Subtraction in binary → easier to add a negative number using 2's complement representation

To convert any binary number from positive to negative, (or the other way):
- Invert all bits & add 1
- e.g. $7: 0111_2 \longrightarrow 1000+1 = 1001$
- Simple rule: copy from right up to & including 1st 1 invert after that
- e.g. $1001 \rightarrow 0111$

2's complement (4-bit)	
0	0000
1	0001
⋮	
7	0111
-8	1000
-7	1001
⋮	
-2	1110
-1	1111

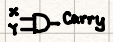
The adder in the processor

Consider binary addition:

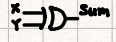
```
  5    0 1 0 1   X
+ 6  + 0 1 1 0   Y
 11    S
```

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Carry: $X \cdot Y$



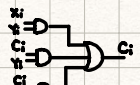
Sum: $x\bar{Y} + \bar{X}Y = X \oplus Y$



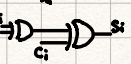
The full-adder □ has a carry-in & carry-out

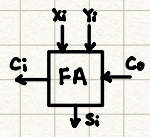
C_i	X_i	Y_i	S_i	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$C_o = X_i Y_i + C_i Y_i + C_i X_i$



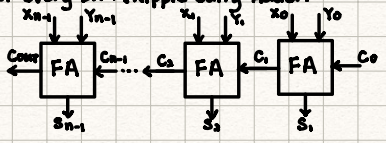
$S_i = C_i \oplus X_i \oplus Y_i$





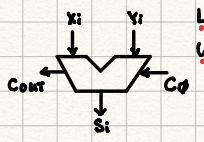
To build a multi-bit adder (adds multi-bit #s),
use one FA for every bit: (Ripple Carry Adder)

```
  X
+ Y
  S
```



Verilog for a 16-bit adder

```
module adder16 (Co, X, Y, S, Cout)
    parameter  n = 16;
    input   Co;
    input   [n-1:0] X, Y;
    output  [n-1:0] S;
    output  Cout;
                        add [or, |]
    assign {Cout, S} = X + Y + Co;
endmodule
```



Arithmetic
Logic → ALU
Unit
multiplication, division

ALU also shifts binary #s left or right by a number of bits
e.g. 0111 → 1110 shift left by 1 bit multiplies by 2
shift right by 1 bit divide by 2

Consider n-bit number: $b_{n-1} b_{n-2} \cdots b_2 b_1 b_0$

Most significant bit (MSB) Least Significant Bit (LSB)