Patt-Ch2 Bits, Data Types, and Operations

2.1 Bit - Binary Digit

- 1: **presence** of a voltage, 0: **absence** of a voltage
- A collection of n bits has 2^n possible states.
- a way of code
  - ***e.g. ASCII Code*** *,total numbers less than 8bits*

2.2 Data Types

> data type: representation of information

2.2.1 Unsigned Integers

An n-bit unsigned int represents 2^n values: **0 - 2^n-1**

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

2.2.2 Signed Integers

2.2.2.1 sign-magnitude and 1's complement(反码)

Signed Integers

With n bits, we have 2^n distinct values.

- **assign about half to positive integers (1 through 2^{n-1}) and about half to negative (- 2^{n-1} through -1)**
- **that leaves two values: one for 0, and one extra**

Positive integers

- **just like unsigned – zero in *most significant* (MS) bit**
  00101 = 5

Negative integers

- **sign-magnitude – set MS bit to show negative, other bits are the same as unsigned**
  10101 = -5
- **one's complement – flip every bit to represent negative**
  11010 = -5
- **in either case, MS bit indicates sign: 0=positive, 1=negative**

2.2.2.2 Problems above

- +0 *00000* and -0 *10000(S-M)* or *11111(1's C)*
- complex arithmetic, even error

```
e.g.  sign-magnitude
      2+(-3)
      0 0 1 0   2
      1 0 1 1   -3
      -------
      1 1 0 1   -5
      ERROR

      1's complement
      4+(-3)
      0 0 1 1   4
      1 1 0 0   -3
      -------
      1 1 1 1   0
      ERROR
```

2.2.2.3 2's Complement(补码)

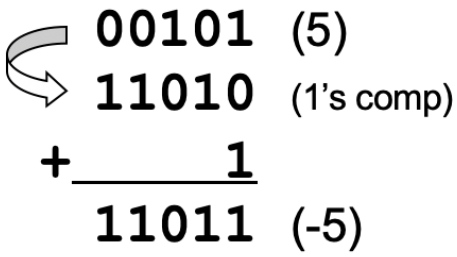
Design Purpose

- *Two's complement* **representation developed to make circuits easy for arithmetic.**
- for each positive number (X), assign value to its negative (-X),
  such that **X + (-X) = 0** with **"normal" addition**, **ignoring carry out**

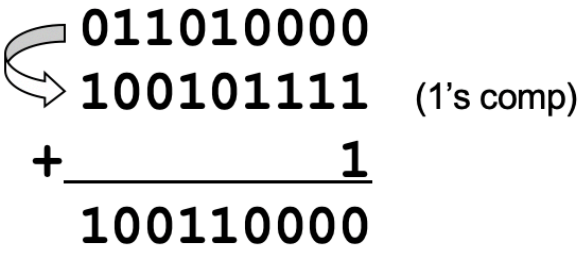
```
e.g.
      0 0 1 0 1   (5)
      1 1 0 1 1   (-5)
      ---------
      0 0 0 0 0   (0)
```

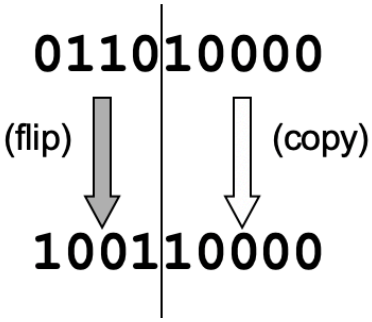
Representation

- Positive or 0: normal binary representation
- Negative:
  - start with positive number
  - flip every bit
  - add 1



- shortcut:
  - start with positive
  - copy bits from right to left until (and including) the first 1
  - Flip remaining bits to the left





Number Range

- MS(*most important*) bit is **sign bit**, weight **-2^{n-1}**
- Range: **$[-2^{n-1}, 2^{n-1}-1]$**
  - The most negative number (-2^{n-1}) has no positive counterpart.

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

以5和-5为例	SM	1's C	2's C
正数	0101	0101	0101
负数	1101	1010	1011

2.3 Floating Point

2.3.1 Representation

- IEEE754 standard
- **32-bit(64 bit for DOUBLE)** normalized floating-point number representation

	Sign	Exponent	Fraction
Bits	1	*8/11*	*23/52*
Rule	0 for `+`, 1 for `-`	unsigned int[1]	digits after point[2]

[1] 8 bits **unsigned integer** can represent [0,255].

exp [1,254] -> **subtract 127** real exp [-126, 127]

exp == 255(1111 1111) -> ∞

exp == 0 -> 0.fraction x 2^{-126}

for 64 bits **DOUBLE**,

exp [1,2046] -> **subtract 1023** real exp [-1022,1023]

exp == 2047(111 1111 1111) -> ∞

exp == 0(000 0000 0000) -> **0.**fraction x 2^{-1022}

> *The purpose is to compare two float numbers easily,that is,camparing every bit*

[2] **scientific counting**——1.fraction x 2^{exp}

The digit before point is always 1 so it's unnecessary to allocate.

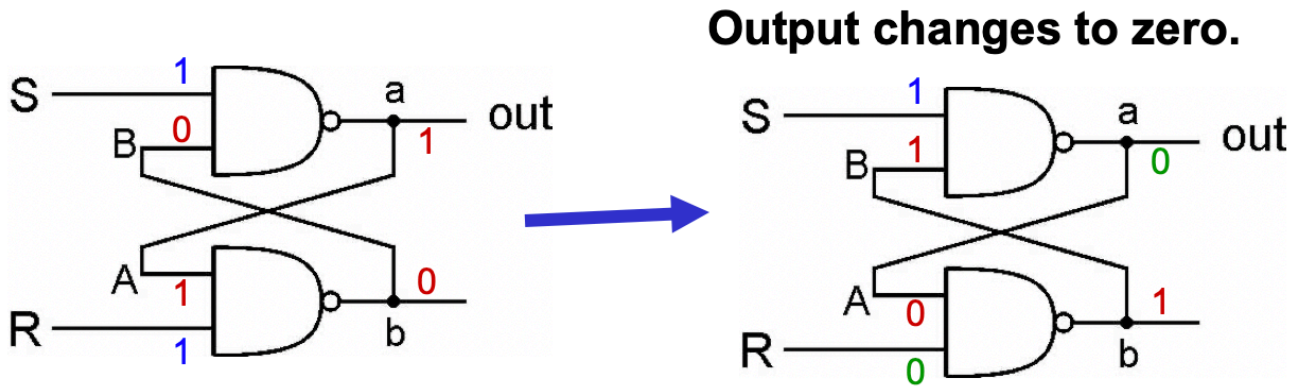
e.g. 1.10011001... then the fraction is 1001loop,until the 23rd bit

*Also have 16 bits floating point,1, 5, 10 for sign,exp and fraction respectively

2.3.2 Infinites

0 11111111 00000000000000000000000 -> +∞

1 11111111 00000000000000000000000 -> -∞



2.3.3 Converting Example

Example 1 Floating -> Decimal

Convert 1 10000001 1010 0000 0000 0000 0000 000 to decimal representation

sign	exp	frac
1	1000 0001	1010 0000 0000 0000 0000 000

exp: $(1000\ 0001)_2$-$(127)_{10}$ = 128+1-127 = 2;

coe:$(-1.101)_2$ = 1+1/2+1/8 = $(-1.625)_{10}$;

Result: -1.625 x 2^2 = -6.25

Example 2 Decimal -> Floating

Convert $(-35.6)_{10}$ to floating representation

sign: 1

$(35)_{10}$ = 32 + 2 + 1 = $(100011)_2$

$(0.6)_{10}$ = $(.1001\text{loop})_2$

$(35.6)_{10}$ = $(10\ 0011.1001\text{loop})_2$ = $(1.00011\ 1001\text{loop} \times 2^5)_2$

tips: How to convert decimal fraction to binary?

```
  |0.6 x2 = 1.2
1 |0.2 x2 = 0.4
0 |0.4 x2 = 0.8
0 |0.8 x2 = 1.6
1 |0.6 x2 = 1.2
loop 1001
```

sign	exp	frac
1	1000 0100	00011 1001 1001 1001 1001 10

2.4 Converting between Binary & Decimal

2.4.1 Converting Binary(2's C) to Decimal

- positive: normal converting
- Negative(*MS bit is 1*):
  - take 2's C to get a positive number
  - normal converting
  - add `-`

$$
\begin{aligned}
X &= 11100110_{two} \\
-X &= 00011010 \\
&= 2^4+2^3+2^1 = 16+8+2 \\
&= 26_{ten} \\
X &= -26_{ten}
\end{aligned}
$$

2.4.2 Converting Decimal to Binary(2's C)

```
01011.101

<-    ->

11    +1/2 + 1/8
```

4.5 The following table represents a small memory. Refer to this table for the following questions.

Address	Data
0000	0001 1110 0100 0011
0001	1111 0000 0010 0101
0010	0110 1111 0000 0001
0011	0000 0000 0000 0000
0100	0000 0000 0110 0101
0101	0000 0000 0000 0110
0110	1111 1110 1101 0011
0111	0000 0110 1101 1001

a. What binary value does location 3 contain? Location 6?

b. The binary value within each location can be interpreted in many ways. We have seen that binary values can represent unsigned numbers, 2's complement signed numbers, floating point numbers, and so forth.
   (1) Interpret location 0 and location 1 as 2's complement integers.
   (2) Interpret location 4 as an ASCII value.
   (3) Interpret locations 6 and 7 as an IEEE floating point number. Location 6 contains number[15:0]. Location 7 contains number[31:16].
   (4) Interpret location 0 and location 1 as unsigned integers.

c. In the von Neumann model, the contents of a memory location can also be an instruction. If the binary pattern in location 0 were interpreted as an instruction, what instruction would it represent?

d. A binary value can also be interpreted as a memory address. Say the value stored in location 5 is a memory address. To which location does it refer? What binary value does that location contain?

Answer:(b.(3)floating point 再次做错，因为再次漏看组合顺序)

a.Location3:0000 0000 0000 0000, Location6: 1111 1110 1101 0011
b.(1)Location0(2's C):7747,location1(2's C): -4059
(2)location4(ASCII):'e'
(3)location6 and 7(floating): 1.101 1001 1111 1110 1101 0011 x 2^-114
(4)location0(unsigned int): 7747,location1(unsigned int):61477
c.ADD instruction in LC3.ADD R1 and R3, and the result will store in R7.
d.location:0110(location6), value containing:1111 1110 1101 0011

3 Operations

3.1 Arithmetic Operations

3.1.1 Addition & Subtraction

- all int have the same number of bits
- ignore carry out
- the result still fits in n-bit 2's C

```
  01101000 (104)      11110110 (-10)
+ 11110000 (-16)    +          (-9)
  01011000 (98)               (-19)

  01101000 (104)      11110110 (-10)
- 00010000 (16)     -          (-9)
  01101000 (104)      11110110 (-10)
+ 11110000 (-16)    +          (9)
  01011000 (88)               (-1)
```

如果只是二进制加法，不考虑进位舍去

> 2.14 Add the following bit patterns. Leave your results in binary form.

e. 1111 + 0001

result 1 0000

3.1.2 Sign Extension

- zero extension: Fill 0 when extending.

```
4-bit          8-bit
0100 (4)       00000100 (still 4)
1100 (-4)      00001100 (12, not -4)
```

- sign extension: Fill sign bit when extending.

Instead, replicate the MS bit -- the sign bit:

4-bit		8-bit	
0100	(4)	00000100	(still 4)
1100	(-4)	11111100	(still -4)

3.1.3 Overflow(溢出)

The 2 only possible overflow situations:

- **positive + positive == negative**, that is, carry to the sign bit, and the sign bit becomes 1 after adding
- **negative + negative == positive**,the sign bit becomes 0 after adding

3.2 Logical Operations

3.2.1 Bit Vector

- View n-bit number as a collection of n logical values
- operation applied to each bit **independently**

3.2.2 NOT

truth table for NOT

A	NOT A
0	1
1	0

3.2.3 AND

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

- **Bit Mask**(掩码) A & 00100000 to reset 1->0

3.2.4 OR

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

3.2.5 XOR(异或)

Exclusive OR

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

3.2.6 DeMorgan's Law *Page 41*

AND OR NOT relationship

~ (~A & ~B) = A | B