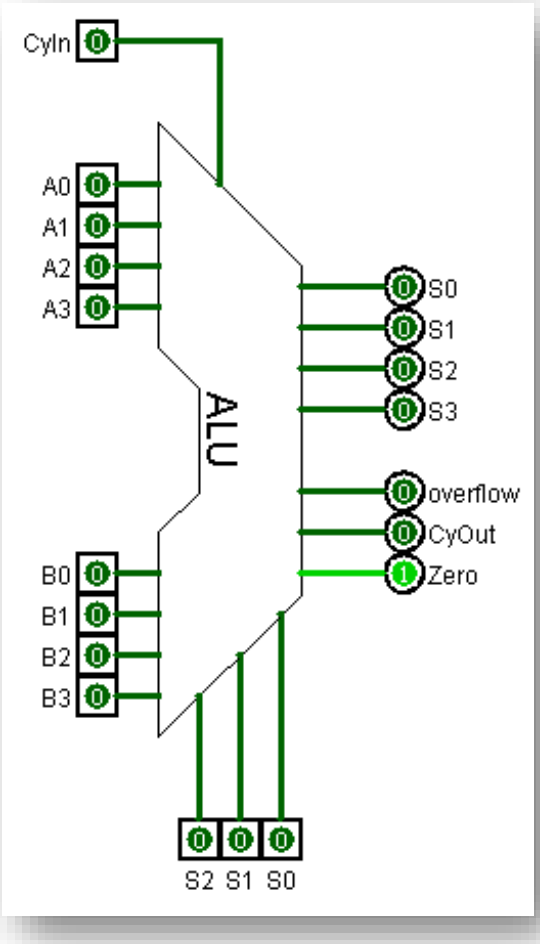
Lógica e Sistemas Digitais

Realizado pelo grupo ?:
Paulo Rosa – 44873



Docente: Prof. José Antão

Introdução:

28/10/18

2.º Trabalho Prático

A generalidade das linguagens de programação disponibiliza os seguintes operadores básicos:

- **Aritméticos** +, -, *, /, %;
- **Lógicos** &, |, ^, !, ~;
- **Deslocamento** (*Shift*) >>, <<;
- **Relacionais** = =, >, <, >=, <=.

Para suportar a realização destas operações, a Unidade Central de Processamento (CPU) dos nossos computadores inclui na sua arquitectura uma unidade funcional denominada por *Arithmetic Logic Unit* (ALU).

Objectivo:

Projectar e realizar uma ALU, segundo o diagrama da Figura 1, com as seguintes características:

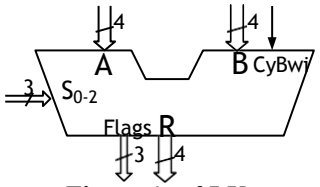


Figura 1 – ALU.

- Executa uma de seis operações (ADD, ADC, SBB, ANL, ASR, RCL), seleccionadas pelos três bits S_2, S_1 e S_0, conforme indicado na Tabela 1 (a codificação poderá ser modificada se for considerada conveniente);

S_{0-2}	Operação		Sigla	Z	CyBwo	OV
000	Adição	R = A + B	ADD	•	•	•
001	Subtracção com *Borrow*	R = A – B – CyBwi	SBB	•	•	•
010	Adição com *Carry*	R = A + B + CyBwi	ADC	•	•	•
100	Lógico AND bit a bit	R = A&B	ANL	•	–	–
110	*Arithmetic Shift Right*	R = A>>B0-1	ASR	•	•	–
111	*Rotate Carry Left*	R = A<<1	RCL	•	•	•

Tabela 1 – Funcionalidade da ALU.

Legenda: • : indicador binário com significado na operação, **- :** indicador binário sem significado na operação.

- Tem por entradas dois operandos A e B, de quatro bits cada, e um operando de um bit, CyBwi (*Carry/Borrow in*), a ser considerado nas operações de adição, subtracção e deslocamento;
- O resultado R é expresso em quatro bits, no mesmo domínio que os operandos;
- A ALU implementa três indicadores binários (*flags*) Z, CyBwo e Ov, sendo um deles qualitativo e os outros de excesso de domínio. Os três indicadores têm o seguinte significado:

Z	*Zero*	Activo, quando a operação realizada tem como resultado o valor zero.
CyBwo	*Carry/Borrow out*	Representa *Carry* na operação de adição e *Borrow* na operação de subtracção, estando activo quando o resultado excede o domínio, entendida a operação em código natural. Na operação A*SR* recebe o último bit deslocado e na operação RCL recebe o bit de maior peso de A.
Ov	*Overflow*	Activo, quando o resultado da adição ou da subtracção excede o domínio, entendida a operação em código dos complementos. Nas operação RCL quando o bit de sinal de R difere do bit de sinal de A.

Realização:

A ALU a desenvolver deve ser constituída por dois módulos, interligados conforme o diagrama de blocos apresentado na Figura 2. O módulo Aritmético realiza as operações ADD, ADC e SBB, enquanto o módulo Lógico/Shift realiza as operações ANL, ASR e RCL.

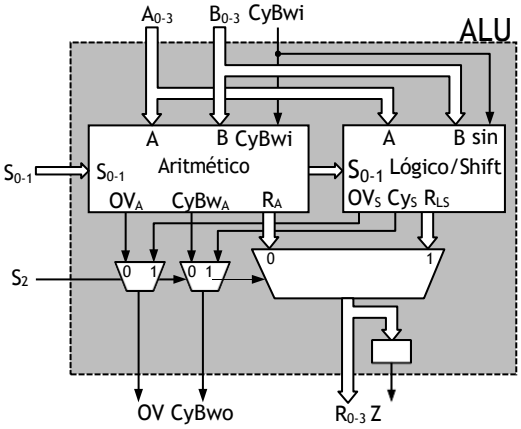


Figura 2 – Diagrama de blocos da ALU.

A arquitectura interna dos dois módulos obedece aos diagramas de blocos apresentados na Figura 3.

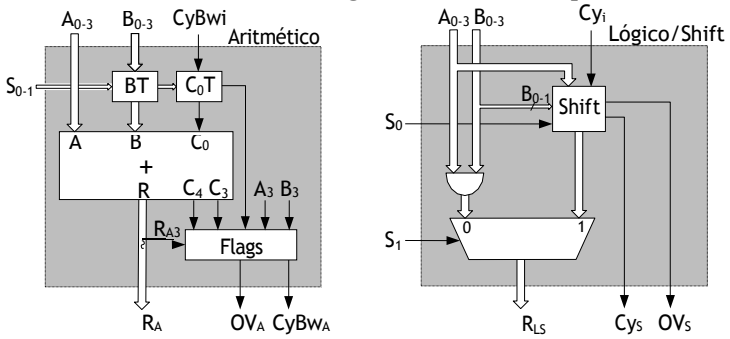


Figura 3 – Diagramas de blocos dos módulos Aritmético e Lógico/Shift.

Módulo Aritmético:

- O módulo aritmético poderia ser realizado por multiplexagem dos operadores adição e subtracção. A arquitectura preconizada implementa uma técnica denominada por *contraction* (contracção) que consiste em reutilizar um dado elemento funcional para a realização de várias funções, obtendo-se uma estrutura mais simples, por adaptação das respectivas entradas e saídas;
- Neste módulo, o elemento central é o somador completo de quatro bits;
- Os elementos BT e C_0T realizam a transformação dos operandos B e CyBwi para que, utilizando o elemento somador, se realizem as operações de adição e subtracção;
- O elemento Flags, implementa os sinais binários OV_A e $CyBw_A$. A sua implementação poderá recorrer a outros sinais disponíveis na estrutura, caso o aluno os considere preferíveis.

Módulo Lógico:

- A arquitectura preconizada recorre à multiplexagem do resultado dos vários operadores, sendo estes realizados por módulos independentes entre si.
- As operações de deslocamento (ASR e RCL) são realizadas por um *Barrel Shifter*. Na operação ASR o número de bits a deslocar é determinado pelos bits B_{0-1}. Nesta operação, o bit Cy_S recebe o último bit deslocado. Na operação RCL, o bit Cys recebe o bit A_3 e A0 recebe o bit CyBwIn

Introdução

O objetivo deste trabalho é desenhar e construir, em hardware, uma ALU com as características(e operações) dadas no enunciado. Lido o enunciado, pode-se então prosseguir para a sua interpretação.

O circuito contém 12 entradas:

Os 4 bits de A
Os 4 bits de B
3 bits S para definir as 6 operações
E 1 bit CyBwIn

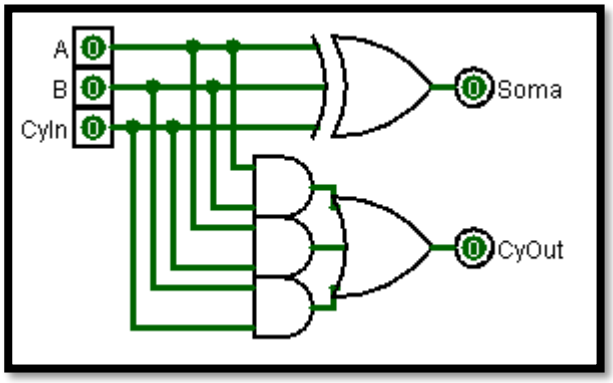
E 7 saídas:

Os 4 bits de R
O bit de Overflow(overflow dos inteiros com sinal, conjunto Z)
O bit de CyBwOut(overflow dos inteiros sem sinal, conjunto N)
E o bit de Zero, que indica se R = 0;

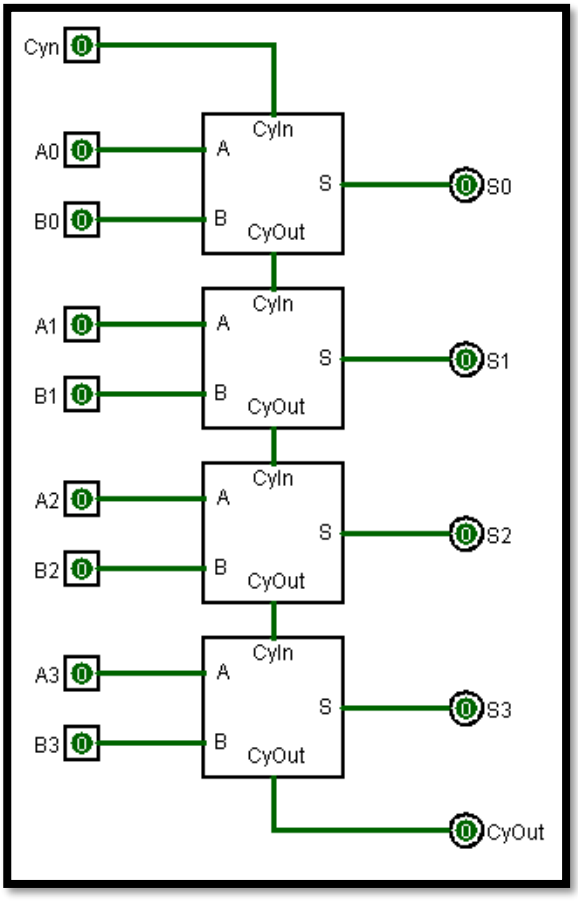
Resolução do Módulo Aritmético

Soma e somador

Em primeiro lugar, fez-se o bloco aritmético. Começando por uma soma singular e depois juntando várias somas que formam o somatório. A soma(R's) está a 1 quando há um número ímpar de inputs a 1 na porta XOR. Quando há pelo menos 2 inputs a 1 neste bloco, então há CarryOut que entra para o próximo bloco de soma como o input CarryIn. Nota que se deve definir a porta XOR como "enable when input is odd number" para esta solução resultar no logisim.



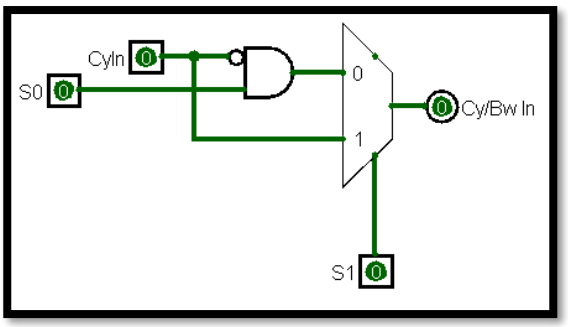
Como estamos queremos resultados a 4 bits, então precisamos de 4 blocos destes.



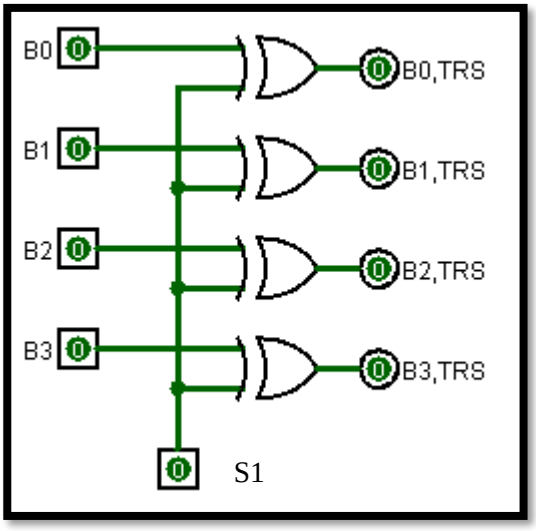
CoT e BT

Depois de construído o somador, podemos definir outros blocos que permitem alterar o comportamento dos A's, B's e CyIn consoante os valores de S0 e S1 para se realizarem as operações que se querem segundo o enunciado.

Um desses é o CoT, que modifica o CyIn antes de entrar para o somador. Quando fazemos a operação (0,0), S1 e S0, respetivamente no bloco aritmético, realizamos A +B em que CyIn não está presente, por isso não influencia a entrada para o somador. Depois, quando fazemos a operação (0,1), realizamos A – B – CyIn, logo, queremos que CyIn se inverta. Finalmente quando fazemos a operação (1,0) realizamos A + B + CyIn, logo queremos com que CyIn esteja presente. Deste modo, usamos um MUX 2x1 e uma porta AND com uma entrada negada para termos como CoT o seguinte circuito:



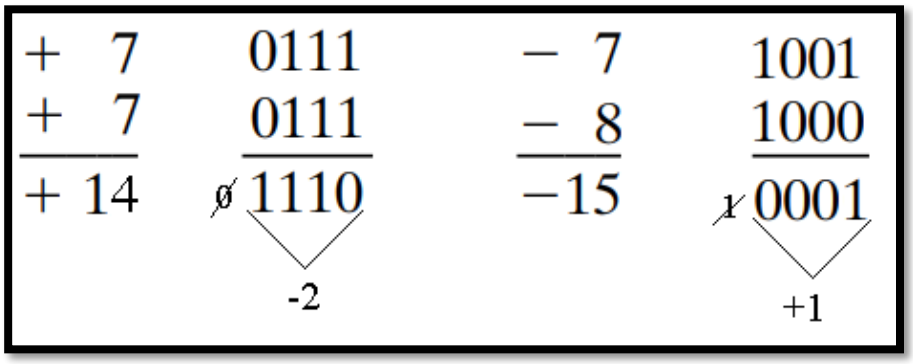
Depois, para podermos ter -B na operação (0,1), temos o bloco BT em que os B's entram e são invertidos com uma porta XOR quando S0 = 1.



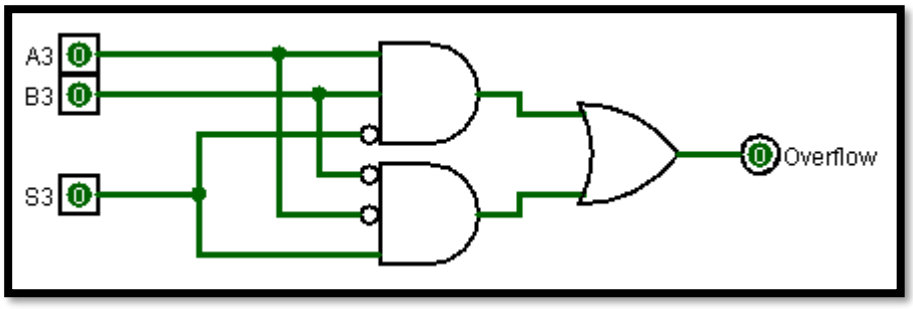
Overflow e CyBwOut

Quando se fala de overflow, fala-se no contexto dos números com sinal e ocorre quando já não se pode representar um resultado de uma conta com o número de bits que temos para representar esse resultado. No nosso caso, temos 4 bits, logo podemos representar números decimais entre -8 e +7. Caso exceda este intervalo, ocorre overflow.

E a lógica e o seu funcionamento é o seguinte: ao somarmos 2 números (A e B) com sinais diferentes não permite overflow porque a magnitude do resultado vai ser sempre igual ou menor a um dos números. Mas quando os números têm o mesmo sinal, é possível que haja overflow, porque eles podem-se combinar para exceder o número de bits que temos. Logo, quando somamos 2 números negativos estamos à espera de obter um resultado negativo e quando somamos 2 números positivos estamos à espera de obter um resultado positivo. Bem, quando ocorre overflow, isto não acontece, porque já esgotamos os bits que temos e "perdemos" o bit sinal. O dois casos de overflow como exemplo:



Logo, temos overflow quando os bits sinal de A e B forem diferentes do bit sinal de R.



E finalmente, a flag CyOut é o carry do somador quando fazemos A+B, mas quando fazemos A-B, o nosso BwOut é o simétrico do CyOut, deste modo, quando estamos a fazer subtração, S0 = 1, por isso ligamos este sinal e o CyOut(C4) do somador a um XOR

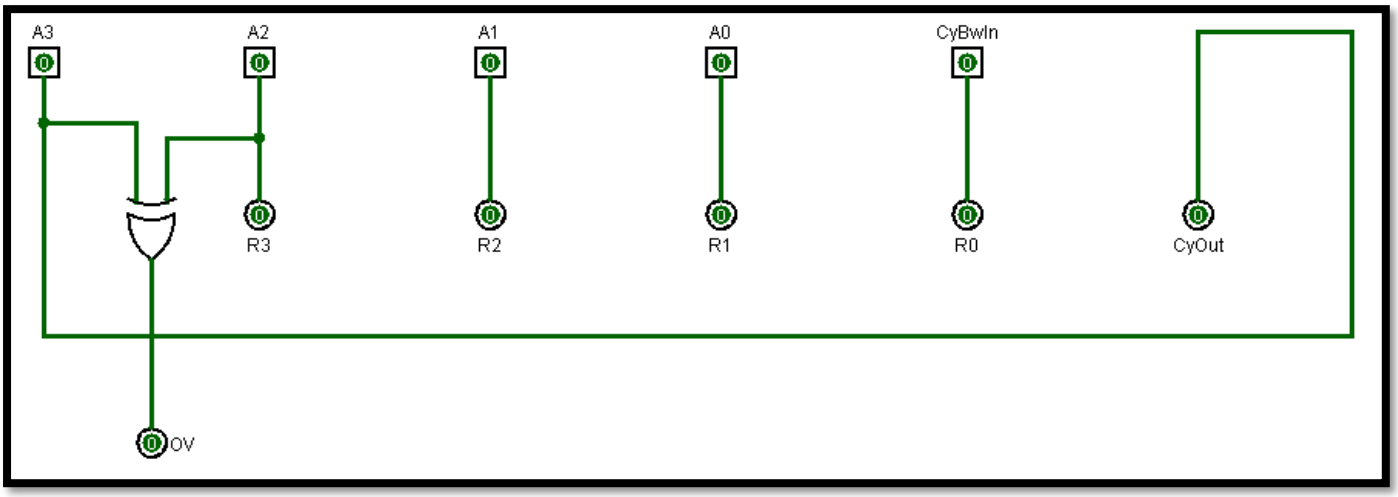
para produzir este efeito.

Resolução do Módulo Lógico

Em segundo lugar, temos o bloco lógico. Em que temos 2 tipos de operações com 4 bits cada e que devem ser contruídas separadamente, ao contrário que no módulo aritmético, o somador tratava de todas as somas, só que só alterávamos alguns aspetos antes de entrar neste. Para tratar disto, foi feito um multiplexer 2x4. Um que recebe as operações do tipo shift e o outro que recebe a do tipo AND(estas são muito diretas já que basta ligar os A's e B's com os bits peso a peso de cada numa porta AND).

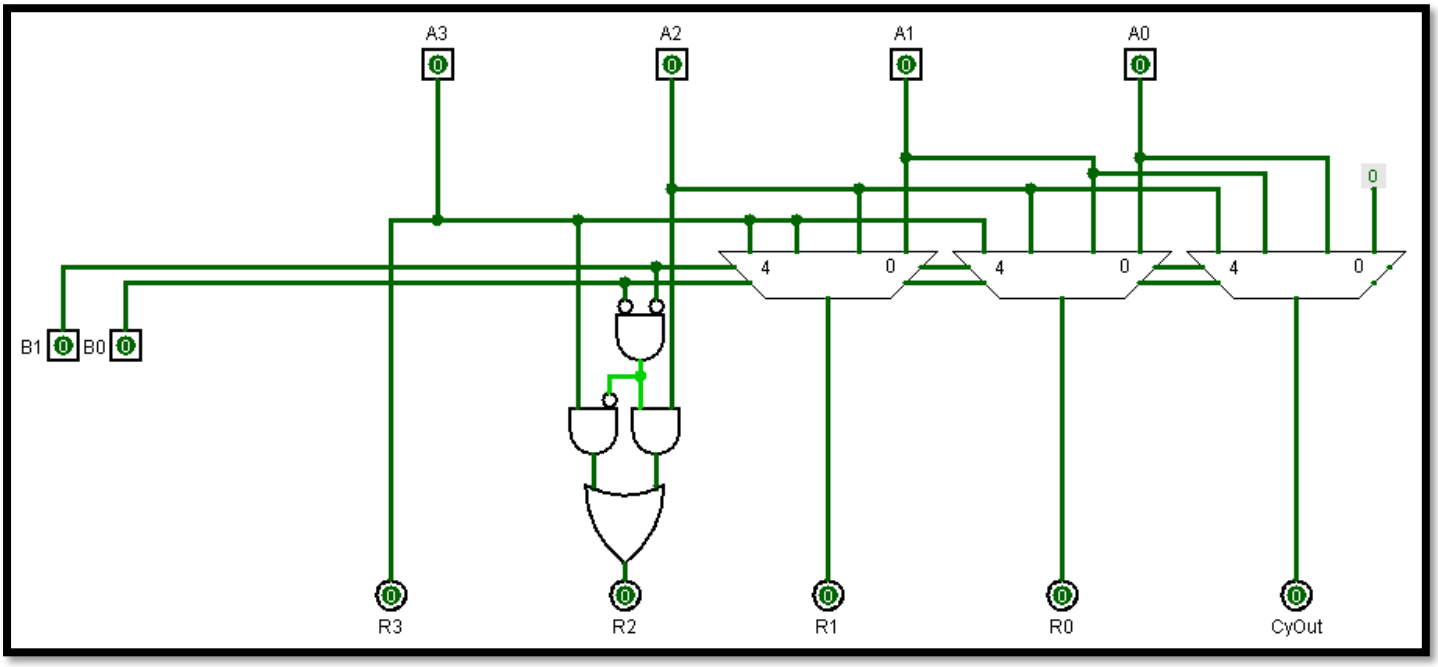
Rotate Carry Left

Este tipo de shift, faz shift de um para a esquerda, sendo que o bit de maior peso de A passa a ser o CyOut e o CyIn o bit de menor peso de A. Esta operação equivale a multiplicar A por 2. E isto implica que pode haver overflow nesta operação. E o caso em que isso acorre é quando o bit sinal original (A3) difere do bit sinal do resultado(R3), então é porque ocorreu overflow.



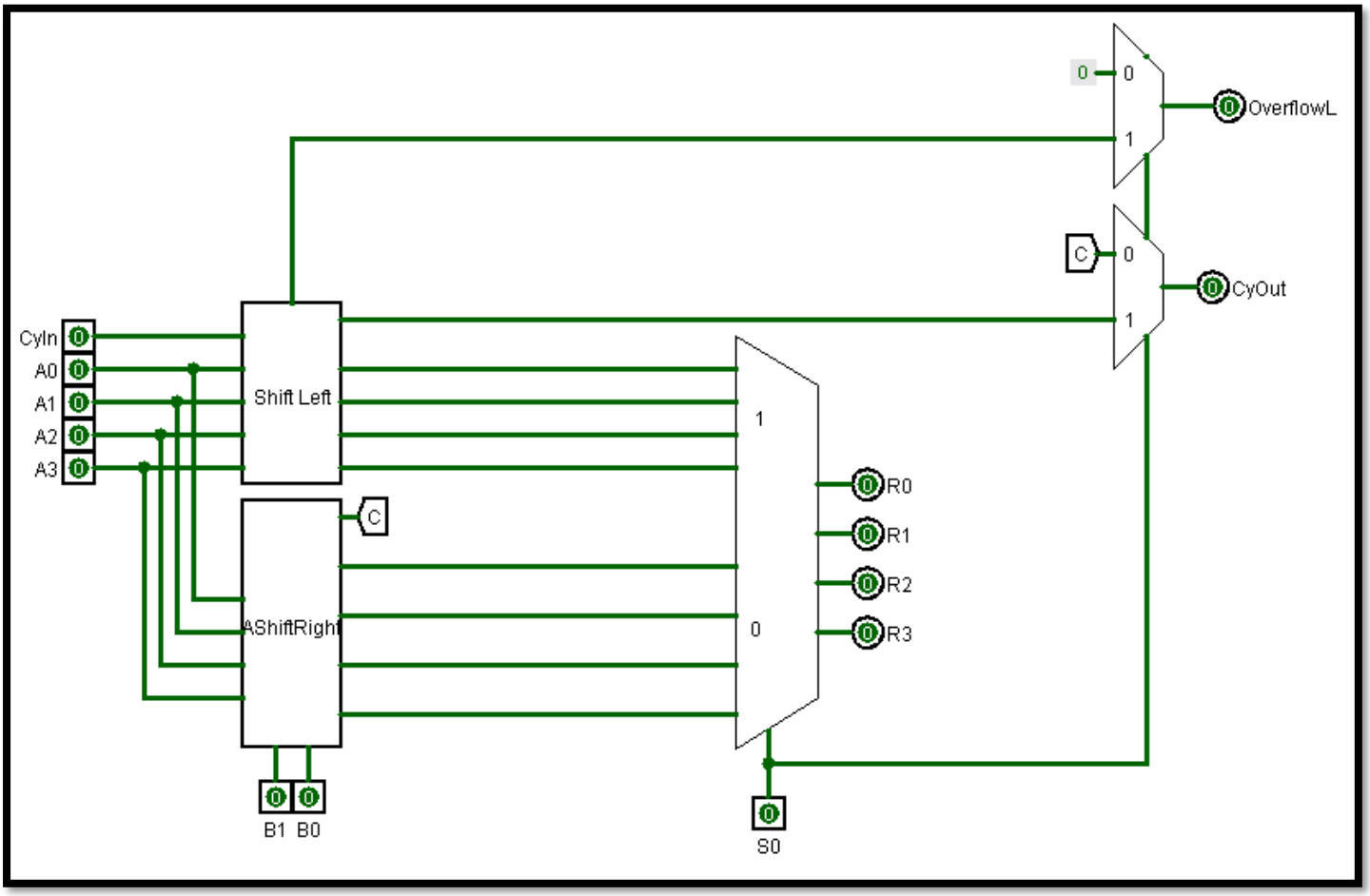
Arithmetic Shift Right

O arithmetic shift right faz shift para a direita segundo os valores de B1 e B0. Ou seja, com esta operação, pode-se dividir por 2 até 3 vezes. E tem como base de funcionamento a utilização de vários multiplexers 4X1. Ao se fazer este shift, pretende-se que o bit sinal nunca mude, por isso, seja qual for o valor de S0 e S1, o bit sinal do resultado deste operção será sempre igual. Neste bloco, decidimos apenas simplificar o R2 por não precisarmos de tantos recursos quando os de um mux 4X1 e termos um melhor aspeto visual deste.



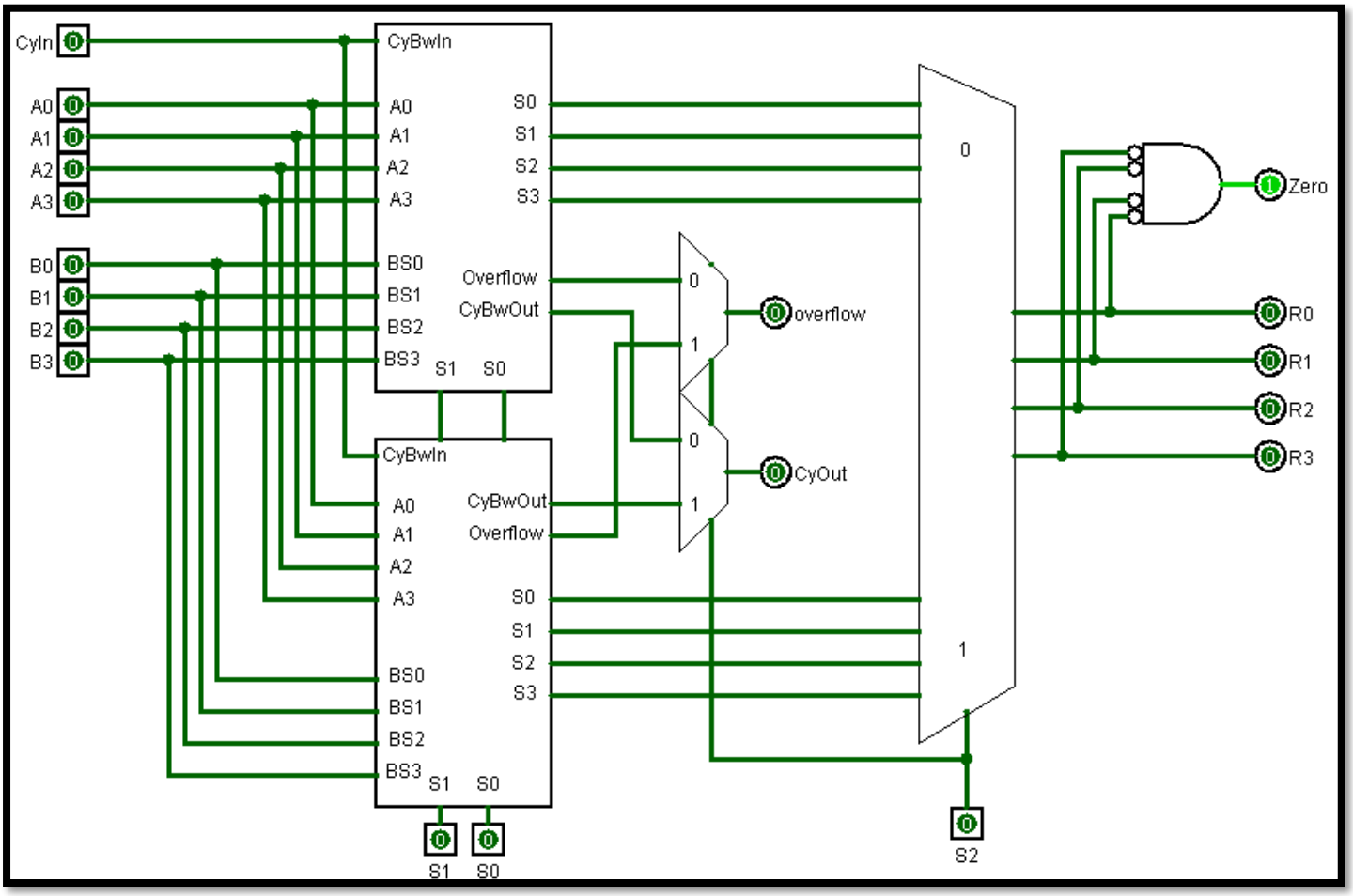
Junção do RCL e ASR num bloco Shift

Depois, é feita a junção do Shift Left com o Arithmethic Shift Right com um Mux 2x4 em que o S0 seleciona qual das duas se deve fazer. Caso se deseja fazer shift left, o overflow da porta lógica é 0



Junção da componente Aritmética com a Lógica

E finalmente temos o circuito final e com um AND de 4 entradas negadas para ligadas a cada R para termos o sinal Zero. E o objeto resultado é igual à o da imagem na capa deste relatório.



Descrição em CUPL

```
/* *************** INPUT PINS *********************/
PIN [1..4] = [A3..0];
PIN [5..8] = [B3..0];
PIN 9 = S0;
PIN 10 = S1;
PIN 11 = S2;
PIN 12 = C0;
PIN 13 = CyBwIn;

/* *************** OUTPUT PINS *****************

PIN 14 = CyBwout;
PIN 15 = Z; /*zero*/
PIN 16 = OverFlow;
PIN [17..20] = [R3..0];
PIN [21..23] = [M0..2];
/* Aritmetico */
C0 = (!CyBwIn&S0) & !S1 # CyBwIn & S1;

[BT0..3] = [B0..3]$S0;

[C1..4] = [A0..3]&[BT0..3] # [A0..3]&[C0..3] # [BT0..3]&[C0..3];

[RA0..3] = [C0..C3] $ [A0..3] $ [BT0..3];

OvA = A3&BT3&!R3 # !A3&!BT3&R3;
CyA = C4 $ S0;

/*Logico*/
/*Shift right*/
CyOutRight = !B1&!B0 & 'b'0 # !B1&B0 &A0 # B1&!B0 &A1 # B1&B0 &A2;

M0 = !B1&!B0 & A0 # !B1&B0 &A1 # B1&!B0 &A2 # B1&B0 &A3;
M1 = !B1&!B0 & A1 # !B1&B0 &A2 # B1&!B0 &A3 # B1&B0 &A3;
M2 = !B1&!B0 & A2 # !B1&B0 &A3 # B1&!B0 &A3 # B1&B0 &A3;
M3 = A3;
```

```
/*Shift left*/
CyOutLeft = A3;

/*L0 = CyBw;
L1 = A0;
L2 = A1;
L3 = A2;*/

[SHIFT0..3] = [CyBwIn,A0,A1,A2]&S0 # [M0..3]&!S0;

[AND0..3] = [A0..3]&[B0..3];

/*AND mais Shifts*/
[RL0..3] = [AND0..3]&!S1  # [SHIFT0..3]&S1;
CyL = CyOutLeft&S0 # CyOutRight&!S0;
OvL = A3 $ A2;

/*LOGICO MAIS ARITMETICO*/
[R0..3] = [RA0..3]&!S2 # [RL0..3]&S2;
OverFlow = OvA&!S2 #  OvL&S2;
CyBwout = CyA&!S2 # CyL&S2;
```

Conclusão

Em suma, todos os objetivos foram realizados. Foram feitas provas e testes para chegar à resolução final, que mostra ser coerente com a lógica do problema segundo o enunciado do trabalho. Foi apreciada e foi interessante a realização deste trabalho, porque representa um circuito que se pode aplicar no mundo real e que, neste caso, se aplica ao funcionamento de um cofre e foi planeado usando os conhecimentos desta cadeira de lógica e sistemas digitais.

FIM