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 Falstad e no Tinkercad.

DETALHAMENTO DO PROBLEMA

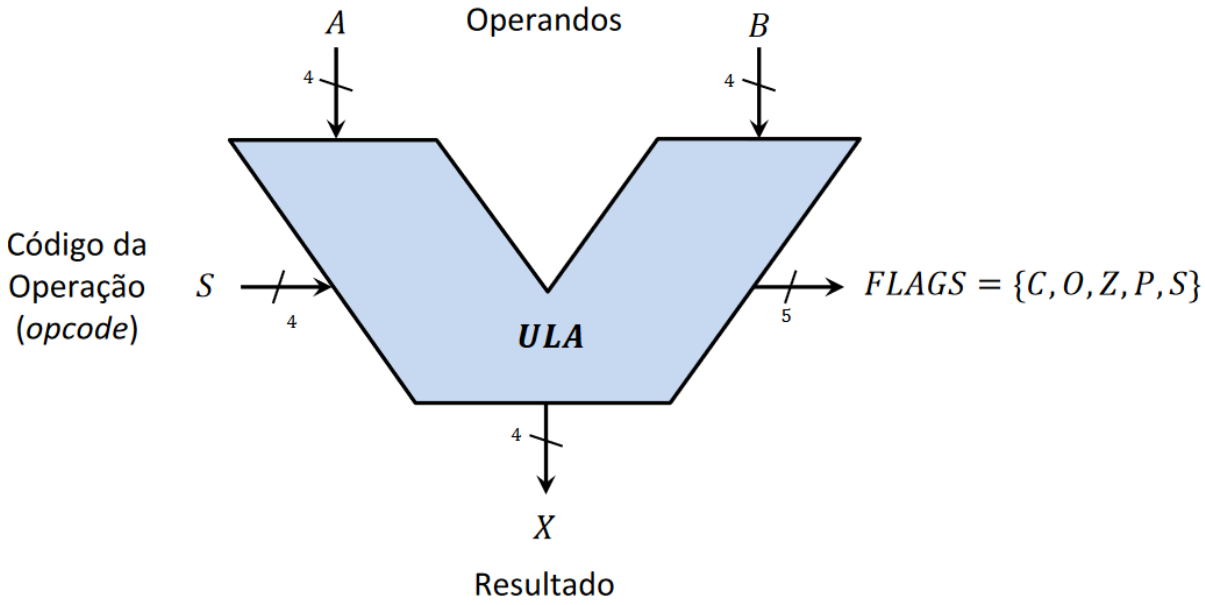
Primeiramente, é importante definir qual é o objetivo do projeto. É necessário que seja desenvolvida uma Unidade Lógica e Aritmética de 4 bits que é capaz de realizar 16 operações diferentes, necessitando assim de 4 bits como entrada de dados para o código da operação. Além disso, é necessário também que a ULA entregue 4 flags específicos, sendo estes o conjunto de flags {C, 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 ^ B
1	0	0	1	X = A v B
1	0	1	0	X = A $\oplus$ B
1	0	1	1	X = !A
1	1	0	0	X = A ^ !B
1	1	0	1	X = !A ^ B
1	1	1	0	X = !(A ^ B)
1	1	1	1	X = !(A v 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, 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 quatro 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

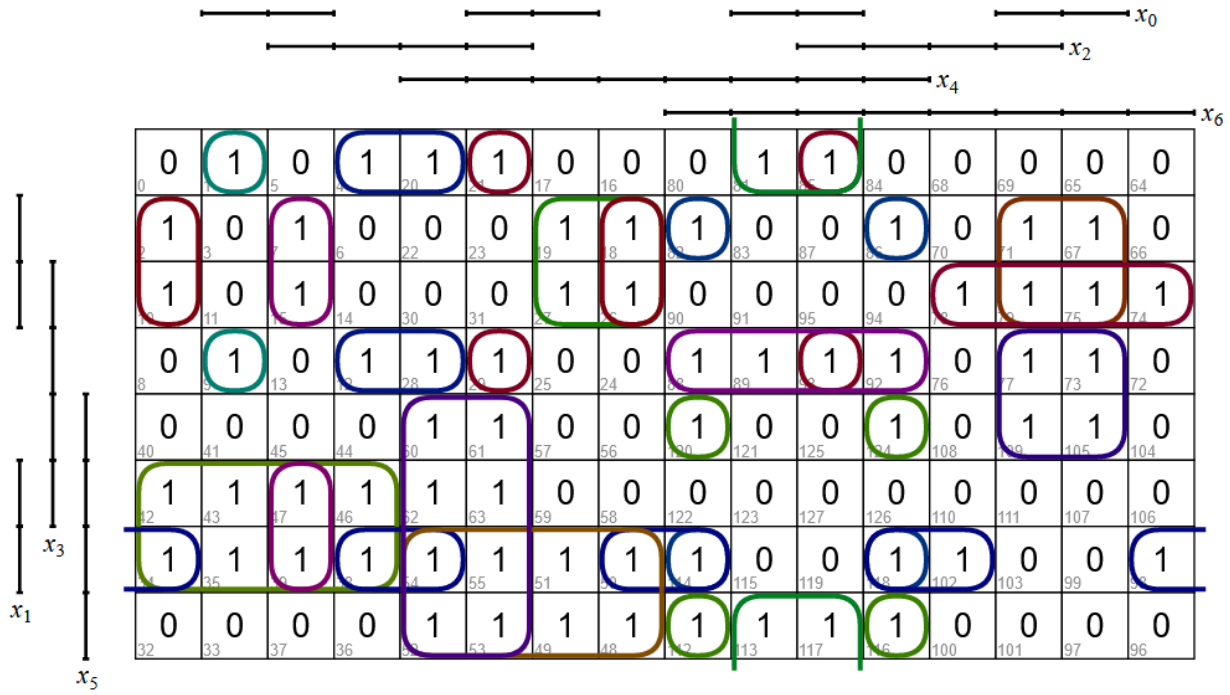
0	0	0	1	0	1	1	0	0
0	0	0	1	1	0	0	1	1
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	1	0	1
0	0	1	0	1	0	0	0	1
0	0	1	0	1	0	1	0	1
0	0	1	0	1	1	0	1	1
0	0	1	0	1	1	1	1	1
0	0	1	1	0	0	0	0	0
0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	0	0	1
0	0	1	1	0	1	1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

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

Em seguida, foi montado o Mapa de Karnaugh para X utilizando uma ferramenta presente no site da Universidade de Marburg:

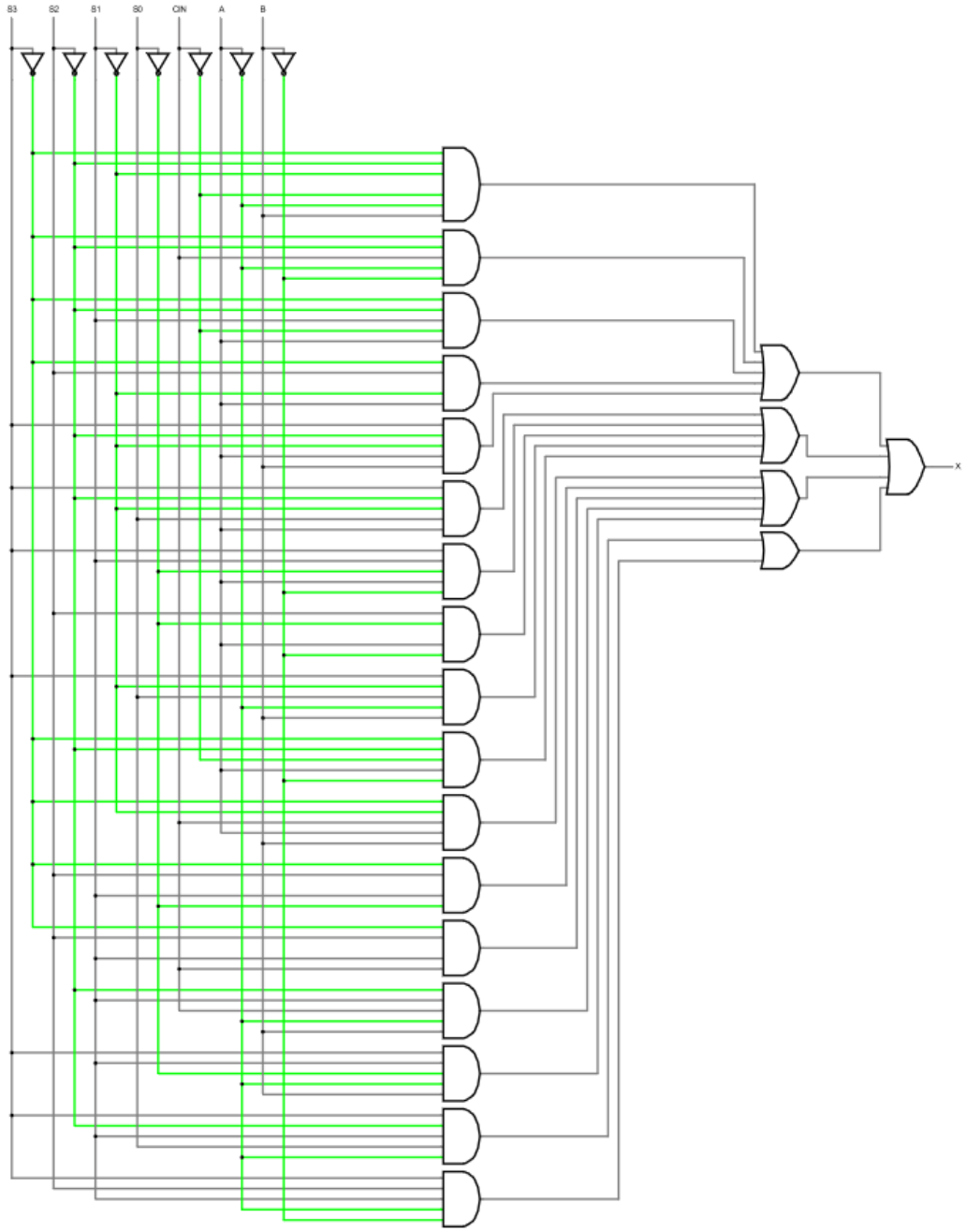
X = {1, 2, 4, 7, 9, 10, 12, 15, 18, 19, 20, 21, 26, 27, 28, 29, 34, 35, 38, 39, 42, 43, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 67, 71, 73, 74, 75, 77, 78, 79, 81, 82, 85, 86, 88, 89, 92, 93, 98, 102, 105, 109, 112, 113, 114, 116, 117, 118, 120, 124}



$y = (\bar{x}_6\bar{x}_5\bar{x}_4\bar{x}_2\bar{x}_1x_0) \vee (\bar{x}_6\bar{x}_5x_2\bar{x}_1\bar{x}_0) \vee (\bar{x}_6\bar{x}_5x_4\bar{x}_2x_1) \vee (\bar{x}_6x_5\bar{x}_4x_1) \vee (x_6\bar{x}_5\bar{x}_4x_1x_0) \vee (x_6\bar{x}_5\bar{x}_4x_3x_1) \vee (x_6x_4\bar{x}_3x_1\bar{x}_0) \vee (x_5\bar{x}_3x_1\bar{x}_0) \vee (x_6\bar{x}_4x_3\bar{x}_1x_0) \vee (\bar{x}_6\bar{x}_5\bar{x}_2x_1\bar{x}_0) \vee (\bar{x}_6\bar{x}_4x_2x_1x_0) \vee (\bar{x}_6x_5x_4\bar{x}_3) \vee (\bar{x}_6x_5x_4x_2) \vee (\bar{x}_5x_4x_2\bar{x}_1x_0) \vee (x_6x_4\bar{x}_3\bar{x}_1x_0) \vee (x_6\bar{x}_5x_4x_3\bar{x}_1) \vee (x_6x_5x_4\bar{x}_1\bar{x}_0)$

Onde (x6, x5, x4, x3, x2, x1, x0) = (s3, s2, s1, s0, cin, a, b)

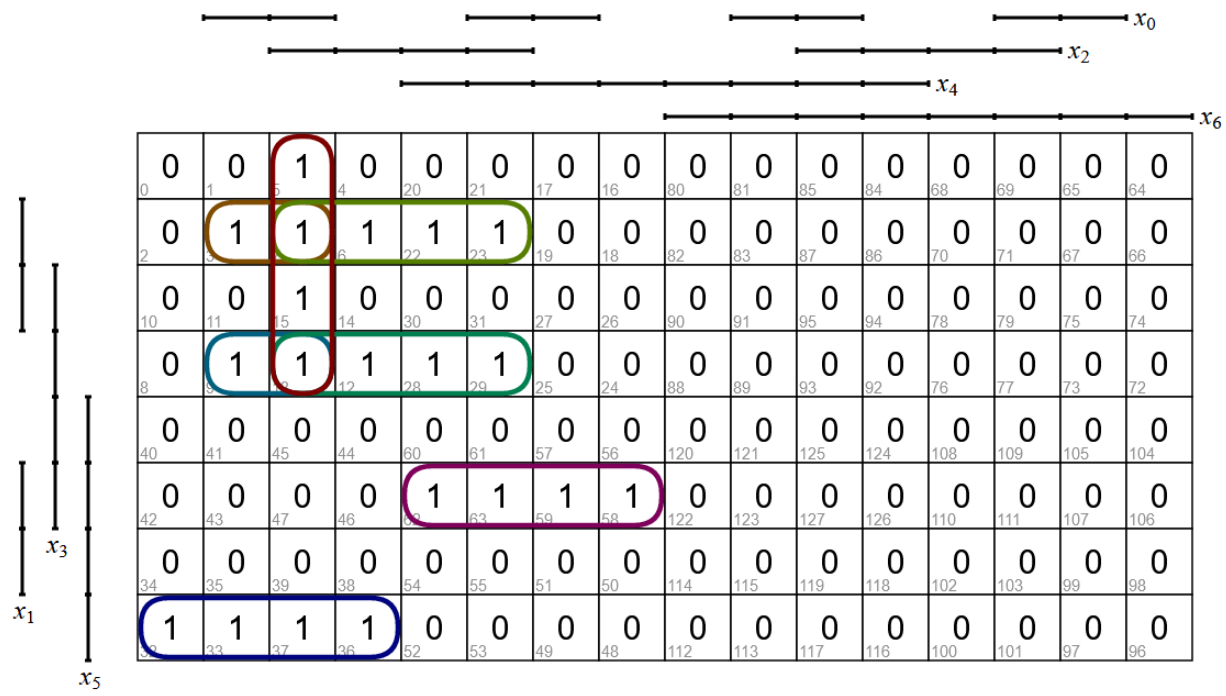
Com a expressão lógica em mãos, foi possível montar o circuito do resultado no circuit.js:



https://tinyurl.com/y3lm8toj

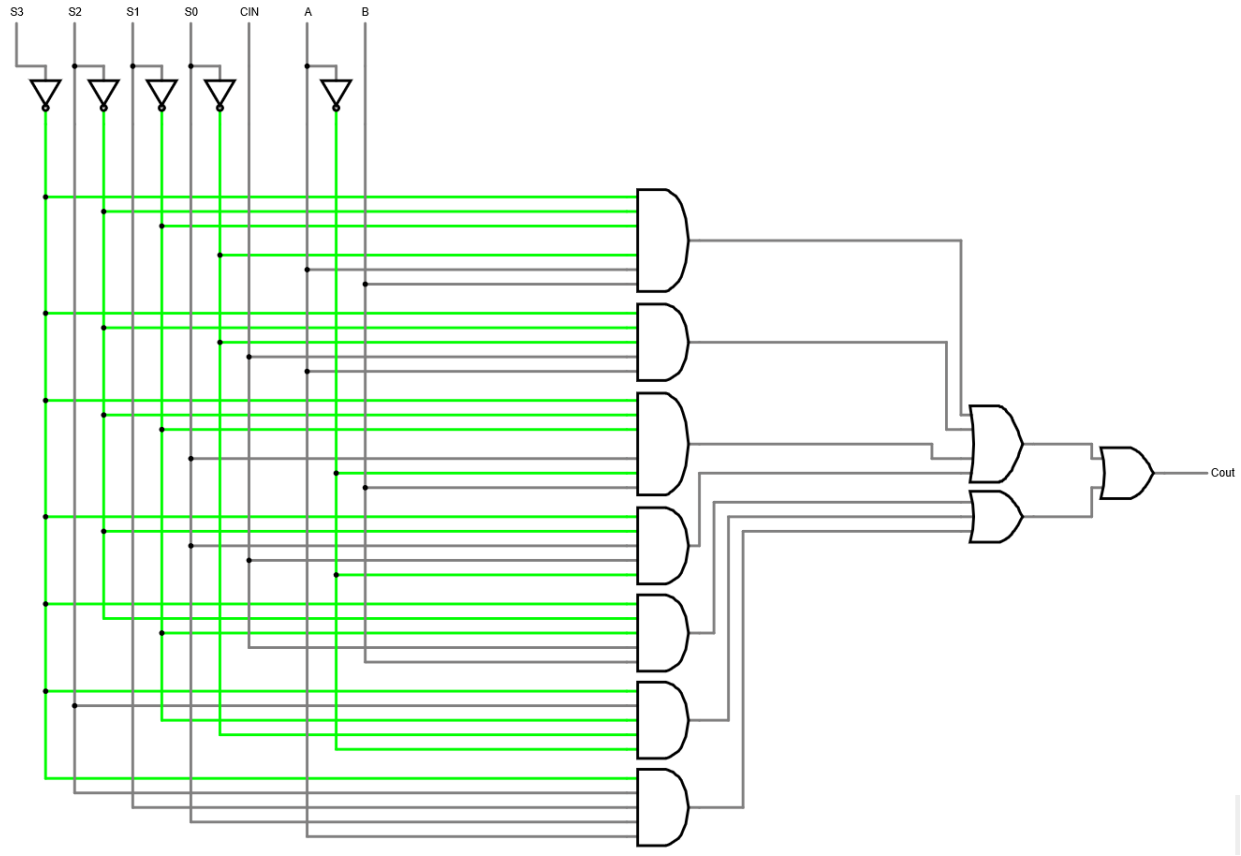
Além disso, também foi montado o Mapa de Karnaugh para o carry:

X = {3, 5, 6, 7, 9, 12, 13, 15, 22, 23, 28, 29, 32, 33, 36, 37, 58, 59, 62, 63}



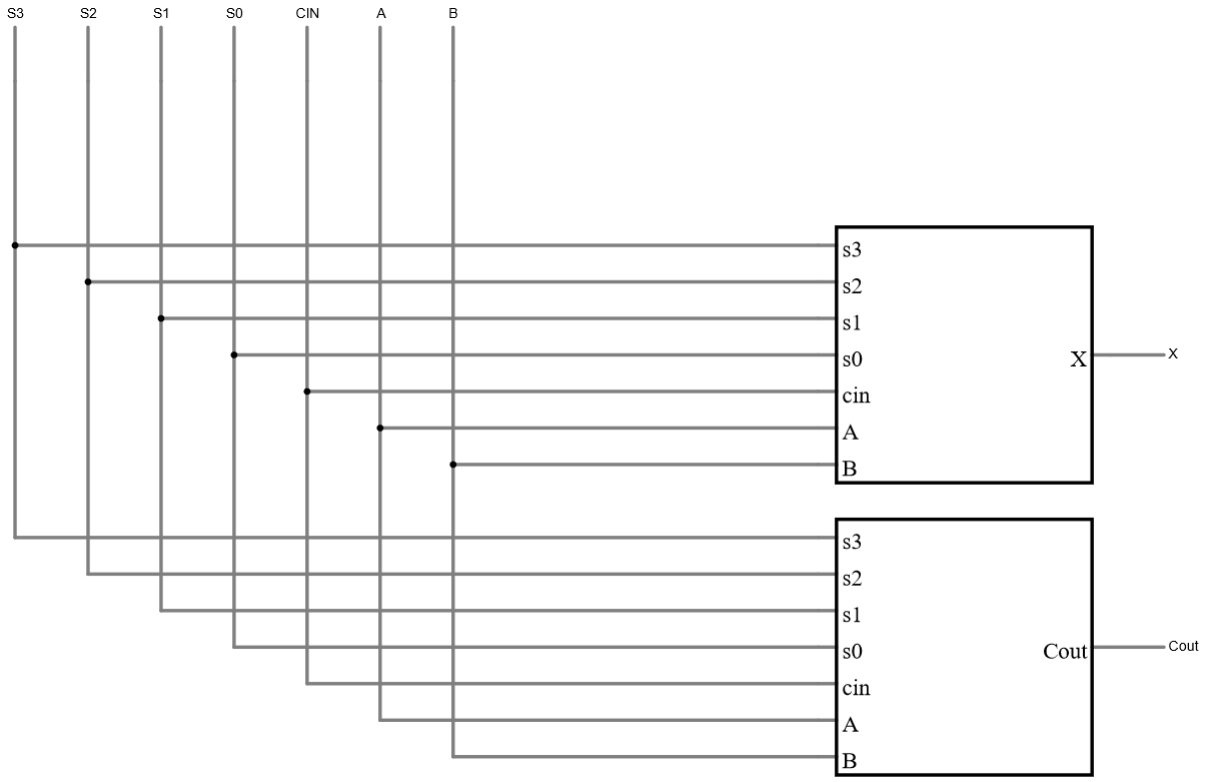
$y = (\bar{x}_6\bar{x}_5\bar{x}_4\bar{x}_3x_1x_0) \vee (\bar{x}_6\bar{x}_5\bar{x}_3x_2x_1) \vee (\bar{x}_6\bar{x}_5\bar{x}_4x_3\bar{x}_1x_0) \vee (\bar{x}_6\bar{x}_5x_3x_2\bar{x}_1) \vee (\bar{x}_6\bar{x}_5\bar{x}_4x_2x_0) \vee (\bar{x}_6x_5\bar{x}_4\bar{x}_3\bar{x}_1) \vee (\bar{x}_6x_5x_4x_3x_1)$

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



https://tinyurl.com/yy4ty8tm

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

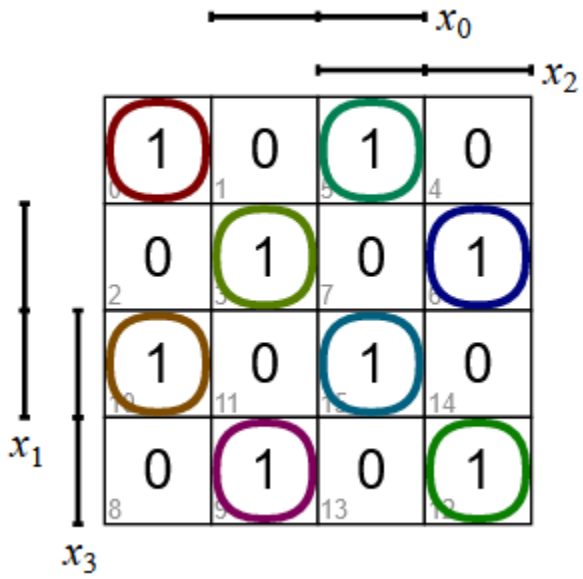


https://tinyurl.com/y5yr9wkx

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

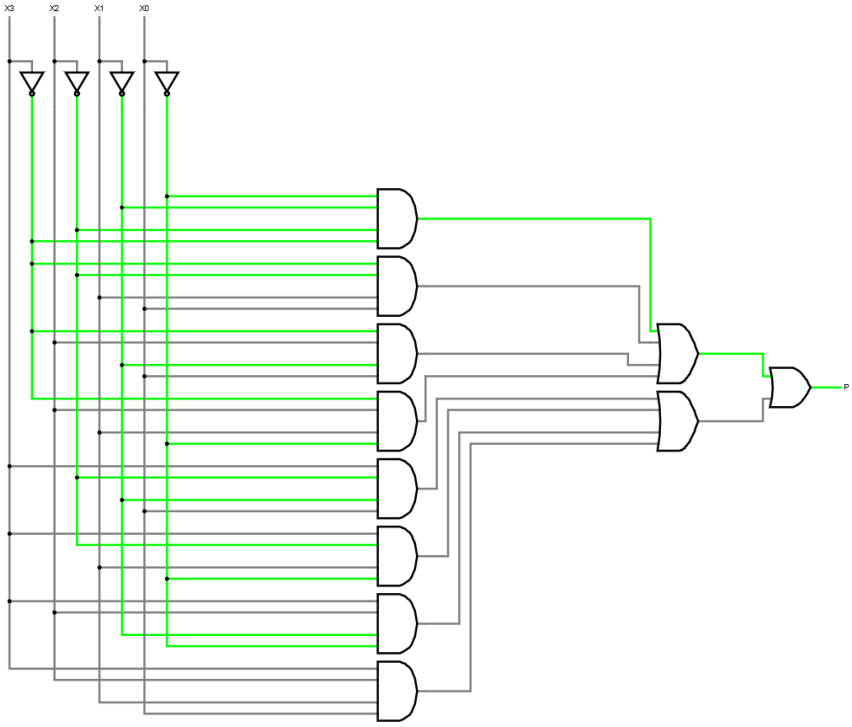
X3	X2	X1	X0	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Mapa de Karnaugh de P:



P = (!x3 !x2 !x1 !x0) ∨ (!x3 !x2 x1 x0) ∨ (!x3 x2 !x1 x0) ∨ (!x3 x2 x1 !x0) ∨ (x3 !x2 !x1 x0) ∨ (x3 !x2 x1 !x0) ∨ (x3 x2 !x1 !x0) ∨ (x3 x2 x1 x0)

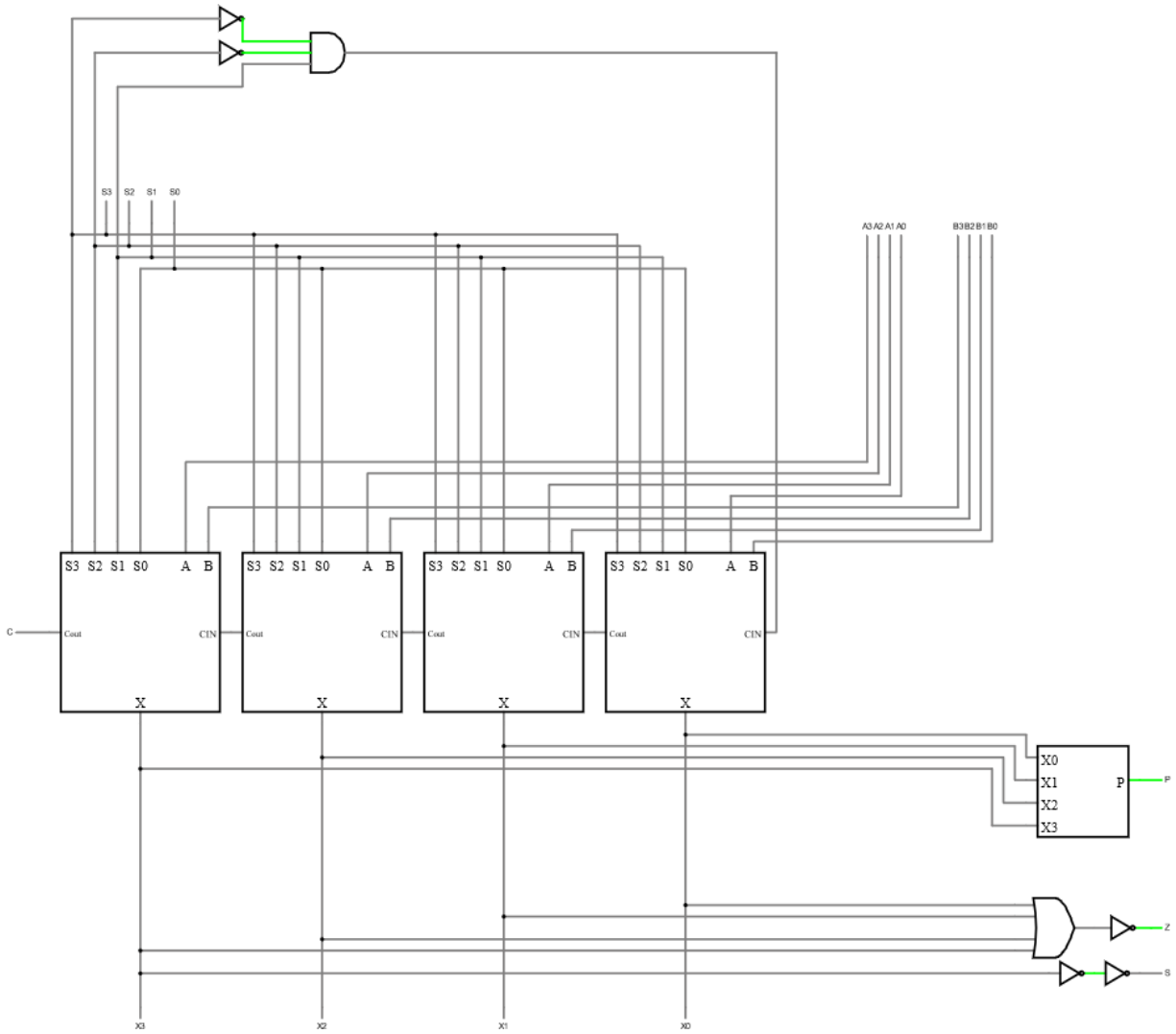
Circuito:



https://tinyurl.com/yyk55zhs

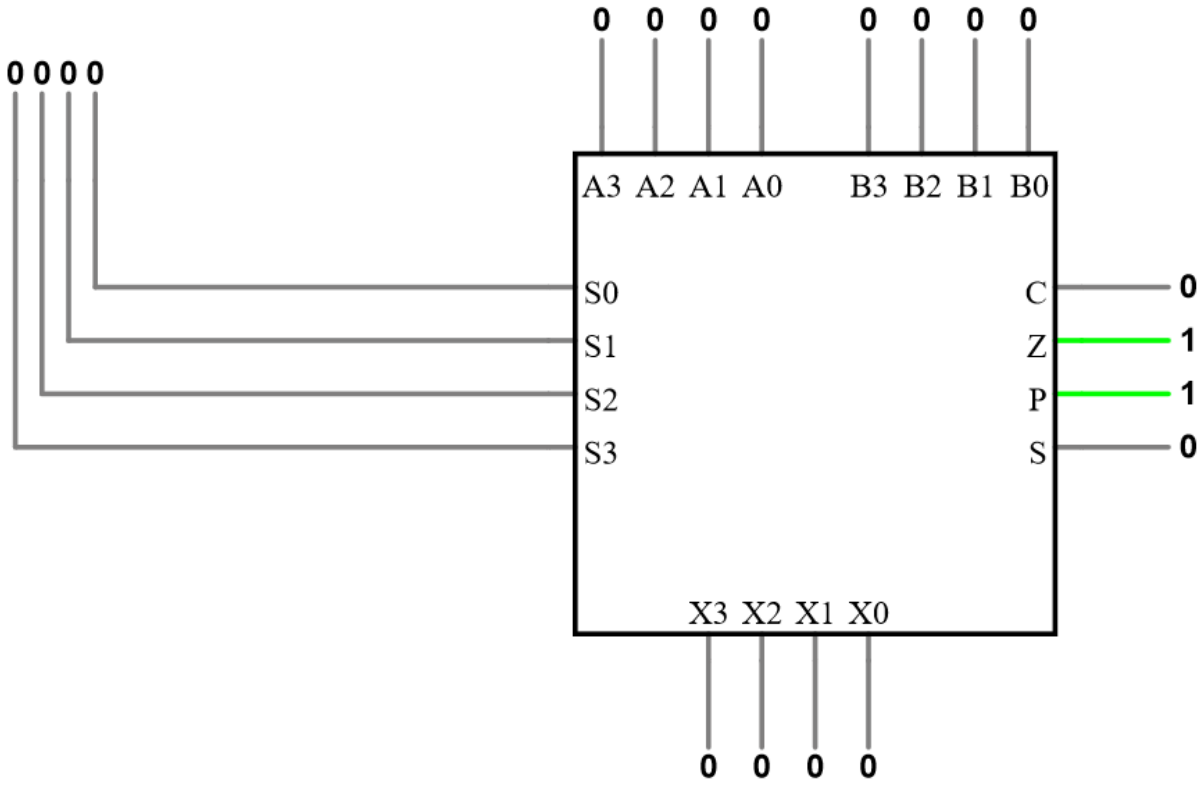
Para o flag Z, foi utilizada apenas a expressão !(A + B + C + D).

Como o flag S é simplesmente o dígito mais significativo do resultado, podemos juntar todos os componentes e finalizar a ULA de 4 bits:



https://tinyurl.com/yx8vxa76

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



https://tinyurl.com/y6nwf2lr

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

REFERÊNCIAS

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