

第二讲 数的算法 - 基础

Lecture 2

Number-Theoretic Algorithms - Basis

明玉瑞 Yurui Ming

yrming@gmail.com

声明

Disclaimer

- ▶ 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识产权，所引材料之知识产权均归原著者所有；若原著者介意之，请联系作者更正及删除。

The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

背景

Background

- ▶ 传统观点认为，计算机程序等于数据结构加算法；一个设计良好的数据结构，应该在尽可能使用较少的时间与空间资源的前提下，支持各种算法的运行。

A conventional point of view interprets the computer programs as the hybridization of data structures plus algorithms. A well-designed data structure can support various algorithms in comparatively less resource include time and space.

- ▶ 但不同种类的数据结构适合不同的算法，部分数据结构甚至是为了解决特定问题而设计出来的。例如B树即为加快树状结构访问速度而设计的数据结构，常被应用在数据库和文件系统上。因此，选择正确的数据结构选择可以提高算法的效率。

Different data structure caters to different algorithms, and some algorithms are tailored to satisfying specific algorithms. For example, the B tree data structure is designed to accelerate accessing the tree structure nodes, and it usually finds applications in database and file systems. Hence, proper choice of data structure can improve the effectiveness of certain algorithms.

标准模板库

Standard Template Library

- 对于一些常用的数据结构，如数组，链表，有序集合等，没有必要重复造轮子，C++的标准模板库提供了一些通用的数据结构，可以在此基础上方便地实现算法实现。

For some commonly-used data structure such as array, linked-list, ordered set, it is not necessary to re-invent the wheel. Luckily, the C++ standard template library provides some generic data structures, which underpin some sophisticated algorithms built on that.

- 标准模板库是基于C++的一个具有工业强度的高效程序库。它包含了诸多在计算机科学领域里所常用的基本数据结构和算法。这些数据结构可以与标准算法一起很好的工作，这为我们的软件开发提供了良好的支持。

Standard template library is an industrial-strength efficient library based on C++. It contains many fundamental data structures and algorithms commonly used in computer science. These data structures work well with standard algorithms, facilitate the software developing process.

示例

Exemplification

- ▶ 下面为利用STL实现插入排序的例子

The following demonstrates implementation of insertion-sort by using STL

- ▶ 为什么需要模板？为什么需要类型？

Why we need template? Further, why we need type?

```
void insertion_sort(vector<int>& v)
{
    for (int i = 1; i < v.size(); i++)
    {
        auto key = v[i];
        int j = i - 1;

        for (; j >= 0 && v[j] > key; j --)
        {
            v[j + 1] = v[j];
        }
        v[j + 1] = key;
    }
}
```

数制表示

Numeric System

- 计算机系统基于元件的电磁学特性建立起来。在宏观层面，这些元件仍工作在模拟状态，即一个状态到另一个状态的转换并不明显。在人的观念里，0~9这几个数字均表示清晰的值或状态，但对物理元件来说，不行。不过，对物理器件而言，有两种状态是比较明确的，如开和关，导通与不导通，电流截止与电流饱和，高电压与低电压，诸如此类。因此，如何充分利用此二值状态实现计算，是构造计算机系统的关键。

The computer is built upon the electro-magnetic properties of electro-magnetic devices. From the macro perspective, these devices are still working in the analog way, which means there is no clear boundary between state transitions. For example, in our conception, the numerals 0 to 9 each represents individual value or state without a cent ambiguity. However, it is not possible for physical devices. For physical devices, there are two states usually clearly separated from each other. For example, switched on and switched off, current cut-off and current saturation, high voltage ($>3.5\text{V}$) and low voltage ($<1.5\text{V}$), etc. Therefore, how to make full use of these binary states to achieve computation is the key to constructing a computer system.

数制表示

Numeric System

- 利用此种二值状态的第一步，是将这种现象抽象化，形成相应的二值数字逻辑及相应的数字算术运算等理论及实践。在算术运算中，一个预备工作是数制转换，即将用十进制表示的数字，用二进制表示。首先我们来了解数值转换的一些算法。

The first step to utilize this binary state is to abstract these phenomena into corresponding theories and practices, for example, the binary digital logic and binary arithmetic operations. For arithmetic operations, a prerequisite is the number system conversion, aka, change from decimal numbers to binary numbers. First, let's understand conversions between number systems.

- 我们首先了解一下十进制到二进制的转换。

First, we will learn the conversion from decimal to binary.

Numeric System

A number in base 10 has the following equivalent representations, exemplified by the number 2947:

$$2947 = 2 * 10^3 + 9 * 10^2 + 4 * 10^1 + 7 * 10^0$$

► 我们反复整除以10，看会发生什么：

Let's recursively divide by 10 to see what will happen:

ively divide by 10 to see what will happen:

$$\begin{array}{rcll} 2947 \div 10 & = & 2 * 10^2 + 9 * 10^1 + 4 * 10^0 & \dots\dots\dots 7 \\ & & \underbrace{\hspace{10em}}_{294} & \\ 294 \div 10 & = & 2 * 10^1 + 9 * 10^0 & \dots\dots\dots 4 \\ & & \underbrace{\hspace{10em}}_{29} & \\ 29 \div 10 & = & 2 * 10^0 & \dots\dots\dots 9 \\ & & \underbrace{\hspace{10em}}_2 & \\ 2 \div 10 & = & 0 & \dots\dots\dots 2 \end{array}$$

数制表示

Numeric System

- 因为不论以10为基，或以2为基，其实际数值是一样的，则下面的表示均等价：

Irrespective of base 10 or base 2, the same numeric value stays the same of different expressions:

$$43 = 4 * 10^1 + 3 * 10^0 = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \dots + a_0 \cdot 2^0$$

- 所以我们可以通过反复整除以2，可以预期得出 a_n 及到 a_0 的系数：

So, we can expect that by iteratively dividing 2, we can find the coefficients from a_n to a_0 :

43	÷ 2 = 21	1
21	÷ 2 = 10	1
10	÷ 2 = 5	0
5	÷ 2 = 2	1
2	÷ 2 = 1	0
1	÷ 2 = 0	1

数制表示

Numeric System

- 总结上面过程，我们可以得出十进制到二进制的转换算法如下：

Summation of the above procedure draws into the following algorithm of converting numbers in base 10 number system to base 2 number system.

```
input: n, bin[]  
do  
{  
    r = n module 2  
    n = floor(n / 2),  
    bin.push_back(r)  
} while (n != 0)  
reverse(bin)  
output: bin
```

数字逻辑

Digital Logic

- ▶ 上面我们从运算的角度探讨了利用二值进行运算的初步，下面来讨论一下二值数字逻辑。

We gave a preliminary touch about binary arithmetic operations, now we discuss some points about binary digital logic.

- ▶ 因为只有两种状态，因此可以用0和1表示。这里的0和1可以和二进制中的0和1对应，但又不完全相同。例如，逻辑的0与1其实没有大小之分，我们不能说逻辑1比0大或者什么的。

We can represent these binary states in the form of 1 and 0, which can correspond to 1 and 0 in the base 2 number system. But they are not exactly the same. For example, from the ordering perspective, comparing the numeric value of logic 1 and logic 0 is meaningless, aka, we cannot say 1 is greater than 0 from the logical perspective.

数字逻辑

Digital Logic

- ▶ 比如，在物理元器件或数字电路中，我们常常接触到高电平（5V）与低电平（0V），若用正逻辑，则5V对应1，0V对应0；若用负逻辑，5V对应0，0V对应1；这里1与0仅代表不同的状态，没有大小之分。

For example, in the fields of physical devices or digital circuits, high voltage (5V) and low voltage are frequently encountered. In terms of positive logic, 5V corresponds to 1 and 0V maps to 0. In terms of negative logic, 5V corresponds to 0 and 0V to 1.

电压/V	二值逻辑		逻辑电平
	正逻辑	负逻辑	
+ 5	1	0	H（高电平）
0	0	1	L（低电平）


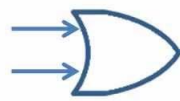


数字逻辑

Digital Logic

- 对上面所述进行抽象，会得出一个学科方向称为布尔代数，其中常量为0与1，或true与false；变量也称布尔变量，取true或false之一。在此基础之上，我们有布尔运算，常见的有与运算，或运算，取否，异或运算等等，部分规则如下：

Abstraction of the above description leads to a mathematics branch called Boolean algebra, in which the constants are 1 and 0, or true and false, with variables named Boolean variables of values in true or false. These underpin the Boolean algebra, with operation such as and, or, not, xor, etc. Rules of them are illustrated below:

NOT		AND			OR			XOR		
x	F	x	y	F	x	y	F	x	y	F
0	1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1	0	1	1
		1	0	0	1	0	1	1	0	1
		1	1	1	1	1	1	1	1	0

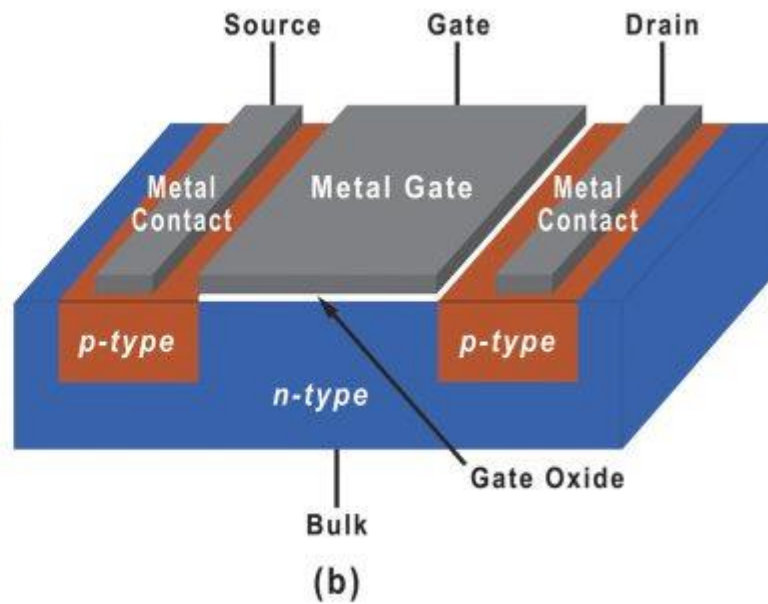
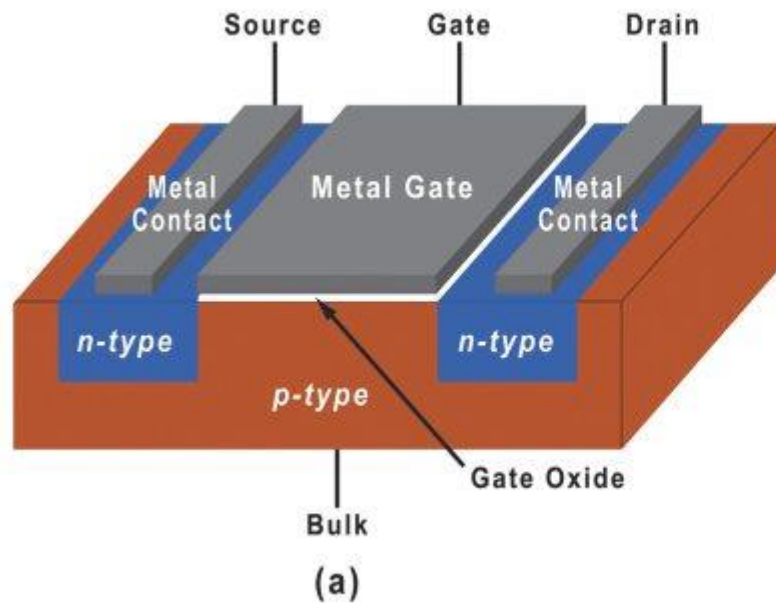


数字逻辑

Digital Logic

- 用半导体电子元器件，很容易实现逻辑门：

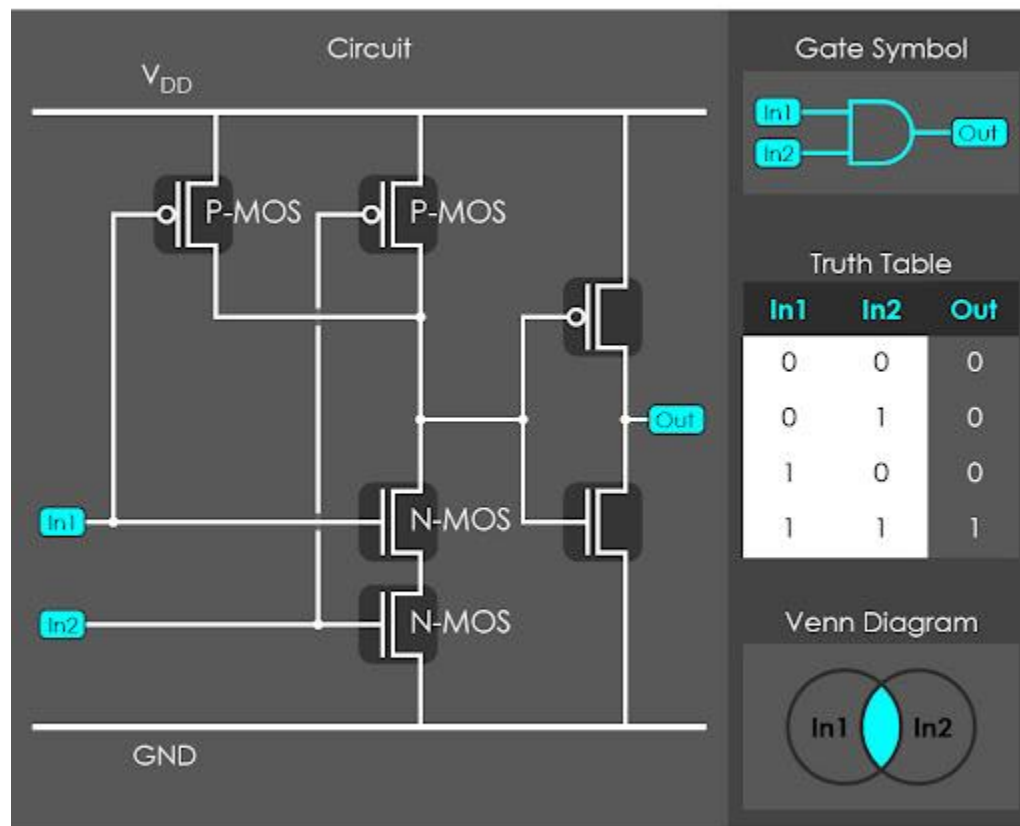
The logic gate can be easily implemented by using semi-conductor devices



数字逻辑

Digital Logic

AND Gate [CMOS]



算术逻辑单元

Arithmetic Logic Unit

- ▶ 算术运算与逻辑运算并不是截然对立的，算术运算可以转换为逻辑运算来做，例如，两个位数是1的二进制数的加法，其规则如下图所示，其中S表示和， C_{out} 表示进位。

The arithmetic operations and logic operations are not totally incompatible with each other. Actually, arithmetic operations can be performed in logic operations. For example, consider the sum of two 1-bit numbers, with the rules illustrated below, where S indicates sum, and C_{out} for carry-on.

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

算术逻辑单元

Arithmetic Logic Unit

- 经过观察知，S的结果为A与B的异或， C_{out} 的结果为A和B的与运算。

From the observation we can draw the conclusion that S results from A xor B, and C_{out} results from A and B.

S	A=0	A=1
B=0	0	1
B=1	1	0

$$S = A'B + B'A = A \text{ xor } B$$

C_{out}	A=0	A=1
B=0	0	0
B=1	0	1

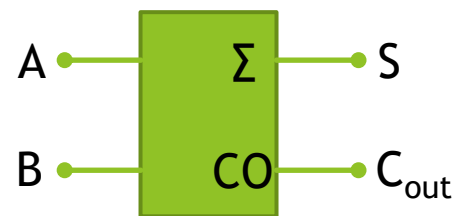
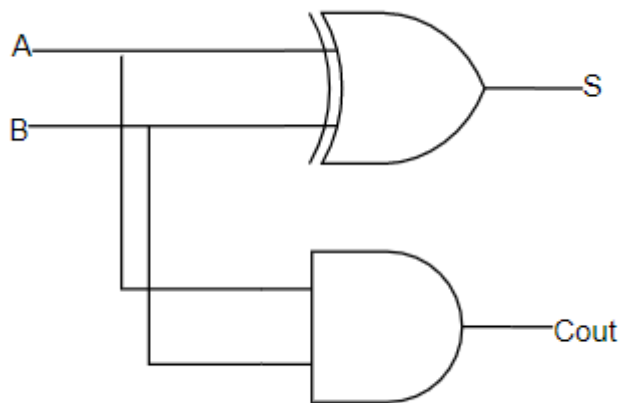
$$C_{out} = AB = A \text{ and } B$$

算术逻辑单元

Arithmetic Logic Unit

- 我们可以用门电路将上面算术运算表达如下：

We can express the above arithmetic sum by using logic gate as illustrated below:

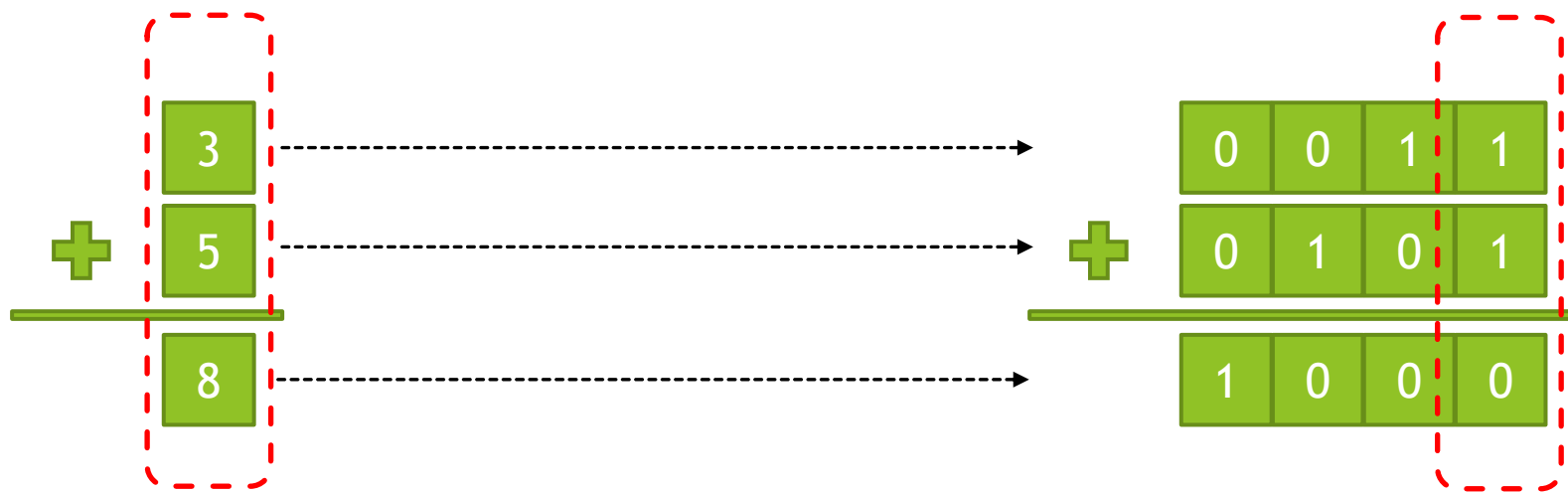


算术逻辑单元

Arithmetic Logic Unit

- 我们知道，实际应用中的数字不可能只有一位，如下图所示：

We know usually in practice the numbers involved in computation cannot be always just one bit, as illustrated below:

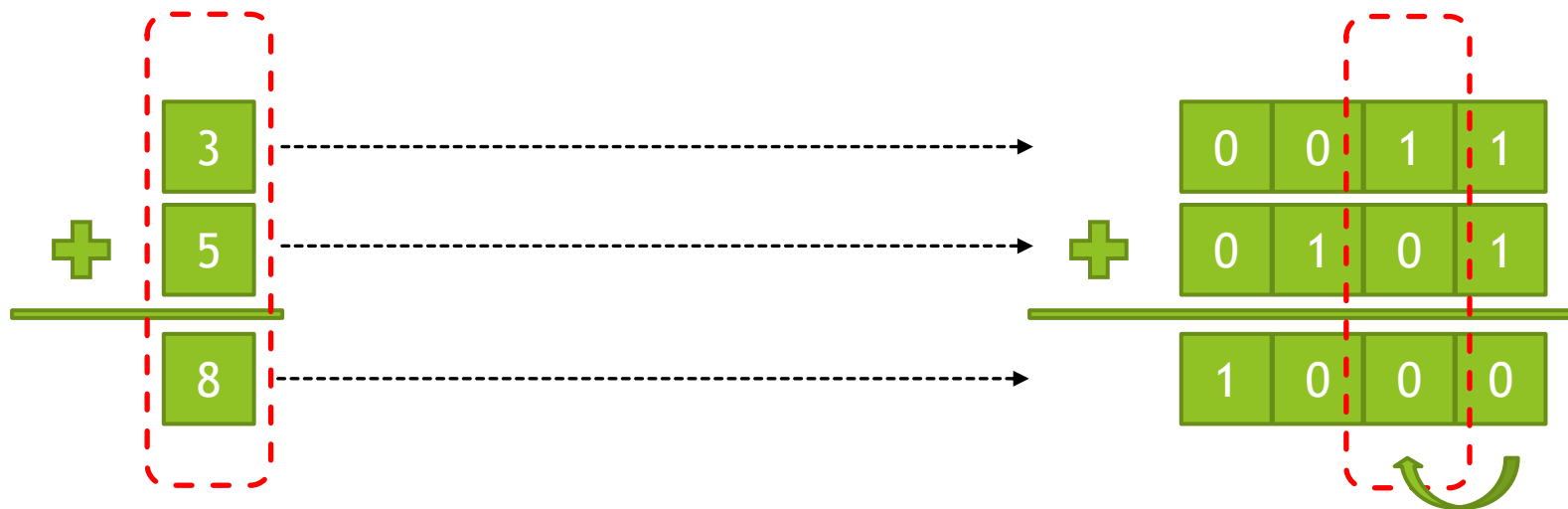


算术逻辑单元

Arithmetic Logic Unit

- ▶ 刚才我们讨论两个1位二进制数的时候，没有考虑从低位的进位，这在实际中是不可避免的，如下图所示：

The addition we considered before doesn't take the carry-in from adjacent lower bit into consideration and this is unavoidable in practice, as shown below:



算术逻辑单元

Arithmetic Logic Unit

- 此时我们要考虑的便如下表所示，其中 C_{in} 表示由低位的进位。

Now we re-consider the addition before and reach the rules shown in the following table, where indicates the carry-in from the adjacent lower bit.

C_{in}	A	B	S	C_{out}
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

算术逻辑单元

Arithmetic Logic Unit

- 接下来，我们分析S及 C_{out} 与A、B及 C_{in} 的关系。

Now we analyze the relation of C_{out} and A among with the terms A, B and C_{in} .

S	AB=00	AB=01	AB=11	AB=10
$C_{in}=0$	0	1	0	1
$C_{in}=1$	1	0	1	0

$$\begin{aligned} S &= A'B'C_{in} + A'BC_{in}' + ABC_{in} + AB'C_{in}' \\ &= A'(B'C_{in} + BC_{in}') + A(BC_{in} + B'C_{in}') \\ &= A'(B \text{ xor } C_{in}) + A(B \text{ xor } C_{in})' \\ &= A \text{ xor } B \text{ xor } C_{in} \end{aligned}$$

$$BC_{in} + B'C_{in}' = (B \text{ xor } C_{in})': \text{ 用看就好}$$

C_{out}	AB=00	AB=01	AB=11	AB=10
$C_{in}=0$	0	0	1	0
$C_{in}=1$	0	1	1	1

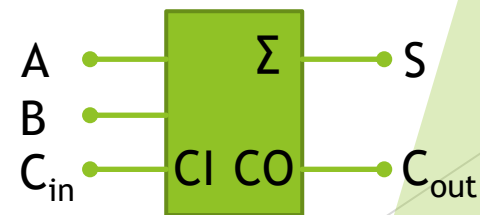
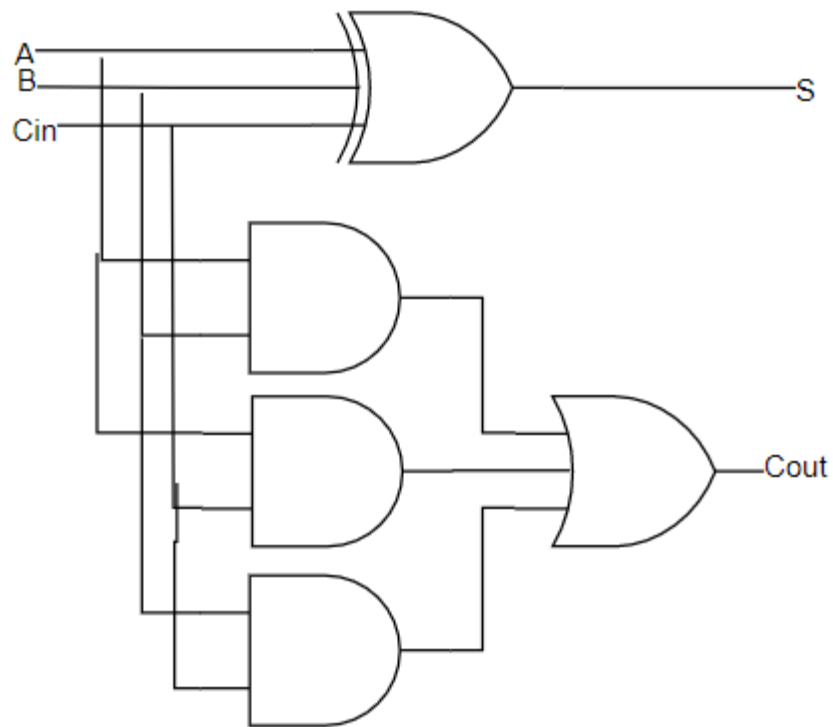
$$\begin{aligned} C_{out} &= AB + BC_{in} + AC_{in} \\ &= A \text{ and } B \text{ or } A \text{ and } C_{in} \text{ or } B \text{ and } C_{in} \end{aligned}$$

算术逻辑单元

Arithmetic Logic Unit

- 我们可以用门电路将上面算术运算表达如下：

We can express the above arithmetic sum by using logic gate as illustrated below:

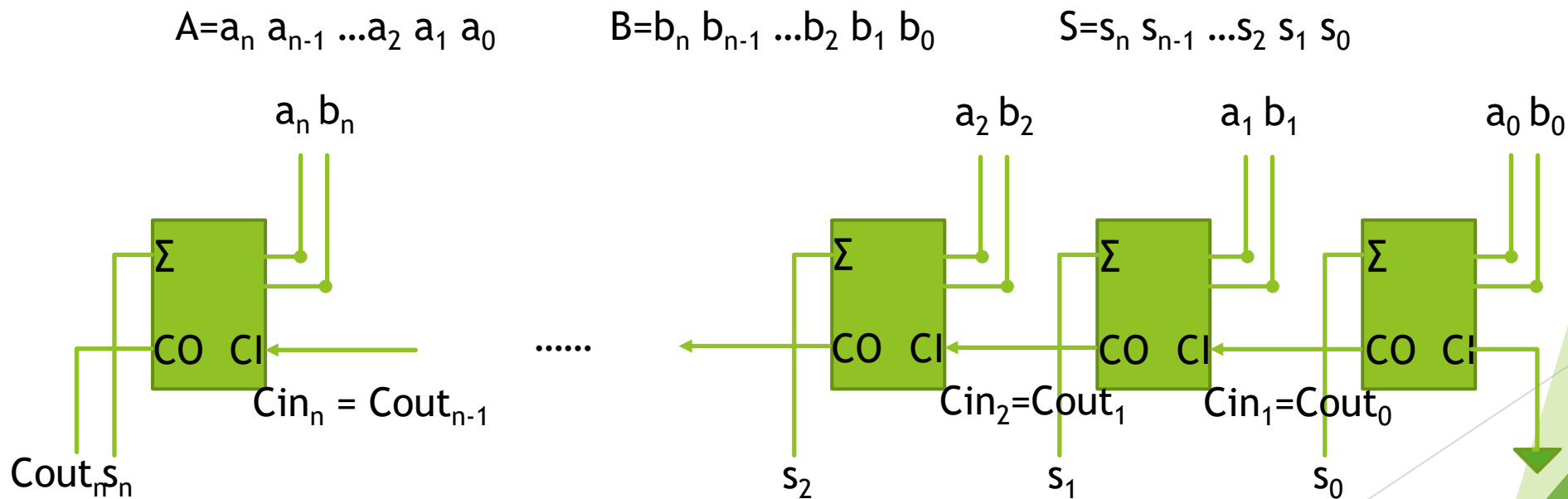


算术逻辑单元

Arithmetic Logic Unit

- 把前面的设计的电路连起来，便可以形成多个二进制位的数字的加法：

To cascade the circuits designed previously, so as to deal with addition of numbers of multiple bits.



算术逻辑单元

Arithmetic Logic Unit

- ▶ 上面我们所展示的电路只是ALU的一部分，因为其只能处理整数的情况；对于浮点数，需要相应的浮点单元电路；而ALU又是CPU的一部分。所以对于C/C++这种可以直接和硬件打交道的语言，我们需要显式声明类型，以便操作时选择最合适的电路。

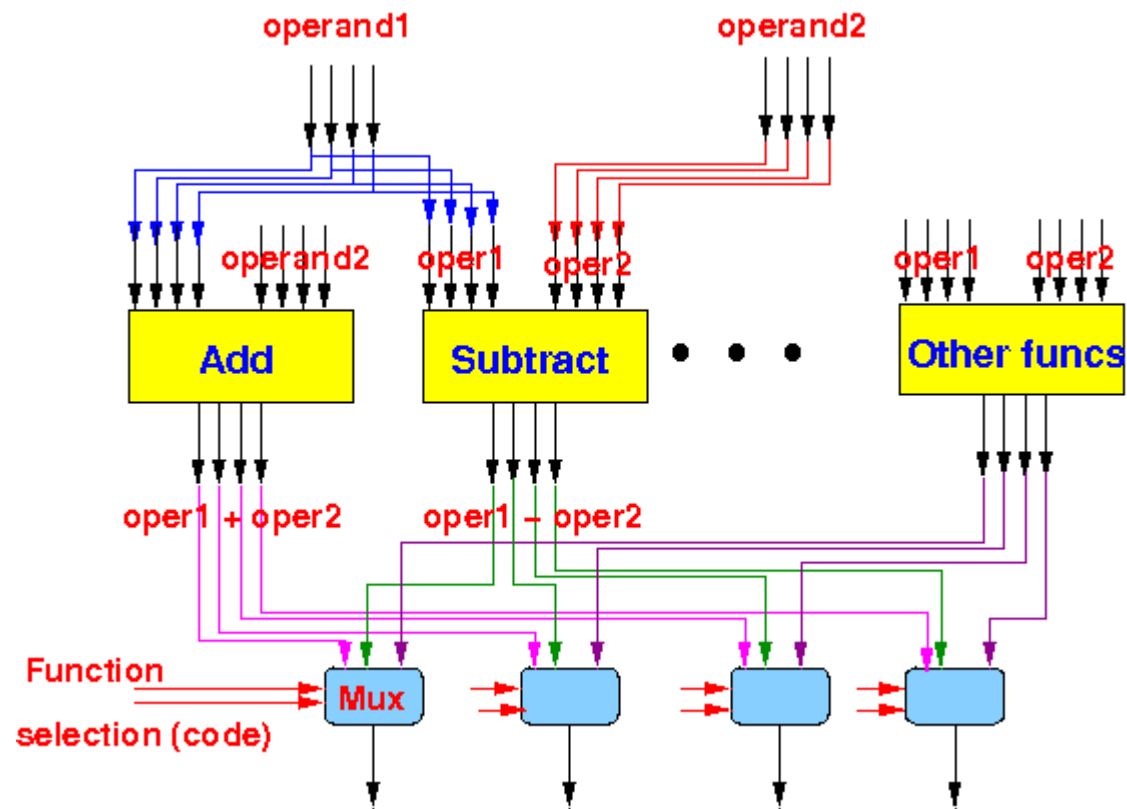
The circuits we exhibited above is just a part of the ALU, due to the fact it can only process the integer case. For floating numbers, it requires other dedicated circuits to cater to the processing. Recursively, ALU is part of CPU. For programming languages C/C++ which can directly interact with hardware, we need explicitly specify the type, in order to choose the most appropriate circuits.

- ▶ 所以，我们有vector<int>, list<double>等

So, we have vector<int>, list<double>, etc.

算术逻辑单元

Arithmetic Logic Unit



习题

Problems

- 阅读相关文献，学习关于小数的10进制表示到二进制表示的转换。

Survey the literature to find out the way of converting float point numbers from base 10 to base 2.

- 基于STL实现10进制到二进制的转换。

Implement the conversion from base 10 to base 2 based on STL.

- 枚举C++语言的基本类型。

Enumerate the basic types of C++.