

# Projeto de ULA

**Cris – DRE: XXXXXXXXXX**

**Gus – DRE: XXXXXXXXXX**

**Leo – DRE: XXXXXXXXXX**

<sup>1</sup>Universidade Federal do Rio de Janeiro (UFRJ)

leonardongc@poli.ufrj.br

## 1. Enunciado

Fazer ULA de 2 operandos(**A** e **B**) de 4 bits com e 8 operações:

- $A + B$
- $A - B$
- $-A$
- $A + 1$
- Até 2 Operações de Comparação Bit a Bit
- O Restante das Operações A Escolha do Grupo Utilizando  $A$  e  $B$

Com os Flags de **Zero, Negativo, Carry & Overflow**

## 2. Escolhas

### 2.1. Lista de Operações Atualizadas

Escolhemos utilizar XOR e ??? pois utilizam submódulos já existentes para outras operações e as operações de deslocamento de bits a direita e a esquerda.

- Soma
- Subtração
- Inversão
- Incremento de 1
- XOR
- ????
- Right Bitshift
- Left Bitshift

### 2.2. Projeto do Somador

Separamos a soma em duas etapas, uma que faz a antecipação dos carries e outra que faz a soma bit a bit com os carries.

#### 2.2.1. Antecipador

Calculado Carry a Carry:

$$Cin_0 = 0$$

$$Cin_1 = A_0.B_0$$

$$\begin{aligned}
Cin_2 &= A_1.B_1 + Cin_1(A_1 + B_1) = A_1.B_1 + A_0.B_0(A_2 + B_2) \\
Cin_3 &= A_2.B_2 + Cin_2(A_2 + B_2) = A_2.B_2 + A_1.B_1 + A_0.B_0(A_2 + B_2)(A_2 + B_2) \\
Cin_4^* &= A_3.B_3 + Cin_3(A_3 + B_3) = A_3.B_3 + A_2.B_2 + A_1.B_1 + A_0.B_0(A_2 + B_2)(A_2 + B_2)(A_3 + B_3)
\end{aligned}$$

### 2.2.2. Somador

Com os carries antecipados é possível calcular a soma bit a bit:

$$F_x = A_x \oplus B_x \oplus Cin_x$$

### 2.3. Projeto do Inversor

O inversor de complemento de 2 tem a seguinte tabela:

$A_3$	$A_2$	$A_1$	$A_0$	$F_3$	$F_2$	$F_1$	$F_0$
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

Repare que para  $A = 1000_{bin} = -8_{dec}$  pela regra ( $F = A + 1$ ) deveria resultar em 1000, mas como não existe representação válida para o número 8 o resultado é 0000 e o flag de overflow é acionado.

$$F_3 = \overline{A_3}.A_2 + \overline{A_3}.A_1 + \overline{A_3}.A_0$$

$$F_2 = \overline{A_2}.A_1 + \overline{A_2}.A_0 + A_2.\overline{A_1}.\overline{A_0}$$

$$F_1 = \overline{A_1}.A_0 + A_1.\overline{A_0}$$

$$F_0 = A_0$$

### 2.4. Projeto do Subtrator

Com os módulos de soma e inversão prontos é possível utilizar seus submódulos para construir uma operação de subtração.

## 2.5. Projeto do Incremento de 1

Utilizamos o módulo de soma substituindo o operando  $B$  pelo vetor  $B_3, B_2, B_1, B_0 = 0$  e a entrada  $Cin_0 = 1$

## 2.6. Projeto do Comparador XOR

Desconsiderando o submódulo antecipador na soma o submódulo somador vale  $A \oplus B$

$$F_x = A_x \oplus B_x$$

## 2.7. Projeto do Comparador ????

## 2.8. Projeto do Left Bit Shifter

Montando a Tabela Verdade Para operações de Shift a esquerda é possível verificar que é certo que não haverá resultado não nulo para  $B > 3$ , portanto podemos considerar apenas os 2 bits menos significativos do operando  $B$ :  $B_1 \& B_0$ :

$B_1$	$B_0$	$F_3$	$F_2$	$F_1$	$F_0$
0	0	$A_3$	$A_2$	$A_1$	$A_0$
0	1	$A_2$	$A_1$	$A_0$	0
1	0	$A_1$	$A_0$	0	0
1	1	$A_0$	0	0	0

$$F_3 = \overline{B_1} \cdot \overline{B_0} \cdot A_3 + \overline{B_1} \cdot B_0 \cdot A_2 + B_1 \cdot \overline{B_0} \cdot A_1 + B_1 \cdot B_0 \cdot A_0$$

$$F_2 = \overline{B_1} \cdot \overline{B_0} \cdot A_2 + \overline{B_1} \cdot B_0 \cdot A_1 + B_1 \cdot \overline{B_0} \cdot A_0$$

$$F_1 = \overline{B_1} \cdot \overline{B_0} \cdot A_1 + \overline{B_1} \cdot B_0 \cdot A_0$$

$$F_0 = \overline{B_1} \cdot \overline{B_0} \cdot A_0$$

## 2.9. Projeto do Right Bit Shifter

O Shift a direita por sua vez forma uma tabela similar com as mesmas entradas:

$B_1$	$B_0$	$F_3$	$F_2$	$F_1$	$F_0$
0	0	$A_3$	$A_2$	$A_1$	$A_0$
0	1	0	$A_3$	$A_2$	$A_1$
1	0	0	0	$A_3$	$A_2$
1	1	0	0	0	$A_3$

$$F_3 = \overline{B_1} \cdot \overline{B_0} \cdot A_3$$

$$F_2 = \overline{B_1} \cdot \overline{B_0} \cdot A_2 + \overline{B_1} \cdot B_0 \cdot A_3$$

$$F_1 = \overline{B_1} \cdot \overline{B_0} \cdot A_1 + \overline{B_1} \cdot B_0 \cdot A_2 + B_1 \cdot \overline{B_0} \cdot A_3$$

$$F_0 = \overline{B_1} \cdot \overline{B_0} \cdot A_0 + \overline{B_1} \cdot B_0 \cdot A_1 + B_1 \cdot \overline{B_0} \cdot A_2 + B_1 \cdot B_0 \cdot A_3$$

Como as operações de translado de bits ocorrem num mesmo domínio, é possível combiná-las acoplando suas tabelas e diferenciando as por um único bit de seleção de operação  $S$  formando um único submódulo:

$S$	$B_1$	$B_0$	$F_3$	$F_2$	$F_1$	$F_0$
0	0	0	$A_3$	$A_2$	$A_1$	$A_0$
0	0	1	0	$A_3$	$A_2$	$A_1$
0	1	0	0	0	$A_3$	$A_2$
0	1	1	0	0	0	$A_3$
1	0	0	$A_3$	$A_2$	$A_1$	$A_0$
1	0	1	$A_2$	$A_1$	$A_0$	0
1	1	0	$A_1$	$A_0$	0	0
1	1	1	$A_0$	0	0	0

Nota: as operações de bitshift estão apenas interessadas no vetor de bits, sendo a representação de sinal ignorada para isso.

### 3. Execução

#### 3.1. Código dos Submódulos

##### 3.1.1. Módulo Antecipador

##### 3.1.2. Módulo Somador

##### 3.1.3. Módulo Inversor

##### 3.1.4. Módulo Shifter

#### 3.2. Implementação dos Flags

##### 3.2.1. Flag de Zero

O flag de zero será acionado para cada operação da forma:

- **Soma**  
Se um operando for inverso do outro(em complemento de 2) ou os números forem zero.
- **Inversão**  
Se o vetor de entrada valer 0.
- **Subtração**  
Se o os operandos forem iguais.
- **Incremento de 1**  
Se o operando for o vetor 1111
- **Comparador XOR**  
Se os operandos forem iguais.
- **Comparador ???**  
???????
- **RBS**  
Se o operando que determina o deslocamento ( $B$ ) for maior que o índice do 1 mais a esquerda do operando ( $A$ ) aciona o flag de 0.
- **LBS**  
Se o vetor de entrada  $A$  valer 0.

### 3.2.2. Flag de Negativo

Da mesma forma que o flag de 0, flag de negativo será acionado:

- **Soma**  
Se os operandos forem negativos ou se um operando negativo for maior que o positivo.
- **Inversão**  
Se o vetor de entrada for diferente de 0 e tiver o bit de sinal 0.
- **Subtração**  
AFFFF.
- **Incremento de 1**  
Se o operando tiver bit de sinal igual a 1 e não for o vetor 1111.
- **Comparador XOR**  
Essa operação não produz negativos
- **Comparador ???**  
???????
- **RBS e LBS**  
As operações de bitshift desconsideram a representação de sinal, portanto não aciona o flag de negativo.

### 3.2.3. Flag de Carry

O flag de zero serve para guardar informação de *vai um* para uma operação futura:

- **Soma**  
Tendo os bits de sinal iguais o flag de carry será o sinal  $Cin_3$  do submódulo antecipador.
- **Inversão**  
  
Inversão apenas ativa carry para o número  $-8$  (vetor 1000) com a saída 0000 zero.
- **Subtração**

Da mesma forma que a soma, com o mesmo sinal.

- **Incremento de 1**  
  
o único vetor que vai um é o número 7, pois é o número que aciona  $Cin_3$  na soma.
- **Comparador XOR**  
Um comparador não gera sinal de vai um.
- **Comparador ???**  
???????
- **RBS**

Translados de bits à direita não geram carries.

- **LBS**

Como o bit de sinal é ignorado para essa operação não faz sentido considerar o vai um.

### 3.2.4. Flag de Overflow

Por sua vez o flag de overflow será acionado quando os 4 bits não forem o suficiente para representar o resultado:

- **Soma**
- **Inversão**

Como a inversão ocorre num espaço fechado de possibilidades representáveis ela não gera flag de overflow

- **Subtração**
- **Incremento de 1**

O mesmo critério da soma.

- **Comparador XOR**

Comparadores bit a bit não geram overflow.

- **Comparador ???**

???????

- **RBS**

- **LBS**