计算机组成原理与接口技术——基于MIPS架构

Mar, 2022

第1讲 计算机基础

杨明
华中科技大学电子信息与通信学院
myang@hust.edu.cn

Agenda

- 内容
  - 计算机系统的发展历史
  - 计算机系统的基本结构
  - 计算机系统的基本工作方式
  - 不同计算机系统的结构模型
  - 计算机系统中的数据、信息的表示方式、存储方式
  - 数据运算基础
- 目的
  - 掌握计算机的基本构成、基本工作原理
  - 掌握不同结构模型计算机系统的特点
  - 掌握不同数制之间的转换、数据在计算机系统中的表示方式、数据的存储格式（大字节序、小字节序）
  - 掌握数据运算基础知识（符号数、无符号数、定点数、浮点数的加减运算规则）

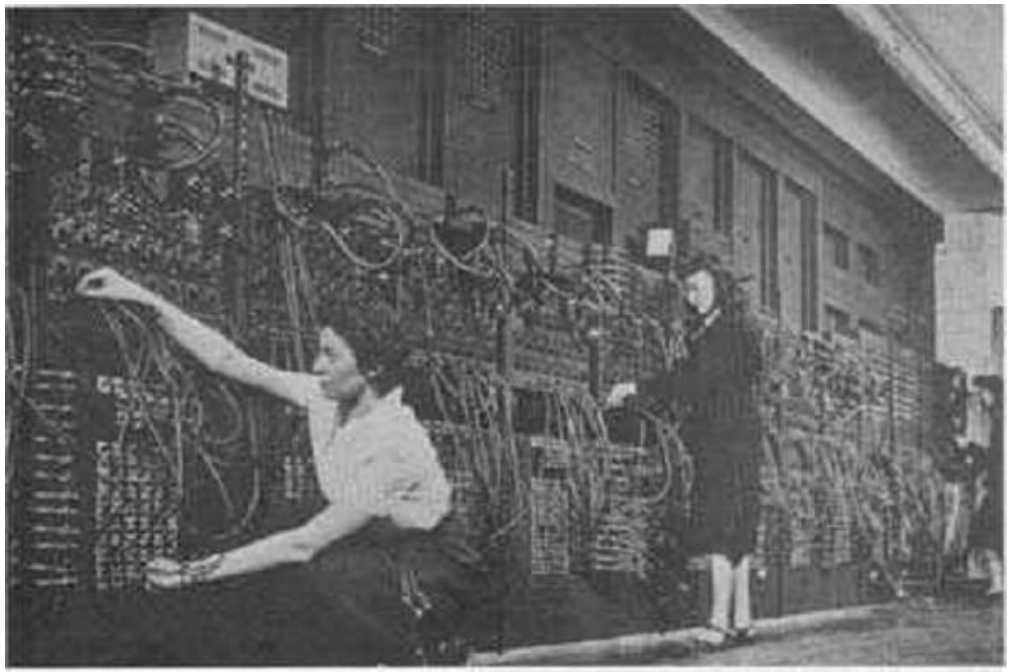
1.1 计算机发展简史

- 什么是计算机（电脑、Computer）？
  - 一种可以接受用户的输入信息，能够存储大量的数据，能够自动、高速、准确地对数据进行处理、运算，并且能够输出各种处理结果的电子设备。
  - 我们平时讲到“计算机”一词，都是指含有硬件（裸机）和软件的计算机系统。

- 微处理器(Micro Processor Unit，MPU)
  - 是指由一片超大规模集成电路制作的具有运算器与控制器功能的中央处理器部件(Central Processor Unit，CPU)，它的主要特征是能够执行指令，但本身还不能构成一台计算机。
  - 以MPU为核心，配上存储器及必要的输入输出接口与外围设备，还有相应的软件系统，因而就组成了运算与处理功能完备、外部设备齐全、能与使用者直接交互的计算机。

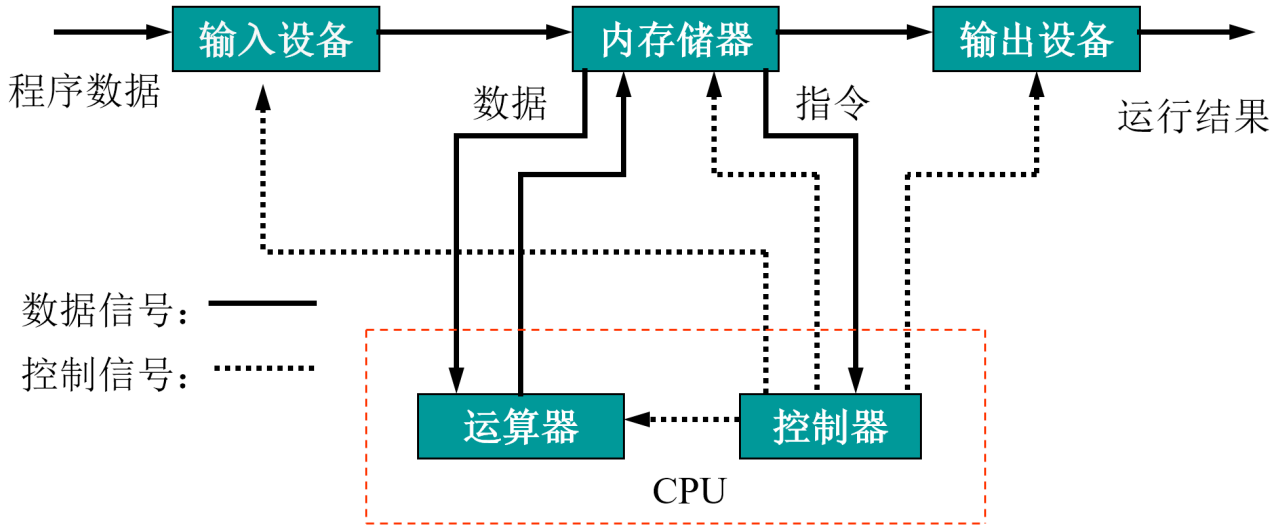
1.1 计算机发展简史

- 第一代：真空管（电子管Vacuum Tube ）1946～59年
  - 46年诞生第1台计算机 ENIAC（Electronic Numerical Integrator And Calculator）
    - 速度 5000次/秒，18800个电子管，功耗150kw，重30吨，占地170m^2 ，造价$100万；
    - 十进制表示/运算，存储器由20个累加器　组成，每个累加器存10位十进制数，每一位由10个真空管表示。
    - 采用手动编程，通过设置开关和插拔电缆来实现。



1.1 计算机发展简史

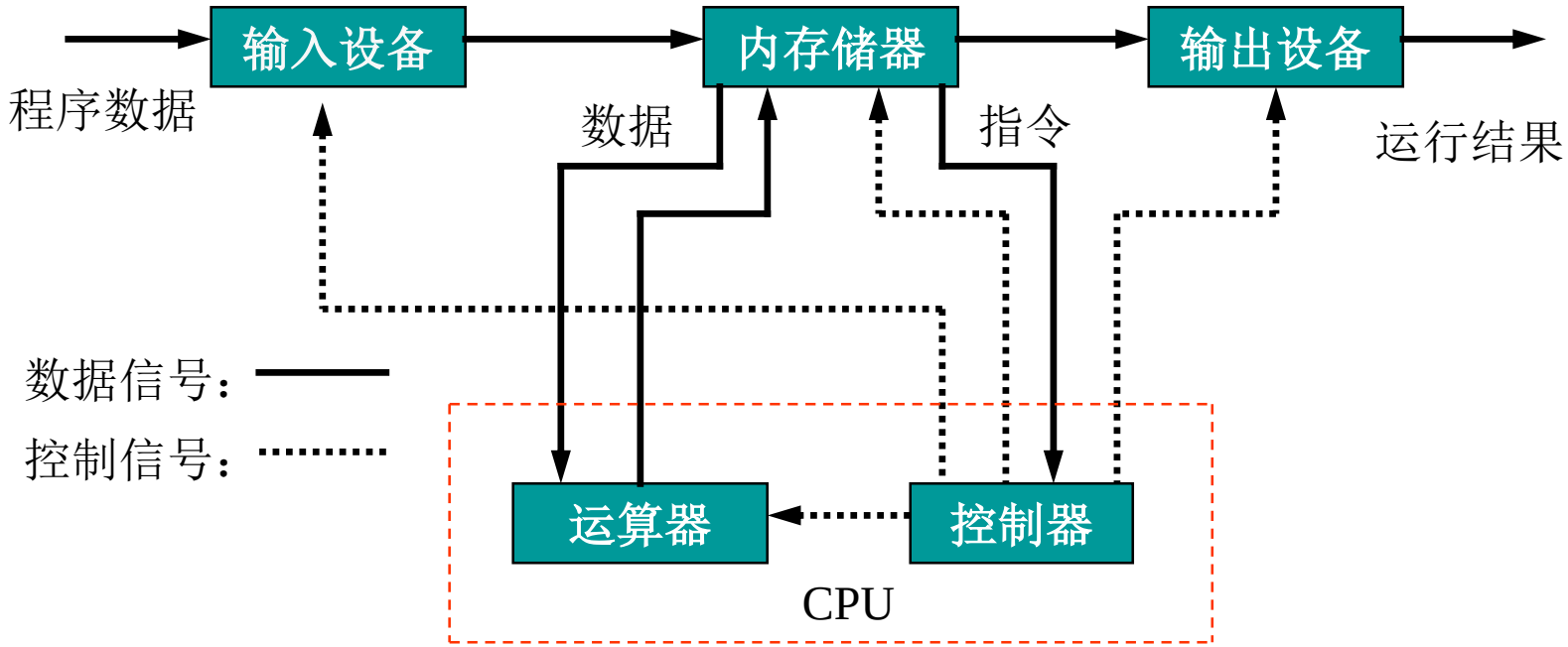
- 第一代：真空管（电子管Vacuum Tube ）1946～59年
  - 冯·诺依曼机（ Von Neumann Machine ）
    - 冯·诺依曼结构的主要思想
      - 计算机应由运算器、控制器、存储器、输入设备和输出设备五个基本部件组成。
      - 各基本部件的功能是：
        - 存储器不仅能存放数据，而且也能存放指令，形式上两者没有区别，但计算机应能区分数据还是指令；
        - 控制器应能自动执行指令；
        - 运算器应能进行加/减/乘/除四种基本算术运算，并且也能进行一些逻辑运算和附加运算；
        - 操作人员可以通过输入设备、输出设备和主机进行通信。



      - 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。
      - 采用“存储程序”工作方式。

1.1 计算机发展简史

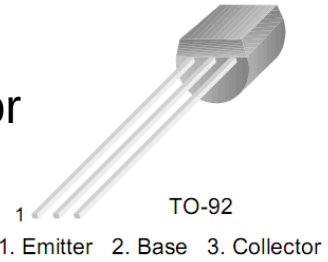
- 第一代：真空管（电子管Vacuum Tube ）1946～59年
  - 冯·诺依曼机（Von Neumann Machine）
    - 45年冯·诺依曼提出“存储程序和程序控制”思想：
      - 当人们要解决问题时，首先将问题程序化，形成指令序列，然后将它存入存储器中，再由CPU的控制器从存储器中逐条取出指令解释，并取出该指令要处理的操作数送往运算器中执行，最后输出程序结果。即计算机的基本工作过程也就是进行“程序存储和程序控制”。



▶ 第二代：晶体管 1959～64年

- 元器件
  - 逻辑元件采用晶体管
  - 内存由磁芯构成
  - 外存为磁鼓与磁带。
- 特点
  - 变址
  - 浮点运算
  - 多路存储器
  - I/O处理机
  - 中央交换结构(非总线结构)。
- 软件：使用高级语言，提供了系统软件。
- 代表机种：IBM 7094 (scientific)、1401 (business)和 DEC PDP-1

晶体管：
Transistor



DEC PDP-1

▶ 第三代：SSI/MSI 1965～70年

- 元器件：
  - 逻辑元件与主存储器均由集成电路（IC）实现。
- 特点：
  - 微程序控制，
  - Cache，
  - 虚拟存储器，
  - 流水线等。

Cray-1

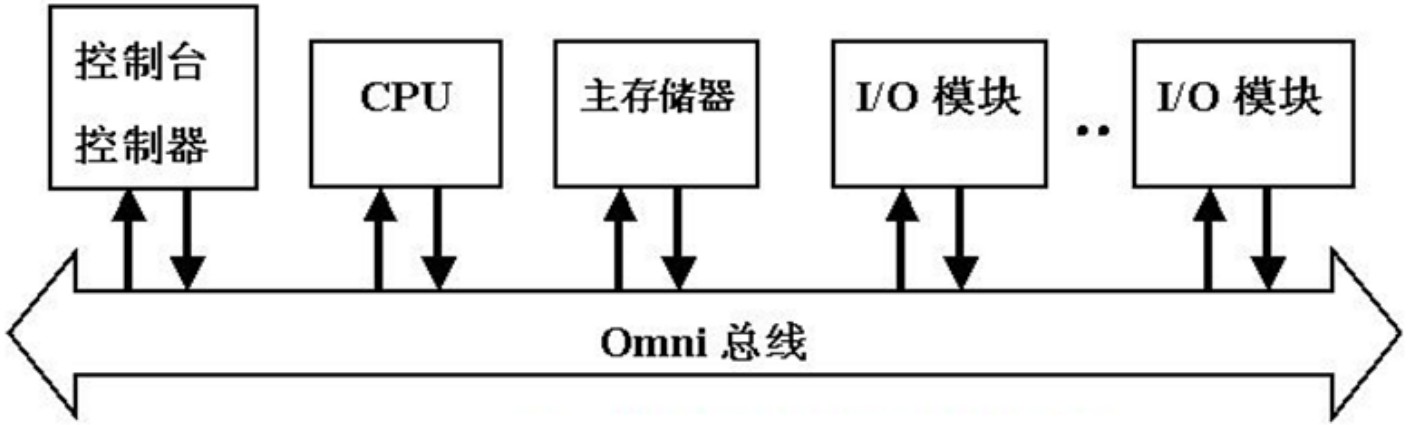
- 代表机种：IBM 360和DEC PDP-8（大/巨型机与小型机同时发展 ）
  - 巨型机(Supercomputer)：Cray-1
  - 大型机(Mainframe)：IBM360系列
  - 小型机(Minicomputer)：DEC PDP-8

- 第三代：SSI/MSI 1965～70年
  - IBM 360系列计算机
    - IBM公司于1964年研制成功
    - 引入“兼容机”(系列机)概念
      - 相同的或相似的指令集
      - 相同或相似的操作系统
      - 更高的速度
      - 更多的I/O端口数
      - 更大的内存容量
      - 更高的价格

IBM 360

- 低端机指令集是高端机的一个子集，称为“向后兼容”。原来机器上的程序可以不改动而在新机器上运行，但性能不同。

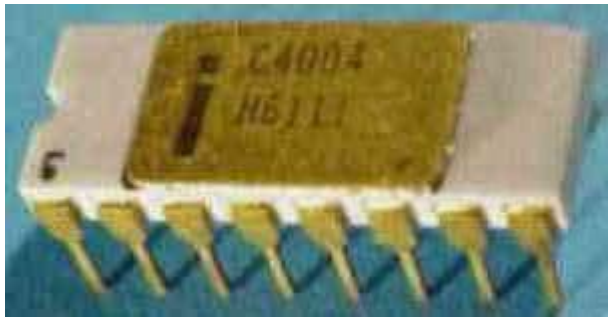
- 第三代：SSI/MSI 1965～70年
  - DEC公司的PDP-8计算机
    - 同在64年出现；
    - 与IBM360相比，价格更低、更小巧，因而被称为小型机（Minicomputer）
    - PDP-8“创造了小型机概念，并使之成为数十亿美元的工业”，使DEC成为了最大的小型机制造商。
    - 主要特点：首次采用总线结构。
      - 具有高度的灵活性，可扩充性好（允许将新的符合标准的模块插入总线形成各种配置），节省器件，体积小，价格便宜
    - Omnibus总线
      - 包含了96个独立的信号通道，用以传送控制、地址和数据信号



PDP-8计算机总线结构图

▶ 第四代：LSI/VLSI/ULSI（1970～至今）

- 微处理器和半导体存储器技术发展迅猛，微型计算机出现。
  - 使计算机以办公设备和个人电脑的方式走向普通用户。
- 半导体存储器
  - 70年Fairchild公司生产出第一个相对大容量半导体存储器
  - 74年位价格低于磁芯的半导体存储器出现，并快速下跌
  - 从70年起，存储密度几乎是每3年提高4倍
- 微处理器
  - 微处理器芯片密度不断增加，使CPU中所有元件放在一块芯片上成为可能。
  - Intel公司71年开发出第一个微处理器芯片4004。
- 特点：共享存储器，分布式存储器及大规模并行处理系统

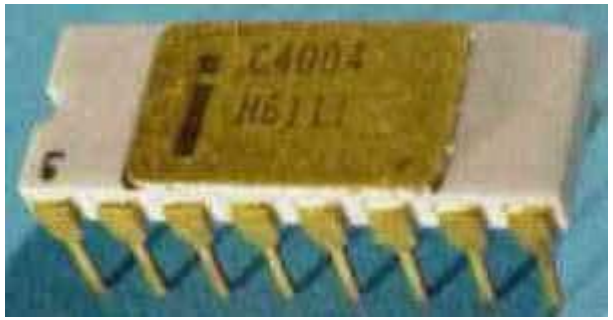


第一代Intel处理器——4004，样子很像小虫子

▶ 第四代：LSI/VLSI/ULSI（1970～至今）

- Intel 80X86系列微处理器的发展

名称/（工作频率）	推出年份	位数	内含晶体管
4004（1MHz）	1971	4	2250
8080	1974	8	0.5万
8085	1976	8	0.6万
8086	1978	16	2.9万
8088/（4.77MHz）	1979	16	2.9万
80186/80188	1982	16	5.6万
80286（20MHz）	1982	16	13万
80386（33MHz）	1985	32	27.5万
80486（66MHz）	1989	32	120万



第一代Intel处理器——4004，样子很像小虫子

1971年，当时还处在发展阶段的Intel推出了世界上第一台微处理器4004。这不但是第一个用于计算器的4位微处理器，也是第一款个人有能力买得起的电脑处理器。
1981年8088芯片首次用于IBM PC机中，开创了全新的微机时代。也正是从8088开始，PC机（个人电脑）的概念开始在全世界范围内发展起来。

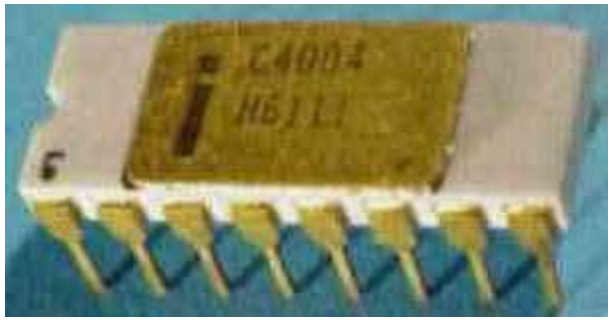
▶ 第四代：LSI/VLSI/ULSI（1970～至今）

- Intel 80X86系列微处理器的发展

名称	推出年份	位数	内含晶体管
80486	1989	32	120 万
Pentium 奔腾	1993.03	32	310 万
Pentium Pro 高能奔腾	1995.11	32	550 万
Pentium MMX 多能奔腾	1997.01	32	450 万
Pentium II 奔腾 2	1997.05	32	750 万
Pentium III 奔腾 3	1999.01	32	950 万
Pentium IV 奔腾 4	2000.11	32	3400 万
超线程奔腾 4	2004	32	1 亿 2500 万
90 纳米超线程奔腾 4	2005	64	1 亿 7000 万
90 纳米双核奔腾 4	2005	64	2 亿 3000 万
65 纳米酷睿双核处理器	2006	32	1 亿 5000 万
65/45nm 酷睿 2 双核处理器	2006	64	2 亿 9500 万

PC机经过50多年来的发展，其速度越来越快、功能越来越强，但不管若何发展，PC的基本结构并没有改变，在结构上仍然采用“冯·诺依曼”结构，功能上也保持着向前的兼容性。

- 第四代：LSI/VLSI/ULSI（1970～至今）
  - Intel 80X86系列微处理器的发展

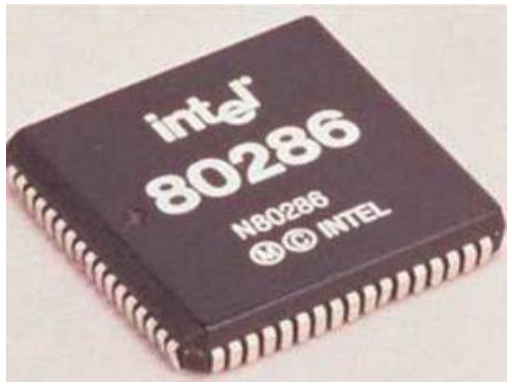


第一代Intel处理器——4004，样子很像小虫子

x86的鼻祖：8086

8088：IBM PC的御用之选

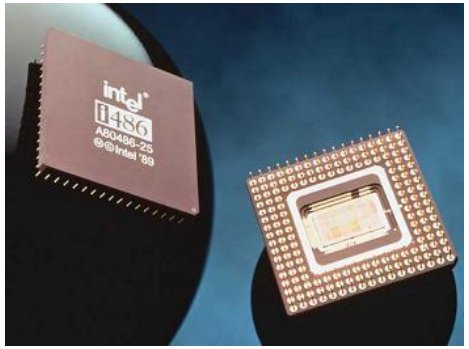
- 第四代：LSI/VLSI/ULSI（1970～至今）
  - Intel 80X86系列微处理器的发展



80286，成就了康柏公司

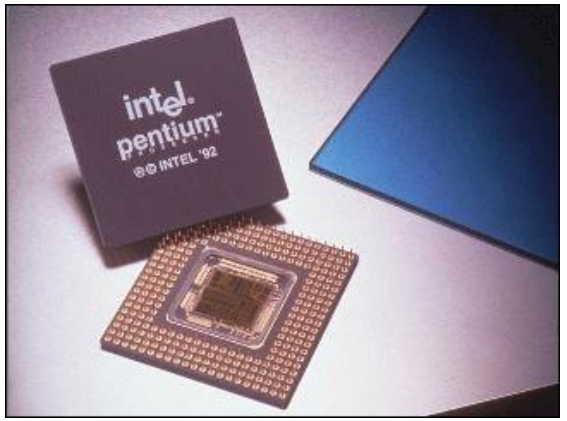
康柏公司率先推出286 PC，成功超越IBM

80386, Intel 第一代32位CPU



80486, 成熟的32位应用，Intel最后一代以数字编号的CPU

- 第四代：LSI/VLSI/ULSI 1970～至今）
  - Intel 80X86系列微处理器的发展



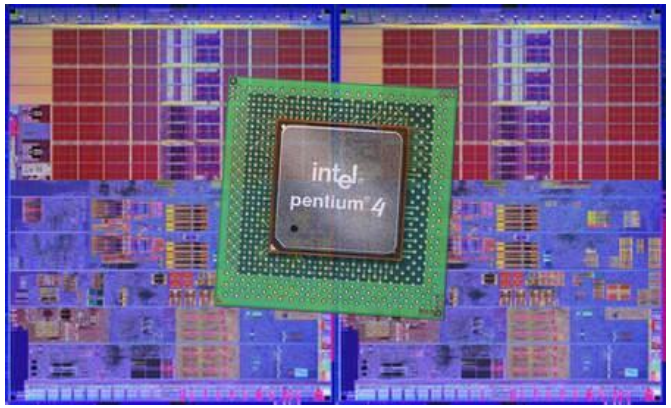
Pentium，第一种不以数字命名的处理器

Pentium II、Celeron—— PC处理器开始分化：Celeron就是简化版的Pentium

- 第四代：LSI/VLSI/ULSI（1970～至今）
  - Intel 80X86系列微处理器的发展

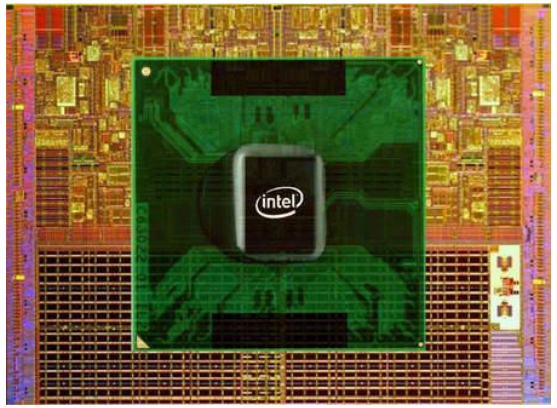
Pentium III，不断的改进：Slot、Socket

2000年，Pentium 4，高噪音低性能的代名词

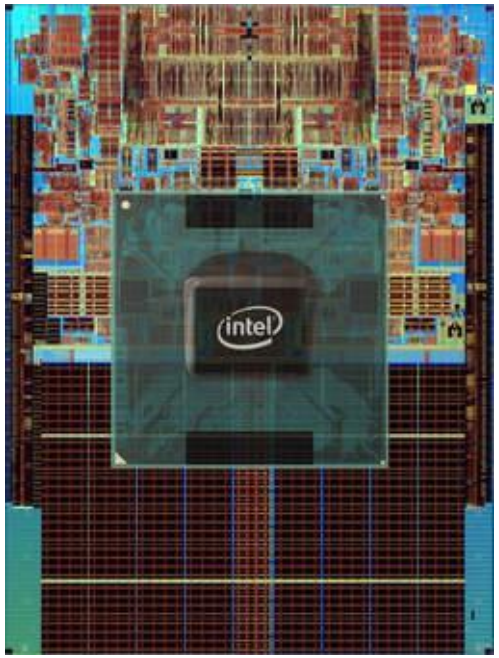


2005年，Pentium 4：开始支持64位，变身双核

- 第四代：LSI/VLSI/ULSI（1970～至今）
  - Intel 80X86系列微处理器的发展



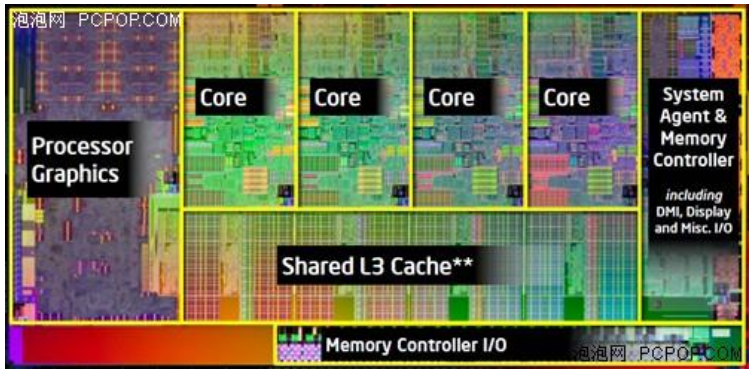
2006年Intel宣布了酷睿双核处理器。这是第一款面向便携式电脑设计的双核处理器，32位



酷睿2——Intel决杀产品，64位，双核/四核，65nm/45nm

- 第四代：LSI/VLSI/ULSI（1970～至今）
  - Intel 80X86系列微处理器的发展

2010年1月份，Intel发布了代号为Clarkdale的新一代酷睿i3/i5处理开始将HD Graphics图形核心跟CPU集成在一块芯片上(i3-530)

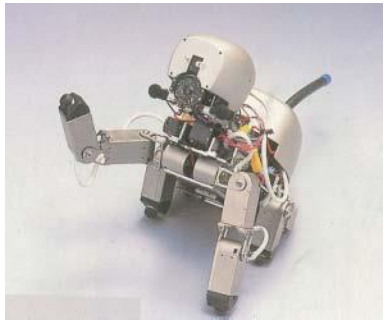


2011年初，Intel发布了全新的Sandy Bridge架构，首次将GPU芯片跟CPU融合在一起(i3-2100)

现在的酷睿i系列分为i3 i5 i7系列，分别对应低、中、高端市场。

1.1 计算机发展简史

- 当前计算机的应用领域
  - 除了通用计算机(PC之外)，计算机系统无处不在
    - 消费电子：PDA、手机、MP3、MP4、数码相机
    - 家电：数字电视、空调、冰箱、微波炉、机顶盒
    - 汽车电子、医疗器械
    - 网络设备、通信设备
    - 机器人
    - …



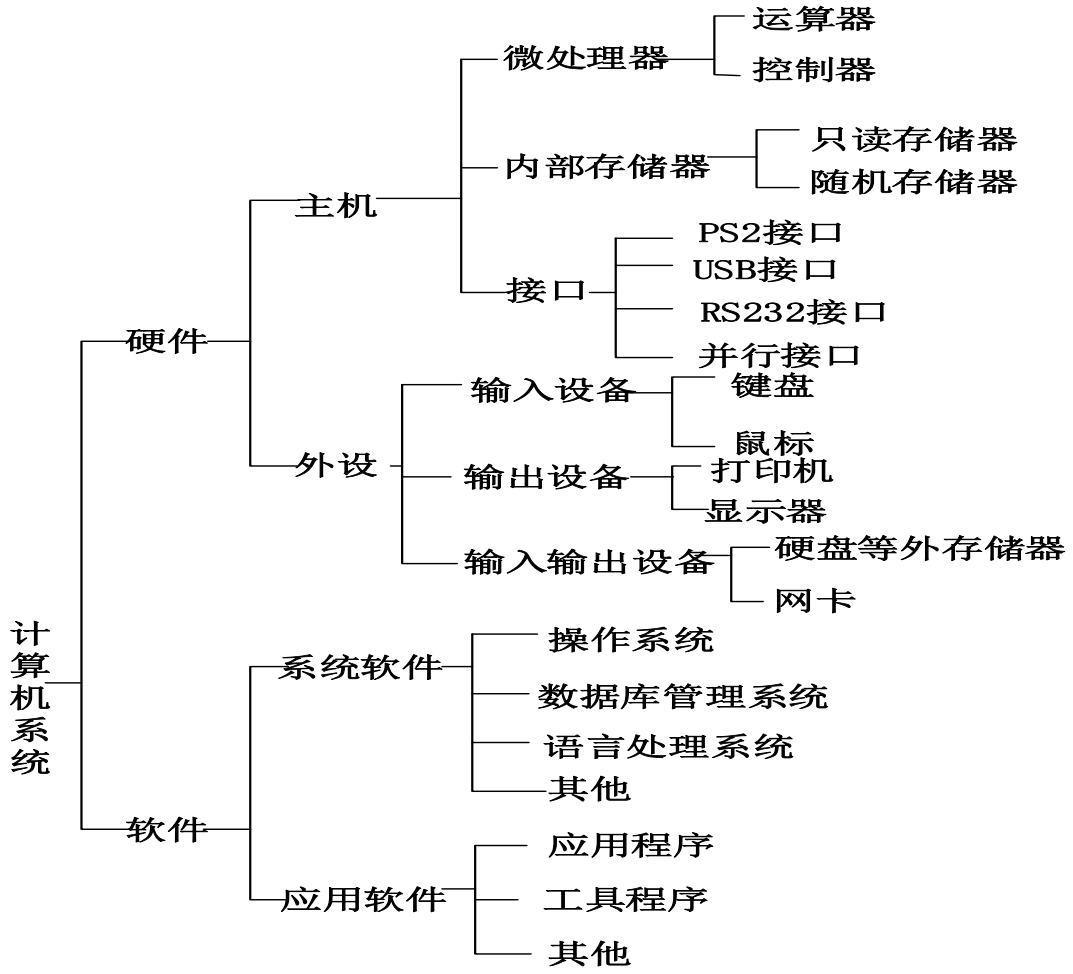
Agenda

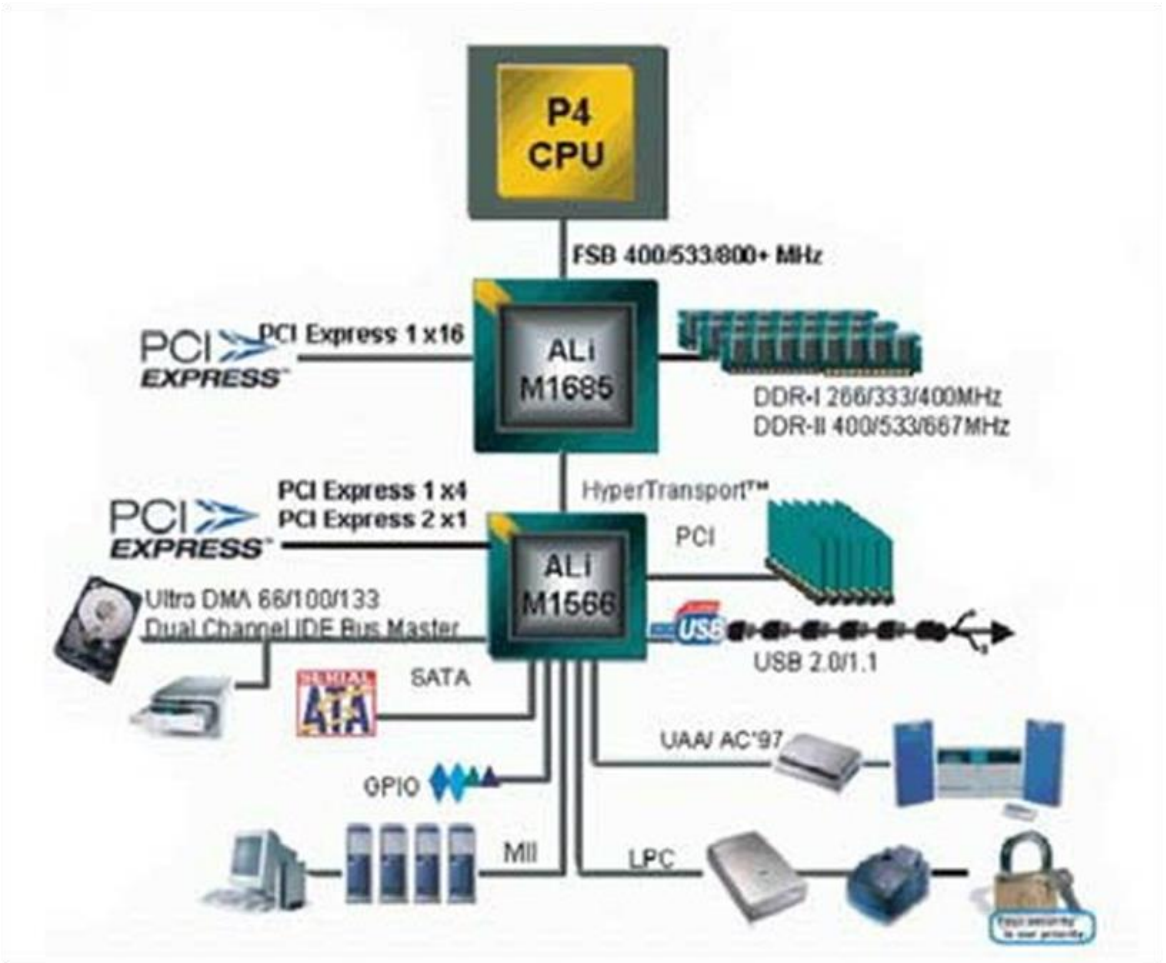
- 内容
  - 计算机系统的发展历史
  - 计算机系统的基本结构
  - 计算机系统的基本工作方式
  - 计算机系统的结构模型
  - 计算机系统中的数据、信息的表示方式、存储方式
  - 数据运算基础
- 目的
  - 掌握计算机的基本构成、基本工作原理
  - 掌握不同结构模型计算机系统的特点
  - 掌握不同数制之间的转换、数据在计算机系统中的表示方式、数据的存储格式（大字节序、小字节序）
  - 掌握数据运算基础知识（符号数、无符号数、定点数、浮点数的加减运算规则）

1.2 计算机系统的构成

▶ 通用计算机的基本构成

- 通用计算机具有计算机的标准形态，其典型产品为PC

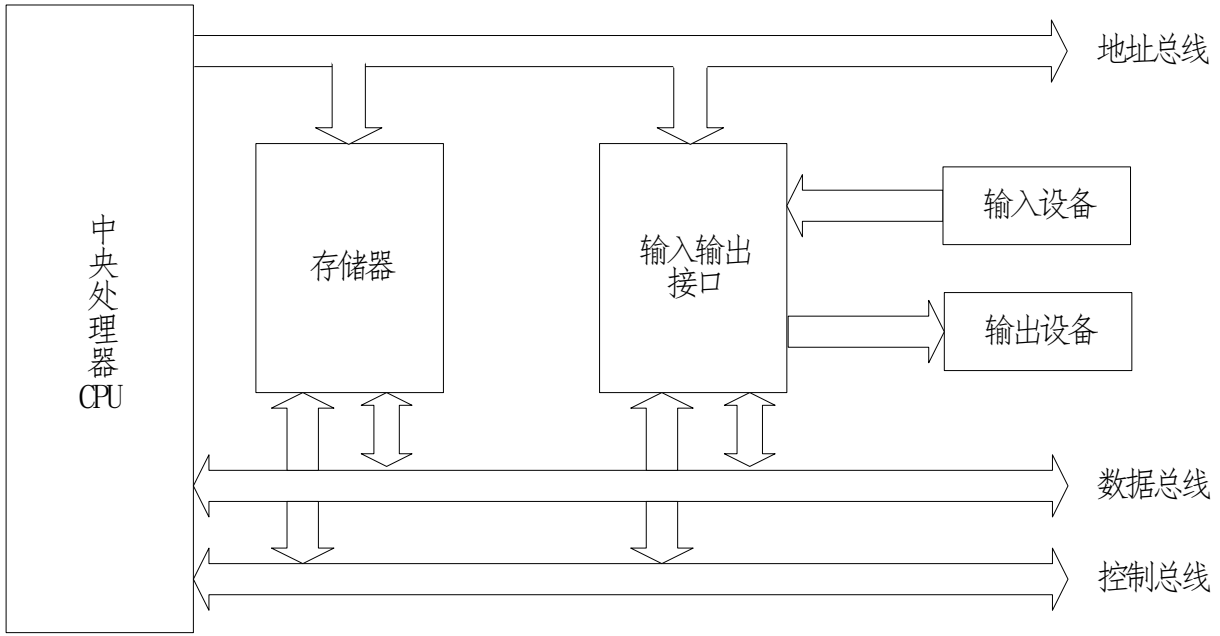




1.2 计算机系统的构成

▶ 计算机硬件结构

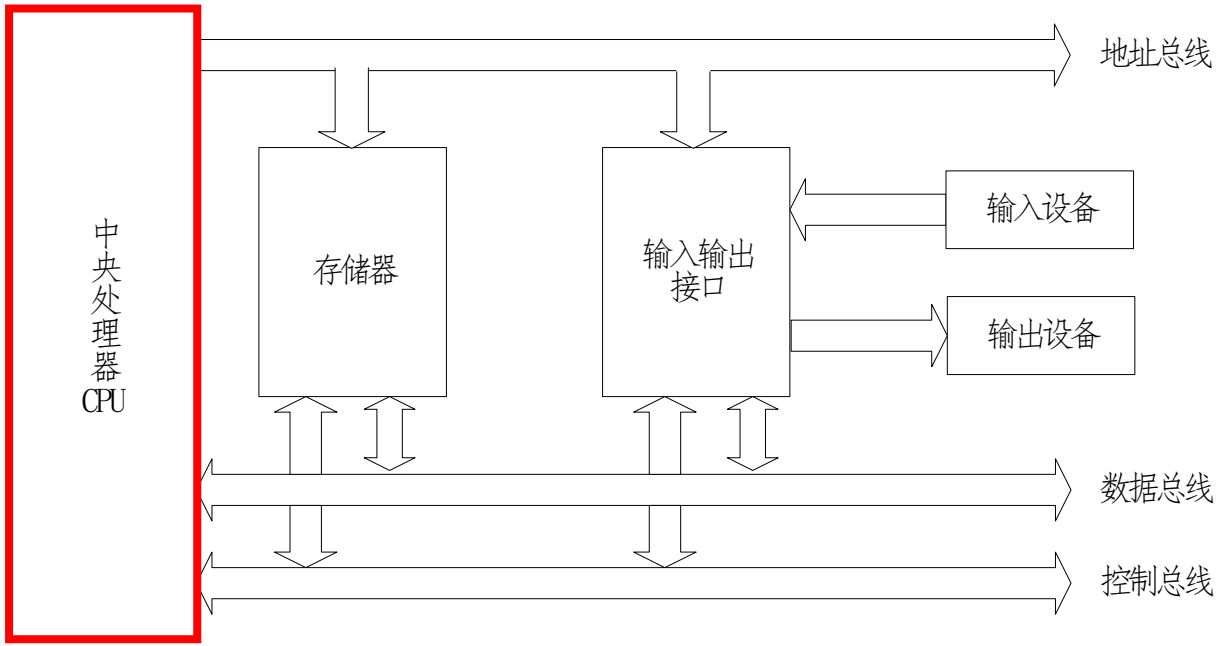
- 中央处理器CPU(central processor unit) / 微处理器(microprocessor unit)
- 存储器(memory)
- 输入输出接口
- 输入输出设备
- 总线(Bus)



1.2 计算机系统的构成

► 计算机硬件结构

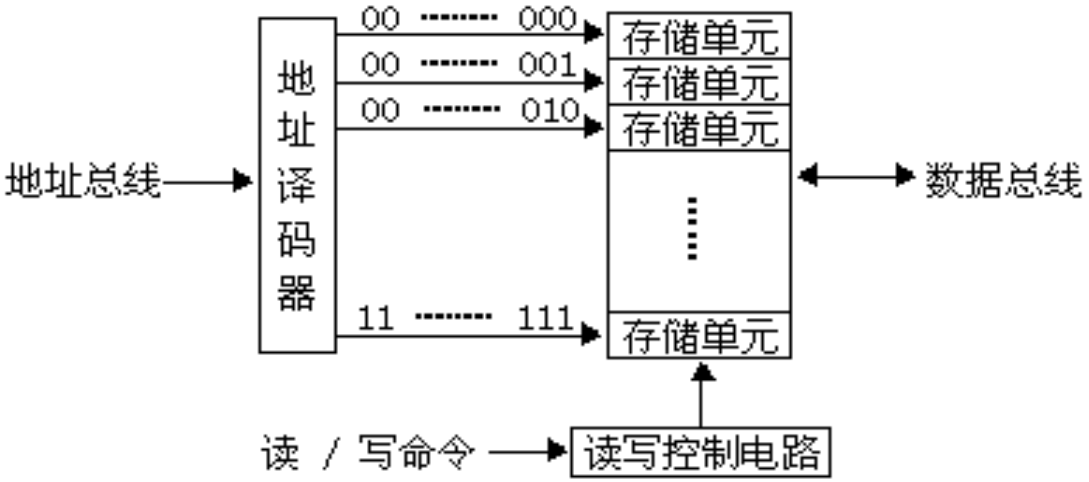
- 中央处理器CPU(central processor unit) / 微处理器(microprocessor unit)
  - 算术逻辑单元ALU(Arithmetic Logic Unit)
    - 用来进行基本的算术运算、逻辑运算、移位等各种数据操作。
  - 控制器（Control Unit）
    - 按一定的顺序从存储器中读取指令，进行译码，在时钟信号的控制下，发出一系列的操作命令，控制整个系统有条不紊地工作。
  - 寄存器（Register）组
    - CPU中有多个寄存器，用来存放操作数、运算的中间结果以及反映运算结果的状态标志等。

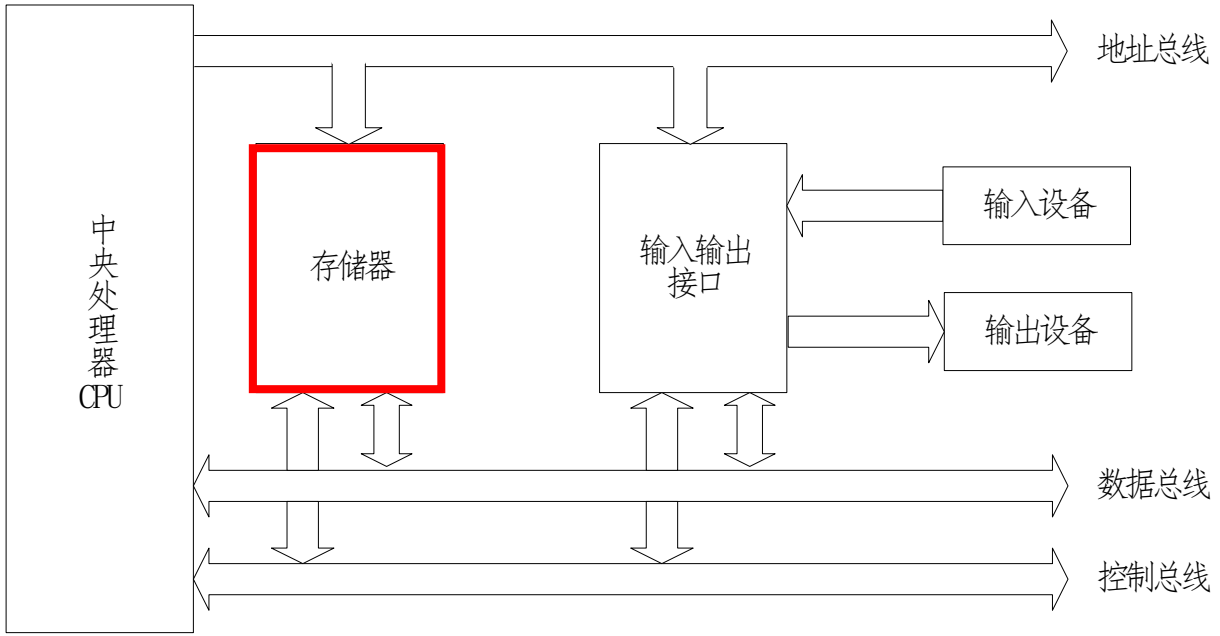


1.2 计算机系统的构成

▶ 计算机硬件结构

- 存储器(memory)
  - 存储器的功能是存储程序、数据和各种信号、命令等信息，并在需要时提供这些信息
  - 不管是程序还是数据，在存储器中都是用二进制的“1”或“0”表示，统称为信息

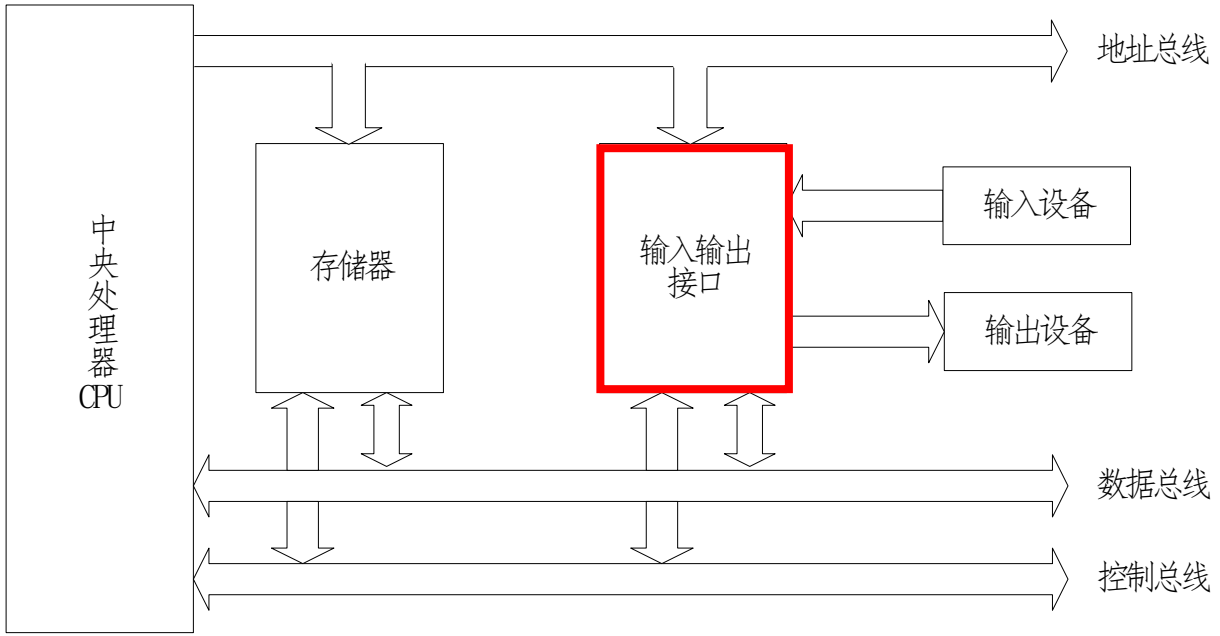




读存储器：CPU将存储单元的信息取到CPU内部；
写存储器：将CPU内部的信息送到存储单元保存；
写操作会改变存储单元信息，读操作不改变存储单元信息。
读/写操作，都需要相应的硬件电路（地址译码、读写控制）

▶ 计算机硬件结构

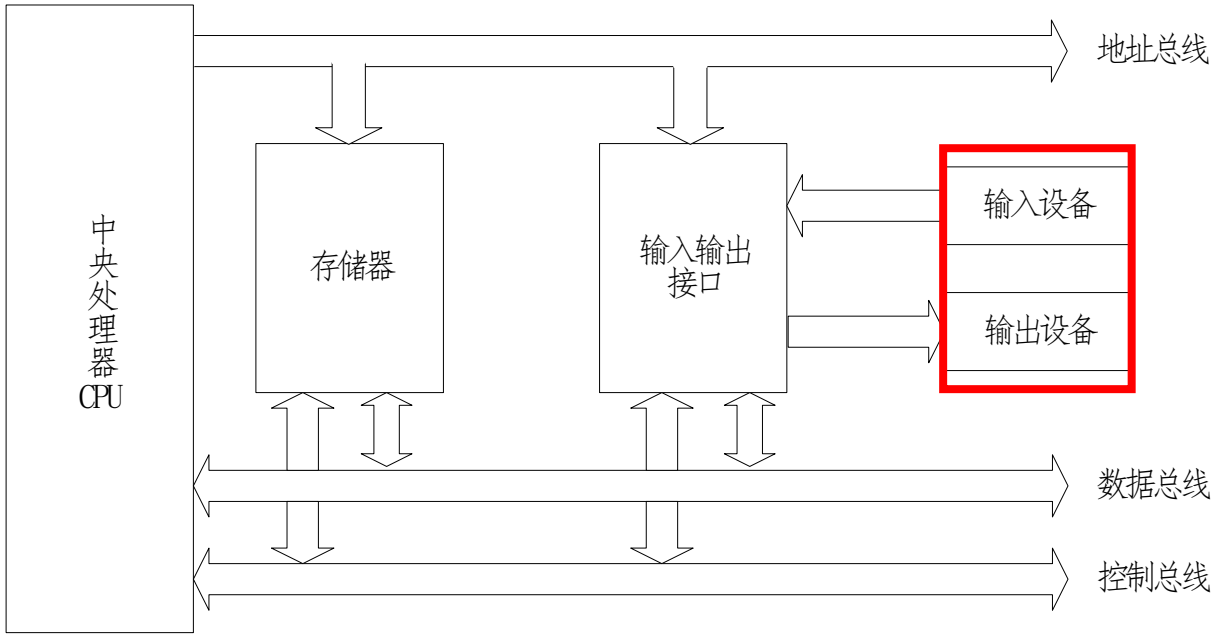
- 输入输出接口
  - 外部设备与CPU之间通过输入输出接口连接
  - 接口的作用
    - 一是外部设备大多数都是机电设备，传送数据的速度远远低于计算机，因而需要接口作数据缓存；
    - 二是外部设备表示信息的格式与计算机不同，由接口完成格式转换；
    - 三是接口还可以向计算机报告设备运行的状态，传达计算机的命令等。

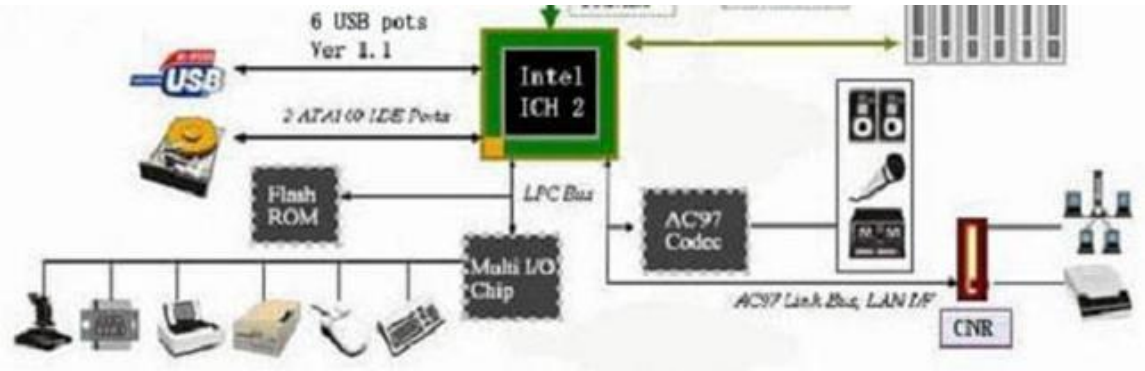


1.2 计算机系统的构成

▶ 计算机硬件结构

- 输入输出设备
  - I/O设备又称为外部设备
  - 输入设备是变换输入信息形式的部件。它将人们熟悉的信息形式变换成计算机能接收并识别的信息形式。
  - 输出设备是变换计算机的输出信息形式的部件。它将计算机处理结果的二进制信息转换成人们或其他设备能接收和识别的形式。

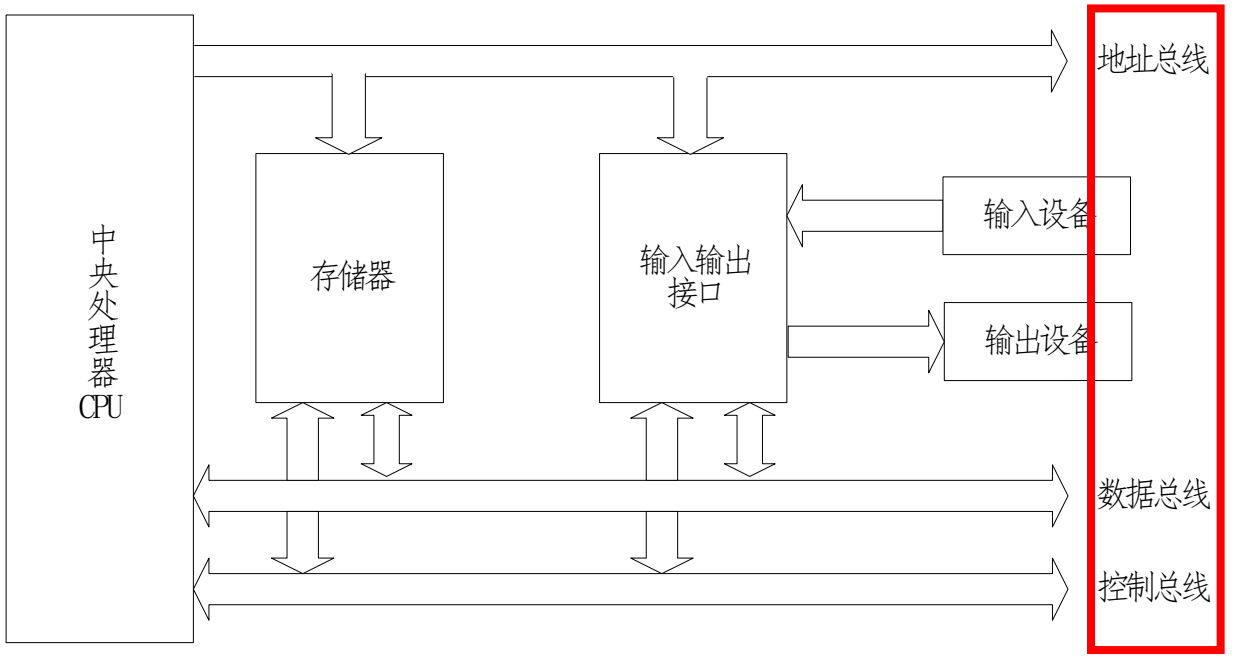




1.2 计算机系统的构成

▶ 计算机硬件结构

- 总线(Bus)
  - 把计算机各个部分有机地连接起来的导线，是各个部分之间进行信息交换的公共通道。
  - 根据传输信息的类型，可以分为3类：
    - 地址总线AB(address bus)：负责传输数据存储位置的一组信号, 采用Ai表示某一位地址总线。
    - 数据总线DB(data bus)：负责传输数据的一组信号线,采用Di表示其中的某一位数据总线
    - 控制总线CB(control bus)：在传输与交换数据时起管理控制作用的一组信号线,读信号 、写信号 、地址锁存允许信号ALE(address latch enable) 、 中断响应信号 (interrupt acknowledge)等

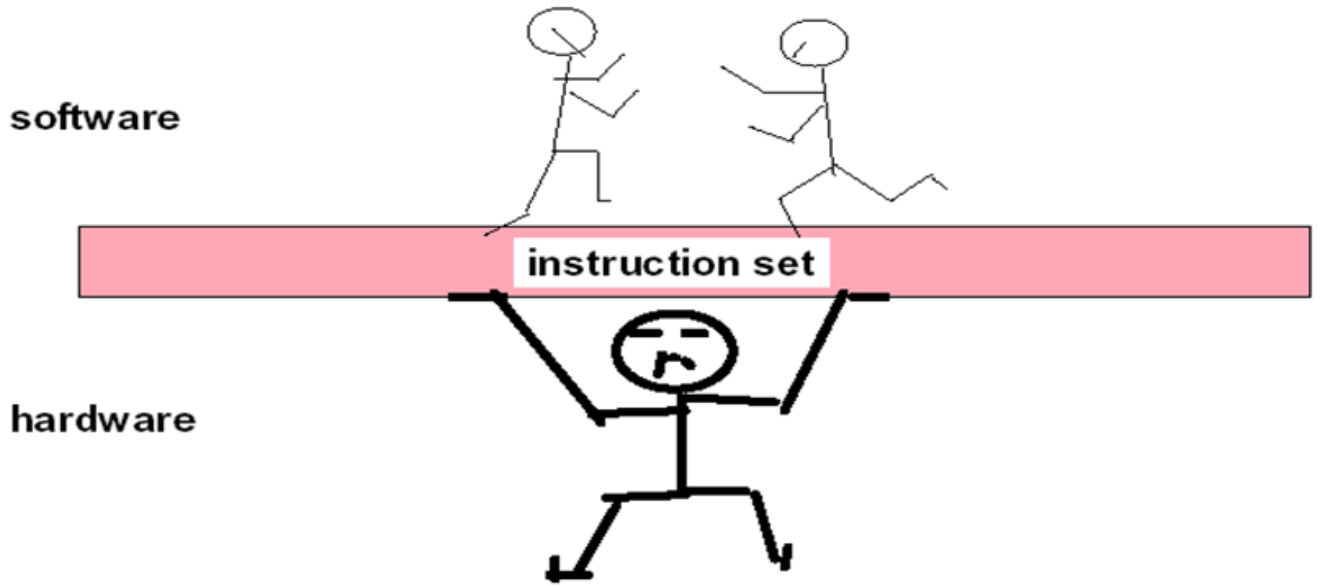


1.3 计算机工作原理

- 软件(Software)
  - 系统软件：用于管理、控制和维护计算机正常运行的程序（操作系统、语言处理程序、程序库、编辑程序等）。
  - 应用软件：用户为解决某一具体问题而专门编写的程序（文字处理软件，图像处理软件，电路设计软件，视、音频播放软件，文件压缩软件）。
- 程序(Program)
  - 指令的有序集合，一组为完成某种任务而编制的指令序列。
  - 为实现自动连续地执行程序，必须先将程序和数据送到具有记忆功能的存储器中保存起来，然后由CPU依据程序中指令的顺序周而复始地取出指令，分析指令，执行指令，直到完成全部指令操作。

1.3 计算机工作原理

- 指令(Instruction)
  - 规定计算机进行某种操作的名令
  - 计算机只能直接识别由0和1组成的编码序列——指令机器码。
- 指令集(instruction set)
  - 计算机所能执行的全部指令称为指令集或指令系统。
  - 软件和硬件的界面： ISA（指令集体系结构，Instruction Set Architecture ）



▶ 计算机语言

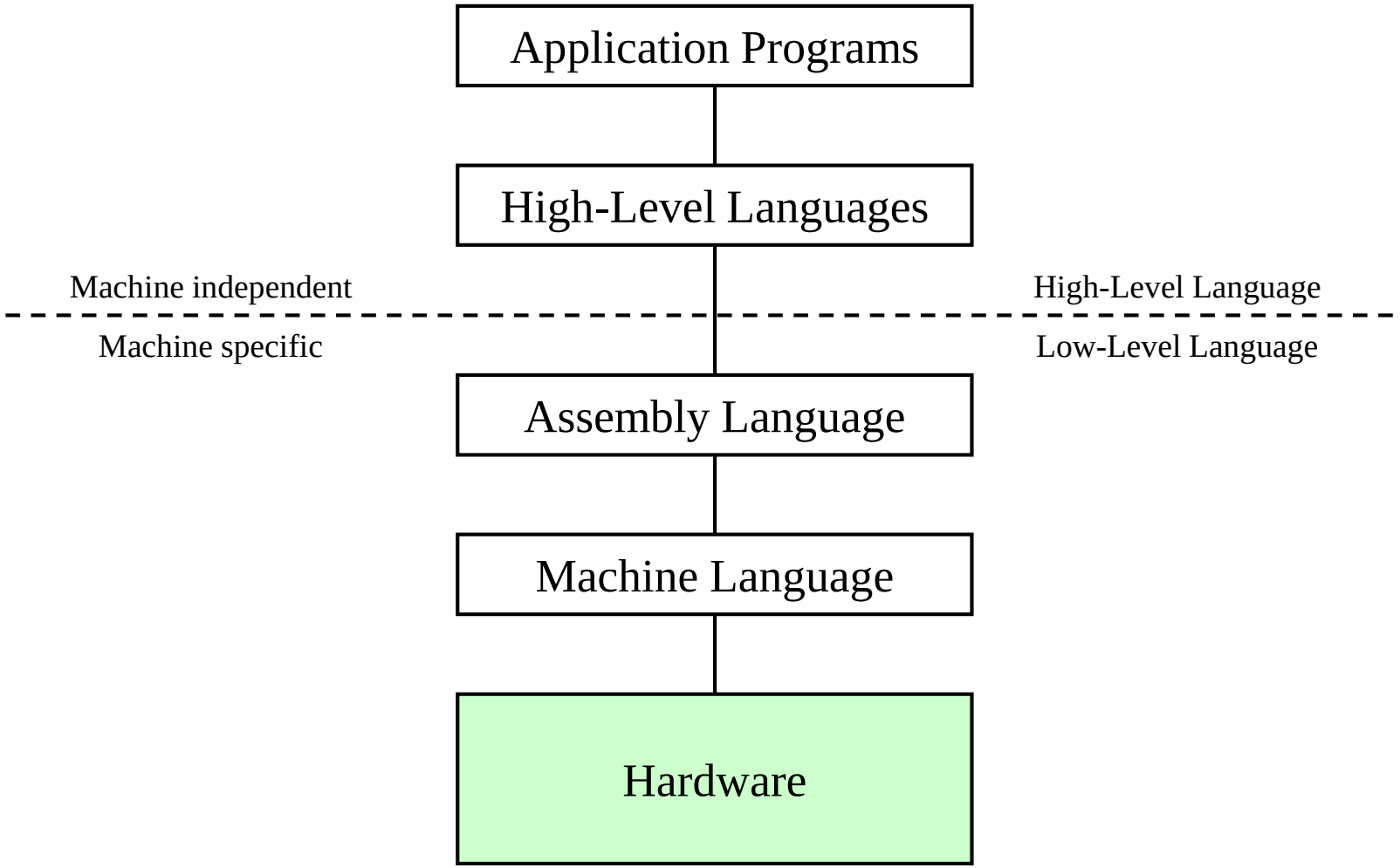
- 机器语言
  - 用二进制码 “0”和 “1”表示，能被计算机直接识别和执行。
  - 编写、调试程序都很繁琐，编写的程序可移植性差，但执行速度很快。

- 汇编语言
  - 是一种面向机器的符号语言，使用指令助记符、地址标号等编写程序，需编译、链接转换成机器代码（.exe文件或.com文件）， 才能执行，运行速度快。
  - 可移植性差，功能不强，但非常利于计算机底层硬件的操作，代码效率也很高。

- 高级语言
  - 是类似于自然语言和数学描述语言的程序设计语言，编写的程序同样需编译、链接转换成机器代码（.exe文件或.com文件），才能执行，运行效率不及汇编语言编写的程序。
  - 可移植性好、易于理解，对底层硬件进行操作时编程复杂。
- 高级语言和汇编语言最终都要由其编译器编译成机器语言，才能被计算机执行。

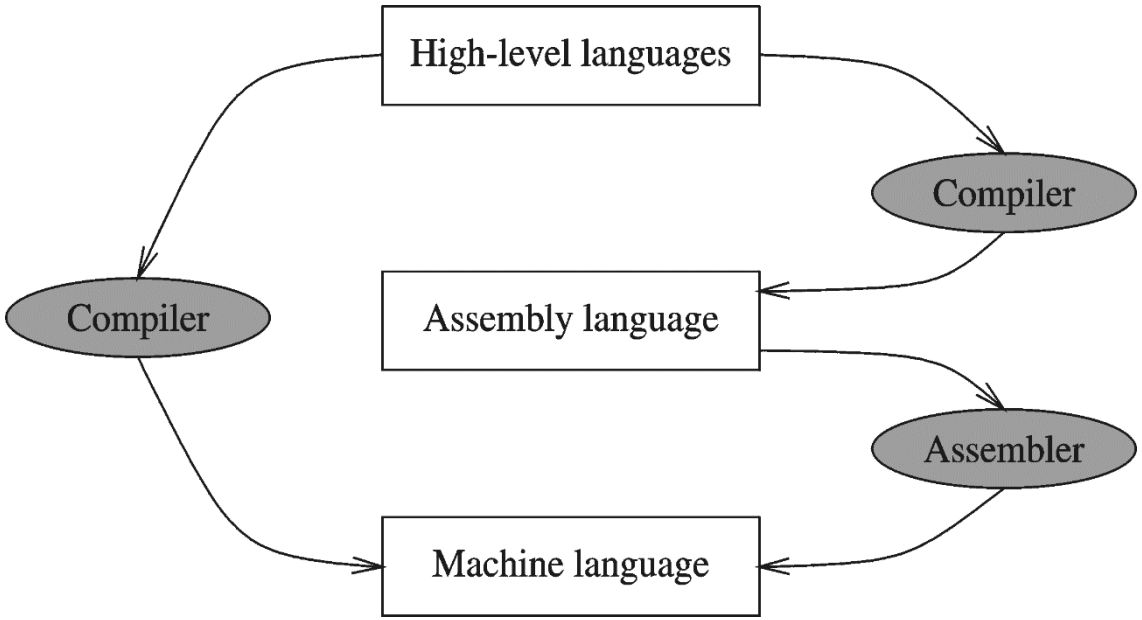
1.3 计算机工作原理

- 计算机语言
  - 机器语言
  - 汇编语言
  - 高级语言



1.3 计算机工作原理

- 计算机语言
  - 机器语言
  - 汇编语言
  - 高级语言



汇编程序(Assembler)：汇编语言源程序→机器语言目标程序

编译程序(Complier)：高级语言源程序→汇编/机器语言目标程序

Program (C Language):

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

MIPS Assembly Language:

```
muli    $2,  $5, 4
add     $2,  $4, $2
lw      $15, 0($2)
lw      $16, 4($2)
sw      $16, 0($2)
sw      $15, 4($2)
jr      $31
```

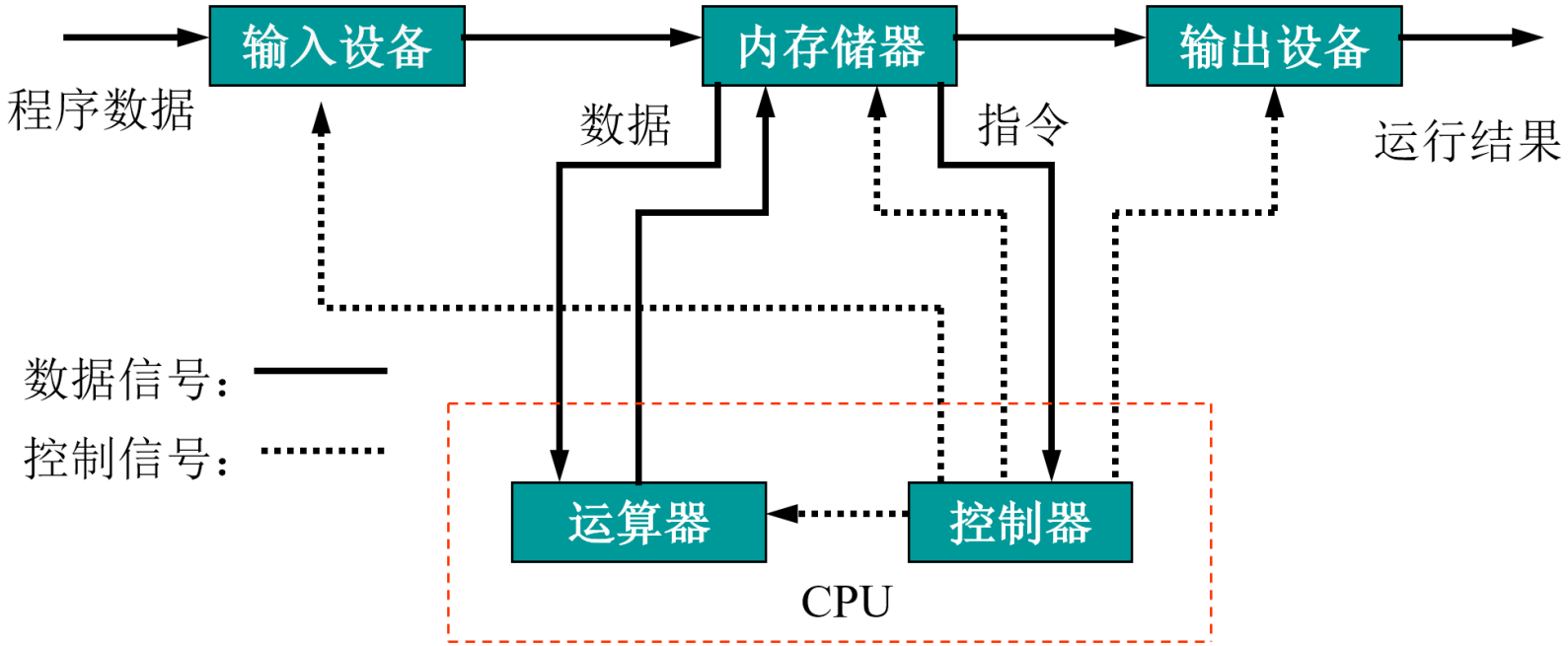
Assembler

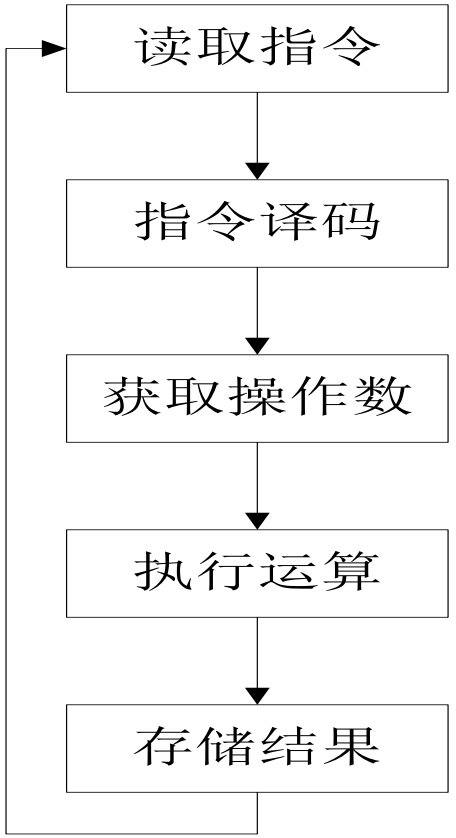
MIPS Machine Language:

```
01A10018
00181821
8C620000
8CF20004
ACF20000
AC320004
03E00008
```

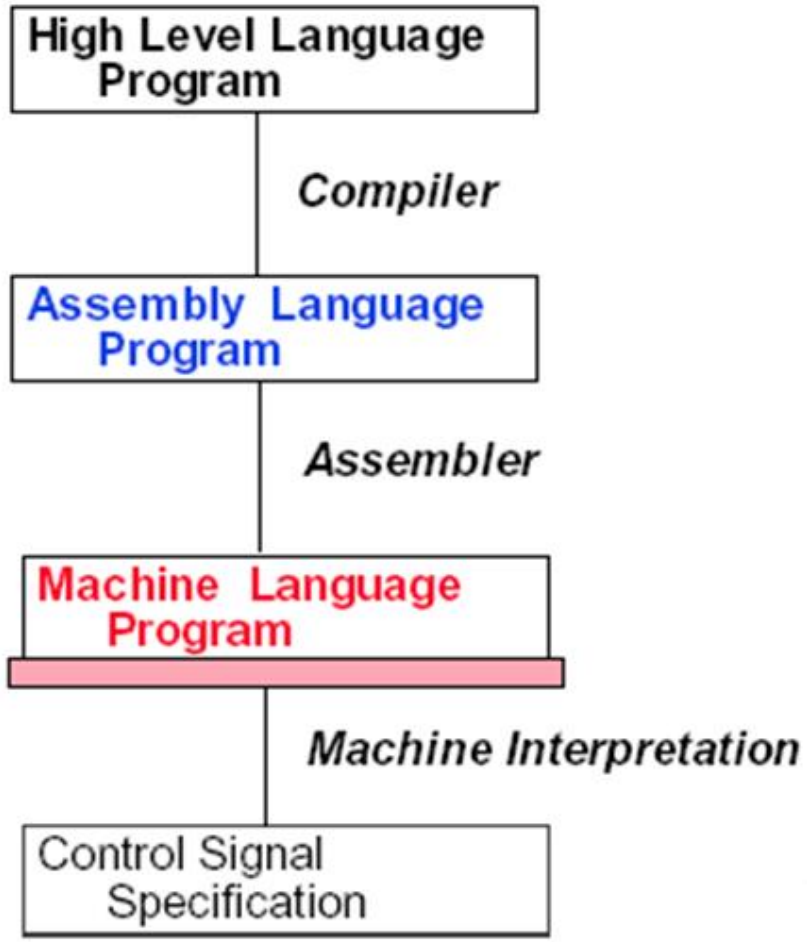
1.3 计算机工作原理

- 冯·诺依曼提出的“存储程序和程序控制”思想：
  - 计算机工作的基本原理就是：首先将问题程序化，形成指令序列，然后将它存入存储器中，再由CPU的控制器从存储器中逐条读取指令，然后译码，获取要操作的数据，执行运算，最后再将结果存到寄存器或存储器。





- 冯·诺依曼提出的“存储程序和程序控制”思想
  - MIPS指令执行过程举例



C语言中的数据交换语句:

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

MIPS汇编指令:

```
lw      $15, 0($2)
lw      $16, 4($2)
sw      $16, 0($2)
sw      $15, 4($2)
```

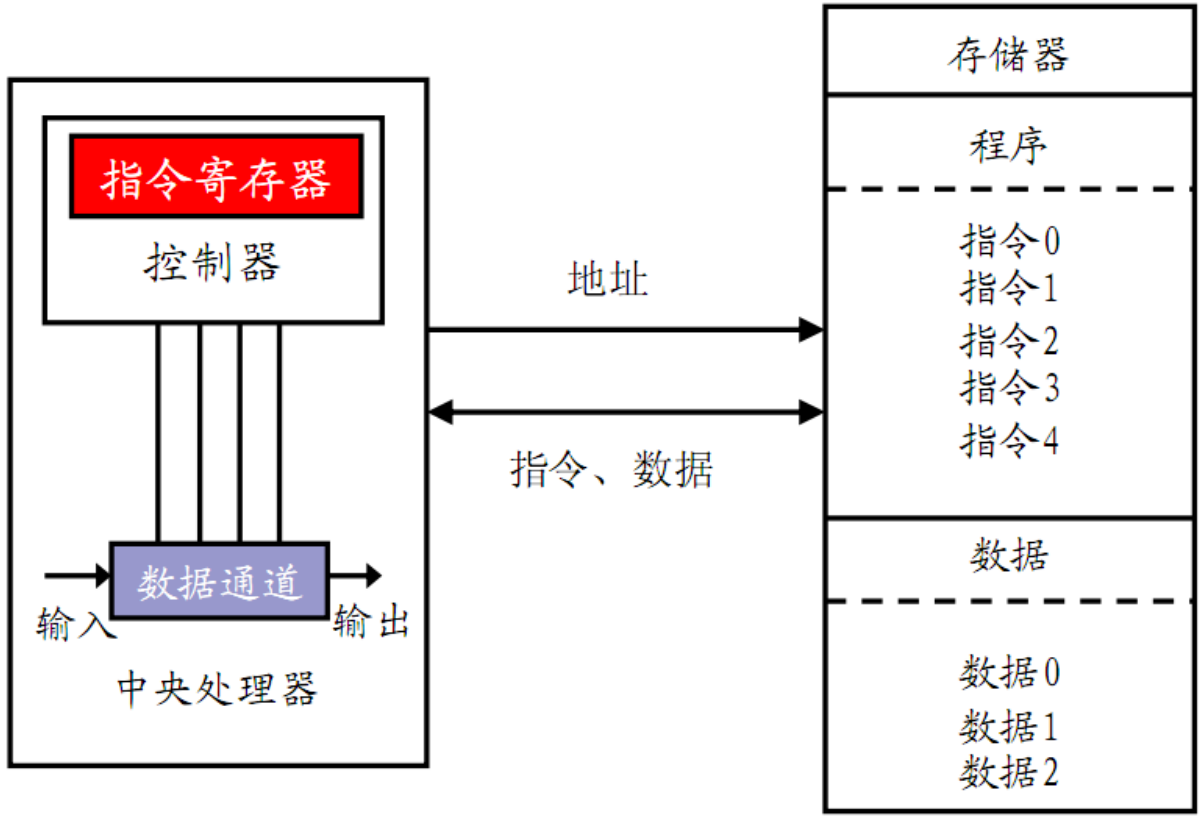
MIPS机器码:

1000 1100 0100 1111 0000 0000 0000 0000
1000 1100 0101 0000 0000 0000 0000 0100
1010 1100 0101 0000 0000 0000 0000 0000
1010 1100 0100 1111 0000 0000 0000 0100

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

1.4 计算机系统的结构模型

- 冯诺依曼体系结构
  - 也称普林斯顿体系结构
  - 程序指令和数据采用统一的存储器
  - 对程序和数据的寻址只能交替进行
  - 代表：
    - Intel公司的x86处理器
    - Freescale（NXP）的HCS08
    - ARM公司的ARM7
    - MIPS
  - 特点：
    - 结构简单，易于实现，成本低
    - 传输效率低

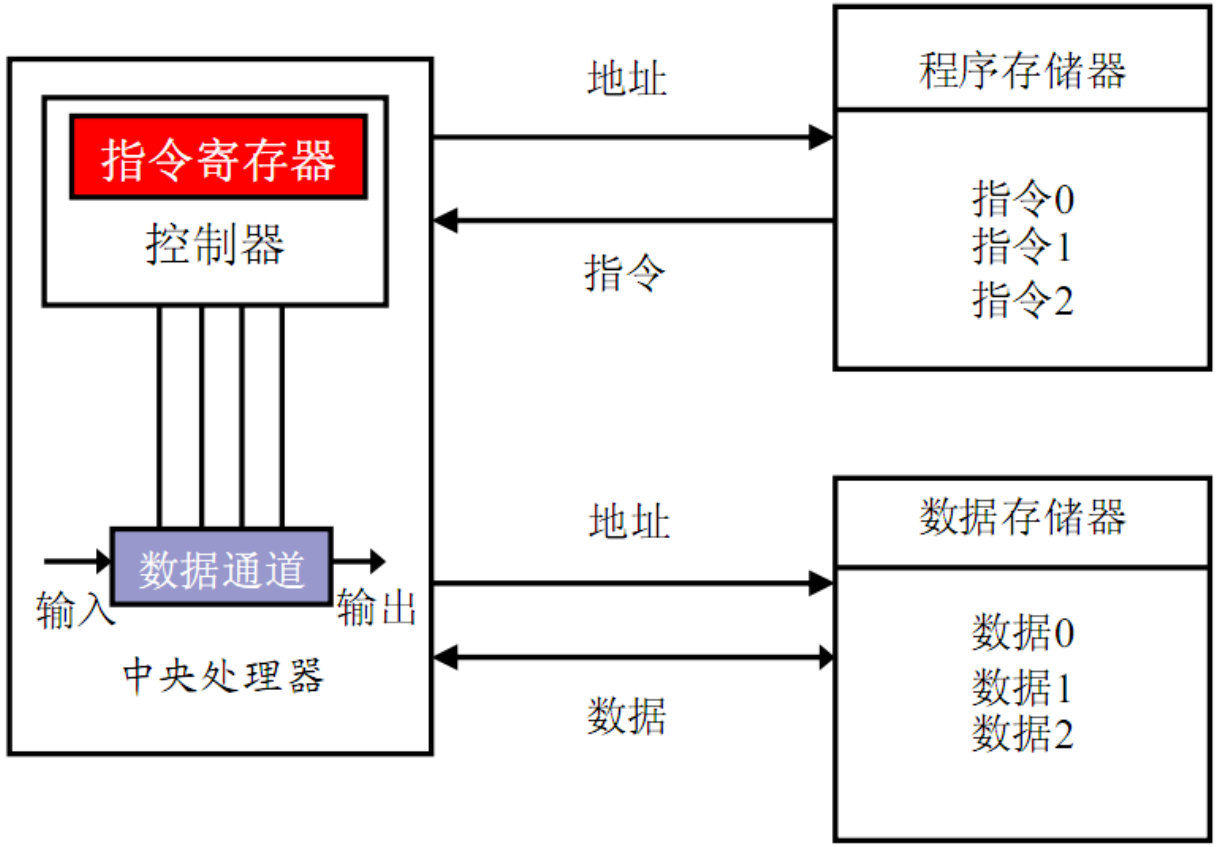


1.4 计算机系统的结构模型

▶ 哈佛体系结构

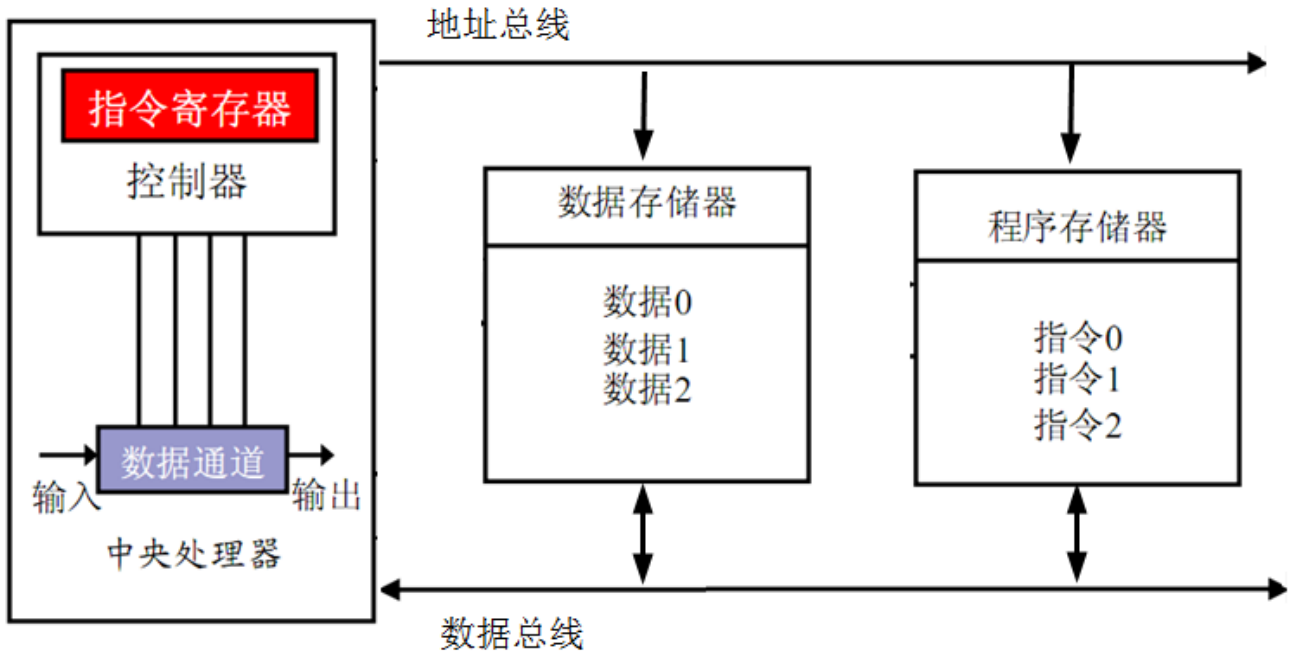
- 程序存储器和数据存储器分开独立编址
- CPU使用两套独立的存储总线
- 程序效率高
- 代表：
  - Zilog公司的Z8系列
  - Microchip公司的PIC系列芯片
  - ARM公司的ARM9、ARM11

- 特点：
  - 传输效率高
  - 结构复杂，扩展外围设备不易



▶ 哈佛体系结构

- 改进的哈佛结构
  - 使用两个独立的存储器模块，分别存储指令和数据，每个存储模块都不允许指令和数据并存；
  - 具有一条独立的地址总线和一条独立的数据总线，利用公用地址总线访问两个存储模块（程序存储模块和数据存储模块），公用数据总线则被用来完成程序存储模块或数据存储模块与CPU 之间的数据传输；
  - 两条总线由程序存储器和数据存储器分时共用。



第1章 作业（一）

- 作业题（P26）
  - 1 . 电子计算机的发展经历了 哪几个阶段? 各个阶段分别具有什么特点？
  - 2. 计算机硬件系统由 哪5 个部分构成? 简述各部分的主要作用 。
  - 3 . 微处理器一般由 哪几部分构成? 各部分分别起什么作用？

- 要求：
  - 在微助教中提交。

Agenda

- 内容
  - 计算机系统的发展历史
  - 计算机系统的基本结构
  - 计算机系统的基本工作方式
  - 计算机系统的结构模型
  - 计算机系统中的数据、信息的表示方式、存储方式
  - 数据运算基础
- 目的
  - 掌握计算机的基本构成、基本工作原理
  - 掌握不同结构模型计算机系统的特点
  - 掌握不同数制之间的转换、数据在计算机系统中的表示方式、数据的存储格式（大字节序、小字节序）
  - 掌握数据运算基础知识（符号数、无符号数、定点数、浮点数的加减运算规则）

1.5 计算机中的信息表示

▶ 数制及其转换

- 二、十、十六进制
  - 计算机为电子器件，以器件的物理状态来表示数，内部使用二进制；
  - 二进制书写、记忆都不方便，故用十六进制数表示二进制；
  - 为符合人们日常习惯，便于人机交流，因而使用十进制数。
- 不同进位制数书写约定（前缀/后缀）：
  - 数后带D或不带任何符号则为十进制数；
  - 带后缀B为二进制数；
  - 带后缀H/前缀0x为十六进制数。
    - 如: 100、100D或(100)10 ，即一百；
      100B、 (100)2 ，即四；
      100H、 (100)16，0x100，即256。

Hexadecimal Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

▶ 数制及其转换

- 1）“K”进制数转为十进制数： “按权展开”

K进制数 $m_{n-1}m_{n-2}\ldots m_1m_0 . m_1m_2\ldots m_p$ (n位整数，p位小数）对应的十进制数：

$$L=m_{n-1}K^{n-1}+m_{n-2}K^{n-2}+\ldots+m_0K^0+m_1K^{-1}+m_2K^{-2}+\ldots+m_pK^{-p} \quad ,(K=2,8,16)$$

如: 11001.11B $=1\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$

$=16+8+0+0+1+0.5+0.25$

$=25.75$

BCD.AH $=11\times16^2+12\times16^1+13\times16^0+10\times16^{-1}$

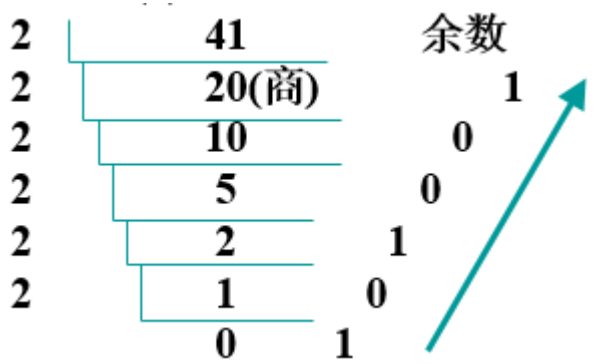
$=11\times256+192+13+10\times0.0625$

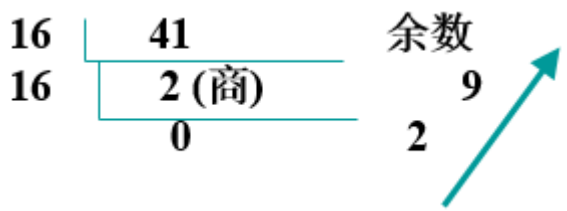
$=3021.625$

练习： BBH = $(?)_{10}$

0111 0111.1100B = $(?)_{10}$

- 数制及其转换
  - 2）十进制数转为“K”进制数
    - 整数部分：“除K取余法”，将十进制数除以K，得到一个商和余数。再将商除以K，又得到一个新的商和余数。如此继续下去，直到商等于0为止。将所得各次余数，以最后余数为最高位，最先得的余数为最低位，倒序排列，就是K进制的转换结果。
      例如：41 = (?) B





所以，41 = 0010 1001B = 29H

验算：10 1001B = 32 + 8 + 1 = 41， 29H = 2 x 16 + 9 = 41

▶ 数制及其转换

- 2）十进制数转为“K”进制数
  - 纯小数部分：“乘K取整法”，先对十进制纯小数乘以K，然后对积的纯小数部分再乘以K，依此继续下去，直到满足所要求的精度或积的小数部分为0终止。把每次得到的整数部分由上（靠近小数点处）而下（远离小数点处）依次正序排列起来，即得所求的K进制的纯小数部分。

例如：$0.125=(?)_2$

乘	纯小数部分	整数部分
2 × 0.125	0.25	0
2× 0.25	0.5	0
2× 0.5	0.0	1

∴ 0.125=0.001B

$0.125=(?)_{16}$

乘	纯小数部分	整数部分
16 × 0.125	0.0	2

∴ 0.125=0.2H

练习：0.75= （？）B =（？）H

▶ 数制及其转换

- 2）十进制数转为“K”进制数
  - 纯小数部分： “乘K取整法”
    - 一个有限的十进制小数并不一定能够用一个有限的二/十六进制小数来表示，由此产生的存储、运算误差——机器误差

如　　0.7=0.1011 0011 (0011 0011…)B

0.7 (* 2)

.4	1	.4	0	.4	0
.8	0	.8	0	.	
.6	1	.6	1	.	
.2	1	.2	1	.	

若保留小数点后8位，0.7 = 0.1011 0011B

= 0.5+0.125+0.0625+0.0078125+0.00390625

= 0.69921875

Δ=|0.7-0.69921875|=0.00078125

▶ 数制及其转换

- 3）二进制转换成十六进制
  - 二进制整数部分由小数点向左，每4位一分，最后不足4位的前面补0；小数部分由小数点向右，每4位一分，最后不足4位的后面补0，然后把每4位二进制数用相应的一位十六进制数代替，便可转换为十六进制数。

    例如： 1101101.101B=(?)H

    0110 1101.1010 =6D.A H

- 4）十六进制转换为二进制
  - 方法： 一位十六进制数用四位二进制数代替

    例如：1D.A8 H=(?)B

    1 D . A 8

    0001 1101 .1010 1000

    ∴ 0.1D.A8H=1 1101 1010.1B

- 二、十六互换，不存在运算的麻烦，仅仅是一种简单的替换关系

▶ 整数的表示

- 符号数：将“+”、“-”号连同数值部分一块编码，以最高位作为符号位，“0”表示“+”，“1”表示“-”，其余位代表数值位，这样的数为符号数

- 无符号数：最高位也代表数值的数

- 机器数（计算机中的数）的位数通常为8的倍数，如：
  - 8 位：Byte，字节
  - 16位：HalfWord，半字
  - 32位：Word，字（MIPS中，字为32b；Intel X86中，字为16位）
  - 64位：Double Word，双字
- 机器数所代表的实际数值，称为机器数的真值；
- 整数在计算机中采用补码表示。

▶ 整数的表示

- 补码求法
  - 正数的原码、补码相同
    - 原码：最高位为符号位，0表示正数，1表示负数，数值位是它的绝对值，这样构成的数码称原码。
  - 负数的补码：从正数原码的最低位向最高位扫描，保留直到第一个“1”的所有位，以后各位（包括符号位）按位取反。

例如　[+4]原 =0000 0100B　　[– 4]原=1000 0100B　　(八位)

[– 4]补=1111 1100B = FCH

求：-4的16位补码？

解：　[+4]原=0000 0000 0000 0100B

根据求补码规则，

[– 4]补=1111 1111 1111 1100B=FFFCH

▶ 整数的表示

- 如何从补码求其所对应的十进制数（真值）？

1、首先根据最高位（8/16/32）判断补码所表示的是正数还是负数：0正1负

0111 1111B　正数

1111 0000B　负数

2、如是正数，按“权”展开即可。

01111111B = 0+64+32+16+8+4+2+1= + 127

3、如是负数，从补码的最低位向最高位扫描，保留直到第一个“1”的所有位，以后各位按位取反，再将所得的数按“权”展开，即得到该补码所对应的负数的绝对值。（注意和“从一个正数的原码求其补码”的法则区分）

1111 0000B

0001 0000B　= 1*16 = 16

所以，　1111 0000B　= –16

“求补”操作：最低位向最高位扫描，保留直到第一个“1”的所有位，以后各位按位取反；“求补”操作不是“求补码”

1.5 计算机中的信息表示

▶ 小数的表示

- 浮点数：小数点不固定的数
- 定点数：小数点固定的数
- 浮点/定点对比
  - 表示的精度与范围不同
    - 一般说来，定点数表示的精度较低，表示的数值范围也小；而浮点数表示范围大，精度也高
  - 计算机中运算的效率不同
    - 一般说来，定点数的运算在计算机中实现起来比较简单，效率较高；而浮点数的运算在计算机中实现起来比较复杂，效率相对较低。
  - 硬件依赖性
    - 一般说来，只要有硬件提供运算部件，就会提供定点数运算的支持，但不一定支持浮点数运算，如有的很多嵌入式开发板就不提供浮点运算的支持。

► 小数的表示

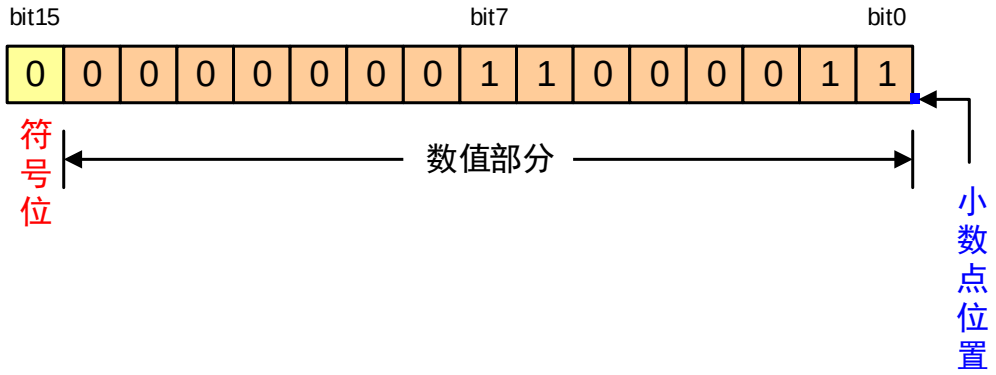
- 定点数的表示与存储
  - 定点小数若用16位二进制表示，最高位是符号位，那么有效位就是15位。小数点之后可以有0 - 15位。我们把小数点之后有n位叫做Qn，例如小数点之后有12位叫做Q12格式的定点小数，而Q0就是我们所说的整数
  - Qn越大，数的范围越小，但精度越高；Qn越小，数的范围越大，但精度越低；

Q 表示法	S 表示法	小数点位置	整数位	小数位	十进制数表示范围	精度
Q0	S15.0	在 D0 之后	15	0	-32768 ≦ x ≦ 32767	2^{0}
Q1	S14.1	在 D1 之后	14	1	-16384 ≦ x ≦ 16383.5	2^{-1}
Q2	S13.2	在 D2 之后	13	2	-8192 ≦ x ≦ 8191.75	2^{-2}
Q3	S12.3	在 D3 之后	12	3	-4096 ≦ x ≦ 4095.875	2^{-3}
Q4	S11.4	在 D4 之后	11	4	-2048 ≦ x ≦ 2047.9375	2^{-4}
Q5	S10.5	在 D5 之后	10	5	-1024 ≦ x ≦ 1023.96875	2^{-5}
Q6	S9.6	在 D6 之后	9	6	-512 ≦ x ≦ 511.984375	2^{-6}
Q7	S8.7	在 D7 之后	8	7	-256 ≦ x ≦ 255.9921875	2^{-7}
Q8	S7.8	在 D8 之后	7	8	-128 ≦ x ≦ 127.99609375	2^{-8}
Q9	S6.9	在 D9 之后	6	9	-64 ≦ x ≦ 63.998046875	2^{-9}
Q10	S5.10	在 D10 之后	5	10	-32 ≦ x ≦ 31.9990234375	2^{-10}
Q11	S4.11	在 D11 之后	4	11	-16 ≦ x ≦ 15.99951171875	2^{-11}
Q12	S3.12	在 D12 之后	3	12	-8 ≦ x ≦ 7.999755859375	2^{-12}
Q13	S2.13	在 D13 之后	2	13	-4 ≦ x ≦ 3.9998779296875	2^{-13}
Q14	S1.14	在 D14 之后	1	14	-2 ≦ x ≦ 1.99993896484375	2^{-14}
Q15	S0.15	在 D15 之后	0	15	-1 ≦ x ≦ 0.999969482421875	2^{-15}

▶ 小数的表示

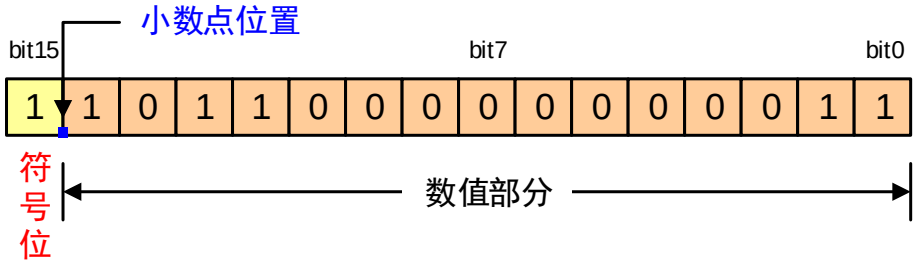
- 定点数的表示与存储

十进制整数195的Q0格式定点表示：0x00c3



Q 表示法	S 表示法	小数点位置	整数位	小数位	十进制数表示范围	精度
Q0	S15.0	在 D0 之后	15	0	-32768 ≦ x ≦ 32767	2^{0}
Q1	S14.1	在 D1 之后	14	1	-16384 ≦ x ≦ 16383.5	2^{-1}
Q2	S13.2	在 D2 之后	13	2	-8192 ≦ x ≦ 8191.75	2^{-2}
Q3	S12.3	在 D3 之后	12	3	-4096 ≦ x ≦ 4095.875	2^{-3}
Q4	S11.4	在 D4 之后	11	4	-2048 ≦ x ≦ 2047.9375	2^{-4}
Q5	S10.5	在 D5 之后	10	5	-1024 ≦ x ≦ 1023.96875	2^{-5}
Q6	S9.6	在 D6 之后	9	6	-512 ≦ x ≦ 511.984375	2^{-6}
Q7	S8.7	在 D7 之后	8	7	-256 ≦ x ≦ 255.9921875	2^{-7}
Q8	S7.8	在 D8 之后	7	8	-128 ≦ x ≦ 127.99609375	2^{-8}
Q9	S6.9	在 D9 之后	6	9	-64 ≦ x ≦ 63.998046875	2^{-9}
Q10	S5.10	在 D10 之后	5	10	-32 ≦ x ≦ 31.9990234375	2^{-10}
Q11	S4.11	在 D11 之后	4	11	-16 ≦ x ≦ 15.99951171875	2^{-11}
Q12	S3.12	在 D12 之后	3	12	-8 ≦ x ≦ 7.999755859375	2^{-12}
Q13	S2.13	在 D13 之后	2	13	-4 ≦ x ≦ 3.9998779296875	2^{-13}
Q14	S1.14	在 D14 之后	1	14	-2 ≦ x ≦ 1.99993896484375	2^{-14}
Q15	S0.15	在 D15 之后	0	15	-1 ≦ x ≦ 0.999969482421875	2^{-15}

十进制纯小数-0.6876的定点表示：0xd803



▶ 小数的表示

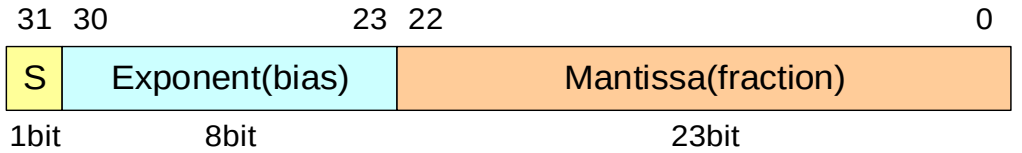
- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
  - IEEE 754 specifies three types or Formats of floating-point numbers:
    - Single (Fortran's REAL*4, C's float), (Obligatory),
    - Double (Fortran's REAL*8, C's double), (Ubiquitous), and
    - Double-Extended (Fortran REAL*10+, C's long double), (Optional).

Table of Formats' Parameters:

Format	Bytes	K+1	N
Single	4	8	24
Double	8	11	53
Double-Extended	≥ 10	≥ 15	≥ 64
(Quadruple	16	15	113)

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



- 一个规格化的32位（单精度）浮点数x的真值：

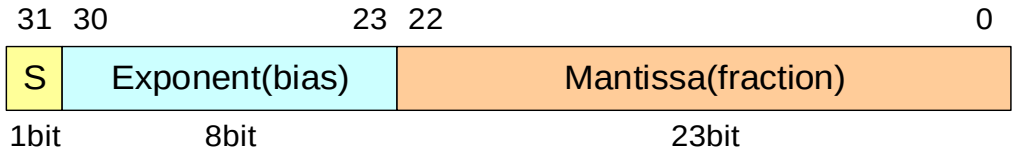
$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$

- 一个规格化的64位（双精度）浮点数x的真值：

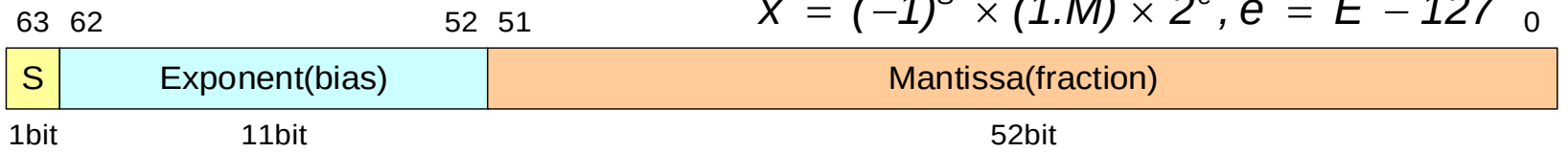
$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$

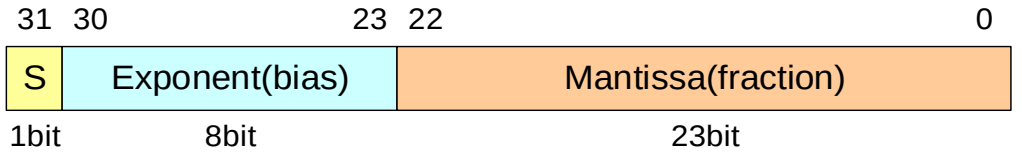


$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

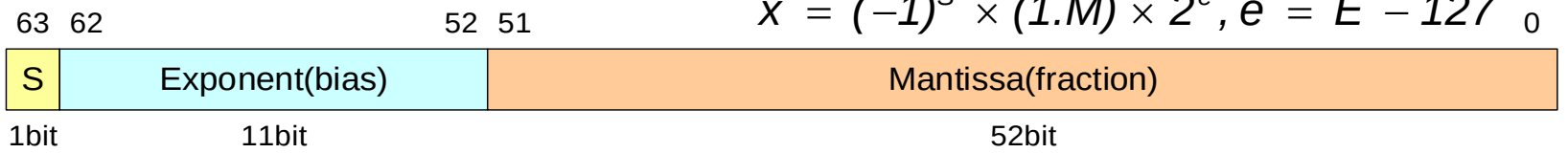
- S: 符号位
  - ▶S=0表示正数，S=1表示负数
- E: 阶码
  - ▶采用移码E来表示正负指数， E=e+127（32位、单精度） or E=e+1023 （64位、双精度）
- M: 尾数
  - ▶尾数域表示的值是1.M（二进制）
  - ▶即尾数域最高有效位应为1（但这个1并不存储），否则要修改阶码同时左右移小数点，使其变成满足这一要求的表示形式——浮点数的规格化表示

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

  - 如十进制数-9.625的32位浮点数格式

    解：9.625 = 1001.101B = 1.001101 X 2^3, e = 3

    得：S = 1（负数），E = 3+127 = 130 = 1000 0010，M = 001101B

    所以，x = 1100 0001 0001 1010 0000 0000 0000 0000B = 0xC11A0000
  - 练习：十进制数20.59375的32位浮点数格式？

    解：20.59375 = 10100.10011B = 1.010010011 X 2^4, e = 4

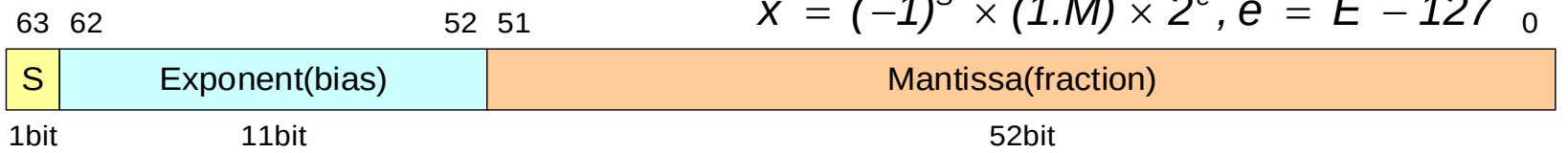
    得：S = 0，E = 4+127 = 131，M = 010010011B

    所以，x = 0100 0001 1010 0100 1100 0000 0000 0000B = 0x41A4C000

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

  - 如32位浮点数41360000H所表示的十进制数

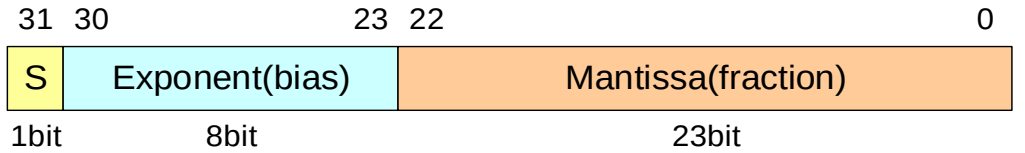
    解：41360000H = 0100 0001 0011 0110 0000 0000 0000 0000B

    得：S = 0，E = 10000010B = 130，M = 1.011011B，e = E-127 = 3

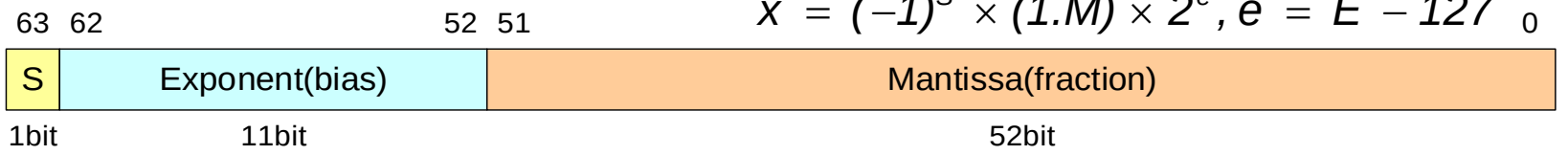
    所以，x = 1.011011B X 2^3 = 1011.011B = 11.375
  - 练习：32位浮点数BE200000H、64位浮点数4052A00000000000所表示的十进制数？

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

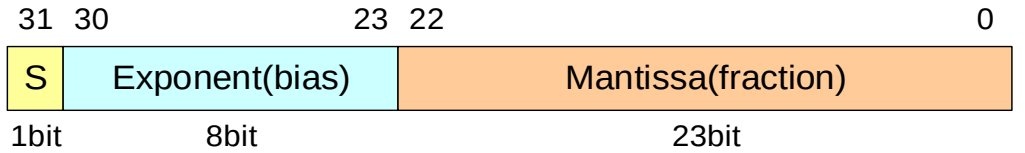
  - 练习：32位浮点数BE200000H、64位浮点数4052A00000000000所表示的十进制数？

解：❶ BE200000H = 1011 1110 0010 0000 0000 0000 0000 0000B
- 符号域S = 1，指数域E = 0111 1100B = 124，尾数域M = 1.010B，e = E-127 = -3
- 所以，x = - 1.010B X 2^{-3} = - 1.25 X 2^{-3} = - 0.15625

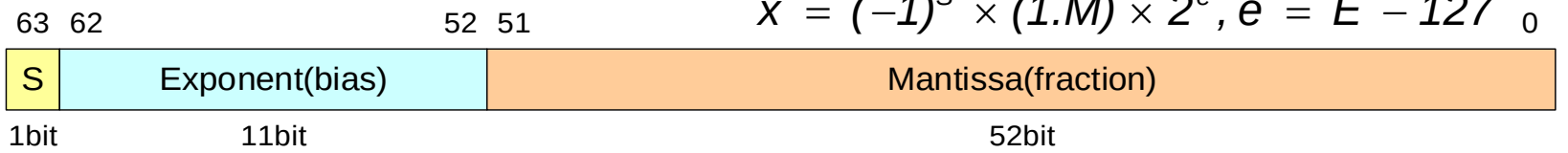
❷ 4052A00000000000 H = 0100 0000 0101 0010 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000B
- 符号域S = 0，E = 100 0000 0101B = 1029，M = 1.0010 1010B，e = E-1023 = 6；
- 所以，x = 1.0010 1010B X 2^6 = 74.5

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$

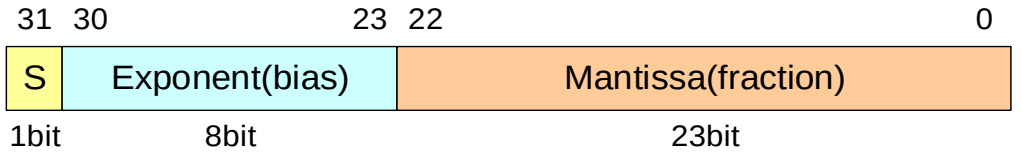


$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

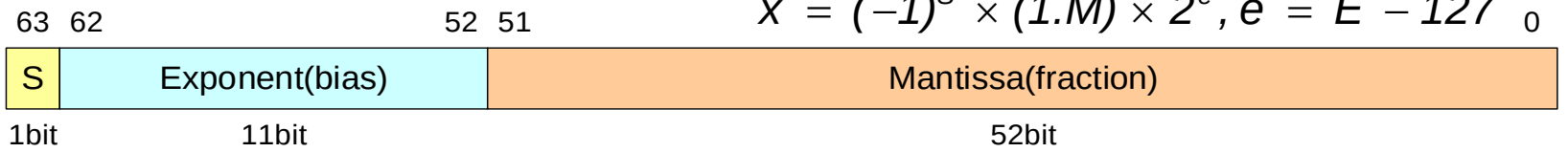
- 特殊值（E全0/1）
  - Zero(0)
    - 指数域 E = 0 and 尾数域 M(F) = 0
    - 可通过符号域S来表示+0 and –0
  - Infinity
    - 当E为最大值（全1）、M = 0时，值为无穷
      - ▶单精度时，8b指数域E最大值: E = 255
      - ▶单精度时，11b指数域E最大值: E = 2047
    - 溢出、除数为0时的结果为无穷值
    - 可通过符号域S来表示+ ∞ and –∞

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



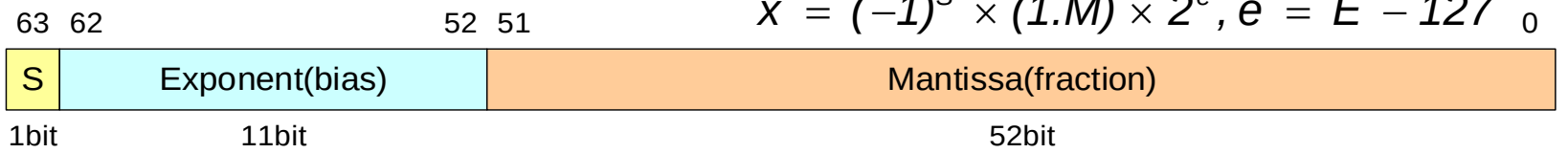
- 特殊值（E全0/1）
  - NaN (Not a Number)
    - NaN is a special value represented with maximum E and M ≠ 0
    - Result from exceptional situations, such as 0/0 or sqrt(negative)
    - Operation on a NaN results is NaN: Op(X, NaN) = NaN

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

- 表示范围
  - 最大值、最小值？

解：❶ 单精度时的正数最大值：0111 1111 0111 1111 1111 1111 1111 1111B（E全0/1的都已有特殊用途）

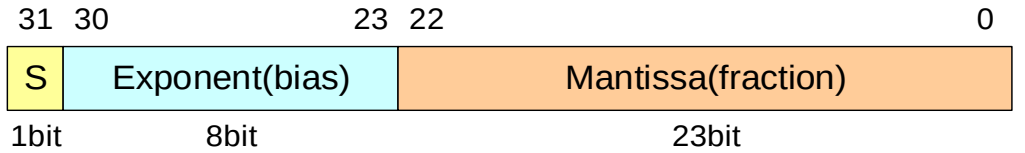
- 符号域S = 0，指数域E = 1111 1110B = 254， e = E-127 = 127，尾数域M = 1.FFFFFEH，
- 所以，x = 1.FFFFFEH X 2^{127} = (2- 2^{-24}) X 2^{127}

❷ 单精度时的正数最小值：0000 0000 1000 0000 0000 0000 0000 0000B（E全0/1的都已有特殊用途）

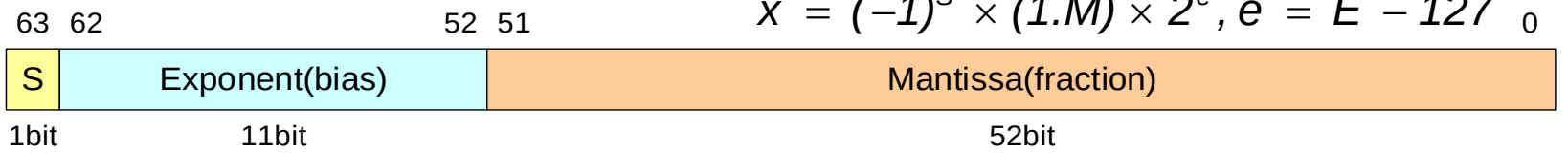
- 符号域S = 0，指数域E = 0000 0001B = 1， e = E-127 = -126，尾数域M = 1.0B，
- 所以，x = 1.0B X 2^{-126} = 2^{-126}

▶ 小数的表示

- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

- 表示范围
  - 最大值、最小值？

解：❶ 单精度时的负数最小值：1111 1111 0111 1111 1111 1111 1111 1111B（E全0/1的都已有特殊用途）

- 符号域S = 1，指数域E = 1111 1110B = 254， e = E-127 = 127，尾数域M = 1.FFFFFEH，
- 所以，x = -1.FFFFFEH X 2^{127} = -(2- 2^{-24}) X 2^{127}

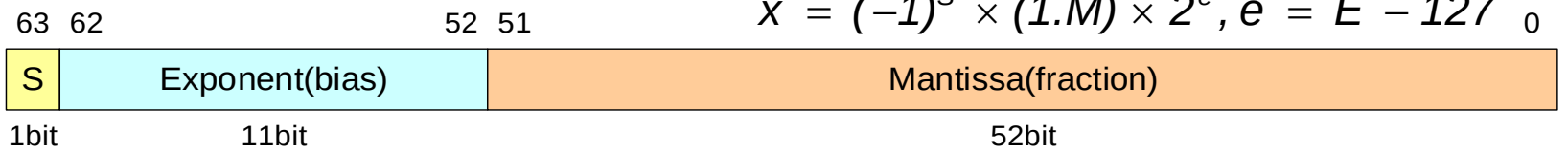
❷ 单精度时的负数最大值：1000 0000 1000 0000 0000 0000 0000 0000B（E全0/1的都已有特殊用途）

- 符号域S = 1，指数域E = 0000 0001B = 1， e = E-127 = -126，尾数域M = 1.0B，
- 所以，x = -1.0B X 2^{-126} = -2^{-126}

▶ 小数的表示

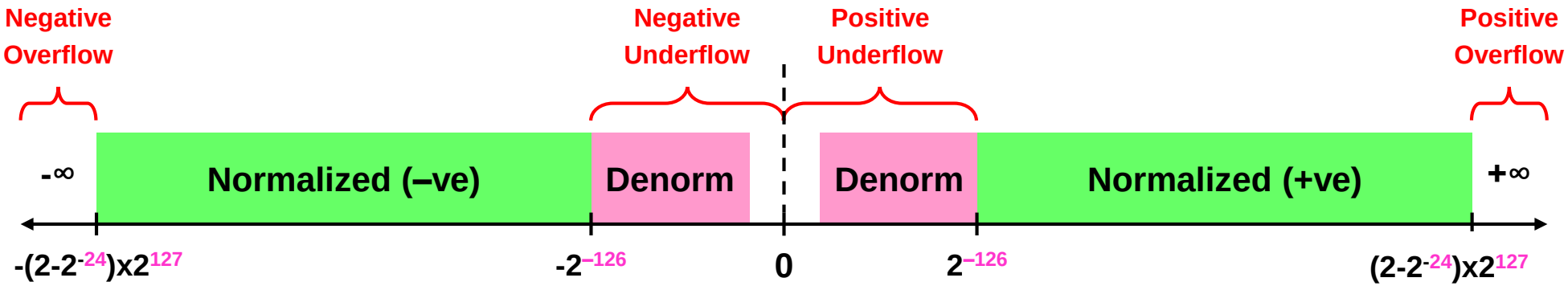
- 浮点数的表示与存储
  - IEEE Standard 754 for Binary Floating-Point Arithmetic
    - IEEE 754 32位浮点数和64位浮点数格式

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$



$$x = (-1)^S \times (1.M) \times 2^e, e = E - 1023$$

- 表示范围
  - 最大值、最小值？



1.5 计算机中的信息表示

▶ 字符的表示

- 计算机如何能够处理各种字符？
  - 字符和字节类型数据之间建立一一对应，处理数据相当于处理字符
  - 数就叫做字符的ASCII码
    - printable characters(20H to 7EH)
      - ▶‘0’ to ‘9’：30H to 39H
      - ▶‘A’ to ‘Z’：41H to 5AH
      - ▶‘a’ to ‘z’：61H to 7AH
      - ▶空格(space)：20H

ASCII（American Standard Code for Information Interchange）

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
20	space	30	0	40	@	50	P	60	‘	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	”	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	’	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o		

printable characters

1.5 计算机中的信息表示

▶ 字符的表示

- 计算机如何能够处理各种字符？
  - 字符和字节类型数据之间建立一一对应，处理数据相当于处理字符
  - 数就叫做字符的ASCII码
    - control characters(00H to 1FH、7FH)
      - ▶00h:NULL character ⇒ used to terminate a string
      - ▶0AH:Line Feed (LF), 换行
      - ▶0DH:Carriage Return (CR), 回车
      - ▶7FH: Delete (DEL)
- 思考：12，34，0123，2012的ASCII?

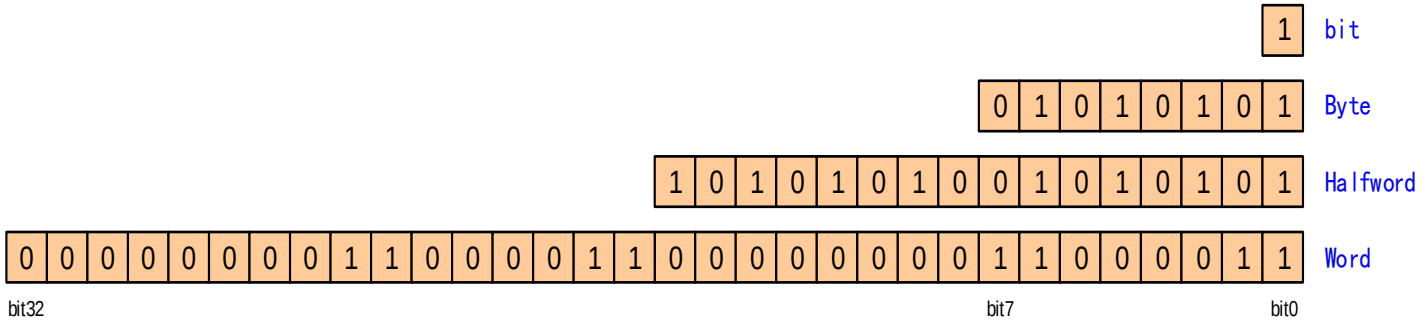
解答：
12_{ASCII} = 3132H
34_{ASCII} = 3334H
0123_{ASCII} = 30313233H
2012_{ASCII} = 32303132H

ASCII（American Standard Code for Information Interchange）

dec	hex	Char	dec	hex	Char	dec	hex	Char	dec	hex	Char
0	00	null	32	20	sp	64	40	@	96	60	'
1	01	soh	33	21	!	65	41	A	97	61	a
2	02	stx	34	22	"	66	42	B	98	62	b
3	03	etx	35	23	#	67	43	C	99	63	c
4	04	eot	36	24	$	68	44	D	100	64	d
5	05	enq	37	25	%	69	45	E	101	65	e
6	06	ack	38	26	&	70	46	F	102	66	f
7	07	bel	39	27	'	71	47	G	103	67	g
8	08	bs	40	28	(	72	48	H	104	68	h
9	09	ht	41	29	)	73	49	I	105	69	i
10	0a	nl	42	2a	*	74	4a	J	106	6a	j
11	0b	vt	43	2b	+	75	4b	K	107	6b	k
12	0c	np	44	2c	,	76	4c	L	108	6c	l
13	0d	cr	45	2d	-	77	4d	M	109	6d	m
14	0e	so	46	2e	.	78	4e	N	110	6e	n
15	0f	si	47	2f	/	79	4f	O	111	6f	o
16	10	dle	48	30	0	80	50	P	112	70	p
17	11	dc1	49	31	1	81	51	Q	113	71	q
18	12	dc2	50	32	2	82	52	R	114	72	r
19	13	dc3	51	33	3	83	53	S	115	73	s
20	14	dc4	52	34	4	84	54	T	116	74	t
21	15	nak	53	35	5	85	55	U	117	75	u
22	16	syn	54	36	6	86	56	V	118	76	v
23	17	etb	55	37	7	87	57	W	119	77	w
24	18	can	56	38	8	88	58	X	120	78	x
25	19	em	57	39	9	89	59	Y	121	79	y
26	1a	sub	58	3a	:	90	5a	Z	122	7a	z
27	1b	esc	59	3b	;	91	5b	[	123	7b	{
28	1c	fs	60	3c	<	92	5c	\	124	7c	|
29	1d	gs	61	3d	=	93	5d	]	125	7d	}
30	1e	rs	62	3e	>	94	5e	^	126	7e	~
31	1f	us	63	3f	?	95	5f	_	127	7f	del

▶ 数的存储

- 数据类型
  - bit /位 (b)
  - Byte / 字节 (8 bits, B)
  - Halfword / 半字 (16 bits, H)
  - Word / 字 (32 bits, W)
  - Doubleword / 双字 (64 bits, D)



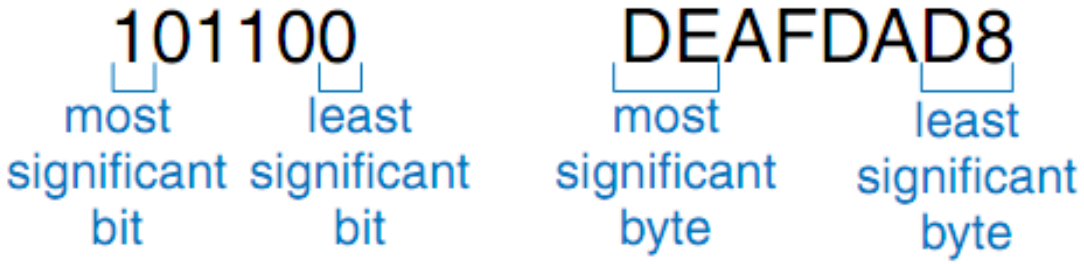
- 每种类型的数据都有范围

Storage Type	Unsigned Range	Powers of 2
Byte	0 to 255	0 to $(2^8 - 1)$
Half Word	0 to 65,535	0 to $(2^{16} - 1)$
Word	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Double Word	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

Storage Type	Unsigned Range	Powers of 2
Byte	–128 to +127	-2^7 to $(2^7 - 1)$
Half Word	–32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Word	–2,147,483,648 to +2,147,483,647	-2^{31} to $(2^{31} - 1)$
Double Word	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

▶ 数的存储

- MSB和LSB
  - MSB： Most significant bits / bytes，最高有效位/最高有效字节
  - LSB：Least significant bits / bytes，最低有效位/最低有效字节



- 数的存储
  - Big-endian和Little-endian
  - 在计算机中，内存可寻址的最小存储单位是字节
    - 多字节数（halfword、word、doubleword）存放在内存时存在字节顺序的问题，即高位字节在前(低地址)，还是低位字节在前(低地址)？
      - PowerPC/HCS08系列中高位字节存放在低地址，即所谓Big endian方式（高对低、低对高）
      - x86系列中则最低位字节存放低地址，即所谓Little endian方式（高对高、低对低）
      - 有些处理器同时支持两种方式，具体由软件在CPU复位后设定（如MIPS R3000）

Figure 6.4 Big-endian and little-endian data storage

Figure 6.4 shows how big- and little-endian machines store the value 0x23456789 in memory word 0. After the load byte instruction, lb $s0, 1($0), $s0 would contain 0x00000045 on a big-endian system and 0x00000067 on a little-endian system.

1.5 计算机中的信息表示

▶ 数的存储

- Big-endian和Little-endian
- 在计算机中，内存可寻址的最小存储单位是字节
  - 多字节数（halfword、word、doubleword）的地址
    - Big endian方式（高对低、低对高）：以MSB所在的地址作为整个数的地址
    - Little endian方式（高对高、低对低）：以LSB所在的地址作为整个数的地址

字数据0x23456789的地址是MSB（23）所在的地址：0

字数据0x23456789的地址是LSB（89）所在的地址：0

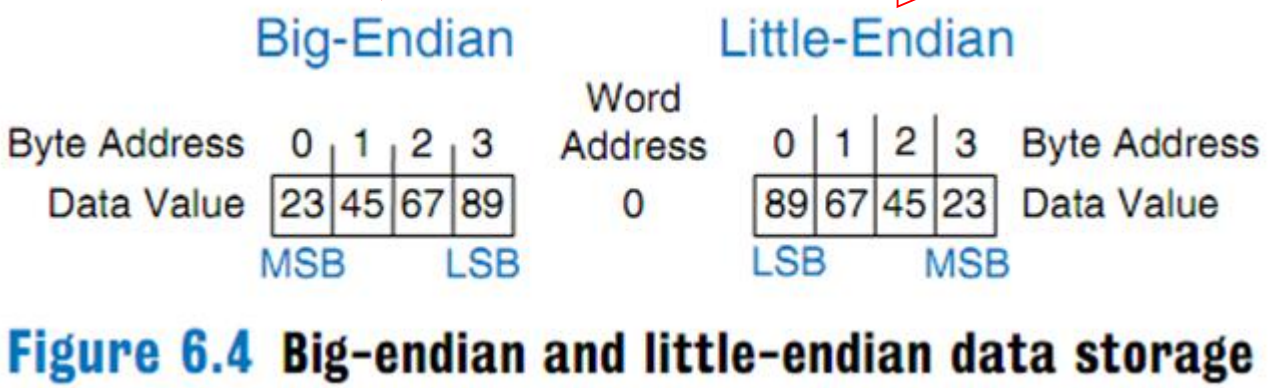


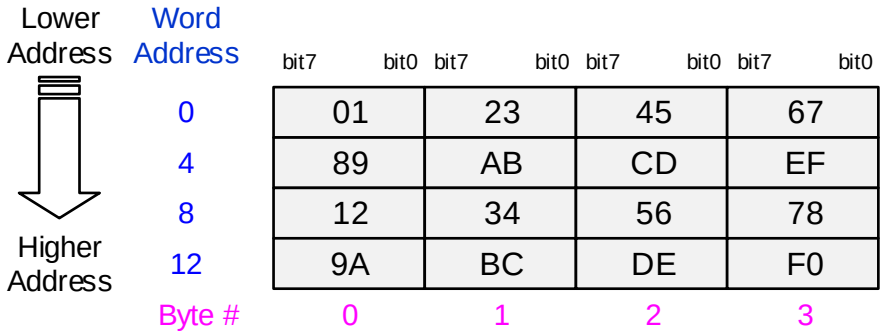
Figure 6.4 Big-endian and little-endian data storage

Figure 6.4 shows how big- and little-endian machines store the value 0x23456789 in memory word 0. After the load byte instruction, lb $s0, 1($0), $s0 would contain 0x00000045 on a big-endian system and 0x00000067 on a little-endian system.

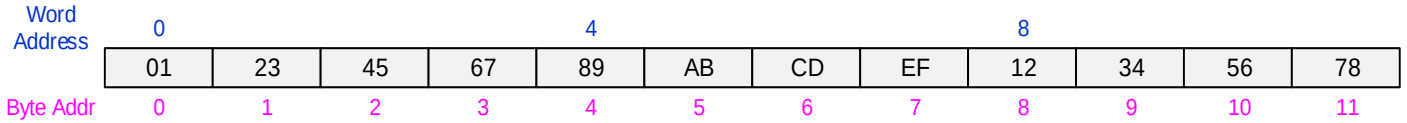
1.5 计算机中的信息表示

▶ 数的存储

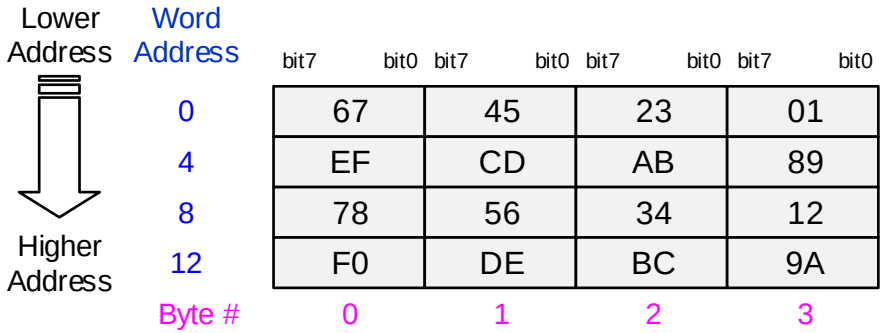
- 0x01234567、0x89ABCDEF、0x12345678、0x9ABCDEF0存入0、4、8、12地址
- Big-endian：高对低、低对高；顺序存放；



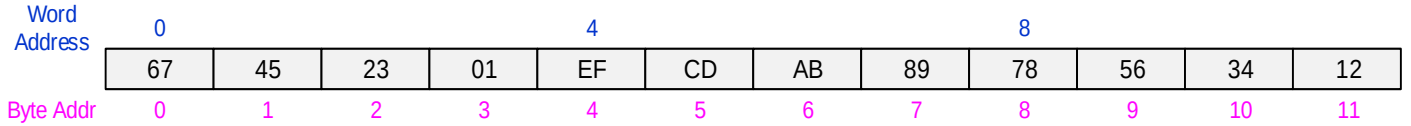
MIPS R3000 processor capabilities, such as cache size, multiprocessor interface, "big-endian" or "little-endian" byte ordering, can be configured immediately after processor reset



- Little-endian：高对高、低对低；如：PC



MIPS processors can operate with either *big-endian* or *little-endian* byte order. SPIM operates with both byte orders. SPIM's byte order is the same as the byte order of the underlying machine that runs the simulator. For example, on a Intel 80x86, SPIM is little-endian, while on a Macintosh or Sun SPARC, SPIM is bigendian.

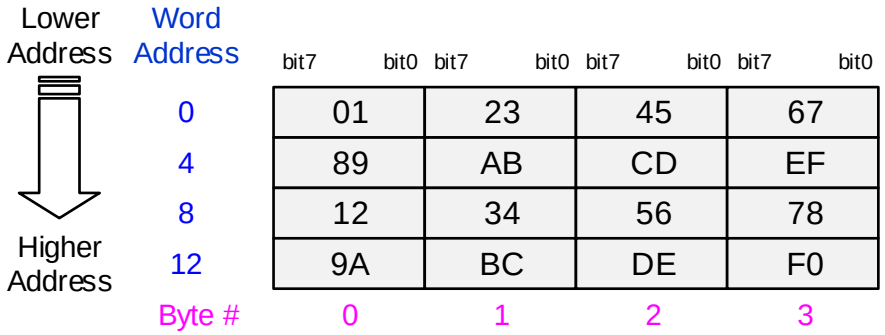


【注意】课本例子都是基于Big endian方式讲解，但是QtSpim运行于X86上，是little-endian —— 验证比较“蹩脚”

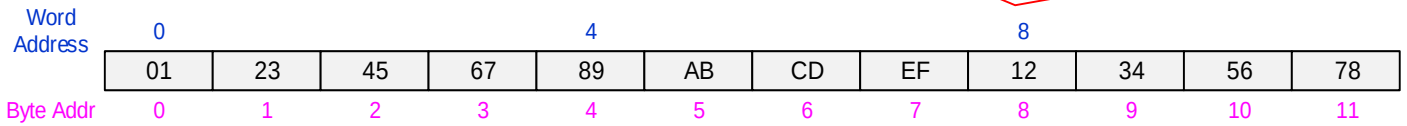
1.5 计算机中的信息表示

▶ 数的存储

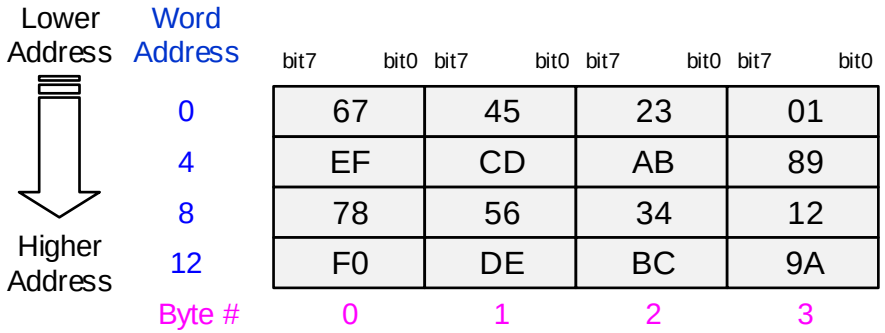
- 0x01234567、0x89ABCDEF、0x12345678、0x9ABCDEF0存入0、4、8、12地址
- Big-endian：高对低、低对高；顺序存放；



Big endian方式（高对低、低对高）：以MSB所在的地址作为整个数的地址：0、4、8、12



- Little-endian：高对高、低对低；如：PC



Little endian方式（高对高、低对低）：以LSB所在的地址作为整个数的地址：0、4、8、12

第1章 作业（二）

作业题（P26）

- 5 . 存储器存取数据的最小单位是什么？存储器的 一般结构 包含哪几部分? 向 存储器存、 取数据分别叫 什么操作？
- 6 . I/O 接口 的主要功能是什么？举例说明 计算机系统中 具有哪些常见I/O 接口 。
- 附1：1.34, 345.6的单精度浮点表示？（最终结果写成16进制）

要求：

- 微助教提交；
- 下次课之前提交；
- 严格要求自己。

Agenda

- 内容
  - 计算机系统的发展历史
  - 计算机系统的基本结构
  - 计算机系统的基本工作方式
  - 计算机系统的结构模型
  - 计算机系统中的数据、信息的表示方式、存储方式
  - 数据运算基础
- 目的
  - 掌握计算机的基本构成、基本工作原理
  - 掌握不同结构模型计算机系统的特点
  - 掌握不同数制之间的转换、数据在计算机系统中的表示方式、数据的存储格式（大字节序、小字节序）
  - 掌握数据运算基础知识（符号数、无符号数、定点数、浮点数的加减运算规则）

▶ 无符号数算数运算

- 两个无符号数相加，由于两个加数均为正数，因此其和也是正数。当和超过其位数所允许的范围时，就向更高位进位，有些CPU内部用进位标志位CF表示。

```
   A4H + 3BH (164+59)
       1010 0100
  +    0011 1011
  ------------------
     0 1101 1111
无进位，CF = 0
```

```
   7EH + C3H (126+195)
       0111 1110
  +    1100 0011
  ------------------
     1 0100 0001
有进位，CF = 1
```

- 两个8/16/32位的无符号数相加，无论有无进位，其和是一个9/17/33位的无符号数（CF为最高位），若要求其所对应的十进制数，只需按权展开。

```
0A4H+3BH = 0 1101 1111B = 0DFH = 13 x 16 + 15 = 223
         = 164 + 59
7EH+0C3H = 1 0100 0001B = 141H =1 x 256 + 4 x 16+1 = 321
         = 126 +195
```

▶ 无符号数算数运算

- 两个8/16/32位的无符号数相减，若被减数大于等于减数，则无借位(CF=0)，结果为正，是一个9/17/33位（CF为最高位）的无符号数，要求其所对应十进制数，只需要按权展开。

```
  A4H - 3BH (164-59)
     1010 0100
-    0011 1011
  ---------------
   0 0110 1001
无借位，CF = 0
```

```
真值：
069H = 6 x 16 + 9 = + 105
```

- 两个8/16/32位的无符号数相减，若被减数小于减数，则有借位(CF=1)，结果为负，是一个9/17/33位（CF为最高位）的补码表示的负数，要求其所对应十进制数，需要对该补码求真值。

```
  7EH - C3H (126-195)
     0111 1110
-    1100 0011
  ---------------
   1 1011 1011
有借位，CF = 1
```

```
真值：
1BBH = - 045H = - 69
```

▶ 无符号数算数运算

- 进位/借位标志位CF
  - 是同一个标志位：加法时为进位标志，减法时为借位标志；
  - 有些CPU内部的寄存器（“状态标志寄存器”）中专门用一个位来表示CF，如X86
  - 有些CPU内部没有此标志位，需要用户自己注意，如MIPS

- 思考：采用二进制完成下列无符号数的运算，并指出是否产生了进位或借位。

  1)0x34+0x98　2) x034-0x98　3)0x1345+0xffff　4)0x1345-0xffff

解答：1)CF=0　2)CF=1
3)CF=1　4)CF=1

```
    34H + 98H
    0011 0100
+   1001 1000
  0 1100 1100
无进位，CF=0
```

```
    1345H + ffffH
    0001 0011 0100 0101
+   1111 1111 1111 1111
  1 0001 0011 0100 0100
有进位，CF=1
```

▶ 有符号数算数运算

- 符号位也参与运算，即把符号数当作无符号数来运算；
- 两个8/16/32位有符号数算术运算结果超出了8/16/32位的表示范围，称之为溢出；
- 在有些型号CPU的内部，会专门用一个溢出标志位OF位来表示两个符号数的运算是否产生了溢出：OF=0，无溢出；OF=1，有溢出。

```
13H + 7EH = 19 + 126
          = 145 > 127
有溢出，OF = 1
```

```
13H - 7EH = 19 - 126
          = -107 > -128
无溢出，OF = 0
```

Storage Type	Unsigned Range	Powers of 2
Byte	–128 to +127	-2^7 to $(2^7 - 1)$
Half Word	–32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Word	–2,147,483,648 to +2,147,483,647	-2^{31} to $(2^{31} - 1)$
Double Word	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

▶ 有符号数算数运算

- 溢出判断方法一：将符号数转换成相应的十进制数进行运算，判断运算结果是否超出8/16/32位的表示范围，超过有溢出，没有超过无溢出。

```
33H + 5AH = 51 + 90
          = 141 > 127
有溢出，OF = 1
```

```
29H - FCH = 41 - (-4)
          = 45 < 127
无溢出，OF = 0
```

- 练习：1）0x34 + 0x98　　2）0x34 – 0x98

```
34H + 98H = 52 + (-104)
          = -52 > -128
无溢出，OF = 0
```

```
34H – 98H = 52 – (-104)
          = 156 > 127
有溢出，OF = 1
```

▶ 有符号数算数运算

- 溢出判断方法二：设符号位向更高位进位为CY，数值部分向符号部分进位为CS ，则溢出：(需要将减法通过**求补**换成加法运算)　　OF= CY ⊕ CS
  - 说明： ⊕ 为“异或”运算符：不同为1，相同为0
  - （ 1⊕0 = 0⊕1 = 1　　1⊕1 = 0⊕0 = 0 ）

```
      33H + 5AH
      0011 0011
  +   0101 1010
  ------------
    0 1000 1101
```

CS=1,CY=0,OF=1,有溢出

```
     29H-FCH
     0010 1001          0010 1001
 -   1111 1100   ➡   +  0000 0100
                    ------------
                    0 0010 1101
```

CS=0,CY=0,OF=0,无溢出

- 练习：1）34H + 98H

```
      0011 0100
  +   1001 1000
  ------------
      1100 1100
```

CS=0,CY=0,OF=0,无溢出

2）34H – 98H

```
      0011 0100          0011 0100
  -   1001 1000   ➡   +  0110 1000
                     ------------
                     0 1001 1100
```

CS=1, CY=0,OF=1,有溢出

$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$

▶ 浮点数运算

- 操作过程

  1）0 操作数检查

  2）比较阶码大小，并完成对阶：指数对其，向指数大的对齐，尾数右移，小数点左移

  3）尾数求和运算　　4）结果规范化　　5）舍入处理　　6）溢出处理

- 例1.5 试用单精度浮点数计算(1.0+123456.789e30)+(-123456.789e30)和1.0+(123456.789e30+(-123456.789e30))的结果

解：

- 1.0表示为单精度浮点数为：0x3f80 0000
  - 0 011 1111 1 000 0000 0000 0000 0000 0000；阶码域0x7f，尾数域0，符号0
- 123456.789e30表示为单精度浮点数为：0x79be371e
  - 0 111 1001 1 011 1110 0011 0111 0001 1110；阶码域0xf3，尾数域0x3e371e，符号0
- -123456.789e30表示为单精度浮点数为：0xf9be371e
  - 1 111 1001 1 011 1110 0011 0111 0001 1110；阶码域0xf3，尾数域0x3e371e，符号1

▶ 浮点数运算

- 操作过程

1）0 操作数检查

2）比较阶码大小，并完成对阶：指数对其，向指数大的对齐，尾数右移，小数点左移

3）尾数求和运算　　4）结果规范化　　5）舍入处理　　6）溢出处理

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$

- (1.0 + 123456.789e30) + (-123456.789e30)
  - (1.0 + 123456.789e30) ：
    - 1.0阶码域0x7f，123456.789e30阶码域0xf3，1.0尾数需右移0xf3-0x7f=0x74=116位才能实现阶码对齐，此时小数点左侧的1根据舍入原则已经丢弃，因此尾数1.0在移位之后变为0
    - 加法运算之后和的尾数即123456.789e30尾数域0x3e371e
    - 得到的规范化结果为123456.789e30：0x79be371e
  - 123456.789e30+ (-123456.789e30)
    - 数据仅符号位不同，尾数运算时直接抵消
    - 规范化之后的结果为0
- 注意：此时计算机得到的结果为错误的结果

▶ 浮点数运算

- 操作过程

1）0 操作数检查

2）比较阶码大小，并完成对阶：指数对其，向指数大的对齐，尾数右移，小数点左移

3）尾数求和运算　　4）结果规范化　　5）舍入处理　　6）溢出处理

$$x = (-1)^S \times (1.M) \times 2^e, e = E - 127$$

- 1.0 + (123456.789e30 + (- 123456.789e30))
  - 123456.789e30+ (-123456.789e30)
    - 数据仅符号位不同，尾数运算时直接抵消
    - 规范化之后的结果为0
  - (1.0 + 0) ：
    - 0为特殊数值，不需要进行对阶，直接得到结果1.0
- 此结果为正确结果
- 当参与运算的两数阶码相差较大，尾数移位可能带来较大误差

▶ 逻辑运算

- 运算规则
  - 与（AND、&、 ×）：多位相与，见0为0，全1为1；
  - 或（OR、|、+）：多位相或，见1为1，全0为0；
  - 非（NOT、/）：1变0，0变1；
  - 异或（XOR、 ⊕ ）；两位异或，不同为1，相同为0
- 如：

```
1&0=0&1=0&0=0        1&1=1
1&1&0=1&0&1=0&0&1=1&1&0=0
&1&1=0               1&1&1=1
0&0&0&0=1&1&0&1=1&0&1&0=0
```

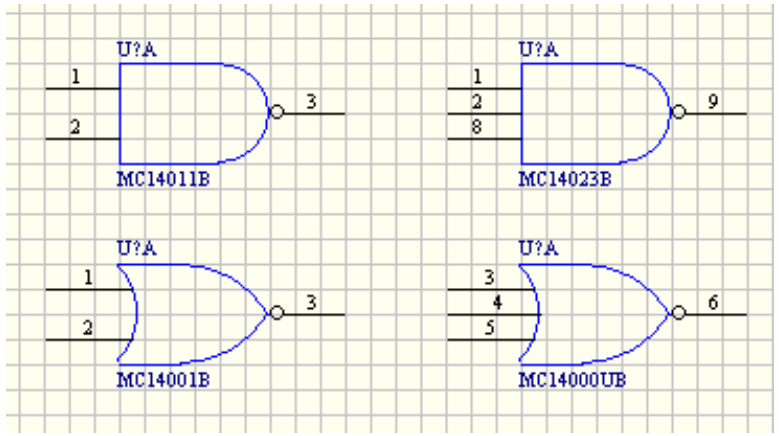
```
1|0=0|1=1|1=1        0|0=0
1|1|0=1|0|1=0|0|1=1|1|0=0|1|1=1
0|0|0=0
0|0|0|1=1|1|0|1=1|0|1|0=1
```

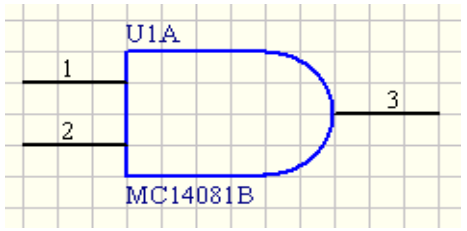
```
/1=0                 /0=1
```

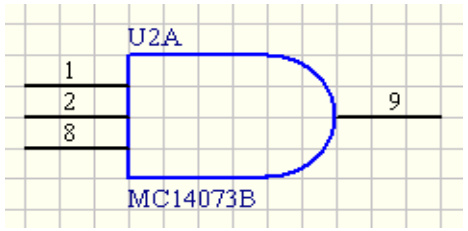
```
1⊕0=0⊕1=1            0⊕0=1⊕1=0
```

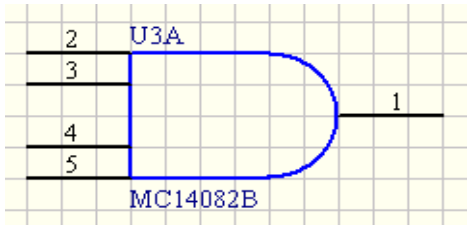
▶ 逻辑运算

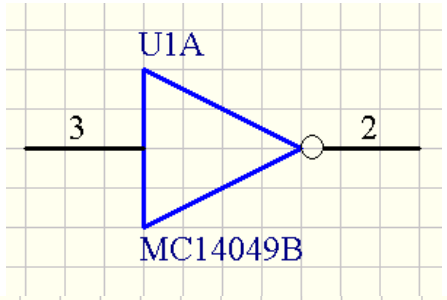
- 运算规则
  - 与（AND、&、 ×）：多位相与，见0为0，全1为1；
  - 或（OR、|、+）：多位相或，见1为1，全0为0；
  - 非（NOT、/）：1变0，0变1；
  - 异或（XOR、 ⊕ ）；两位异或，不同为1，相同为0
- 在数字电路中，有相应的集成电路实现相应的逻辑功能：

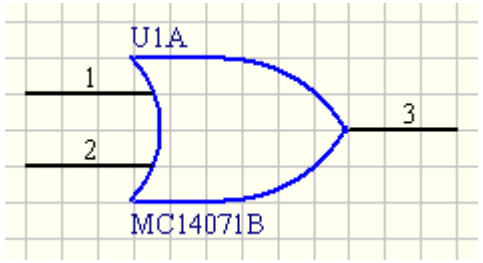


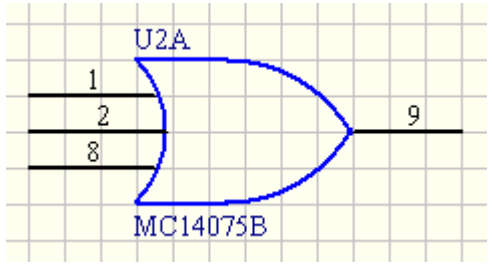


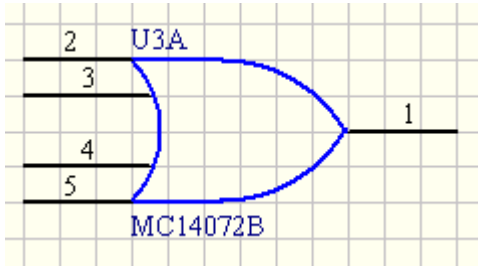


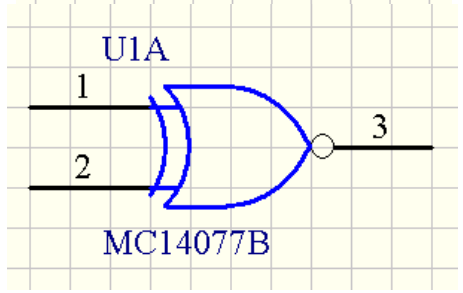












▶ 逻辑运算

- 多个二进制数之间的与、或、异或运算只需按位各自运算即可
- 如：

```
      F0H & 5AH
      1111 0000
  &   0101 1010
  ---------------
      0101 0000
所以，F0H & 5AH = 50H
```

```
      F0H | 5AH
      1111 0000
  |   0101 1010
  ---------------
      1111 1010
所以，F0H | 5AH = FAH
```

```
      F0H ⊕ 5AH
      1111 0000
  ⊕   0101 1010
  ---------------
    0 1010 1010
所以，F0H⊕5AH = AAH
```

- 与运算时，与0的位被清0，与1的位保持不变
- 或运算时，或0的位保持不变，或1的位被置1
- 异或运算时，异或0的位保持不变，异或1的位被取反

1.7 C语言数据类型的含义

▶ 典型的数据类型对应32位机的数据位宽

- 例1. 6某大字节序的计算机系统内存中存放有如表1- 6所示数据，试指出地址0x1234 5678表示的char，short int，int，float，double型数据符号数和无符号数的十进制值？

地址	字节0	字节1	字节2	字节3	字节4	字节5	字节6	字节7
0x1234 5678	0x81	0x82	0x83	0x84	0x85	0x86	0x87	0x88

数据类型	字节数
char	1
short int	2
int	4
long	4
char *	4
float	4
double	8

- 整型数据转换为十进制结果

多字节数（halfword、word、doubleword）的地址
Big endian方式（高对低、低对高）：以MSB所在的地址作为整个数的地址
Little endian方式（高对高、低对低）：以LSB所在的地址作为整个数的地址

数据类型	16进制值	10进制值
Char	0x81	-127
Unsigned char	0x81	129
Short int	0x8182	-32382
Unsigned short	0x8182	33154
int	0x81828384	-2122153084
unsigned	0x81828384	2172814212

1.7 C语言数据类型的含义

- 典型的数据类型对应32位机的数据位宽

数据类型	字节数
char	1
short int	2
int	4
long	4
char *	4
float	4
double	8

  - 例1. 6某大字节序的计算机系统内存中存放有如表1- 6所示数据，试指出地址0x1234 5678表示的char，short int，int，float，double型数据符号数和无符号数的十进制值？

地址	字节0	字节1	字节2	字节3	字节4	字节5	字节6	字节7
0x1234 5678	0x81	0x82	0x83	0x84	0x85	0x86	0x87	0x88

  - Float型结果

    - Float型的16进制为0x81828384
      - 二进制为：1000 0001 1000 0010 1000 0011 1000 0100B
        - 符号域S = 1，指数域E = 0000 0011B = 3，e = E-127 = -124，尾数域 M = 1.000 0010 1000 0011 1000 0100B，所以，x = - M X 2^{-124}
      - 因此，十进制为： -(1.000 0010 1000 0011 1000 0100B) X 2^{-124} = -4.7943174E-38

1.7 C语言数据类型的含义

- 典型的数据类型对应32位机的数据位宽

数据类型	字节数
char	1
short int	2
int	4
long	4
char *	4
float	4
double	8

  - 例1. 6某大字节序的计算机系统内存中存放有如表1- 6所示数据，试指出地址0x1234 5678表示的char，short int，int，float，double型数据符号数和无符号数的十进制值？

地址	字节0	字节1	字节2	字节3	字节4	字节5	字节6	字节7
0x1234 5678	0x81	0x82	0x83	0x84	0x85	0x86	0x87	0x88

  - Double型结果

    - Double型的16进制为0x8182838485868788
      - 二进制为：1000 0001 1000 0010 1000 0011 1000 0100 1000 0101 1000 0110 1000 0111 1000 1000B
        - 符号域S = 1，指数域E = 000 0001 1000B = 24，e = E-1023 = -999，尾数域M = 1.283848586878 8H，所以，x = - M X 2^{-999}
      - 因此，十进制为： -(1.283848586878 8H) X 2^{-999} = -2.159775099417 1683E-301

第1章 作业（三）

- 作业题（P26-P28）
  - 12
  - 13
  - 14
  - 15
  - 18
  - 19
  - 20
  - 21

- 要求：
  - 12-15题，做好，后面给参考答案；
  - 18-21题，微助教提交；

Thanks