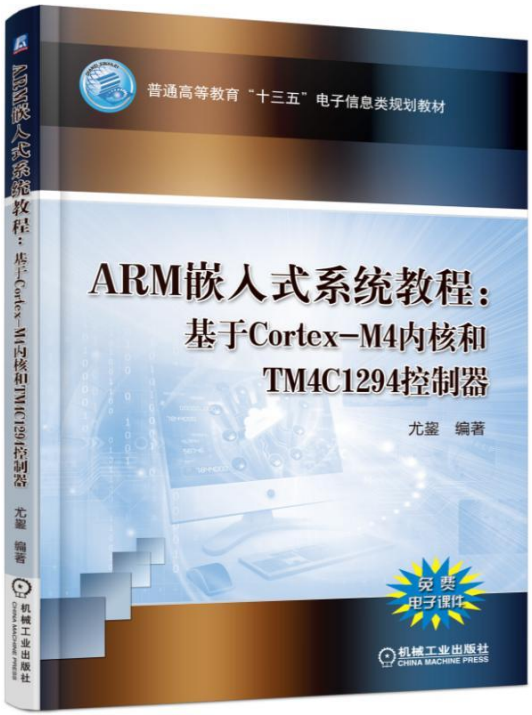
微机系统与接口概述



上课时间、地点：
单周：周二 1-2节
教七-10B
每周四 3-4节
教七-10B
实验地点：电工电子中心

参考书籍：
16/32位微机原理、汇编语言及接口技术教程 钱晓捷 机械工业出版社
ARM Cortex-M3与Cortex-M4权威指南（第3版）

本课程教材

课堂教学： 嵌入式系统教程--基于Tiva C系列ARM Cortex-M4微控制器[M] 沈建华编著，北京航空航天大学出版社。

《嵌入式系统教程--基于Tiva C系列ARM Cortex-M4微控制器》以ARM Cortex-M4内核MCU TM4C123x为核心，详细讲述MCU应用相关的各种外设 模块的原理和编程结构，并给出操作例程代码，包括 电源与时钟管理、存储器、通用输入/输出（GPIO）、定时器、PWM、异步和同步通信接口（UART、SPI、 I2C等）、模拟外设（ADC、DAC、AC）等。同时，对 嵌入式软件设计方法、嵌入式C语言基础、RTOS等作 了简明阐述。*后介绍MCU的软硬件开发环境、软件 库，以及低功耗设计和电磁兼容性基础等。

实验教学： ARM嵌入式系统教程：基于Cortex-M4内核和TM4C1294控制器[M]. 北京：机械工业出版社

《ARM嵌入式系统教程--基于Cortex-M4内核和M4C1294控制器》从微处理器系统的基本组成和工作原理开始介绍，便于初学者了解最基本的嵌入式系统的工作原理。本书以TI公司的Cortex－M4处理器TM4C1294NCPDT为核心（该芯片是TI公司目前的主力ARM芯片），详细介绍了该芯片的组成部件及结构特点，重点介绍了外设接口、常用通信接口及模拟接口，每部分都有相应的例程供读者理解。所有例程均在TI公司的CCS开发环境中进行了实际运行测试，并且详细介绍了CCS的使用方法及开发步骤，对于读者学习使用TI公司的其他嵌入式产品也有很大的帮助。

课程类别：大类专业基础课

计算机硬件系列课程之一

计算机组成原理

微机原理及接口技术

计算机体系结构

课程特点

以技术为主

面向应用

软硬件相结合

先修课程

数字逻辑与结构

提供硬件基础

计算机组成原理

确立计算机部件功能

掌握计算机工作原理

程序设计

建立必备软件基础

熟练应用C++语言程序设计

复习C语言的基础知识（力扣网，牛客网）

课程定位

电机学

微特电机及系统

电力传动技术

自动控制原理

电力系统稳定分析

电力系统暂态分析

发电厂电机部分

电力系统自动化

电气检测技术 ⟷ **微机系统与接口技术** ⟷ 电力电子技术

电机控制装置

电机变频器、调速器
家用电器
电动汽车、高铁、飞行器
机器人

电力系统自动化装置

调频器
APF、SVG
智能电网

课程目标

- 掌握微机系统的组成和工作原理；
- 能够使用C语言编写计算机程序；
- 掌握常见外设的工作原理及其接口技术；
- 初步具备开发微机系统软件、硬件的能力。

- 课程的特点：
  - 没有复杂的理论推导，只有简单的记忆
  - 重视应用
  - 知识点比较散

成绩评定

平时20%：作业15%，点名5%（三次点名不到取消期末考试资格）

期中考试20%：第九周进行

期末考试50%：填空、简答、综合设计

课程设计：10%

- 课程设计：
  - 教师发布多个课程设计题目
  - 学生自由组队，2-3人一组
  - 学生以组为单位，选择题目，完成课程设计要求（包括完成设计任务、PPT讲解、设计报告撰写）
  - 每个课程设计共18分，其中PPT答辩12分（由其他学生打分），设计报告6分（由教师、助教打分）。计算出总分后，按照约定的比例（比例可由组内商量决定，也可由任课教师决定），分给组内的人员。每人满分10分，最多附加2分。
  - 课程设计可选择不做（如果有自信靠其他成绩达到及格的话。）

学习方法很重要

复习并掌握先修课的有关内容

课堂：听讲与理解、适当笔记

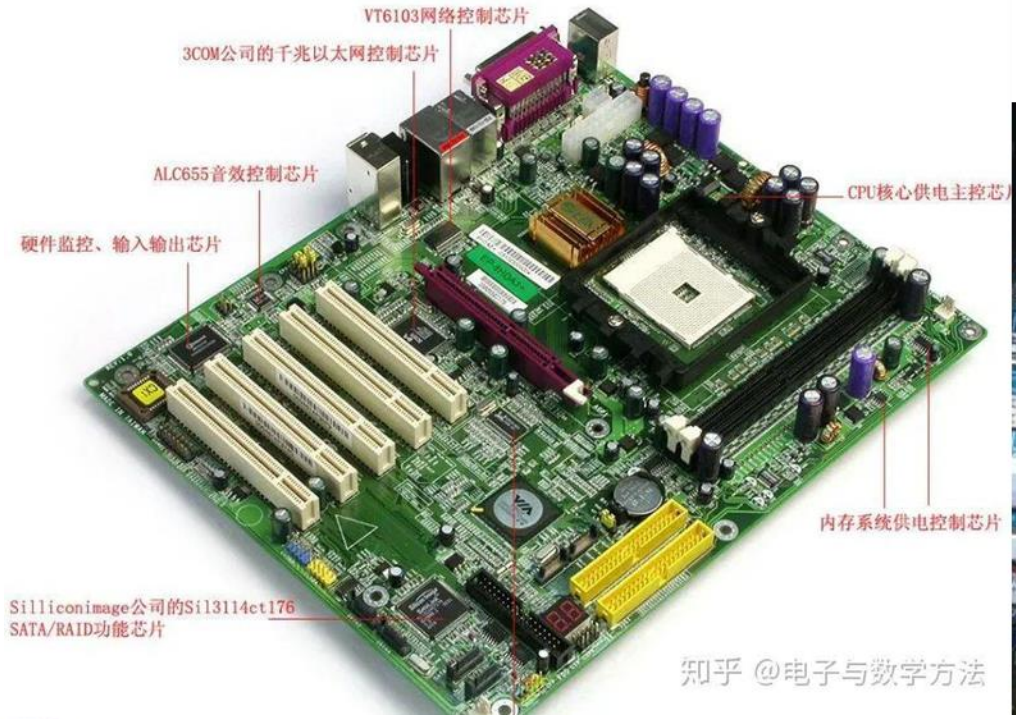
课后：认真读书、完成作业

实验：充分准备、勇于实践

第1章 微型计算机系统概述

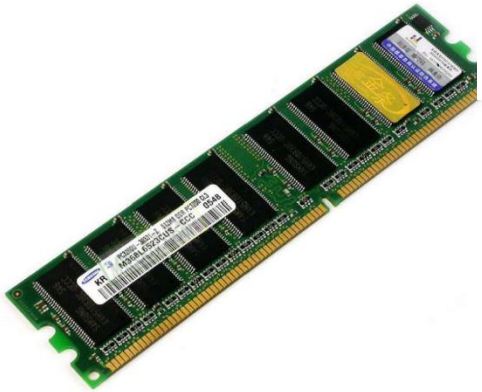
- 微型计算机简称微机，俗称电脑，其准确的称谓应该是微型计算机系统。它可以简单地定义为：在微型计算机硬件系统的基础上配置必要的外部设备和软件构成的实体。

- IBM PC系列机的主机板



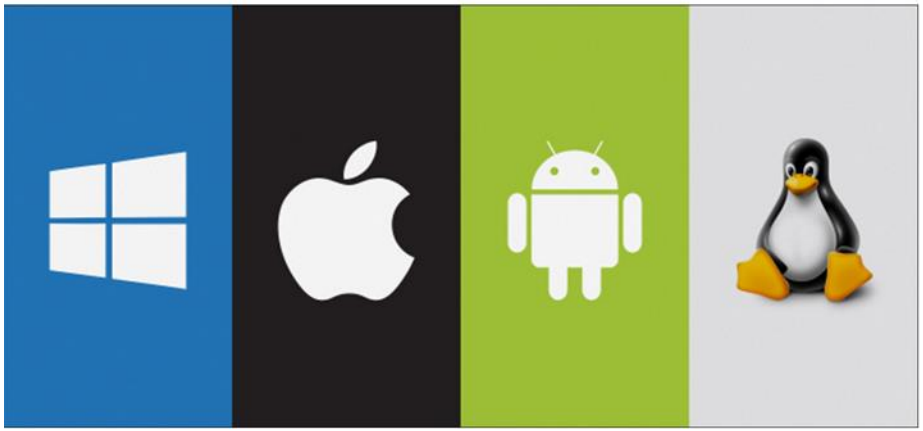
第1章 微型计算机系统概述

- 微型计算机的系统外设：
  - 硬盘、内存、显卡
  - CPU



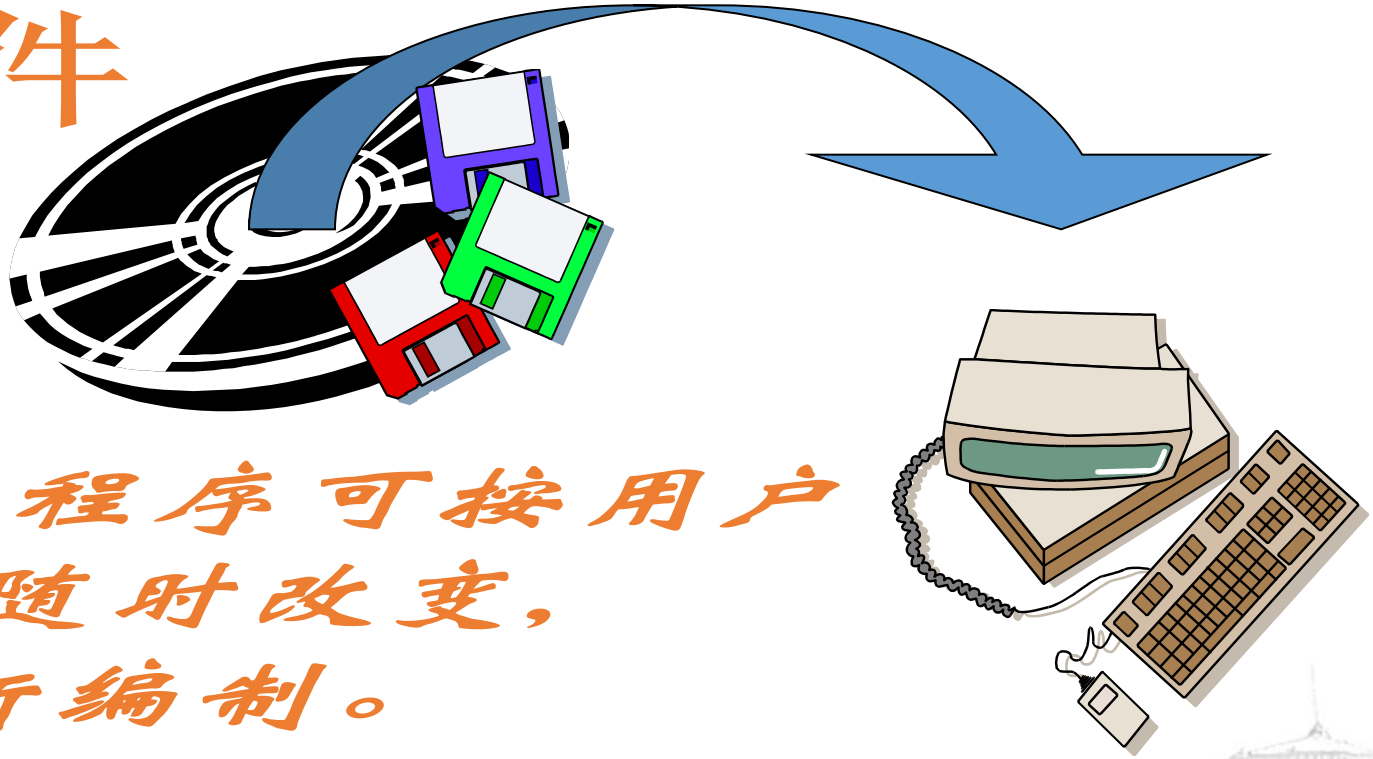
- 微型计算机的系统外设：
  - 键盘鼠标、显示器、摄像头

- 微型计算机的软件系统



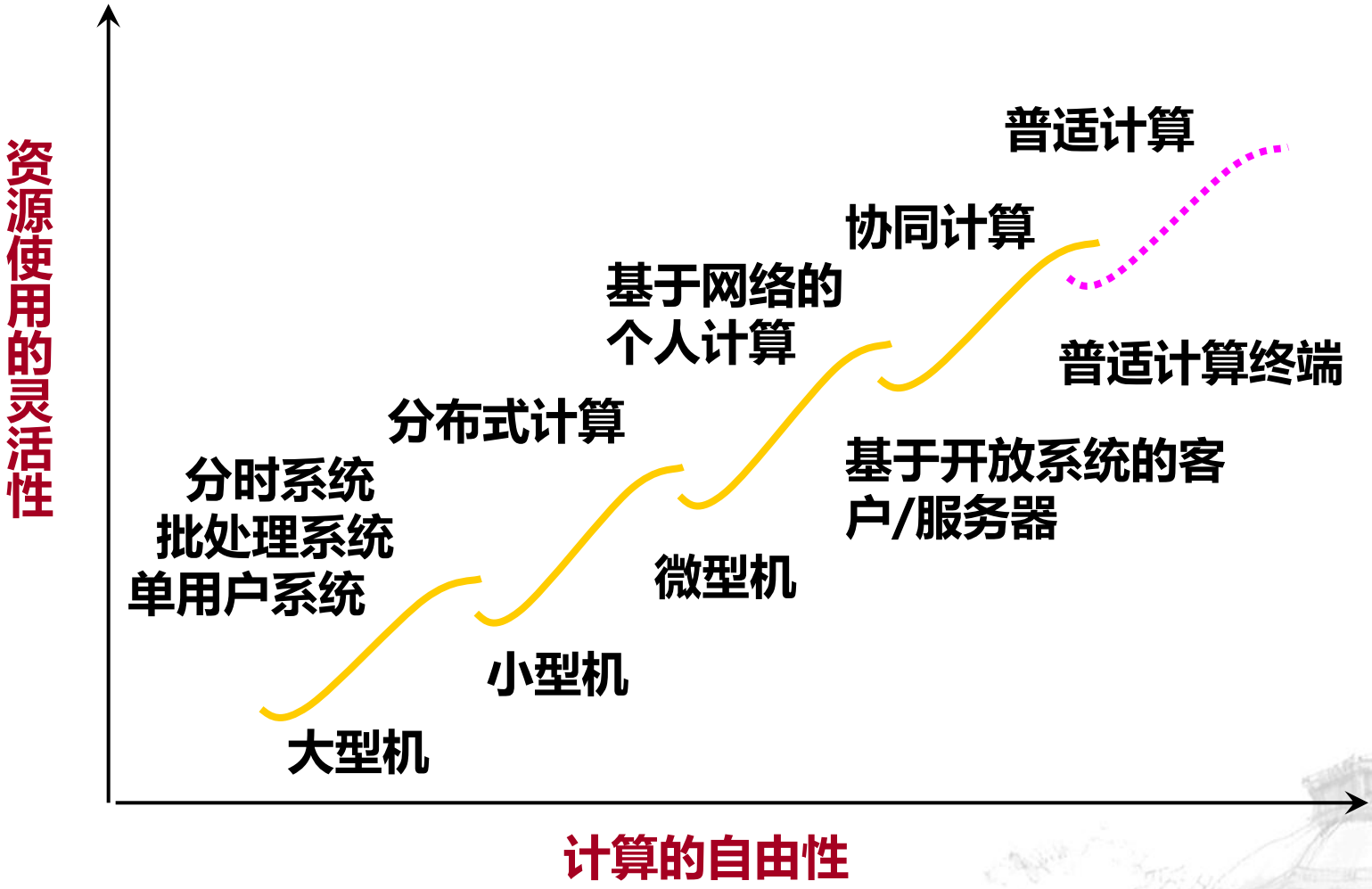
通用计算机—看得见的计算机

软件



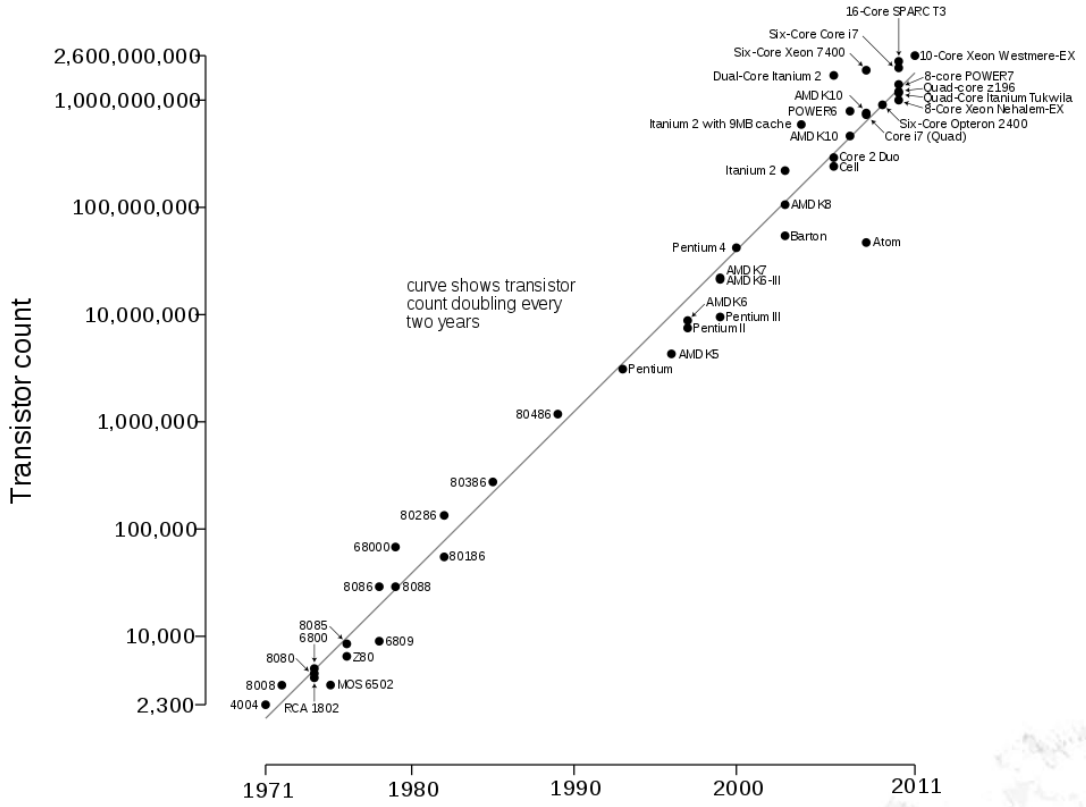
应用程序可按用户需要随时改变，即重新编制。

计算的发展过程

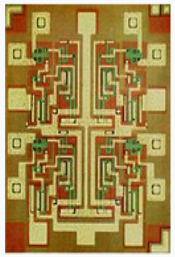

资源使用的灵活性
计算的自由性
大型机
小型机
微型机
分布式计算
基于网络的
个人计算
协同计算
普适计算
分时系统
批处理系统
单用户系统
基于开放系统的客
户/服务器
普适计算终端

- 摩尔定律

当价格不变时，集成电路上可容纳的元器件的数目，约每隔18-24个月便会增加一倍，性能也将提升一倍。



半导体工艺



10 μm – 1971年
6 μm – 1974年
3 μm – 1977年
1.5 μm – 1982年
1 μm – 1985年
800 nm – 1989年
600 nm – 1994年
350 nm – 1995年
250 nm – 1997年
180 nm – 1999年
130 nm – 2001年
90 nm – 2004年
65 nm – 2006年
45 nm – 2008年
32 nm – 2010年
22 nm – 2012年
14 nm – 2014年
10 nm – 2017年
7 nm – 2018年
5 nm – 2020年
3 nm – ~2022年
2 nm – ~2024年

1.1 微型计算机的发展和应用

- 1946年，世界上出现第一台数字式电子计算机ENIAC（电子数据和计算器）
- 发展到以大规模集成电路为主要部件的第四代，产生了微型计算机
- 1971年，Intel公司设计了世界上第一个微处理器芯片Intel4004，开创了一个全新的计算机时代

1.1.1 微型计算机的发展

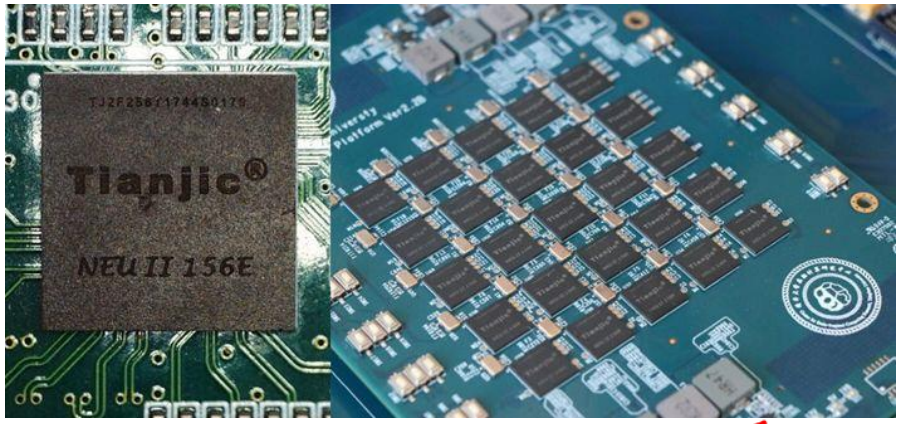
- 第1代：4位和低档8位微机
  4004→4040→8008
- 第2代：中高档8位微机
  Z80、I8085、M6800，Apple-II微机
- 第3代：16位微机
  8086→8088→80286，IBM PC系列机

1.1.1 微型计算机的发展（续）

- 第4代：32位微机
  - 80386→80486→Pentium→Pentium II →Pentium III →Pentium 4
  - 32位PC机、Macintosh机、PS/2机
- 第5代：64位微机

  Itanium、64位RISC微处理器芯片

  微机服务器、工程工作站、图形工作站



据世界顶级科技杂志，英国《自然》（Nature）杂志报道，中国清华大学类脑计算研究中心施路平团队研发的“天机芯”（Tianjic）迎来重大突破！在指甲盖大小的芯片里，安装了含大约40,000个神经元和1000万个突触，28-nm芯片由156个统一功能核心（FCore）组成，还结合了人工神经网络和生物网络的基本构建模块 - 轴突、突触、树突和体细胞块。

1.1.2 微型计算机的应用

- 计算机应用通常分成如下各个领域
  - 科学计算，数据处理，实时控制
  - 计算机辅助设计，人工智能，……
- 由于微型计算机具有如下特点
  - 体积小、价格低
  - 工作可靠、使用方便、通用性强……
- 所以，可以分为两个主要应用方向

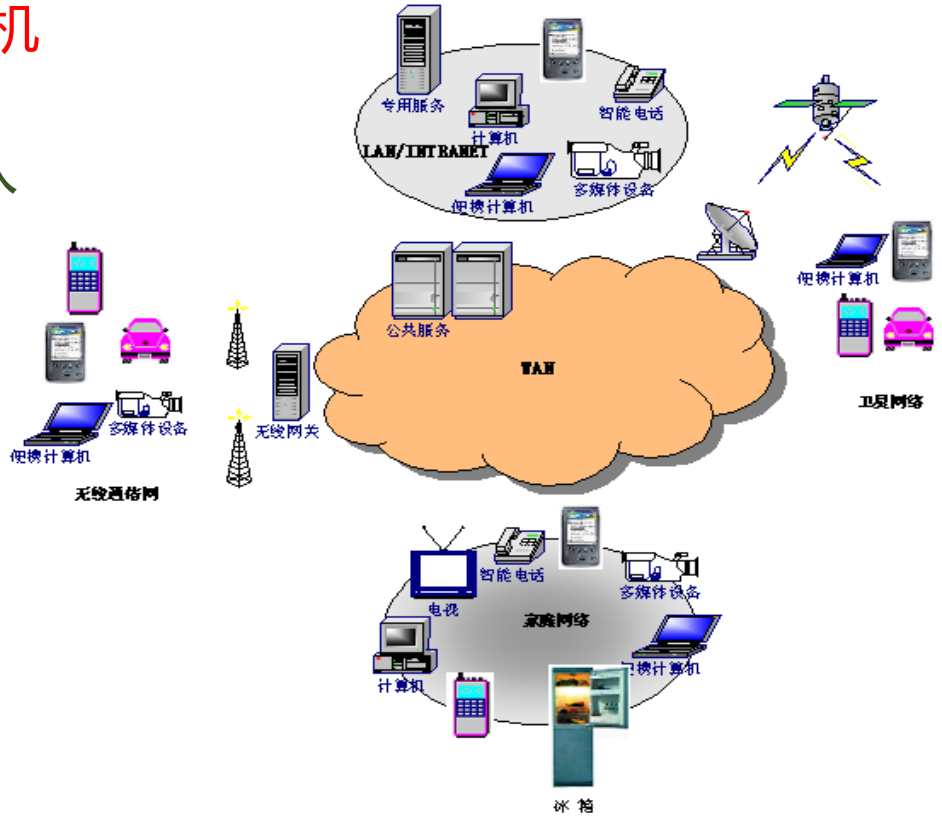
1.1.2 微型计算机的应用

- 用于数值计算、数据处理及信息管理方向
  - 通用微机，例如：PC微机
  - 功能越强越好、使用越方便越好

- 用于过程控制及嵌入应用方向
  - 专用微机，例如：工控机、单片机、数字信号处理器
  - 可靠性高、实时性强
  - 程序相对简单、处理数据量小

嵌入式系统无处不在

- 无处不在的计算机是计算机与使用者的比率达到和超过100:1的阶段
- 无处不在的计算机包括通用计算机和嵌入式计算机系统
- 在100:1比例中95%以上都是嵌入式计算机系统，并非通用计算机



嵌入式设备无处不在，

但桌面系统还依然有用

嵌入式系统定义

- 看不见的计算机，一般不能被用户编程，它有一些专用的I/O设备，对用户的接口是应用专用的。
- An embedded system is a computer system contained within some larger device or product with the intent purpose of providing monitoring and control services to that device.
- “Any sort of device which includes a programmable computer but itself is not intended to be a general-purpose computer.”
- 通常将嵌入式计算机系统简称为嵌入式系统。
- IEEE: “Device used to control，monitor，or assist the operation of equipment，machinery or plants”.
- 嵌入式系统是以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

嵌入式系统定义

一个嵌入式系统是一个硬件与软件的集合体。

硬件：嵌入式处理器、外设

软件：操作系统软件、应用程序

嵌入式系统是将先进的计算机技术、半导体技术和电子技术与各个行业的具体应用相结合后的产物。

包含有计算机，但又不是通用计算机的计算机应用系统。

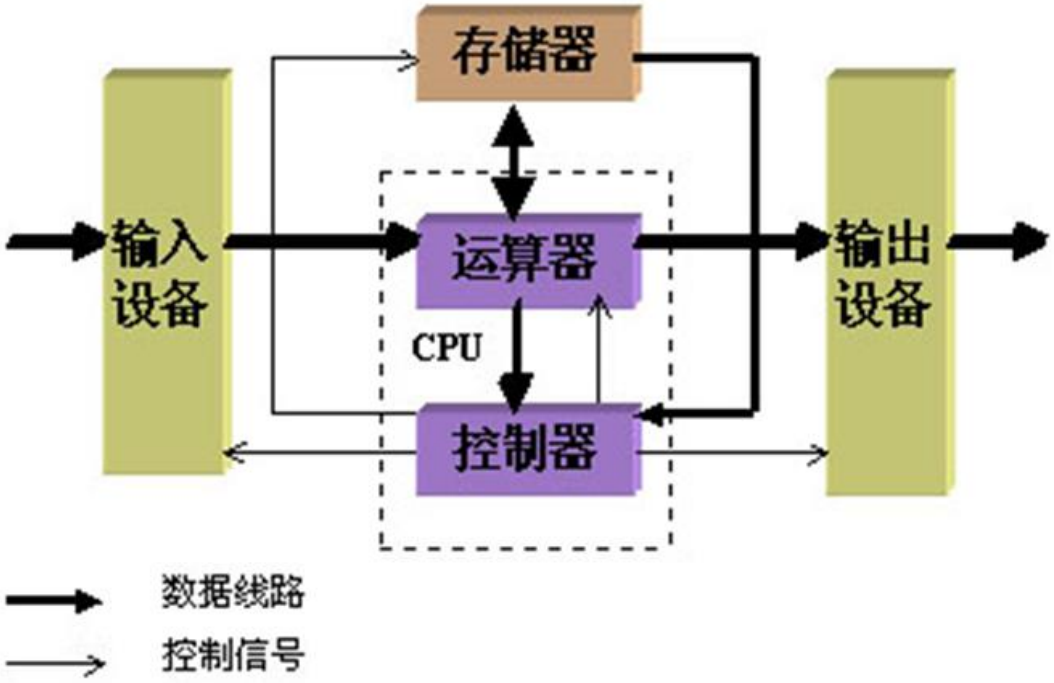
通用计算机与嵌入式系统对比

特征	通用计算机	嵌入式系统
形式和类型	• 看得见的计算机。 • 按其体系结构、运算速度和结构规模等因素分为大、中、小型机和微机。	• 看不见的计算机。 • 形式多样，应用领域广泛，按应用来分。
组成	• 通用处理器、标准总线和外设。 • 软件和硬件相对独立。	• 面向应用的嵌入式微处理器，总线和外部接口多集成在处理器内部。 • 软件与硬件是紧密集成在一起的（程序固化）
开发方式	• 开发平台和运行平台都是通用计算机	• 采用交叉开发方式，开发平台一般是通用计算机，运行平台是嵌入式系统。
二次开发性	• 应用程序可重新编制	• 一般不能再编程

嵌入式系统应用

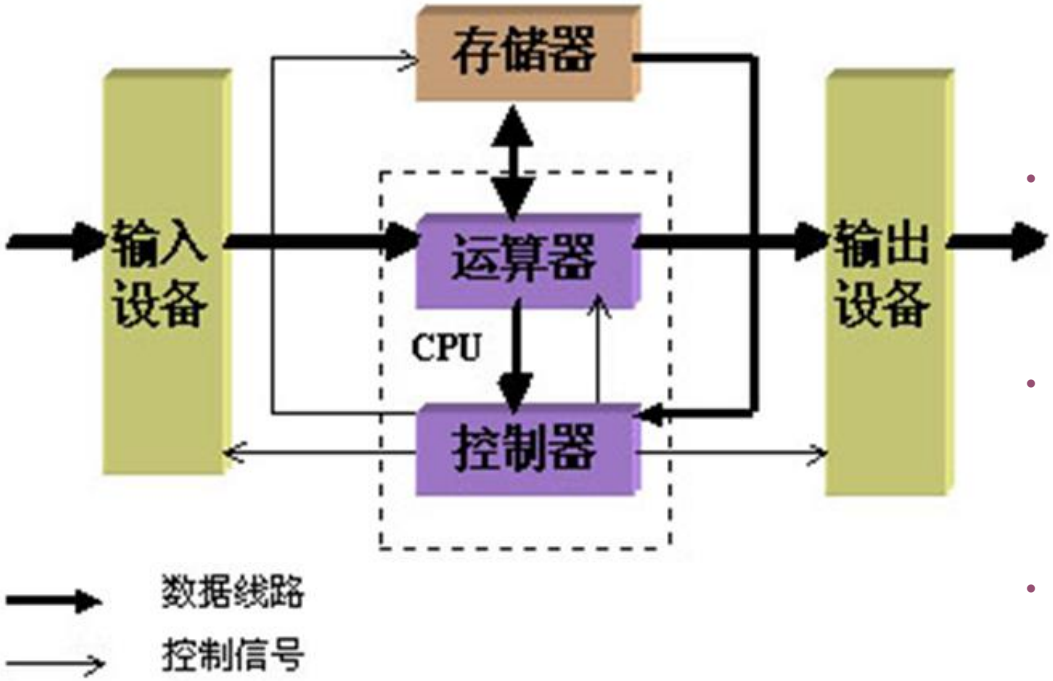
- **个人通信与娱乐系统**：手机、数码相机、音乐播放器、可穿戴电子产品、PSP游戏机
- **家电类产品**：数字电视、扫地机器人、智能家电
- **办公自动化**：打印机，复印机、传真机
- **医疗电子类产品**：生化分析仪、血液分析仪、CT
- **网络通信类产品**：通信类交换设备、网络设备(交换机、路由器、网络安全)
- **汽车电子类产品**：引擎控制、安全系统、汽车导航与娱乐系统
- **工业控制类产品**：工控机、交互式终端(POS、ATM)、安全监控、数据采集与传输、仪器仪表
- **军事及航天类产品**：无人机、雷达、作战机器人

冯·诺依曼体系结构



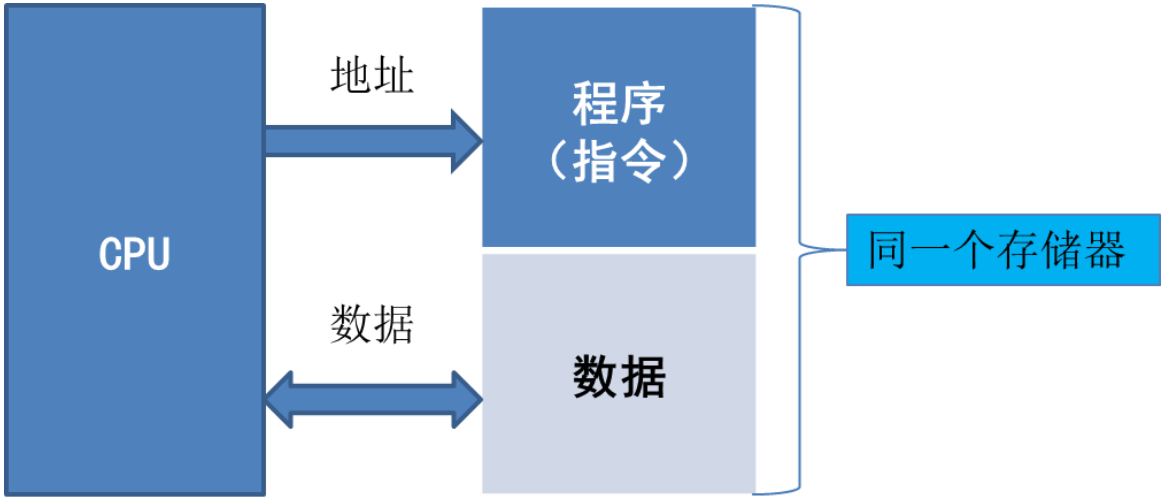
- 计算机处理的数据和指令用二进制数表示；
- 采用存储程序方式，指令和数据存储在存储器中；
- 顺序执行程序的每一条指令；
- 由存储器、运算器、控制器、输入设备和输出设备五大部件组成计算机系统，并规定了这五部分的基本功能。

冯·诺依曼体系结构



- 存储器：用来存放数据和程序
- 运算器：主要运行算数运算和逻辑运算，并将中间结果暂存到运算器中
- 控制器：主要用来控制和指挥程序和数据的输入运行，以及处理运算结果
- 输入设备：用来将人们熟悉的信息形式转换为机器能够识别的信息形式，常见的有键盘，鼠标等
- 输出设备：可以将机器运算结果转换为人们熟悉的信息形式，如打印机输出，显示器输出等

- 冯诺依曼结构

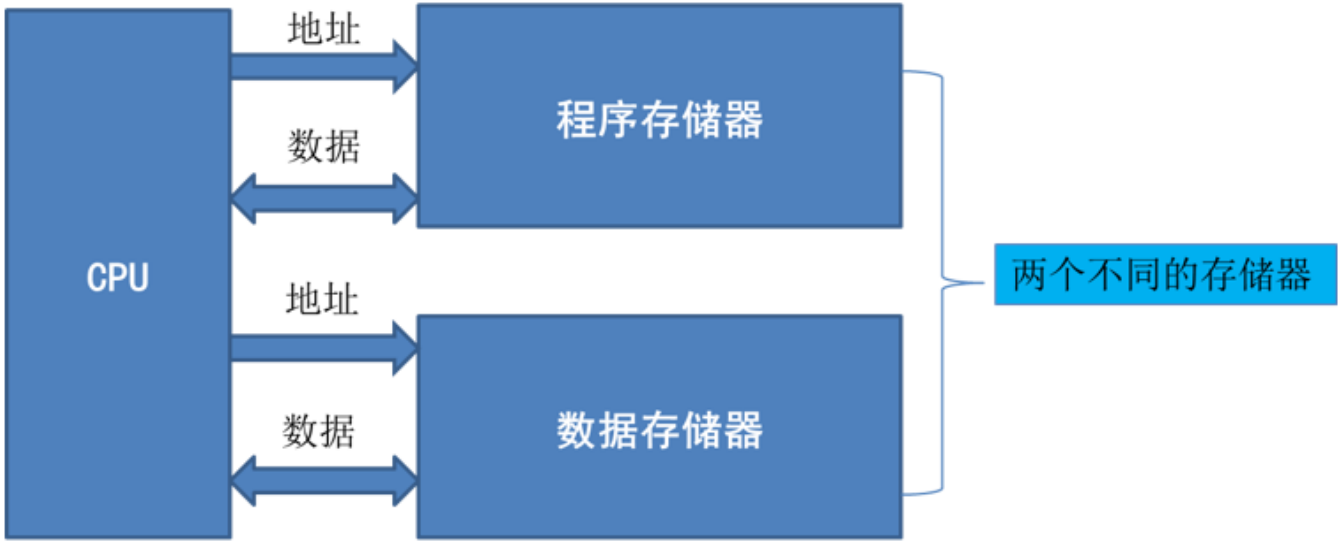


冯诺依曼结构

不能同时取指令和取操作数

程序指令和数据的宽度相同

哈佛结构



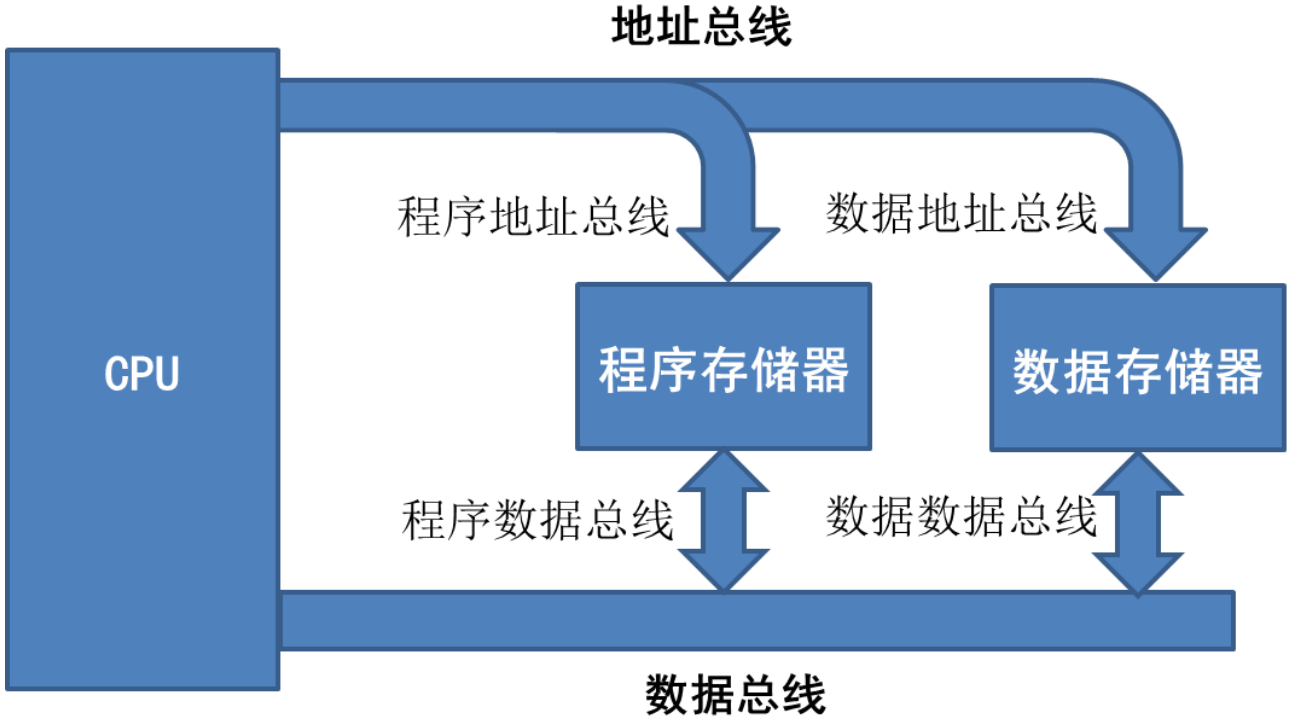
哈佛结构

将程序指令存储和数据存储分开

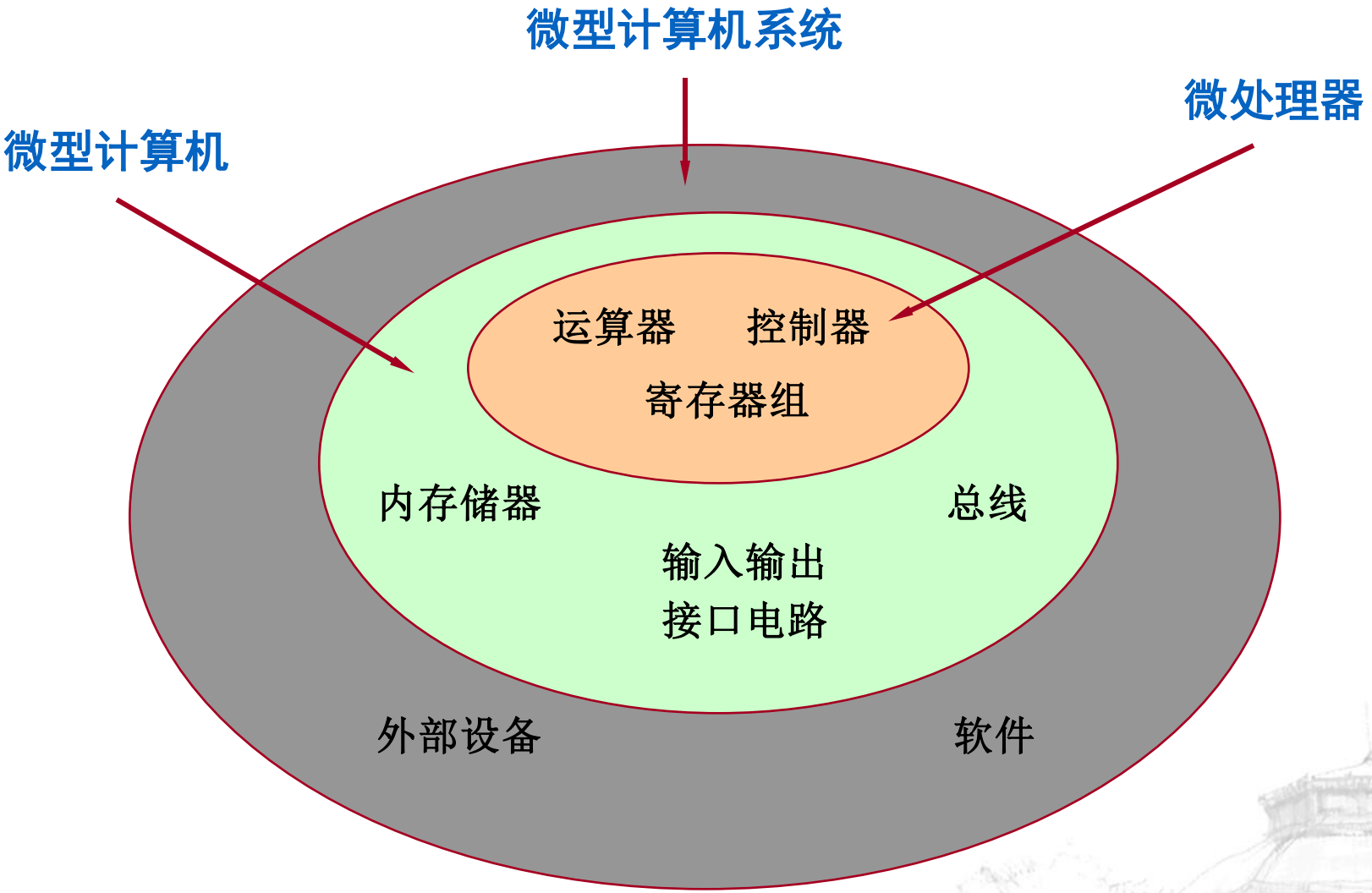
数据和指令的储存可以同时进行，可以使指令和数据有不同的数据宽度

总线过多，复杂，成本高

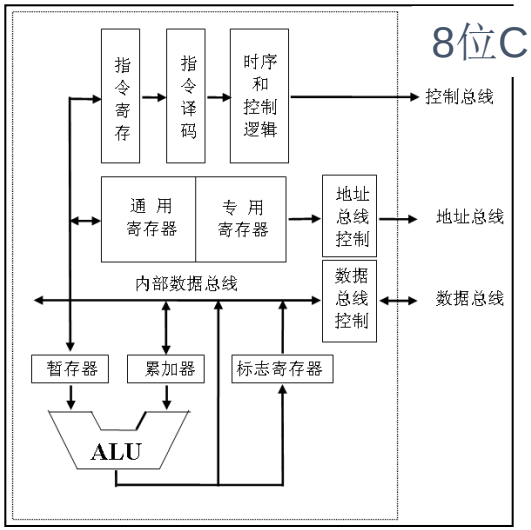
- 改进型哈佛结构

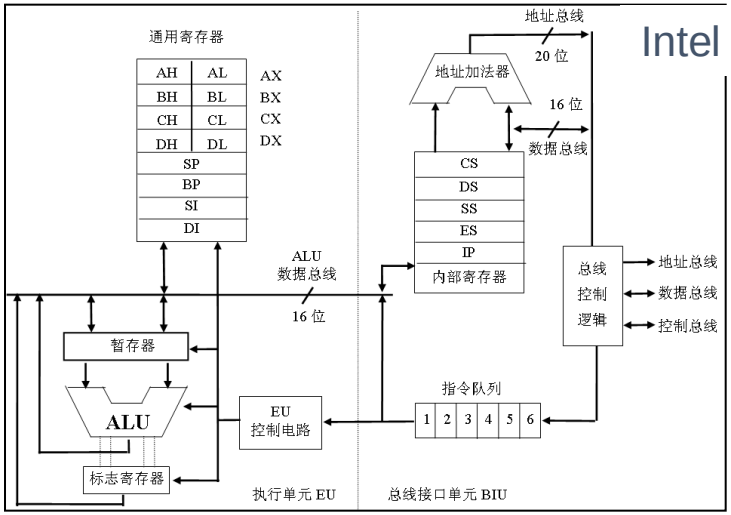


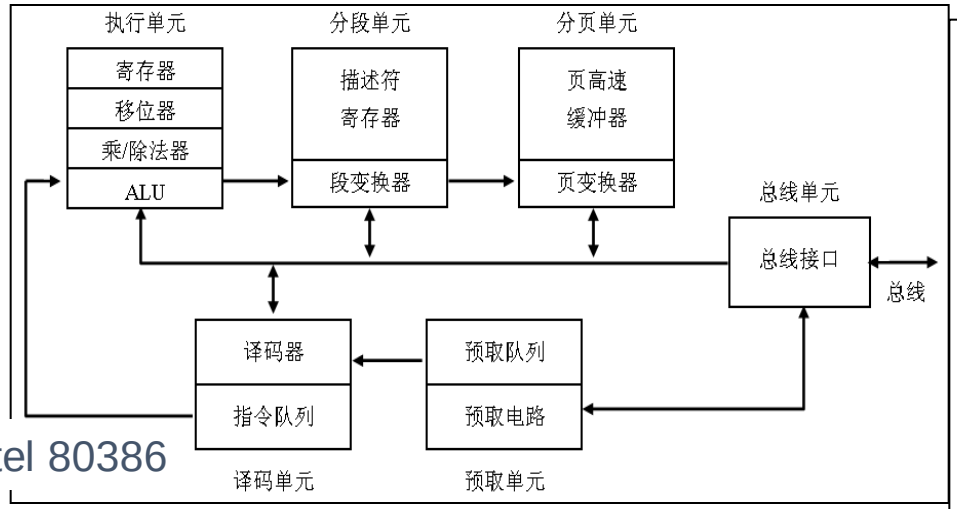
改进型哈佛结构

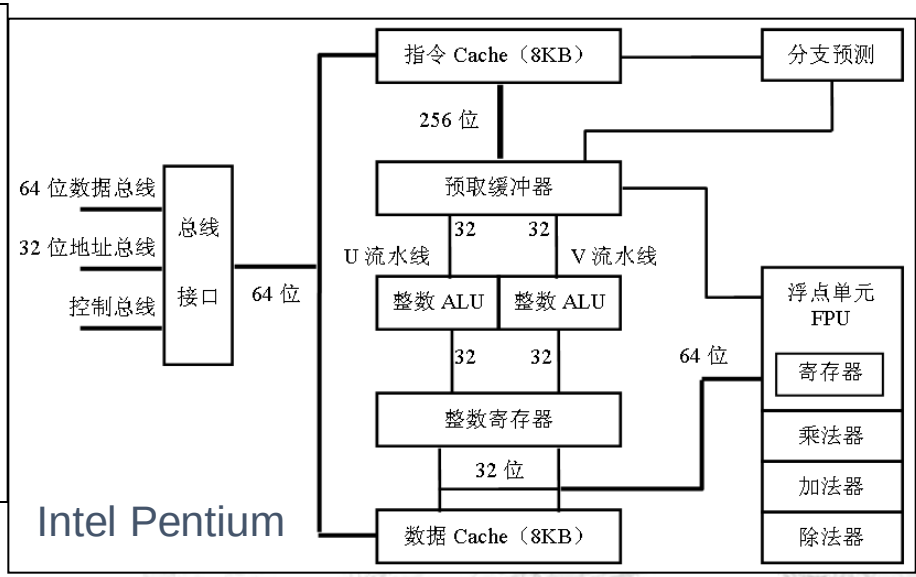

微型计算机系统
微型计算机
微处理器
运算器
控制器
寄存器组
内存储器
总线
输入输出
接口电路
外部设备
软件

• 从应用角度看到的处理器内部结构









1.2.1 微型计算机的硬件组成

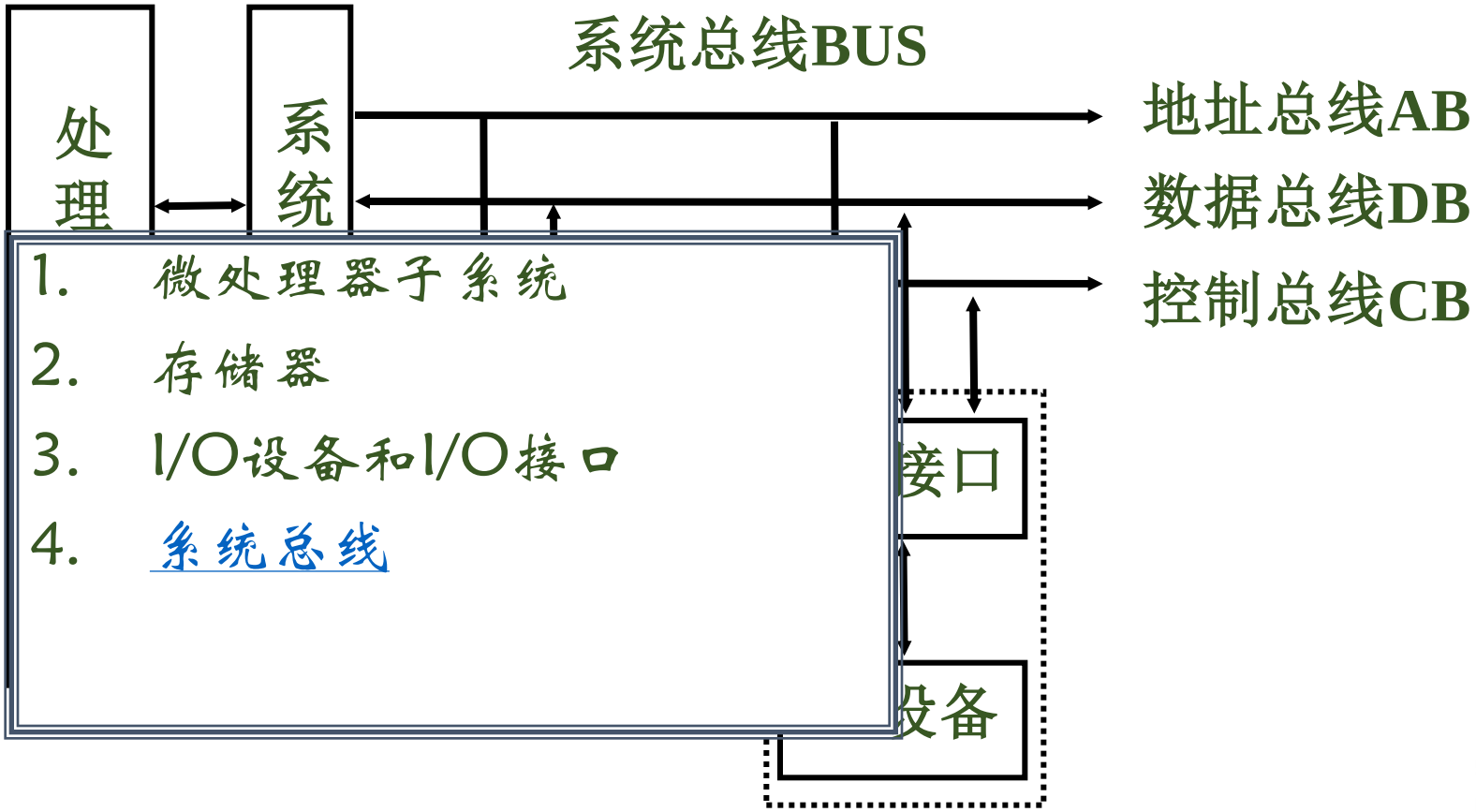


图1.1 微型计算机的系统组成

系统总线

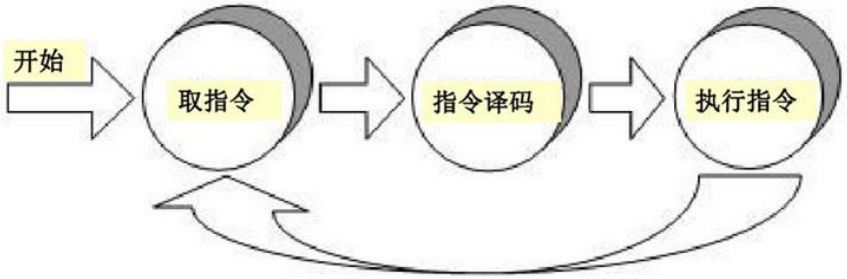
- 总线是指传递信息的一组公用导线
- 总线是传送信息的公共通道
- 微机系统采用总线结构连接系统功能部件
- 总线信号可分成三组
  - 地址总线AB：传送地址信息
  - 数据总线DB：传送数据信息
  - 控制总线CB：传送控制信息

➢ 总线信号

- 地址总线AB
  - 输出将要访问的内存单元或I/O端口的地址
  - 地址线的多少决定了系统直接寻址存储器的范围
- 数据总线DB
  - CPU读操作时，外部数据通过数据总线送往CPU
  - CPU写操作时，CPU数据通过数据总线送往外部
  - 数据线的多少决定了一次能够传送数据的位数
- 控制总线CB
  - 协调系统中各部件的操作，有输出控制、输入状态等信号
  - 控制总线决定了系统总线的特点，例如功能、适应性等

➢CPU的工作原理

- 程序的执行过程实际上是不断地取出指令、分析指令、执行指令的过程。



（1）预先把指挥计算机如何进行操作的指令序列（就是程序）和原始数据输入到计算机内存中，每条指令中明确规定了计算机从哪个地址取数，进行什么操作，然后送到什么地方去等步骤。

（2）计算机在执行时，先从内存中取出第一条指令，通过控制器的译码器接收指令的要求，再从存储器中取出数据进行指定的运算和逻辑操作等，然后再按地址把结果送到内存中，如果需要向硬盘等存储设备存储数据，还需要将内存中的该数据存储到硬盘中。接下来取出第2条指令，在控制器的指挥下完成规定操作，依次进行下去，直到遇到停止指令。

（3）计算机中基本上有两股信息在流动，一种是数据，即各种原始数据、中间结果和程序等，另一种信息是控制信息，它控制机器的各种部件执行指令规定的各种操作。

（1）通过命令记录员找到当前执行到的命令，并将命令提取出来放到命令控制器中的指令暂存处

（2）接着控制器中的命令解释器对命令进行解释，并产生相应的控制信号

（3）在控制器的控制下，将两个数再从存储器中提取出来，分别放到运算器的两个数据缓存区中

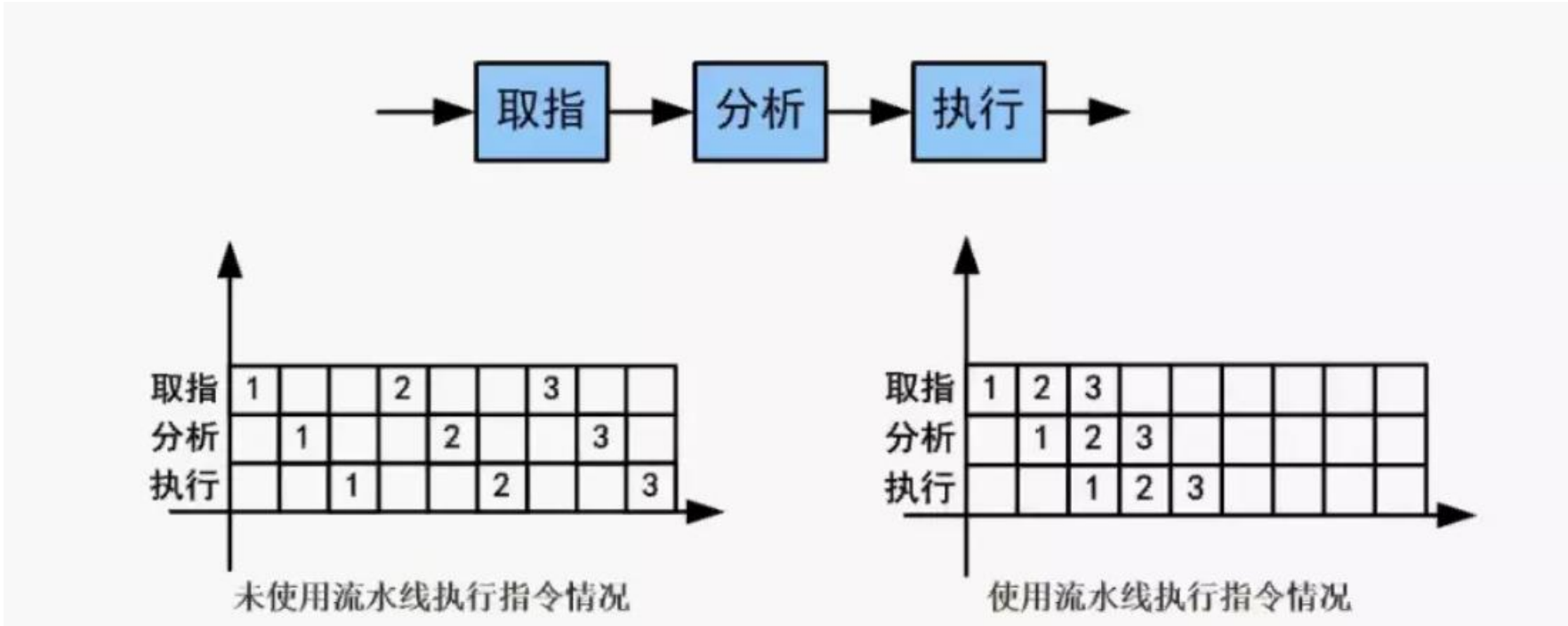
（4）接着控制器产生一个控制信号告诉计算电路做这两个数的加法，相加得到运算结果.

（5）把运算结果运算结果写回存储器指定位置。

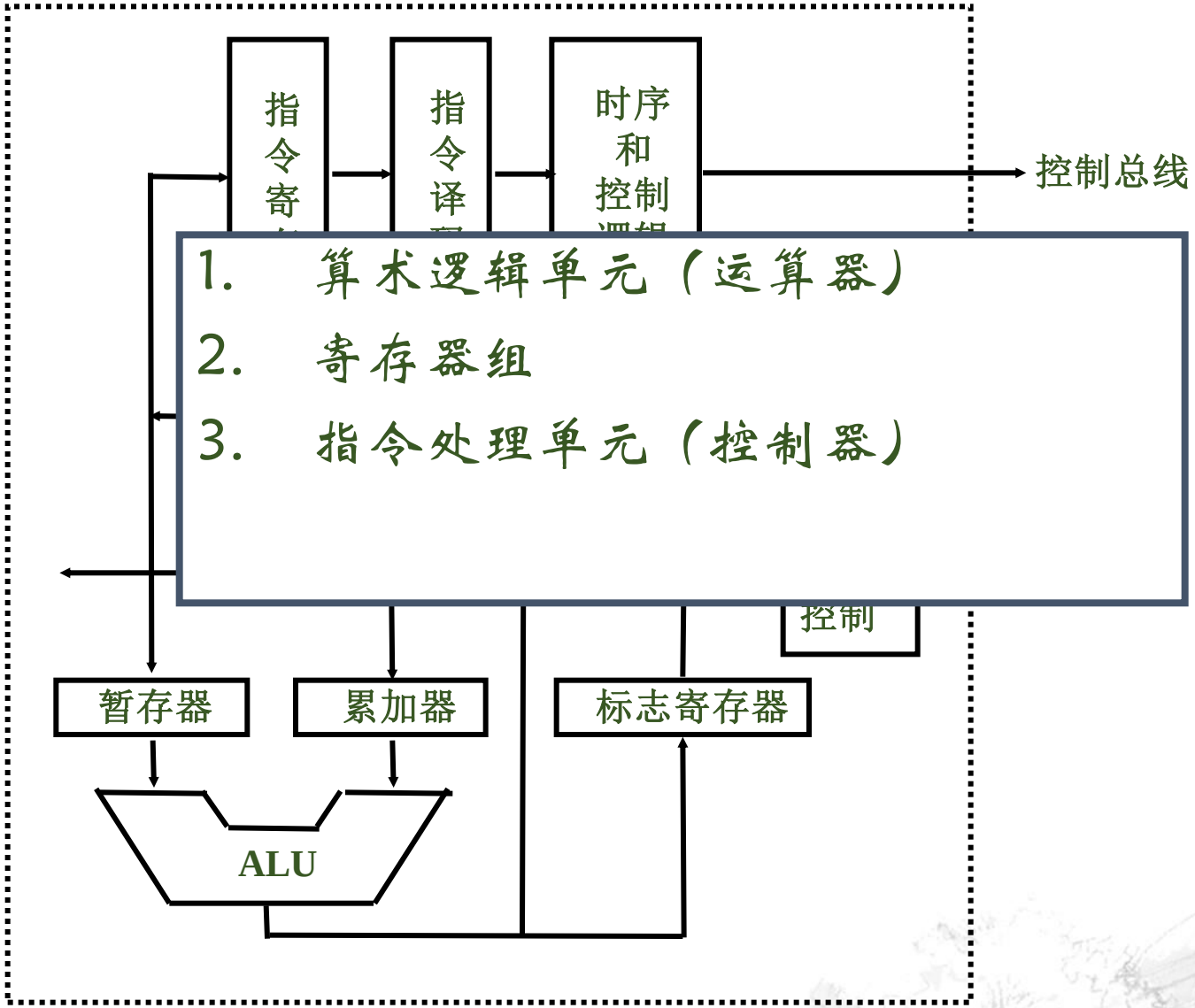
（6）完这条命令后，转到下一条命令继续执行.

- 流水线技术

流水线是指在程序执行时多条指令重叠进行操作的一种准并行处理实现技术。



1.2.2 微处理器的基本结构



1. 算术逻辑单元ALU

- 计算机的运算器，负责处理器所能进行的各种运算，主要就是算术运算和逻辑运算
- 累加器结构的处理器
  - 累加器（Accumulator）
    - 提供一个操作数
    - 保存运算结果
- 标志（Flag）寄存器
  - 反映运算结果的辅助信息
  - 例如:有无进借位、是否为零、是否为负等
  - 也称为程序状态字（PSW）

2. 寄存器（Register）

- 处理器内部需要高速存储单元，用于暂时存放程序执行过程中的代码和数据
- 透明寄存器
  - 对应用人员不可见、不能直接控制的寄存器
- 可编程（Programmable）寄存器
  - 具有引用名称、供编程使用
  - 通用寄存器

    数量较多、使用频度较高，具有多种用途
  - 专用寄存器

    只用于特定目的

3. 指令处理单元

- 处理器的控制单元，它控制指令的执行和信息的传输
- 指令执行的过程
  - 取指：指令处理单元将指令从主存取出，并通过总线传输到处理器内部的指令寄存器
  - 译码：指令处理单元通过指令译码电路获得该指令的功能
  - 执行：指令处理单元的时序和控制逻辑按一定的时间顺序发出和接收相应信号，完成指令所要求的操作

1.2.3 8088/8086的功能结构

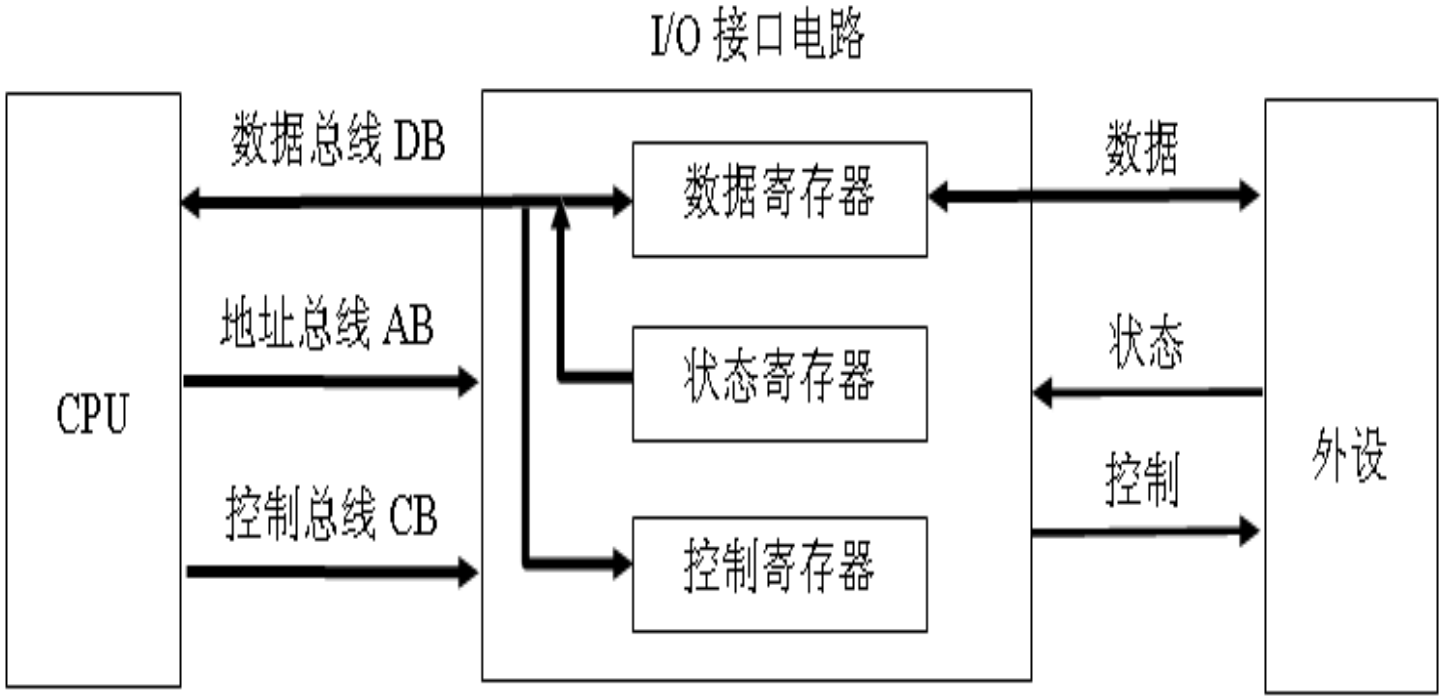
- 8088的内部结构从功能分成两个单元
  - 总线接口单元BIU——管理8088与系统总线的接口，负责CPU对存储器和外设进行访问
  - 执行单元EU——负责指令的译码、执行和数据的运算
- 两个单元相互独立，分别完成各自操作
- 两个单元可以并行执行，实现指令取指和执行的流水线操作

输入输出接口

I/O接口概述

- 微机的外部设备多种多样
  - 工作原理、驱动方式、信息格式、以及工作速度方面彼此差别很大
  - 它们不能与CPU直接相连
  - 必须经过中间电路（I/O接口）再与系统相连
- I/O接口是位于基本系统与外设间、实现两者数据交换的控制电路
  - 在PC机主板上的可编程接口电路
  - 系统总线插槽中的电路卡（Card）

I/O接口的典型结构



I/O地址＝外设端口，对应接口寄存器

1. 通用接口内部组成

- **数据寄存器**
  - 保存处理器与外设之间交换的数据
  - 数据输入寄存器：保存从输入设备获取的数据，处理器选择合适的方式进行读取
  - 数据输出寄存器：保存处理器发往输出设备的数据，适时到达输出设备
- **状态寄存器**
  - 保存外设当前的工作状态信息
- **控制寄存器**
  - 保存处理器控制接口电路和外设操作的有关信息

2. 通用接口外部特性

- 接口电路的外部特性由其引出信号来体现
- I/O接口处于处理器与外设之间：
- **面向微处理器一侧的信号**
  - 与处理器总线或系统总线类似
  - 有数据信号、地址信号和控制信号等
- **面向外设一侧的信号**
  - 与外设有关
  - 外设数据信号、外设状态信号和外设控制信号

3. 通用接口基本功能

- **数据缓冲**
  - 匹配快速的处理器与相对慢速的外设的数据交换
  - 缓冲：实现接口双方数据传输的速度匹配
- **信号变换**
  - 把信号相互转换为适合对方的形式
  - 计算机直接处理的信号
    - 数字量（0和1组成的信号编码）
    - 开关量（只有两种状态的信号）
    - 脉冲量（低脉冲信号，高脉冲信号）

- 计算机主机有多种与外设传送数据的方式
- 通过处理器执行I/O指令完成
  - 无条件传送
  - 查询传送
  - 中断传送
- 以硬件为主，加快传输速度
  - 直接存储器存取（DMA）
  - 使用专门的I/O处理机

4. 软件编程（汇编语言）

- 接口芯片具有可编程性（Programmable）
- 命令字（控制字）
  - 写入接口芯片、选择工作方式、控制数据传输
- 初始化程序
  - 选择I/O接口工作方式、设置原始工作状态等
- 驱动程序
  - 操纵I/O接口完成具体工作

硬件接口电路需要软件编程配合工作

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
```

包含头文件

```
void main(void)
{
    // Enable the GPIO module.
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlDelay(1);

    // Configure PA1 as an output.
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_1);
    // Loop forever.
    //
    while(1)
    {
        // Set the GPIO high.
        ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, GPIO_PIN_1);
        // Delay for a while.
        ROM_SysCtlDelay(1000000);
        // Set the GPIO low.
        ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0);
        // Delay for a while.
        ROM_SysCtlDelay(1000000);
    }
}
```

由main函数开始执行
包含一个while无限循环

- C语言的语法回顾

关键字:

1). 数据类型(常用char, short, int, long, unsigned, float, double)

2). 运算和表达式(=, +, -, *, while, do-while, if, goto, switch-case)

3). 数据存储(auto, static, extern, const, register, volatile, restricted),

4). 结构(struct, enum, union,typedef),

5). 位操作和逻辑运算(<<, >>, &, |, ~, ^, &&),

6). 预处理(#define, #include, #error, #if...#elif...#else...#endif等),

7). 平台扩展关键字(__asm, __inline, __syscall)

C语言的语法回顾

数据类型:

C语言提供的typedef就是用于处理这种情况的关键字，在大部分支持跨平台的软件项目中被采用，典型的如下:

```
typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned int uint32_t;
.....
typedef signed int int32_t;
```

➢C语言的语法回顾

内存管理和存储架构:

C语言允许程序变量在定义时就确定内存地址，通过作用域，以及关键字extern，static，实现了精细的处理机制，按照在硬件的区域不同，内存分配有三种方式:

1). **从静态存储区域分配**。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2). **在栈上创建**。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

3). **从堆上分配**，亦称动态内存分配。程序在运行的时候用 malloc 或 new 申请任意多少的内存，程序员自己负责在何时用 free 或 delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但同时遇到问题也最多。

➢C语言的语法回顾

内存管理和存储架构:

```c
#include <stdio.h>
#include <stdlib.h>

static int st_val;                      //静态全局变量 -- 静态存储区
int ex_val;                             //全局变量 -- 静态存储区
int main(void)
{
   int a = 0;                           //局部变量 -- 栈上申请
   int *ptr = NULL;                     //指针变量
   static int local_st_val = 0;         //静态变量
   local_st_val += 1;
   a = local_st_val;
   ptr = (int *)malloc(sizeof(int)); //从堆上申请空间
   if(ptr != NULL)
   {
      printf("*p value:%d", *ptr);
      free(ptr);
      ptr = NULL;
      //free后需要将ptr置空，否则会导致后续ptr的校验失效，出现野指针
   }
}
```

➢C语言的语法回顾

内存管理和存储架构:

- **数组**是由相同类型元素构成，当它被声明时，编译器就根据内部元素的特性在内存中分配一段空间
- C语言也提供多维数组，
- **指针**则是提供使用地址的符号方法，C语言的指针具有最大的灵活性，在被访问前，可以指向任何地址，这大大方便了对硬件的操作，但同时也对开发者有了更高的要求。

不同类型指针加法的运算结果

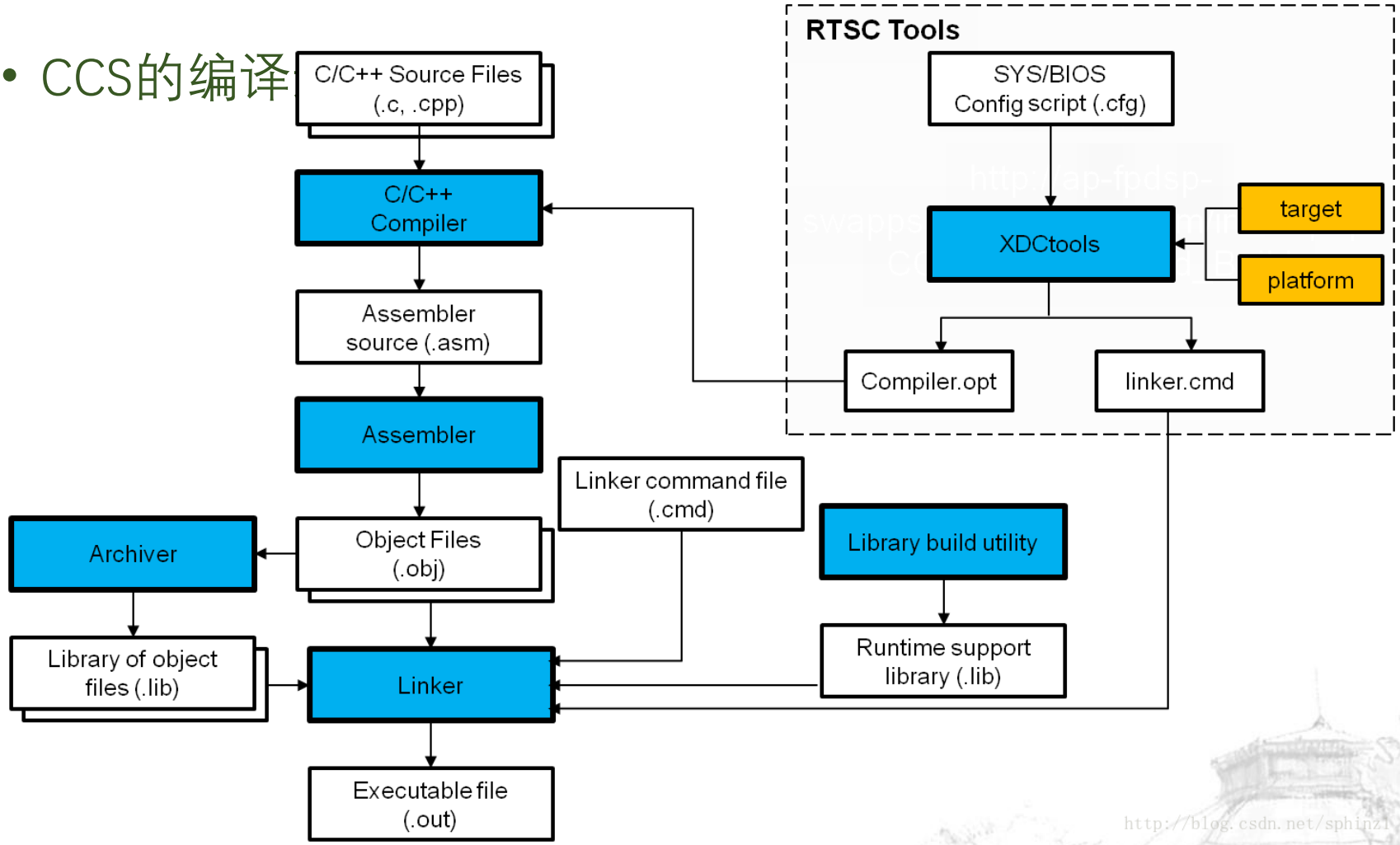
```
int main(void)
{
  char cval[] = "hello";
  int i;
  int ival[] = {1, 2, 3, 4};
  int arr_val[][2] = {{1, 2}, {3, 4}};
  const char *pconst = "hello";
  char *p;
  int *pi;
  int *pa;
  int **par;

  p = cval;
  p++;              //addr增加1
  pi = ival;
  pi+=1;            //addr增加4
  pa = arr_val[0];
  pa+=1;            //addr增加4
  par = arr_val;
  par++;           //addr增加8
  for(i=0; i<sizeof(cval); i++)
  {
```

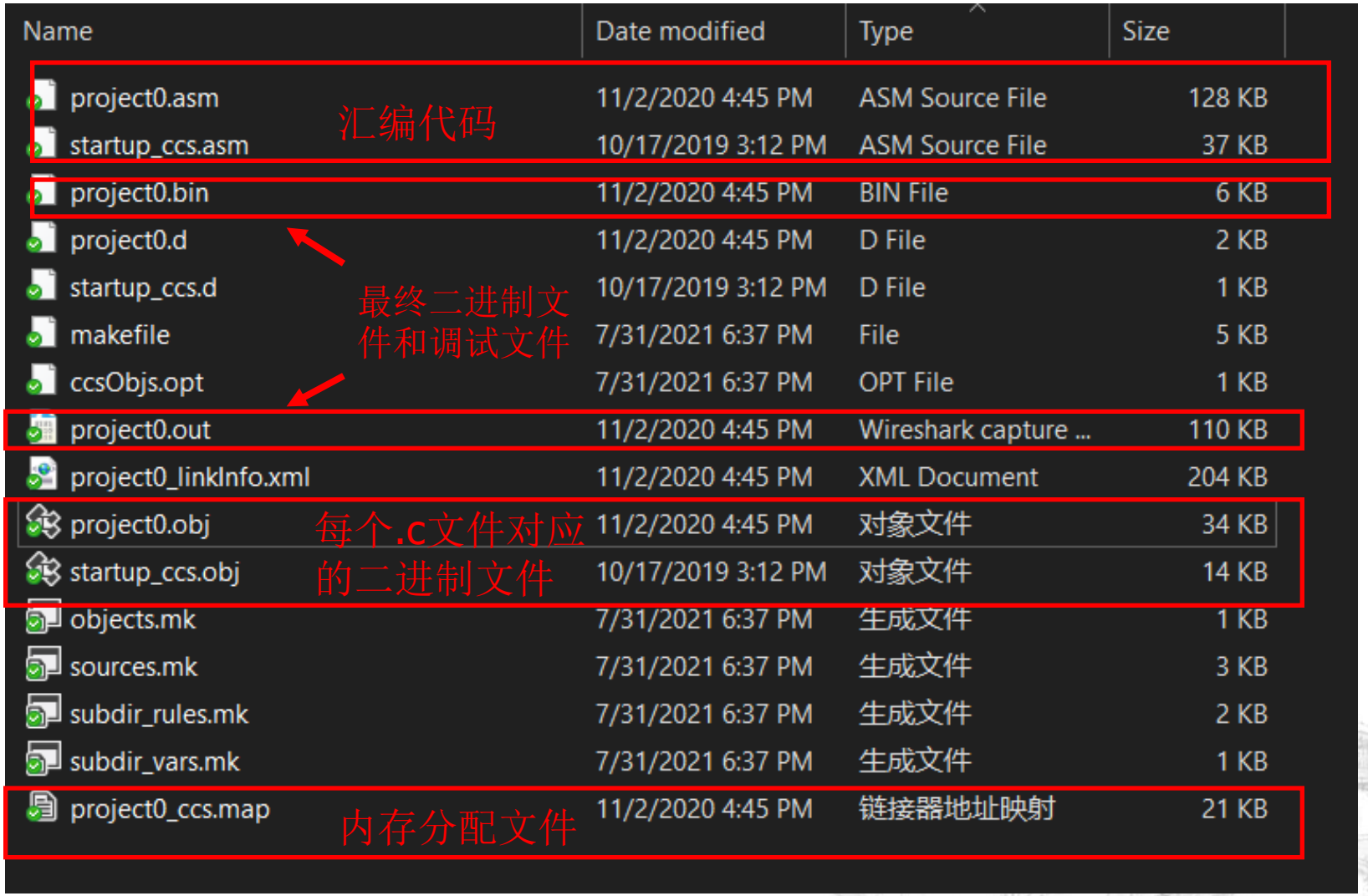
C语言编译过程

- 预处理(Preprocessing)，即完成宏定义和include 文件展开等工作;
- 编译(Compilation)，根据编译参数进行不同程序的优化，编译成汇编代码;
- 汇编(Assemble)，用汇编器把上一阶段生成的汇编代码进一步生成目标代码;
- 链接(Linking)，用链接器把上一阶段生成的目标代码、其他一些相关的系统提供的目标代码（如crtx.o）和系统或用户提供的库链接起来，生成最终的执行代码。生成可执行文件。

- CCS的编译

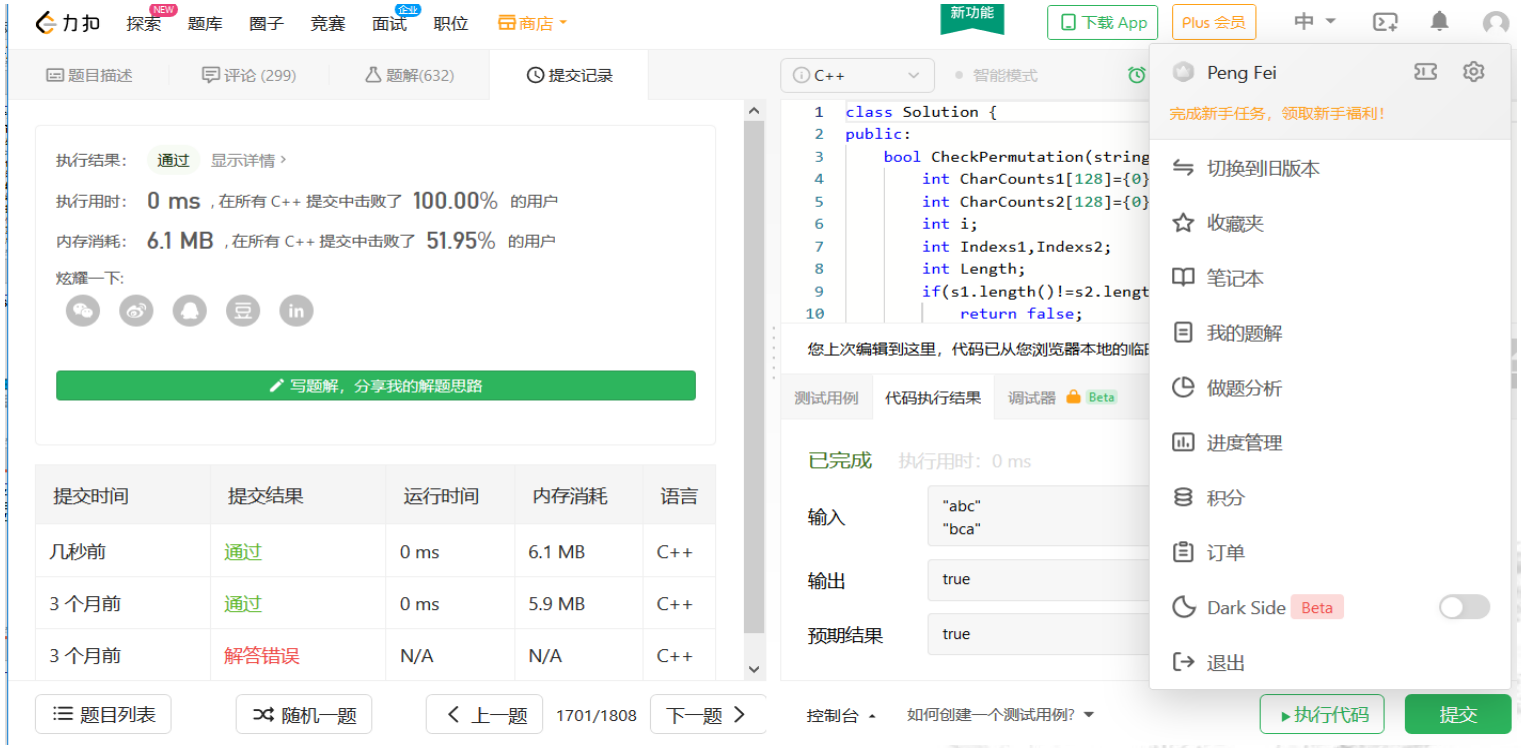


编译过程生成的文件


Name
Date modified
Type
Size
project0.asm
11/2/2020 4:45 PM
ASM Source File
128 KB
startup_ccs.asm
10/17/2019 3:12 PM
ASM Source File
37 KB
汇编代码
project0.bin
11/2/2020 4:45 PM
BIN File
6 KB
project0.d
11/2/2020 4:45 PM
D File
2 KB
startup_ccs.d
10/17/2019 3:12 PM
D File
1 KB
makefile
7/31/2021 6:37 PM
File
5 KB
最终二进制文件和调试文件
ccsObjs.opt
7/31/2021 6:37 PM
OPT File
1 KB
project0.out
11/2/2020 4:45 PM
Wireshark capture ...
110 KB
project0_linkInfo.xml
11/2/2020 4:45 PM
XML Document
204 KB
project0.obj
11/2/2020 4:45 PM
对象文件
34 KB
每个.c文件对应的二进制文件
startup_ccs.obj
10/17/2019 3:12 PM
对象文件
14 KB
objects.mk
7/31/2021 6:37 PM
生成文件
1 KB
sources.mk
7/31/2021 6:37 PM
生成文件
3 KB
subdir_rules.mk
7/31/2021 6:37 PM
生成文件
2 KB
subdir_vars.mk
7/31/2021 6:37 PM
生成文件
1 KB
project0_ccs.map
11/2/2020 4:45 PM
链接器地址映射
21 KB
内存分配文件

➢作业

- 1. 在力扣或者牛客网做三道简单C语言算法题
- 2. 要求在自己电脑IDE上做，调试完成后再提交到力扣上检验，不可直接用力扣的在线IDE做。
- 3. 提交作业需包括本地IDE代码、结果截图，力扣运行截图。



1.2.5 计算机中的数据表示

在计算机内部，数据都是以二进制的形式存储和运算的。

➢ 位

位(bit)简写为b，音译为比特，是计算机存储数据的最小单位，是二进制数据中的一个位，一个二进制位只能表示0或1两种状态，要表示更多的信息，就得把多个位组合成一个整体，每增加一位，所能表示的信息量就增加一倍。

➢ 字节

字节(Byte)简记为B，规定一个字节为8位，即1Byte = 8bit。字节是计算机数据处理的基本单位，并主要以字节为单位解释信息。每个字节由8个二进制位组成。通常，一个字节可存放一个ASCII码，两个字节存放一个汉字国际码

➢ 字

字(Word)是计算机进行数据处理时，一次存取、加工和传送的数据长度。一个字通常由一个或若干个字节组成，由于字长是计算机一次所能处理信息的实际位数，所以，它决定了计算机数据处理的速度，是衡量计算机性能的一个重要标识，字长越长，性能越好。计算机型号不同，其字长是不同的，常用的字长有8位、16位、32位和64位。

- **机器数**
  - 在计算机内部，任何信息都以二进制代码表示(即0与1的组合来表示)。一个数在计算机中的表示形式，称为机器数。
- **真值**
  - 机器数所对应的原来的数值称为真值

1.2.6 计算机中的数据表示

- 原码
  - 原码表示法即用机器数的最高位代表符号(若为0，则代表正数，若为1，则代表负数)，数值部分为真值的绝对值的一种表示方法。
- 反码
  - 正数的反码不变，负数的反码是对应正数的原码取反
- 补码（计算机中使用的编码）
  - 正数的补码不变，负数的补码是其对应正数的反码+1。

1.2.7 计算机中的数据表示

十进制	+73	–73	+127	–127	+0	–0
二进制(真值)	+1001001B	–1001001B	+1111111B	–1111111B	+0000000B	–0000000B
原码	01001001B	11001001B	01111111B	11111111B	0000000B	10000000B
反码	01001001B	10110110B	01111111B	10000000B	00000000B	11111111B
补码	01001001B	10110111B	01111111B	10000001B	0000000B	00000000B

- 十进制与二进制之间的转换

例如把52换算成二进制数，计算结果如图：

```
2| 52 ……0
 2| 26 ……0
  2| 13 ……1
   2| 6 ……0
    2| 3 ……1
     1 ……1
```

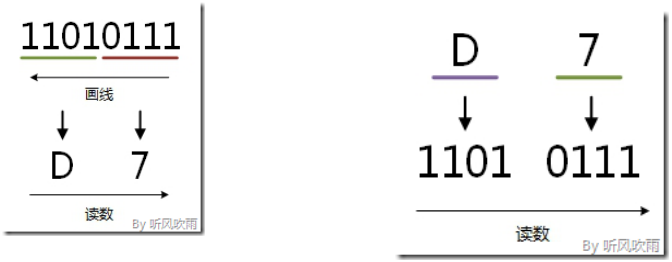
例如要把-52换算成二进制：

1.先取得52的二进制：00110100

2.对所得到的二进制数取反：11001011

3.将取反后的数值加一即可：11001100

- 二进制与十六进制之间的转换



- 十进制和十六进制转换

➢使用计算器进行数制转换

- MODE
- BASE-N
- SHIFT + DEC十进制
- SHIFT + HEX十六进制
- SHIFT + BIN二进制

1.2.8 计算机中的数据表示

- BCD码
  - 常用8421 BCD码。
  - Gray码(相邻的2个数只有一位不同)
- ASCII
  - 7位二进制编码。
  - 表示字母、数字字符和控制字符
- Unicode
  - 16位编码
  - 对世界上所有语言大部
  - https://unicode-table.c

- ASCII码表

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

C语言编程中所用的数据类型

数据类型	位数	范围（有符号）	范围（无符号）
char,int8_t,uint8_t	8	-128 – 127	0 – 255
short, int16_t, uint16_t	16	-32768 – 32767	0 – 65535
int, int32_t, uint32_t	32	-2^31 – 2^31-1	0 – 2^32 -1
long	32	-2^31 – 2^31-1	0 – 2^32 -1
Long long, int64_t, uint64_t	64	-2^63 – 2^63-1	0 – 2^64 -1
float	32	-3.403*10^38 – 3.403*10^38	
Double, long double	64	-1.798*10^308 – 1.798*10^308	

- 小数的表示方法：
  - 直观的小数表示法
  - 定点数
    - 约定数值的小数点固定在某一位置，称为定点表示法，简称为定点数。
  - 浮点数
    - 小数点位置可以任意浮动，称为浮点表示法，简称为浮点数

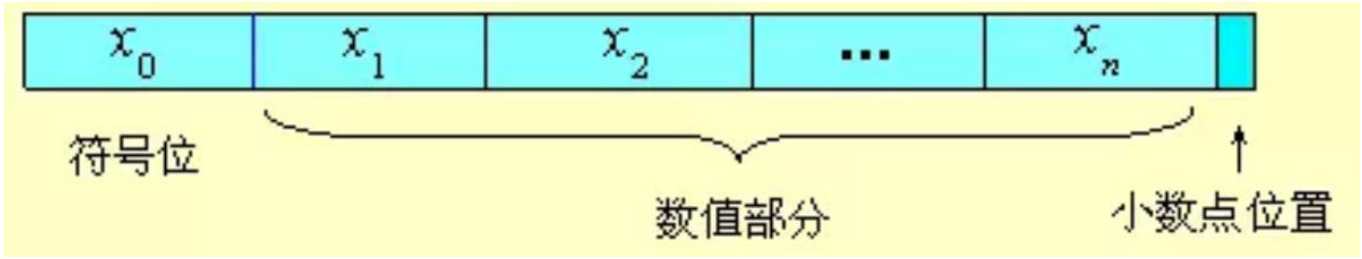
- 直观的小数表示法

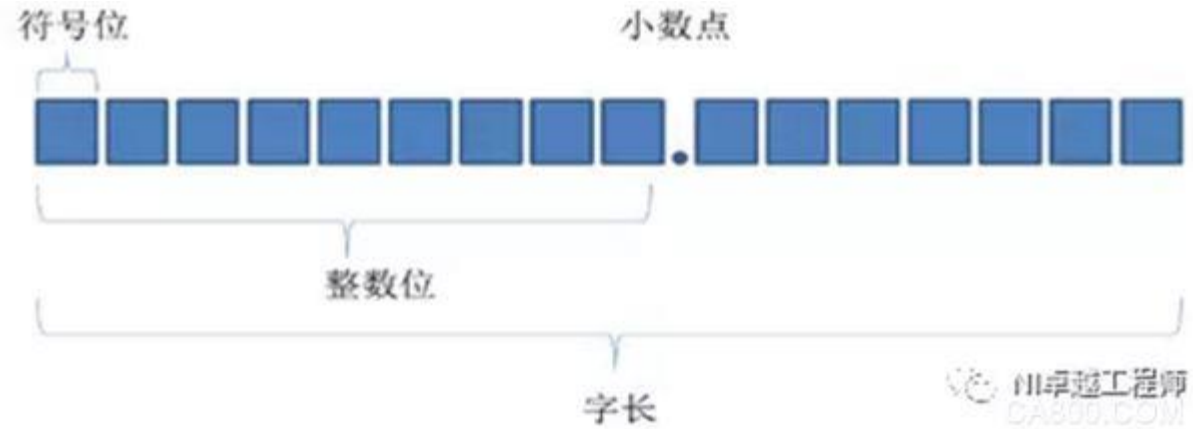
- 用10000表示10.000，10001表示10.001

- 用10000表示100.00，10001表示100.01

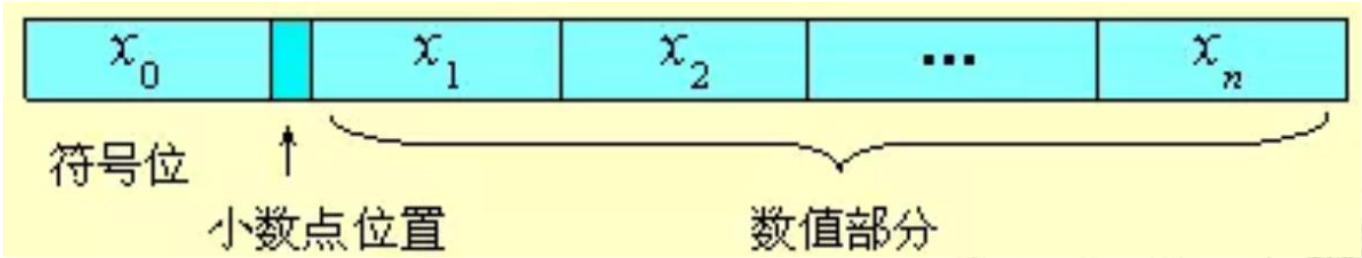
- ……
- 表示方法自己定义，不统一，注意取值范围
- 加减运算，不受影响
- 乘除运算，受影响，需要多余乘除法

➢定点数





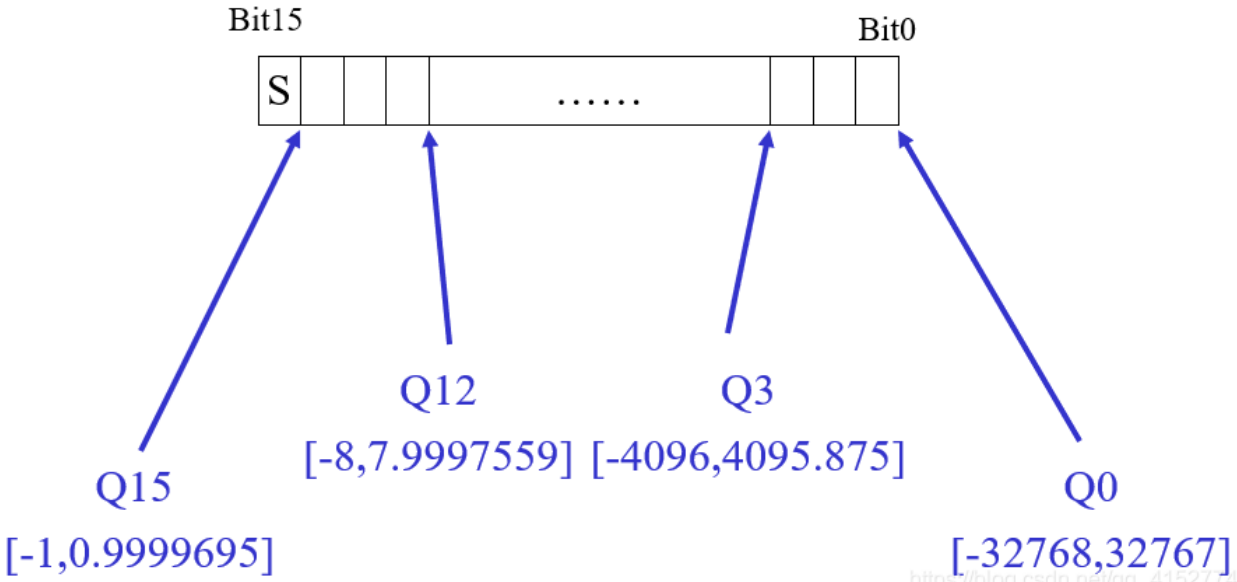
Q*f*：称作“Q 格式”，Qn 表示 n 个小数的位数。



➢定点数

数的定标：确定定点格式中小数点的位置，假设一个16位的有符号二进制数，Q=0，那么小数点就定标在LSB，也就是最低位（准确说应该是最低位的后面）；Q=3，那么就是小数点在从右往左数第四位和第三位之间。

数的定标——确定定点格式数据中小数点的位置



对于N位的有符号二进制数，用Q值法表示，范围是[-2^(N-Q-1), 2^(N-Q-1)- 2^(-Q)]
对于N位的无符号二进制数，用Q值法表示，范围是[0, 2^(N-Q)- 2^(-Q)]

Q格式

Q格式是一种定点小数表示方法，具体来说，二者的转换关系为：

**定点数 = 浮点数 * 2^Q **

Q15在两个字节下，Q15表示小数位有15位，最高位表示符号位。则表示的范围：[-1<X<0.9999695]。15位即为2的15次方为32768.所以我们将1放大了32768倍。其精度即为1/32768=0.000030517578125。因为15位数为0x7fff即32767，所以正数最大就是32767／32768=0.99996948≈0.9999695

在四个字节下的，第一位表示符号位，16位表示整数位，15位表示小数位。Q15的范围则为[-65536，65535.9999695]

在C2000中，统一使用32位来表示。QN格式对应数据类型为_iqN，表示使用N位来表示小数，其余32-N位表示整数，故Q0其实就是一般的32位整数。具体的Q1~Q30的取值范围及精度见下表：

Data Type	Range		Resolution/Precision
	Min	Max	
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015
_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

```
typedef   long    _iq12;   /* Fixed point data type: Q12 format      */
typedef   long    _iq11;   /* Fixed point data type: Q11 format      */
typedef   long    _iq10;   /* Fixed point data type: Q10 format      */
typedef   long    _iq9;    /* Fixed point data type: Q9 format       */
typedef   long    _iq8;    /* Fixed point data type: Q8 format       */
typedef   long    _iq7;    /* Fixed point data type: Q7 format       */
typedef   long    _iq6;    /* Fixed point data type: Q6 format       */
typedef   long    _iq5;    /* Fixed point data type: Q5 format       */
typedef   long    _iq4;    /* Fixed point data type: Q4 format       */
typedef   long    _iq3;    /* Fixed point data type: Q3 format       */
typedef   long    _iq2;    /* Fixed point data type: Q2 format       */
typedef   long    _iq1;    /* Fixed point data type: Q1 format       */
```

➢使用IQMath库进行运算

IQmath库是TI C28x系列DSP中使用的一个高度优化且高精度的数学库，用于使用定点算法实现浮点运算。

使用方法

添加库文件IQmath.lib或IQmath_f32.lib，后者用于带FPU单元的DSP；

包含头文件IQmathLib.h；

修改CMD文件，添加其中与IQmath相关的部分。

_IQ类型代表使用GLOBAL_Q定义的精度，GLOBAL_Q在IQmathLib.h文件中定义，默认为：

```
#ifndef GLOBAL_Q
#define GLOBAL_Q 24 /* Q1 to Q29 */
#endif
```

可直接更改IQmathLib.h文件中的定义，也可使用如下方法覆盖此定义：

```
#define GLOBAL_Q 21 /* Set the Local Q value */
#include <IQmathLib.h>
```

➢使用IQMath库进行运算

算数运算及性能评估表如下：

对于相同_IQN格式的两个数之间的加减法，可以使用+、-直接进行，注意不要溢出即可；对于不同_iqN格式的两个数，需要转换为相同_IQN格式后再进行加减。

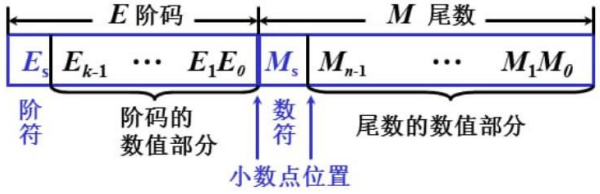
Function Name	IQ Format	Execution Cycles	Accuracy (in bits)	Program Memory (words)	Input format	Output format	Remarks
Arithmetic Functions							
IQNmpy	1-30	~ 6	32 bits	NA	IQN*IQN	IQN	INTRINSIC
IQNrmpy	1-30	17	32 bits	13 words	IQN*IQN	IQN	
IQNrsmpy	1-30	21	32 bits	21 words	IQN*IQN	IQN	
IQNmpyI32	1-30	~ 4	32 bits	NA	IQN*long	IQN	C-MACRO
IQNmpyI32int	1-30	22	32 bits	16 words	IQN*long	long	
IQNmpyI32frac	1-30	24	32 bits	20 words	IQN*long	IQN	
IQNmpyIQX		~ 7	32 bits	NA	IQN*IQN	IQN	INTRINSIC
IQNdiv	1-30	63	28 bits	71 words	IQN/IQN	IQN	

➢使用IQMath库进行运算

三角函数及性能评估表如下：

Function Name	IQ Format	Execution Cycles	Accuracy (in bits)	Program Memory (words)	Input format	Output format	Remarks
Trigonometric Functions							
IQNasin	1-29	154		82 words	IQN	IQN	Note A
IQNsin	1-29	46	30 bits	49 words	IQN	IQN	
IQNsinPU	1-30	40	30 bits	41 words	IQN	IQN	
IQNacos	1-29	170		93 words	IQN	IQN	Note A
IQNcos	1-29	44	30 bits	47 words	IQN	IQN	
IQNcosPU	1-30	38	29 bits	39 words	IQN	IQN	
IQNatan2	1-29	109	26 bits	123 words	IQN	IQN	
IQNatan2PU	1-29	117	27 bits	136 words	IQN	IQN	
IQatan	1-29	109	25 bits	123 words	IQN	IQN	

➢浮点数



$$N=M\times R^E$$

M_s	代表浮点数的符号
n	其位数反映浮点数的精度
k	其位数反映浮点数的表示范围
E_s 和 k	共同表示小数点的实际位置

尾数M：为定点小数，尾数的位数决定了浮点数有效数值的精度，尾数的符号代表了浮点数的正负，因此又称为数符。尾数一般采用原码和补码表示。

阶码E：为定点整数，阶码的数值大小决定了该浮点数实际小数点位置与尾数的小数点位置（隐含）之间的偏移量。阶码的位数（k）决定了浮点数的表示范围。阶码的符号叫阶符。

浮点数可以使用三部分表示：数符（S）,阶码（E）,尾数（M）。

阶码的底R：一般为2、8或16，且隐含规定。

例如：352.47 = 3.5247 * 10^2

178.125转化为二进制为 10110010.001，又可表示为：1.0110010001 乘以 2的111次方（111是7的二进制表示），其中10110010001 这部分被称作尾数(M)111这部分被称作阶码（E）正负被称作数符（S）：0表示正数， 1表示负数。

- 浮点数使用方便
- 运算非常复杂
- 需要特殊的浮点运算单元，一般只有高端CPU中才会有，成本较高，而且智能处理加法、减法和乘法。
- 其他数学计算，比如除法，开方，三角函数等，则计算更加复杂耗时。
- 定点数与浮点数之间相互转换，也消耗计算时间。

- 作业

1. 冯·诺伊曼计算机的基本设计思想是什么？哈佛结构的计算机，与冯·诺伊曼计算机相比，有哪些优缺点？
2. 什么是总线，总线通常有哪3组信号？各组信号的作用是什么？
3. 1）计算机的字长是什么含义？2）简述处理器中的流水线技术。
4. 将下列十六进制无符号整数，转换为十进制真值。
1) 0FFH　　2) 0H　　3) 5EH　　4) EFH
5. 如果上题中的十六进制数为8位有符号整数，请将其转换位十进制真值
6. 将下列十进制数转换位压缩BCD码
1）12　　2）24　　3）68　　4）99
7. 将下列二级制补码表示的有符号整数转换为十进制真值
1）0000 0000b　2）0111 1111b　3）1000 0001b　4）1100 0111b