ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

LEIM -Licenciatura Engenharia informática e multimédia

Computação Física

Trabalho 1

Circuitos combinatórios e Circuitos Sequenciais

Turma: LEIM23D

Trabalho realizado por: Miguel Távora Nº45102
João Cunha Nº45412
Luís Farinha Nº45147

Docente: Carlos Carvalho

Data de entrega:5/4/2019

Índice

Índice de Tabelas

Índice de Figuras

1.Introdução/Objetivos

O trabalho prático foi feito com o intuito de aprender, melhorar e aplicar sobre tudo o que aprendemos sobre circuitos combinatórios e sequênciais. Neste sentido tinhamos dois números a 4 bits cada, através desses dois números teríamos de realizar quatro operações diferentes sendo elas a soma, subtração, ASR(*Arithmetic Shift Right*) e a operação lógica NOR. Para além dos dois números a 4 bits teríamos ainda um número a 2 bits que são os bits de seleção, onde cada operação terá uma combinação binária diferente e será a forma como irá ser selecionada a operação pretendida.

Para obter a expressão lógica simplificada das operações foram construídas tabelas de verdade e uma máquina de estados ASM(*Algorithmic State Machinet*) onde foi possível construir os mapas de Karnaugh.

No final, foi montado um circuito com LED's para ver a funcionalidade da ALU, com base nos módulos de controlo e funcionais fazendo uma simulação em software dos programas.

[1] "O módulo funcional é um diagrama de blocos constituído por todos os dispositivos hardware disponibilizado pelos fabricantes, tais como, multiplexers, demultiplexers, comparadores, codificadores, descodificadores, flip-flops, registos, contadores, memórias, etc. "

[1] "O módulo de controlo é um circuito combinatório ou uma máquina de estados que aciona os dispositivos existentes no módulo funcional. Para tal, o módulo de controlo tem como entradas, sinais vindos do módulo funcional, e como saídas, sinais que comandam os dispositivos constituintes do módulo funcional."

O principal objetivo deste trabalho, foi desenvolver expressões lógicas através das respetivas tabelas e mapas. Utilizar circuítos combinatórios e circuítos sequênciais de forma a otimizar e melhorar a sua funcionalidade, facultando ao utilizador uma experiência mais direta.

2.Modelo “Caixa preta” da ALU

A unidade lógica e aritmetica do inglês *Arithmetic Logic Unit* (ALU), é circuito digital que realiza operações lógicas apartir dos quais quando devidamente implementados é possível realizar operações artimeticas. A ALU está divida em diversos níveis de abstração sendo o maior nível de abstração o Modelo de “Caixa preta”. Na figura observa-se que a ALU tem como entrada os dois números binários A e B constituídos por 4 bits cada e o selector(C) de 2 bits. Através das combinações possíveis de C será possível efectuar operações que veremos mais à frente. Onde depois dessas operações será possível obter o Resultado(R) de 4 bits e tem três flags. Cada flag tem um significado diferente, o Carry/Borrow(Cy/Bw) são indicadores de erro nas operações sendo apenas analisados para números naturais(0,+15), a flag overflow(Ov) é também um indicador de erro mas apenas para números relativos e a flag zero (Z) indica se o número está todo a zeros ou não.

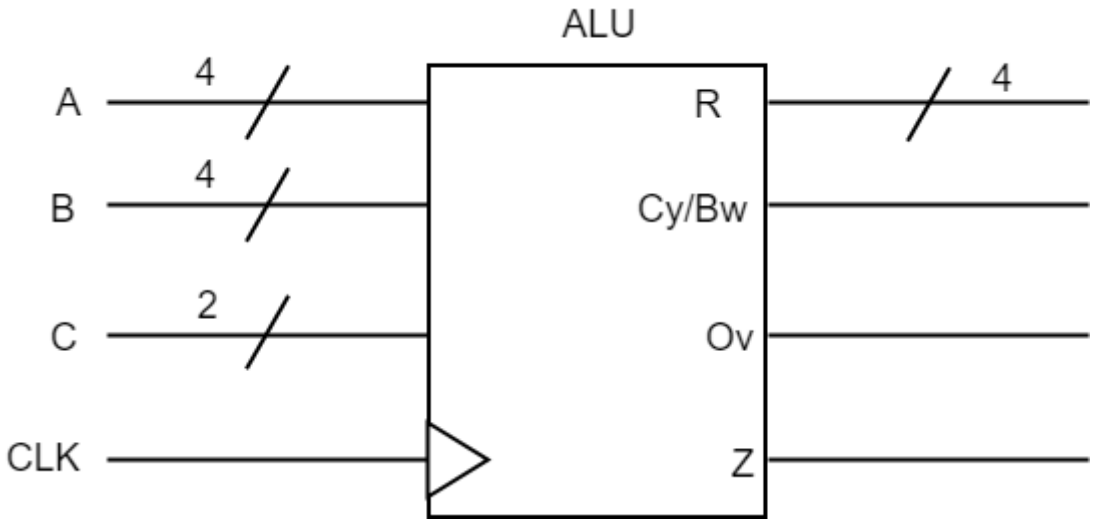


Figura 1 - Modelo "Caixa Preta" da ALU

3.Operações Aritméticas e Lógica

O trabalho foi divide em modulos em quatro módulos dos quais resultam :a soma, a subtração, o ARS e o NOR. Sendo a seleção feita por combinações dos valores lógicos de cada um dos bits do selector, conforme se observa na tabela de verdade:

Controlo		Operações Aritmeticas e Logica			
C_1	C_0	Soma	Subtração	Aritmetic Shift Right	NOR
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Tabela 1 - Tabela de verdade dos Operadores

De onde resulta o seguinte circuito combinatório:

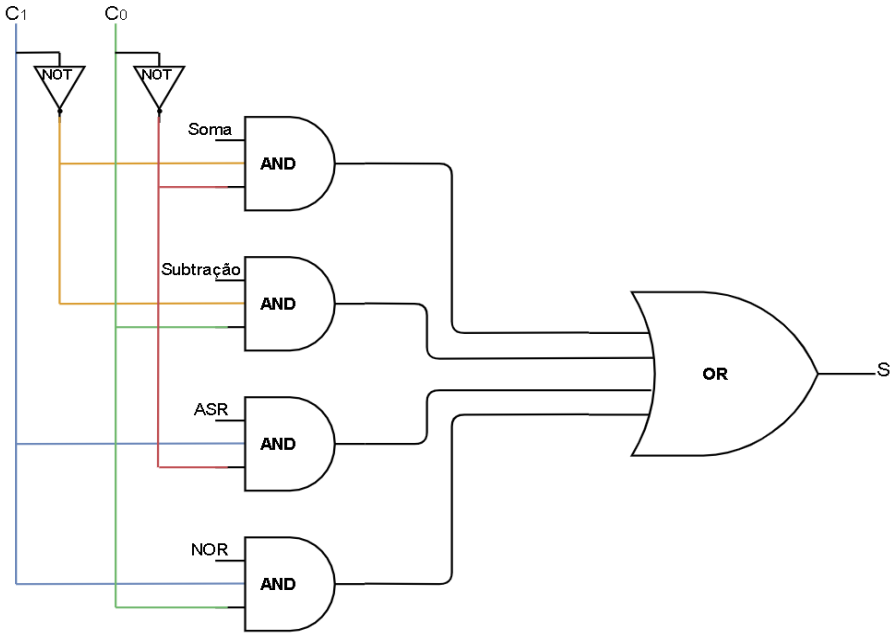


Figura 2 - Circuito combinatório do seletor

Estrutura do trabalho – entradas e saídas de cada bit

Representação esquematizada

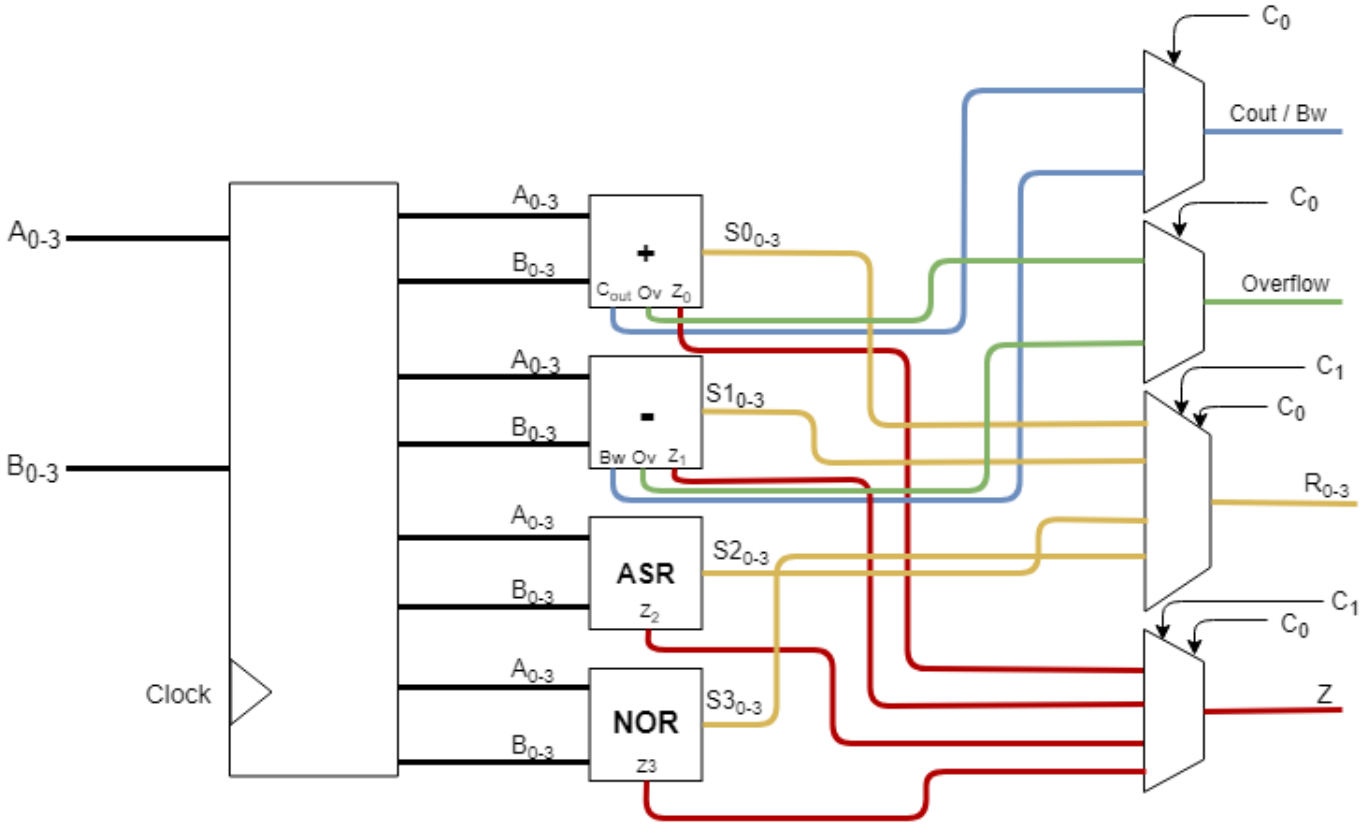


Figura 3 - Modulo esquematizado das operações

A partir do esquema representado é possível observar que entram dois números e são realizadas quatro operações nomeadamente a soma, a subtração, o Aritmetic Shitft Right (ASR) e o NOR. Para cada operação entra os 4 bits de cada número e são executadas todas as ações. Na saída temos o resultado de 4 bits e 3 flags, sendo que a flag do Carry/Borrow e de Overflow apenas são relevantes na operação de soma e de subtração. Como foi referido anteriormente a flag Z indica se o número é zero ou não, a flag Carry/Borrow são indicadores de erro para números naturais e o Overflow para números relativos. De seguida os bits de controlo selecionam o resultado correto, utilizando mux4x1 para o R e Z, e mux2x1 para o Carry/Borrow e Overflow.

4.Operações

4.1Operação Soma

A soma consiste numa expressão lógica que opera bit a bit, entre os operandos A e B, de 4 bits cada obtendo o Resultado também de 4 bits e as respetivas flags.

	C_{y2}	C_{y1}	C_{y0}		
	A_3	A_2	A_1	A_0	
+	B_3	B_2	B_1	B_0	
C_{yOut}	S_3	S_2	S_1	S_0	

Figura 4 - Exemplo de uma soma bit a bit

A partir do exemplo foi criada uma tabela de verdade com o Carry Out(CyOut) e com o Resultado (S):

A	B	CyIn	CyOut	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabela 2 - Tabela de verdade da soma

Para chegar às expressões lógicas foi preciso construir as tabelas de verdade e também os mapas de Karnaugh:

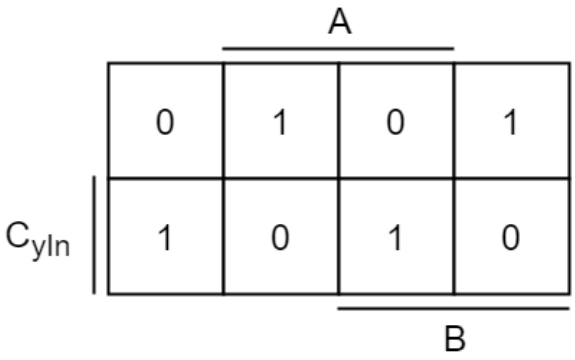


Figura 6 - Mapa de Karnaugh do Resultado

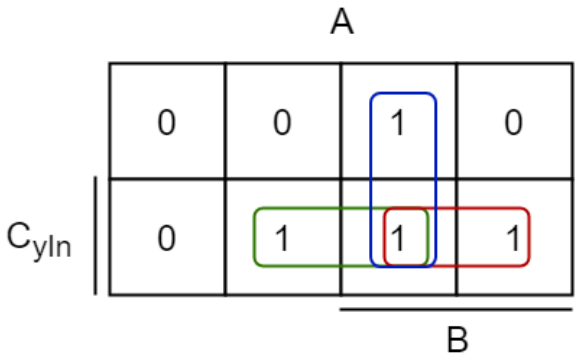


Figura 5 - Mapa de Karnaugh do Carry

A partir dos mapas e da tabela foi possível obter as seguintes expressões lógicas:

$$S = A \oplus B \oplus Cy$$

$$Cy_{out} = Cy(A \oplus B) + A.B$$

Com a obtenção das expressões seguintes foi possível construir os seguintes circuitos lógicos:

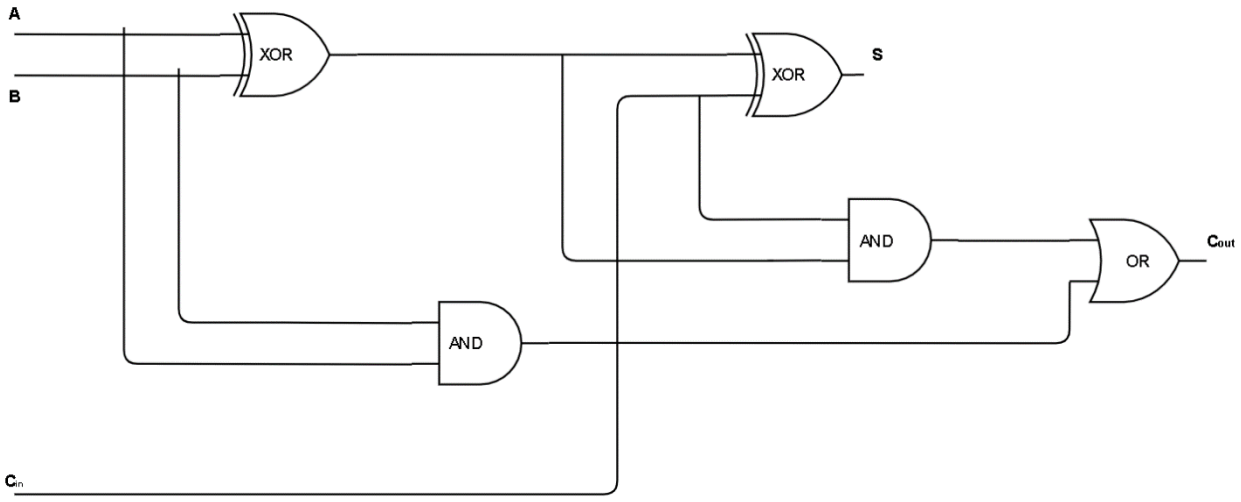


Figura 7 - Circuito logico do somador

A partir da tabela de verdade do Overflow obtivemos a seguinte expressão:

$$Ov = A_3 \oplus B_3 \oplus S_3 \oplus Cy_3$$

A	B	S	Carry	Ov
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabela 3 - Tabela de verdade do Overflow

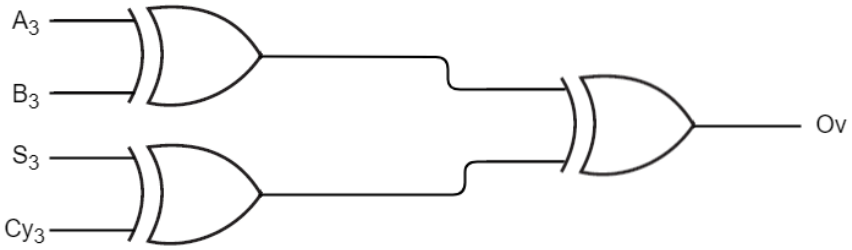


Figura 8 - Circuito logico do Overflow da soma

Uma representação mais geral do modelo da soma:

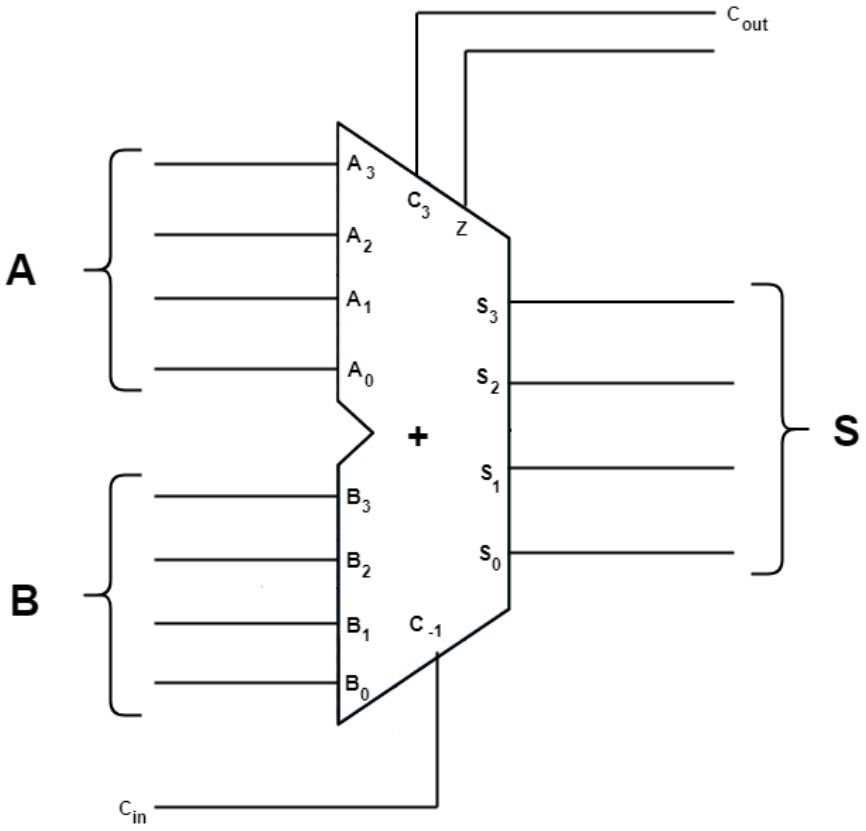


Figura 9 - Módulo funcional da soma

O Módulo funcional esta divido em 4 módulos mais pequenos que operam todos de forma igual como é demonstrado na Figura 10.

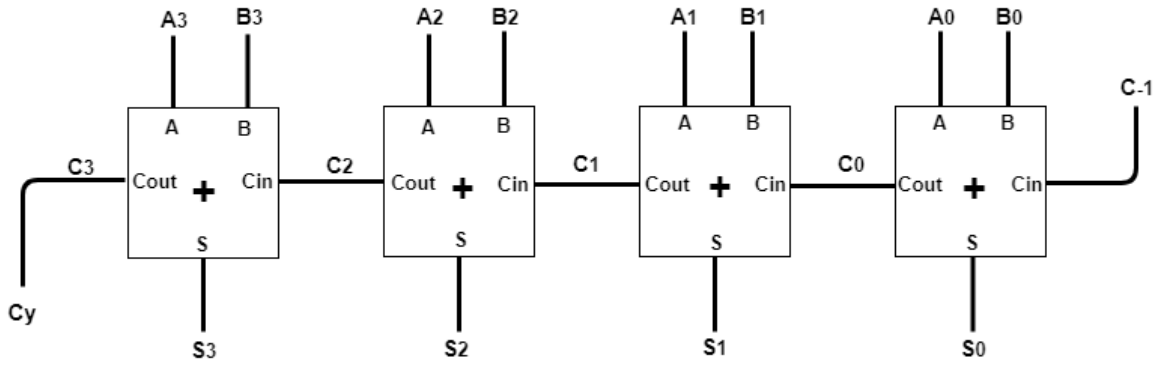


Figura 10 - Exemplo detalhado da adição utilizando módulos de soma (hardware)

As Flags no somador e o que representam:

A flag de Carry é um indicador de erro da operação soma para números naturais, que no caso vão de (0-15). Sempre que o Carry(Cy) for igual a 1 é porque não é possível representar o resultado com o número de bits disponível. Quer isto dizer que seria necessário mais bits para a sua representação, logo o resultado obtido não está correto, porque o resultado excedeu os 4 bits da sua representação. O overflow tal como o Carry representa um erro associado à soma mas para números relativos (-8,+7), tal como o Carry é quando não é possível de representar com os 4 bits.

Código:

```
282 bool Ovlbit(bool A, bool B, bool S, bool Cy){
283   return A^B^S^Cy;
284 }
285 void somador4bits(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
286   Cylbit = 0;
287   S0[0] = somadorlbit(aa0,bb0,0);
288   S0[1] = somadorlbit(aa1,bb1,Cylbit);
289   S0[2] = somadorlbit(aa2,bb2,Cylbit);
290   S0[3] = somadorlbit(aa3,bb3,Cylbit);
291   Z0 = mux2x1(1,0,(S0[0]||S0[1]||S0[2]||S0[3]));
292   Ov0 = Ovlbit(aa3,bb3,S0[3],Cylbit);
293   CylbitH = Cylbit;
294 }
295 bool somadorlbit(bool A,bool B,bool CarryIn){
296   bool S = CarryIn^A^B;
297   //bool S = CarryIn&((!A&&!B)||(A&&B))||!CarryIn&&((!A&&B)||(A&&!B));
298   Cylbit = A&&B || CarryIn&&(A^B);
299   return S;
300 }
```

4.1Operação Subtração

Para a subtração tal como para a soma foi construída a tabela de verdade seguinte:

A	B	BwIn	BwOut	S
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabela 4 - Tabela de verdade da subtração

Para chegar às expressões lógicas foi necessário construir os mapas de Karnaugh utilizando as tabelas de verdade:

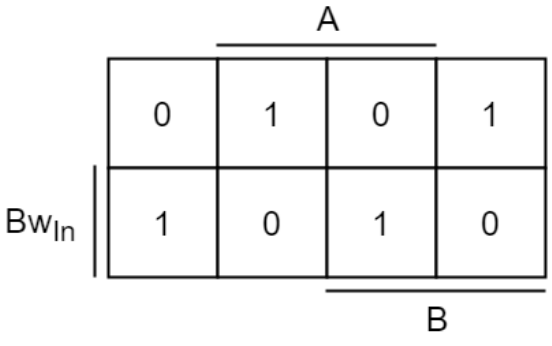


Figura 11 - Mapa de Karnaugh do Resultado

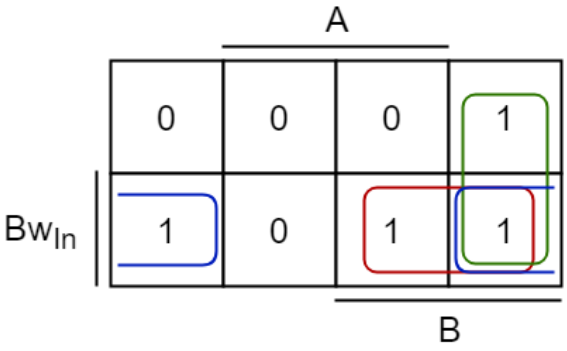


Figura 12 - Mapa de Karnaugh do Bwout

A partir dos mapas e da tabela foi possível obter as expressões lógicas seguintes:

$$S = (A \oplus B) \oplus Bw$$

$$Bw_{out} = \bar{A}.B + ((\overline{A \oplus B}).Bw)$$

$$Ov = A_3 \oplus \overline{B_3} \oplus S_3 \oplus \overline{Bw_3}$$

Com as expressões foi construído o seguinte circuíto lógico:

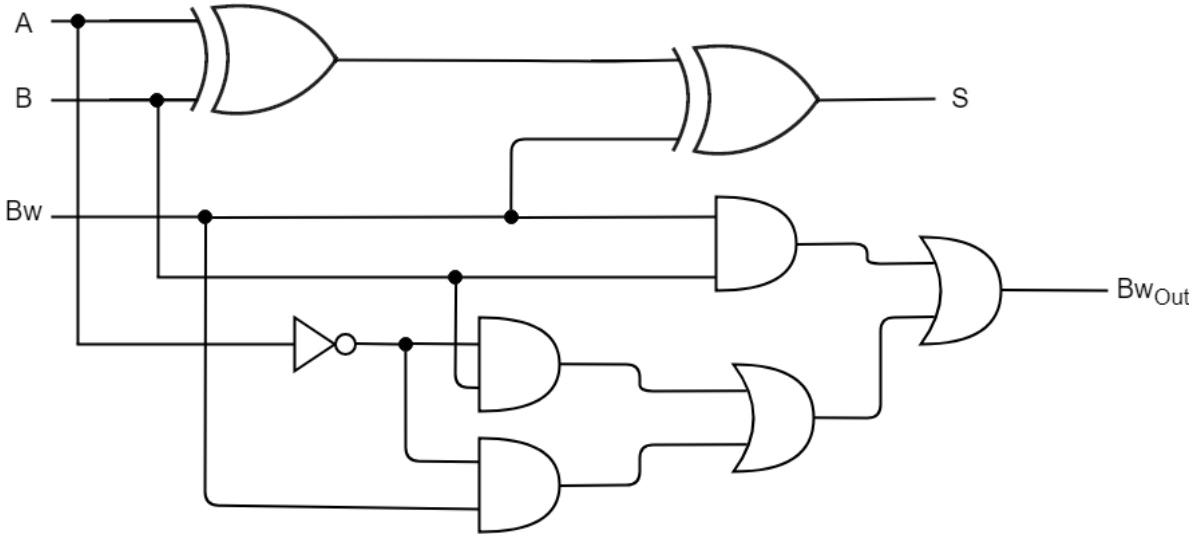


Figura 13 - Circuito lógico da subtração

Com a tabela seguinte foi possível construir as expressões lógicas seguintes:

$$Ov = A_3 \oplus \overline{B_3} \oplus S_3 \oplus \overline{Bw_3}$$

A	!B	S	!Bw	Ov
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabela 5 - Tabela Overflow da subtração

O respetivo circuito lógico obtido diretamente da expressão lógica:

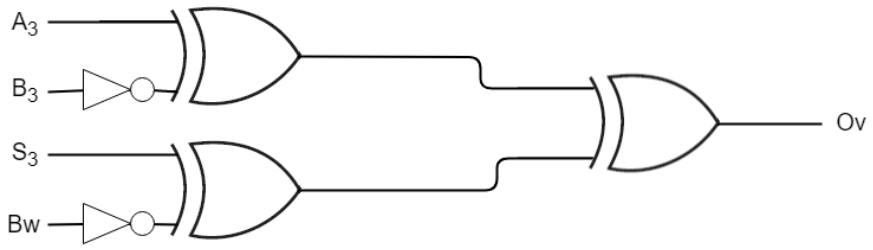


Figura 14 - Circuito lógico Overflow subtração

A subtração será feita à custa da soma, com o código de complementos (complemento para dois) onde se complementa o número binário B, adicionando 1 à soma. Isso acontece Através da formula S = A-B (=) S = A+(-B). Realizando duas operações com apenas um módulo, reduzindo a quantidade de módulos necessários e optimizando o código.

	C_{y2}	C_{y1}	C_{y0}		
	A_3	A_2	A_1	A_0	
	$!B_3$	$!B_2$	$!B_1$	$!B_0$	
+	0	0	0	1	
!Carry = Borrow	S_3	S_2	S_1	S_0	

Figura 15 - Exemplo de uma soma bit a bit

Uma representação mais geral do modelo da subtração de A com B utilizando a código de complementos

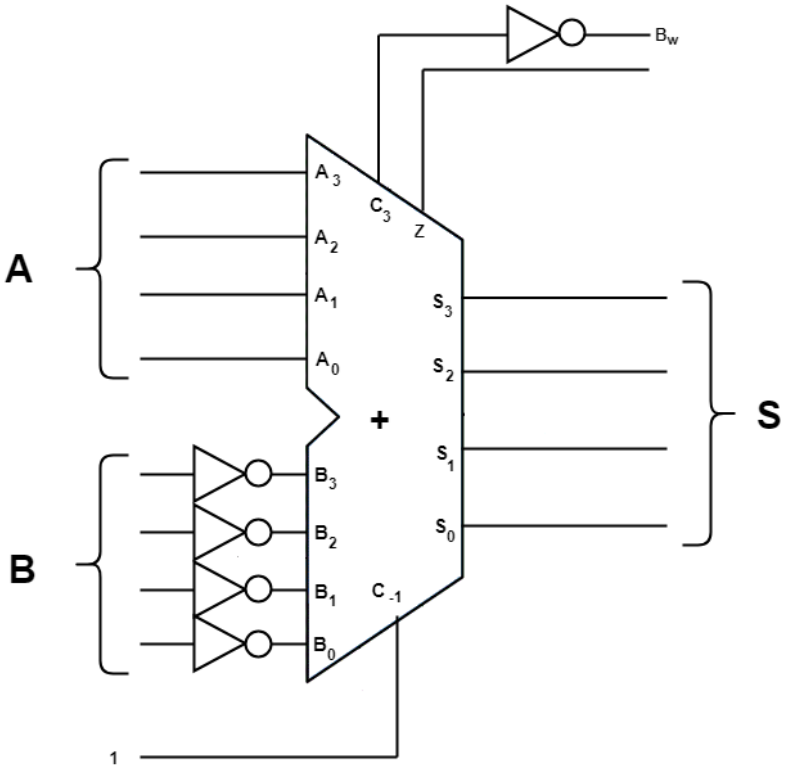


Figura 16 - Módulo funcional da subtração

O Módulo funcional esta divido em 4 módulos mais pequenos que operam todos de forma igual como é demonstrado na Figura 5.

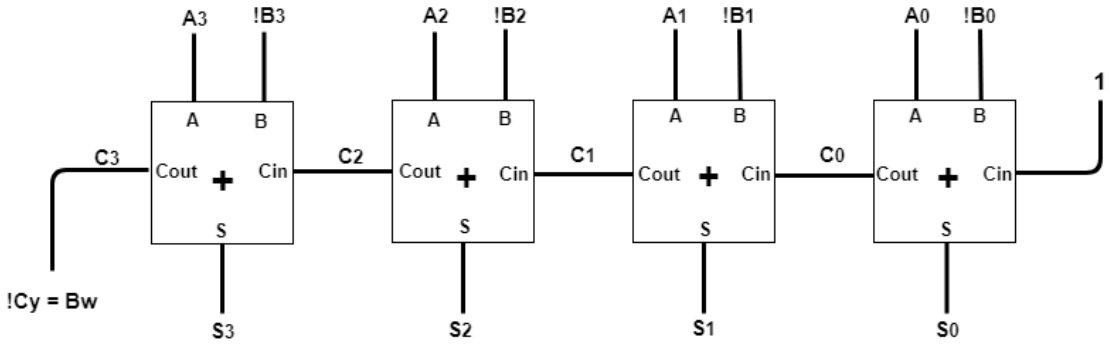


Figura 17 - Exemplo detalhado da subtração utilizando módulos de soma (hardware)

As Flags do subtrator e o representam:

A flag Borrow(Bw) tal como o Carry é uma flag de representação de erro para números naturais, que acende quando não é possível representar o resultado com os bits disponíveis. Como na subtração foi utilizado a opereção soma, a flag representante é o Carry e não o Borrow, mas como o Borrow é o contrário do Carry é possível obter o Borrow pela negação do Carry. O Overflow da subtração tal como o da soma, também indica o erro para subtrações de números relativos, resultando num número que não é possível obter com 4 bits. A flag Z indica se o número está todo a zero ou não.

$$Cy = 0 \Rightarrow Bw = 1$$
$$Cy = 1 \Rightarrow Bw = 0$$

Código:

```
266 void subtrator4bits(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
267   Sl[0] = somadorlbit(aa0,!bb0,1);
268   Sl[1] = somadorlbit(aa1,!bb1,Cylbit);
269   Sl[2] = somadorlbit(aa2,!bb2,Cylbit);
270   Sl[3] = somadorlbit(aa3,!bb3,Cylbit);
271   Zl = mux2x1(1,0,(Sl[0]||Sl[1]||Sl[2]||Sl[3]));
272   Bwlbit = !Cylbit;
273   Ovl = Ovlbit(aa3,!bb3,Sl[3],!Bwlbit);
274 }
```

4.3 NOR

O NOR é a junção das funções NOT e OR, sendo por isso o complemento do OR onde só toma valor 1 quando ambas as variáveis estiverem a 0.

Tabela de Verdade:

NOR

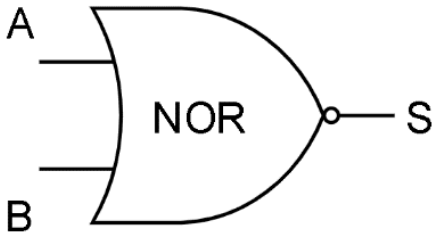
A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

S = /A./B

Mapas Karnaugh:

NOR	B	
	1	0
A	0	0

Expressão lógica:



Código:

```
256 void NOR4bit(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
257   S3[0] = NOR1bit(aa0, bb0);
258   S3[1] = NOR1bit(aa1, bb1);
259   S3[2] = NOR1bit(aa2, bb2);
260   S3[3] = NOR1bit(aa3, bb3);
261   Z3 = mux2x1(1,0,(S3[0]||S3[1]||S3[2]||S3[3]));
262 }
263 bool NOR1bit(bool A, bool B){
264   return (!A&&!B);
265 }
```

4.4 MUX's

O conceito de multiplexagem consiste em escolher apenas uma entre n entradas possíveis e liga-la à saída. Sendo neste caso um dispositivo com 2^Nentradas e 1 saída. O MUX(multiplexer lógico) coloca na saída o valor lógico presente na entrada determinados pelos bits de seleção.

Tabela de verdade do MUX 2x1

MUX 2x1

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$Z = /S.A + S.B$

Tabela de verdade do MUX 4x1

C_1	C_0	X_3	X_2	X_1	X_0	S
0	0	0	0	0	**1**	1
0	1	0	0	**1**	0	1
1	0	0	**1**	0	0	1
1	1	**1**	0	0	0	1

$S = /C_1./C_0.X_0 + /C_1.C_0.X_1 + C_1./C_0.X_2 + C_1.C_0.X_3$

Mapa de karnaugh do MUX 2x1

MUX2x1	A			
	0	**0**	**1**	**1**
C	**0**	**1**	**1**	**0**
		B		

Expressões lógicas:

S = /C.A+C.B

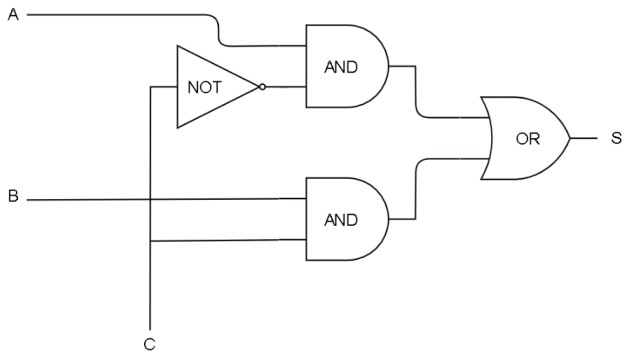


Figura 19 - MUX2x1

S = /C1./C0.X0+ /C1.C0.X1 + C1./C0.X2 + C1.C0.X3

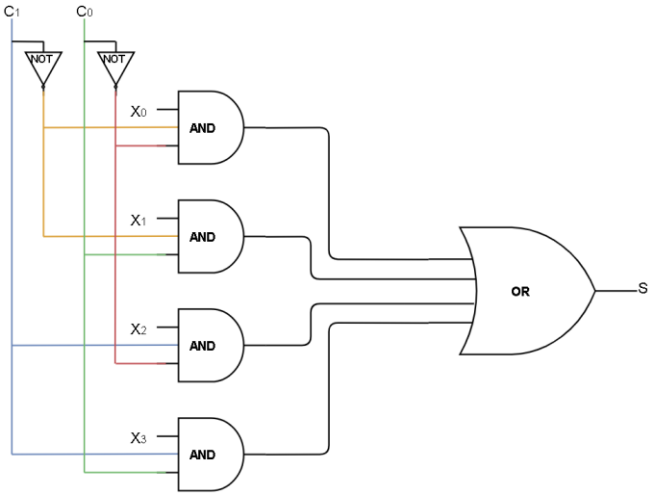


Figura 18 - MUX4x1

Código:

```
303 bool mux2x1(bool A, bool B, bool C){
304   return (!C)? A:B;
305 }
306
307 bool mux4x2(bool AA, bool BB, bool CC, bool DD, bool E1, bool E0){
308   return (!E1&&!E0&&AA || !E1&&E0&&BB || E1&&!E0&&CC || E1&&E0&&DD);
309 }
```

4.5 Aritmetic Shift Right (ASR)

Entradas e saídas ASR

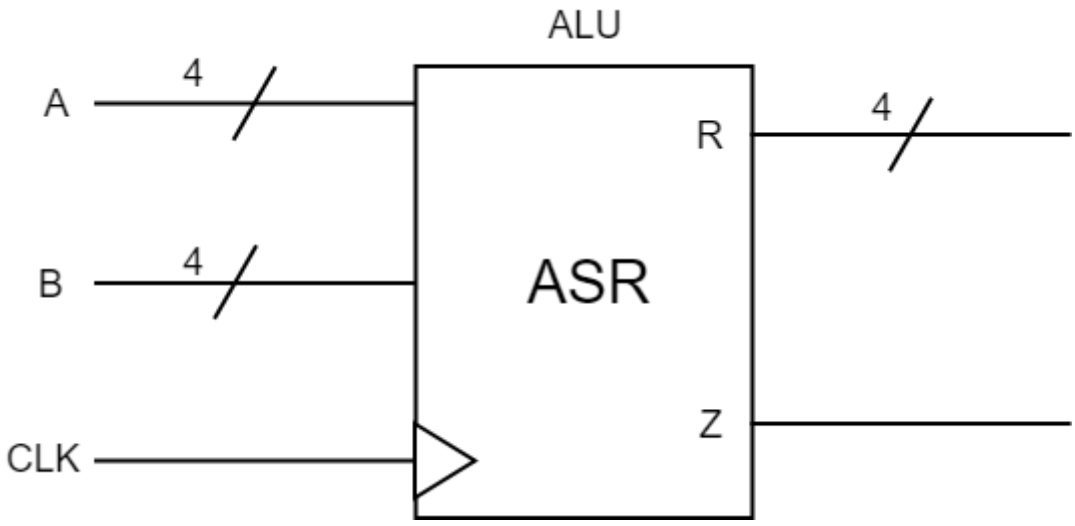


Figura 20 - Modulo do ASR

Algoritmo do deslocamento aritmético para a direita ASR

$$ASR = \frac{A}{2^B} \qquad ou \qquad A \gg B$$

Como podemos ver na fórmula, o A está a ser dividido por 2^B, e como A é um número binário, esta divisão é relativamente simples, se considerarmos A = 0100 (+4) e B = 0001 (+1):

$$\frac{0100}{0010} = 0010 \;\; ou \;\; \frac{4}{2^1} = (+2)$$

Como podemos ver na figura, a parte numérica do número A anda uma casa para a direita e o bit sinal é adicionado a esquerda:

$$A_3A_2A_1A_0 \gg 0001 = A_3A_3A_2A_1$$

Desenho modulo funcional ASR Combinatório:

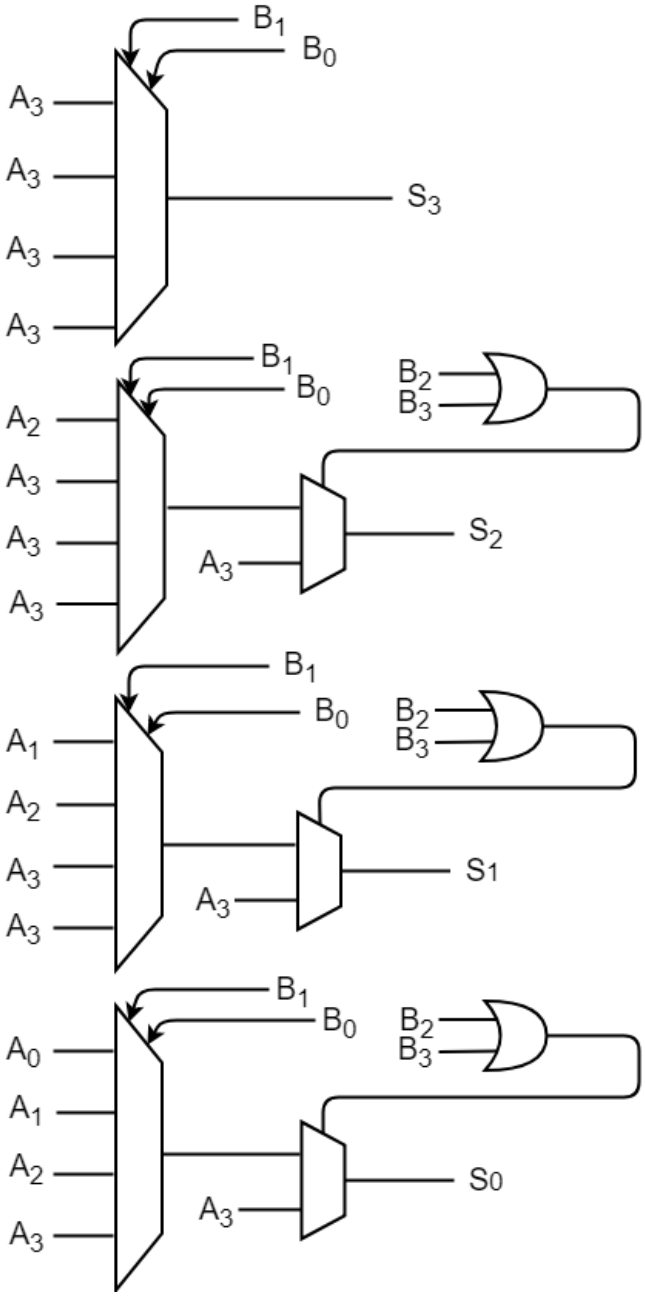


Figura 21 - modulo funcional ASR (Combinatório)

Primeiramente foi realizado o ASR de forma combinatória para mais tarde utilizar no projeto. Utilizando vários mux4x1 e mux2x1 é possível determinar o resultado final da operação, embora na parte sequencial só seja feito "1 shift de cada vez". Por isso esta parte é utilizada para imprimir a solução correta apenas para ajudar o utilizador a interpretar o resultado obtido na parte sequencial.

Desenho modulo funcional ASR Sequencial:

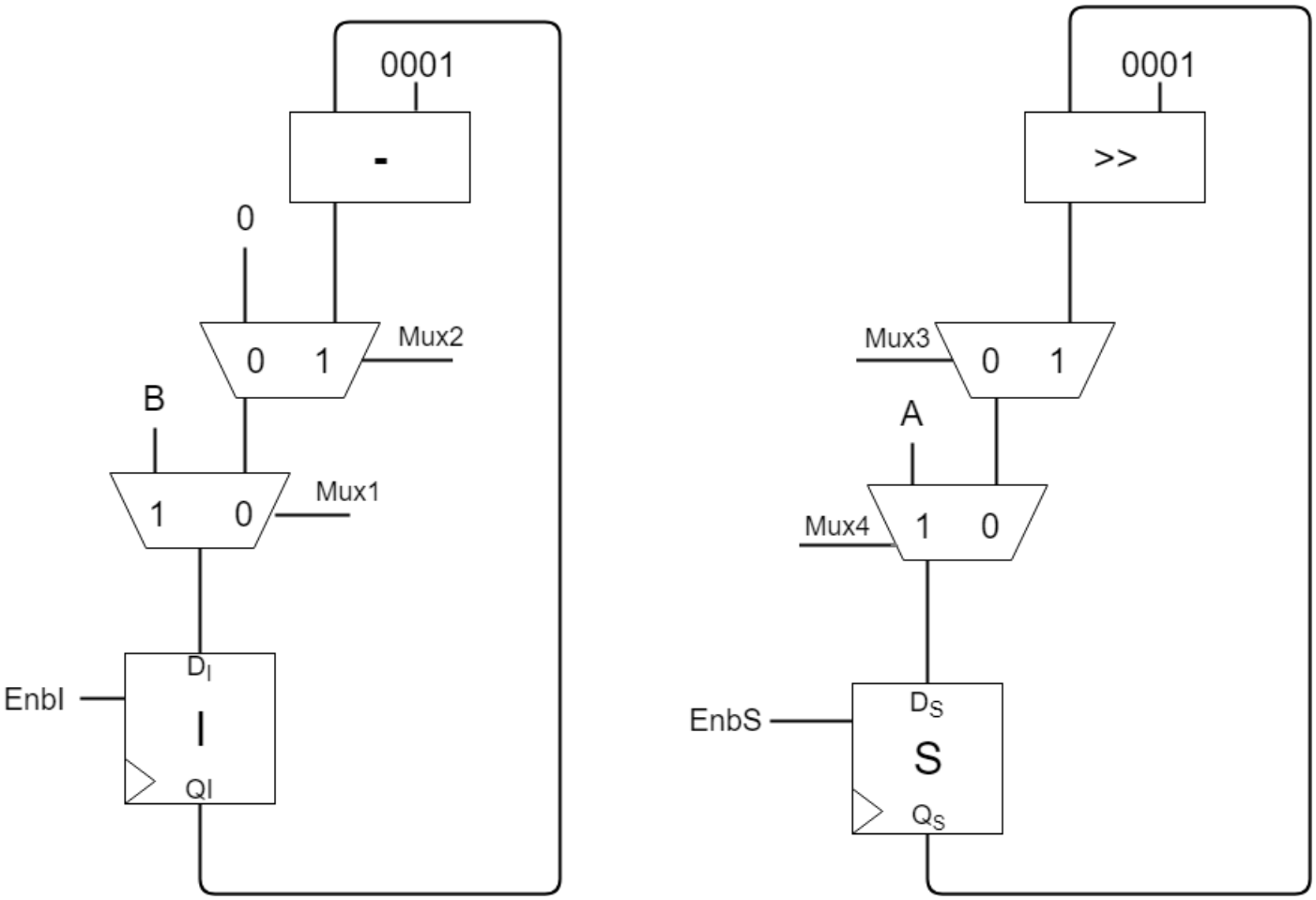


Figura 22 - modulo funcional ASR (Sequencial)

Entradas e saídas do modulo de controlo:

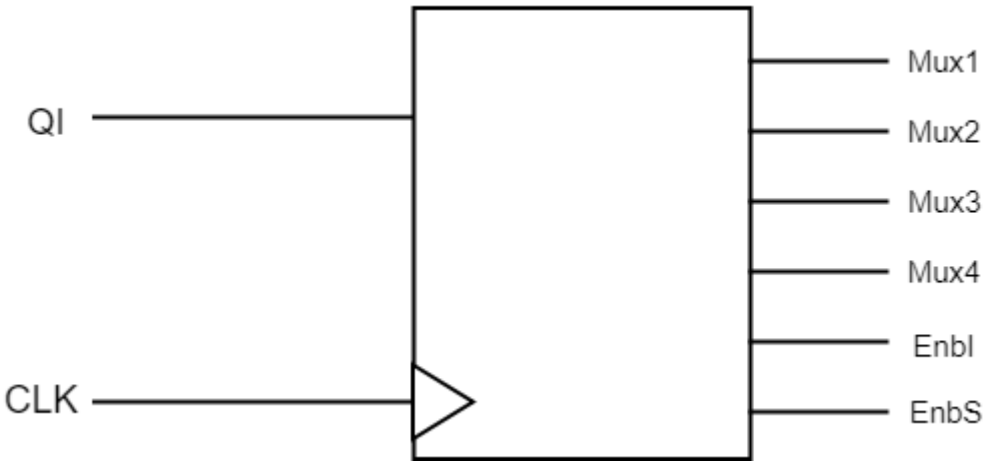


Figura 23 - Modulo de control ASR

ASM do modulo de Controlo e expressões logicas de F.E.S e F.S

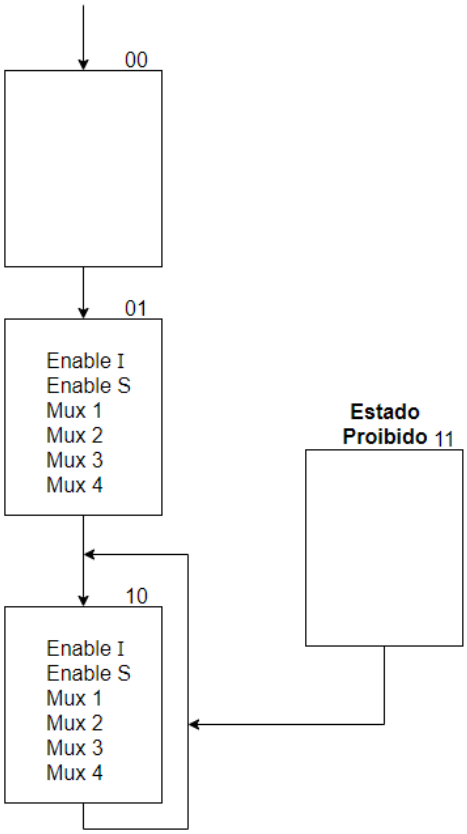


Figura 24 - ASM do Modulo de controlo

Q1*	Q0*	Q1	Q0	D1	D0
0	0	0	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0

Tabela 6 - Calculo FES

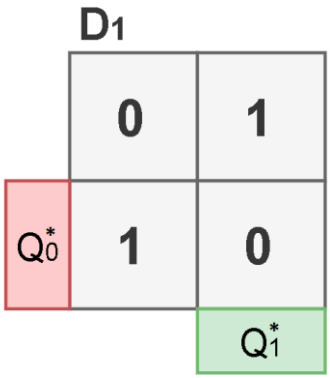


Figura 26 - Mapas de Karnaugh D1

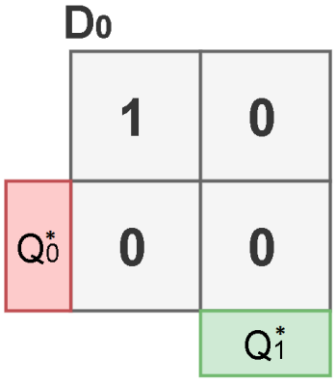


Figura 25 - Mapas de Karnaugh D0

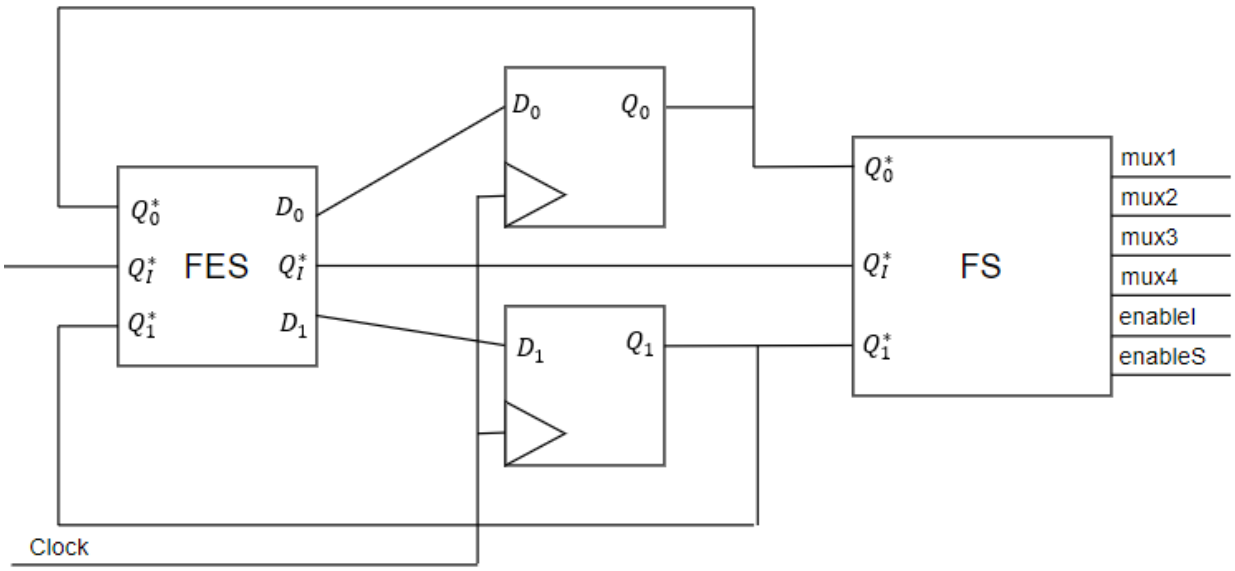


Figura 27 - Modelo de Moore-Mealey utilizado

FES:

$D0 = \overline{Q_1^*}.\overline{Q_0^*}$

$D1 = Q_1^* \oplus Q_0^*$

FS:

Mux1 = Q_0^*

Mux2 = $Q_I^*[3] + Q_I^*[2] + Q_I^*[1] + Q_I^*[0]$

Mux3 = $Q_I^*[3] + Q_I^*[2] + Q_I^*[1] + Q_I^*[0]$

Mux4 = Q_0^*

enableI = 1

enableS = 1

Montagem:

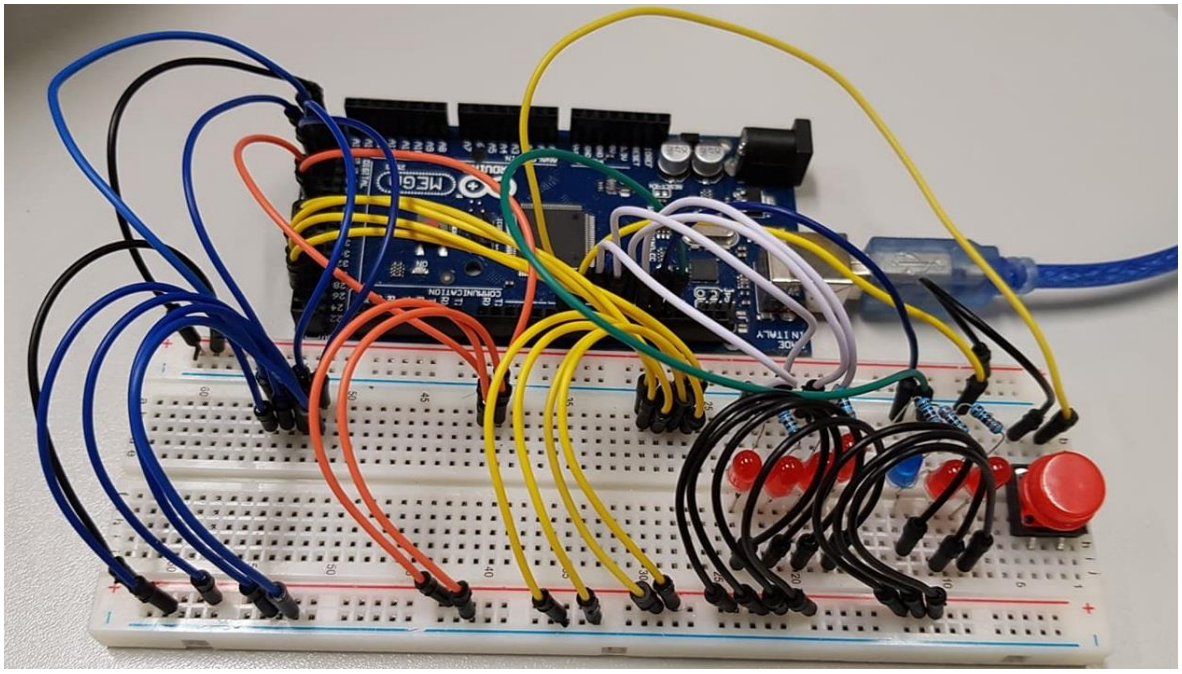


Figura 28 - Fotografia montage 1

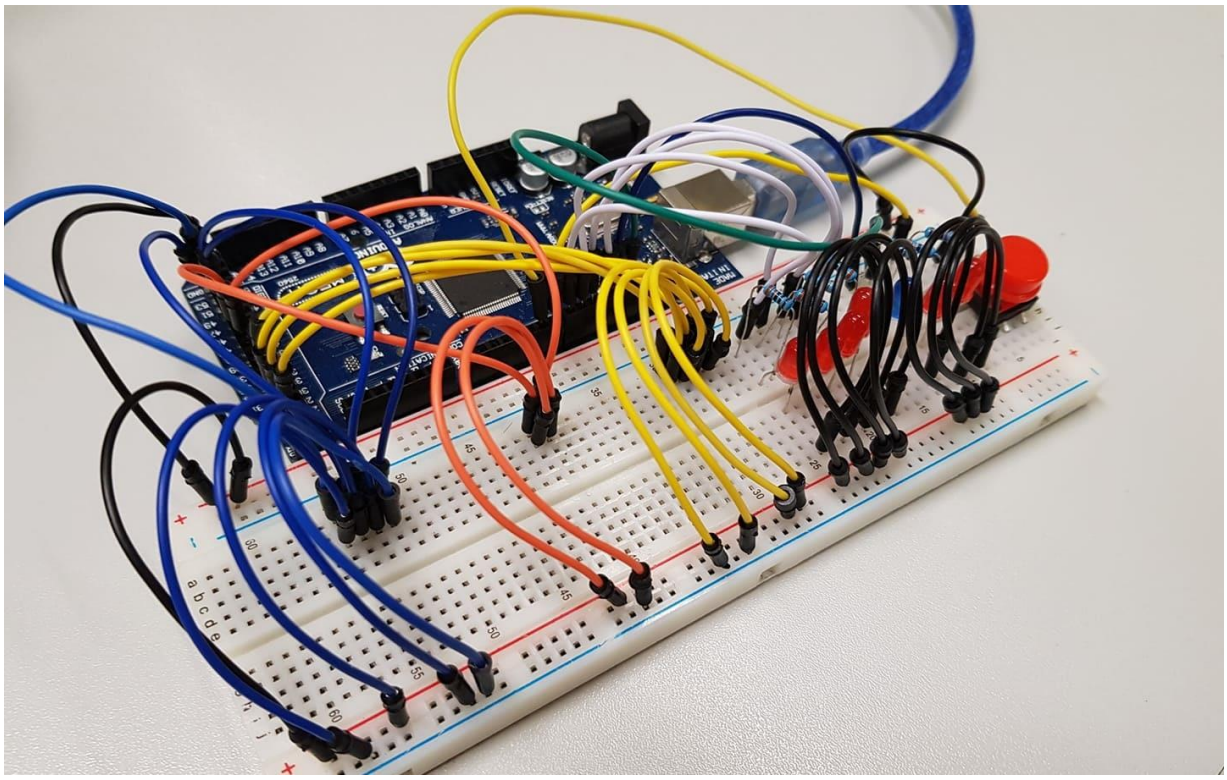


Figura 29 - Fotografia montage 2

5.Conclusões

No presente trabalho o grupo foi capaz de adquirir competências tais como:

-Esquematizar o problema em diversas parcelas mais simples de entender e por em prática.

-Elaborar tabelas de verdade e mapas de karnaugh para a obtenção de expressões lógicas, tendo por base o valor lógico das entradas, ter-se-ão as saídas pretendidas.

-Desenho dos circuitos combinatórios que representam as expressões lógicas obtidas.

-Definição de algoritmos para efetuar o ASR através de subtrações sucessivas.

-desenho de máquinas de estados que represente, conforme os valores das variáveis e condição de transição.

-Desenho do módulo funcional para explicar de que forma os dados são encaminhados para a saída.

-Utilização de flip-flops e registos.

-Simulação de tudo no Arduíno com LED's

A maior dificuldade do grupo foi a Implementação do ASR devido à sua complexidade e pelo facto de ser necessário registos e enables.

6.Bibliografia

[1] Folhas de Computação Física, Jorge Pais, 2018/2019

[2] Folhas de Computação Física, Carlos Carvalho, 2019

Anexo (Código)

```
#define A0 47
#define A1 49
#define A2 51
#define A3 53
#define B0 23
#define B1 25
#define B2 27
#define B3 29
#define C0 39
#define C1 41

#define LED0 11
#define LED1 10
#define LED2 9
#define LED3 8
#define LED4 12
#define LED5 13
#define LED6 7
#define button 6

bool A[] = {0,0,0,0};
bool B[] = {0,0,0,0};
bool C[] = {0,0};
bool Cylbit,Bwlbit, CylbitH, BwlbitASR;

bool S0[] = {0,0,0,0};
bool S1[] = {0,0,0,0};
bool S2[] = {0,0,0,0};
bool S3[] = {0,0,0,0};

bool Z0,Z1,Z2,Z3;

bool Ov0,Ov1;

////ALU OUT/////
bool SOUT[] = {0,0,0,0};
bool CyBwOUT;
bool OvOUT;
bool Zout;


////ASR////
bool mux1, mux2, mux3, mux4, enableI, enableS;
bool clock1 = 0;
bool subIn[] = {0,0,0,0};
bool subOut[] = {0,0,0,0};
bool shiftRightOUT[] = {0,0,0,0};
bool Q = 1;

```

```
bool mux2x1Out[] = {0,0,0,0};
bool mux2x1Out2[] = {0,0,0,0};
bool mux2x1Out3[] = {0,0,0,0};
bool mux2x1Out4[] = {0,0,0,0};
bool registoIOut[] = {0,0,0,0};
bool registoSOut[] = {0,0,0,0};
bool ZeroASR = 0;

bool Q0 = 0, Q1 = 0, D0 = 0, D1 = 0;

void setup() {
  Serial.begin(9600);
  pinMode(A0, INPUT_PULLUP);
  pinMode(A1, INPUT_PULLUP);
  pinMode(A2, INPUT_PULLUP);
  pinMode(A3, INPUT_PULLUP);
  pinMode(B0, INPUT_PULLUP);
  pinMode(B1, INPUT_PULLUP);
  pinMode(B2, INPUT_PULLUP);
  pinMode(B3, INPUT_PULLUP);
  pinMode(C1, INPUT_PULLUP);
  pinMode(C0, INPUT_PULLUP);
  pinMode(button, INPUT_PULLUP);
  pinMode(LED0, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED5, OUTPUT);
  pinMode(LED6, OUTPUT);
  getAandBandC();
  alu(A,B,C[1],C[0]);
  aluOUTprint();
  //printNumbersBinary();
  //printNumbers();

}

void loop() {

  getAandBandC();
  moduloControlo();
  alu(A,B,C[1],C[0]);
  ledF();
  if(clock1){
    printASR();
  }

```

```
void getAandBandC(){
  A[3] = !digitalRead(A3);
  A[2] = !digitalRead(A2);
  A[1] = !digitalRead(A1);
  A[0] = !digitalRead(A0);


  B[3] = !digitalRead(B3);
  B[2] = !digitalRead(B2);
  B[1] = !digitalRead(B1);
  B[0] = !digitalRead(B0);

  C[1] = !digitalRead(C1);
  C[0] = !digitalRead(C0);
  clock1 = !digitalRead(button);



}

void ledF(){
  digitalWrite(LED0,SOUT[0]);
  digitalWrite(LED1,SOUT[1]);
  digitalWrite(LED2,SOUT[2]);
  digitalWrite(LED3,SOUT[3]);
  digitalWrite(LED4,OvOUT);
  digitalWrite(LED5,CyBwOUT);
  digitalWrite(LED6,Zout);


}
void alu(bool A[], bool B[], bool CC1, bool CC0){

  somador4bits(A[3], A[2], A[1], A[0], B[3], B[2], B[1], B[0]);
  subtrator4bits(A[3], A[2], A[1], A[0], B[3], B[2], B[1], B[0]);
  shiftRight2(A[3], A[2], A[1], A[0], B[3], B[2], B[1], B[0]);
  shiftRight4(A[3], A[2], A[1], A[0], B[3], B[2], B[1], B[0]);
  NOR4bit(A[3], A[2], A[1], A[0], B[3], B[2], B[1], B[0]);

  SOUT[3]= mux4x2(S0[3],S1[3],registoSOut[3],S3[3], CC1,CC0);
  SOUT[2]= mux4x2(S0[2],S1[2],registoSOut[2],S3[2], CC1,CC0);
  SOUT[1]= mux4x2(S0[1],S1[1],registoSOut[1],S3[1], CC1,CC0);
  SOUT[0]= mux4x2(S0[0],S1[0],registoSOut[0],S3[0], CC1,CC0);
  OvOUT = mux2x1(Ov0,Ov1,CC0);
  Zout = mux4x2(Z0,Z1,Z2,Z3,CC1,CC0);
  CyBwOUT = mux2x1(Cy1bitH,Bw1bit,CC0);

}
```

```
150 void moduloControlo(){
151   if(clock1){
152     FES();
153     flipFlopD0(D0);
154     flipFlopD1(D1);
155     FS();
156   }
157 }
158
159 void FES(){
160   D0 = !Q1&&!Q0;
161   D1 = Q1^Q0;
162
163 }
164 void FS(){
165   mux1 =Q0;
166   mux2 =(registoIOut[3] || registoIOut[2] || registoIOut[1]|| registoIOut[0]);
167   mux3 =(registoIOut[3] || registoIOut[2] || registoIOut[1]|| registoIOut[0]);
168   mux4 = Q0;
169   enableI = 1;
170   enableS = 1;
171 }
172 void flipFlopD0(bool D0){
173   Q0 = (clock1)?D0:Q0;
174 }
175
176 void flipFlopD1(bool D1){
177   Q1 = (clock1)?D1:Q1;
178 }
179
180 void shiftRight2(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
181   mux2x1Out2[3] = mux2x1(0,subOut[3],mux2);
182   mux2x1Out2[2] = mux2x1(0,subOut[2],mux2);
183   mux2x1Out2[1] = mux2x1(0,subOut[1],mux2);
184   mux2x1Out2[0] = mux2x1(0,subOut[0],mux2);
185
186   mux2x1Out[3] = mux2x1(mux2x1Out2[3], bb3,mux1);
187   mux2x1Out[2] = mux2x1(mux2x1Out2[2], bb2,mux1);
188   mux2x1Out[1] = mux2x1(mux2x1Out2[1], bb1,mux1);
189   mux2x1Out[0] = mux2x1(mux2x1Out2[0], bb0,mux1);
190   RegistoI(mux2x1Out[3], mux2x1Out[2], mux2x1Out[1], mux2x1Out[0], enableI&&clock1);
191
192   subASR(registoIOut[3], registoIOut[2], registoIOut[1], registoIOut[0], 0, 0, 0, 1);
193
194   mux2x1Out2[3] = mux2x1(0,subOut[3],mux2);
195   mux2x1Out2[2] = mux2x1(0,subOut[2],mux2);
196   mux2x1Out2[1] = mux2x1(0,subOut[1],mux2);
197   mux2x1Out2[0] = mux2x1(0,subOut[0],mux2);
198
```

```
198
199   mux2x1Out[3] = mux2x1(mux2x1Out2[3], bb3,mux1);
200   mux2x1Out[2] = mux2x1(mux2x1Out2[2], bb2,mux1);
201   mux2x1Out[1] = mux2x1(mux2x1Out2[1], bb1,mux1);
202   mux2x1Out[0] = mux2x1(mux2x1Out2[0], bb0,mux1);
203
204   shiftRight3();
205 }
206 void shiftRight3(){
207   mux2x1Out3[3] = mux2x1(registoSOut[3],shiftRightOUT[3],mux3);
208   mux2x1Out3[2] = mux2x1(registoSOut[2],shiftRightOUT[2],mux3);
209   mux2x1Out3[1] = mux2x1(registoSOut[1],shiftRightOUT[1],mux3);
210   mux2x1Out3[0] = mux2x1(registoSOut[0],shiftRightOUT[0],mux3);
211
212
213   mux2x1Out4[3] = mux2x1(mux2x1Out3[3], A[3], mux4);
214   mux2x1Out4[2] = mux2x1(mux2x1Out3[2], A[2], mux4);
215   mux2x1Out4[1] = mux2x1(mux2x1Out3[1], A[1], mux4);
216   mux2x1Out4[0] = mux2x1(mux2x1Out3[0], A[0], mux4);
217
218   RegistoS(mux2x1Out4[3], mux2x1Out4[2], mux2x1Out4[1], mux2x1Out4[0], enableS&&clock1);
219   shiftRight(registoSOut[3],registoSOut[2],registoSOut[1],registoSOut[0], 0, 0, 0, 1);
220
221   mux2x1Out3[3] = mux2x1(registoSOut[3],shiftRightOUT[3],mux3);
222   mux2x1Out3[2] = mux2x1(registoSOut[2],shiftRightOUT[2],mux3);
223   mux2x1Out3[1] = mux2x1(registoSOut[1],shiftRightOUT[1],mux3);
224   mux2x1Out3[0] = mux2x1(registoSOut[0],shiftRightOUT[0],mux3);
225
226   mux2x1Out4[3] = mux2x1(mux2x1Out3[3], A[3], mux4);
227   mux2x1Out4[2] = mux2x1(mux2x1Out3[2], A[2], mux4);
228   mux2x1Out4[1] = mux2x1(mux2x1Out3[1], A[1], mux4);
229   mux2x1Out4[0] = mux2x1(mux2x1Out3[0], A[0], mux4);
230
231   Z2 = mux2x1(1,0,(registoSOut[0]||registoSOut[1]||registoSOut[2]||registoSOut[3]));
232
233
234 }
235
236 void RegistoS(bool m3, bool m2, bool m1, bool m0, bool clock2){
237   registoSOut[3] = mux2x1( registoSOut[3], m3, clock2);
238   registoSOut[2] = mux2x1( registoSOut[2], m2, clock2);
239   registoSOut[1] = mux2x1( registoSOut[1], m1, clock2);
240   registoSOut[0] = mux2x1( registoSOut[0], m0, clock2);
241 }
242
```

```
void subASR(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
  BwlbitASR = 0;
  subOut[0] = subtratorlbit(aa0,bb0,0);
  subOut[1] = subtratorlbit(aa1,bb1,BwlbitASR);
  subOut[2] = subtratorlbit(aa2,bb2,BwlbitASR);
  subOut[3] = subtratorlbit(aa3,bb3,BwlbitASR);
}
void RegistoI(bool m3, bool m2, bool m1, bool m0, bool clock2){
  registoIOut[3] = mux2x1( registoIOut[3], m3, clock2);
  registoIOut[2] = mux2x1( registoIOut[2], m2, clock2);
  registoIOut[1] = mux2x1( registoIOut[1], m1, clock2);
  registoIOut[0] = mux2x1( registoIOut[0], m0, clock2);
}
void NOR4bit(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
  S3[0] = NOR1bit(aa0, bb0);
  S3[1] = NOR1bit(aa1, bb1);
  S3[2] = NOR1bit(aa2, bb2);
  S3[3] = NOR1bit(aa3, bb3);
  Z3 = mux2x1(1,0,(S3[0]||S3[1]||S3[2]||S3[3]));
}
bool NOR1bit(bool A, bool B){
  return (!A&&!B);
}
void subtrator4bits(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
  S1[0] = somadorlbit(aa0,!bb0,1);
  S1[1] = somadorlbit(aa1,!bb1,Cylbit);
  S1[2] = somadorlbit(aa2,!bb2,Cylbit);
  S1[3] = somadorlbit(aa3,!bb3,Cylbit);
  Z1 = mux2x1(1,0,(S1[0]||S1[1]||S1[2]||S1[3]));
  Bwlbit = !Cylbit;
  Ov1 = Ovlbit(aa3,!bb3,S1[3],!Bwlbit);



}

bool subtratorlbit(bool A, bool B, bool BwIn){
  bool S = !A&&!B&&BwlbitASR || !A&&B&&!BwlbitASR ||A&&!B&&!BwlbitASR || A&&B&&BwlbitASR;
  BwlbitASR = B&&BwIn || !A&&B || !A&&BwIn;
  return S;
}

bool Ovlbit(bool A, bool B, bool S, bool Cy){
  return A^B^S^Cy;
}
```

```
288 void somador4bits(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
289   Cylbit = 0;
290   S0[0] = somadorlbit(aa0,bb0,0);
291   S0[1] = somadorlbit(aa1,bb1,Cylbit);
292   S0[2] = somadorlbit(aa2,bb2,Cylbit);
293   S0[3] = somadorlbit(aa3,bb3,Cylbit);
294   Z0 = mux2x1(1,0,(S0[0]||S0[1]||S0[2]||S0[3]));
295   Ov0 = Ovlbit(aa3,bb3,S0[3],Cylbit);
296   CylbitH = Cylbit;
297
298
299 }
300 bool somadorlbit(bool A,bool B,bool CarryIn){
301   bool S = CarryIn^A^B;
302   //bool S = CarryIn&(((!A&&!B)||(A&&B))||!CarryIn&&((!A&&B)||(A&&!B));
303   Cylbit = A&&B || CarryIn&&(A^B);
304   return S;
305
306 }
307
308
309 bool mux2x1(bool A, bool B, bool C){
310   return (!C)? A:B;
311 }
312
313 bool mux4x2(bool AA, bool BB, bool CC, bool DD, bool E1, bool E0){
314   return (!E1&&!E0&&AA || !E1&&E0&&BB || E1&&!E0&&CC || E1&&E0&&DD);
315 }
316
317 void shiftRight(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){//A numero relativo
318   bool sinal = aa3;
319   bool SS0 = mux4x2(aa0,aa1,aa2,aa3, bb1,bb0);
320   bool SS1 = mux4x2(aa1,aa2,aa3,sinal, bb1,bb0);
321   bool SS2 = mux4x2(aa2,aa3,sinal,sinal, bb1,bb0);
322   bool SS3 = mux4x2(aa3,sinal,sinal,sinal, bb1,bb0);
323   shiftRightOUT[0] = mux2x1(SS0,sinal,bb2||bb3);
324   shiftRightOUT[1] = mux2x1(SS1,sinal,bb2||bb3);
325   shiftRightOUT[2] = mux2x1(SS2,sinal,bb2||bb3);
326   shiftRightOUT[3] = mux2x1(SS3,sinal,bb2||bb3);
327   ZeroASR = mux2x1(1,0,(registoSOut[0]||registoSOut[1]||registoSOut[2]||registoSOut[3]));
328 }
```

```
329 void shiftRight4(bool aa3, bool aa2, bool aa1, bool aa0, bool bb3, bool bb2, bool bb1, bool bb0){
330   bool sinal = aa3;//A numero relativo, funcao auxiliar para saber a resposta final
331   bool SS0 = mux4x2(aa0,aa1,aa2,aa3, bb1,bb0);
332   bool SS1 = mux4x2(aa1,aa2,aa3,sinal, bb1,bb0);
333   bool SS2 = mux4x2(aa2,aa3,sinal,sinal, bb1,bb0);
334   bool SS3 = mux4x2(aa3,sinal,sinal,sinal, bb1,bb0);
335   S2[0] = mux2x1(SS0,sinal,bb2||bb3);
336   S2[1] = mux2x1(SS1,sinal,bb2||bb3);
337   S2[2] = mux2x1(SS2,sinal,bb2||bb3);
338   S2[3] = mux2x1(SS3,sinal,bb2||bb3);
339 
340 }
341 
342 void printASR(){
343   if(clock1){
344     Serial.println();
345     Serial.println("CLOCK");
346     Serial.println();
347   }
348   Serial.print("SubOUT  ");
349   Serial.print(subOut[3]);
350   Serial.print(subOut[2]);
351   Serial.print(subOut[1]);
352   Serial.println(subOut[0]);
353   Serial.print("muxOut2  ");
354   Serial.print(mux2x1Out2[3]);
355   Serial.print(mux2x1Out2[2]);
356   Serial.print(mux2x1Out2[1]);
357   Serial.println(mux2x1Out2[0]);
358   Serial.print("muxOut  ");
359   Serial.print(mux2x1Out[3]);
360   Serial.print(mux2x1Out[2]);
361   Serial.print(mux2x1Out[1]);
362   Serial.println(mux2x1Out[0]);
363   Serial.print("RegistoI  ");
364   Serial.print(registoIOut[3]);
365   Serial.print(registoIOut[2]);
366   Serial.print(registoIOut[1]);
367   Serial.println(registoIOut[0]);
368   Serial.print("ASROUT  ");
369   Serial.print(shiftRightOUT[3]);
370   Serial.print(shiftRightOUT[2]);
371   Serial.print(shiftRightOUT[1]);
372   Serial.println(shiftRightOUT[0]);
373   Serial.print("muxOut3  ");
374   Serial.print(mux2x1Out3[3]);
375   Serial.print(mux2x1Out3[2]);
376   Serial.print(mux2x1Out3[1]);
377   Serial.println(mux2x1Out3[0]);
```

```
  Serial.print("muxOut4  ");
  Serial.print(mux2x1Out4[3]);
  Serial.print(mux2x1Out4[2]);
  Serial.print(mux2x1Out4[1]);
  Serial.println(mux2x1Out4[0]);
  Serial.print("RegistoS  ");
  Serial.print(registoSOut[3]);
  Serial.print(registoSOut[2]);
  Serial.print(registoSOut[1]);
  Serial.println(registoSOut[0]);
  Serial.print("FlagZero ");
  Serial.println(ZeroASR);
  Serial.println();
  Serial.println();


  delay(200);



}
void aluOUTprint(){
  Serial.print(A[3]);
  Serial.print(A[2]);
  Serial.print(A[1]);
  Serial.print(A[0]);
  if(!C[1]&&!C[0]){
    Serial.print(" + ");
  }
  else if(!C[1]&&C[0]){
    Serial.print(" - ");
  }
  else if(C[1]&&!C[0]){
    Serial.print(" >> ");
  }
  else{
    Serial.print(" NOR ");
  }
  Serial.print(B[3]);
  Serial.print(B[2]);
  Serial.print(B[1]);
  Serial.print(B[0]);
  Serial.print(" = ");
  if(C[1]&&!C[0]){
      Serial.print(S2[3]);
      Serial.print(S2[2]);
      Serial.print(S2[1]);
      Serial.println(S2[0]);
  }else{
```

```
426   }else{
427       Serial.print(SOUT[3]);
428       Serial.print(SOUT[2]);
429       Serial.print(SOUT[1]);
430       Serial.println(SOUT[0]);
431   }
432   Serial.print("Cy/Bw = ");
433   if(!C[1]){
434     Serial.println(CyBwOUT);
435   }
436   else{
437     Serial.println("-");
438   }
439   Serial.print("Ov = ");
440   if(!C[1]){
441     Serial.println(OvOUT);
442   }
443   else{
444     Serial.println("-");
445   }
446   Serial.print("Z = ");
447   Serial.println(Zout);
448   Serial.println();
449 }
450
451 void printNumbersBinary(){
452
453   Serial.print("A = ");
454   Serial.print(A[3]);
455   Serial.print(A[2]);
456   Serial.print(A[1]);
457   Serial.println(A[0]);
458
459   Serial.print("B = ");
460   Serial.print(B[3]);
461   Serial.print(B[2]);
462   Serial.print(B[1]);
463   Serial.println(B[0]);
464   Serial.println();
465
466   Serial.print(A[3]);
467   Serial.print(A[2]);
468   Serial.print(A[1]);
469   Serial.print(A[0]);
470   Serial.print(" + ");
471   Serial.print(B[3]);
472   Serial.print(B[2]);
473   Serial.print(B[1]);
474   Serial.print(B[0]);
```

```
475   Serial.print(" = ");
476   Serial.print(S0[3]);
477   Serial.print(S0[2]);
478   Serial.print(S0[1]);
479   Serial.println(S0[0]);
480   Serial.print("Cy/Bw = ");
481   if(!0){
482     Serial.println(CylbitH);
483   }
484   else{
485     Serial.println("-");
486   }
487   Serial.print("Ov = ");
488   if(!0){
489     Serial.println(Ov0);
490   }
491   else{
492     Serial.println("-");
493   }
494   Serial.print("Z = ");
495   Serial.println(Z0);
496   Serial.println();
497
498   Serial.print(A[3]);
499   Serial.print(A[2]);
500   Serial.print(A[1]);
501   Serial.print(A[0]);
502   Serial.print(" - ");
503   Serial.print(B[3]);
504   Serial.print(B[2]);
505   Serial.print(B[1]);
506   Serial.print(B[0]);
507   Serial.print(" = ");
508   Serial.print(S1[3]);
509   Serial.print(S1[2]);
510   Serial.print(S1[1]);
511   Serial.println(S1[0]);
512   Serial.print("Cy/Bw = ");
513   if(!0){
514     Serial.println(Bw1bit);
515   }
516   else{
517     Serial.println("-");
518   }
519   Serial.print("Ov = ");
520   if(!0){
521     Serial.println(Ov1);
522   }
```

```
523   else{
524     Serial.println("-");
525   }
526   Serial.print("Z = ");
527   Serial.println(Z1);
528   Serial.println();
529
530   Serial.print(A[3]);
531   Serial.print(A[2]);
532   Serial.print(A[1]);
533   Serial.print(A[0]);
534   Serial.print(" >> ");
535   Serial.print(B[3]);
536   Serial.print(B[2]);
537   Serial.print(B[1]);
538   Serial.print(B[0]);
539   Serial.print(" = ");
540   Serial.print(S2[3]);
541   Serial.print(S2[2]);
542   Serial.print(S2[1]);
543   Serial.println(S2[0]);
544   Serial.print("Cy/Bw = ");
545   if(!l){
546     Serial.println(CyBwOUT);
547   }
548   else{
549     Serial.println("-");
550   }
551   Serial.print("Ov = ");
552   if(!l){
553     Serial.println(OvOUT);
554   }
555   else{
556     Serial.println("-");
557   }
558   Serial.print("Z = ");
559   Serial.println(Z2);
560   Serial.println();
561
562   Serial.print(A[3]);
563   Serial.print(A[2]);
564   Serial.print(A[1]);
565   Serial.print(A[0]);
566   Serial.print(" NOR ");
567   Serial.print(B[3]);
568   Serial.print(B[2]);
569   Serial.print(B[1]);
570   Serial.print(B[0]);
```

```
570   Serial.print(B[0]);
571   Serial.print(" = ");
572   Serial.print(S3[3]);
573   Serial.print(S3[2]);
574   Serial.print(S3[1]);
575   Serial.println(S3[0]);
576   Serial.println();
577   Serial.print("Cy/Bw = ");
578   if(!1){
579     Serial.println(CyBwOUT);
580   }
581   else{
582     Serial.println("-");
583   }
584   Serial.print("Ov = ");
585   if(!1){
586     Serial.println(OvOUT);
587   }
588   else{
589     Serial.println("-");
590   }
591   Serial.print("Z = ");
592   Serial.println(Z3);
593   Serial.println();
594
595 }
```

```
596 void printNumbers(){
597   Serial.print("A = ");
598   Serial.println(A[0] + 2*A[1] + 4*A[2] + 8*A[3]);
599 
600   Serial.print("B = ");
601   Serial.println(B[0] + 2*B[1] + 4*B[2] + 8*B[3]);
602 
603   Serial.print(A[0] + 2*A[1] + 4*A[2] + 8*A[3]);
604   Serial.print( " + " );
605   Serial.print(B[0] + 2*B[1] + 4*B[2] + 8*B[3]);
606   Serial.print(" = ");
607   Serial.println(S0[0] + 2*S0[1] + 4*S0[2] + 8*S0[3] + 16*Cylbit);
608 
609   Serial.print("Carry = ");
610   Serial.println((Cylbit)?"TRUE":"FALSE");
611   Serial.println("");
612   Serial.print(A[0] + 2*A[1] + 4*A[2] + 8*A[3]);
613   Serial.print( " - " );
614   Serial.print(B[0] + 2*B[1] + 4*B[2] + 8*B[3]);
615   Serial.print(" = ");
616   if(!Bwlbit){
617 
618     Serial.println(S1[0] + 2*S1[1] + 4*S1[2] + 8*S1[3]);
619     Serial.println("BORROW = FALSE");
620   }else{
621     Serial.print("-");
622     Serial.println(!S1[0] + 2*!S1[1] + 4*!S1[2] + 8*!S1[3] + 1);
623 
624     Serial.println("BORROW = TRUE");
625   }
626   Serial.println("");
627 }
```