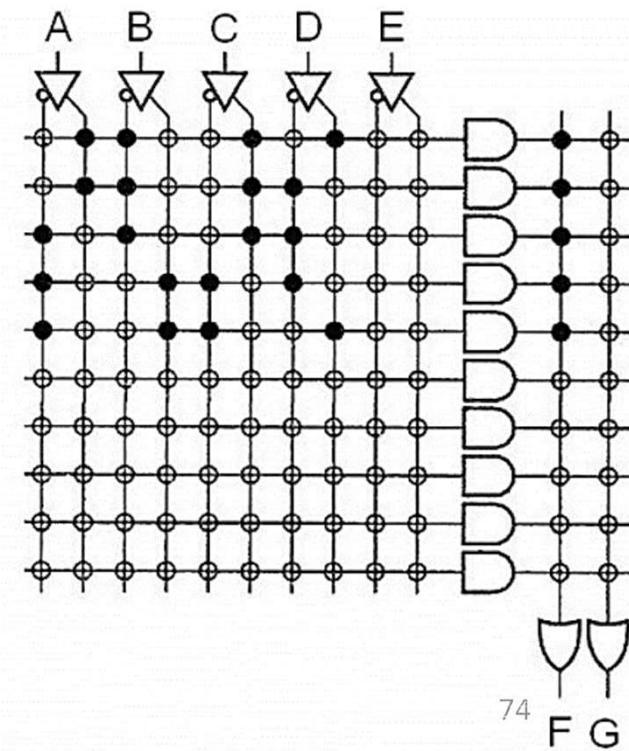
Programmable Logic Device

Exercício

- 41) [POSCOMP 2010] Considere o circuito digital apresentado no diagrama a seguir. Ressalte-se que, por convenção, chaves representadas por círculos escuros representam conexões fechadas e chaves representadas por círculos vazados representam conexões abertas.

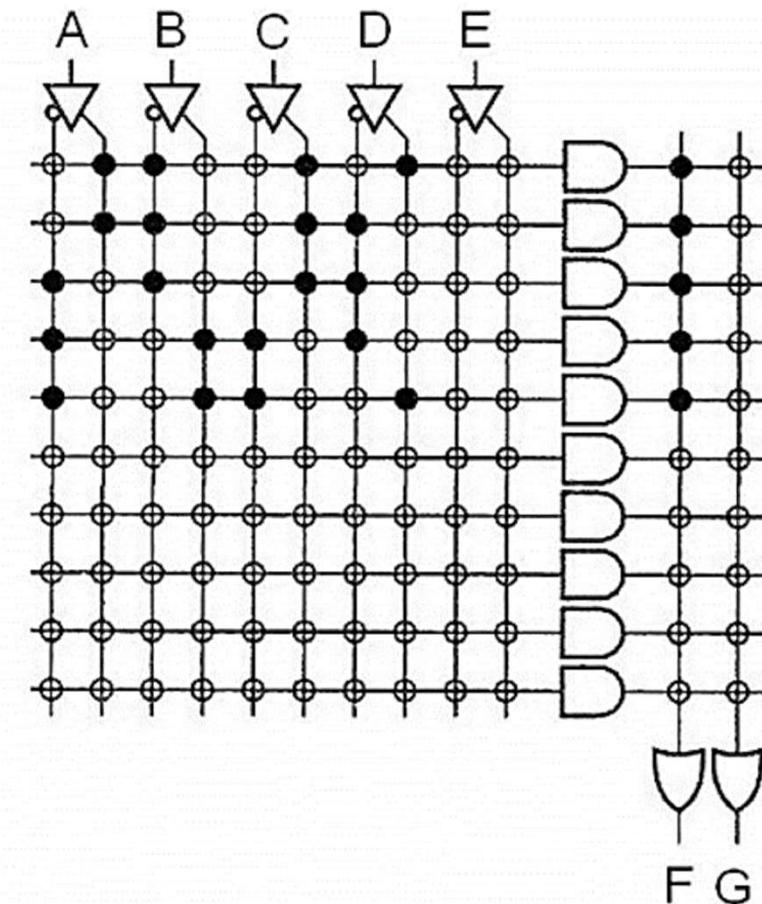
Assinale a alternativa correta:

- a) O circuito representa uma implementação em PAL da função
 $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C.$
- b) O circuito representa uma implementação em FPGA da função
 $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}.$
- c) O circuito representa uma implementação em PLA da função
 $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C.$
- d) O circuito representa uma implementação em PAL da função
 $G = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C.$
- e) O circuito representa uma implementação em PLA da função
 $G = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C.$



Programmable Logic Device

Exercício



$A\bar{B}CD$
 $A\bar{B}C\bar{D}$
 $\bar{A}\bar{B}C\bar{D}$
 $\bar{A}B\bar{C}\bar{D}$
 $\bar{A}B\bar{C}D$

$A\bar{B}C$
 $\bar{B}C\bar{D}$
 $\bar{A}B\bar{C}$

| | | | | |
|----|----|----|----|----|
| | 00 | 01 | 11 | 10 |
| 00 | | | | 1 |
| 01 | 1 | 1 | | |
| 11 | | | | |
| 10 | | 1 | 1 | |

$$F = \bar{A}B\bar{C} + \bar{B}C\bar{D} + A\bar{B}C$$

$$F = \bar{A}B\bar{C} + \bar{B}C\bar{D} + A\bar{B}C$$

Programmable Logic Device
Exercício

$$F = \bar{A}B\bar{C} + \bar{B}C\bar{D} + A\bar{B}C$$

Assinale a alternativa correta:

- a) O circuito representa uma implementação em PAL da função $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C$.
- b) O circuito representa uma implementação em FPGA da função $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}$.
- c) O circuito representa uma implementação em PLA da função $F = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C$. ?
- d) O circuito representa uma implementação em PAL da função $G = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C$.
- e) O circuito representa uma implementação em PLA da função $G = \bar{A}B\bar{C} + B\bar{C}D + A\bar{B}C$.

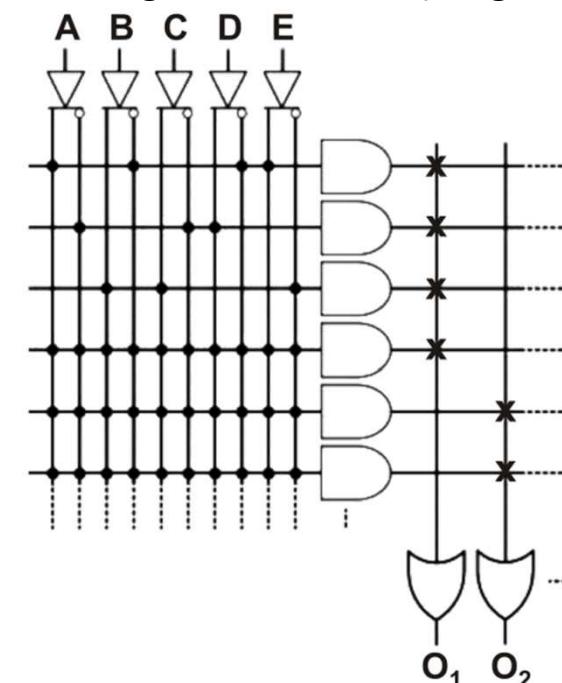
Programmable Logic Device

Exercício

- 50) (POSCOMP 2013) A figura a seguir mostra a representação de um fragmento de PAL (Programmable Array Logic):

Considerando que um “x” representa uma conexão permanente na matriz de portas OR e que um círculo negro representa uma conexão ativa na matriz de portas AND, assinale a alternativa que apresenta, corretamente, a expressão lógica correspondente à saída O_1 .

- a) $\bar{A}\bar{B}\bar{D}E + A\bar{C}\bar{D} + \bar{B}C\bar{E}$
- b) $\bar{A}BD\bar{E} + ACD + \bar{B}\bar{C}E$
- c) $A\bar{B}\bar{D}E + \bar{A}\bar{C}D + BC\bar{E}$
- d) $A\bar{B}DE + A\bar{C}D + \bar{B}C\bar{E}$
- e) $ABD\bar{E} + \bar{A}\bar{C}\bar{D} + \bar{B}\bar{C}E$



OBS.: Essa é uma representação simplificada de PAL. Cada porta AND possui 10 entradas e cada porta OR possui 4 entradas.

Programmable Logic Device

Exercício

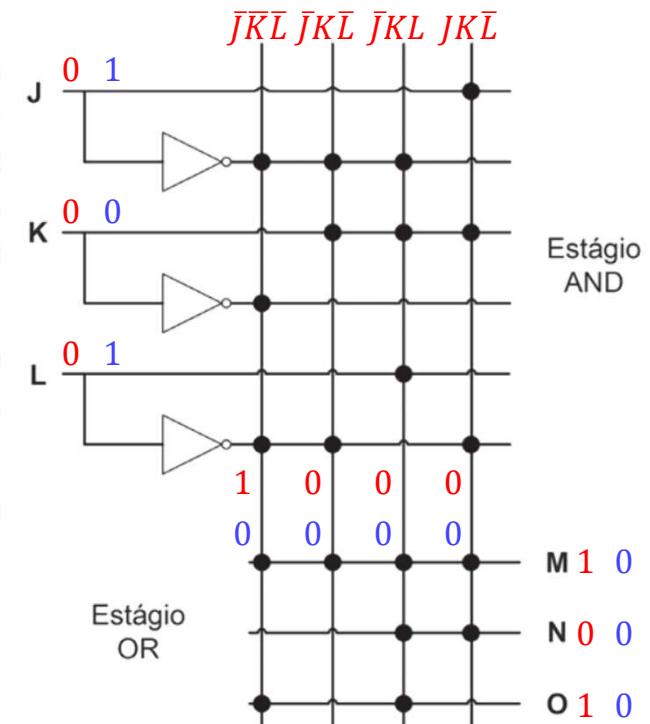
- 23) (ENADE 2014) Um componente bastante usado em circuitos lógicos é a matriz lógica programável (ou PLA, do inglês Programmable Logic Array). Uma PLA usa como entrada um conjunto de sinais e os complementos desses sinais (que podem ser implementados por um conjunto de inversores).

A lógica é implementada a partir de dois estágios: o primeiro é uma matriz de portas AND, que formam o conjunto de termos-produto (também chamados *mintermos*); o segundo estágio é uma matriz de portas OR, cada uma efetuando uma soma lógica de qualquer quantidade dos *mintermos*. Cada um dos *mintermos* pode ser o resultado do produto lógico de qualquer dos sinais de entrada ou de seus complementos.

É comum, em lugar de desenhar todas as portas lógicas de cada um dos estágios, representar apenas a posição das portas lógicas em uma matriz, conforme ilustra a figura a seguir.

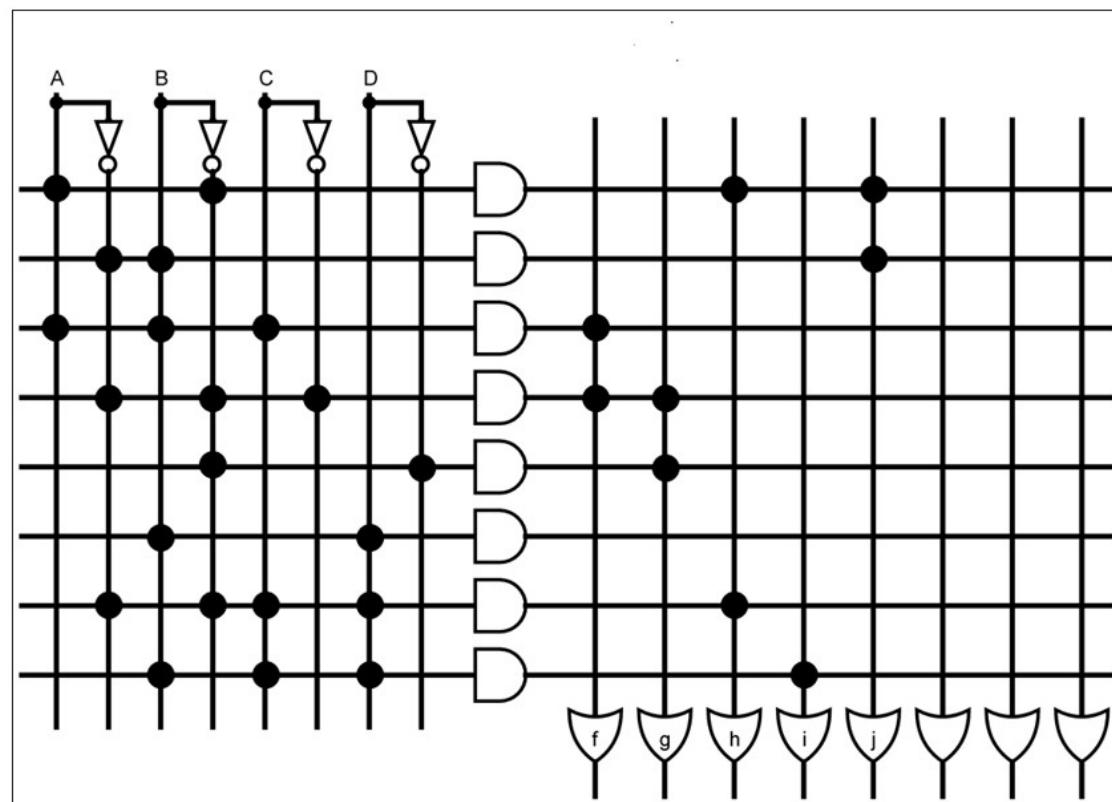
A partir da figura apresentada, infere-se que as entradas $JKL = 000$ e $JKL = 101$ levam a saídas MNO iguais, respectivamente, a

- a) 000 e 000.
- b) 000 e 010.
- c) 100 e 101.
- d)** 101 e 000.
- e) 101 e 010.



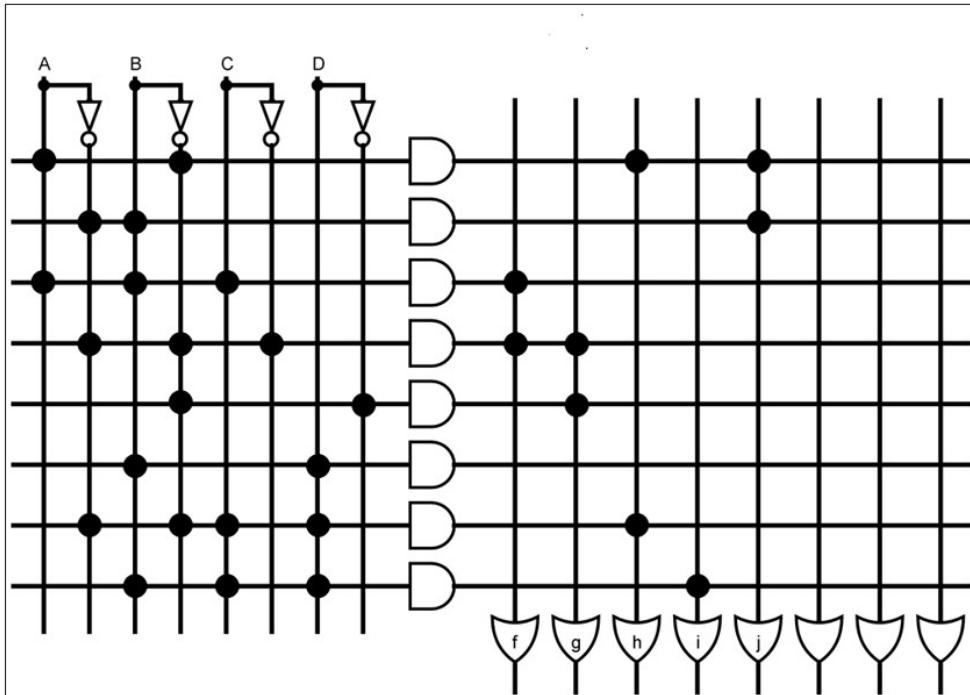
Programmable Logic Device
Exercício

- 1) Considere o seguinte *PLA*. Quais as equações para f, g, h, i e j ?



Programmable Logic Device
Exercício

- 1) Considere o seguinte *PLA*. Quais as equações para f, g, h, i e j ?



$$F = ABC + \bar{A}\bar{B}\bar{C}$$

$$G = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{D}$$

$$H = A\bar{B} + \bar{A}\bar{B}CD \quad H = A\bar{B} + \bar{B}CD$$

$$I = BCD$$

$$J = A\bar{B} + \bar{A}B \quad J = A \oplus B$$

Programmable Logic Device

Exercício

- 2) Obter as funções F_2 , F_3 e F_5 a partir de quatro entradas (I_8 , I_4 , I_2 e I_1) que indicam se a entrada binária representa um número divisível por 2, 3 e 5, respectivamente. Assuma que a entrada nunca será 0, 1 ou 2.
- a) Use *Mapa de Karnaugh* para obter as funções;
 - b) Implemente as três funções utilizando *ROM*;
 - c) Implemente as três funções utilizando *PAL*;
 - d) Implemente as três funções utilizando *PLA*;

Programmable Logic Device

Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente as três funções utilizando *ROM*;
- c) Implemente as três funções utilizando *PAL*;
- d) Implemente as três funções utilizando *PLA*;

| A | B | C | D | F2 | F3 | F5 |
|---|---|---|---|----|----|----|
| 0 | 0 | 0 | 0 | X | X | X |
| 0 | 0 | 0 | 1 | X | X | X |
| 0 | 0 | 1 | 0 | X | X | X |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |

| | | | | |
|----|----|----|----|----|
| | 00 | 01 | 11 | 10 |
| 00 | X | X | | X |
| 01 | 1 | | 1 | 1 |
| 11 | 1 | | 1 | 1 |
| 10 | 1 | | 1 | 1 |

$$F_2 = \bar{C}\bar{D} + C\bar{D} = \bar{D}$$

| | | | | |
|----|----|----|----|----|
| | 00 | 01 | 11 | 10 |
| 00 | X | X | | X |
| 01 | | 1 | | |
| 11 | | | 1 | |
| 10 | | | | 1 |

$$F_5 = \bar{A}\bar{C}D + \bar{B}CD + ABCD$$

| | | | | |
|----|----|----|----|----|
| | 00 | 01 | 11 | 10 |
| 00 | X | X | 1 | X |
| 01 | | | | 1 |
| 11 | 1 | | 1 | |
| 10 | | 1 | | |

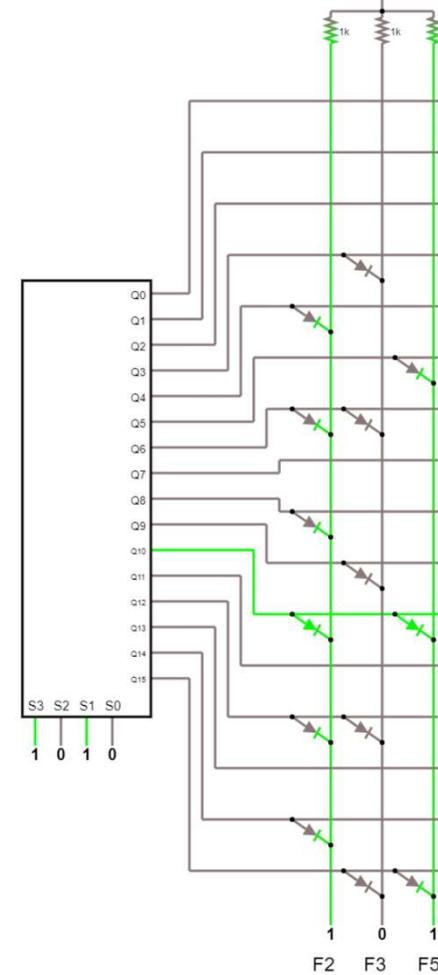
$$F_3 = \bar{A}\bar{B} + \bar{A}C\bar{D} + \bar{B}\bar{C}D + AB\bar{C}\bar{D} + ABCD$$

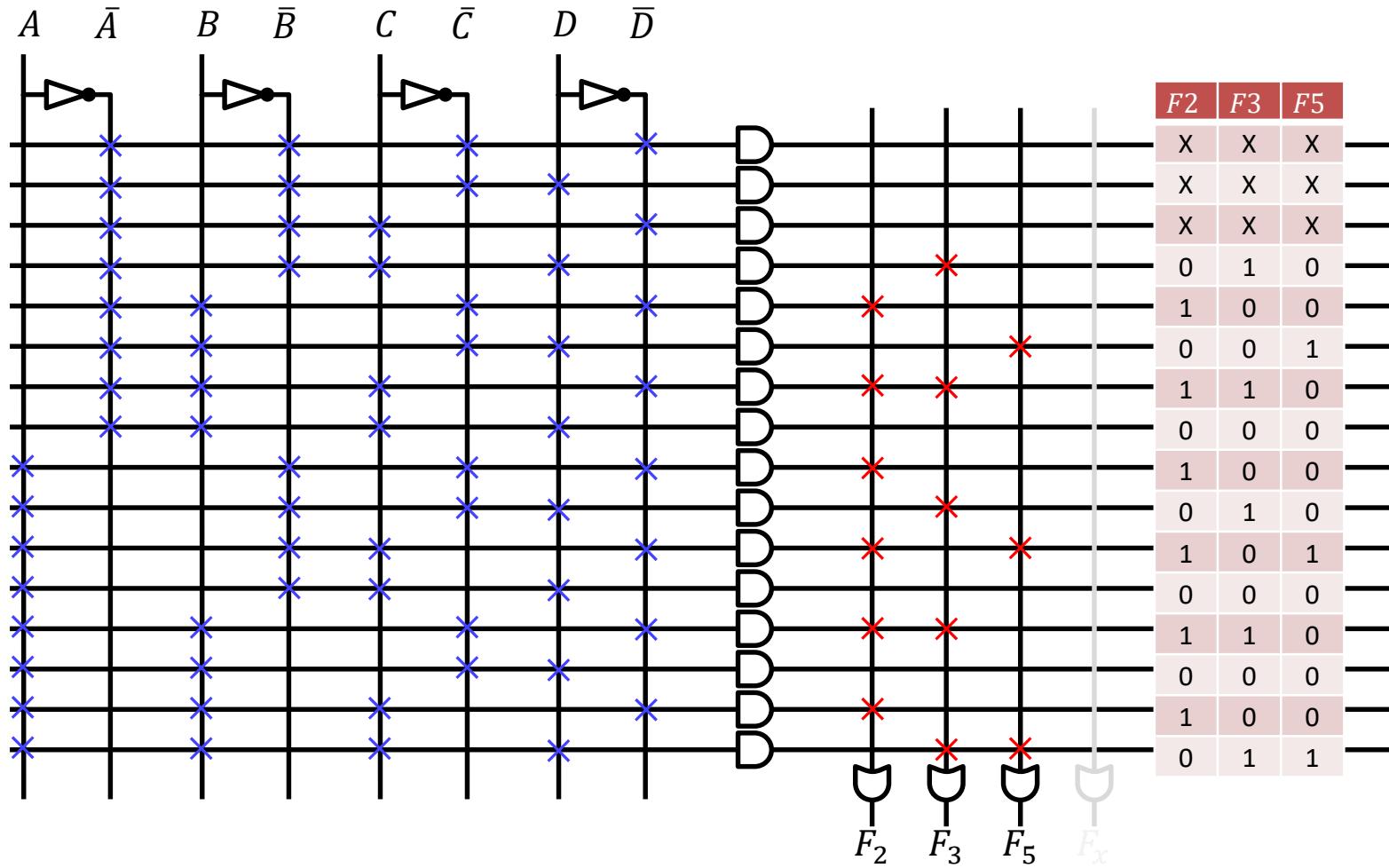
Programmable Logic Device

Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente as três funções utilizando *ROM*;
- c) Implemente as três funções utilizando *PAL*;
- d) Implemente as três funções utilizando *PLA*;

| A | B | C | D | F2 | F3 | F5 |
|---|---|---|---|----|----|----|
| 0 | 0 | 0 | 0 | X | X | X |
| 0 | 0 | 0 | 1 | X | X | X |
| 0 | 0 | 1 | 0 | X | X | X |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |





Programmable Logic Device

Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente as três funções utilizando *ROM*;
- c) **Implemente as três funções utilizando *PAL*;**
- d) Implemente as três funções utilizando *PLA*;

$$F_2 = \bar{C}\bar{D} + C\bar{D} = \bar{D}$$

$$F_3 = \bar{A}\bar{B} + \bar{B}\bar{C}D + \bar{A}C\bar{D} + A\bar{B}\bar{C}D + ABCD$$

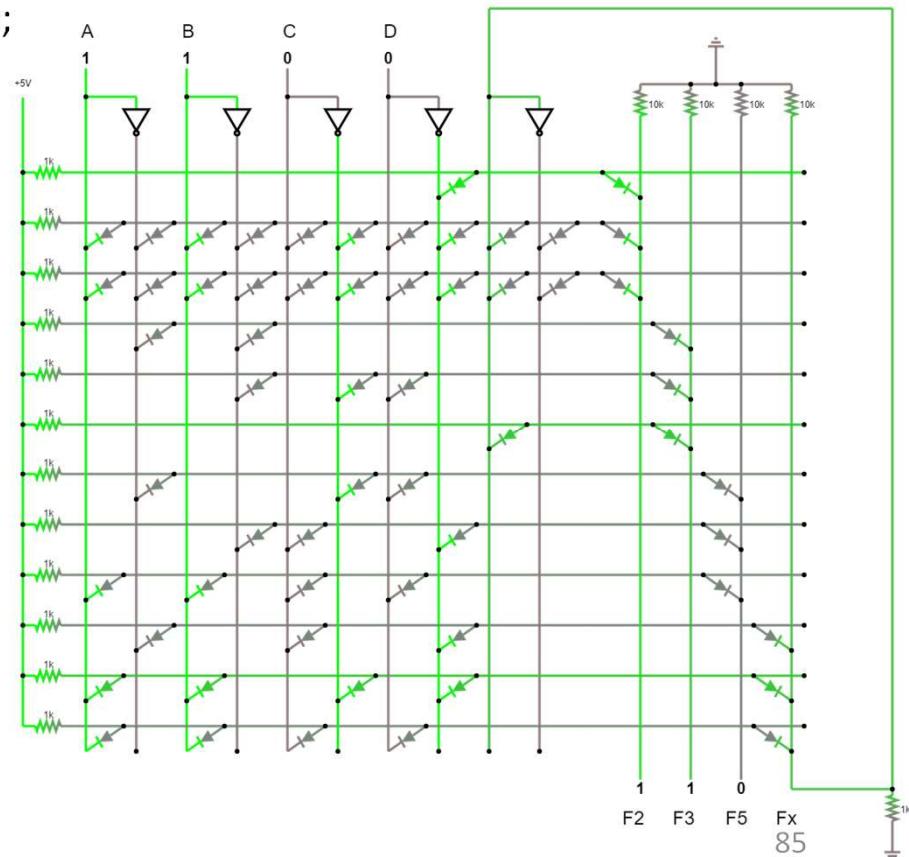
$$F_5 = \bar{A}\bar{C}D + \bar{B}\bar{C}\bar{D} + ABCD$$

$$F_2 = \bar{D}$$

$$F_3 = \bar{A}\bar{B} + \bar{B}\bar{C}D + F_x$$

$$F_x = \bar{A}C\bar{D} + A\bar{B}\bar{C}D + ABCD$$

$$F_5 = \bar{A}\bar{C}D + \bar{B}\bar{C}\bar{D} + ABCD$$



85

Programmable Logic Device

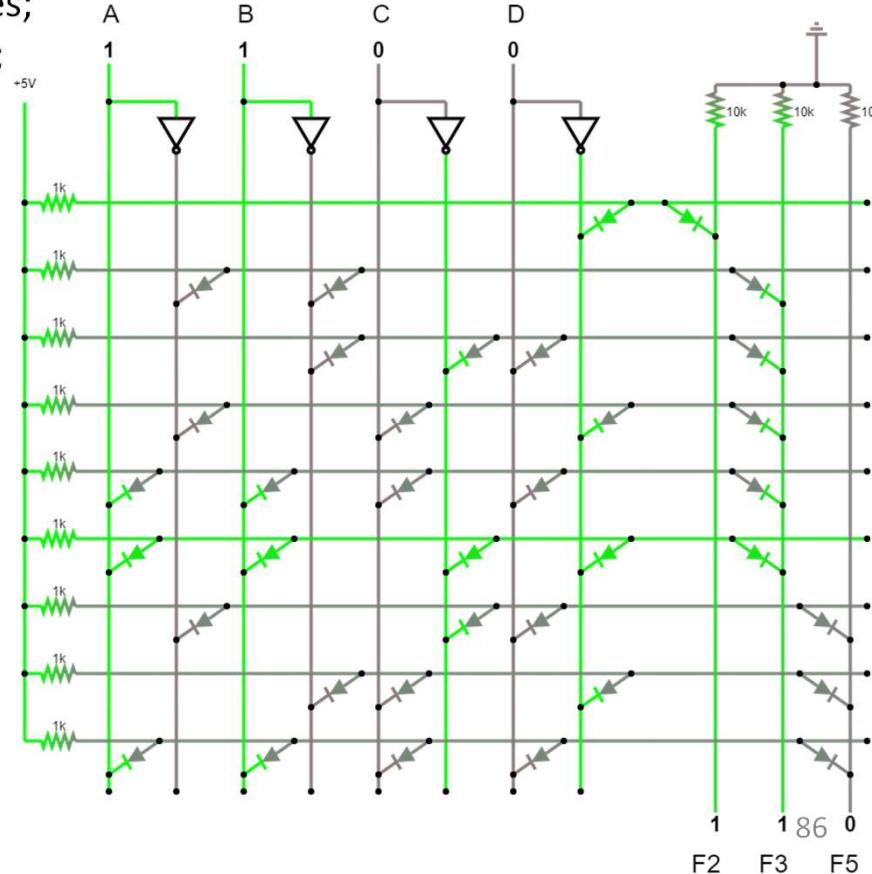
Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente as três funções utilizando *ROM*;
- c) Implemente as três funções utilizando *PAL*;
- d) Implemente as três funções utilizando *PLA*;**

$$F_2 = \bar{C}\bar{D} + C\bar{D} = \bar{D}$$

$$F_3 = \bar{A}\bar{B} + \bar{B}\bar{C}D + \bar{A}\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + ABCD$$

$$F_5 = \bar{A}\bar{C}D + \bar{B}\bar{C}\bar{D} + ABCD$$



Programmable Logic Device

Exercício

- 3) Implementar uma função que avalia uma entrada de 4 bits e detecta se ela forma uma sequência ininterrupta de 1's (uma única transição de $0 \Rightarrow 1$ ou de $1 \Rightarrow 0$).

Os 7 casos são: 0001, 0011, 0111, 1111, 1110, 1100, 1000.

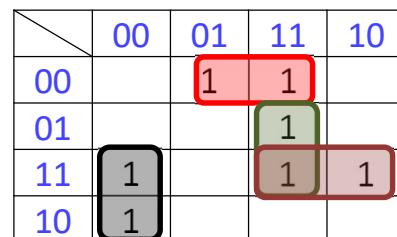
- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente a solução utilizando *ROM*;
- c) Implemente a solução utilizando *PAL*;
- d) Implemente a solução utilizando *PLA*;

Programmable Logic Device

Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente a solução utilizando *ROM*;
- c) Implemente a solução utilizando *PAL*;
- d) Implemente a solução utilizando *PLA*;

| A | B | C | D | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

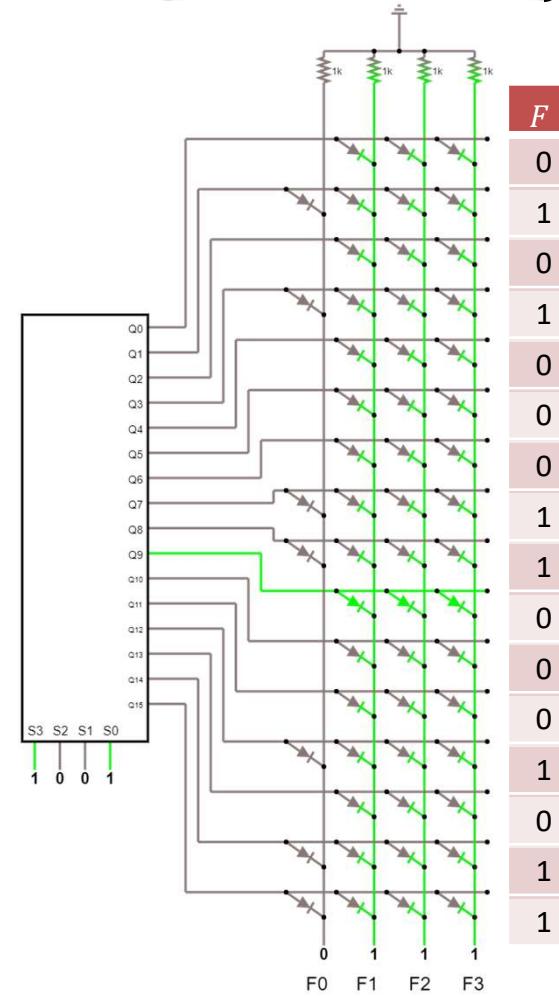


$$F = \bar{A}\bar{B}D + BCD + ABC + A\bar{C}\bar{D}$$

Programmable Logic Device

Exercício

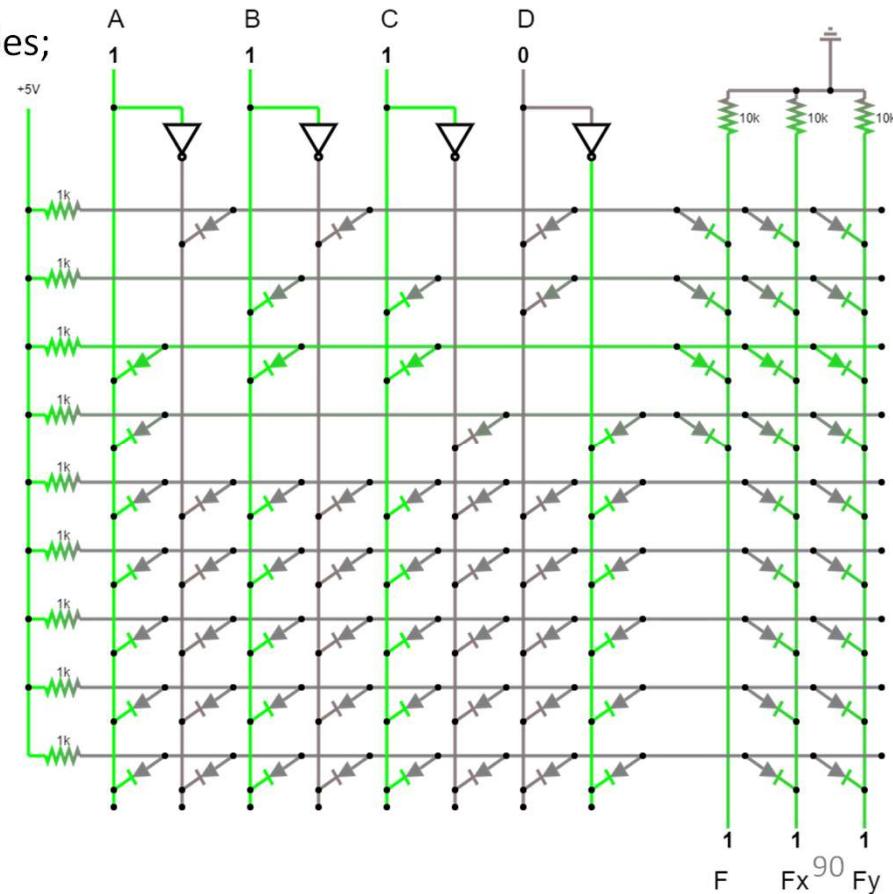
- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente a solução utilizando *ROM*;
- c) Implemente a solução utilizando *PAL*;
- d) Implemente a solução utilizando *PLA*;



Programmable Logic Device Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente a solução utilizando *ROM*;
- c) Implemente a solução utilizando *PAL*;
- d) **Implemente a solução utilizando *PLA*;**

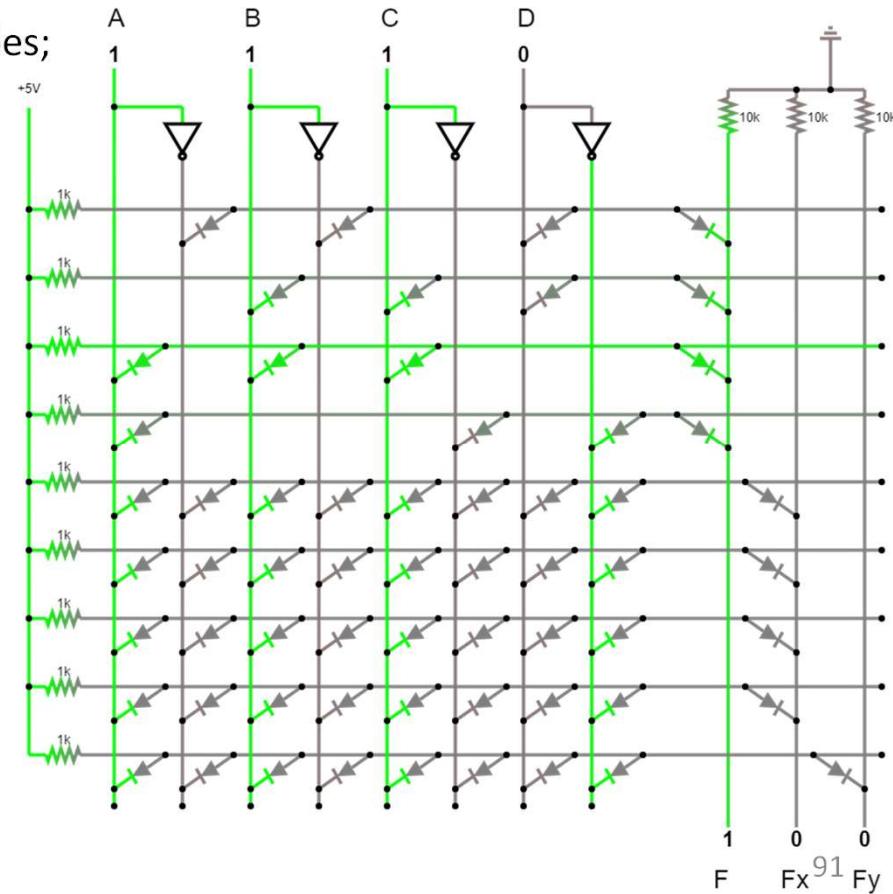
$$F = \bar{A}\bar{B}D + BCD + ABC + A\bar{C}\bar{D}$$



Programmable Logic Device Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
- b) Implemente a solução utilizando *ROM*;
- c) **Implemente a solução utilizando *PAL*;**
- d) Implemente a solução utilizando *PLA*;

$$F = \bar{A}\bar{B}D + BCD + ABC + A\bar{C}\bar{D}$$

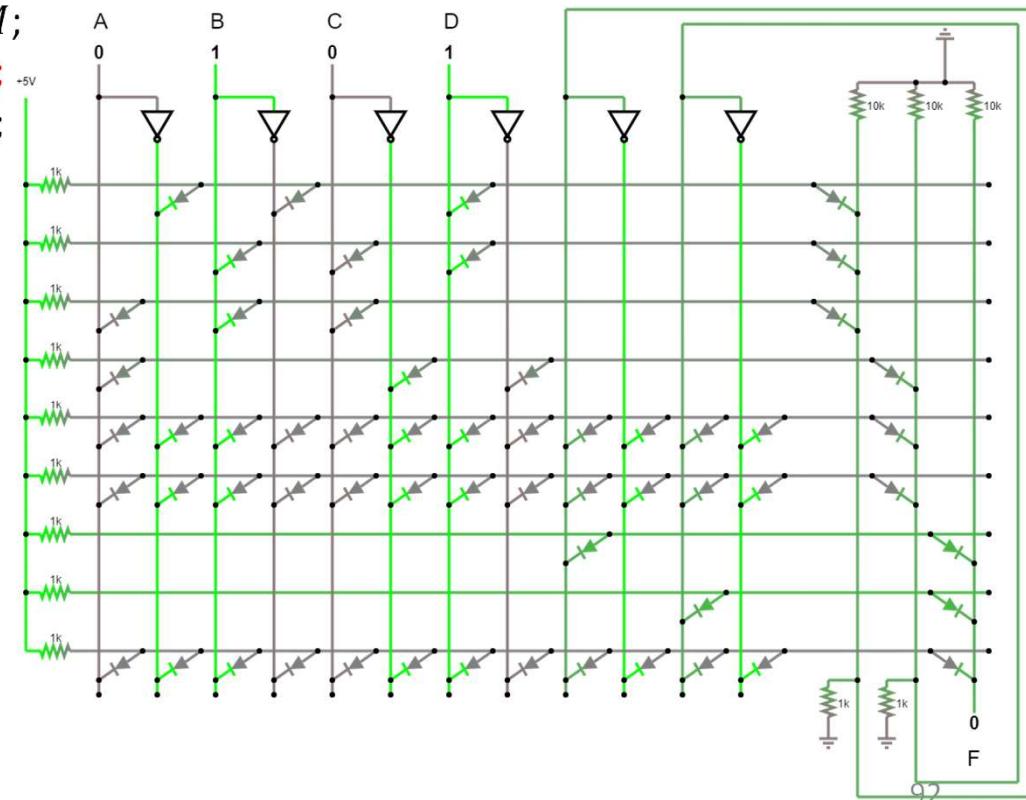


Programmable Logic Device

Exercício

- a) Use *Mapa de Karnaugh* para obter as funções;
 - b) Implemente a solução utilizando *ROM*;
 - c) **Implemente a solução utilizando *PAL*;**
 - d) Implemente a solução utilizando *PLA*;

$$F = \bar{A}\bar{B}D + BCD + ABC + A\bar{C}\bar{D}$$



Programmable Logic Device

Exercício

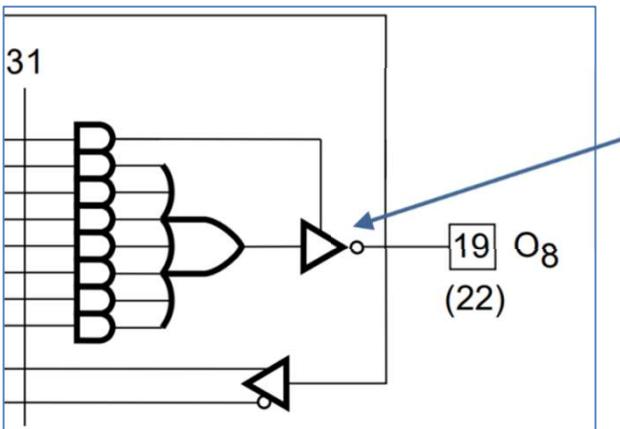
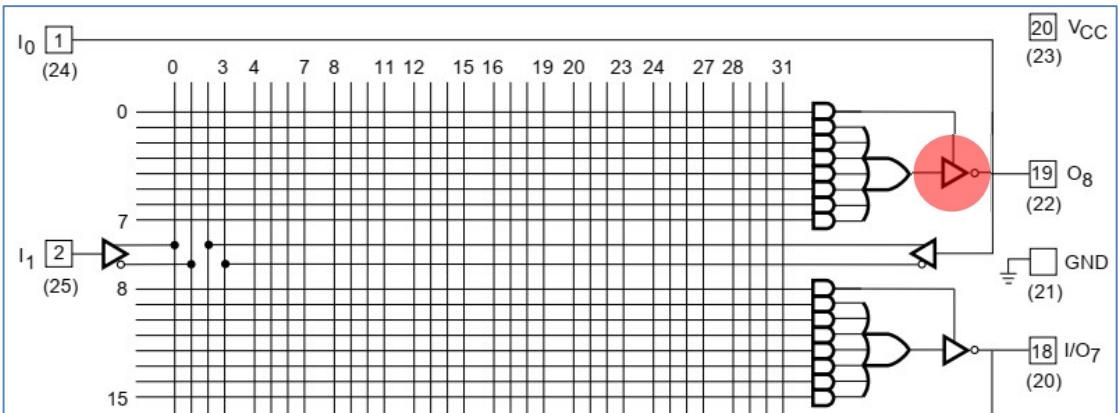
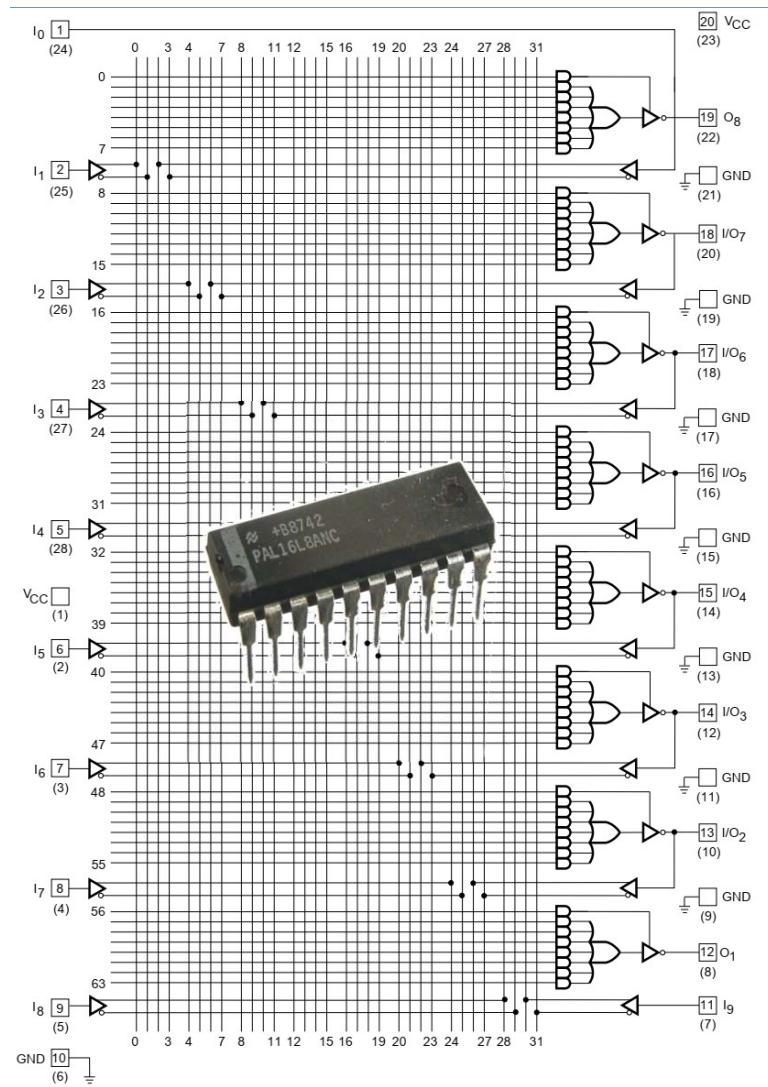
- 4) Implementar um conversor de código do padrão *XS3* para *BCD*.
- Use *Mapa de Karnaugh* para obter as funções;
 - Implemente a solução utilizando *PLA*;

A tabela verdade é dada por:

| XS3 (EXC de 3) | | | | BCD | | | |
|----------------|---|---|---|-----|---|---|---|
| A | B | C | D | P | Q | R | S |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

PAL

PAL16L8 Family

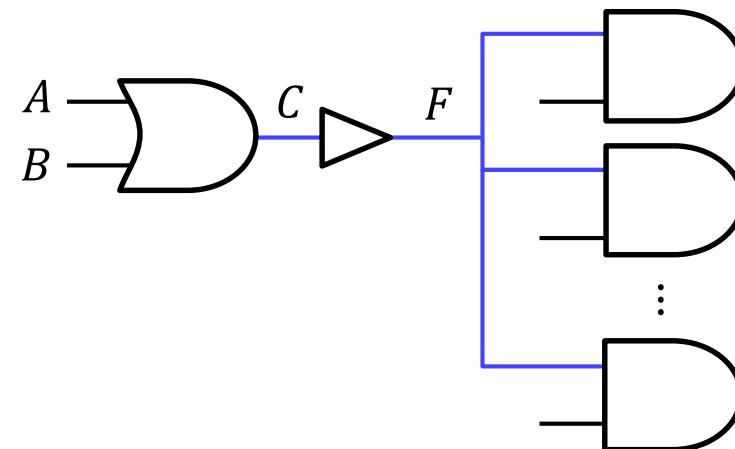


Three-State Buffer
Tri-State Buffer
3-State Buffer

<http://www.applelogic.org/files/PAL16R8.pdf>

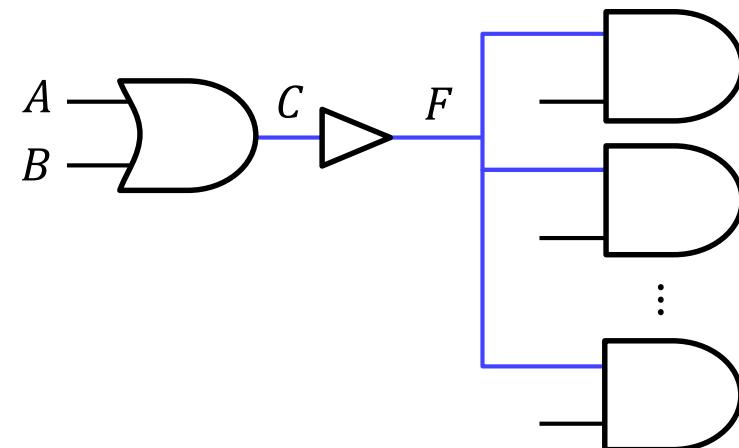
Three-State Buffer

- A saída de uma porta lógica pode servir como entrada de um número limitado de outras portas lógicas;
- A partir de um certa quantidade há uma degradação no desempenho do sistema digital;
- Um buffer simples pode ser utilizado para restabelecer o nível da saída de uma porta lógica;



Three-State Buffer

- O símbolo do buffer se assemelha a porta *NOT*, porém sem o círculo na saída;
- Este é um buffer não-inversor e os valores lógicos de entrada e saída do buffer são iguais ($F == C$);
- Normalmente, um circuito lógico não funcionará adequadamente se uma saída lógica for conectada diretamente a duas ou mais portas ou dispositivos lógicos;

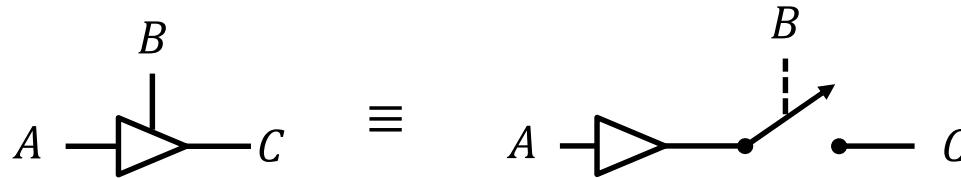


Three-State Buffer

- Por exemplo, se uma porta com uma saída 0 (baixa tensão) e outra com saída 1 (alta tensão) são conectadas, a tensão resultante pode ser algum valor intermediário que não representa claramente 0 ou 1.
- Em alguns casos, podem ocorrer danos às portas se as saídas forem conectadas diretamente;
- O uso de lógica *Three-State* permite que saídas de duas ou mais portas ou outro dispositivo lógico possam ser conectados;

Three-State Buffer

- A seguinte figura ilustra o equivalente lógico de *Three-State*:

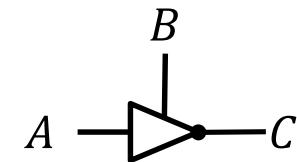
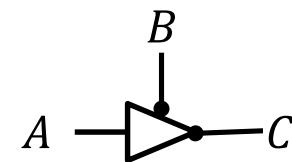
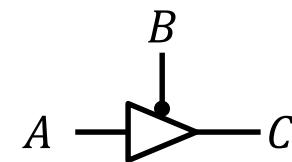
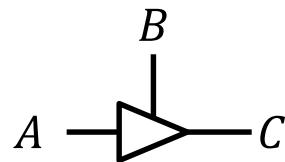


| B | A | C |
|---|---|---|
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- Para B igual a 1, o *Three-State* estará habilitado, e a saída C será igual a A ;
- Para B igual a 0, a saída C atua como um circuito aberto;
- De fato, quando B é 0, a saída C é efetivamente desconectada da saída do buffer e nenhuma corrente irá fluir;
- Esta condição de isolamento é muitas vezes referenciada como um estado *Hi-Z* (*alta impedância*) da saída, uma vez que o circuito oferece uma resistência ou impedância muito alta ao fluxo de corrente;

Three-State Buffer

- Variações do *Three-State Buffer*;



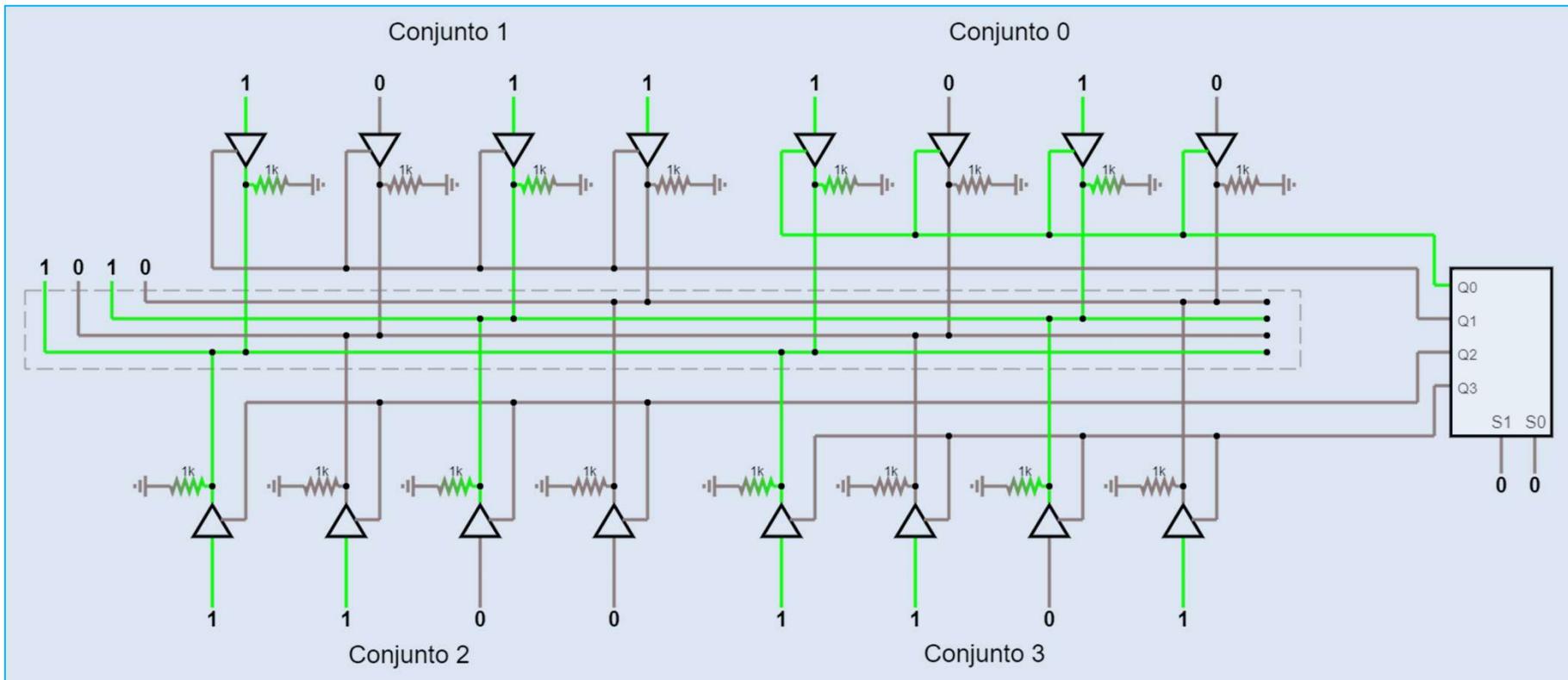
| B | A | C |
|---|---|---|
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| B | A | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | Z |
| 1 | 1 | Z |

| B | A | C |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | Z |
| 1 | 1 | Z |

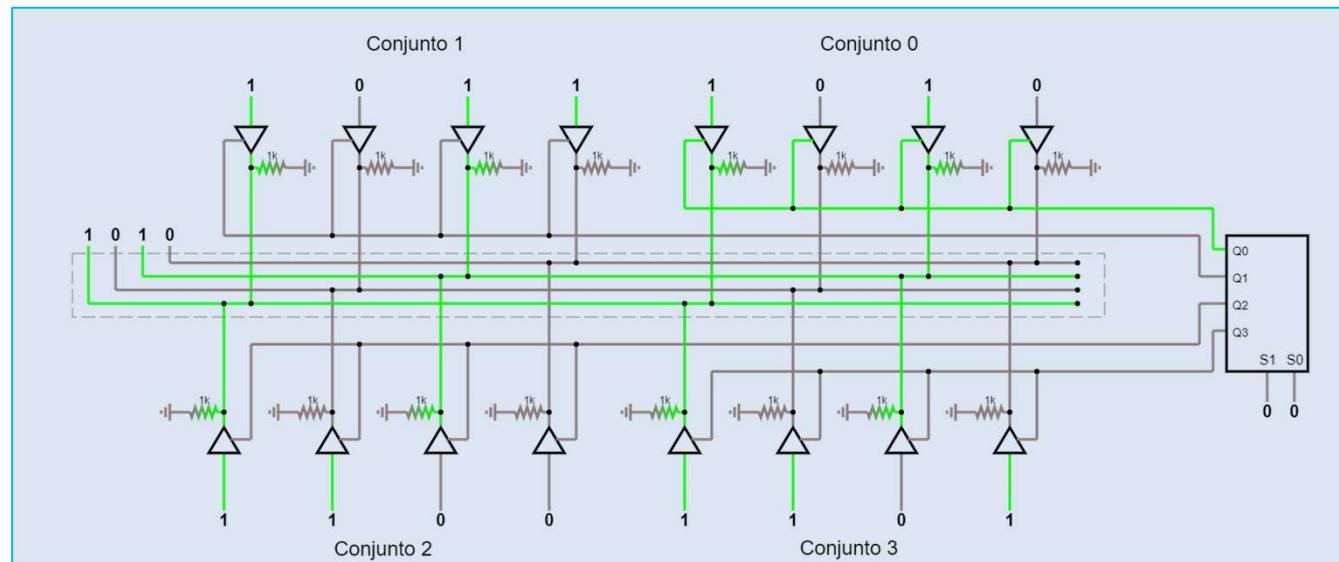
| B | A | C |
|---|---|---|
| 0 | 0 | Z |
| 0 | 1 | Z |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Three-State Buffer



Three-State Buffer

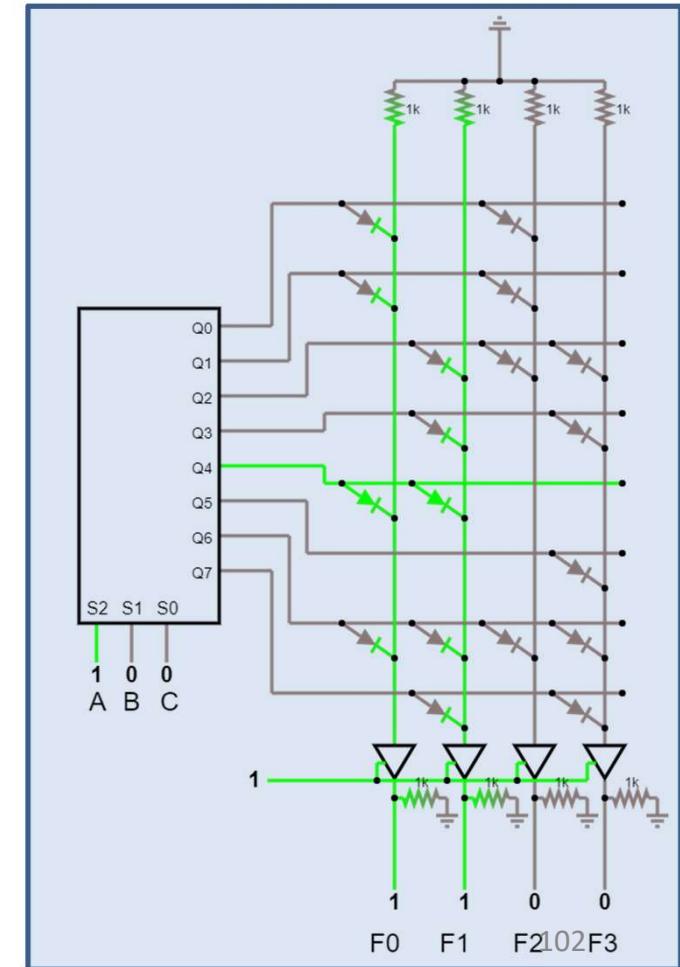
- Neste circuito, os *Three–States* estão sendo utilizados para controlar o acesso ao barramento de dados;
- Um *Decoder* seleciona o conjunto de dados habilitado a utilizar o barramento;



<https://tinyurl.com/yzvzneme>

Three-State Buffer

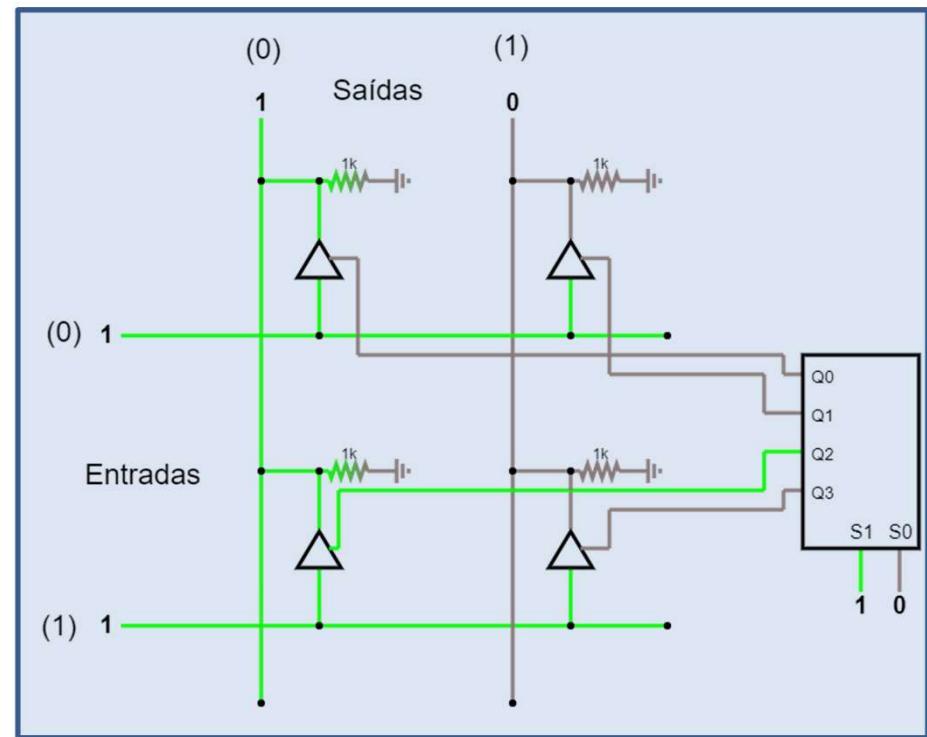
- Neste circuito, os *Three-States* estão sendo utilizados para controlar o acesso da *ROM* ao barramento de dados;
- Uma entrada digital habilita a transferência do dado selecionado da *ROM*;



<https://tinyurl.com/yewvpsmk>

Three-State Buffer

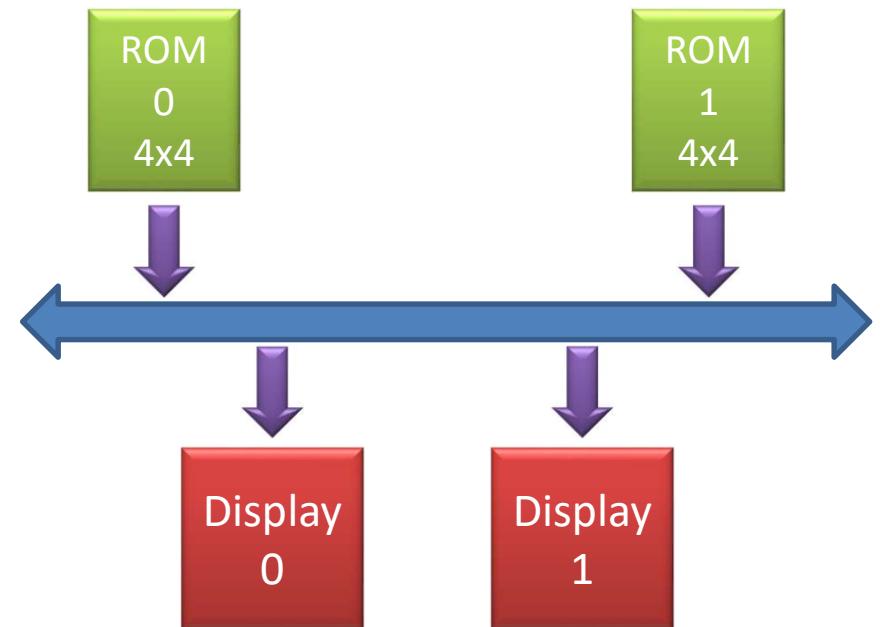
- Neste circuito, os *Three-States* estão sendo utilizados para fazer o roteamento entre duas entradas possíveis para duas saídas possíveis;
- O *Decoder* faz o papel da lógica de chaveamento;
- A entrada S_1 indica qual a origem e S_0 qual a saída escolhida;



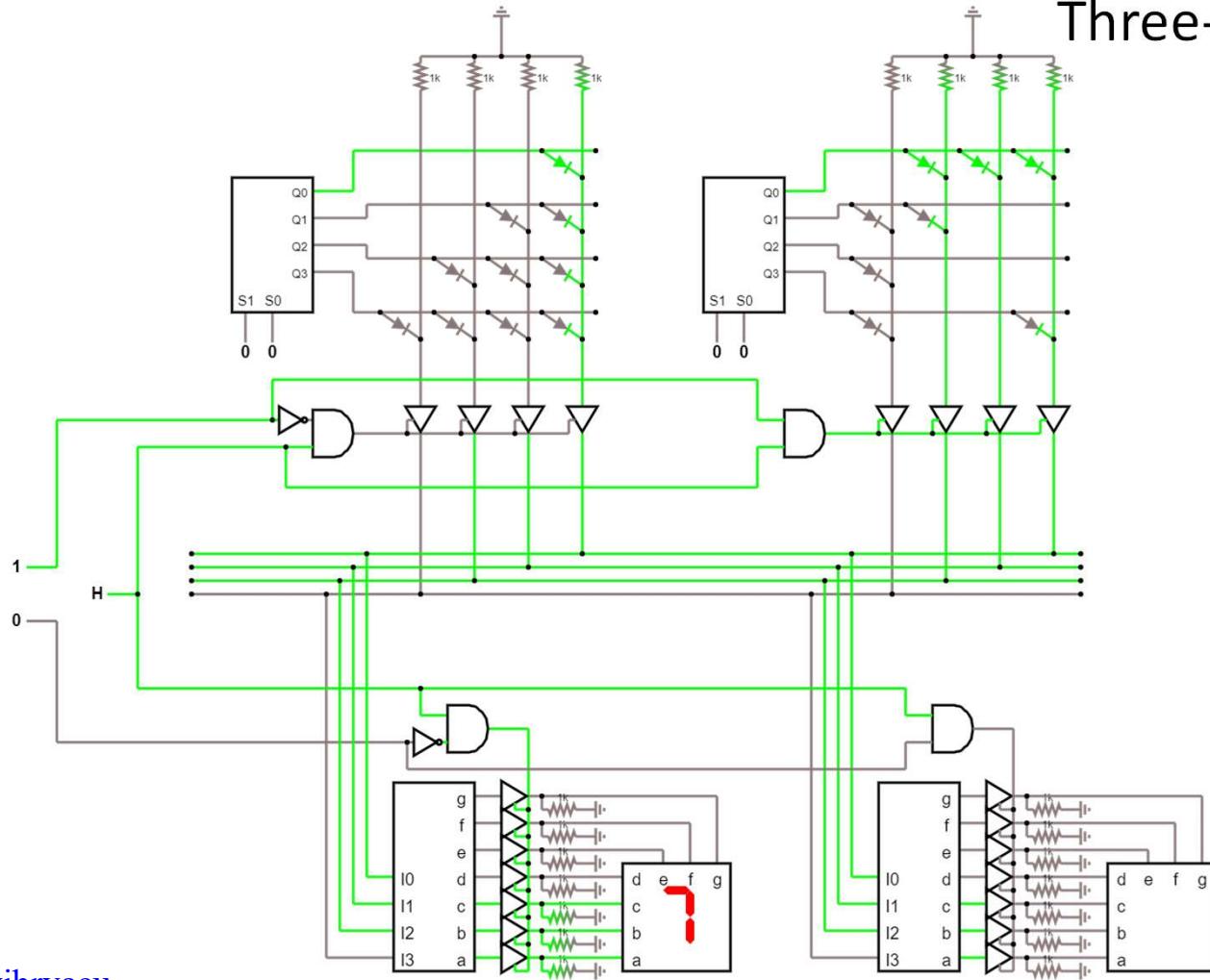
Three-State Buffer

Exercício

- 1) Um determinado circuito deve dispor de duas *ROMs*, dois Displays de 7 Segmentos e um barramento compartilhado, além dos elementos de controle.
- Construir um circuito completo com os elementos de controle que permita selecionar uma *ROM* e um Display;
 - Cada uma das *ROMs* terá um endereço de memória pré-ajustado;
 - A informação da *ROM* escolhida será apresentada no Display escolhido;
 - O Display não selecionado deve ficar apagado;
 - A informação só aparecerá no Display quando um sinal *WRITE* estiver habilitado;
 - A solução deverá contemplar *Three-State Buffers*.

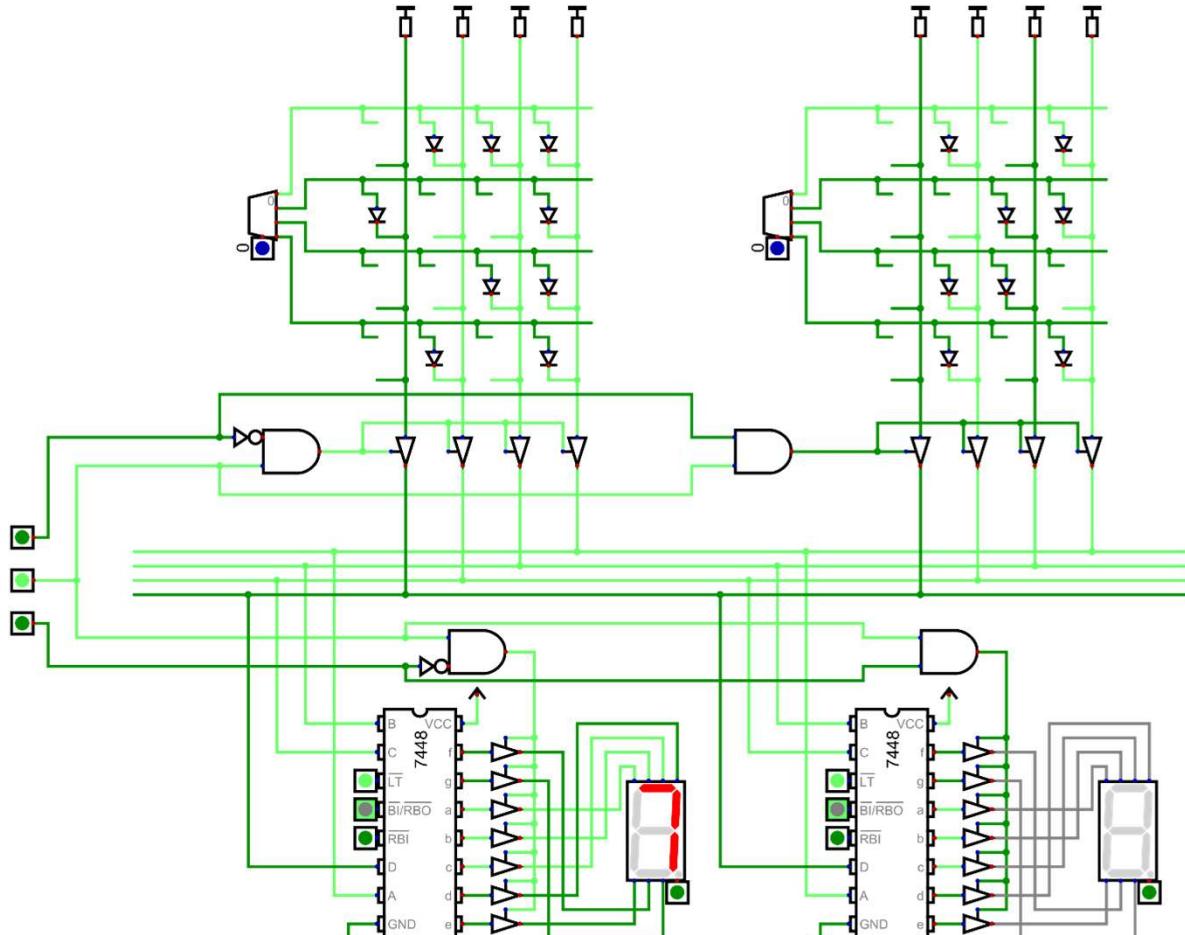


Three-State Buffer *Exercício*



<https://tinyurl.com/yjbrvacu>

Three-State Buffer *Exercício*



<https://github.com/hneemann/Digital/releases/latest/download/Digital.zip>

Sumário

- 1. Revisão – Sistemas de Numeração
- 2. Revisão – Representação de Dados
- 3. Revisão – Operações com Binários
- 4. Álgebra Booleana
- 5. Simplificação de Expressões
- 6. Mapa de Karnaugh
- 7. Elementos Lógicos Universais
- 8. Circuitos Combinacionais
- 9. Circuitos Sequenciais

- 1. Somador / Subtrator
- 2. Comparadores
- 3. Codificador/decodificador
- 4. Multiplexador/Demux
- 5. Geradores de paridade
- 6. Circuitos Específicos
- 7. Multiplicadores / Divisores
- 8. ULA
- 9. PLD/PLA/PAL/FPGA/ROM

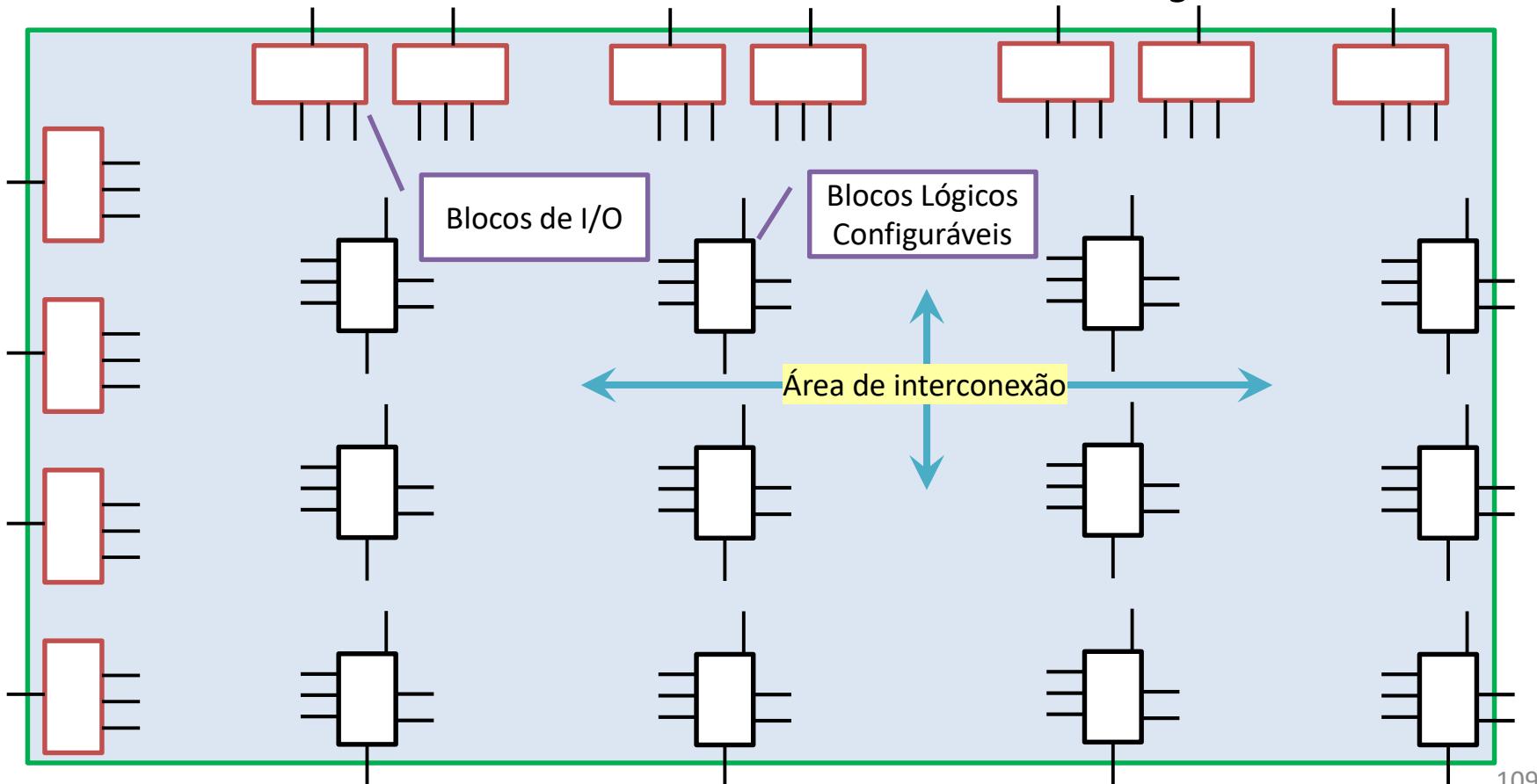
FPGA

Field-Programmable Gate Arrays

- Um *FPGA* contém uma matriz de blocos de células lógicas idênticos com interconexões programáveis;
- O usuário pode programar tanto as células lógicas, quanto as conexões entre estas células;
- A seguinte figura exemplifica um layout de um *FPGA* típico;

FPGA

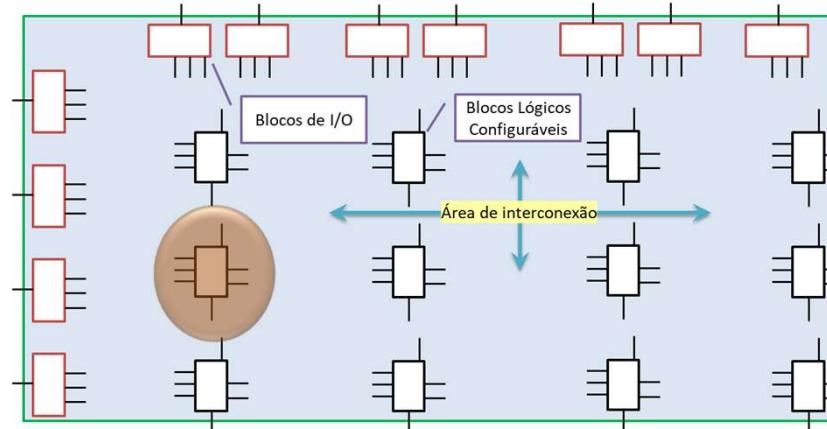
Field-Programmable Gate Arrays



FPGA

Field-Programmable Gate Arrays

- O interior do *FPGA* consiste de uma matriz de células lógicas, também chamadas de Blocos Lógicos Configuráveis (*CLBs*);
- A matriz de *CLBs* é envolvida por um anel de blocos de interface de *E/S*;
- Esses blocos de *E/S* conectam os sinais dos *CLB* aos pinos de entrada do *CI*;
- O espaço entre os *CLBs* é formado por trilhas usadas para rotear as conexões entre as saídas e entradas do *CLB*;

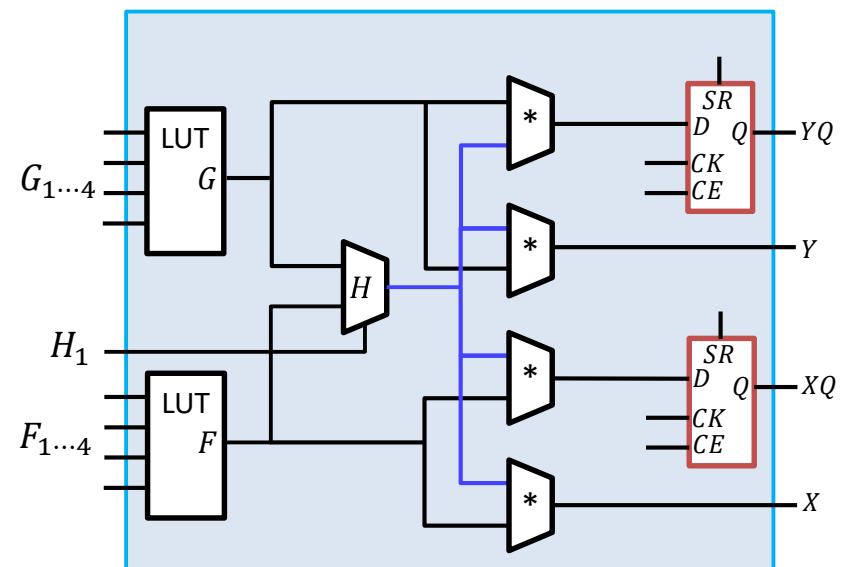


Roth Jr, Charles H; Kinney, Larry L; Fundamentals of Logic Design, Seventh Edition. Cengage Learning, 2013 .

FPGA

Field-Programmable Gate Arrays

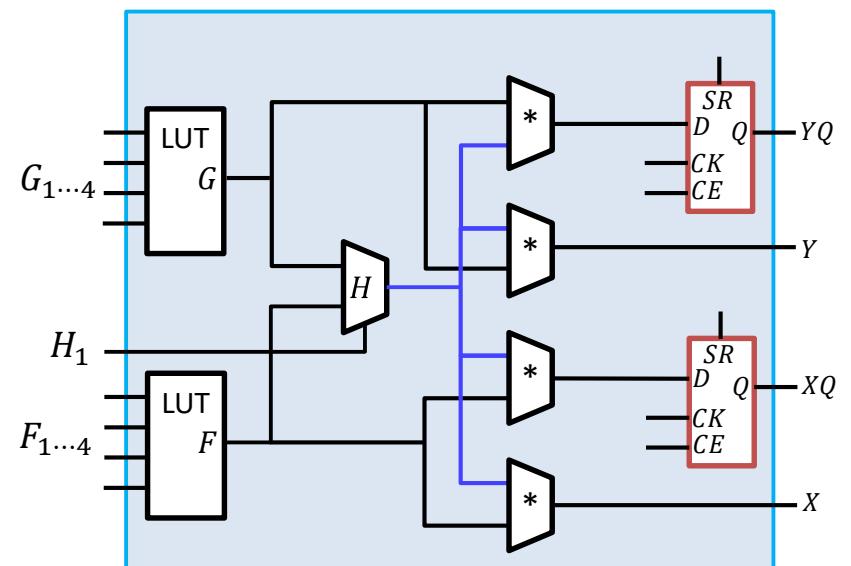
- A figura ao lado mostra uma versão simplificada de um *CLB*;
- Este *CLB* contém dois geradores de função, dois *flip-flops* e vários multiplexadores para rotear os sinais internamente ao *CLB*;
- Cada gerador de função tem quatro entradas, podendo implementar funções de até quatro variáveis;
- Os geradores de função são implementados como *Look-up Table (LUTs)*;



FPGA

Field-Programmable Gate Arrays

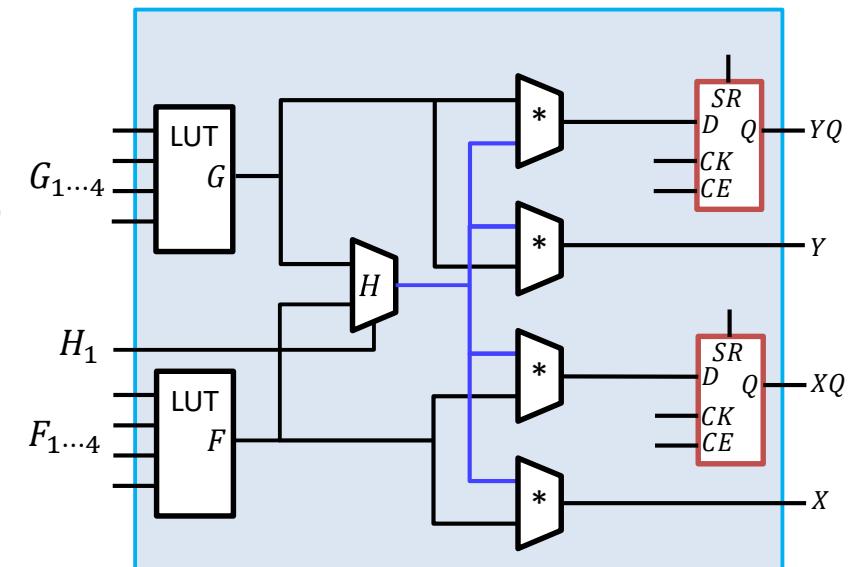
- Uma *LUT* de quatro entradas é essencialmente uma *ROM* reprogramável com palavras de 16×1 bit;
- A *LUT* não realiza operação lógica, apenas disponibiliza a tabela verdade da função programada;
- Esta *ROM* armazena a tabela verdade para a função que está sendo gerada;
- O multiplexador *H* seleciona *F* ou *G* dependendo do valor de H_1 ;
- O *CLB* tem duas saídas combinacionais (*X* e *Y*) e duas saídas *flip-flop* (*XQ* e *YQ*).



FPGA

Field-Programmable Gate Arrays

- As saídas X e Y e as entradas do *flip-flop* são selecionadas por multiplexadores programáveis;
- As entradas selecionadas desses *MUXes* são programadas quando o *FPGA* for configurado;
- Por exemplo, a saída X pode vir do gerador de função F e a saída Y do multiplexador H ;
- A operação dos *flip-flops* será descrita no módulo de circuitos sequenciais;



Sumário

1. Revisão – Sistemas de Numeração
2. Revisão – Representação de Dados
3. Revisão – Operações com Binários
4. Álgebra Booleana
5. Simplificação de Expressões
6. Mapa de Karnaugh
7. Elementos Lógicos Universais
8. Circuitos Combinacionais
9. Circuitos Sequenciais
 1. Latches
 2. Flip-Flop
 3. Registradores
 4. Contadores
 5. Máquina de Estados
 6. *Memória RAM*

Memória RAM

- Vimos anteriormente as memórias *ROM*, cujas características principais são:
 - ✓ Armazenamento permanente (não-volátil);
 - ✓ Utilização em diversas aplicações, tais como:
 - ❑ Microprogramação;
 - ❑ Rotinas de biblioteca;
 - ❑ Programação de sistema (*BIOS – Basic Input Output System*);
 - ❑ Tabelas de funções lógicas;

Memória RAM

- Uma segunda categoria, a das memórias *RAM*, é necessária para dar flexibilidade aos sistemas digitais;
- Uma *RAM* é uma memória onde os dados podem ser lidos ou gravados por meio de acesso direto ao endereço selecionado – *Random Access Memory*;
- O processo de gravação substitui o dado anterior pela nova informação;
- O processo de leitura permite obter a informação armazenada em um determinado endereço da *RAM*, sem que o dado seja perdido;
- Por ser de natureza volátil, a *RAM* é normalmente usada para armazenamento de dados de curto prazo;

Memória RAM

- A família *RAM* é dividida em duas categorias principais:
 - ✓ *RAM Estática (SRAM)*; e
 - ✓ *RAM Dinâmica (DRAM)*;
- As *SRAMs* geralmente usam *latches* ou algum arranjo com transistores como elemento de armazenamento e podem, portanto, reter os dados indefinidamente, enquanto conectadas a uma fonte de tensão;
- As *DRAMs* usam capacitores como elementos de armazenamento e não podem reter dados por muito tempo sem que os capacitores sejam recarregados por um processo chamado *refresh*;

Memória RAM

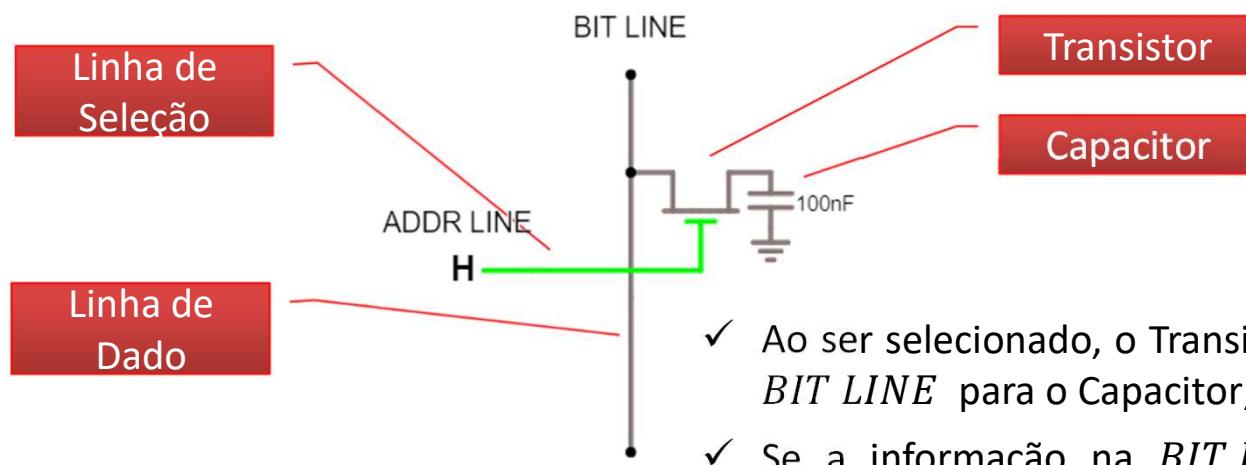
- Tanto as *SRAMs* quanto as *DRAMs* perderão os dados armazenados quando a alimentação for removida e, portanto, são classificadas como memórias voláteis;
- Os dados podem ser lidos muito mais rápido em *SRAMs* do que em *DRAMs*;
- No entanto, as *DRAMs* apresentam algumas vantagens em relação as *SRAMs*:
 - ✓ podem armazenar muito mais dados para um mesmo espaço físico;
 - ✓ custo menor, devido a maior simplicidade do circuito de armazenamento do bit.

Memória RAM *DRAM*

- As células de uma *DRAM* utilizam capacitor como elemento de armazenamento do bit de dado;
- A vantagem desse tipo de célula é a simplicidade, e consequentemente a possibilidade de construção de chips com extensos *arrays* de memória, resultando em menor custo por bit;
- A desvantagem é que o capacitor não pode manter sua carga por um longo período de tempo e perderá o bit de dados armazenado, a menos que sua carga seja atualizada periodicamente;
- A atualização da carga dos capacitores requer circuitos adicionais e complica a operação da *DRAM*.

Memória RAM *DRAM*

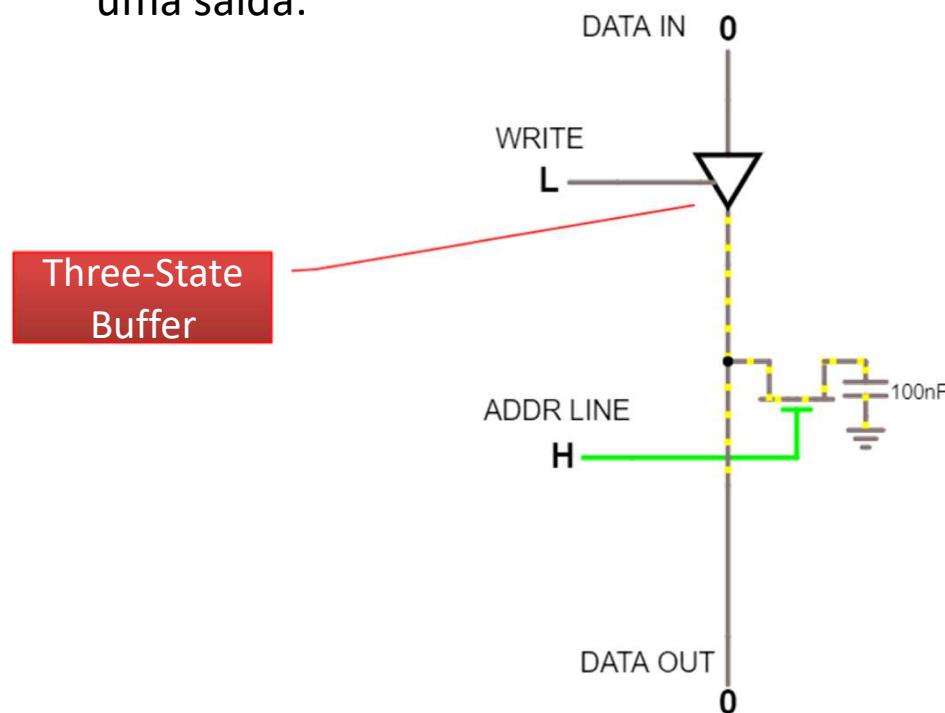
- O seguinte circuito equivale a uma célula capaz de armazenar um bit de dado:



- ✓ Ao ser selecionado, o Transistor transfere carga da *BIT LINE* para o Capacitor;
- ✓ Se a informação na *BIT LINE* for 1, então o Capacitor será carregado;
- ✓ Caso contrário, será descarregado;

Memória RAM DRAM

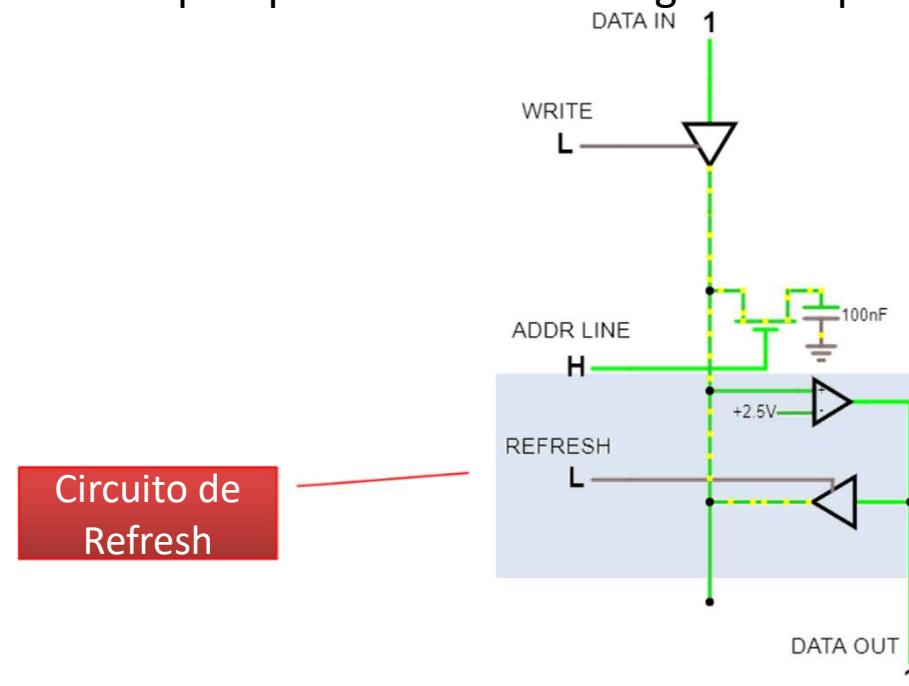
- Completando o circuito, adicionaremos uma entrada controlada por um *Three-State* e uma saída:



- ✓ O *Three-State* é controlado por uma chave instantânea, que ao ser acionada, transfere o Dado de entrada para a memória;
- ✓ Imediatamente esta informação estará disponível na saída;
- ✓ Neste exemplo, foi colocado um capacitor de maior capacidade para que a retenção da informação fosse maior no simulador utilizado.

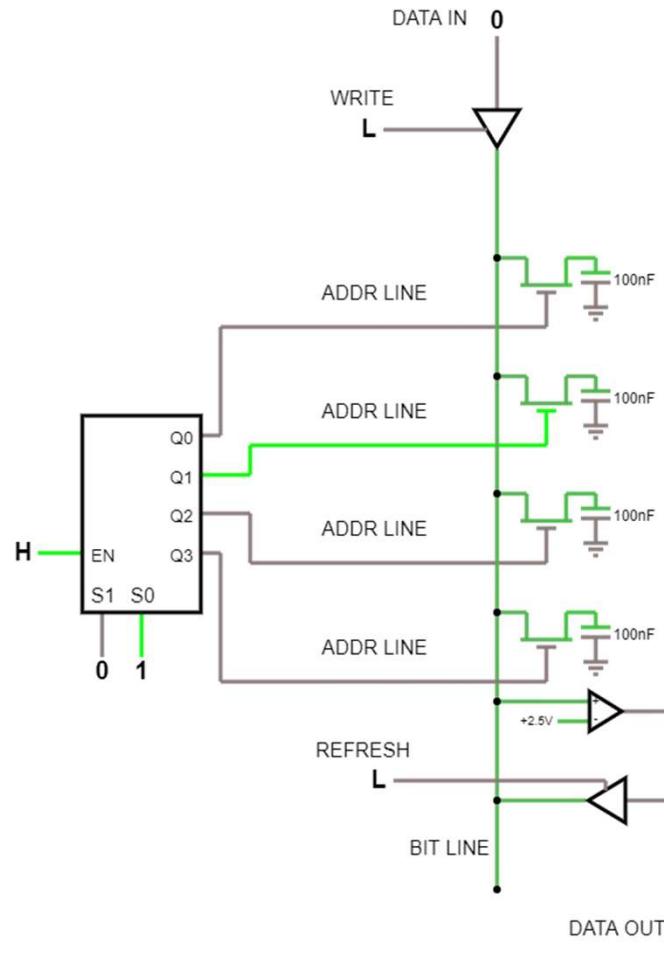
Memória RAM DRAM

- O Capacitor vai perdendo carga com o tempo, o que requer um processo de *refresh*, sempre que um limite de carga é ultrapassado:



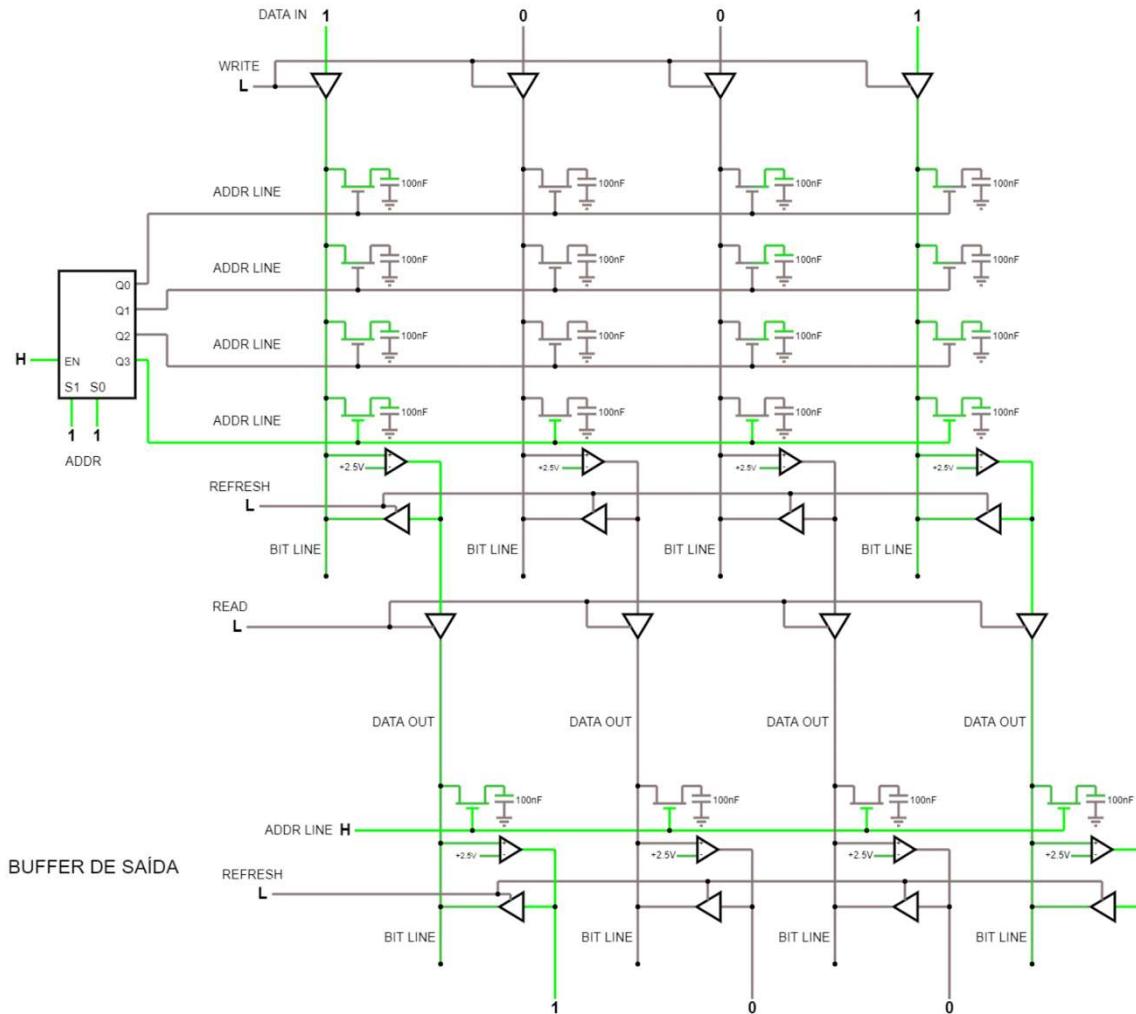
- ✓ O circuito de *refresh* compara o nível de tensão da linha de dado com um determinado valor limite;
- ✓ Estando acima deste limite, então o valor da linha é 1, e um reforço de carga é acionado;
- ✓ O circuito é ativado periodicamente ou sempre que uma operação é realizada.

Memória RAM DRAM



- Agora podemos formar uma coluna destas células com um circuito de *refresh* compartilhado;
- A seleção da linha de dados será feito por um *decoder*;

Memória RAM DRAM



- Agora podemos formar um *array* gerando uma *DRAM* de 4×4 ;
- A leitura da informação armazenada também é controlada por uma entrada (*READ*) que copiará a linha selecionada para um *BUFFER DE SAÍDA*;
- Este *BUFFER* foi criado com uma *DRAM* de 1×4 ;

Memória RAM *DRAM*

- A principal aplicação das *DRAMs* é na formação da memória principal dos computadores;
- A diferença entre *DRAMs* e *SRAMs* é o tipo de célula de memória;
- Como já visto, a célula de memória *DRAM* consiste em um transistor e um capacitor, sendo muito mais simples do que a célula *SRAM*;

Memória RAM *DRAM*

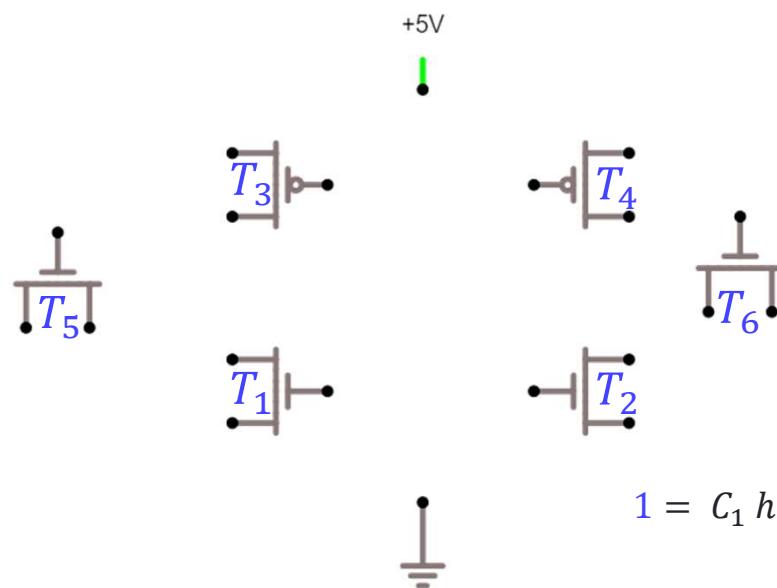
- Isso proporciona densidades muito maiores com *DRAMs*, resultando em capacidades superiores, embora o tempo de acesso seja muito mais lento;
- Adicionalmente, como a carga armazenada em um capacitor será lentamente drenada, a célula *DRAM* requer uma operação de *refresh* frequente para preservar o dado armazenado;
- Este requisito resulta em circuitos mais complexos do que em uma *SRAM*;
- Na sequência, a estrutura das *SRAMs*.

Memória RAM *SRAM*

- As *SRAMs* são caracterizadas por células de memória formadas por *latches* ou uma combinação equivalente de seis transistores;
- Uma vez mantida a alimentação elétrica, uma célula de memória estática pode reter um estado 1 ou 0 indefinidamente;
- Não há perda de carga, logo é desnecessário um circuito de *refresh*;
- Os *latches* serão apresentados no módulo de circuitos sequencias.

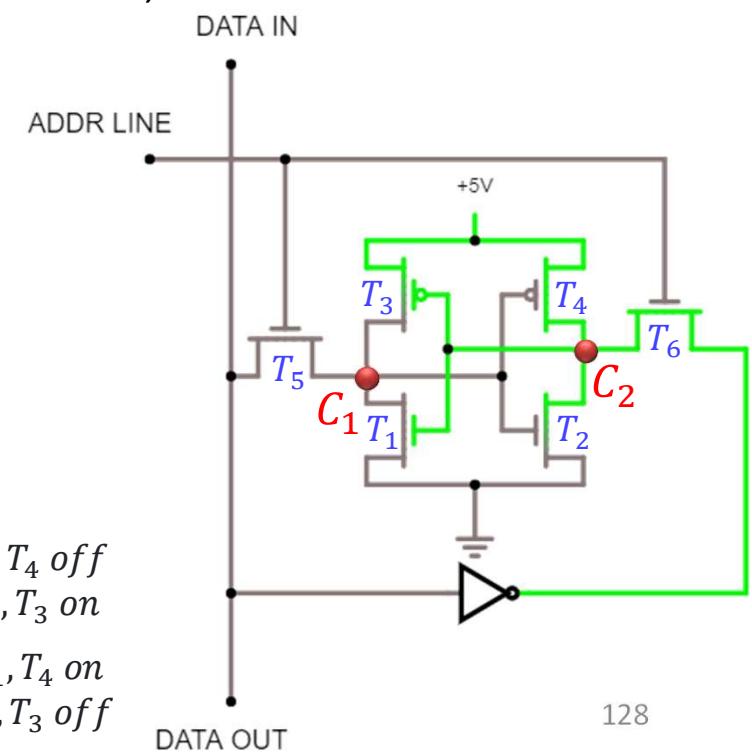
Memória RAM SRAM

- São necessário 6 transistores e uma fonte para cada célula;

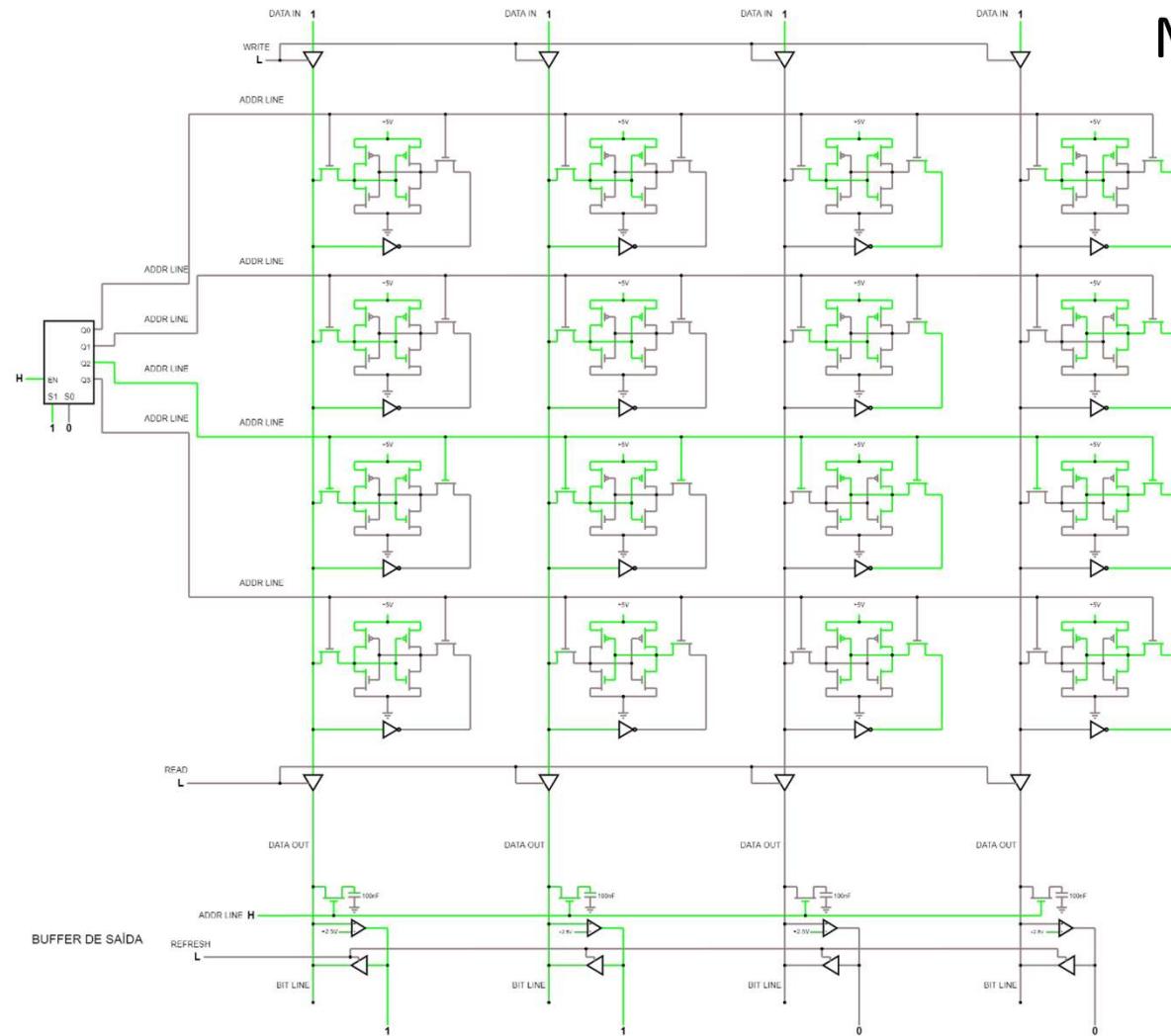


$1 = C_1 \text{ high}, C_2 \text{ low} \Rightarrow T_1, T_4 \text{ off}$
 $T_2, T_3 \text{ on}$

$0 = C_2 \text{ high}, C_1 \text{ low} \Rightarrow T_1, T_4 \text{ on}$
 $T_2, T_3 \text{ off}$



Memória RAM SRAM



Memória RAM *SRAM*

- Uma das principais aplicações das *SRAMs* é nas memórias *cache*;
- A memória *cache* é uma memória relativamente pequena e de alta velocidade que armazena as instruções ou dados mais recentemente referenciados na memória principal (usualmente maior e mais lenta);
- Um *cache* de primeiro nível (cache *L1* ou *cache primária*) é geralmente integrada ao chip do processador e tem capacidade de armazenamento bastante limitada;

Memória RAM *SRAM*

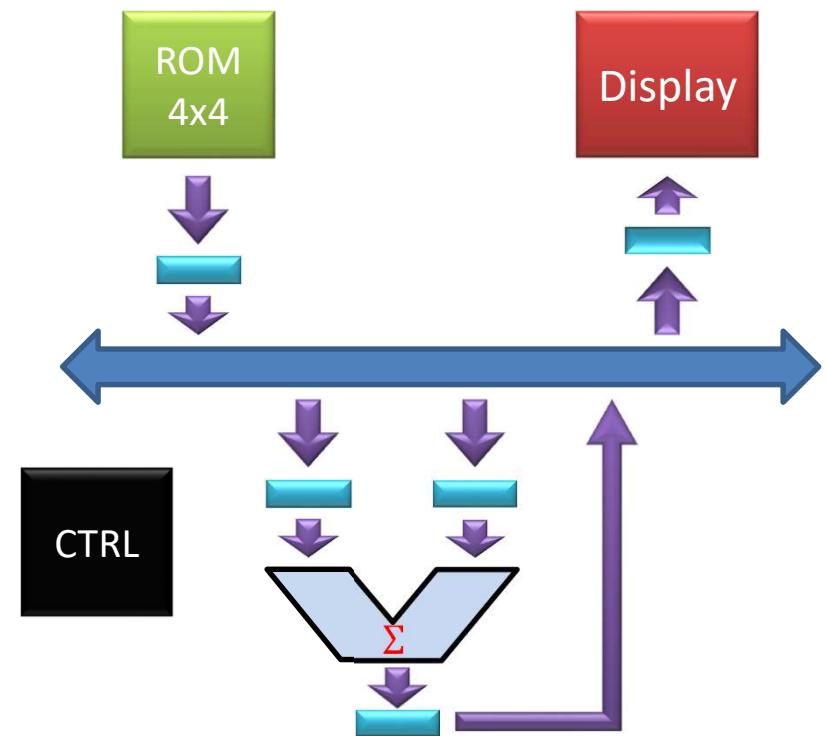
- Uma *cache* de segundo nível (cache *L2* ou *cache secundária*) também pode ser integrada ao processador ou formar um conjunto de memória separado, externo ao processador;
- Geralmente tem uma capacidade de armazenamento maior do que a *cache L1*;
- Diferentes sistemas podem ter *caches* de diversos outros níveis (*L3*, *L4*, etc.), mas *L1* e *L2* são as mais comuns.

Three-State Buffer

Exercício

- 1) Elaborar um circuito de 4 bits para somar dois valores contidos em uma *ROM* e apresentar o resultado em um Display. Seguem os requisitos:

- Haverá apenas um circuito *somador* de 4 bits;
- A soma deverá ser obtida executando-se manualmente passos de transferência de dados de/para os *buffers*;
- Deverá haver um barramento compartilhado entre os dispositivos;
- Deverá haver uma lógica de controle do fluxo da informação e ativação dos dispositivos;
- Os *buffers* podem ser do tipo *DRAM* ou *SRAM*;
- O dado de um endereço da *ROM* deve ir para um dos *buffers* do *somador* e o outro dado para o segundo *buffer*;



Three-State Buffer Exercício

- 1) Elaborar um circuito de 4 bits para somar dois valores contidos em uma *ROM* e apresentar o resultado em um Display. Seguem os requisitos:

- Exemplo de ações manuais:
 - Preparar dado 1 da *ROM*;
 - Transferir dado do buffer da *ROM* para buffer A/B do *somador*;
 - Preparar dado 2 da *ROM*;
 - Transferir dado do buffer da *ROM* para buffer B/A do *somador*;
 - Executar a soma;
 - Transferir valor do buffer R para barramento;
 - Apresentar dado;

