Unit 2.1 - Binary Numbers

Binary digit:	b_4	b_{4-1}	b_{4-2}	...	b_1	b_0
Value:	2^4	2^{4-1}	2^{4-2}		2^1	2^0

$\Rightarrow$ value $= \sum_i^1 b_i \cdot 2^i$

Max value with k bits: $1+2+4+\ldots+2^{k-1} = 2^k - 1$

With 8 bits, can represent 256 different binary numbers $\Rightarrow 2^8 = 256$

0000 0000
0000 0001
...
0111 1111
1000 0000
...
1111 1111

$\Rightarrow$ reserving last digit on left for negative numbers leaves 127 possible numbers to be used with 8 bits

$\hookrightarrow \frac{256-2}{2} = 127$

Decimal $\rightarrow$ Binary

$87_{decimal} = 64 + 16 + 4 + 2 + 1$
$= 2^6 + 2^4 + 2^2 + 2^1 + 2^0$

$\hookrightarrow$ start with largest factor 2

$\Rightarrow$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	1
+	64	0	16	0	4	2	1

Base 10

e.g. 789_{10}

$7 \cdot 10^2$ $8 \cdot 10^1$ $9 \cdot 10^0$

2020-09-07

Unit 2.2 - Binary Addition

```
    1 1 1
  0 0 0 1 0 1 0 1
+ 0 1 0 1 1 1 0 0
  0 1 1 1 0 0 0 1
```

↖ carry the one

Overflow

```
1   1 1 1
  1 0 0 1 0 1 0 1
+ 1 1 0 1 1 1 0 0
1 0 1 1 1 0 0 0 1
```

- overflow bit will be truncated
- ∴ in a computer, cannot do real integer addition

Half Adder

carry - □ 0

```
? ? ? ? ? a ? ?
? ? ? ? ? b ? ?
```

□ - sum

↳ carry over a must = ∅

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

↳ truth table for chip

Full Adder

carry c

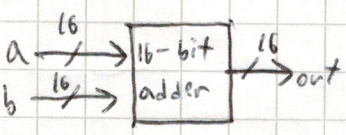
```
? ? ? ? ? a ? ?
? ? ? ? ? b ? ?
```

sum

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

16-Bit Adder

⇒ 15 full adders with one half-adder for right most bit



2025-04-07

Unit 2.3 - Negative Numbers

Sign Bit

0000 →	0	→ 0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-0	-8 (8)
1001	-1	-7 (9)
1010	-2	-6 (10)
1011	-3	-5 (11)
1100	-4	-4 (12)
1101	-5	-3 (13)
1110	-6	-2 (14)
1111	-7	-1 (15)

- inelegant
- messy hardware config.

Two's Complement

- represent negative number $-x$ using the positive number: $2^4 - x$
- positive numbers in range $0 \ldots 2^{4-1} - 1$
- negative numbers in range $-1 \ldots -2^{4-1}$

Addition in 2's Complement

$$\begin{array}{r} -2 \\ +-3 \\ \hline \end{array} \Rightarrow \begin{array}{r} 14 \\ +13 \\ \hline \end{array} \Rightarrow \begin{array}{r} 1110 \\ +1101 \\ \hline 11011 \end{array}$$

$11011 = 27_{ten}$

$1011 = 11_{ten}$

$11 \Rightarrow \boxed{-5}$ in 2's complement, correct answer

- Representation is modulo 2^4
  Addition is modulo 2^4

Unit 2.3 - Negative Numbers (continued)

Computing $-x$ → solving this allows us to subtract: $y - x = y + (-x)$
↳ addition is solved

input: x
output: $-x$ (in 2's complement)

idea: $2^n - x = 1 + (2^n - 1) - x$

$2^n - 1 = 1111\ 1111_2$ → ~~just flipping~~ $(2^n - 1) - x$ is just flipping each bit

eg. input: 4 (0100)
output: -4 → 2's complement = 12_{10}, 1100

```
 1111
-0100
------
 1011
```

```
 1011
+   1
------
 1100
```

→ to add 1, start at right most bit, flip 1s to 0s and stop when you flip a 0 to 1

eg
```
 1111 1111
-1010 1100
----------
 0101 0011
```
→ then add one

☆ Using two's complement and method for addition, can have a method for subtraction

to increment binary number by 1:

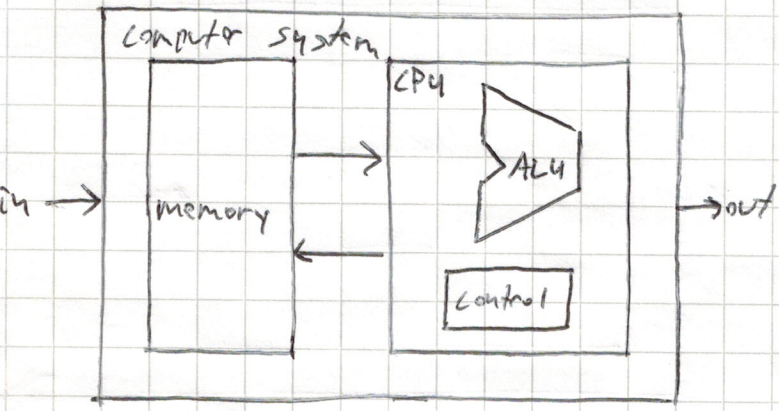
Half Adder (a = in[0], b = true, out = out[0], carry = c_1) → pipe carry forward
Full Adder (a = in[1], b = false, c = c_1, out = out[1], carry = c_2
⋮
Full Adder (a = in[15], b = false, c = c_{15}, out = out[15], carry = c_{16})

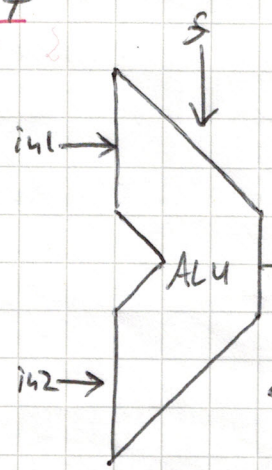
↳ first b input is true, subsequent b inputs are false

2020-04-07

Unit 2.4 - Arithmetic Logic Unit

Von Neumann Architecture

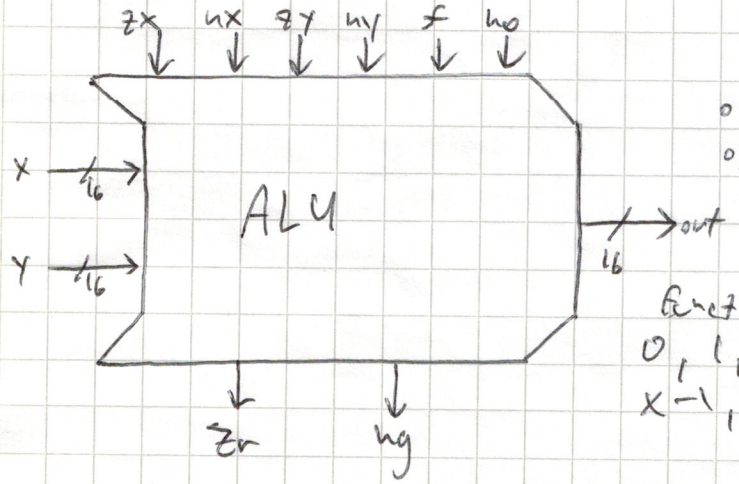




- ALU computes a function on the two inputs and outputs the result
- f is one of a family of predefined arithmetic and logical functions
- the operations an ALU can do is a hardware/software trade off

arithmetic operations: int addition/multiplication/division...
logical operations: And, or, Xor...

The Hack ALU



inputs: two 16-bit 2's complement value
output: one 16-bit 2's complement value
- function to compute set by 6 1-bit control inputs
- Also outputs two 1-bit control outputs

functions:
0, 1, -1, x, y, !x, !y, -x, -y, x+1, y+1, x-1, y-1, x+y, x-y, y-x, x&y, x|y

Hack ALU operation

zx	nx	zy	ny	f	no	out
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	out(x,y)=

→ subsequent operations are done on processed inputs

Control outputs — zr and ng

if out==0 then zr=1, else zr=0
if out<0 then ng=1, else ng=0

} comes into play when building complete architecture