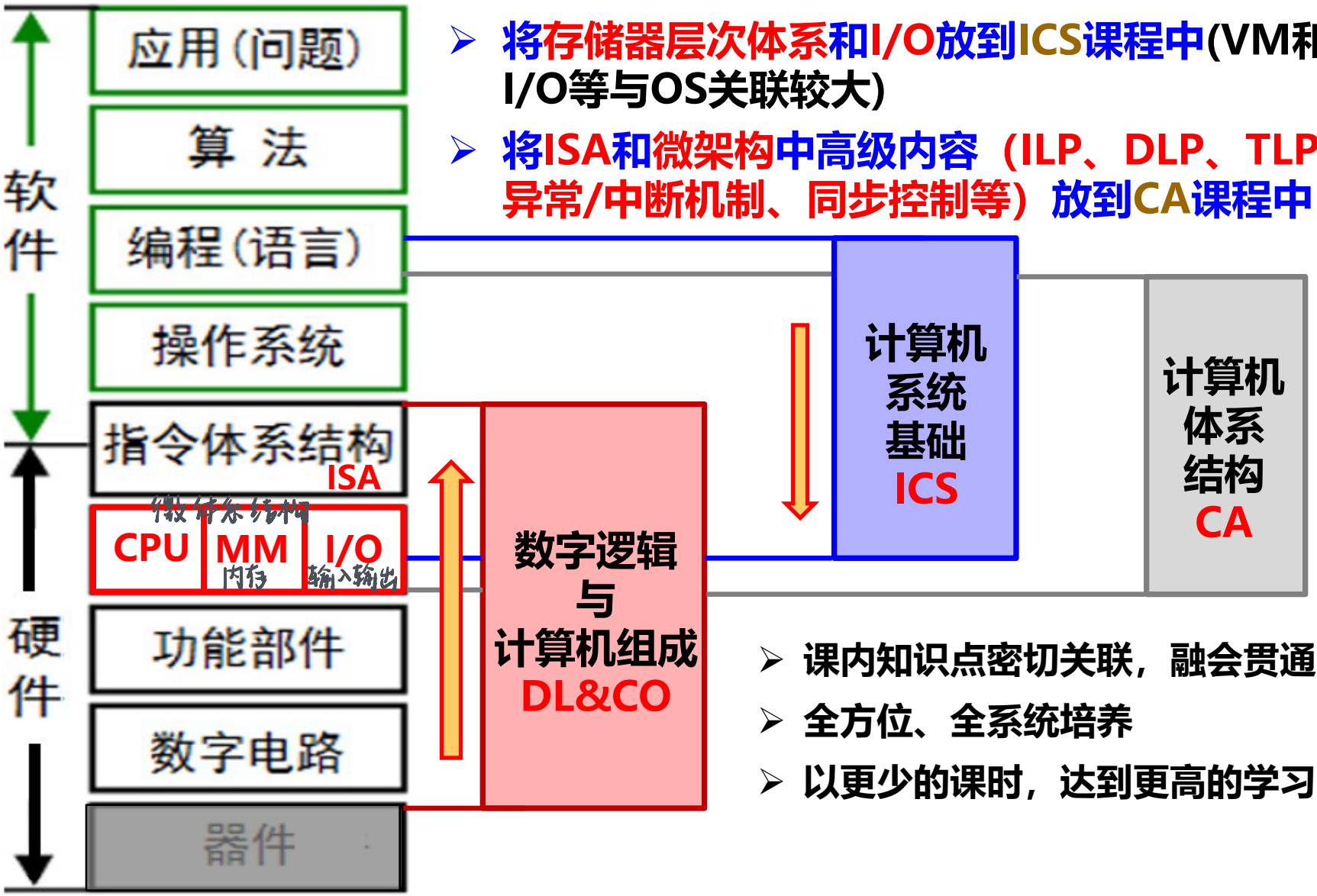
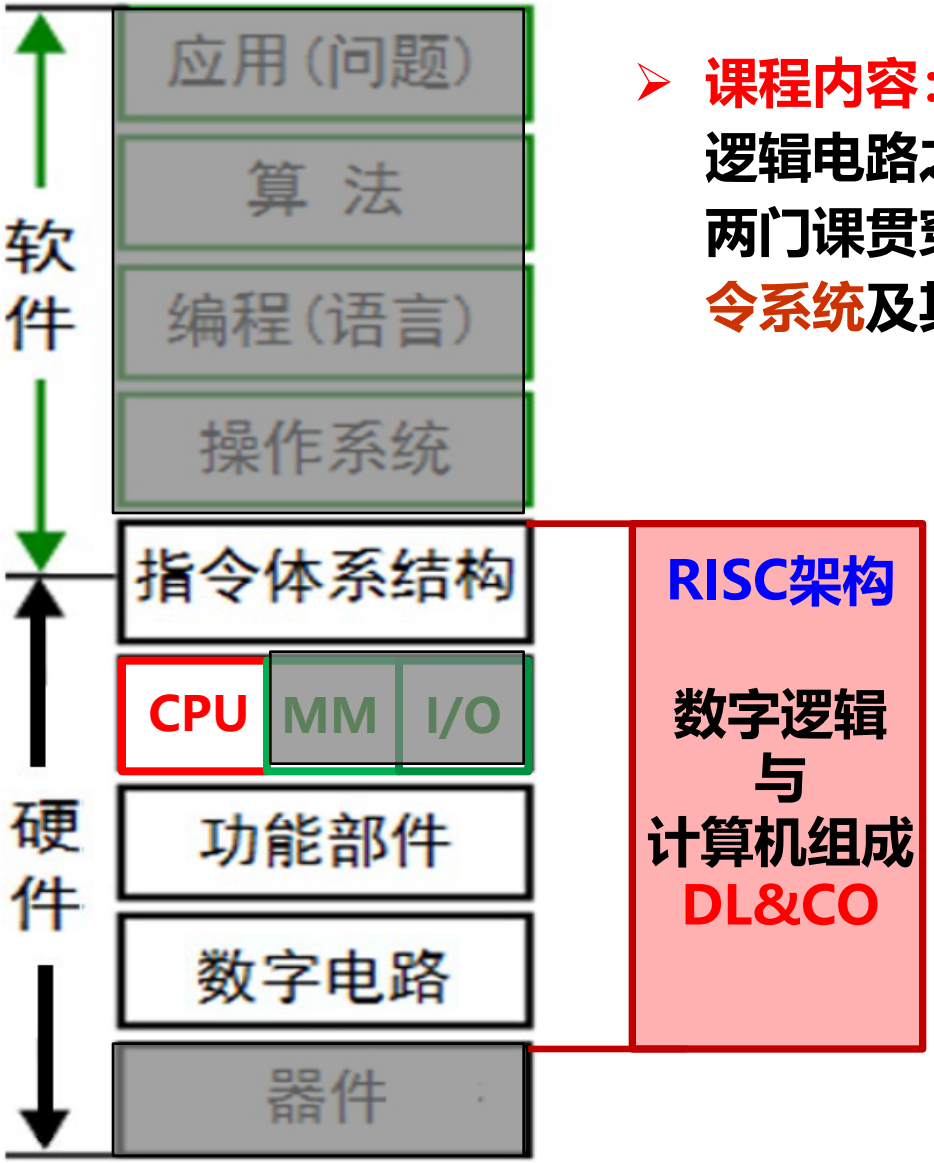
学什么——相关课程体系安排

- 将ISA、基本流水线CPU留在DL&CO课程中
- 将存储器层次体系和I/O放到ICS课程中(VM和I/O等与OS关联较大)
- 将ISA和微架构中高级内容（ILP、DLP、TLP、异常/中断机制、同步控制等）放到CA课程中



- 课内知识点密切关联，融会贯通
- 全方位、全系统培养
- 以更少的课时，达到更高的学习目标

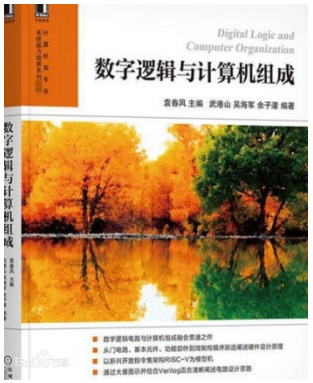
本课程“数字逻辑与计算机组成”安排



- **课程内容：**将**数据通路和控制器**设计作为数字逻辑电路之后的数字系统设计案例，将传统的两门课贯穿起来；以**RISC-V**为模型机介绍**指令系统**及其**单周期CPU**和**流水线CPU**设计。

- **教材内容：**包含传统组原教材中的主要内容（本课程不涉及其中MM和IO等），若不开设ICS课程，也可使用DL&CO教材完成完整知识体系的教学

课程基本信息



- **课程名称 Digital Logic and Computer Organization**
  - **数字逻辑与计算机组成**
- **教材:**
  - **《数字逻辑与计算机组成》，袁春风 武港山 吴海军 余子濠，机械工业出版社**
- **主要参考教材:**
  - 《数字逻辑与计算机组成习题解答与实验教程》
  - 《数字设计和计算机体系结构（原书第2版）》David Money Harris、Sarah. L. Harris 著，陈俊颖 译，机械工业出版
  - 《计算机组成与系统结构（第2版）》袁春风主编，清华大学出版社，2015 年
  - 《计算机组成与设计》，袁春风 余子濠，高等教育出版社，2020.10
- **在线学习**
  - 中国大学MOOC平台有多门“数字逻辑电路”、“计算机组成原理”
  - 本课堂录屏（杨若瑜）
  - 超星大视频（袁春风，2011年，计算机组成原理），B站可以观看
  - 爱课程中精品资源共享课（袁春风，2015年，计算机组成原理）

实验及考核方式 ★★

- 关于本课程的配套实验，可以先阅读此文:

https://mp.weixin.qq.com/s/V7QyK2VG7WFtrPTaCLQYFA

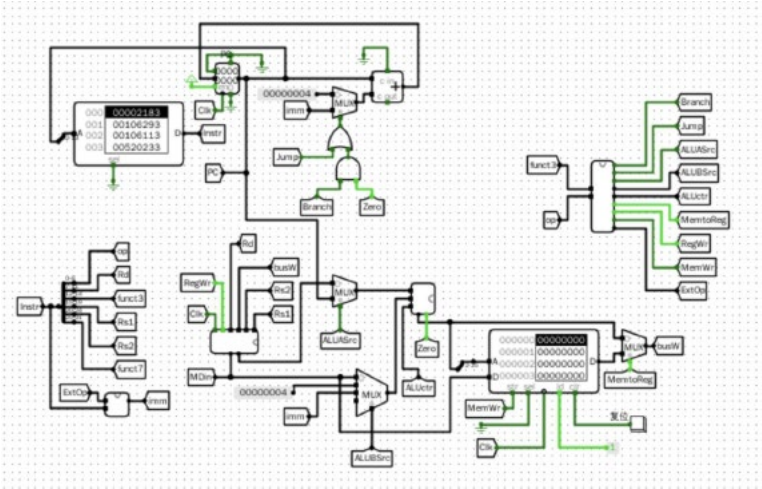
- 登录并访问课堂

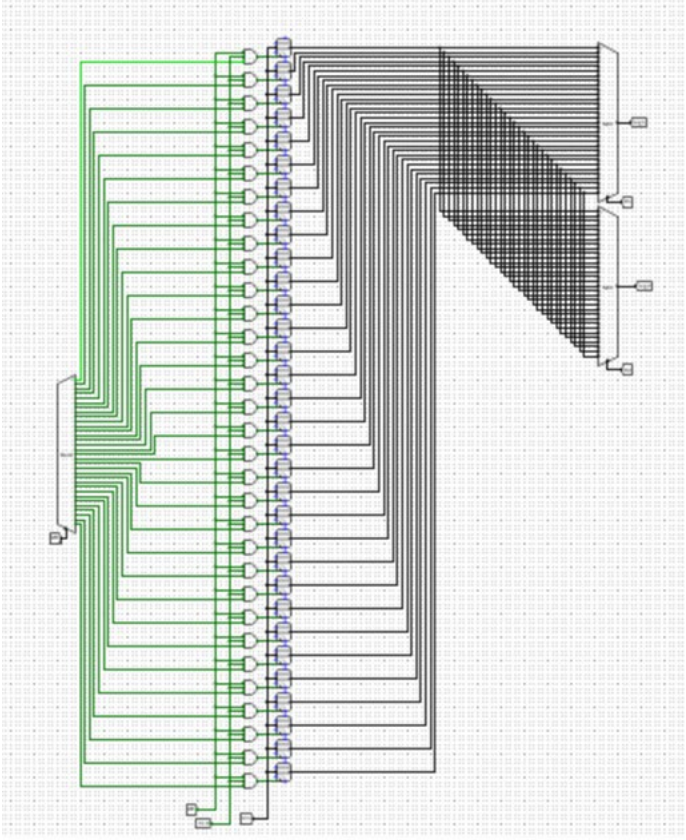
http://114.212.10.241/classrooms/kiteasnv?code=TWNVY

  - **实验1：基本逻辑部件设计**
  - **实验2：组合逻辑电路设计**
  - **实验3：同步时序电路设计**
  - **实验4：加法器和ALU设计**
  - **实验5：取指令部件设计**
  - **实验6：单周期CPU设计与测试**

- **考核方式**
  - **课后作业、问答等平时成绩：15%**
  - **六次实验成绩：35% （包括实验验收+报告）**
  - **期末考试成绩：50%**

- **请加入并关注QQ群——922962775**





—不要看不起简单的知识

—不要畏惧困难的知识

—不要割裂的看待各专业课

—有些必须死记硬背，但更多需要理解

—理解“计算机”的解剖结构（不只是硬件）

—找到自己喜欢的位置深入研究

—多问多想

第1章 二进制编码

第一讲 计算机系统概述

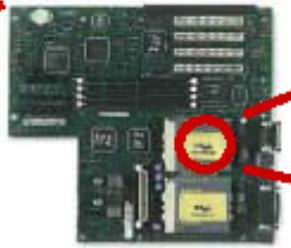
第二讲 二进制数的表示

第三讲 数值数据的编码表示

第四讲 非数值数据的编码表示及数据的宽度和存储排列

解剖一台计算机硬件（逻辑层次）

计算机通常包括主机和外设

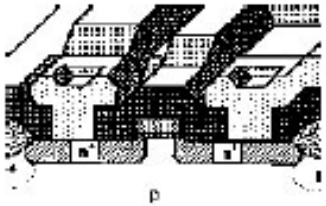


主机中包含多个电路板

每个电路板中有十几个集成电路

每个集成电路中有十几个模块

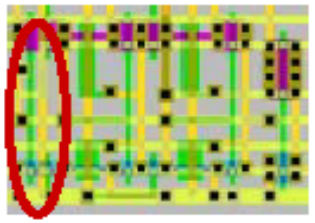
每个门电路实现基本的逻辑运算



所有信息都用二进制编码表示并存储

Gate:

每个单元中有十几个门电路



Cell:

每个模块中有上千万个单元

计算机是什么？ 一些关键词

- 十进制、二进制
- 运算，程序，算法，编程
- 存储、存储器

用二进制进行运算、
可以编程序实现各种功能、
得把程序“保管（存储）”好
……
—— 计算机一直是这样的吗？

- 寄存器、内存、外存（硬盘）

快、容量小 ——> 慢、容量大

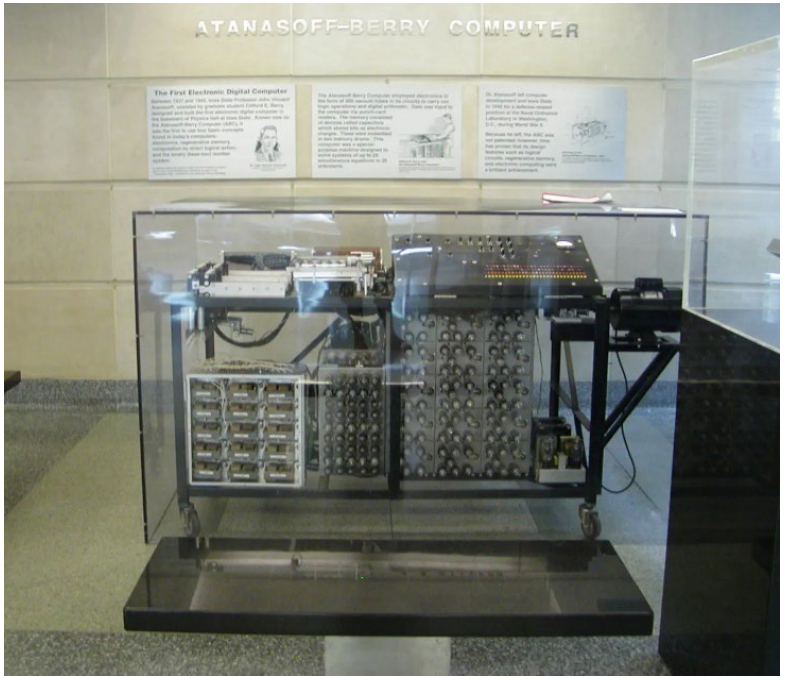
第一讲 计算机系统概述

- 冯.诺依曼结构计算机
  - 冯.诺依曼结构基本思想
  - 计算机硬件的基本组成
- 程序的表示和执行过程
  - 计算机如何实现程序的执行
  - 计算机硬件和软件的接口： ISA
- 计算机系统抽象层
  - 机器级语言和高级编程语言
  - 翻译程序：汇编、编译、解释

世界上第一台电子计算机ABC（非通用）

- 阿塔纳索夫-贝瑞计算机（Atanasoff-Berry Computer，简称ABC）——爱荷华州立大学的约翰·文特森·阿塔纳索夫（John Vincent Atanasoff）和他的研究生克利福特·贝瑞（Clifford Berry）在1937年设计，不可编程，仅仅设计用于求解线性方程组，并在1942年成功进行了测试。

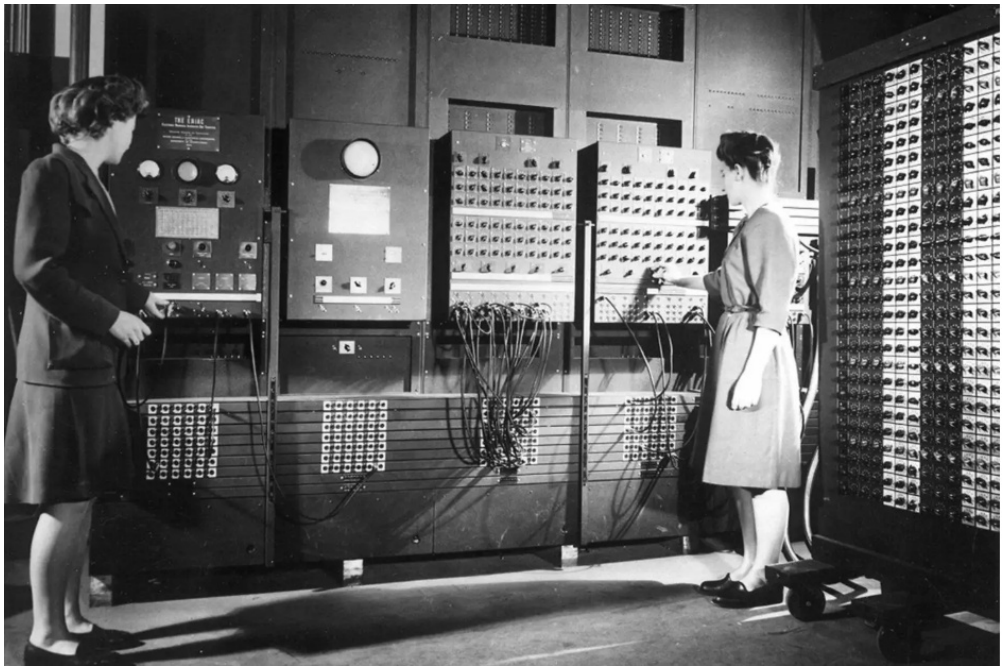
- 这台计算机是电子与电器的结合，电路系统中装有300个电子真空管执行数值计算与逻辑运算，机器使用电容器来进行数值存储，数据输入采用打孔读卡方法，采用二进制



世界上第一台通用电子计算机ENIAC

- 1945年，电子数字积分计算机，是世界上第一台通用计算机，也是继ABC之后的第二台电子计算机，它是图灵完全的电子计算机，能编程，解决不同的计算问题。

- 非存储程序
- 非冯诺依曼结构



*冯·诺依曼的故事

- 1944年，冯•诺依曼参加原子弹的研制工作，涉及到极为困难的计算。
- 1944年夏的一天，诺依曼巧遇美国弹道实验室的军方负责人戈尔斯坦，他正参与ENIAC的研制工作。
- 冯•诺依曼被戈尔斯坦介绍加入ENIAC研制组，1945年，在共同讨论的基础上，冯•诺依曼以“关于EDVAC的报告草案”为题，起草了长达101页的总结报告，发表了全新的“存储程序通用电子计算机方案”。

Electronic
Discrete
Variable
Automatic
Computer

现代计算机的原型

1946年，普林斯顿高等研究院（the Institute for Advance Study at Princeton，IAS）让冯·诺依曼设计“存储程序”计算机，其依据就是这份报告。被称为IAS计算机（1951年建成，即EDVAC）。

世界上第一台现代、存储程序式、通用、冯诺依曼结构、电子计算机?

- 1948年6月的曼彻斯特小型机（Manchester Baby） 是第一。
- EDSAC——电子延迟存储自动计算器（Electronic delay storage automatic calculator）是第二，由英国剑桥大学在1949年5月建成，
- EDVAC本身——冯诺依曼的草案启发了全世界，自己却由于某些技术和非技术原因，直到1951年才建成。

冯·诺依曼结构的核心？

- 在那个报告中提出的计算机结构被称为冯·诺依曼结构。

- 冯·诺依曼结构最重要的思想是什么？

  “存储程序(Stored-program)” 工作方式：

  任何要计算机完成的工作都要先被编写成程序，然后将程序和原始数据送入主存并启动执行。一旦程序被启动，计算机应能在不需操作人员干预下，自动完成逐条取出指令和执行指令的任务。

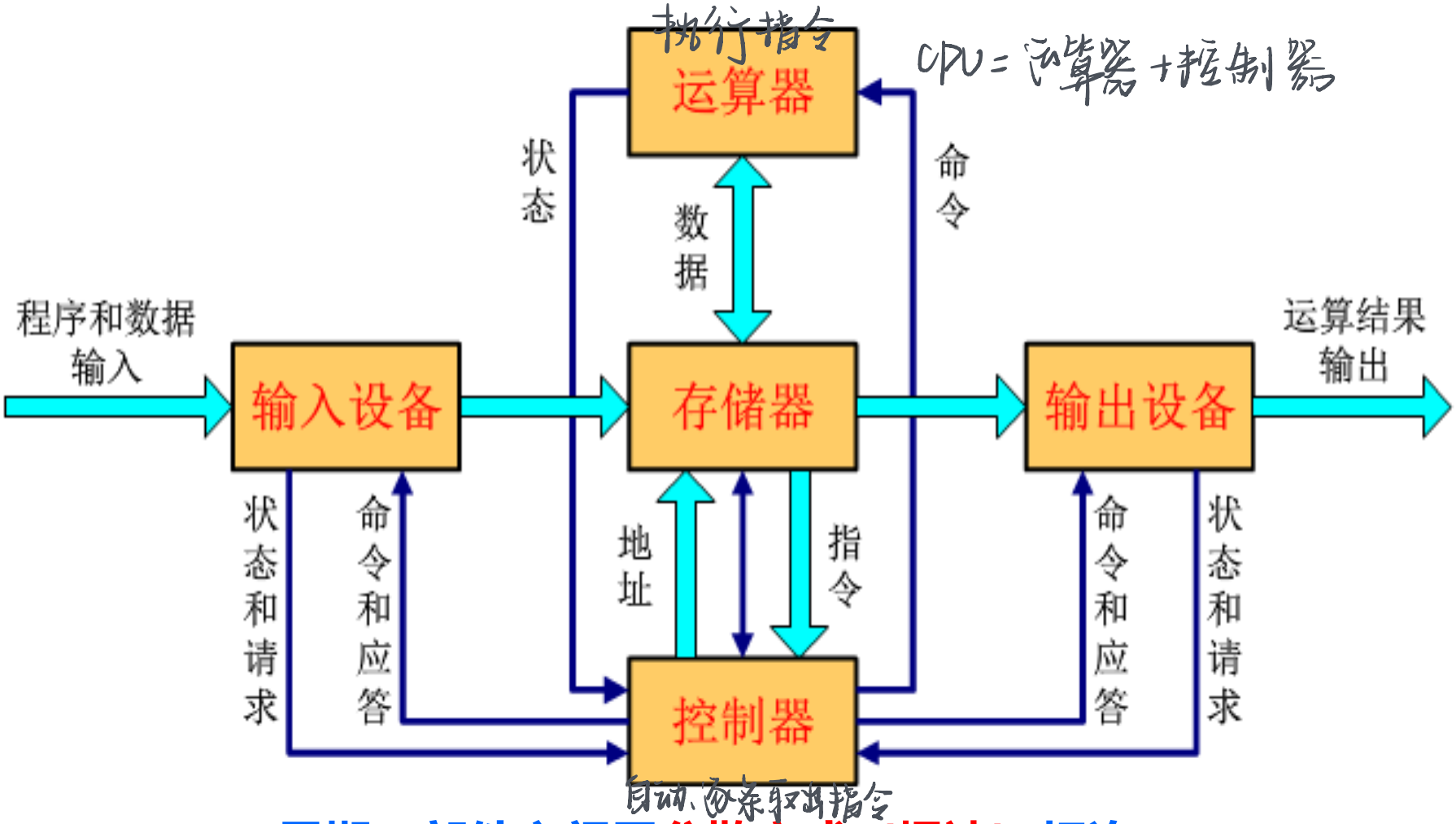
  顺序

- 冯·诺依曼结构计算机也称为冯·诺依曼机器（Von Neumann Machine）。
- 几乎现代所有的通用计算机大都采用冯·诺依曼结构。

“冯·诺依曼结构”计算机应该可以做哪些事？

- 应该有个主存，用来存放程序和数据
- 应该有一个自动逐条取出指令的部件
- 还应该有具体执行指令（即运算）的部件
- 程序由指令构成
- 指令描述如何对数据进行处理
- 应该有将程序和原始数据输入计算机的部件
- 应该有将运算结果输出计算机的部件

冯.诺依曼结构计算机模型



早期，部件之间用分散方式（慢速）相连

现在，部件之间大多用总线方式相连

趋势，总线+点对点（分散方式）高速连接

△ 归纳：冯诺依曼结构的完整思想

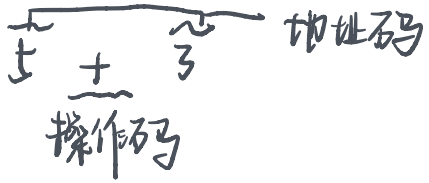
CPU：运算器+控制器
存储器
I/O：输入/出设备

综上所述

1. 计算机应由运算器、控制器、存储器、输入设备和输出设备五个基本部件组成。
2. 各基本部件的功能是：
   - 存储器不仅能存放数据，而且也能存放指令，形式上两者没有区别，但计算机应能区分数据还是指令；
   - 控制器应能自动取出指令来执行；
   - 运算器应能进行加/减/乘/除四种基本算术运算，并且也能进行一些逻辑运算和附加运算；
   - 操作人员可以通过输入设备、输出设备和主机进行通信。

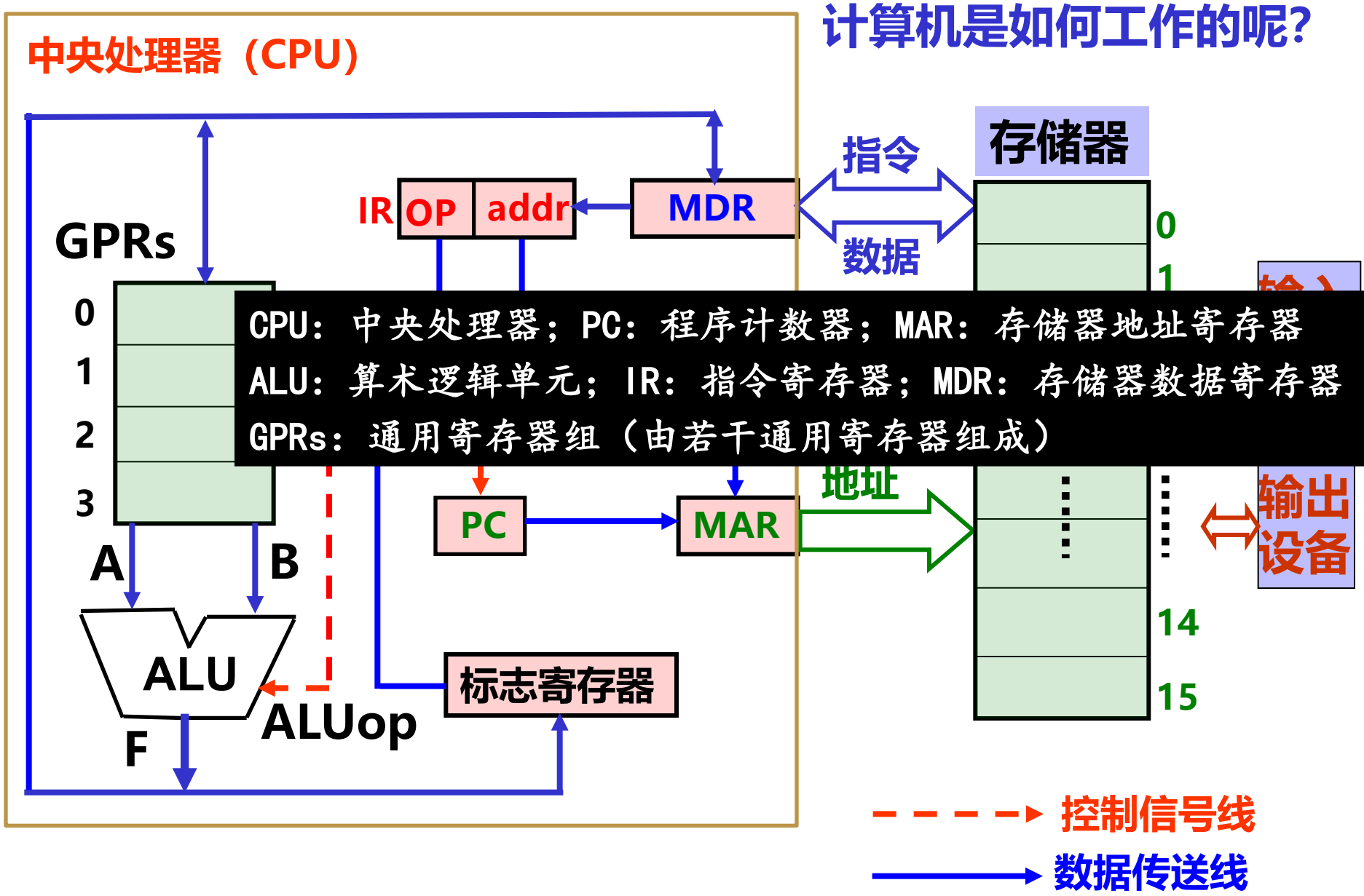
码：编码

3. 内部以二进制表示指令和数据。每条指令由操作码和地址码两部分组成。操作码指出操作类型，地址码指出操作数的地址。由一串指令组成程序。



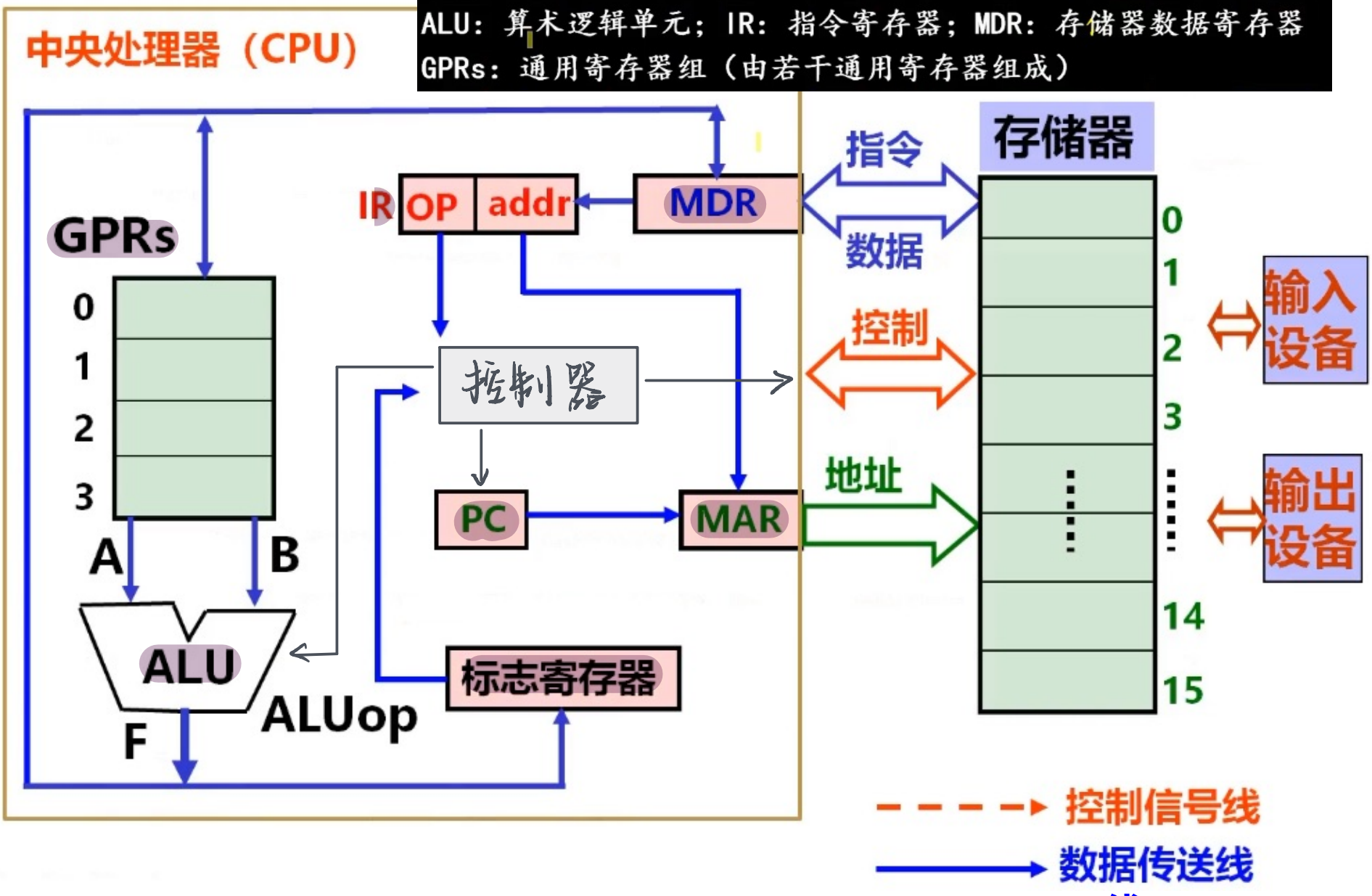
4. 采用“存储程序”工作方式。

现代计算机结构模型


计算机是如何工作的呢?
中央处理器（CPU）
GPRs
0
1
2
3
IR
OP
addr
MDR
指令
数据
存储器
0
1
14
15
CPU：中央处理器；PC：程序计数器；MAR：存储器地址寄存器
ALU：算术逻辑单元；IR：指令寄存器；MDR：存储器数据寄存器
GPRs：通用寄存器组（由若干通用寄存器组成）
PC
MAR
地址
输出设备
A
B
ALU
ALUop
F
标志寄存器
控制信号线
数据传送线

现代计算机结构模型

CPU：中央处理器；PC：程序计数器；MAR：存储器地址寄存器
ALU：算术逻辑单元；IR：指令寄存器；MDR：存储器数据寄存器
GPRs：通用寄存器组（由若干通用寄存器组成）



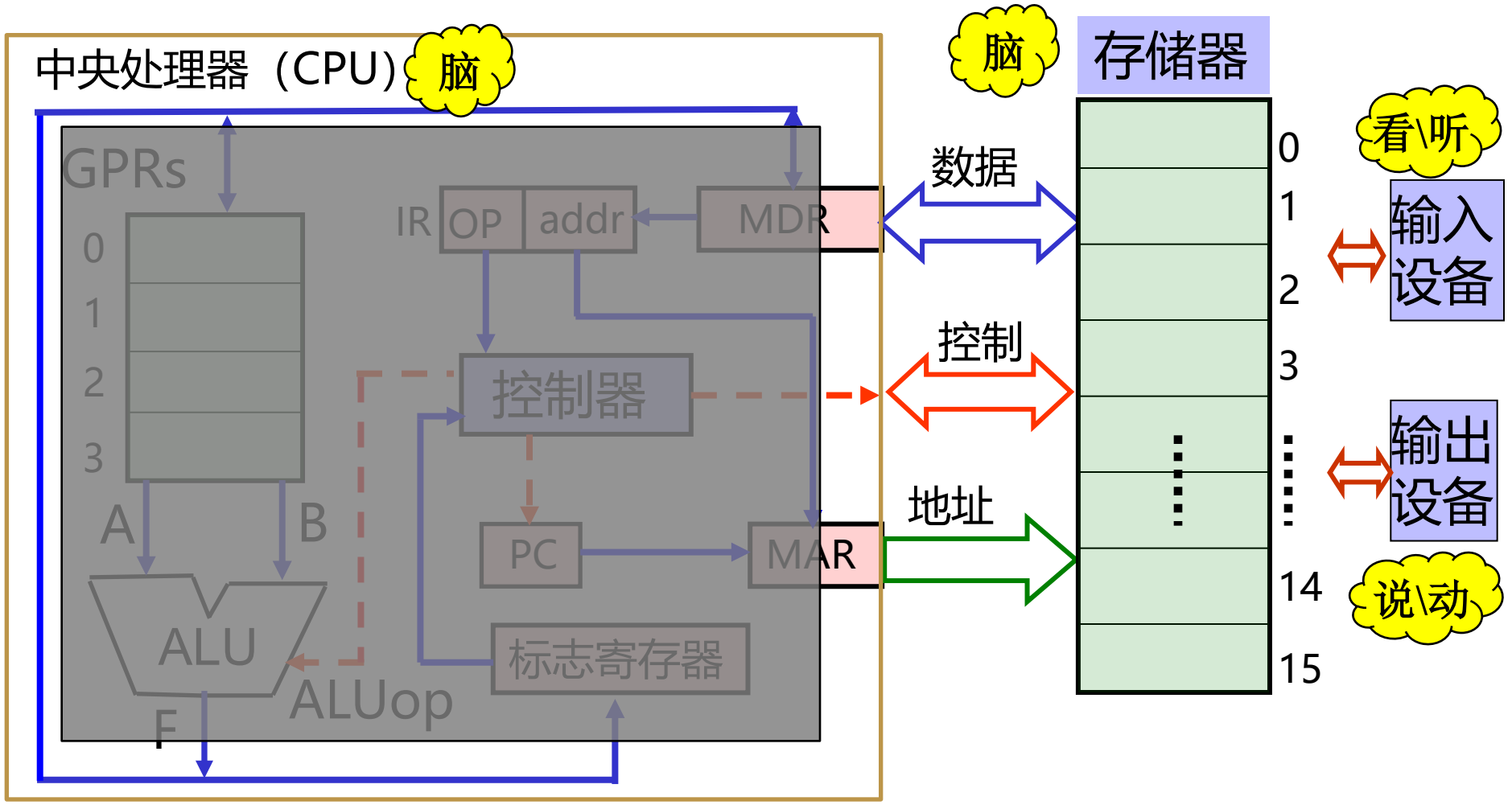
第一讲 计算机系统概述

- 冯.诺依曼结构计算机
  - 冯.诺依曼结构基本思想
  - 计算机硬件的基本组成
- 程序的表示和执行过程
  - 计算机如何实现程序的执行
  - 计算机硬件和软件的接口： ISA
- 计算机系统抽象层
  - 机器级语言和高级编程语言
  - 翻译程序：汇编、编译、解释

一个比喻

先想象一下你是怎样按菜谱做出一桌菜的?

再思考一下你看见一个人之后是怎样判断ta是谁的?



计算机是如何工作的？

“存储程序”工作方式
（忽略输入输出部分）

- **程序在执行前**

  数据和指令事先存放在存储器中：形式上没有差别，都是0/1序列

  每条指令和每个数据都有地址（有“存储”就必有“地址”）

  指令按序存放（注意！不一定按序执行！）

  程序起始地址置入PC

- **开始执行程序后：计算机能自动取出一条一条指令执行，在执行过程中无需人的干预。以下步骤在控制器的协调下完成。**

  第一步：根据PC取指令　　第二步：指令译码

  第三步：按地址取操作数　　第四步：指令执行

  第五步：按地址回写结果　　第六步：修改PC的值

  第七步：按新的PC，回到第一步，继续执行下一条指令

注意：步骤3和5可能不需要

程序由指令组成，若所有指令执行完，则程序执行结束

——思考一下：操作系统、PPT等等什么时候会执行完？

计算机是如何工作的？——指令的执行

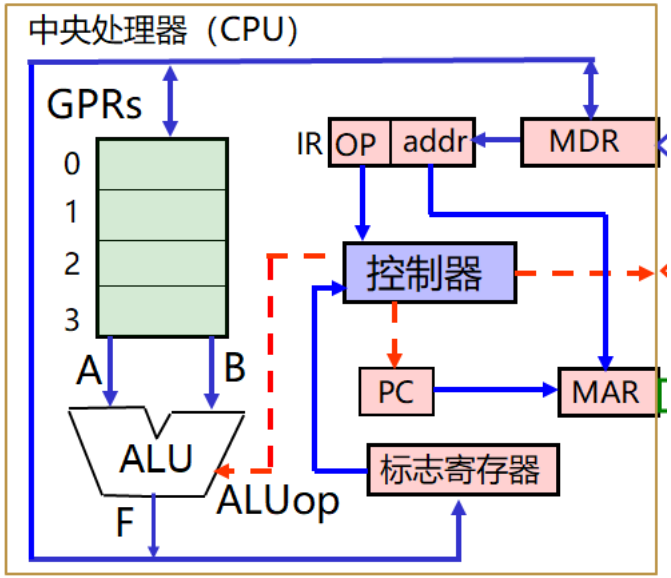
- **指令执行过程中，指令和数据被从存储器取到CPU，存放在CPU内的寄存器中，指令可以在IR中，数据在GPR中。**

指令中通常包括以下信息：

操作码（加减等）

源操作数1 或/和 源操作数2（的地址）

目的操作数地址

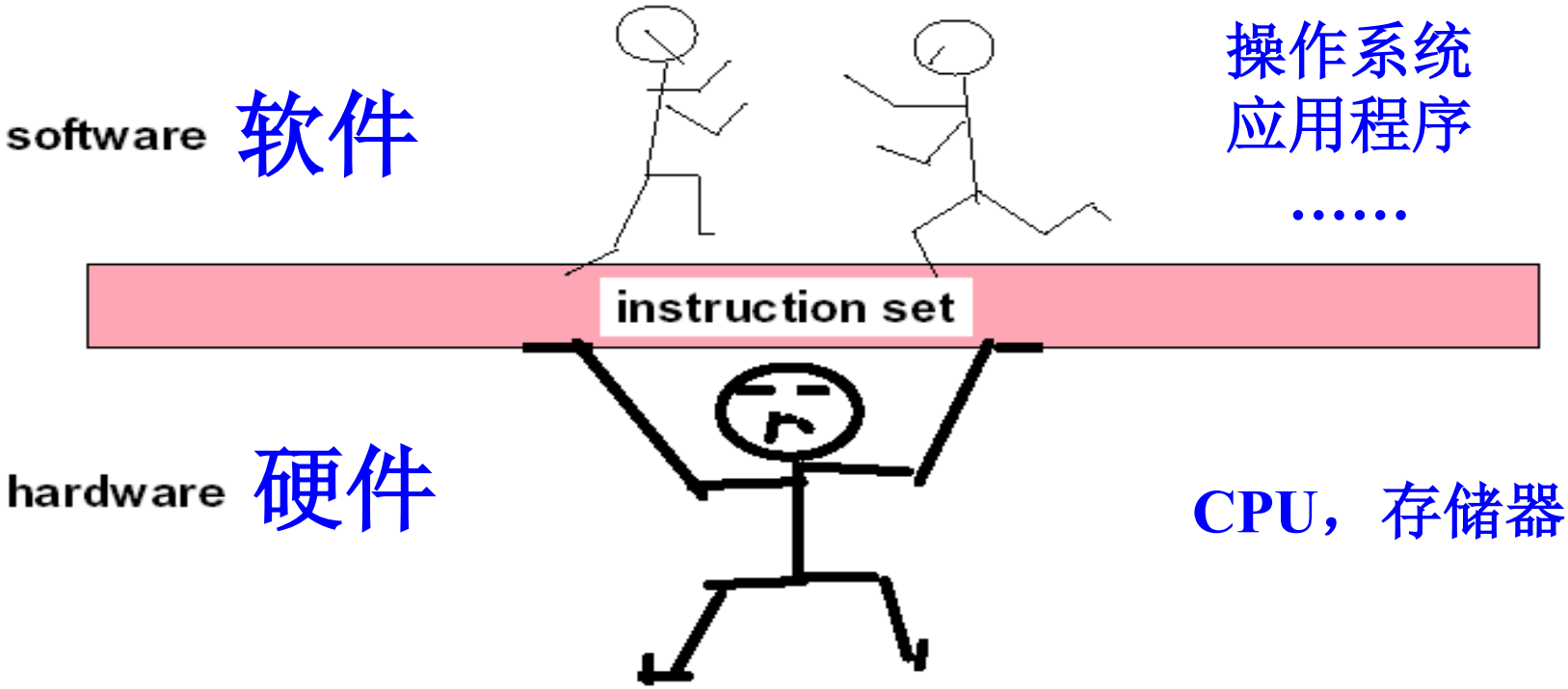


可见，在“计算机”的工作中

——（硬件） 和 “程序” （软件）缺一不可

—— 但是如何实现软硬件的合作呢?

软硬件交界面：ISA



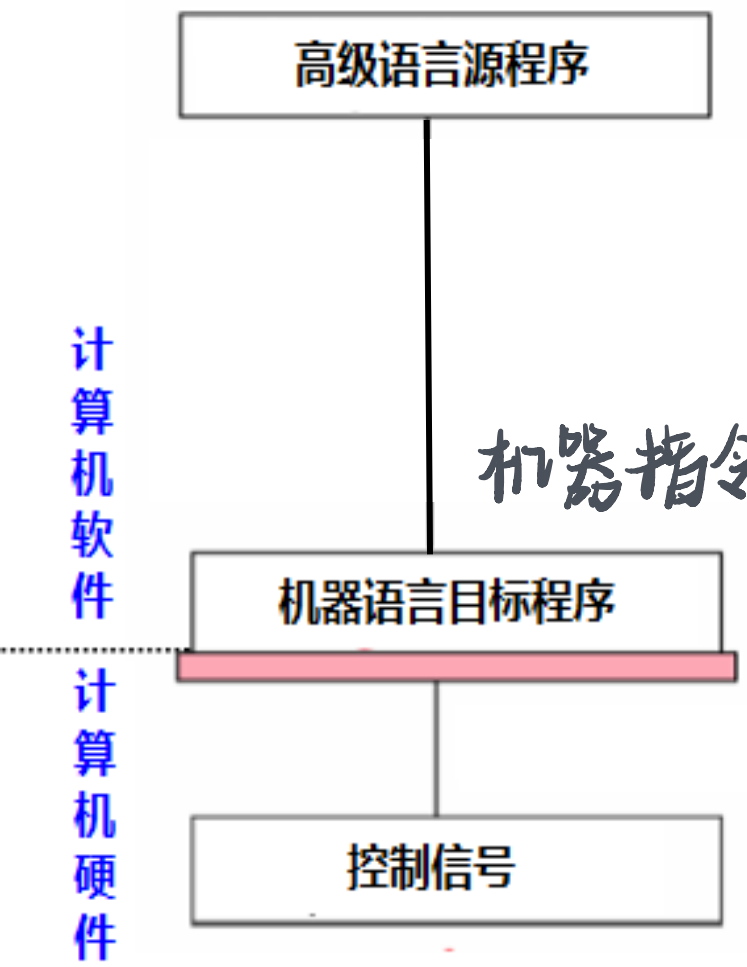
软、硬件界面：指令集体系结构（Instruction Set Architecture，ISA）

有时简称系统结构、体系结构，指令系统，甚至简称“架构”

指令集体系结构（ISA）

- **ISA指Instruction Set Architecture，即指令集体系结构**
- **ISA是一种规约（Specification）**
  - 可执行的指令的集合，包括指令格式、操作种类以及每种操作对应的操作数的相应规定；
  - 指令可以接受的操作数的类型；
  - 操作数所能存放的寄存器组的结构，包括每个寄存器的名称、编号、长度和用途；
  - 操作数所能存放的存储空间的大小和编址方式；
  - 操作数在存储空间存放时按照大端还是小端方式存放；
  - 指令获取操作数的方式，即寻址方式；
  - 指令执行过程的控制方式，包括程序计数器、条件码定义等。
- **ISA在计算机系统中是必不可少的一个抽象层，Why?**
  - **没有它，软件无法使用计算机硬件！**
  - **没有它，一台计算机不能称为“通用计算机”**

软硬件层次（简化）




```
tmp = a[i];
a[i] = a[i+1];
a[i+1] = tmp;
```

```
100011 00010 01000 0000000000000000
100011 00010 01001 0000000000000100
101011 00010 01001 0000000000000000
101011 00010 01000 0000000000000100
```

机器语言程序一定是0、1序列

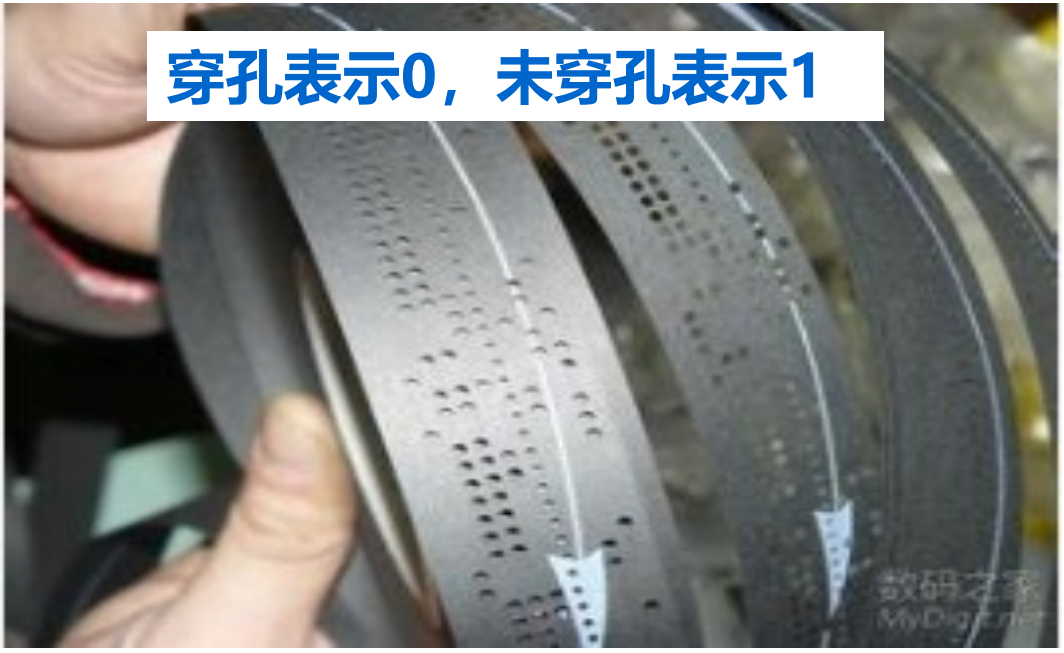
机器语言程序能被硬件直接执行

第一讲 计算机系统概述

- 冯.诺依曼结构计算机
  - 冯.诺依曼结构基本思想
  - 计算机硬件的基本组成
- 程序的表示和执行过程
  - 计算机如何实现程序的执行
  - 计算机硬件和软件的接口：ISA
- 计算机系统抽象层
  - 机器级语言和高级编程语言
  - 翻译程序：汇编、编译、解释

最早的程序开发过程

- 用机器语言编写程序，并记录在纸带或卡片上



输入：按钮、开关；所有信息都是0/1序列！
输出：指示灯等

例：第1条转移指令执行之后要跳到第4条指令去执行

```
0: 0101 0110
1: 0010 0100
2: ……
3: ……
4: 0110 0111
5: ……
6: ……
```

太原始了，无法忍受，咋办？

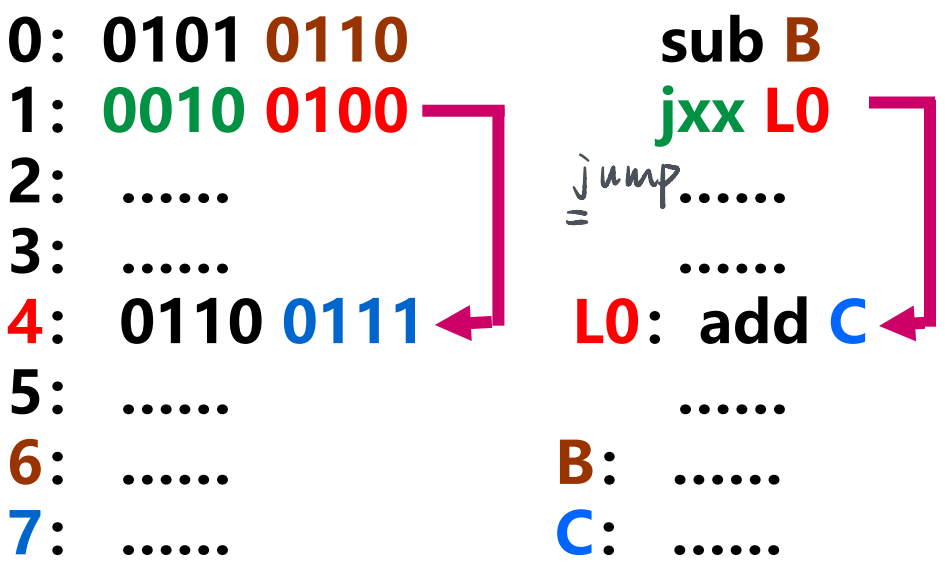
用符号表示而不用0/1表示！

若在第4条指令前加入新指令，则需重新计算第1条指令中的转移地址（不再是4了），然后重新打孔。不灵活！

书写、阅读困难！

用汇编语言开发程序

- 若用符号表示跳转位置和变量位置，是否简化了问题?
- 于是，汇编语言出现
  - 用助记符表示操作码
  - 用标号表示位置
  - 用助记符表示寄存器
  - .....



用汇编语言编写的优点是:

不会因为增减指令而需要修改其他指令

不需记忆指令二进制操作码，写程序比机器语言方便

可读性也比机器语言强

在第4条指令前加指令时不用改变sub、jxx和add指令中的编号（地址码）！

不过，这带来新的问题，是什么呢?

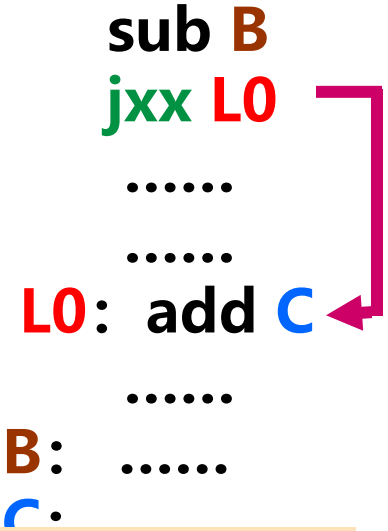
程序员容易了，可机器（硬件）不认识这些指令了!

需将汇编语言转换为机器语言!

用汇编程序转换

进一步认识机器级语言

- 汇编语言源程序由汇编指令构成
  - 用助记符和标号来表示的指令（与机器指令一一对应）
- 指令又是什么呢?
  - 包含操作码和操作数或其地址码
    （机器指令用二进制表示，汇编指令用符号表示）
  - 可以描述： 存取数，若干数的算术逻辑运算
    判断是否转移执行 等等



- 想象用汇编语
  （例如，用
  - 需要描述的
    硬件结构的

机器级语言 包括： 汇编语言 、 机器语言

都是面向计算机硬件的

汇编程序 能够将汇编语言源程序A转换为机器语言源程序B

A和B中的指令一定是一一对应的

B能被硬件直接执行

A不能被硬件直接执行

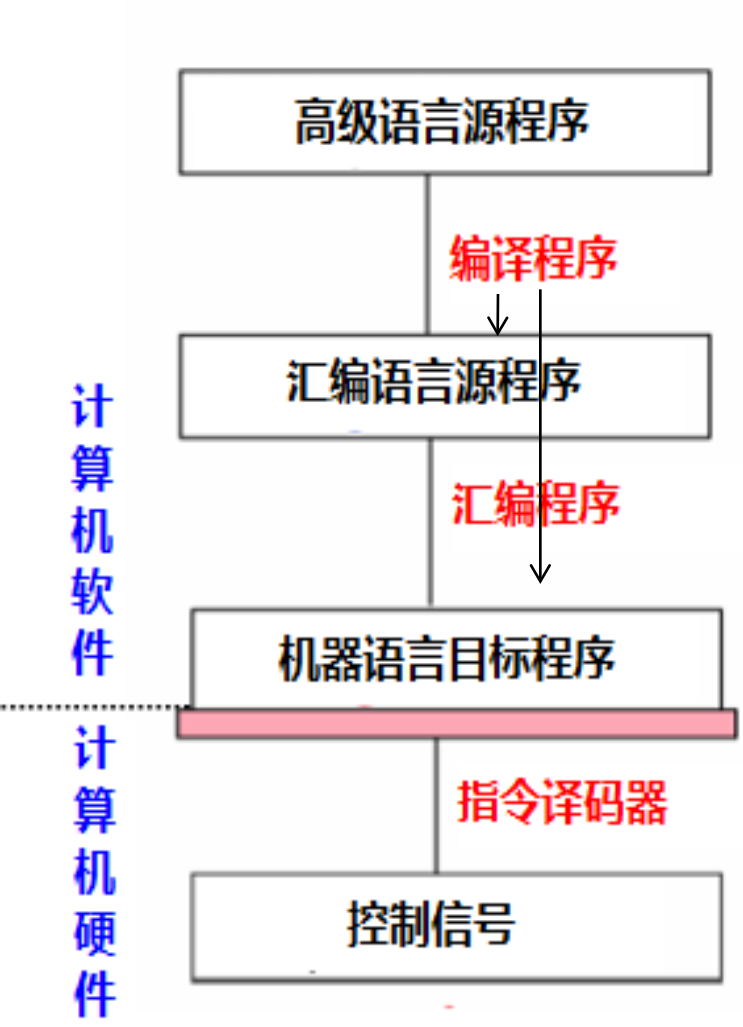
用高级语言开发程序

- **随着技术的发展，出现了许多高级编程语言**
  - 它们与具体机器结构无关
  - 面向算法描述，比机器级语言描述能力强得多
  - **高级语言中一条语句对应几条、几十条甚至几百条指令**
  - 有“面向过程”和“面向对象”的语言之分
  - **处理逻辑分为三种结构**
    - **顺序结构、选择结构、循环结构**

现在，几乎所有程序员都用高级语言编程，但最终要将高级语言转换为机器语言程序

  - **有两种转换方式：“编译”和“解释”**
    - 编译程序(Complier)：将高级语言源程序转换为机器级目标程序文件，最终即可通过启动目标程序来完成执行
    - 解释程序(Interpreter)：将高级语言语句逐条翻译成机器指令并立即执行，不生成目标文件。

软硬件层次（完整版）




```
tmp = a[i];
a[i] = a[i+1];
a[i+1] = tmp;
```

寄存器编号

```
lw $8, 0($2)
lw $9, 4($2)
sw $9, 0($2)
sw $8, 4($2)
```

每条指令由操作码和若干地址码组成

100011	00010	01000	0000000000000000
100011	00010	01001	0000000000000100
101011	00010	01001	0000000000000000
101011	00010	01000	0000000000000100

…, EXTop=1, ALUSelA=1, ALUSelB=11, ALUop=add, IorD=1, Read, MemtoReg=1, RegWr=1, …

任何高级语言程序最终通过执行若干条机器指令来完成！

开发和运行程序需什么支撑？

- 最早的程序开发很直接——
  - 直接输入指令和数据，启动后把第一条指令地址送PC开始执行
- 现代计算机用高级语言编程
- 用高级语言开发程序需要复杂的支撑环境（怎样的环境？）
  - 需要编辑器编写源程序
  - 需要一套翻译转换软件处理各类源程序
    - 编译方式：预处理程序、编译器、汇编器、链接器
    - 解释方式：解释程序
  - 需要一个可以执行程序的界面（环境）
    - GUI方式：图形用户界面
    - CUI方式：命令行用户界面

语言处理程序

（高级）语言处理系统 ＋

人机接口 ＋

语言的运行时系统

操作系统内核

操作系统

现代（传统）计算机系统的层次

语言处理系统包括：各种语言处理程序（如编译、汇编、链接）、运行时系统（如库函数、调试、优化等功能）

操作系统包括人机交互界面、提供服务功能的内核例程

支撑程序开发和运行的环境由**系统软件**提供

最重要的系统软件是**操作系统**和**语言处理系统**

语言处理系统运行在操作系统之上，操作系统利用指令管理硬件

归纳一下：软件（Software）

- System software(系统软件) - 简化编程，并使硬件资源被有效利用
  - 操作系统（Operating System）：硬件资源管理，用户接口
  - 语言处理系统：翻译（语言处理）程序+ Linker, Debug, etc …
    - 翻译程序(Translator)有三类：

      编译程序(Complier)：高级语言源程序→汇编/机器目标程序

      汇编程序(Assembler)：汇编语言源程序→机器目标程序

      解释程序(Interpreter)：将高级语言语句逐条翻译成机器指令并立即执行,不生成目标文件。
  - 其他实用程序: 如：磁盘碎片整理程序、备份程序等
- Application software(应用软件) - 解决具体应用问题/完成具体应用
  - 各类媒体处理程序：Word/ Image/ Graphics/…
  - 管理信息系统 (MIS)
  - Game，…

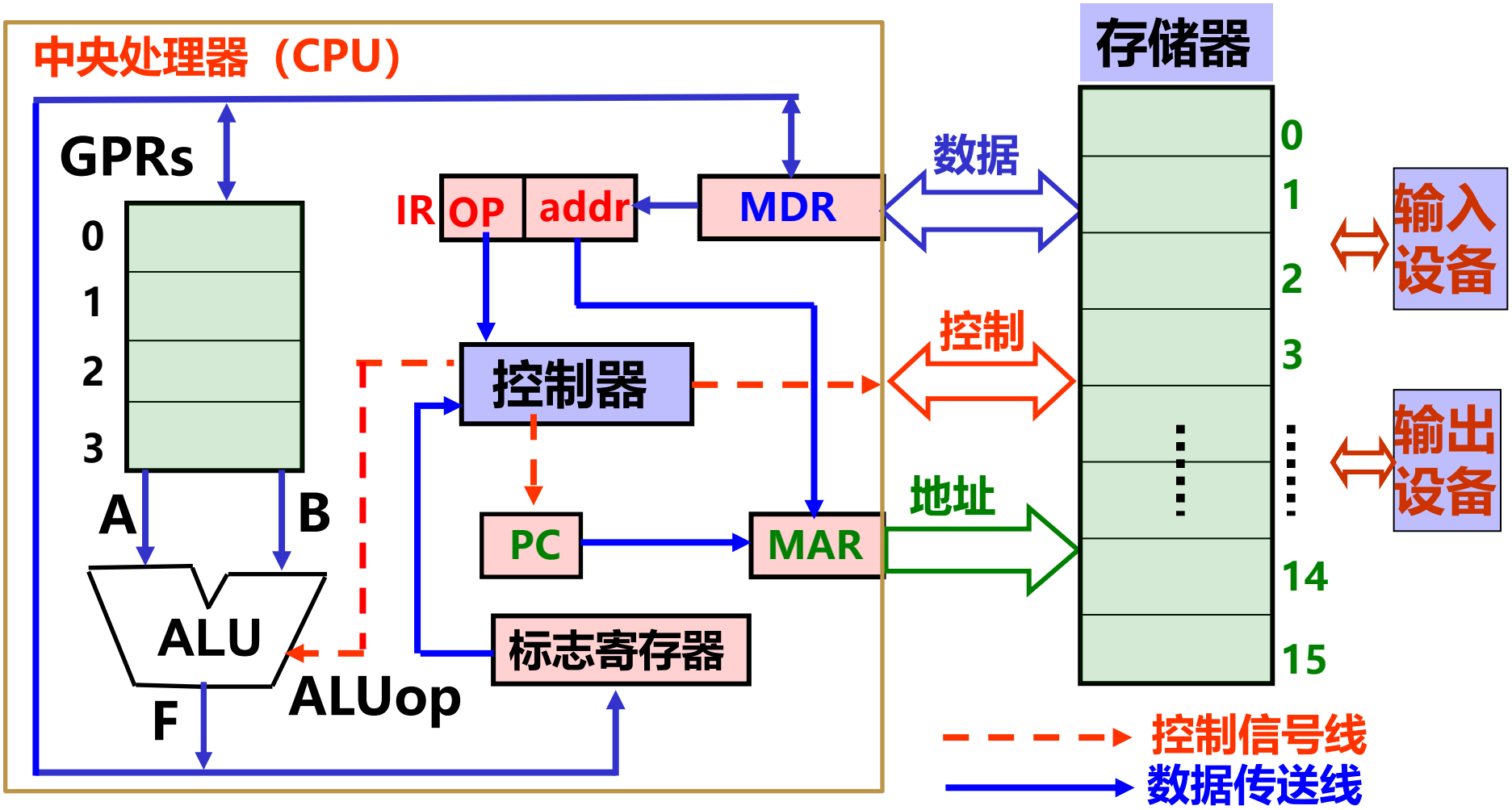
对于以下结构的机器，设计出几条指令

8位模型机M：8位定长指令字，4个GPR，16个主存单元

Load M#，R#　　（取数：将存储单元M内容装入寄存器R）

Store R#，M#　　（存数：将寄存器R内容装入存储单元M）

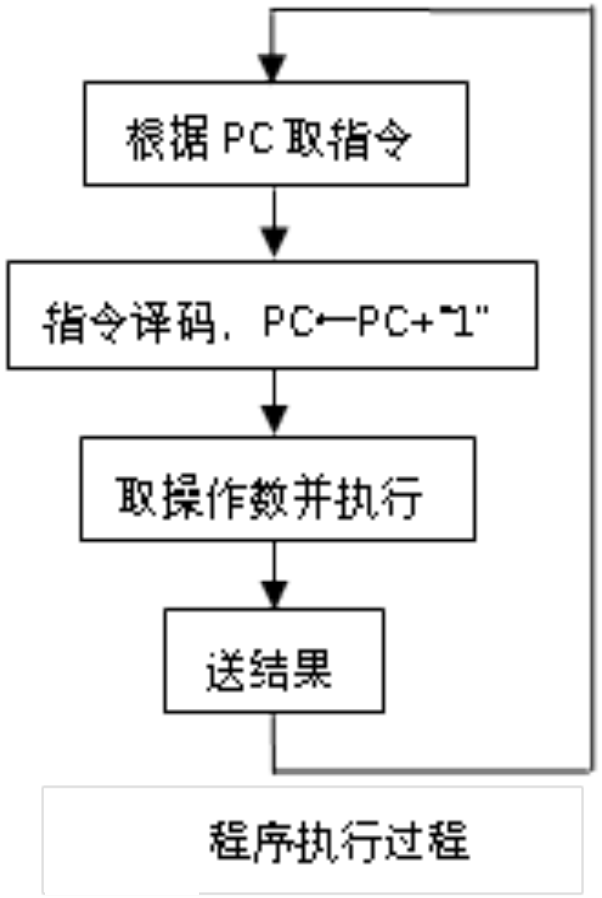
Add R#，R#　（加法，还有Sub，Mul等；操作数还可“R# M#”等）



程序和指令执行过程举例（仅为示意）

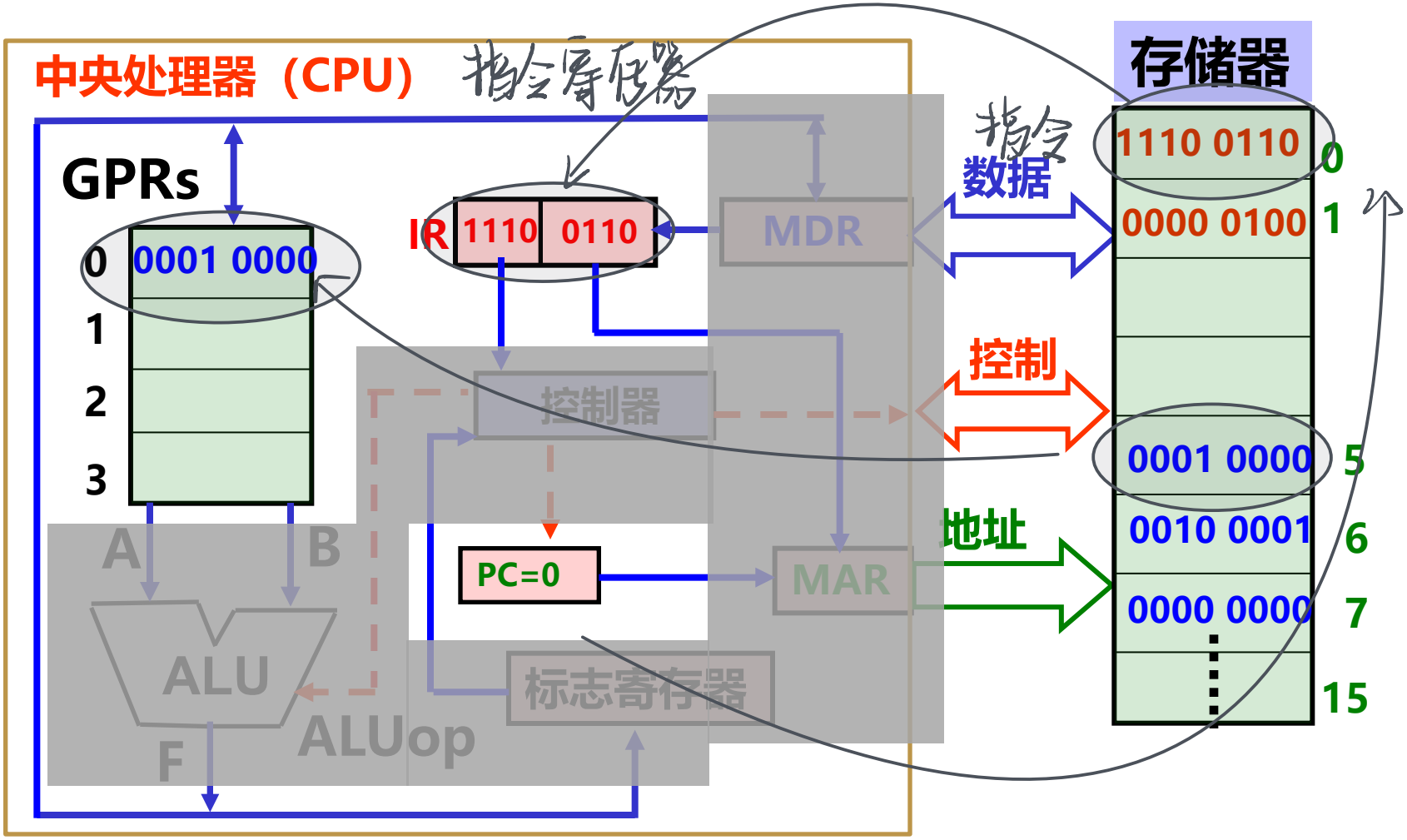
若在M上实现“z=x+y”，x和y分别存放在主存5和6号单元中，结果z存放在7号单元中，则程序在主存单元中的初始内容可为：

主存地址	主存单元内容	内容说明
0	1110 0110	取数x操作
1	0000 0100	传送操作
2	1110 0101	取数y操作
3	0001 0001	加操作
4	1111 0111	存数操作
5	0001 0000	操作数x
6	0010 0001	操作数y
7	0000 0000	结果z，初始值为0

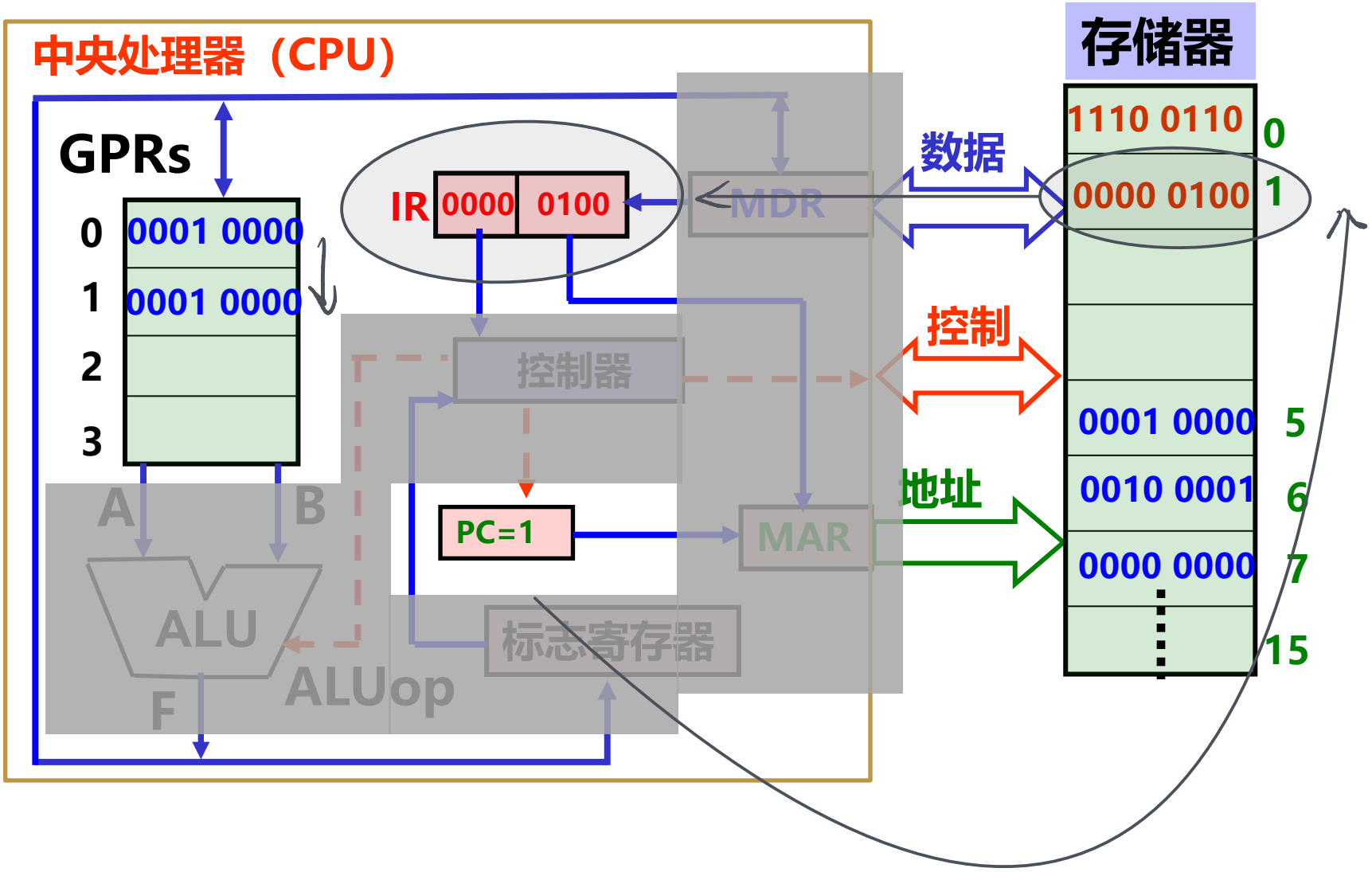


程序执行过程及其结果?

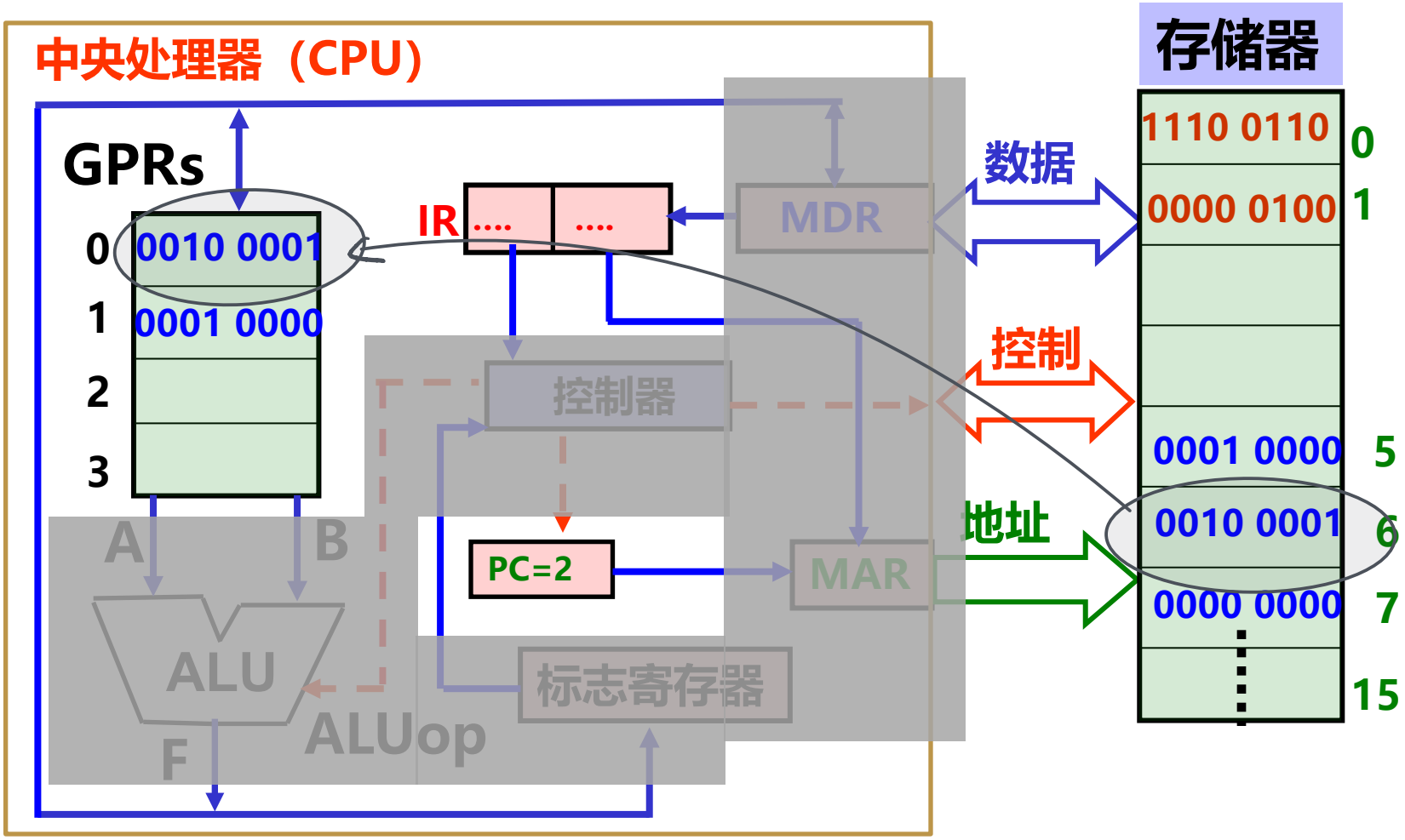
执行过程示意PC=0 取数x



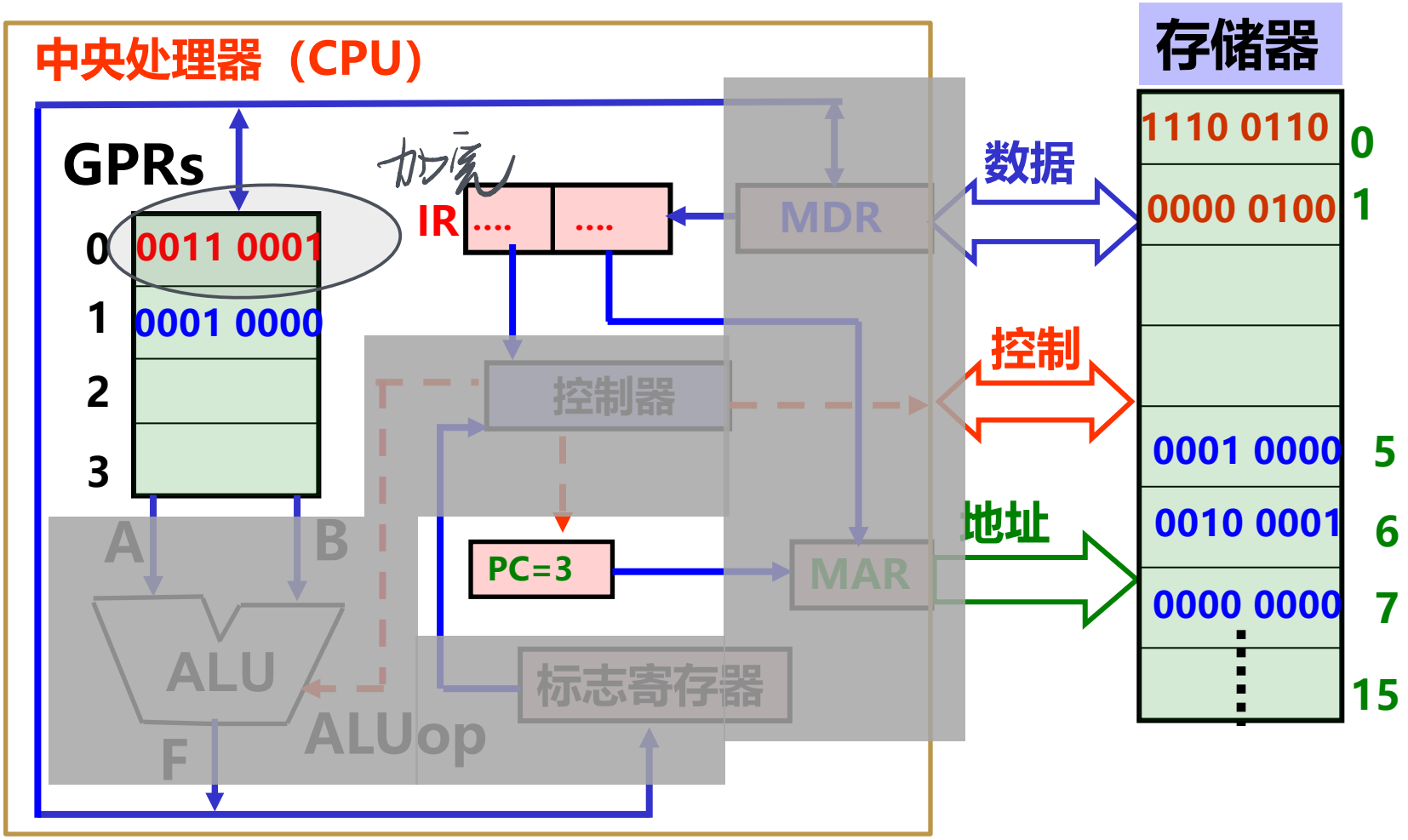
执行过程示意PC=1传送



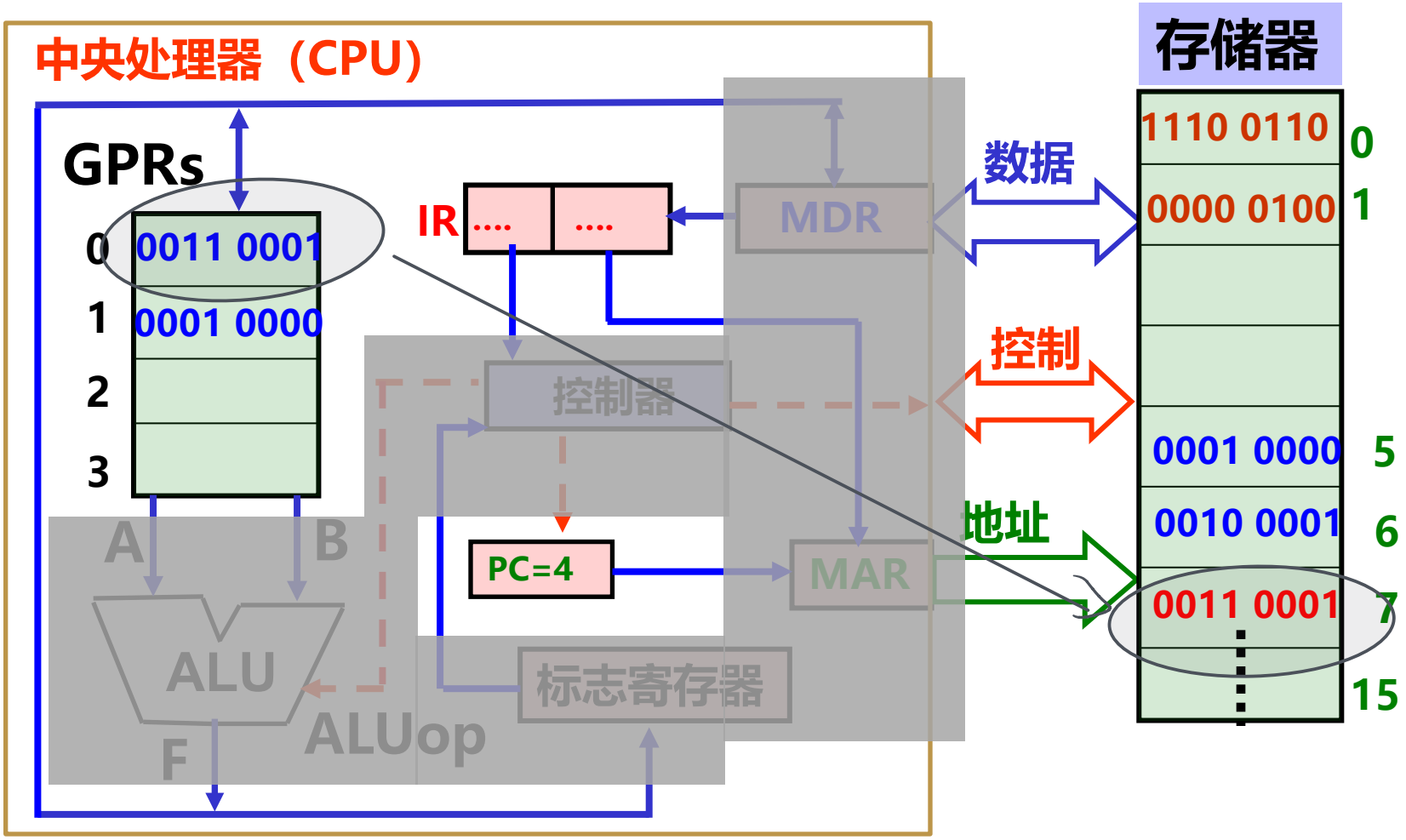
执行过程示意PC=2取数y



执行过程示意PC=3加法



执行过程示意PC=4存数



还是计算机系统的层次（再细化一点）

功能**转换**：上层是下层的**抽象**，下层是上层的**实现**

底层为上层提供支撑环境！

程序执行效果（结果和性能等）

不仅取决于

算法、程序编写

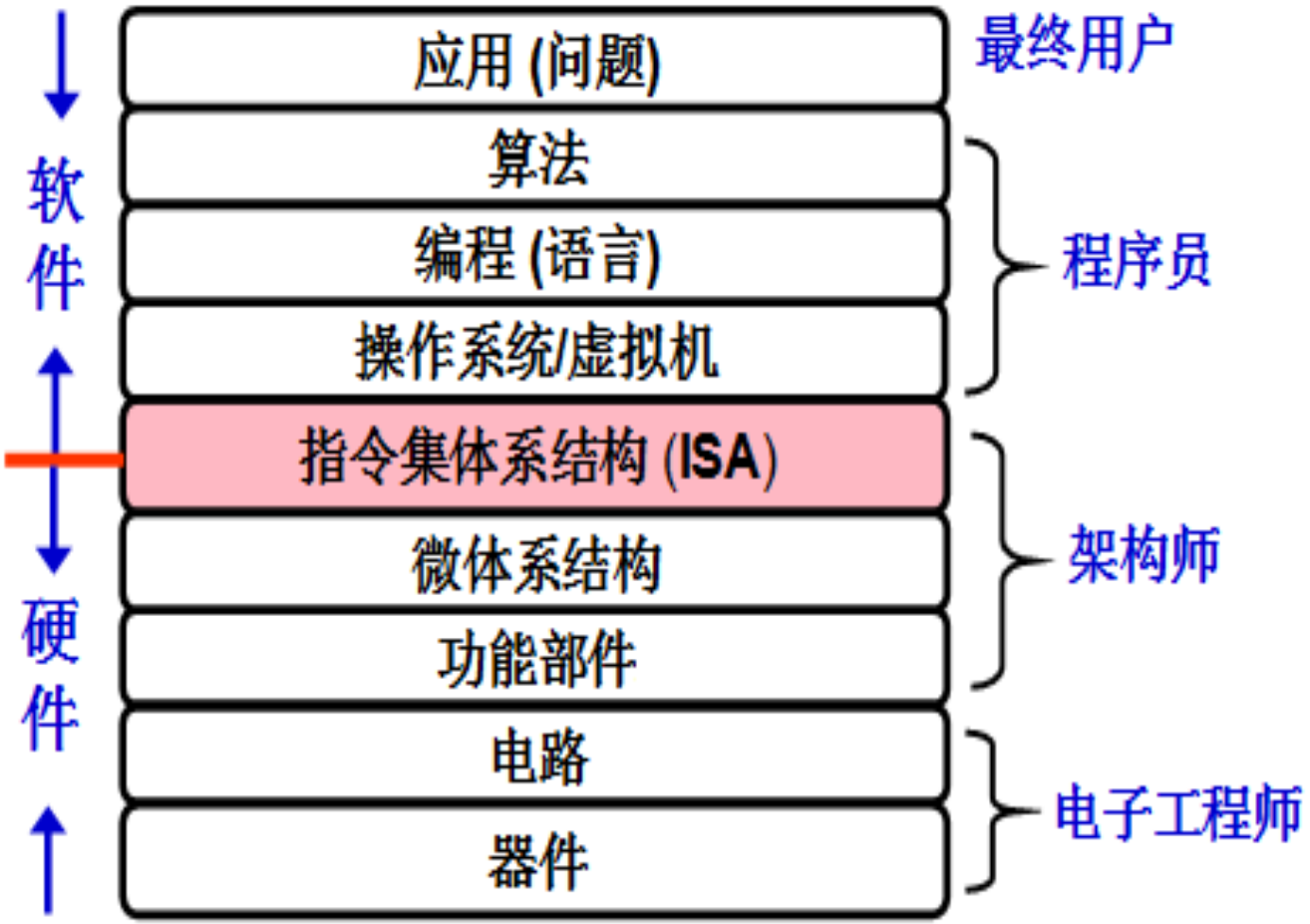
而且取决于

语言处理系统

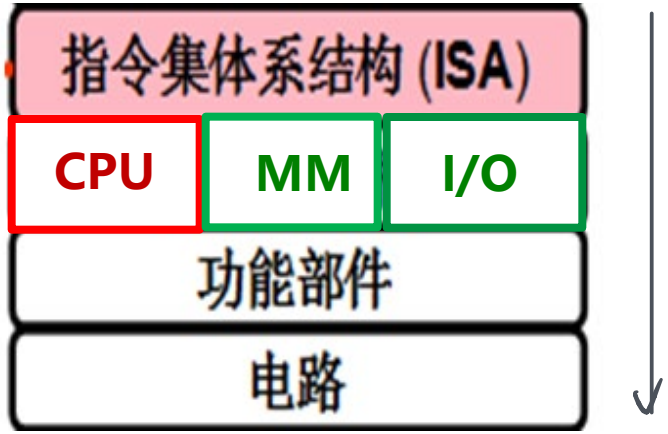
操作系统

ISA

微体系结构



本课程教学内容安排（再重复一下）



二进制编码

六次实验

- 数字逻辑电路
- 硬件描述语言（其它课）
- 运算功能部件
- 指令集体系结构
- 中央处理器（CPU）
- 存储器层次结构（ICS）
- 系统互连与输入/出（ICS）

第二讲：二进制编码表示

主　要　内　容

- 计算机的外部信息和内部数据
- 进位计数制
  - 十进制
  - 二进制
  - 八进制和十六进制
- 二进制数与其他计数制数之间的转换
  - R进制数与十进制数之间的转换
  - 二、十六进制数之间的转换
  - 十进制数→二进制数的简便方法

再来看看（ISA）

冯诺依曼计算机的

“存储程序”工作方式——程序（指令）、数据、执行

（计算机硬件能够理解并执行的只有二进制机器指令）

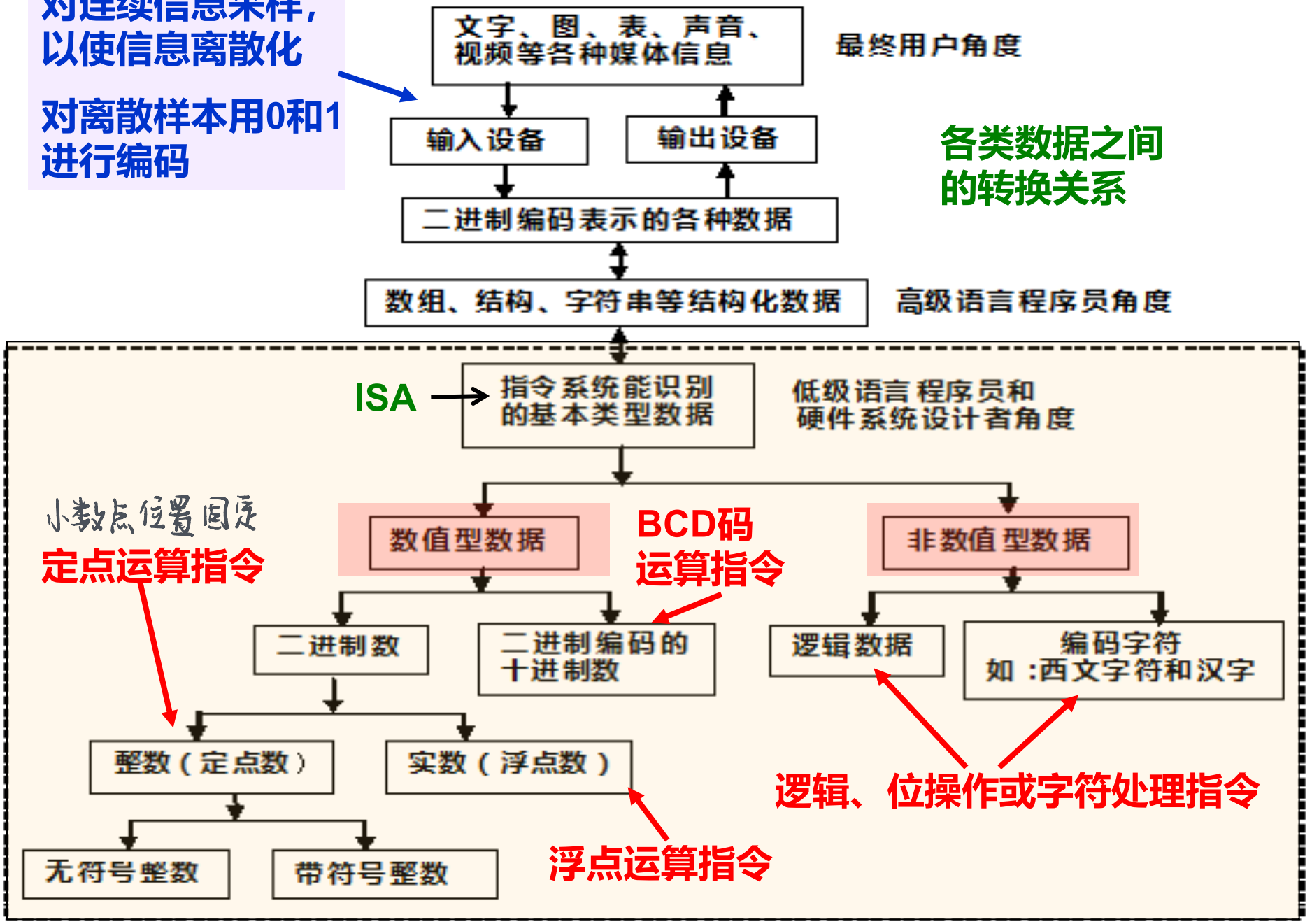
（计算机硬件能够处理的只有二进制数据）

- **ISA(指令集体系结构)规定了如何使用硬件**
  - 可执行的指令有哪些；每条指令有多长等等。
  - 指令可以接受的操作数的类型；
  - 操作数能存放到哪些地方：寄存器，内存；
  - 指令执行过程的控制方式，包括程序计数器如何自增等。

如果外部世界的一切信息都要用计算机来处理？

对连续信息采样，以使信息离散化

对离散样本用0和1进行编码



信息的二进制编码

- 机器级数据分两大类：
  - 数值数据：无符号整数、带符号整数、浮点数（实数）、十进制数
  - 非数值数据：逻辑数（包括位串）、西文字符和汉字
- 计算机内部所有信息都用二进制（即：0和1）进行编码
- 用二进制编码的原因：
  - 制造二个稳定态的物理器件容易
  - 二进制编码、计数、运算规则简单
  - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算
- 真值和机器数
  - 机器数：用0和1编码的计算机内部的0/1序列
  - 真值：机器数真正的值，如：现实中带正负号的数

Decimal / Binary（十 / 二进制数）

- 十进制数5836.47 表示为10的幂:

$$5 \times 10^3 + 8 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 7 \times 10^{-2}$$

- 二进制数11001表示为2的幂:（转换为十进制数）

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

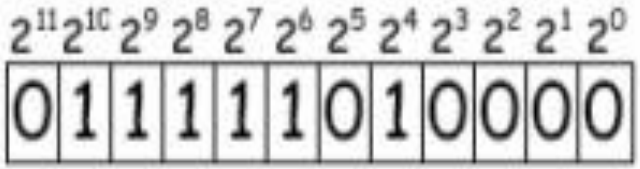
$$= 16 + 8 + 0 + 0 + 1 = 25$$

- 用一个下标表示数的基（ radix / base)

  或用后缀B-二进制（H-十六进制（前缀0x-）、O-八进制）

  $11001_2 = 25_{10}$、 11001B= 25

Octal / Hexadecimal (八 / 十六进制数)



$$v = \sum_{i=0}^{n-1} 2^i b_i$$

$2^3=8$ **$2^4=16$**

3720_8 **$7D0_{16}$**

Octal - base 8
000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

Hexadecimal - base 16	
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

一个8进制数字用3位二进制数字表示
一个16进制数字用4位二进制数字表示

计算机用二进制表示所有信息!

为什么要引入 8 / 16进制?

8 / 16进制是二进制的简便表示。便于阅读和书写!

它们之间对应简单，转换容易。

在机器内部用二进制，在屏幕或其他外部设备上表示时，转换为10进制或8/16进制数，可缩短长度

现在基本上都用16进制数表示机器数

数制之间的转换

(1) 二、八、十六进制数的相互转换

① 八进制数转换成二进制数

$(13.724)_8=(\ 001\ 011\ .\ 111\ \ 010\ \ 100\ )_2=(1011.1110101)_2$

② 十六进制数转换成二进制数

$(2B.5E)_{16}=(00101011\ .\ 01011110)_2=(101011.0101111)_2$

③ 二进制数转换成八进制数

$(0.10101)_2=(\ 000\ .\ 101\ \ 010\ )_2=(\ 0.52)_8$

④ 二进制数转换成十六进制数

$(11001.11)_2=(\ 0001\ \ 1001\ .\ 1100\ )_2=(\ 19.C\ )_{16}$

二进制数转换为8（16）进制数之前，
注意补足二进制位数为3（4）的整数倍
并保持数值不变（整数和小数部分各自单独考虑）

数制之间的转换

(2) R进制数 => 十进制数

按“权”展开

例1: $(10101.01)_2=1x2^4+1x\ 2^2+1x2^0+1x2^{-2}=(21.25)_{10}$

例2: $(307.6)_8=3x8^2+7x8^0+6x8^{-1}=(199.75)_{10}$

例1: $(3A.\ 1)_{16}=3x16^1+10x16^0+1x16^{-1}=(58.0625)_{10}$

(3) 十进制数 => R进制数

整数部分和小数部分分别转换

① 整数(integral part)----“除基取余，上右下左”

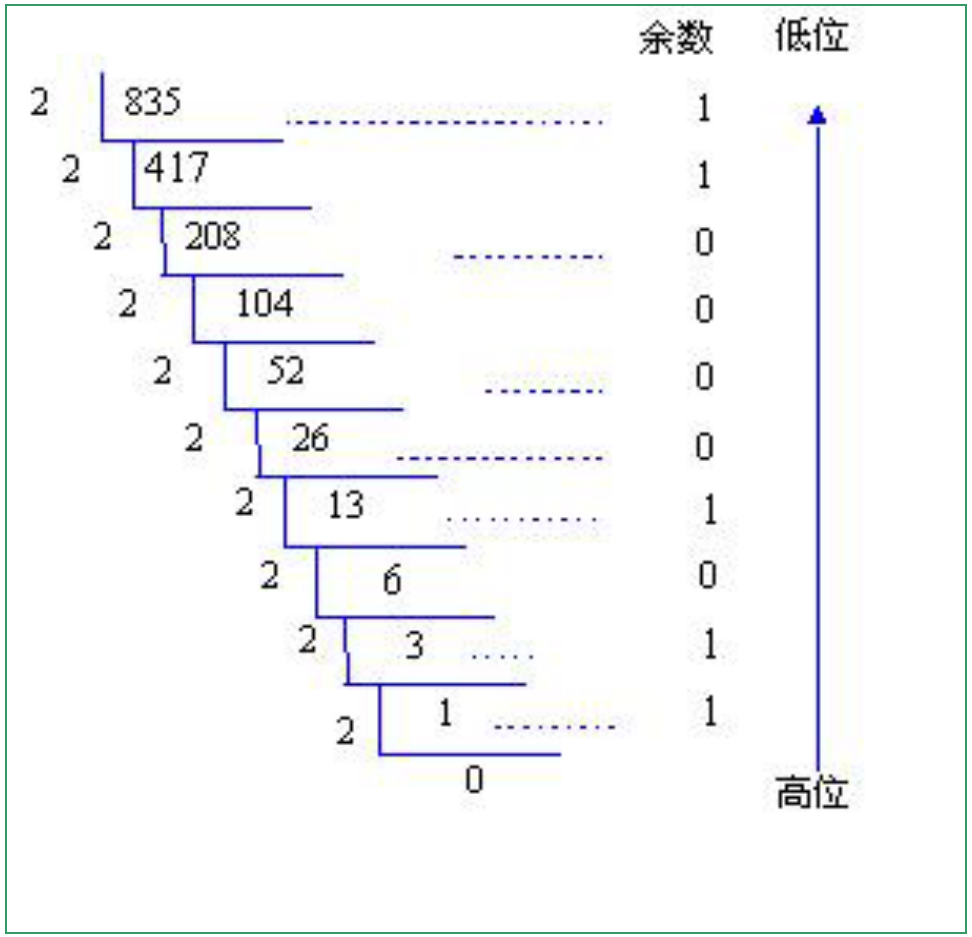
② 小数(fractional part)----“乘基取整，上左下右”

理论上的做法

10进制→2进制

例1: $(835.6785)_{10}=(1101000011.1011)_2$

整数----“除基取余，上右下左”　　小数----“乘基取整，上左下右”



0.6875×2=1.375　整数部分=1　(高位)

0.375×2=0.75　整数部分=0　↓

0.75×2=1.5　整数部分=1　↓

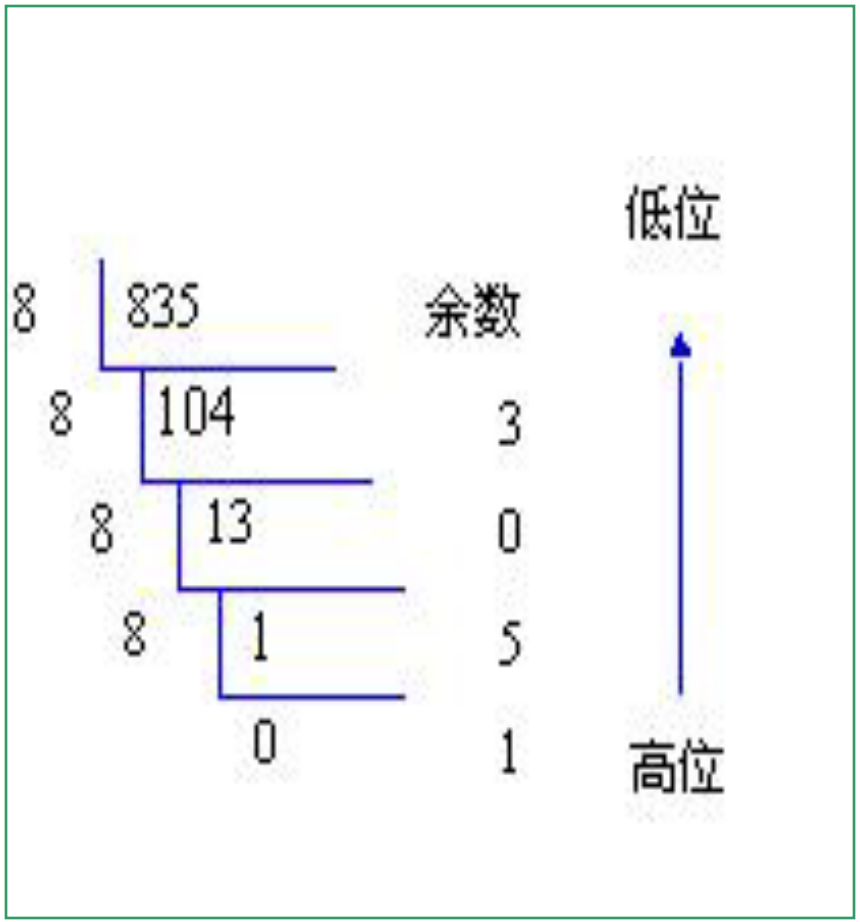
0.5×2=1.0　整数部分=1　(低位)

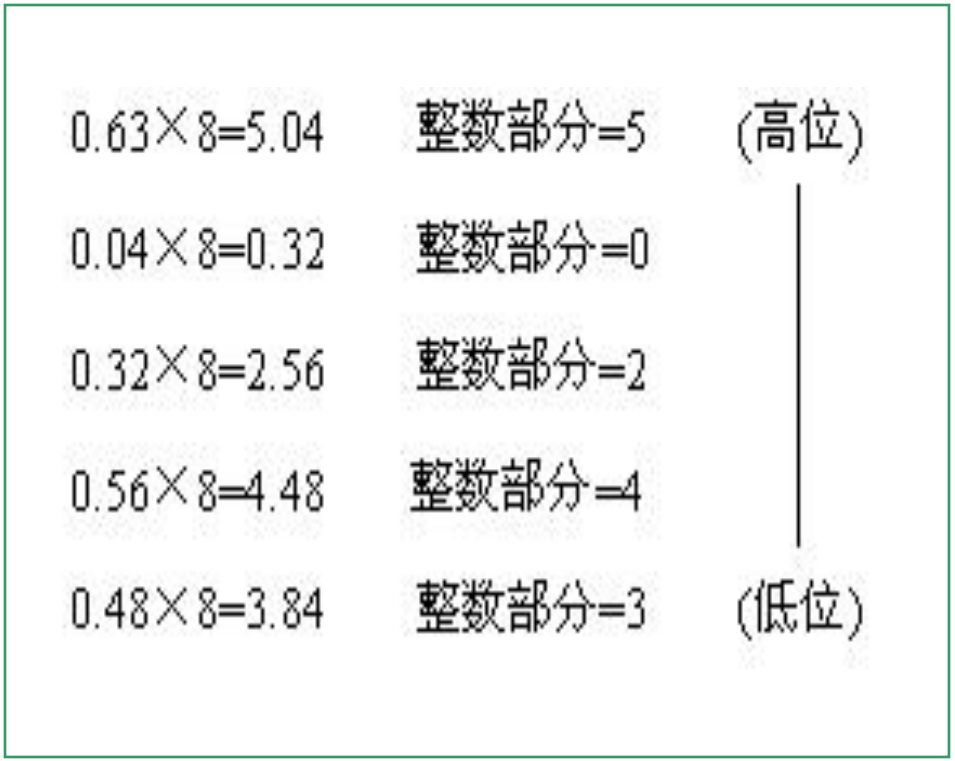
10进制→8进制

例2: $(835.63)_{10}=(1503.50243\cdots)_8$

小数----“乘基取整，上左下右”

有可能乘积的小数部分总得不到0，此时得到一个近似值。





数制之间的转换（简便方法）

(3) 十进制数 => R进制数

实际按简便方法先转换为二进制数，再按需转换为8/16进制数

整数：2、4、8、16、32、64、128、256、512、1024、2048、4096、8192、16384、32768、65536

小数：0.5、0.25、0.125、0.0625、0.03125、……

例：4123.25 = 4096+16+8+2+1+0.25

=1 0000 0001 1011.01B

=$(101B.4)_{16}$

4023 =(4096-1)-64-8

=1111 1111 1111B-100 0000B-1000B

=1111 1011 0111B

= FB7H=$(FB7)_{16}$

第三讲：数值数据的编码表示

主　要　内　容

- 定点数的表示
  - 定点数的二进制编码
    - 原码、补码、移码表示
  - 定点整数的表示
    - 无符号整数、带符号整数
- 浮点数的表示
  - 浮点数格式和表示范围
  - IEEE754浮点数标准
    - 单精度浮点数、双精度浮点数
    - 特殊数的表示形式
- 十进制数的二进制编码（BCD码）

数值数据的表示

- 数值数据表示的三要素
  - 进位计数制
  - 定、浮点表示
  - 如何用二进制编码

  即：要确定一个数值数据的值必须先确定这三个要素。

  例如， 1011001这个数的值是多少? 答案是：不知道！

- 确定使用二进制
- 定/浮点表示（解决小数点问题）

  定点数编码是最基础的工作
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- 定点数的编码（解决正负号问题）
  - 原码、补码、反码、移码 （反码很少用）

Sign and Magnitude （原码的表示）

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

1位符号位
+
3位数值位

Decimal	Binary
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

- 容易理解，　但是：
  - ✓ 0 的表示不唯一，故不利于程序员编程
  - ✓需额外对符号位进行处理，故不利于硬件设计
  - ✓加、减运算方式不统一
  - ✓特别当 a<b时，实现 a-b比较困难

原码表示整数或小数（默认1位符号位）

CPU必需知道定点整数/小数

定点整数（8位原码）

0001 1000B　11000B

1001 1000B　-11000B

定点小数（8位原码）

0001 1000B　0.0011B

1001 1000B　-0.0011B

定点整数：小数点固定在数值位最右，无需显式表达

定点小数：小数点固定在数值位最左，无需显式表达

从 50年代开始，整数都采用补码来表示，
但浮点数的尾数用原码定点小数表示

补码特性 - 模运算（modular运算）

重要概念：在一个模运算系统中，一个数与它除以“模”后的余数等价。

时钟是一种模12系统（0 ≡ 12 ≡ 24 ≡ 36， 3 ≡ 15 ， 6 ≡ 18 ）

假定钟表时针指向10点，要将它拨向6点， 则有两种拨法：

① 倒拨4格：10- 4 = 6

② 顺拨8格：10+8 = 18 ≡ 6 (mod 12)

模12系统中： 10- 4 ≡ 10+8 (mod 12)

- 4 ≡ 8 (mod 12)

则，称8是- 4对模12的补码 （即：- 4的模12补码等于8）。

同样有 -3 ≡ 9 （mod 12）

-5 ≡ 7 （mod 12）等

（正数的补码就是它自己）

结论1： 一个负数的补码等于模减该负数的绝对值。

结论2： 对于某一确定的模，某数减去小于模的另一数(4)，总可以用该数加上另一数的相反数(-4)的补码来代替。

补码（modular运算）：+ 和– 的统一

模运算系统例子：利用负数的补码

例1：“钟表”模运算系统

假定时针只能顺拨，从10点倒拨5格后是几点？

$10-5 \equiv 10+(12-5) = 10+7 \equiv 5 \pmod{12}$

例2：“4位十进制数” 模运算系统

假定算盘只有四档，且只能做加法，则在算盘上计算

9028-1713等于多少？

$9028-1713 \equiv 9028+(10^4-1713)$

$=9028+8287$

$=17315$

$\equiv 7315 \pmod{10^4}$

（超过模10000的部分被丢弃）

只有低4位留在算盘上。

计算机中的运算器是模运算系统（有限位数）

暂时无需区分符号位和数值位

8位二进制加法器模运算系统

模是多少？ 2^8

计算0111 1111 - 0100 0000 = ？

0111 1111 + (- 0100 0000) = 0111 1111 + (2^8- 0100 0000)

=0111 1111 + 1100 0000 = 1 0011 1111 (mod 2^8)

= 0011 1111

只留8位余数，1被丢弃

week 1

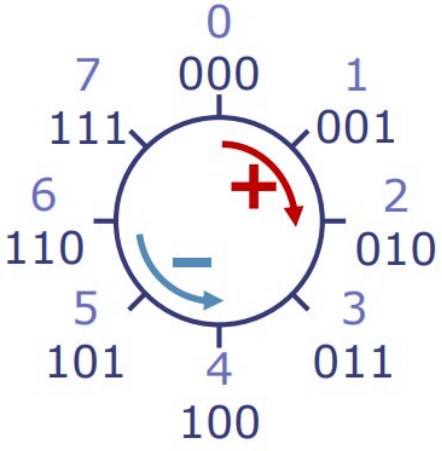
结论： 一个负数的补码等于对应正数（该负数的绝对值）的“各位取反、末位加1”

也就是说：并不会真的用减法去计算 (2^8- 0100 0000)

运算器是模运算系统（负数去哪了？）

注意：计算机中运算器只有有限位。假定为n位，则运算结果只能保留低n位，其模为2^n。

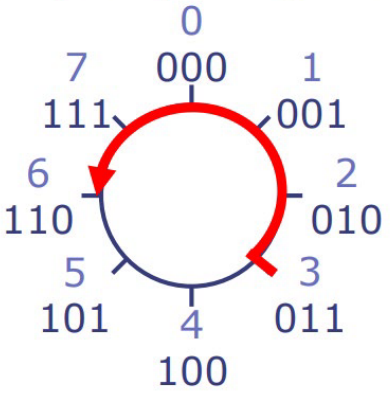
模为2^3



-2 ≡ 6（mod 8）

结果到底是-2 还是 6呢?

Example: (3 – 5) mod 2^3 ?



0~7

用三位二进制模运算，计算3-2

3: 011

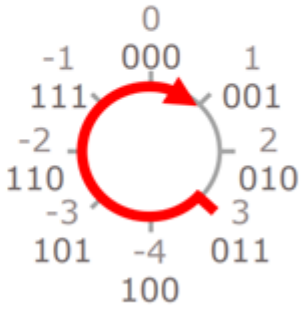
2: 010

-2: 110

$$\begin{array}{r} 011 \\ +\ 110 \\ \hline 1001 \end{array}$$

保留三位

这个1意味着什么?



跨越了0位置

0123
-4
-3
-2
-1

补码的完整定义

此时区分符号位和数值位

4位中最高位即为符号位
1代表负数
0代表正数

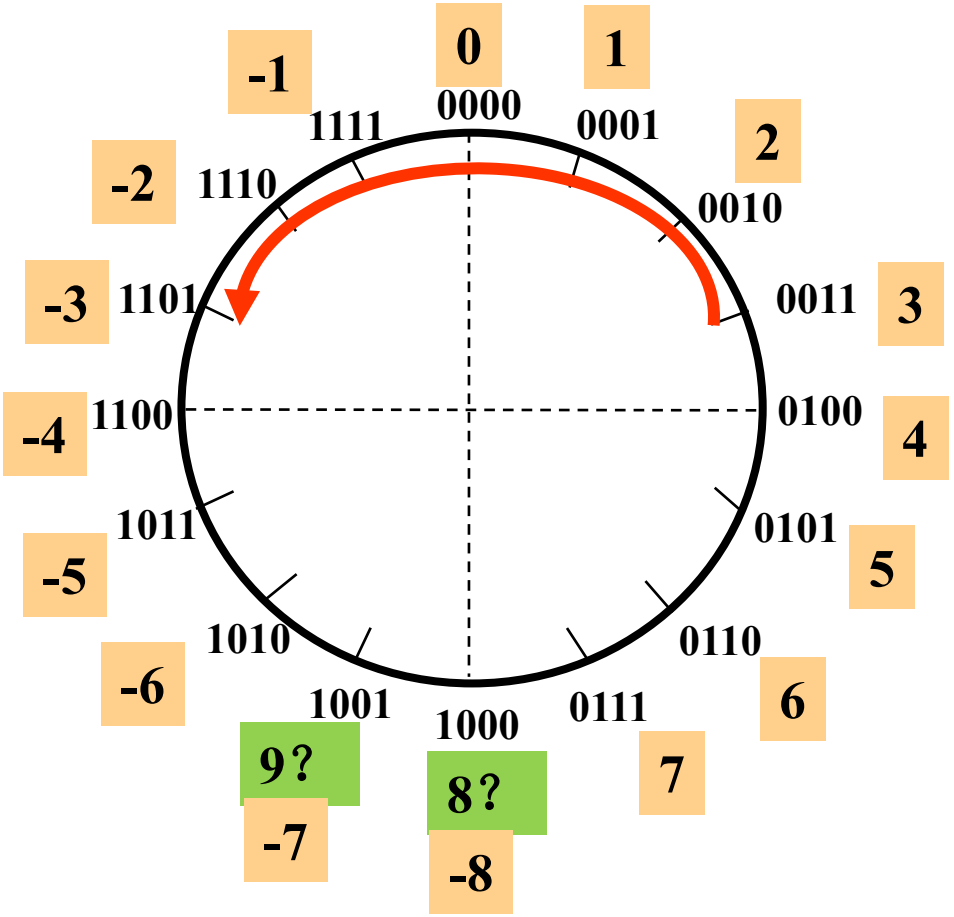
补码的定义 假定补码（机器数）有n位，则：

定点整数：$[X]_{补}= 2^n + X \quad (-2^{n-1} \le X < 2^{n-1}, \ mod\ 2^n)$

注：计算机中并不使用补码表示定点小数，略

当n=4时，共有16个机器数：
0000 ~ 1111，可看成是模为2^4的钟表系统。

真值的范围为 -8 ~ +7



3: 0011
6: 0110
-6: 1010

$3-6 = 3 + [-6]_{补}$

$$\begin{array}{r} 0011 \\ +\ 1010 \\ \hline 1101 \end{array}$$

求特殊数的补码

假定机器数有n位，且只有1位是符号位

——也就是一个补码包括1位符号位和n-1位数值位

$[X]_{补} = 2^n + X \quad (-2^{n-1} \le X < 2^{n-1}, \mod 2^n)$

① $[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0） （mod 2^n）

② $[-1]_{补} = 2^n - 0\ldots01 = 11\ldots1$（n个1） （mod 2^n）

③ $[+0]_{补} = [-0]_{补} = 00\ldots0$（n个0）

求一般数的补码——通用方法，简便转换

例: 设机器数有8位，求123和-123的补码表示。

如何快速得到123的二进制表示？

解: 123 = 127- 4 = 01111111B - 100B = 01111011B

-123= - 01111011B

$[+01111011]_{补}= 2^8 + 01111011 = 100000000 + 01111011$

$= 01111011 \pmod{2^8}$，即 7BH。

$[-01111011]_{补}= 2^8 – 01111011 = 10000\ 0000 – 0111\ 1011$

$= 1111\ 1111 – 0111\ 1011 +1$

$= 1000\ 0100 +1$

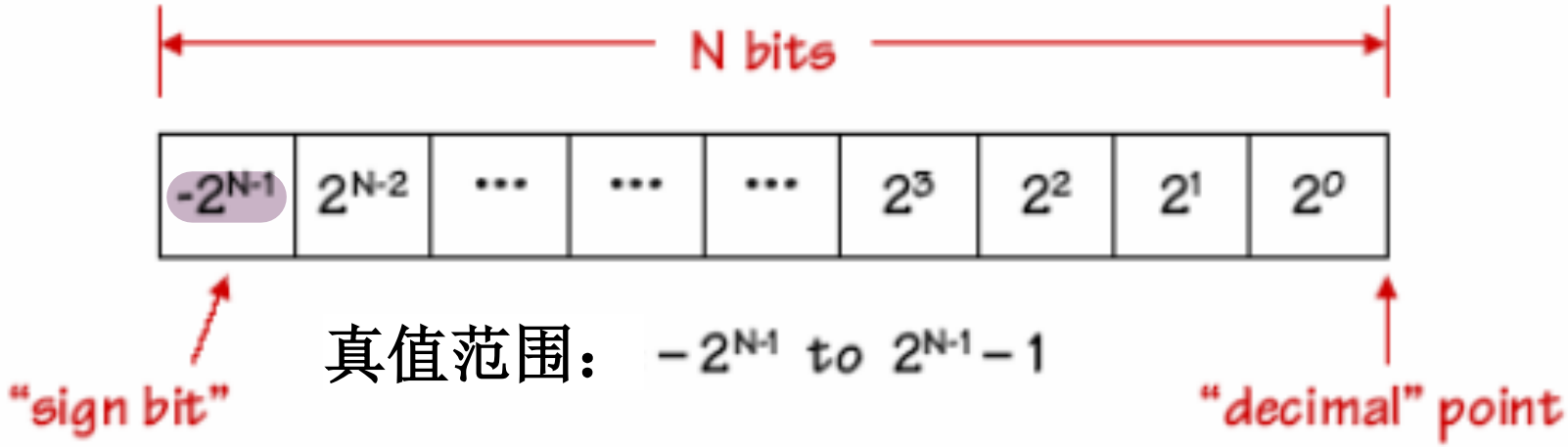
$= 1000\ 0101$，即 85H。

1000 0100

各位取反，末位加1

* 求补码的真值——通用方法，简便转换

根据补码各位上的“权”求补码的值



8-bit 2's complement example:

$$11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$$

符号为0，则为正数，数值部分相同

符号为1，则为负数，数值各位取反，末位加1

例如：补码“11010110”的真值为：-0101010=-(32+8+2)=-42

补码的总结，变形补码

- 正数：符号位（sign bit）为0，数值部分不变
- 负数：符号位为1，数值部分“各位取反，末位加1”

变形（模4）补码：双符号，用于存放可溢出的中间结果。

Decimal	补码	变形补码	Decimal	Bitwise Inverse	补码	变形补码
0	0000	00000	-0	1111	0000	00000
1	0001	00001	-1	1110	1111	11111
2	0010	00010	-2	1101	1110	11110
3	0011	00011	-3	1100	1101	11101
4	0100	00100	-4	1011	1100	11100
5	0101	00101	-5	1010	1011	11011
6	0110	00110	-6	1001	1010	11010
7	0111	00111	-7	1000	1001	11001
8	1000	01000	-8	0111	1000	11000

+0和-0表示唯一

值太大，用4位补码无法表示，故“溢出”！但用变形补码可保留符号位和最高数值位。

Excess (biased) notion- 移码表示

- 什么是“excess (biased) notation-移码表示”？

  将每一个数值加上一个偏置常数（ Excess / bias）

- 一般来说，当机器数（编码）位数为n时，bias取 2^{n-1}

  假设n=4: $E_{biased} = E + 2^3$ (bias= 2^3 =1000B)

  $[X]_{补} = 2^4 + X$

  -8 (+8) ~ 0000B

  -7 (+8) ~ 0001B

  …

  0 (+8) ~ 1000B

  …

  +7 (+8) ~ 1111B

  0的移码表示唯一

  此时移码和补码仅第一位正好相反

  移码里面不存在符号位的概念

移码主要用来表示浮点数阶码——为了简化浮点数的编码和计算

归纳1： Signed integer（带符号整数）

- 计算机必须能处理正数(positive) 和负数(negative)，包含符号位
- 有三种定点编码方式（注意！不包括移码！）
  - Signed magnitude （原码）

    现用来表示浮点（实）数的尾数
  - One’s complement （反码）

    现已不用于表示数值数据
  - Two’s complement （补码）

    50年代以来，所有计算机都用补码来表示定点整数
- 为什么用补码表示带符号整数？
  - 补码运算系统是模运算系统，加、减运算统一
  - 数0的表示唯一，方便使用
  - 比原码多表示一个最小负数（提示：原码+0和-0不一样）
  - 与移码相比，其符号位和真值的符号对应关系更直接

归纳2：Unsigned integer(无符号整数)

- 一般在全部是正数运算且不出现负值结果的场合下，可使用无符号数表示。例如，地址运算，编号表示，等等
- 无符号数的编码中没有符号位，也无需使用原码补码移码
- 能表示的最大值 > 位数相同的带符号整数的最大值
  - 例如，8位无符号整数最大是255（1111 1111） 2^n-1

    而8位带符号整数最大为127（0111 1111）$2^{n-1}-1$
- 总是整数，所以很多时候就简称为“无符号数”

不同类型数值的相互转换

例：在32位机器上输出si, usi, i, ui的十进制（真值）和十六进制值（机器数）是什么？

16位带符号数 short si = -32768;

16位无符号数 unsigned short usi = si;

32位带符号数 int i = si;

32位无符号数 unsingned ui = usi ;

提示：

$32768=2^{15}$

=1000 0000 0000 0000B

Short表数范围是

-2^{15}~$2^{15}-1$

	真值	16进制机器数
si	-32768	80 00
usi	32768	80 00
i	-32768	FF FF 80 00
ui	32768	00 00 80 00

机器数不变，但真值变了

16位带符号数 →32位带符号数

16位无符号数 →32位无符号数

现象：

带符号整数：符号扩展

无符号整数：0扩展

注意：这里每一次赋值都是CPU执行指令而完成的“运算”，所以总能做正确的事

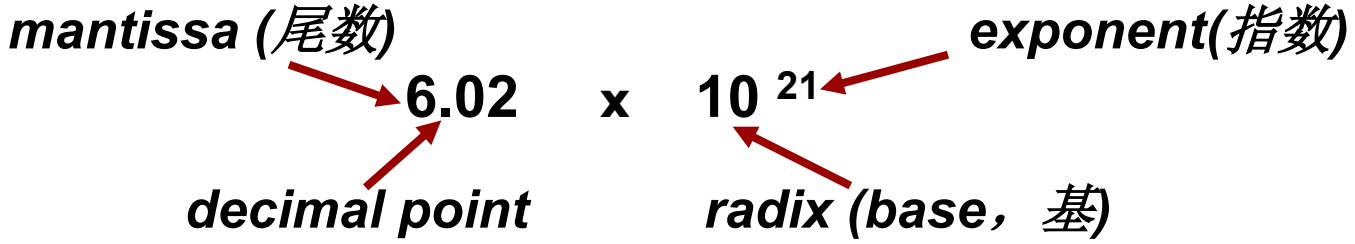
原码
补码
移码

定点整数 { 无符号整数 二进制
带符号整数 补码

小数？实数？

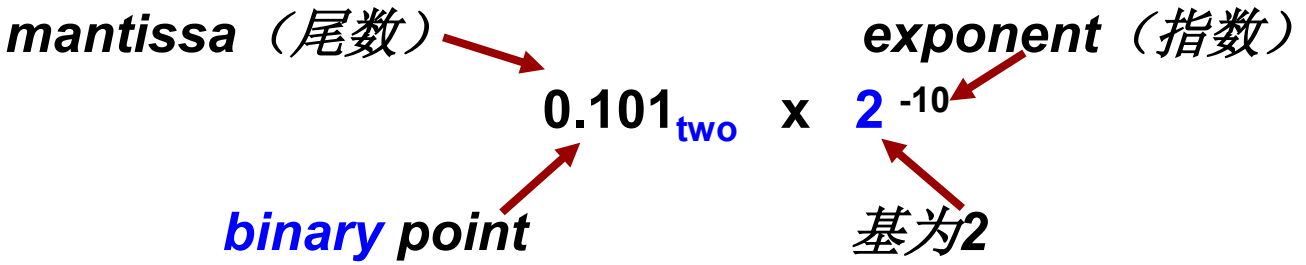
科学计数法(Scientific Notation)与浮点数

十进制的例子:



- Normalized form（规格化形式）: 尾数的小数点前只有一位非0数
- 同一个数有多种表示形式。例：对于数 1/1,000,000,000
  - Normalized (唯一的规格化形式): 1.0×10^{-9}
  - Unnormalized（非规格化形式不唯一）: 0.1×10^{-8}, 10.0×10^{-10}

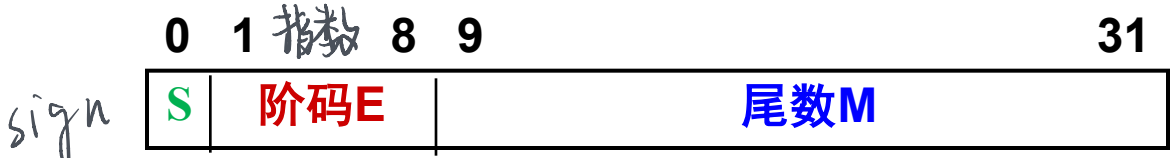
二进制的科学计数法表示：



只要对尾数和指数分别编码，就可表示一个浮点数（即：实数）

浮点数(Floating Point)的第1个例子

例：画出下述格式的规格化数的表示范围。规定如下（注意三种颜色）：



$+/-0.1xxxxx \times 2^{E-128}$

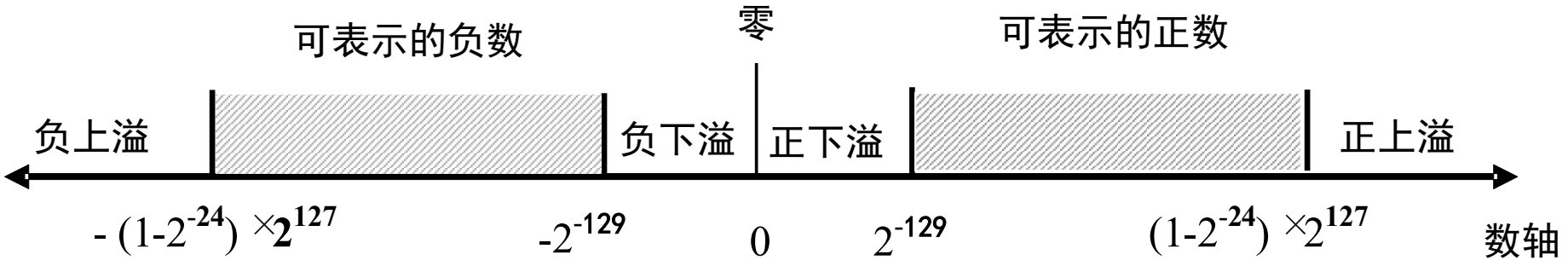
是一个32位浮点数，其中：第0位数符S；第1～8位为8位移码表示阶码E（偏置常数为128）；第9～31位为24位二进制原码小数表示的尾数M。

规格化要求尾数必须是0. 1xxxx形式（小数点后第一位总是1），且约定第一位默认的“1”不明显表示出来。最终就用23个数位表示24位尾数。

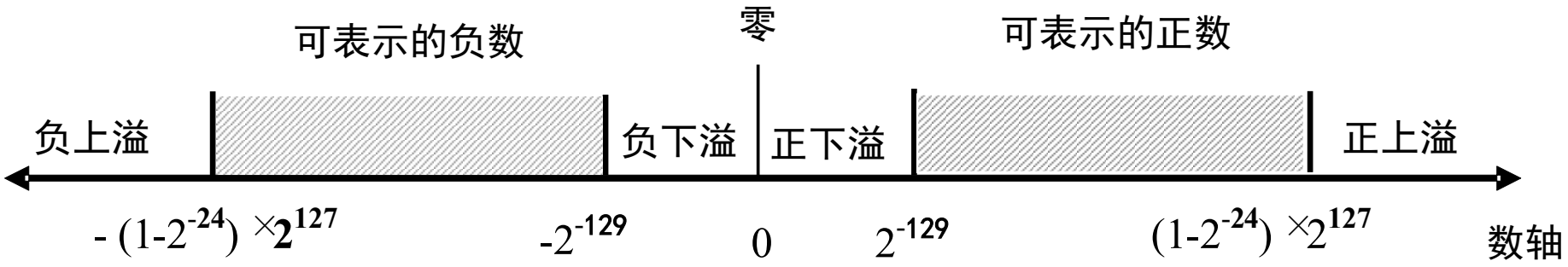
最大正数：$0.11\ldots1 \times 2^{11\ldots1-128}$　对应真值$(1-2^{-24}) \times 2^{127}$

最小正数：$0.10\ldots0 \times 2^{00\ldots0-128}$　对应真值$(1/2) \times 2^{-128}$

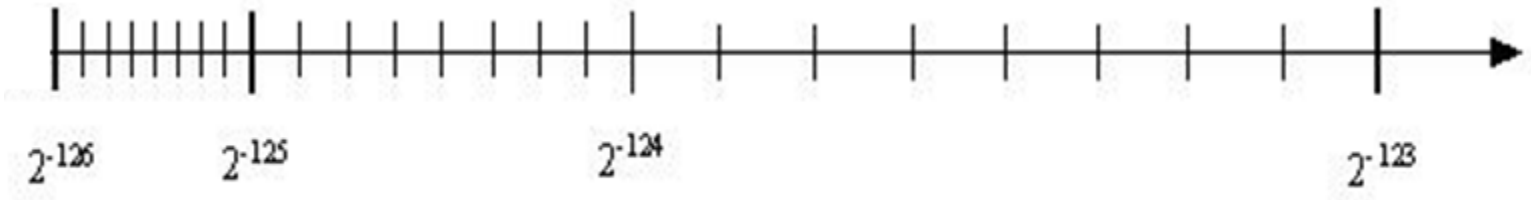
因为原码是对称的，所以其表示范围关于原点对称。



浮点数(Floating Point)的第1个例子（续）



- 机器0：尾数为0 或 落在下溢区中的数
- 浮点数范围比定点数大，但数的个数没变多，故数之间更稀疏且不均匀，也不连续（比如第二小的正数是0.10……1x$2^{00\cdots0-128}$ ）



浮点数还能怎么表示？

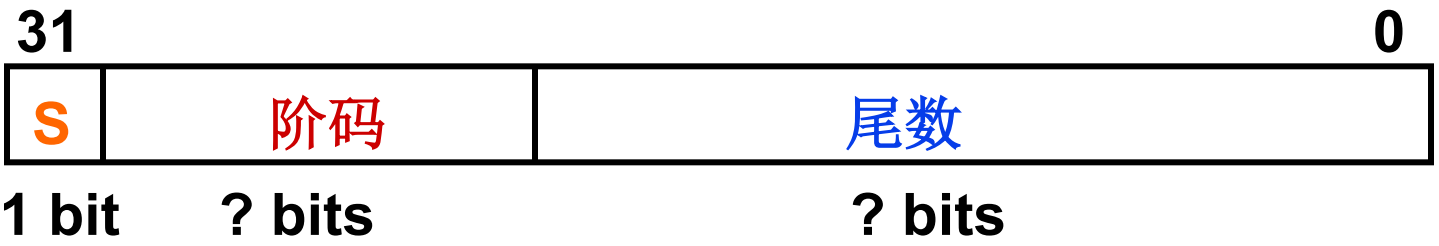
规定：小数点前总是“1”，故可隐含表示

注意：和前面例子的规定不太一样，这里更合理（多表示了一位尾数的有效数字）

- Normal format（规格化数形式）：

  +/-1.xxxxxxxxxx × $2^{\text{Exponent}-?}$

- 32-bit 规格化数：



S 是符号位（Sign）

阶码用移码来表示（Exponent）

尾数表示 xxxxxxxxxx，不含尾数部分小数点前面的1

- 早期的计算机，各自定义自己的浮点数格式

问题：浮点数表示不统一会带来什么问题？

“Father” of the IEEE 754 standard

直到80年代初，各个机器内部的浮点数表示格式还没有统一
因而相互不兼容，机器之间传送数据时，带来麻烦

1970年代后期，IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE 754的制定

现在所有计算机都采用IEEE 754来表示浮点数

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.

1989
ACM Turing
Award Winner!

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

Prof. William Kahan

IEEE 754 浮点数标准

规格化数：+/-1.xxxxxxxxxx$_{two}$ x $2^{Exponent-127}$

Single Precision单精度：32位 (双精度Double Precision 类似 64位)

S	Exponent	Significand
1 bit	8 bits	23 bits

- S (符号位)： 1 表示negative ; 0表示 positive
- Exponent（阶码-机器数）：对应的是“指数”-真值
  - SP规格化数阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
  - 是移码，偏置常数为127 (single)

全0和全1用来表示特殊值

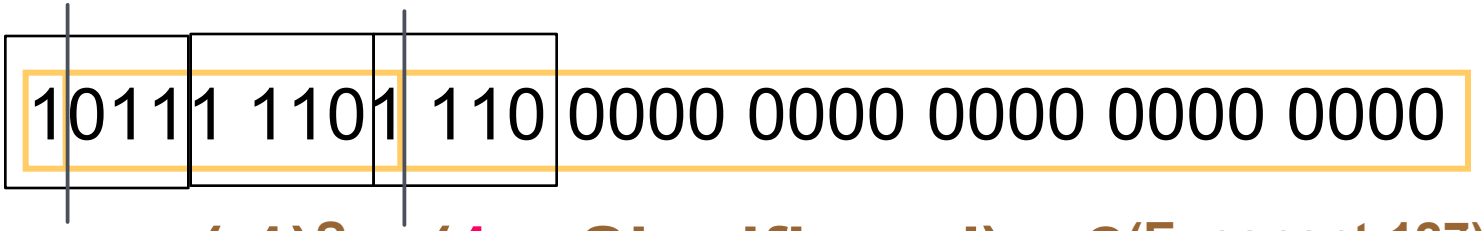
0 ∞

- Significand（尾数）：是原码
  - 规格化尾数最高位总是1，所以隐含表示，省1位
  - 1 + 23 bits （ single)

SP: $(-1)^S$ x (1 + Significand) x $2^{(Exponent-127)}$

例：二进制浮点数转换为十进制真值

BEE00000H 是一个 IEEE 754 单精度浮点数的机器数表示



$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

- **Sign**: 1 => negative
- **Exponent**:
  - $0111\ 1101_{two} = 125_{ten}$
  - 按偏置常数计算真实指数: 125 - 127 = -2
- **Significand**:

  $1 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 0\times2^{-5} + \ldots$

  $=1+2^{-1}+2^{-2} = 1+0.5+0.25 = 1.75$
- **真值**: $-1.75_{ten}\times2^{-2} = -0.4375$

例：十进制真值转换为二进制浮点数

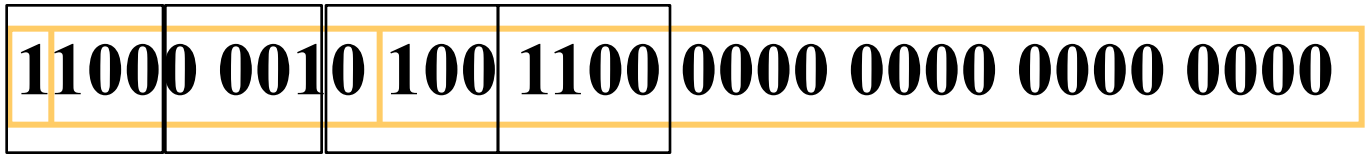
-12.75

1. 初始真值: -12.75
2. 整数部分转换:

   $12 = 8 + 4 = 1100_2$
3. 小数部分转换:

   $.75 = .5 + .25 = .11_2$
4. 规格化:

   $1100.11 = 1.10011 \times 2^3$
5. 阶码计算: $127 + 3 = 128 + 2 = 1000\ 0010_2$



最终浮点数的16进制表示为：C14C0000H

Normalized numbers（规格化数）

前面的定义都是针对规格化数（normalized form）

How about other patterns?

Exponent	Significand	Object
1-254	anything implicit leading 1	Norms
0	0	?
0	nonzero	?
255	0	?
255	nonzero	?

0000 0000	非零或全零
1111 1111	非零或全零

怎么表示0？

exponent: 全0

significand: 全0

sign? 1和0都行

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

怎么表示+∞/-∞ ？

∞ ： infinity

浮点数

In FP, 除数为0的结果是 +/- ∞, 不是溢出异常.（整数除0为异常）

为什么要这样处理?

- 可以利用+∞/-∞作比较。 例如：X/0>Y可作为有效比较

+∞/-∞的表示：

- Exponent : 全1(11111111B = 255)
- Significand: 全0

+∞ : 0 11111111 00000000000000000000000

-∞ : 1 11111111 00000000000000000000000

可能的运算：

5.0 / 0 = +∞, -5.0 / 0 = -∞

5+(+∞) = +∞, (+∞)+(+∞) = +∞

5 - (+∞) = -∞, (-∞) - (+∞) = -∞ etc

怎么表示非数（“Not a Number”）？

Sqrt (- 4.0) = ？　　0/0 = ？

- **Called Not a Number (NaN) - “非数”**

NaN的表示：

Exponent = 255

Significand: 任意非0值

NaNs 对编程调试有帮助

可能的运算：

sqrt (-4.0) = NaN　　0/0 = NaN

op (NaN,x) = NaN　　+∞+(-∞) = NaN

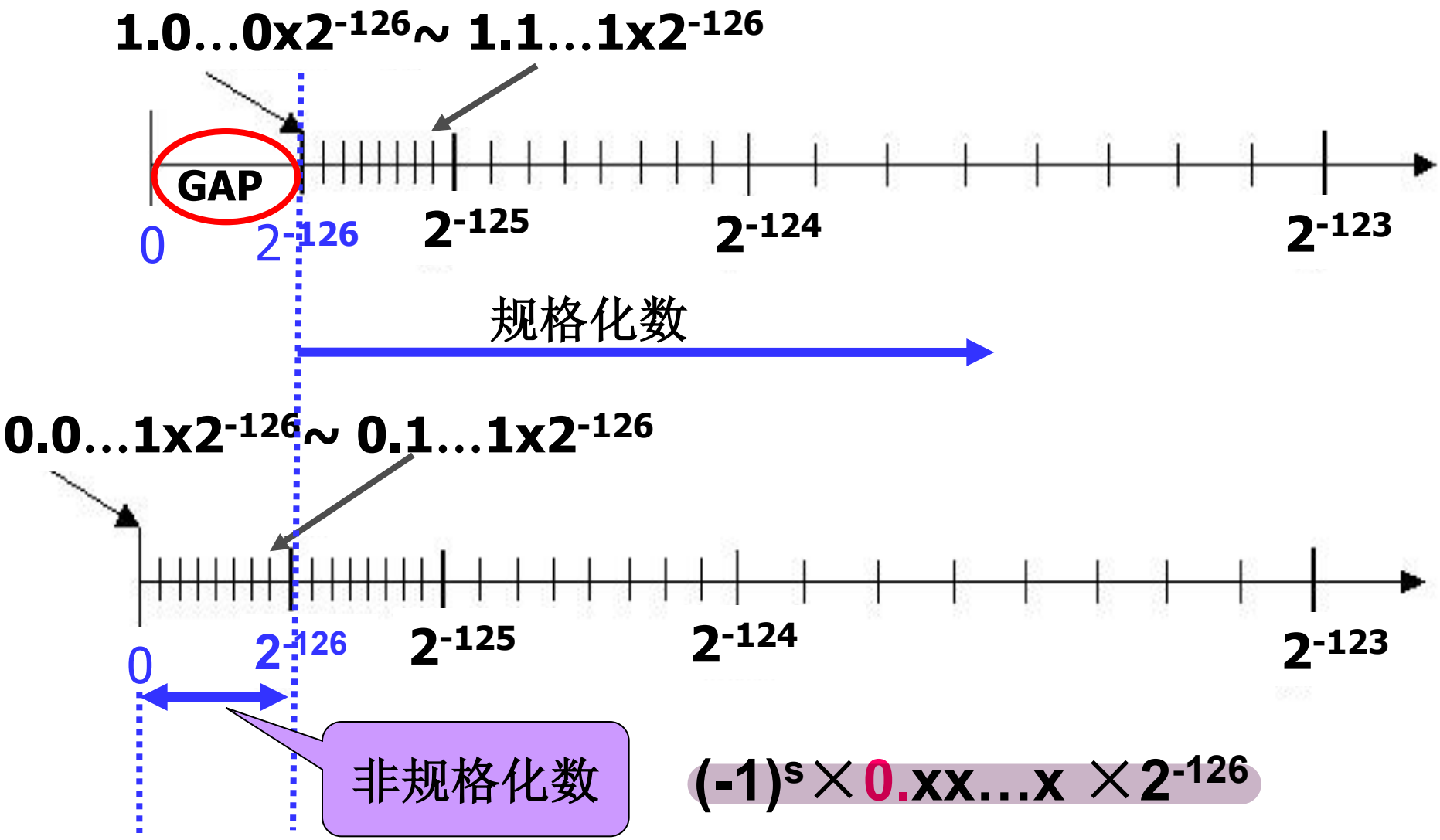
+∞- (+∞) = NaN　　∞/∞ = NaN

etc.

怎么表示Denorms(非规格化数)

Exponent	Significand	Object
1-254	anything implicit leading 1	规格化数
0	0	+/-0
0	nonzero	非规格化数
255	0	+/- ∞
255	nonzero	NaN非数

非规格化数的表示



$(-1)^s \times 0.xx\ldots x \times 2^{-126}$

Exponent： 全0

Significand: 任意非0值

IEEE 754表示的一些问题

- 表数范围?

  单精度可表示最大正数: +1.11…1x 2^{127}　　约 +3.4 x 10^{38}

  双精度呢?　　约 +1.8 x 10^{308}

- 数据转换时可能发生的问题? i是32位补码，f是float，d是double

  i 和 (int) ((float) i))　　不一定相等（尾数23+1）（int是32位）

  i 和 (int) ((double) i))　　相等（尾数52+1）

  f 和 (float) ((int) f))　　不一定相等

  d 和 (double) ((int) d))　　不一定相等

- FP参与加法时的不同计算顺序可能带来的问题?

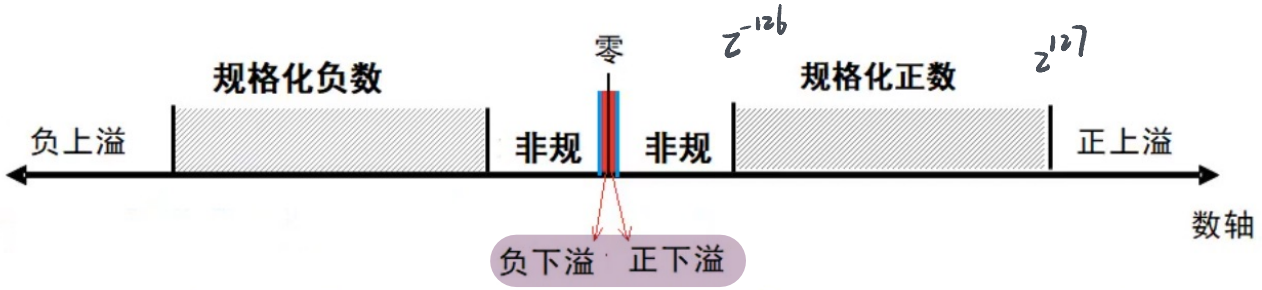
  x = – 1.5 x 10^{38},　y = 1.5 x 10^{38},　z = 1.0

  (x+y)+z = (–1.5x10^{38}+1.5x10^{38}) +1.0 = 1.0

  x+(y+z) = –1.5x10^{38}+ (1.5x10^{38}+1.0) = 0.0

最后归纳：浮点数的范围和精度

- 原码、补码的表数范围：
  - 和机器数中除符号位之外的数值位的位数有关
- **浮点数的范围**
  - **主要与阶码的范围有关**
- **浮点数的精度**
  - **与尾数的位数和如何规格化有关**



* 用BCD码表示十进制数

- 编码思想： 每个十进数位（0-9）至少有4位二进制表示。而4位二进制位可组合成16种状态，去掉10种状态后还有6种冗余状态。
- 编码方案（注：可以额外用专门的四位编码表示正负号，不过很少用）

1. 十进制有权码
   - 每个十进制数字的4个二进制位（称为基2码）都有确定的权。
     8421码是最常用的十进制有权码。也称自然BCD（NBCD）码。
2. 十进制无权码

   从0000－0101表示0－9

   - 每个十进制数位的4个基2码没有确定的权。
   - 用的较多的是余3码和格雷码。
   - 余3码：由8421码加上0011形成。当两个十进制数字之和是10时，和的二进制编码值正好是16，而且0和9，1和8，…，5和4的余3码互为反码（各二进制位都相反）。
   - 格雷码（Gray Code）：任意两个相邻的编码只有一位二进位不同。格雷码有多种编码形式。
3. 其他编码方案 （5中取2码、独热码等）

原码
补码
移码

定点整数 { 无符号整数 二进制
带符号整数 补码

浮点数(IEEE754)

{ 尾数（原码） 定点小数
阶码（移码）

十进制数Decimal ○ BCD（Binary coded Decimal）码

补充说明：补码可以表示小数，原码可以表示整数，浮点数也可以有各种编码方式。但考虑到兼容性、合理性等，统一采用了相对最优的编码方案。

- 四位二进制1001B（不含符号位）、十六进制9H ——对应真值+9
- 四位原码（含1位符号位）1001 ——对应真值-1
- 四位补码（含1位符号位）1001 ——对应真值-7
- 8421码 1001 ——对应真值+9

10在计算机中有几种可能的表示？
——原码，补码，无符号数，浮点数，BCD码

-10呢？
——原码，补码，浮点数，BCD码

回顾和补充（关于原码，默认1位符号位）

定点整数（8位原码）

0001 1000B　11000B

1001 1000B　-11000B

定点小数（8位原码）

0001 1000B　0.0011B

1001 1000B　-0.0011B

如果是浮点数尾数，那么一定是原码定点小数

其对应真值还要看规格化的要求

假设规格化要求是——小数点前有且只有一个1

0001 1000B　1.0011B

1001 1000B　-1.0011B

编译、汇编的过程将真值转换为机器数

执行指令时可判断所拿到的机器数属于哪种类型

第三讲小结

- 在机器内部编码后的数称为机器数，其值称为真值
- 定义数值数据有三个要素：进制、定点/浮点、编码
- 整数的表示
  - 无符号数：正整数，用来表示地址等；带符号整数：用补码表示
- 浮点数的表示
  - 符号；尾数：定点小数；指数（阶）：定点整数（基不用表示）
- 浮点数的范围
  - 正上溢、正下溢、负上溢、负下溢；与阶码的位数和基的大小有关
- 浮点数的精度：与尾数的位数和是否规格化有关
- 浮点数的表示（IEEE 754标准）：单精度SP（float）和双精度DP（double）
  - 规格化数(SP)：阶码1~254，尾数最高位隐含为1
  - “零”（阶为全0，尾为全0)
  - ∞ (阶为全1，尾为全0)
  - NaN (阶为全1，尾为非0)
  - 非规格化数 (阶为全0，尾为非0，隐藏位为0)
- 十进制数的二进制表示（BCD码）
  - 有权BCD码（8421码）、无权BCD码（余3码、格雷码等）

第四讲　非数值数据、数据的排列和存储

主　要　内　容

- ◆非数值数据的表示
  - •逻辑数据、西文字符、汉字
- ◆数据的宽度
- ◆数据的存储排列
  - •大端方式、小端方式

逻辑数据的编码表示

- **表示**
  - 用一位表示 。例如，真：1 / 假：0
  - N位二进制数可表示N个逻辑数据，或一个位串
- **运算**
  - 按位进行

$$\begin{array}{r} 0101 \\ 1001 \\ \hline 0001 \end{array} \qquad \begin{array}{r} 0101 \\ 1001 \\ \hline 1101 \end{array} \qquad \begin{array}{l} 0101 \\ 1010 \end{array} \qquad \begin{array}{l} 0101 \\ 0010 \end{array}$$

  - 如: 按位与 / 按位或 / 逻辑左移 / 逻辑右移 等
- **识别**
  - 逻辑数据和数值数据在形式上并无差别，也是一串0/1序列，使用时靠指令来识别。
- **位串**
  - 用来表示若干个状态位或控制位（OS中使用较多）

西文字符的编码表示

- **特点**
  - 是一种拼音文字，用有限几个字母可拼写出所有单词
  - 只对有限个字母和数学符号、标点符号等辅助字符编码
  - 所有字符总数不超过256个，使用7或8个二进位可表示
- **表示（常用编码为7位ASCII码）**
  - 十进制数字：0/1/2.../9
  - 英文字母：A/B/.../Z/a/b/.../z
  - 专用符号：+/-/%/*/&/.....
  - 控制字符（不可打印或显示）

了解对应的ASCII码！

- **操作**
  - 字符串操作，如:传送/比较　等

ASCII 码表

	$b_6b_5b_4$ =000	$b_6b_5b_4$ =001	$b_6b_5b_4$ =010	$b_6b_5b_4$ =011	$b_6b_5b_4$ =100	$b_6b_5b_4$ =101	$b_6b_5b_4$ =110	$b_6b_5b_4$ =111
$b_3b_2b_1b_0$=0000	NUL	DLE	SP	0	@	P	`	p
$b_3b_2b_1b_0$=0001	SOH	DC1	!	1	A	Q	a	q
$b_3b_2b_1b_0$=0010	STX	DC2	“	2	B	R	b	r
$b_3b_2b_1b_0$=0011	ETX	DC3	#	3	C	S	c	s
$b_3b_2b_1b_0$=0100	EOT	DC4	$	4	D	T	d	t
$b_3b_2b_1b_0$=0101	ENQ	NAK	%	5	E	U	e	u
$b_3b_2b_1b_0$=0110	ACK	SYN	&	6	F	V	f	v
$b_3b_2b_1b_0$=0111	BEL	ETB	‘	7	G	W	g	w
$b_3b_2b_1b_0$=1000	BS	CAN	(	8	H	X	h	x
$b_3b_2b_1b_0$=1001	HT	EM	)	9	I	Y	i	y
$b_3b_2b_1b_0$=1010	LF	SUB	*	:	J	Z	j	z
$b_3b_2b_1b_0$=1011	VT	ESC	+	;	K	[	k	{
$b_3b_2b_1b_0$=1100	FF	FS	,	<	L	\	l	|
$b_3b_2b_1b_0$=1101	CR	GS	-	=	M	]	m	}
$b_3b_2b_1b_0$=1110	SO	RS	.	>	N	^	n	~
$b_3b_2b_1b_0$=1111	SI	US	/	?	O	_	o	DEL

* 汉字及国际字符的编码表示

- **特点**
  - 汉字是表意文字，一个字就是一个方块图形。
  - 汉字数量巨大，总数超过6万字，给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入和输出等带来了一系列问题。
- **编码形式**
  - 有以下几种汉字代码：
  - 输入码：对汉字用键盘按键进行编码表示，用于输入 (较键盘组合)
  - 内码：用于在系统中进行存储、查找、传送等处理
  - 字模点阵码或轮廓描述：描述汉字字模的点阵或轮廓，用于输出

问题：西文字符有没有输入码？有没有内码？(ASCII)
有没有字模点阵或轮廓描述？

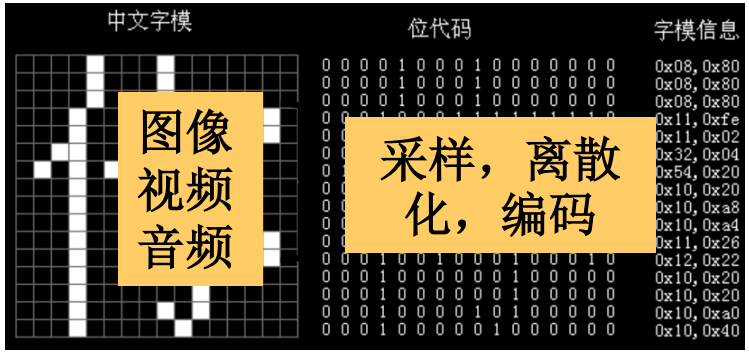
* 汉字内码、字模点阵码(轮廓描述)

- 至少需16位二进制才能表示一个汉字内码。为什么？ $2^{16}=65536$
  - 由汉字的总数决定！
- 可在GB2312国标码的基础上产生汉字内码
  - 为与ASCII码区别，将国标码的两个字节的第一位置“1”后得到一种汉字内码

- 为便于打印、显示汉字，汉字字形必须预先存在机内
  - 字库 (font)：所有汉字形状的描述信息集合
  - 不同字体 (如宋体、仿宋、楷体、黑体等) 对应不同字库
  - 从字库中找到字形描述信息，然后送设备输出

    通过汉字内码，可以确定其在字库中的位置，也就找到了字形信息

- 字形主要有两种描述方法：
  - 字模的点阵描述（图像方式）
  - 字模的轮廓描述（图形方式）
    - 直线向量轮廓
    - 曲线轮廓（True Type字形）



数据的基本宽度（复习）

- 比特（bit）是计算机中处理、存储、传输信息的最小单位

- 字节 (Byte)，二进制信息的计量单位，也称“位组”
  - 1个字节=8个bit
  - 现代存储器通常按字节编址
  - 此时，字节是最小可寻址单位

(addressable unit)

地址编号	存储内容
0_{10}	0010 0110B
1_{10}	1010 1100B
……	……
n_{10}	1101 0011B

- 字 (word) 也经常用来作为数据的长度单位
  - 1个字一般来说是16个bit 2个字节

“字”（数据的宽度）和 “字长”

- **“字”和 “字长”的概念不同**
  - **“字长”指定点运算数据通路的宽度。**

    （CPU内部有进行数据运算、存储和传送的部件，这些部件的宽度基本上要一致，才能相互匹配。因此，”字长”等于CPU内部定点运算部件的位数、通用寄存器的宽度等 ——学到后面章节就更明白了）
  - **“字”表示被处理信息的单位，用来度量数据类型的宽度。**
  - **字和字长的宽度可以一样，也可不同。**

    例如，x86体系结构定义“字”的宽度一直都是16位

    但从386开始字长就是32位了。

    再比如：IA-32的字长肯定是32位，

    但是“字” 16位，双字DWORD： 32位等 （都是数据的宽度单位）

数据量的度量单位

- 存储二进制信息时的度量单位要比字节或字大得多
- 主存容量经常使用的单位，如：
  - “千字节”（KB），1KB=2^{10}字节=1024B
  - “兆字节”（MB），1MB=2^{20}字节=1024KB
  - “千兆字节”（GB），1GB=2^{30}字节=1024MB
  - “兆兆字节”（TB），1TB=2^{40}字节=1024GB
- 主频和带宽使用的单位，如：
  - “千比特/秒”（kb/s），1kbps=10^3 b/s=1000 bps
  - “兆比特/秒”（Mb/s），1Mbps=10^6 b/s =1000 kbps
  - “千兆比特/秒”（Gb/s），1Gbps=10^9 b/s =1000 Mbps
  - “兆兆比特/秒”（Tb/s），1Tbps=10^{12} b/s =1000 Gbps

* 数据量的度量单位

- **硬盘和文件使用的单位**
  - **不同的硬盘制造商和操作系统用不同的度量方式，因而比较混乱**
  - **为避免歧义，国际电工委员会（IEC）给出了二进制前缀字母定义，可用不同的前缀表示所采用的度量方式**

十进制前缀			IEC 定义的二进制前缀			值差
单词	前缀	值	单词	前缀	值	（%）
kilobyte	KB/kB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

程序中数据类型的宽度

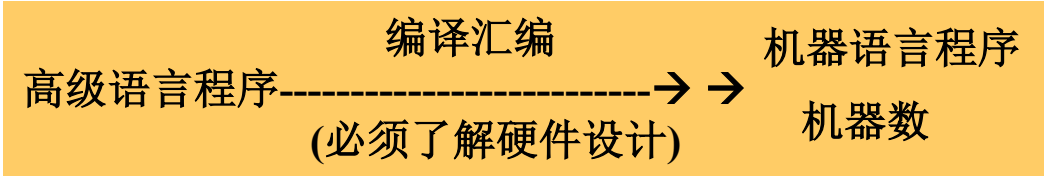
- 高级语言支持多种类型、多种长度的数据
- 不同机器上表示的同一种类型的数据可能宽度不同
- 程序中的数据有相应的机器级表示方式和相应的处理指令
  (在第五章指令系统介绍具体指令)

C语言中数值数据类型的宽度 (单位：字节)

C声明	典型32位机器	Compaq Alpha机器
char	1	1
short int	2	2
int	4	4
long int	4	8
char*	4	8
float	4	4
double	8	8

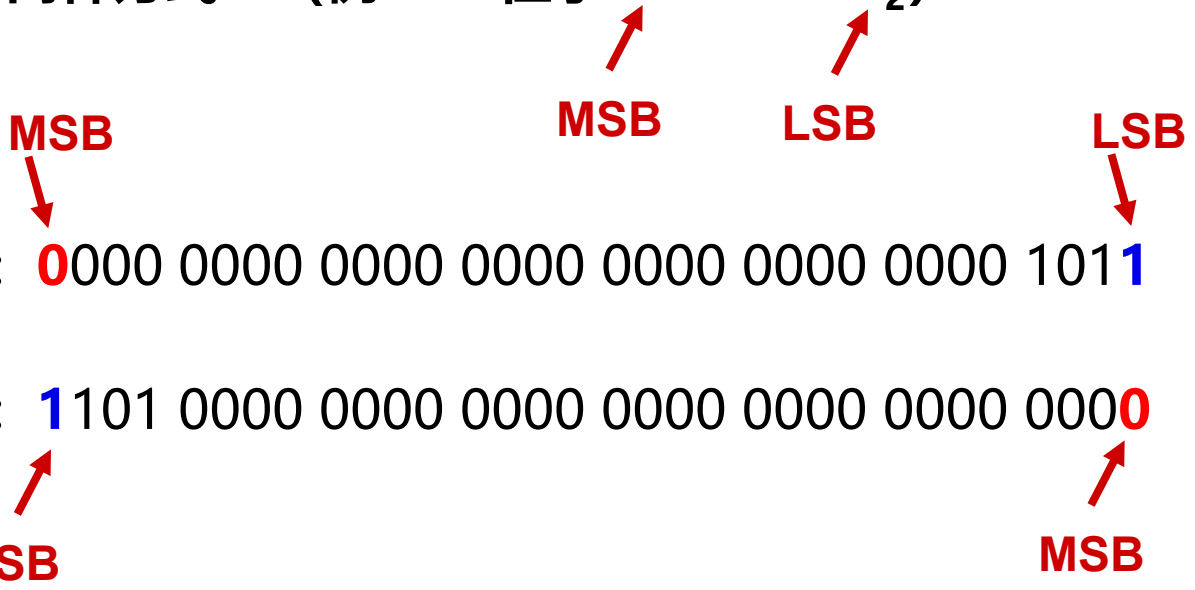
Compaq Alpha是64位机器，即字长为64位

从表中看出：同类型数据并不是所有机器都采用相同的宽度，分配的字节数随机器字长和编译器的不同而不同。



数据（机器数）的存储和排列顺序

- 机器数的位排列顺序有两种方式：（例：32位字: 0...01011$_2$）



- 用LSB(Least Significant Bit)来表示最低有效位
- 用MSB来表示最高有效位

数据（机器数）的存储和排列顺序

- 如果以字节为一个排列单位，则
- LSB(Least Significant Byte)表示最低有效字节
- MSB表示最高有效字节

  - 80年代开始，几乎所有通用机器都用字节编址
  - ISA设计时要考虑的两个问题：
    - 如何根据一个地址取到一个32位的字？ - 字的存放问题
    - 一个字能否存放在任何地址边界？ - 字的边界对齐问题

- 存放好之后，什么叫做【数据的地址】？
  - 【连续若干单元中最小的编号】（从小地址开始存放数据）
    - 若一个short型（16位）数据存放在单元0x0100和0x0101中
    - 那么它的地址是——0x0100
    - 它的值呢?

大端方式和小端方式

例如，若 32位int i = -65535，其地址为内存100号单元（即占100#～103#），则用“取数”指令访问100号单元取出 i 时，必须清楚 i 的4个字节是如何存放的。

0000 0000 0000 0000 1111 1111 1111 1111

1111 1111 1111 1111 0000 0000 0000 0001

$65535=2^{16}-1$

$[-65535]_{补}=FFFF0001H$

Word:

FF	FF	00	01
103	102	101	100
MSB			LSB
100	101	102	103

little endian word 100# 小端

big endian word 100# 大端

大端方式（Big Endian）：MSB所在的地址是数的地址

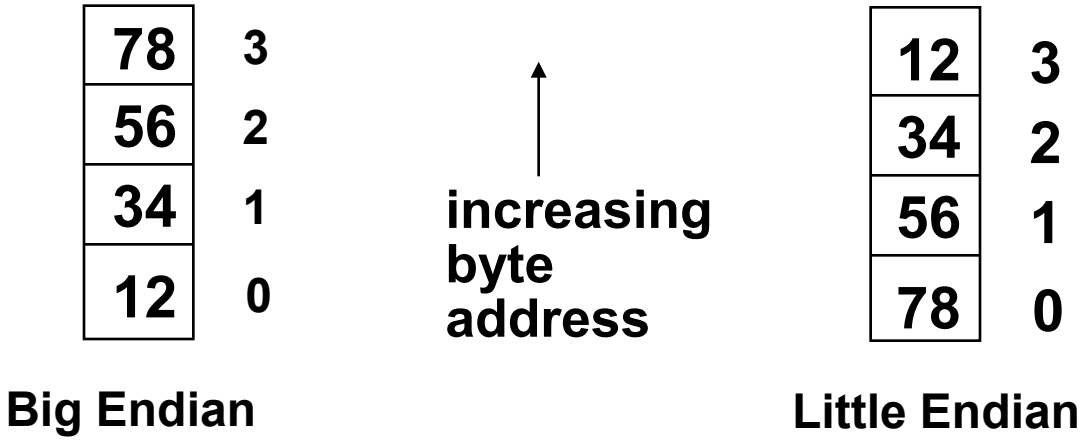
e.g. IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

小端方式（ Little Endian）：LSB所在的地址是数的地址

e.g. Intel 80x86, DEC VAX

有些机器两种方式都支持，可通过特定控制位来设定采用哪种方式。

Byte Swap Problem（字节交换问题）



上述存放在0号单元的数据（占4个字节）是什么？ 12345678H

存放方式不同的机器间程序移植或数据通信时需要进行顺序转换

音、视频和图像等文件格式或处理程序都涉及到字节顺序问题

ex. Little endian: GIF, PC Paintbrush, Microsoft RTF,etc

Big endian: Adobe Photoshop, JPEG, MacPaint, etc

第四讲小结

- 非数值数据的表示
  - 逻辑数据用来表示真/假或N位位串，按位运算
  - 西文字符：用ASCII码表示
  - 汉字：汉字输入码、汉字内码、汉字字模码
- 数据的宽度
  - 位、字节、字（不一定等于字长），k/K/M/G/…有不同的含义
- 数据的存储排列
  - 数据的地址：连续若干单元中最小的地址，即：从小地址开始存放数据
    - 若一个short型（16位）数据si存放在单元0x0100和0x0101中，那么si的地址是什么——0x0100
  - 大端方式：用MSB存放的地址表示数据的地址
  - 小端方式：用LSB存放的地址表示数据的地址

本章作业

- **教材第1章习题：**
- **3、4、5、6、9、12（5）、14、17**

- **作业提交截止时间： 9月21号晚上24:00**