PCS 3115 (PCS2215)

Sistemas Digitais I

Módulo 11 – Circuitos Combinatórios
Blocos Básicos

Prof. Dr. Marcos A. Simplicio Jr.

versão: 3.0 (Jan/2016)

Blocos básicos

- Multiplicadores
- ULA
- Gerador/Detector de Paridade
- Exercícios

Multiplicador binário

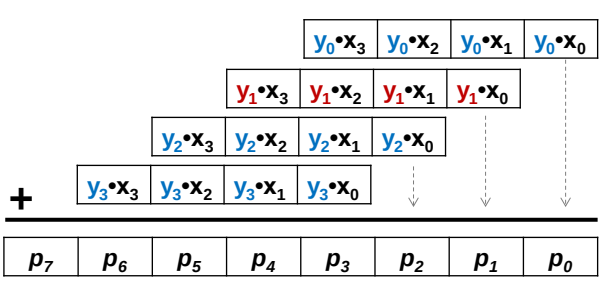
Algoritmo “shift-and-add”: apenas dois registradores para dados parciais

```
          1011  Multiplicando
        × 1101  Multiplicador
        --------
          0000  Produto parcial inicial
x 1     +  1011  Multiplicando deslocado p/ esquerda de 0 posições
        --------
         01011  Produto parcial
x 0     + 0000  Multiplicando deslocado p/ esquerda de 1 posição
        -------
        001011  Produto parcial
x 1     +  1011  Multiplicando deslocado p/ esquerda de 2 posições
        ---------
        0110111  Produto parcial
x 1     +  1011  Multiplicando deslocado p/ esquerda de 3 posições
        ----------
        10001111  Produto
```


Multiplicador binário

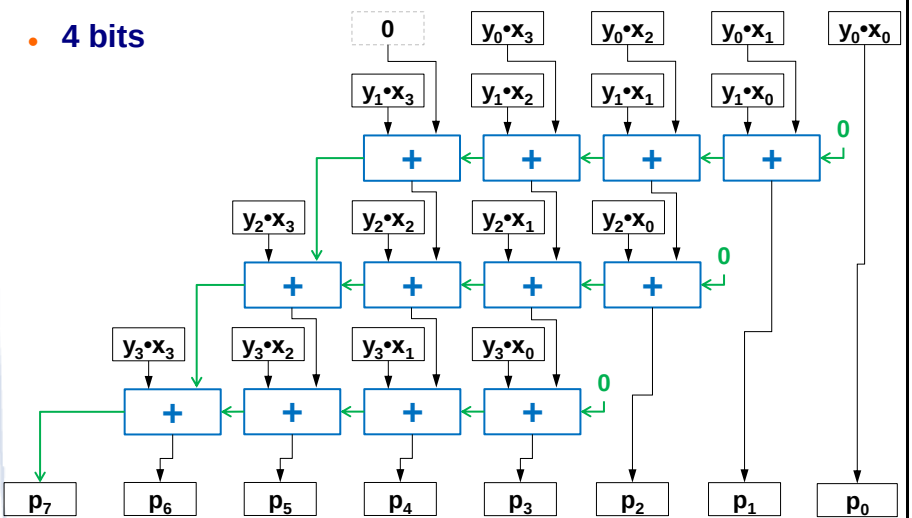
- Processo **não precisa ser sequencial**, usando registradores: pode-se construir um circuito combinatório que implementa cada fase do “laço de shift-and-add”

```
   1011   x
   1101   y
   -------
   1011
  0000
 1011
1011
-------------
10001111   p
```



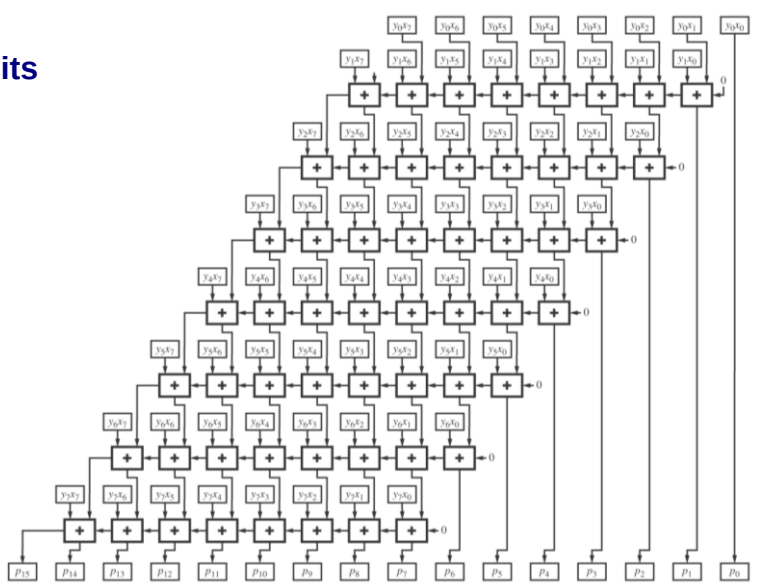
Multiplicador binário

- 4 bits



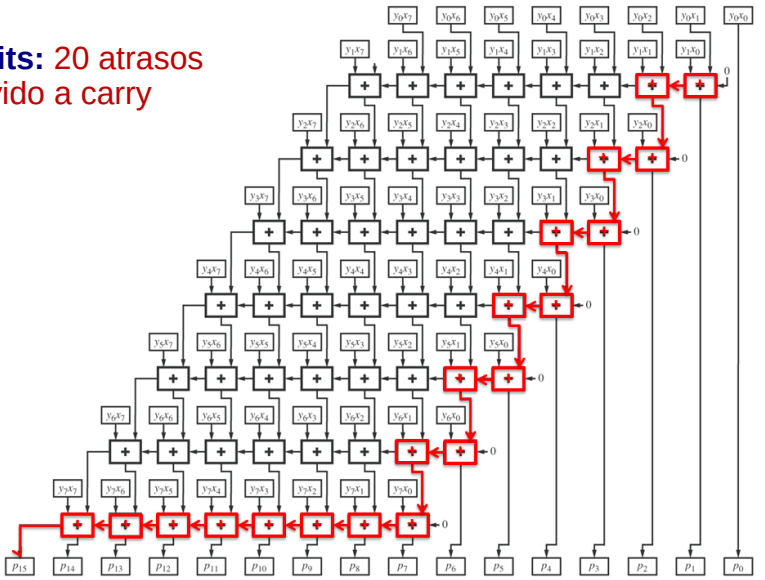
Multiplicador binário

- 8 bits



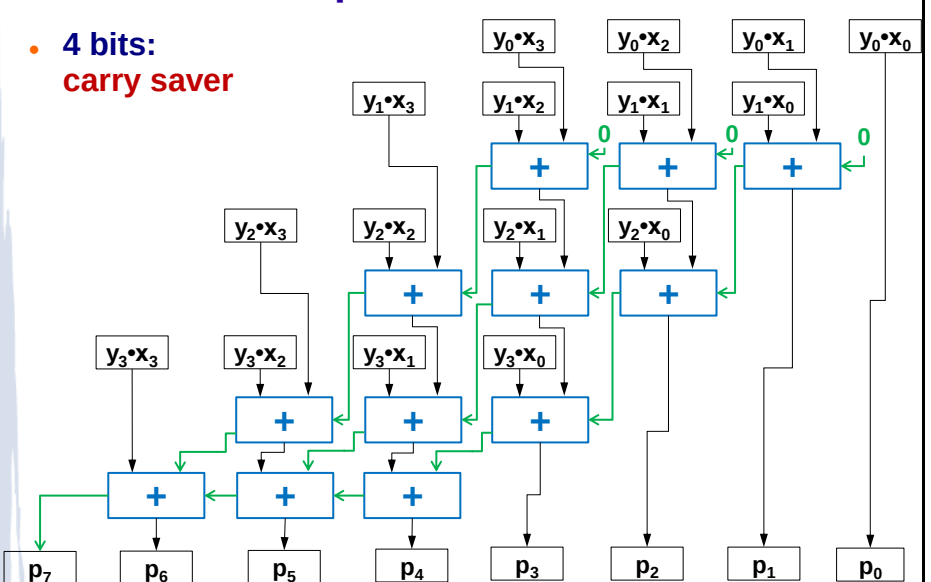
Multiplicador binário

- **8 bits:** 20 atrasos devido a carry



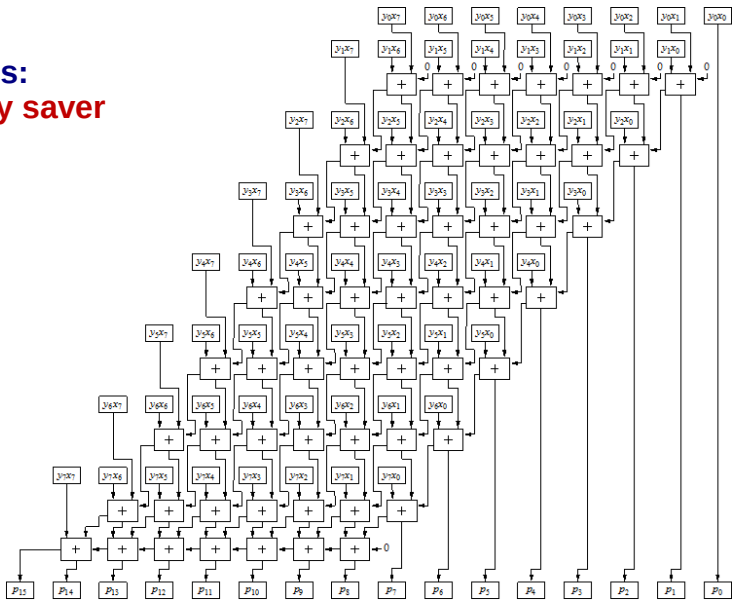
Multiplicador binário

- **4 bits:**
  carry saver



Multiplicador binário

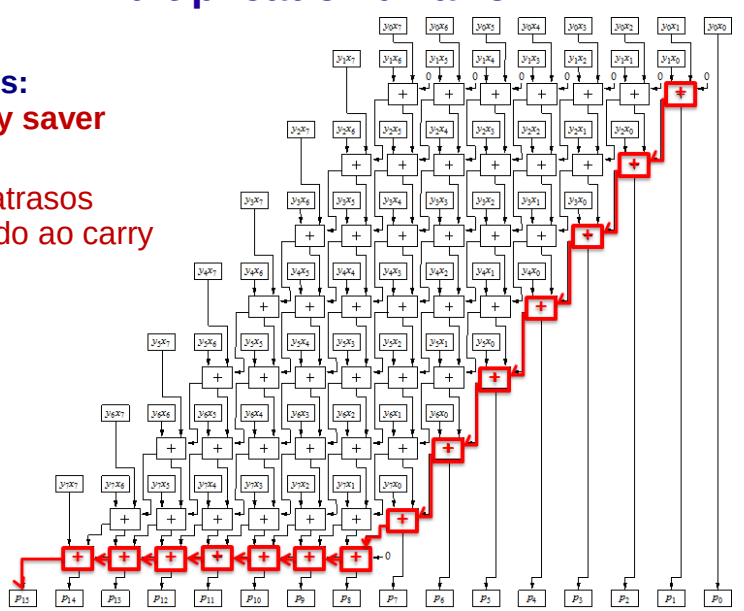
- **8 bits:**
  carry saver



Multiplicador binário

- **8 bits:**
  carry saver

➔ 14 atrasos devido ao carry

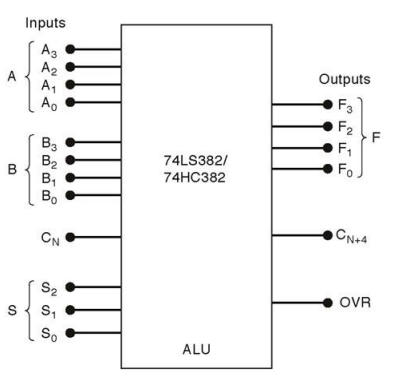


Unidade Lógica Aritmética (ULA)

- Um **Bloco Lógico Funcional** que dispõe de um **repertório básico** e **limitado** de
  - **Operações Lógicas**: and, or, inversão, etc.
  - e/ou **Operações Aritméticas**: soma, subtração, etc.
- Há vários **CIs** que, embora **não tenham a capacidade** de **ULAs de computadores** (pessoais ou de grande porte), fornecem algumas **operações** sobre **dados binários** de entrada.

Unidade Lógica Aritmética (ULA)

ALU 74LS382/74HC382



A = 4-bit input number
B = 4-bit input number
C_N = carry into LSB position
S = 3-bit operation select inputs
F = 4-bit output number
C_{N+4} = carry out of MSB position
OVR = overflow indicator

(a)

Function Table

S_2	S_1	S_0	Operation	Comments
0	0	0	CLEAR	$F_3F_2F_1F_0 = 0000$
0	0	1	B minus A	Needs $C_N = 1$
0	1	0	A minus B	Needs $C_N = 1$
0	1	1	A plus B	Needs $C_N = 0$
1	0	0	A ⊕ B	Exclusive-OR
1	0	1	A + B	OR
1	1	0	AB	AND
1	1	1	PRESET	$F_3F_2F_1F_0 = 1111$

Notes: S inputs select operation.
OVR = 1 for signed-number overflow.

(b)

Unidade Lógica Aritmética (ULA)

4 bits de operações:

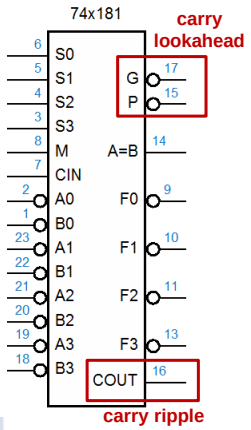
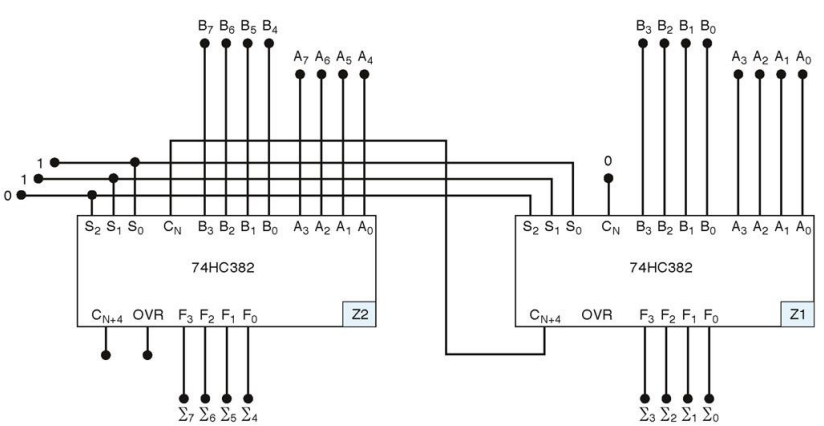


Table 5-52 Functions performed by the 74x181 4-bit ALU.

Inputs				Function	
S3	S2	S1	S0	*M = 0 (arithmetic)*	*M = 1 (logic)*
0	0	0	0	F = A minus 1 plus CIN	F = A′
0	0	0	1	F = A · B minus 1 plus CIN	F = A′ + B′
0	0	1	0	F = A · B′ minus 1 plus CIN	F = A′ + B
0	0	1	1	F = 1111 plus CIN	F = 1111
0	1	0	0	F = A plus (A + B′) plus CIN	F = A′ · B′
0	1	0	1	F = A · B plus (A + B′) plus CIN	F = B′
0	1	1	0	F = A minus B minus 1 plus CIN	F = A ⊕ B′
0	1	1	1	F = A + B′ plus CIN	F = A + B′
1	0	0	0	F = A plus (A + B) plus CIN	F = A′ · B
1	0	0	1	F = A plus B plus CIN	F = A ⊕ B
1	0	1	0	F = A · B′ plus (A + B) plus CIN	F = B
1	0	1	1	F = A + B plus CIN	F = A + B
1	1	0	0	F = A plus A plus CIN	F = 0000
1	1	0	1	F = A · B plus A plus CIN	F = A · B′
1	1	1	0	F = A · B′ plus A plus CIN	F = A · B
1	1	1	1	F = A plus CIN	F = A

Unidade Lógica Aritmética (ULA)



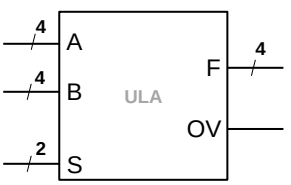
Notes: Z1 adds lower-order bits.
Z2 adds higher-order bits.
Σ_7–Σ_0 = 8-bit sum.
OVR of Z2 is 8-bit overflow indicator.

Expansão da ULA: dois chips 74HC382, conectados para operar como um somador de palavras de 8 bits.

Exercício -- ULA

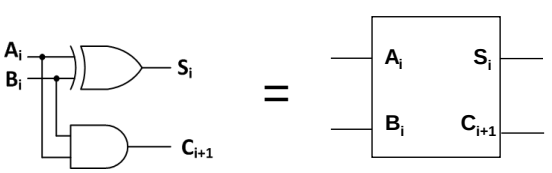
- Projetar a estrutura interna de uma ULA como a mostrada abaixo, para que ela tenha as seguintes funcionalidades considerando A e B números inteiros com sinal (complemento de dois):

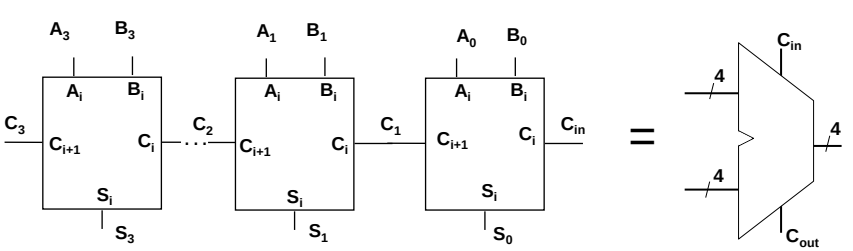
S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



Exercício -- ULA

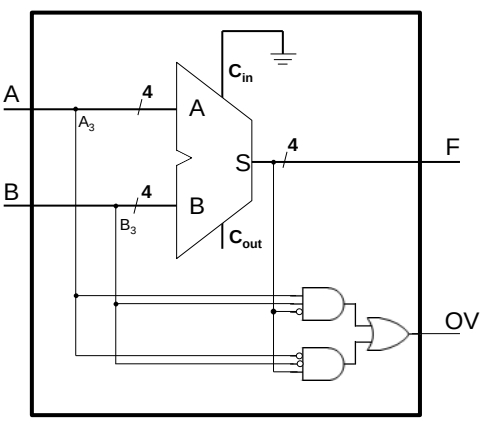
S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma





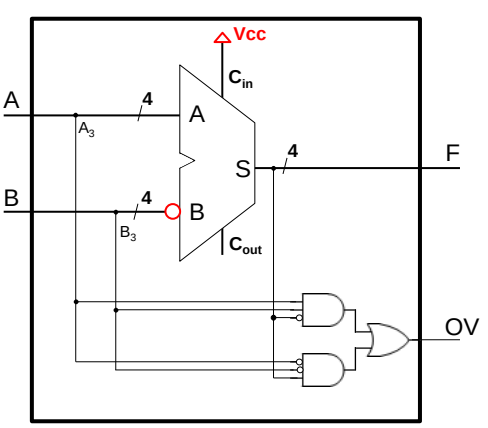
Exercício -- ULA

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma



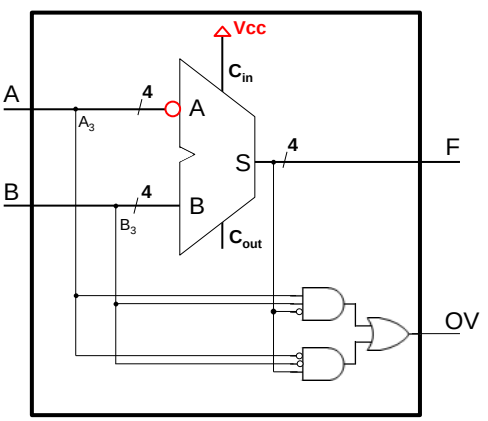
Exercício -- ULA

S1	S0	F	OV	Comentário
0	1	A menos B	1 se overflow	subtração



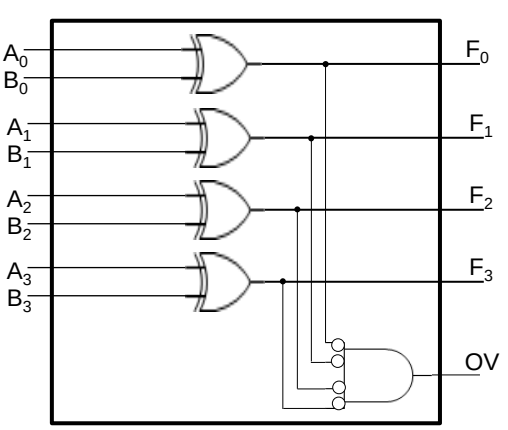
Exercício -- ULA

S1	S0	F	OV	Comentário
1	0	B menos A	1 se overflow	subtração inversa



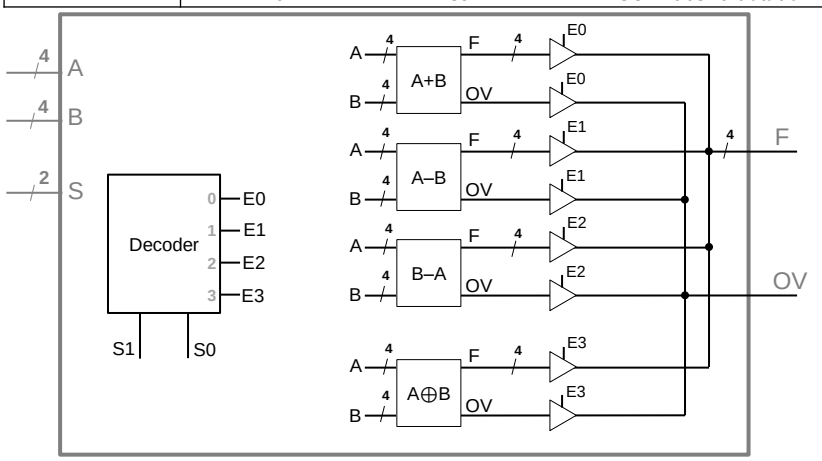
Exercício -- ULA

S1	S0	F	OV	Comentário
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



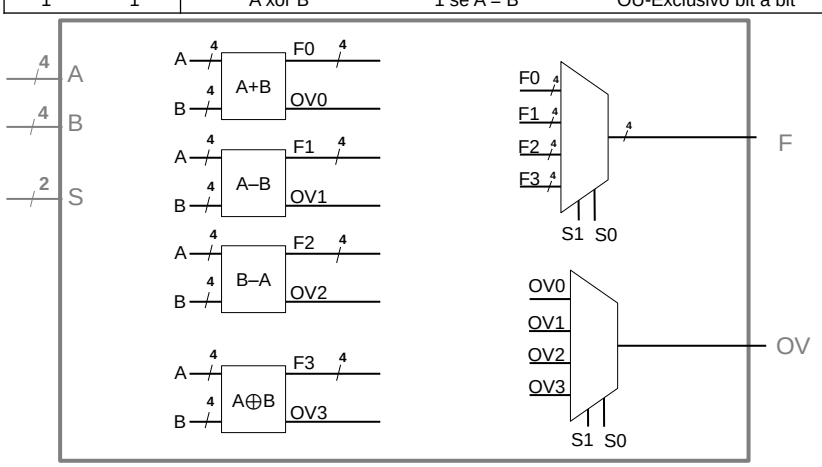
Exercício: Solução 1

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



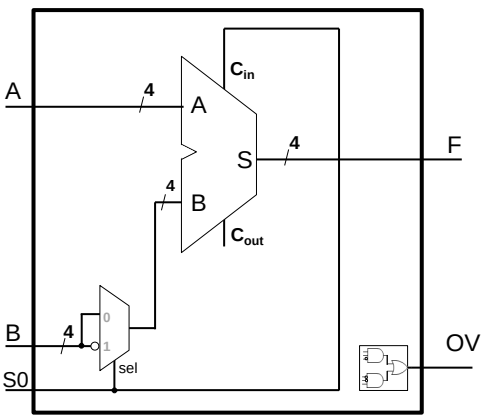
Exercício: Solução 2

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



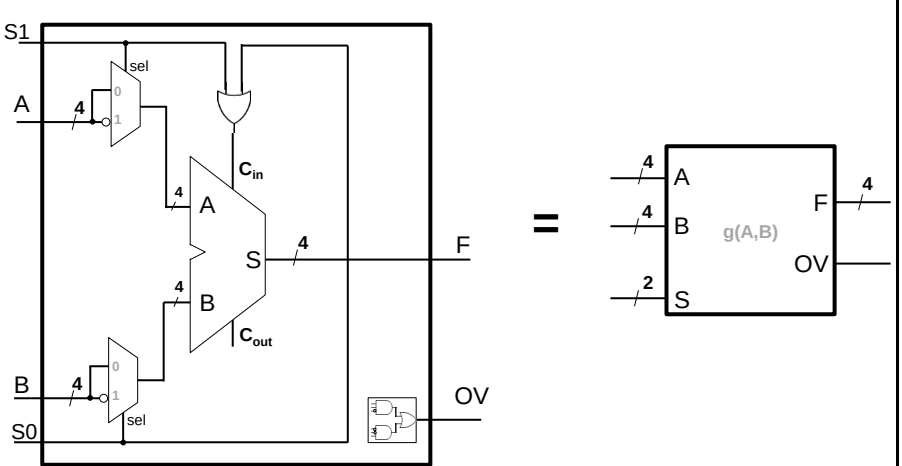
Exercício: Otimizações…

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração



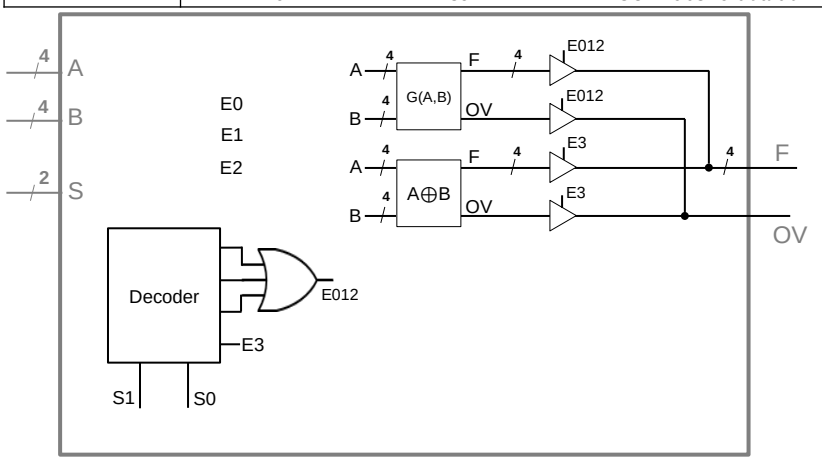
Exercício: Otimizações…

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa



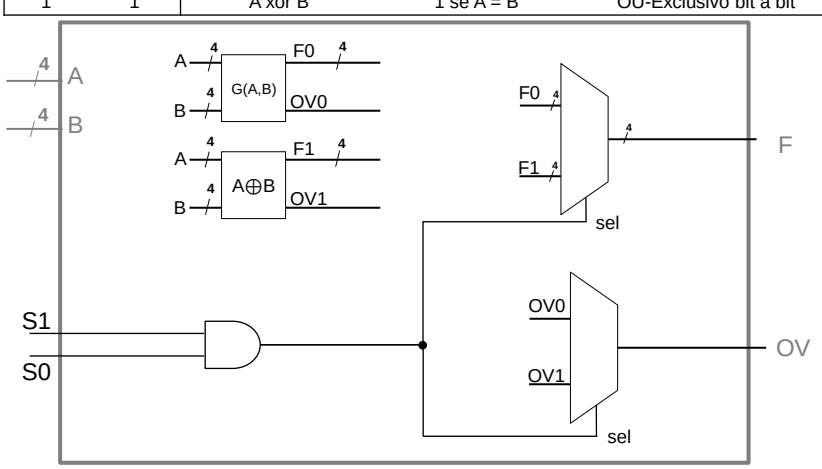
Exercício: Solução 1b

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



Exercício: Solução 2b

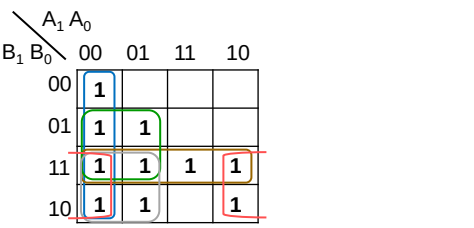
S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A = B	OU-Exclusivo bit a bit



Exercício: Uma modificação

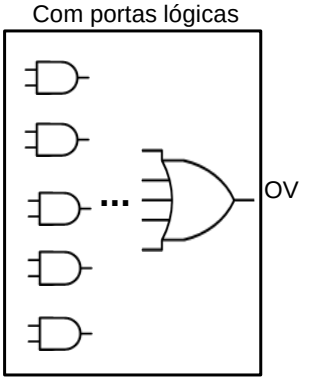
S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A_1A_0 <= B_1B_0	OU-Exclusivo bit a bit

- Implementar com: (a) portas lógicas, (b) decoder 4:16, (c) mux 16:1, (d) mux 8:1, (e) mux 4:1



$OV = A_1'A_0' + B_1B_0 + A_1'B_0 + A_1'B_1 + B_1A_0'$

Com portas lógicas

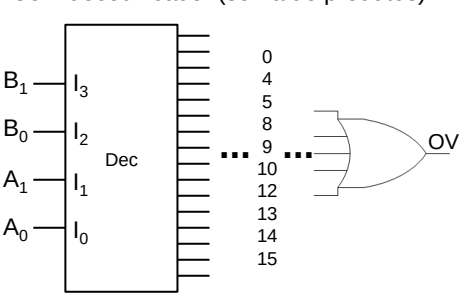


Exercício: Uma modificação

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A_1A_0 <= B_1B_0	OU-Exclusivo bit a bit

	B_1	B_0	A_1	A_0	OV
0	0	0	0	0	**1**
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	**1**
5	0	1	0	1	**1**
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	**1**
9	1	0	0	1	**1**
10	1	0	1	0	**1**
11	1	0	1	1	0
12	1	1	0	0	**1**
13	1	1	0	1	**1**
14	1	1	1	0	**1**
15	1	1	1	1	**1**

Com decodificador (soma de produtos)

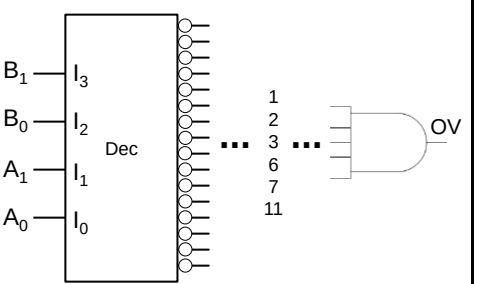


Exercício: Uma modificação

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se $A_1A_0 <= B_1B_0$	OU-Exclusivo bit a bit

	B_1	B_0	A_1	A_0	OV
0	0	0	0	0	1
1	0	0	0	1	**0**
2	0	0	1	0	**0**
3	0	0	1	1	**0**
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	**0**
7	0	1	1	1	**0**
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	**0**
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

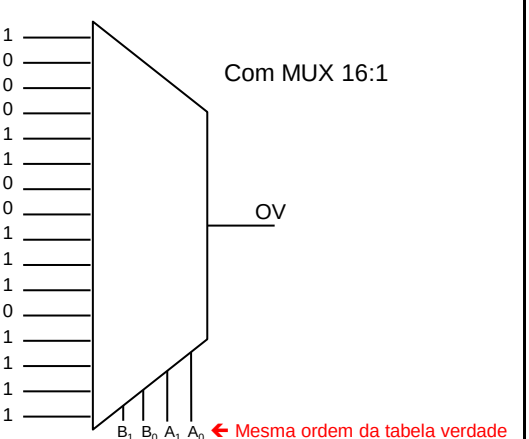
Com decodificador (produto de somas)



Exercício: Uma modificação

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se $A_1A_0 <= B_1B_0$	OU-Exclusivo bit a bit

	B_1	B_0	A_1	A_0	OV
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

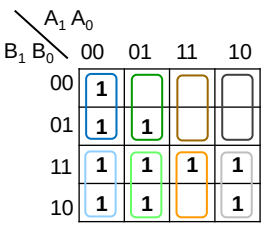


Com MUX 16:1

← Mesma ordem da tabela verdade

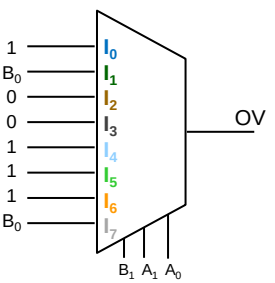
Exercício: Uma modificação

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A_1A_0 <= B_1B_0	OU-Exclusivo bit a bit



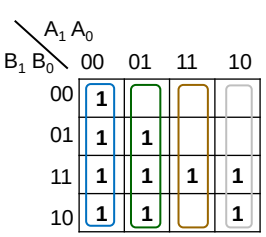
$OV = (B_1'A_1'A_0')\cdot 1 + (B_1'A_1'A_0)\cdot B_0 +$
$(B_1'A_1A_0)\cdot 0 + (B_1'A_1A_0')\cdot 0 +$
$(B_1A_1'A_0')\cdot 1 + (B_1A_1'A_0)\cdot 1 +$
$(B_1A_1A_0)\cdot B_0 + (B_1A_1A_0')\cdot 1$

Com MUX 8:1



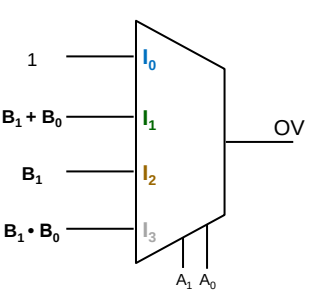
Exercício: Uma modificação

S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inversa
1	1	A xor B	1 se A_1A_0 <= B_1B_0	OU-Exclusivo bit a bit



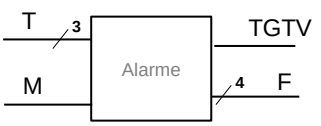
$OV = (A_1'A_0')\cdot 1 + (A_1'A_0)\cdot(B_1 + B_0) +$
$(A_1A_0)\cdot(B_1\cdot B_0) + (A_1A_0')\cdot B_1$

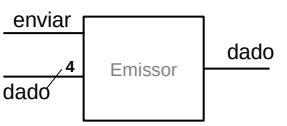
Com MUX 4:1



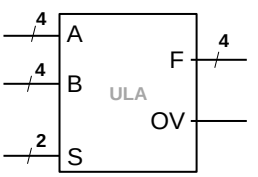
Exercício: Usando a ULA

- Usando a ULA criada e NOTs, deseja-se construir um circuito de **alarme** remoto a ser conectado a um emissor. A especificação é:
  - Se um sensor de temperatura T, de 3 bits, acusar um valor acima de V ("TGTV"=1), então deve ser enviado F = T–V a uma central
    - O valor de V é configurável pela entrada M: M=0 ➔ $V=3_{10}$; M=1 ➔ $V=5_{10}$
    - Por questões de compatibilidade com o sistema na central, a saída F deve ser: F= T–V caso M = 0, e F = compl1(T–V) se M=1



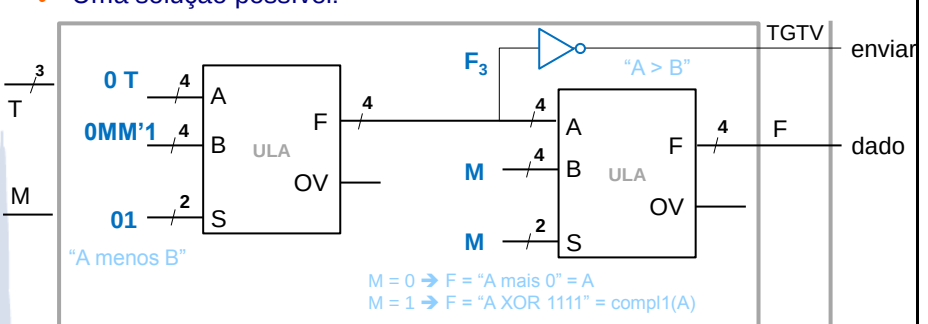


S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inv.
1	1	A xor B	1 se A = B	OU-Exclusivo



Exercício: Usando a ULA

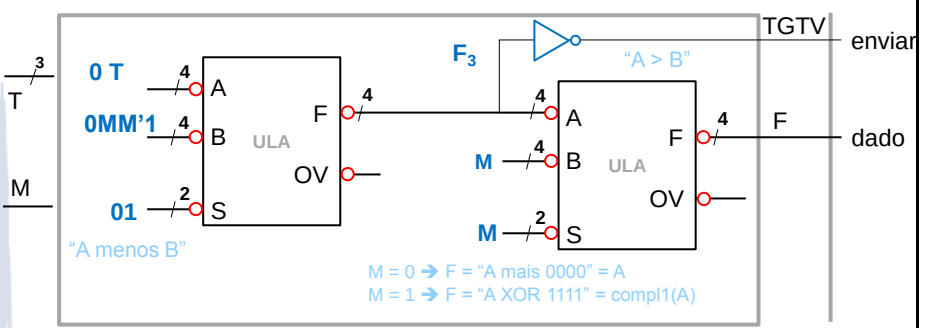
- Uma solução possível:



S1	S0	F	OV	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inv.
1	1	A xor B	1 se A = B	OU-Exclusivo

Exercício: Usando a ULA

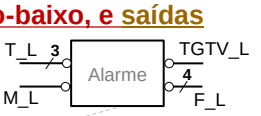
- Ao enviar o projeto para produção, você percebe que a ULA utilizada opera no modo **ativo baixo**. O que muda no projeto?

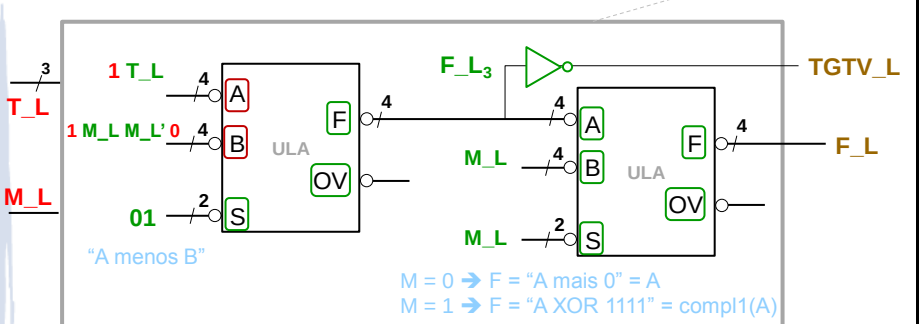


S1_L	S0_L	F_L	OV_L	Comentário
0	0	A mais B	1 se overflow	soma
0	1	A menos B	1 se overflow	subtração
1	0	B menos A	1 se overflow	subtração inv.
1	1	A xor B	1 se A = B	OU-Exclusivo

Exercício: Usando a ULA

- **Circuito inteiro** em ativo-baixo: basta que os sinais ligados nas **entradas sejam consideradas como ativo-baixo, e saídas sejam consideradas ativo-baixo**
  - Notação: "_L"

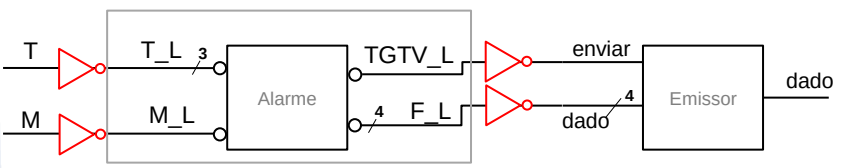




- **Sinais internos não são afetados**

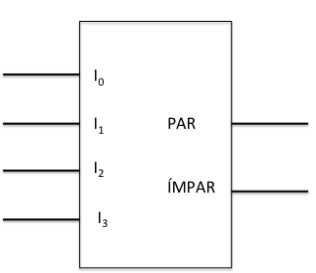
Exercício: Usando a ULA

- **Circuito inteiro** com lógica negativa: basta que os sinais ligados nas entradas sejam consideradas como ativo-baixo, e saídas sejam consideradas ativo-baixo
  - A única coisa que mudaria mais profundamente é o **acoplamento** entre módulos ativo-alto e ativo-baixo

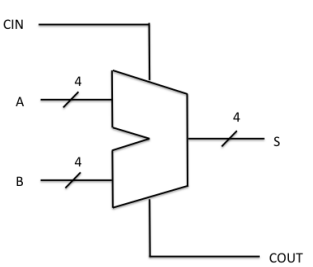


Exercício (PREC 2016)

- Seja A um número binário de 4 bits. Projete um circuito que calcule $A - 1_{10}$ caso A tenha paridade par e $A - 2_{10}$ caso A tenha paridade ímpar. A subtração deve ser calculada em Complemento de 2. Utilize (somente!) um gerador de paridade de 4 bits (Figura 1) e um somador completo de 4 bits (Figura 2).



Gerador de Paridade



Somador Completo de 4 bits

Lição de Casa

- Leitura Obrigatória:
  - Capítulo 6 do Livro Texto, ênfase em 6.4, 6.5, 6.7, 6.8.
- Exercícios obrigatórios:
  - Capítulo 6 do Livro Texto.

Bibliografia Adicional

- Fregni, Edson; Saraiva, Antônio. *Engenharia do Projeto Lógico Digital*. Editora Edgard Blücher Ltda. São Paulo, SP, Brasil, 1.995;
- Ranzini, Edith; Fregni, Edson. Teoria da Comutação: Introdução aos Circuitos Digitais (Partes 1 e 2). Notas de Aula de PCS214, PCS/EPUSP, Agosto de 1.996.
- Tocci, Ronald; Widmer, Neal S. Sistemas Digitais – Princípios e Aplicações. 8ª Edição, Pearson/Prentice-Hall, São Paulo, 2.003.