让计算机学做二进制乘除法

摘要：运用模仿学习算法，让计算机在人的指导下，学会做二进制乘除法。进行二进制乘除法运算的过程中，需要涉及一些条件判断，在不同的条件下执行不同的命令。而模仿学习的主要任务，就是让计算机通过分析、比较各个样本（由用户输入），明白什么情况下该执行该执行什么命令。

关键词：模仿学习、谓词逻辑

作者：蔡超联

实验编号：33

编程语言：C++

实验成功日期：实验3年（公元2018年）4月1日

目录

[〇、关于模仿学习 1](#_Toc510703224)

[一、元操作及元谓词 3](#_Toc510703225)

[1.一元函数 3](#_Toc510703226)

[2.二元函数 3](#_Toc510703227)

[3.零元谓词 3](#_Toc510703228)

[4.一元谓词 3](#_Toc510703229)

[二、寄存器及状态命题 4](#_Toc510703230)

[三、任务A：让计算机学会做二进制乘法 5](#_Toc510703231)

[1.二进制乘法的计算过程 5](#_Toc510703232)

[2.训练过程 6](#_Toc510703233)

[（1）第一个样本：1101X1011 6](#_Toc510703234)

[（2）第二个样本：1110X101 9](#_Toc510703235)

[3.检验效果 10](#_Toc510703236)

[四、任务B：让计算机学会做二进制除法（恢复余数法） 11](#_Toc510703237)

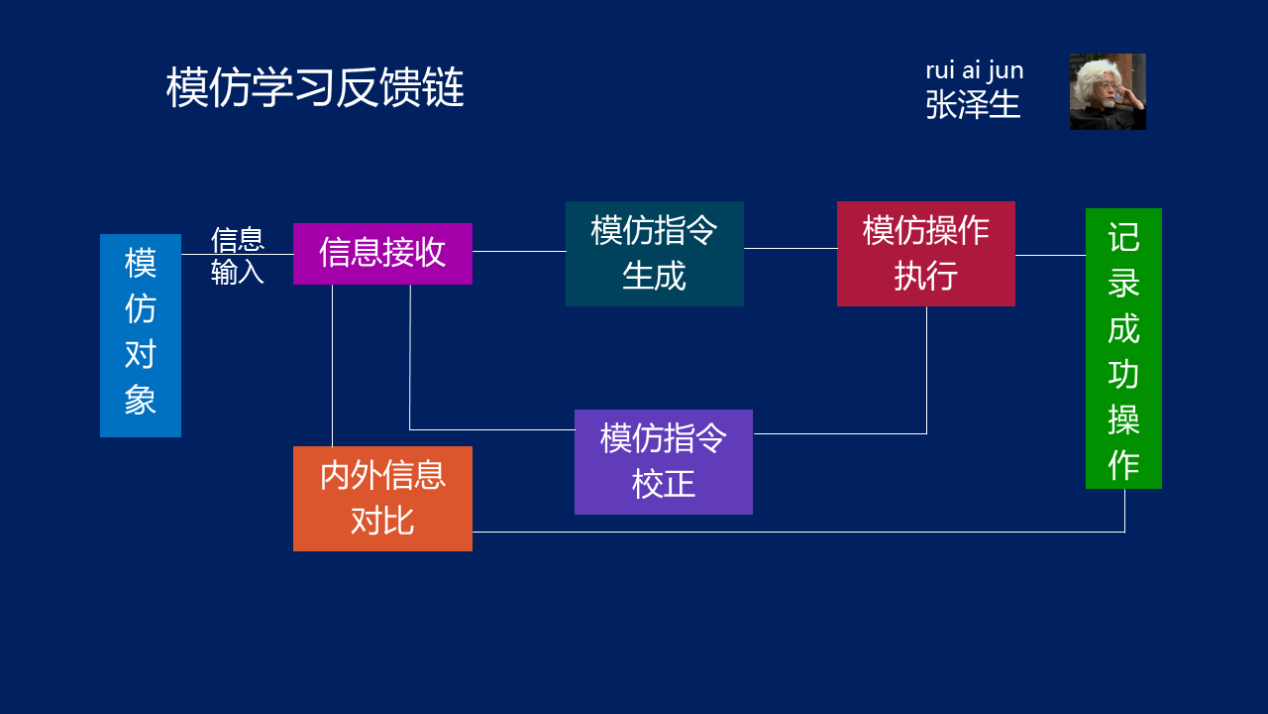
[1.二进制除法的计算过程 11](#_Toc510703238)

[2.训练过程：以1011÷1101为例 14](#_Toc510703239)

[3.检验效果 15](#_Toc510703240)

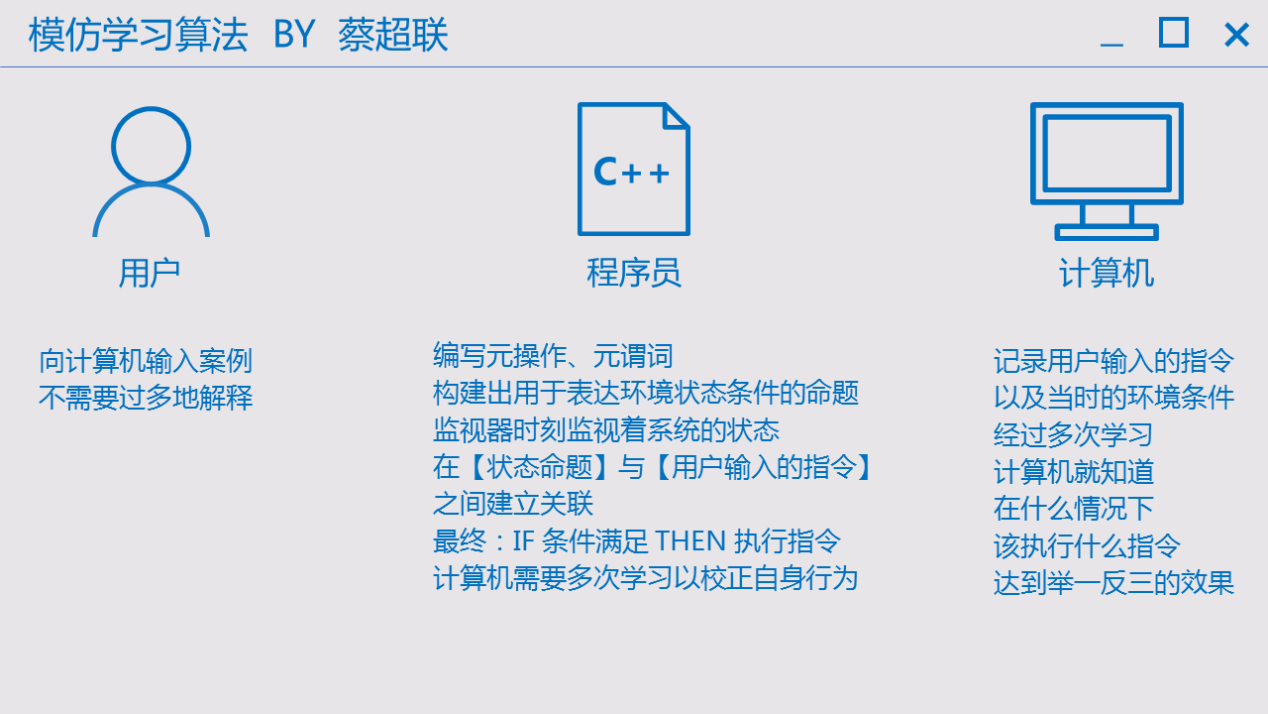
# 〇、关于模仿学习

模仿学习过程最早是由ruiaijun先生提出的，他给出的定义是：模仿学习就是一个机体按照外界某种活动的模式进行模仿，从而产生自身活动方式的过程。模仿学习是人类向长辈、老师学习的主要方式，它大大降低人类个体自己摸索学习知识的时间。人工智能也需要大量通过模仿学习获取知识，获取操作能力。



模仿学习，最关键的一点就是“举一反三”。老师的示范，说到底只是一种“样本”，具有特殊性。学生真正要学会解决一类问题，还得对老师给出的“样本”进行分析，找出一般性的规律，将其转化为自身的经验。

我对ruiaijun提出的模仿学习过程进行了进一步的抽象，如下图所示：



在PC机上，用户通过键盘输入命令，来向计算机灌输自己解决问题的经验（样本）。在不同的情境下（或者不同的题设条件下），用户输入的命令是不同的。计算机所要做的事情，就是：记录用户输入的指令以及当时的环境条件。比如用户输入命令C，此时的环境条件是S1={P1，P2，P5，P6，P7}。用户下一次输入命令C时，环境条件S2={P1，P2，P3，P4，P5，P8}。计算机需要对S1、S2两个集合做交运算，得到S={P1，P2，P5}，并将S集合与C命令之间建立起关联，表示当S集合下的所有命题都成立时，才应当执行C命令；换句话说，S集合内的条件的合取命题（做and运算），是执行C命令的前提条件。

# 一、元操作及元谓词

## 1.一元函数

（1）左移：将寄存器左移一位，低位补0

（2）右移：将寄存器右移一位，高位补0

（3）加1：使寄存器的值加上1

## 2.二元函数

（1）加：它接受两个参数A、B，效果是：使得A等于A+B。本程序没有减法运算，根据计算机补码的原理，减去一个数等于加上它的补码。

## 3.零元谓词

（1）首次运行：如果循环计数器C=1，表示当前进行的是第一轮循环，则返回true，否则返回false。

（2）不是首次运行

## 4.一元谓词

（1）正数：如果寄存器的值不等于0，且最高位为0，则返回true

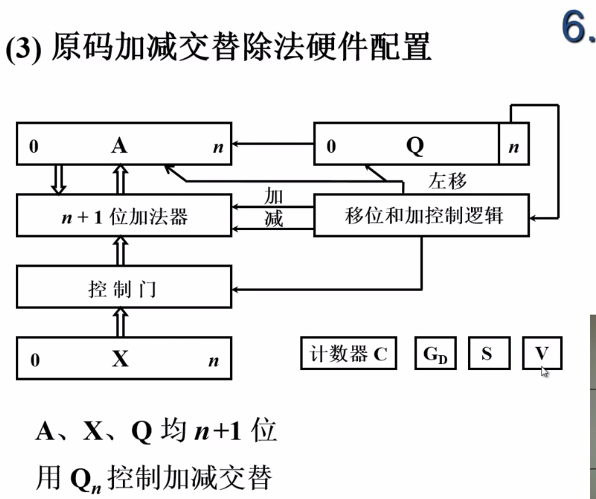
（2）负数：如果寄存器的最高位为1，则返回false。

（3）非负：如果寄存器的最高位为0，则返回true。（在加减交替法实现除法运算的试验中，会用到这个谓词）

（4）以0结尾：如果寄存器的最低位是0，则返回true。

（5）以1结尾：如果寄存器的最低位是1，则返回true。

# 二、寄存器及状态命题



A：用于存储余数，它的初始值==被除数

Q：用于存储商

X：用于存储除数

AQ：A和Q共同构成AQ，就如同AH和AL共同构成AX寄存器；如果对AQ进行左移操作，会同时造成A和Q的值左移。

C：计数器，相当于for循环里的计数变量，初始值为0。预先规定循环的次数就是被除数的位数。禁止对计数器进行加减操作。

注意：

本实验中，乘法和除法共用一套电路。Q寄存器也可存放乘数。

零元谓词本身可以单独构成一个命题，如：首次运行、不是首次运行。

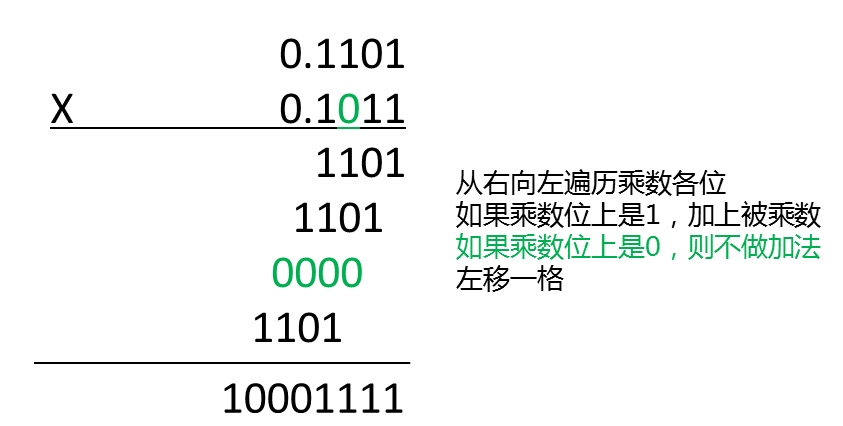
一元谓词和一个寄存器结合，可构成一句描述状态的命题，如：

以1结尾（Q）、正数（A）、非负（AQ）。

# 三、任务A：让计算机学会做二进制乘法

## 1.二进制乘法的计算过程

下图展示手算二进制乘法的过程



计算机算二进制乘法的过程，和人的计算过程有些类似：

数值部分按原码1位乘法规则计算

1. 从低到高取乘数中1位判断；
2. 为0不做，为1加被乘数；
3. 上一步结果右移一位。

循环第一步，直到乘数每一位判断完毕。



最后算出来的积，就是AQ寄存器的值。

乘法运算接受X、Y两个乘数，运算结果就是积，不用多说。

计算机计算二进制乘法的过程，可以写成如下形式：

1. 如果（寄存器Q的末位是1） 则 P加上X
2. AQ寄存器整体右移一位

第二步运算是无条件操作，在任何情况下都会执行；

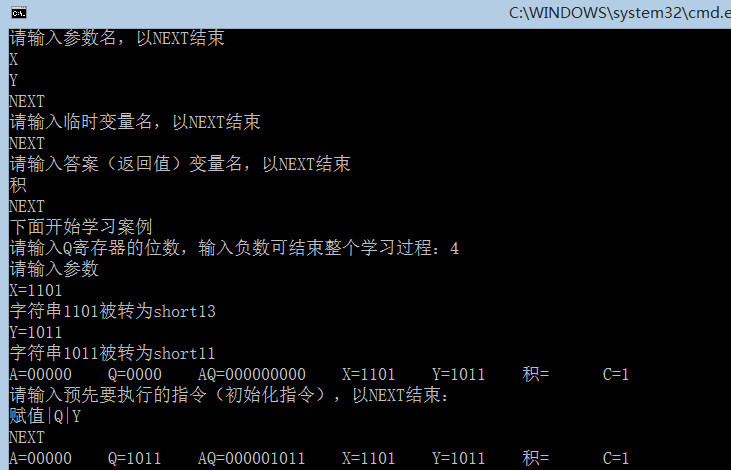
第一步运算是有条件的，仅当寄存器Q末位是1的情况下才会去执行。

## 2.训练过程

### （1）第一个样本：1101X1011

我们要计算的是0.1101×0.1011，只需要输入小数部分就可以了，X=1101，Y=1011.

Q寄存器的长度设为4位。A寄存器自动被扩展到5位，最高位用于处理溢出。



在循环之前，先令寄存器Q的初值为Y。输入命令：赋值|Q|Y。在本实验中，函数和参数、参数和参数之间用管道号（|）分隔。

赋值以后，我们看到Q寄存器的值变为1011，AQ寄存器的值变为000001011.

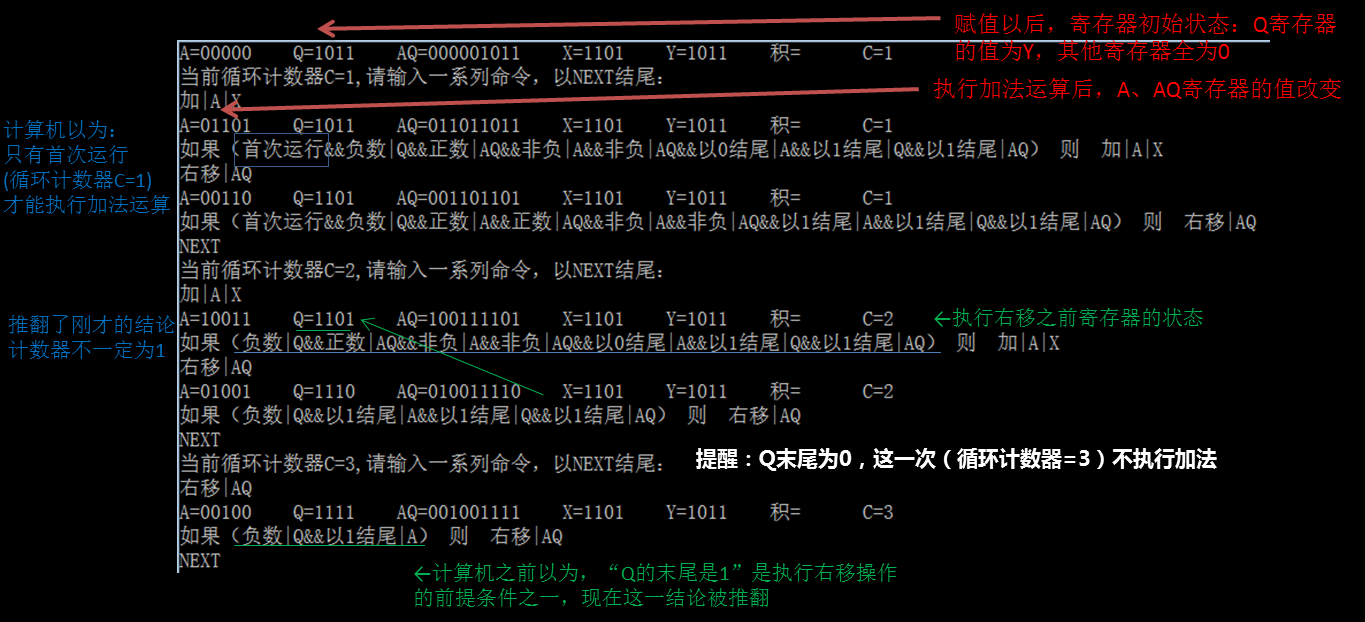
现在开始第一轮循环，控制台显示循环计数器C等于1.

输入命令：加|A|X

你键入命令的这一刻，计算机就开始对状态进行监测，它发现以下状态命题为真：

1. 首次运行（即C==1）——无意义，稍后排除
2. Q寄存器为负数（在原码和补码里面，最高位为1就是负数）——这个对本实验没有意义
3. AQ寄存器非负、A寄存器非负——无意义
4. AQ寄存器为正数——无意义
5. A寄存器以0结尾——无意义
6. Q寄存器以1结尾
7. AQ寄存器以1结尾（命题⑥成立，则此命题必然成立，反之亦然）

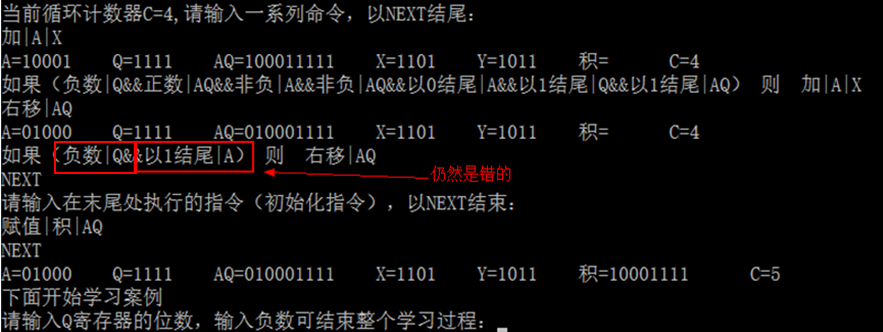
接下来，输入命令：右移|AQ。由控制台输出可以看出，计算机以为仅当Q寄存器以1结尾时，才能执行右移操作。稍后，它会通过学习样本，来推翻这一结论。



如上图所示，在第2轮循环中，我们又向计算机输入了【A寄存器加上X】的指令，这个时候，计算机会对当前状态进行检测，它发现“首次运行”这个命题不再成立了，因为当前已经不是第一轮循环了，于是它便推翻了这一结论，将“首次运行”这一状态命题从【前提条件】里删去。

在第3轮循环中，一开始Q寄存器的值就是1110，它的末位是0，因此这次不执行加法运算，直接右移。

计算机之前以为，“Q的末尾是1”是执行右移操作的前提条件之一，现在这一结论被推翻。



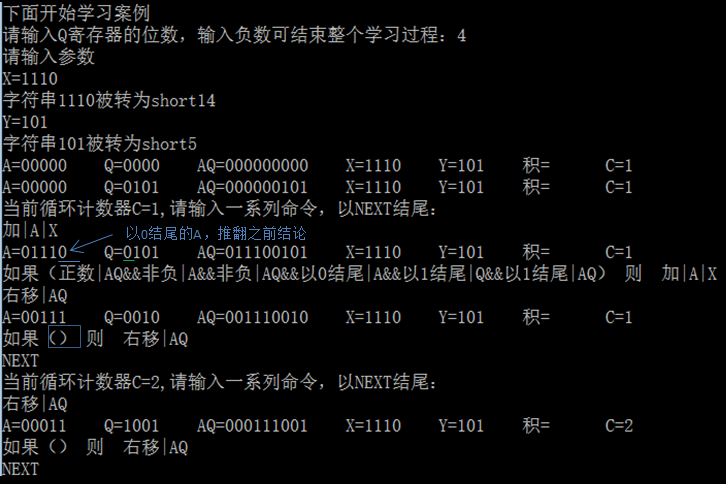
如上图所示，接下来又进行了第四轮循环，用户输入最后一步语句“右移|AQ”以后，计算机仍然认为“Q的最高位是1”和“A的最低位是1”是执行该命令的前提条件。为什么它会犯这种错误呢？

我们一开始规定，存放乘数的寄存器的长度为4位，我们输入的乘数Y=1011，刚好占满4位，最高位是1.

如果在Q寄存器长度不变的情况下，我们输入一个长度为3位的乘数，比如101，将它放入Q寄存器以后最高位自动补零，这样“Q的最高位为1”这一结论就被推翻了。

同理，我们输入一个结尾为0的被乘数（如1110），将它放入A寄存器里，如此A寄存器的末位就是0，这样一来“A的最低位是1”这一结论也就被推翻了。

### （2）第二个样本：1110X101





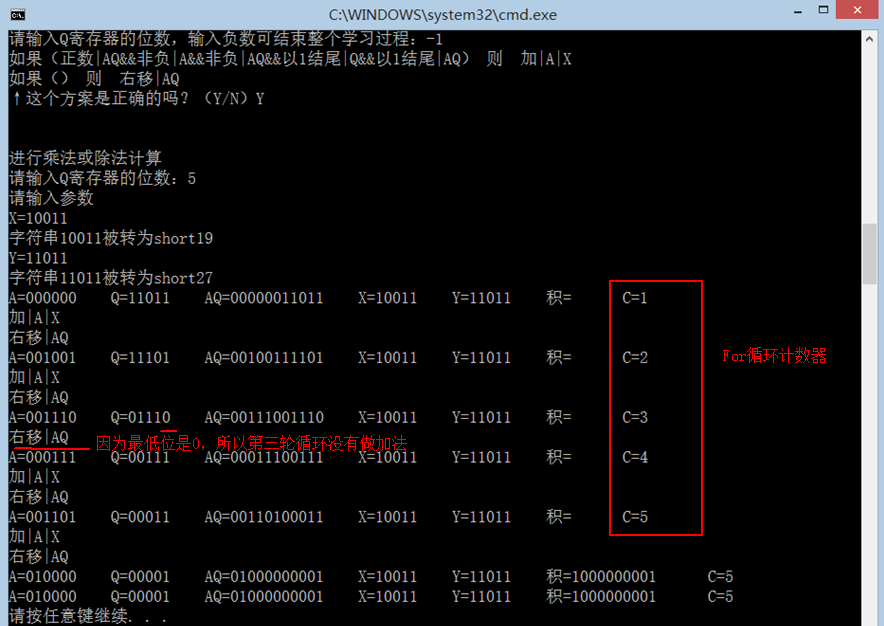
学习完毕，结果仍然有点不对劲，为什么计算机认为“A寄存器为正数（最高位为0）”是执行加法运算的前提条件之一？



看一下这张图就明白了。执行上一步加法之后，A寄存器的最高位的确有可能是1.但是每次执行加法以后必须右移，右移以后最高位会被填充为0，这样一来A寄存器的最高位只能是0而不可能是1，这是符合事实的。

## 3.检验效果

现在让计算机计算11011X11011，即19X27

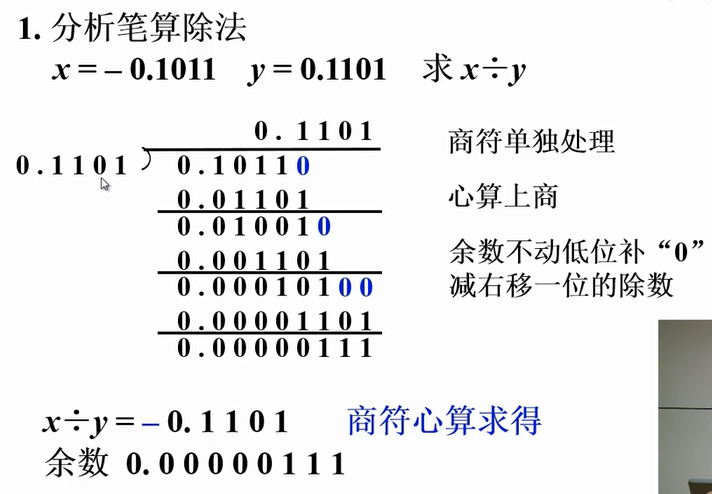


运算结果为10 0000 0001，正确。

# 四、任务B：让计算机学会做二进制除法（恢复余数法）

## 1.二进制除法的计算过程

计算机里的除法运算是参照竖式除法运算的过程，采用位移和加减法运算（减一个数，实际上就是加上它的补码）的迭代过程实现的。首先我们来看一下竖式除法运算的过程。



计算机做两个正数的除法，参照竖式笔算的过程。但是有以下几处不同：

1. 笔算中，比较被除数和除数的大小是根据人眼观察实现的，而计算机中需要经过减法运算进行。
2. 如果被除数减去除数，所得余数为正，上商1；所得余数为负，上商0（本实验中寄存器左移一位后面自动补零，所以上商0等于什么也没做）。笔算中余数添0以后和除数比较，而计算机采用部分余数左移后和除数比较，左移出界的部分余数的高位都是0，对运算不会造成影响。
3. 计算机进行定点数除法运算时，要求被除数必须小于除数，因为，被除数大于等于除数时，商为带小数，会发生溢出。☆

“恢复余数法”求X/Y的商q和余数R的方法如下：

* 1. R1=X-Y

如果R1<0，则上商q0=0。然后恢复余数：R1=R1+Y

如果R1>=0，则上商q0=1。这时被除数X大于等于Y，做溢出处理。

* 1. 如果第i次的部分余数为Ri，则第i+1次的部分余数为R(i+1), R(i+1)=2\*Ri-Y

如果R(i+1) <0，则上商0，同时恢复余数：R(i+1)=R(i+1)+Y

如果R(i+1) >=0，则上商1

* 1. 循环执行第2步，直到求得所需位数的商。

线性化的描述：

for(C=1 to 除数的位数)

{

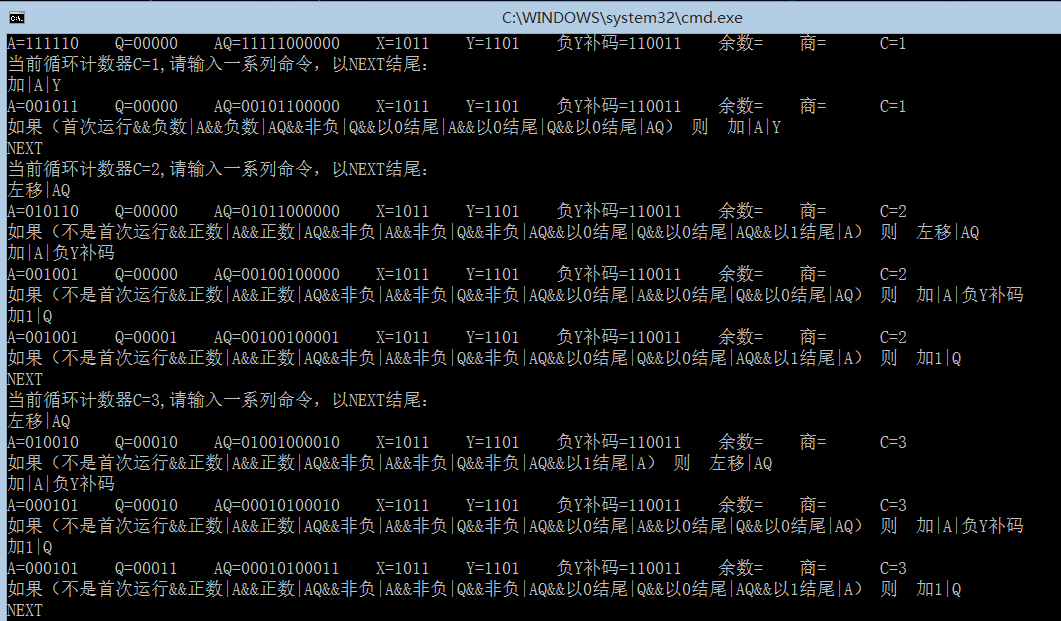
1. 如果(C≠1) 则 左移AQ；
2. A减去Y；//A加上Y的补码
3. 如果（A>0） 则 Q加上1；//上商1
4. 如果（A<0） 则 A加上Y；//恢复余数

}

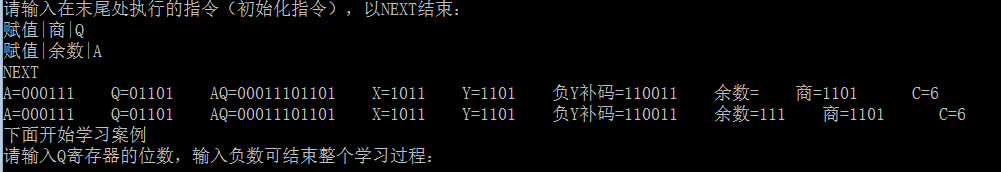


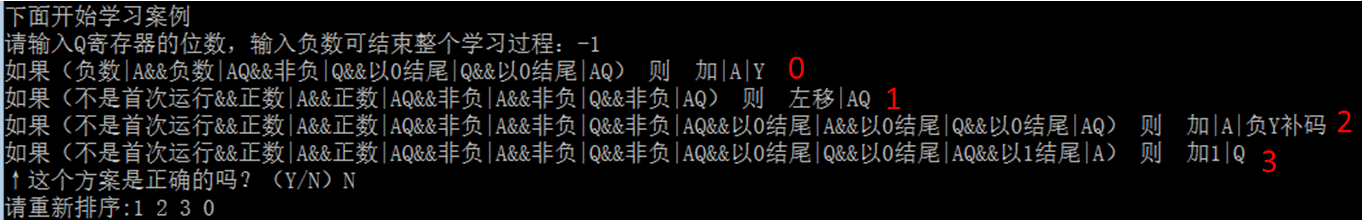
## 2.训练过程：以1011÷1101为例





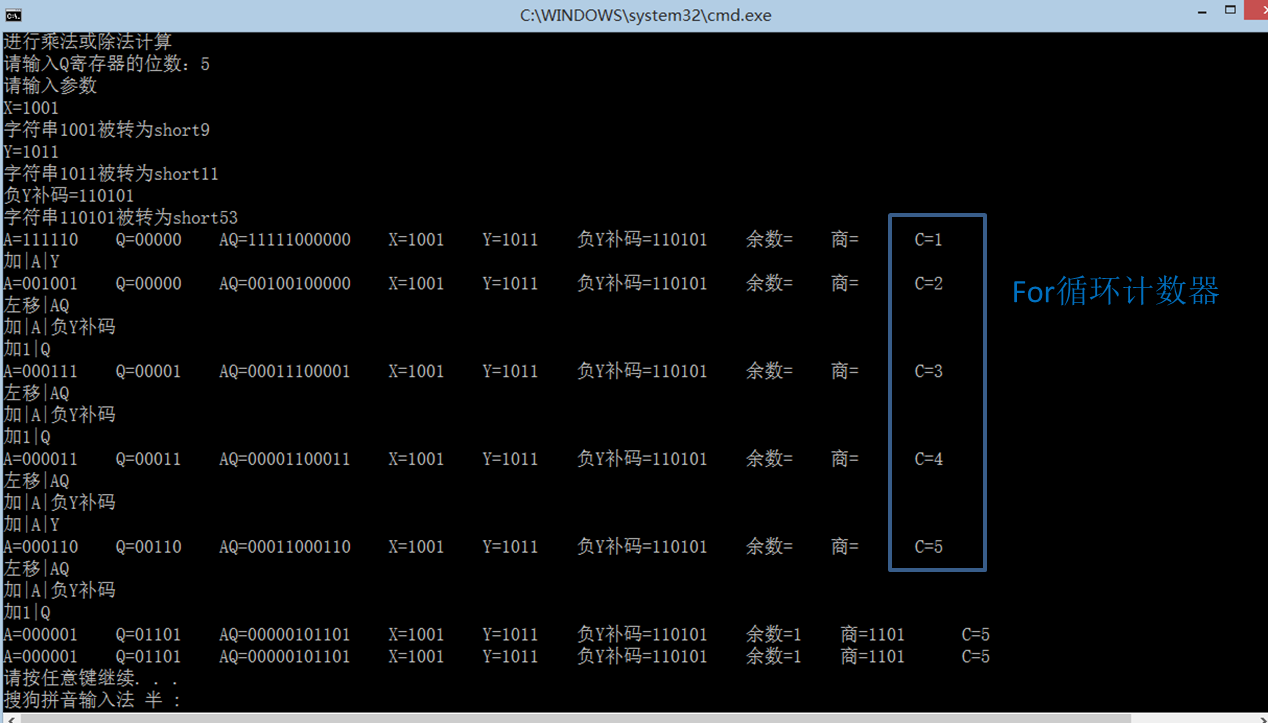






## 3.检验效果

计算1001÷1011



计算10111 ÷ 11011

