# 计算机组成原理

## 第一章 计算机系统概论

### 1.1 计算机系统简介

计算机系统由软件和硬件组成，软件又分为系统软件和应用软件。

将高级语言程序翻译成机器语言程序的软件称为翻译程序。翻译程序分为：

1. 编译程序

将源程序一次全部翻译成机器语言程序，而后再执行机器语言程序。因此，只要机器源程序不变，就无需再次进行翻译。

1. 解释程序

将源程序的一条语句翻译成对应于机器语言的一条语句，并且立即执行这条语句，如此重复直至完成源程序的全部翻译任务。特点是翻译一次执行一次。

五级层次结构：

1. 虚拟机器（高级语言机器）

用编译程序翻译成汇编语言程序或其他中间语言程序

1. 虚拟机器（汇编语言机器）

用汇编程序翻译成机器语言程序

1. 虚拟机器（操作系统机器）

用机器语言解释操作系统，控制并管理计算机系统全部硬件和软件资源

1. 传统机器（机器语言机器）

用微程序解释机器指令

1. 微程序机器（微指令系统）

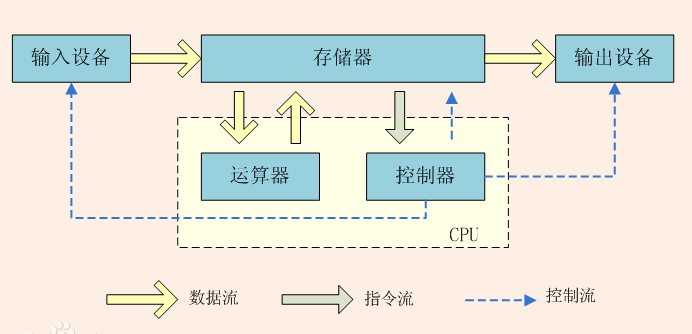
由硬件直接执行微指令

### 1.2 计算机的基本组成

#### 1.2.1 Von Neumann计算机

特点：

1. 计算机由运算器、存储器、控制器、IO设备五大部件组成；
2. 指令和数据以同等地位存放于存储器中，并可按地址访问；
3. 指令和数据均用二进制数表示；
4. 指令由操作码和地址码组成，操作码用来表示操作的性质，地址码用来表示操作数在存储器中的位置；
5. 指令在存储器内按顺序存放，通常是按顺序执行的
6. 机器以运算器为中心（现代计算器以存储器为中心）



现代计算机可认为由三大部分组成：

1. CPU Central Processing Unit 中央处理器
2. I/O设备 Input/Output equipment
3. 主存储器 Main Memory

#### 1.2.2 计算机工作步骤

1. 主存储器

作用是存放指令和数据，并能由CPU直接随机存取

包括存储体M，各种逻辑部件及控制电路等。

存储体由许多**存储单元**组成，每个存储单元包含若干个存储原件，每个存储原件能存放一位二进制代码。一个存储单元存储一串二进制代码，称这串二进制代码为一个**存储字**，这串二进制代码的位数称为**存储字长**。存储字长、指令字长、数据字长三者可不相同，但必须是字节的整数倍。

主存的工作方式：按照存储单元的地址号来实现对存储字各位的存取。称为按地址存取方式。为了实现按地址访问的方式，主存中必须配备两个寄存器MAR和MDR。

1. **MAR** Memory Address Register 存储器地址寄存器

用来存放欲访问的存储单元的地址，其位数对应存储单元的个数

1. **MDR** Memory Data Register 存储器数据寄存器

用来存放从存储体某单元取出的代码或者准备往某存储单元存入的代码，其位数与存储字长相等。

随着硬件技术的发展，主存都制成大规模集成电路的芯片，而将MAR和MDR集成在CPU之中。

1. 运算器

至少包括3个寄存器（现代计算机内部往往设有通用寄存器组）和一个算数逻辑单元ALU。

一般有ACC（Accumulator）累加器，MQ（Multiplier-Quotient Register）乘商寄存器，X 操作数寄存器。

1. 控制器

计算机的神经中枢，由它指挥各部件自动、协调的工作。工作流程：

1. 取指过程：

命令存储器读出一条指令

1. 分析阶段：

对指令进行分析，指出该指令要完成什么样的操作，并按寻址特征指明操作数的地址

1. 执行阶段：

根据操作数所在的地址以及指令的操作码完成某种操作

控制器由**程序计数器、指令寄存器、控制单元**组成

* + - 1. PC Program Counter 程序计数器

存放当前欲执行指令的地址，与MAR之间有一条直接通路

* + - 1. IR Instruction Register 指令寄存器

用来存放当前的指令，内容来自MDR

* + - 1. CU Control Unit 控制单元

用来分析当前指令完成所需的操作，发送微操作指令序列

1. I/O

### 1.3 计算机硬件的主要指标

1． **机器字长**

指CPU一次能处理数据的位数，通常与CPU的寄存器位数有关。也将直接影响加法器（或ALU）、数据总线、存储字节的位数。

2． **存储容量**

包括主存容量和辅存容量。

主存容量指主存中存放二进制代码的总位数，存储容量=存储单元个数\*存储字长

3. **运算速度**

现在普遍采用单位时间内执行指令的平均条数来衡量，并用**MIPS**作为计量单位

**MIPS**： Million Instruction Per Second 百万条指令每秒

**CPI**： Cycle Per Instruction 单条指令所需的时钟周期

**FLOPS**: Floating Point Operation Per Second 浮点运算次数每秒

## 第二章 计算机的发展及应用

1. 第一代电子管计算机
2. 第二代晶体管计算机
3. 第三代集成电路计算机
4. 微型计算机

## 第三章 系统总线

### 3.1 总线的基本概念

计算机系统的五大部件之间的互连方式有两种：

1. 分散连接 各部件之间使用单独的连线
2. 总线连接 将各部件连到一组公共信息传输线上

总线是连接多个部件的信息传输线，是各部件共享的传输介质。

在某一时刻，只允许有一个部件向总线发送信息，而多个部件可以同时从总线上接收相同的信息。

### 3.2 总线的分类

1. **片内总线**

芯片内部的总线，比如在CPU芯片内部，寄存器与寄存器之间、寄存器与算数逻辑单元ALU之间都由片内总线连接

2． **系统总线**

指CPU、主存、I/O设备各大部件之间的信息传输线

1. **数据总线**

传输各功能部件之间的数据信息

数据总线宽度是衡量系统性能的一个重要参数

1. **地址总线**

指出数据总线上的源数据或目的数据在主存单元的地址或I/O设备的地址

地址总线的位数与存储单元的个数有关

1. **控制总线**

用来发出各种控制信号的传输线，控制各部件在不同时刻获得总线使用权。

还能监视各部件状态，如查询该设备处于忙还是闲

3. **通信总线**

用于计算机系统之间或与其他系统之间通信

### 3.3 总线特性及性能指标

#### 3.3.1 总线特性

* 机械特性
* 电气特性
* 功能特性
* 时间特性

#### 3.3.2 总线性能指标

1. 总线宽度

指数据总线的根数，如8位、32位

1. 总线带宽

单位时间内总线上传输数据的位数 MBps

1. 时钟同步/异步
2. 总线复用

一条信号线上分时传送两种信号，多路复用技术

1. 信号线数

地址总线、数据总线和控制总线三种总线数的综合

1. 总线控制方式
2. 其他指标

如负载能力、电源电压、总线宽度能否扩展

#### 3.3.3 总线标准

系统与各模块、模块与模块之间的一个互连的标准界面。界面的任一方只需根据总线标准的要求完成自身一方接口的功能要求，而无须了解对方接口与总线的连接要求。

ISA Industrial Standard Architecture

EISA Extended Industrial Standard Architecture

VESA Video Electronic Standard Association

**PCI** Peripheral Component Interconnect

AGP Accelerated Graphics Port

USB Universal Serial Bus

### 3.4 总线结构

#### 3.4.1 单总线结构

将CPU、主存、I/O设备都挂在一组总线上

#### 3.4.2 多总线结构

将速度较低的IO设备从单总线上分离出来，形成主存总线与IO总线分开的结构

### 3.5 总线控制

#### 3.5.1 总线判优控制

总线上连接的各类设备按其对总线有无控制功能分为主设备和从设备。从设备只能响应从主设备发来的总线命令。当多个主设备同时要使用总线时，就由总线控制器的判优、仲裁逻辑按一定的优先等级顺序确定哪个主设备能使用总线。

总线判优控制可分为集中式和分布式，集中式将控制逻辑集中在一处（如CPU），分布式将控制逻辑分散在与总线连接的各个部件或设备上。

1. 链式查询

按链式方式查询各个接口是否有总线请求，缺点是优先级低的设备很难获得请求

1. 计数器定时查询

是一种循环方法，通过发出计数值，使部件获得使用权

1. 独立请求方式

响应速度快，优先次序控制灵活

#### 3.5.2 总线通信控制

总线周期可分为四个阶段：

1. 申请分配阶段

由需要使用总线的主模块提出申请，经总线仲裁机构决定下一传输周期的总线使用权授予某一申请者

1. 寻址阶段

取得了使用权的主模块通过总线发出本次要访问的从模块的地址及相关命令，启动参与本次传输的从模块

1. 传数阶段

主模块和从模块进行数据交换

1. 结束阶段

主模块的有关信息均从系统总线上撤除，让出总线控制权

总线通信控制主要解决通信双方如何获知传输开始和传输结束，以及通信双方如何协调如何配合。通常有四种方式：同步通信、异步通信、半同步通信、分离式通信

1. 同步通信

通信双方由同一时标控制数据传送，

优点是明确、统一，配合简单一致

缺点是属于强制性同步，会受到木桶效应的影响

1. 异步通信

采用握手方式进行通信，分为：

1. 不互锁方式

主模块发出请求信号不必等待从模块的回答信号，而是经过一段时间确认从模块已收到请求信号后便撤销其请求信号

如CPU向主存写信息

1. 半互锁方式

主模块发出请求信息必须等待从模块的回答信号后再撤销其请求信号

如多机系统中某个CPU访问共享存储器

1. 全互锁方式

在半互锁方式的基础上，从模块发出的回答信号必须确认主模块请求信号已撤销才能撤销其回答信号

如网络通信

1. 半同步通信

加入WAIT响应信号线，保留同步通信的基本特点，但是通过WAIT响应信号线采用插入时钟等待周期的措施来协调通信双方的配合问题

1. **分离式通信**

基本思路：将一个传输周期分解为两个子周期。

在第一个子周期中，主模块A在获得总线使用权后将命令、地址、自身编号以及其他有关信息发送到系统总线上，被系统从模块B接收下来，然后放弃总线使用权。

在第二个子周期，当B模块收到A模块发来的命令信号后，经过内部操作将A模块需要的数据准备好，申请总线使用权，发送信息。

特点是：

1. 各模块欲占用总线使用权都需提出申请
2. 得到使用权后，模块在限定的时间内向对方传送信息，采用同步方式传送，不等待对方的应答信号
3. 各模块在准备数据的过程中都不占用总线，使总线可接受其他模块的请求
4. 总线被占用时都在做有效工作，不存在空闲等待时间。

## 第四章 存储器

### 4.1 概述

存储器性能成为计算机系统的核心，因为CPU速度现在惊人的高，而存取器的速度与它很难适配，而且大多IO都是与存储器直接交互。

**RAM** Random Access Memory 随机存储器

存储器的任意一个存储单元的内容都可以随机存取，而且存取时间与存储单元的物理位置无关。计算机中的主存都采用这种随机存储器。

与之相对的有串行访问存储器，对存储单元进行读写时，需按其物理位置的先后顺序寻找地址。

**ROM** Read Only Memory 只读存储器

通常存放固定不变的程序、常数等，或用于操作系统的固化，与RAM一起作为主存的一部分，同一构成主存的地址域。

按照在计算机系统中的作用不同，存储器主要分为主存储器、辅助存储器、缓冲存储器

1. **主存储器**

直接和CPU交换信息

1. **辅助存储器**

用来存放当前暂时不用的程序和数据

1. **缓冲存储器 Cache**

用在两个速度不同的部件之中，比如CPU和主存之间，它的速度快于主存

### 4.2 主存储器

#### 4.2.1 简介

主存与MAR、MDR通信，完成CPU发出的读写操作

主存的主要技术指标是**存储容量**和**存储速度**

1. 存储容量

=存储单元个数\*存储字长

1. 存储速度

Memory Access Time 访问时间，指执行一次存储器操作使用的时间

Memory Cycle Time 存取周期，指连续两次独立的存储器操作所需的最小间隔时间

1. 存储器带宽

表示单位时间内存储器存取的信息量 字节/秒

提高存储器带宽的方式：

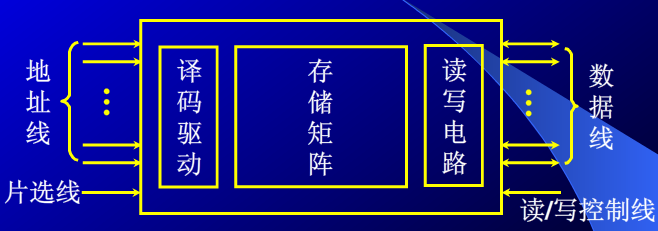
缩短存取周期，增加存储字长（使每个周期可读写更多二进制位数），增加存储体

#### 4.2.2 半导体存储芯片

在一个芯片内集成具有记忆功能的存储矩阵、译码驱动电路和读/写电路等。

通过地址总线，数据总线和控制总线与外部连接。

其中地址线是单向输入的，数据线是双向的，地址线和数据线的位数反映存储芯片的容量。如10根地址线和4根数据线芯片容量为4K位。



译码驱动方式有线选法和重合法。

线选法是一根字选择先选中一个存储单元；重合法是把线分为两份，一份确定x坐标，一份确定y坐标，由x,y坐标得出存储单元。

#### 4.2.3 RAM

* **静态RAM** Static RAM

电源掉电时，原存信息丢失，属于易失性半导体存储器

* **动态RAM** Dymamic RAM

#### 4.2.4 ROM

存放系统程序、标准子程序和各类常数等。

#### 4.2.5 存储器与CPU的连接

a) 存储容量的扩展

单片存储芯片容量有限，故必须将若干存储芯片连在一起才能组成足够容量的存储器。

位扩展

字扩展

b) 存储器与CPU的连接

特别要注意片与片之间的地址线、数据线和控制线的连接。

#### 4.2.6 存储器的校验

##### 4.2.6.1 汉明码

任何一种编码是否具有检测能力和纠错能力，与**编码的最小距离**有关。编码的最小距离，是指任意两组合法代码之间的最少二进制位数的差异。

L – 1 = D + C，且D>=C

L为码距 D为检测错误位数 C为纠正错误位数

即编码最小距离L越大，其检验错误的位数D越大，纠正错误的位数C也越大。

假设欲检测的二进制代码为n位，为使其具有纠错能力，需添加k位检测位，组成n+k位的代码。为了能准确对错误定位及指出代码没错，新增添的检测位数k应满足：

2^k >= n + k +1

###### 汉明码的组成

检测位放置在1,2,4,8……上，这些检测位的位置设置是为了保证他们能分别承担n+k位信息中不同数位所组成的小组的奇偶检测任务，使检测位和它们所负责的小组中1的个数为奇数或偶数。

如C1的小组包含1,3,5,7等，C2的小组包含2,3,6,7等

###### 汉明码的纠错过程

汉明码的检错过程实际上是对传送后的汉明码形成新的检测位Pi，根据Pi的状态找出错误的位置。Pi = Ci异或所有其组内成员。

汉明码常常被用在纠错一位的场合，若欲实现检错两位，使用时还得增添一位检测位。

#### 4.2.7 提高访存速度的措施

主存的存取速度已经成为计算机系统的瓶颈。

1. **单体多字系统**

由于程序和数据在存储体内是连续存放的，因此CPU访存取出的信息也是连续的，如果可以在一个存取周期内，从同一地址取出多条指令，再逐条将指令送至CPU执行，就增大了存储器的带宽，提高了单体存储器的工作速度。

1. **多体并行系统**

采用多体模块组成的存储器，每个模块都相互独立能并行工作又能交叉工作。当一个体与CPU交换信息时，另一个体可以同时与外部设备进行直接存储器访问。

**高位交叉编址**的存储器（**顺序存储**）因为一个体内的地址是连续的，有利于扩充容量，连续读取n个字所需的时间t1为T+(n-1)\*t

**低位交叉编址**的存储器（**交叉存储**）有利于提高存储器的带宽，连续读取n个字所需的时间为nT

1. **高性能存储芯片**

### 4.3 高速缓冲存储器

#### 4.3.1 概述

程序访问的局部性和CPU相对于主存的性能优势迫使Cache的出现。

采用映射的方式简历主存与缓存的连接。具体做法是：

主存 m | b

缓存 c | b

一般而言，Cache容量越大，其CPU的命中率就越高。但是当Cache的容量达到一定值时，命中率已不因容量的增大而又明显的提高。如4GB一般配32KB的Cache命中率即可达95%

##### 基本结构

1. **Cache存储体**

以块为单位与主存交换信息，为加速Cache与主存的调动，主存大多采用多体结构，且Cache访存的优先级最高

1. **地址映射变换机构**

将CPU送来的主存地址转换为Cache地址，主要是将主存的块号（高位地址）与Cache块号间的转换

1. **替换机构**

若Cache内容已满，无法接受来自主存块的信息时，就从Cache内的替换机构按一定的替换算法来确定应从哪个Cache块返回主存

1. **Cache的读写操作**

对Cache块写入的信息，必须与被映射的内存块内的信息完全一致，一般采用以下几种策略：

1. Write-through Store-through

写操作时数据既写入Cache又写入主存

1. Write-back Copy-back

写操作时只把数据写入Cache而不写入主存，但当Cache数据被替换出去时才写回主存。为了识别Cache中的数据是否与主存一致，Cache中的每一块要增设一个标志位，表示清或浊。当然多处理器中这种策略会引起一些问题。

##### Cache的改进

1. **多级缓存**
2. **统一缓存**和**分立缓存**

选择策略与主存结构与机器对指令的执行控制方式有关

* 1. 统一缓存是指指令和数据都存放在统一缓存内的Cache
  2. 分立缓存是指指令和数据分别存放在两个缓存中（流水线、超前控制）

#### 4.3.2 Cache-主存地址映射

##### 直接映射

每个主存块只与一个缓存块相对应。

记得i为缓存块号，j为主存块号，C为缓存块数

有 i = j mod C

实现简单，只需要利用主存地址的某些位即可直接判断，主存被分为 t | c | b其中c | b对应缓存。

缺点是不够灵活，因为每个主存块只能固定的对应某个缓存块，即使缓存内还空着很多位置也不能占用。

##### 全相联映射

允许主存中的每一字块映射到Cache中的任何一块位置上

只适合小容量的Cache使用，虽然命中率高，空间利用率高但是速度慢。

##### 组相联映射

对直接映射和全相联映射的一种折中。把Cache分为Q组，每组有R块，并有以下关系：

i = j mod Q

i为缓存的组号，j为主存的块号。

#### 4.3.3 替换策略

1. FIFO

2. LRU Least Recently Used

### 4.4 辅助存储器

#### 4.4.1 概述

与主存相比，辅助存储器容量大、速度慢、价格低、可脱机保存信息。

##### 4.4.2 循环冗余校验码

**Cyclic Redundancy Check**，循环冗余校验码，发现并纠正信息在存储或传送过程中连续出现的多位错误代码。是基于**模2运算**而建立编码规律的校验码。

##### CRC码的编码方式

CRC码就是用多项式M(x)\*x^k,除以生成多项式G(x)(产生校验码的多项式)，所得余数作为校验位。为了得到k位余数作为校验位，G（x）必须是k+1位。

##### CRC码的译码和纠错

将收到的循环校验码用约定的生成多项式G(x)去除，如果无措，则余数应该为0。

并不是任何一个k+1位多项式都可以作为生成多项式。从检错和纠错的要求出发，生成多项式应满足以下要求：

1. 任何一位发生错误，余数都不为0
2. 不同位发生错误应使余数不同
3. 对余数继续做模2除，应使余数循环

## 第五章 输入输出系统

### 5.1 程序中断方式

中断服务程序的流程：

1. **保护现场**

由中断隐指令保存程序的断点

由中断服务程序保存通用寄存器和状态寄存器的内容

1. **中断服务**
2. **恢复现场**
3. **中断返回**

### 5.2 DMA方式 Direct Memory Access

采用了外设和内存直接交换数据的方式，不通过CPU也不需要CPU暂停现行程序为设备服务。

由于DMA接口和CPU共享主存，就有可能出现两者争用主存的冲突，通常采用如下三种方法：

1. 停止CPU访问主存
2. 周期挪用、周期窃取
3. DMA与CPU交替访问

## 第六章 计算机的运算方法

### 6.1 无符号数和有符号数

计算机中的数均放在寄存器中，通常称寄存器的位数为机器字长。

1. **原码**

带符号的绝对值表示

1. **补码**

取反加一

因为0在补码中只有一种表示形式，所以补码比原码和反码多表示一个负数

1. **反码**

通常用来作为由原码求补码或者由补码求原码的中间过渡

**反码中的0有两种表示形式**

1. **移码**

移码就是在真值上加一个常数2^n

同一个真值的移码和补码仅差一个符号位

共同点：

1. 三种机器数的最高位均为符号位；
2. 当真值为正时，原码、补码、反码的表示形式均相同，符号位为0，数值部分与真值相同
3. 当真值为负时，原码、补码、反码表示形式不同，反码是原码的按位取反，补码是原码的取反加一

### 6.2 数的定点表示和浮点表示

#### 6.2.1 小数点的定点表示

小数点固定在某一位置的数称为定点数

#### 6.2.2 浮点表示

浮点数被表示为N = s \* r^j

S为尾数（可正可负）,j为阶码(可正可负)，r是基数。在计算机中，r可取2、4、8等

为了提高数据精度以及便于浮点数的比较，规定浮点数的尾数用纯小数表示。将尾数最高位为1的浮点数称为规格化数，浮点数表示成规格化形式后，精度最高。

##### 浮点数的表示形式

浮点数由**阶码j**和**尾数S**组成

j | S

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 符号位S | 阶码j | 尾数S | 总位数 |
| 短实数 | 1 | 8 | 23 | 32 |
| 长实数 | 11 | 52 | 64 |
| 临时实数 | 15 | 64 | 80 |

阶码用移码表示，真值都被加上一个常数（偏移量），尾数部分都规格化表示（非0的有效位最高位总是1。但是在IEEE标准中，1.XXXXX，小数点前的1默认被省略。

##### 浮点数的表示范围

浮点数可能发生上溢和下溢

1. 上溢： 浮点数阶码大于最大阶码，通常进行中断溢出处理
2. 下溢： 浮点数阶码小于最小阶码，通常按0处理

### 6.3 定点运算

#### 移位

符号位不变。

|  |  |  |
| --- | --- | --- |
| 真值 | 码制 | 添补代码 |
| 正数 | 原码、反码、补码 | 0 |
| 负数 | 原码 | 0 |
| 补码 | 左移添0 |
| 右移添1 |
| 反码 | 1 |

有符号数的移位称为**算术移位**

无符号数的移位称为**逻辑移位**

#### 加减法

现代计算机中都采用补码作加减法运算。

##### 溢出判断

1. **一位符号位判断溢出**

实际参加操作的两个数符号相同，结果符号与之不同，则为溢出

为了节省时间，通常用符号位产生的进位与最高有效位产生的进位异或操作后，按结果进行判断

1. **两位符号位判断溢出**

当2位符号位不同时，表示溢出。最高位符号位永远代表真正的符号。

在采用双符号位方案时，寄存器或主存中的数仍然只需要1位符号位，但是传递到加法器时要传递两次

##### 乘法运算

1. 乘法运算可用移位和加法来实现，起始状态部分积为0
2. 由乘数的末位值确定被乘数是否与原部分积相加，然后右移一位，形成新的部分积；同时，乘数也右移一位，由次低位作新的末位，空出最高位放部分积的最低位；
3. 每次做加法时，被乘数仅仅与原部分积的高位相加，其低位被移至乘数所空出的高位位置

需要加速可用原码二位乘，用两位乘数的状态来决定新的部分积如何形成。