GTA / UFRJ
GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO

Politécnica UFRJ

COPPE UFRJ

Circuitos Lógicos

Aula 6

cruz@gta.ufrj.br http://gta.ufrj.br/~cruz

Poli – Dept. de Eng. Eletrônica e de Computação
Universidade Federal do Rio de Janeiro

Na última aula

- Codificação
- Representando números inteiros com 0's e 1's
  - Conversão binário – decimal
  - Conversão binário – hexadecimal
  - Representação de números negativos
    - Sinal-magnitude
    - Complemento a 2
  - Código de Gray
- Representação de caracteres

Hoje

- Circuito somador
- Multiplexadores e demultiplexadores
- Codificadores e decodificadores
  - BCD para display 7 segmentos

Assuntos administrativos

- Continuação da Prática 2 na 3ª-feira
  - Quem já fez não precisa vir
- Prova 1 dia 16 de maio
  - Revisão na aula anterior
- Trabalho 1
  - Entrega 30 de maio (mais de 1 mês para fazer)
- Prova 2
  - 6 de julho
- Entrega do trabalho prático 2
  - 11 de julho
- Prova final
  - 13 de julho
- Prova de 2ª chamada
  - 20 de julho

Assuntos administrativos

- Lista 2
  - Entrega até hoje
- Lista 3
  - Entrega até dia 04/05
- Lista 4
  - Entrega até dia 11/05
- Lista 5
  - Entrega até dia 18/05

Projeto de Circuitos

Padrões de projeto

- Arquitetura de software e hardware enfrenta problemas similares várias vezes
- Problemas similares provavelmente têm soluções similares
- Engenharia conhece padrões repetidos
  - Cria módulos “padrão”
  - Cria arquiteturas “padrão”

Padrões de projeto

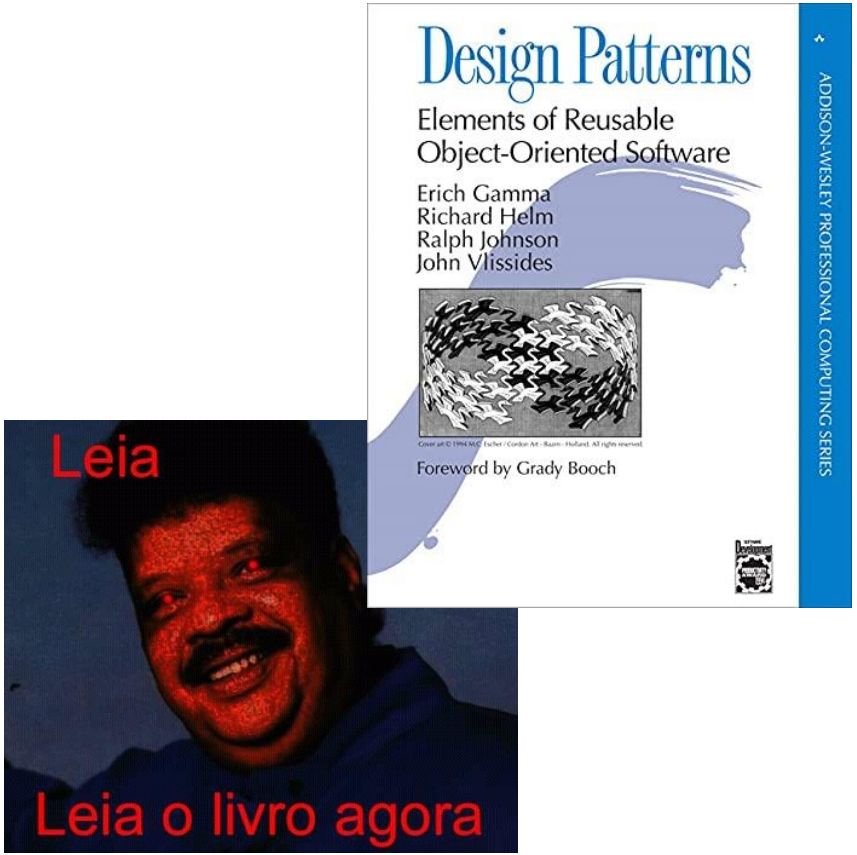
- Arquitetura de software e hardware enfrenta problemas similares várias vezes
- Problemas similares provavelmente têm soluções similares
- Engenharia conhece padrões repetidos
  - Cria módulos “padrão”
  - Cria arquiteturas “padrão”

Módulo: um pedaço de código ou hardware capaz de executar uma função
Arquitetura: o jeito como esses módulos se relacionam

Leia o livro

Gamma et al., Design Patterns - Elements of Reusable Object-Oriented Software

Ou pelo menos estude padrões de design

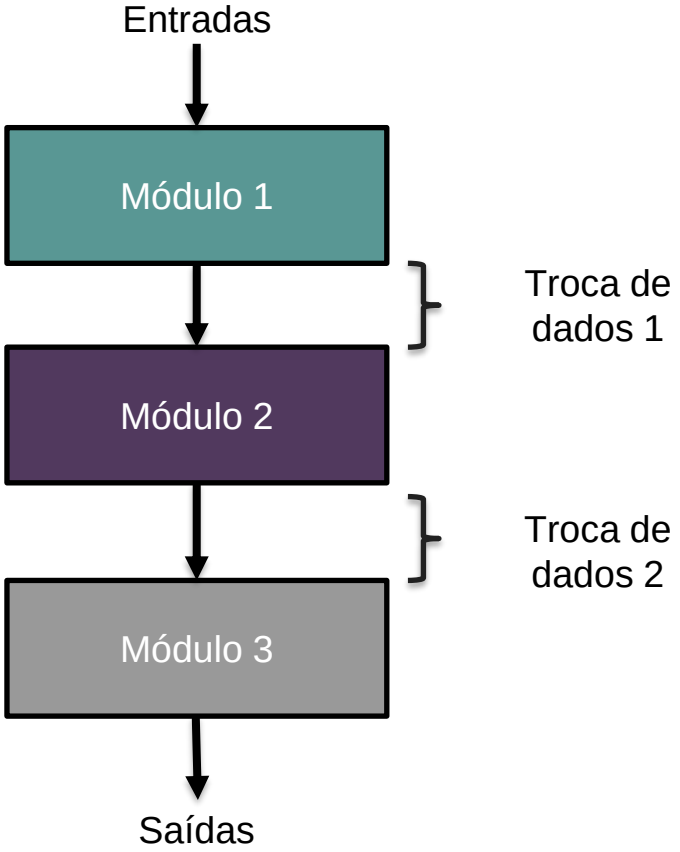


Modularização - dividir para conquistar

- Dividir para conquistar
  - Problema dividido em etapas
- Resolver cada etapa

- Vantagens
  - Cada etapa é testável
    - Erros são detectáveis
  - Módulos são reutilizáveis

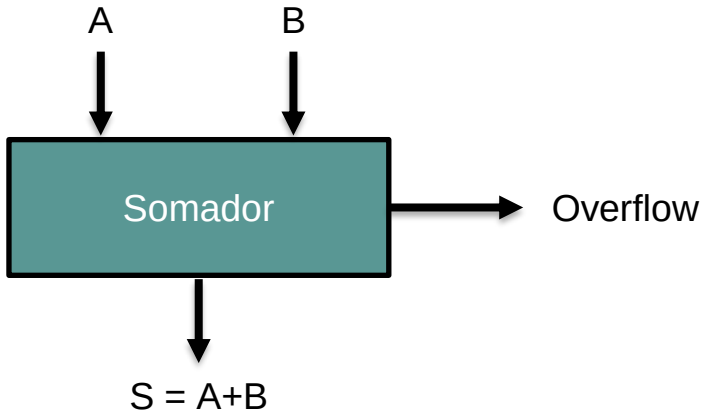
- Desvantagens
  - Dividir é um novo problema
  - Não garante solução ótima



Circuito somador

Somador

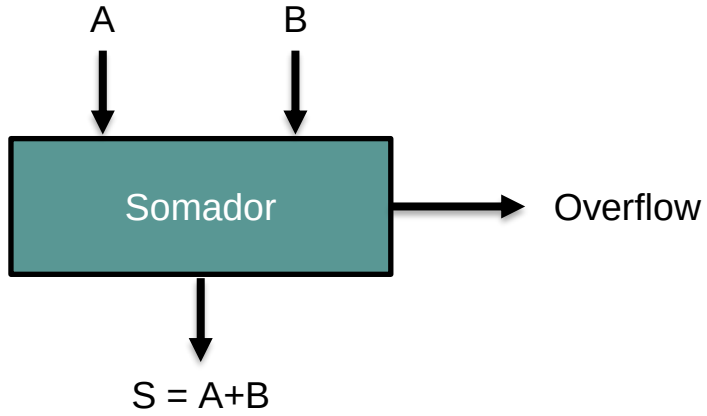
- Recebe duas variáveis A e B
  - n bits de entrada cada
  - Representam números em binário
- Entrega a soma de A com B
  - n bits
  - Bit **overflow**, informando se a soma “estourou” a capacidade de representação



Atenção: Soma não é OR!

As variáveis e entrada

- Recebe duas variáveis A e B
  - n bits de entrada cada
  - Representam números em binário
- Entrega a soma de A com B
  - n bits
  - Bit **overflow**, informando se a soma “estourou” a capacidade de representação



Atenção: Soma não é OR!

As variáveis de entrada e saída

- Entradas
  - Letras do início do alfabeto
    - A e B no nosso caso
  - Contém n bits
    - $A_0, A_1, A_2, \ldots A_n$
    - $B_0, B_1, B_2, \ldots B_n$
- Saídas
  - Soma é denotada por letra do fim do alfabeto
    - S no nosso caso
    - $S_0, S_1, S_2, \ldots S_n$
  - Estouro é denotado por C_{out}
    - *Overflow*

No nosso caso

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0
A	A_3	A_2	A_1	A_0
B	B_3	B_2	B_1	B_0
Saída	S_3	S_2	S_1	S_0
Estouro				C_{out}

Tabela verdade para 4 bits

- 8 variáveis de entrada
  - 2^8 configurações de entrada
- 5 Variáveis de saída
  - 5 tabelas com 2^8 configurações

B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0	S_3	S_2	S_1	S_0	C_{out}
0	0	0	0	1	1	0	0	1	1	0	0	0
0	0	0	0	1	1	0	1	1	1	0	1	0
0	0	0	0	1	1	1	0	1	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	1	0
0	0	0	0	1	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	1	1	0	0	1	0
0	0	0	0	1	0	1	0	1	0	1	0	0
0	0	0	0	1	0	1	1	1	0	1	1	0
0	0	0	0	0	1	0	0	0	1	0	0	0

Soma bit a bit – o Bit 0

- Se ambos forem 0, saída é 0
- Se um for 0 e o outro 1, saída é 1
- Se ambos forem 1, saída é 0
  - Soma “estoura”
    - Próximo bit deve receber +1

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S				1	13
C_{out}				0	0

Soma bit a bit – o Bit 0

- Se ambos forem 0, saída é 0
- Se um for 0 e o outro 1, saída é 1
- Se ambos forem 1, saída é 0
  - Soma “estoura”
    - Próximo bit deve receber +1

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S				1	13
C_{out}				0	0

- Bit 0
  - Soma é 1
  - Não tem estouro

Diferença entre *carry* e *overflow*

- *Carry*
  - Representa o “vai um” de uma soma
  - Bit a bit
- *Overflow*
  - Representa um transbordamento da capacidade de representação
  - Tem a ver com um conjunto de bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0
A	A_3	A_2	A_1	A_0
B	B_3	B_2	B_1	B_0
Saída	S_3	S_2	S_1	S_0
Estouro				C_{out}

Soma bit a bit – o Bit 1

- O estouro do Bit 0 deve ser considerado
  - Soma passa a ser de 3 bits
    - A_1
    - B_1
    - C_0
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S				1	13
C_{out}				0	0

Soma bit a bit – o Bit 1

- O estouro do Bit 0 deve ser considerado
  - Soma passa a ser de 3 bits
    - A_1
    - B_1
    - C_0
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S			0	1	13
C_{out}			1	0	0

- Bit 1
  - Soma é 0
  - Tem estouro

Soma bit a bit – o Bit 2

- O estouro do Bit 0 deve ser considerado
  - Soma é de 3 bits
    - A_2
    - B_2
    - C_1
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S			0	1	13
C_{out}			1	0	0

Soma bit a bit – o Bit 2

- O estouro do Bit 0 deve ser considerado
  - Soma é de 3 bits
    - A_2
    - B_2
    - C_1
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S		1	0	1	13
C_{out}		1	1	0	0

- Bit 2
  - Soma é 1
  - Tem estouro

Soma bit a bit – o Bit 3

- O estouro do Bit 0 deve ser considerado
  - Soma é de 3 bits
    - A_3
    - B_3
    - C_2
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S		1	0	1	13
C_{out}		1	1	0	0

Soma bit a bit – o Bit 3

- O estouro do Bit 0 deve ser considerado
  - Soma é de 3 bits
    - A_3
    - B_3
    - C_2
  - Estouro considera mesmos 3 bits

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S	1	1	0	1	13
C_{out}	0	1	1	0	0

- Bit 3
  - Soma é 1
  - Não tem estouro

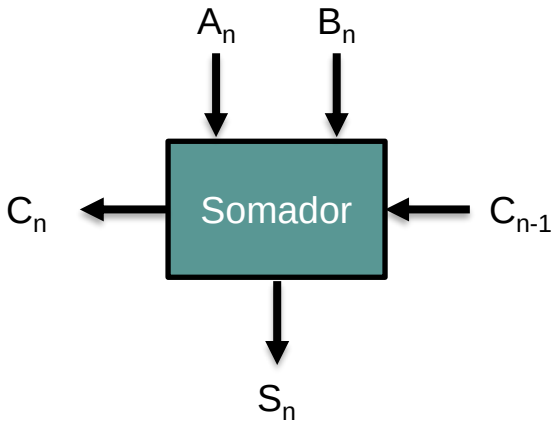
Resultado da soma

- Bits de S são o resultado da soma
  - Devem ser 13
    - Quando convertidos

Variável	Bit 3 2^3	Bit 2 2^2	Bit 1 2^1	Bit 0 2^0	Dec.
A	0	1	1	1	7
+ B	0	1	1	0	6
S	1	1	0	1	13
C_{out}	0	1	1	0	0

Modularização da soma

- S_n
  - A_n
  - B_n
  - C_{n-1}
- C_n
  - A_n
  - B_n
  - C_{n-1}



Modularização da soma

- S_n
  - A_n
  - B_n
  - C_{n-1}
- C_n
  - A_n
  - B_n
  - C_{n-1}

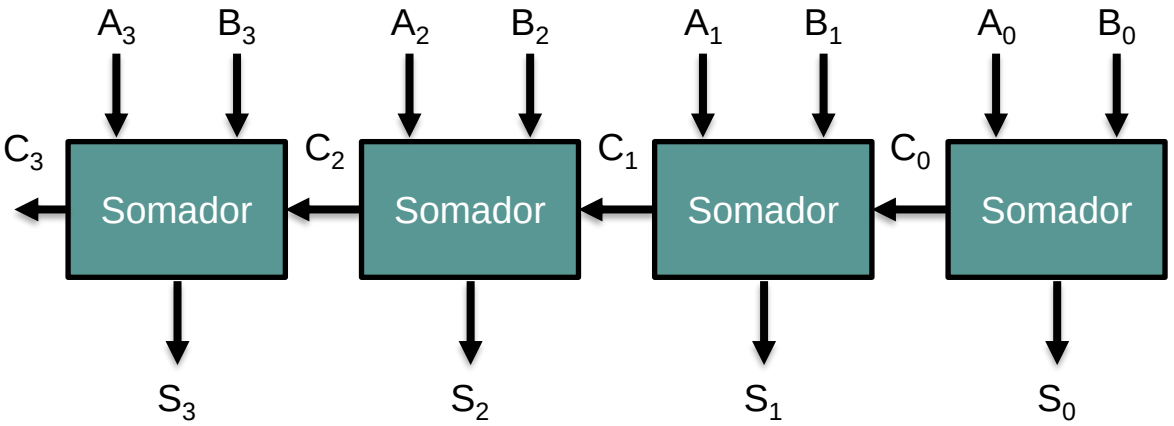


Tabela verdade módulo somador

- S_n deve ser 1 quando 1 ou 3 variáveis de entrada forem 1
- C_n deve ser 1 quando 2 ou mais variáveis de entrada forem 1

C_{n-1}	B_n	A_n	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Expressão para S_n

$$
\begin{aligned}
&S_n \\
&= (\overline{C_{n-1}} \cdot \overline{B_n} \cdot A_n) \\
&+ (\overline{C_{n-1}} \cdot B_n \cdot \overline{A_n}) \\
&+ (C_{n-1} \cdot \overline{B_n} \cdot \overline{A_n}) \\
&+ (C_{n-1} \cdot B_n \cdot A_n)
\end{aligned}
$$

C_{n-1}	B_n	A_n	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Simplificando S_n

$$S_n = (\overline{C_{n-1}} \cdot \overline{B_n} \cdot A_n) + (\overline{C_{n-1}} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot \overline{B_n} \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$S_n = (\overline{C_{n-1}} \cdot ((\overline{B_n} \cdot A_n) + (B_n \cdot \overline{A_n}))) + (C_{n-1} \cdot ((\overline{B_n} \cdot \overline{A_n}) + (B_n \cdot A_n)))$$

Sabemos que:

$$(\overline{B_n} \cdot A_n) + (B_n \cdot \overline{A_n}) = A_n \oplus B_n$$

E que:

$$(\overline{B_n} \cdot \overline{A_n}) + (B_n \cdot A_n) = \overline{A_n \oplus B_n}$$

Assim:

$$S_n = (\overline{C_{n-1}} \cdot (A_n \oplus B_n)) + \left(C_{n-1} \cdot (\overline{A_n \oplus B_n})\right)$$

Então, chamando $A_n \oplus B_n$ de D, temos

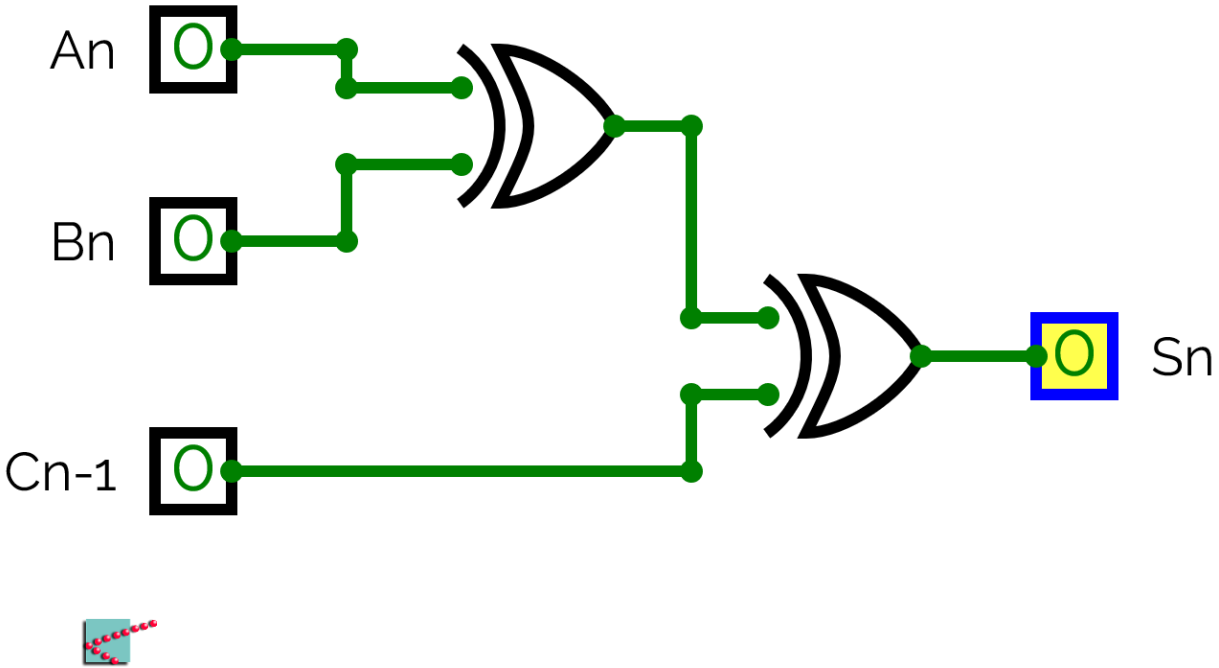
$$S_n = (\overline{C_{n-1}} \cdot D) + (C_{n-1} \cdot \overline{D}) = C_{n-1} \oplus D$$

Por fim,temos:

$$S_n = C_{n-1} \oplus (A_n \oplus B_n)$$

Circuito para S_n

$$S_n = C_{n-1} \oplus (A_n \oplus B_n)$$



Expressão para C_n

$$
\begin{aligned}
& C_n \\
& = (\overline{C_{n-1}} \cdot B_n \cdot A_n) \\
& + (C_{n-1} \cdot \overline{B_n} \cdot A_n) \\
& + (C_{n-1} \cdot B_n \cdot \overline{A_n}) \\
& + (C_{n-1} \cdot B_n \cdot A_n)
\end{aligned}
$$

C_{n-1}	B_n	A_n	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Simplificando C_n

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot \overline{B_n} \cdot A_n) + (C_{n-1} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

Simplificando C_n

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot \overline{B_n} \cdot A_n) + (C_{n-1} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n)) + (C_{n-1} \cdot B_n \cdot A_n)$$

Simplificando C_n

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot \overline{B_n} \cdot A_n) + (C_{n-1} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n)) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n))$$

Simplificando C_n

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot \overline{B_n} \cdot A_n) + (C_{n-1} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n)) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n))$$

Chamando de $B_n \cdot A_n$ de D:

$$C_n = (\overline{C_{n-1}} \cdot D) + (C_{n-1} \cdot D) + (C_{n-1} \cdot (A_n \oplus B_n))$$

Simplificando C_n

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot \overline{B_n} \cdot A_n) + (C_{n-1} \cdot B_n \cdot \overline{A_n}) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n)) + (C_{n-1} \cdot B_n \cdot A_n)$$

$$C_n = (\overline{C_{n-1}} \cdot B_n \cdot A_n) + (C_{n-1} \cdot B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n))$$

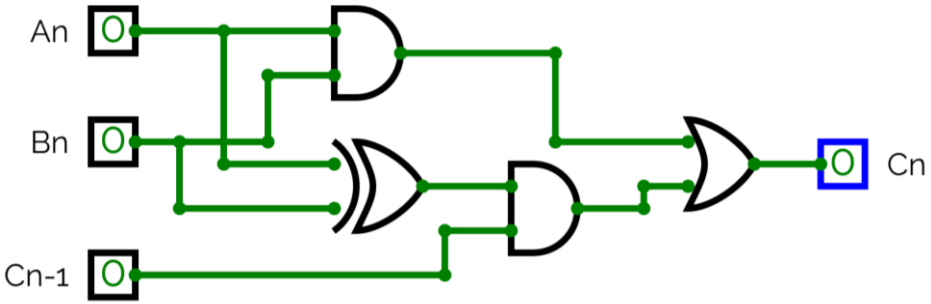
Chamando de $B_n \cdot A_n$ de D:

$$C_n = (\overline{C_{n-1}} \cdot D) + (C_{n-1} \cdot D) + (C_{n-1} \cdot (A_n \oplus B_n))$$

$$C_n = (B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n))$$

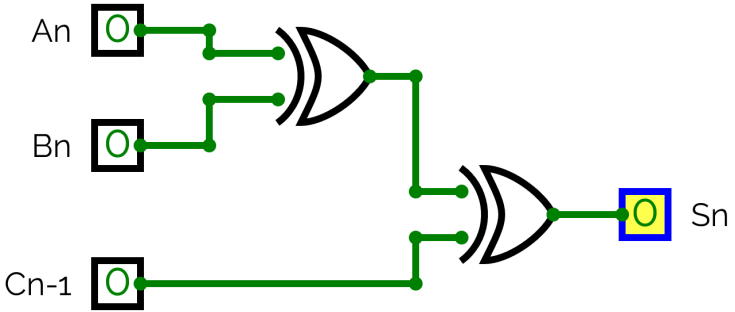
Circuito de C_n

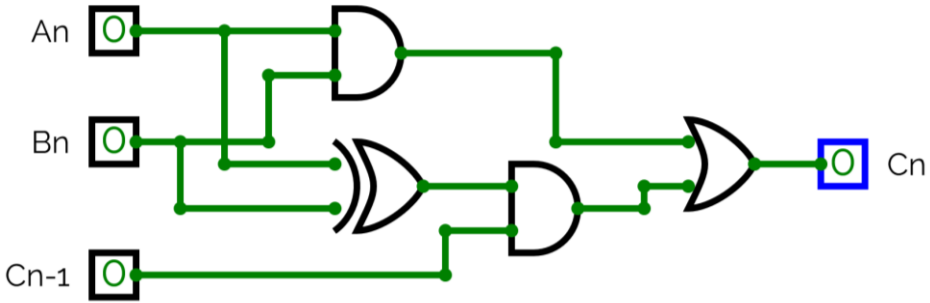
$$C_n = (B_n \cdot A_n) + (C_{n-1} \cdot (A_n \oplus B_n))$$



Circuito somador

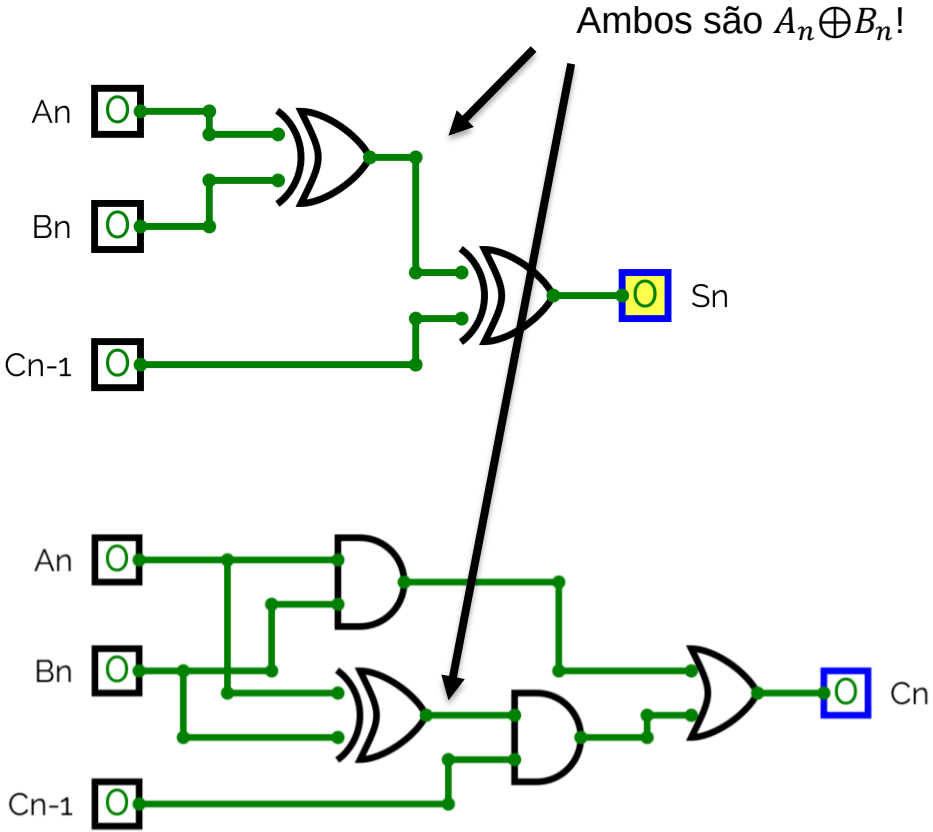
- Podemos aproveitar operações comuns às duas saídas
  - Deve ser feito com cuidado!





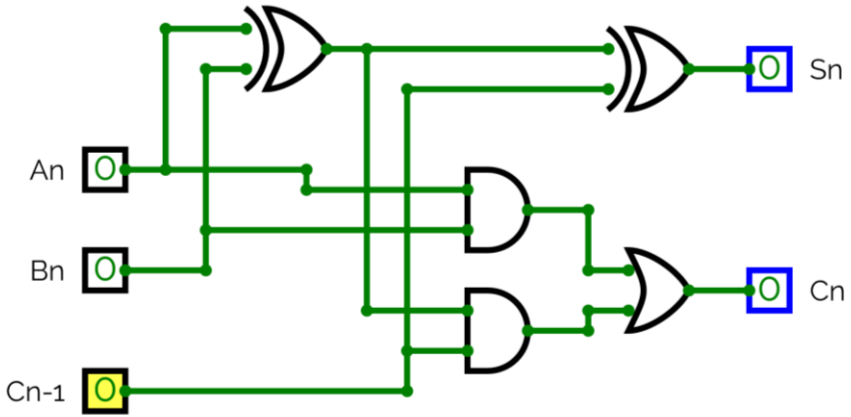
Circuito somador

- Podemos aproveitar operações comuns às duas saídas
  - Deve ser feito com cuidado!



Circuito somador

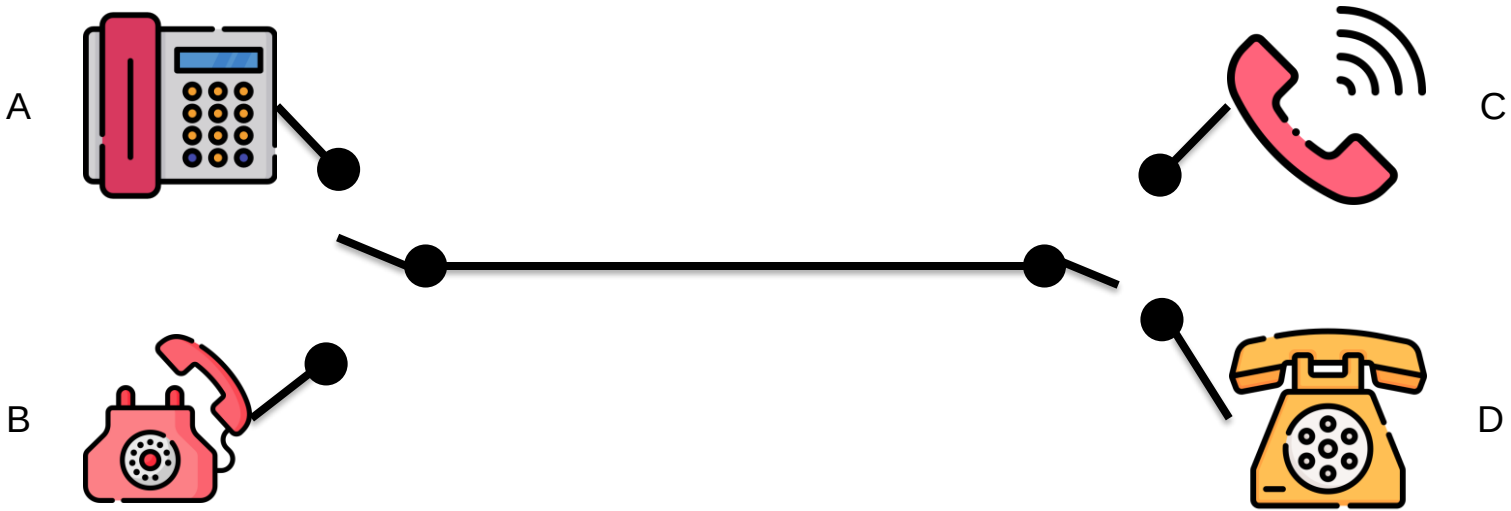
- Podemos compartilhar $A_n \oplus B_n$ entre as saídas



Multiplexadores e demultiplexadores

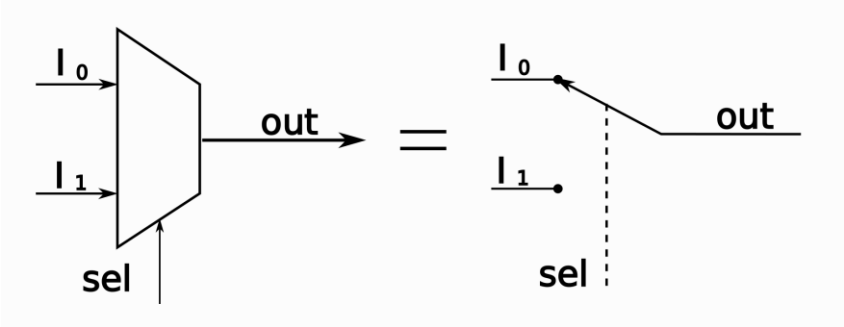
Chaveamento é importante

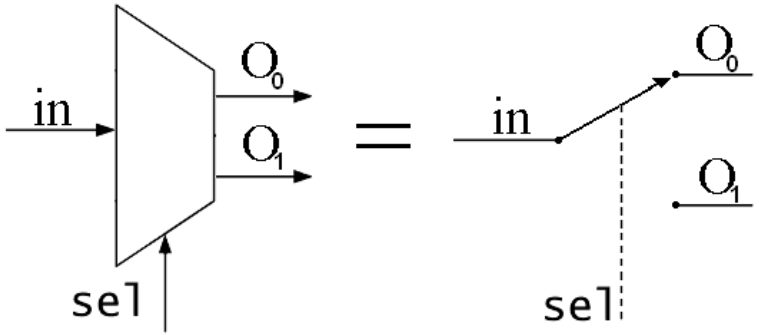
- Pessoas A e B estão de um lado do país e querem falar com C e D, que estão do outro
- Apenas um canal de comunicação
- Múltiplas entradas devem ser chaveadas com múltiplas saídas



Chaveamento é importante

- Pessoas A e B estão de um lado do país e querem falar com C e D, que estão do outro
- Apenas um canal de comunicação
- Múltiplas entradas devem ser chaveadas com múltiplas saídas

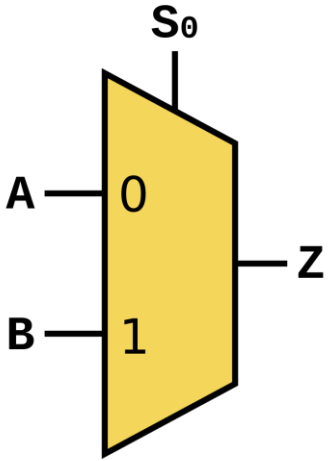




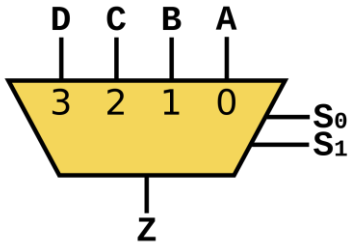
Imagens retiradas
da Wikipedia

Multiplexador (MUX)

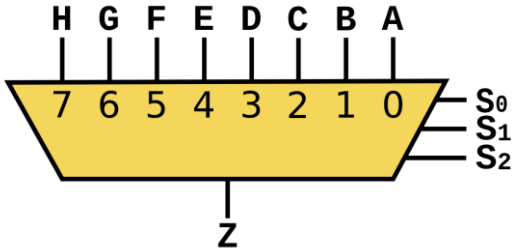
- Dois conjuntos de entradas
  - n seletores
  - 2^n linhas de dados
- 1 única saída
- Seletor “chaveia” entre diferentes linhas de dados



Mux 2:1



Mux 4:1



Mux 8:1

Imagens retiradas da Wikipedia

Como funciona?

- Dois conjuntos de entradas
  - n seletores
  - 2^n linhas de dados
- Seletor “chaveia” entre diferentes linhas de dados

S_1	S_0	Z
0	0	A
0	1	B
1	0	C
1	1	D

Mux 4:1

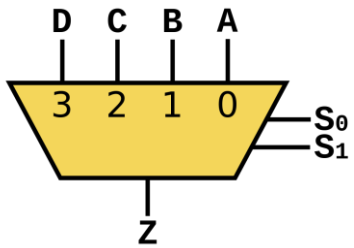


Imagem retirada da Wikipedia

Se a gente lembrar

- $x \cdot 1 = x$
  - AND com 1 “deixa passar”
- $x + 0 = x$
  - OR com 0 “deixa passar”

S_1	S_0	Z
0	0	A
0	1	B
1	0	C
1	1	D

Mux 4:1

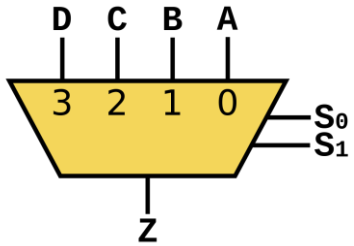


Imagem retirada da Wikipedia

Opção de implementação

- $x \cdot 1 = x$
  - AND com 1 “deixa passar”
- $x + 0 = x$
  - OR com 0 “deixa passar”

$$
\begin{aligned}
Z \\
&= (\overline{S_1} \cdot \overline{S_0} \cdot A) \\
&+ (\overline{S_1} \cdot S_0 \cdot B) \\
&+ (S_1 \cdot \overline{S_0} \cdot C) \\
&+ (S_1 \cdot S_0 \cdot D)
\end{aligned}
$$

Mux 4:1

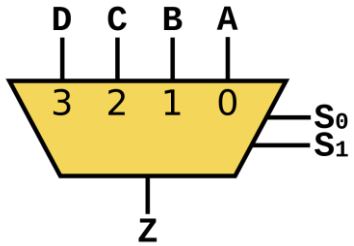
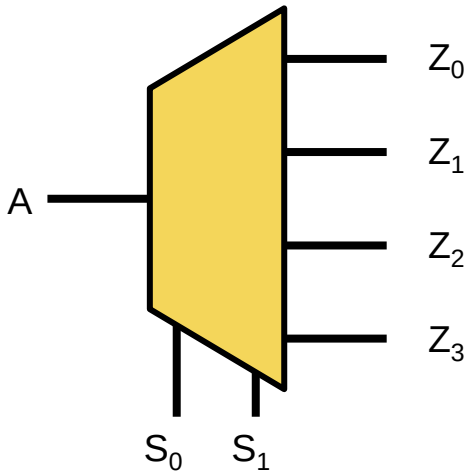


Imagem retirada da Wikipedia

S_1	S_0	Z
0	0	A
0	1	B
1	0	C
1	1	D

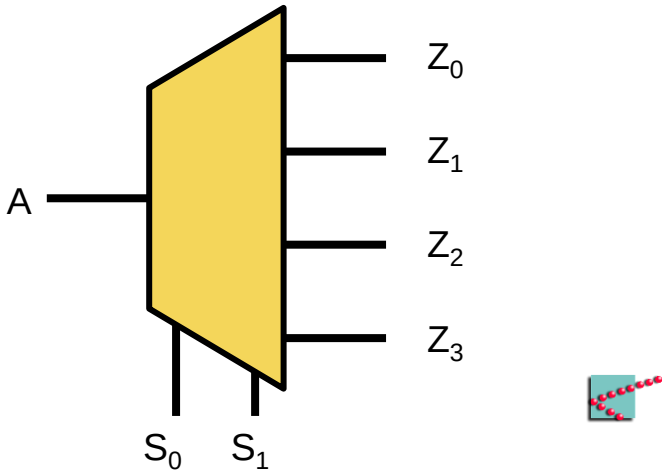
Demultiplexador (DEMUX)

- Dois conjuntos de entradas
  - n seletores
  - 1 linhas de dados
- 2^n saídas
- Seletor “chaveia” a linha de dados entre diferentes saídas



Demultiplexador (DEMUX)

- Dois conjuntos de entradas
  - n seletores
  - 1 linhas de dados
- 2^n saídas
- Seletor “chaveia” a linha de dados entre diferentes saídas

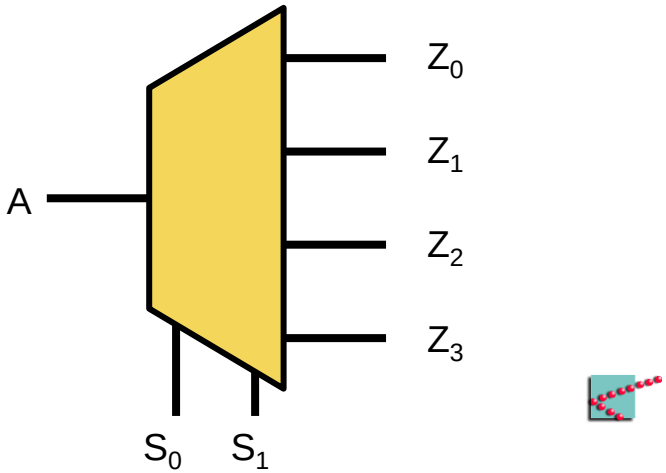


S_1	S_0	A	Z_3	Z_2	Z_1	Z_0
0	0	0				0
0	0	1				1
0	1	0			0	
0	1	1			1	
1	0	0		0		
1	0	1		1		
1	1	0	0			
1	1	1	1			

Se a gente lembrar

- $x \cdot 1 = x$
  - AND com 1 “deixa passar”
- $x + 0 = x$
  - OR com 0 “deixa passar”

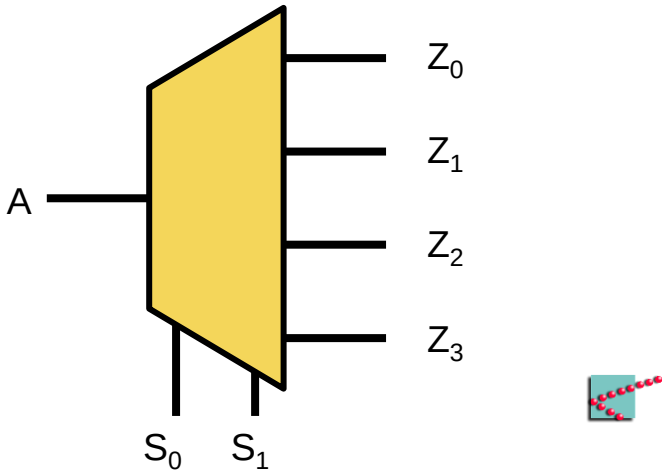
$$Z_0 = (\overline{S_1} \cdot \overline{S_0} \cdot A)$$
$$Z_1 = (\overline{S_1} \cdot S_0 \cdot A)$$
$$Z_2 = (S_1 \cdot \overline{S_0} \cdot A)$$
$$Z_3 = (S_1 \cdot S_0 \cdot A)$$



Problema!

- Saídas recebem 0 quando deveriam estar inativas!!!
- Como resolver?

Saberemos em aulas futuras

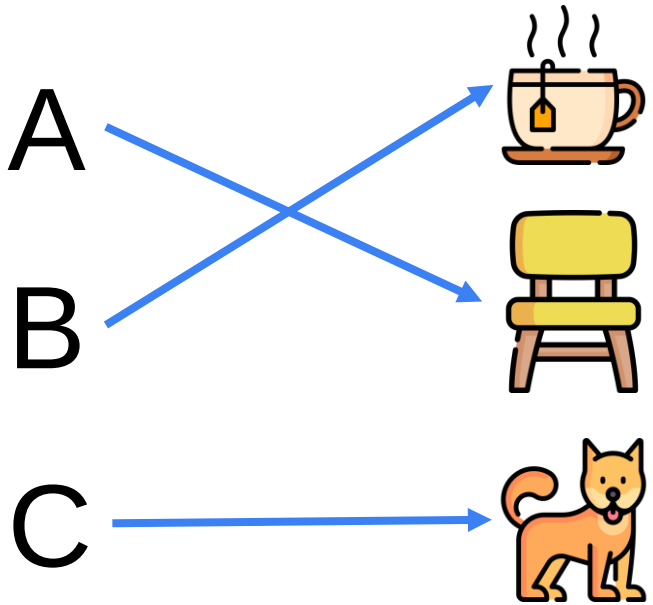


S_1	S_0	A	Z_3	Z_2	Z_1	Z_0
0	0	0				0
0	0	1				1
0	1	0			0	
0	1	1			1	
1	0	0		0		
1	0	1		1		
1	1	0	0			
1	1	1	1			

Codificação

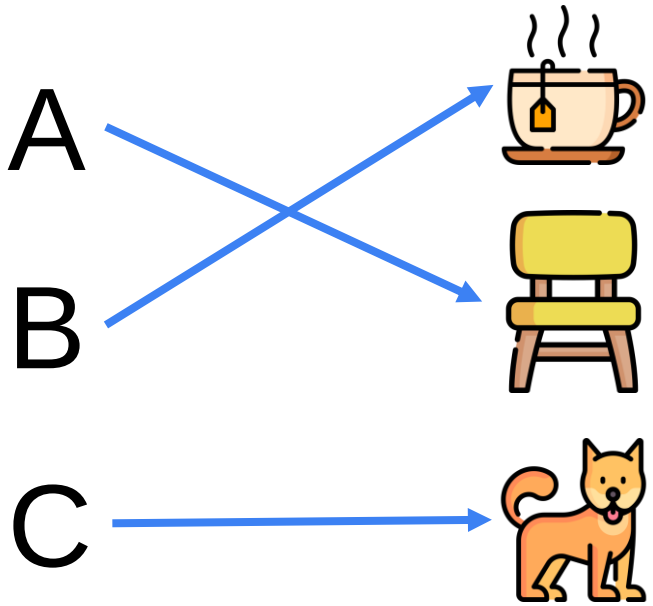
Codificar

- Sejam F e D alfabetos fonte e destino
- Uma codificação é uma associação de cada símbolo de F a cada símbolo de D
  - Função bijetora



Codificar

- Alfabetos podem ser quaisquer coisas imagináveis
  - Letras
  - Algarismos
  - Números
  - Palavras
  - Sons faláveis
- Codificação também
  - Escrita
    - Letras – sons faláveis
  - Numeração
    - Quantidades – números
  - Vocabulário
    - Palavras – ideias



Representação em binário

- Sequência de bits representa um símbolo de um alfabeto
  - Alfabeto pode ser:
    - Números inteiros
    - Algarismos
    - Letras
    - Cores
    - Estados de uma variável
    - ....

Codificadores e decodificadores

Definição geral

- Codificador transforma dados ‘naturais’ em sua versão binária
- Decodificador transforma dados binários em sua versão ‘natural’

Algumas vezes o natural também é binário!

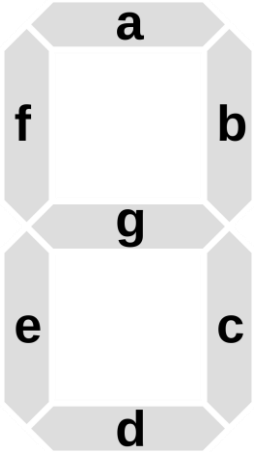
Circuitos codificadores e decodificadores

- Capazes de *traduzir* de um código binário para outro

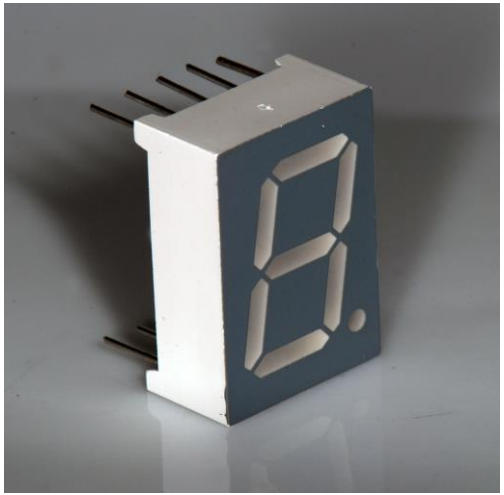
- Exemplos
  - Sinal-magnitude para complemento de 2
  - Complemento de 2 para sinal-magnitude
  - ASCII para binário
  - Binário para ASCII
  - BCD para display 7 segmentos
  - Hexadecimal para display 7 segmentos

Display 7 segmentos

- Leds organizados em 8
- Leds nomeados a, b, c, d, e, f, g
- GND comum
- Terminal para cada um dos leds

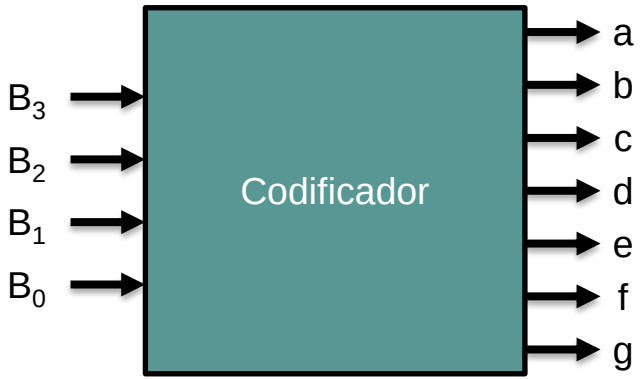


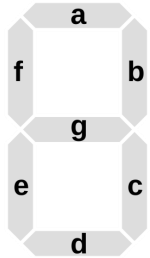
Imagens retiradas da Wikipedia



Codificador BCD – 7 segmentos

- Recebe 4 variáveis
  - Representam um número de 4 bits
- Codifica o número para exibição em um display de 7 segmentos





Não tem jeito

- Tabela verdade pra cada uma das variáveis de saída
  - 2^4 configurações cada
- Algumas vão poder reaproveitar entradas de outras

Considerações finais

- Modularização é uma importante ferramenta
- Soma é simplificada se usarmos uma modularização
- Comunicação depende de portas lógicas
- Codificadores convertem diferentes códigos
  - Códigos diferentes possuem diferentes utilidades

Créditos

Os ícones desta apresentação foram feitos por Freepic e retirados de www.flaticon.com.

GTA / UFRJ

GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO

www.gta.ufrj.br