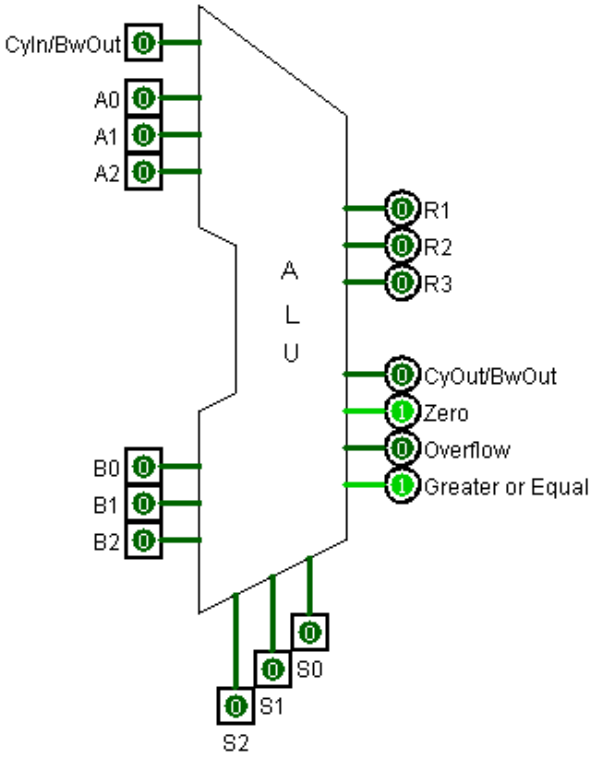
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

17/18v

Lógica e Sistemas Digitais

ALU



Realizado pelo grupo 8:

Paulo Rosa	**44873**
Gonçalo Santos	**44587**
Diogo Gouveia	**44884**

Docente: José Paraizo

29/3/18

Enunciado

Introdução:

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:

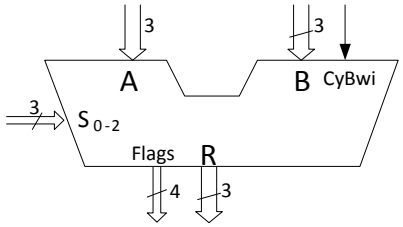


Figura 1 – ALU.

- Tem por entradas dois operandos A e B, de três 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 três bits, no mesmo domínio que os operandos;
- Executa uma de oito operações (ADC, SBB, INC, DEC, ANL, NOT, RCL, ASR), seleccionadas pelos três bits S_2, S_1 e S_0, conforme indicado na Tabela 1;

S_{0-2}	Operação		Sigla	Z	CyBwo	OV	GE
000	Adição com carry	R = A + B+CyBwi	ADC	•	•	•	–
001	Subtracção com borrow	R = A □ B □CyBwi	SBB	•	•	•	•
010	Incremento	R = A + 1	INC	•	•	•	–
011	Decremento	R = A □ 1	DEC	•	•	•	–
100	AND Lógico bit a bit	R = A&B	ANL	•	–	–	–
101	NOT Lógico bit a bit	R = ~A	NOT	•	–	–	–
110	*Rotate Carry Left*	R = A<<1	RCL	•	•	•	–
111	*Arithmetic Shift Right*	R = A>>B_{0-1}	ASR	•	•	•	–

Tabela 1 – Funcionalidade da ALU.

A ALU implementa quatro indicadores binários (*flags*), sendo dois deles qualitativos e os outros de excesso de domínio:

Z	*Zero*	Activo, quando a operação realizada tem como resultado o valor zero.
CyBwo	*Carry/Borrow out*	Representa *Carry* na adição e *Borrow* na subtracção, estando activo quando o resultado excede o domínio, entendido em código natural. Na operação *SHR* 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, entendido em código dos complementos. Nas operações SHR e RCL quando o bit de sinal de R difere do bit de sinal de A.
GE	*Greater or Equal*	quando activo indica que A é maior ou igual a B, tomando os operandos como valores inteiros com sinal.

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 ADC, SBB, INC e DEC enquanto o módulo Lógico/Shift realiza as operações ANL, NOT, RCL e ASR

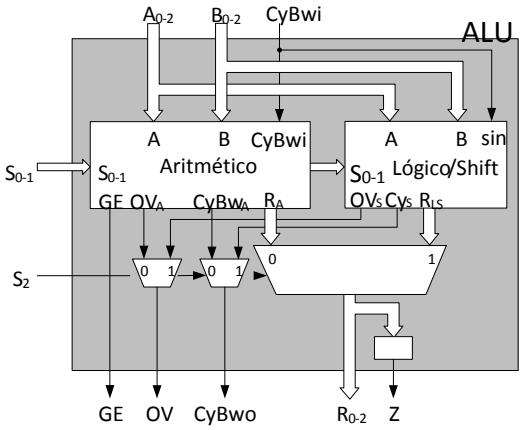


Figura 2 – Diagrama de blocos da ALU

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

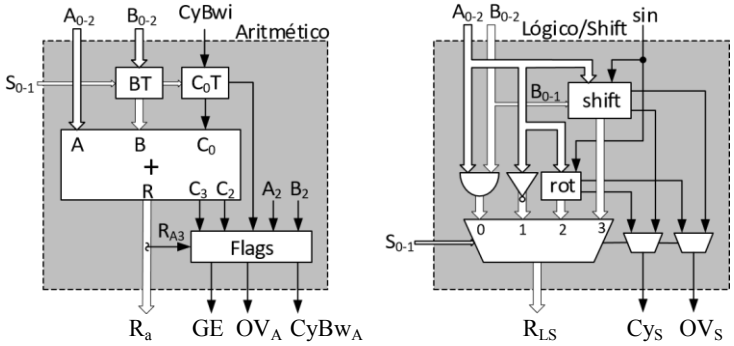


Figura 3 – Diagramas de blocos dos módulos Aritmético e Lógico/Shift.

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, subtracção, incremento e decremento.
- O elemento Flags, implementa os sinais binários OV_A, $CyBw_A$ e GE. A sua implementação poderá recorrer a outros sinais disponíveis na estrutura, caso o aluno os considere preferíveis.

Módulo Lógico:

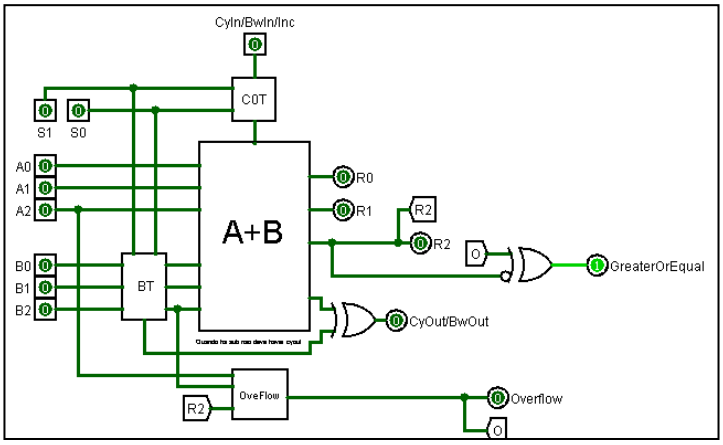
- A arquitectura preconizada recorre à multiplexagem do resultado dos vários operadores, sendo estes realizados por módulos independentes entre si.
- As operações de deslocamento (ASR e RCL) são realizadas por um Barrel Shifter, em que, na operação ASR o número de bits a deslocar é determinado pelos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

1. Introdução:

A ALU (Arithmetic Logic Unit) é um circuito digital que realiza operações aritméticas e lógicas e é uma construção base para a formação do CPU (Central Processing Unit). Neste projeto, tem 8 operações: 4 aritméticas, no módulo aritmético, e 4 lógicas, no módulo lógico.

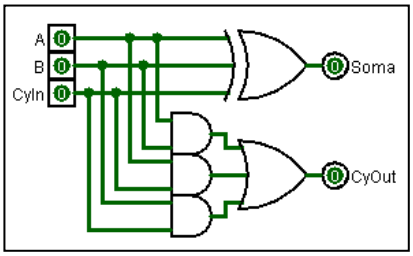
2. Realização do projecto:

2.1 Módulo aritmético:

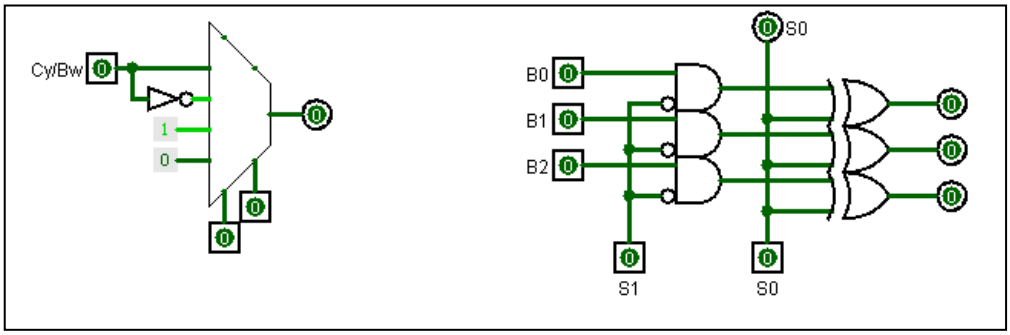


2.1.1 Operações:

No módulo aritmético temos 4 operações: A + B+CyBwi, A-B-CyBwi, A+1 e A-1. Que são realizadas a partir do nosso Somador(sequência de somas) que é feito a partir do circuito Soma



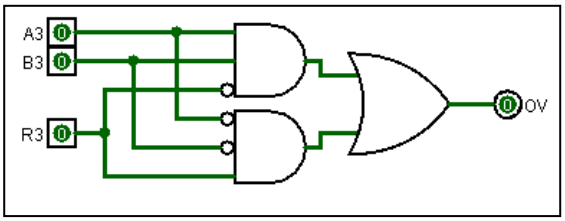
Para acomodar o Carry/BwIn com as diferentes operações, foi criado o C0T e o BT



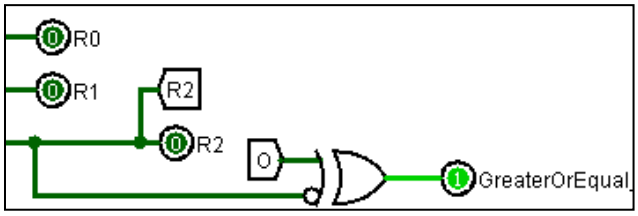
Em que se S = 0, Pode haver carry e a soma é normal, se S = 1; há o inverso de carry e o simétrico de B, se S = 2, Há carry (Inc de +1) e o B não pode afetar esta operação e se S = 3, do C0T sai 0 e do B sai tudo a valor 1, o que produz o efeito de A-1.

2.1.2 Flags:

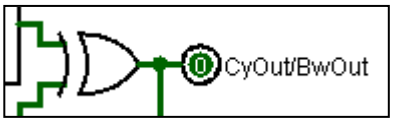
Ocorre overflow quando a adição ou subtração de dois números excede a magnitude pela qual eles podem ser apresentados com o número desejado de bits. E isto acontece quando o valor do bit de maior peso dos operandos for diferente do bit de maior peso do resultado. De tal forma foi criado o nosso overflow:



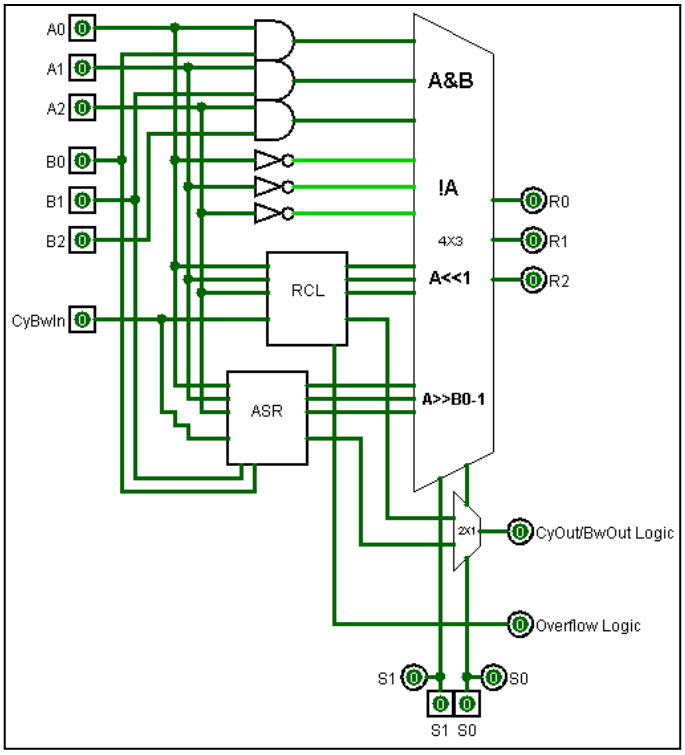
Ocorre GreaterOrEqual(em inteiros com sinal) quando A>B. E se A-B = R, entao A>B. E se A-B = -R então é porque A<B. Com isto em mente foi criado o GE.



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 do somador um XOR para produzir este efeito.

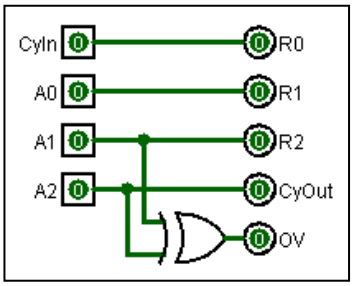


2.2 Módulo Lógico:



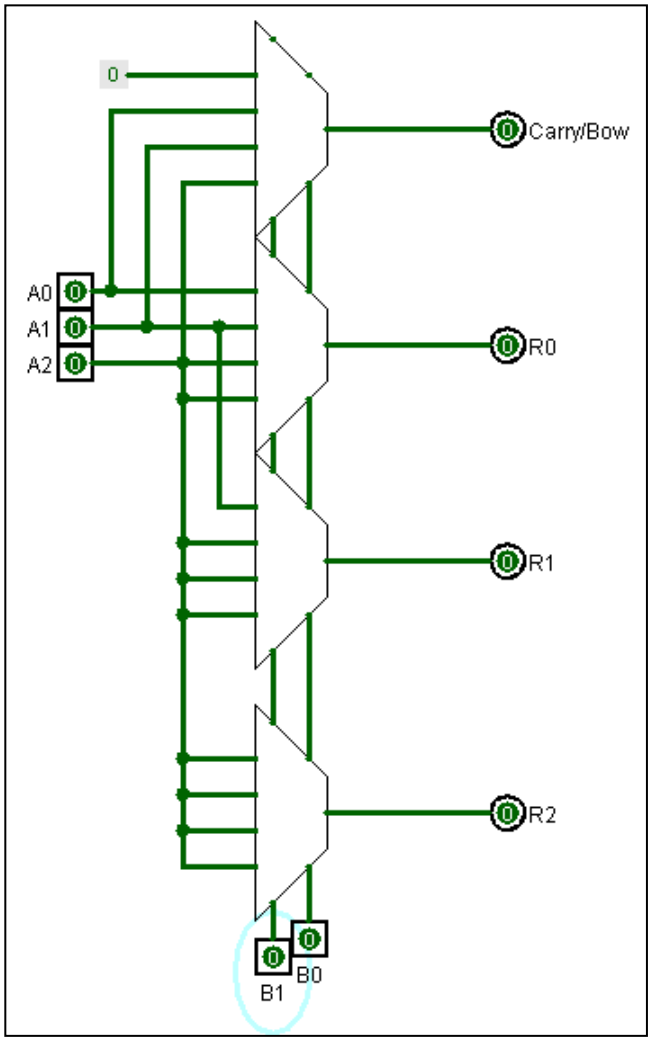
2.2.1 Operações:

Aqui encontramos as 4 operações do módulo lógico: A&B, !A, A<<1, A>>B0-1. E que são geridas pelo MUX 4X3. Com S = 0, a operação A&B é realizada com *ands*, e a se S = 1, há *nots* para formar o !A, e o *RotateCarryLeft* foi realizado da sequinte forma:

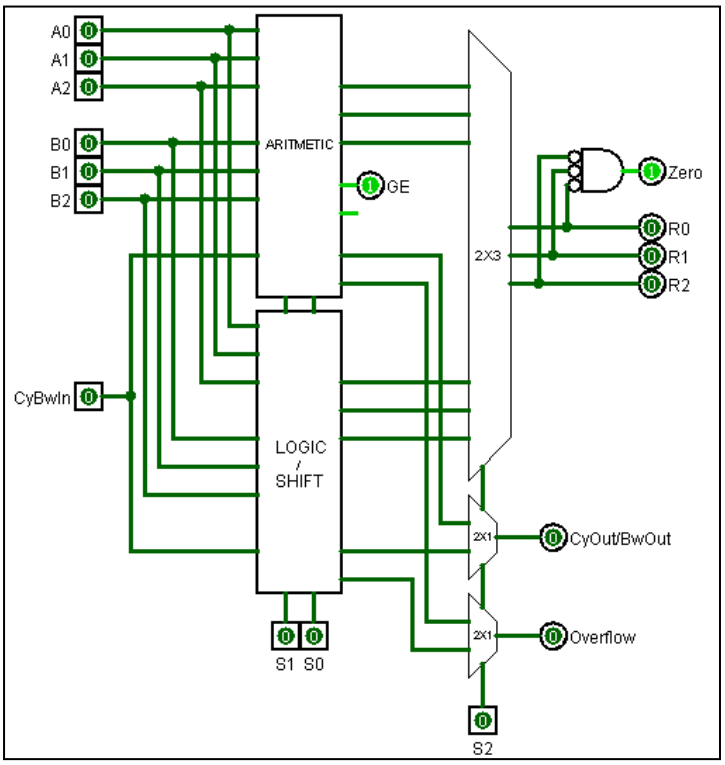


Sendo a transição de um só bit, há uma transmissão/shift direto nesta operação. E ocorre overflow se o bit sinal de chegada é diferente do de partida.

No *ArithmethicShiftRight*, são usados vários MUX's e ligações para satisfazer as influências do B1 e B0 nesta operação. Não havendo *serial in* porque houve um erro no enunciado. E não há overflow porque o bit sinal do 'A' mantem-se.



2.3 Aritmethic + Logic:



Por fim, temos um MUX 2X3 e dois MUX 2x1 para gerir os R's e para gerir os CyOut's e Overflows respectivamente.

2.4 Em CUPL:

```
/* *************** INPUT PINS *********************/
PIN [1..3] = [A2..0];
PIN [4..6] = [B2..0];
PIN [7..9] = [S0..S2];
PIN 10 = CyBwin;

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

PIN 14 = Zero;
PIN 15 = CyBwout;
PIN 16 = OverFlow;
PIN 17 = GE;
PIN 18 = R0;
PIN 19 = R1;
PIN 20 = R2;
PIN 21 = C1;
PIN 22 = C2;
PIN 23 = C3;


Add = !S2 & !S1 & !S0;
Sub = !S2 & !S1 & S0;
IncA = !S2 & S1 & !S0;
DecA = !S2 & S1 & S0;
And = S2 & !S1 & !S0;
Not = S2 & !S1 & S0;
Rcl = S2 & S1 & !S0;
Shr = S2 & S1 & S0;

/* Aritmetico */

C0 = Add & CyBwin   #   Sub & !CyBwin   #   IncA   #   DecA & 'b'0; /* aka acomodador de funcao '

[BT0..2] = ([B0..2] & S1) $ S0;

[C1..3] = [A0..2]&[BT0..2] # [A0..2]&[C1..3] # [BT0..2]&[C1..3]; /*os carrys ver se e 2 ou 3*/

[RA0..2] = [C1..C3] $ [A0..2] $ [BT0..2];

/*Flags*/

CyBwoutA = C3 $ S0;

OvA = A2&B2&!R2 # !A2&!B2&R2;

GE = (!R2) $ OvA;


/* Logico */


[ASR0..2] = !B0&!B1&[A0..2] # !B0&B1&[A1,A2,CyBwin] # B0&!B1&[A1, A2, A2] # B0&B1&[A2, A2, A2];

CyLRight = !B0&B1 & A0   #   B0&!B1 & A1   #   B0&B1 & A2;
```

```
RCL0 = CyBwin;
RCL1 = A0;
RCL2 = A1;
CyLLeft = A2;

AEB0 = A0&B0;
AEB1 = A1&B1;
AEB2 = A2&B2;


[RL0..2] = And & ([AEB0..AEB2])    #    Not&(![A0..2])    #    Shr&([ASR0..2])  #    Rcl&([RCL0..2]);


/*flags*/
OVL = A1 $ A2;
CyL = !S0 & CyLLeft    #    S0 & CyLRight;


/*tudo junto*/

RR0 = !S2&RA0 # S2&RL0;
RR1 = !S2&RA1 # S2&RL1;
RR2 = !S2&RA2 # S2&RL2;

[R0..2] = [RR0, RR1, RR2];

CyBwout = !S2&CyBwoutA # S2&CyL;
OverFlow = !S2&OvA # S2&OVL;

Zero = !R0 & !R1 & !R2;
```

3. Conclusões:

Em suma, todos os objetivos foram realizados. Tivemos que investigar e aprender novos conceitos. Nós entendemos a lógica do exercício e apreciamos a sua realização. E esperemos aplicar estes conceitos quando construirmos o CPU.