

Universidade Estadual do Oeste do Paraná (UNIOESTE)

Centro de Engenharias e ciências exatas (CECE)

Ciência da Computação

TRABALHO – SISTEMAS DIGITAIS

Gabriel José Biudes Lino

Leonardo Huang

Professor Jorge Habib Hanna El Khouri

Foz do Iguaçu, PR

Maio de 2022

INTRODUÇÃO

Este documento foi produzido com o intuito de documentar e descrever todo o processo de desenvolvimento do trabalho proposto em sala de aula. Sendo assim, ao decorrer do documento, será mostrado um resumo da arquitetura do sistema digital produzido para desenvolver o papel de uma unidade lógica e aritmética, assim como os circuitos completos desenvolvidos no Circuit.js.

DETALHAMENTO DO PROBLEMA

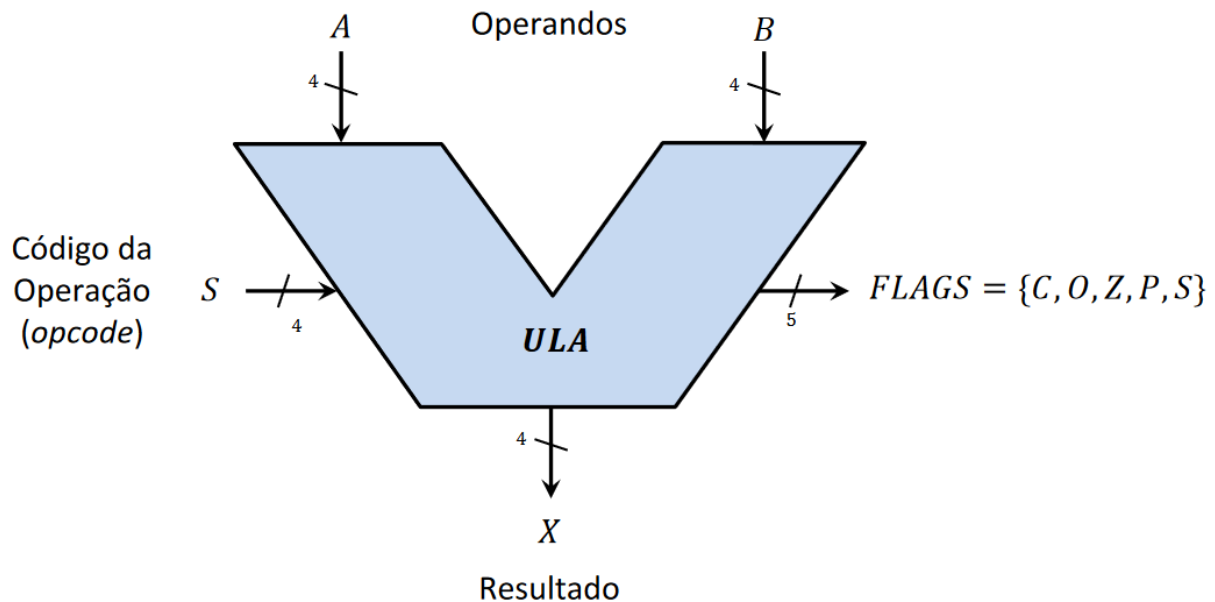
Primeiramente, é importante definir qual é o objetivo do projeto. É necessário que seja desenvolvida uma Unidade Lógica e Aritmética de 4 bits que é capaz de realizar 16 operações diferentes, necessitando assim de 4 bits como entrada de dados para o código da operação. Além disso, é necessário também que a ULA entregue 5 flags específicos, sendo estes o conjunto de flags {C, O, Z, P, S}.

Tabela de operações:

D3	D2	D1	D0	OPERAÇÃO
0	0	0	0	$X = A + B$
0	0	0	1	$X = A - B$
0	0	1	0	$X = A + 1$
0	0	1	1	$X = A - 1$
0	1	0	0	$X = -A$
0	1	0	1	$X = A$
0	1	1	0	$X = -1$
0	1	1	1	$X = A + A$
1	0	0	0	$X = A \wedge B$
1	0	0	1	$X = A \vee B$
1	0	1	0	$X = A \oplus B$
1	0	1	1	$X = !A$
1	1	0	0	$X = A \wedge !B$
1	1	0	1	$X = !A \wedge B$
1	1	1	0	$X = !(A \wedge B)$
1	1	1	1	$X = !(A \vee B)$

ARQUITETURA DA SOLUÇÃO

Analisando de forma mais superficial, a “caixinha preta” responsável pela resolução do problema se mostra conforme definido no enunciado do projeto:



Para isso, será montada uma tabela verdade com as entradas de dados possíveis e seus respectivos resultados. A partir desta tabela, será apresentado o Mapa de Karnaugh para os valores de X e do Carry Out, que nos dará as expressões lógicas. Com estas em mãos, é possível montar os circuitos no circuit.js, produzindo uma ULA de 1 bit. A partir da ULA de 1 bit, será montada uma ULA de 4 bits, onde também será possível mostrar os cinco flags restantes.

CIRCUIT.JS

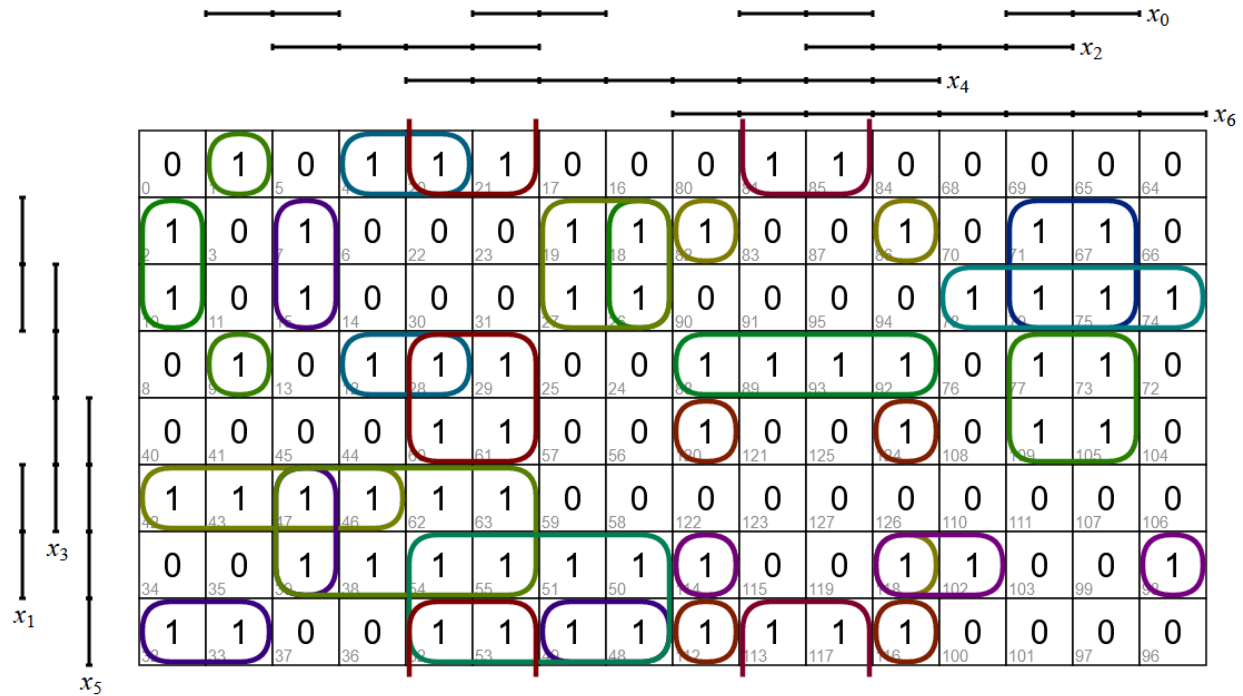
Primeiramente, foi criada uma tabela verdade contendo todas as possibilidades para as entradas de {s3, s2, s1, s0, cin, A, B} e suas respectivas saídas em Cout e X:

S3	S2	S1	S0	CIN	A	B	COUT	X
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	0	1
0	0	0	0	1	0	1	1	0
0	0	0	0	1	1	0	1	0
0	0	0	0	1	1	1	1	1
0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	1	1	1
0	0	0	1	0	1	0	0	1

0	0	0	1	0	1	1	0	0
0	0	0	1	1	0	0	1	1
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	1	0	1
0	0	1	0	1	0	0	0	1
0	0	1	0	1	0	1	0	1
0	0	1	0	1	1	0	1	0
0	0	1	0	1	1	1	1	0
0	0	1	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	0	0	1
0	0	1	1	0	1	1	0	1
0	0	1	1	1	0	0	1	1
0	0	1	1	1	0	1	1	1
0	0	1	1	1	1	0	0	0
0	0	1	1	1	1	1	0	0
0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0	1
0	1	0	0	0	1	0	0	0
0	1	0	0	0	1	1	0	0
0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	1	1	0
0	1	0	0	1	1	0	0	1
0	1	0	0	1	1	1	0	1
0	1	0	1	0	0	0	0	0
0	1	0	1	0	0	1	0	0
0	1	0	1	1	0	0	0	0
0	1	0	1	1	1	0	0	1
0	1	0	1	1	1	1	0	1
0	1	1	0	0	0	0	0	1
0	1	1	0	0	0	1	0	1
0	1	1	0	0	1	0	0	1

0	1	1	0	0	1	1	0	1
0	1	1	0	1	0	0	0	1
0	1	1	0	1	0	1	0	1
0	1	1	0	1	1	0	0	1
0	1	1	0	1	1	1	0	1
0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1	0
0	1	1	1	0	1	1	1	0
0	1	1	1	1	0	0	0	1
0	1	1	1	1	0	1	0	1
0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0	0
1	0	0	0	0	1	1	0	1
1	0	0	0	1	0	0	0	0
1	0	0	0	1	0	1	0	0
1	0	0	0	1	1	0	0	0
1	0	0	0	1	1	1	0	1
1	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	0	0
1	0	0	1	1	0	1	0	1
1	0	0	1	1	1	0	0	1
1	0	0	1	1	1	1	0	1
1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	1
1	0	1	0	0	1	0	0	1
1	0	1	0	0	1	1	0	0
1	0	1	0	1	0	0	0	0
1	0	1	0	1	0	1	0	1
1	0	1	0	1	1	0	0	1
1	0	1	0	1	1	1	0	0
1	0	1	1	0	0	0	0	1
1	0	1	1	0	0	1	0	1
1	0	1	1	0	1	0	0	0

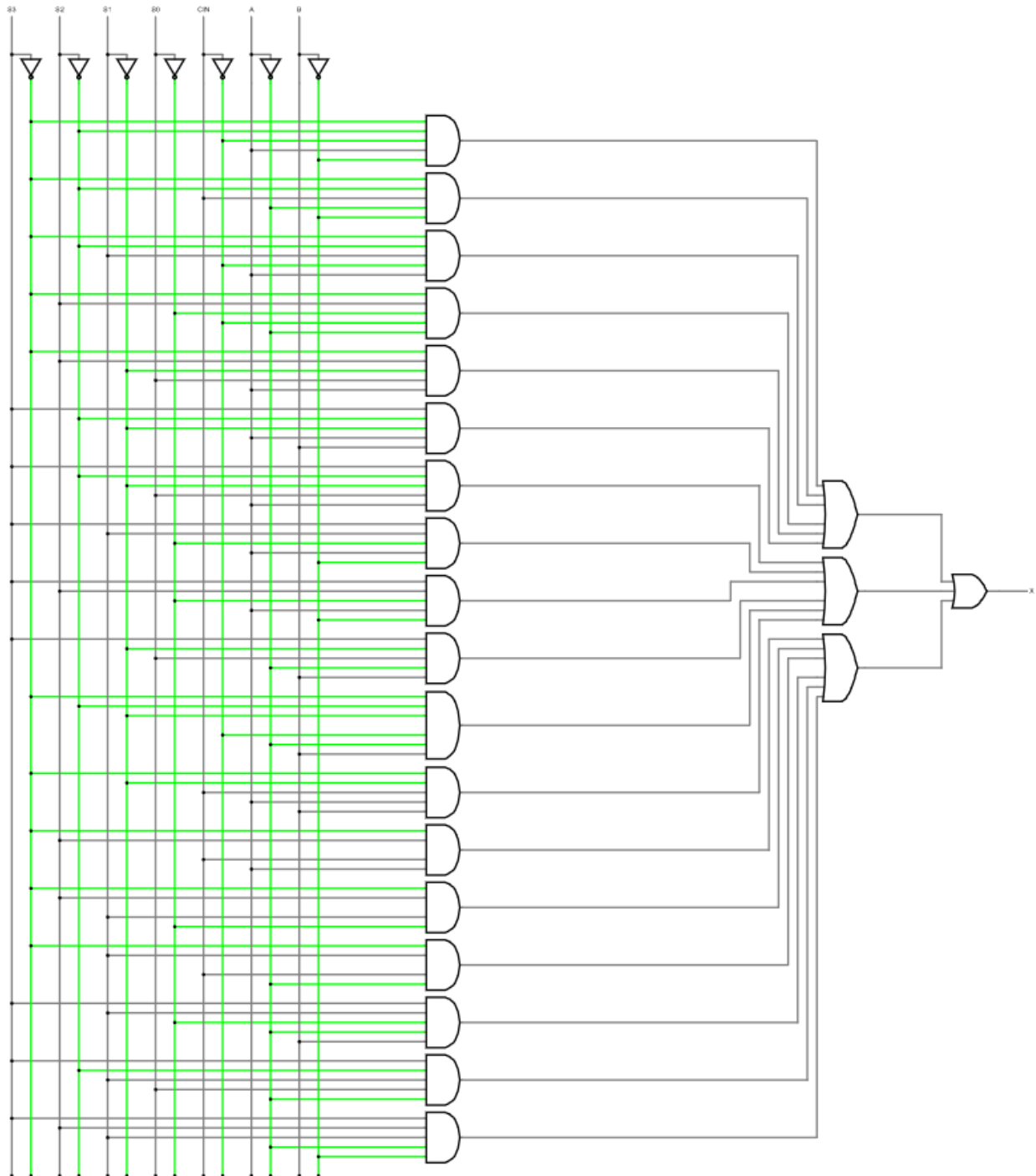
$X = \{1, 2, 4, 7, 9, 10, 12, 15, 18, 19, 20, 21, 26, 27, 28, 29, 32, 33, 38, 39, 42, 43, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 67, 71, 73, 74, 75, 77, 78, 79, 81, 82, 85, 86, 88, 89, 92, 93, 98, 102, 105, 109, 112, 113, 114, 116, 117, 118, 120, 124\}$



$$y = (\bar{x}_6\bar{x}_5\bar{x}_2x_1\bar{x}_0) \vee (\bar{x}_6\bar{x}_5\bar{x}_2x_1\bar{x}_0) \vee (\bar{x}_6\bar{x}_5x_4\bar{x}_2x_1) \vee (\bar{x}_6x_5\bar{x}_3\bar{x}_2\bar{x}_1) \vee (\bar{x}_6x_5\bar{x}_4x_3x_1) \vee (x_6\bar{x}_5\bar{x}_4x_1x_0) \vee (x_6\bar{x}_5\bar{x}_4x_3x_1) \vee (x_6x_4\bar{x}_3x_1\bar{x}_0) \vee (x_6x_5\bar{x}_3x_1\bar{x}_0) \vee (x_6\bar{x}_4x_3\bar{x}_1x_0) \vee (\bar{x}_6\bar{x}_5\bar{x}_4\bar{x}_2\bar{x}_1x_0) \vee (\bar{x}_6\bar{x}_4x_2x_1x_0) \vee (\bar{x}_6x_5x_2x_1) \vee (\bar{x}_6x_5x_4\bar{x}_3) \vee (\bar{x}_6x_4x_2\bar{x}_1) \vee (x_6x_4\bar{x}_3\bar{x}_1x_0) \vee (x_6\bar{x}_5x_4x_3\bar{x}_1) \vee (x_6x_5x_4\bar{x}_1\bar{x}_0)$$

Onde $(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = (s_3, s_2, s_1, s_0, \text{cin}, a, b)$.

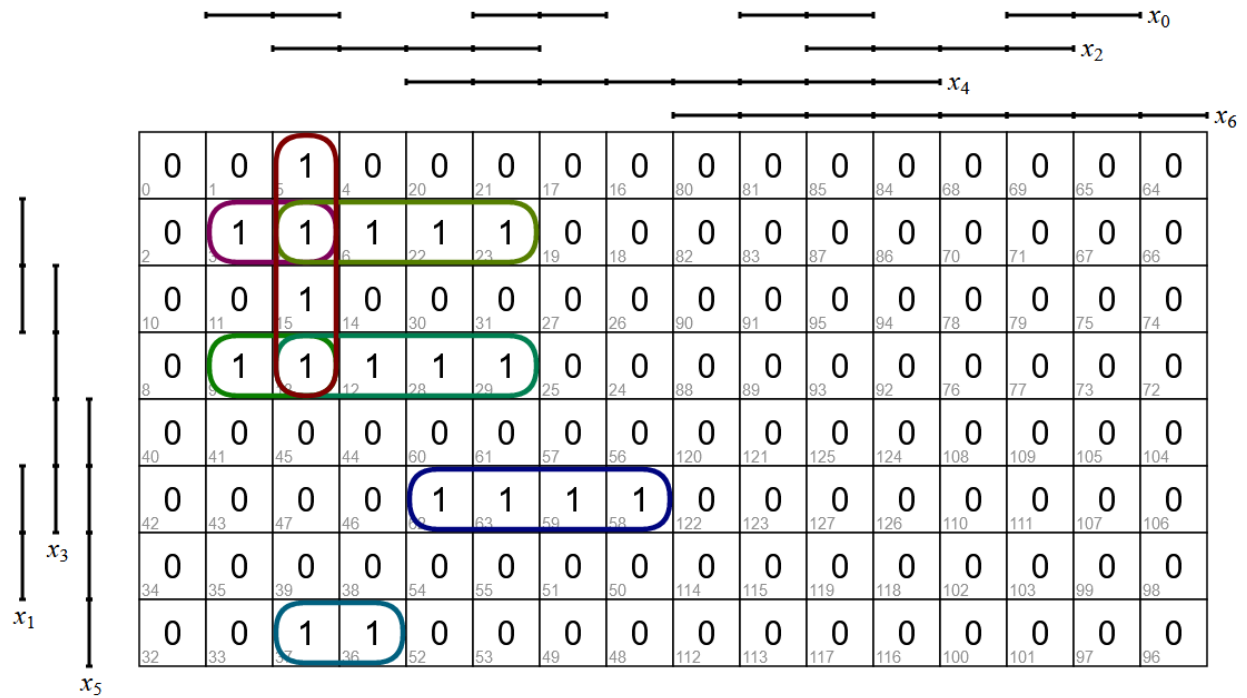
Com a expressão lógica em mãos, foi possível montar o circuito do resultado no circuit.js:



<https://tinyurl.com/22j26kqi>

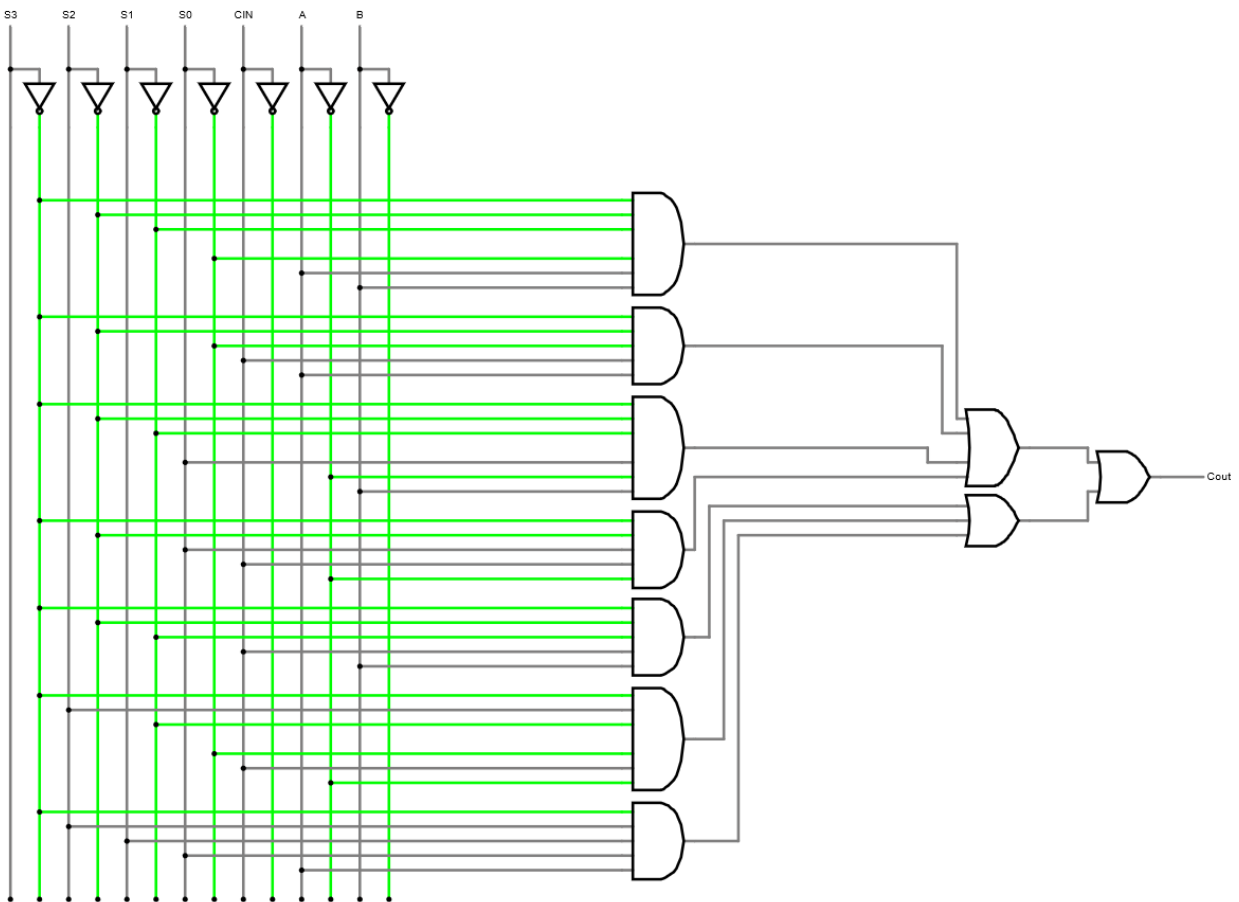
Além disso, também foi montado o Mapa de Karnaugh para o carry:

$$X = \{3, 5, 6, 7, 9, 12, 13, 15, 22, 23, 28, 29, 36, 37, 58, 59, 62, 63\}$$



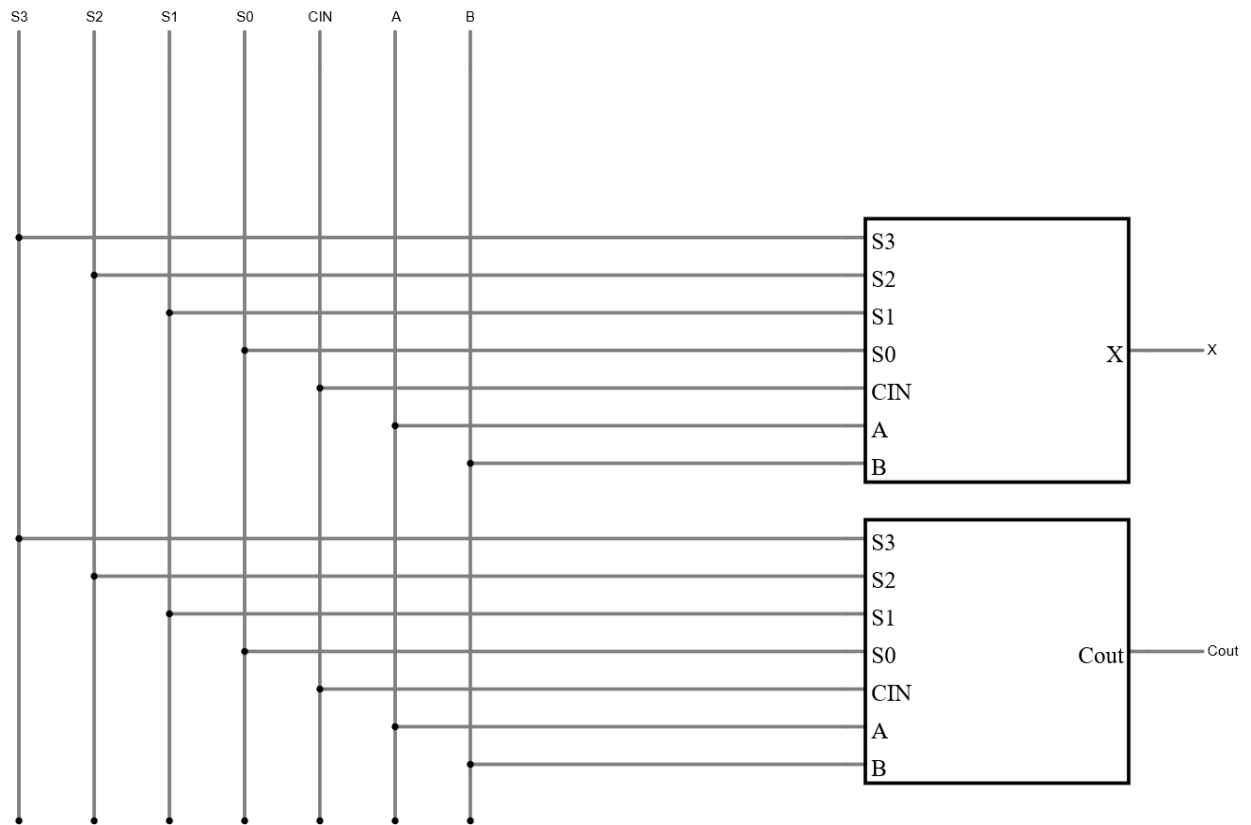
$$y = (\bar{x}_6 \bar{x}_5 \bar{x}_4 \bar{x}_3 x_1 x_0) \vee (\bar{x}_6 \bar{x}_5 \bar{x}_3 x_2 x_1) \vee (\bar{x}_6 \bar{x}_5 \bar{x}_4 x_3 \bar{x}_1 x_0) \vee (\bar{x}_6 \bar{x}_5 x_3 x_2 \bar{x}_1) \vee (\bar{x}_6 \bar{x}_5 \bar{x}_4 x_2 x_0) \vee (\bar{x}_6 x_5 \bar{x}_4 \bar{x}_3 x_2 \bar{x}_1) \vee (\bar{x}_6 x_5 x_4 x_3 x_1)$$

Com a expressão lógica em mãos, foi possível montar o circuito do carry out no circuit.js:



<https://tinyurl.com/2cwk85d4>

Assim, com ambos os circuitos em mãos, é possível fazer uma ULA de 1 bit:

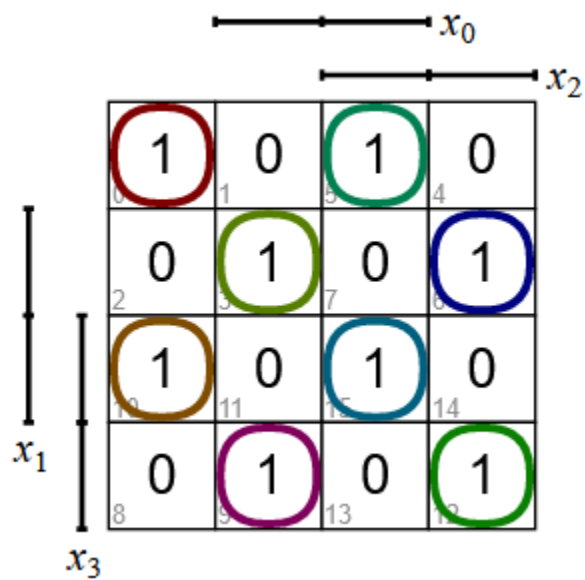


<https://tinyurl.com/27emf66o>

Após isso, foi montada a tabela verdade para o flag P:

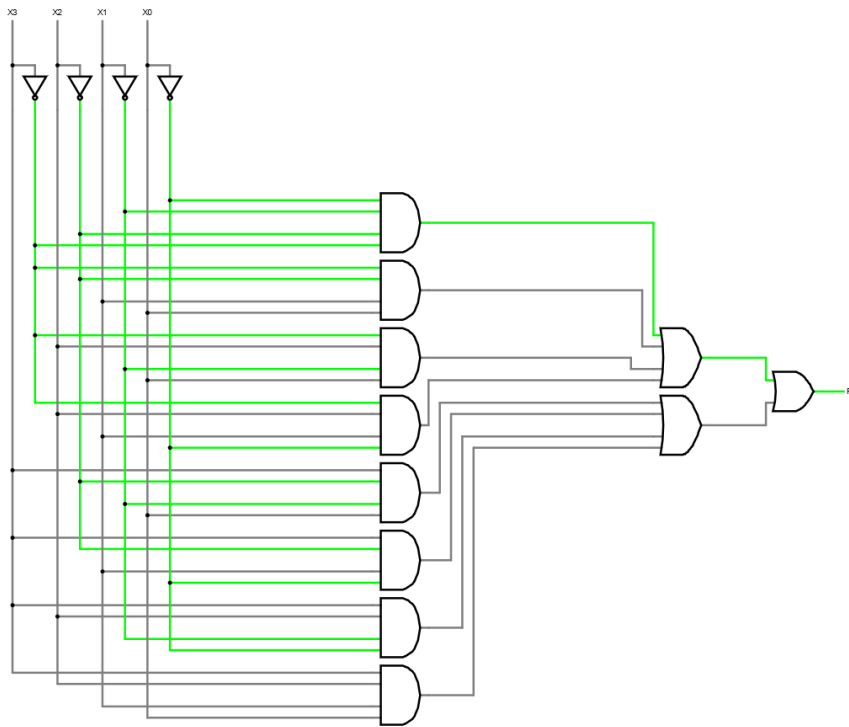
x3	x2	x1	x0	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Mapa de Karnaugh de P:



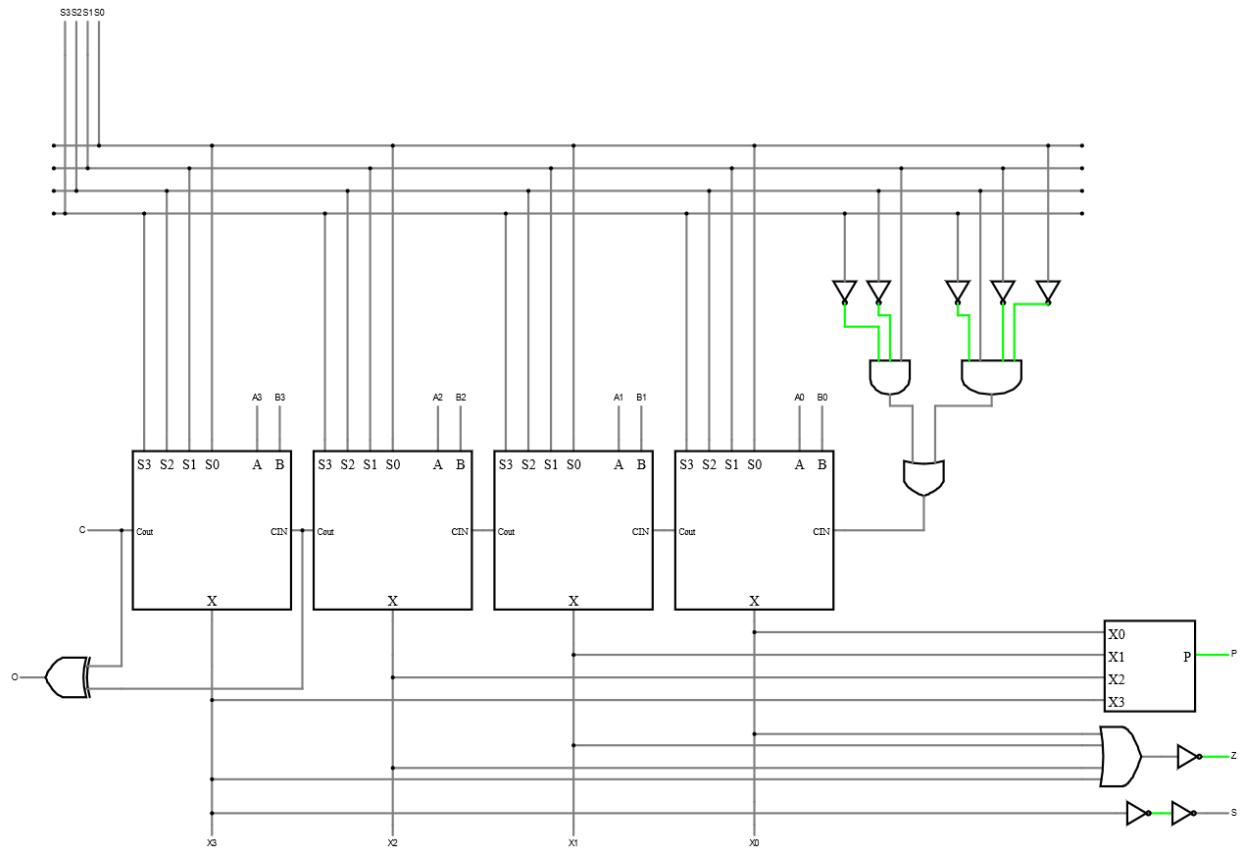
$$P = (!x3 !x2 !x1 !x0) \vee (!x3 !x2 x1 x0) \vee (!x3 x2 !x1 x0) \vee (!x3 x2 x1 !x0) \vee (x3 !x2 !x1 x0) \vee (x3 !x2 x1 !x0) \vee (x3 x2 !x1 !x0) \vee (x3 x2 x1 x0)$$

Circuito:



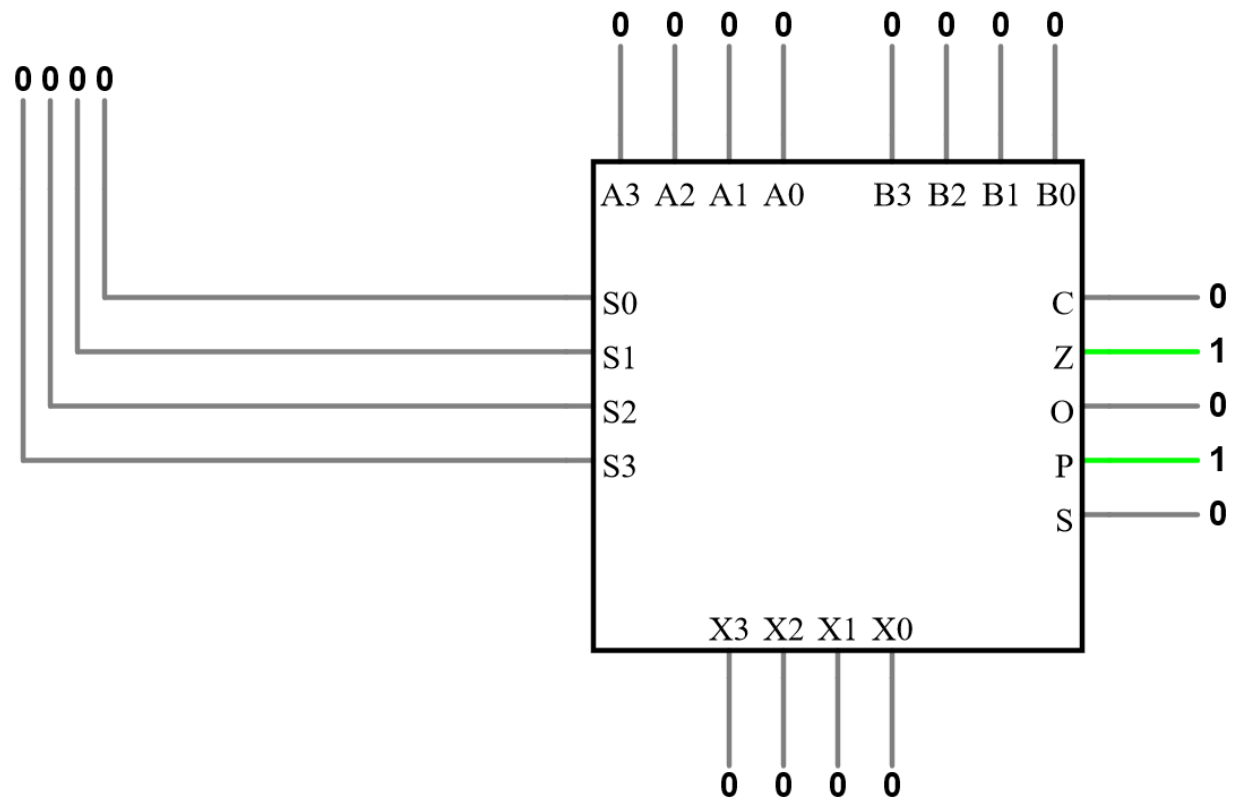
<https://tinyurl.com/yyk55zhs>

Como o flag S é simplesmente o dígito mais significativo do resultado E o flag O é um xor entre os últimos 2 carry outs, podemos juntar todos os componentes e finalizar a ULA de 4 bits:



<https://tinyurl.com/29y4frpd>

Assim, juntando tudo em uma caixinha preta, temos o circuito seguinte:



<https://tinyurl.com/22yf2bvx>

Que conclui, assim, a ULA de 4 bits no circuit.js.

REFERÊNCIAS

<https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/code/karnaughmap/>