计算机组成原理

作者

梁嘉朗　赵乾皓

王娜　眭思悦

苏庆栋　高萁

刘元　任腾龙

董作岭　傅显坤

审核校对：王渊均　排版：沈珈莹

版本号：V2.0

此版本包括计组第 1、4、5、6、7、8、9、10 章以及补充的数字逻辑部分，

同时向 1.0 版本的作者徐卫霞、杜泽林、孙吉鹏致敬

修订时间：2020.1

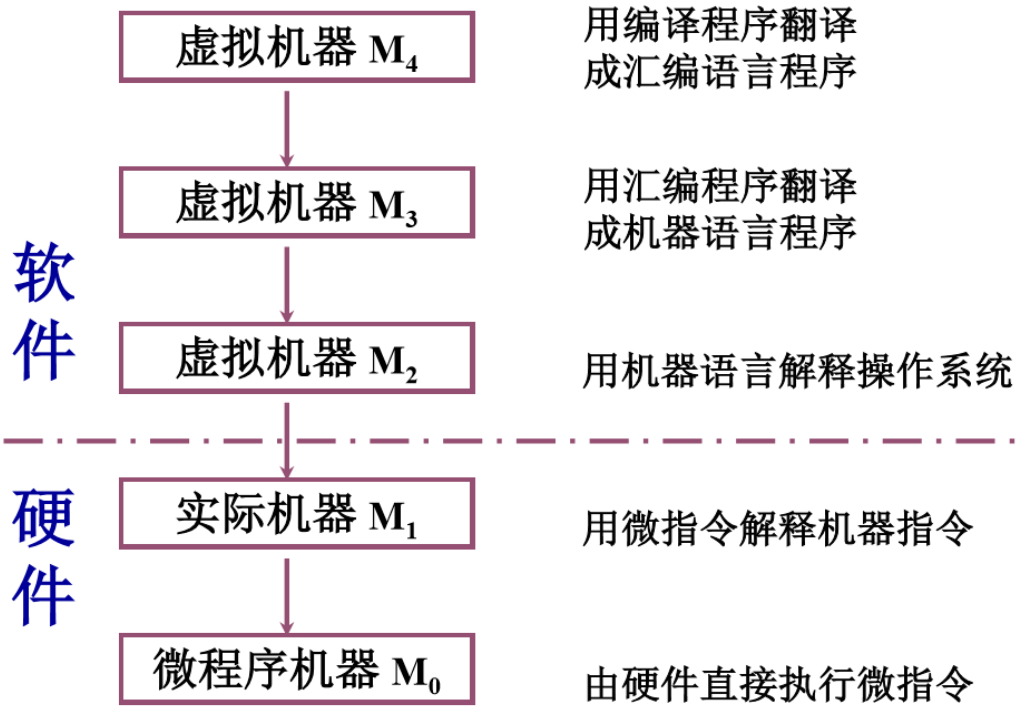
第1章 计算机系统概论

本章概述

计算机系统是由**硬件**和**软件**两大部分组成的，而这本书也是围绕着两个部分进行讲解的。在硬件部分，我们需要了解计算机各个部件的结构。在软件部分，我们需要了解程序如何进行，也就是指令和数据如何发挥作用。

1.1 计算机系统简介

- 如下图，我们可以看到现代计算机系统的多级层次结构



- 在这里我们介绍上图的五个机器分别的功能和运作：
  最顶层的虚拟机器M_4是**高级语言机器**，它的任务是将高级语言翻译成汇编语言，再将汇编语言送入机器M_3。
  虚拟机器M_3是**汇编语言机器**，当M_2接收到汇编语言后，把汇编语言翻译成机器语言程序，送入下一级的M_2。
  虚拟机器M_2是**操作系统机器**。操作系统接收到上一级传送的机器语言后，机器语言可通过操作系统实现控制和管理计算机软件和硬件资源的操作。
  实际机器M_1为**机器语言机器**，它的任务是接收从操作系统送来的机器指令，并用更详细的**微指令**解释机器指令。关于机器指令，我们会在第七章进行讨论，而关于微指令我们会在本书第十章详细介绍。
  M_0为**微程序机器**。它的任务是接收微指令，并通过硬件执行微指令。

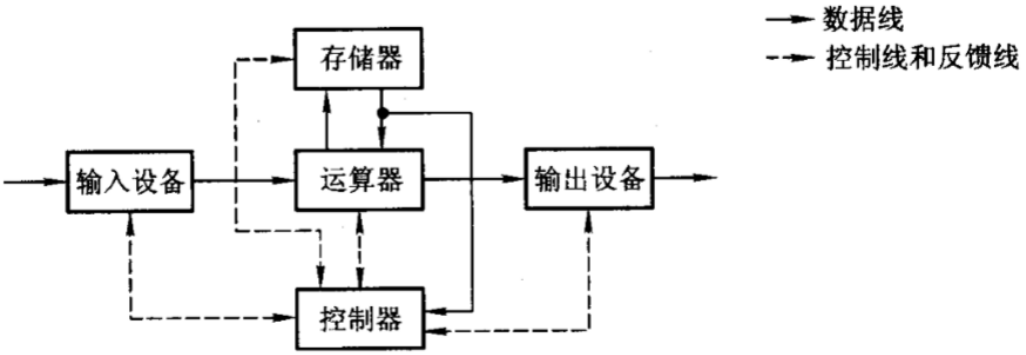
在这本书中，我们主要讨论的是从实际机器M1和微程序机器M0的组成原理以及设计思想。

1.2 计算机的基本组成

1.2.1 冯诺依曼计算机的特点

- 计算机由运算器、存储器、控制器、输入设备和输出设备五大部件组成。
  指令和数据以同等的地位存放于存储器内，并可按地址寻访。
  指令和数据均用二进制数表示。
  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。
  指令在存储器内按顺序存放。通常指令是按顺序执行的，在特定条件下，可根据运算结果或根据设定的条件改变执行顺序。**这是冯诺依曼机器的核心特征。**
  机器以运算器为中心，输入输出设备与存储器间的数据传送通过运算器完成。

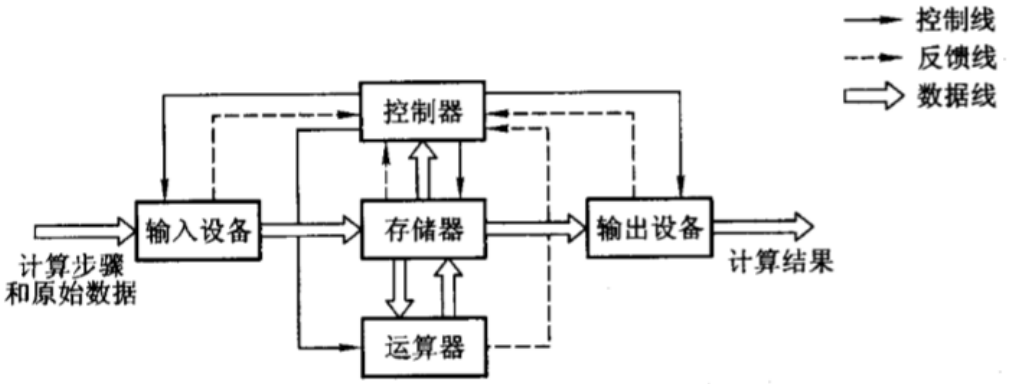
- 由下图可见，这是典型的冯诺依曼计算机结构框图。这个机器以运算器为核心，数据的输入输出必须要经过运算器，这使得运算器的工作变得繁忙，最终导致运算器成为计算机系统的瓶颈。



1.2.2 计算机的硬件框图

- 由上面典型的冯诺依曼计算机系统我们看到，运算器成为了制约计算机系统的瓶颈。因此我们对其进行了结构上的改进，于是就有了以存储器为中心的计算机结

构。

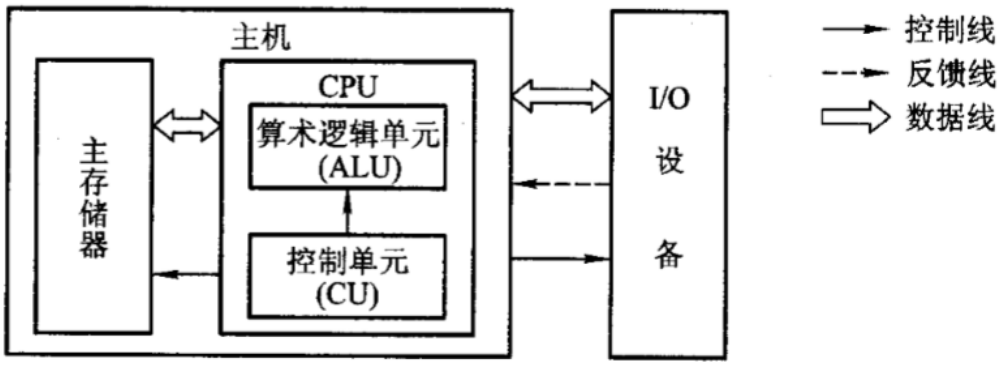


- 在这里我们可以看到，冯诺依曼计算机的五大部件，分别是**运算器**、**存储器**、**控制器**、**输入设备和输出设备**。

1.2.3 现代计算机的结构

- 现代计算机系统经过多年的发展，运算器和控制器到后面就集成在同一芯片上，它们合起来之后就统称**中央处理器(CPU)**。另外，我们把输入输出设备也统称为**I/O设备**。

- 如下图所示，现代计算机可认为由三大部分组成：**CPU**、**I/O设备**以及**主存储器**。CPU和主存合起来可称为**主机**，I/O设备可称为**外部设备**。

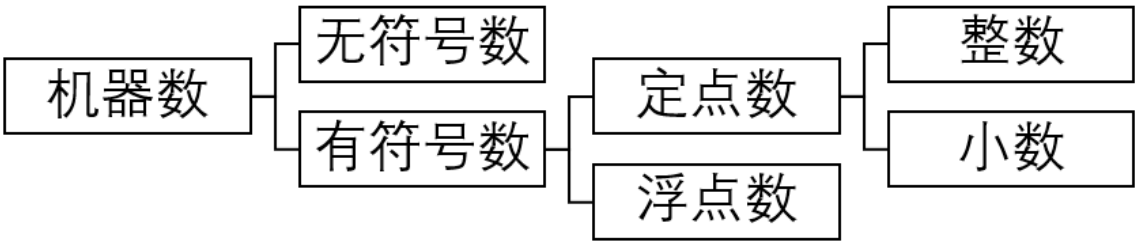


- 下面我们来依次介绍这五大部件的结构和功能。下面的介绍只是大致的介绍，目的是帮助同学们对计算机各部件的结构及功能有大致的了解，具体的介绍及细节请同学们翻看书本或其他章节的知识见解。

1.2.4 运算器的结构与功能

- 运算器用来完成**算术运算**和**逻辑运算**，并将运算的中间结果暂时存在运算器里。而对于运算器的结构我们需要了解进位器的结构。

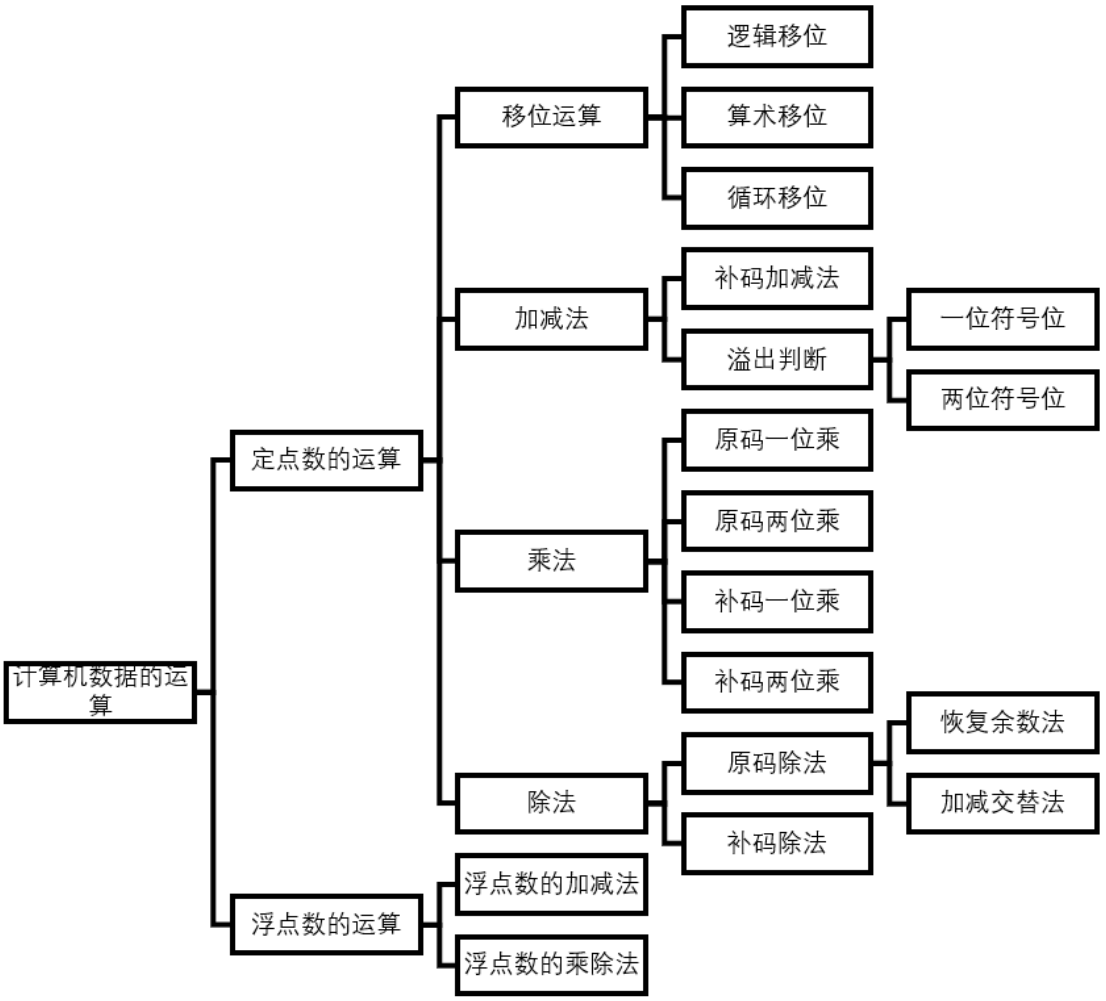
- 首先我们要了解计算机是如何表示数的，下图给出了各种数的分类以及要注意的概念。
  真值是**机器数**所代表的的真实值，机器数就是把真值的符号"数字化"的数。
  由于无符号数只能表示非负数，实际上可被有符号数代替。因为有符号数表示的范围更广，所有我们后面所有的讨论都是基于有符号数的。
  定点数表示的小数，只能表达绝对值小于1的小数。
  浮点数由**阶码**和**尾数**组成。为了提高浮点数的精度，其尾数必须为规格化数。浮点数在表示范围、数的精度、溢出处理和程序编程方面均优于定点数。但在运算规则、运算速度及硬件成本方面不如定点数。



- 除了计算机中数的分类，我们还要了解机器数的四种表示法：**原码**、**补码**、**反码**、**移码**。

- 了解完计算机如何表示数，下面我们来了解计算机中的运算。如下图为计算机数据运算的体系。
  因为补码加减运算规则简单且易于实现，因此定点数的加减法普遍采用补码的加减法。
  定点数的**乘法和除法**运算实际上是由多次的移位运算、加法运算和构成的。
  浮点数的加减法总共分五步进行，分别是**对阶**、**尾数求和**、**规格化**、**舍入**和溢出判

断。



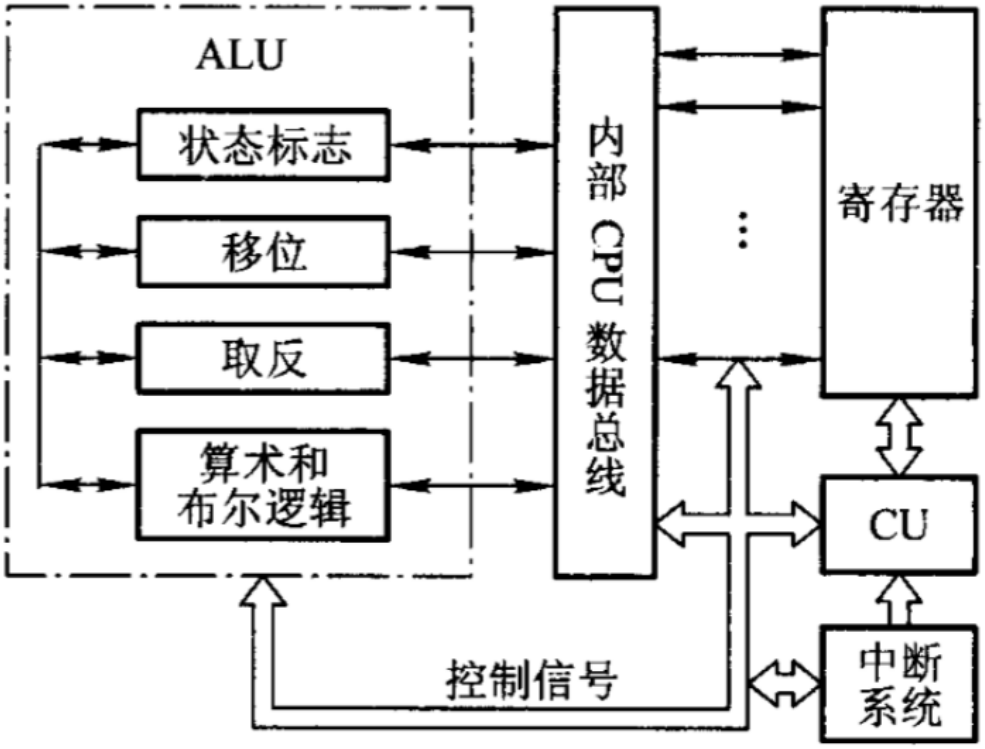
1.2.5 控制器的结构与功能

- 控制器用来控制指挥程序和数据的输入、运行以及处理运算的结果,而这里所指的的控制器就是**控制单元(CU)**。这里的控制器和运算器共同构成CPU。计算机之所以能自动协调的工作，是由于控制单元的统一指挥。

- 对计算机而言，一旦程序进入存储器后，就可由计算机自动完成取指令和执行指令的任务，控制器就是专用于此项工作的。控制器主要负责协调并控制计算机各部件执行程序的指令序列，其基本功能是**取指令**、**分析指令**和**执行指令**。

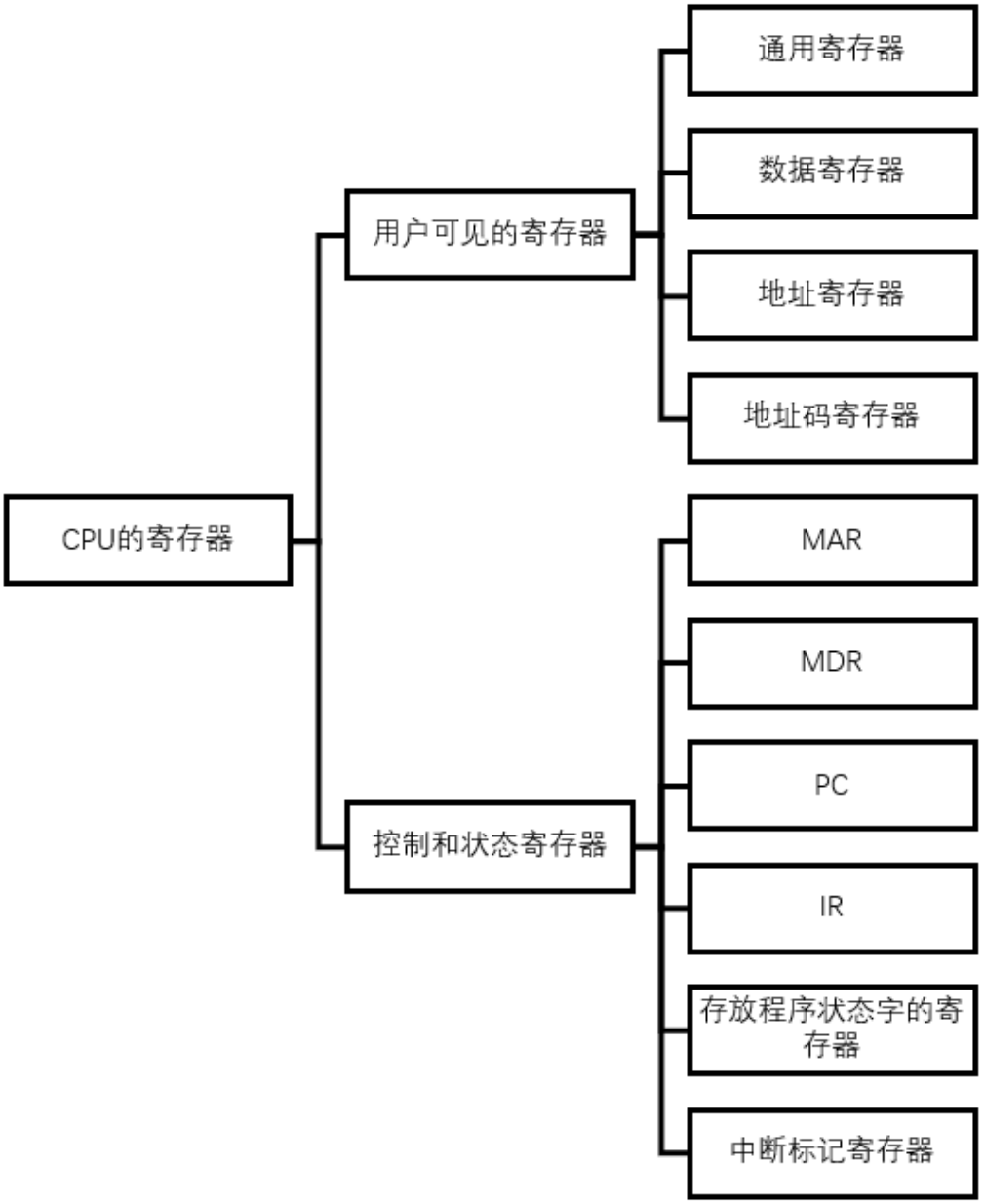
- 下图展示的是CPU的内部结构。**ALU**、**寄存器**、**中断系统**和**CU**分别构成了CPU的四大部分。
  ALU负责完成算术运算和逻辑运算，内含有存放操作数的寄存器。
  寄存器负责存放各种数据。
  中断系统负责处理异常情况和特殊请求。

CU负责对指令进行译码，并发出各种操作命令序列。



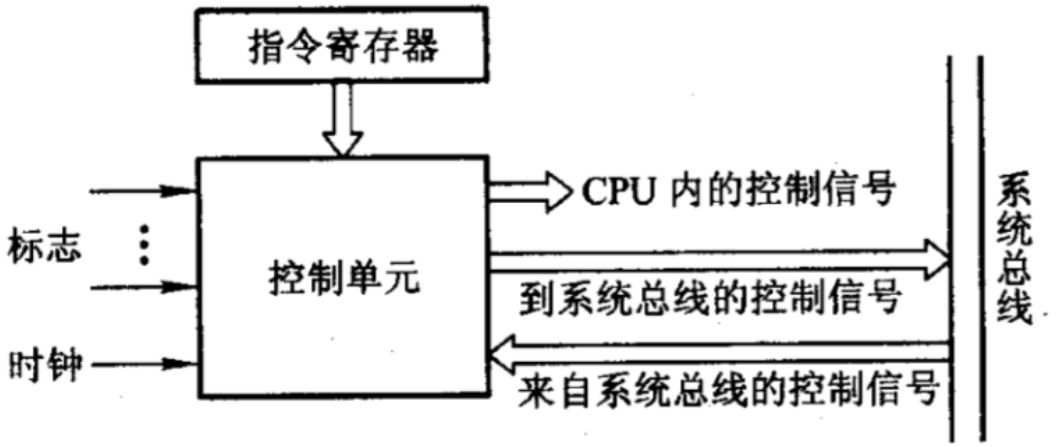
- CPU中的寄存器大致可分为两类：一类是**用户可见寄存器**，用户可对这类寄存器编程，通过优化让CPU因使用这类寄存器而减少对主存的访问次数。另一类属于**控制和状态寄存器**，用户不可对这类寄存器编程，它们被控制部件使用，以控制CPU的

操作，也可以被带有特权的操作系统程序使用，从而控制程序的执行。

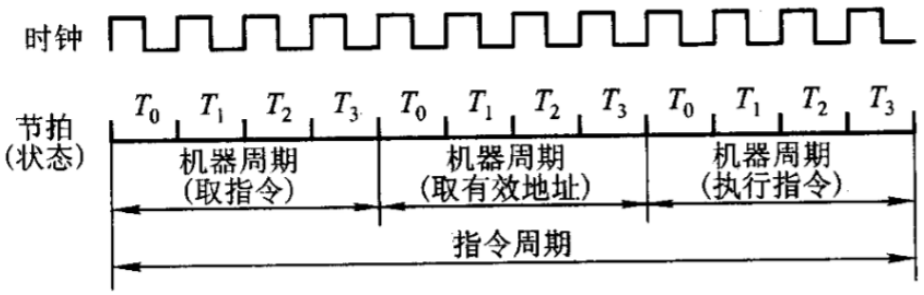


- 控制单元具有发出各种微操作命令(即控制信号)序列的功能。计算机执行程序的过程中，控制单元要发出各种微操作命令，不同的机器指令对应不同的指令。
  如下图，控制单元接收的输入信号有：**时钟信号**、**指令寄存器的指令**、**标志(CPU当前所处的状态)和来自系统总线(控制总线)的控制信号**；控制单元的输出信号有**CPU内的控制信号**和**送至系统总线(控制总线)的信号。**
  现代计算机的CPU都集成在一个硅片内，在芯片内采用**内部总线**的方式可节省芯片

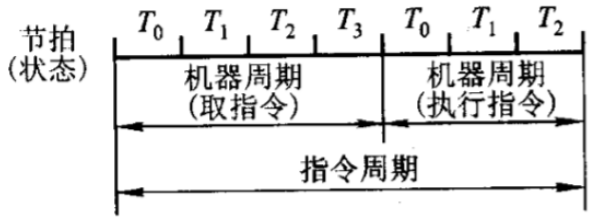
内寄存器之间的连线，使芯片内各个部件布局更合理。



- **机器周期**可看作是所有指令执行过程中的一个基准时间。
  时钟周期是控制计算机操作的最小时间单位，每个节拍的宽度证号对应一个时钟周期。
  机器周期、节拍(状态)组成了多级时序系统。下图反映了指令周期、机器周期、节拍(状态)的关系。一个指令周期包含若干个机器周期，一个机器周期包含若干个时钟周期(节拍)，每个指令周期内的机器周期可以不等，每个机器周期内的节拍数也可能不等。



(a) 定长的机器周期



(b) 不定长的机器周期

- 控制单元控制一条指令执行的过程，实质上是一次执行一个确定的**微操作序列**的过程。如何形成控制不同微操作序列所采用的时序控制方式称为**CU的控制方式。**常见的控制方式有**同步控制**、**异步控制**、**联合控制**和**人工控制。**

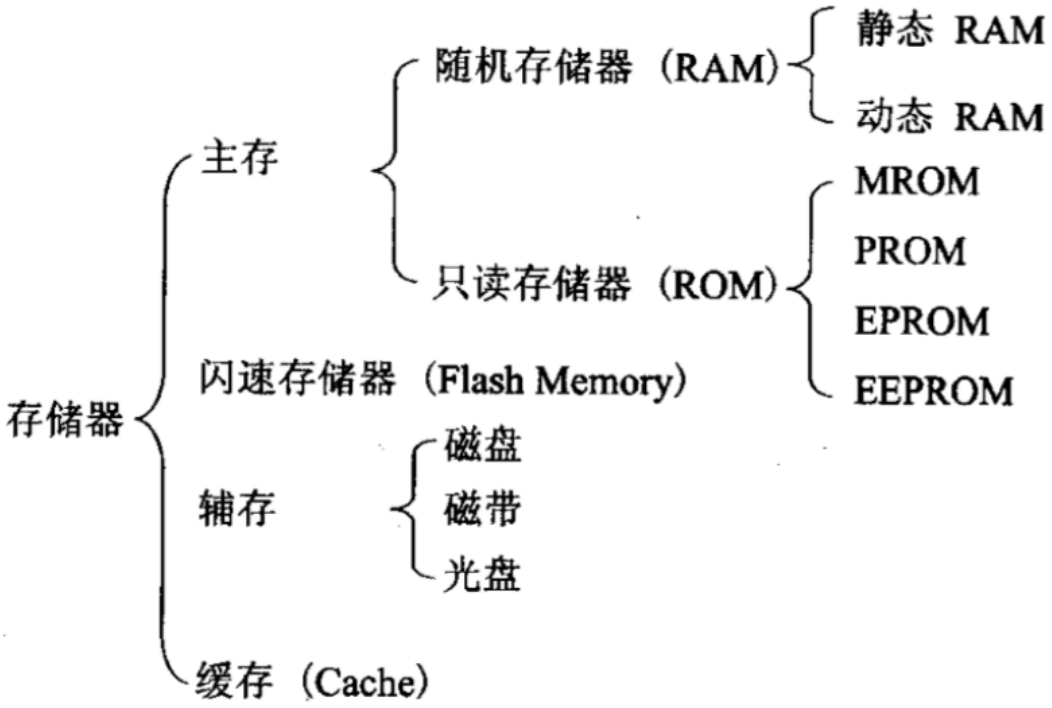
- 控制单元有两种设计方法，分别是**组合逻辑设计**和**微程序设计**。不同的设计方法，使得控制单元内部的结构不同，但最终输出的控制信号是一样的。
  组合逻辑设计的思想是：根据指令微操作的节拍安排，列出微操作命令的操作时间表，然后写出每一个微操作命令(控制信号)的逻辑表达式，最后根据逻辑表达式画出相应的组合逻辑电路。这种设计模式的缺点是随着微操作命令增多，线路更复杂，测试更困难。
  微程序设计的出现克服了组合逻辑设计线路复杂的缺点。其设计思想是：把一条机器指令编写成一个微程序，每个微程序包含若干条微指令，每条微指令对应一个或多个微操作命令。

1.2.6 存储器的结构与功能

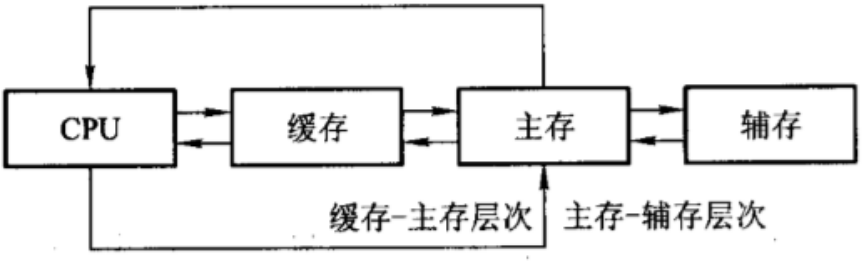
- 存储器是计算机系统中的记忆设备，用来存放程序和数据。这部分重点介绍主存储器的分类、工作原理、组成方式以及与其他部件(如CPU)的联系。

- 存储器的种类繁多，从不同的角度对存储器可作不同的分类。本书介绍的分类方式有按存储介质分类、按存储方式分类和按在计算机的作用分类。
  按**存储介质**分类，存储器可分为半导体存储器、磁表面存储器、磁芯存储器和光盘存储器。
  按**存取方式分**类，存储器可分为随机存储器、只读存储器、顺序存取存储器和直接存取存储器。
  按**在计算机中的作用**分类，存储器只要分为主存储器、辅助存储器、和缓冲存储器。

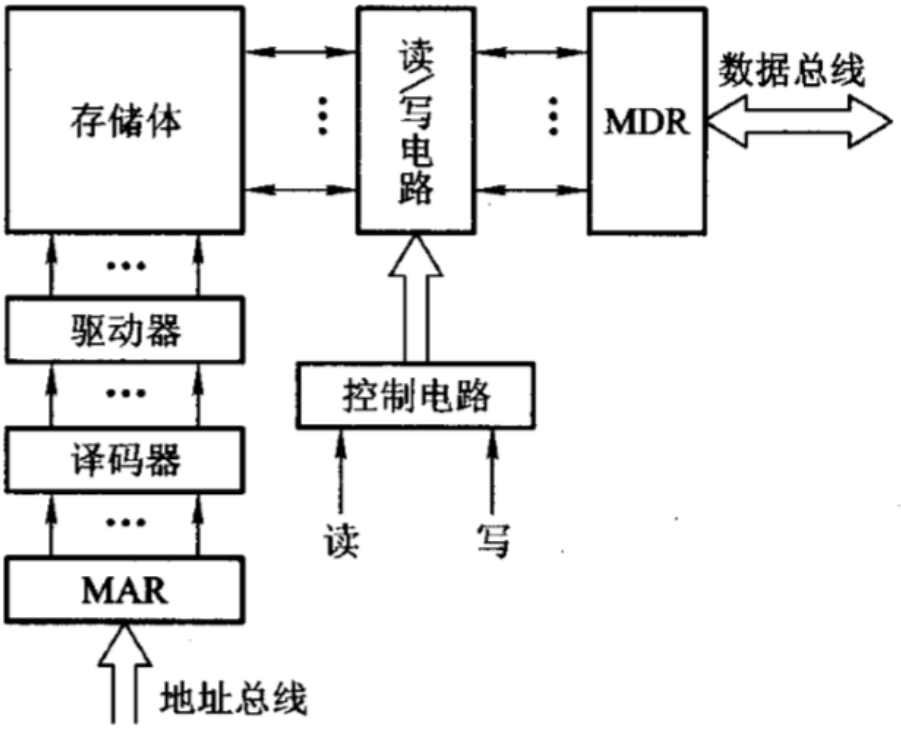
综上所述，存储器分类如下图所示。



- 存储器的**层次结构**主要体现在**Cache-主存**和**主存-辅存**这两个层次结构上。
  缓存-主存层次主要解决CPU和主存速度不匹配的问题。
  主存-辅存层次主要解决存储系统的容量问题。



- 如下图，**主存储器**包括存储体和各种逻辑部件及控制电路等。而存储体是有多个半导体存储芯片组成的，半导体存储芯片的译码驱动方式有**线选法**和**重合法**。



- **随机存取存储器**是构成主存的重要存储器，按其存储信息原理的不同，可分为**静态RAM**和**动态RAM**。
  静态RAM是用触发器工作原理来存储信息的，属于易失性半导体存储器。
  动态RAM是靠电容存储电荷的原理寄存信息。电容上的电荷一般只能维持有限时间，因此即使电源不掉电，其存储的信息也会自动消失。因此，动态RAM会对其所有存储单元回复一次原状态，这个过程将成为**刷新**。刷新通常的三种方式为**集中刷新**、**分散刷新**、**异步刷新**。
  目前，动态RAM的应用比静态RAM更广泛。其原因有：动态RAM的集成度远高于静态RAM、动态RAM芯片引脚和封装尺寸更少、动态RAM功耗更小，价格更低。因此动态RAM广泛用于计算机主存。
  另一方面，静态RAM速度快于动态RAM且不需要刷新。因此，静态RAM通常用于缓冲存储器。

- **只读存储器**是构成计算机主存的另一重要部分。其类型有无法改变原始状态的只读存储器**掩模ROM**、一次性编程的只读存储器**PROM**、可擦除可编程的只读存储器**EPROM**、可局部擦写和全部擦写的**EEPROM**等。

- 前面提到的主存储器的存储体，还有多篇存储芯片组成的。若干存储芯片连载一起，称为**存储容量的扩展**，扩展方式有**位扩展**、**字扩展**和**字、位同时扩展**。在扩展过程中，存储芯片与CPU相连。其中片与片之间的**地址线**、**数据线**和**控制线**的连接是存储容量扩展的重点和难点。

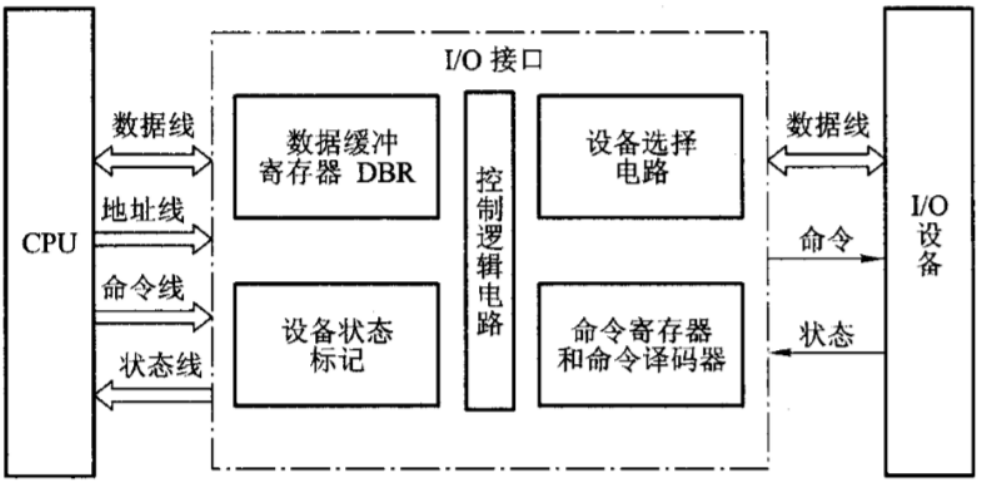
- 书本介绍了多种**提高访存速度**的措施，分别是采用高速器件、采用层次结构以及调整主存结构。而提到的主存结构分别有**单体多字系统**、**多体并行系统**。
  另外，采用高性能存储芯片也是提高访存速度的措施之一。书中介绍了三种芯片，分别是**SDRAM**、**RDRAM**以及**带Cache的DRAM**。

- Cache-主存是计算机存储器的另一个重要的层次结构。Cache的出现使CPU可以不直接诶访问主存，而与高速Cache交换信息，解决了主存与CPU速度不匹配的问题。
  Cache和主存地址的映射方式分为**直接相连**、**全相联映射**以及**组相联映射**。
  当主存块需要调入Cache并且它的可用空间位置又被占满时，需要替换掉Cache的数据，这就产生了相应的替换策略，比如**先进先出算法**、**近期最少使用算法**和**随机法**。

1.2.7 I/O设备的结构与功能

- 输入和输出设备在后续的发展中被归类为**I/O设备**。输入输出系统设计的内容及其繁杂，既包括具体的各类I/O设备，有包括各种不同的I/O设备如何与主机交换信息。
  本章重点分析I/O设备与主机交换信息的三种控制方式，及其相应的接口功能和组成。

- I/O设备和CPU连接，靠的是**I/O接口**。I/O接口通常是指主机与I/O设备之间设置的一个硬件电路及其相应的软件控制。
  I/O接口的结构是根据不同功能和控制方式决定的。I/O接口的功能有**选址**、**传送命令**、**传送数据**、**反应I/O设备工作状态**。
  下图为I/O接口的基本组成。



- 在I/O设备在和主机传送信息时，有多种**控制方式**，本书只讲三种，分别是**程序查询方式**、**程序中断方式**和**DMA方式**。

程序查询方式是由CPU通过不断查询I/O设备是否已做好准备，从而控制I/O设备与主机交换信息。
程序中断方式指的是CPU在启动I/O设备后，不再查询设备是否已准备就绪，CPU选择继续执行自身程序。只有当I/O设备准备就绪并向CPU发出中断请求后才予以响应。
DMA方式指的是I/O设备直接与主存交换信息而不占用CPU。对前两种方法来说，I/O的信息都必须经过CPU才能到达主存。对比前两种方法，DMA方法提高了CPU的资源利用率。

1.2.8 总线的结构与功能

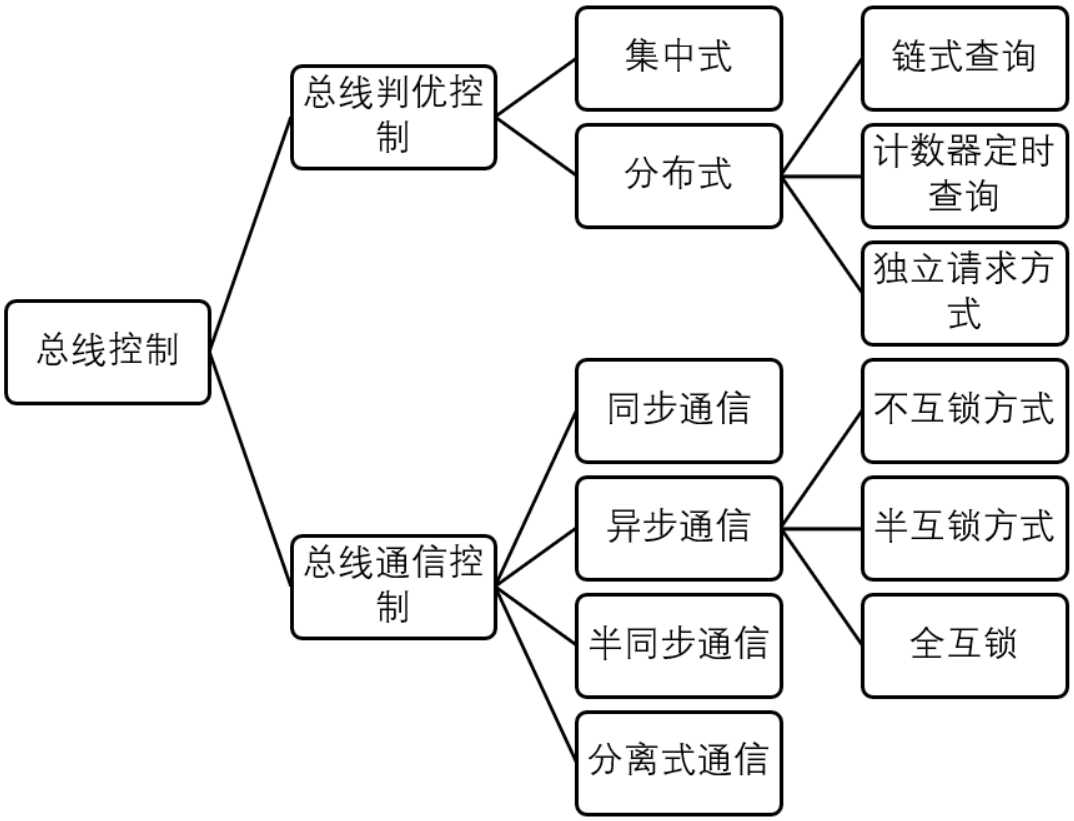
- **总线**是连接上述这么多个部件的信息传输线，是各部件共享的传输介质。总线实际上是又许多传输线或通路组成，二进制代码就是通过总线传输的。

- 总线可分为三类，分别是:
  片内总线是指芯片内部的总线。
  系统总线是指CPU、主存、I/O设备各大部件之间的信息传输线。系统总线根据系统总线传输的信息分为：**数据总线**、**地址总线**和**控制总线**。
  通信总线是连接不同计算机系统或计算机与其他系统的总线。

- 总线结构通常可分为**单总线结构**和**多总线结构**。

- 总线上连接这多个部件，部件之间如何协调工作需要由总线控制器统一管理。管理的内容主要包括**判优控制**和**通信控制**。
  总线判优控制是指当多个设备同时要是用总线时，总线控制器要按一定的优先等级顺序确定哪个主设备能使用总线。
  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。

下图是总线控制的分类图



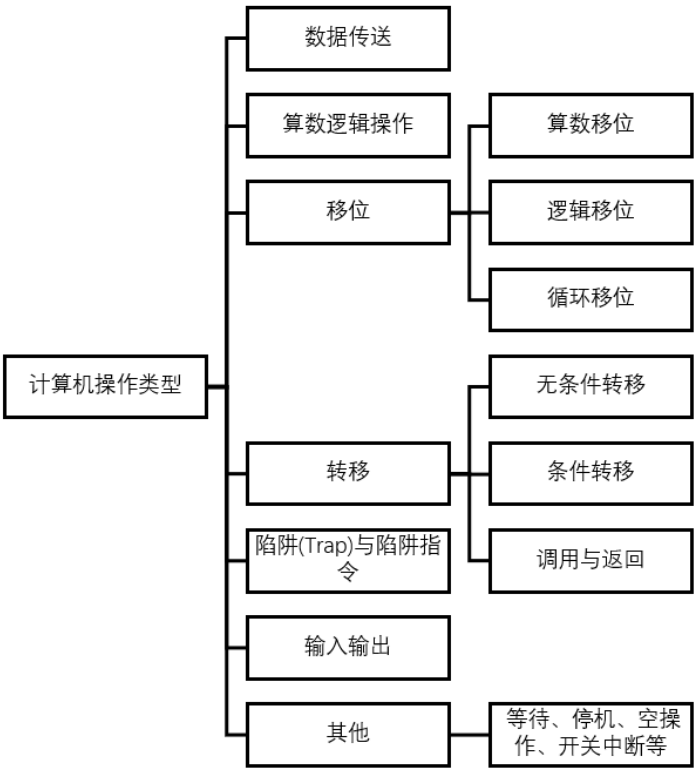
1.2.9 计算机的指令系统

- 计算机是连续执行每一条机器语句而实现全自动工作的。而这每一条机器语言的语句称为指令，全部机器指令的集合称为机器的**指令系统**。因此机器指令系统集中反映了机器的功能。

- 指令是由**操作码**和**地址码**两部分组成。操作码的长度可以是固定的，也可以是变化的。由于操作码长度不固定会增加指令译码和分析的难度，因此通常采用**扩展操作码技术**，使操作码长度对地址数的减少而增加。

- 指令字长取决于**操作码的长度**、**操作数地址的长度**和**操作数地址的个数**。不同机器的指令字长是不同的。
  机器中操作数的类型有**地址**、**数字**、**字符**和**逻辑数据**等。

不同的机器，操作类型也是不同的，但几乎所有机器都有如所展示的下图操作。



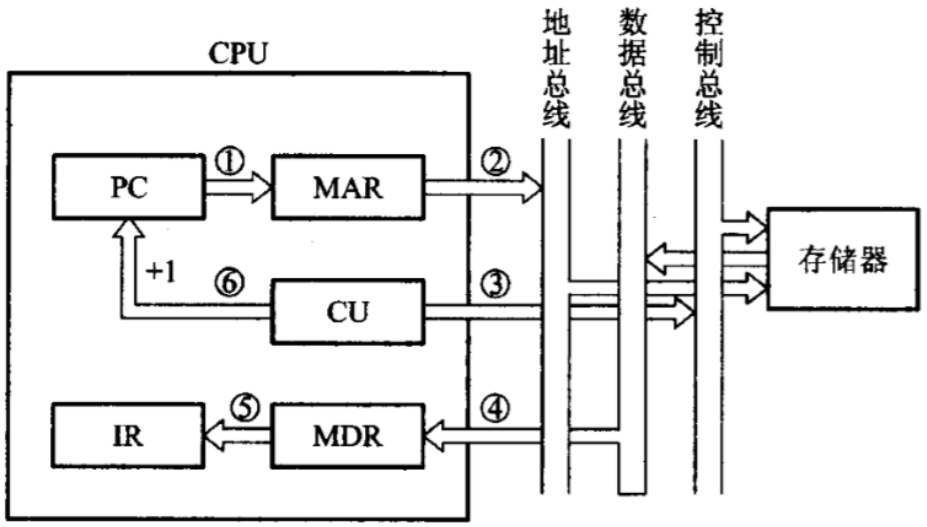
- 寻址方式是指确定本条指令的数据地址以及下一条将要执行的指令地址的方法，主要分为指令寻址和数据寻址两大类。
  指令寻址指的是寻找下一条指令的地址，它分为**顺序寻址**和**跳跃寻址**两种。
  指令寻址的方式种类较多，因此在指令字中必须设一个字段来指明是哪一种寻址方式。书上介绍的寻址方式有立即寻址、直接寻址、隐含寻址、间接寻址、寄存器寻址、寄存器间接寻址、基址寻址、变址寻址、相对寻址和堆栈寻址。

- 指令格式集中体现了指令系统的功能。在确定指令格式时，应从以下五个方面综合考虑：操作类型、数据类型、指令格式、寻址方式和寄存器个数。

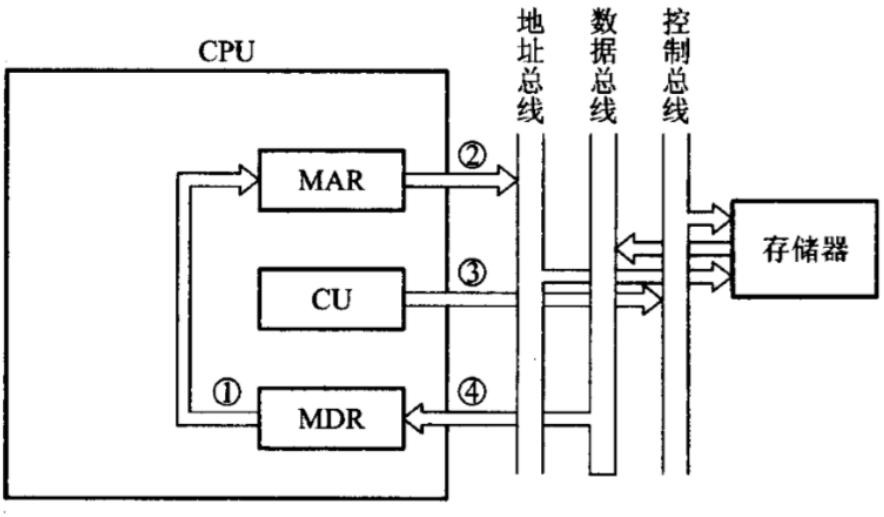
- RISC即精简指令系统计算机，与其对应的是CISC，即复杂指令系统计算机。与CISC相比，RISC机的主要优点有：充分利用VLSI芯片的面积、提高计算机运算速度、便于设计、有效支持高级程序语等。
  RISC的主要特点有：
  复杂指令的功能有频率高的简单指令的组合来实现。
  指令长度固定，指令格式种类少，寻址方式种类少。
  只有取数、存数指令访问存储器，其余指令操作都在寄存器内完成。
  CPU中有多个通用寄存器。
  采用流水线技术，大部分指令在一个时钟周期内完成。采用超标量和超流水线技术，可是每条指令的平均执行时间小于一个时钟周期。
  控制器采用组合逻辑控制，不用微程序控制
  采用优化的编译程序。

- CPU每取出并执行一条指令所需的全部时间称为**指令周期**，也即CPU完成一条指令的时间。
  由于各种指令操作功能不同，因此各种指令的指令周期是不相同的。一个完整的指令周期应该包括**取指**、**间址**、**执行**和**中断**这四个子周期。

- 取指周期操作如下：
  现行指令地址送至存储器地址寄存器，记为PC→MAR。
  向主存发送读命令，启动主存作读操作，记作1→R。
  将MAR(通过地址总线)所指的主存单元中的内容(指令)经数据总线读到MDR内，记作M(MDR)→MDR.
  将MDR的内容送至IR，记作MDR→IR。
  取回来的指令的操作码送至CU，记作OP(IR)→CU。
  形成下一条指令的地址，记作(PC)+1→PC。

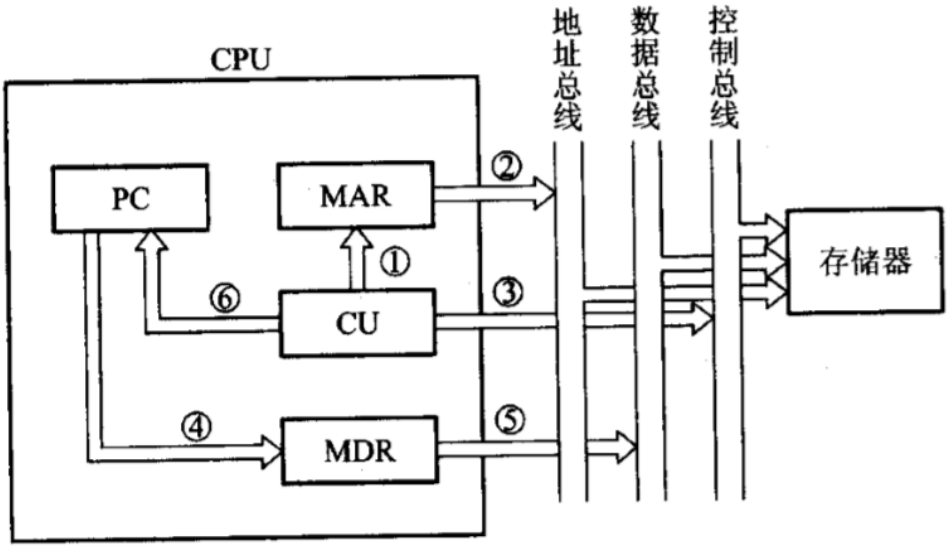


- 间址周期完成取操作数有效地址的任务，操作如下：
  将指令的地址码部分(形式地址)送至存储器地址寄存器，记作Ad(IR)→MAR。
  向主存发送读命令，启动主存作读操作，记作1→R。
  将MAR(通过地址总线)所指的主存单元中的内容(有效地址)经数据总线读至MDR内，记作M(MAR)→MDR。
  将有效地址送至指令寄存器的地址字段，记作MDR→Ad(IR)。这个操作在有些机器中可省略。

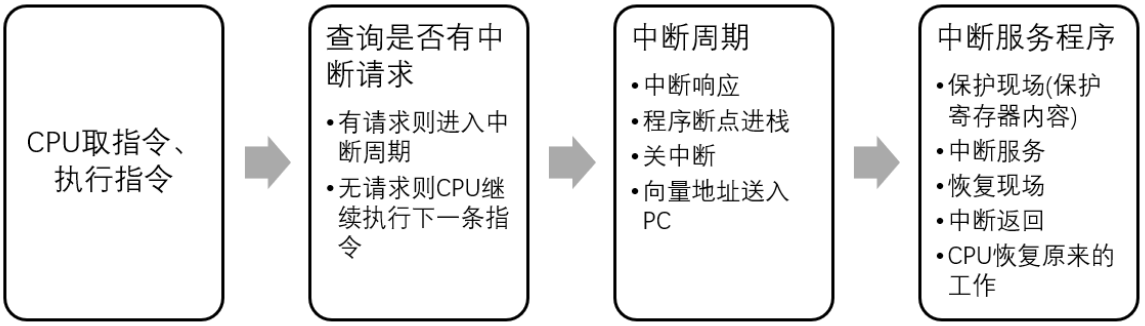


- 执行周期
  不同指令的执行周期的微操作是不同的，非访存指令、访存指令和转移指令的指令周期各自不同。这里就不再赘述了。

- 中断周期
  在指令执行周期结束的时刻，CPU要查询是否有中断请求，如果有则进入中断周期。要注意，中断周期是为中断服务程序做准备的。中断周期的操作如下：
  将特定地址"0"送至存储器地址寄存器，记作0→MAR。
  向主存发写命令，启动存储器作写操作，记作1→W。
  将PC的内容(程序断点)送至MDR，记作PC→MDR。
  将MDR的内容(程序断点)通过数据总线写入到MAR(通过地址总线)所指示的主存单元(0地址单元)，记作MDR→M(MAR)。
  将向量地址形成部件的输出送至PC，记作向量地址→PC，为下一条指令的取指周期作准备。
  关中断，将允许中断触发器清零，记作0→EINT(该操作课直接由硬件线路完成)。
  因此整个中断周期要做的事情有:**保护断点**、**寻找中断服务程序的入口地址**、**软件关中断**。



- 另外，中断周期后计算机进入中断服务程序。下图展示了中断服务程序的流程，在中断服务程序中计算机的操作依次为：**保护现场(保护寄存器内容)**、**设备服务**、**恢复现场**、**开中断**以及**中断返回**。



- 整个中断过程中需要注意的有以下几点：
  各中断源是通过**中断请求触发器**向CPU发送中断请求的。这些触发器课集中设在CPU内，也可分散设置在接口内。
  CPU内的中断系统通过**中断判优逻辑**确定优先响应哪一个中断源的请求。中断判优可以用**硬件实现**，如通过链式排队器或CPU内排队器实现；也可以用**软件排队**实现，即编写查询程序实现。
  中断服务程序入口地址，指的是程序的首条指令的地址。查找中断服务程序入口地址有两种方法，分别是**硬件向量法**和**软件查询法**，实际上两种方法是和排队方式相对应的。
  中断过程中出现了新的中断请求，这种情况称为**多重中断**。其实现条件为：提前设置**开中断指令**和优先级别高的中断源有权中断优先级别低的中断源。

为保证优先级别高的中断源课打断优先级别低的中断源，可以采用**屏蔽技术**，屏蔽技术可改变中断源的优先等级。屏蔽技术由**屏蔽触发器**和**屏蔽字**组成。

- 由上面分析我们可知，完成一条指令实际上可以分多个阶段。因此在这里我们引入**指令流水**的概念。指令流水是指，把一条指令的操作分成多个细小的步骤，每个步骤由专门的电路完成。指令相互重叠执行，大大增加了CPU的执行效率。关于指令流水我们需要关注以下几点。
  流水线过程中会出现三种相关，使流水线不断流实现起来很困难，这三种相关分别是**结构相关**、**数据相关**和**控制相关**。
  流水线的性能通常用**吞吐率**、**加速比**和**效率**三项指标来衡量。
  为了在一个时钟周期内产生更多条指令的结果，流水线中的多发技术随之发明。流水线中的多发技术分别有：**超标量技术**、**超流水线技术**和**超长指令字技术**。
  将指令的整个执行过程用流水线进行分段处理的流水线结构称为**指令流水线结构**，应用于部件级的流水技术称为**运算流水线结构**。

- 无论控制单元采用的设计模式是**组合逻辑设计**，还是**微程序设计**，所有的微操作的控制信号都是由控制单元发出的。

- 我们下面以取指周期、间址周期、中断周期为例，介绍在组合逻辑设计模式中，微操作的节拍安排。每个节拍内都可以进行一个或多个微操作，每个微操作都有对应的来自CU的控制信号。

- 取指周期的节拍安排：
  T_0：PC→MAR，1→R。
  T_1：M(MAR)→MDR，(PC)+1→PC。

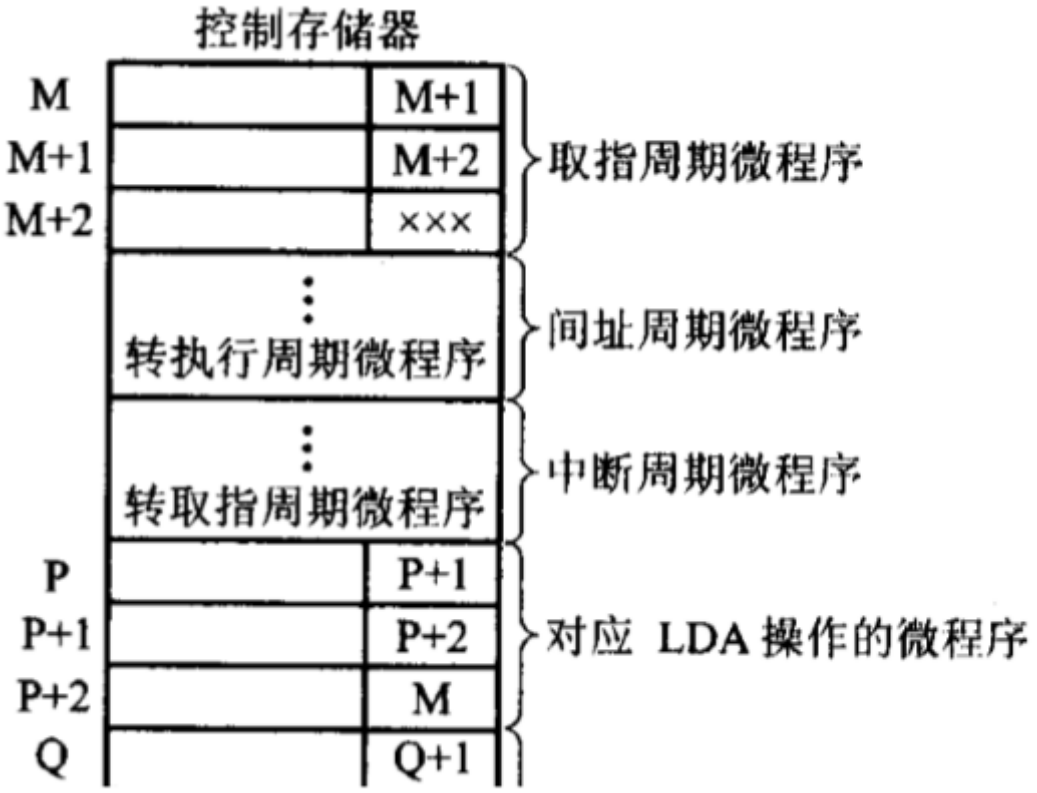
- 间址周期的节拍安排：
  T_0：Ad(IR)→MAR，1→R。
  T_1：M(MAR)→MDR。
  T_2：MDR→Ad(IR)。

- 不同的指令在执行周期进行的操作也不一样，因此此处就不详细介绍各条指令在执行周期所执行的微操作了。

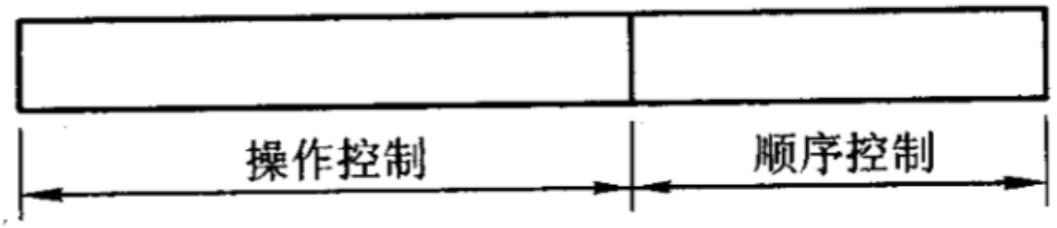
- 中断周期的节拍安排：
  T_0：0→MAR，1→W。
  T_1：PC→MDR。
  T_2：MDR→M(MAR)，向量地址→PC。

- **微程序设计**可以看成组合逻辑设计的升级版。其实现原理是把机器指令编写成一个微程序，一个微程序包含了若干条微指令，每条微指令对应一个或多个微操作命令。因此，多条微指令的组合即可实现多种复杂的功能，便于调试、修改和增删机器指令。

微程序控制单元的核心部件是一个**控制存储器**。如图，每一条机器指令都与一个操作性质命名的微程序对应。



- 控制存储器里面存放的就是**微指令**。如下图，微指令共分两个字段，左边的为操作**控制字段**，该字段发出各种操作信号；另一个为**顺序控制**字段，负责指出下调微指令的地址，以控制微指令序列的执行顺序。



- 微指令的编码方式又称微指令的**控制方式**，它是指如何对微指令控制字段进行编码，以形成控制信号。编码方式主要有：**直接编码(直接控制)方式**、**字段直接编码方式**、**字段间接编码方式**、**混合编码**以及其他多种方式。
- 微指令**序列地址**的形成大致有两种方式，分别是**直接由微指令的下地址字段指出**和**根据机器指令的操作码形成**。实际上除了以上两种常用的，微指令序列地址形成方

式还有**增量计数法**、**分支转移**、**通过测试网络形成**以及**由硬件产生微程序入口地址**。

- 微指令的格式与微指令的编码方式有关，通常分为**水平型微指令**和**垂直型微指令**两种。

- 完成一条微程序分为两个阶段，分别是**取微指令**和**执行微指令**，由这两个阶段产生了**串行微程序控制**和**并行微程序控制**两种运行方式。

- 微程序设计控制单元的主要任务是编写对应各条机器指令的微程序，主要步骤是先写出**对应机器指令的全部微操作及节拍安排**，然后**确定微指令格式**，最后编写出每条微指令的二进制代码，即**微指令码点**。

1.2.10 计算机的工作步骤

用计算机解决实际问题包括两大步骤。一个是上机前的准备，另一个是上机运行。
上机前准备的工作可以归纳为：建立数学模型、确定计算方法和编制解题程序。
上机运行则是计算机无情地执行指令的过程了。在这里我们来大致了解一下计算机是如何执行指令的。

- 例如我们通过计算机求:$(ax+b)x+c$的结果。在这里我们可以把过程分为一下步骤：
  通过键盘输入未知数x的值。
  将x取至运算器中。
  乘以a，的ax，存入ACC中。
  加b，得$ax+b$，存入ACC中。
  乘以x，得$(ax+b)x$，存入ACC中。
  加c，得$(ax+b)x+c$，存入ACC中。

将上述操作步骤写成计算机对应的机器指令，我们就完成了程序的编写。下一步我们要做的是让计算机执行对应的指令。

指令和数据存于主存单元的地址	指令		注释
	操作码	地址码	
0	000001	0000001000	取数 x 至 ACC
1	000100	0000001001	乘 a 得 ax,存于 ACC 中
2	000011	0000001010	加 b 得 $ax+b$,存于 ACC 中
3	000100	0000001000	乘 x 得 $(ax+b)x$,存于 ACC 中
4	000011	0000001011	加 c 得 ax^2+bx+c,存于 ACC 中
5	000010	0000001100	存数,将 ax^2+bx+c 存于主存单元
6	000101	0000001100	打印
7	000110		停机
8	x		原始数据 x
9	a		原始数据 a
10	b		原始数据 b
11	c		原始数据 c
12			存放结果

- 由图可知，该程序有8条指令，每条指令都要经过取指、分析、执行三个阶段。因此(ax+b)x+c程序的运行过程如下：
  将程序通过输入设备送至计算机
  程序首地址→PC
  启动程序运行
  取第一条指令：PC→MAR→M→MDR→IR ,(PC)+1→PC
  分析指令：OP(IR)→CU
  执行指令：Ad(IR)→MAR→M→MDR→ACC
  重复进行“取指令→分析指令→执行指令”的过程，直到该程序最后一条指令执行完。
  打印结果
  停机

1.3 计算机硬件的主要技术指标

1.3.1 机器字长

机器字长是指CPU一次能处理数据的位数，通常与CPU的寄存器位数有关。字长越长，数的表示范围越大。

1.3.2 存储容量

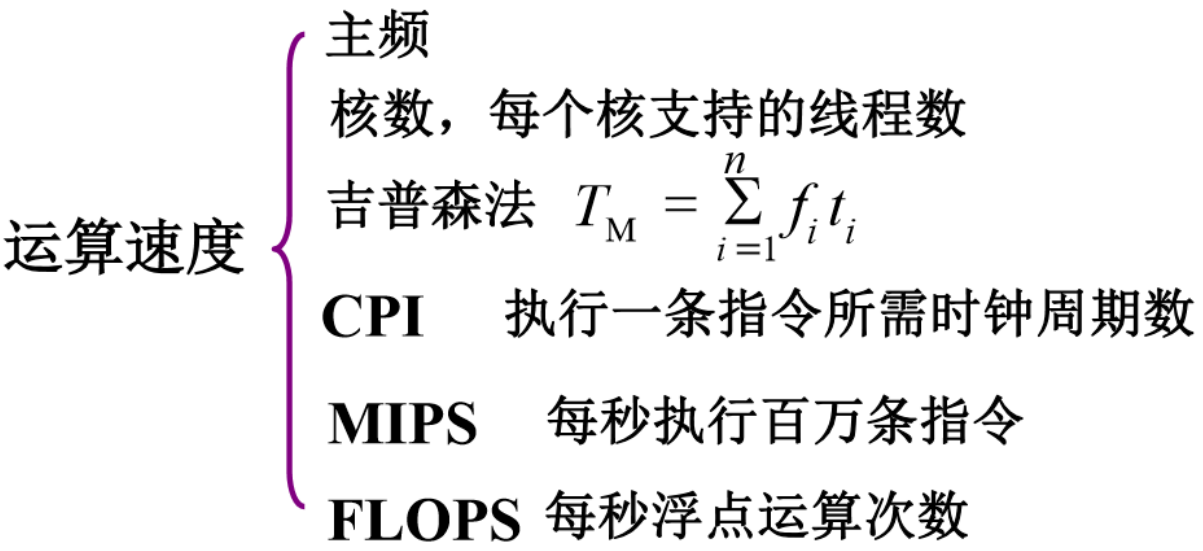
存储器的容量应该包括主存容量和辅存容量。即

> 存储容量 = 存储单元个数 × 存储字长

计算机中，**MAR的位数**反映了存储器的存储单元的个数，**MDR的位数**反应了每个存储单元的存储字长。因此，我们可以通过上面两个参数求得存储容量。

1.3.3 运算速度

计算机的运算速度



第4章 存储器

本章概述

本章重点介绍存储器的分类、工作原理、组成方式以及其他部件（如CPU）的联系。此处还介绍了高速缓冲存储器、磁表面存储器等的基本组成和工作原理。旨在使读者真正建立起如何用的存储器组成具有层次结构的存储系统的概念。

4.1 概述

4.1.1 存储器分类

1、按存储介质分类

（1）半导体存储器

由半导体器件组成的存储器。

- 优点：体积小、功耗低、存取时间短
- 缺点：当电源消失时，所存信息也随之丢失，是一种易失性存储器。

按其材料不同又可分为双极型（TTL）半导体存储器（特点：高速）和MOS半导体存储器（特点：高集成度，制造简单，成本低廉，功耗小，广泛应用）。

（2）磁表面存储器

在金属或塑料基体的表面上涂一层磁性材料作为记录介质，工作时磁层随**载磁体**高速运转，用**磁头**在磁层上进行读/写操作。

特点：非易失性。

> 按载磁体形状的不同，可分为磁盘、磁带和磁鼓。

（3）磁芯存储器

磁芯是由**硬磁材料**做成的**环形元件**，在磁芯中穿有驱动线（通电流）和读出线，这样便可以进行读/写操作。

特点：非易失性。体积过大、工艺复杂、功耗太大，目前几乎已不采用。

（4）光盘存储器

是应用**激光**在记录介质（**磁光材料**）上进行读/写的存储器。

特点：非易失性。记录密度高、耐用性好、可靠性高、可互换性强，越来越用于计算机系统。

2、按存取方式分类

（1）随机存储器（RAM）

在程序的执行过程中**可读可写**。存储器的任何一个存储单元的内容都可以随机存取，存取时间与存取单元的物理位置无关。

（2）只读存储器（ROM）

在程序的执行过程中**只读**。通常存放固定不变的程序、常数和汉字字库，甚至用于操作系统的固化。与随机存储器可共同作为主存的一部分，统一构成主存的地址域。

（3）串行访问存储器

对存储单元进行读/写操作时，按其物理位置的先后顺序寻找地址。这种存储器由于信息所在位置不同，使得读/写时间均不相同。

- 顺序存取存储器：不论信息处在哪个位置，读/写时必须从其介质的始端开始按顺序寻找，如磁带。
- 直接存取存储器：在对磁盘读/写时，首先直接指出该存储器中的某个小区域（磁道），然后再顺序寻访，直至找到位置，如磁盘。

3、按在计算机中的作用分类

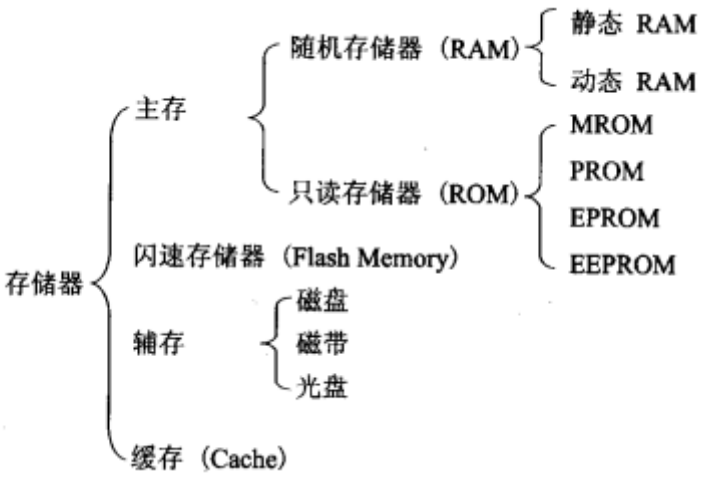
主存储器（简称主存）：可以和CPU直接交换信息。

辅助存储器（简称辅存）：是主存的后援存储器，用来存放当前暂时不用的程序和数据，不能直接和CPU交换信息。

> 二者相比，主存速度快、容量小、每位价格高；辅存速度慢、容量大、每位价格低。

缓冲存储器（简称缓存）：用在两个速度不同的部件之中，起到缓冲作用。

综上所述，存储器分类如下图：



4.1.2 存储器的层次结构

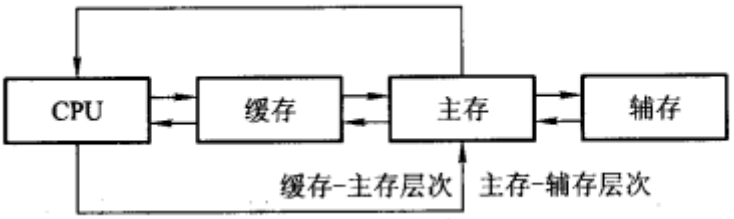
1、存储器三个主要特性的关系

存储器有3个主要性能指标：速度、容量和每位价格（简称位价）。

下图反映了三者的关系。图中由上至下，位价越来越低，速度越来越慢，容量越来越大，CPU访问的频度越来越少。

2、缓存 — 主存层次和主存 — 辅存层次

实际上，存储系统层次结构主要体现在缓存 — 主存和主存 — 辅存这两个存储层次上。如下图所示。

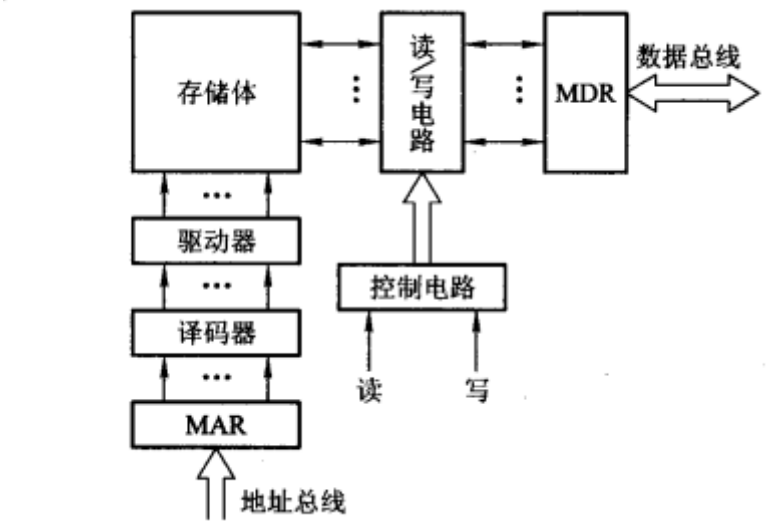


> 缓存 — 主存层次主要解决CPU和主存速度不匹配的问题。
> 主存 — 辅存层次主要解决存储系统的容量问题。

4.2 主存储器

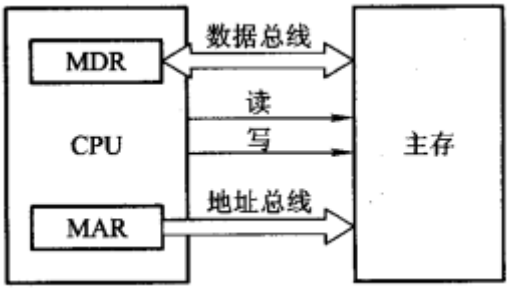
4.2.1 概述

1、主存的基本组成



2、主存与CPU的联系

如下图所示，图中的驱动器、译码器和读写电路均制作在存储芯片中，而MAR和MDR制作在CPU芯片内。存储芯片和CPU芯片可通过总线连接。



3、主存中存储单元地址的分配

主存各存储单元的空间位置是由单元地址号来表示的，而地址总线是用来指出存储单元地址号，根据该地址可读出或写入一个存储字。

不同的机器存储字长不同。

8位二进制数表示一个字节，存储字长都取8的倍数。

4、主存的技术指标

（1）存储容量

指主存能存放二进制代码的总位数，即 **存储容量=存储单元个数×存储字长**

它的容量也可用字节总数来表示，即 **存储容量=存储单元个数×存储字长/8**

（2）存储速度

- 存储时间（又称存储器的访问时间）：指启动一次存储器操作（读或写）到完成该操作所需的全部时间。

> 分为读出时间（从存储器接收到有效地址开始，到产生有效输出所需的全部时间）和写入时间（从存储器接收到有效地址开始，到数据写入被选中单元为止所需的全部时间）。

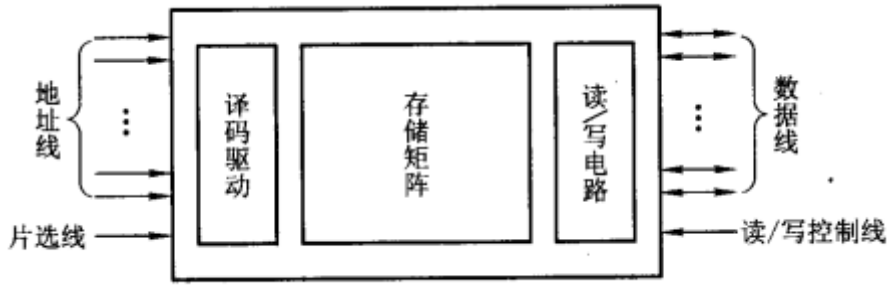
- 存储周期：指存储器进行连续两次独立的存储器操作（如连续两次读操作）所需的最小间隔时间，通常存取周期大于存取时间。

（3）存储器带宽
表示单位时间内存储器存取的信息量，单位可用**字/秒**或**字节/秒**或**位/秒**表示。
提高存储器带宽的措施：

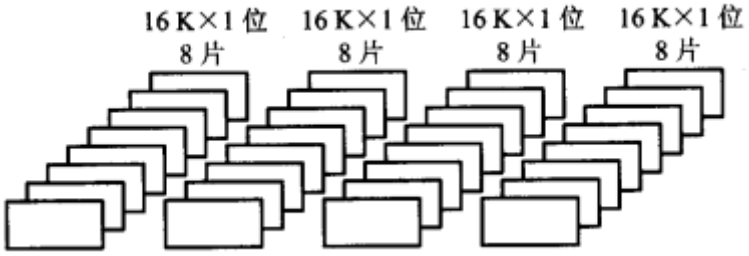
- 缩短存取周期
- 增加存储字长，使每个存取周期可读/写更多的二进制位数。
- 增加存储体

4.2.2 半导体存储芯片的简介

1、半导体存储芯片的基本结构
译码驱动能把地址总线送来的地址信号翻译成对应存储单元的选择信号，该信号在读/写电路的配合下完成对被选中单元的读/写操作。
读/写电路包括读出放大器和写入电路，用来完成读/写操作。
存储芯片通过地址总线、数据总线和控制总线与外部连接。
下图为存储芯片的基本结构



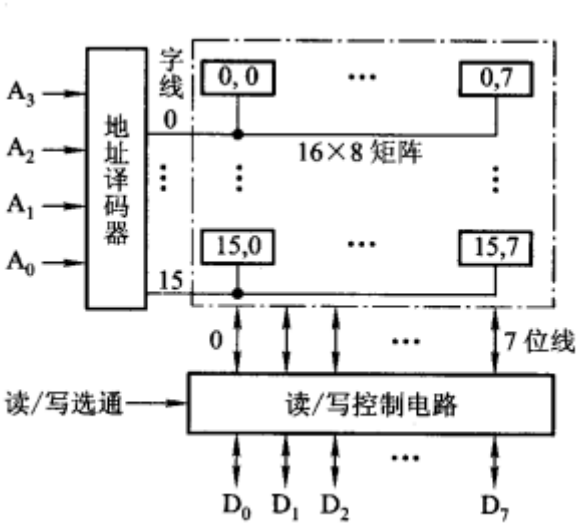
地址线是单向输入的，其位数与芯片容量有关。
数据线是双向的，其位数与芯片可读出或写入的数据位数有关。数据线的位数与芯片容量有关。
地址线和数据线的位数共同反应存储芯片的容量。
控制线主要有读/写控制线和片选线两种。
存储芯片片选线的作用：用16K×1位的存储芯片组成64K×8位的存储器



2、半导体存储芯片的译码驱动方式

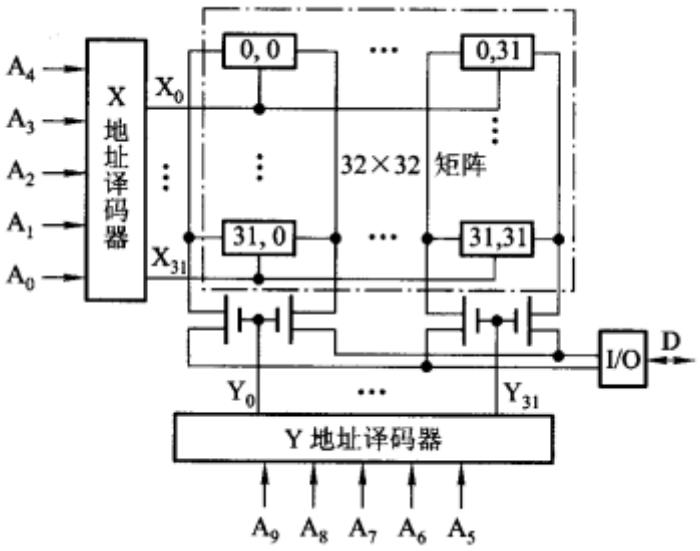
（1）线选法

下图是一个16×1字节线选法存储芯片的结构示意图。特点是用一根字选择线（字线），直接选中一个存储单元的各位。结构简单，只适用于容量不大的存储芯片。



（2）重合法

下图是一个1K×1位重合法结构示意图。

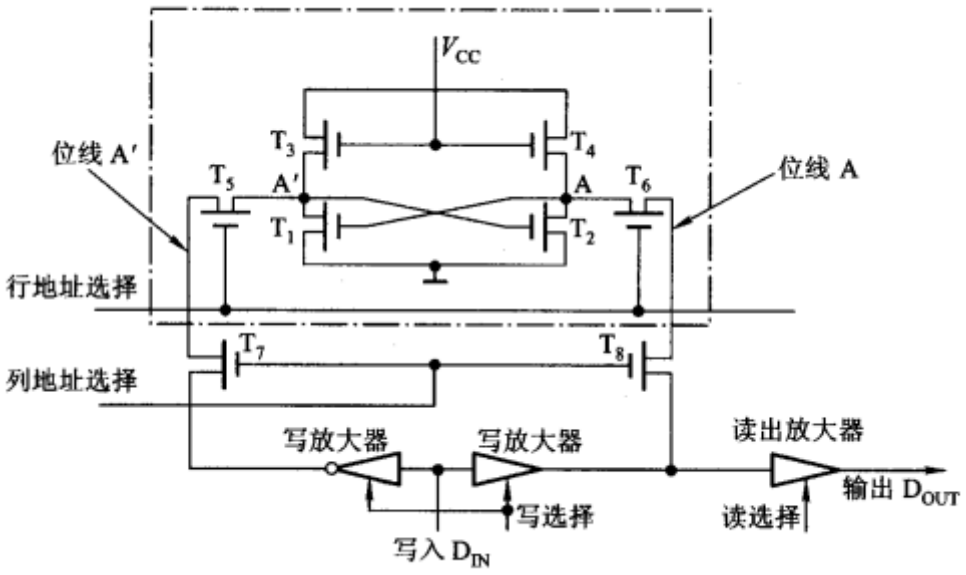


4.2.3 随机存取存储器

随机存取存储器按其存储信息的原理不同，可分为静态RAM和动态RAM两大类。

1、静态RAM（SRAM）

（1）静态RAM基本单元电路



T_1~T_4 触发器

T_5、T_6 行开关

T_7、T_8 列开关，一列共用

A 触发器原端

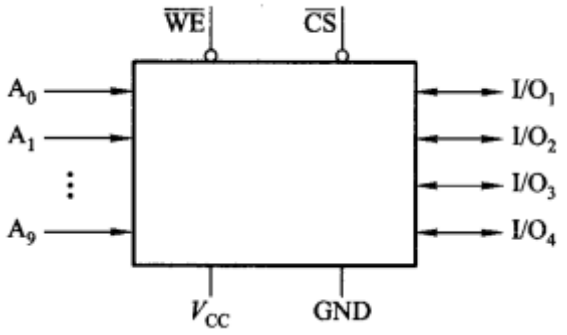
A` 触发器非端

//静态RAM属于易失性半导体存储器

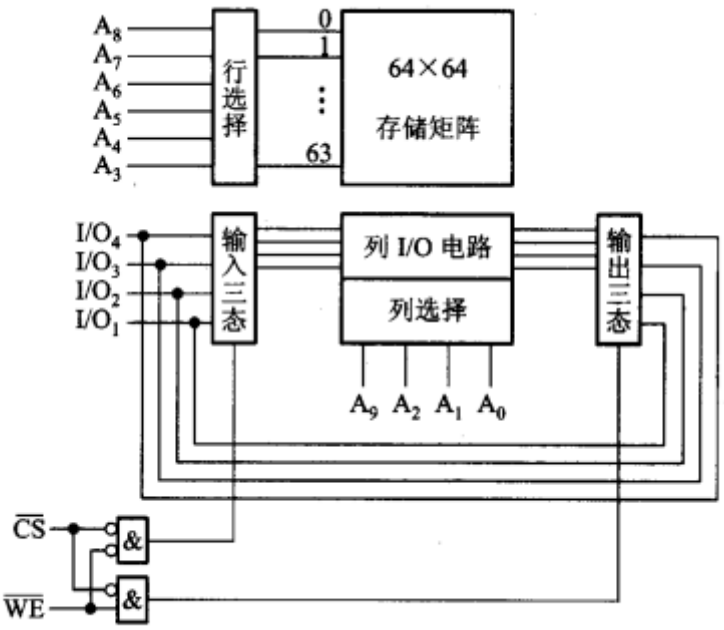
（2）静态RAM芯片举例

Intel2114芯片

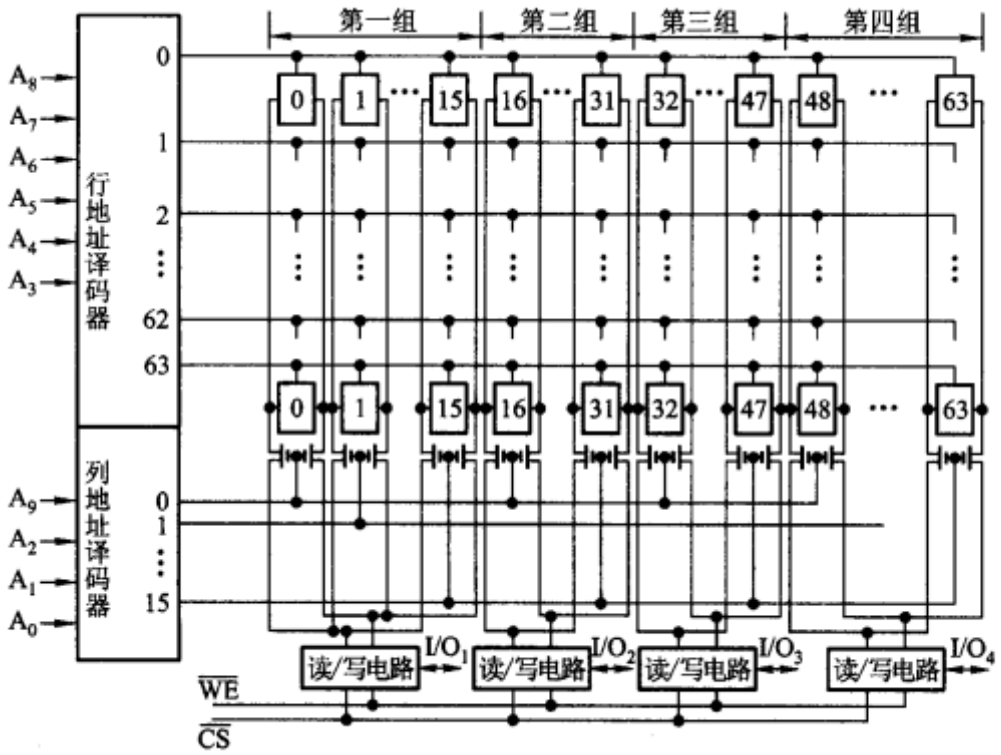
下图为Intel2114外特性示意图



下图为2114芯片结构示意图



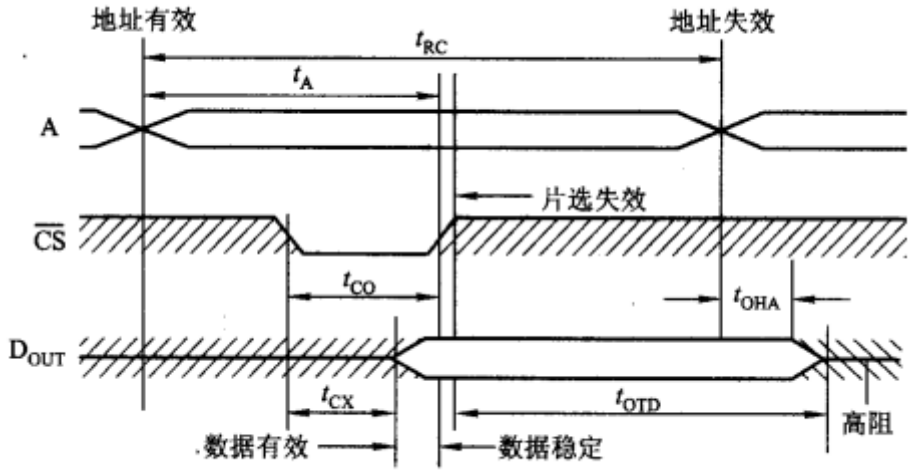
下图为2114RAM芯片内的存储矩阵结构



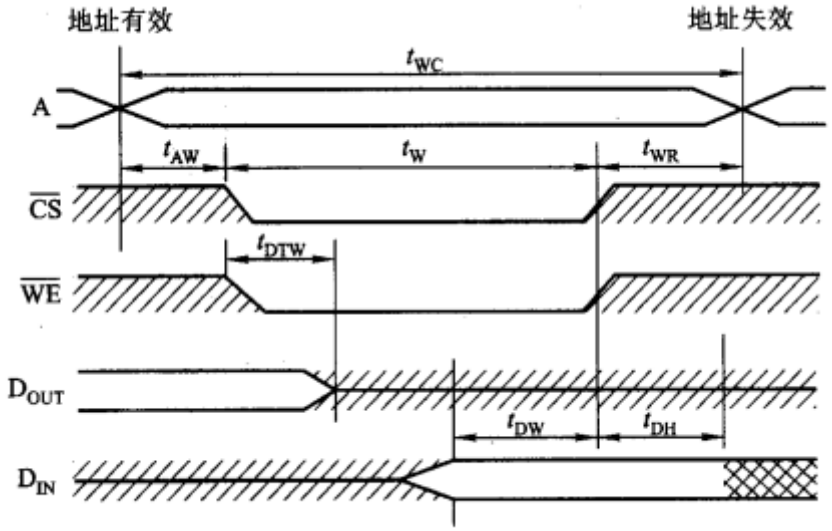
//详细图解见课本76-79页

（3）静态RAM读/写时序

1）读周期时序



2）写周期时序



//详细过程解析见课本78-80页

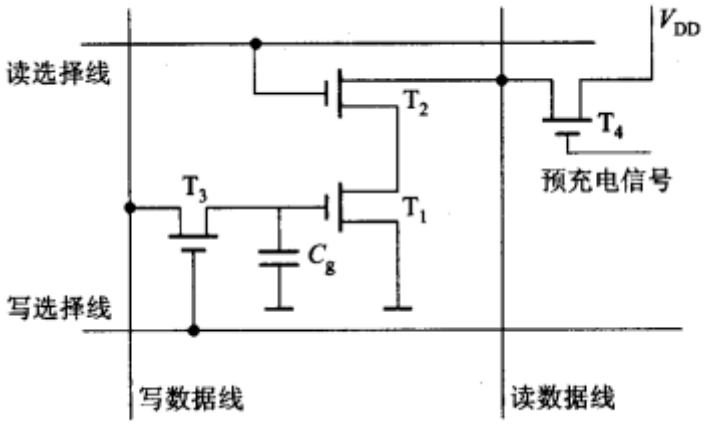
2、动态RAM（DRAM）

（1）动态RAM的基本单元电路

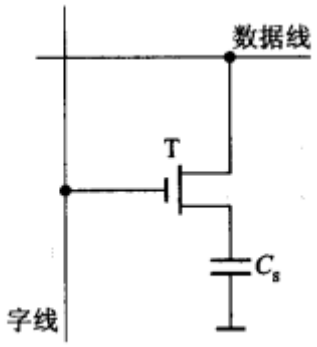
常见的动态RAM基本单元电路靠电容存储电荷的原理来寄存信息。若电容上存有足够多的电荷表示存“1”，电容上无电荷表示存“0”。

再生或刷新：电容上的电荷一般只能维持1~2ms。因此即使电源不掉电，信息也会丢失，为此，必须在2ms内对其所有存储单元恢复一次原状态，这个过程称为再生或刷新。

下图为三管MOS动态RAM基本单元电路。



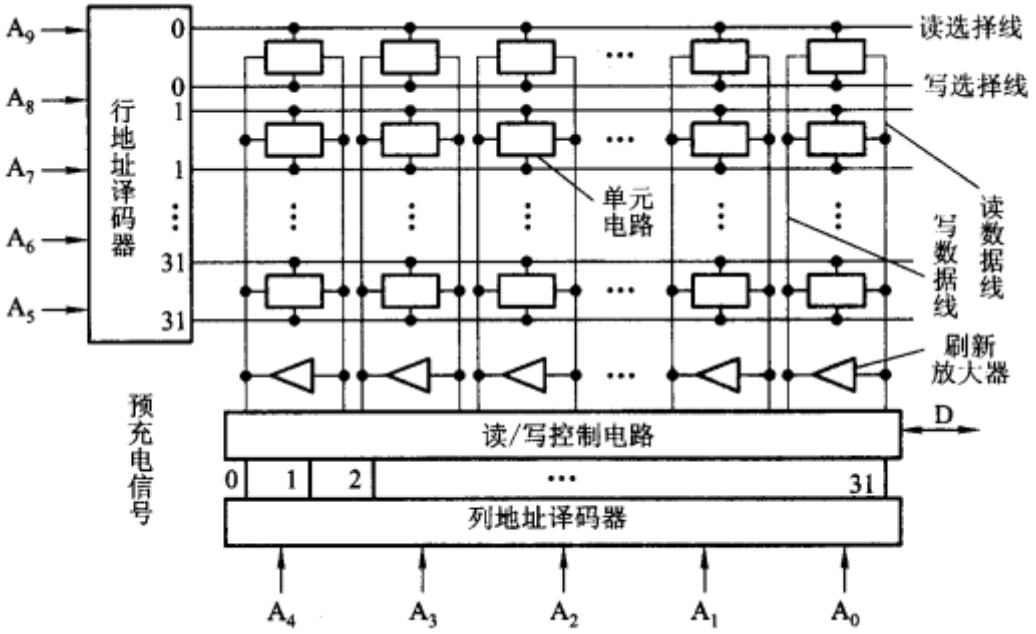
为了提高集成度，将三管电路进一步简化，得到单管MOS动态RAM基本单元电路，如下图。



（2）动态RAM芯片举例

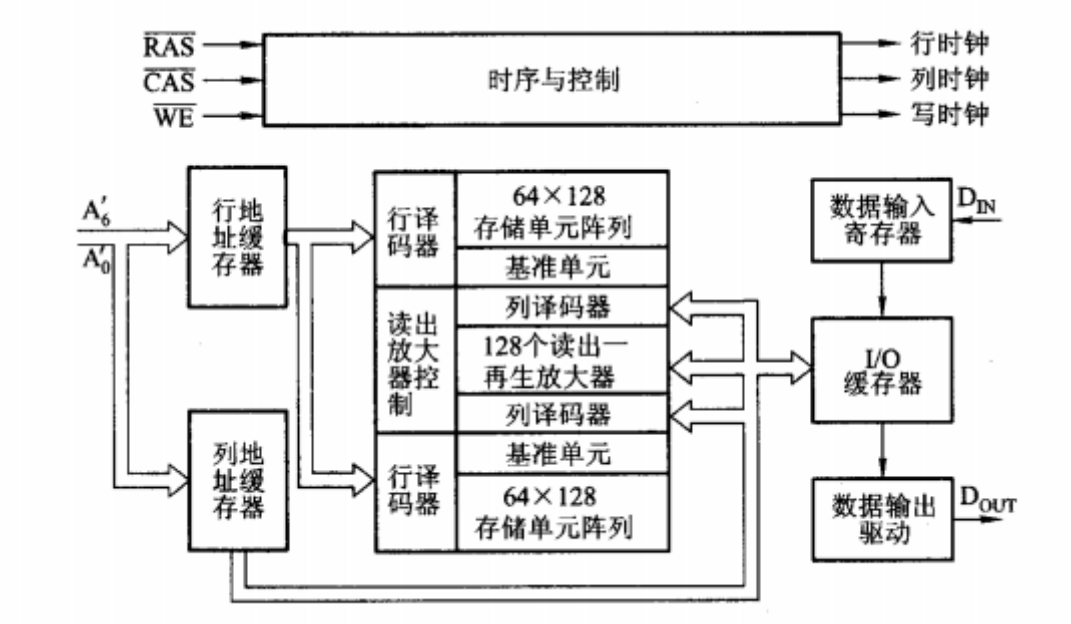
1）三管动态RAM芯片

下图为1K×1位的存储芯片



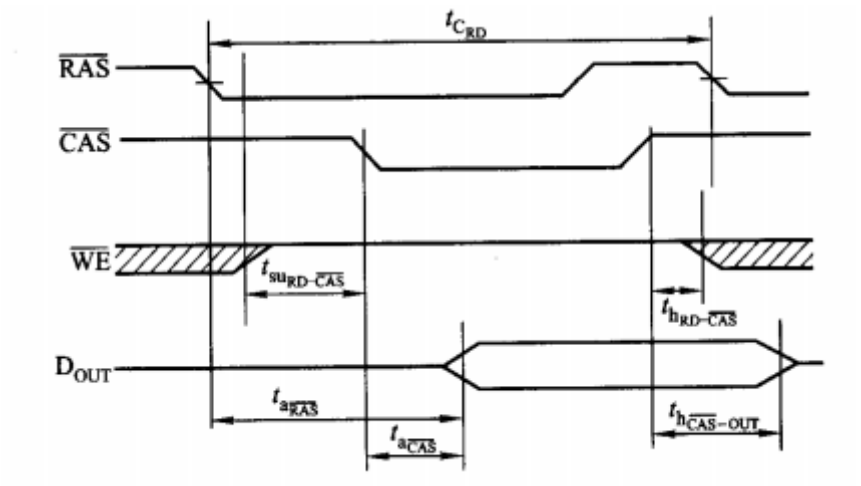
2）单管动态RAM芯片

下图为单管动态RAM芯片结构示意图

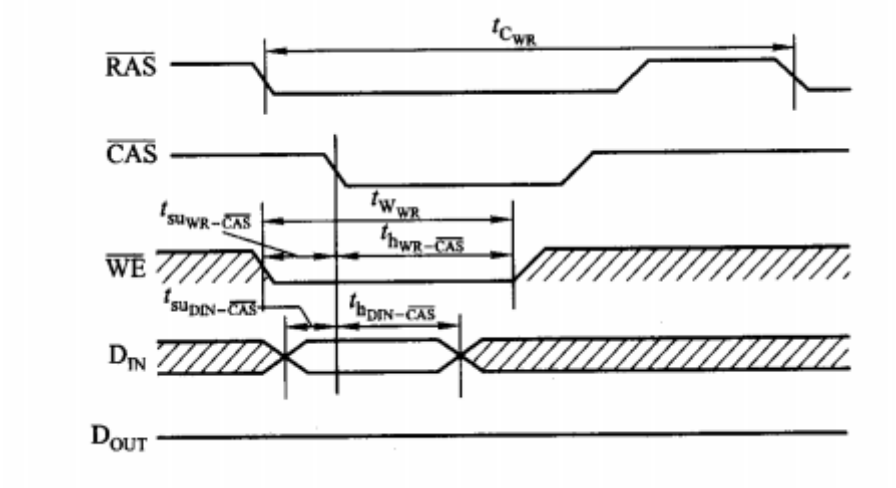


（3）动态RAM时序

1）读时序



2）写时序



//记得根据示意图以及课本解析，看懂静态RAM基本电路的读/写操作过程、Intel2114RAM矩阵（64×64）读/写操作过程、静态RAM的读/写操作过程、三管动态RAM芯片的读/写操作过程、单管动态RAM芯片的读/写操作过程以及静态RAM和动态RAM的读/写时序，如果还没有看懂，不要急着向下哦~

（4）动态RAM的刷新

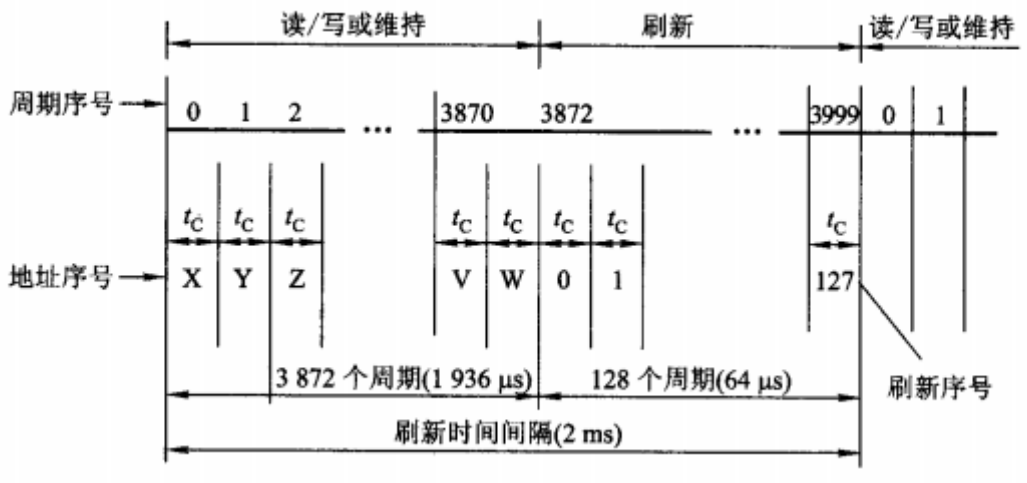
刷新：先将原存信息读出，再由刷新放大器形成原信息并重新写入的再生过程。

刷新周期（再生周期）：规定在一定的时间内，对动态RAM的全部基本单元电路必作一次刷新，一般取2ms，这个时间称为刷新周期，又称再生周期。

1）集中刷新

集中刷新是在规定的一个刷新周期内，对全部存储单元集中一段时间逐行进行刷新，此刻必须停止读/写操作。

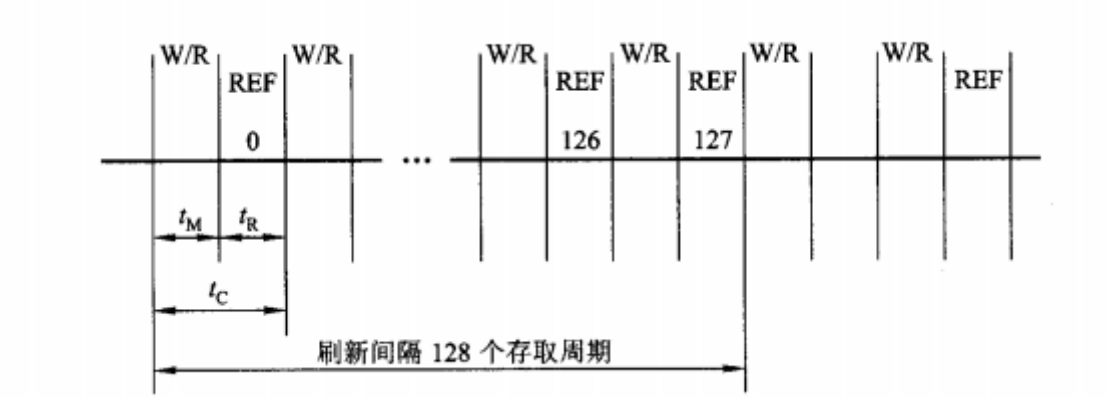
时间分配图：



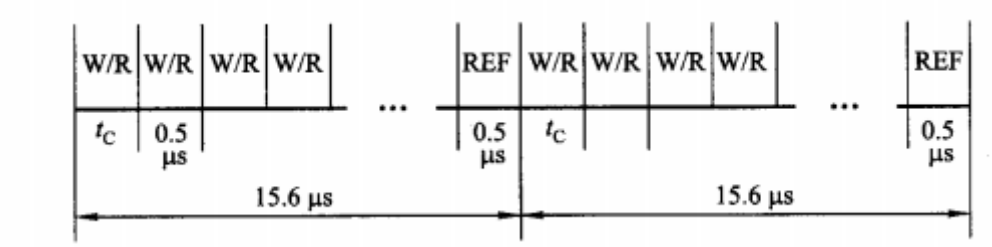
2）分散刷新

分散刷新是指对每行存储单元的刷新分散到每个存取周期内完成。

不存在死区；存取周期长；系统速度降低。
时间分配图：



3）异步刷新
异步刷新是前两种方式的结合，既可缩短“死时间”，又充分利用最大刷新间隔为2ms的特点。
时间分布图：

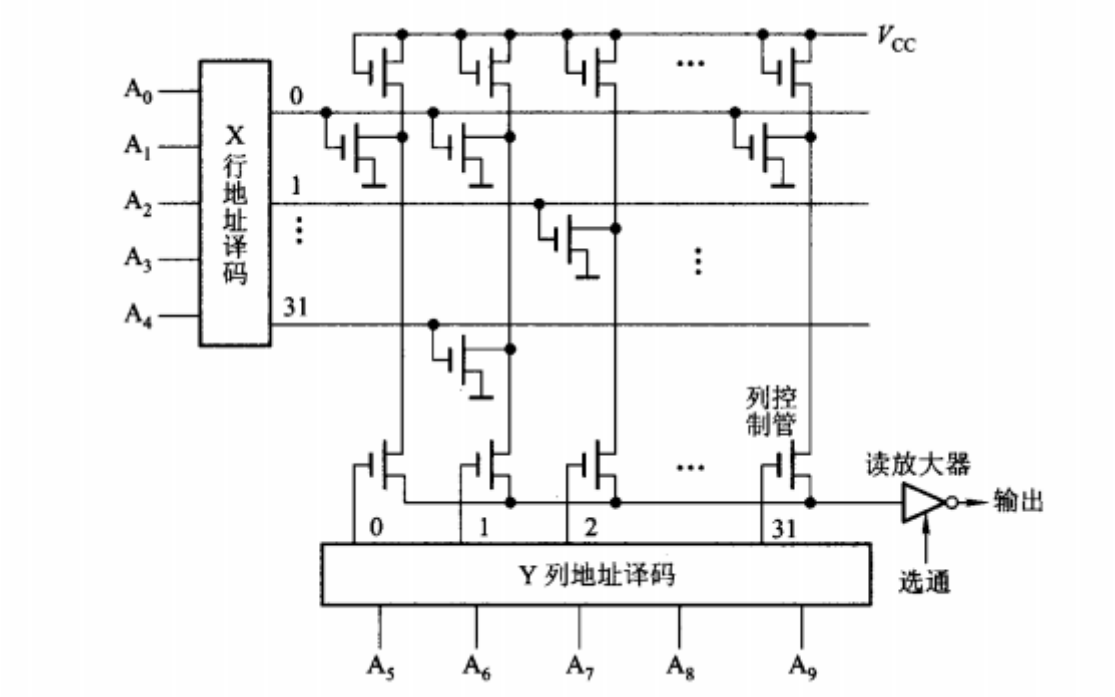


//这三种刷新方式很重要，一定要配合课本的（86-87页）文字解析，搞懂三种刷新的时间过程。
3、动态RAM与静态RAM的比较
动态RAM的应用比静态RAM广泛：
①在同样大小的芯片中，动态RAM的集成度远高于静态RAM。
②动态RAM行、列地址按先后顺序输送，减少了芯片引脚，封装尺寸页减少。
③动态RAM的功耗比静态RAM小。
④动态RAM的价格比静态RAM的价格便宜。
动态RAM也有缺点：
①由于使用动态元件（电容），因此它的速度比静态RAM低。
②动态RAM需要再生，故需配置再生电路，也需要消耗一部分功率。通常，容量不大的高速缓冲存储器大多用静态RAM实现。

4.2.4 只读存储器

对半导体ROM而言，基本器件为两种：MOS型和TTL型。
1、掩模ROM

下图为MOS型掩模ROM，容量为1K×1位，采用重合法驱动。



> 行、列地址线分别经行、列译码器，各有32行、列选择线。

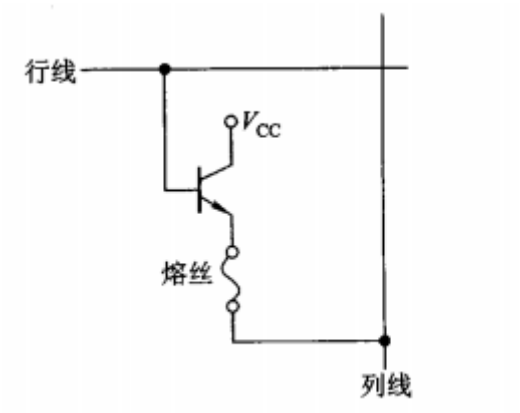
> 行选择线和列选择线交叉处既有耦合元件MOS管，也可没有。

> 列选择线各控制一个列控制管，32个列控制管的输出端共连一个放大器。

2、PROM

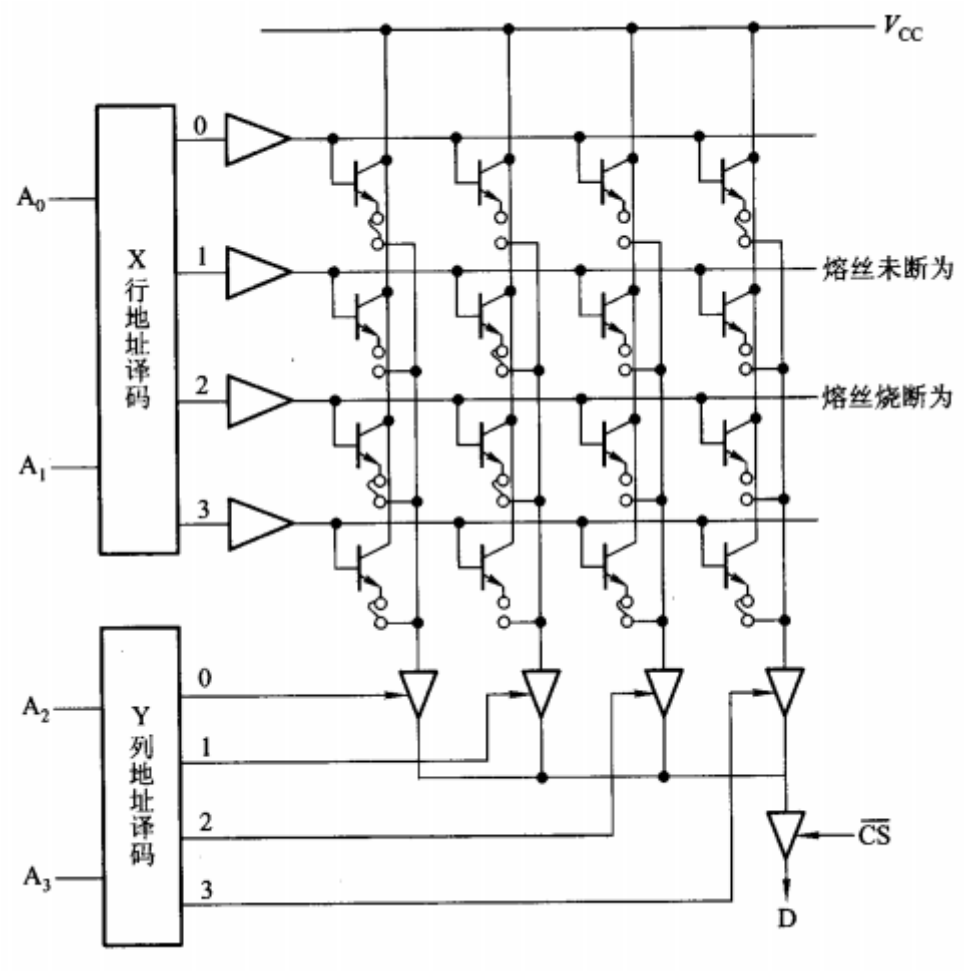
PROM是可以实现**一次性编程**的只读存储器。

下图示意一个有双极型电路和熔丝构成的基本单元电路。



> 在这个电路中，基极由行线控制，发射极与列线之间形成一条镍铬合金薄膜制成的熔丝（可用光刻技术实现），集电极接电源 V_{cc} 。熔丝断和未断可区别所存信息是“1”或“0”。

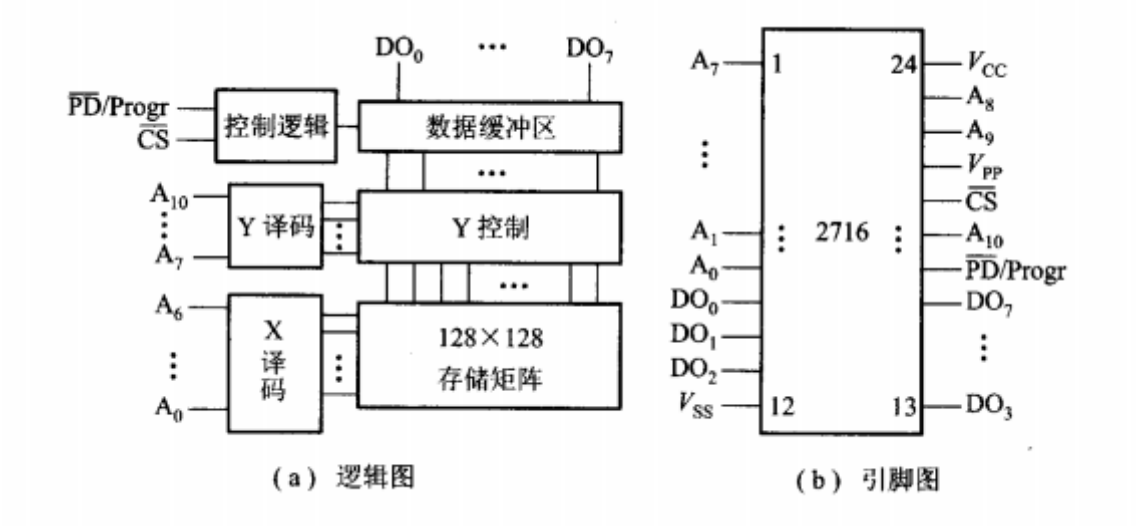
下图是由上图所示的基本单元电路构成的16 × 1位双极型镍铬熔丝式PROM芯片。用户在使用前，可按需将信息存入行、列交叉的耦合元件内。



3、EPROM

EPROM是一种可擦除可编程只读存储器，它可以由用户对其所存信息作任意次的改写。

下图为2716型EPROM的逻辑图和引脚图。



（a）逻辑图　　（b）引脚图

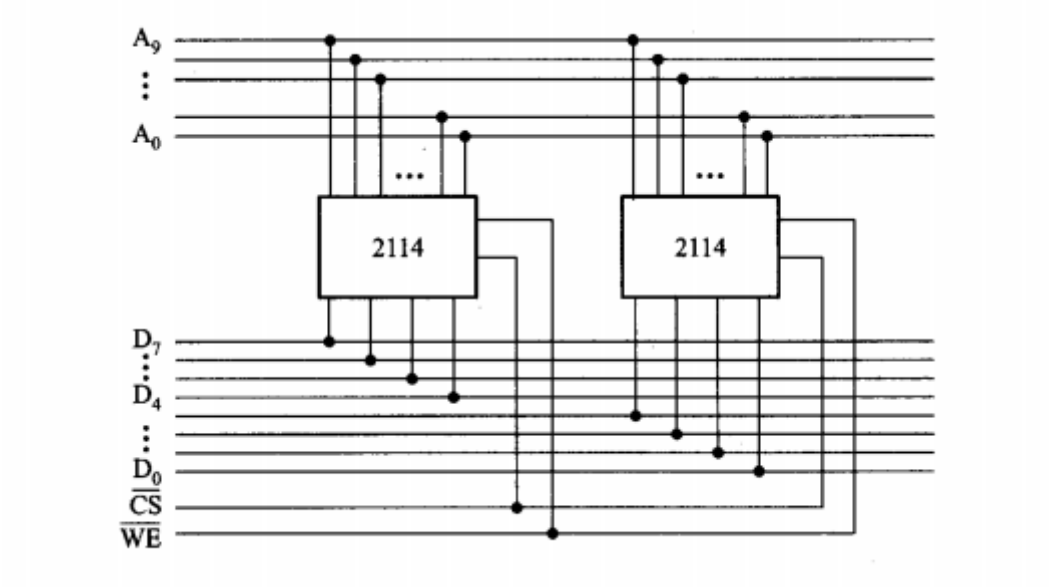
这类芯片的外引脚除地址线、数据线外，还有两个电源引出头V_{cc}和V_{pp}。
其中V_{cc}接+5V；V_{pp}平时接+5V。当其接+25V时用来完成编程。
V_{ss}为地。
$\overline{CS}$为片选端，读出时为低电平，编程写入时为高电平。
$\overline{PD}$/progr是功率下降/编程输入端，在读出时为低电平。

> EPROM的改写可用两种方法：
> 一是用紫外线照射，但擦除时间比较长，而且不能对个别需改写的单元进行单独擦除或重写。
> 二是用电气方法将存储内容擦除，再重写。甚至在联机条件下，用字擦除方式或页擦除方式，既可局部擦鞋，又可全部擦写，这种EPROM就是EEPROM。

4.2.5 存储器与CPU的连接

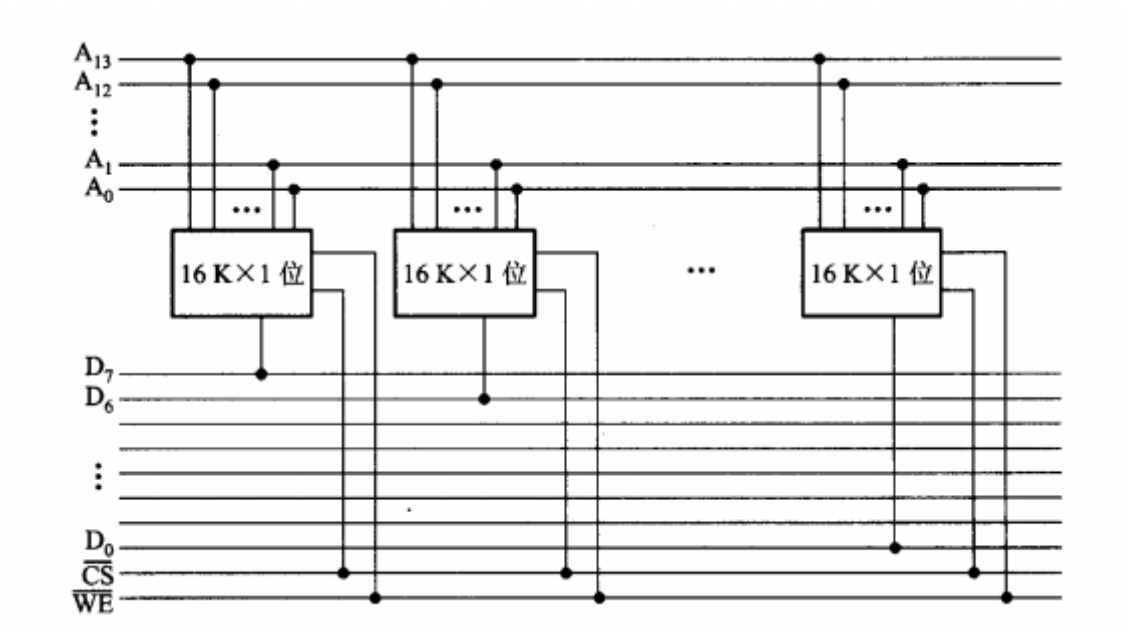
1、存储容量的扩展
将若干存储芯片连在一起才能组成足够容量的存储器，称为存储容量的扩展，通常有位扩展和字扩展。
（1）位扩展
位扩展指增加存储字长。

例如下图所示，2片1K × 4位的芯片可组成1K × 8位的存储器。



> 图中2片2114的地址线 A_9~A_0、$\overline{CS}$、$\overline{WE}$ 都分别连在一起，其中一片的数据线作为高4位 D_7~D_4,另一片的数据线作为低四位D_3 ~ D_0 。这样便构成了一个1K × 8位的存储器。

又如下图所示，将8片16K × 1位的存储芯片连接，可组成一个16K × 8位的存储器。



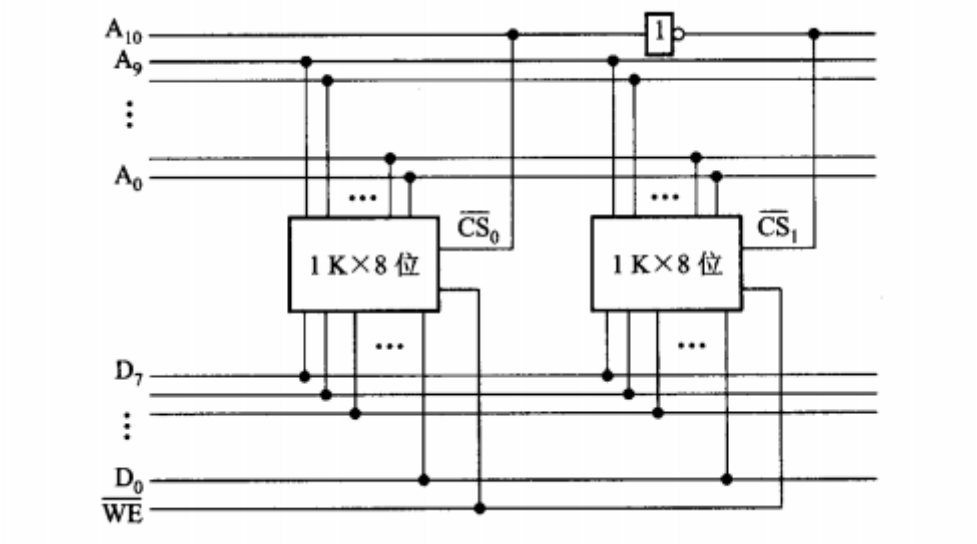
(注意，画图时不需要将所有连线画出或者所有位写出，只需画出首尾的几根或写出首尾的几个，中间用省略号表示，如上述例子中所示。)

（2）字扩展

字扩展指增加存储器字的数量。

例如下图，用2片1K × 8位的存储芯片可组成一个2K × 8位的存储器，即存储字增加了一

倍。

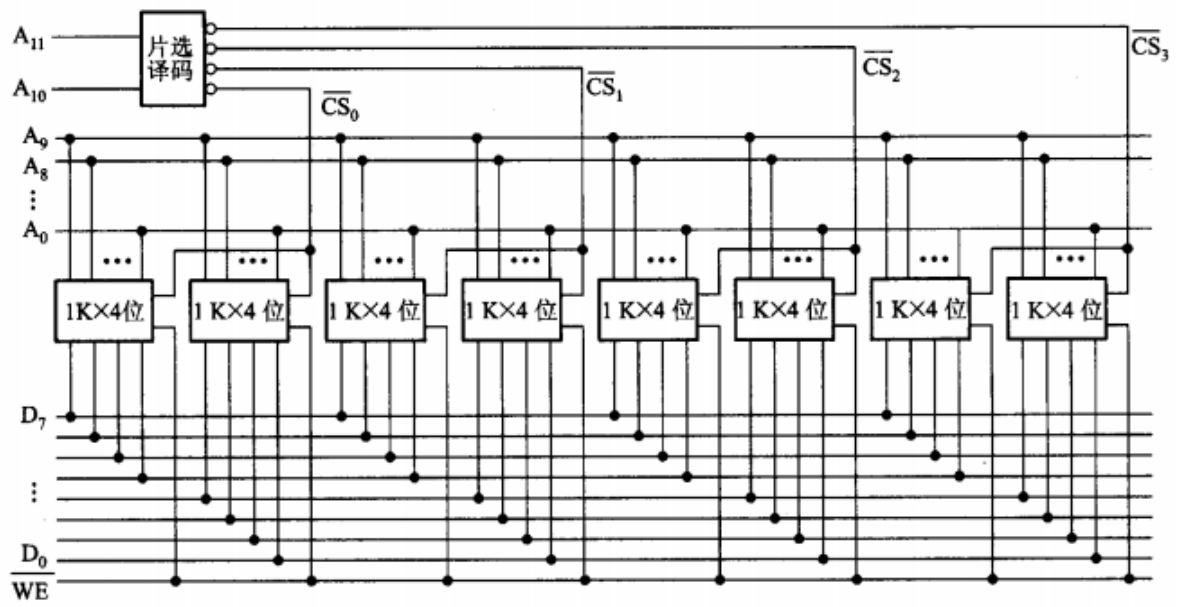


> 在此，将A_{10}用做片选信号。由于存储芯片的片选输入端要求低电平有效，故当A_{10}为低电平时，$\overline{CS_0}$有效，选中左边的1K × 8位芯片；当A_{10}为低电平时，反相后$\overline{CS_1}$有效，选中右边的1K × 8位芯片。右上角为反相器。

（3）字、位扩展

字、位扩展是既增加存储字的数量，又增加存储字长。

例如下图，是用8片1K × 4位的芯片组成4K × 8位的存储器。



> 由图可见，每2片构成一组1K × 8位的存储器。地址线A_{11}、A_{10}经片选译码器得到四个片选信号$\overline{CS_0}$、$\overline{CS_1}$、$\overline{CS_2}$、$\overline{CS_3}$，分别选择其中1K × 8位的存储芯片。$\overline{WE}$为读/写控制信号。

（画图时如果存储芯片数量过多，不必将所有芯片画出，画出部分芯片代表整体即可）
（字扩展和位扩展属于考试重点，一定要自己搞清楚原理并且动手画图）

2、存储器与CPU的连接

存储芯片与CPU芯片相连时，特别要注意片与片之间的地址线、数据线和控制线的连接。

（1）地址线的连接

存储芯片的容量不同，其地址线数也不同，CPU的地址线数往往比存储芯片的地址线多。

通常总是将**CPU地址线的低位**与存储芯片的地址线相连，**CPU地址线的高位**或在存储芯片扩充时用，或用作其他用途，如片选信号等。

> 例如，设CPU地址线为16位A_{15} ~ A_0,1K × 4位的存储芯片仅有10根地址线A_9 ~ A_0。此时，可将CPU的低位地址A_9 ~ A_0与存储芯片地址线A_9 ~ A_0相连。又如，当用16K × 1位存储芯片时，则其地址线由14根A_{13} ~ A_0，此时可将CPU的低位地址A_{13}~ A_0与存储芯片地址线A_{13} ~ A_0相连。

（2）数据线的连接

CPU的数据线数与存储芯片的数据线数也不一定相等，此时必须对存储芯片扩位，使其**数据位数与CPU的数据线数相等。**

（3）读/写命令线的连接

CPU读/写命令线一般可直接与存储芯片的读/写控制端相连，通常高电平为读，低电平为写。

（4）片选线的连接

片选线的连接是CPU与存储芯片正确工作的关键。

片选有效信号与CPU的访存控制信号$\overline{MREQ}$（低电平有效）有关。

若CPU访问I/O，则$\overline{MREQ}$为高电平，表示不要求存储器工作。

片选有效信号还和地址有关。CPU的地址线往往多于存储芯片的地址线，故那些未与存储芯片连上的高位地址必须和访存控制信号共同产生存储芯片的片选信号。

（5）合理选择存储芯片

存储芯片类型（RAM或ROM）和数量的选择，通常选用ROM存放系统程序、标准子程序和各类常数等，RAM则是为用户编程而设置的。

存储器与CPU连接的解题思路

分析用多少芯片完成存储系统

↓

计算芯片的地址线

↓

字选完成片内寻址

（确定多少CPU地址线直接连接到存储芯片）

↓

是否全译码方式

↓

确定片选方式（把地址线的高位与译码芯片相连）

（请结合课本94-97页的习题例4.1和例4.2理解解题方式，并能独立解出存储器与CPU连接的习题）

4.2.6 存储器的校验

在计算机运行过程中，大量的数据需要存储，难免会出现错误。为了及时发现错误和改正错误，通常将数据配成汉明编码。

4.2.6.1 汉明码的组成

汉明码具有一位纠错能力。

> **编码的最小距离：**
> 在一种编码系统中，任意两组合法代码之间的最少二进制位数的差异。

根据纠错理论得：

$L-1=D+C \quad 且D \geq C$

也就是编码距离 L 越大，检测错误位数 D 越大，纠正错误的位数 C 越大，而且检错能力恒大于等于纠错能力。因此，如果在信息编码中增加若干检测位，就能显著提高检错纠错能力。

（1）检测位位数k的确定

假设检测的二进制代码位数为 n 位，需要加入 k 位检测位，组成$n+k$位代码。为准确检测、定位错误，k 应满足：

$$2^k \geq n+k+1$$

例如，一段含4位2进制数的编码，需要至少3位检测位。

（2）检测位位置的确定

k 的位数确定后，接下来设定它们在传送代码中的位置。

> 注意：
> 对于一段$n+k$位的代码，在讨论检测位的位置及取值时，我们有两种编号方式。一种为自左

> 向右编号为第1，2，3，...n+k 位；另一种为自右向左编号。两者的校验本质是一致的，只是形式不同。在此节中，我们采用授课老师采用的自右向左编号的方式，即第二种。

设 $n+k$位代码自右至左依次编为第1,2,3,...$n+k$位，而将k位检测位记作 C_i, (i =1,2,4,...2^{k-1} ），分别安插在 $n+k$位代码编号的第1,2,4,...，2^{k-1}位上。
例如，我们要传递的信息为$A_4A_3A_2A_1$(即$n=4$)，根据$2^k \geq n+k+1$易知$k=3$，则$n+k$位编码如下：

二进制序号	7	6	5	4	3	2	1
名称	A_4	A_3	A_2	C_4	A_1	C_2	C_1

（3）检测位检测任务的确定
检测位位置的设置是为了保证它们能承担$n+k$位信息中不同“小组”的奇偶检测任务，具体分配如下:
C_1 检测的g_1小组包含第1,3,5,7,9,11...位。
C_2 检测的g_2小组包含第2,3,6,7,10,11,14,15...位。
C_4 检测的g_4小组包含第4,5,6,7,12,13,14,15...位。
C_8 检测的g_8小组包含第8,9,10,11,12,13,14,15,24...位。
...
那么这些小组是如何划分的呢?

- 每个小组g_i有且仅有一位为它所独占,即g_i小组独占第2^{i-1}位(i=1,2,3,⋯)。
- 每两个小组g_i和g_j共同占有一位是其他小组没有的，即每两小组g_i和g_j共同占有第$2^{i-1}+2^{j-1}$位(i,j=1,2,3,⋯)。
- 每三个小组g_i,g_j和g_l共同占有第$2^{i-1}+2^{j-1}+2^{l-1}$位(i,j=1,2,3,⋯) ,是其他小组没有的。
- 依次类推，便可确定每组所包含的各位。

小技巧:
例如，我们要传递的信息为$A_4A_3A_2A_1$，$n+k$位编码如下：

二进制序号	7	6	5	4	3	2	1
名称	A_4	A_3	A_2	C_4	A_1	C_2	C_1

A_4的序号为7，按权展开为：4,2,1
A_3的序号为6，按权展开为：4,2
A_2的序号为5，按权展开为：4,1
A_1的序号为3，按权展开为：2,1

接下来，分析各检测位的小组包含的位：

对于 C_4 即寻找**含权为4**的数据位：A_4,A_3,A_2（即第5,6,7位）

对于 C_2 即寻找**含权为2**的数据位：A_4,A_3,A_1（即第3,6,7位）

对于 C_1 即寻找**含权为1**的数据位：A_4,A_2,A_1（即第3,5,7位）

（未标明检测位自身所在的位）

（4）检测位取值的确定

仍以（2）中的例子为例，令$A_4A_3A_2A_1=0101$，则：

二进制序号	7	6	5	4	3	2	1
名称	0	1	0	C_4	1	C_2	C_1

检测位的取值为“检测任务小组”内各位的异或：

$C_4=A_4\bigoplus A_3\bigoplus A_2=0\bigoplus 1\bigoplus 0=1$

$C_2=A_4\bigoplus A_3\bigoplus A_1=0\bigoplus 1\bigoplus 1=0$

$C_1=A_4\bigoplus A_2\bigoplus A_1=0\bigoplus 0\bigoplus 1=1$

故0101的汉明码应为$A_4A_3A_2C_4A_1C_2C_1$，即0101101。

> 读者会发现，这个编码与教材书上同为0101的例子，而产生汉明码为0100101不一致，这是因为我们的编号顺序与教材不一致，想必读者已经体会到了**说明编号规则**的重要性。

4.2.6.2 汉明码的纠错过程

汉明码的纠错过程实际上是对传递后的编码形成新的检测位，根据新的检测位就可以得知传递过程是否产生错误，错误在哪一位。记新的检测位为P_i,P_i的状态是由原检测位C_i及其所在的小组内各位异或产生。

$P_4=C_4\bigoplus A_4\bigoplus A_3\bigoplus A_2$

$P_2=C_2\bigoplus A_4\bigoplus A_3\bigoplus A_1$

$P_1=C_1\bigoplus A_4\bigoplus A_2\bigoplus A_1$

以前例为例：

二进制序号	7	6	5	4	3	2	1
名称	0	1	0	1	1	0	1

假设传递后的信息如上，那么：

$P_4=C_4\bigoplus A_4\bigoplus A_3\bigoplus A_2=1\bigoplus 0\bigoplus 1\bigoplus 0=0$

$P_2=C_2\bigoplus A_4\bigoplus A_3\bigoplus A_1=0\bigoplus 0\bigoplus 1\bigoplus 1=0$

$P_1=C_1\bigoplus A_4\bigoplus A_2\bigoplus A_1=1\bigoplus 0\bigoplus 0\bigoplus 1=0$

根据配偶原则，传递后形成的新检测位全为 0 表示传递没有出错。
反之，假设收到的传递后的编码为

二进制序号	7	6	5	4	3	2	1
名称	0	1	0	1	1	1（此位出错）	1

此时新的检测位取值为：

$$P_4 = C_4 \bigoplus A_4 \bigoplus A_3 \bigoplus A_2 = 1 \bigoplus 0 \bigoplus 1 \bigoplus 0 = 0$$
$$P_2 = C_2 \bigoplus A_4 \bigoplus A_3 \bigoplus A_1 = 1 \bigoplus 0 \bigoplus 1 \bigoplus 1 = 1 \text{（非0）}$$
$$P_1 = C_1 \bigoplus A_4 \bigoplus A_2 \bigoplus A_1 = 1 \bigoplus 0 \bigoplus 0 \bigoplus 1 = 0$$

P_2出错，意味着C_2, A_4, A_3, A_1中有出错，又结合P_4 P_1没有出错，分析知是C_2出错，将其改正即得到正确的传递结果。

> 小技巧：
> $P_4P_2P_1$的值为010,读者有没有发现这个数看作二进制正好是2！事实上，这个一个可行的结论，$P_4P_2P_1$传入“38译码器”，输出端Y_0表示传递正确，Y_1至Y_7对应第1~7位出错。

> 提醒：
> 本节我们采用的是自右向左的编号方式，相信读者理解本节后，对于教材的另一种编号方式下的汉明码，也能够轻松学会。

4.3 高速缓冲存储器

4.3.1 概述

1、问题的提出
为什么我们需要高速缓冲存储器（Cache）呢?

原因有二：

- CPU与主存之间加一级缓存，主存可将CPU需要的信息提前送至缓存，在主存与I/O设备交换时，CPU可以直接从缓存读取信息，不必等待而影响效率。
- 由于主存速度的提高始终跟不上CPU的发展，高速缓存Cache可以解决主存与CPU速度不匹配的问题。

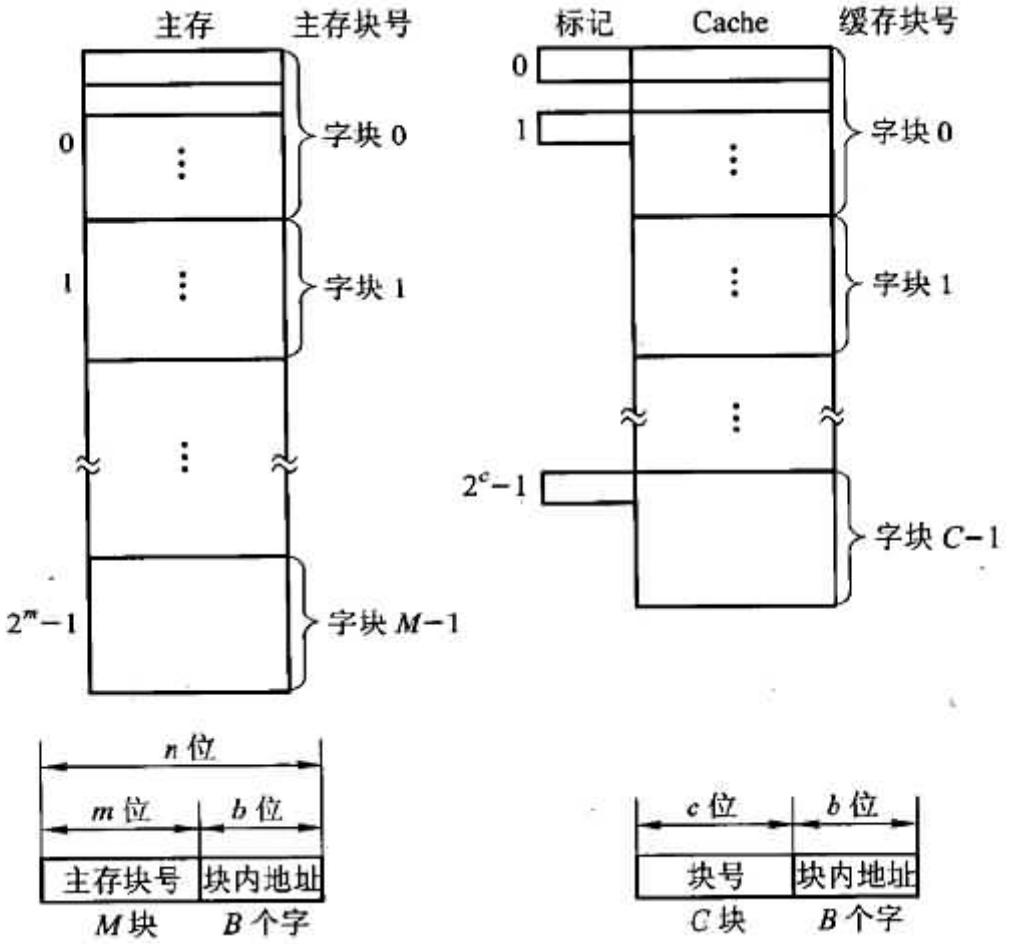
那么，我们的设想是否可能呢?

通过分析发现，在一段时间内，CPU从主存取指令或者取数据时只是对主存局部区域的访问。这是由于指令和数据在主存内都是连续存放的，不是随机的，而是相对的簇聚，这就使得CPU在执行程序时，访存具有相对的局部性，这就称为**程序访问的局部性原理**。

根据这一原理，只要将CPU近期要用到的程序和数据提前从主存送到Cache，那么就可以实现CPU在一定时间内只访问Cache而无须访问主存。

2、Cache的工作原理

下图是Cache-主存存储空间的基本结构图



分析：

设主存由2^n个可编址的字组成，每个字有唯一的n位地址。为了将容量远大于缓存的主存与缓存建立映射关系，将主存和缓存都分成若干字块，每个字块内有若干字，并使得字块一样大（即含有同样多的字）。这样设置的目的是便于在需要时将主存的某个字块整体调入Cache区，也就是基于前面提到的**程序访问的局部性原理**的考虑。

如此，主存和Cache的地址都发生了变化。在寻找地址时，需要先找“字块”，再在字块内找“字”。故，地址分为两部分。对于主存：高m位表示主存的块地址，低b位表示块内地址，则$2^m = M$表示主存的块数。同理，缓存的高c位表示缓存的块地址，低b位表示块

内地址，则$2^c = C$表示缓存块数，易知C远小于M。主存和缓存都用b位表示块内地址，即$B = 2^b$反映了块的大小，称B为**块长**。

由于主存容量远大于缓存容量（主存块数远大于缓存块数），因此，一个缓存块不可能惟一地、永久地只对应一个主存块，故每个缓存需要设一个标记以表示当前存放的事哪一个主存块，该标记的内容**相当于主存块的编号**。

注意：

- 任何时刻都有一些主存块位于缓存块中。
- 若CPU想读取的主存的某个字已在Cache中，则直接访问Cache（CPU与Cache之间通常一次传送一个字，即字传送）。
- 而如果CPU想读取的某个字不在Cache中，此时需要将该字所在的字块整体一次调入Cache中（字块传送）。

命中与不命中：
上述“注意”第二条称为“CPU访问Cache命中”，第三条称为“CPU访问Cache不命中”。CPU读取信息时，将主存地址的高m位（或m位的一部分）与缓存块的标记进行比较，以判断所读的信息是否已经在缓存中。

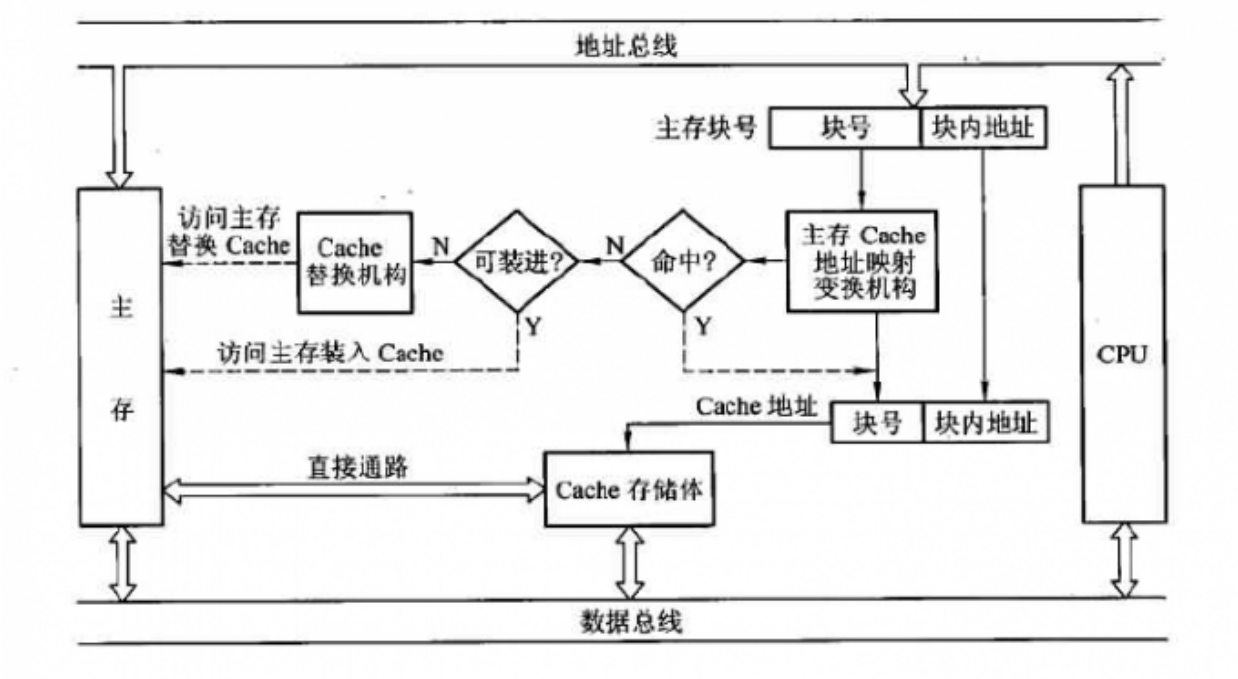
命中率：
Cache的容量与块长是影响Cache效率的重要因素。通常用**命中率**来衡量**Cache的效率**。在一个程序执行期间，设N_c为访问 Cache的总命中次数,N_m为访问主存的总次数，则命中率h为:

$$h = \frac{N_c}{N_c + N_m}$$

设t_c为命中时的Cache访问时间，t_m为未命中时的主存访问时间，$1 - h$表示未命中率，则Cache-主存系统的平均访问时间t:
$t_a = ht_c + (1 - h)t_m$
用e表示访存效率，则有：

$$e = \frac{t_c}{t_a} \times 100\% = \frac{t_c}{ht_c + (1 - h)t_m} \times 100\%$$

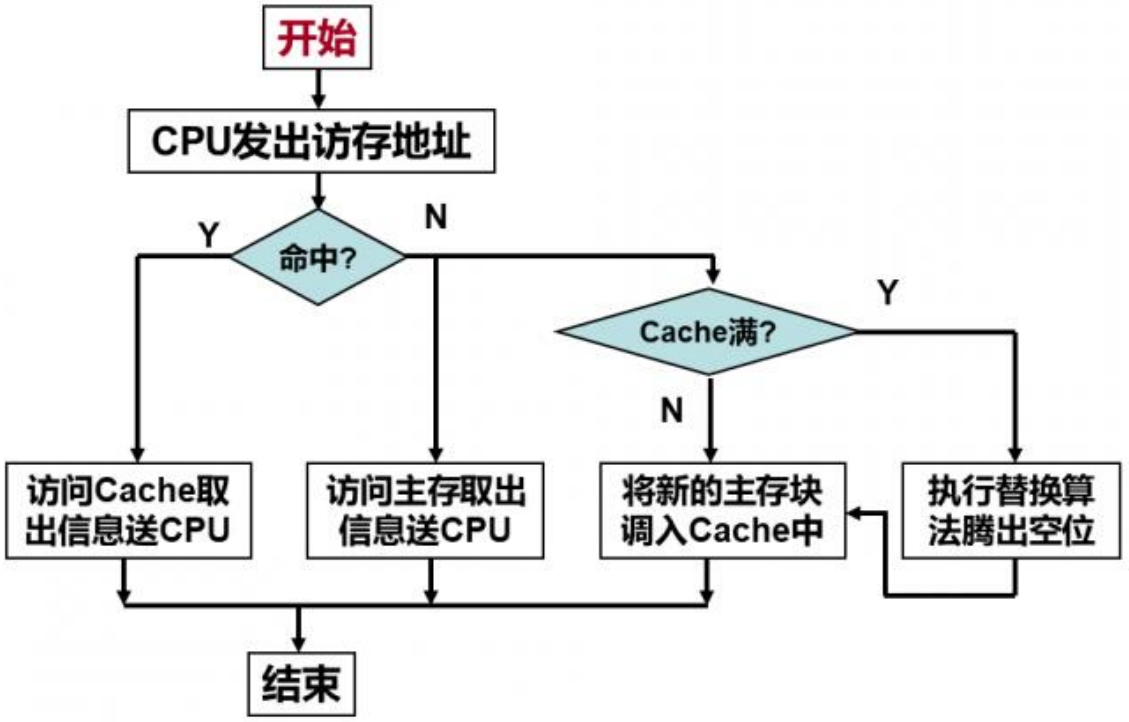
3、Cache的基本结构
Cache的基本结构如下图：



分析：

它主要由Cache存储体、地址映射变换机构、 Cache替换机构几大模块组成。

1. Cache存储体以块为单位与主存交换信息。
2. 地址映射变换机构是将CPU送来的主存地址转换为Cache地址。
   如果转换后的Cache块已与CPU欲访问的主存块建立了对应关系，即已命中，则CPU直接访问Cache。否则，CPU不仅将该字从主存取出，同时将它所在的主存块一并调入Cache。在调入时，若Cache处于未被装满的状态则调入。反之就得采用替换策略。
3. 当Cache已满，无法接受来自主存块的信息时，就由替换机构按一定的替换算法来确定移出哪个块返回主存，把新的主存块调入。
4. Cache的读写操作
   读操作如下图所示：



写操作比较复杂，因为对Cache块内写入信息，本质是对其对应的主存块相应位置的信息的改写，必须与被映射的主存块内的信息完全一致。因此会出现如何使Cache与主存内容保持一致的问题。目前主要采用以下两种方法：

- 写直达法（Write-through）
  又称存直达法（Store-through），即进行写操作时，既写入缓存又写入主存。易知，这种方法随时保证主存与缓存的数据一致性，但同时该方法增加了访存次数。
- 写回法（Write-back）
  又称拷回法（Copy-back）。它的原理是进行写操作时，只把数据写入缓存，仅当Cache中的数据被替换出去时才写入主存。容易理解写回法Cache中的数据会和主存中的不一致。为了解决这个问题，Cache中的每一块要增设一个标志位，该位有两个状态：“清”（未被修改过，与主存一致）和“浊”（被修改过，与主存不一致）。如此，在Cache替换时，“清”的块不必写回主存，因为此时Cache中的数据与主存中是一致的，在写Cache时，将标志位设为“浊”，替换时将此Cache块写回主存，同时置为“清”。

两种方法各有特点。写直达法中。读操作时不涉及对主存的写操作，更新策略比较容易实现。在写操作中，既要写入Cache又要写入主存，因此写直达法的“写”操作时间就是访问主存的时间。写回法中读操作Cache失效发生数据替换时，被替换的块需写回主存，增加了Cache的复杂性。这种方法对主存的写操作只发生在块替换时，而且对Cache中的一个块的多次写操作只需要一次写入主存，故可减少写操作次数。“写”操作时只写入Cache，故“写”操作时间就是访问Cache的时间，因此速度快。

4、Cache的改进

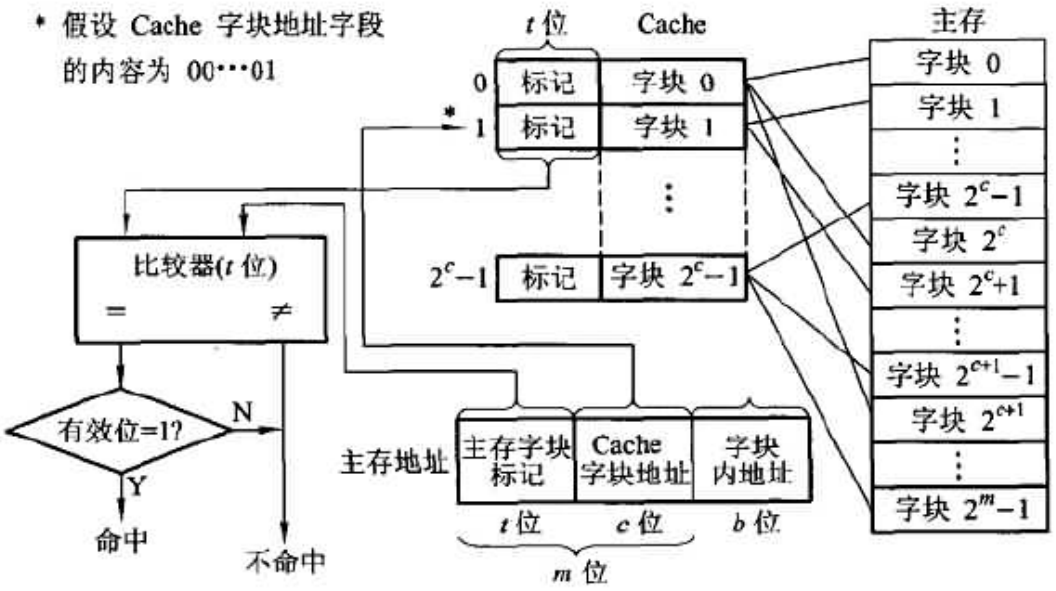
1. 增加 Cache 的级数
2. 统一缓存和分立缓存的选用
   参看课本P_{114}。

4.3.2 Cache——主存地址映射

由主存地址映射到Cache地址称为地址映射。地址映射方式很多，有直接映射（固定的映射关系）、全相联映射（灵活性大的映射关系）、组相联映射（上述两种映射的折中）。

1、直接映射

下图反映了直接映射方式中主存与缓存的对应关系



要点：

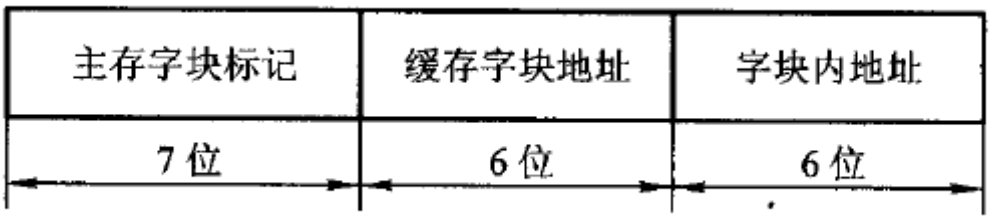
1. 图中每个主存块只和一个缓存块建立对应关系，映射关系式为：
   $i = j \mod C$ 或 $i = j \bmod 2^c$
   其中，i为缓存块号，j为主存块号，C为缓存块数。
2. 每个缓存块对应于若干个主存块。
3. 主存地址被分成三部分：低b位是指Cache的块内地址，与缓存字块大小有关。中间c位为字块地址。高 t位是指主存字块标记，它被记录在建立了对应关系的缓存块的”标记”位中。
4. 当缓存接到CPU送来的主存地址，根据中间c位找到相应的Cache字块，将字块的“标记”与主存地址的高t位相符判断。若符合且有效位为“1”(**有效位**用来识别

Cache块中的数据是否有效，因为有时Cache中的数据是无效的），表示该Cache块已和主存的某块建立了对应关系，则可根据b位块内地址从Cache块中取得数据；若不符合（或有效位为“0”），则从主存读入新的字块来替代旧的字块，同时将信息送往CPU,并修改 Cache“标记”，将有效位置于“1”。

直接映射的缺点：

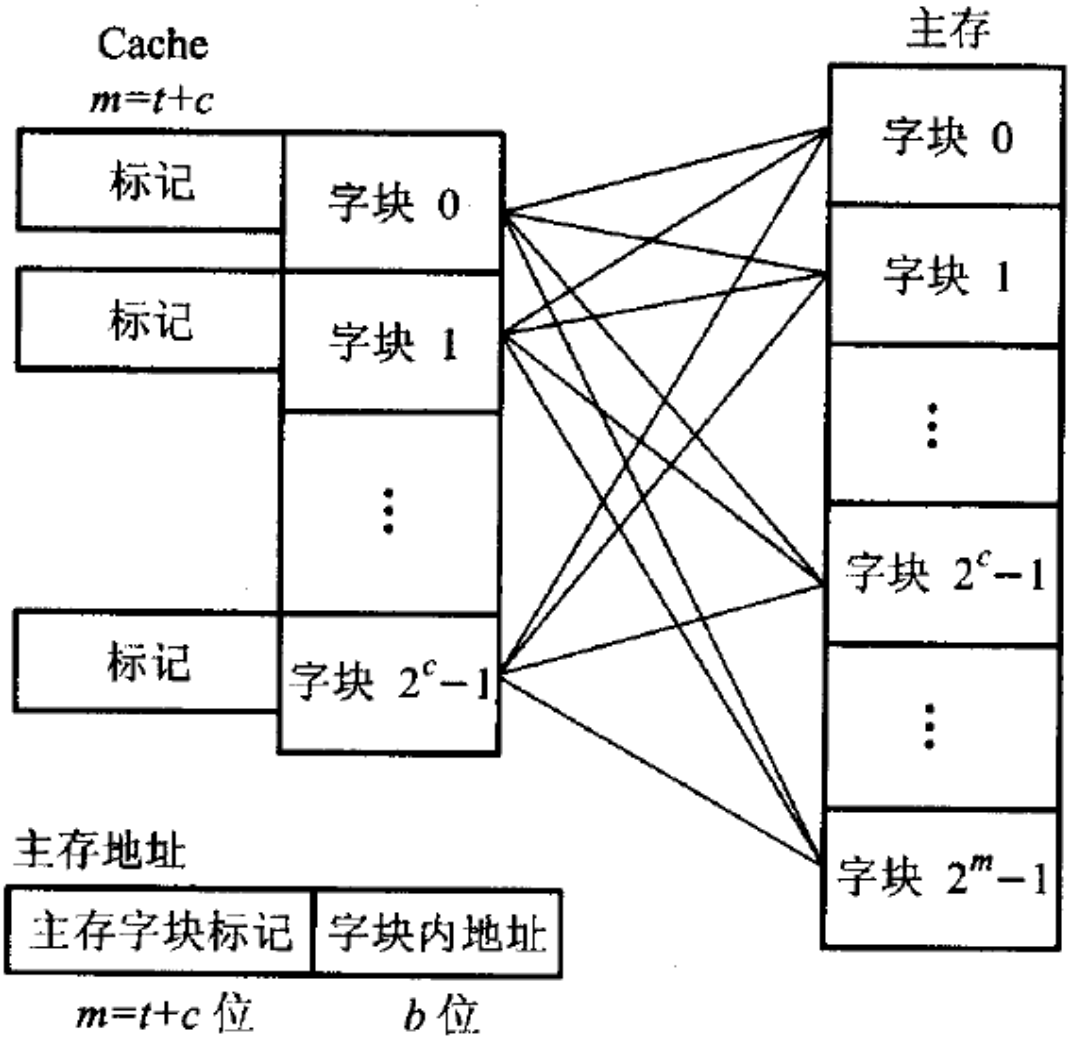
1.不够灵活，每个主存块只能“固定地”对应某个缓存块，会出现某时某缓存空着的情况，使缓存的存储空间得不到充分的利用。

2.如果程序要重复访问对应同一缓存位置的不同主存块，就要不停地进行替换，降低命中率。

> 例题：
> 假设主存容量为512KB，Cache容量为4KB，每个字块为16个字，每个字32位。请画出直接映射方式下主存地址各字段位数示意图。
> 答：
> 根据 Cache容量为4KB($2^{12} = 4K$),Cache地址为12位。由每字32位（即每个字 4 B）知Cache共有4KB/4B=1 K字。因每个字块16个字，故Cache中有1K/16 = 64块。
> 在直接映射方式下，主存地址字段的各段分配如下图所示。
>
> 
>
>
> 其中字块内地址为6位(4位表示16个字，2位表示每字 4字节），缓存共64块，故字块地址为6位，主存字块标记为主存地址长度与Cache地址长度之差 ，即19-12=7位。

2、全相联映射

下图反映了全相联映射方式中主存与缓存的对应关系：

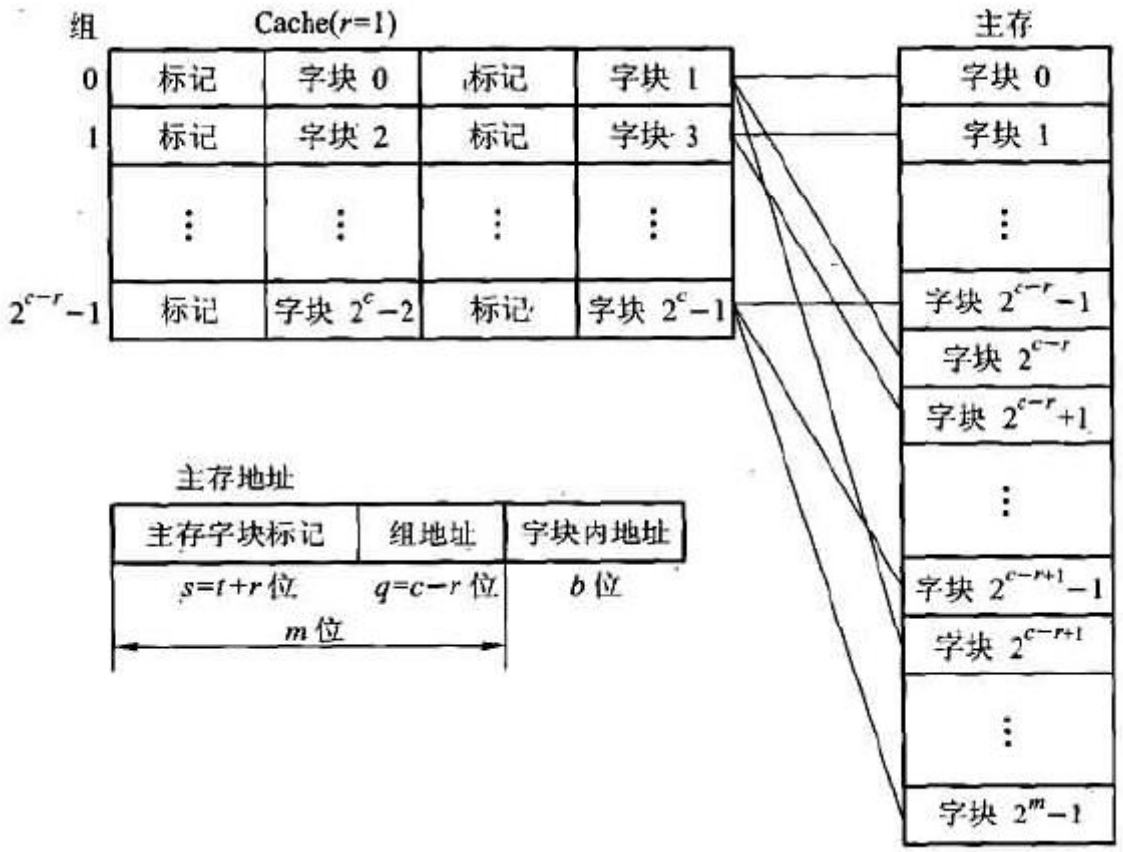


要点：

1. 全相联映射允许主存中**每一字块**映射到Cache中的**任何一块**位置上。
2. 这种映射方式可以从已被占满的Cache中替换出任一旧字块。
3. 与直接映射相比，它的主存字块标记从 t位增加到 $t+C$位，这就使“标记”的位数增多，而且访问 Cache 时主存字块标记需要和Cache的 **全部“标记”** 位进行比较，才能判断出所访问主存地址的内容是否已在Cache内。
4. 这种映射方式灵活，命中率也更高，缩小了块冲突率。但所需的逻辑电路甚多，成本较高，实际的Cache还要采用各种措施来减少地址的比较次数。

3、组相联映射

下图反映了组相联映射方式中主存与缓存的对应关系：

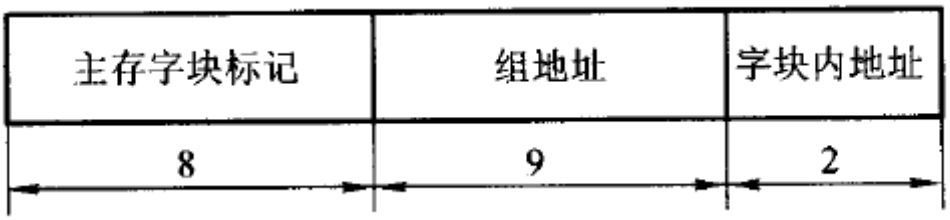


要点：

1. 组相联映射是对直接映射和全相联映射的一种折衷。主存块映射Cache的“组”时相当于直接映射，Cache“组”内的后续对应相当于全相联映射。
2. Cache字块地址字段由直接映射中的c位变为组地址字段q位，且$q = c - r$, 其中2^c,表示Cache的总块数，2^q表示Cache的分组个数，2^r表示每组内包含的块数。主存字块标记字段由直接映射的t 位变为$s = t + r$位。
3. 假设Cache分为Q组，每组有R块，则有如下关系：
   $i = j \ mod \ Q$
   其中，i为缓存的组号，j为主存的块号。某一主存块按模Q映射到缓存的第i组内。
4. 显然，主存的第j块会映射到Cache的第i组内，两者**一一对应**，属于直接映射关系；另一方面，主存的第j块可以映射到Cache的**第i组内中的任一块**，这又体现出全相联映射关系。
5. 可见，组相联映射的性能及其复杂性介于直接映射和全相联映射两者之间，当$r = 0$ 时是直接映射方式，当 $r = c$ 时是全相联映射方式。

举个例子：

- 假设 $c=5, q=4$, 则 $r=c-q=1$ 。其实际含义为：Cache共有 $2^c=32$个字块，共分为$2^q=16$ 组，每组内包含$2^1=2$ 块。组内2块的组相联映射又称为**二路组相联。**
- 根据假设，组相联映射的含义是：主存的某一字块可以按模16映射到Cache某组的任一字块中。即主存的第0,16,32…字块可以映射到Cache第0组的2个字块中的任一字块；主存的第15,31,47…字块可以映射到Cache第15组中的任一字块。

> 例题：
> 假设主存容量为$512K \times 16$位，Cache容量为$4K \times 16$位，块长为 4 个 16 位的字，访存地址为字地址。在二路组相联映射方式下，设计主存的地址格式。
> 答：
> 每个字 16 位，Cache共$4K$字，每块 4 个字,块内地址需要 2 位。共$1K$个字块，由二路组相联的条件，一组内有 2 块，得Cache共分 512 组，即组地址需要 9 位。主存字块标记为 19 - 9 - 2 = 8 位，其地址格式如下图所示：
>
> 
>

4.3.3 替换策略

前面我们提到，当我们需要将主存块调入缓存块但是相应的位置被占用了时，就需要进行数据的替换，因此产生了替换策略的问题。在直接映射中，由于某个主存块只与一个固定的Cache字块有映射关系，因此替换策略很简单。而在组相联和全相联映射中，主存块可以写入Cache中若干位置，就产生了“替换哪个Cache字块更好？”的问题，即所谓**替换算法问题。**常用的替换算法**有先进先出算法**、**近期最少使用算法**和**随机法。**

1、先进先出（First-In-First-Out，FIFO）算法

FIFO 算法选择最早调入 Cache 的字块进行替换，容易实现，开销小，但没有根据访存的局部性原理，故不能提高 Cache 的命中率。

2、近期最少使用(Least Recently Used, LRU)算法

LRU 算法比较好地利用访存局部性原理，替换出近期用得最少的字块。它比较复杂，一般采用简化的方法，只记录每个块最近一次使用的时间。LRU 算法的平均命中率比FIFO高。

3、随机法

随机法是随机地确定被替换的块，比较简单，可采用一个随机数产生器产生一个随机的被替换的块，它也没有根据访存的局部性原理，故不能提高 Cache 的命中率。

4.4 辅助存储器

辅助存储器作为主存的后援设备又称为外部存储器，简称外存，它与主存一起组成了存储器系统的主存——辅存层次。

这一部分作为扩展内容，读者可根据情况自行阅读。

参看课本P_{123}。

第5章 输入输出系统

本章概述

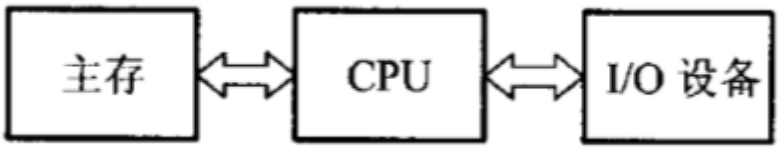
本章重点分析I/O设备与主机交换信息的三种控制方式(程序查询、中断和DMA)及其相应接口功能和组成

5.1 概述

5.1.1 输入输出系统的发展概况

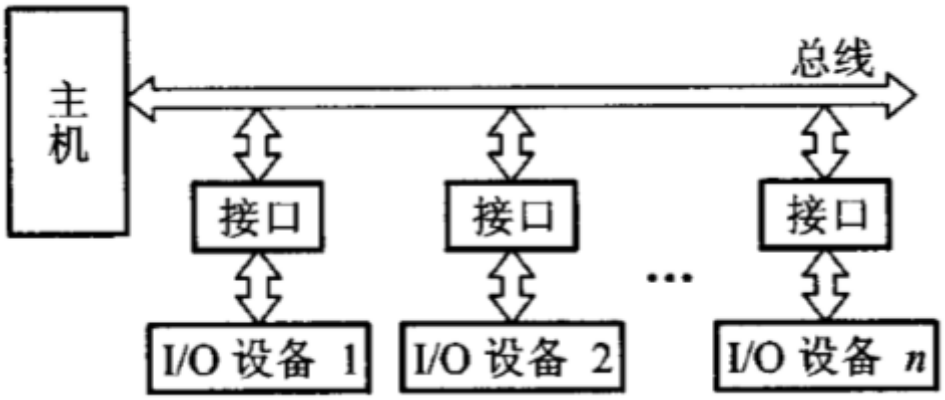
1.早期阶段

分散连接，CPU和I/O串行工作，程序查询方式



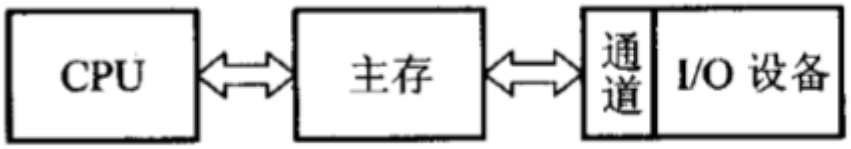
2.接口模块和DMA阶段(本章主要讲的这个阶段)

总线连接，CPU和I/O设备并行工作，中断方式、DMA方式



3.具有通道结构的阶段

常用于大中型计算机系统，用通道来负责管理I/O设备以及实现主存与I/O设备之间交换信息的部件



4.具有I/O处理机的阶段

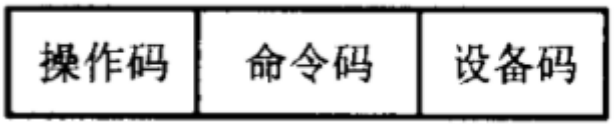
基本独立于主机工作，既可完成I/O控制，又可完成码制转换、格式处理、数据块检错、纠错等功能

5.1.2 输入输出系统的组成

1.I/O软件

（1）I/O指令(CPU指令的一部分)

是机器指令的一类，格式与其他指令有相似之处也有不同点，可以和其他机器指令字长相等



操作码字段：可作为I/O指令与其他指令的判别码

命令码字段：具体操作

设备码字段：多台I/O设备的选择码

（2）通道指令

具有通道的I/O系统专门设置的指出参与传送(读或写)的数据组在主存中的首地址、末地址(字节数)

2.I/O硬件

（1）有接口的I/O系统中主要包括I/O设备和接口模块两大块

（2）通道系统中主要包括I/O设备、设备控制器和通道组成

5.1.3 I/O设备与主机的联系方式

1.I/O设备编址方式

通常将I/O设备码看做地址码

（1）统一编址

将I/O地址码看做存储器地址的一部分(占用了存储空间，减少了主存容量，不需要专用I/O指令)

（2）不统一编址

I/O地址与存储器地址分开，对I/O设备的访问必须通过专用的I/O指令(不影响主存容量，需要专用I/O指令)

2.设备寻址：通过设备选择电路识别是否被选中

3.传送方式

（1）串行：同一时间只传送一位信息，速度慢、只需要一根数据线一根地线(I/O设备与主机距离远时较为合理)

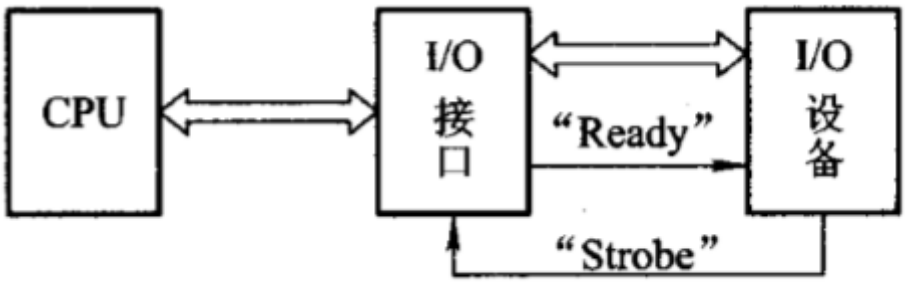
（2）并行：同时发出多位信息，速度快、需要的数据线多

4.联络方式

（1）立即响应

（2）异步工作采用应答信号

异步并行应答



（3）同步工作采用同步时标

5.I/O设备与主机的连接方式

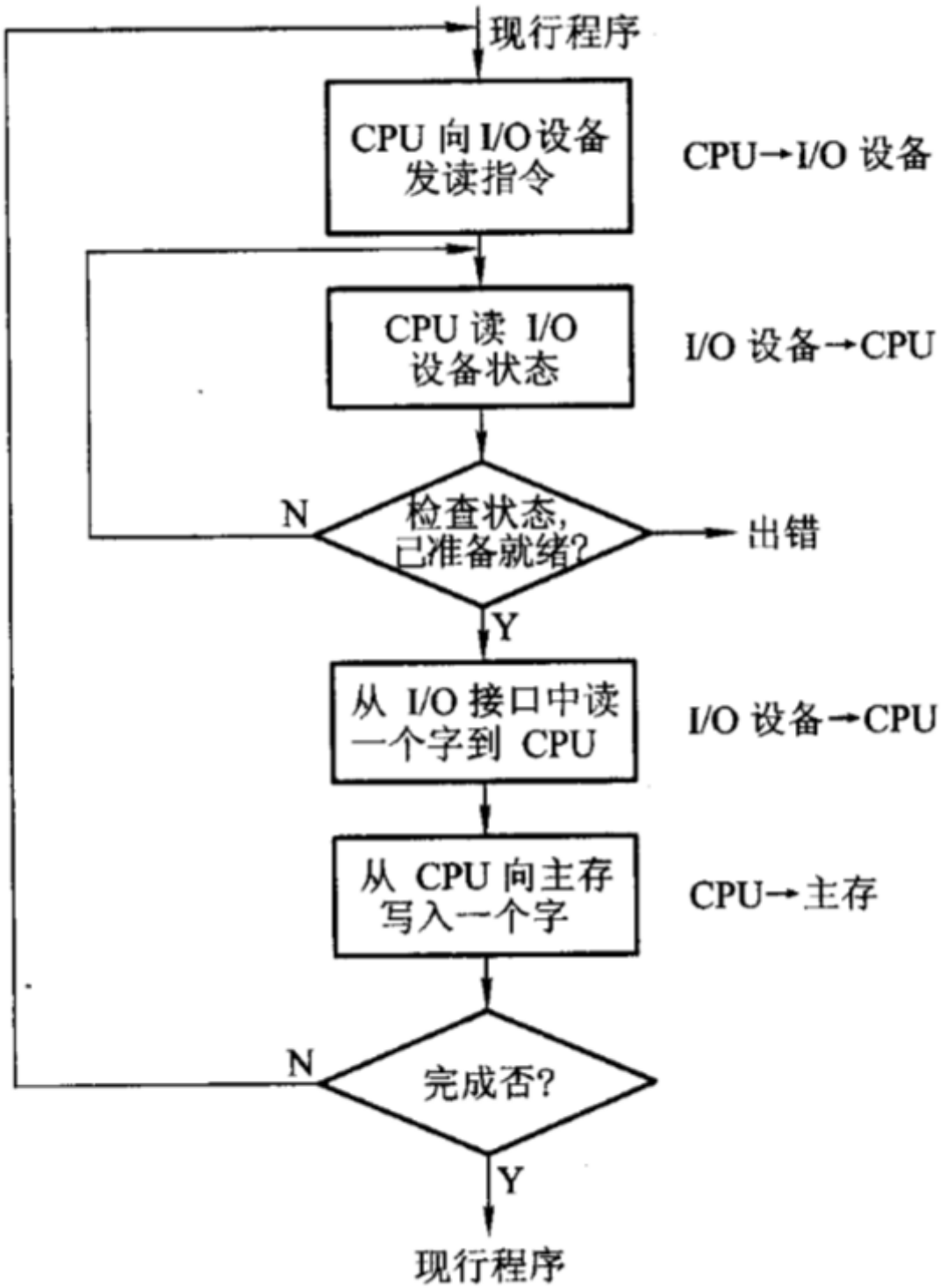
（1）辐射式连接：设备增删比较困难，出现在计算机发展的初级阶段

（2）总线连接：便于增删设备

5.1.4 I/O设备与主机信息传送的控制方式

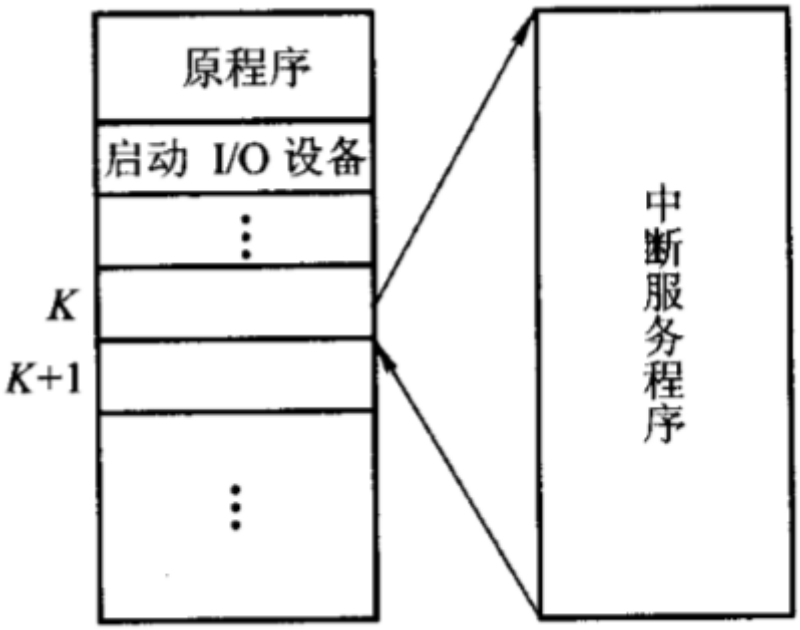
1.程序查询方式

CPU不断查询I/O设备是否准备好(踏步等待)直至准备好了，CPU和I/O设备串行工作，效率低



2.程序中断方式

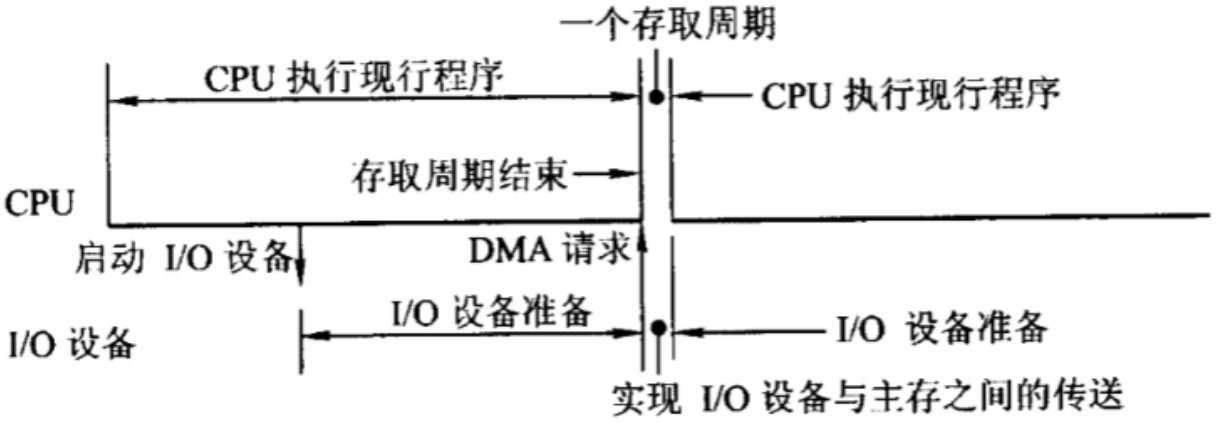
CPU不查询设备是否准备好了，而是设备准备好后通知CPU，发出中断请求，没有踏步等待，CPU 和 I/O 并行工作，提高效率



3.DMA方式

进一步改善，避免I/O设备与主存进行数据交换造成的占用CPU内部寄存器、浪费CPU资源

主存和 I/O 之间有一条直接数据通道，不中断现行程序，CPU 和 I/O 并行工作，DMA占用存取周期称为周期挪用（周期窃取），进一步提高CPU资源利用率



5.2 I/O设备(不考)

5.3 I/O接口

5.3.1 概述

接口可以看做是两个系统或两个部件之间的交接部分，既可以是两个硬件设备之间的连接电路，也可以是两个软件的共同逻辑边界。I/O接口指主机与I/O设备之间的硬件电路及其相应软件控制

1.可实现I/O设备的选择

2.可实现数据缓冲

3.可实现数据串并模式转换

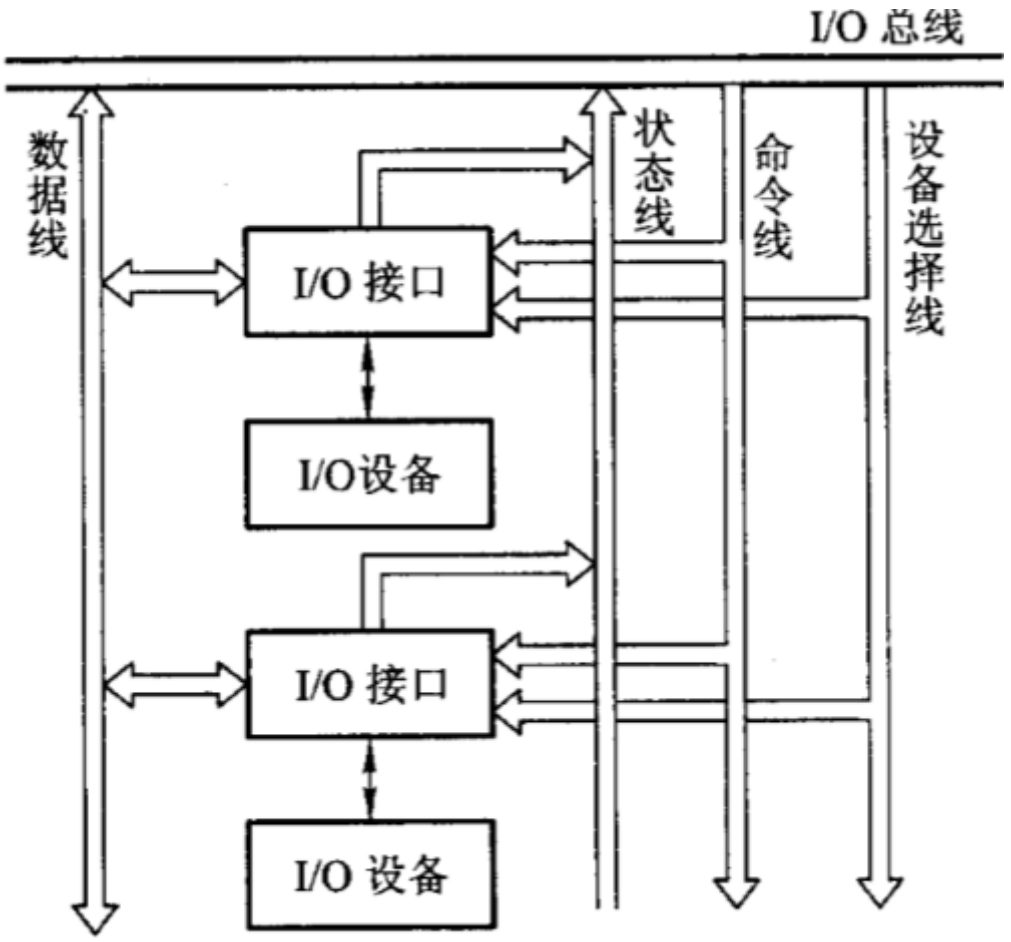
4.可实现电平转换

5.可传送控制指令

6.可监视设备工作状态

5.3.2 接口的功能和组成

1.总线连接方式的I/O接口电路



（1）数据线

与主机之间数据代码的传送线，根数通常等于存储字长的位数或者字符的位数，通常是双向的(若为单向必须要用两组)

（2）设备选择线

传送设备码，根数取决于I/O指令中设备码的位数，同样可以一组双向(也可两组单向)

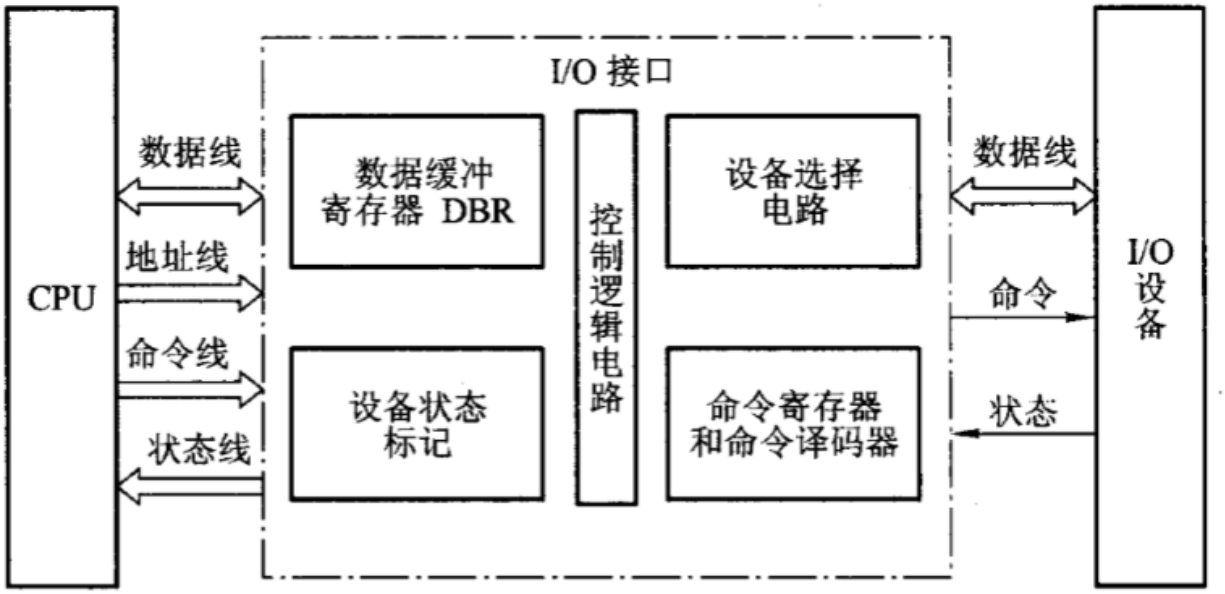
（3）命令线

主要用以传输CPU向设备发出的各种信号，单向，根数与命令信号多少有关

（4）状态线

将设备的状态报告给主机的信号线，单向

2.接口功能和组成



（1）选址功能——设备选择电路

（2）传送命令功能——命令寄存器、命令译码器

（3）传送数据功能——数据缓冲寄存器

（4）反映I/O设备工作状态的功能——设备状态标记

5.3.3 接口类型

1.按数据传送方式分类：并行接口(Intel 8255)、串行接口(Intel 8251)

2.按功能选择的灵活性分类：可编程接口(Intel 8255、 Intel 8251)、不可编程接口(Intel 8212)

3.按通用性分类：通用接口(Intel 8255、 Intel 8251)、专用接口(Intel 8279、 Intel 8275)

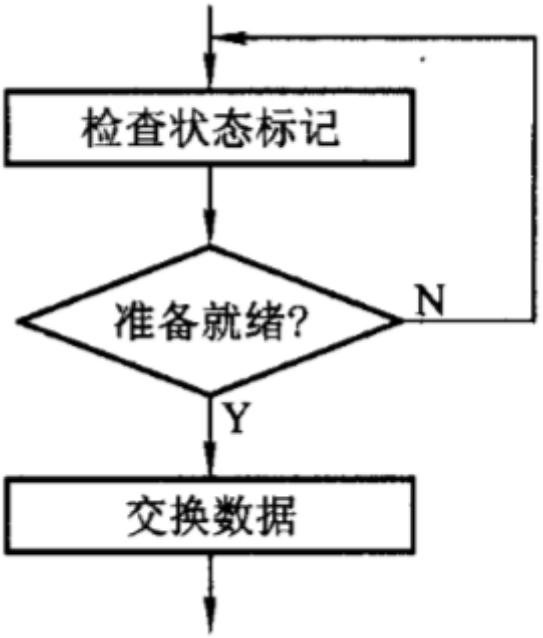
4.按数据传送的控制方式分类：中断接口(Intel 8259)、DMA 接口(Intel 8257)

5.4 程序查询方式

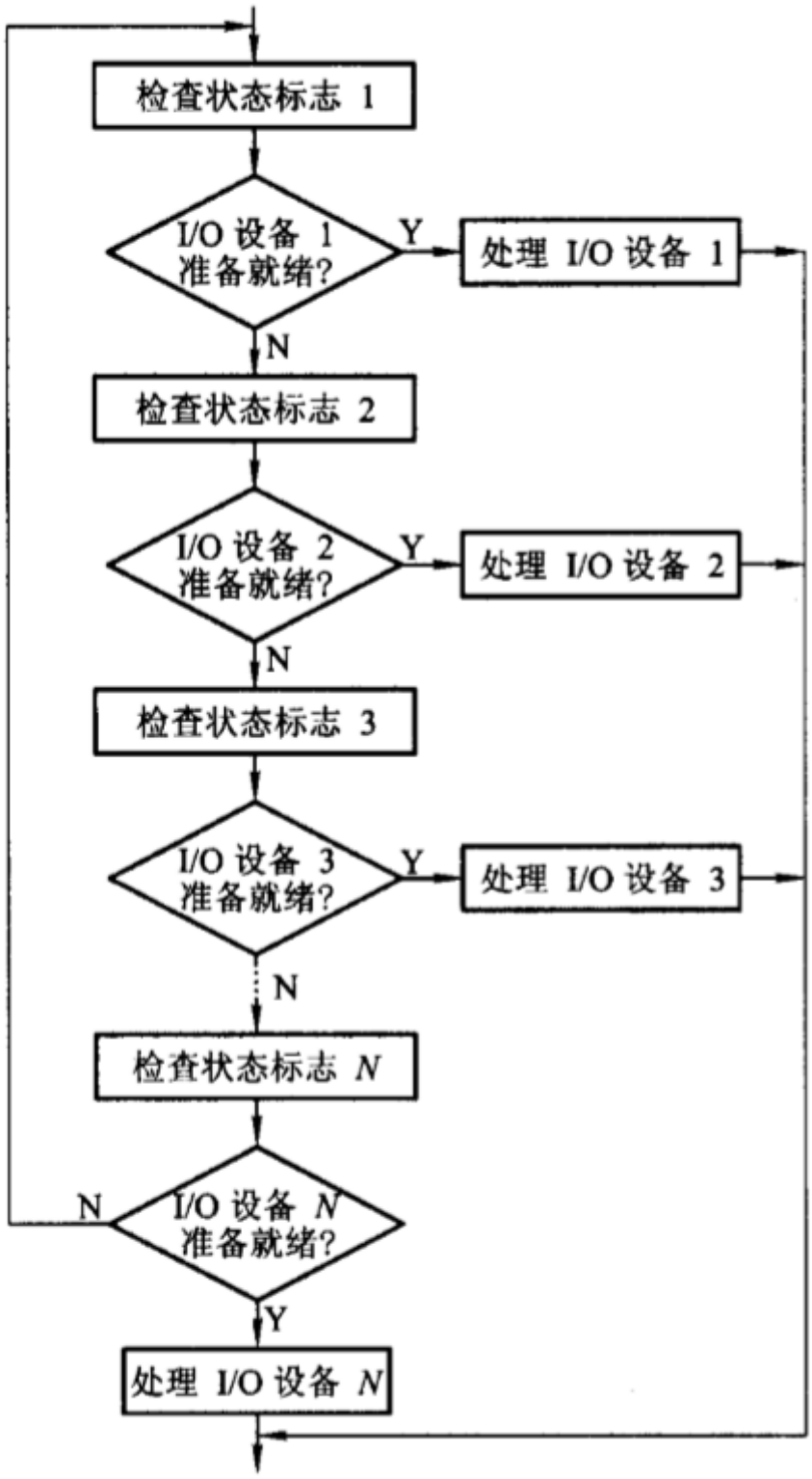
5.4.1 程序查询流程

当I/O设备较多时，要按各个设备的优先级进行逐级查询

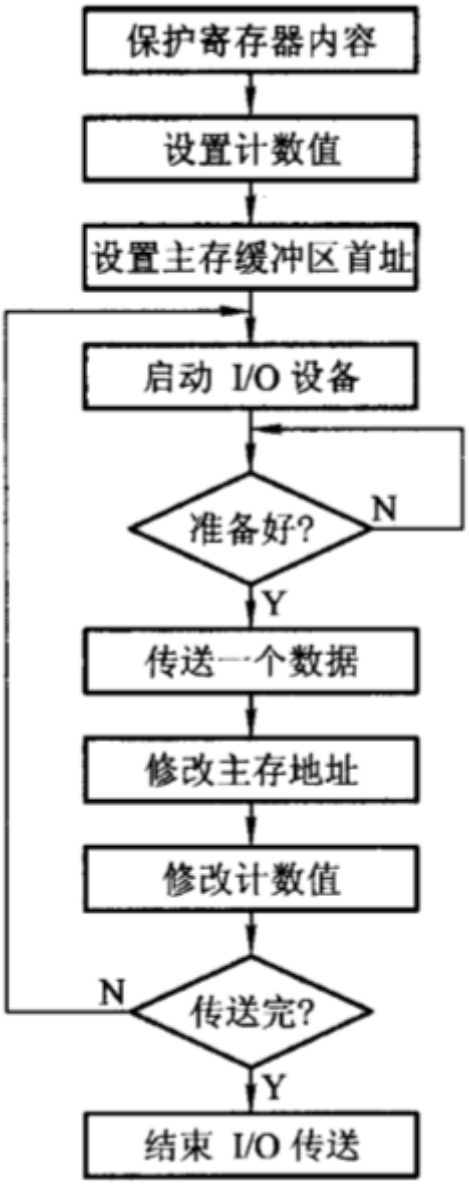
单个设备查询流程



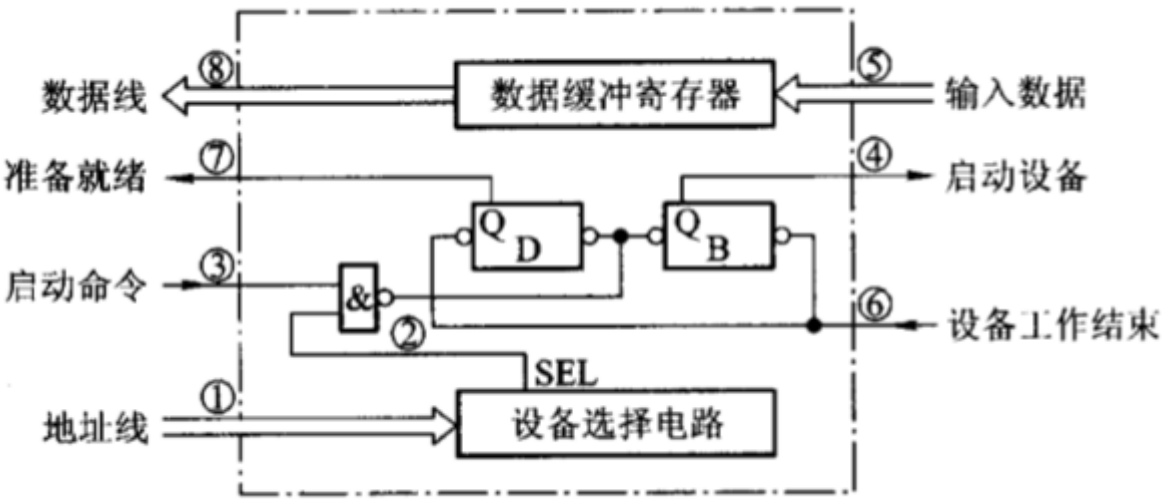
多个设备查询流程


检查状态标志 1
I/O 设备 1
准备就绪?
Y
处理 I/O 设备 1
N
检查状态标志 2
I/O 设备 2
准备就绪?
Y
处理 I/O 设备 2
N
检查状态标志 3
I/O 设备 3
准备就绪?
Y
处理 I/O 设备 3
N
检查状态标志 N
N
I/O 设备 N
准备就绪?
Y
处理 I/O 设备 N

程序查询方式的总流程



5.4.2 程序查询方式的接口电路



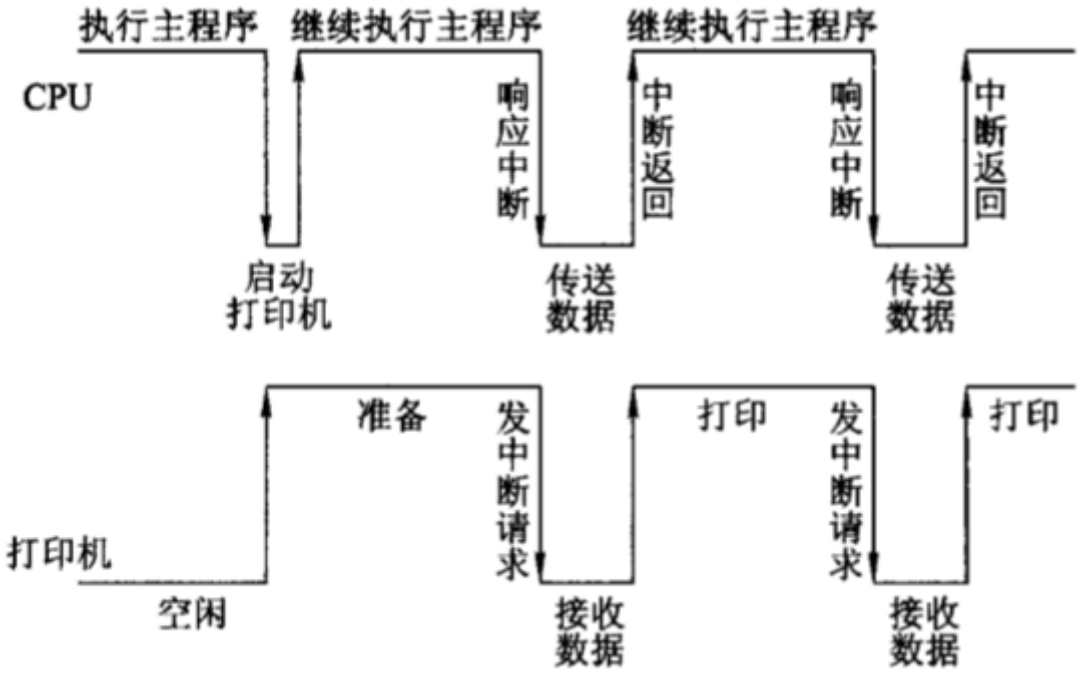
5.5 程序中断方式(重点)

5.5.1 中断的概念

计算机在执行程序的过程中出现异常情况或者特殊请求时，计算机停止现行程序先处理异常情况或者特殊请求，处理完后再继续执行原程序。中断能合理发挥效能提高效率。

5.5.2 I/O中断的产生

I/O设备与主机交换信息时设备速度慢，在设备准备的同时CPU运行现行程序提高利用率，准备好后发出中断请求

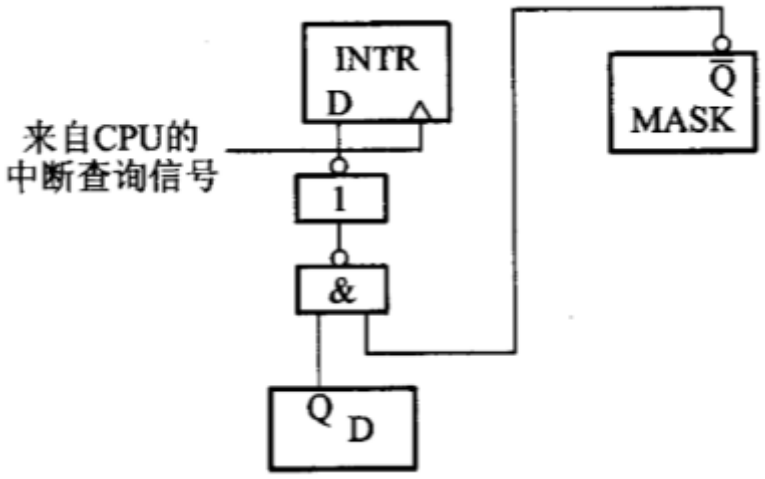


5.5.3 程序中断方式的接口电路

1.中断请求触发器和中断屏蔽触发器

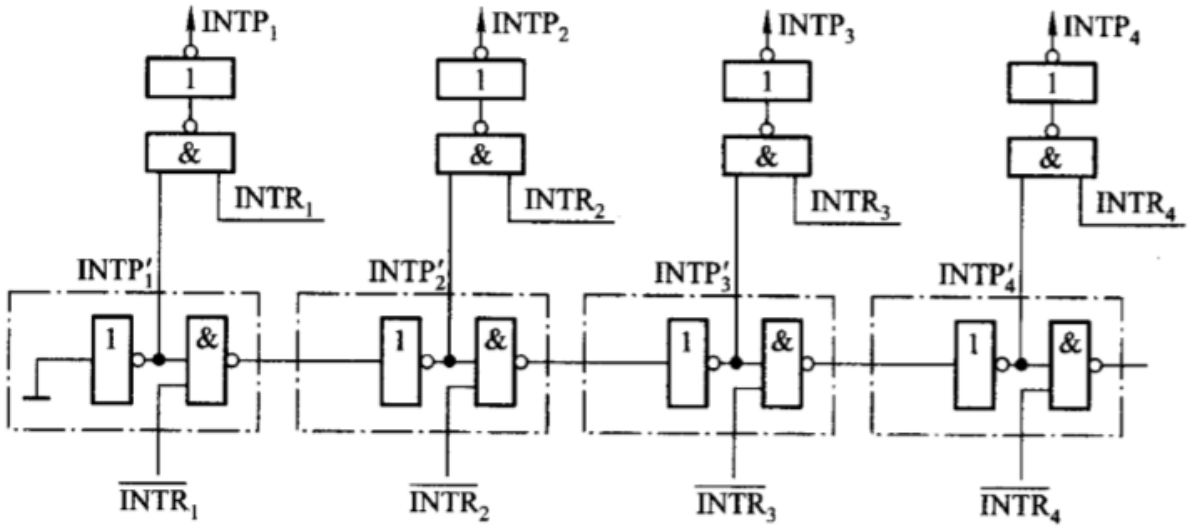
INTR：中断请求触发器(INTR=1有请求)

MASK：中断屏蔽触发器(MASK=1被屏蔽)



2.排队器

当多个中断源同时提出请求时，速度越高的设备优先级越高进行排队



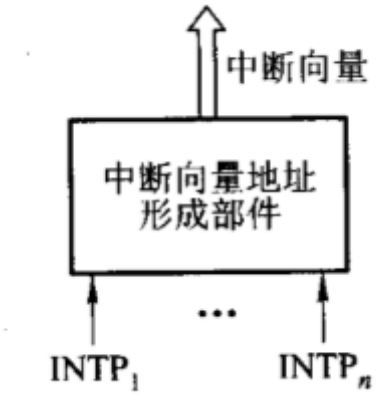
3.中断向量地址形成部件(设备编码器)

1.软件法(第八章讲)

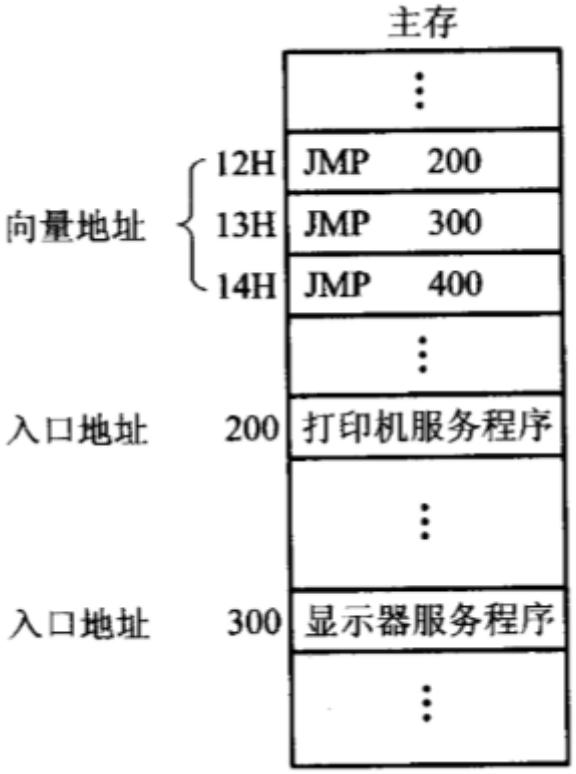
2.硬件法

由硬件产生向量地址，再由向量地址找到 入口地址

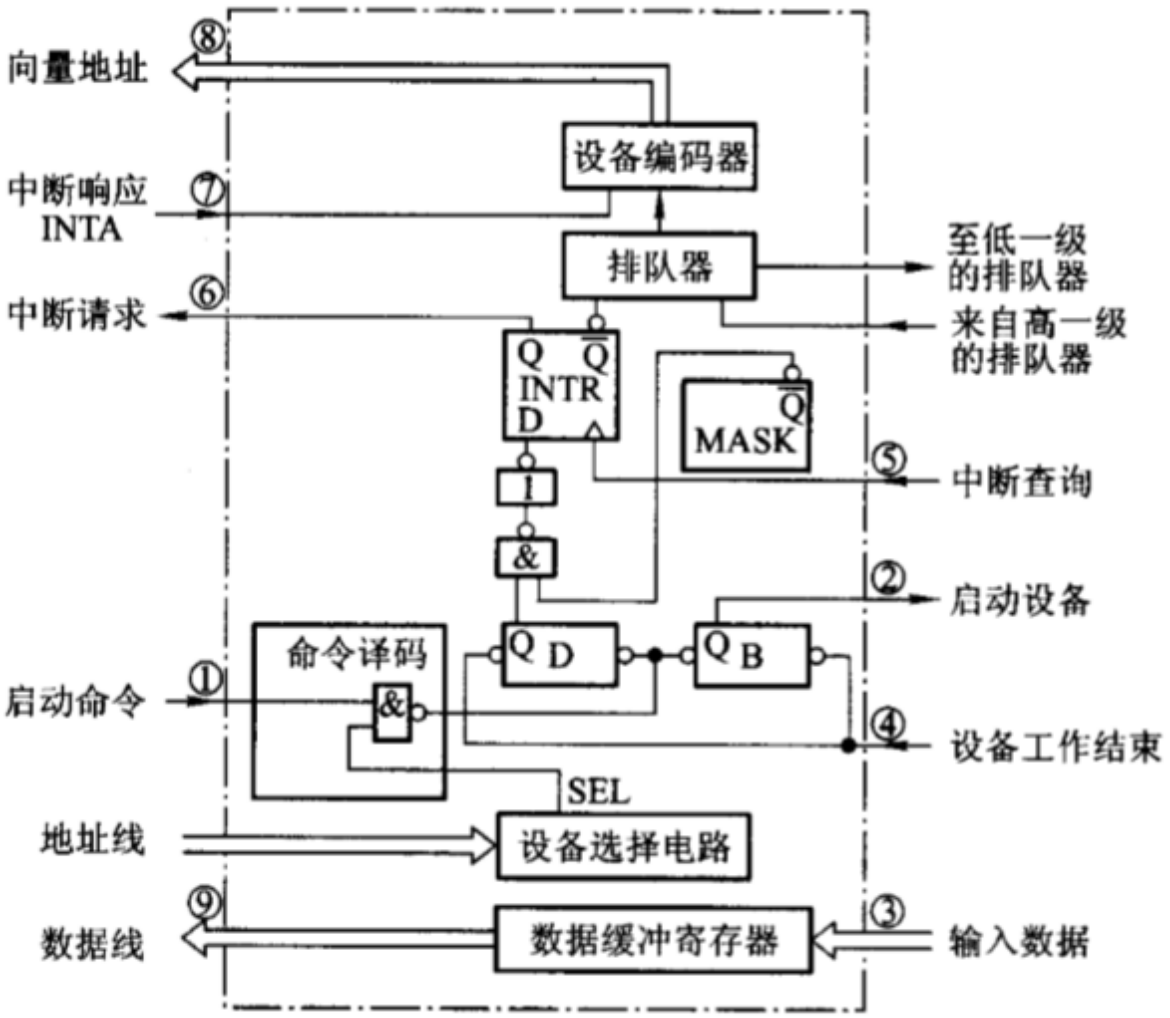
向量地址位数与中断源个数有关



向量地址与中断服务程序入口地址是两个不同的概念，通过向量地址寻找入口地址



4.程序中断方式接口电路的基本组成



5.5.4 I/O中断处理过程

1.CPU中断的条件和时间(5.5.3.4结合上图)

条件：允许中断触发器EINT=1，(EINT触发器可以用开中断指令置为1，可用关中断指令置为0或硬件自动复位)

时间：在每条指令执行阶段的结束前(当D=1(随机)且MASK=0时)

2.I/O中断处理过程(以输入设备为例)

（1）CPU发启动设备命令，B置1，D置0

（2）接口启动输入设备

（3）输入设备将数据送入数据缓存器

（4）设备向接口发送工作结束信号，D置1，B置0，标志设备准备就绪

（5）若D为1，MASK为0时在指令执行阶段的结束时由CPU发出中断查询信号

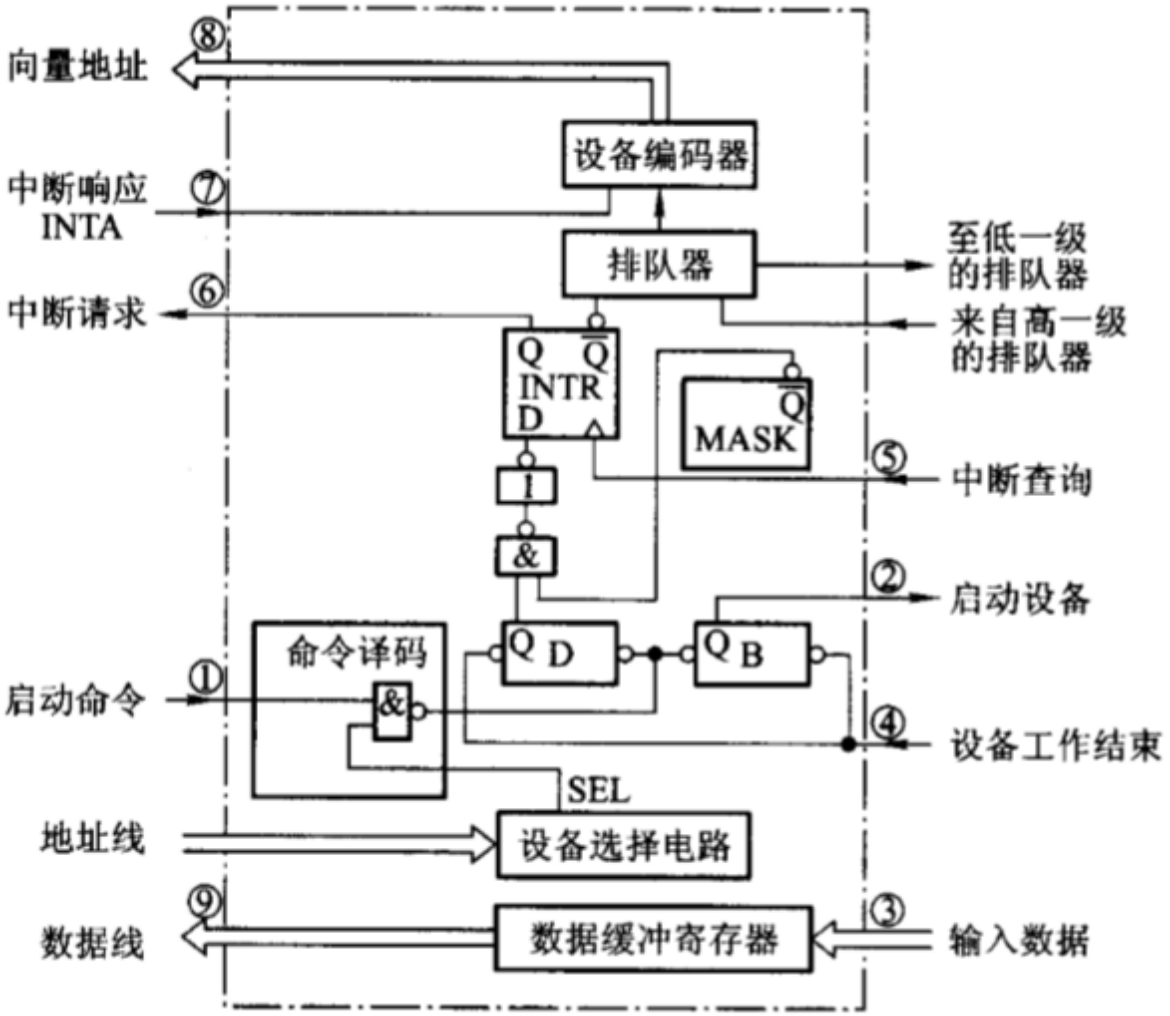
（6）INTR置1，标志设备向CPU提出中断请求，INTR送至排队器进行中断判优

（7）若CPU允许中断(EINT为1)则进入中断响应程序，形成向量地址

（8）向量地址送至PC作为下一条指令的地址

（9）这条指令结束后开始执行中断服务程序，进入中断服务阶段

(10)中断服务程序结束后返回至原程序断点处



5.5.5 中断服务程序的流程

1.保护现场

（1）程序断点的保护由中断隐指令完成

（2）寄存器内容的保护由中断服务程序(进栈指令)完成

2.中断服务(设备服务)

对不同的 I/O 设备具有不同内容的设备服务

3.恢复现场

出栈指令

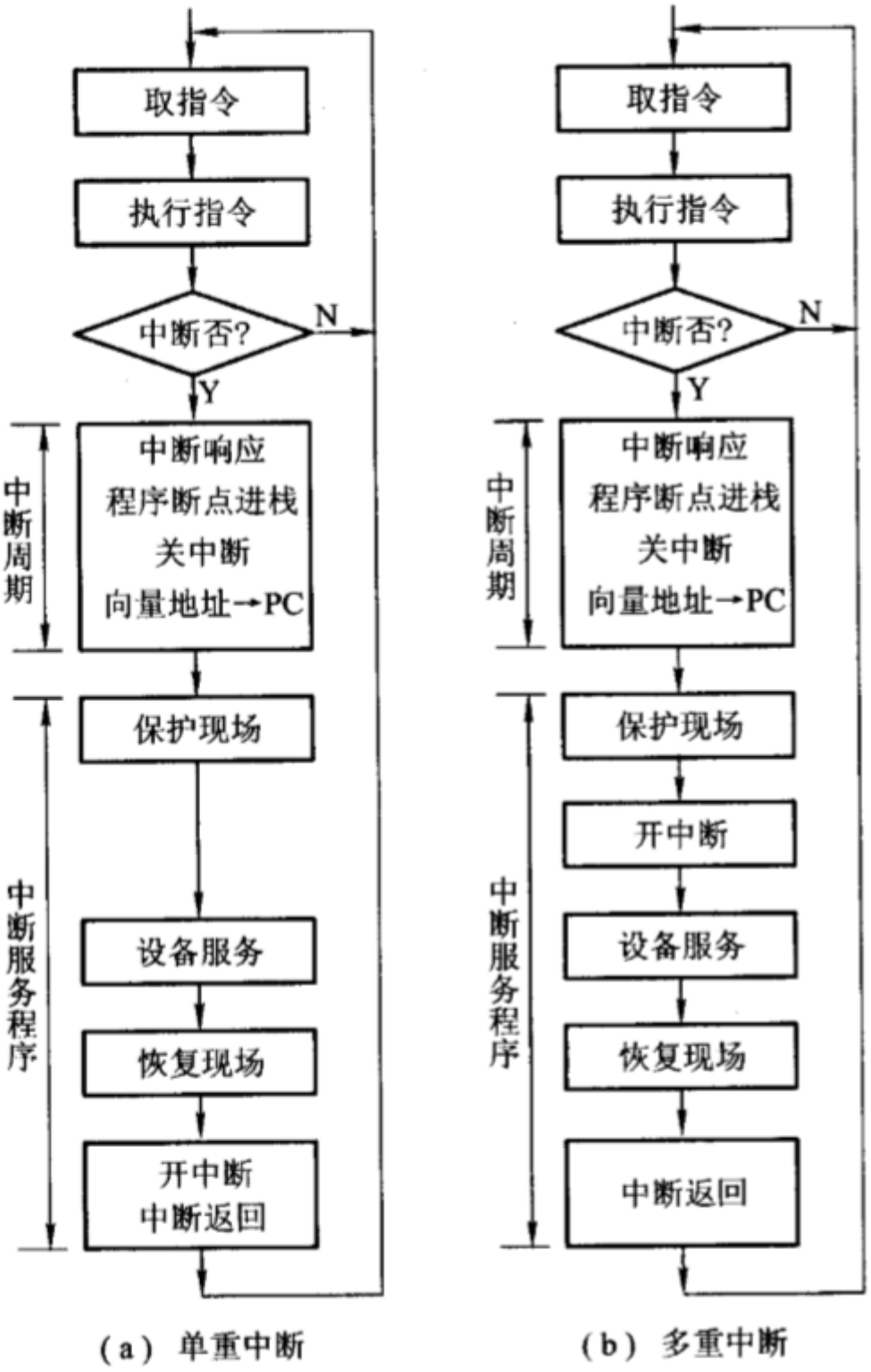
4.中断返回

中断返回指令

单重中断：允许中断现行的中断服务程序

多重中断：允许级别更高的中断源中断现行的中断服务程序

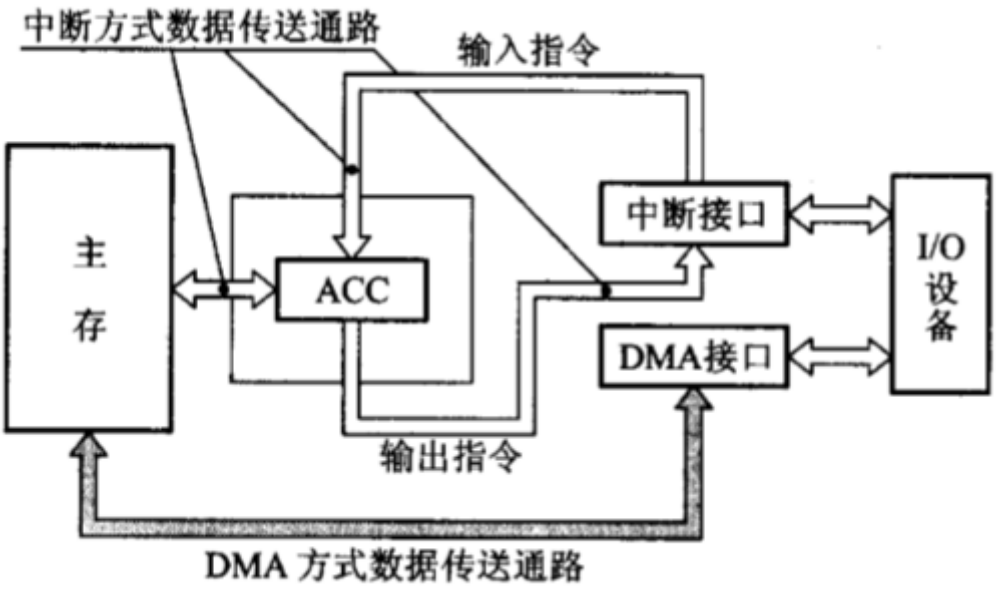
两种处理方式的中断服务程序略有区别在于开中断的设置时间不同



5.6 DMA方式(重点)

5.6.1 DMA方式的特点

1.DMA和程序中断两种方式的数据通路



2.DMA与主存交换数据的三种方式

（1）停止CPU访问主存

控制简单，CPU 处于不工作状态或保持状态，未充分发挥 CPU 对主存的利用率

（2）周期挪用(周期窃取)

既实现了I/O传送又较好发挥了主存和CPU的效率，较为常用(常用于I/O读写周期大于主存周期时)

CPU此时不访存

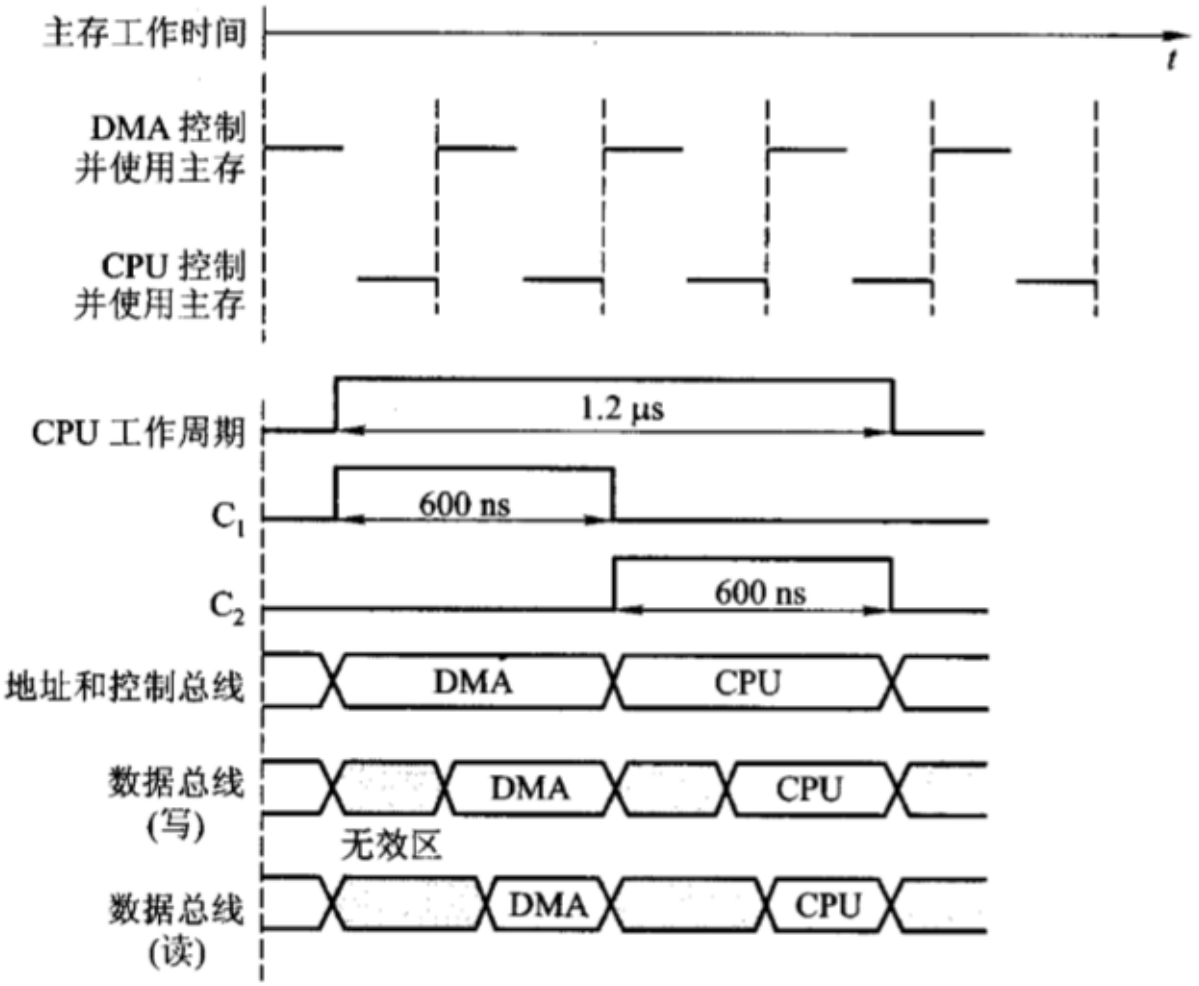
CPU正在访存：等CPU这次访问结束

CPU与DMA同时请求访存：I/O设备优先

（3）DMA与CPU交替访问

条件：CPU工作周期比主存存取周期长

不需要总线使用权的申请，建立，归还过程，具有很高的DMA效率，但逻辑更复杂



(c) DMA与CPU交替访问

5.6.2 DMA接口的功能和组成

1.DMA接口的功能

（1）向CPU申请DMA传送

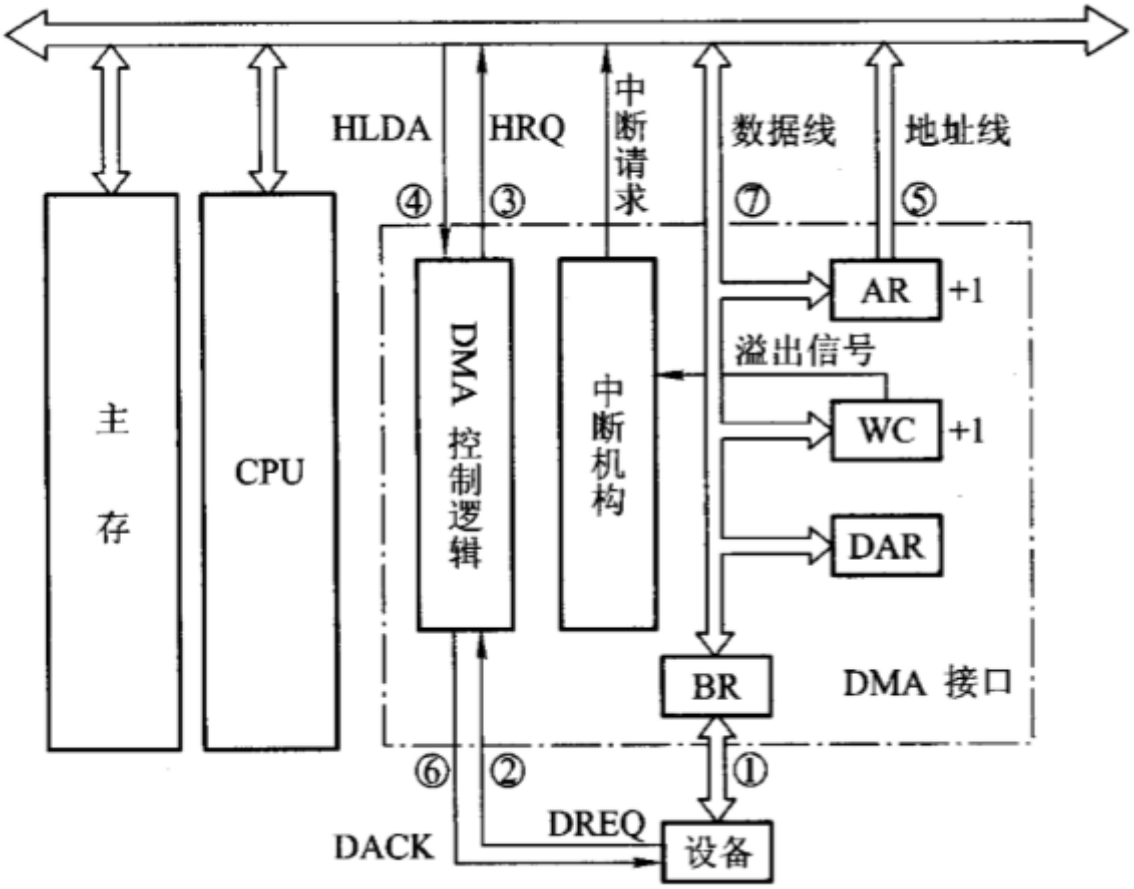
（2）处理总线控制权的转交

（3）管理系统总线、控制数据传送

（4）确定数据传送的首地址和长度，修正传送过程中的数据地址和数据长度

（5）DMA传送结束时，给出操作完成信号

2.DMA接口的基本组成



（1）主存地址寄存器(AR)：存放主存中需要交换数据的地址

（2）字计数器(WC)：记录传送数据的总字数，通常以交换字数的补码值预置

（3）数据缓冲寄存器(BR)

（4）DMA控制逻辑

（5）中断机构：向CPU提出中断请求

（6）设备地址寄存器(DAR)

5.6.3 DMA的工作过程

1.DMA传送过程

（1）预处理

DMA接口开始工作前需要对它进行预置

通知 DMA 控制逻辑传送方向(入/出)

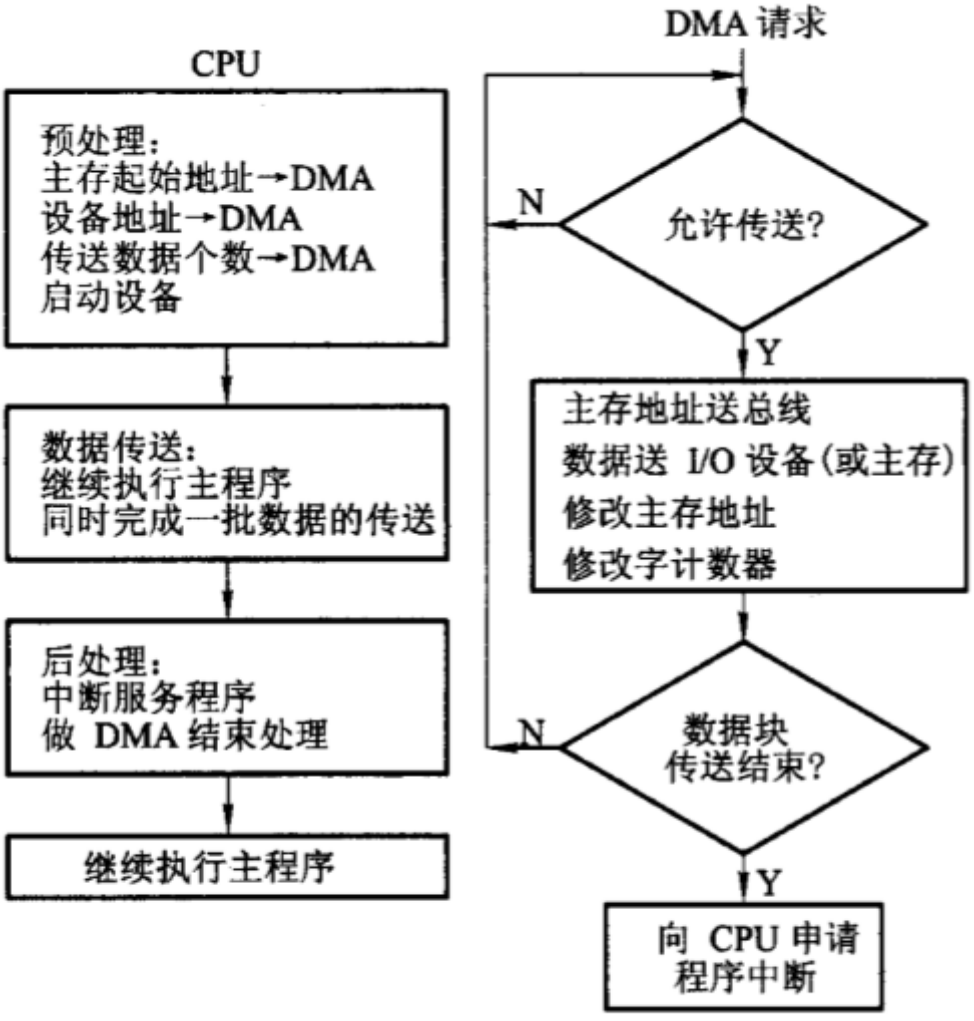
设备地址(DMA的DAR)，并启动设备

主存地址(DMA的AR)

传送字数(DMA的WC)

（2）数据传送

DMA是以数据块为单位传送的



（a） DMA 传送过程　　（b） 数据传送阶段的细化

（3）后处理

校验送入主存的数是否正确

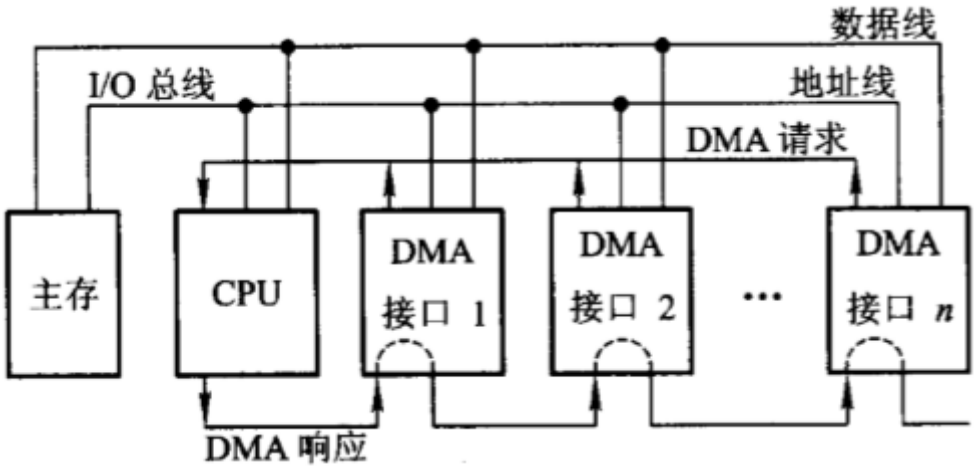
是否继续用DMA

测试传送过程是否正确，错则转诊断程序

2.DMA与系统的连接方式

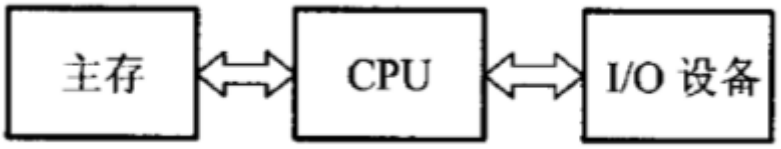
（1）具有公共请求线的DMA请求

具有公共请求线，CPU发出响应信号用链式查询方式通过DMA接口，首先选中的设备就获得总线控制权



（2）独立的DMA请求

每个DMA接口各有一对独立的DMA请求线和响应线，由CPU优先判别机构裁决先后顺序



3.DMA方式与程序中断方式的比较

	中断方式	DMA 方式
(1) 数据传送	程序	硬件
(2) 响应时间	指令执行结束	存取周期结束
(3) 处理异常情况	能	不能
(4) 中断请求	传送数据	后处理
(5) 优先级	低	高

5.6.4 DMA的接口类型

1.选择型DMA接口

特点：在物理上连接多个设备，在逻辑上只允许连接一个设备

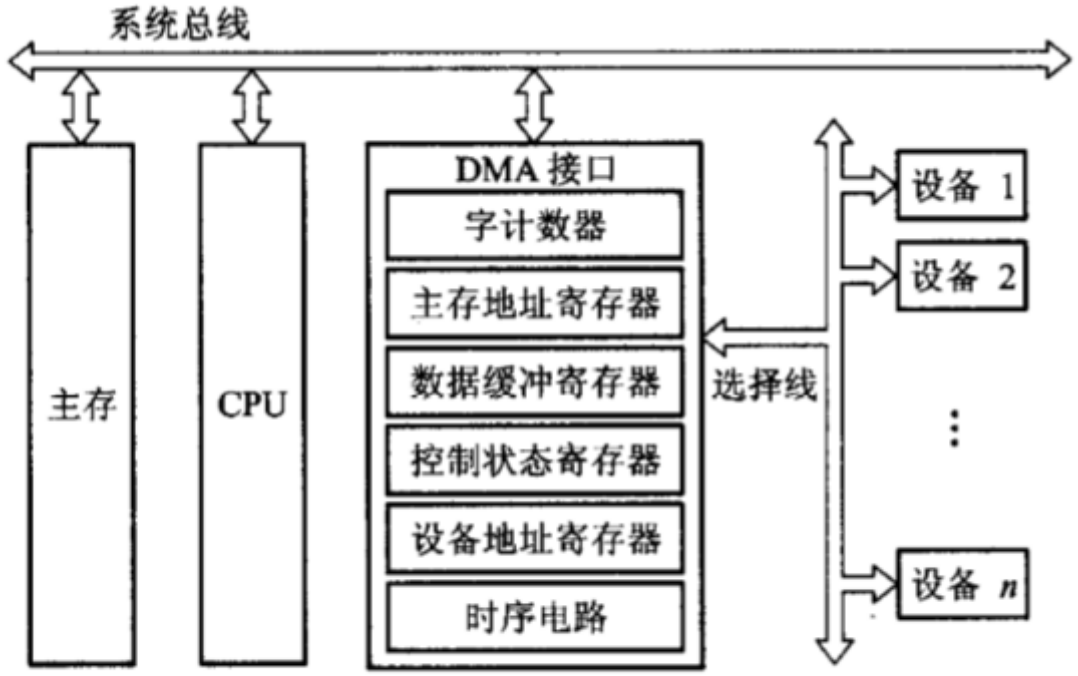
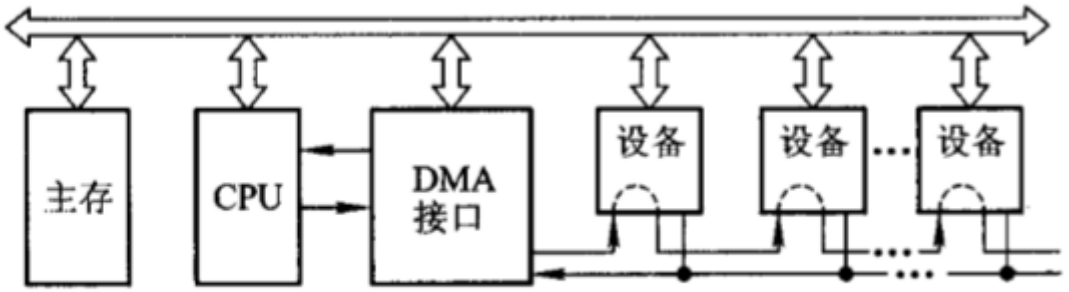


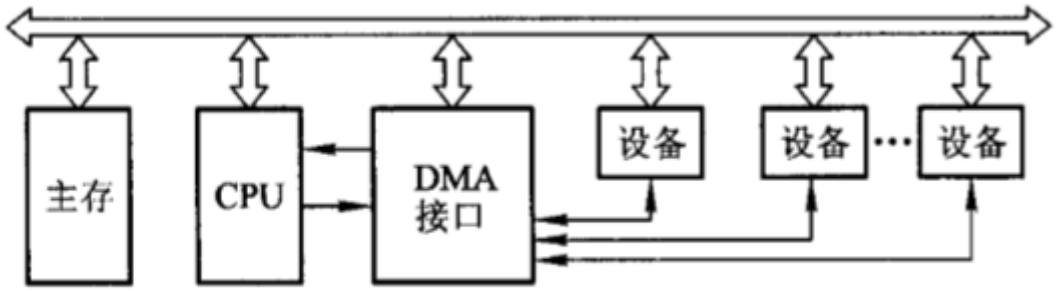
图 5.50　选择型 DMA 接口的逻辑框图

2.多路型DMA接口

特点：在物理上连接多个设备，在逻辑上允许连接多个设备

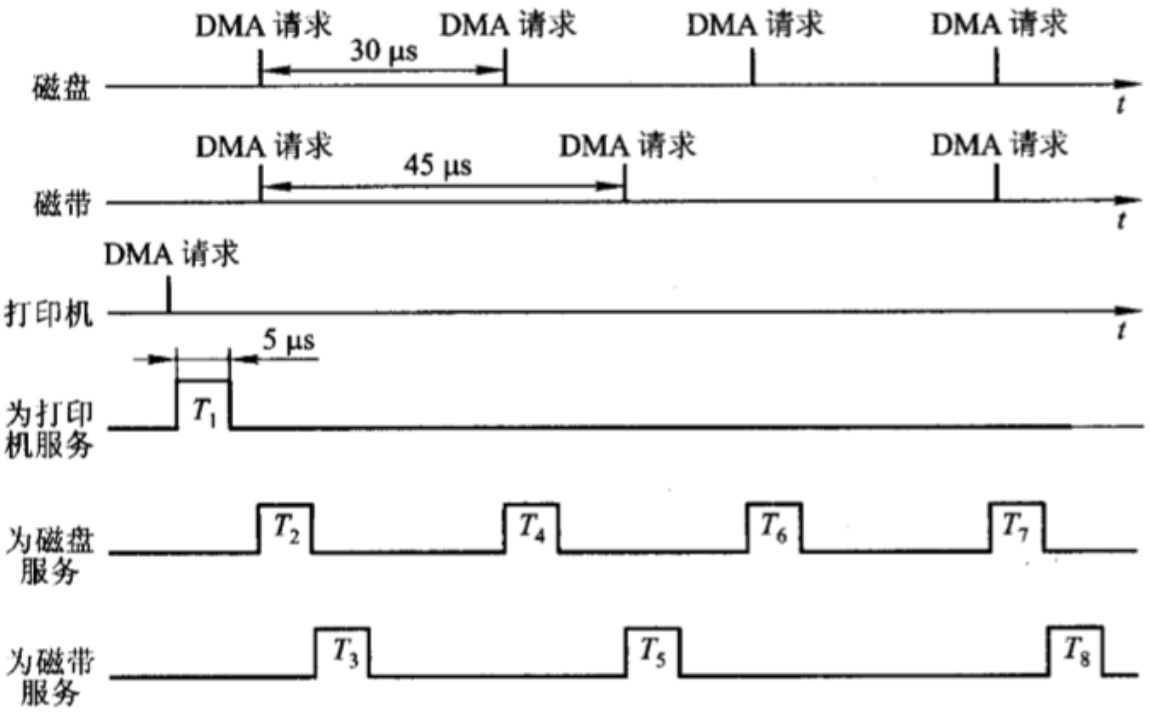


（a） 链式多路型DMA接口



（b） 独立请求多路型DMA接口

多路型工作原理



第6章 计算机的运算方法

本章概述

符号数，原补码以及加减法运算

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**。

6.1.1 无符号数

无符号数，指的是没有符号的数，也就是说寄存器中的每一位都可以用来存放数值。

因为没有符号，所以 8 位寄存器可以存放$2^8 = 256$范围大小的数字，即 0 ~ 255。同理，16 位寄存器可以存放$2^{16} = 65536$范围大小的数字，即 0 ~ 65535。

6.1.2 有符号数

有符号数与无符号数恰好相反，在存放有符号数的时候，要在寄存器中留出对应的位来存放正负符号。

6.1.2.1 机器数与真值

计算机通常用 “0” 表示 “正”，用 “1” 表示 “负”，从而将符号数字化并保存在寄存器中。

我们将符号放在 **有效数字** 的前面，就组成了 **有符号数**。

例如： `+1100` 在机器中就表示为 `01100` ，而 `-1100` 在机器中就表示为 `11100` 。

6.1.2.2 原码表示法

对于整数:

$$[x]_{原} = \begin{cases} 0,x & 0 \leq x < 2^n \\ 2^n - x & -2^n < x \leq 0 \end{cases}$$

其中，x 为真值，n 为整数的位数。注意，在上述公式中，逗号是用来隔开符号位和数值位的形式。

比如：当真值$x = +1110$时，$[x]_{原} = 0,1110$，当$x = -1110$时，$[x]_{原} = 2^4 - (-1110) = 1,1110$。

对于小数:

$$[x]_{原} = \begin{cases} x & 0 \leq x < 1 \\ 1 - x & -1 < x \leq 0 \end{cases}$$

其中，x 为真值，n 为小数的位数。

比如：当真值$x = 0.1101$时，$[x]_{原} = 0.1101$，当$x = -0.1101$时，$[x]_{原} = 1 - (-0.1101) = 1.1101$。

原码的特点：简单、直观，但是在计算机中实现其加减法不方便。

6.1.2.3 补码表示法

(1) 补的概念

补这个概念在时钟里面非常常见。例如在时钟里面想要从 6 点拨到 9 点，那么既可以顺时针拨 3 小时（+3），也可以逆时针拨 15 小时（-15），那么就可以得到：“+9 是 -3 以12为模的补数”，记作

$$-3 \equiv +9 \qquad (mod 12)$$

也可以称作“对模 12 而言，-3 和 -9 是互为补数的”。

注意以下要点：

- 一个负数可用它的正补数来替代，而这个正补数可以用模加上负数本身求得。
- 一个正数和一个负数互为补数时，他们绝对值之和即为模数。
- 正数的补数即该正数本身。

(2) 补码的定义

对于整数：

$$[x]_{补} = \begin{cases} 0, x & 0 \le x < 2^n \\ 2^{n+1} + x & -2^n < x \le 0 \end{cases} \quad (mod\ 2^{n+1})$$

其中，x 为真值，n 为整数的位数。

比如：当真值$x = +1010$时，$[x]_{补} = 0, 1010$，当$x = -1101$时，$[x]_{补} = 2^{4+1} - 1101 = 10000 - 1101 = 1, 0011$。

对于小数：

$$[x]_{补} = \begin{cases} x & 0 \le x < 1 \\ 2 + x & -1 < x \le 0 \end{cases} \quad (mod\ 2)$$

其中，x 为真值，n 为小数的位数。

比如：当真值$x = 0.1001$时，$[x]_{补} = 0.1001$，当$x = -0.0110$时，$[x]_{补} = 2_{10} + 0.1001_2 = 10.0000_2 - 0.0110_2 = 1.1010$。

证明可得$[+0]_{补} = [-0]_{补} = 0.0000$，即 “零” 的表示形式只有一种。

符号位的长度与模数有关。例如当模数为 4 时，又形成了双符号位的补码，如$x = -0.1001$，对$(mod\ 2^2)$而言，$[x]_{补} = 2^2 + x = 100.0000 - 0.1001 = 11.0111$

> **小技巧1**：当真值为负时，补码可用 **原码除符号位外，每位取反**，末位加 1 求得。

> **小技巧2**：还有一种简单的计算方案：
> 假设有一个数的原码是 `1,00010101010111010111110000`，想算补码，那么我们首先从右往左一个一个找，找什么呢？找第一次出现的 “1”。在本例子中，“1” 出现在 **从右往左数** 第 5 位。
> 现在不要眨眼，我们从右侧第 6 位开始，把后面 **除了符号位以外** 的东西全部 **按位取反**，得到 `1,11101010101000010100010000`，这就是我们要的补码！！！！！！
> 验证一下，看看真的是这样吗？小数也是这样吗？请读者自行验证。

6.1.2.4 反码表示法

对于整数：

$$[x]_{反} = \begin{cases} 0, x & 0 \le x < 2^n \\ (2^{n+1} - 1) + x & -2^n < x \le 0 \quad (mod\ (2^{n+1} - 1)) \end{cases}$$

其中，x 为真值，n 为整数的位数。

比如：当真值$x = +1101$时，$[x]_{反} = 0,1101$，当$x = -1101$时，$[x]_{反} = 2^{4+1} - 1 + 1101 = 1,0010$。

对于小数：

$$[x]_{反} = \begin{cases} x & 0 \le x < 1 \\ (2 - 2^{-n}) + x & -1 < x \le 0 \end{cases}$$

其中，x 为真值，n 为小数的位数。

比如：当真值$x = 0.0110$时，$[x]_{反} = 0.0110$，当$x = -0.0110$时，$[x]_{反} = (2 - 2^{-4}) - 0.0110 = 1.1001$。

综上所述，三种机器数的特点：

- 三种机器数的最高位均为符号位。符号位和数值部分之间可用“.”（对于小数）或“,”（对于整数）隔开。
- 当真值为正时，原码、补码和反码的表示形式均相同，即符号位用“0”表示，而数值部分与真值相同。

- 当真值为负时，原码、补码和反码的表示形式不同，但其符号位都用“1”表示，而数值部分有这样的关系，即补码是原码的“求反加1”，反码是原码的“每位求反”。

6.1.2.5 移码表示法

由于补码带有符号位很难直接判断真值大小，因此需要想办法把数字处理一下，这样计算机进行大小比较的时候就可以把符号位也当作一个正常的数字位进行比较了。

移码的定义为：

$$[x]_{移} = 2^n + x \quad (-2^n \leq x < 2^n)$$

，其中，x为真值，n为整数的位数。

比如：当真值$x = 10100$时，$[x]_{移} = 1,10100$，当$x = -10100$时，$[x]_{移} = 2^5 - 10100 = 0,01100$。

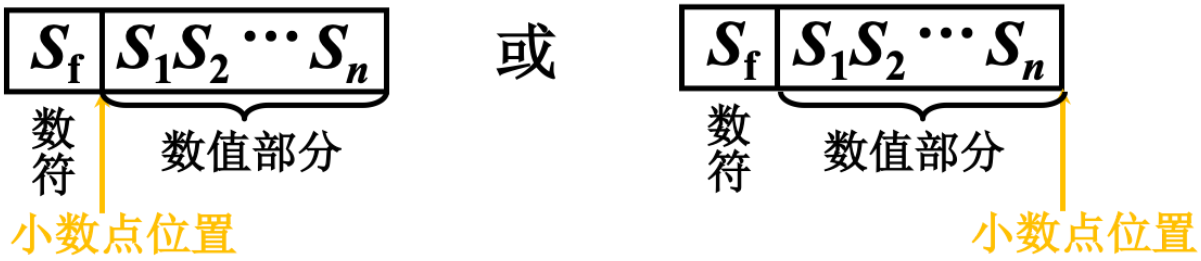
并且，$[+0]_{移} = [-0]_{移} = 1,00000$，是唯一的。

另外，同一个真值的移码和补码仅差一个符号位，因此只需要将补码的符号位由“0”修改为“1”，或者从“1”修改为“0”，即可得到移码。

6.2 数的定点表示和浮点表示

6.2.1 定点表示

小数点固定在某一位置的数称为定点数，有以下两种表示方式：



当小数点位于数符和第一数值位之间时（如上如左），机器内的数为纯小数；当小数点位于数值位之后时（如上如右），机器内的数为纯整数。

另外，采用定点数的机器称为定点机。数值部分的位数 n 决定了定点机中数的表示范围。在定点机中，由于小数点的位置固定不变，因此当机器处理的数不是纯小数或纯整数时，必须乘一个比例因子，否则会造成“溢出”。

6.2.2 浮点表示

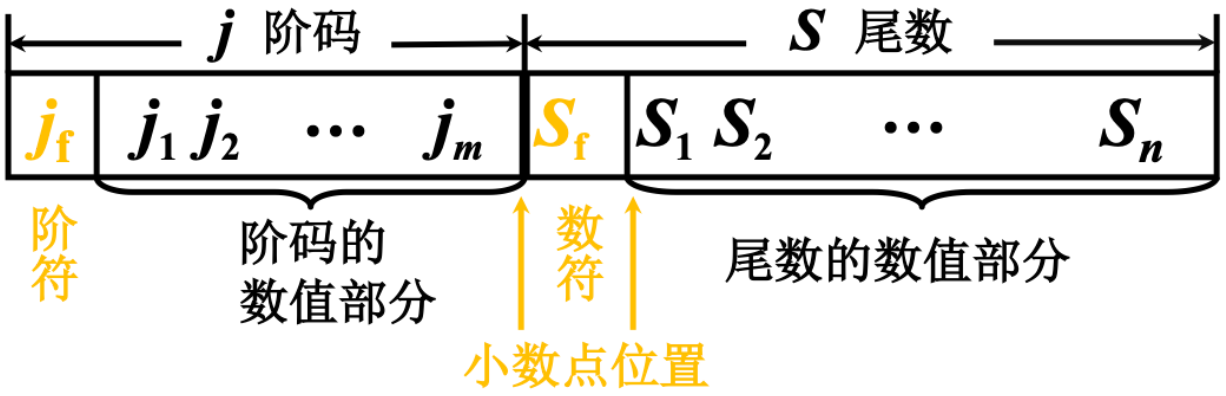
浮点数的通常表示方式：

$$N = S \times r^j$$

其中，S 为尾数（小数，可正可负），j 为阶码（整数，可正可负），r 是基数（或者基值）。在计算机中，r 常取 2，4，8，16 等。

6.2.2.1 浮点数的表示形式

浮点数在机器中的表示形式如下图所示。采用这种数据格式的机器称为浮点机。



其中，记阶码数值部分的位数为 m，尾数数值部分的位数为 n。则在浮点数中：

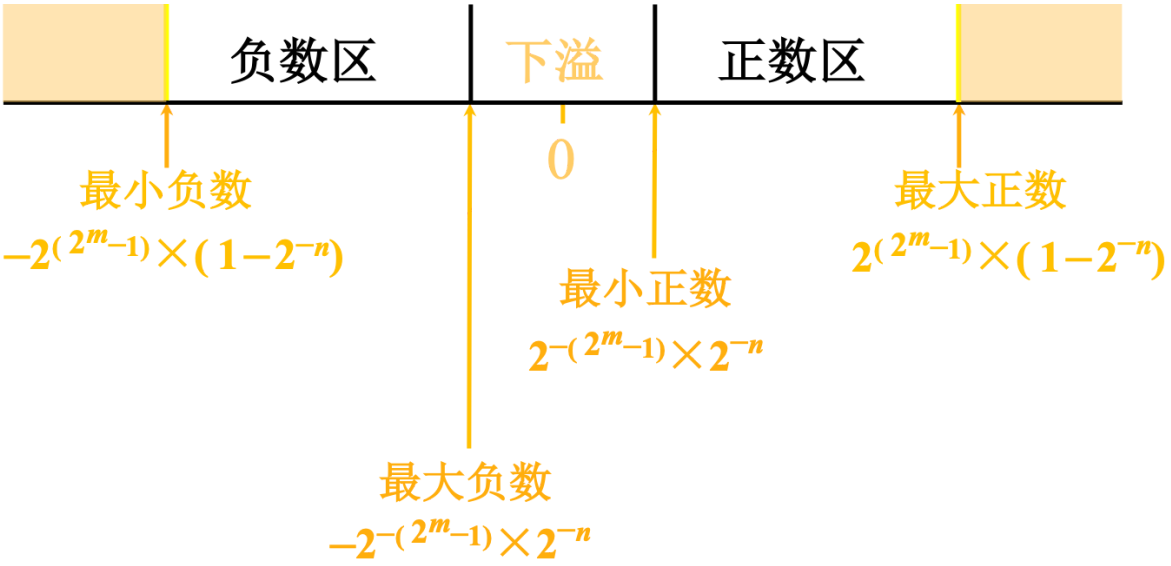
- m 反映了浮点数的表示范围以及小数点的实际位置
- n 反映了浮点数的精度
- S_f代表浮点数的符号

整个浮点数可以被看作是两个数的结合，前一个阶码是整数，后一个尾数是小数。

6.2.2.2 浮点数的表示范围

以浮点数通式$N = S \times r^j$为例，设浮点数阶码的数值位取 m 位，尾数的数值位取 n 位，当浮点数为非规格化数*时，它在数轴上的表示范围如下图所示：

> *对于某些过小的数，如 `1.23e-130`，允许的阶数位数不能满足阶数大小的需要，这时可能就会在尾数前添加前导“0”，如将其表示为 `0.000123e-126`，这样的数就是非规格化数。



阶码和尾数的分配将会直接影响浮点数的表示范围和精度。通常来讲：

- 对于短实数（总位数32位）：阶码 8 位（含阶符 1 位），尾数 24 位（含数符 1 位）
- 对于长实数（总位数64位）：阶码 11 位（含阶符 1 位），尾数 53 位（含数符 1 位）
- 对于临时实数（总位数80位）：阶码 15 位（含阶符 1 位），尾数 65 位（含数符 1 位）

6.2.2.3 浮点数的规格化形式

尾数变为规格化数能够提高浮点数的精度（能够减少计算过程中出现 0 或者 NaN 的可能性）。

通过修改阶码并左右移位尾数的方法，可以使非规格化数变为规格化数，这个过程称为 **规格化**。规格化数的 **形式** 和规格化的 **过程** 对于 **基数不同** 的浮点数是不同的。例如在浮点数通式$N = S \times r^j$中：

- $r = 2$时，尾数最高位为 1 的数为规格化数。规格化时，尾数左移 1 位，阶码减 1（该过程称为向左规格化，简称 **左规**）；尾数右移 1 位，阶码加 1（该过程称为向右规格化，简称 **右规**）。
- $r = 4$时，尾数最高两位不全为 0 的数为规格化数。规格化时，尾数左移 2 位，阶码减 1；尾数右移 2 位，阶码加 1。
- $r = 8$ 时，尾数最高三位不全为 0 的数为规格化数。规格化时，尾数左移 3 位，阶码减 1；尾数右移 3 位，阶码加 1。

r 更大的规格化同理。
图为实现算数左移右移的硬件框图：

在浮点机中，一旦基数确定后就不再变了。由于基数并不体现在浮点数的存储中（基数是隐含的），所以不同基数的浮点数表示完全相同。

一般来说，基数 r 越大：

- 可表示的浮点数范围越大，所表示的数个数越多
- 浮点数的精度越低

6.2.3 定点数和浮点数的比较

- 当浮点机和定点机中数的位数相同时，浮点数的表示范围比定点数的大得多。
- 当浮点数为规格化数时，其相对精度远比定点数高。
- 浮点数运算要分阶码部分和尾数部分，而且运算结果都要求规格化，故浮点运算步骤比定点运算步骤多，运算速度比定点运算的低，运算线路比定点运算的复杂。
- 在溢出的判断方法上，浮点数是对规格化数的阶吗进行判断，而定点数是对数值本身进行判断。

总之，浮点数在数的表示范围、数的精度、溢出处理和程序编程方面（不取比例因子）均优于定点数。但在运算规则、运算速度及硬件成本方面又不如定点数。因此，究竟选用定点数还是浮点数应根据具体应用综合考虑。

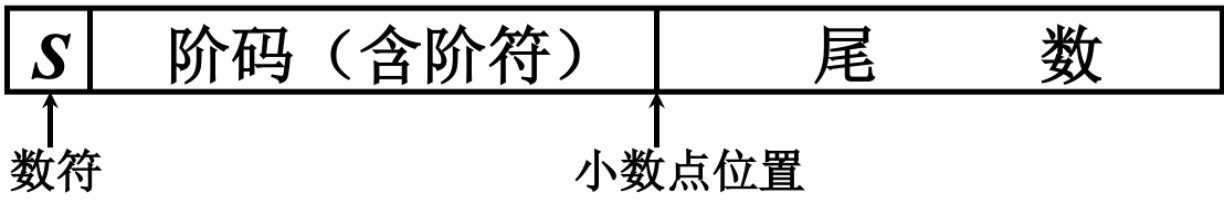
一般来说，通用的大型计算机大多采用浮点数，或同时采用定、浮点数；小型、微型及某些专用机、控制机则大多采用定点数。

6.2.4 补充：机器零

- 当浮点数 **尾数为 0** 时，不论其阶码为何值，均按机器零处理
- 当浮点数 **阶码等于或小于它所表示的最小** 时，不论尾数为何值，按机器零处理

> 例如当 m = 4，n = 10，阶码和尾数都用补码表示时，机器零为
>
> - 尾数为 0： `X,XXXX;0.0000000000`
> - 阶码等于或小于它所表示的最小： `1,0000;X,XXXXXXXXXX` （阶码 = -16 时）
>   当阶码用移码，尾数用补码表示时，机器零为 `0,0000;0.0000000000`，这有利于机器中“判零”电路的实现。

6.2.5 IEEE 754 标准



尾数为规格化表示，非 “0” 的有效位最高位为 “1”（隐含）。

阶码和尾数的分配：

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

6.3 定点运算

6.3.1 移位运算

6.3.1.1 移位的意义

例如在生活中，15m 可以写成 1500cm，从数字而言，1500 相当于 15 相对于小数点左移了两位（同时补充了“0”）；15 相当于 1500 向右移了两位（同时删掉了多余的“0”）。

从本质上来说，左移的过程使得数的绝对值扩大，右移使得绝对值缩小。

对于二进制计算机来说，左移或右移 n 位相当于乘以或者除以$2^n\ (n=1,2,...,n)$。

另外，由于移位后会出现空缺，因此补充空缺到底是用 0 还是 1，与机器采用有符号数还是无符号数有关。

6.3.1.2 算数移位

有符号数的移位称为算数移位。

算数移位的规则如下：

正负	码制	添补代码
正数	原码、反码、补码	0
负数	原码	0

正负	码制	添补代码
	补码	左移添 0
		右移添 1
	反码	1

注意算数移位的重要特点：**不论是正数还是负数，移位后其符号位均不变。**

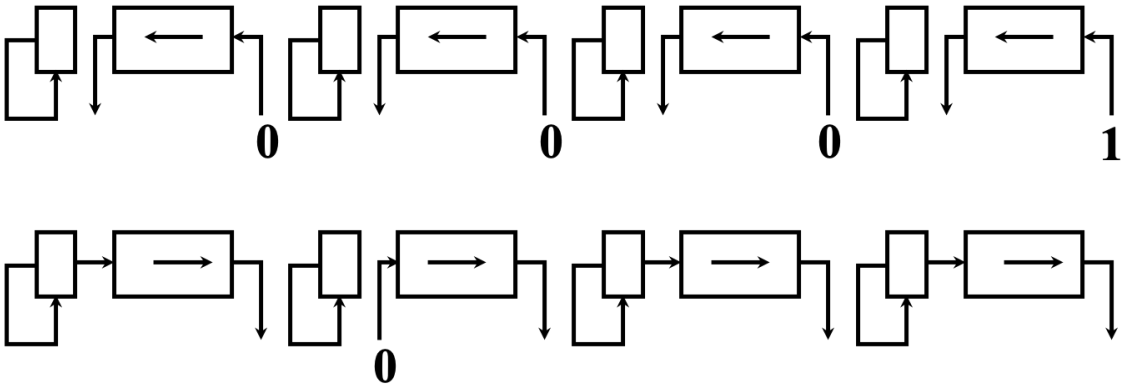
另外，从表格我们可以得出的结论：

- 机器数为正时，不论是左移还是右移，添补代码均为 0。
- 由于负数的原码数值部分与真值相同，所以在移位时只需要使符号位不变，其余空位均添 0 即可。
- 由于负数的反码各位除符号位以外与负数的原码正好相反，所以移位后所添的代码应与原码相反，即全部添 1。
- 分析任意数的补码发现，当对其由低位向高位找到第一个“1”时，在此“1”左边的各位与对应的反码相同，而在此“1”右边的各位（包括“1”本身）与对应的原码相同。所以负数的补码左移时，因空位出现在低位，则添补的代码与原码相同，即添 0；右移时因空位出现在高位，则添补的代码与反码相同，即添 1。

关于位移以后的结果：

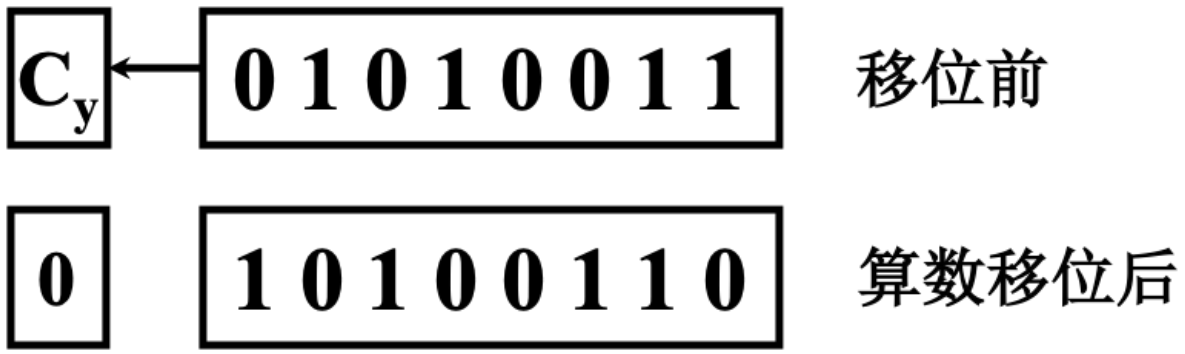
- 对于正数，三种机器数移位后符号位均不变，左移时最高数位丢 1，结果出错；右移时最低数位丢 1，影响精度。
- 对于负数，三种机器数移位后符号位均不变;
    - 负数的原码左移时最高数位丢 1，结果出错；右移时最低数位丢 1，影响精度。
    - 负数的补码左移时最高数位丢 0，结果出错；右移时最低数位丢 1，影响精度。
    - 负数的反码左移时最高数位丢 0，结果出错；右移时最低数位丢 0，影响精度。

下图为实现算数左移右移的硬件框图：



	（a）真值为正	（b）负数的原码	（c）负数的补码	（d）负数的反码
←丢 1	出错	出错	正确	正确
→丢 1	影响精度	影响精度	影响精度	正确

另外，为了避免算数移位时丢失数据，可以采用带进位（C_y）的移位，示意图如下（下图是左移）：



6.3.1.3 逻辑移位

无符号数的移位称为逻辑移位。

规则：逻辑左移时，高位移丢，低位添 0；逻辑右移时，低位移丢，高位添 0。

6.3.2 加减法运算

加减法运算是计算机中最基本的运算。由于减法运算可以通过加负值的方式代替，所以加减法被放在一起进行讨论。

6.3.2.1 补码加减运算的基本公式

对于整数：

$$[A]_{补}+[B]_{补}=[A+B]_{补}(mod\ 2^{n+1})$$

$$[A-B]_{补}=[A]_{补}+[-B]_{补}(mod\ 2^{n+1})$$

对于小数：

$$[A]_{补}+[B]_{补}=[A+B]_{补}(mod\ 2)$$

$$[A-B]_{补}=[A]_{补}+[-B]_{补}(mod\ 2)$$

6.3.2.2 溢出判断

(1) 用一位符号判断溢出

不论作加法还是作减法，只要实际参加操作的两个数（减法时即为被减数和“求补”以后的减数）符号相同，结果又与原操作数的符号不同，即为溢出。

计算机中采用这种方法进行判断时，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，按其结果进行判断。若异或结果为 1，即为1溢出；异或结果为 0，则无溢出。

(2) 用两位符号位判断溢出

2 位符号位的补码，即 **变形补码**，是以 4 为模的，其定义为：

$$[x]_{补'}=\begin{cases}x & 0\le x<1\\4+x & -1\le x<0\end{cases}\quad(mod\ 4)$$

用变形补码作加法时，2 位符号位要连同数值部分一起参加运算，而且高位符号位产生的进位自动丢失，便可得正确结果，即

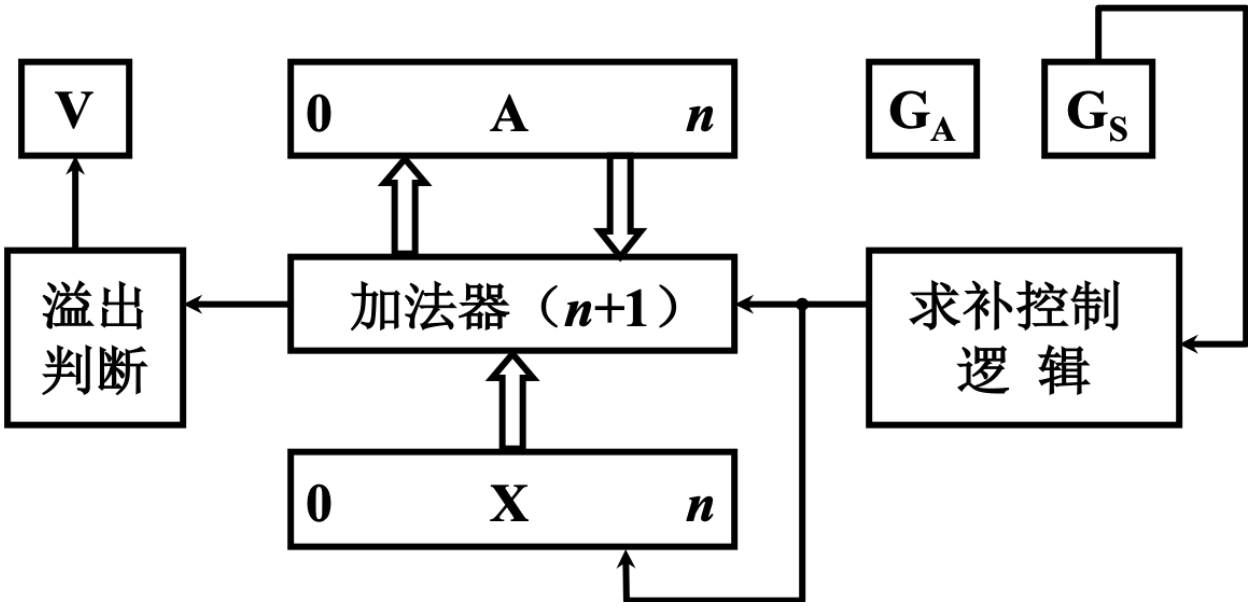
$$[x]_{补'}+[y]_{补'}=[x+y]_{补'}$$

变形补码判断溢出的原则是：当 2 位符号位不同时，表示溢出，否则，无溢出。

注意：无论是否发生溢出，高位（从左到右第 1 位）符号位永远代表真正的符号。

另外，采用双符号位方案时，寄存器或主存中的操作数只需要保留一位符号位即可。

6.3.2.3 补码定点加减法所需的硬件配置



图中，寄存器A、X、加法器的位数相等，其中A存放被加数（或被减数）的补码，X存放加数（或减数）的补码。

当作减法时，由“求补控制逻辑”将$\overline{X}$送至加法器，并使加法器的最末位来进位为 1，以达到对减数求补的目的。

运算结果溢出时，通过溢出判断电路置“1”溢出标记V。G_A为加法标记，G_S为减法标记。

6.3.3 乘法运算

6.3.3.1 笔算乘法改进

计算过程：

1. 乘法运算可用移位和假发来实现，两个 n 位数相乘，总共需要进行 n 次加法运算和 n 次加法移位。
2. 由乘数的末位值确定被乘数是否与原部分积想家，然后右移一位，形成新的部分积，同时，乘数也右移移位，由次低位作新的末位，空出最高位放部分积的最低位。
3. 每次作加法时，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。

这样做对计算机的好处：计算机很容易实现这种运算规则。一个寄存器放被乘数，一个寄存器存放乘积的高位，另一个寄存器放乘数及乘积的低位，再配上加法器和其他电路，即可构成乘法器。而加法只在部分积高位进行，所以不仅节省了器材，而且缩短运算时间。

6.3.3.2 原码乘法

由于原码与真值的表示只差一个符号，因此可以直接使用上述的讨论结果，再加以符号位处理即可。

(1) 原码一位乘运算规则

原码一位乘的运算规则如下：

1. 乘积的符号位由两位原码符号位异或运算结果决定。
2. 乘积的数值部分由两数绝对值相乘，其通式为：

$$
\begin{aligned}
x^* \cdot y^* &= x^*(0.y_1y_2\cdots y_n) \\
&= x^*(y_1 2^{-1} + y_2 2^{-2} + \cdots + y_n 2^{-n}) \\
&= 2^{-1}(y_1x^* + 2^{-1}(y_2x^* + 2^{-1}(\cdots + 2^{-1}(y_{n-1}x^* + \underbrace{2^{-1}(y_nx^* + \underbrace{0}_{z_0})}_{z_1})}_{z_2}\cdots)))
\end{aligned}
$$

$$
\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{z_n}
$$

$$
\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{z_{n-1}}
$$

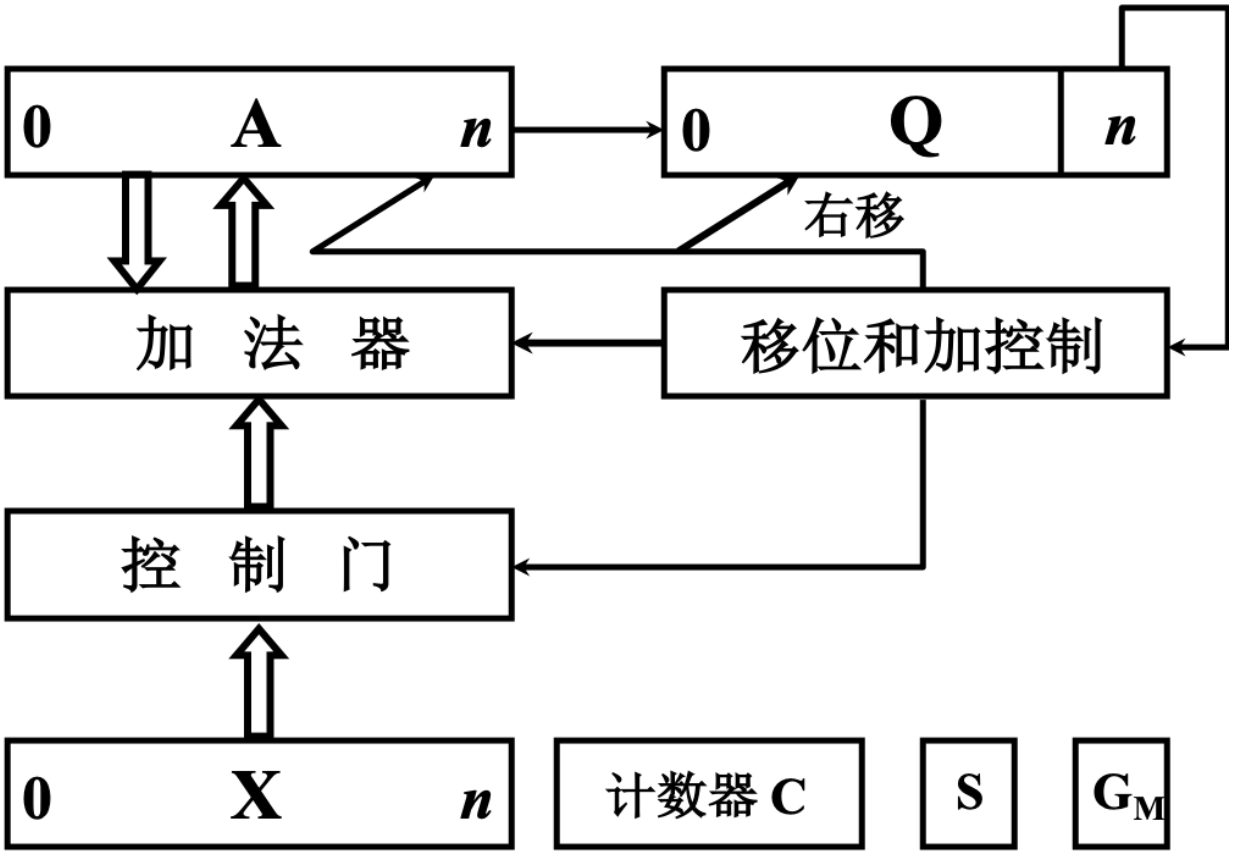
$$
\cdots
$$

，再令 z_i 表示第 i 次部分积，则上式可写成如下递推公式：

$$
\begin{aligned}
z_0 &= 0 \\
z_1 &= 2^{-1}(y_n \cdot x^* + z_0) \\
z_2 &= 2^{-1}(y_{n-1} \cdot x^* + z_1) \\
&\vdots \\
z_i &= 2^{-1}(y_{n-i+1} \cdot x^* + z_{i-1}) \\
&\vdots \\
z_n &= 2^{-1}(y_1 \cdot x^* + z_{n-1})
\end{aligned}
$$

注意，由于乘积的数值部分是 **两数绝对值** 相乘的结果，故原码一位乘法运算过程中的右移操作均为 **逻辑右移。**

(2) 原码一位乘所需的硬件配置



图中，A、X、Q 均为 $n+1$ 位的寄存器，其中 X 存放被乘数的原码，Q 存放乘数的原码。移位和加控制电路受末位乘数 Q_n 的控制（当 $Q_n=1$ 时，A 和 X 内容相加后，A、Q 右移一位；当 $Q_n=0$ 时，只作 A、Q 右移一位的操作）。计数器 C 用于控制逐位相乘的次数。S 存放乘积的符号。G_M 为乘法标记。

(3) 原码两位乘

原码两位乘与原码一位乘一样，符号位的运算和数值部分是分开进行的，但原码两位乘是用两位乘数来决定新的部分积如何形成，因此可以提高运算速度。

两位乘数的四种状态如下：

乘数$y_{n-1}y_n$	**新的部分积**
0 0	新部分积等于原部分积右移两位
0 1	新部分积等于原部分积加被乘数后右移两位
1 0	新部分积等于原部分积加 2 倍被乘数后右移两位
1 1	新部分积等于原部分积加 3 倍被乘数后右移两位

由于在计算机中获得 3 倍被乘数比较困难，因此将乘以 3 分两步完成，第一步先完成减 1 倍被乘数的操作，第二步完成加 4 倍被乘数的操作。而加 4 倍被乘数的操作实际上是由

比“11”高的两位乘数代替完成的，可以看作是在高两位乘数上加“1”，这个“1”可以暂存在 C_j 触发器中。机器完成 C_j 置“1”，即意味着对高两位乘数加 1，也即要求高两位乘数代替本两位乘数“11”来完成加 4 倍被乘数的操作。

由此可得原码两位乘运算规则：

乘数判断位 $y_{n-1}y_n$	标志位 C_j	操作内容
0 0	0	$z\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 保持“0”
0 1	0	$z+x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 保持“0”
1 0	0	$z+2x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 保持“0”
1 1	0	$z-x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_ 置“1”
0 0	1	$z+x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 置“0”
0 1	1	$z+2x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 置“0”
1 0	1	$z-x^*\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 保持“1”
1 1	1	$z\rightarrow$ 2 位，$y^*\rightarrow$ 2 位，C_j 保持“1”

表中，z表示原有部分积，x^*表示被乘数的绝对值，y^*表示乘数的绝对值，$\rightarrow$ 2 表示右移两位。当进行$-x^*$运算时，一般都采用加$[-x^*]_{补}$来实现。这样，参与原码两位乘运算的操作数是绝对值的补码，因此运算中右移两位的操作也必须按补码右移规则完成。尤其注意的是，乘法过程中可能要加 2 倍被乘数，即$+[2x^*]_{补}$，使部分积的绝对值大于 2。为此，只有对部分积取 3 位符号位，且以最高符号位作为真正的符号位，才能保证运算过程的正确无误。

此外，为了统一用两位乘数和一位C_j共同配合管理全部操作，与原码一位乘不同的是，需要在乘数（当乘数位数为偶数时）的最高位前增加两个 0。这样，当乘数最高两个有效位出现“11”时，需将C_j置“1”，再与所添补的两个 0 结合呈 001 状态，以完成加x^*的操作（此步不必移位）。

6.3.3.3 补码乘法

(1) 补码一位乘运算规则
设被乘数$[x]_{补}=x_0.x_1x_2\cdots x_n$
乘数$[y]_{补}=y_0.y_1y_2\cdots x_n$

1. 被乘数 x 符号任意，乘数 y 符号位正：
   $[x\cdot y]_{补}=[x]_{补}\cdot[y]_{补}=[x]_{补}\cdot y$对照原码一位乘的两个表达式，可见当乘数 y 为正数时，不管被乘数 x 符号如何1，都可以按照原码乘法的规则运算，即

$$
\begin{aligned}
[z_0]_{补} &= 0 \\
[z_1]_{补} &= 2^{-1}(y_n[x]_{补} + [z_0]_{补}) \\
[z_2]_{补} &= 2^{-1}(y_{n-1}[x]_{补} + [z_1]_{补}) \\
&\vdots \\
[z_i]_{补} &= 2^{-1}(y_{n-i+1}[x]_{补} + [z_{i-1}]_{补}) \\
&\vdots \\
[x \cdot y]_{补} = [z_n]_{补} &= 2^{-1}(y_1[x]_{补} + [z_{n-1}]_{补})
\end{aligned}
$$

2. 被乘数 x 符号任意，乘数 y 符号为负：

$$
[x \cdot y]_{补} = [x]_{补}(0.y_1 y_2 \cdots y_n) + [-x]_{补}
$$

由此可得，当乘数为负时是把乘数的补码$[y]_{补}$去掉符号位，当成一个正数与$[x]_{补}$相乘，然后加上$[-x]_{补}$进行校正，也称校正法，用递推公式表示如下：

$$
\begin{aligned}
[z_0]_{补} &= 0 \\
[z_1]_{补} &= 2^{-1}(y_n[x]_{补} + [z_0]_{补}) \\
[z_2]_{补} &= 2^{-1}(y_{n-1}[x]_{补} + [z_1]_{补}) \\
&\vdots \\
[z_i]_{补} &= 2^{-1}(y_{n-i+1}[x]_{补} + [z_{i-1}]_{补}) \\
&\vdots \\
[z_n]_{补} &= 2^{-1}(y_1[x]_{补} + [z_{n-1}]_{补}) \\
[x \cdot y]_{补} &= [z_n]_{补} + [-x]_{补}
\end{aligned}
$$

3. 被乘数 x 和乘数 y 符号均为任意

比较法是 Booth 夫妇首先提出来的，又叫 Booth 算法。

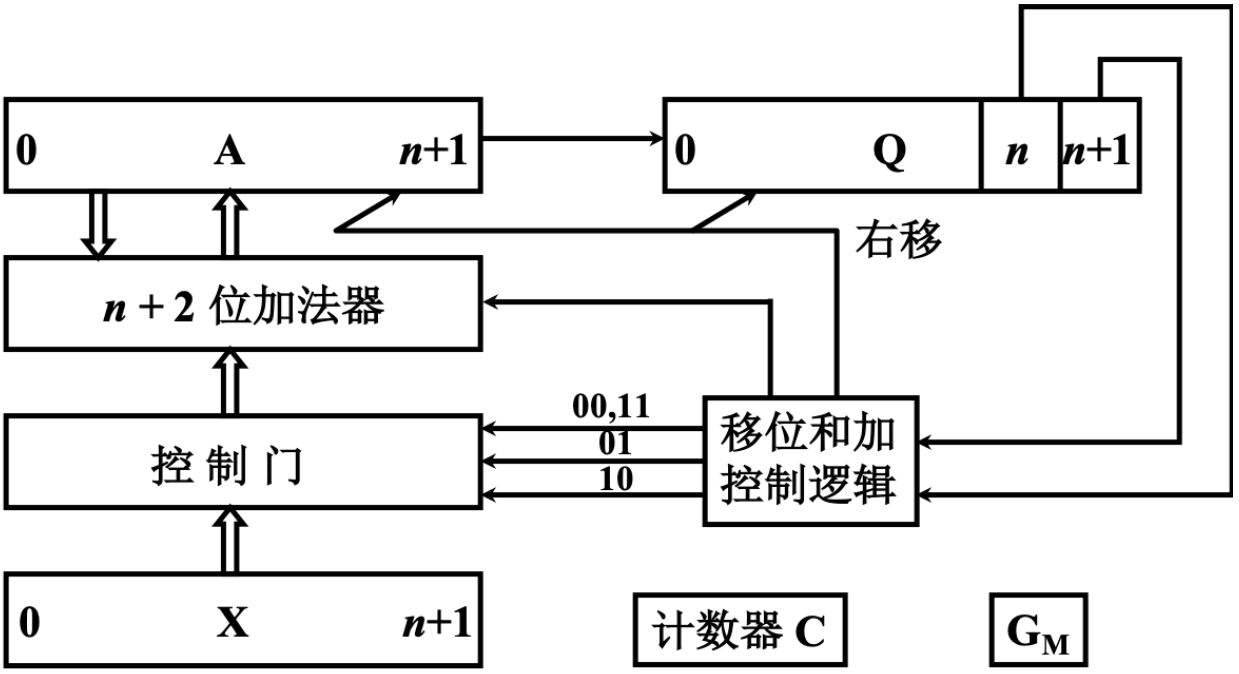
按照补码乘法校正法规则，其基本算法可用一个统一的公式表示为

$$
[x \cdot y]_{补} = [x]_{补}(0.y_1 y_2 \cdots y_n) - [x]_{补} \cdot y_0
$$

，其递推公式为

$$
\begin{aligned}
[z_0]_{补} &= 0 \\
[z_1]_{补} &= 2^{-1}\{[z_0]_{补} + (y_{n-1} - y_n)[x]_{补}\} \\
[z_2]_{补} &= 2^{-1}\{[z_1]_{补} + (y_n - y_{n-1})[x]_{补}\} \\
&\vdots \\
[z_i]_{补} &= 2^{-1}\{[z_{i-1}]_{补} + (y_{n-i+2} - y_{n-i+1})[x]_{补}\} \\
&\vdots \\
[z_n]_{补} &= 2^{-1}\{[z_{n-1}]_{补} + (y_2 - y_1)[x]_{补}\} \\
[x \cdot y]_{补} &= [z_{n+1}] = [z_n]_{补} + (y_1 - y_0)[x]_{补}
\end{aligned}
$$

(2) 补码比较法（Booth 算法）所需的硬件配置



图中A、X、Q均为$n+2$位寄存器，其中X存放被乘数的补码（含两位符号位），Q存放乘数的补码（含最高 1 位符号位和最末 1 位附加位），移位和加控制逻辑受Q寄存器末 2 位乘数控制。当其为 01 时，A、X内容相加后A、Q右移一位；当其为 10 时，A、X内容相减后A、Q右移一位。计数器C用于控制逐位相乘的次数，G_M为乘法标记。

6.3.4 除法运算

6.3.4.1 原码除法

原码除法和原码乘法一样，符号位是单独处理的。商符由两数符号位进行异或运算求得，商值由两数绝对值相除(x^*/y^*)求得。

(1) 恢复余数法

特点：当余数为负时，需加上除数，将其恢复成原来的余数。

由上所述，商值确定的是通过比较被除数和除数的绝对值大小，即$x^* - y^*$实现的，而计算机内只设加法器，故需将$x^* - y^*$操作变为$[x^*]_{补} + [-y^*]_{补}$的操作。

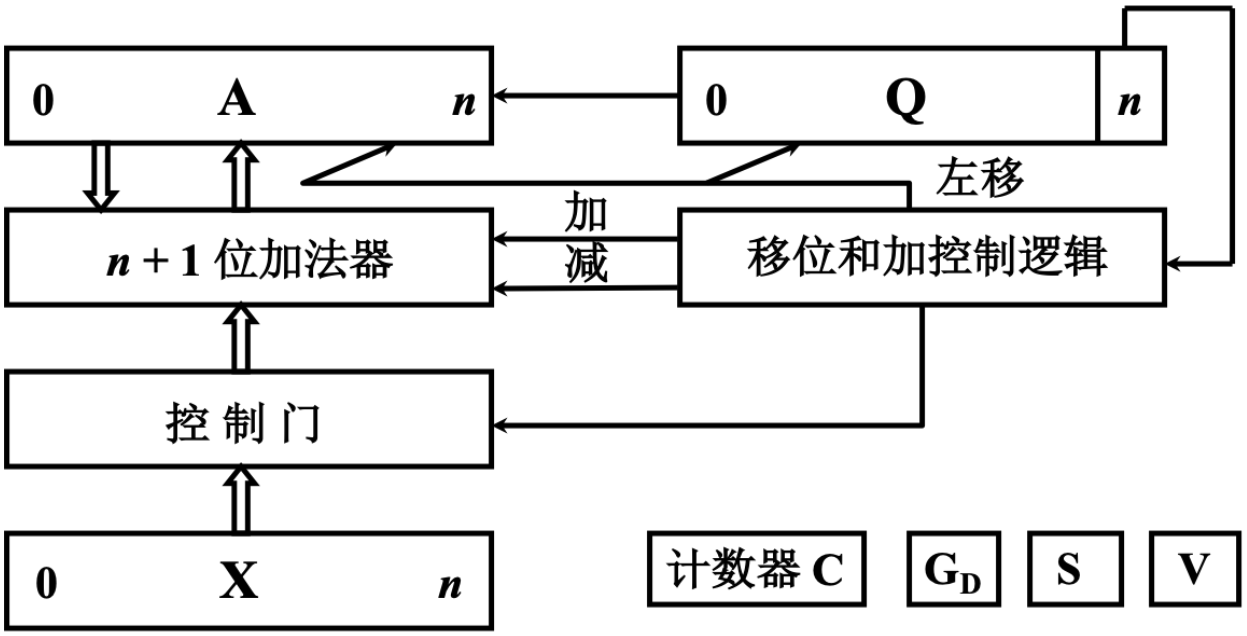
(2) 不恢复余数法（加减交替法）

加减交替法可以认为是恢复余数法的一种改进算法。

原码恢复余数法可以归纳为：

- 当$R_i > 0$，商上“1”，做$2R_i - y^*$的运算。
- 当$R_i < 0$，商上“0”，做$2R_i + y^*$的运算。

(3) 原码加减交替法所需的硬件配置



图中A、X、Q均为$n+1$位寄存器，其中A存放被除数的原码，X存放除数的原码。移位和加控制逻辑受Q的末位Q_n控制（$Q_n = 1$作减法，$Q_n = 0$作加法），计数器C用于控制逐位相除的次数n，G_D为除法标记，V为溢出标记，S为商符。

6.3.4.2 补码除法（加减交替法）

(1) 商值的确定

1. 比较被除数（余数）和除数绝对值的大小

- 当被除数与除数同号时，作减法，若得到的余数与除数同号，表示“够减”，否则表示“不够减”。

- 当被除数与除数异号时，作加法，若得到的余数与除数异号，表示“够减”，否则表示“不够减”。

算法比较表：

比较$[x]_{补}$与$[y]_{补}$的符号	求余数	比较$[R_i]_{补}$与$[y]_{补}$的符号
同号	$[x]_{补}-[y]_{补}$	同号，表示“够减”
异号	$[x]_{补}+[y]_{补}$	异号，表示“够减”

2. 商值的确定
上商的算法可归纳为以下两点：

- 如果$[x]_{补}$与$[y]_{补}$同号，商为正，则“够减”时上商“1”，“不够减”时上商“0”（按原码规则上商）。
- 如果$[x]_{补}$与$[y]_{补}$异号，商为负，则“够减”时上商“0”，“不够减”时上商“1”（按反码规则上商）。

简化的商值确定：

$[R]_{补}$与$[y]_{补}$	商值
同号	1
异号	0

(2) 商符的形成
商符是在求商值的过程中自动形成的：

- 当$[x]_{补}$与$[y]_{补}$同号时，$[x]_{补}-[y]_{补}$所得的余数$[R_0]_{补}$必与$[y]_{补}$异号，上商“0”，与商符（正）一致。
- 当$[x]_{补}$与$[y]_{补}$异号时，$[x]_{补}+[y]_{补}$所得的余数$[R_0]_{补}$必与$[y]_{补}$同号，上商“1”，与商符（负）一致。

商符还可以用于判断商是否溢出（不考虑小数补码运算时，商等于“-1”的情况）：

- 当$[x]_{补}$与$[y]_{补}$同号时，若$[R_0]_{补}$与$[y]_{补}$同号，上商“1”，即溢出。
- 当$[x]_{补}$与$[y]_{补}$异号时，若$[R_0]_{补}$与$[y]_{补}$异号，上商“0”，即溢出。

(3) 新余数的形成

$[R_i]_{补}$与$[y]_{补}$	商	新余数$[R_{i+1}]_{补}$

与	商	新余数
同号	1	$[R_{i+1}]_{补}=2[R_i]_{补}+[-y]_{补}$
异号	0	$[R_{i+1}]_{补}=2[R_i]_{补}+[y]_{补}$

如果对商的精度没有要求，一般可采用“末位恒置 1”法，这种方法操作简单，易于实现，最大误差仅为2^{-n}。

(4) 小结

- 补码除法共上商 n +1 次（末位恒置 1），第一次为商符
- 第一次商可用于判断溢出
- 总共执行 n 次加操作和 n 次移位操作
- 用移位的次数判断除法是否结束
- 精度误差最大为2^{-n}

(5) 补码除和原码除（加减交替法）比较

	原码除	补码除
商符	$x_0 \oplus y_0$	自然形成
操作数	绝对值补码	补码
上商原则	余数的正负	比较余数和除数的符号
上商次数	$n+1$	$n+1$
加法次数	$n+1$	n
移位	逻辑左移	逻辑左移
移位次数	n	n
第一步操作	$[x^*]_{补}-[y^*]_{补}$	同号：$[x^*]_{补}-[y^*]_{补}$ 异号：$[x^*]_{补}+[y^*]_{补}$

6.4 浮点四则运算

经过6.2的学习之后，我们已经对浮点数有了一定的了解，在6.4中我们将会介绍浮点数的四则运算（主要是加减）。

6.4.1 浮点加减

现在假设我们有两个浮点数

$x = s_x * r^{j_x}$

$y = s_y * r^{j_y}$

而当二者的阶码不等时，由于小数点位置不同，所以无法直接进行加减运算。所以浮点数的加减通常按照以下步骤进行：

1. 对阶
2. 尾数求和
3. 规格化
4. 舍入
5. 溢出判断

对阶

对阶的目的是小阶向大阶看齐，所以阶小的尾数向右位移，每右移一位，阶码+1，至相等为止。但可能会丢失数码影响精度。

例（参见原书270页）：

$x = 0.1101 * 2^{01}$

$y = (-0.1010 * 2^{11})$

求x+y

进行加法前，需先进行对阶，求出阶差

$[\Delta_j]_{补} = [i_x]_{补} - [j_y]_{补} = 00,01 + 11,01 = 11,10$

求出阶码之后，按小阶向大阶看齐的原则，将x的尾数右移两位，则对阶完毕。

尾数求和

只需按定点数的运算规则进行运算即可

规格化

当尾数求和（差）的结果不满足双符号位补码规格化的形式时，就需要规格化。规格化分左规和右规。

- 左规

  使用条件：尾数出现00.0xxx……x或11.1xxx……x时。尾数左移一位，阶码减一，直至符合00.1xxx……x或11.0xxx……x时。

- 右规

使用条件： 尾数出现01.xxx……x或10.xxx……x时。在定点运算时算作溢出，但在浮点运算时，只需尾数右移一位，阶码+1即可处理

舍入

在对阶和右规的过程中，可能会丢失尾数，影响精度，使用用舍入法提高精度。

- “0舍1入”法

  若被移去的最高位是0，则舍去（+0），若最高位是1，则末尾+1.但可能使尾数仍然溢出，此时需要再做一次右规。所以又可以理解为，舍去谁就加谁。

- “恒置1”法

  不论移去的最高位是0还是1，始终让右移后的尾数恒置1.但也同样存在使尾数变大变小的两种可能。

溢出判断
浮点机的溢出由阶码的符号决定：

$[j]_{补}=01,xx.....x$时为上溢；$[j]_{补}=10,xx.....x$时为下溢，按机器零处理。

第7章 指令系统

本章概述

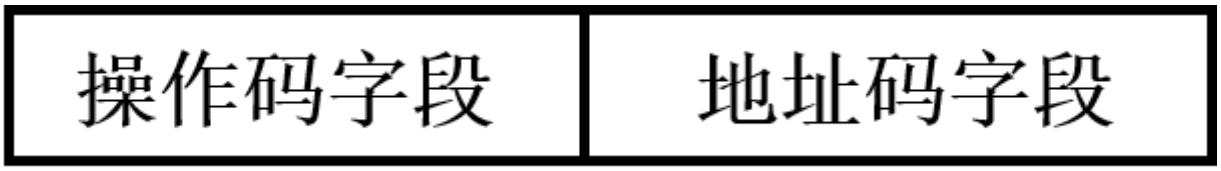
本章主要介绍机器指令系统的分类、常见的寻址方式、指令格式以及RISC技术。

7.1 机器指令

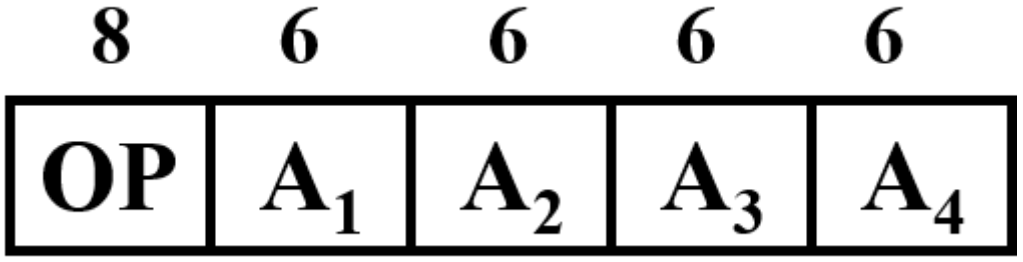
1.机器指令：每一条机器语言的语句。
2.指令系统：全部机器指令的集合。

7.1.1 指令的一般格式

指令是由操作码+地址码两部分组成的。



1.操作码：指明该指令所要完成的操作，其位数反映了机器的操作种类。
（1）操作码长度固定：用于指令字长较长的情况。
（2）操作码长度不固定：操作码分散在指令字的不同字段中。通常采用扩展操作码技术，使操作码的长度随地址数的减少而增加。
***扩展操作码的设计原则**：
a.不允许短码是长码的前缀。
b.各指令的操作码一定不能重复。
关于扩展操作码详见书P301图7.2、P302例7.1，以及网上一些比较平易近人的讲解：
https://blog.csdn.net/nuo_Shar/article/details/79146830
2.地址码：指出该指令的源操作数的地址、结果的地址以及下一条指令的地址。
（1）四地址指令：



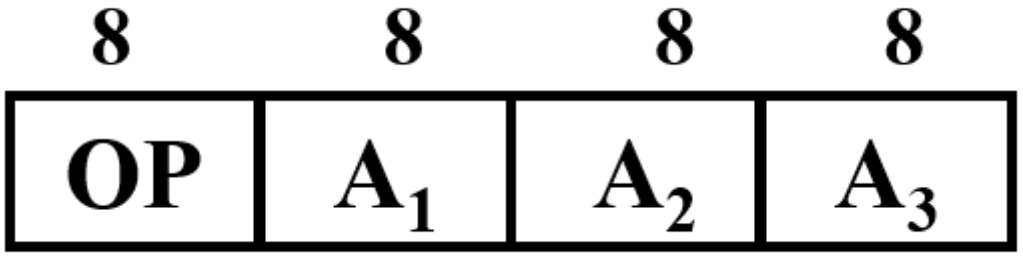
OP为操作码，A_1为第一操作数地址，A_2为第二操作数地址，A_3为结果地址，A_4为下一条指令地址。

完成操作：$(A_1)OP(A_2)\rightarrow A_3$

访存次数：4（取指令一次，取两个操作数两次，存放结果一次）

操作数寻址范围：2^6

（2）三地址指令：



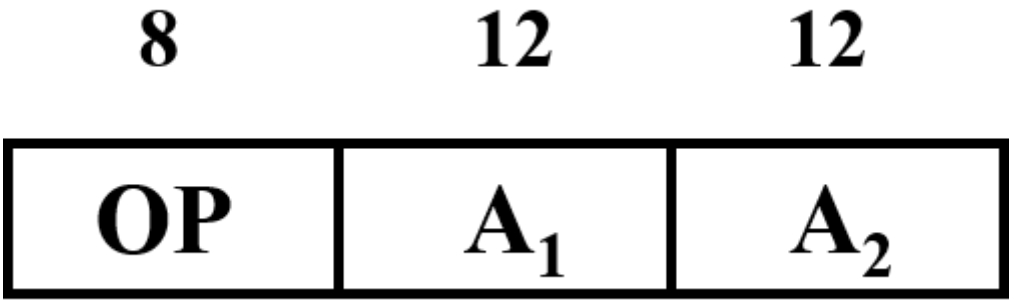
与四地址指令的区别：省去A_4，由程序计数器PC自动形成下一条指令的地址。

完成操作：$(A_1)OP(A_2)\rightarrow A_3$

访存次数：4（取指令一次，取两个操作数两次，存放结果一次）

操作数寻址范围：2^8

（3）二地址指令：



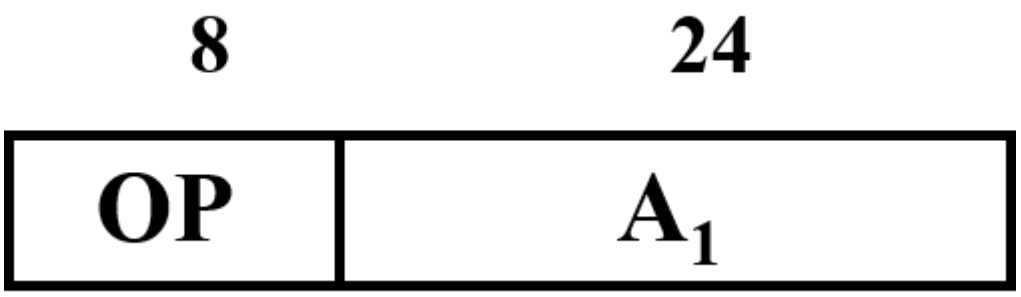
与三地址指令的区别：结果的地址不再由A_3给出，而是由A_1或A_2代替，或者由ACC存放结果

完成操作：$(A_1)OP(A_2)\rightarrow A_1$或$(A_1)OP(A_2)\rightarrow A_2$或$(A_1)OP(A_2)\rightarrow ACC$

访存次数：若结果存于A_1或A_2，需4次访存；若结果存于ACC，需3次访存。

操作数寻址范围：2^{12}

（4）一地址指令：



与二地址指令的区别：ACC既存放参与操作的操作数，又存放运算的中间结果
完成操作：(ACC)OP(A_1)→ACC
访存次数：2（取指令一次，取操作数一次）
操作数寻址范围：2^{24}
（5）零地址指令：
在指令字中无地址码，如空操作、停机这类指令只有操作码。

7.1.2 指令字长

1.指令字长取决于操作码的长度+操作数地址的长度+操作数地址的个数。
2.指令字长、存储字长、机器字长的区分：
（1）指令字长：机器指令中二进制代码的总位数。
（2）存储字长：一个存储单元存储一串二进制代码（存储字），这串二进制代码的位数称为存储字长。存储字长可以是8位、16位、32位等。
（3）机器字长：计算机进行一次整数运算所能处理的二进制数据的位数。
当指令字长固定时，指令字长=存储字长。
当指令字长可变时，指令字长按字节的倍数变化。

7.2 操作数类型和操作类型

7.2.1 操作数类型

常见的操作数类型有地址（无符号整数）、数字（定点数、浮点数、十进制数）、字符（ASCII）、逻辑数据（逻辑运算）等。

7.2.2 数据在存储器中的存放方式

1.存储器可按字节、半字、字、双字访问。
2.字节的次序有两种：低字节为低地址和高字节为低地址。
详见书P306图7.3、7.4

7.2.3 操作类型

1.数据传送：数据传送包括寄存器与寄存器、寄存器与存储单元、存储单元与存储单元之间的传送。
2.算术逻辑操作：实现算术运算和逻辑运算。
3.移位：移位可分为算术移位、逻辑移位和循环移位三种。
4.**转移**：使用转移类指令，可以改变计算机执行指令的顺序。
（1）无条件转移：不受任何条件约束直接转移到下一条需执行指令地址。如“JMP X”。
（2）条件转移：根据当前指令的执行结果来决定是否需要转移。
（3）调用与返回：调用指令可实现从一个程序转移到另一个程序的操作。调用指令（CALL）一般与返回指令（RETURN）配合使用。每一条CALL指令对应一条RETURN指令。返回地址可以存放在寄存器内、子程序的入口地址内、栈顶内。
（4）陷阱（Trap）与陷阱指令：陷阱是一种意外事故的中断。
5.输入输出

7.3 寻址方式

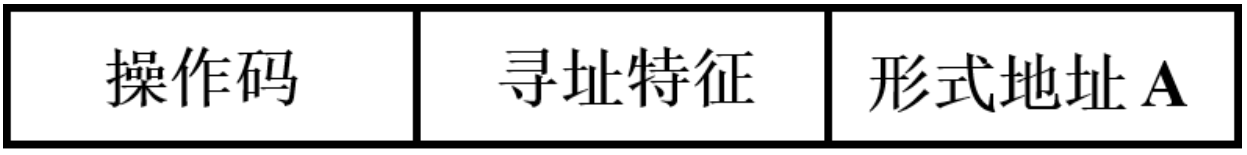
1.寻址方式：本条指令的数据地址+下一条将要执行的指令地址。
2.寻址方式分为指令寻址+数据寻址两大类。

7.3.1 指令寻址

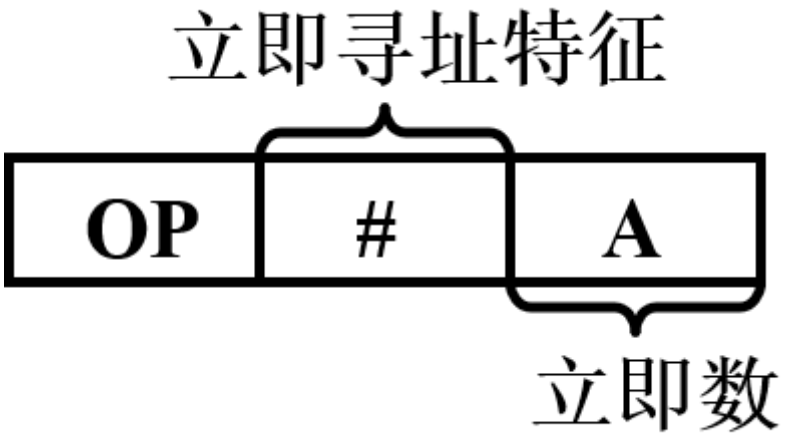
指令寻址分为顺序寻址+跳跃寻址两种。
1.顺序寻址：(PC)+1→PC。通过程序计数器PC加一，自动形成下一条指令的地址。
2.跳跃寻址：通过转移类指令实现。

7.3.2 数据寻址

1.数据寻址的指令格式如下图所示：

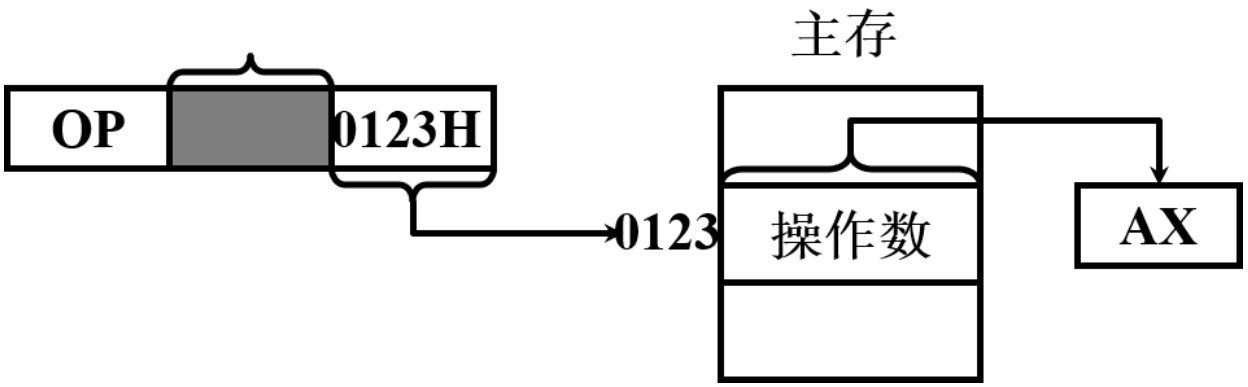


（1）寻址特征：用来指明属于哪种寻址方式。
（2）形式地址（A）：指令字中的地址，不代表操作数的真实地址。
（3）有效地址（EA）：操作数的真实地址，由寻址方式和形式地址共同确定。
下面假设指令字长=存储字长=机器字长。
2.各类寻址方式：
（1）立即寻址：形式地址A就是操作数本身（不是操作数地址），用补码表示。



特点：a.执行阶段不访存b.A 的位数限制了立即数的范围

（2）直接寻址：EA=A，有效地址由形式地址直接给出。



特点：a.执行阶段访问一次主存b.A的位数限定了操作数寻址范围c.操作数地址不易修改

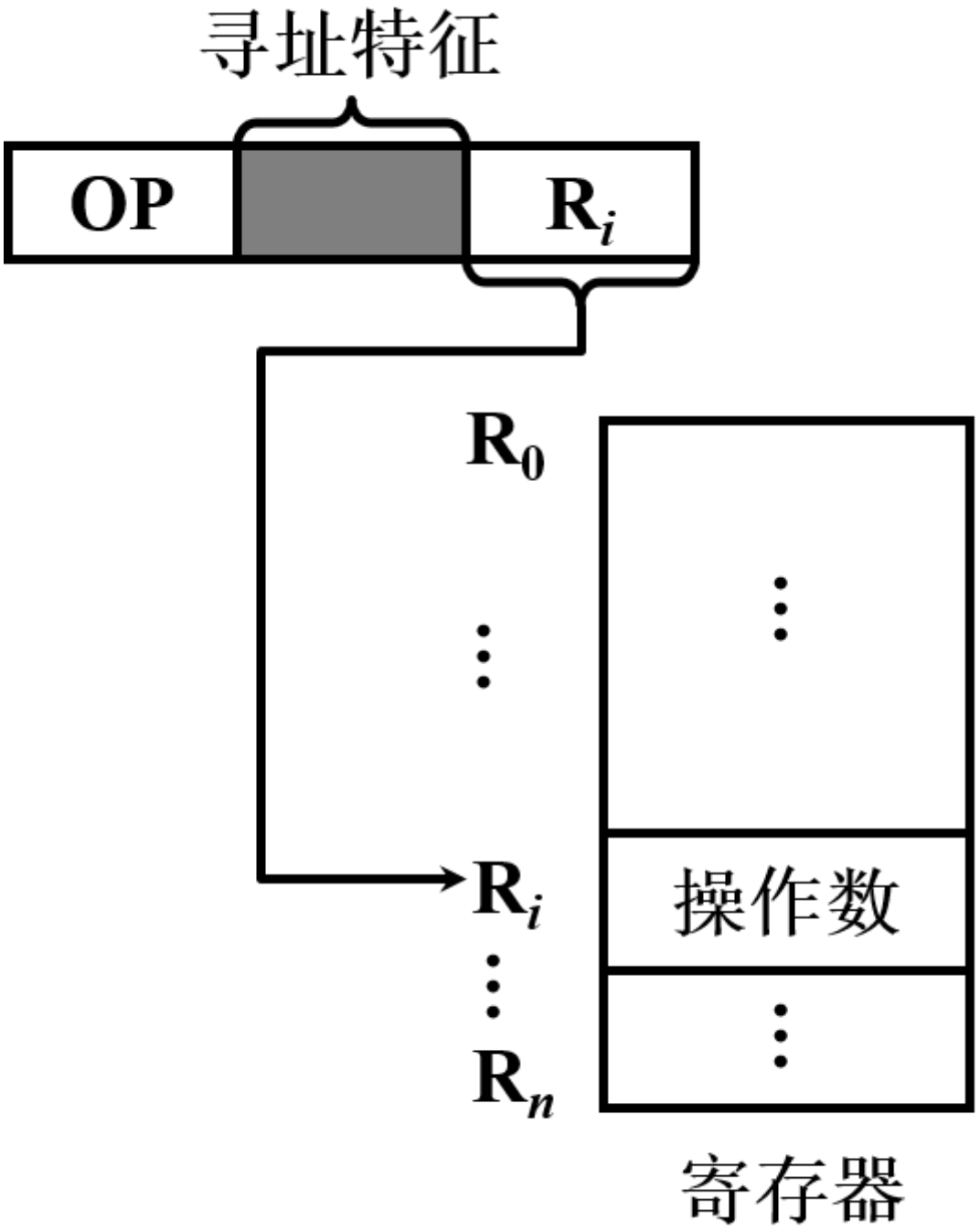
（3）隐含寻址：操作数的地址隐含在操作码或某个寄存器中。如一地址格式的加法指令。详见书P312图7.10。

特点：少了一个地址，利于缩短指令字长

（4）间接寻址：EA=(A)，有效地址由形式地址间接提供（类似于指针）。可以多次间址，多次间址需根据存储器的最高位确定几次访存。详见书P312图7.11。

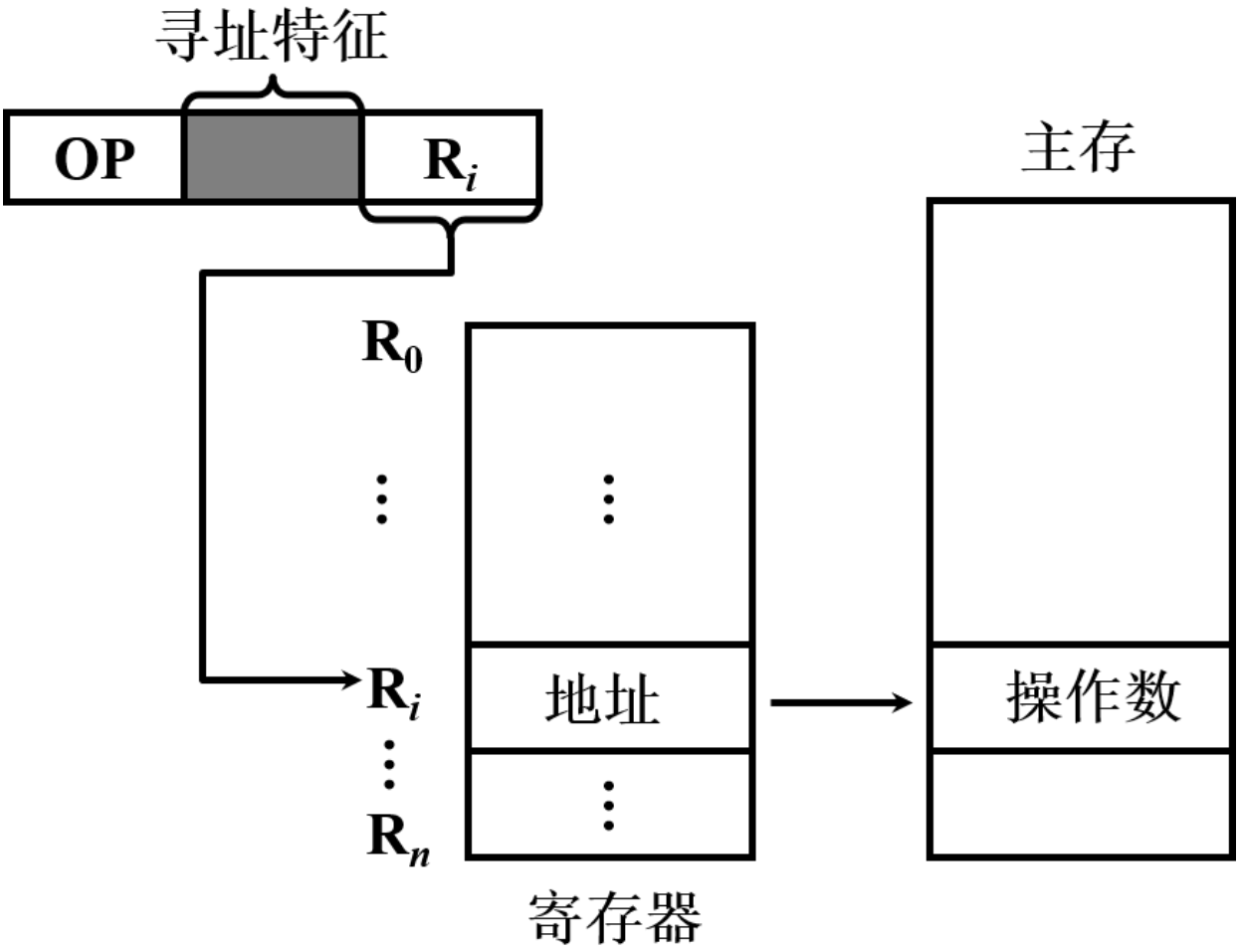
特点：a.指令在执行阶段要多次访存，执行时间延长b.可扩大寻址范围（有效地址EA的位数大于形式地址A的位数）c.便于编制程序

（5）寄存器寻址：EA=R_i，地址码字段直接指出寄存器的编号。



特点：a.执行阶段不访存，只访问寄存器，执行速度快b.寄存器个数有限，可缩短指令字长

（6）寄存器间接寻址：EA=(R_i)，有效地址EA在寄存器中。



特点：a.指令执行阶段需要访存（有效地址在寄存器中，操作数在存储器中）b.便于编制循环程序

（7）基址寻址：EA=(BR)+A，其中BR为基址寄存器（专用），也可用通用寄存器作为基址寄存器。详见书P314图7.15。

特点：a.扩大操作数的寻址范围（基址寄存器位数大于形式地址A的位数）b.有利于多道程序运行c.基址寄存器内容由操作系统或管理程序确定d.执行过程中，基址寄存器内容不变（作为基地址），形式地址A可变（作为偏移量）

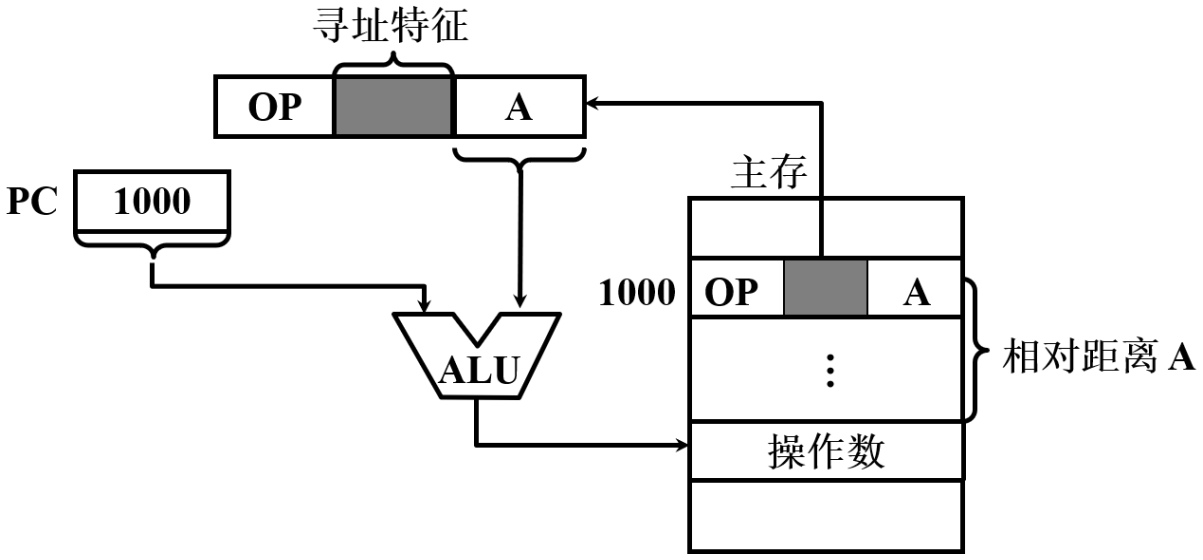
注意：通用寄存器作基址寄存器时，用户指定哪个寄存器作基址寄存器，但其内容仍由操作系统确定。

（8）变址寻址：EA=(IX)+A，其中IX为变址寄存器（专用），也可用通用寄存器作为变址寄存器。详见书P315图7.16。

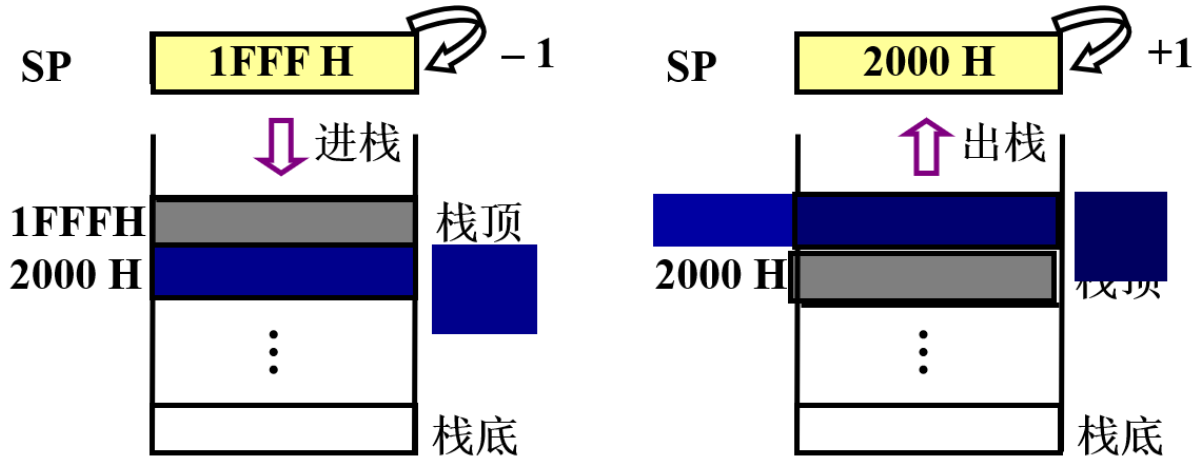
特点：a.扩大操作数的寻址范围（变址寄存器位数大于形式地址A的位数）b.变址寄存器内容由用户给定c.执行过程中，变址寄存器内容可变（作为偏移量），形式地址A不变（作为基地址）d.便于处理数组问题

基址寻址和变址寻址可以相互比较理解

（9）相对寻址：EA=(PC)+A，A是相对于当前指令地址的位移量（可正可负，用补码表示）。*为相对寻址特征。



特点：a.A的位数决定操作数的寻址范围b.便于程序浮动c.广泛应用于转移指令
可结合书P318例7.2理解（注意CPU从存储器取出一个字节时，自动完成(PC)+1→PC）。
（10）堆栈寻址：堆栈分为硬堆栈（多个寄存器）和软堆栈（指定的存储空间），特点是先进后出。栈顶地址由SP指出，有效地址EA隐含在堆栈指针SP中。进栈(SP)-1→SP，出栈(SP)+1→SP。



特点：a.指令中可以少一个地址字段（有效地址隐含在SP中）b.进栈出栈要修改地址指针，进栈(SP)-Δ→SP，出栈(SP)+Δ→SP。Δ取值与主存编址方式有关：按字编址Δ取1；按字节编址需根据存储字长是几个字节构成才能确定Δ，如字长16位时Δ取2（16/2=8）。

7.4 指令格式举例

不重要，跳过。

7.5 RISC技术

1.RISC(Reduced Instruction Set Computer)：精简指令系统计算机。
2.CISC(Complex Instruction Set Computer)：复杂指令系统计算机。

7.5.1 RISC的产生和发展

1.80-20规律：典型程序中80%的语句仅仅使用处理机中20%的指令-引发RISC技术

7.5.2 RISC的主要特征

1.RISC的主要特点：
（1）选用使用频度较高的一些简单指令，复杂指令的功能由简单指令来组合。
（2）指令长度固定、指令格式种类少、寻址方式少。
（3）访存指令只有LOAD/STORE。
（4）CPU中有多个通用寄存器。
（5）控制器采用组合逻辑控制。
（6）采用流水技术，大部分指令在1个时钟周期完成。
（7）采用优化的编译程序。
2.CISC的主要特点：
（1）系统指令复杂庞大，各种指令使用频度相差大。
（2）指令长度不固定、指令格式种类多、寻址方式多。
（3）可以访存的指令不受限制。
（4）CPU中设有专用寄存器。
（5）各种指令执行时间相差很大，大多数指令需多个时钟周期才能完成。
（6）控制器大多采用微程序控制。
（7）难以用优化编译生成高效的目标代码程序。

7.5.3 RISC和CISC的比较

1.RISC更能充分利用VLSI芯片的面积。
2.RISC更能提高运算速度。
3.RISC便于设计，可降低成本，提高可靠性。
4.RISC有利于编译程序代码优化。
5.RISC不易实现指令系统兼容。

第8章 CPU的结构和功能

本章概述

计算机最重要的是什么？CPU!本章就从CPU的结构和功能入手，讨论机器完成一条指令的全过程以及优化这个过程的流水技术。此外，我们还会进一步了解中断技术。

8.1CPU的结构

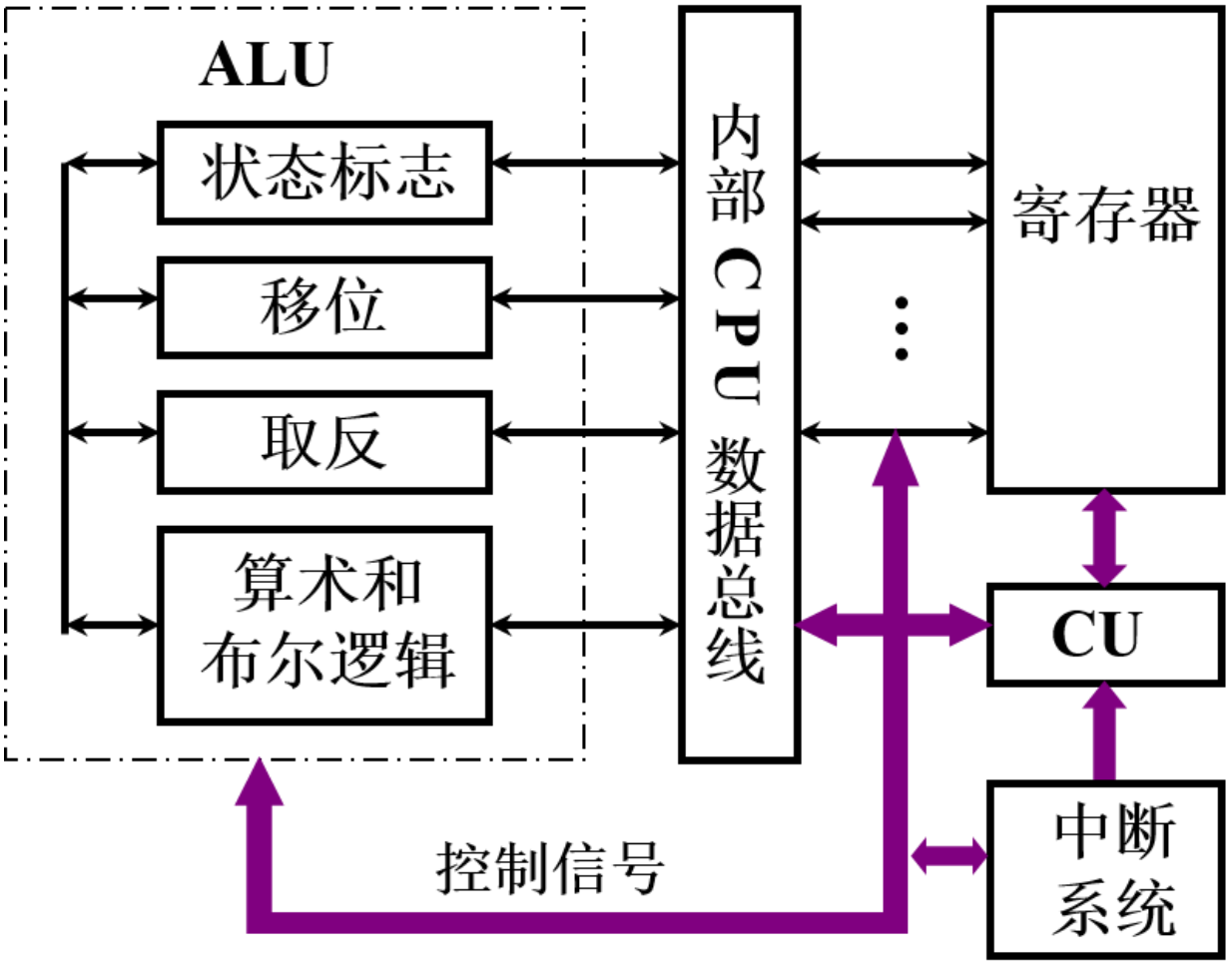
8.1.1 CPU的功能

CPU实质包括运算器和控制器两部分，而控制器负责协调并控制计算机各部件执行程序的指令序，其基本功能就是：取指令，分析指令，执行指令，控制程序的输入和运算结果的输出，对总线的管理，处理中断。

CPU的功能：
1.指令控制：控制程序的顺序执行
2.操作控制：产生完成每条指令所需的控制命令
3.时间控制：对各种操作加以时间的限制
4.数据加工：对数据进行算术运算和逻辑运算
5.处理中断

8.1.2 CPU结构框图

CPU的四大部分：
ALU(算术逻辑单元)，寄存器(组),中断系统和CU(控制单元)。



8.1.3 CPU的寄存器

1.用户可见寄存器：
(1)通用寄存器：存放操作数，或者作为满足某种寻址方式所需的寄存器
(2)数据寄存器：存放操作数，允许两个连读的寄存器存放双倍字长的值
(3)地址寄存器：存放地址。具有通用性，也可用于特殊的寻址方式
(4)条件码寄存器：存放条件码。条件码可被测试作为分支运算的依据。
2.控制和状态寄存器
(1)MAR：存储器地址寄存器，用于存放将被访问的存储单元的地址。
(2)MDR：存储器数据寄存器，用于存放欲存入存储器中的数据或者最近从存储器读出来的数据
(3)PC：程序计数器，存放现行指令的地址，通常具有计数功能
(4)IR：指令寄存器，存放当前预执行的指令
(5)存放程序状态字PSW的寄存器
(6)中断标记寄存器

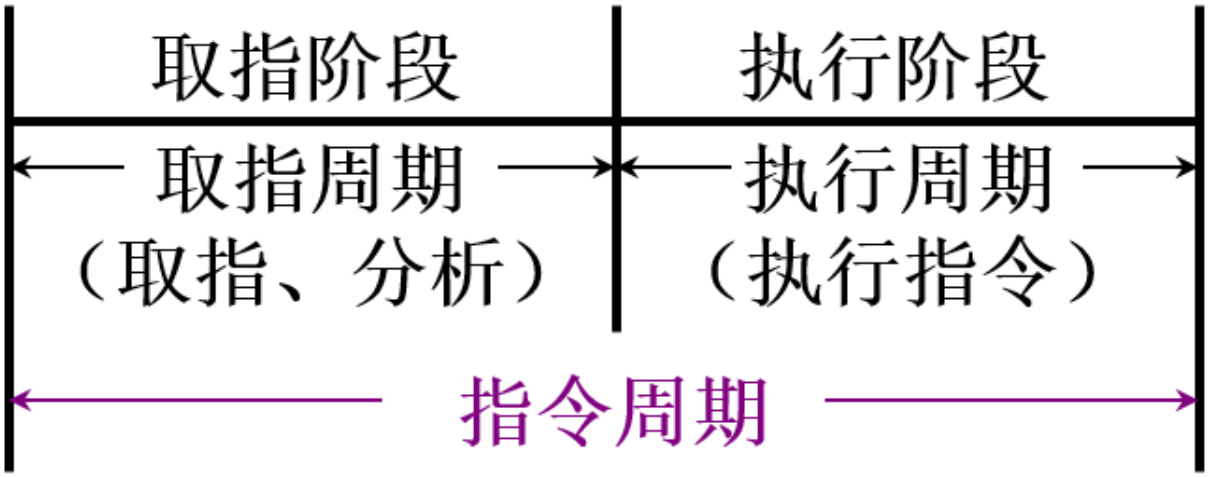
8.1.4 控制单元和中断系统

控制单元CU是提供完成计算机全部指令操作的微操作命令序列部件。
微操作命令序列的形成方法：
1.组合逻辑设计方法，是硬连线逻辑
2.微程序设计方法，是存储逻辑。
(具体内容参见第4篇)

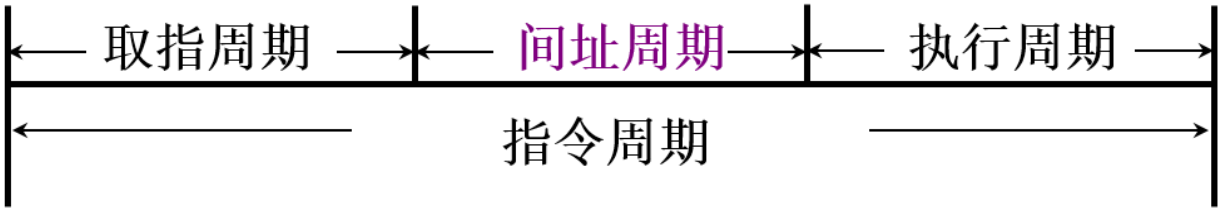
8.2 指令周期

8.2.1 指令周期的基本概念

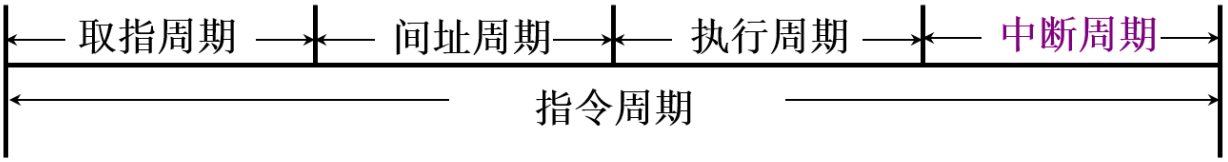
指令周期即CPU每取出并执行一条指令所需的全部时间。分为取指周期（取指令和分析指令）和执行周期（执行指令）。
取指周期–>执行周期



另外，
若是间接取址，还有间址周期；
取指周期–>间址周期–>执行周期



若在以上基础上还有中断，则还有中断周期。
取指周期–>间址周期–>执行周期–>中断周期



取指周期是为了取指令，间址周期是为了取有效地址，执行周期是为了取操作数，中断周期是为了保存程序断点。这四个周期又称为CPU的工作周期，为了区别它们，在CPU内设置4个标志触发器，分别为FE、IND、EX、INT

8.2.2 指令周期的数据流

1.取指周期的数据流
PC存放现行指令的地址，将该地址送到MAR，MAR将其传到地址总线，由CU通过控制总线向存储器发送读命令，读出来的命令经过数据总线传送到MDR，再送到IR，最后CU控制PC内容加一，形成下一条指令的地址。
2.间址周期的数据流
取指周期结束后，CU检查IR中的内容，确定是否含有间址操作。若有，则MDR中的数据传输到MAR，再传送到地址总线，CU通过控制总线向存储器发送读命令，读取出来的数据经数据总线传送到MDR并保存。
3.执行周期的数据流
由于指令的操作不同，所以执行周期的数据流也是不同的，无法用统一的数据流图表示。
4.中断周期的数据流
CU将用于保存程序断点的存储器特殊地址送到MAR，再送到地址总线上，CU通过控制总线向存储器发存储命令，并将PC的内容送到MDR，使程序断点经过数据总线存入存储器。此外，CU将中断服务程序的入口地址送到PC，为下一个指令周期的取指周期做准备。

8.3 指令流水

提高访存速度的方法：
1.提高存储芯片的性能
2.从体系结构上，采用多体、Cache等分级存储来提高存储器的性价比
提高主机和I/O交换信息的速度：
1.DMA方式
2.多总线结构
提高运算速度：
1.高速芯片和快速进位链
2.改进算法
提高处理机速度：
1.提高器件的性能

2.改进系统结构和开发系统的并行性

并行包括同时性（多个事件同一时刻发生）和并发性（多个事件同一时间段发生）两个方面。

并行性的级别：

作业级或程序级、任务级或进程级、指令之间级和指令内部级。前两级是粗粒度（过程级，用算法实现），后两级是细粒度（指令级，用硬件实现）。

8.3.1 指令流水原理

指令流水就是上述细粒度并行性的一项重要技术。

指令预取：由指令部件取出一条指令并将它暂存起来，若执行部件空闲就将暂存的指令传给执行部件执行。与此同时，指令部件又可以取出下一条指令并暂存起来。

执行效率不可能加倍的原因：

1.指令的执行时间一般大于取指时间。

2.当遇到条件转移指令时，下一条指令是不可知的。必须等到执行阶段结束后才能获知条件是否成立，从而决定下一条指令的地址，造成时间损失。（猜测法来减少时间损失）

为了进一步提高处理速度，将指令的处理过程分为更细的几个阶段：

1.取指(FI)：从存储器取出一条指令并暂存入指令部件的缓冲区

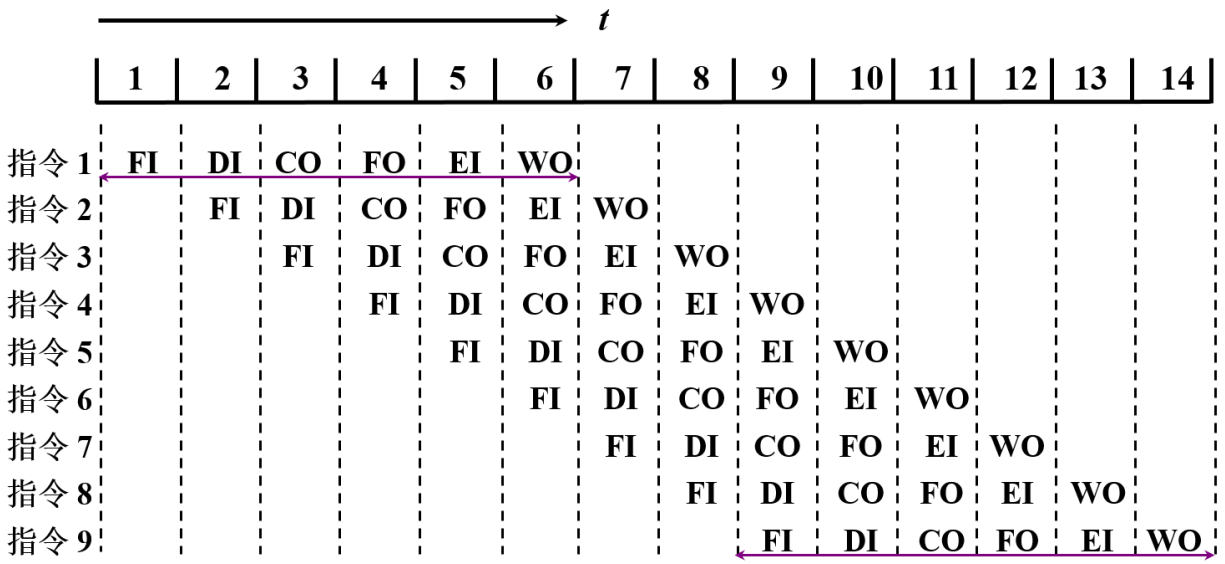
2.指令译码(DI)：确定操作性质和操作数地址的形成方式

3.计算操作数地址(CO)：计算操作数的有效地址

4.取操作数(FO)：从存储器里取出操作数（若操作数在寄存器中则无需此阶段）

5.执行指令(EI)：执行指令所需操作，并将结果存于目的位置（寄存器）

6.写操作数(WO)：将结构存入存储器。



8.3.2 影响流水线性能的因素

流水线不断流实现困难的原因：

1.结构相关（资源相关）：多条指令进入流水线后硬件资源满足不了指令重叠执行的要求。

解决方法：

（1）让流水线在完成前一条指令对数据的存储器访问时，暂停取后一个指令的操作。

（2）设置两个独立的存储器分别存放操作数和指令。

（3）指令预取技术（基于访存周期很短的情况）

2.数据相关：指令在流水线中重叠执行时，后继指令需要用到前面指令的执行结果。

解决方法：

（1）后推法：停顿后继指令的运行，直至前面指令的结果已经生成

（2）定向技术（旁路技术、相关专用通路技术）：直接将执行结果送到其他指令所需的地方。

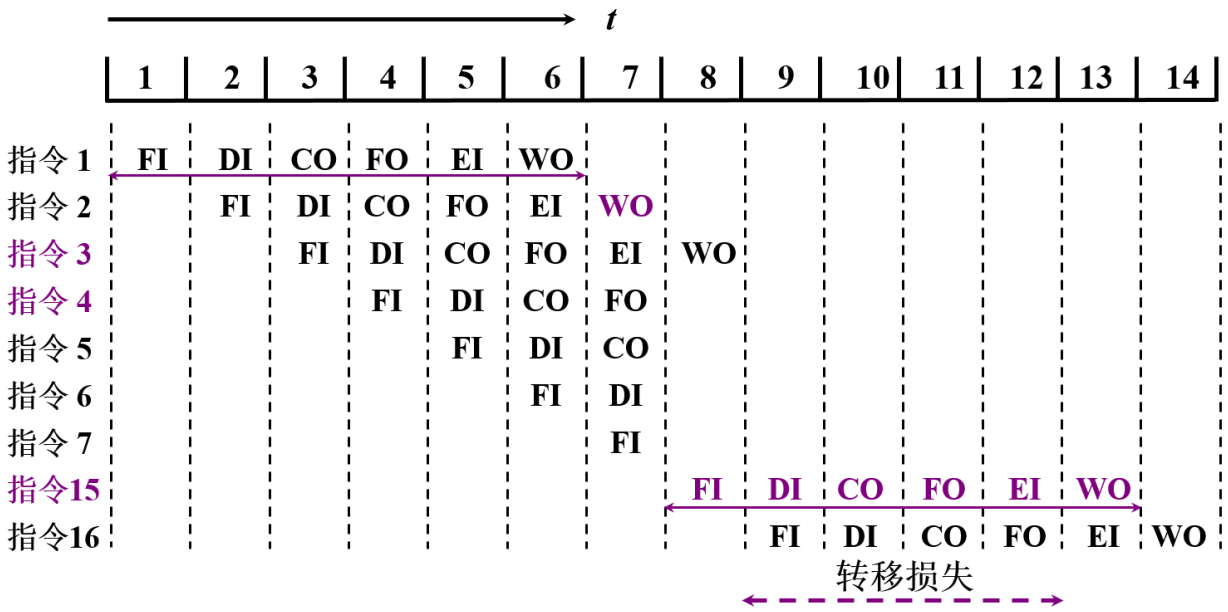
3.控制相关：流水线遇到分支指令和其他改变PC值的指令。

解决方法：

尽早判别转移是否发生，尽早生成转移目标地址；预取转移成功或不成功两个控制流方

向上的目标指令；加快和提前形成条件码；提高转移方向的猜准率等。

设 指令3 是转移指令



8.3.3 流水线性能

设有m段指令，每段时间为t，有n条指令。

1.吞吐率：单位时间内流水线完成指令或输出结果的数量。

最大吞吐率：流水线在连续流动达到稳定状态后的吞吐率。

T=1/t

实际吞吐率：总是小于最大吞吐率。

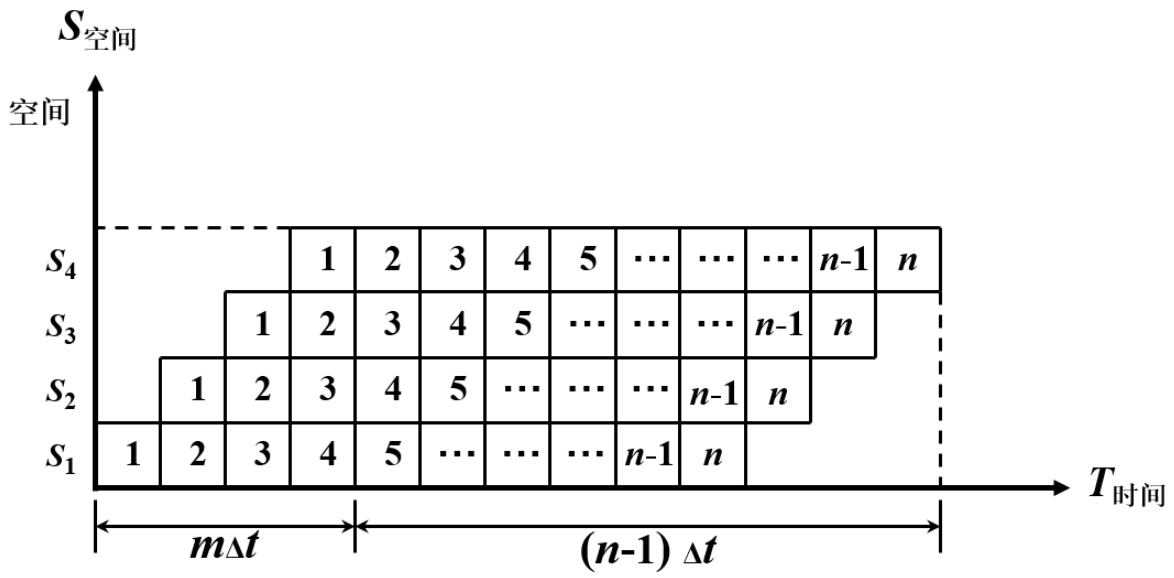
T=n/mt+(n-1)t

2.加速比：m段流水线的速度与等功能的非流水线的速度之比。

S=nmt/mt+(n-1)t

3.效率：流水线中各功能段的利用率。

E=mnt/m(m+n-1)t



8.3.4 流水线中的多发技术

1.超标量技术
2.超流水线技术
3.超长指令字技术
感兴趣的朋友详见课本356、357页

8.3.5 流水线结构

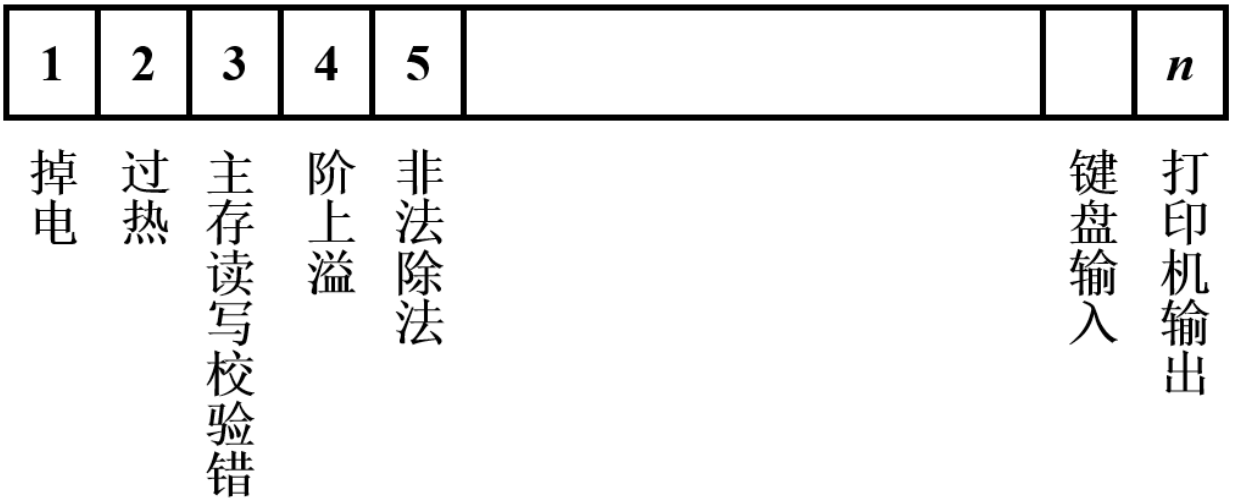
1.指令流水线结构：
将指令的整个执行过程用流水线进行分段处理。
2.运算流水线：
流水技术在部件级的应用，例如浮点数乘法中尾数乘本身就可以用流水线。

8.4 中断系统

8.4.1 概述

引起中断的因素：
1.人为设置的中断（自愿中断）
2.程序性事故
3.硬件故障
4.I/O设备
5.外部事件
中断源：引起中断的各个因素。
1.不可屏蔽中断：CPU不能禁止响应，如电源掉电

2.可屏蔽中断：CPU可根据该中断源是否被屏蔽来确定是否给与响应。



8.4.2 中断请求标记和中断判优逻辑

为了判断是哪个中断源提出请求，在中断系统中必须设置中断请求标记触发器，简称中断请求触发器，记为INTR，其状态为1时，表示中断源有请求。中断请求触发器越多，说明计算机处理中断能力越强。

当有多个中断源发出请求，中断系统必须根据优先顺序予以响应，称为中断判优。

判优顺序的依据：

若中断源得不到及时地响应致使机器工作出错的严重程度。对于I/O设备，速度高的设备优先级高。

中断判优的实现：

1.硬件排队：

（1）链式排队器

（2）设在CPU内的排队器，优先顺序按1234由高向低排列。

2.软件排队：

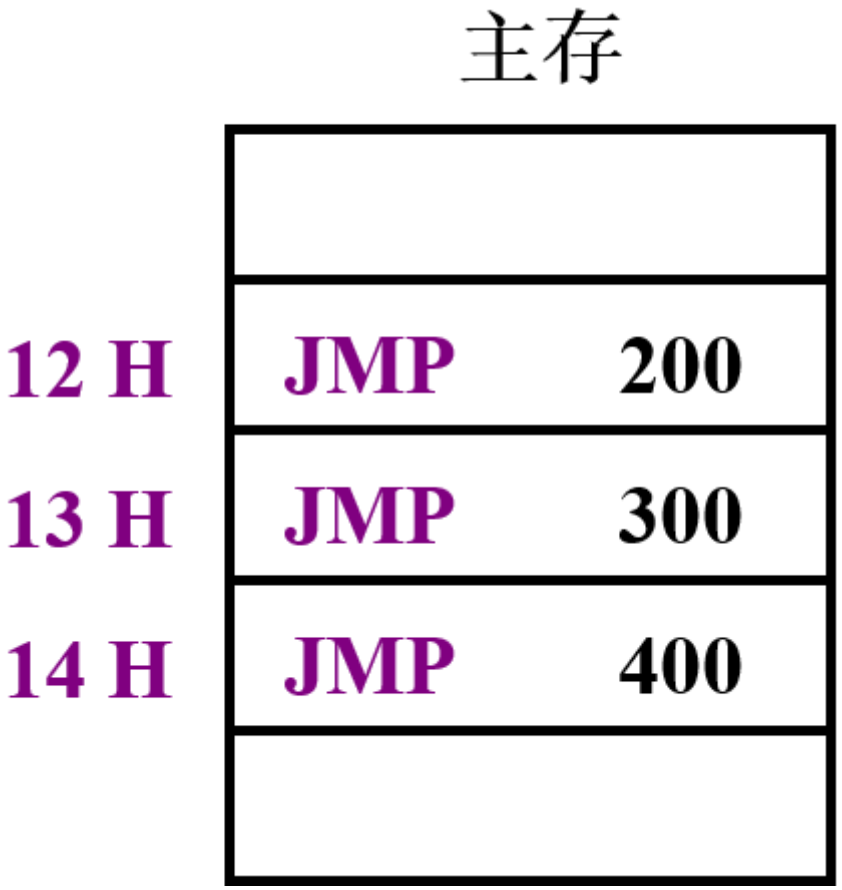
通过编写查询程序实现。程序按照优先等级从高到低查询各中断源是否有请求。

8.4.3 中断服务程序入口地址的寻找

1.硬件向量法：

利用硬件产生向量地址，再由向量地址找到中断服务程序的入口地址。

（1）无条件转移指令



（2）设置向量地址表

2.软件查询法：

用软件寻找中断服务程序入口地址。

8.4.4 中断响应

响应中断的条件：

EINT=1（允许中断触发器为1）且INTR=1（中断请求标记触发器为1）

响应中断的时间：

CPU在指令执行周期结束后，响应中断源的请求。

具体为：中断请求触发器的数据端来自各个中断源，当他们有请求时，数据端置1，当且仅当CPU发出的中断查询信号输入到触发器的时钟端时，才将INTR置1.

中断隐指令：即CPU在中断周期内由硬件自动完成的一条指令。

1.保护程序断点

将当前程序计数器PC的内容保存到存储器中

2.寻找中断服务程序的入口地址

（1）在中断周期内，将向量地址送至PC，使CPU执行下一条无条件转移指令，转至中断服务程序的入口地址。

（2）将软件查询入口地址的程序首地址送至PC，使PC执行中断识别程序，找到入口地址。

3.关中断

使EINT置0

8.4.5 保护现场和恢复现场

保护现场：保护程序断点；保护CPU内部各寄存器内容的现场。

恢复现场：在中断返回前，必须将寄存器的内容恢复到中断处理前的状态，由中断服务程序完成。

8.4.6 中断屏蔽技术

多重中断（中断嵌套）：

当CPU在执行某个中断服务程序时，另一个中断源又提出了新的中断请求，而CPU又响应了这个请求，暂时停止正在运行的服务程序，转去执行新的中断服务程序。

实现多重中断的条件：

1.提前设置开中断指令

多重中断开中断指令的位置前于单重中断。

2.优先级别高的中断源有权中断优先级别低的中断源

优先级	屏　　蔽　　字
1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2	0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
3	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
4	0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
5	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
6	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
⋮	⋮
15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
16	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

屏蔽技术：

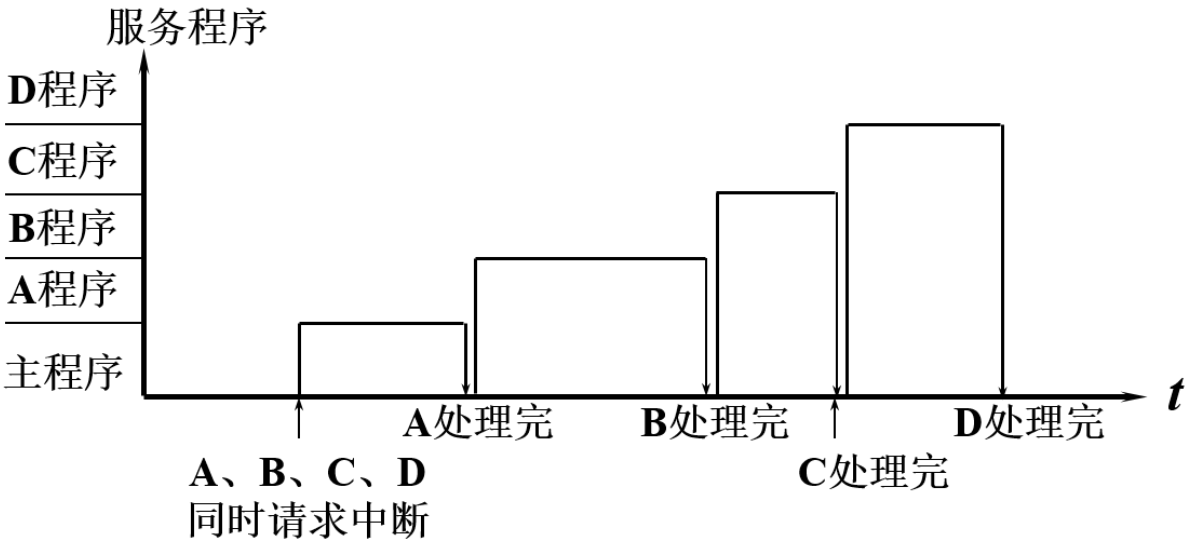
屏蔽技术可以改变优先等级。

优先等级包含响应优先级和处理优先级。

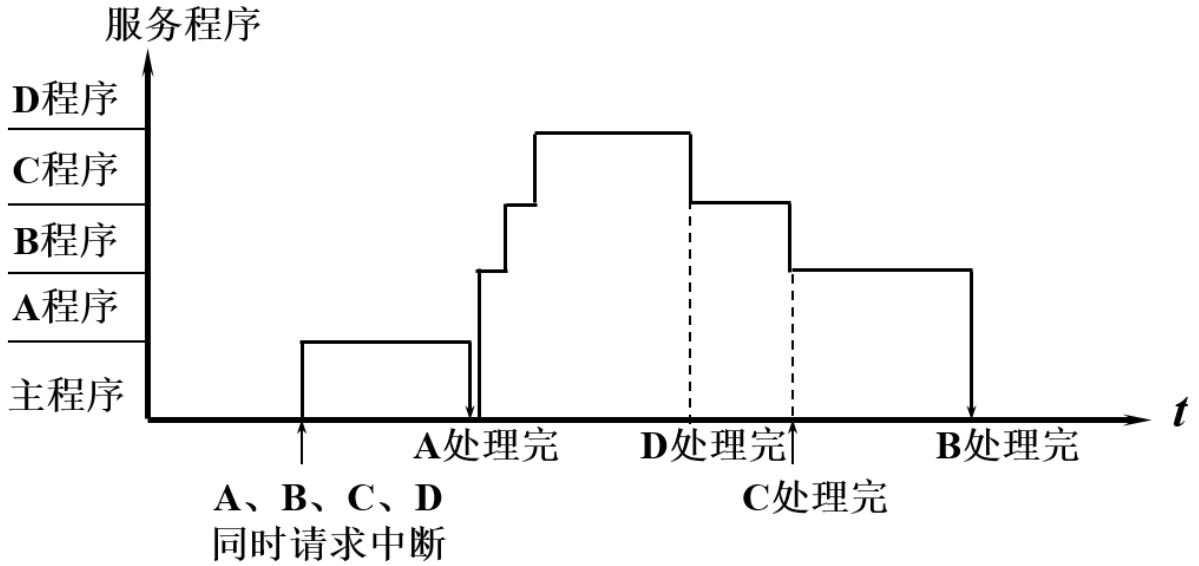
响应优先级是CPU响应各中断源请求的优先次序，由硬件线路设置好，不易改动。

处理优先级是CPU实际对各中断源请求的处理优先次序。若不采用屏蔽技术，响应的优先次序就是处理的优先次序。

中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1



CPU 执行程序轨迹（原屏蔽字）



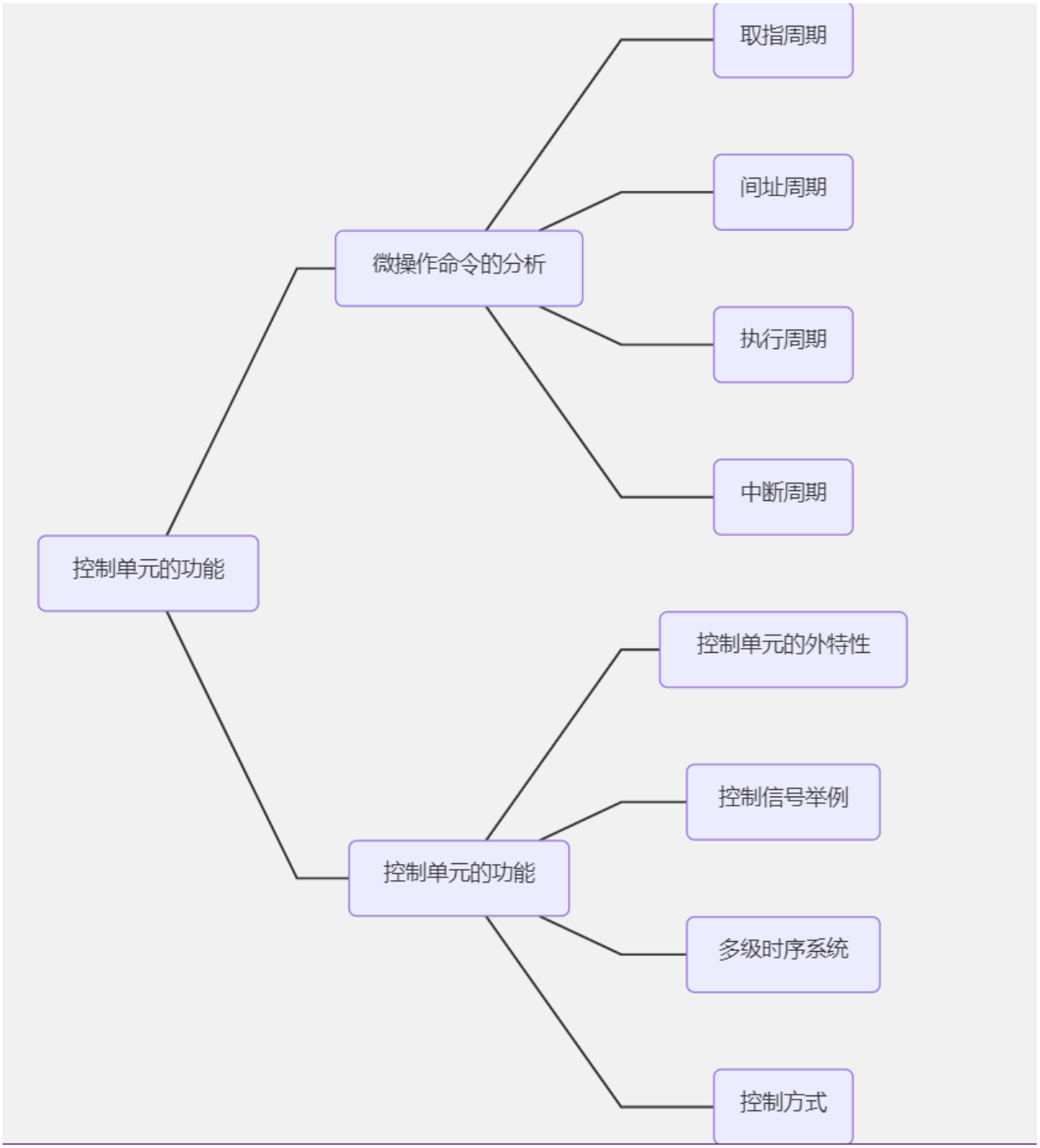
CPU 执行程序轨迹（新屏蔽字）

采用屏蔽技术的中断服务程序：

保护现场–>置屏蔽字–>开中断–>中断服务–>关中断–>恢复现场–>恢复屏蔽字–>开中断–>中断返回

第9章 控制单元的功能

本章概述



- 本章结合指令周期的4个阶段，着重分析控制单元为完成不同指令所发出的各种操作命令（控制信号），是第10章控制单元的设计的基础。

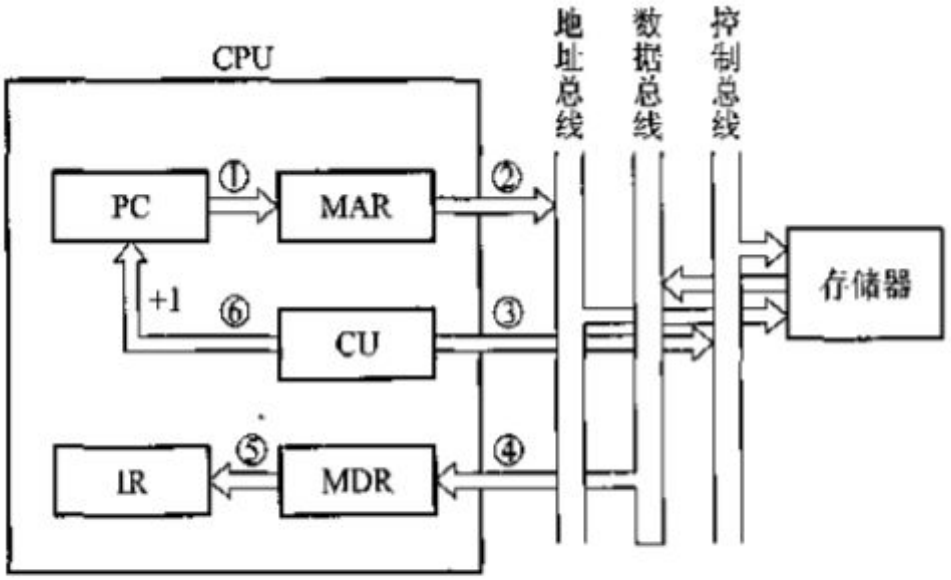
9.1 微操作命令的分析

- 完成一条指令分 4 个工作周期
  取指周期

间址周期
执行周期
中断周期

9.1.1 取指周期

- 取指周期数据流

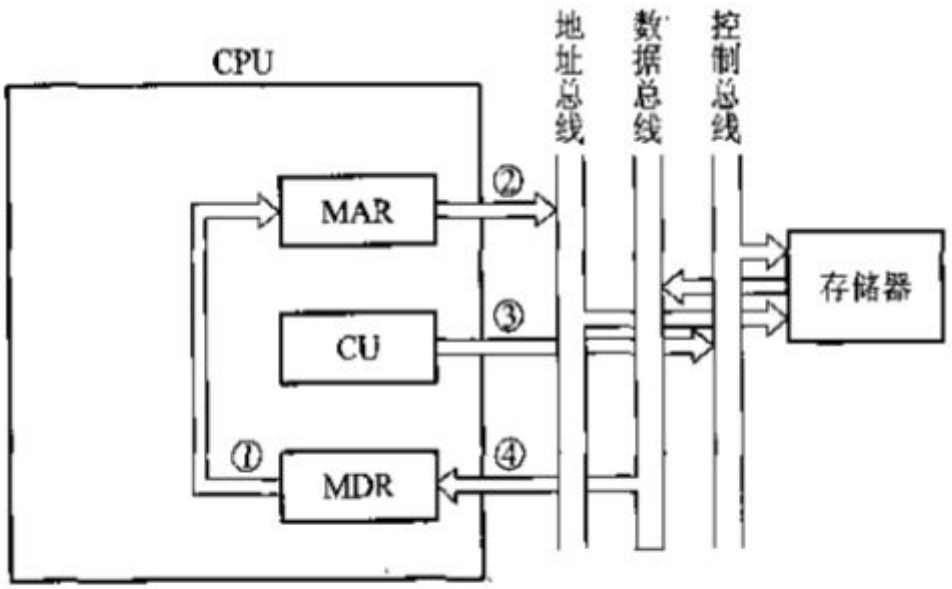


取指周期数据流

- 取指操作：
  PC→MAR→地址线
  1→R
  M(MAR)→MDR
  MDR→IR
  OP(IR)→CU
  (PC)+1→PC

9.1.2 间址周期

- 间址周期数据流



间址周期数据流

- 间址操作:
  Ad(IR)→MAR
  1→R
  M(MAR)→MDR
  MDR→Ad(IR)

9.1.3 执行周期

- 非访存指令
  (1) CLA　清A　0→ACC
  (2) COM　取反　ACC→ACC
  (3) SHR　算术右移　L(ACC)→R(ACC), ACC_0→ACC_0
  (4) CSL　循环左移　R(ACC)→L(ACC), ACC_0→ACC_n
  (5) STP　停机指令　0→G
- 访存指令
  （1）加法指令　ADD　X
  Ad(IR)→MAR
  1→R
  M(MAR)→MDR
  (ACC)+(MDR)→ACC
  (2) 存数指令　STA　X
  Ad(IR)→MAR

1→W

ACC→MDR

MDR→M(MAR)

(3) 取数指令　LDA　X

Ad(IR)→MDR

1→R

M(MAR)→MDR

MDR→ACC

- 转移类指令

(1) 无条件转移指令　JMP　X

Ad(IR)→PC

(2) 条件转移（负则转）指令　BAN　X

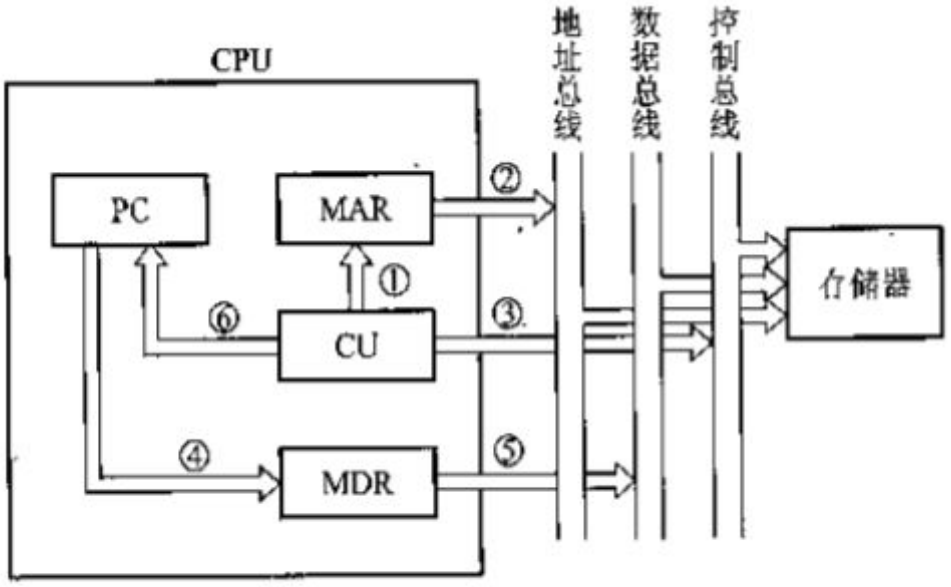
$A_0 \cdot Ad(IR)+A_0 \cdot (PC) \rightarrow PC$

- 三类指令的指令周期

三类指令的指令周期

9.1.4 中断周期

- 中断周期数据流

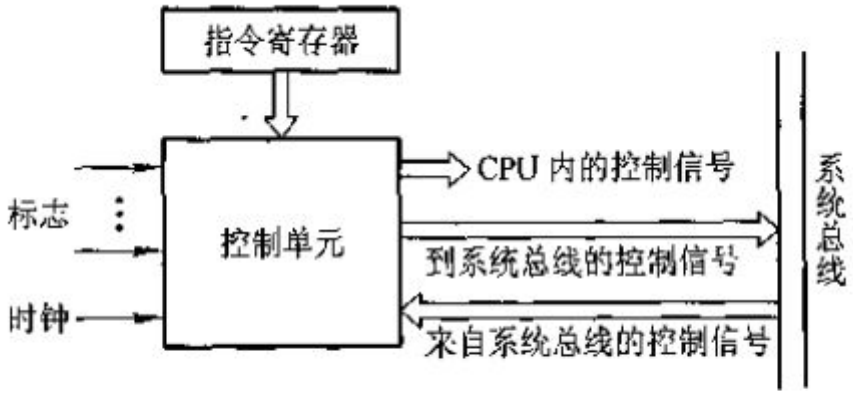


中断周期数据流

- 中断操作(程序断点存入“0”地址)
  0→MAR
  1→W
  PC→MDR
  MDR→M(MAR)
  向量地址→PC
  0→EINT(置"0")
  (如果程序断点存入堆栈，只需将上述第1步0→MAR改为(SP)-1→SP)

9.2 控制单元的功能

9.2.1 控制单元的外特性



控制单元外特性

1.输入信号
(1)时钟

- CU 受时钟控制，一个时钟脉冲发一个操作命令或一组需同时执行的操作命令

(2)指令寄存器OP (IR)→CU

- 控制信号与操作码有关

(3)标志

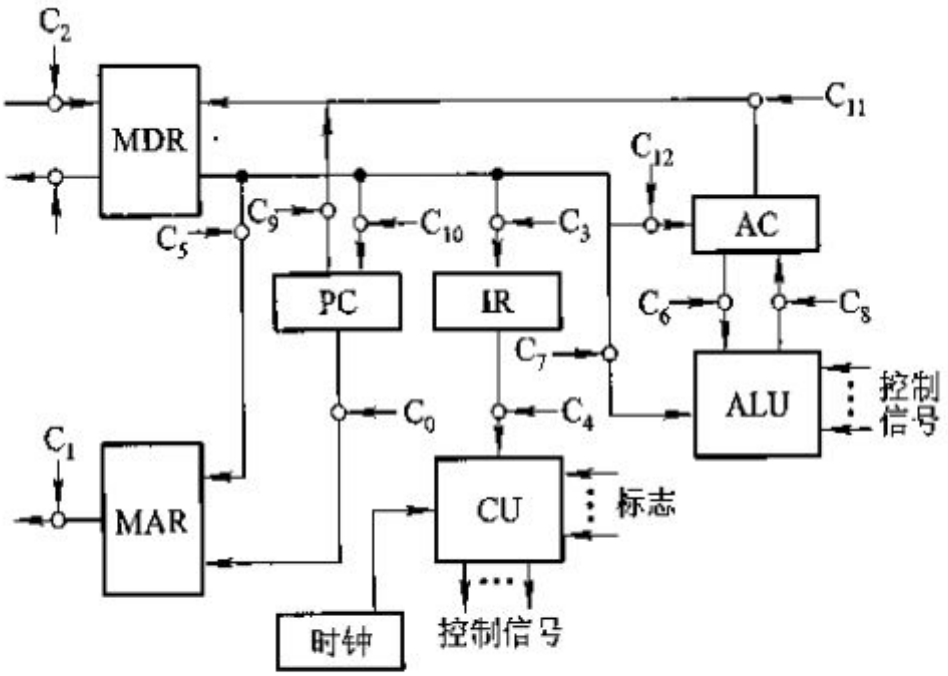
- CU受标志控制

(4) 外来信号

2.输出信号
(1) CPU 内的各种控制信号
(2)送至控制总线的信号

9.2.2 控制信号举例

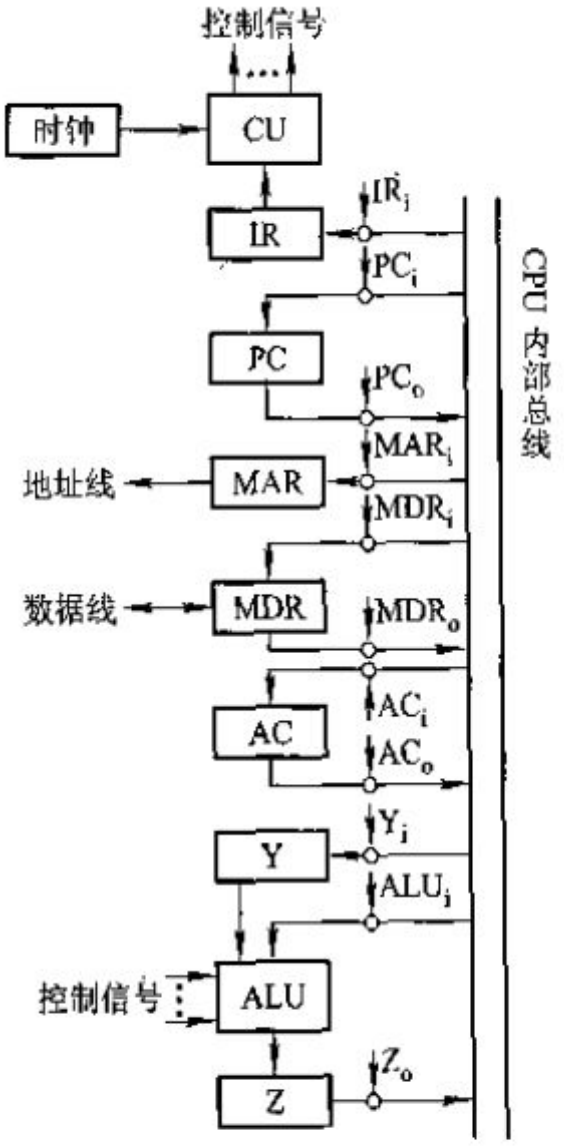
1.不采用 CPU 内部总线的方式



未采用 CPU 内部总线方式的数据通路和控制信号

取指周期、间址周期、执行周期控制信号详见课本p380

2.采用CPU内部总线的方式



采用 CPU 内部总线方式的
数据通路和控制信号

取指周期、间址周期、执行周期控制信号详见课本p382

9.2.3 多级时序系统

1.机器周期

(1) 机器周期的概念

- 所有指令执行过程中的一个基准时间

(2) 确定机器周期需考虑的因素

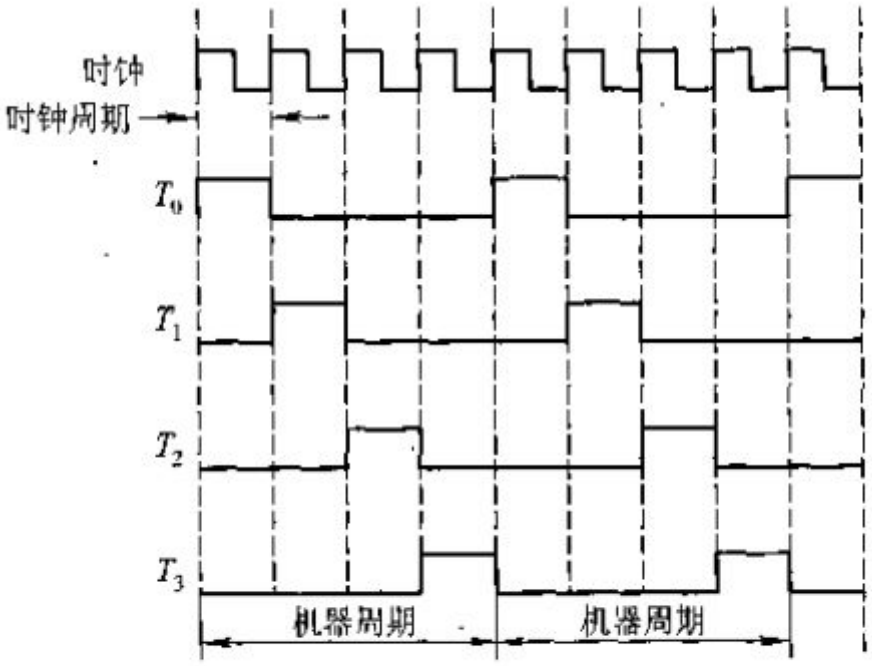
- 每条指令的执行步骤
- 每一步骤所需的时间

(3) 基准时间的确定

- 以完成最复杂指令功能的时间为准
- 以访问一次存储器的时间为基准
- 若指令字长 = 存储字长，则取指周期 = 机器周期

2.时钟周期（节拍、状态）

- 一个机器周期内可完成若干个微操作
- 每个微操作需一定的时间
- 将一个机器周期分成若干个时间相等的时间段（节拍、状态、时钟周期）
- 时钟周期是控制计算机操作的最小单位时间
- 用时钟周期控制产生一个或几个微操作命令

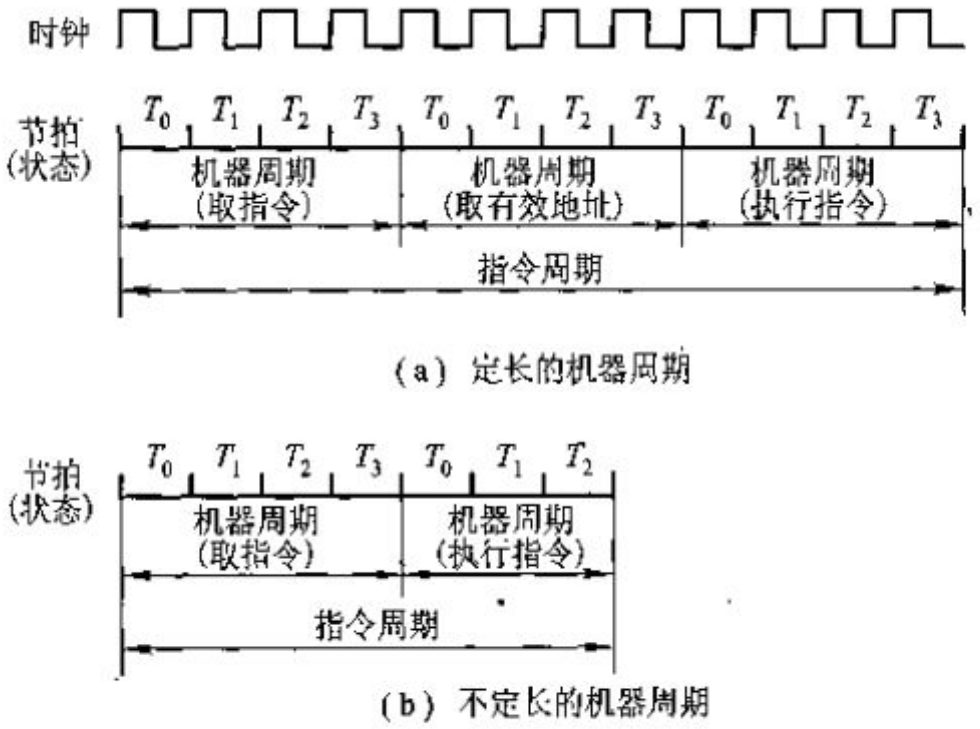


机器周期、时钟周期和节拍的关系

3.多级时序系统

- 机器周期、节拍（状态）组成多级时序系统

- 一个指令周期包含若干个机器周期
- 一个机器周期包含若干个时钟周期



指令周期、机器周期、节拍和时钟周期的关系

4.机器速度与机器主频的关系

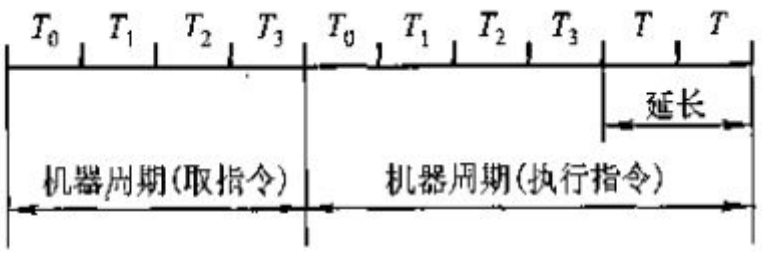
- 机器的主频 f 越快机器的速度也越快
- 在机器周期所含时钟周期数相同的前提下，两机平均指令执行速度之比等于两机主频之比
- 机器速度不仅与主频有关，还与机器周期中所含时钟周期（主频的倒数）数以及指令周期中所含的机器周期数有关

9.2.4 控制方式

产生不同微操作命令序列所用的时序控制方式

- 同步控制方式
  任一微操作均由 统一基准时标 的时序信号控制
  (1)采用定长的机器周期
  以最长的微操作序列和最繁的微操作作为标准
  机器周期内节拍数相同

(2)采用不定长的机器周期



延长机器周期示意

(3)采用中央控制和局部控制相结合的方法

局部控制的节拍宽度与中央控制的节拍宽度一致

其他控制方式（异步控制方式、联合控制方式、人工控制方式）见课本p389

第10章 控制单元的设计

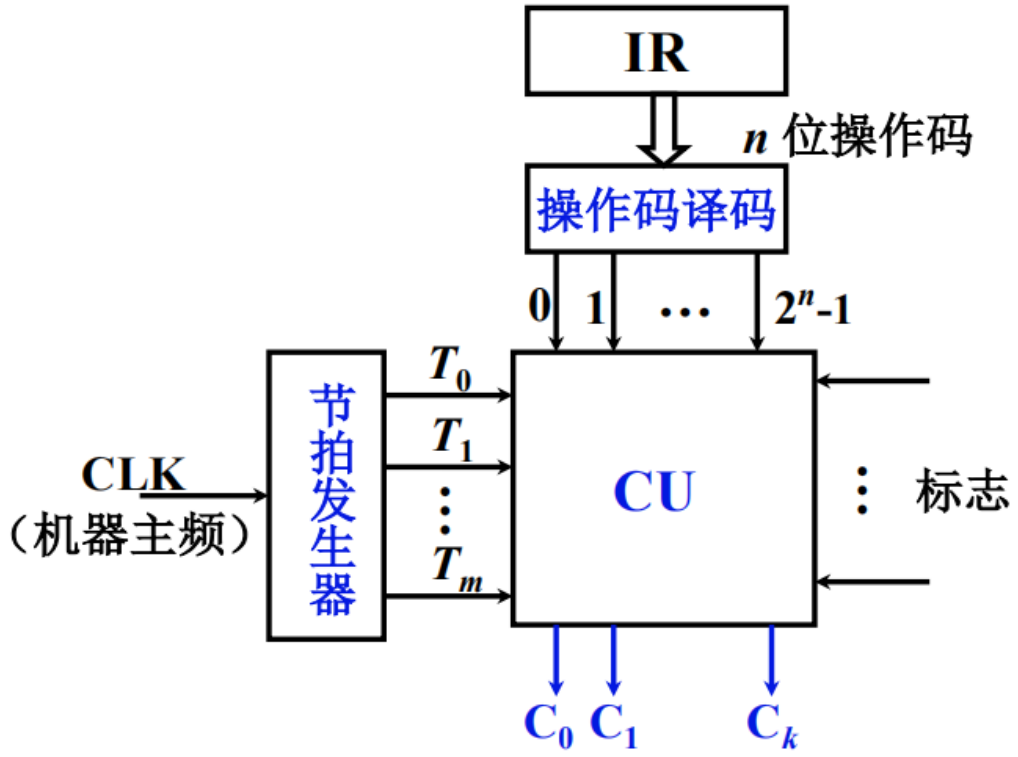
本章概述

本章以十条机器指令为例，介绍控制单元的两种设计方法，旨在使读者初步掌握设计控制单元的思路，为今后的设计计算机打下初步基础。**（下学期的课程设计一定会用到这一部分内容，尤其是微程序设计。）**

10.1 组合逻辑设计

10.1.1 组合逻辑控制单元框图

带译码和节拍输入的控制单元框图：



其中控制单元的时钟输入实际上是一个脉冲序列，其频率即为机器的主频，它使CU能按一定的节拍发出各种控制信号。

10.1.2 微操作的节拍安排（重点）

1.特点：

采用规整的同步控制方式，一个 机器周期内有3个节拍（时钟周期），CPU 内部结构采用非总线方式。

2.安排微操作时序的原则：

原则一 注意微操作的先后顺序，因为有些微操作的次序是不容改变的。

原则二 被控对象不同 的微操作（硬件资源不冲突）尽量安排在 一个节拍 内完成，节省时间。

原则三 占用时间较短的微操作尽量安排在一个节拍内完成并允许有先后顺序。

3.节拍安排：

1）取指周期：

T0 PC->MAR　　1->R

T1 M (MAR)->MDR　　(PC) + 1->PC

T2 MDR->IR　　OP (IR)->ID(Decoder)

分析：T0，T1周期符合原则二，T2符合原则三。

2）间址周期：

T0 Ad (IR)->MAR　　1->R

T1 M (MAR)->MDR

T2 MDR->Ad (IR)

3）执行周期：

3.1）非访存指令：

①清除累加器指令CLA

T0

T1

T2 0->AC(**由于要严格遵循同步控制的原则，所以即使只有一步微操作也要安排三个节拍。**)

② 累加器取反指令COM

T0

T1

T2 $AC^{取反}$->AC

③算术右移一位指令SHR

T0

T1

T2 L(AC)->R(AC),AC_0->AC_0

④循环左移一位指令CSL

T0

T1

T2 R(AC)->L(AC),AC_0->AC_n

⑤停机指令STP

T0

T1

T2 0->G

3.2）访存指令：

⑥加法指令ADDX

T0 AD(IR)->MAR,1->R

T1 M(MAR)->MDR

T2 (AC)+(MDR)->AC

⑦存数指令STA X

T0 AD(IR)->MAR,1->W

T1 AC->MDR

T2 MDR->M(MAR)

⑧取数指令LDA X

T0 AD(IR)->MAR,1->R

T1 M(MAR)->MDR

T2 MDR->AC

3.3）转移类指令：

⑨无条件转移指令JMP X

T0

T1

T2 AD(IR)->PC

⑩有条件转移（负则转）指令BAN X

T0

T1

T2 A_0Ad(IR)+$A_0^{取非}$ (PC)->PC

执行周期的这十条指令的微操作一定要记牢**这是重点！！！**

4)中断周期：

T0 （SP)-1->MAR,1->W **（硬件关中断）**

T1 PC->MDR

T2 MDR->M(MAR),向量地址->PC **中断隐指令完成**

10.1.3 组合逻辑的设计步骤

1、安排每条指令中微操作的节拍

2、列出微操作命令的操作时间表（具体列出：每个微操作命令和哪些指令的哪些机器周期以及哪些节拍有关）

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	ADD	STA	LDA	JMP
FE 取指	T_0		PC ⟶ MAR	1	1	1	1	1	1
			1 ⟶ R	1	1	1	1	1	1
	T_1		M(MAR) ⟶ MDR	1	1	1	1	1	1
			(PC) +1 ⟶ PC	1	1	1	1	1	1
	T_2		MDR ⟶ IR	1	1	1	1	1	1
			OP(IR) ⟶ ID	1	1	1	1	1	1
		I	1 ⟶ IND			1	1	1	1
		$\overline{\text{I}}$	1 ⟶ EX	1	1	1	1	1	1

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	ADD	STA	LDA	JMP
IND 间址	T_0		Ad (IR) ⟶ MAR			1	1	1	1
			1 ⟶ R			1	1	1	1
	T_1		M(MAR) ⟶ MDR			1	1	1	1
	T_2		MDR ⟶ Ad (IR)			1	1	1	1
		$\overline{\text{IND}}$	1 ⟶ EX			1	1	1	1

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	ADD	STA	LDA	JMP
EX执行	T_0		Ad (IR) ⟶ MAR			1	1	1	
			1 ⟶ R			1		1	
			1 ⟶ W				1		
	T_1		M(MAR) ⟶ MDR			1		1	
			AC ⟶ MDR				1		
	T_2		(AC)+(MDR)→AC			1			
			MDR ⟶ M(MAR)				1		
			MDR ⟶ AC					1	
			0 ⟶ AC	1					

3、写出每一个微操作命令的逻辑表达式并化简

以上图红色的命令**M(MAR)->MDR**为例进行微操作命令的化简：

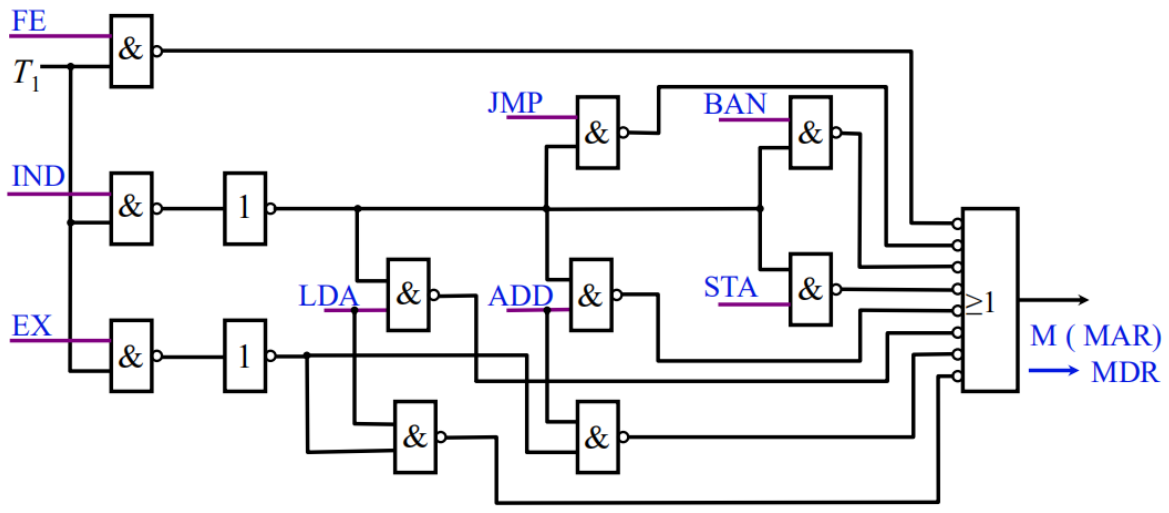
M (MAR) -> MDR

= FE · T1 + IND ·T1 (ADD + STA + LDA + JMP + BAN)+ EX ·T1 (ADD +LDA)

= T1{ FE + IND (ADD + STA + LDA + JMP + BAN) + EX (ADD +LDA) }

（其中FE为取指周期，IND为间址周期，EX为执行周期）

4、画出相应的组合逻辑电路图



由上面的逻辑电路图可以看出，组合逻辑电路设计是非常庞杂的。

组合逻辑电路的特点

1. 思路清晰，简单明了

2. 庞杂，调试困难，修改困难

3. 速度快（RISC采用）

什么是RISC机器？见课本P327

10.2 微程序设计

10.2.1 微程序设计思想的产生（1和2了解即可，3需掌握）

1.采用了“存储逻辑”的设计思想

将控制器所需要的微操作命令，以微代码的形式编成微指令，存在专门的存储器中，执行机器指令时，从该存储器中取出微指令，产生执行机器指令所需的微操作命令序列。

2.采用了“程序设计”的技术

把一条机器指令所需要的微操作命令序列，以微指令的形式编成一段微程序，整个指令系统就编出一整套微程序，采用“顺序、转移、多分支”等程序设计的技术进行微程序的设计。

3.微程序控制器的基本概念

微（操作）命令：构成控制信号序列的最小单位

微操作：由微命令控制实现的最基本操作。

微指令：若干个微命令的组合。

微周期：指从控制存储器中读取一条微指令并执行相应的微操作所需的时间。

微程序：一系列微指令的有序集合。

控制存储器：存放微程序的只读存储器。（**微程序控制单元的核心部件。**）

10.2.2 微程序控制单元框图及工作原理

1.机器指令对应的微程序

1)由于任何一条机器指令的取指操作是相同的，因此将取指指令统一编写成一个微程序。

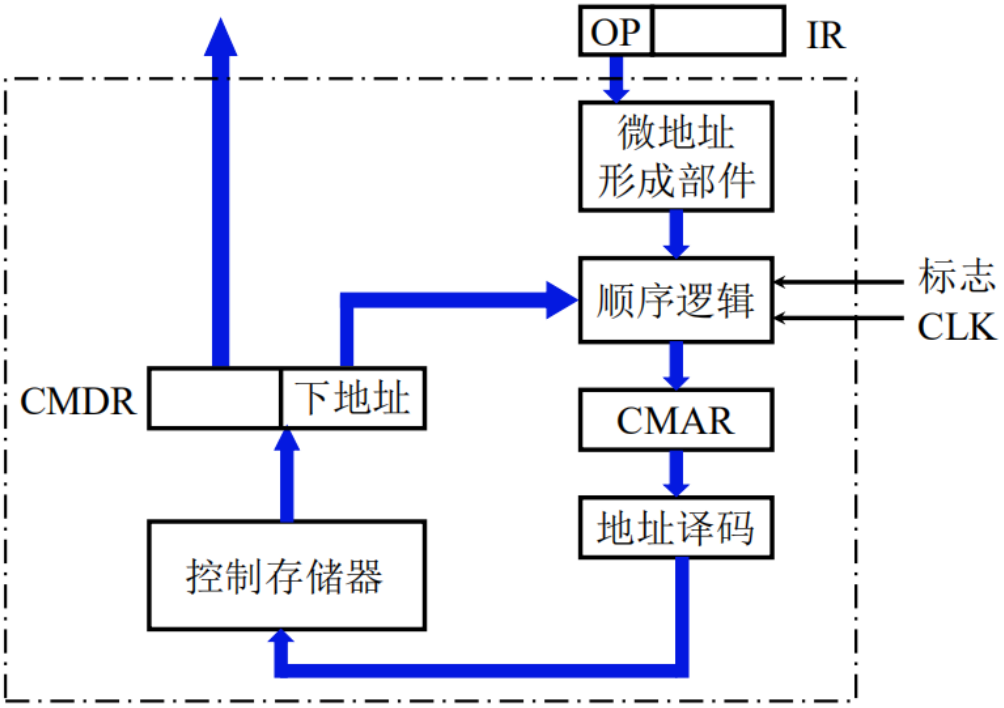
2)指令如果是间址，其操作也是可以预测的，也可以编写对应间址周期的微程序。

3)当然中断隐指令可由一个对应中断周期的微程序表示。

4)这样控制存储器中的微程序个数应该为机器指令数加上取指周期，间址周期和中断周期3个微程序。

2.微程序控制单元的基本框图

1）框图



图中的控制储存器是微程序控制单元的核心部件，用于储存全部的微程序。
CMAR是控存地址寄存器，用来存放欲读出的微指令地址。
CMDR是控存数据寄存器，用来存放从控存读出的微指令。
2）微指令的基本格式：
微指令的基本格式，共分为两个字段，一个是操作控制字段，用于发出各种控制信号。一个是顺序控制字段，可以指出下一条微指令的地址（**简称下地址**）。
3）工作原理：
位程序控制单元，通过逐条取出微指令，发出各种微操作命令，从而实现从主存逐条取出，分析并执行机器指令，以达到运行程序的目的。

10.2.3 微指令的编码方式

1.直接编码（直接控制）方式
原理：在微指令的操作控制字段中，每一位代表一个微操作命令。
特点：速度快，但是可能造成控存容量大。
2.字段直接编码方式（显式编码）
原理：将微指令的操作控制字段分成若干段，将一组互斥的微操作命令放在一个字段内。
特点：以较少的二进制信息表示较多的微操作命令信号，执行速度较慢。
3.字段间接编码方式（隐式编码）
原理：一个字段中的某些微指令还需要另一个字段中的某些微指令来解释。
特点：进一步缩短了微指令的字长，但是削弱了微指令的并行控制能力。
4.混合编码
将直接编码和字段编码混合使用。

5.其他：

比如：设置常数字段……

10.2.4 微指令序列地址的形成

1.直接由微指令的下地址字段指出：（断定方式）

2.根据机器指令的操作码形成

微指令的地址由操作码经微地址形成部件形成。**微地址形成部件**实际上是一个编码器，其输入是指令操作码，输出是对应该机器指令微程序的首地址。

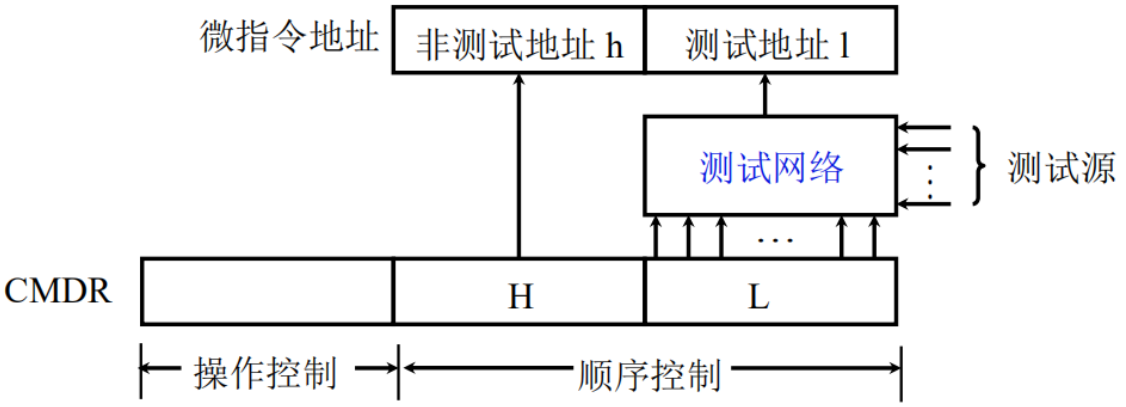
3.增量计数法

要求微指令的地址是连续的。

4.分支转移法

当遇到条件转移指令时，下一条微指令的地址由该方式形成。

5.通过测试网络形成

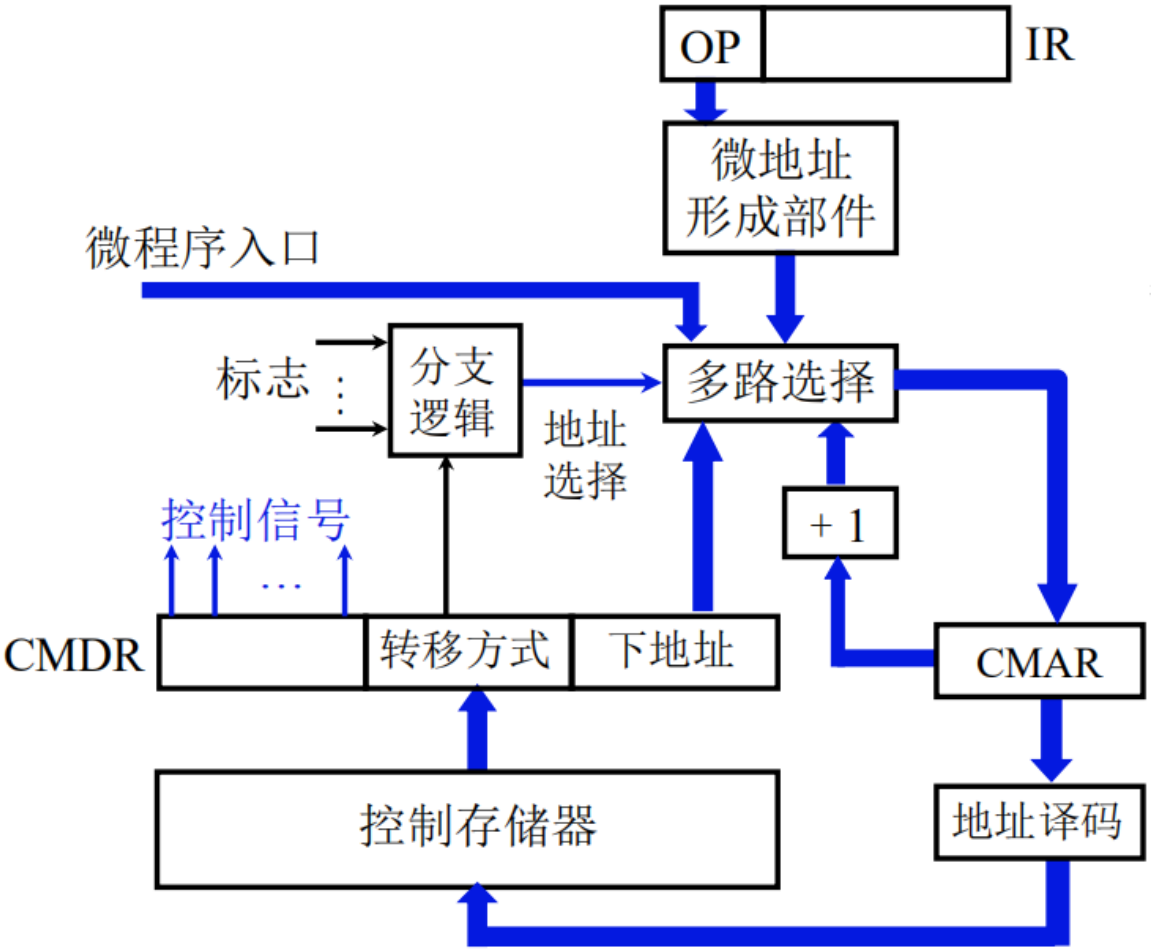


6.由硬件产生微程序的入口地址

第一条微指令地址由专门 硬件 产生。

中断周期：由 硬件 产生 中断周期微程序首地址。

综上：后续微指令地址形成方式原理图



10.2.5 微指令格式

1.水平型微指令

一次能定义并执行多个并行操作。

比如：直接编码、字段直接编码、字段间接编码、直接和字段混合编码。

2.垂直型微指令:

类似机器指令操作码的方式，由微操作码字段规定微指令的功能.

3.两种微指令格式的比较:

(1) 水平型微指令比垂直型微指令并行操作能力强，灵活性强

(2) 水平型微指令执行一条机器指令所要的微指令数目少，速度快

(3) 水平型微指令用较短的微程序结构换取较长的微指令结构

(4) 水平型微指令与机器指令差别大

10.2.6 静态微程序设计和动态微程序设计

静态 微程序无须改变，采用 ROM

动态 通过 改变微指令 和 微程序 改变机器指令，有利于仿真，采用 EPROM

微程序设计举例

具体步骤：

1.写出对应机器指令的微操作及节拍安排

2.确定微指令格式

3.编写微指令码点

补充章节：数字逻辑

1. 基本逻辑关系

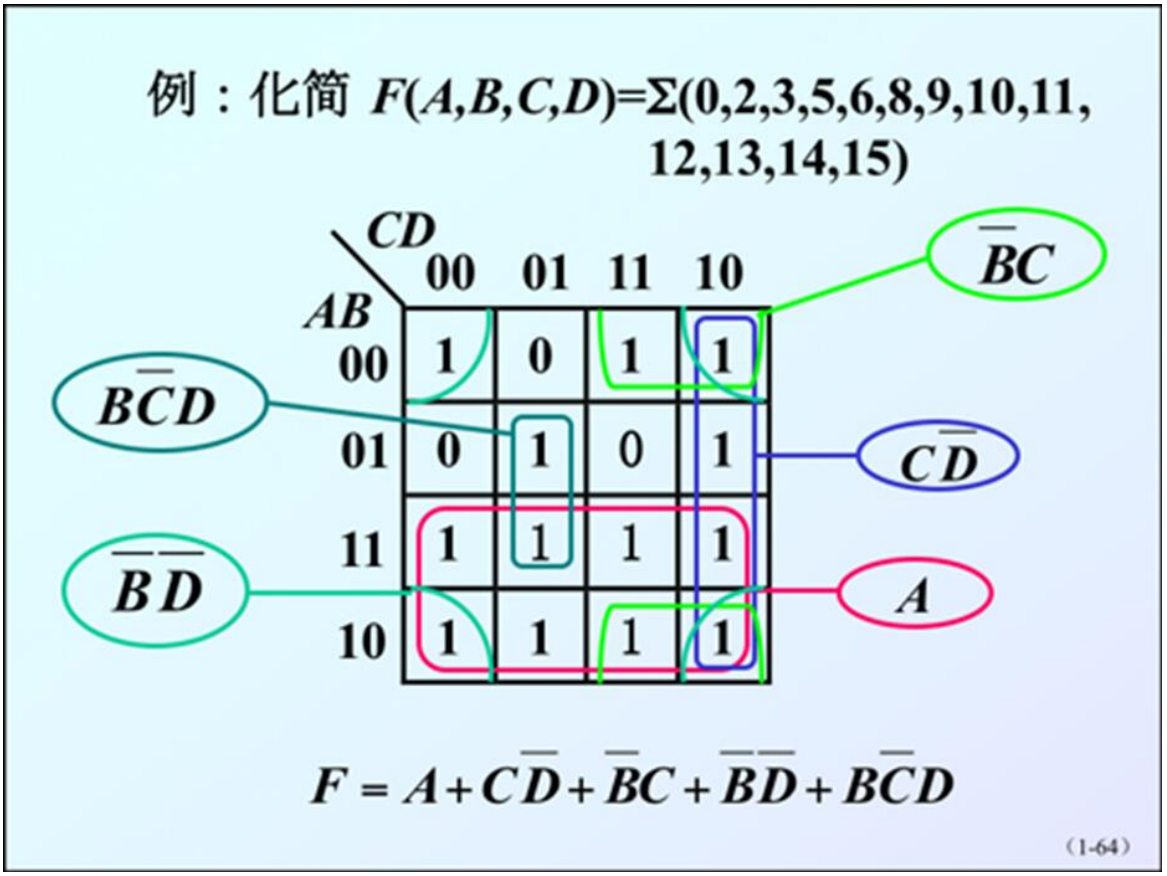
- 与（•）逻辑乘法
- 或（+）逻辑加法
- 非

2. 最小项

- 定义： 在一个具有n个变量的逻辑函数中，如果一个与项包含了所有n个的变量，而且每个变量都是以原变量或反变量的形式作为一个因子仅出现一次，那么这样的与项就称为该逻辑函数的一个最小项。对于n个变量的全部最小项共有2^n个。
- 最小项编号：为了表达方便，人们通常用mi表示最小项，其下标i为最小项的编号。编号的方法是：最小项中的原变量取1，反变量取0，则最小项取值为一组二进制数，其对应的十进制数便为该最小项的编号。

3. 卡诺图化简（详见上课时 数字逻辑1 课件后半部分 一言难尽提一提重点）

1. 逻辑相邻：相邻单元输入变量的取值只能有一位不同
2. 若两个最小项中只有一个变量以原、反状态相区别，则称它们为逻辑相 邻。逻辑相邻的项可以合并，消去一个因子。
3. 相邻单元的个数是2^N个，并组成矩形时，可以合并。
4. 先找面积尽量大的组合进行化简，可以减少更多的因子。
5. 各最小项可以重复使用。
6. 所有的1都被圈过后，化简结束。
7. 化简后的逻辑式是各化简项的逻辑和。
8. 圈的数目越少越简；圈内的最小项越多越简。
9. 卡诺图中所有的 1 都必须圈到，不能合并的 1 必须单独画圈。
   课件上例题：山软智库不拥有此版权，向原作者致敬！


例：化简 F(A,B,C,D)=Σ(0,2,3,5,6,8,9,10,11,12,13,14,15)
CD
AB
00 01 11 10
00 1 0 1 1
01 0 1 0 1
11 1 1 1 1
10 1 1 1 1
$\overline{B}C$
$B\overline{C}D$
$C\overline{D}$
$\overline{B}\,\overline{D}$
A
$F = A + C\overline{D} + \overline{B}C + \overline{B}\,\overline{D} + B\overline{C}D$
(1-64)