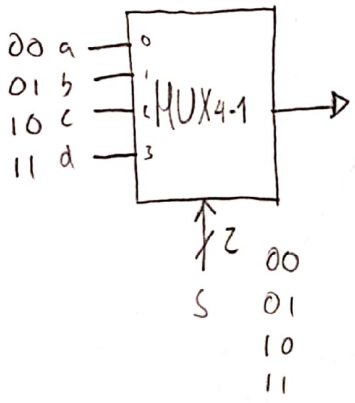
Práctica 01

Objetivo global práctica 01

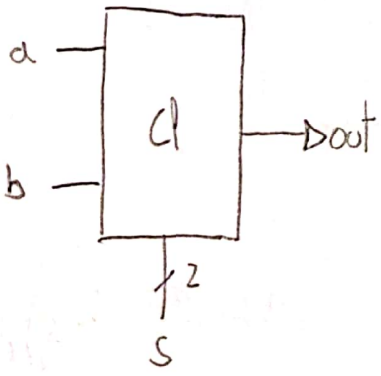
Realizar una ALU de 4 bits capaz de realizar sumas y restas en C2 y generar los bits de flags asociados, así como varias operaciones lógicas

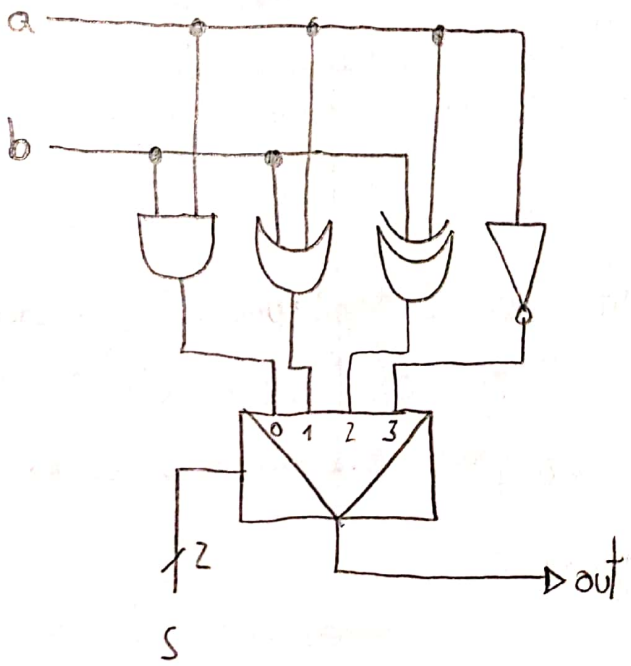
Objetivo 1



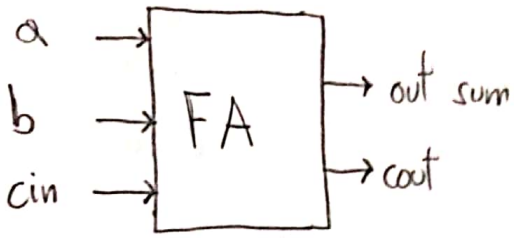
S => seleccion (entrada selección)

Objetivo 2.1 Celda lógica



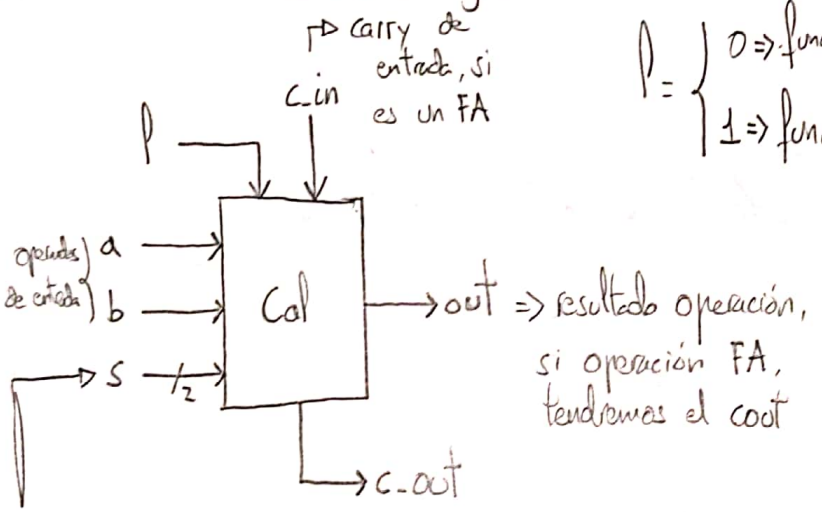


Objetivo 2.2 Full adder



Concatenación => podemos usar un operando de concatenación, útil para los Full adder

Objetivo 2.3. Celda aritmético lógica



p bit de selección

operaciones

p = { 0 => funcionará como un FA (aritméticas)
p = { 1 => funcionara como un CL (lógicos)

=> resultado operación, si operación FA, tendremos el cout

Si es operación lógica podría determinar si es AND, OR XOR, NOT

↳ Solo tiene sentido cuando sea una operación aritmética

Mux2_1

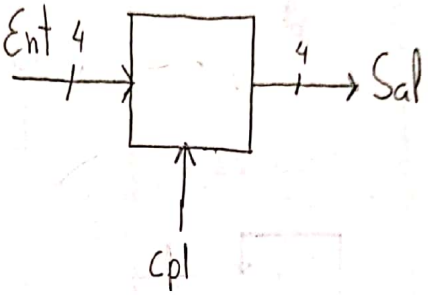
```
module mux2_1(output wire out, input wire a, input wire b, input wire s);
    assign out= s ? b:a;
    // operación condicional de C, Sintaxis [condición ? valor_si_cierta : valor_si_falsa]
endmodule
```

Objetivo 3. Complemento

En función de una señal de control cpl, de forma que podamos dejar pasar un dato sin modificar o hacer su C1. Elaborar el módulo con el prototipo siguiente:

```
Compl1(output wire [3:0] Sal, input wire [3:0] Ent, input wire cpl);
```

PISTA: mirar la implementación del MUX2_1. El operador condicional as puede ayudar



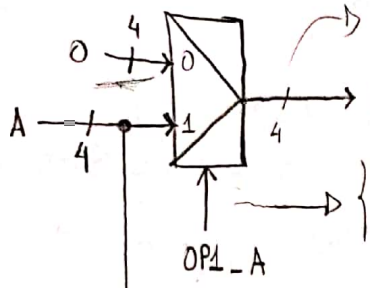
Objetivo 4. Camino de datos y operación de la ALU.

	L	ALUop	R	zero	carry	sign
aritmética	0	00	A+B	1 si (R=0)	1 si (A+B) tiene acarreo	Bit más significativo de R
	0	01	A-B	1 si (R=0)	1 si (A-B) tiene acarreo	Bit más significativo de R
	0	10	C2(A)	1 si (R=0)	1 si C2(A) tiene acarreo	Bit más significativo de R
	0	11	C2(B)	1 si (R=0)	1 si C2(B) tiene acarreo	Bit más significativo de R
lógica	1	00	A and B	1 si (R=0)	no importa	no importa
	1	01	A or B	1 si (R=0)	no importa	no importa
	1	10	A xor B	1 si (R=0)	no importa	no importa
	1	11	C1(A)	1 si (R=0)	no importa	no importa
		códigos de operación				

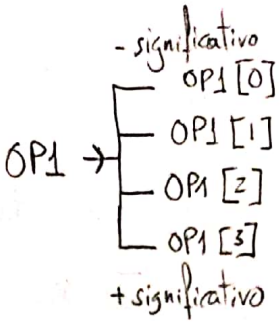
A-B => la implementaremos, sumandole el C2 del sustraendo al primer operando

LD = A+(-B) = A+C2(B) = A+C1(B)+1; Op1=A, Op2=C1(B), Cin_0 = 1

Mux 2-1

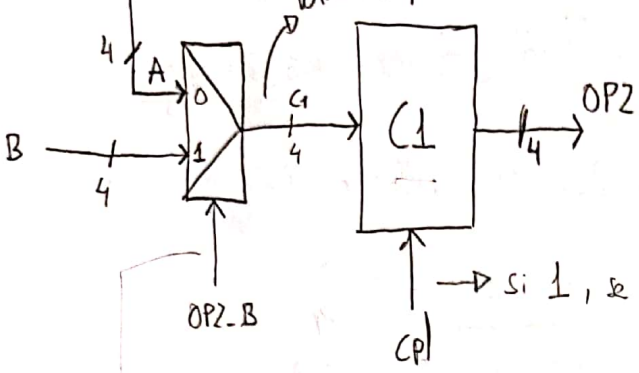


0, dejará pasar 0 a OP1
1, dejará pasar A a OP1



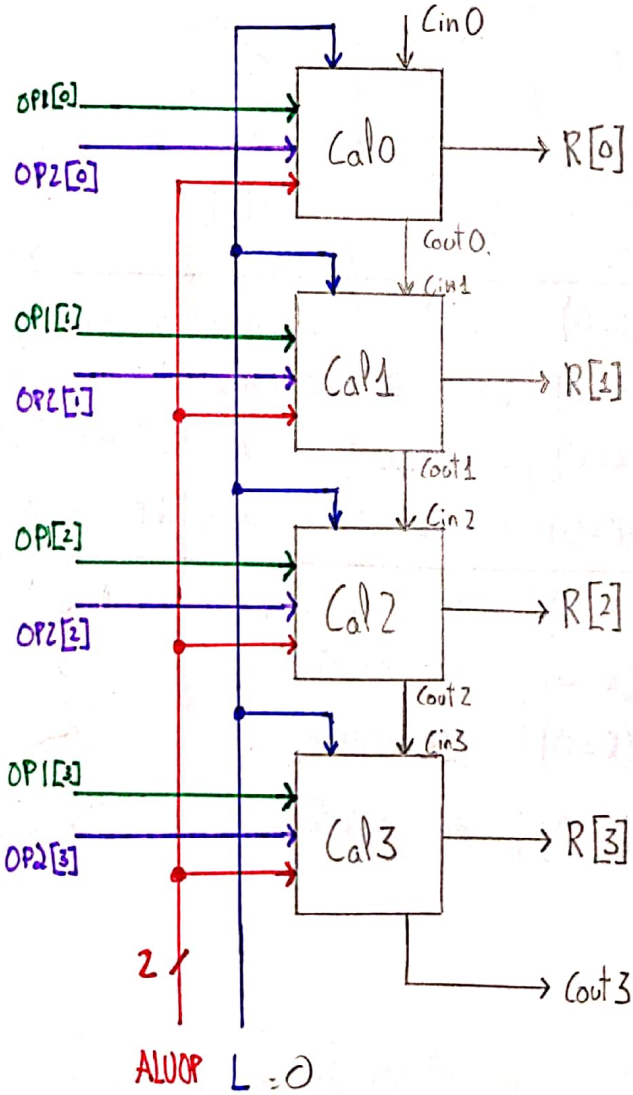
II
OP2

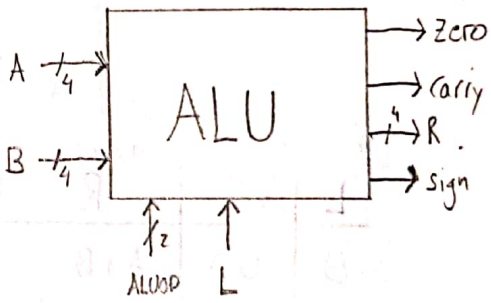
módulo del objetivo 3



Si 1, se complementará la entrada del módulo C1

0, dejará pasar A a C1
1, dejará pasar B a C1





+

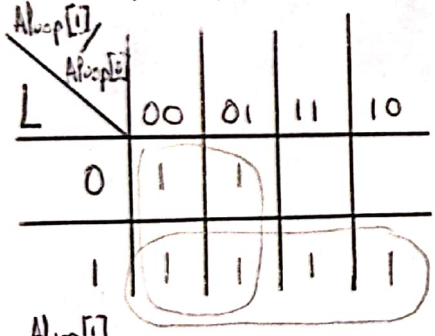
-

Objetivo 5. Diseñar funciones lógicas

Hay que diseñar las señales de control op1_A, op2_B, cpl y CinO, que dependerán de las entradas L, AluOp[1] y AluOp[0] que definirán la operación a ejecutar por la ALU.

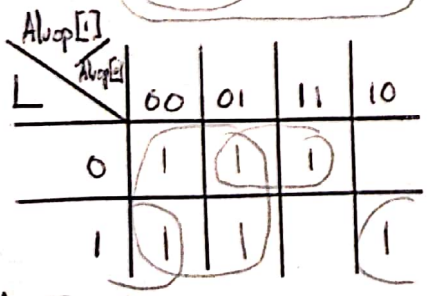
No olvidar las señales zero, carry y sign.

Oper	L	AluOp[1]	AluOp[0]	op1_A	op2_B	cpl	CinO
A+B	0	0	0	1	1	0	0
A-B	0	0	1	1	1	1	1
C2(A)	0	1	0	0	0	1	1
C2(B)	0	1	1	0	1	1	1
A and B	1	0	0	1	1	0	0
A or B	1	0	1	1	1	0	0
A xor B	1	1	0	1	1	0	0
C1(A)	1	1	1	1	0	0	0



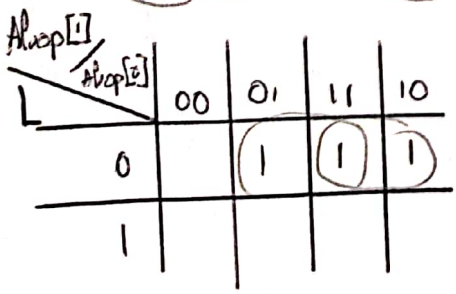
$\Rightarrow op1\_A = \overline{AluOp[1]} + L$

assign op1_A = (~AluOp[1])|L



$\Rightarrow op2\_B = \overline{AluOp[1]} + \overline{L} \cdot AluOp[0] + L \cdot \overline{AluOp[0]}$

assign op2_B = ((~AluOp[1]))|((~L) & AluOp[0])|(L & (~AluOp[0]))



$\Rightarrow cpl = CinO = \overline{L} \cdot AluOp[0] + \overline{L} \cdot AluOp[1]$

assign cpl = ((~L) & AluOp[0])|((~L) & AluOp[1])

=

CinO