



中國人民大學

RENMIN UNIVERSITY OF CHINA

信息学院

SCHOOL OF INFORMATION

程序设计荣誉课程

1. 基础2——计算机与程序运行

授课教师：游伟 副教授、孙亚辉 副教授

授课时间：周二14:00 – 15:30，周四10:00 – 11:30（教学三楼3304）

上机时间：周二18:00 – 21:00（理工配楼二层机房）

课程主页：<https://www.youwei.site/course/programming>

目录

1. 计算模型与体系结构
2. 计算机系统的组成
3. 信息在计算机中的表示
4. 程序与程序设计语言
5. C/C++语言介绍

1.1 计算模型与体系结构

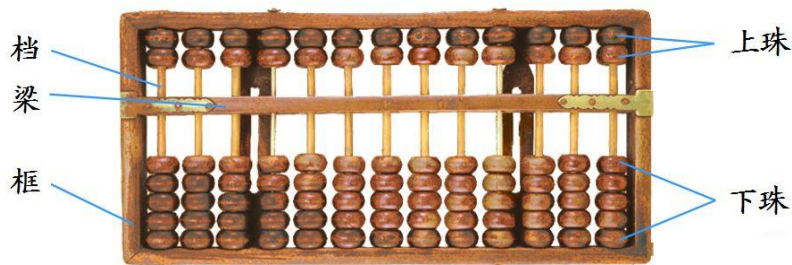
■ 计算：对目标对象的数据或信息进行表示、加工、变换和展示

- 数值计算
- 逻辑计算
- 更复杂的计算



■ 计算模型：刻画计算的抽象的形式系统或数学系统

- 图灵机模型 (TM)：通用计算模型，可描述任何理论上可计算的问题
- 随机存取机模型 (RAM)：与图灵机模型等价，但操作更简单



加法口诀 (+3)

- 三上三
- 三下五去二
- 三去七进一

更多内容：《计算理论导论荣誉课程》（第6学期）

图灵机模型 (TM)

■ 直观描述:

- 3个部件: 有限状态控制器、无穷线性带和读写头
- 3个动作: 改变控制器状态、改写当前格符号、将读写头左移/右移一格
- 1套规则: 根据当前格局 (当前状态+当前格符号), 决定采取什么动作 (当前状态改变+当前格符号+读写头移动)

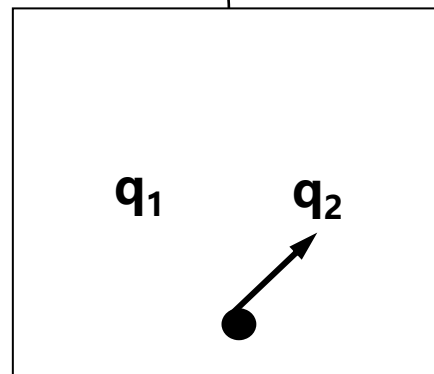
规则

当前格局	采取的动作
(q ₂ , a)	(q ₁ , b, R)
(q ₂ , b)	(q ₁ , a, R)
(q ₁ , b)	(q ₂ , a, L)
(q ₁ , a)	(q ₂ , b, L)

纸带



读写头



状态控制器

图灵机模型 (TM)

思考：

- 如果用十进制表示 x ，情况如何？
- 如何用图灵机模型计算 $x+y$ ？

■ 示例：用TM模型计算 $x+1$

■ 编码：用二进制表示 x ，用 $*$ 与符号带上的其它符号分隔开

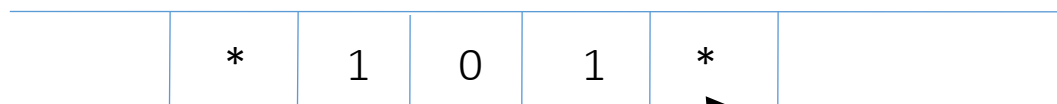
■ 符号集：0, 1, *

■ 状态集：start (初始)，inc (增加)，noncarry (不进位)，
carry (进位)，overflow (溢出)，halt (停止)

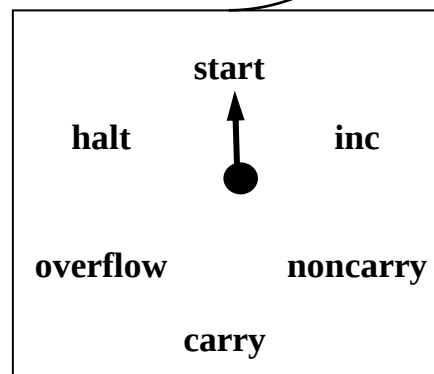
规则

当前格局	采取的动作
(start, *)	(inc, *, L)
(inc, 0)	(noncarry, 1, L)
(inc, 1)	(carry, 0, L)
(inc, *)	(halt, *, -)
(noncarry, 0)	(noncarry, 0, L)
(noncarry, 1)	(noncarry, 1, L)
(noncarry, *)	(halt, *, -)
(carry, 0)	(nocarry, 1, L)
(carry, 1)	(carry, 0, L)
(carry, *)	(overflow, 1, L)
(overflow, 0/1/*)	(halt, *, -)

纸带



读写头

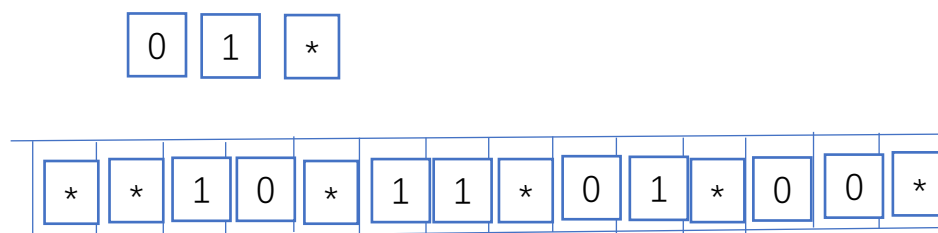


状态控制器

示例：计算 $5+1$

- 状态集: start0, start1, noncarry, carry, inc, noinc, dbinc, halt
- 规则:

(start0, *)	(noncarry, *, L)
(start1, *)	(carry, *, L)
(noncarry, 0)	(noinc, 0, L)
(noncarry, 1)	(inc, 0, L)
(carry, 0)	(inc, 0, L)
(carry, 1)	(dbinc, 0, L)
(noinc, 0/1)	(start0, 0/1, L)
(inc, 0)	(start0, 1, L)
(inc, 1)	(start1, 0, L)
(dbinc, 0/1)	(start1, 0/1, L)
(noncarry, *)	(halt, *, -)
(carry, *)	(halt, 1, -)



e.g. 计算12+6

by石岩

图灵机之 x+y

• 样例: (DEC)11+6=17
(BIN):1011+110=10001

• 状态: sta (开始【初始位置】), c(进位1),c2 (进位2)
[3], uc (不进位), halt (停止)

当前格局	动作
(sta,*)	(uc,*,L)
(uc,0)	(uc,0,L)
(uc,1)	(c,1,L)
(uc,0)	(uc,0,L)
(uc,1)	(uc,1,L)
(c,0)	(c,0,L)
(c,1)	(c2,1,L) [4]
(c,0)	(uc,1,L)
(c,1)	(c,0,L)
(c2,0)	(c,0,L) [4]
(c2,1)	(c,1,L) [4]

- 思源:
 1. 模仿人类传统手算法: 相信自然选择, 大概简便
 2. (高精度加法)

	*[2]	1	0[1]	0	1	1	1	1	0	*
--	------	---	------	---	---	---	---	---	---	---

(c,*)	(halt,1,-)
(uc,*)	(halt,*, -)

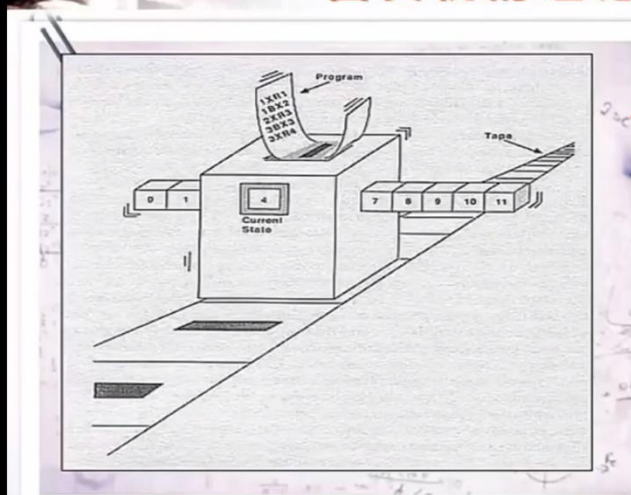
- 说明:
 - [1]: 补齐
 - [2]: 终止符/只写一个因为每位相加后至多进一位。
证明的思路: 设相加的两个数(二进制)中, 位数较多的数有n位, 则另一个数最大为1……11 (共n个1), 该数最大亦为1……11 (共n个1), 和为100……00 (共n个0), 则有n+1位。故知, 最多位数为n的两个数之和不超过n+1位。
 - [3]: 所谓进几, 包含了“_”的加数。
 - [4]: 注意到“_”与“ ”交替, 由这三处可知不可能以“c2”的状态走到“_”或“*”。

基本组成部分：输入带、读写头、状态寄存器、控制规则
工作原理：如何通过状态转换和读写操作来处理信息

by谢骏希



图灵机的理论意义



- 可计算性的判定；
- 意义：
 - ◆ 给出了一个可实现的通用计算模型；
 - ◆ 引入了通过“读写符号”和“状态改变”进行运算的思想；
 - ◆ 证实了基于简单字母表完成复杂运算的能力；
 - ◆ 引入了存储区、程序、控制器等概念的原型；

二进制图灵机

- ```

* q0.0.0.Rq0q.1.1.R.R1R0.0.0.ROR0.1.1.R.R1R0.##.R.RR0R1.0.0.ROR1.1.1.R.R1R1.##.R.RR1R0.+,+R.RR0R1
+,+R.RR1R0.0.0.RORR0.1.1.R.R1R0.##.R.RR0R1.0.0.R.R1R1.1.1.R.R1R1.##.R.RR1R0.+=,=L.LL
R1.+=,=L.LL.R0R.F.F.LLORR0.T.T.LLORR1.F.F.LL.LL.R1.T.T.LL.LL0.Q.LL.LL0.L.LL.LL0.L.LL.LL1.L.LL.LL1.L.F.L
.LL.LL.0.L.LL.LL1.L.LL.LL1.Q.L.LL.LL+.0.L.Ltag.LL+.1.L.Ltag.LL.##.L.Ltag.Ltag.0.##.L.Ltag.Ltag.0.##.L
.Ltag.LL.1.L.LL.0.L.LL1.L.LL.LL.B.B.R.q0q0.##.R.NextPartNextPart.##.R.NextPartNextPart.0.##.R.NextPartNextPart
.1.##.R.printPrint0.F.F.printPrint0.T.T.R.printPrint0.=.R.printPrint0.0.0.R.printPrint0.1.1.R.printPrint1
.F.F.R.printPrint0.T.T.R.printPrint1.=.T.T.R.printPrint1.0.0.R.printPrint1.1.1.R.printPrint0.B.0.L.backprint1.B.
.1.F.backback.0.0.L.backback.1.1.L.backback.+=.L.backback.F.F.L.backback.T.T.L.backback.##.R.continuecontinue
inue.F.R.printContinue.T.T.R.printContinue.+=.R.qa

```

### Problem2 四位二进制数加法

- 设计一个可以完成4位二进制数加法的图灵机。
- 格式要求：
  - 输入：(4位0或者1)+(4位0或者1)=
  - 输出的“4位0或者1”表示要做加法的二进制数
  - 停机时纸带：...####(5位0或者1)B###...
  - 停机时纸带上的“5位0或者1”作为答案
  - 结果不足5位时注意加前导0
  - 起始位置应为输入中“=”位置右侧一格
  - 以 B 作为结尾
  - # 为 Γ 中任意字母。

### 题解

Part1. 每次读取前一个数的最后一位，加到后一个数的相应数位（已经得到结果的相应数位用“T”“F”覆盖（分别表示 1 和 0），便于迅速找到下一个要累加的数位），相加后直接覆盖第二个数的相应数位，然后回溯，回溯时注意考虑进位；由于需要固定输出 5 位数字（含前导 0），故在第一次个位累加完回溯时将“+”开辟为最高数位；回溯时我们顺带将第一个数已经取过的最后一位打上 # 标记，保证下一次取时就是 # 串前的最后一位。

Part2. 此时原先第二个数的五个数位上已经是所要求的结果，将其转移至“=”后即可。

[illegible]

### Problem3 任意位二进制数加法

- 设计一个可以完成任意位二进制数加法的图灵机。
- 格式要求：
  - 输入：(若干0或者1)+(若干0或者1)=
    - 输入的“若干0或者1”中**不含前导0**
    - **每一个操作数的长度大于等于1**
  - 停机时纸带：...###(若干0或者1)B###...
    - 停机时纸带上的“若干0或者1”作为答案
    - 不应含有前导0
    - 答案的起始位置应为输入中“=”位置右侧一格
    - 以**B**作为**结尾**
    - #为Γ中任意字母。

### 题解

整体思路完全与 Problem2 相同，此题需要注意的细节是输出时不能包含前导 0，所以我们考虑在每次回溯时对第二个数向前多“开辟”一个数位的空间（容易证明：新“开辟”的空间只会占用原先“+”的位置以及我们已经打上 # 标记的无用位置），最后转移时去掉前面的一堆前导 0 即可。（细节：注意 T/F 与 0/1 的两种情况都要写上）

符号集：0, 1, \*

状态集：start（初始），inc（增加），dec（减少），  
overflow（溢出），x（移至x），y（移至y），halt（停止）

| 当前格局      | 采取的动作          | 当前格局             | 采取的动作     |
|-----------|----------------|------------------|-----------|
| (start,*) | (dec,*,R)      | (x,0)            | (x,0,L)   |
| (dec,0)   | (dec,1,R)      | (x,1)            | (x,1,L)   |
| (dec,1)   | (x,0,L)        | (x,*)            | (inc,*,L) |
| (dec,*)   | (halt,*, -)    | (y,0)            | (y,0,R)   |
| (inc,0)   | (y,1,R)        | (y,1)            | (y,1,R)   |
| (inc,1)   | (inc,0,L)      | (y,*)            | (dec,*,R) |
| (inc,*)   | (overflow,1,L) | (overflow,0/1/*) | (y,*,R)   |

备注：为了简化规则，  
y倒序书写，即左侧为  
低位，右侧为高位，  
x正常书写

优缺点：规则应该较  
为简单，但y较大时  
整个流程耗时长，是  
加上了循环的x+1

实现二进制加法为什么难：

实现二进制加法的图灵机虽然看起来简单，但实际上涉及到一些挑战，主要原因如下：by陈一鸣

状态复杂性：

图灵机需要设计足够的状态来处理不同的情况，如读取每一位数字、处理进位、调整符号等。为了确保所有可能的情况都被涵盖，需要设计多个状态来处理这些任务，并在它们之间切换。

符号管理：

在图灵机中，除了处理二进制数字，还需要处理额外的符号，比如加号、空白符号等。这些符号的处理需要额外的状态和规则，以确保图灵机能够正确地读取和写入这些符号。

进位处理：

二进制加法涉及到进位的处理。图灵机必须能够跟踪进位并在必要时将其传递到前面的位。这要求图灵机在计算过程中能够在正确的时机移动到适当的位置，并正确地处理进位。

读写头的移动：

图灵机的读写头需要在纸带上进行精确的移动。实现二进制加法时，读写头必须能够在加数和结果之间准确地移动，且每次移动都要根据当前状态和读到的符号做出正确的操作。

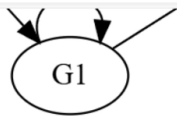
错误处理和调试：

设计和调试图灵机的状态转移表可能非常繁琐。任何状态转移错误或符号处理不当都会导致图灵机无法正确计算结果。因此，在实现过程中需要仔细验证每一步的逻辑。

效率：

虽然图灵机理论上能够完成所有计算任务，但它们可能不是高效的。在图灵机上实现复杂的操作（如二进制加法）可能会非常慢，尤其是在处理大型输入时

以上由chatgpt4o生成



## 图灵机模拟动画



|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| 1 | 1 | 0 | 1 | + | 1 | 1 | 0 | 1 | 0 | 1  | B  | B  | B  | B  | B  | B  | B  | B  |    |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

当前已使用规则: q3, B, B, L, qa  
当前将使用规则:  
当前步数: 0

OUT,T,B,R,OUT  
OUT,F,B,R,OUT  
OUT,+,B,R,OUT  
OUT,1,1,L,q2  
OUT,0,0,L,q2  
OUT,I,I,L,q2  
OUT,O,O,L,q2  
q2,B,B,R,q3  
q3,1,1,R,q3  
q3,0,0,R,q3  
q3,I,1,R,q3  
q3,O,0,R,q3  
q3,B,B,L,qa

1101+110101

符号：

1,0,B(Blank),T,F,I,0; //在"+"左边用T/F标记1/0, 在右边用I/0标记1/0;

状态：

SCAN //用于从左向右扫描第一个数至 "+"  
RE //用于从"+"开始向左越过所有已经被标记的数, 获得最靠右的未标记数, 并把这位的1标记为T, 0标记为F  
G0 //用于携带"0"至空格或者右数左起被标记的第一位  
G1 //用于携带"1"至空格或者右数左起被标记的第一位  
P0 //在右数未被标记的末位加0, 并在当位把1标记为I, 把0标记为0  
P1 //在右数未被标记的末位加1, 并在当位把1标记为I, 把0标记为0  
Carry //标记进位  
BACK //用于读写头从右数返回至"+"  
OUT //擦除两数和外的符号  
STOP //运行到此停机

输入规定：

输入形如xxxx+xxxx, 其中x为0或1; 并规定右边的数一定比前面的数位大;

之所以如此繁琐, 是因为没有对输入数据进行预处理, 而把**对齐**的操作交给图灵机来完成。这要求我们用不同符号区别**已完成加法**的位和**未完成加法**的位, 并有足够的状态和规则来处理它们

指令集:

SCAN,0,0,R,SCAN;  
SCAN,1,1,R,SCAN  
SCAN,+,+,L,RE

//左起向右扫描至"+", 向左一格切换模式为RE

RE,F,F,L,RE  
RE,T,T,L,RE  
RE,0,F,R,G0  
RE,1,T,R,G1  
RE,B,B,R,OUT

//从"+"开始, 向左越过所有的T/F, 获得左数未被标记的末位

G0,F,F,R,G0  
G0,T,T,R,G0  
G0,+,+,R,G0  
G0,0,0,R,G0  
G0,1,1,R,G0  
G0,B,B,L,P0  
G0,0,0,L,P0  
G0,I,I,L,P0

//向右至第一个空格或I/O, 向左一格切换模式为P0

P0,0,0,L,BACK  
P0,1,I,L,BACK

演示: 110+111  
得到: 11F+11I

P0,0,0,L,BACK  
P0,1,I,L,BACK

G1,F,F,R,G1  
G1,T,T,R,G1  
G1,+,+,R,G1  
G1,0,0,R,G1  
G1,1,1,R,G1  
G1,B,B,L,P1  
G1,0,0,L,P1  
G1,I,I,L,P1

//向右至第一个空格或I/O,向左一格切换模式为P1

P1,0,I,L,BACK  
P1,1,0,L,CARRY  
P1,+,I,L,RE

演示: 111+1101  
得到: 11T+111O

CARRY,0,1,L,BACK  
CARRY,+,1,L,BACK  
CARRY,1,0,L,CARRY

把P1切换为CARRY,保证了每次加法只有一位被标记

注意进位可能发生在+上,此时把+看作0

BACK,1,1,L,BACK

BACK,0,0,L,BACK

BACK,+,+,L,RE

RE,F,F,L,RE

RE,T,T,L,RE

RE,0,F,R,G0

RE,1,T,R,G1

RE,B,B,R,OUT

通过BACK回到加号，回到加号后切换状态为RE，重复之前的操作

!!!!!! 当进位发生在加号上，需要新的条件从BACK回到RE

故增加判断条件

BACK,F,F,L,RE

BACK,T,T,L,RE

当左数每一位都已经被读完，从左数最右端空白起，将T和F都写为空白

RE,B,B,R,OUT

OUT,T,B,R,OUT

OUT,F,B,R,OUT

OUT,+,B,R,OUT

OUT,1,1,L,STOP

OUT,0,0,L,STOP

OUT,I,I,L,STOP

OUT,0,0,L,STOP

示例: BBB1101+1011BBB

经过扫描阶段后, 读写头来到+, 左走一格, 切换状态为C1并改写原位置的1为T

得到: BBB110T+1011BBB

进入C1状态后向右只读到空白, 左走一格, 切换状态为P1, 在右数最低位上加1, 将1改写为O, 发生进位, 状态切换为CARRY

得到: BBB110T+101OBBB

得到: BBB110T+110OBBB //至此完成了两数最低位的对齐和相加

继续运行我们的指令 //此后每次完成一位加法

得到: BBB11FT+110OBBB 虽然加号被1进位顶掉了, 但是由于我们给BACK加了新的判

得到: BBB1TFT10OOBBB 断条件, 即遇到T/F时切换状态为RE, 所以程序还能正常运行

得到: BBBTTFT1IOOOBBB

back跳过了这个1, 太遗憾了, 这使得我们的输出很不统一, 但只能将就着用了, 因为如果把加号处的1变成I, 将直接影响到对应位相加

BACK从I出发

向左直到第一个B, 切换状态为OUT

最后得到: BBBBBBBB1IOOOBBB

即1101+1011=11000

虽然很丑陋, 但至此终究完成了计算

//我们可以再写一个图灵机程序, 保留1, 0, B, 并将所有的I改写为1, O改写为0; 这并不困难; 留给同学们课后完成 (QAQ)

特别鸣谢我的舍友容忍我中午不睡觉敲键盘

参考文献:

1、【计算机科学导论实验】加法图灵机模拟 题解

[https://blog.csdn.net/qq\\_39417033/article/details/137610642](https://blog.csdn.net/qq_39417033/article/details/137610642)

2、操作系统基础 01 - 起源(一) 图灵机

<http://shangyang.me/2019/11/17/os-basic-02-turing-machine/>

3、chatgpt4o

以下为完整指令集

符号:

1,0,B(Blank),T,F,I,O;//在"+"左边用T/F标记1/0, 在右边用I/O标记1/0;

状态:

SCAN //用于从左向右扫描第一个数至 "+"  
RE //用于从"+"开始向左越过所有已经被标记的数, 获得最靠右的未标记数, 并把这位的1标记为T,0标

记为F

G0 //用于携带"0"至空格或者右数左起被标记的第一位  
G1 //用于携带"1"至空格或者右数左起被标记的第一位  
P0 //在右数未被标记的末位加0, 并在当位把1标记为I,把0标记为O  
P1 //在右数未被标记的末位加1, 并在当位把1标记为I,把0标记为O  
Carry //标记进位  
BACK //用于读写头从右数返回至"+"  
OUT //擦除两数和外的符号  
STOP //运行到此停机

输入规定:

输入形如xxxx+xxxx,其中x为0或1;并规定右边的数一定比前面的数位数大;

指令集:

SCAN,0,0,R,SCAN;  
SCAN,1,1,R,SCAN  
SCAN,+,+,L,RE //左起向右扫描至"+", 向左一格切换模式为RE  
  
RE,F,F,L,RE  
RE,T,T,L,RE  
RE,0,F,R,G0  
RE,1,T,R,G1  
RE,B,B,R,OUT //从"+"开始, 向左越过所有的T/F, 获得左数未被标记的末位  
  
G0,F,F,R,G0  
G0,T,T,R,G0  
G0,+,+,R,G0  
G0,0,0,R,G0  
G0,1,1,R,G0  
G0,B,B,L,P0  
G0,O,O,L,P0  
G0,I,I,L,P0 //向右至第一个空格或I/O,向左一格切换模式为P0

P0,0,O,L,BACK  
P0,1,I,L,BACK

G1,F,F,R,G1  
G1,T,T,R,G1  
G1,+,+,R,G1  
G1,0,0,R,G1  
G1,1,1,R,G1  
G1,B,B,L,P1  
G1,O,O,L,P1  
G1,I,I,L,P1

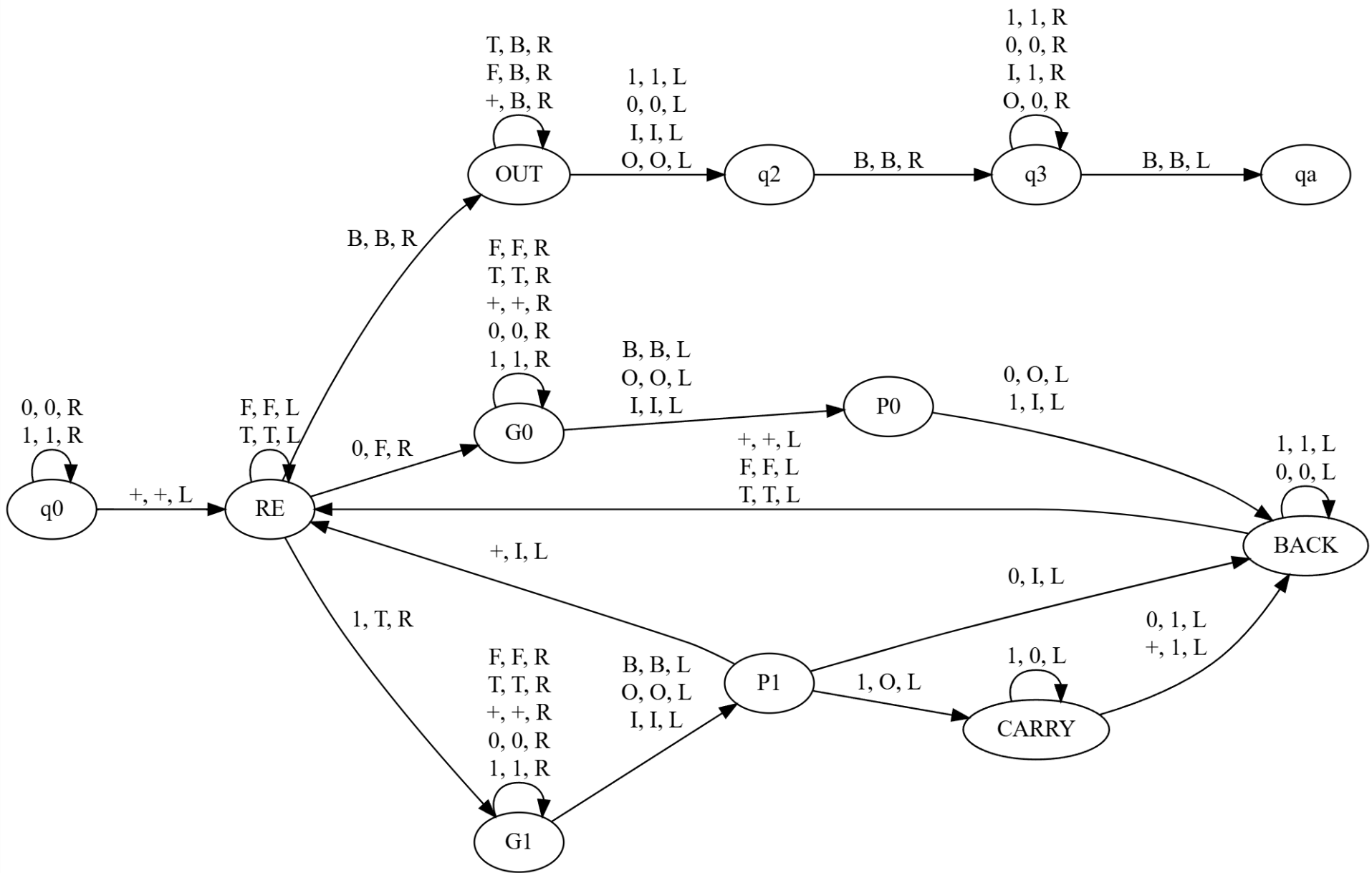
//向右至第一个空格或I/O,向左一格切换模式为P1

P1,0,I,L,BACK  
P1,1,O,L,CARRY  
P1,+,I,L,RE

CARRY,0,1,L,BACK  
CARRY,+,1,L,BACK  
CARRY,1,0,L,CARRY

BACK,1,1,L,BACK  
BACK,0,0,L,BACK  
BACK,+,+,L,RE  
BACK,F,F,L,RE  
BACK,T,T,L,RE

RE,B,B,R,OUT  
OUT,T,B,R,OUT  
OUT,F,B,R,OUT  
OUT,+,B,R,OUT  
OUT,1,1,L,STOP  
OUT,0,0,L,STOP  
OUT,I,I,L,STOP  
OUT,O,O,L,STOP //擦除 除两数和以外的数字和字母



# 关于图灵机的一些问题？

- 给定一个图灵机的描述，判断纸带上的一个初始输入能否被该图灵机接收/拒绝（判断对错）
- 给定一个图灵机的描述，给出一个能被该图灵机所接收/拒绝的初始输入（举正/反例）
- 给定一个图灵机的描述，判断该图灵机是否能完成一个所需的功能（验证完备性）
- 给定两个完成相同功能的图灵机，判断谁的效率更高？（算法复杂度）
- 给定一个图灵机的描述，求出一个完成相同功能但效率更高的图灵机（算法优化）
- 判断一个所需的功能可否用图灵机进行描述（可计算性）

# 附加题：YOJ-832

图灵机有一条无限长的纸带，纸带分成了一个一个小方格。有一个机器头在纸带上移来移去。机器头有一组内部状态，还有一些固定的程序。在每个时刻，机器头都要从当前纸带上读入一个方格信息，然后结合自己的内部状态查找程序表，根据程序输出信息到纸带方格上，并转换自己的内部状态，然后进行移动。**当进入接收状态或者拒绝状态时，程序结束。**下面是图灵机形式化的定义：

图灵机 $\mathcal{Q}$ 是一个7元组， $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ ，其中 $Q, \Sigma, \Gamma$ 都是有限集并且满足一下七个条件：

1.  $Q$ 是状态集合，
2.  $\Sigma$ 是输入字母表，且不含包空符号 (blank symbol)  $\sqcup$ ，
3.  $\Gamma$ 是磁带的字母表，且 $\sqcup \in \Gamma, \Sigma \subset \Gamma$ ，
4.  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 是转换函数，
5.  $q_0 \in Q$ 是开始状态，
6.  $q_{accept} \in Q$ 是接收状态，
7.  $q_{reject} \in Q$ 是拒绝状态，且 $q_{reject} \neq q_{accept}$ 。

图灵机的当前状态、当前的磁带内容、读头所在位置构成了图灵机的配置。

\* 简单起见，所有的状态用阿拉伯数字0,1,2,3,4,5...依次表示，特别地，**开始状态即q0=0。**

\* 对于转换函数，用一个例子来说明。例如，若某条规则为 (state0, a)→(state1, d, R)

意味着当状态为state0，机器头读取到的符号为a时，将状态变为state1，并修改当前机器头指向的方格中的符号为d，最后将机器头向右移动一格

|               |   |     |   |       |  |
|---------------|---|-----|---|-------|--|
|               |   | 机器头 |   |       |  |
| state0: (空符号) | a | b   | c | (空符号) |  |
|               |   |     |   |       |  |
|               |   | 机器头 |   |       |  |
| state1: (空符号) | d | b   | c | (空符号) |  |

(state0, a)→(state1, d, R)

\* 特别地，**将空符号视为‘#’**，保证转换函数中不会出现空符号

\* 通常我们认为纸带有左边界，并给纸带的每一个方格按如下规则编号

|   |   |   |   |   |   |       |
|---|---|---|---|---|---|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | ..... |
|---|---|---|---|---|---|-------|

若某一时刻，机器头在第0格，并且根据规则，机器头在下一步时将向左移动，那么此时机器头将会停留在第0格不动。

\* 你**不需要考虑图灵机的停机问题**，即保证给出的图灵机不存在“死循环”的情况。

现在，给定一个图灵机T，试给出被T接受的一个输入状态（当一组输入通过图灵机上的操作最后进入接受状态时，即认为该输入被图灵机接受）

## 【输入格式】

接下来的输入描述一个图灵机 $T$

第一行一个整数 $n_{state}$ ,表示图灵机的状态数。并且这 $n_{state}$ 个状态具体为 $0, 1, 2, 3, \dots, n_{state} - 1$

第二行一个整数 $n_{acc}$ , 表示接受状态的集合大小。

接下来一行包括 $n_{acc}$ 个整数, 用空格分开, 表示 $n_{acc}$ 个接受状态。

接下来一行一个整数 $n_{rej}$ , 表示拒绝状态的集合大小。

接下来一行包括 $n_{rej}$ 个整数, 用空格分开, 表示 $n_{rej}$ 个拒绝状态。

接下来一行包括一个整数 $m$ , 表示磁带字母表的大小(不含空符号)

接下来一行包括一个字符串, 表示磁带字母表(不含空符号)

接下来一行包括一个整数 $n_{rules}$ , 表示图灵机的转换规则数目

接下来 $n_{rules}$ 行, 每行形如 $state_0 \ ch_0 \ state_1 \ ch_1 \ L/R/-$ ,

表示当状态为 $state_0$ , 机器头读取到的字符为 $ch_0$ 时, 下一个状态为 $state_1$ ,将机器头指向的方格中的符号由 $ch_0$ 变为 $ch_1$ ;

- 若最后一个字符为 $L$ , 则机器头向左移动一格
- 若最后一个字符为 $R$ , 则机器头向右移动一格
- 特别地, 最后一个字符为 $-$ 时, 当且仅当 $state_1$ 为接受或者拒绝状态, 此时程序结束, 机器头的移动方向没有意义

## 【输出格式】

第一行包括机器头指向的位置

第二行包括初始状态下的纸带 (右侧的空字符可以省略)

### 【输入示例】

```
4
1
3
0
2
01
4
0 0 1 1 R
1 1 2 1 R
2 1 2 1 R
2 0 3 1 -
```

### 【输出示例】

```
1
#010
```

### 【样例解释】

该图灵机接受形如01.....10的串

### 【数据范围】

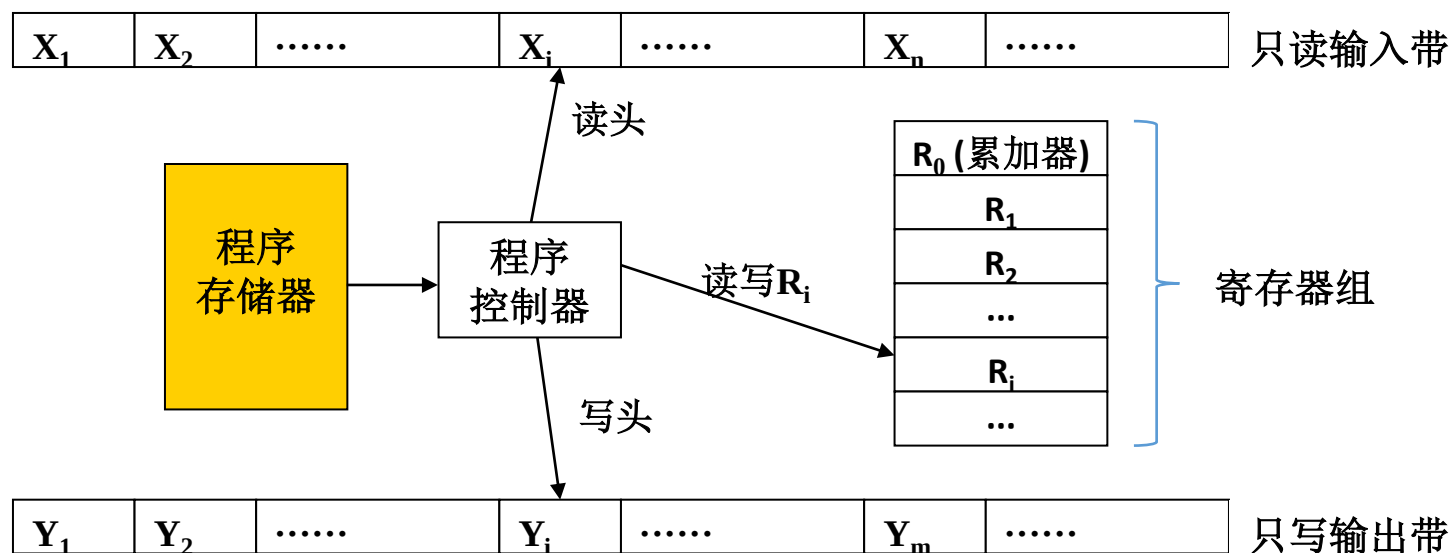
- $n_{state} \leq 11000$ ,  $\Sigma$ 中所有字符ascii码范围在0~255
- 保证存在串 $s$ , 满足
  - $T \text{ accpet } s$
  - $|\Sigma|^{|s|} \times Step(T, s) \leq 10000000$ , 其中 $Step(T, s)$ 表示串 $s$ 在图灵机 $T$ 上执行的步数

# 随机存取机模型 (RAM)

思考：RAM和TM的联系与区别

## ■ 组成部件：

- 只读输入带由一系列方格和带头组成，每输入一次，带头右移一格
- 只写输出带由一系列方格和带头组成，每输出一次，带头右移一格
- 程序控制部件由指令计数器、指令译码器构成，用来控制程序走向执行
- 程序存储部件用来存储程序
- 寄存器组由个数无限、按序编号的寄存器组成



# 随机存取机模型 (RAM)

## ■ 指令集:

- Input: 输入数据
- Output: 输出数据
- Load: 将普通寄存器中的值载入累加器
- Store: 将累加器中的值存入普通寄存器
- Add: 将指定寄存器的值与累加器的值相加, 结果存入累加器

## ■ 示例: 用RAM模型计算 $x+y$

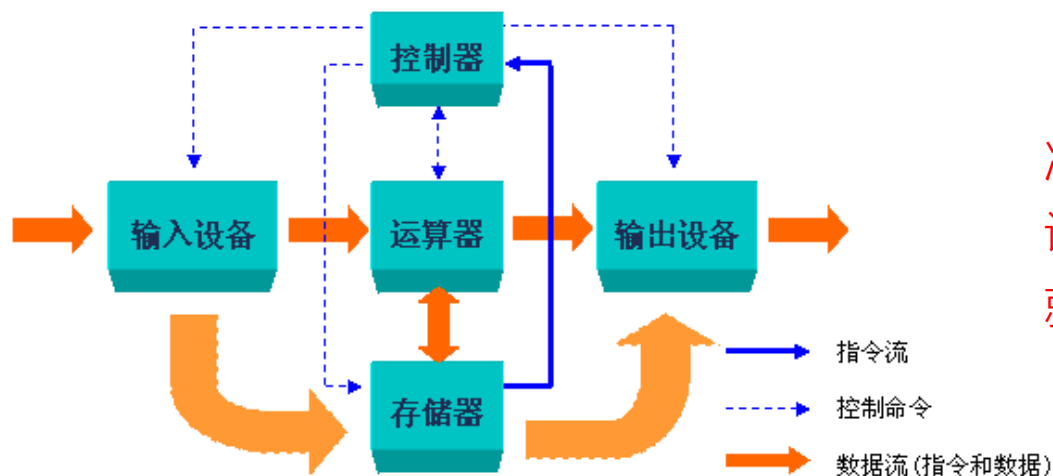
```
1. Input R1
2. Input R2
3. Load R1 //R0 = R1
4. Add R2 //R0 = R0 + R2
5. Store R3 //R3 = R0
6. Output R3
```

# 1.1 计算模型与体系结构

## ■ 体系结构：计算模型的物理实现

### ■ 冯·诺依曼体系结构

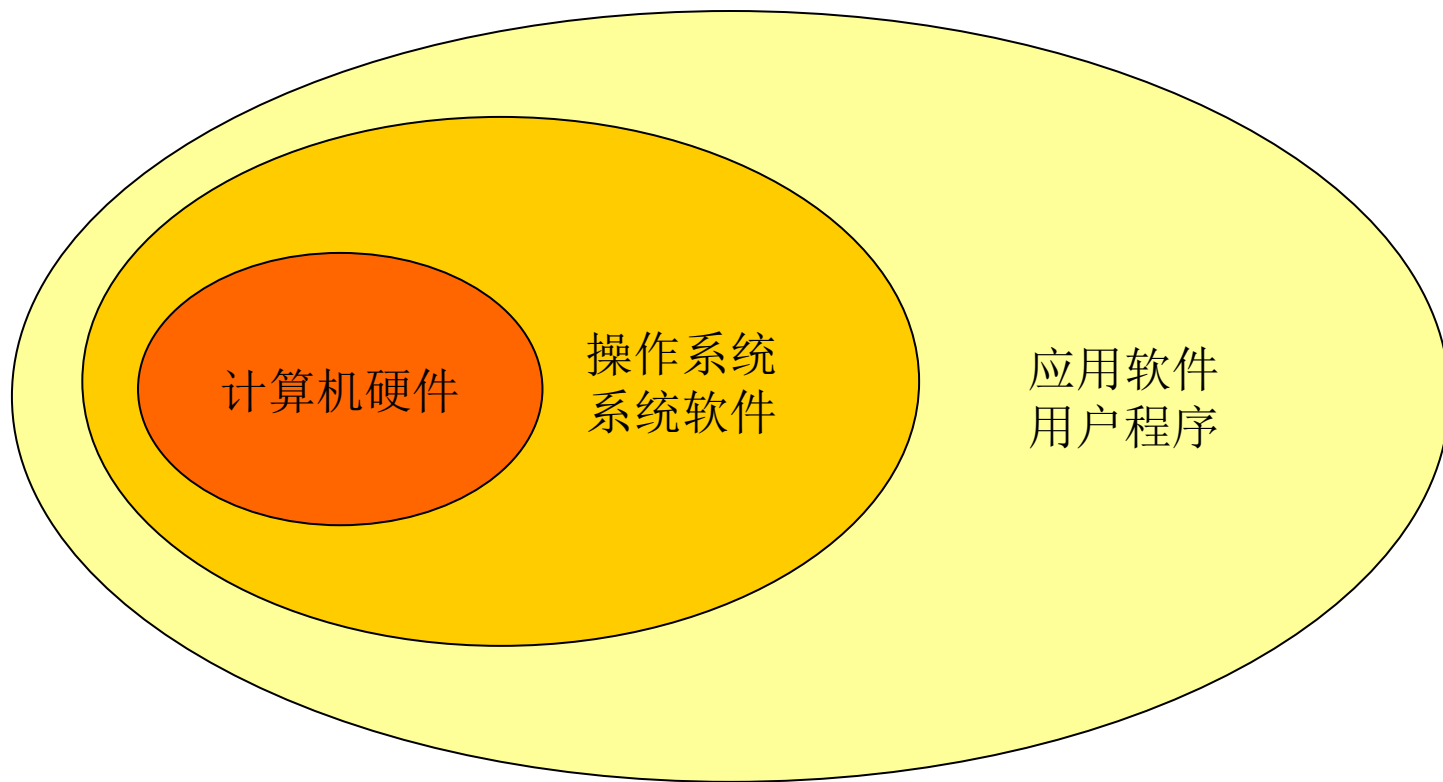
- 计算机硬件由**运算器**、**控制器**、**存储器**、**输入/输出设备**组成
- 数据和程序以**二进制代码**形式存放
- 控制器根据存放在存储器中的**程序**来工作



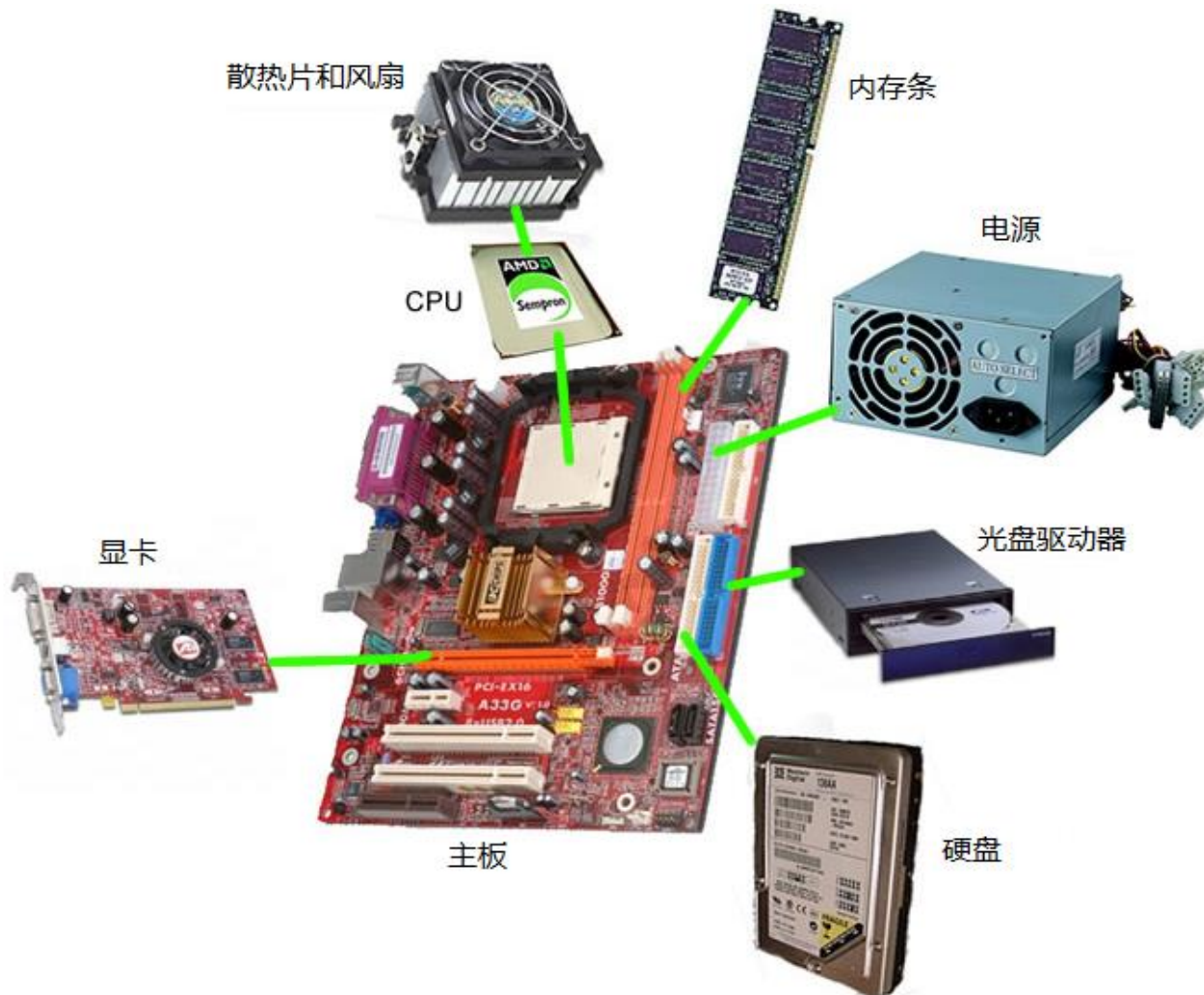
冯·诺依曼体系结构计算机是一种通用设计，只需修改存储器中的指令序列，就能控制计算机做不同的事情

更多内容：《计算机系统基础I》（第3学期）

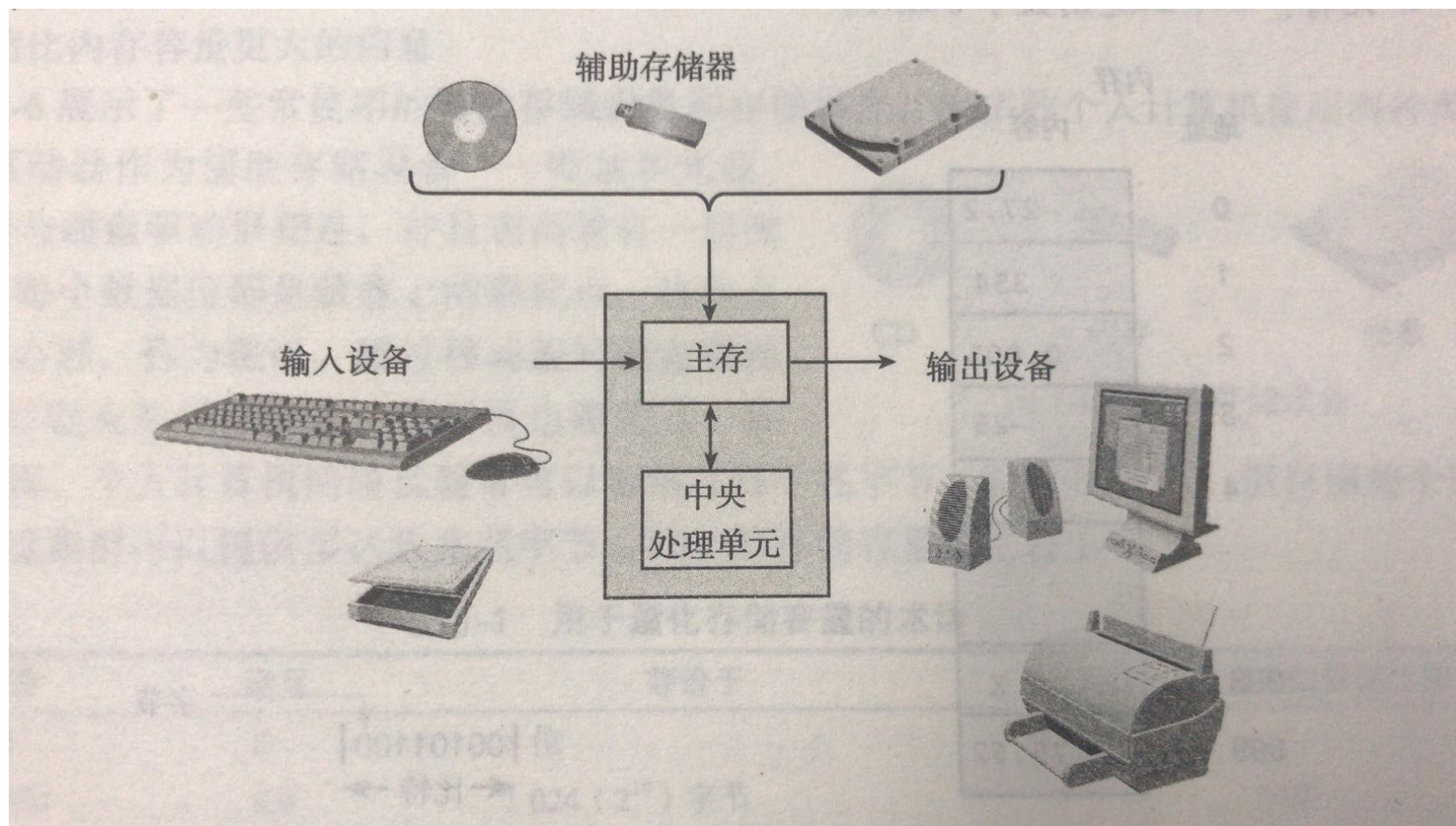
## 1.2 计算机系统的组成



## 1.2.1 计算机硬件



## 1.2.1 计算机硬件



更多内容：《计算机系统基础I》（第3学期）、《计算机系统实现I》（第5学期）

# 中央处理器（CPU）

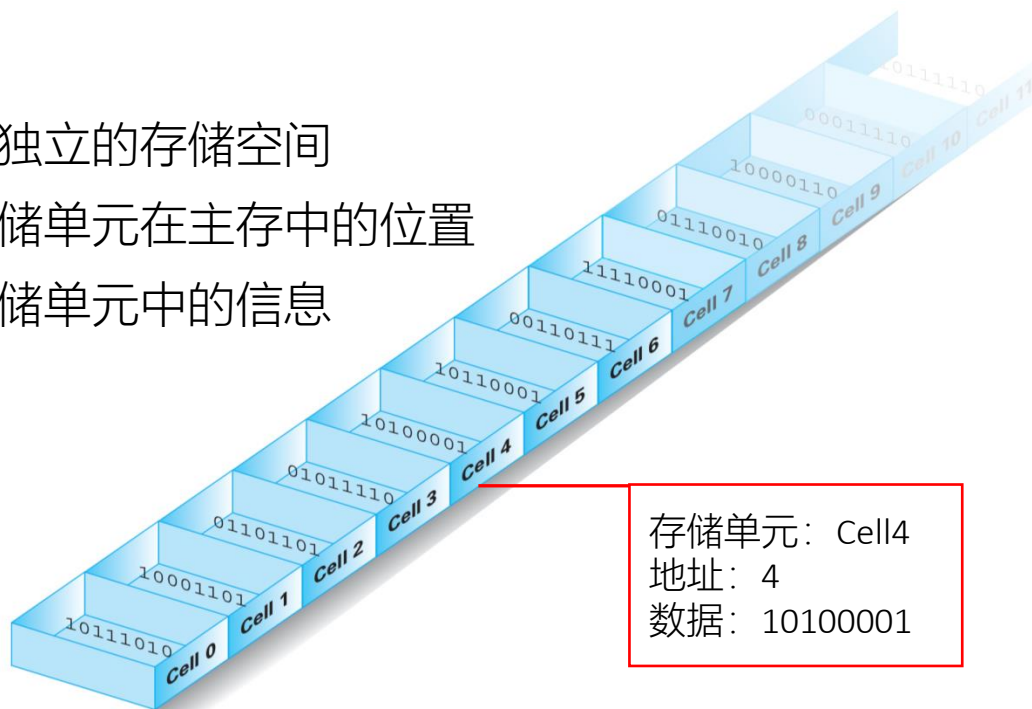
- 中央处理器是计算机中执行处理数据指令的器件
- CPU主要由两部分组成：运算器和控制器
  - 运算器进行加减法等算术运算以及比较两个数之类的逻辑运算
  - 控制器包括指令控制器（识别和翻译指令代码）、时序控制器（为每条指令按时间顺序提供控制信号）、总线控制器（控制CPU的内外部总线）和中断控制器（控制各种各样的中断请求）
- 字长：CPU可以同时处理的位数
  - 字长较长的处理器在每个处理周期内可以处理更多的数据
  - 当前的个人计算机通常都是32位或64位处理器

# 主存储器 (Memory)

- 主存用于存放正在运行的程序和需要立即处理的数据
- 主要有两类主存：随机存储器 (RAM) 和只读存储器 (ROM)
  - RAM用于存储临时的数据（如程序运行时的数据），断电后数据消失
  - ROM用于存储永久的数据（如开机引导程序），断电后数据不会消失

## ■ 存储结构

- 存储单元：主存中独立的存储空间
- 存储单元地址：存储单元在主存中的位置
- 存储单元内容：存储单元中的信息



存储单元：Cell4  
地址：4  
数据：10100001

## 1.2.2 计算机软件

### ■ 系统软件：计算机工作时必须配备的软件

- 操作系统：管理计算机硬件资源，控制程序运行，为用户提供交互界面
- 语言处理程序：将高级语言源程序翻译成计算机能识别的目标程序
- 数据库管理系统：存储和管理海量数据，方便进行查询和增删改操作

### ■ 应用软件：为某种特定用途而开发的软件

- 文本编辑：UltraEdit, Notepad++, Word, .....
- 图像处理：Photoshop, ACDSee, 美图秀秀, .....
- 视频播放：暴风影音, 优酷视频, 腾讯视频, .....
- 网页浏览：Chrome, Edge, Firefox, Surfari, .....
- .....

更多内容：《计算机系统基础II》（第4学期）、《计算机系统实现II》（第6学期）

# 1.3 信息在计算机中的表示

## ■ 信息与数据

- 信息：各种事物的变化和特征的反映
- 数据：信息的载体，如数值、文字、图像等
- 人理解的是信息，计算机处理的是数据

## ■ 计算机中的数据

- 数值数据：表示量的大小、正负，如整数、小数等
- 字符数据：表示一些符号、标记，如英文字母A~Z、a~z、数字0~9、各种专用字符+、-、/、（）.....及标点符号等
- 其他数据：声音数据、图像数据等（需要特定的编码/解码规则）

更多内容：《计算机系统基础I》（第3学期）

# 1.3 信息在计算机中的表示

## ■ 数据单位

- 信息在计算机中用二进制表示，最小的数据单位是一个数位 (bit)
- 8个数位被称为1个字节 (Byte)，是计算机中表示存储容量的基本单位
- 存储容量的计量单位有字节B、千字节KB、兆字节MB、十亿字节GB及万亿字节TB等，换算关系如下：

$$1 \text{ B} = 8 \text{ bit}$$

$$1 \text{ KB} = 1024 \text{ B} = 2^{10} \text{ B}$$

$$1 \text{ MB} = 1024 \text{ KB} = 2^{20} \text{ B}$$

$$1 \text{ GB} = 1024 \text{ MB} = 2^{30} \text{ B}$$

$$1 \text{ TB} = 1024 \text{ GB} = 2^{40} \text{ B}$$

# 1.3.1 进制及其转换

## ■ 基本概念

- 数制：用一组固定的**数码**和一套统一的**规则**来表示数目的方法
- 进位计数制：按照进位方式计数的数制
- 基数：某种进制中允许使用的基本数码的个数
- 位权：一个数码处在不同位置上代表的值不同。每个数码所代表的数值等于该数码乘以一个与数码所在位置相关的常数。这个常数称为位权。

## ■ 计算机科学常用的进制

- 二进制 (Binary)
- 八进制 (Octonary)
- 十六进制 (Hexadecimal)

# 十进制整数 → 非十进制整数

余数法：除基数取余数。

示例：

$$(75)_{10} = ( ? )_8$$

|   |    |   |
|---|----|---|
| 8 | 75 | 3 |
| 8 | 9  | 1 |
| 8 | 1  | 1 |
|   | 0  |   |

低 ↑ 高

结果为  $(75)_{10} = (113)_8$

# 十进制小数 → 非十进制小数

**进位法：**乘基数取整。用十进制**小数乘基数**，当积为0或达到所要求的精度时，**将整数部分由上而下排列**。示例： $(0.625)_{10} = (?)_2$

$$\begin{array}{r} 0.625 \\ \times 2 \\ \hline 1.250 \\ \times 2 \\ \hline 0.50 \\ \times 2 \\ \hline 1.0 \end{array}$$

高  
↓  
低

**结果** $(0.625)_{10} = (0.101)_2$

整数=1

整数=0

整数=1      小数值=0

## 十进制数 → 非十进制数

例： $(81.65)_{10} = (? )_2$ ，要求精度为小数六位。

### 整数与小数分别转换

$(0.65)_{10} = ( ? )_2$ ，**进位法**—乘基数取整。

得： $(0.65)_{10} = (0.101001)_2$

$(81)_{10} = ( ? )_2$ ，**余数法**—除基取余法

得： $(81)_{10} = (1010001)_2$

综合得： $(81.65)_{10} = (1010001.101001)_2$

**注意：**小数转换不一定能算尽，只能算到一定精度的位数为止，故要产生一些误差。当位数较多时，这个误差就很小了。

# 非十进制数 → 十进制数

利用多项式表示法（**位权多项式法**）把各非十进制数按权展开求和。

**转换公式：**

$$(s)_N = \sum_n^{-m} k_i N^{i-1}$$

$$(S)_N = (k_n k_{n-1} \cdots k_1 \cdot k_0 k_{-1} \cdots k_{-m})_N$$

例：  $(1011.1)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$   
 $= 8 + 0 + 2 + 1 + 0.5$   
 $= (11.5)_{10}$

---

## 二进制与八进制之间的转换

从**小数点**开始，将二进制数的整数和小数部分**每三位**分为**一组**，**不足**三位的分别在整数的最高位前和小数的最低位后加“0”补足，然后每组用等值的八进制码替代，即得目的数。

例：  $(11010111.0100111)_B = (?)$

0

0 110 1011 1 0100 111 00

↓ ↓ ↓ ↓ ↓ ↓  
3 2 7 2 3 4

三位合一位

得：  $(11010111.0100111)_B = (327.234)_O$

$(327.234)_O = (?)_B$

一位拆三位

小数点为界

## 二进制与十六进制之间的转换

从**小数点**开始，将二进制数的整数和小数部分**每四位**分为**一组**，**不足**四位的分别在整数的最高位前和小数的最低位后加“0”补足，然后每组用等值的十六进制码替代，即得目的数。

例9:  $111011.10101\text{ B} = 3\text{B}.\text{A}8\text{ H}$

00111011.10101000



3



B



A



8

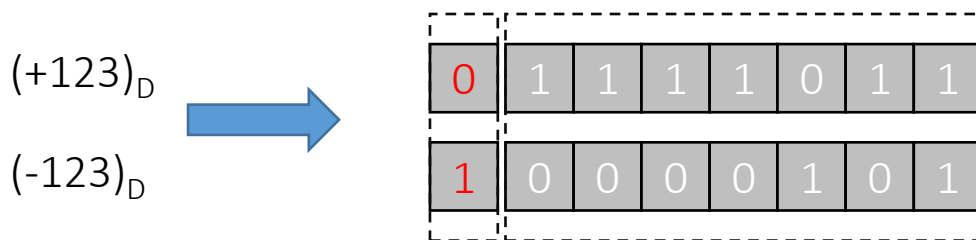
四位合一

一位拆四位

## 1.3.2 数值的表示

### ■ 真值与机器数

- 真值：在计算机外部，用正负号表示的实际数值
- 机器数：在计算机内部，将正负号数字化后得到的数值编码结果



符号位

### ■ 定点数与浮点数

- 定点数：约定所有数值数据的小数点隐含在某一个固定位置上
- 浮点数：小数点的位置可按需要浮动



定点整数： $(+123)_D$



定点小数： $(+0.9609375)_D$

# 定点整数的表示

注意:

- $B2U$ 、 $B2T_\omega$ 、 $B2F$ 是双射
- $B2S_\omega$ 、 $B2O_\omega$ 是满射，但不是单射，因为存在两个0 (+0 和 -0)

- 假设一个二进制整数有 $\omega$ 位，位向量 $\vec{x} = [x_{\omega-1}, x_{\omega-2}, \dots, x_0]$



- 无符号整数

$$B2U(\vec{x}) \stackrel{\text{def}}{=} \sum_{i=0}^{\omega-1} x_i \cdot 2^i$$

值的范围:  $[0, 2^\omega - 1]$

- 有符号整数

- 原码 (Sign-Magnitude) : 除符号位以外，其余位保持原样

$$B2S_\omega(\vec{x}) \stackrel{\text{def}}{=} (-1)^{x_{\omega-1}} \cdot \left( \sum_{i=0}^{\omega-2} x_i \cdot 2^i \right) \quad \text{值范围: } [-2^{\omega-1} + 1, 2^{\omega-1} - 1], \text{ 共 } 2^\omega - 1 \text{ 个数}$$

- 反码 (Ones'-Complement) : 反码 + |负数| =  $2^\omega - 1$  ( $\omega$ 位二进制所能表示的最大无符号整数)

$$B2O_\omega(\vec{x}) \stackrel{\text{def}}{=} -x_{\omega-1} \cdot (2^{\omega-1} - 1) + \sum_{i=0}^{\omega-2} x_i \cdot 2^i \quad \text{值范围: } [-2^{\omega-1} + 1, 2^{\omega-1} - 1], \text{ 共 } 2^\omega - 1 \text{ 个数}$$

- 补码 (Two's-Complement): 补码 + |负数| =  $2^\omega$  (模数), 补码=反码+1

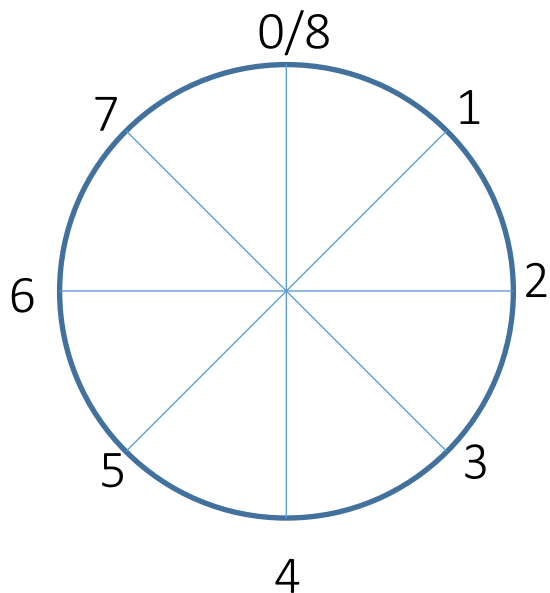
$$B2T_\omega(\vec{x}) \stackrel{\text{def}}{=} -x_{\omega-1} \cdot 2^{\omega-1} + \sum_{i=0}^{\omega-2} x_i \cdot 2^i \quad \text{值范围: } [-2^{\omega-1}, 2^{\omega-1} - 1], \text{ 共 } 2^\omega \text{ 个数}$$

- 移码 (Frame Shift) : 移码 = 补码 +  $2^{\omega-1}$

$$B2F(\vec{x}) \stackrel{\text{def}}{=} (-x_{\omega-1} \cdot 2^{\omega-1})^{(1-x_{\omega-1})} + \sum_{i=0}^{\omega-2} x_i \cdot 2^i \quad \text{值范围: } [-2^{\omega-1}, 2^{\omega-1} - 1], \text{ 共 } 2^\omega \text{ 个数}$$

# 如何理解原码、反码、补码、移码

原码 (方便人看) → 反码 (原码向补码过度) → 补码 (方便机器计算) → 移码 (方便比较大小)



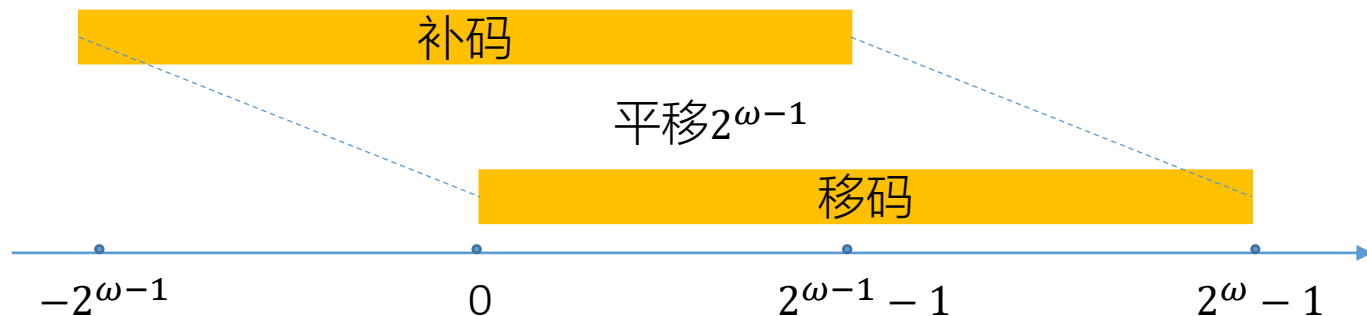
假如当前箭头指向4, 想要将箭头指向2, 需要怎么操作:

1. 往逆时针方向拨2个单位:  $(4 - 2) \bmod 8 = (4 + (-2)) \bmod 8 = 2$

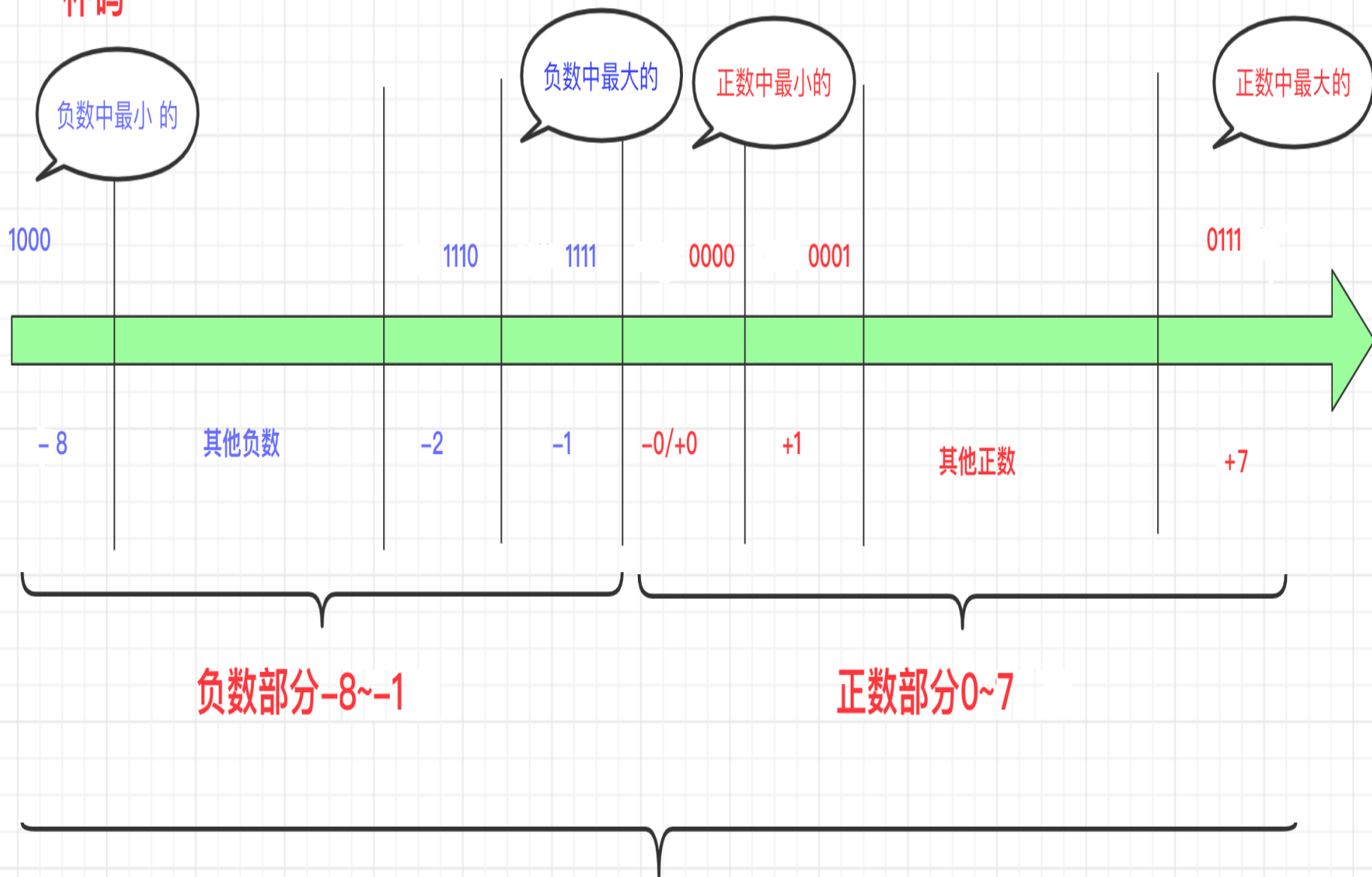
2. 往顺时针方向拨6个单位:  $(4 + 6) \bmod 8 = 10 \bmod 8 = 2$

即: 往逆时针方向拨 (减法) 可以用往顺时针方向拨 (加法) 替代!

-2 和 +6 在模数8的情况下是互补的:  $(-2)_D = [1110]_{\text{补}}$ ,  $(+6)_D = [0110]_{\text{补}}$



# 补码

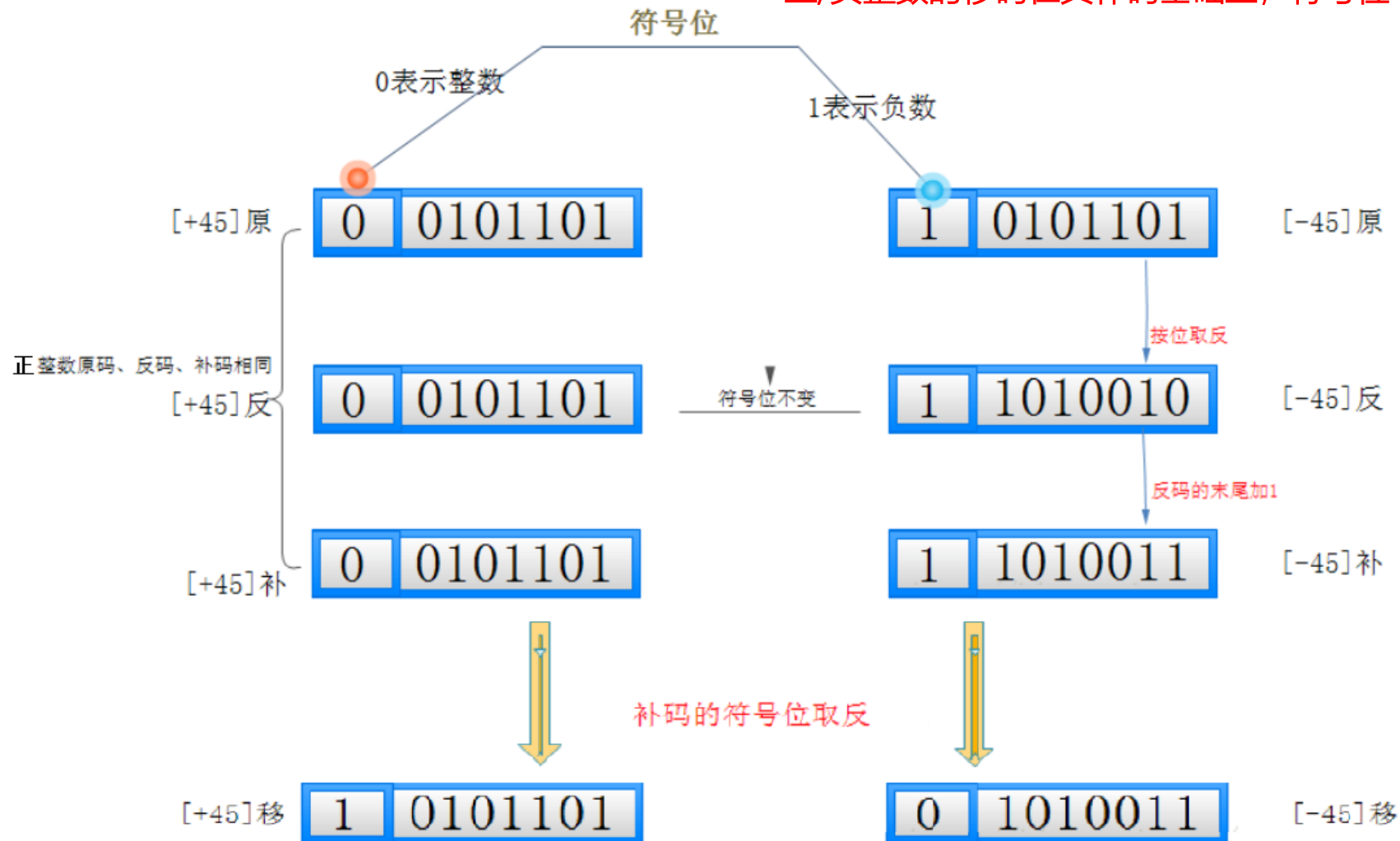


4 位, 数值范围: -8~7 共 16 个数

# 定点整数的表示

规律：

- 正整数的原码、反码、补码相同
- 负整数的原码符号位为1，其余位不变，
- 负整数的反码在其原码基础上，按位取反，
- 负整数的补码在其反码基础上，末位加1
- 正/负整数的移码在其补码基础上，符号位取反



# 编码的意义

- 硬件局限：只有加法器，没有减法器
- 编码目标：
  - 运算的操作数和运算结果使用相同的编码
  - 能简单判断正负和比较值的大小
  - 运算过程不需要判断正负和比较值的大小
  - 所有的减法操作都能转换成加法操作
- 在绝大多数计算机中，使用补码来表示定点整数

# 有符号整数几种编码方式的比较 (假定 $\omega=4$ )

## ■ 原码

- 在反码表示中, 数值0有两种表示形式 ( $[0000]_{\text{原}}$ 和 $[1000]_{\text{原}}$ )
- 计算 $(+1)_{\text{D}}+(+1)_{\text{D}}$ :  $[0001]_{\text{原}}+[0001]_{\text{原}}=[0010]_{\text{原}}=(+2)_{\text{D}}$
- 计算 $(+1)_{\text{D}}+(-1)_{\text{D}}$ :  $[0001]_{\text{原}}+[1001]_{\text{原}}=[1010]_{\text{原}}=(-2)_{\text{D}}$
- 计算 $(-1)_{\text{D}}+(-1)_{\text{D}}$ :  $[1001]_{\text{原}}+[1001]_{\text{原}}=[0010]_{\text{原}}=(+2)_{\text{D}}$

## ■ 反码

- 在反码表示中, 数值0有两种表示形式 ( $[0000]_{\text{反}}$ 和 $[1111]_{\text{反}}$ )
- 计算 $(+1)_{\text{D}}+(+1)_{\text{D}}$ :  $[0001]_{\text{反}}+[0001]_{\text{反}}=[0010]_{\text{反}}=(+2)_{\text{D}}$
- 计算 $(+1)_{\text{D}}+(-1)_{\text{D}}$ :  $[0001]_{\text{反}}+[1110]_{\text{反}}=[1111]_{\text{反}}=(-0)_{\text{D}}$
- 计算 $(-1)_{\text{D}}+(-1)_{\text{D}}$ :  $[1110]_{\text{反}}+[1110]_{\text{反}}=[1100]_{\text{反}}=(-3)_{\text{D}}$

# 有符号整数几种编码方式的比较 (假定 $\omega=4$ )

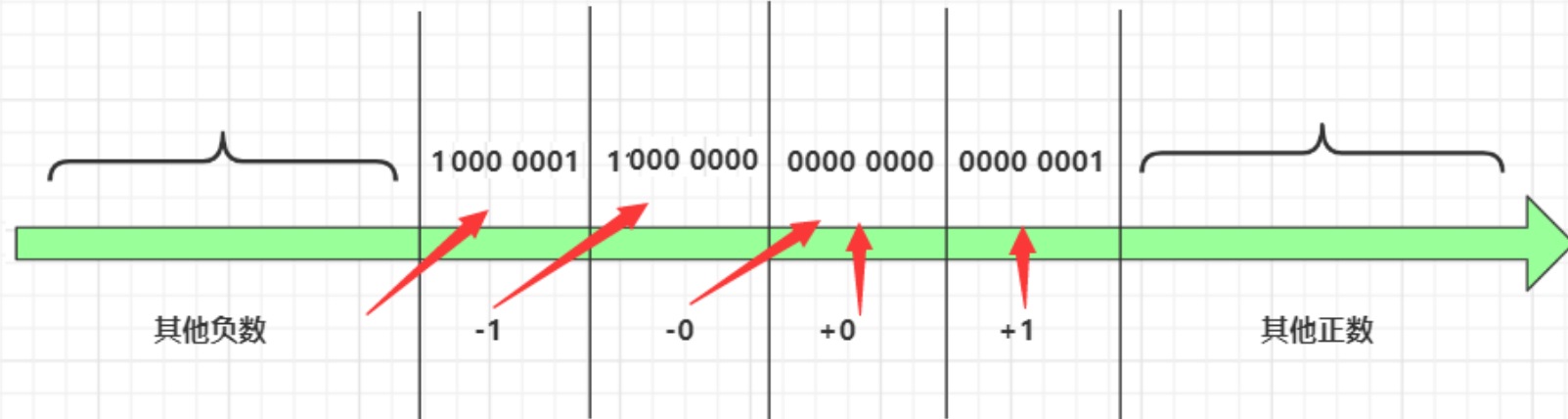
## ■ 补码

- 在补码表示中, 数值0只有一种表示形式 ( $[0000]_{\text{补}}$ 表示0,  $[1000]_{\text{补}}$ 表示-8)
- 计算 $(+1)_D + (+1)_D$ :  $[0001]_{\text{补}} + [0001]_{\text{补}} = [0010]_{\text{补}} = (+2)_D$
- 计算 $(+1)_D + (-1)_D$ :  $[0001]_{\text{补}} + [1111]_{\text{补}} = [0000]_{\text{补}} = (0)_D$
- 计算 $(-1)_D + (-1)_D$ :  $[1111]_{\text{补}} + [1111]_{\text{补}} = [1110]_{\text{补}} = (-2)_D$

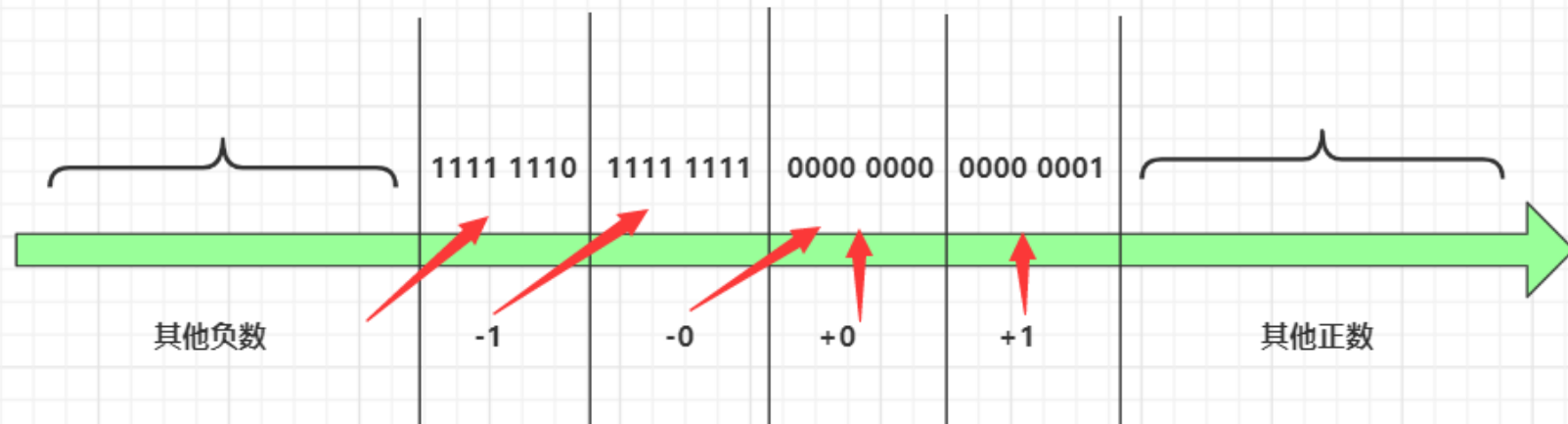
## ■ 移码

- 移码不使用符号位判断正负, 而是通过检查是否大于 $2^{\omega-1}$ 来判断正负
- 计算 $(+1)_D + (+1)_D$ :  $[1001]_{\text{移}} + [1001]_{\text{移}} = [0010]_{\text{移}} = (-6)_D$
- 计算 $(+1)_D + (-1)_D$ :  $[1001]_{\text{移}} + [0111]_{\text{移}} = [0000]_{\text{移}} = (-8)_D$
- 计算 $(-1)_D + (-1)_D$ :  $[0111]_{\text{移}} + [0111]_{\text{移}} = [1110]_{\text{移}} = (+6)_D$

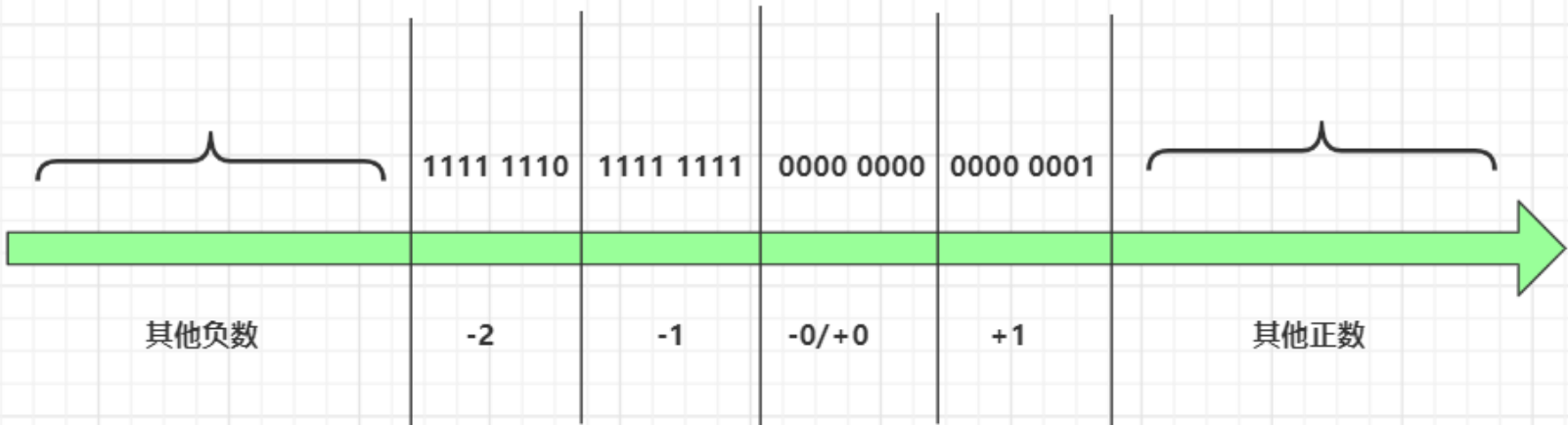
原码



反码

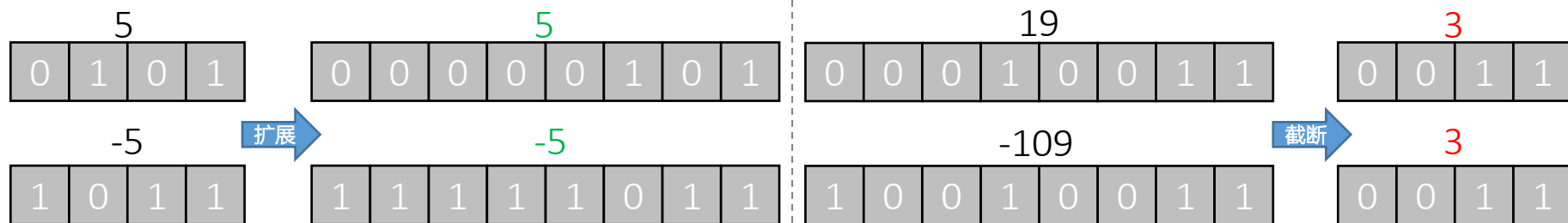


补码



# 定点整数的扩展与截断

- 假设一个二进制整数有 $\omega$ 位，位向量 $\vec{x} = [x_{\omega-1}, x_{\omega-2}, \dots, x_0]$ ，现欲将其变成 $\omega'$ 位，位向量 $\vec{x}'$
- 扩展（当 $\omega' > \omega$ 时）：从数值角度考虑（保值）
  - 无符号整数的零扩展： $\vec{x}' = [0, \dots, 0, x_{\omega-1}, x_{\omega-2}, \dots, x_0]$
  - 有符号整数补码表示形式的符号扩展： $\vec{x}' = [x_{\omega-1}, \dots, x_{\omega-1}, x_{\omega-1}, \dots, x_0]$
- 截断（当 $\omega' < \omega$ 时）：从位级角度考虑（不保值）
  - 截断无符号整数和有符号整数： $\vec{x}' = [x_{\omega'-1}, x_{\omega'-2}, \dots, x_0]$
  - 规律： $x' = x \bmod 2^{\omega'}$



# 有符号整数和无符号整数之间的转换

## ■ 从数值角度考虑

- 对于在两种形式中都能表示的值，保持不变
- 将负数转换成无符号数，可能会得到0
- 待转换的无符号数超出补码能表示的范围，可能会得到所能表示的最大值

## ■ 从位级角度考虑（大多数C语言的实现）

- 规则：保持位值不变，只是改变了解释这些位的方式，从而数值可能改变
- 无符号整数转换为有符号整数的补码表示形式

$$U2T_{\omega}(x) = \begin{cases} x, & x \leq 2^{\omega-1} - 1 \\ x - 2^{\omega}, & x > 2^{\omega-1} - 1 \end{cases} \quad (0 \leq x \leq 2^{\omega} - 1)$$

- 有符号整数的补码表示形式转换为无符号整数

$$T2U_{\omega}(x) = \begin{cases} x + 2^{\omega}, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (-2^{\omega} \leq x \leq 2^{\omega} - 1)$$

无符号整数：19

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

有符号整数的补码形式：19

无符号整数：147

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

有符号整数的补码形式：-109

# 浮点数的表示

- 科学计数法:  $N = M \times R^E$

- $M$ : 尾数,  $R$ : 基值,  $E$ : 阶码

- 示例:  $0.00303 = 3.03 \times 10^{-3} = 0.303 \times 10^{-2}$

- $19880000 = 1.988 \times 10^7 = 0.1988 \times 10^8$

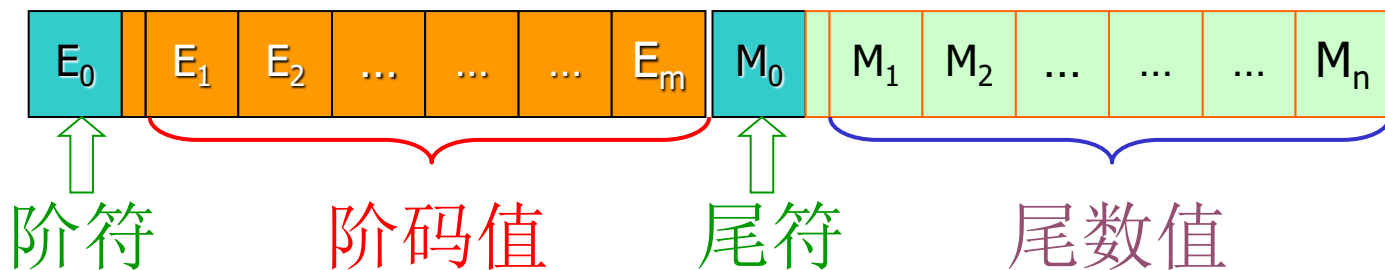
- 一个机器浮点数由**尾数**和**阶码**及其**符号**位组成

- **尾数**: 用**定点小数**表示, 给出有效数字的位数, 决定了浮点数的**表示精度**

- **阶码**: 用**定点整数**表示, 指明小数点所在位置, 决定了浮点数的**表示范围**

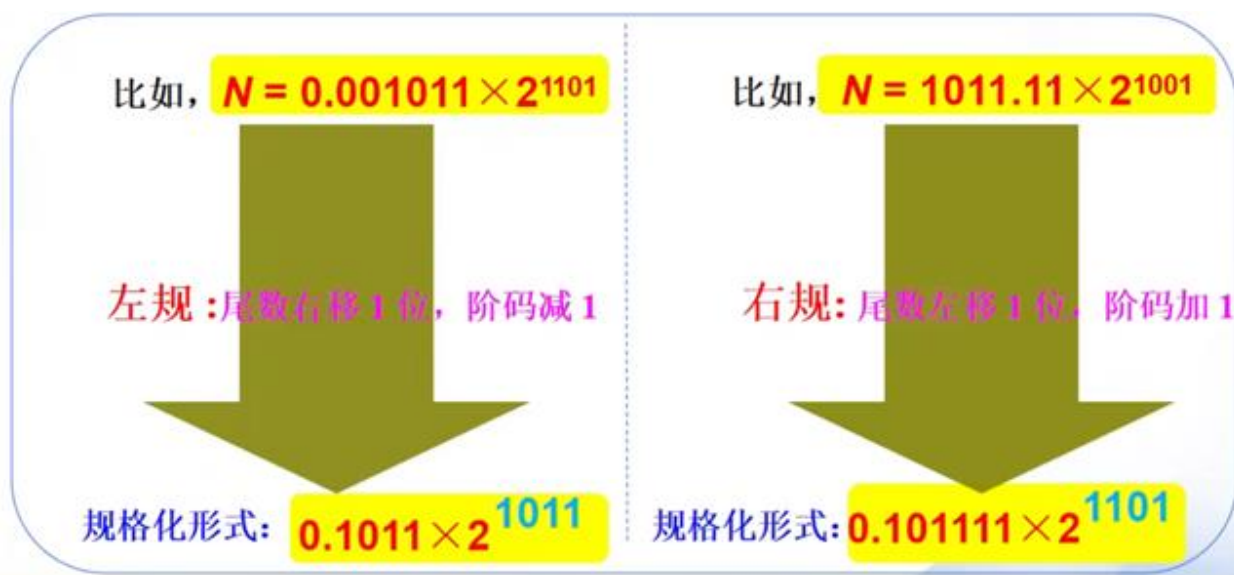
- 尾数越长, 表示的精度越高; 阶码越长, 表示的范围越广

- 在固定长度的浮点数格式内, 表示范围和表示精度是一对矛盾

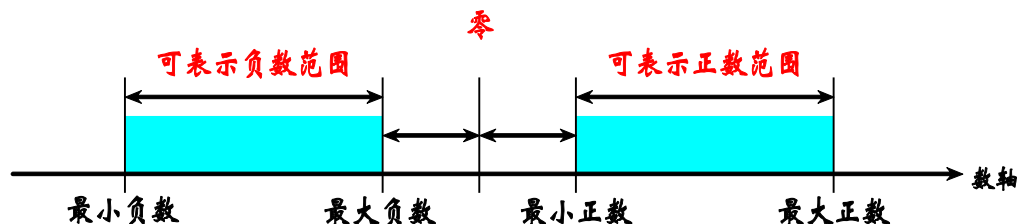


# 浮点数的规格化

- 目的：为了数据表示的唯一性，为了提高数据的表示精度
- 要求：尾数部分为纯小数且最高有效位为1，从而保证有n个有效数字
- 规格化通过尾数移位和修改阶码实现
  - 尾数的小数点每向左移动1位，阶码相应加1
  - 尾数的小数点每向右移动1位，阶码相应减1



# 浮点数的表示范围



$$\begin{aligned} \text{最大正数: } \text{Max}(+) &= S_{\max} \times r^{j_{\max}} \\ &= 0.11\dots1 \times 2^{11\dots1} \\ &= (1-2^{-n}) \times 2^{2^m-1} \end{aligned}$$

$$\begin{aligned} \text{最小负数: } \text{Min}(-) &= -S_{\max} \times r^{j_{\max}} \\ &= -0.11\dots1 \times 2^{11\dots1} \\ &= -(1-2^{-n}) \times 2^{2^m-1} \end{aligned}$$

$$\begin{aligned} \text{最小正数: } \text{Min}(+) &= S_{\min} \times r^{j_{-\max}} \\ &= 0.00\dots01 \times 2^{-11\dots1} \\ &= (2^{-n}) \times 2^{-(2^m-1)} \end{aligned}$$

$$\begin{aligned} \text{最大负数: } \text{Max}(-) &= -S_{\min} \times r^{j_{-\max}} \\ &= -0.00\dots01 \times 2^{-11\dots1} \\ &= -(2^{-n}) \times 2^{-(2^m-1)} \end{aligned}$$

非规格化形式

$$\begin{aligned} \text{最大正数: } \text{Max}(+) &= S_{\max} \times r^{j_{\max}} \\ &= 0.11\dots1 \times 2^{11\dots1} \\ &= (1-2^{-n}) \times 2^{2^m-1} \end{aligned}$$

$$\begin{aligned} \text{最小负数: } \text{Min}(-) &= -S_{\max} \times r^{j_{\max}} \\ &= -0.11\dots1 \times 2^{11\dots1} \\ &= -(1-2^{-n}) \times 2^{2^m-1} \end{aligned}$$

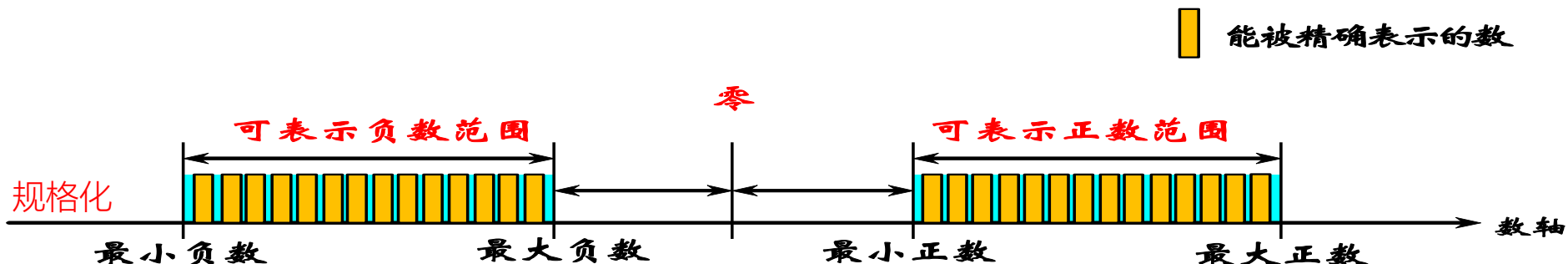
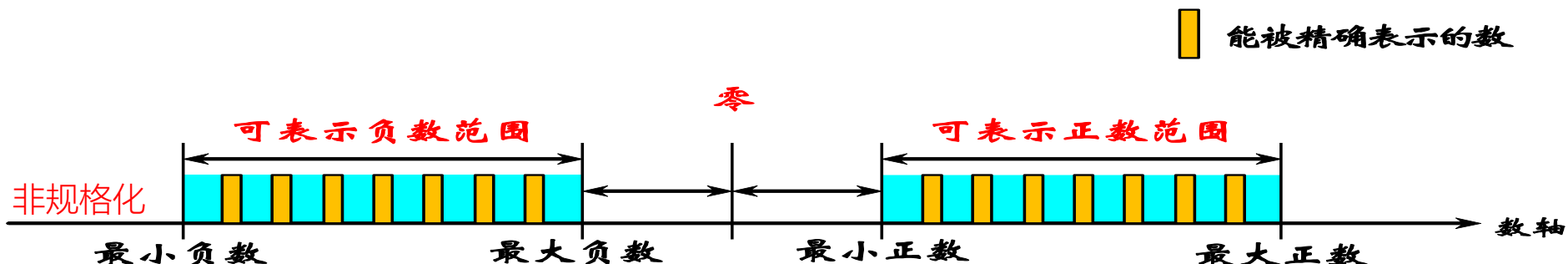
$$\begin{aligned} \text{最小正数: } \text{Min}(+) &= S_{\min} \times r^{j_{-\max}} \\ &= 0.10\dots0 \times 2^{-11\dots1} \\ &= (2^{-1}) \times 2^{-(2^m-1)} \end{aligned}$$

$$\begin{aligned} \text{最大负数: } \text{Max}(-) &= -S_{\min} \times r^{j_{-\max}} \\ &= -0.10\dots0 \times 2^{-11\dots1} \\ &= -(2^{-1}) \times 2^{-(2^m-1)} \end{aligned}$$

规格化形式

注：n为尾数位数，m为阶码位数

# 规格化的影响



损失一定的表示范围，以提高表示精度

## 1.3.3 字符的表示

- 字符必须按特定规则变成二进制编码才能输入计算机
- 字符编码：规定用怎样的二进制码来表示字符
- 西文字符：ASCII编码
  - **ASCII** (American Standard Code for Information Interchange, 美国信息交换标准码)。通用的是7位ASCII码，用7位二进制数表示一个字符的编码。共有 $2^7=128$ 个不同的编码
  - 主要的字符及其编码：
    - 48 ~ 57为10个阿拉伯数字0 ~ 9
    - 65 ~ 90为26个大写英文字母A ~ Z
    - 97 ~ 122为26个小写英文字母a ~ z
    - 32为空格，13为回车

# ASCII表

ASCII 值 控制字符

|     |     |
|-----|-----|
| 0   | NUT |
| 1   | SOH |
| 2   | STX |
| 3   | ETX |
| 4   | EOT |
| 5   | ENQ |
| 6   | ACK |
| 7   | BEL |
| 8   | BS  |
| 9   | HT  |
| 10  | LF  |
| 11  | VT  |
| 12  | FF  |
| 13  | CR  |
| 14  | SO  |
| 15  | SI  |
| 16  | DLE |
| 17  | DC1 |
| 18  | DC2 |
| 19  | DC3 |
| 20  | DC4 |
| 21  | NAK |
| 22  | SYN |
| 23  | TB  |
| 24  | CAN |
| 25  | EM  |
| 26  | SUB |
| 27  | ESC |
| 28  | FS  |
| 29  | GS  |
| 30  | RS  |
| 31  | US  |
| 127 | DEL |

可显示字符 →

← 控制字符

| 二进制       | 十进制 | 十六进制 | 图形        | Binary    | Decimal | Hex | Graphic | Binary    | Decimal | Hex | Graphic |
|-----------|-----|------|-----------|-----------|---------|-----|---------|-----------|---------|-----|---------|
| 0010 0000 | 32  | 20   | (空格) (sp) | 0100 0000 | 64      | 40  | @       | 0110 0000 | 96      | 60  | `       |
| 0010 0001 | 33  | 21   | !         | 0100 0001 | 65      | 41  | A       | 0110 0001 | 97      | 61  | a       |
| 0010 0010 | 34  | 22   | "         | 0100 0010 | 66      | 42  | B       | 0110 0010 | 98      | 62  | b       |
| 0010 0011 | 35  | 23   | #         | 0100 0011 | 67      | 43  | C       | 0110 0011 | 99      | 63  | c       |
| 0010 0100 | 36  | 24   | \$        | 0100 0100 | 68      | 44  | D       | 0110 0100 | 100     | 64  | d       |
| 0010 0101 | 37  | 25   | %         | 0100 0101 | 69      | 45  | E       | 0110 0101 | 101     | 65  | e       |
| 0010 0110 | 38  | 26   | &         | 0100 0110 | 70      | 46  | F       | 0110 0110 | 102     | 66  | f       |
| 0010 0111 | 39  | 27   | '         | 0100 0111 | 71      | 47  | G       | 0110 0111 | 103     | 67  | g       |
| 0010 1000 | 40  | 28   | (         | 0100 1000 | 72      | 48  | H       | 0110 1000 | 104     | 68  | h       |
| 0010 1001 | 41  | 29   | )         | 0100 1001 | 73      | 49  | I       | 0110 1001 | 105     | 69  | i       |
| 0010 1010 | 42  | 2A   | *         | 0100 1010 | 74      | 4A  | J       | 0110 1010 | 106     | 6A  | j       |
| 0010 1011 | 43  | 2B   | +         | 0100 1011 | 75      | 4B  | K       | 0110 1011 | 107     | 6B  | k       |
| 0010 1100 | 44  | 2C   | ,         | 0100 1100 | 76      | 4C  | L       | 0110 1100 | 108     | 6C  | l       |
| 0010 1101 | 45  | 2D   | -         | 0100 1101 | 77      | 4D  | M       | 0110 1101 | 109     | 6D  | m       |
| 0010 1110 | 46  | 2E   | .         | 0100 1110 | 78      | 4E  | N       | 0110 1110 | 110     | 6E  | n       |
| 0010 1111 | 47  | 2F   | /         | 0100 1111 | 79      | 4F  | O       | 0110 1111 | 111     | 6F  | o       |
| 0011 0000 | 48  | 30   | 0         | 0101 0000 | 80      | 50  | P       | 0111 0000 | 112     | 70  | p       |
| 0011 0001 | 49  | 31   | 1         | 0101 0001 | 81      | 51  | Q       | 0111 0001 | 113     | 71  | q       |
| 0011 0010 | 50  | 32   | 2         | 0101 0010 | 82      | 52  | R       | 0111 0010 | 114     | 72  | r       |
| 0011 0011 | 51  | 33   | 3         | 0101 0011 | 83      | 53  | S       | 0111 0011 | 115     | 73  | s       |
| 0011 0100 | 52  | 34   | 4         | 0101 0100 | 84      | 54  | T       | 0111 0100 | 116     | 74  | t       |
| 0011 0101 | 53  | 35   | 5         | 0101 0101 | 85      | 55  | U       | 0111 0101 | 117     | 75  | u       |
| 0011 0110 | 54  | 36   | 6         | 0101 0110 | 86      | 56  | V       | 0111 0110 | 118     | 76  | v       |
| 0011 0111 | 55  | 37   | 7         | 0101 0111 | 87      | 57  | W       | 0111 0111 | 119     | 77  | w       |
| 0011 1000 | 56  | 38   | 8         | 0101 1000 | 88      | 58  | X       | 0111 1000 | 120     | 78  | x       |
| 0011 1001 | 57  | 39   | 9         | 0101 1001 | 89      | 59  | Y       | 0111 1001 | 121     | 79  | y       |
| 0011 1010 | 58  | 3A   | :         | 0101 1010 | 90      | 5A  | Z       | 0111 1010 | 122     | 7A  | z       |
| 0011 1011 | 59  | 3B   | ;         | 0101 1011 | 91      | 5B  | [       | 0111 1011 | 123     | 7B  | {       |
| 0011 1100 | 60  | 3C   | <         | 0101 1100 | 92      | 5C  | ]       | 0111 1100 | 124     | 7C  |         |
| 0011 1101 | 61  | 3D   | =         | 0101 1101 | 93      | 5D  | ^       | 0111 1101 | 125     | 7D  | }       |
| 0011 1110 | 62  | 3E   | >         | 0101 1110 | 94      | 5E  | _       | 0111 1110 | 126     | 7E  | ~       |
| 0011 1111 | 63  | 3F   | ?         | 0101 1111 | 95      | 5F  |         |           |         |     |         |

## 1.3.3 字符的表示

### ■ 中文字符：

- **国标码**：常用汉字6763个，其中一级汉字3755个，二级汉字3008个。  
国标码用两个字节来表示一个汉字，每个字节的最高位为0
- **机内码**：在计算机内部对汉字进行存储、处理的编码。一个汉字的内码用两个字节存储，每个字节的最高位为1，以区别于ASCII码



图： 国标码 (4D3C)<sub>H</sub> (0100110100111100)<sub>B</sub>  
机内码 (CDBC)<sub>H</sub> (1100110110111100)<sub>B</sub>

灵： 国标码 (4169)<sub>H</sub> (0100000101101001)<sub>B</sub>  
机内码 (C1E9)<sub>H</sub> (1100000111101001)<sub>B</sub>

## 1.4 程序与程序设计语言

- 程序设计语言：指挥计算机工作的命令。没有程序，计算机将无法进行工作

- 程序设计语言的分类

- 按照人与机器的交互程度分类
- 按照程序的运行方式分类
- 根据程序语言解决问题的方法分类

更多内容：《程序设计II荣誉课程》（第2学期） 《编译原理》（第4学期）

## 1.4.1 按照人与机器的交互程度分类

### ■ 机器语言：由“0”和“1”组成的二进制码构成

- 优点：速度快，不需要翻译
- 缺点：依赖机器，可读性差，难以掌握

### ■ 汇编语言：用符号和助记符来代表机器语言

- 优点：速度快，可读性较好
- 缺点：依赖机器，较难掌握，需要翻译

### ■ 高级语言：用类英语描述对机器的指令

- 优点：可读性好，易于掌握；不依赖机器，可实现跨平台
- 缺点：翻译需要时间开销

```
1. int i, sum = 0;
2. for (i = 0; i < 10; i++) sum = sum + i;
```

高级语言

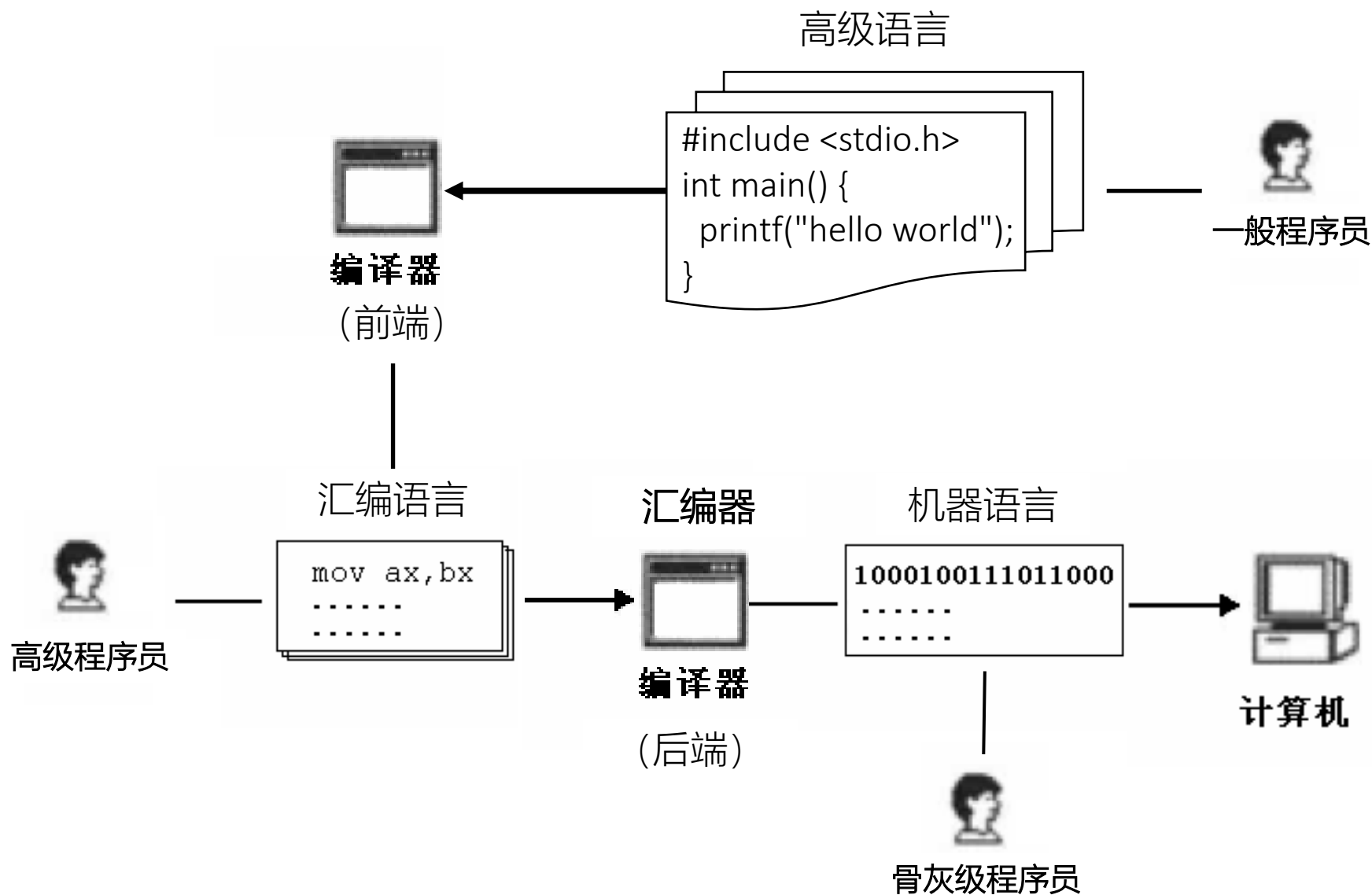
```
1. 1131: movl $0x0,-0x8(%rbp)
2. 1138: movl $0x0,-0x4(%rbp)
3. 113f: jmp 114b <main+0x22>
4. 1141: mov -0x4(%rbp),%eax
5. 1144: add %eax,-0x8(%rbp)
6. 1147: addl $0x1,-0x4(%rbp)
7. 114b: cmpl $0x9,-0x4(%rbp)
8. 114f: jle 1141 <main+0x18>
9. 1151: mov $0x0,%eax
```

汇编语言

83 45 fc 01

机器语言

## 1.4.1 按照人与机器的交互程度分类

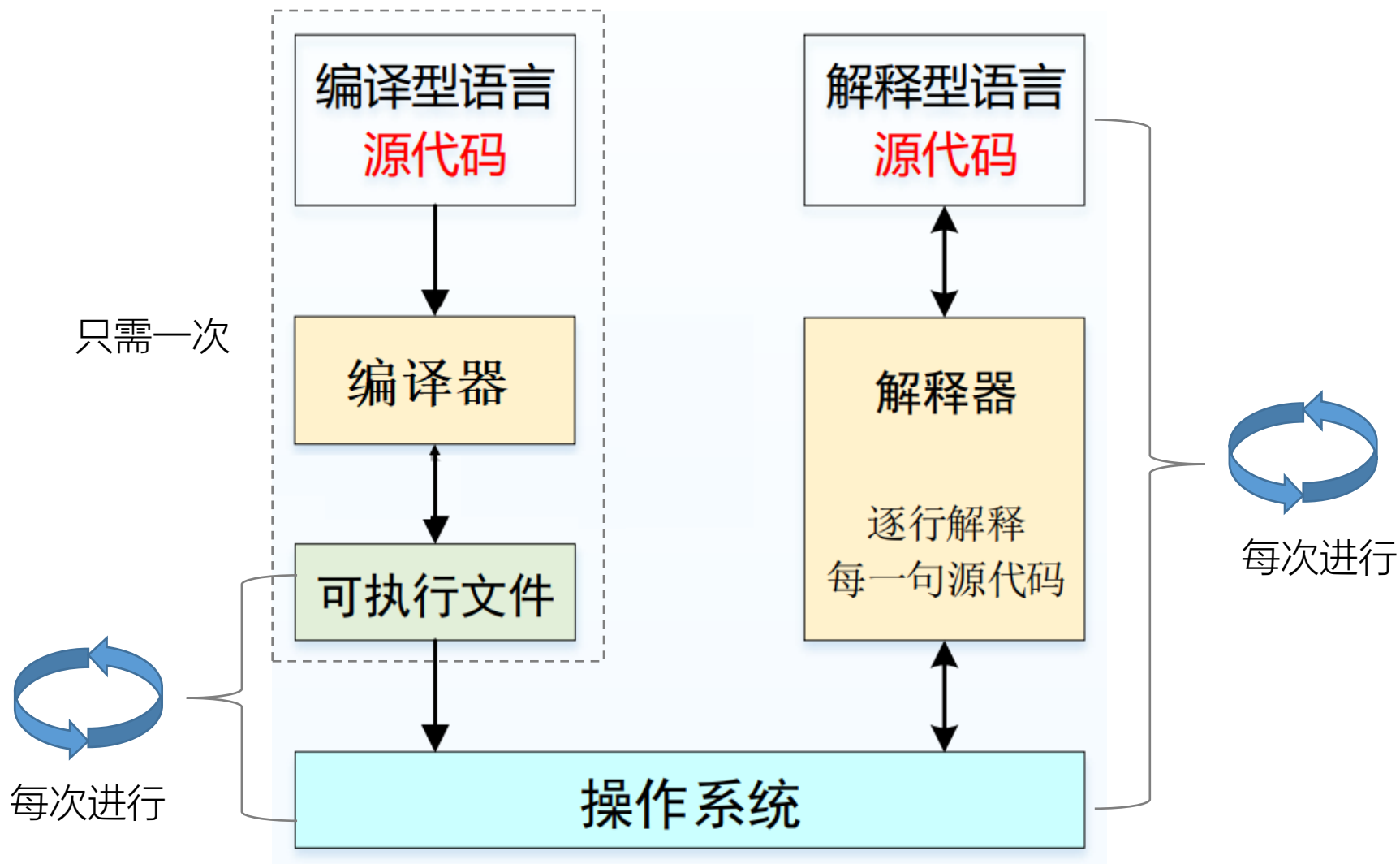


## 1.4.2 按照程序的运行方式分类

- 编译型语言：使用专门的编译器，针对特定的平台，将高级语言源代码一次性的编译成可被该平台硬件执行的机器码，并包装成该平台所能识别的可执行程序的格式
  - 优点：运行速度快，代码效率高
  - 缺点：代码需要编译才能运行，可移植性差，只能在兼容的系统上运行
  - 常见的编译型语言：C, C++, Pascal, .....
- 解释型语言：每次运行时，都使用专门的解释器对源程序逐行解释成特定平台的机器码并立即执行
  - 优点：可移植性好，只要有解释环境，可以在不同的操作系统上运行
  - 缺点：需要解释环境，运行起来比编译慢，占用资源也要多一些，代码效率低，不仅要给用户程序分配空间，解释器本身也占用了系统资源
  - 常见的解释型语言：Python, JavaScript, MATLAB

## 1.4.2 按照程序的运行方式分类

思考：Java是编译型语言还是解释型语言？



## 1.4.3 根据程序语言解决问题的方法分类

### ■ 面向过程的程序语言

- 按功能划分为若干个基本**模块**，形成一个树状结构
- 各模块间的关系尽可能简单，功能上相对独立
- 模块内部由顺序、选择和循环三种基本结构组成
- 其模块化实现的具体方法是使用**子函数**

### ■ 面向对象的程序语言

- 将数据及其操作方法封装在一起，作为一个整体——**对象**
- 对同类型对象抽象出其共性，形成**类**
- 类通过一个简单的外部接口，与外界发生关系
- 对象与对象之间通过消息进行通信

### ■ 函数式语言

### ■ 逻辑式语言

### ■ 专用语言

## 1.4.3 根据程序语言解决问题的方法分类

- 以下棋程序为例

- 面向过程：描述下棋的过程（逻辑）

- 步骤：1. 绘制棋盘；2. 行棋；3. 更新棋盘；4. 判断输赢；  
5. 返回步骤2；6. 输出结果

- 特点：步骤清楚，设计简单；但程序是一个整体，修改麻烦

- 面向对象：描述下棋涉及的对象及其行为关系

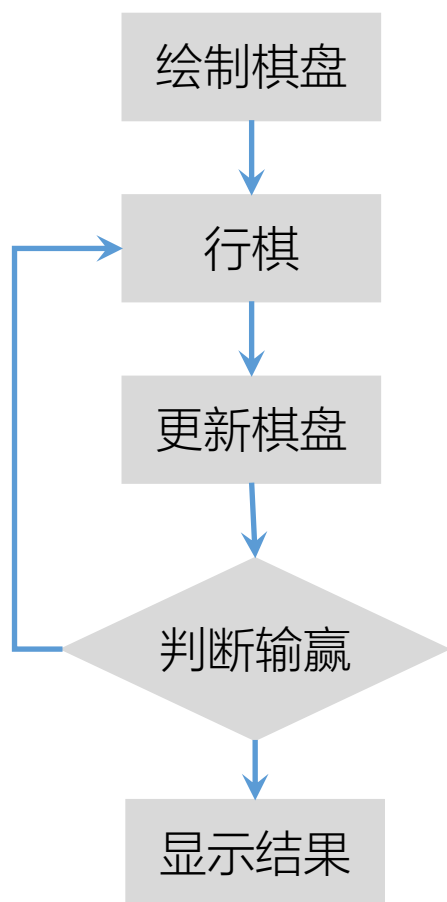
- 对象：对弈者、棋盘系统、规则系统

- 关系：对弈者获得用户的行棋位置后告知棋盘系统，棋盘系统在屏幕上更新棋局，同时利用规则系统判定棋局（犯规、输赢）

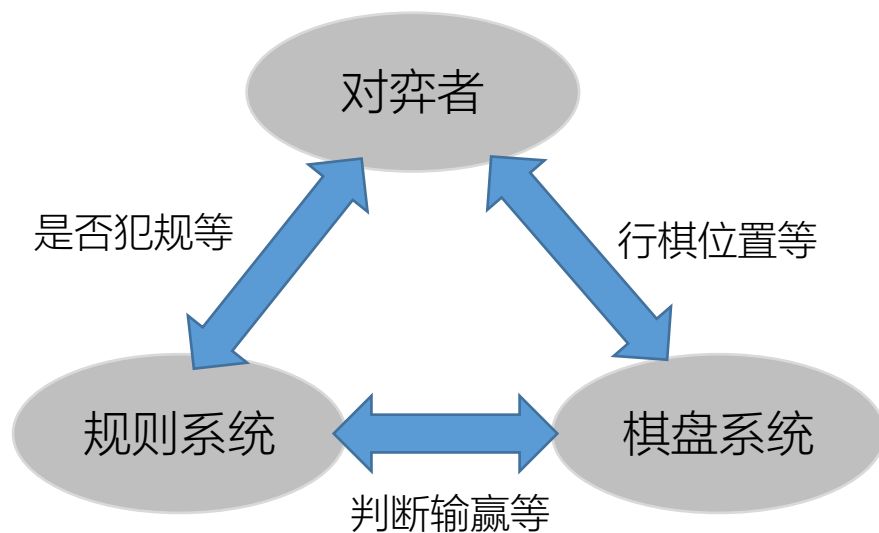
- 特点：设计复杂；但框架确定后，修改和替换方便

## 1.4.3 根据程序语言解决问题的方法分类

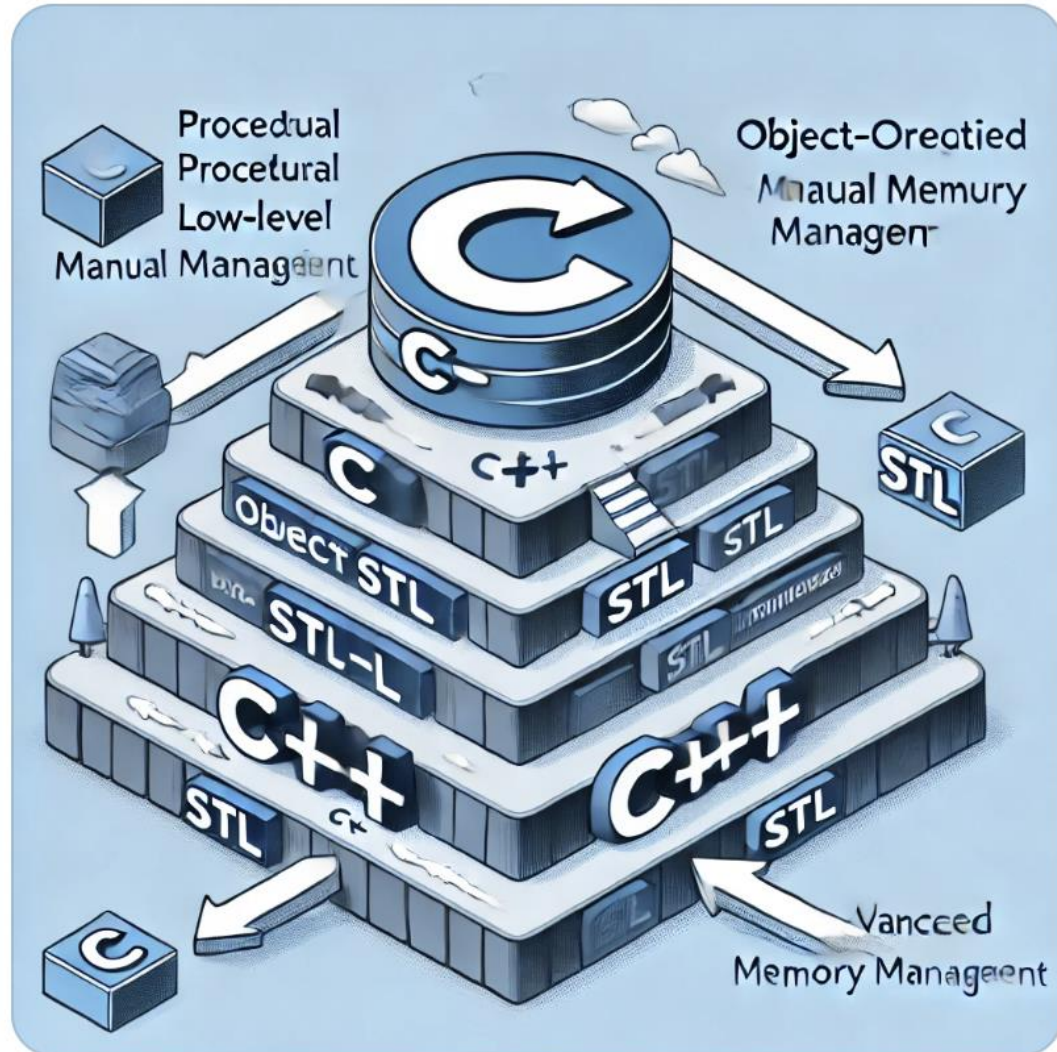
### 面向过程



### 面向对象



## 1.5 C/C++语言介绍



## 1.5.1 C语言介绍

- C语言是一门面向过程的编译型“高级语言”。它既有高级语言的特点，又具有低级汇编语言的特点
- C语言可以作为系统设计语言，编写工作系统应用程序，也可以作为应用程序设计语言，编写不依赖计算机硬件的应用程序
- 为什么学习C语言？
  - C语言相比其它高级语言如（C++，Java，C#）是低级语言，通过它可以更好地了解计算机底层工作原理。比如数据在内存中是如何存储的，如何直接访问内存中的数据等等。
  - C语言是其他任何高级语言的基础，学好C语言，就可以更容易掌握其他语言。语言都是相通的，C更专注于语言的实质，而不需要分散更多精力在集成开发环境的使用和抽象的数据概念上。

# C语言程序示例

```
1. #include <stdio.h> /*包含标准输入输出头文件*/
2. int main(int argc, char **argv) /*主函数*/
3. { /*函数体开始*/
4. printf("Hello World") ; /*调用库函数，输出字符串常量*/
5. return 0; /*返回值*/
6. }
```

```
1. #include <stdio.h> /*包含标准输入输出头文件*/
2. #define C 1 /*定义常量C的值为1*/
3. int main(int argc, char **argv) /*主函数*/
4. { /*函数体开始*/
5. int x; /*定义整数类型变量x*/
6. scanf("%d", &x); /*调用库函数，读入一个整数值，存入变量x中*/
7. x = x + C; /*计算x+C的值，结果存入变量x中*/
8. printf("%d\n", x); /*调用库函数，输出结果*/
9. return 0; /*返回值*/
10. }
```

## 1.5.2 C++语言介绍

- C++语言是一门面向对象的编译型“高级语言”，是对C语言的扩展
- C++语言继承了 C 语言的高效、灵活和底层控制的特点，同时在此基础上增加了面向对象编程、泛型编程、命名空间、异常处理等现代编程特性
- 为什么学习C++语言？
  - 相比于C语言，C++语言提升了代码的可读性、可维护性和可扩展性
  - C++语言更适合开发大型复杂的软件系统，而 C 语言则更适合底层开发和嵌入式系统等领域

# C++语言程序示例

```
1. #include <iostream> /*包含标准输入输出头文件*/
2. int main(int argc, char **argv) /*主函数*/
3. { /*函数体开始*/
4. std::cout << "Hello World" ; /*调用库函数，输出字符串常量*/
5. return 0; /*返回值*/
6. }
```

```
1. #include <iostream> /*包含标准输入输出头文件*/
2. #define C 1 /*定义常量C的值为1*/
3. int main(int argc, char **argv) /*主函数*/
4. { /*函数体开始*/
5. int x; /*定义整数类型变量x*/
6. std::cin >> x; /*调用库函数，读入一个整数值，存入变量x中*/
7. x = x + C; /*计算x+C的值，结果存入变量x中*/
8. std::cout << x; /*调用库函数，输出结果*/
9. return 0; /*返回值*/
10. }
```