位运算

符号	描述	运算法则
&	与	两个位都为1时，结果才为1
|	或	两个位都为0时，结果才为0
~	取反	名字可知，0变1、1变0
^	异或	两个位相同为0，相异为1
<<	左移	将所有二进制位向左移动若干位，丢弃高位，并在低位补0
>>	右移	将所有二进制位向右移动若干位，对于无符号数，高位补充零；对于有符号数，不同的编译器有不同的处理方法，有的是补充符号位（即算术右移），有的是补充0（即逻辑右移）。

与运算 &　　0&0=0　　0&1=0　　1&0=0　　1&1=1

或运算 |　　0|0=0　　0|1=1　　1|0=1　　1|1=1

非运算 ~　　~1=0　　~0=1　　~10001=01110（二进制）

异或运算 ^　　0^0=0　　0^1=1　　1^0=1　　1^1=0

同或运算⊙　　0 ⊙ 0=1　　0 ⊙ 1=0　　1 ⊙ 0=0　　1 ⊙ 1=1

左移 <<　　等于乘以 2

右移 >>　　等于除以 2

```cpp
#include<iostream>// 左移
using namespace std;
int main()
{
	int a,b;

	a=5 <<1;
	cout<<a<<endl;

	b=5 <<2;
	cout<<b;

	return 0;
}
```

```
10
20
```

```cpp
#include<iostream>// 右移
using namespace std;
int main()
{
	int a,b;

	a=5 >>1;
	cout<<a<<endl;

	b=5 >>2;
	cout<<b;

	return 0;
}
```

```
2
1
```

```cpp
#include<iostream>// 与运算 &
using namespace std;
int main()
{
	int a,b,c,d,e;

	a=0&0;		cout<<a<<endl;

	b=0&1;		cout<<b<<endl;

	c=1&0;		cout<<c<<endl;

	d=1&1;		cout<<d<<endl;

	e=4&5;		cout<<e;

	return 0;
}
```

```
0
0
0
1
4
```

```cpp
#include<iostream>// 或运算 |
using namespace std;
int main()
{
	int a,b,c,d,e;

	a=0|0;		cout<<a<<endl;

	b=0|1;		cout<<b<<endl;

	c=1|0;		cout<<c<<endl;

	d=1|1;		cout<<d<<endl;

	e=4|5;		cout<<e;
	return 0;
}
```

```
0
1
1
1
5
```

按位取反，需要了解一下原码、补码、反码、取反。

原码：
正数是其二进制本身；
负数是符号位为 1，数值部分取 X 绝对值的二进制。

反码：
正数的反码和原码相同；
负数是符号位为 1，其它位是原码取反。

补码：
正数的补码和原码，反码相同；
负数是符号位为 1，其它位是原码取反，末位加 1。（反码末尾减 1）（或者说负数的补码是其绝对值反码末位加 1）

取反就是简单的 0 变 1,1 变 0 ；

二进制数在内存中的存放形式，二进制数在内存中是以补码的形式存放的。

以计算正数 9 的按位取反为例，计算步骤如下（注：前四位为符号位）：
- 原码　　：0000 1001
- 算反码 ：0000 1001 （正数反码同原码）
- 算补码 ：0000 1001 （正数补码同反码）
- 补取反 ：1111 0110 （全位 0 变 1，1 变 0）
- 算反码 ：1111 0101 （末位减 1）
- 算原码 ：1111 1010 （其他位取反）

总结规律：$\sim x = -(x+1)$

```cpp
#include<iostream>// 非运算（取反）~
using namespace std;
int main()
{
	int a,b,c,d,e;
	a=~1;		cout<<a<<endl;

	b=~0;		cout<<b<<endl;
	//~ 表示按位取反，即在数值的二进制表示方式上，将 0 变为 1，将 1 变为 0
	c=~9;		cout<<c<<endl;

	d=~-4;		cout<<d<<endl;
	return 0;
}
```

```
-2
-1
-10
4
```

```cpp
#include<iostream>// 异或运算 ^
using namespace std;
int main()
{
	int a,b,c,d,e;

	a=0^0;		cout<<a<<endl;

	b=0^1;		cout<<b<<endl;

	c=1^0;		cout<<c<<endl;

	d=1^1;		cout<<d<<endl;

	e=4^5;		cout<<e;

	return 0;
}
```

```
0
1
1
0
1
```

程序中的所有数在计算机内存中都是以二进制的形式存储的，位运算就是直接对整数在内存中的二进制位进行操作（补码和符号位也参与运算，后面再补充补码的知识），如果是十进制数间进行位运算，则将十进制数先转换成二进制数再运算。位运算符及其运算规则如下：

按位与 (&)

两个都是 1 才是 1. 例如：

```
 00101      5
 11100      28
&------------------------
 00100      4
```

& 运算通常用于二进制数的取位操作，例如一个数“&1”的结果就是取二进制数的最末位。这可以用来判断一个整数的奇偶性，二进制的最末位为 0 表示该数为偶数，最末位为 1 表示该数是奇数。对于一个数 a，a=a&(a-1) 的作用是将 a 的二进制数最右边的 1 变为 0。

按位或 (|)

只要有一个为 1 就为 1。例如：

```
 00101      5
 11100      28
|------------------------
 11101      29
```

| 运算通常用于二进制数特定位上的无条件复制，例如一个数“|1”的结果就是把二进制最末位强行变成 1. 如果需要把二进制数最末位变成 0，对这个数“|1”之后再减一就可以了，其实际意义就是把这个数强行变成最接近的偶数。

按位异或 (^)

相同为 0，不同为 1（可以理解为“半加”，也就是不进位的二进制数加法）。例如：

```
 00101      5
 11100      28
^------------------------
 11001      25
```

^ 运算可以用于简答的加密，比如我的某个密码是 123456，但又担心密码被泄露，所以我使用 19960706 作为秘钥，123456 ^ 19960706= 20017602，我就把 20017602 记下来，等要使用密码时再计算 20017602 ^ 19960706 = 123456，就得到我的密码了。

加法的逆运算是减法，乘法的逆运算是除法，^ 运算的逆运算是它本身。由此可以写出不需要中间变量的 swap(交换两个数) 过程。

按位取反 (~)

单目运算符，1 变成 0，0 变成 1，例如：

```
 00101      5
~    ------------------
 11010      -6
```

注意 11010 为补码。

使用 ~ 运算时要格外小心，需要注意整数类型有没有符号。如果 ~ 的对象是无符号整数 (unsigned)，那么得到的值就是它与该类型上界的差。

左移 (<<)

a<<b 就表示 a 转为二进制后左移 b 位 (在后面添加 b 个 0)。例如 10 的二进制数为 1100100，而 110010000 转成十进制数是 400，那么 100<<2=400。可以看出 a<<b 的值实际上就是 a 乘以 2 的 b 次方，因为在二进制数后面添加一个 0 就相当于该数乘以 2。

通常认为 a<<1 比 a*2 更快，因为前者是更底层的一些操作。因此程序中乘以 2 的操作可以考虑用左移一位来代替。

定义一些常量可能会用到 << 运算。你可以使用 1<<16-1 来表示 65535。很多算法和数据结构要求数据规模必须是 2 的幂，此时可以用 << 来定义 MAX_N 等常量。

右移 (>>)

与 << 相似，a>>b 表示把 a 转为二进制数后右移 b 位，低位舍弃 b 位，高位补 b 位符号位 (正数补 0、负数补 1)。

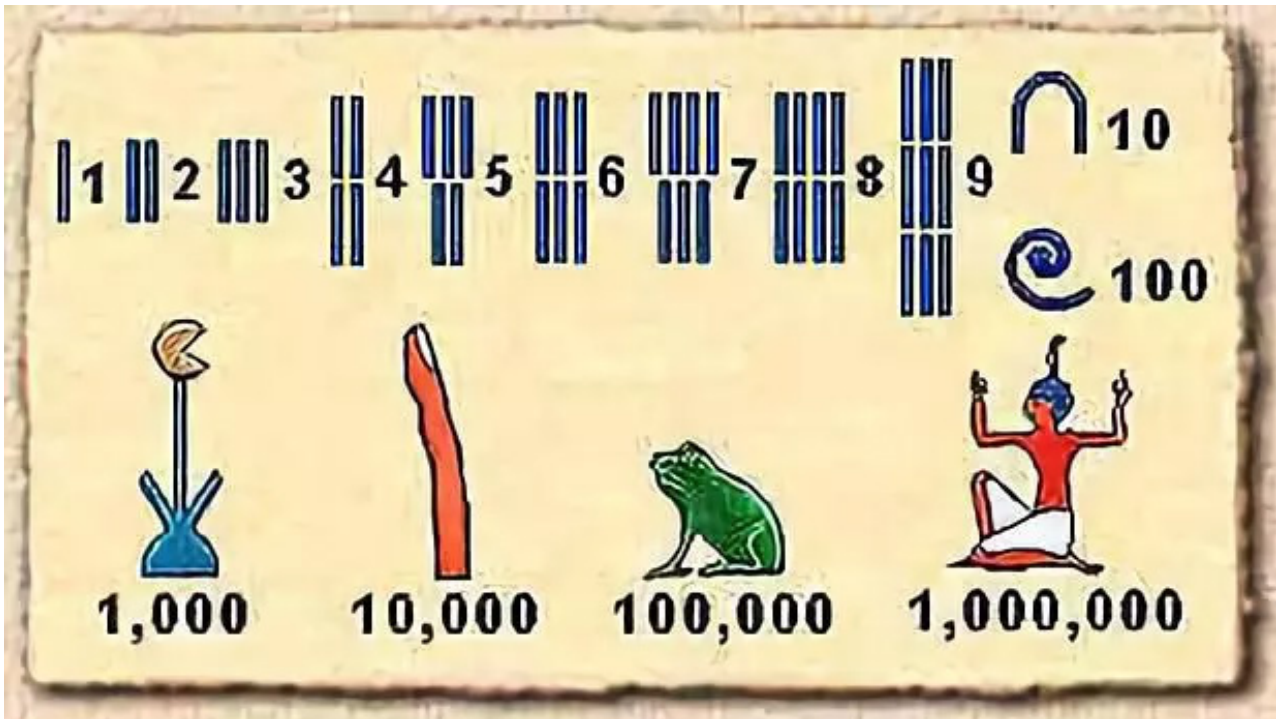
右移运算对于正数，相当于 a 除以 2 的 b 次方 (取商)，用 >> 代替除法运算可以使程序效率大大提高。

进制简史

（选自全景式数学）

十进制一直以来都是人类使用的最广的一种记数体系。

关于十进制的历史，时间最早的确切记录是在古埃及，时间大概是 5000 年前左右。中国关于十进制最早的可靠记载，是出现在殷墟中出土的甲骨文上，大约是 3000 年前左右。



古埃及十进制

实际上，在古代世界独立开发的有文字的记数体系中，除了巴比伦文明楔形数字的 60 进制和玛雅数字的 20 进制外（下文会讲到），几乎全部为十进制。

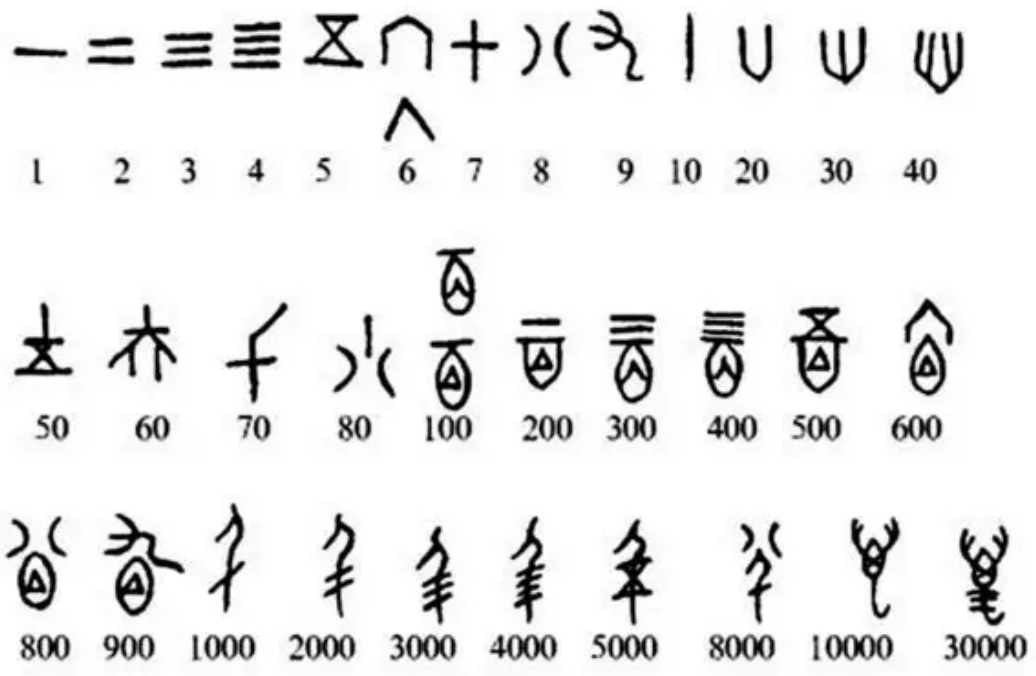
亚里士多德称人类普遍使用十进制，只不过是绝大多数人生来就有 10 根手指这样一个解剖学事实的结果。

也就是说，人类对于数字有了认识地直接原因是直立行走而解放了我们的双手，从而才有了可以数数的手指。

所以，我们从小就用手指头来帮助计算，其实是一种遗传自远古祖先的本能。

同样的，有十进制，就必然有五进制，就是看在计算时，习惯运用一只手还是两只手。

实际上，如果单纯的按照用双手帮助计算的角度，五进制比十进制更实用。因为在使用五进制时，你的左手数到大于五的时候，你的右手可以伸出一个手指当做一个五，从而双手最多能数到 30，只有到 30 以上时才需要借助其他东西来帮助。而使用十进制时，同样的情况只能数到 10。



甲骨文十进制

而对美洲印第安人的几百个部落所进行的研究也证实了这一点：

将近三分之一的人在使用十进制，另有大约三分之一的人在使用五进制或者五进制和十进制混合使用，剩下的不到三分之一的人在使用二进制，而那些使用三进制的人则不到百分之一。二十进制（以 20 为基数），出现在大约百分之十的部落中。

因此，我们现在十进制成为主流而不是五进制的原因，是因为当脱离手指，进行记录的时候，十进制对五进制的优势就很明显了。

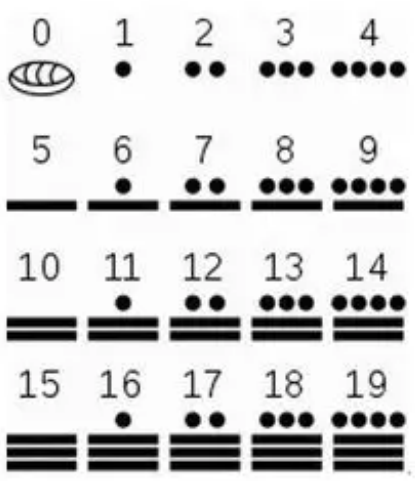
另外，“eleven”（11）和“twelve”（12）最初的意思应该是“one over”（大于 1）和“two over”（大于 2），这也表明了十进制在早期的支配地位。

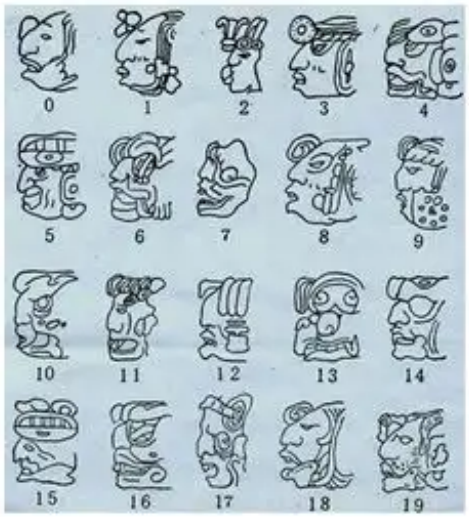
二十进制

二十进制是一种玛雅人创造的独特记数体系。这种体系的根源也无法得知。所以很多学者也按照亚里士多德对十进制论述的逻辑，认为二十进制是以 10 根手指加 10 根脚指为基数的记数方法。

不过，对于这种说法，每次想象到玛雅人掰着脚趾头数数或者计算时的画面时，总有一丝亲切感，因为我小时候也这样做过。这也是本能使然！

今天的法语中，vingt 是 20 的意思，而 quatre vingt 是四个 20，意思是 80，所以被认为是二十进制的残留。

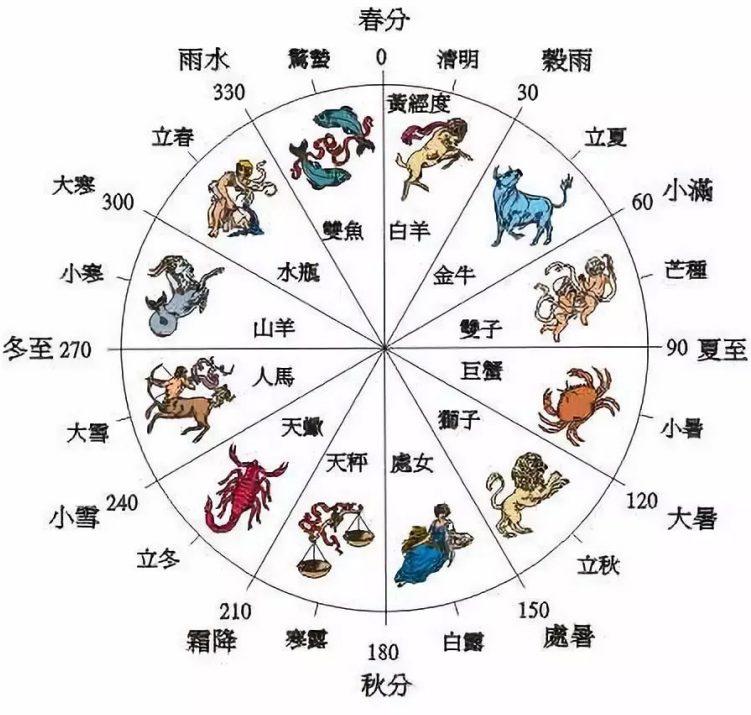




● ×8000=8000
● ×400= 400
● ×20= 20
● ×1= 1
8421

雅玛数字有两套表示方式：横点和头像。竖式位算法，20进制。

玛雅二十进制



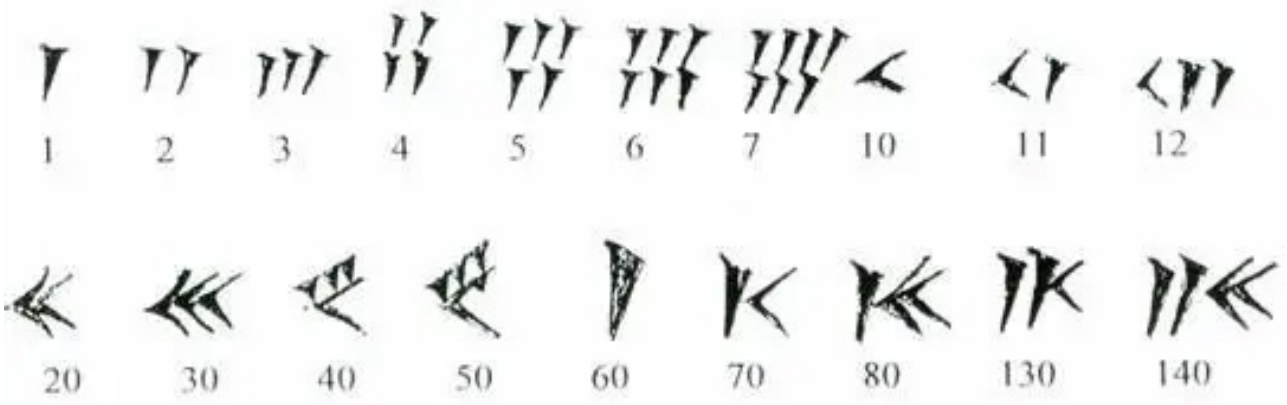
十二进制

关于十二进制，一般的文明都使用这种进制来作为记时的计量方法。　　这或许是由于天体运行周期的原因，比如，一年中月球绕地球转十二圈，还比如将白天夜晚分别划分为 12 部分。黄道十二宫、十二星座、十二地支、十二生肖、二十四节气、二十四小时都可以算这个体系下的具体应用。

不过，还是有人喜欢从生物学角度去理解这个问题，认为这和人类一只手有十二节指骨有关（不包括姆指，一根手指有三节指骨）。

六十进制

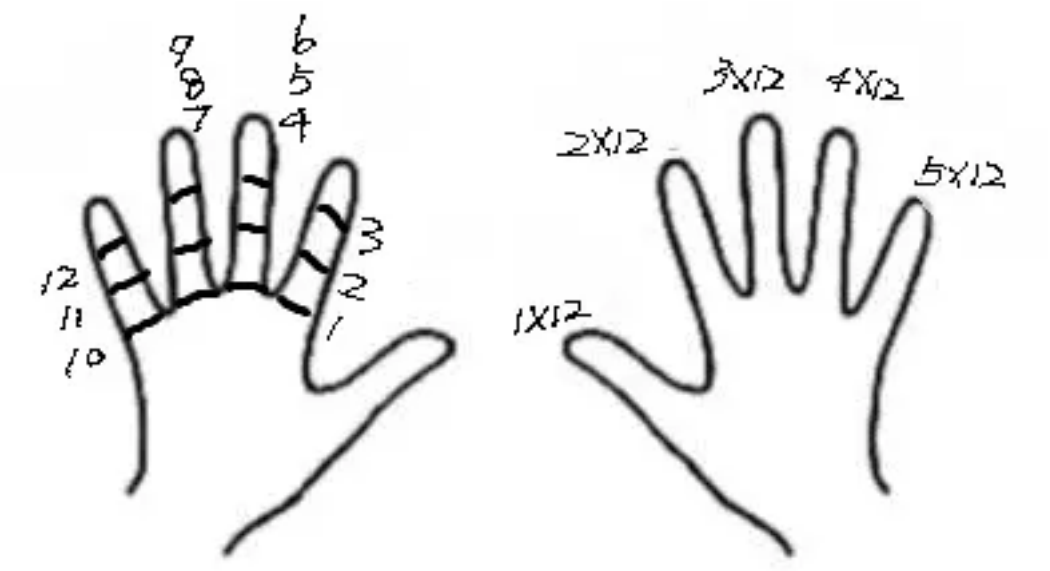
关于六十进制，也在我之前的文章中提到过。认为六十进制是为了在计算时（特别是涉及到几何计算时）可以轻而易举地细分为二分之一、三分之一、四分之一、六分之一、十分之一、十二分之一、十五分之一、二十分之一和三十分之一，这样就使得十位上能经得住尽可能的细分。



巴比伦六十进制

也有学者认为六十进制是两种更早进位制的自然组合：一种是十进制，另一种是六进制。让我奇怪的是，这次没有学者把这个进制和手指结合起来想。

不过，结合我前面说五进制时的脑补和十二进制关于手指的使用，我最终发现，六十进制也没有逃脱这个利用手指数数的逻辑框架。



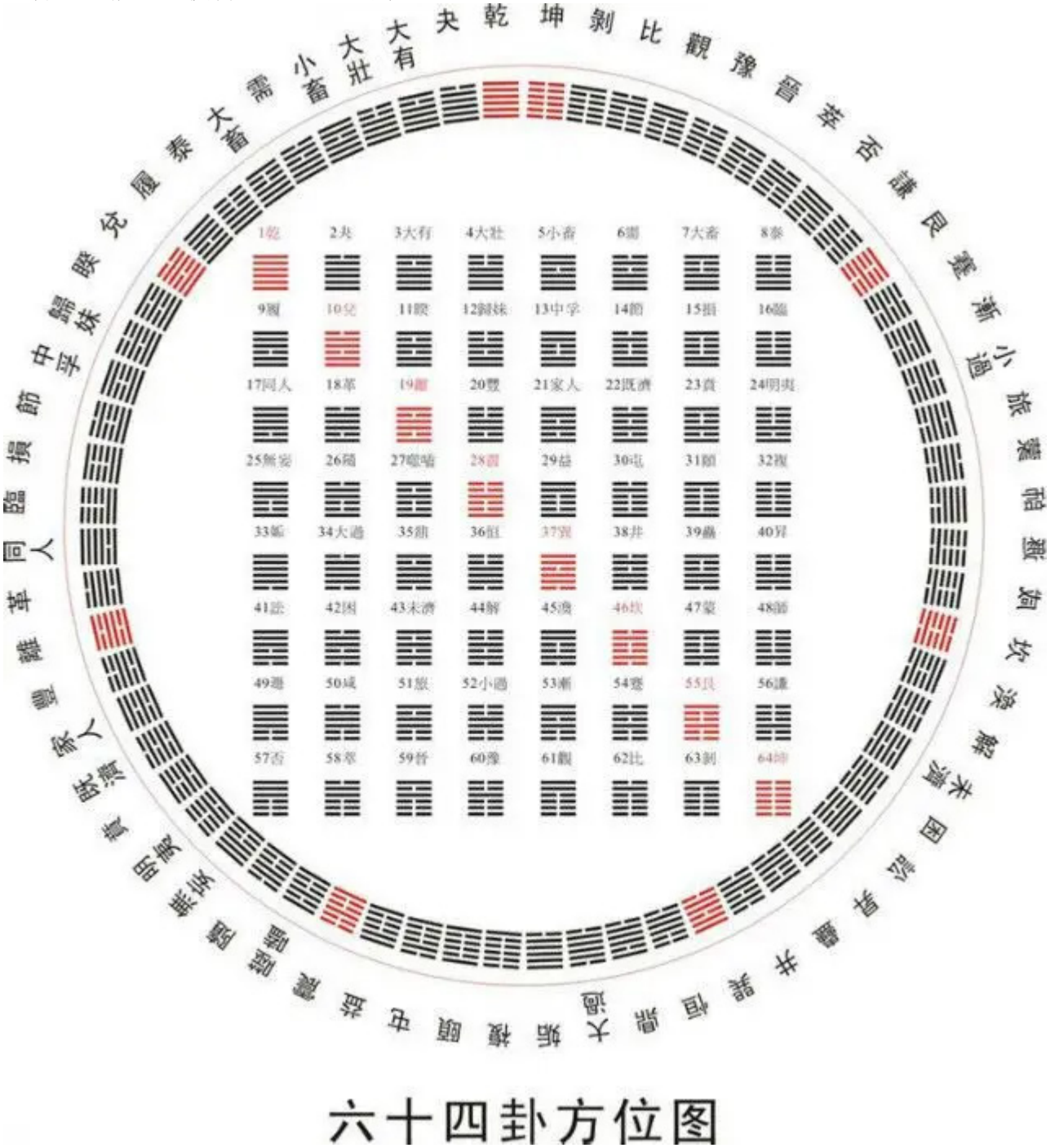
另外，尽管我们的社会形态是以十进制为基础，六十进制在现在依然还存在与时间和角的度量中。

二进制

说到二进制的起源，就不得不说一本伟大的书，和一个伟大的人。这本书就是《易经》，这个人就是莱布尼茨。关于《易经》，它在我们文化中的地位就像它二进制体系所传达的哲学象征一样——是一切的源头。这个“二进制”并非一种记数体系，虽然它也有“计算”的功能，但只是借由这种“计算”来，通神灵，知祸福。

而现代意义上的二进制，是由莱布尼茨发现的。关于他的这个发现是否是受到《易经》的启发，历来有很多争论。同时，他还设计了一台可以完成数码计算的机器，这是关于计算机的最初设想。

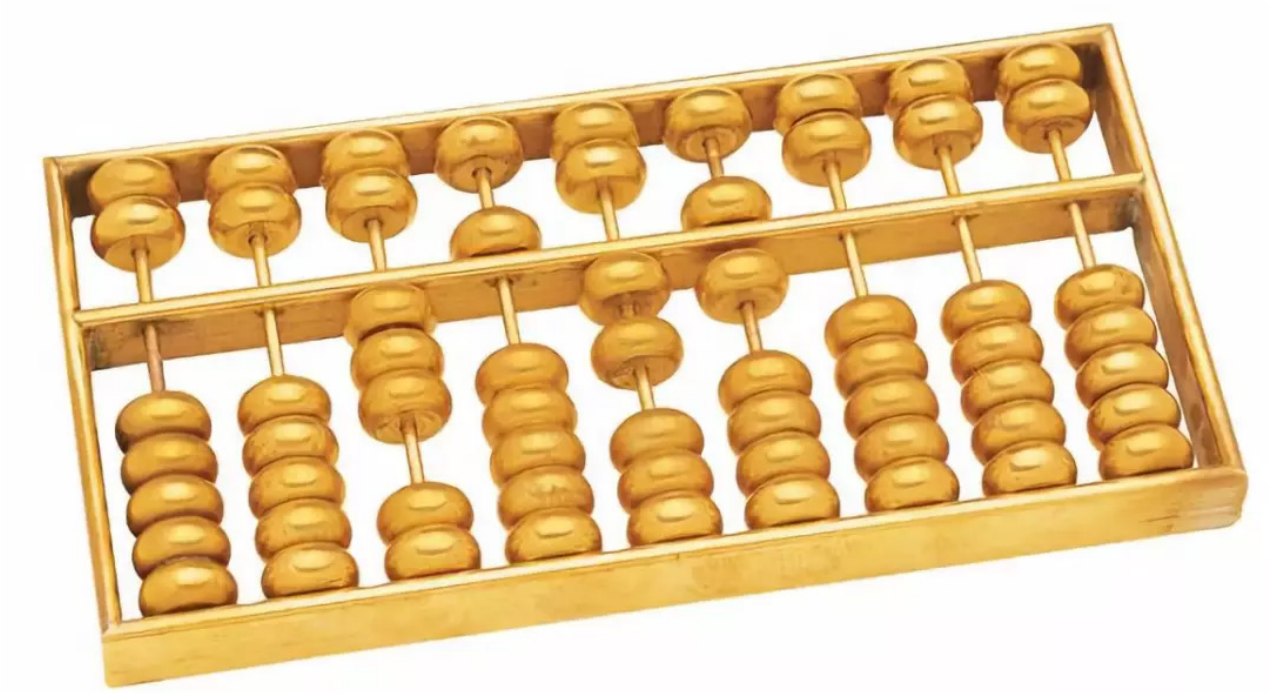
到现在，二进制不仅在基数上同《易经》一样是 0 和 1，并且因为最近流行的“万物皆比特”的概念，使得它在哲学的高度上也和《易经》统一了。



六十四卦方位图

四进制、八进制、十六进制

八进制和十六进制，最早采用的应该是中国人。“半斤八两”这个说法就是很好的一个佐证。对于“半斤八两”，有一种解释是：在古代定秤的时候，以天上的星星为准。北斗七星，南斗六星，福禄寿三星，总共十六星。所以，一斤为十六两，半斤既是八两。半斤八两的计算工具就是中国古人用了几千年的算盘。



对照上图，我们会发现算盘的设计其实是很有蹊跷的。

仔细想想的话，如果算盘是针对十进制而设计的话，那么上挡只需要一珠，下挡四珠，那就足够应付计算了，那古人为什么不将多出的两珠去掉？

然后，你用八进制的角度去看，因为不论上挡或下挡，若每珠代表数 1，那么每位的最大计数值是七，七正是八进制的最大基数。

再用十六进制的角度去看，若上挡每颗珠子代表数五，下挡每颗珠子代表数 1，那么每位的最大计数值是十五，十五正是十六进制的最大基数。

所以，算盘是用来计算十六进制的，而不是现在的十进制。只不过十进制也能用它来勉强计算而已。现在，“半斤八两”和算盘都已经成为历史概念和物品了。

不过，我依然还会记得用算盘一遍一遍地从 1 加到 100，一直到 5050 出现在算盘为止。

现代意义上的八进制和十六进制，再加上四进制，这三种进制体系都是和从二进制直接演化过来的。

其中四进制现在已经基本弃用。而八进制和十六进制还广泛地应用在计算机领域，其主要原因是因为它们能够和二进制几乎无缝地兼容。

另外，在印欧语系中，8 的单词是源自于 4 的倍数形式，在拉丁文中，表示 9 的 novem 也很有可能是与 novus（新的）有关的，意思是一个新序列的开始。这样一些单词也许可以解释为：它们暗示了，四进制和八进制的记数法曾经持续存在过一段时间。