Week 2

Binary numbers – any positive integer x can be represented in binary with 2^n bits, where $n = \lceil \log_2 x \rceil$

0	0
1	1
2	10
3	11
4	100
5	101
⋮	

N bits → **2^N possibilities**

In general, for any base b:

$$(x_n x_{n-1} \cdots x_0)_b = \sum_{i=0}^{n} x_i \cdot b^i$$

Binary → Decimal

$$101 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$
$$= 4 + 1$$
$$= 5$$

Binary addition

Break it down into:

1. Half adder (2 bits)
2. Full adder (3 bits)
3. Adder (arbitrary number of bits)

Decimal → Binary

$87_{decimal} = 64 + 16 + 4 + 2 + 1$

$= 0\,1\,0\,1\,0\,1\,1$

2's complement

$-x \rightarrow 2^n - x$

Addition with 2's complement is "free"!

Representing Negative Numbers

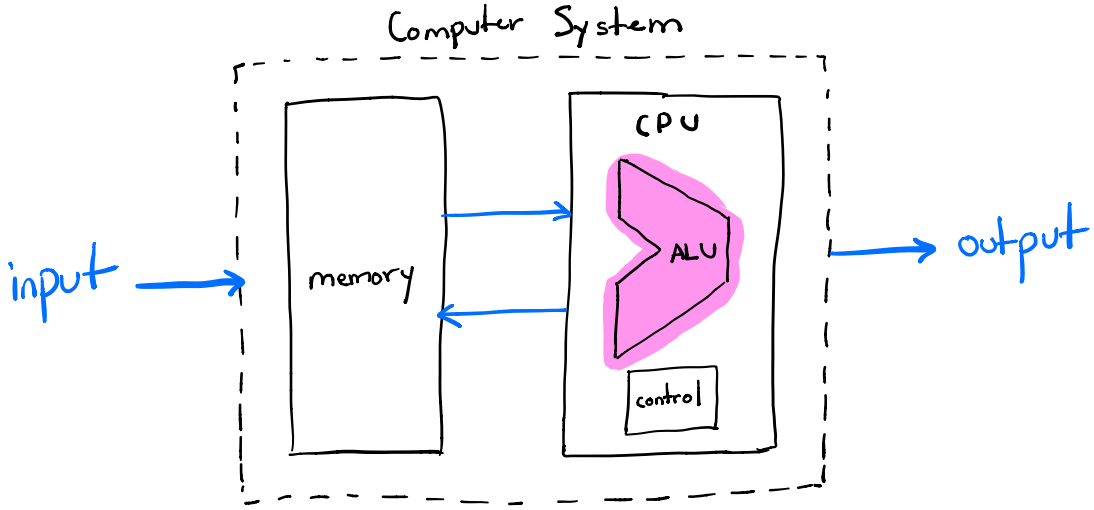
Options:

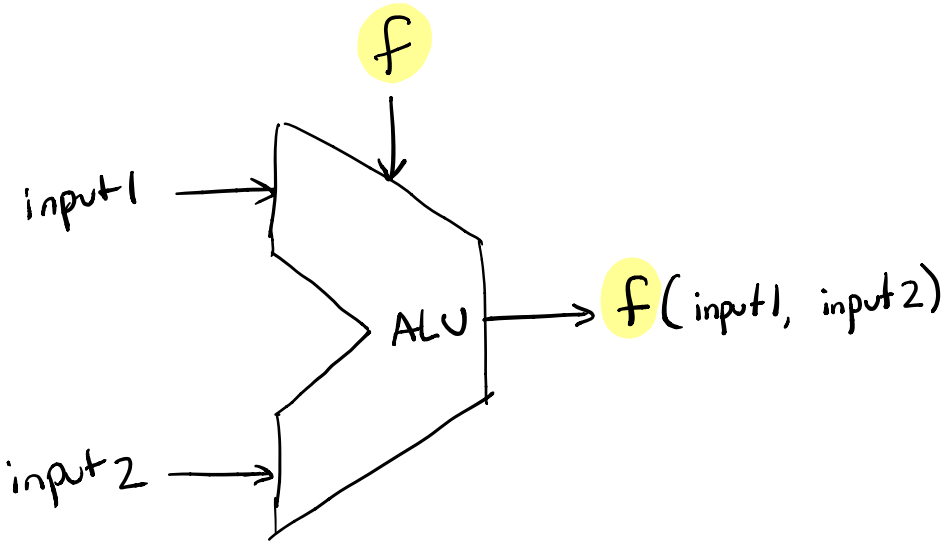
- signed magnitude representation (SMR)
- 1's complement
- 2's complement ← what most computers use today!

Negation

invert all the bits of x and add 1!

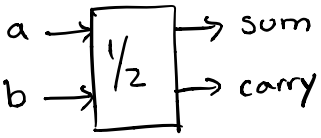
Von Neumman Architecture

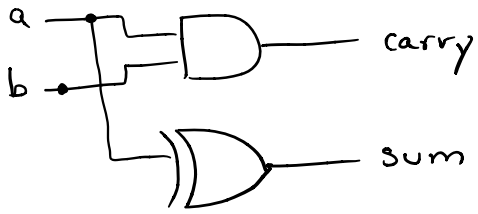




f is one out of a family of pre-defined arithmetic and logical functions.

Half adder

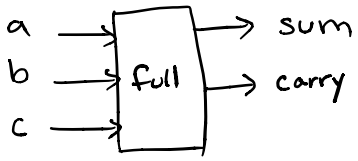


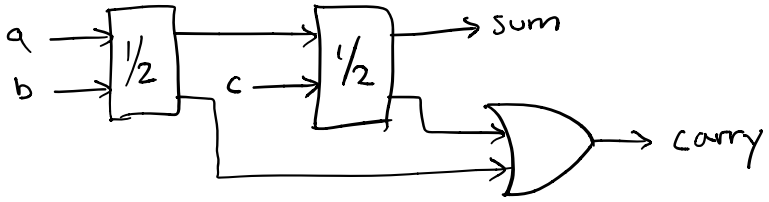


a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

xor! and!

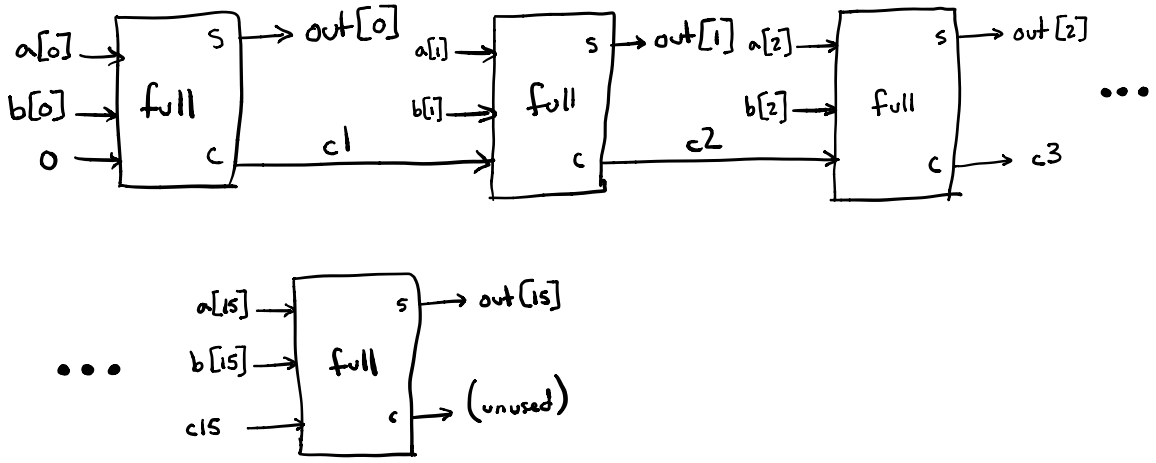
Full adder



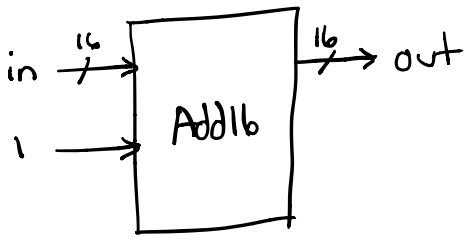


a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Add16

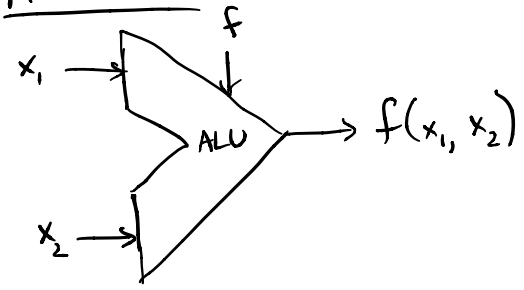

a[0]
b[0]
0
full
s
c
out[0]
c1
a[1]
b[1]
full
s
c
out[1]
c2
a[2]
b[2]
full
s
c
out[2]
c3
a[15]
b[15]
c15
full
s
c
out[15]
(unused)

Inc16


in
16
1
Add16
16
out

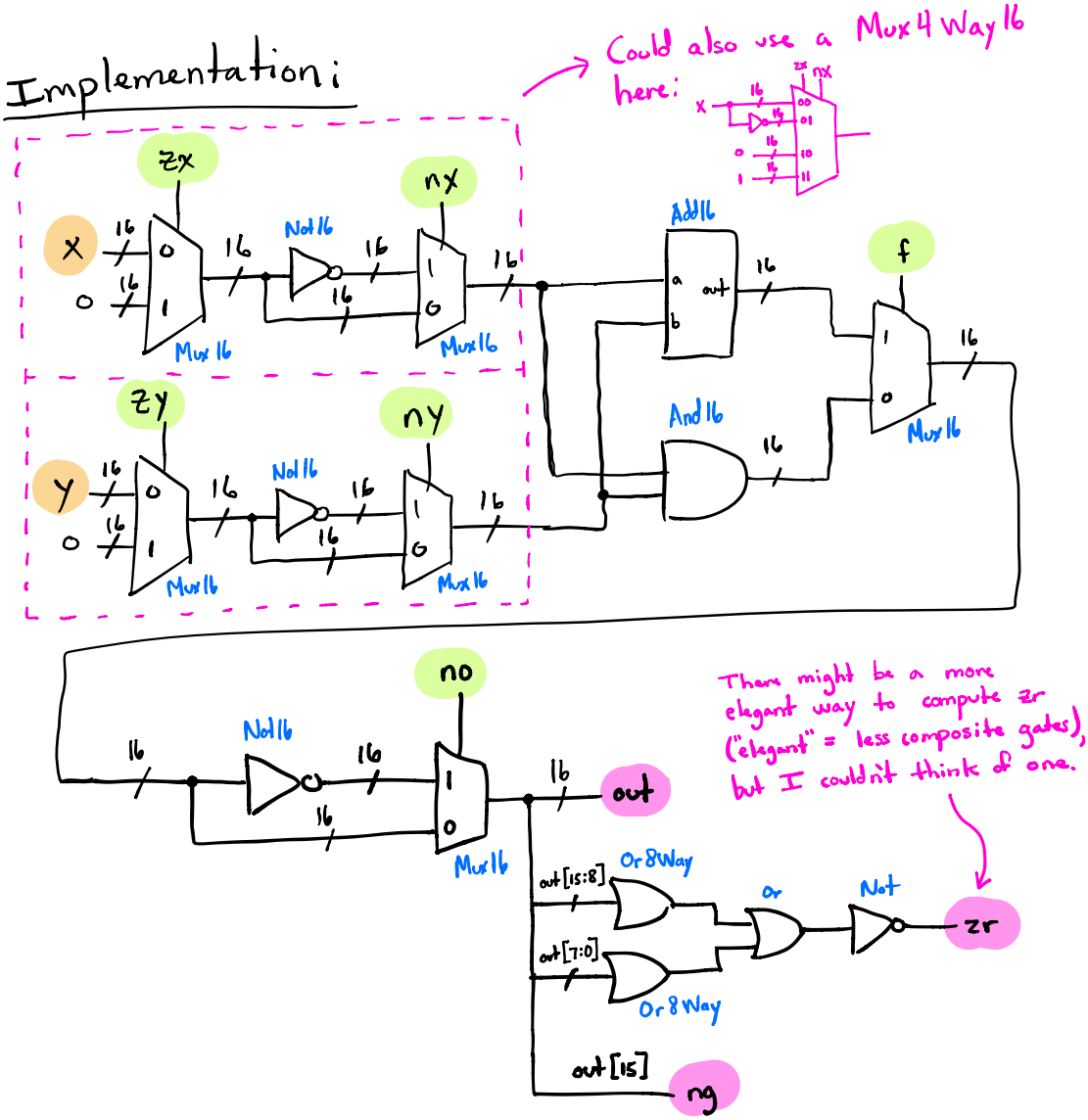
ALU (arithmetic logic unit)

Abstraction:



Computes one of several functions on the two inputs, and outputs the result.

Implementation:



How does y−x work?

zx = 0 zy = 0 f = x+y → out = ~(x + ~y)
nx = 0 ny = 1 no = 1

Assume n bits.

1) $\sim x = 2^n - 1 - x$

$$x + \sim y = x + 2^n - 1 - y$$

$$\sim(x + \sim y) = 2^n - 1 - (x + 2^n - 1 - y)$$

$$= 2^n - 1 - x - 2^n + 1 + y$$

$$= y - x$$

Example:

x = 1110 (-2)
y = 0101 (5) → 7

↓

x = 1110
y = 1010

x+y = 1000
!(x+y) = 0111 → 7

Perspective

Are the chips we built standard? Yes, except for the ALU, which is greatly simplified compared to typical ALUs. This is in line with a major theme of the course, which is **simplicity** and **elegance**.

e.g. multiplication and division are deferred to software. This trades **performance** and **efficiency** for **simplicity**

Some efficiency/optimization discussion

- Can make the 16-bit adder "faster" (in terms of worst-case propagation delay) by using a carry-lookahead technique → compute the carry bits independently
  - Since propagation delay in real implementations is not modeled in this course, optimizations/efficiency considerations like this are a "dont-care"