第 1 讲 进制转换与计算机编码(原码、反码、补码)

■ 进制转换

■ X 进制整数的表示

核心概念：X 进制整数的每一位数字，乘以对应“位权”（$X^{位数-1}$）后相加，即为其十进制数值。

■ 数位、数字、位权的关系

以“X 进制数 998424353”为例（共 9 位）：

数位（从右数）	第一位	第二位	第三位	第四位	第五位	第六位	第七位	第八位	第九位
数字	3	5	3	4	2	4	8	9	9
位权（X^{n-1}）	X^0	X^1	X^2	X^3	X^4	X^5	X^6	X^7	X^8

关键注意点

X 进制的**基数 X** 必须大于每一位数字的最大值（例如：若数字最大是 9，则 X≥10）。

位权的指数是“**数位序号-1**”（从右数第一位的位权是X^0）。

■ 常见进制的特点（4 种）

进制类型	数字范围	前缀标识	特殊对应关系
十进制	0~9	0D/0d	日常生活最常用
二进制	0~1	0B/0b	-
八进制	0~7	数字 0	1 位八进制 = 3 位二进制
十六进制	0～9、A(10)～F(15)	0X/0x	1 位十六进制 = 4 位二进制

■ 二进制转十进制

核心方法：按权展开求和（每位数码×对应位权，再相加）

位权规则：

整数部分：从右（小数点左）开始，位权为 $2^0, 2^1, 2^2 \ldots$

小数部分：从左（小数点右）开始，位权为 $2^{-1}, 2^{-2}, 2^{-3} \ldots$

示例计算（二进制 1101.1001 转十进制）：

$$(1101.1001)_2 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$$

$$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0 + 0.0625$$

$$= (13.5625)_{10}$$

■ X 进制转十进制

核心方法：按权展开求和（每位数码×对应位权，再相加）

位权规则：

整数部分（小数点左）：从右到左，位权为 $X^0, X^1, X^2 \ldots$

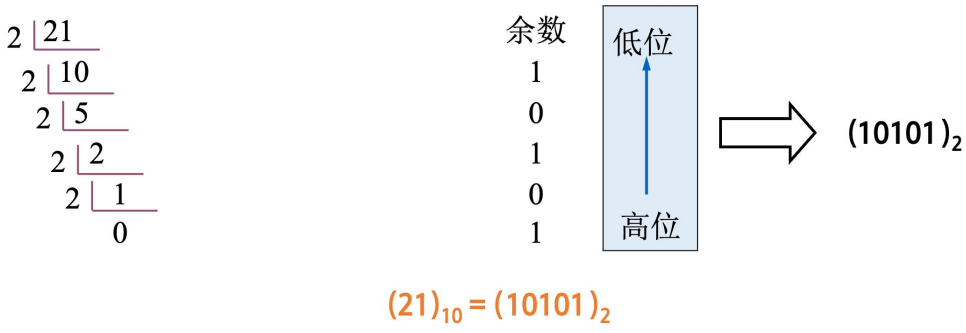
小数部分（小数点右）：从左到右，位权为 $X^{-1}, X^{-2}, X^{-3} \ldots$

示例计算（X 进制数 2048.1024 转十进制）：

$$(2048.1024)_{10} = 2\times X^3 + 0\times X^2 + 4\times X^1 + 8\times X^0 + 1\times X^{-1} + 0\times X^{-2} + 2\times X^{-3} + 4\times X^{-4}$$

■ 十进制转二进制（整数部分）

转换方法：除 2 取余法（除 2 取余直到商为 0，余数逆序排列）



$(21)_{10} = (10101)_2$

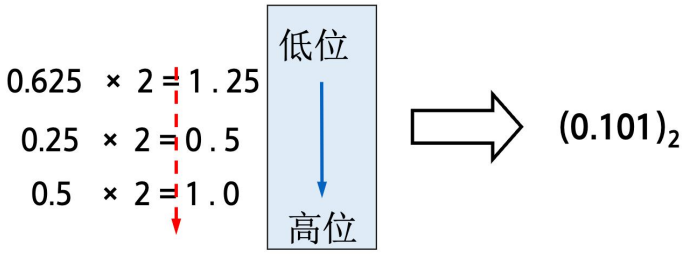
■ 十进制转 X 进制（整数部分）

十进制整数转换为 X 进制整数的方法为：除 X 取余直到商为 0，逆序排列。
十进制整数 2365 转为八进制整数过程如下：

$$(2365)_{10} = (4475)_8$$

■ 十进制转二进制：小数部分

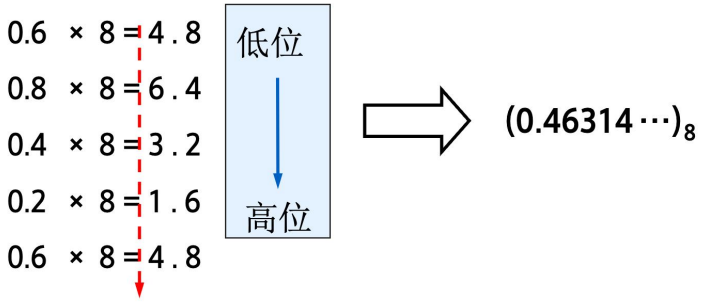
乘 2 取整直到小数部分为 0 或出现循环节，顺序排列。
十进制小数 0.625 转为二进制小数过程如下：



$$(0.625)_{10} = (0.101)_2$$

■ 十进制转 X 进制：小数部分

乘 X 取整直到小数部分为 0 或出现循环节，顺序排列。
十进制小数 0.6 转为八进制小数过程如下：

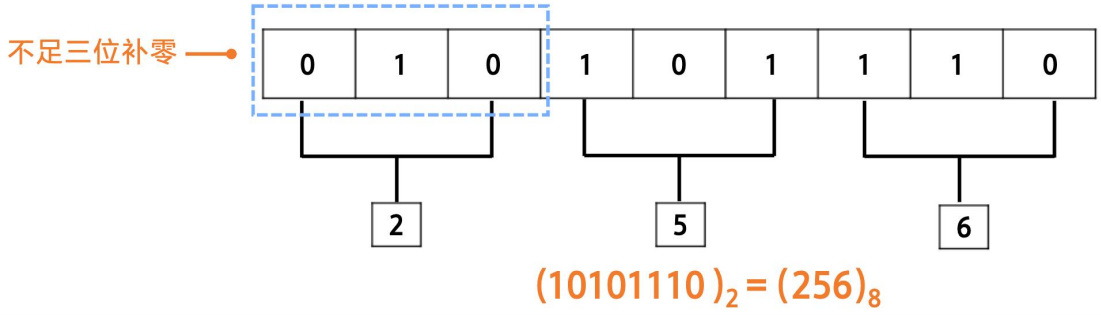


$$(0.6)_{10} = (0.46314\cdots)_8$$

因此，绝大部分浮点数无法用二进制精确表示

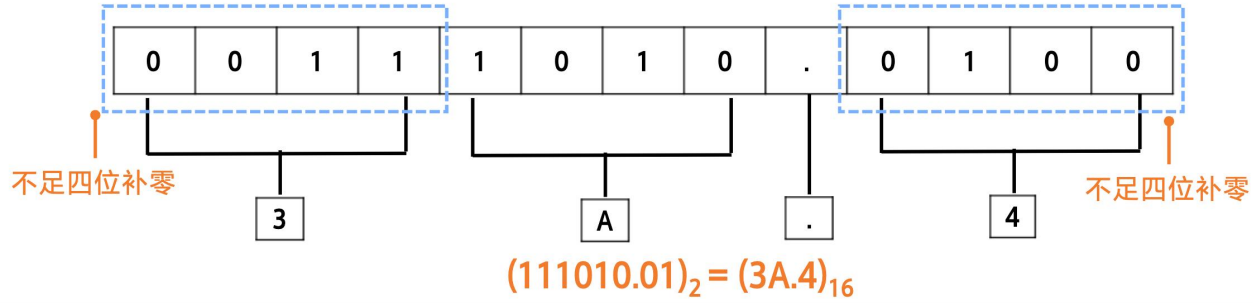
■ **二进制转八进制：三合一**

整数部分从低位向高位每三个二进制数分为一组，转换成一位八进制数字。高位不足三位前补零。
小数部分从高位向低位每三个二进制数分为一组，转换成一位八进制数字。低位不足三位后补零。
将二进制数 10101110 转换为八进制：



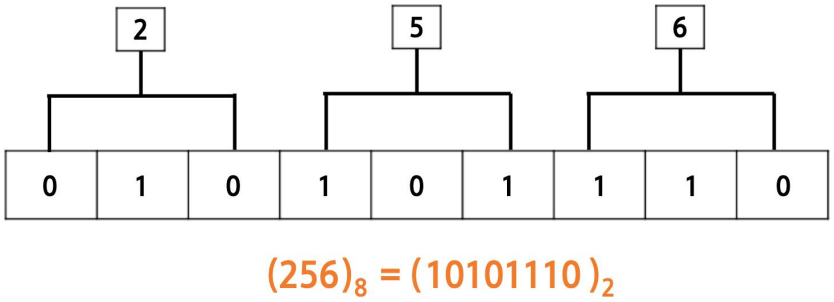
■ **二进制转十六进制：四合一**

整数部分从低位向高位每四个二进制数分为一组，转换成一位十六进制数字。高位不足四位前补零。
小数部分从高位向低位每四个二进制数分为一组，转换成一位十六进制数字。低位不足四位后补零。
将二进制数 111010.01 转换为十六进制：



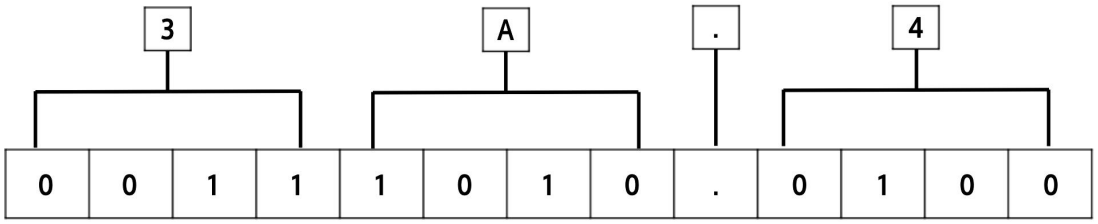
■ **八进制转二进制：一分三**

将每一位八进制数字转换为三位二进制数字，把转换后的二进制数字拼在一起后，分别抹掉整数部分的高位零和小数部分的末尾零。
将八进制数 256 转换为二进制：



■ **十六进制转二进制：一分四**

将每一位十六进制数字转换为四位二进制数字，把转换后的二进制数字拼在一起后，分别抹掉整数部分的高位零和小数部分的末尾零。
将十六进制数 3A.4 转换为二进制：



$$(3A.4)_{16} = (111010.01)_2$$

■ **流操作算子**

■ **以 X 进制输出整数**

```
int a = 100;
cout << dec << a << endl;
cout << hex << a << endl;
cout << oct << a << endl;
```

分别以十进制，十六进制，八进制输出 a

流操作算子	作用	备注
dec	以十进制形式输出整数	默认
hex	以十六进制形式输出整数	影响以后所有输出
oct	以八进制形式输出整数	影响以后所有输出

■ **保留 n 位小数**

```
double a = 3.1415;
double b = 2.718;
double c = a * b;
cout << fixed << setprecision(3) << c;
```

输出结果小数部分精度到第三位

流操作算子	作用	备注
fixed	以普通小数形式输出浮点数	
setprecision(n)	设置输出浮点数的精度为 n	一般与 fixed 连用

■ 设置宽度

```
cout << setw(10) << 123 << endl;
cout << setw(10) << "hello" << endl;
```

输出宽度为 10，整数，字符串靠右对齐

```
string s1, s2;
cin >> setw(4) >> s1 >> setw(3) >> s2;
```

将前四个字符读入进 s1，后三个字符读入进 s2

流操作算子	作用	备注
setw(w)	指定输出宽度为 w 个字符，或输入字符串时读入 w 个字符	只影响下一次输出/输入

■ 计算机编码（原码、反码、补码）

机器数：计算机中二进制表示形式，最高位为符号位（0 正、1 负）
真值：机器数对应的实际数值（如机器数 10000011 真值为 - 3）

■ 原码 = 符号位+数值位

最高位为符号位，用 0 表示正数，用 1 表示负数，其余位表示这个数值的大小。

原码能表示的范围是$-(2^{n-1}-1) \sim (2^{n-1}-1)$。

■ 反码

当数字是正数的时候，它的反码与它的原码相同；当数字是负数的时候，它的反码是：先保持原码的符号位〝1〞不变，然后其余位全部取反（1 变成 0，0 变成 1）。

数字	原码	反码	备注
22	00010110	00010110	每一位都一样
-22	10010110	11101001	最高位一样，其他位都反

反码加法：在加法结果不溢出的情况下，遵循“符号位参与运算+循环进位”规则，即最高位产生的进位需加到结果的最低位。
因为减法可以通过加法来实现，在这个规则下，就解决了原码直接进行加减运算出错的问题。
由于 0 在反码里有两种不同的表示方法，在进行数学运算时，仍然存在问题，比如比较-0 和+0 的值是否相等。

■ 补码

当数字是正数的时候，它的补码与它的原码、反码均相同；当数字是负数的时候，它的补码是它的反码加 1。

数字	原码	反码	补码	备注
22	00010110	00010110	00010110	每一位都一样
-22	10010110	11101001	11101010	

由于 0 在补码中只有一种表示方式，因此补码彻底解决了-0 问题，成为计算机中整数的标准存储方式。
补码加法：遵循“符号位参与运算+进位舍弃”规则，即最高位产生的进位直接舍弃。
补码减法：通过加法来实现。

■ 常用计算

补码转真值：符号位 1（负数）→补码取反 + 1 得绝对值；符号位 0（正数）→直接转十进制
例：补码 10111011→取反 01000100→+1=01000101（69）→真值 - 69

负数补码计算：原码→符号位不变其余取反（反码）→+1
例：-6（16 位）→原码 1000000000000110→反码 1111111111111001→补码 1111111111111010（FFFA）