计算机组成原理

数字

|  |  |
| --- | --- |
| 编 制 人： | Lizy |
| 发布日期： | 2023.3.15 |

更改记录

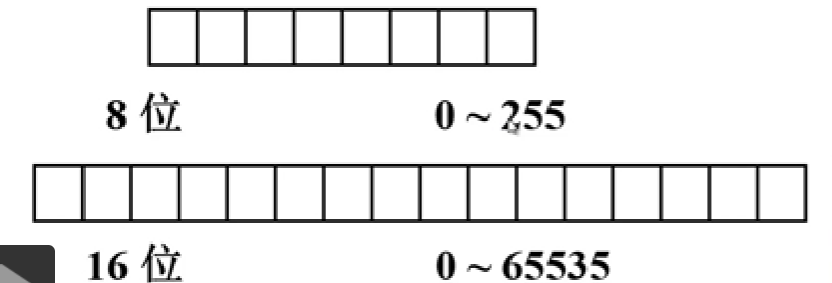
|  |  |
| --- | --- |
| **发布日期** | **更改描述** |
| 20230303 | 第一版发布 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# 计算机的运算方法

## 无符号数和有符号数

### 无符号数

只有数值。保存在寄存器中，寄存器的位数反映了无符号数的表示范围。



### 有符号数

有正负号的数。包括符号和数值2部分。

#### 机器数与真值

1. 定义：

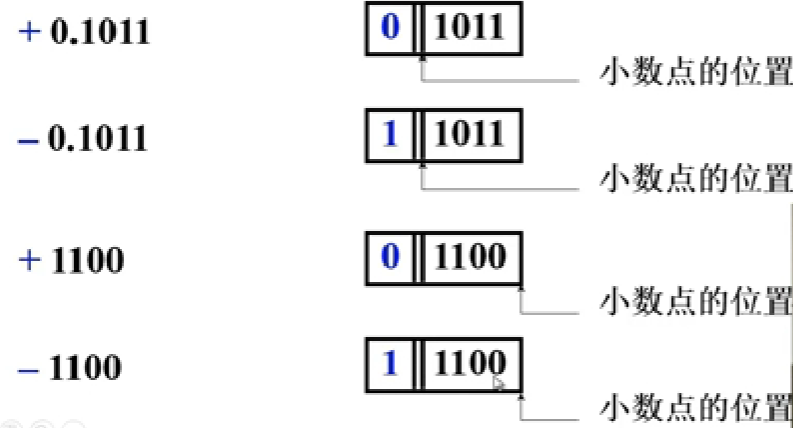
保存在计算机中数叫做**机器数**；

实际的值为**真值**；

1. 符号位：

0：正值；1：负值；

1. 小数点：以约定方式给出



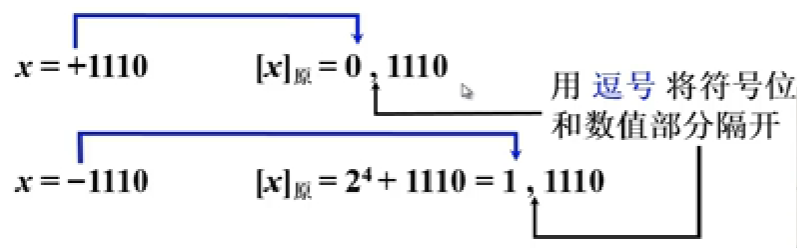
1. 定点计算机：小数点的位置是**固定**的计算机；
2. 定点计算机分为2类：

* 小数定点器：小数点在符号位后面；
* 整数定点器：小数点在整数后面；

#### 原码表示法

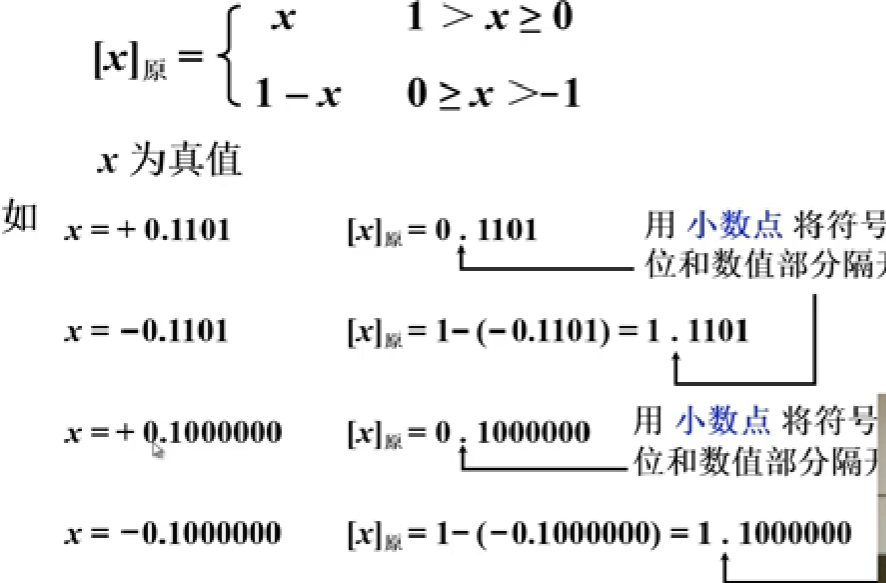
1. 定义：符号位数字化；**带符号的绝对值**；

* 整数：用“**，**”将符号位和数值隔开



* 小数：用“**.**”将符号位和数值隔开

真值范围:**（0.a ~ (-0.a)）**

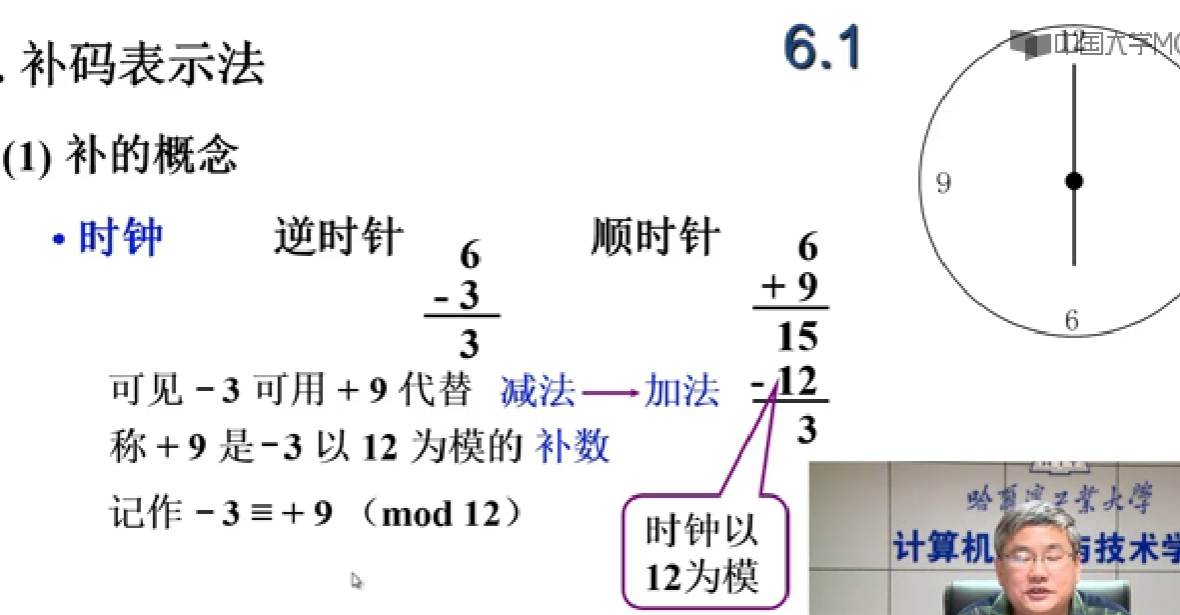


1. 特点：简单、直观，但是做加法时也要用减法操作；

#### 补码表示法

1. 补的概念：

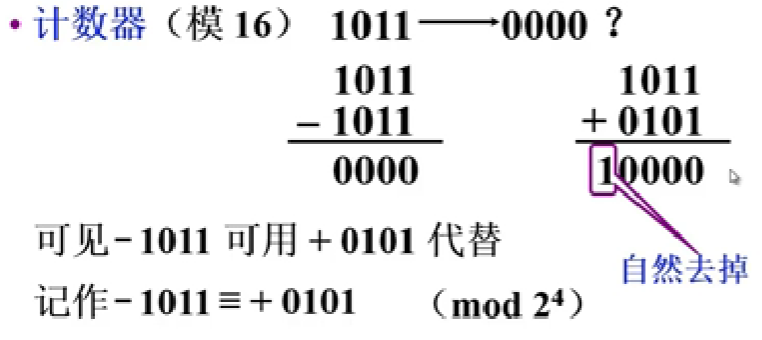
+9是-3以12为模的补数；



1. 结论：

一个负数加上“模”即得该负数的补数；

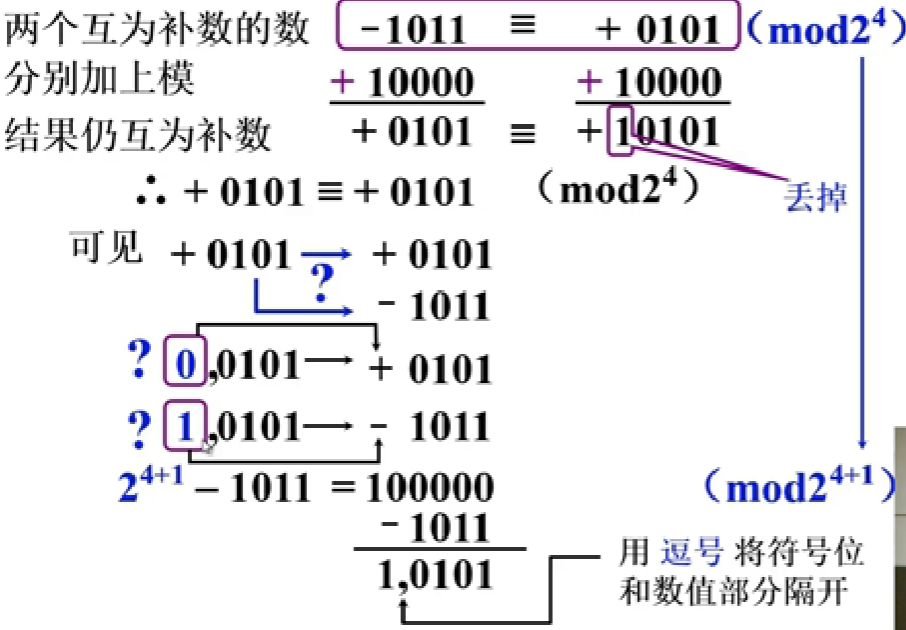
一个正数和一个负数互为补数时，他们的绝对值之和即为“模数”；



1011 + 0101 = 10000；

寄存器为4位，则其模为24（16即10000）；

1. 正数的补数即为其本身：

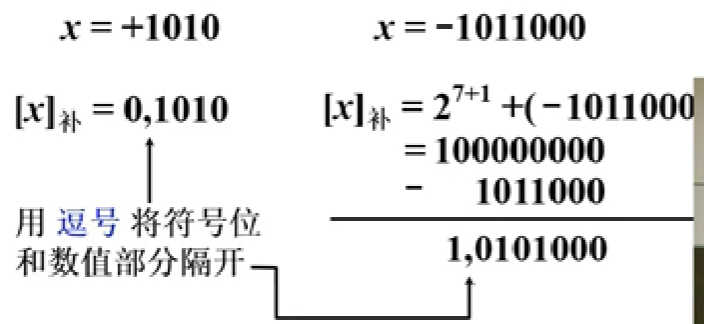


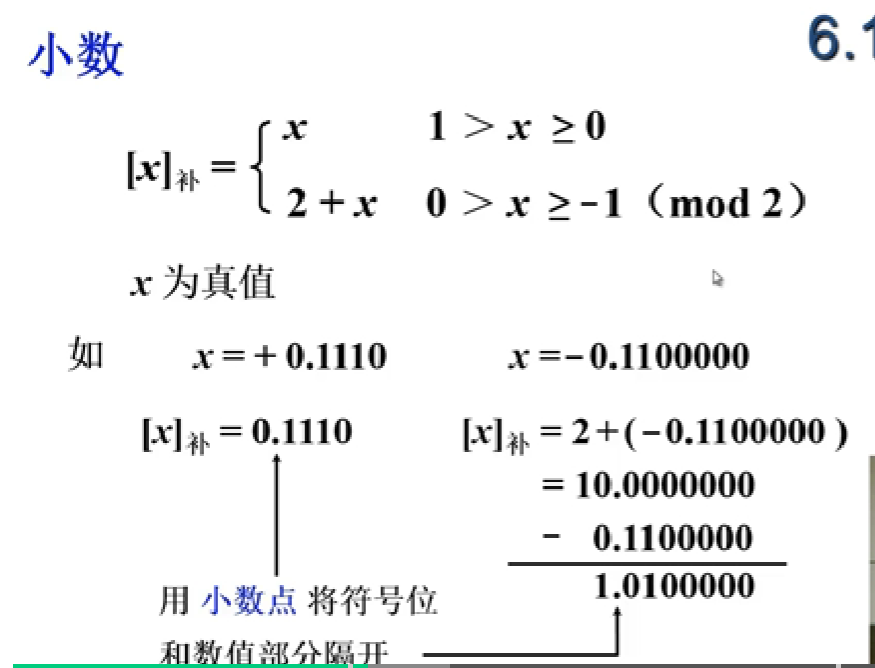
1. 补码定义：

正数的补码就是其本身；

负数的补码等于：符号位 |（2n+1 + 负数）；其中n为数值位数

这里“模”是**n+1**次方





小数的“模”是**2**；

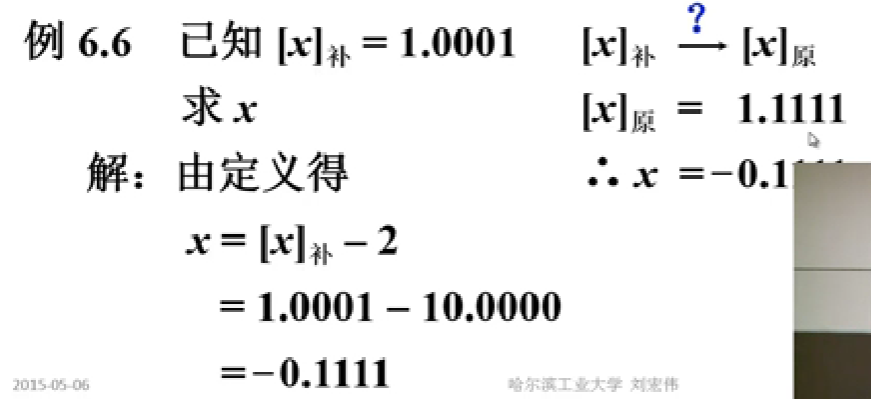
负数的补码：**2 + 负数；**

1. 求补码的快捷方式

**负数**的补码：原码符号位不变，数值位**取反+1**；小数也同样适用；

设x = -1010

* 数值取反+1 ： 0101 + 1 = 0110
* 添加符号位：10110



1. 真值为**负数**时：**补码 --> 原码：**符号位不变，数值取反 + 1

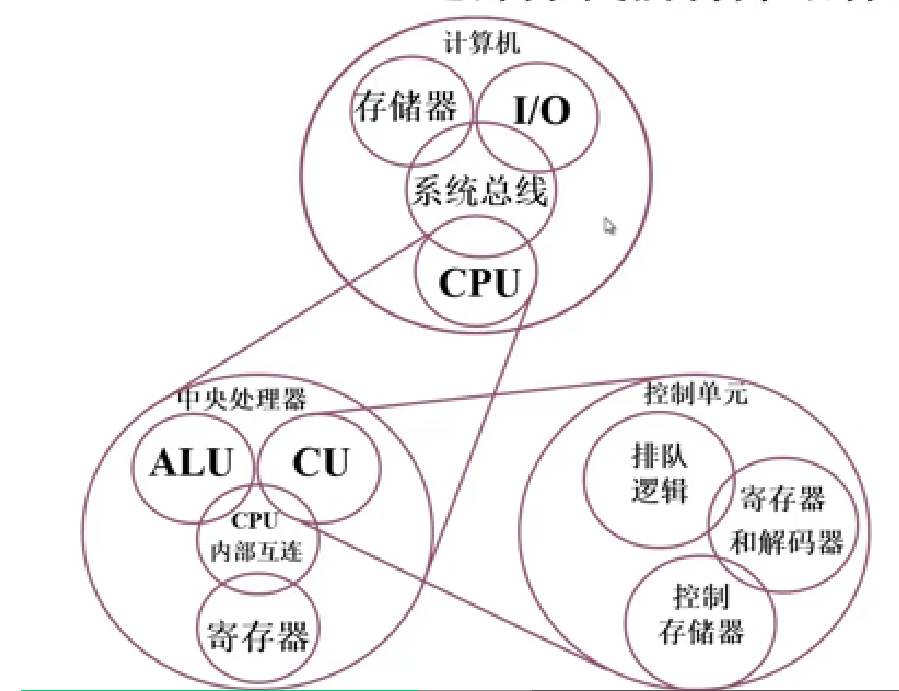
#### 反码表示法

## 数的定点

…..

**第三部分软硬件接口 CPU的控制器CU**

# 指令系统





高级语言->机器指令（0 1代码）后，才能由计算机硬件按序执行；

**机器指令**：计算机系统的CPU可以**直接识别**并**执行**的操作命令；

**指令集**：一个机器可以处理的所有**机器指令构成的集合**；

**指令集**就是计算机系统内**软件和硬件的交界面**，软件通过指令系统告诉计算机硬件做什么操作；硬件通过指令系统把运算结果和硬件状态返回给软件；

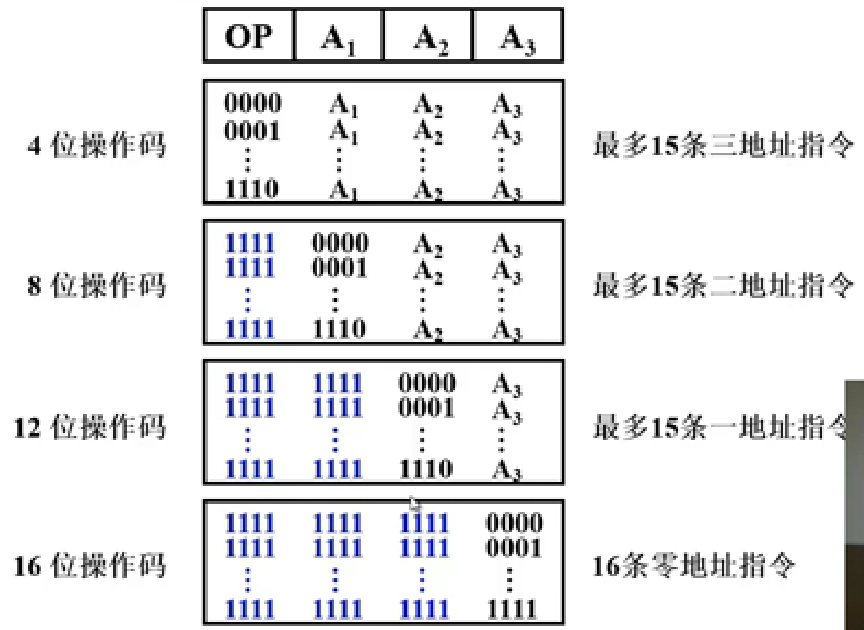
## 机器指令

### 指令格式



1. **操作码**：反应机器做什么操作（有的包括了对什么数据进行操作和寻址方式等）；

* 长度固定：用于指令字长较长的情况，RISC,固定操作码长度是为了方便译码；如IBM 370，操作码8位；
* 长度可变：操作码分散在指令字的不同字段中；如x86；
* **扩展**操作码技术：一种指令优化技术，即让操作码的位数随地址数的减少而增加，不同地址数的指令可以具有不同的操作码长度：

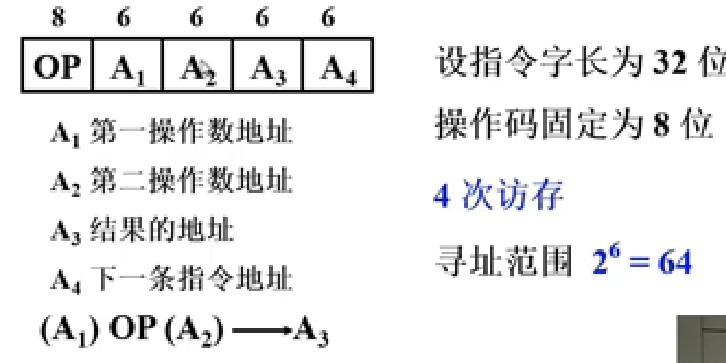


原则：短操作码一定不能是长操作码的前缀，否则对操作码的译码很难识别出来；

一般，高频出现的指令用短操作码，不常出现的指令用长操作码；

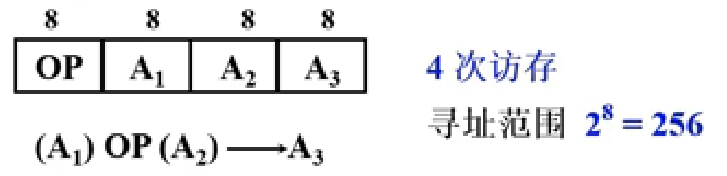
1. **地址码**：

* 四地址：如果地址字段均指示主存的地址，则完成一条四地址指令，共需访问4次存储器：取指令一次，取2个操作数两次，存放结果一次；



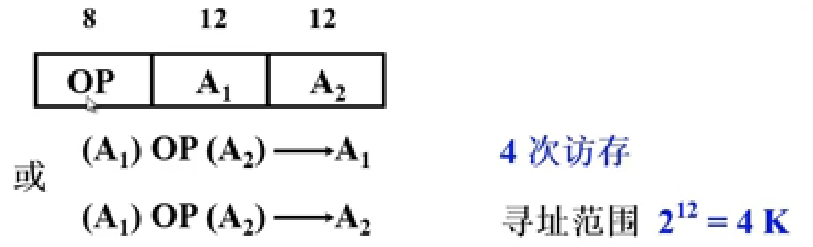
缺点：四地址的寻址范围很小，如何减少地址码个数增加地址码字段长度，从而增加寻址范围；用PC代替A4；

* 三地址：用PC代替A4，减少了地址码个数，增加了地址码字段长度，增加了寻址范围；

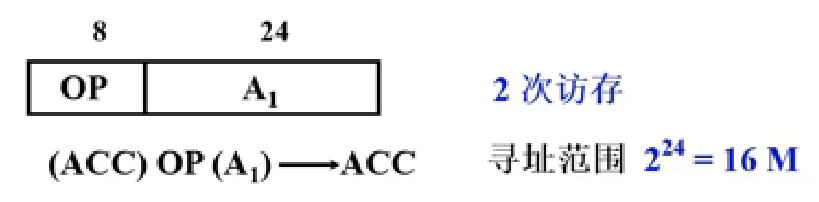


改进：进一步减少地址码个数，增加地址码字段长度，用A1或A2代替A3；将越运算结果不保存在A3中，而是保存在某个参加运算的原操作数地址中；

* 二地址：将运算结果保存在原操作数地址中；



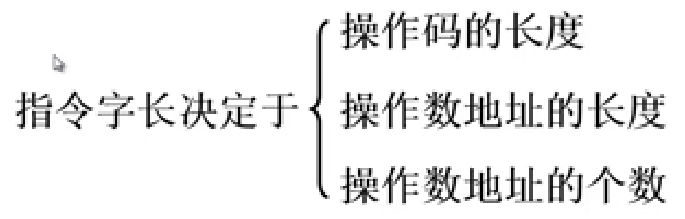
* 若将运算结果存放在一个指定的寄存器（如ACC）中可以降低访存次数，若将某个原操作数直接放在寄存器中，运算后的结果也放在该寄存器中，可以进一步降低访存次数、扩大寻址范围，如ACC代替A1或A2；
* 一地址：寄存器ACC代替A1或A2，运算结果放在ACC中；



寻址范围大大增加；

* 零地址：无地址码

### 指令字长



1. **指令字长固定**：

指令字长 ≤ 存储字长；

1. **指令字长可变**

按字节的倍数变化

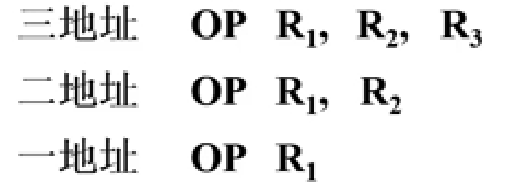
### 小结

1. 当用一些硬件资源代替指令字中的地址码字段后：

* 可扩大指令的寻址范围；
* 可缩短指令字长；
* 可减少访存次数；

1. 当指令的地址字段为寄存器时：

* 可缩短指令字长；
* 指令执行阶段不访存；



## 操作数类型和操作种类

### 操作数类型

操作数：被操作码操作的数；

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 逻辑运算

### 数据在存储器中的存放方式（字节编址）（操作数）

1. 存放方式

* 大端：高字节先存



* 小端：低字节先存



1. 机器字长和存储字长：

* 机器字长：CPU一次能处理的二进制位数，一般等于内部存储器的位数，如32位机器、64位机器；（每个机器的机器**字**长可以不一样，与架构有关，DSP的字长16位）；
* 存储字长：一个存储单元存储的一串二进制的位数；

1. 存储字长64位（双字），机器字长32位时：

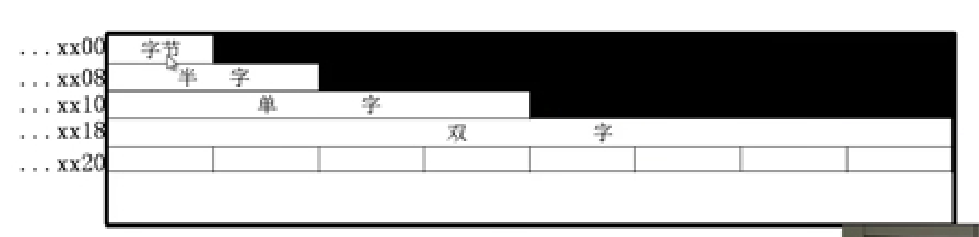
* 从任意位置开始存储：



优点：不浪费存储资源；

缺点：除了访问一个字节之外，访问其他类型的数据都可能花费两个存储周期的时间。读写控制比较复杂；判断是否跨存储字；

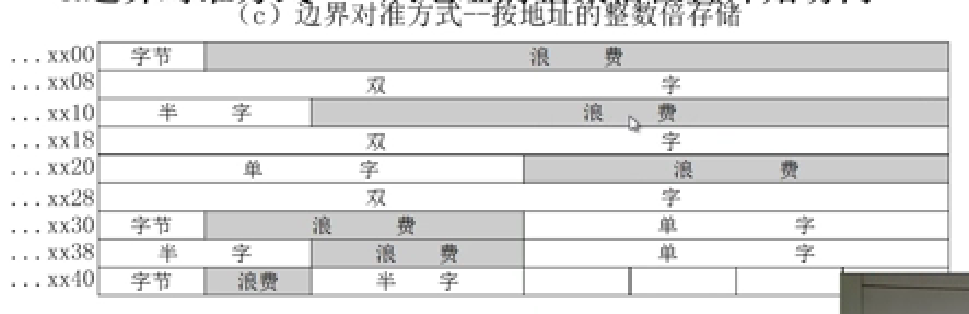
* 从一个存储字的起始位置开始存储：



优点：无论访问何种类型的数据，在一个周期内均可完成，读写控制简单；

缺点：内存浪费太大；

* 边界对准方式——从地址的整数倍位置开始访存



边界对准：数据存放的起始地址是数据类型长度的整数倍；

前2中方案的折中，在一个周期内既可以完成存储访问，空间浪费也不太严重；

### 操作类型（操作码）

1. 数据传送

源 寄存器 寄存器 存储器 存储器

目的 寄存器 存储器 寄存器 存储器

例如 MOVE STORE/MOVE/PUSH LOAD/MOVE/POP MOVE

置“1”，清“0”；

1. 算数逻辑操作

算数：加、减、乘、除、增1、减1、求补、浮点运算、、、；

逻辑：与、或、非、异或、位操作、、、；

1. 移位操作

算数移位 逻辑移位

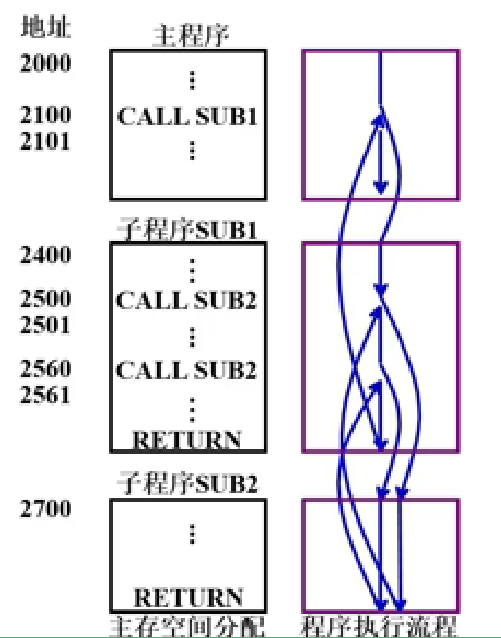
循环移位（带进位和不带进位）

1. 转移

* 无条件转移 JMP
* 条件转移 分支指令
* 结果为零转 （Z = 1）JZ
* 结果溢出转 （O = 1）JO
* 结果有进位转 （C = 1）JC
* 跳过一条指令 SKP



* 调用和返回



* 陷阱（Trap）与陷阱指令

意外事故的中断：非法、越界、除数为0。。。

* 一般不提供给用户直接使用

在出现事故时，由CPU自动产生并执行（隐指令、硬件实现）；

* 设置供用户使用的陷阱指令

OS提供用户使用的陷阱指令，完成系统调用(软中断、调试时）

1. 输入输出

如果外设按统一编址方式，可以用访问存储器的方式（MOVE）访问外设，不用输入输出指令；

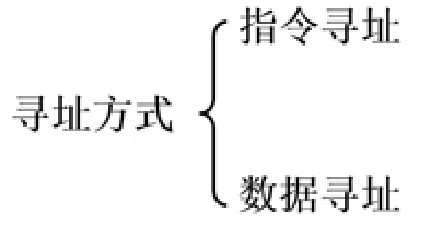
如果外设/端口是单独编址，则需要输入输出指令；

入 端口中的内容——>CPU的寄存器 eg: IN AL , n

出 CPU的寄存器——>端口寄存器 eg: OUT n, AL

## 寻址方式

寻址方式：确定本条指令的操作数地址、下一条要执行指令的指令地址；

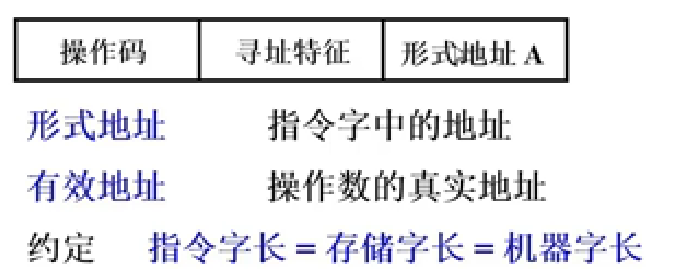


### 指令寻址

1. **顺序寻址** （PC）+ **1** ——> PC
2. **跳跃寻址** 由转移指令指出下一条指令的地址

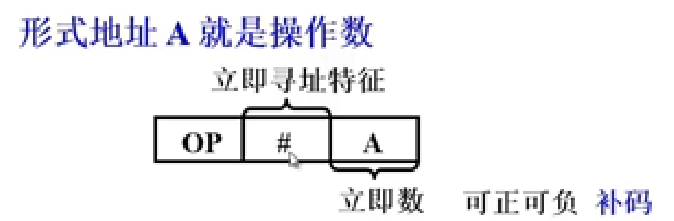


### 数据寻址



1. 立即寻址

形式地址A就是操作数；直接参与操作码的运算，其格式为：

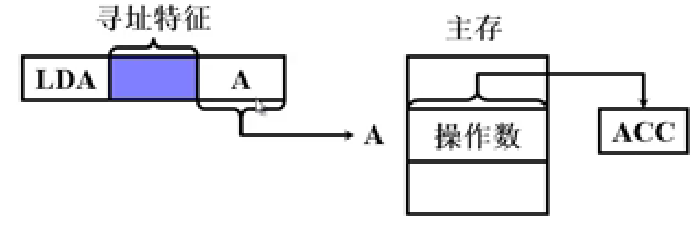


指令执行阶段不访存；

A的位数限制了立即数的范围；

1. 直接寻址

有效地址 = 形式地址（EA = A） A就是操作数在内存单元的实际地址；



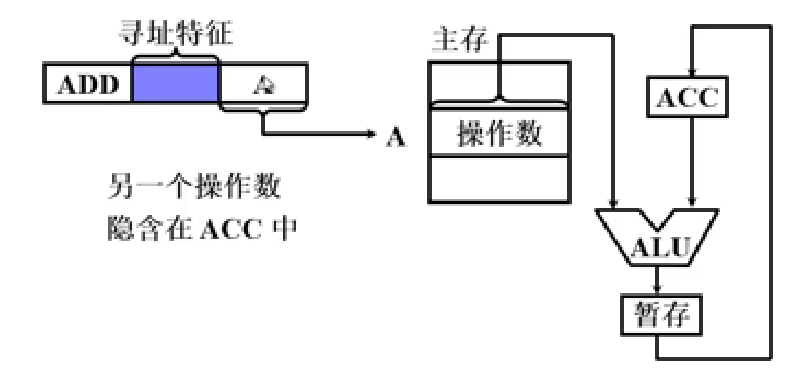
执行阶段访问一次存储器；

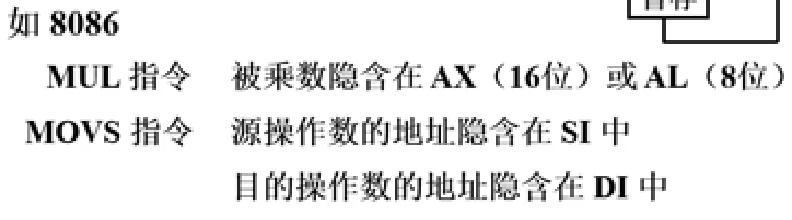
A的位数决定了该指令操作数的寻址范围；

操作数的地址不易修改（必须修改A）；

1. 隐含寻址

操作数地址隐含在操作码中；

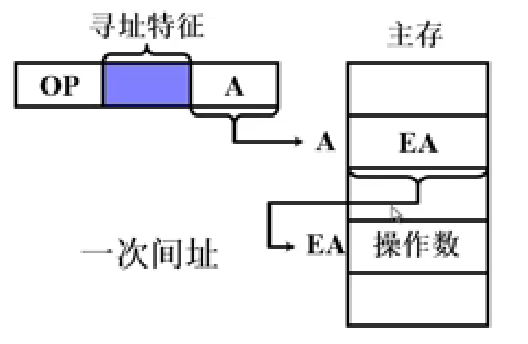




指令字中少了一个地址字段，可缩短指令字长；

1. 间接寻址

数据在内存中的有效地址由形式地址间接提供；EA = （A）；

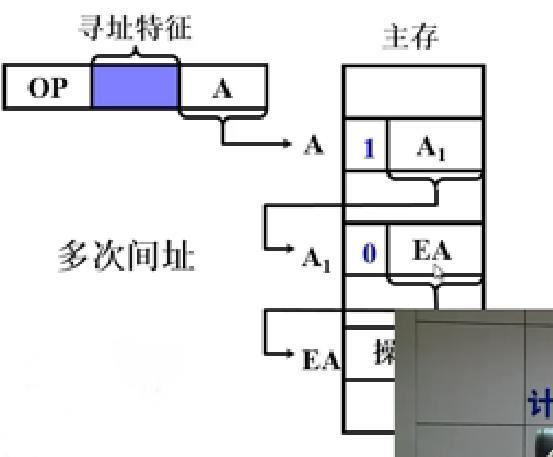


一次间址时：

指令执行阶段2次访存；

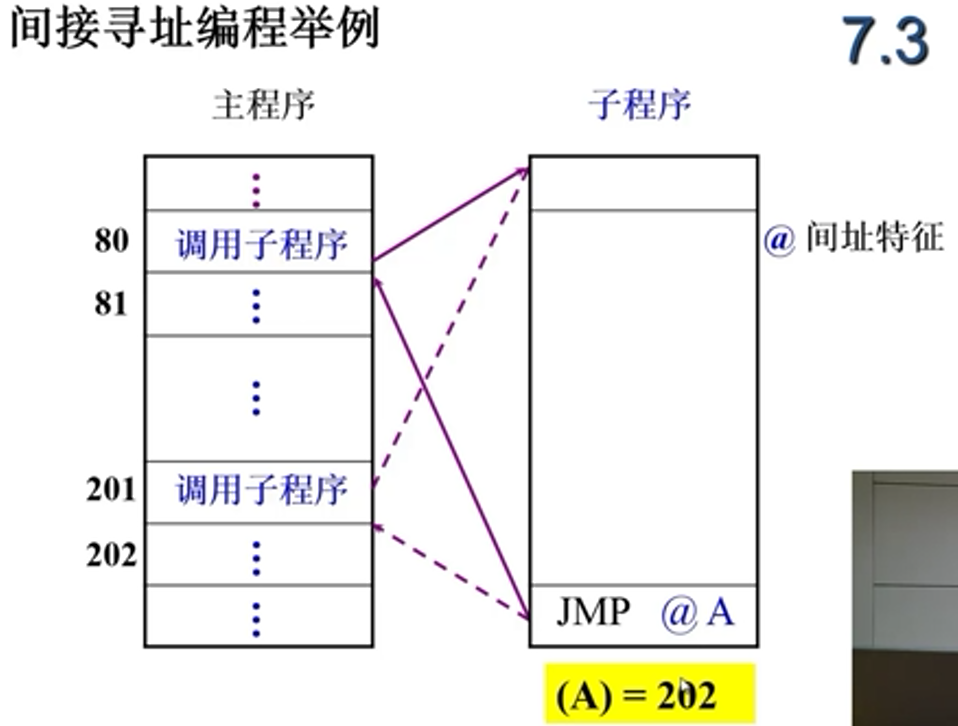
可扩大寻址范围；

便于编制程序；



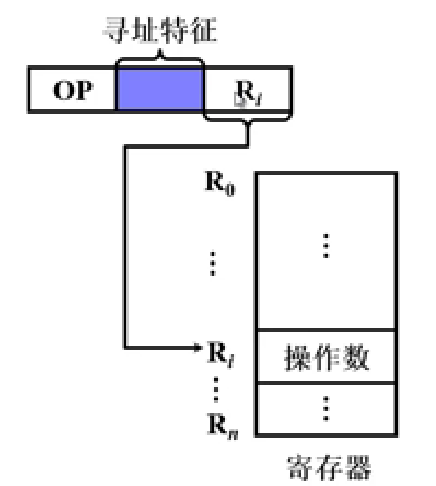
多次间址：

指令执行阶段多次访存；



1. 寄存器寻址

有效地址即寄存器编号，EA = Ri；



执行阶段不访存，只访问寄存器，执行速度快；

寄存器个数有限，可缩短指令字长；

1. 寄存器间接寻址

数据在内存中的有效地址由寄存器间接提供，EA = （Ri），操作数的地址保存在寄存器中；



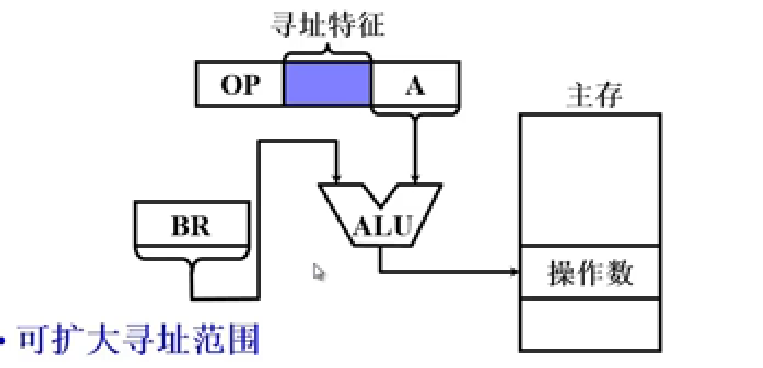
有效地址在寄存器中，操作数在存储器中，执行阶段需要访存储器；

便于编制循环程序；

1. 基址寻址

* 采用**专用寄存器**做基址寄存器；

EA = (BR) + A BR为基址寄存器



可扩大寻址范围；

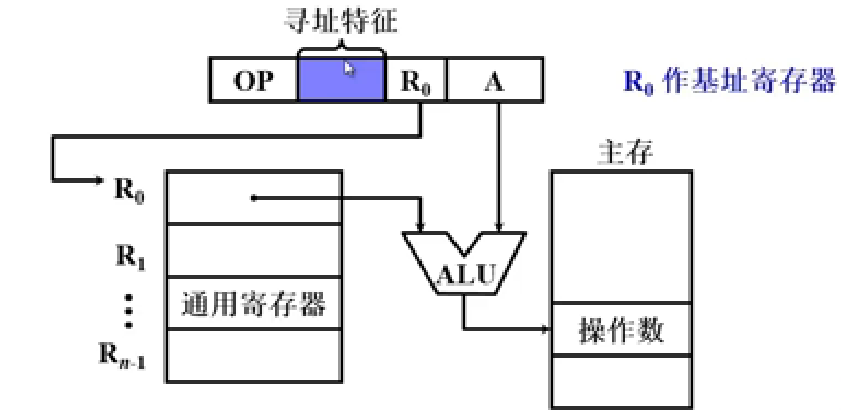
**有利于多道程序；（程序动态定位）**

BR内容由操作系统或管理程序确定；

在程序的执行过程中**BR内容不变**，形式地址A可变；（基址地址不变，相对位置改变，如，外扩FLASH）

* 采用**通用寄存器**做基址寄存器

R0是寄存器编号，用来存放基址；



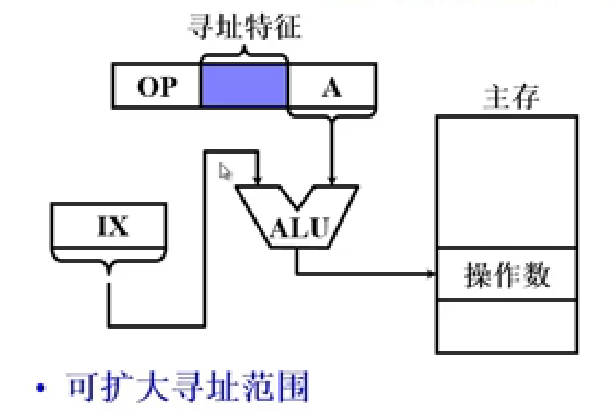
由用户指定哪个通用寄存器作为基址寄存器；

基址寄存器的内容由操作系统确定；

在程序的执行过程中**R0内容不变**，形式地址A可变；

1. 变址寻址

EA = (IX) + A IX为编址寄存器（专用），通用寄存器也可作为编址寄存器；



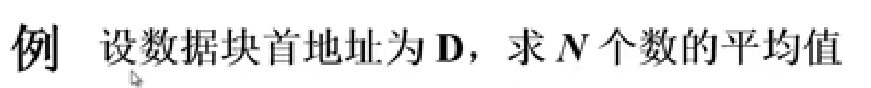
可扩大寻址范围；

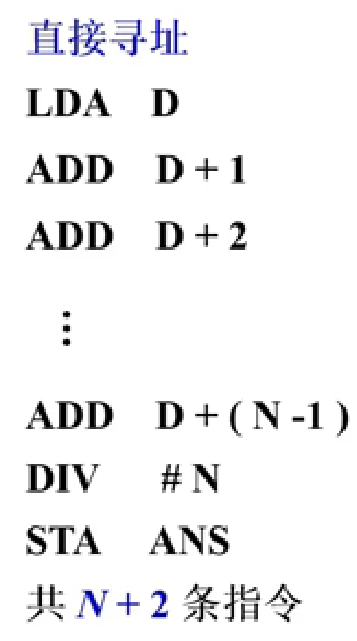
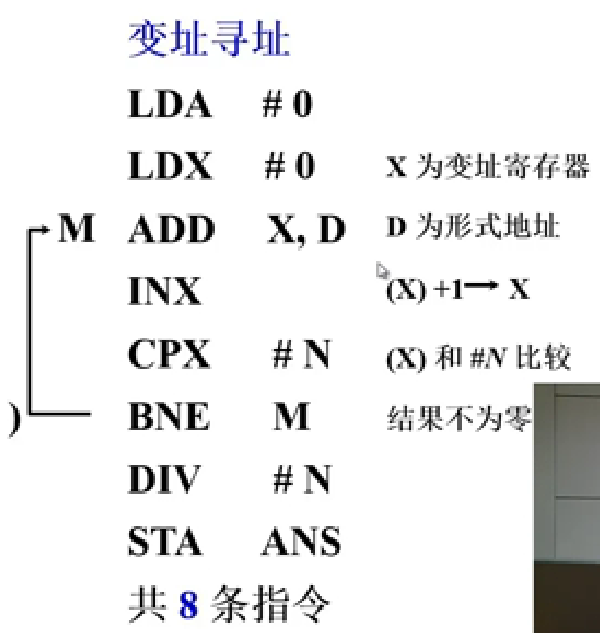
IX的内容由**用户**给定；

在程序的执行过程中**IX内容可变**，形式地址A不变；（相对位置不变，基址地址改变，如，内存中不同进程的基地址）

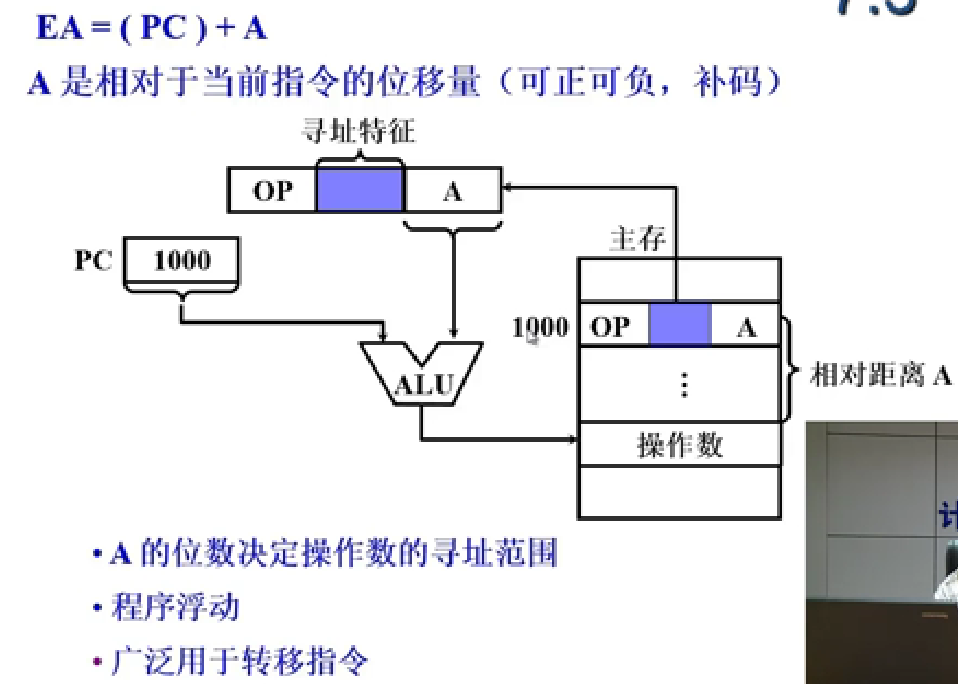
便于处理数组问题；

实例：



1. 相对寻址

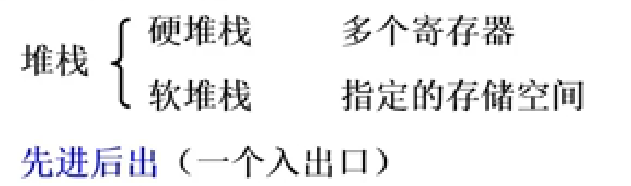


实例：

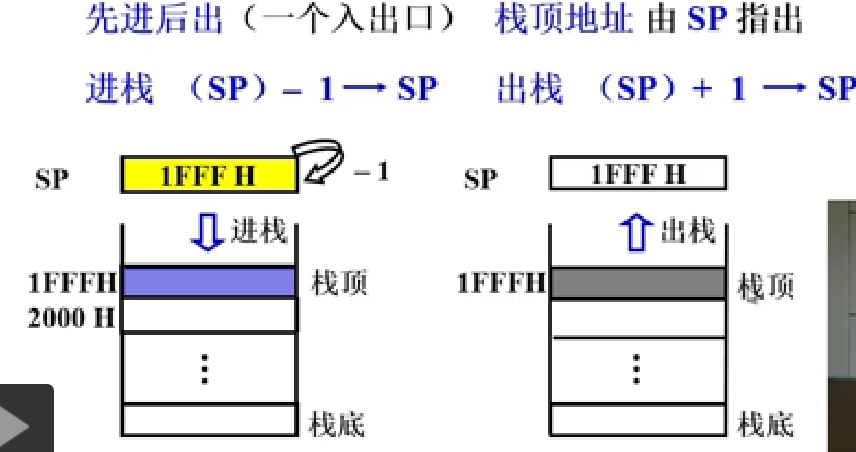


1. 堆栈寻址

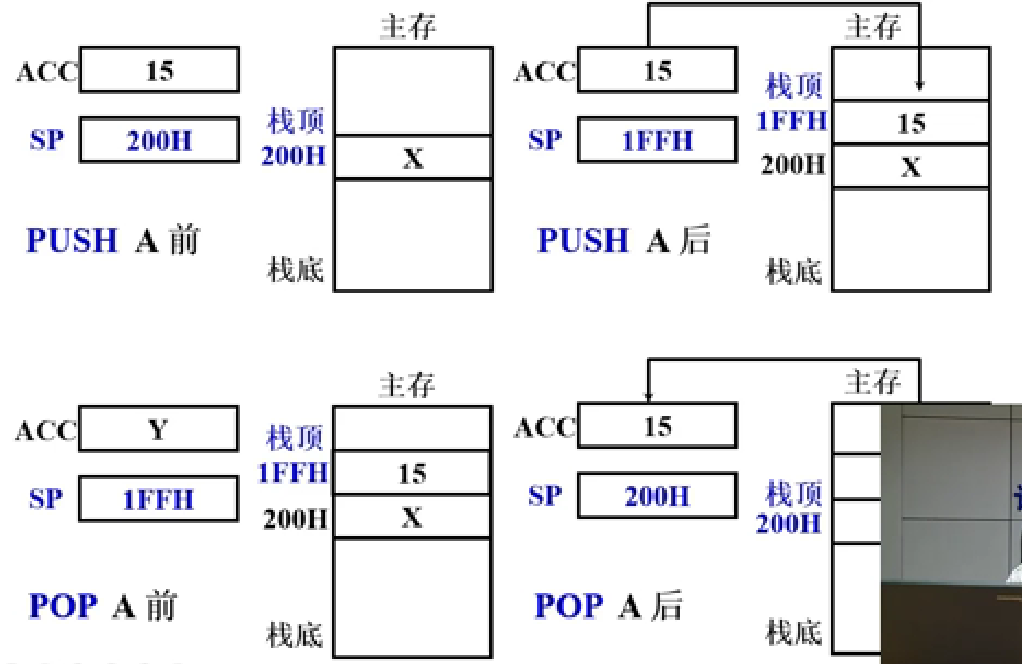
* 堆栈特点：先进后出；



栈顶地址小于栈底地址（栈底地址大，倒扣的桶）



* 举例：



* SP的修改与主存编址方法有关



## 指令格式举例

### 设计指令格式时应考虑的各种因素

1. 指令系统的兼容性；
2. 其他因素

操作类型 包括指令个数及操作的难易长度

数据类型 确定哪些数据类型可参与操作

指令格式 指令字长是否固定

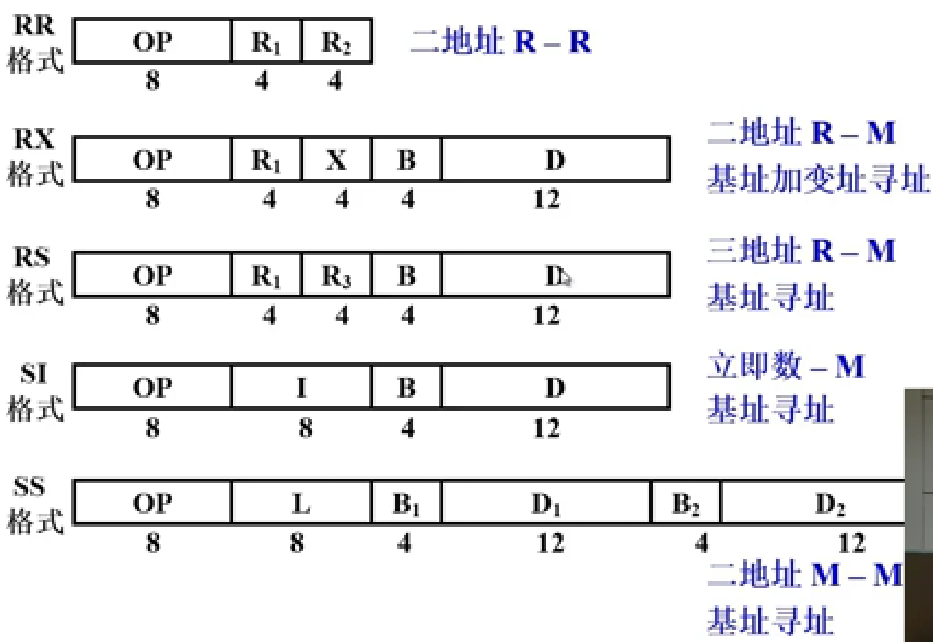
操作码位数、是否采用扩展操作码技术

地址码位数、地址个数、寻址方式类型

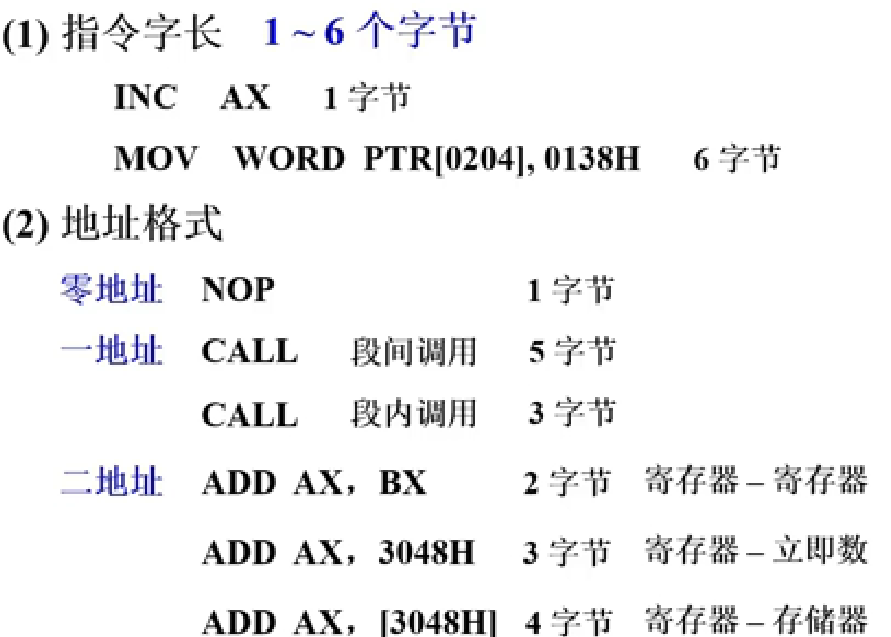
寻址方式 指令寻址、操作数寻址

寄存器个数 寄存器的多少直接影响指令的执行时间

### IBM 360 指令格式



### Intel 8086 指令格式——复杂指令集

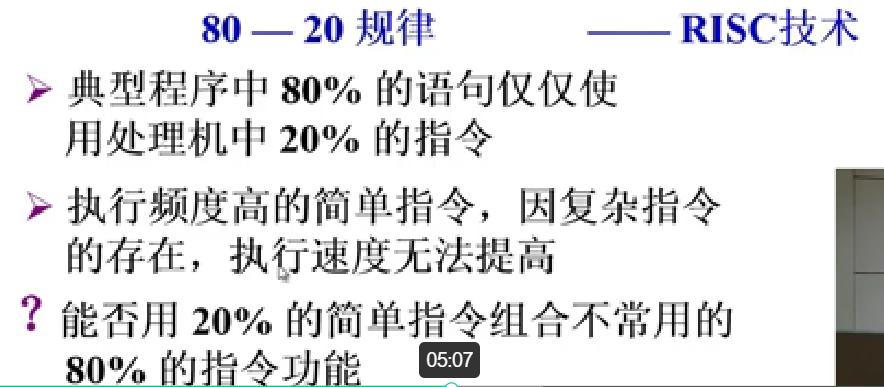


## RISC技术

### RISC的产生和发展

RISC（Reduced Instruction Set Computer，精简指令集）

CISC（Complex Instruction Set Computer，复杂指令集）



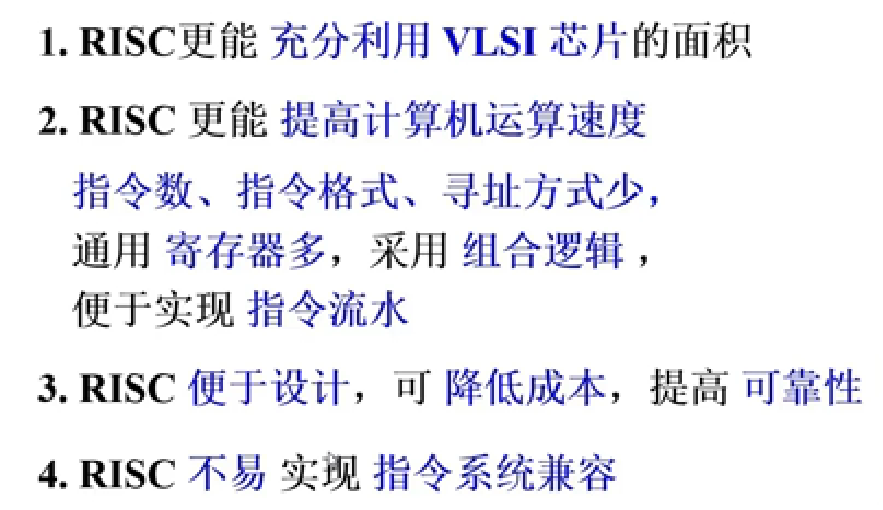
### RISC的主要特征

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，译码简单；
3. 只有LOAD/STORE指令**访存**；
4. CPU中有多个通用寄存器；
5. 采用**流水**技术、**一个时钟周期**内完成一条指令；
6. 采用组合逻辑（硬件）实现控制器，提高速度；

### CISC的主要特征

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，导致译码复杂，CPU硬件部分也很复杂；
3. 访存指令不受限；
4. CPU中设有专用寄存器；
5. 大多数指令需要**多个**时钟周期执行完毕；
6. 采用微程序控制器（因为用硬件方式太过复杂）；

### CISC和RISC的比较



按照采用的指令集，CPU可以分为**复杂指令集（CISC）和精简指令集（RISC）**两大类。x86架构是复杂指令集的代表，而Arm架构、MIPS架构、Alpha、PowerPC架构等是精简指令集的代表，各架构特点比较如下：



# CPU的结构和功能

## CPU的结构

### CPU的功能

CPU包括控制器和运算器；

1. 控制器的功能

* **取指令**：从内存中将指令取出；——指令控制
* **分析指令**：对操作码进行译码；——操作空座
* **执行指令，发出各种操作命令：如在内存或寄存器中取操作数，进行**运算，将结果存储在内存或寄存器中；——时间控制
* 控制程序输入及结果的输出；
* 总线管理
* 处理异常情况和特殊请求；——处理中断

1. 运算器的功能

实现算数运算和逻辑运算；——数据加工

### CPU结构框图

总线按位置分：

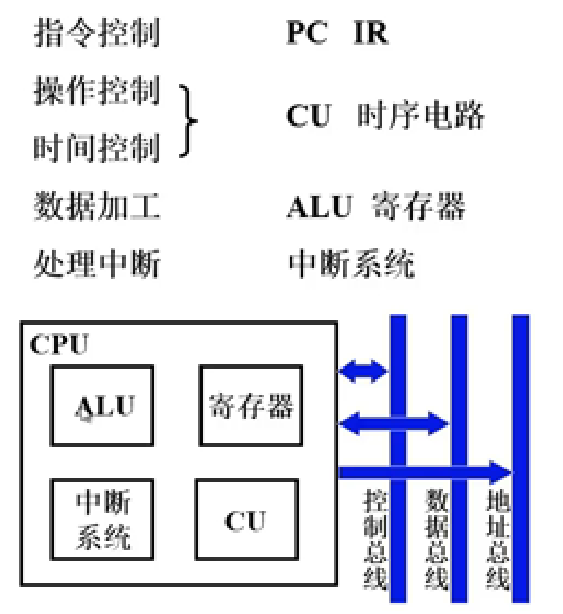
片内总线：系统内部（寄存器）；

系统总线：计算各部件之间（CPU与外设）；

通信总线：计算机与计算机或其他设备；

**CPU与系统总线**

把CPU当作一个黑盒，CPU通过**系统总线**和计算机的其他部件进行通信，系统总线分为控制总线（双向）、数据总线（双向）和地址总线（单向）；



### CPU的寄存器

1. 用户可见寄存器

**通用寄存器** 存放操作数

可作某种寻址方式所需的专用寄存器

**数据寄存器** 存放操作数（满足各种数据类型）

两个寄存器拼接存放双倍子长数据

**地址寄存器** 存放地址，其位数应满足最大的地址范围

用于特殊的寻址方式 段基值 栈指针

**条件码寄存器** 存放条件码，可作程序分支的依据

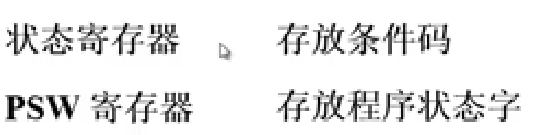
如正、负、零、溢出、进位等；

1. 控制和状态寄存器

**控制寄存器**：



**状态寄存器**：指令执行的结果或软硬件状态



### 控制单元CU和中断系统

1. CU产生全部指令的微操作命令**序列**

组合逻辑设计 硬连线逻辑

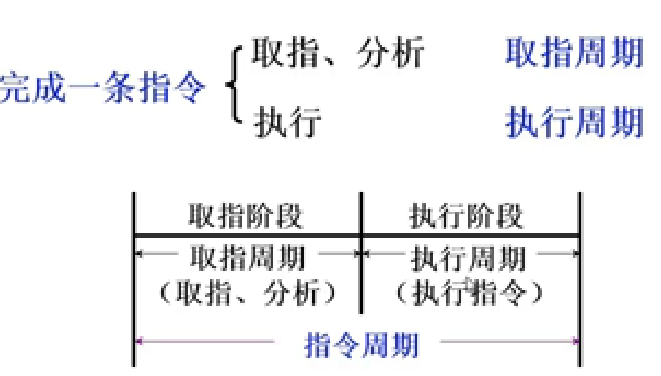
微程序设计 存储逻辑

1. 中断系统

## 指令周期

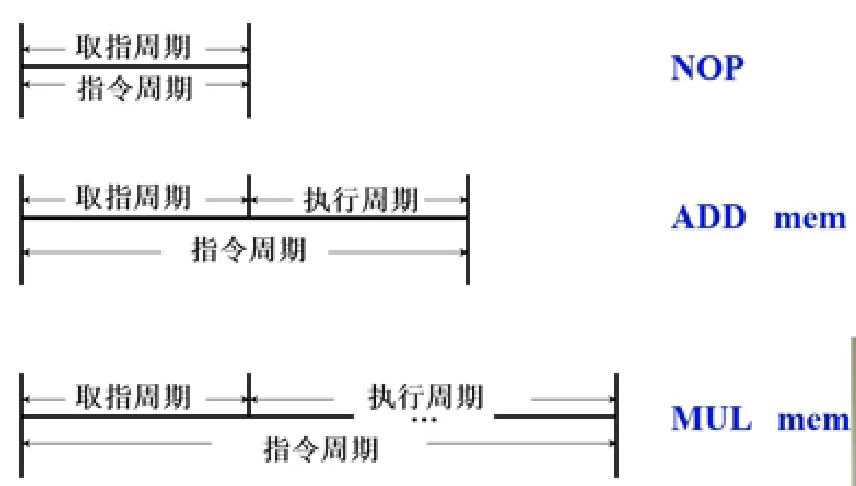
### 指令周期的基本概念

指令周期：完成取指令、分析指令、执行指令、存储结果的全部时间；

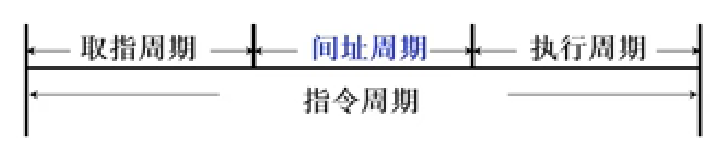


1. 每条指令的指令周期不同

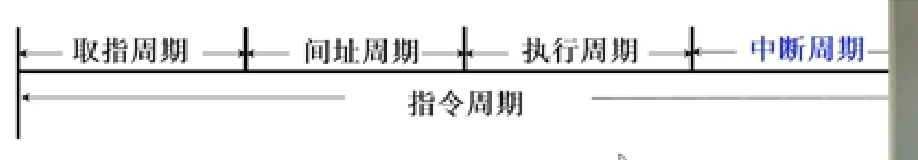
即使同一个设备，不同指令的指令周期长度或所包含的机器周期的个数也可能不同；



1. 具有间接寻址的指令周期

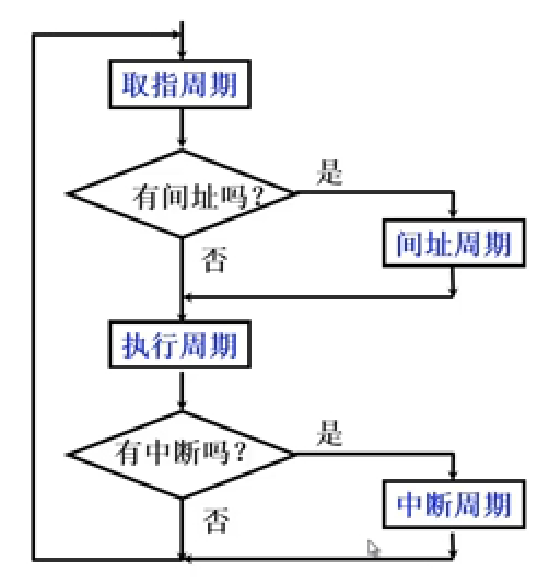


1. 具有中断周期的指令周期



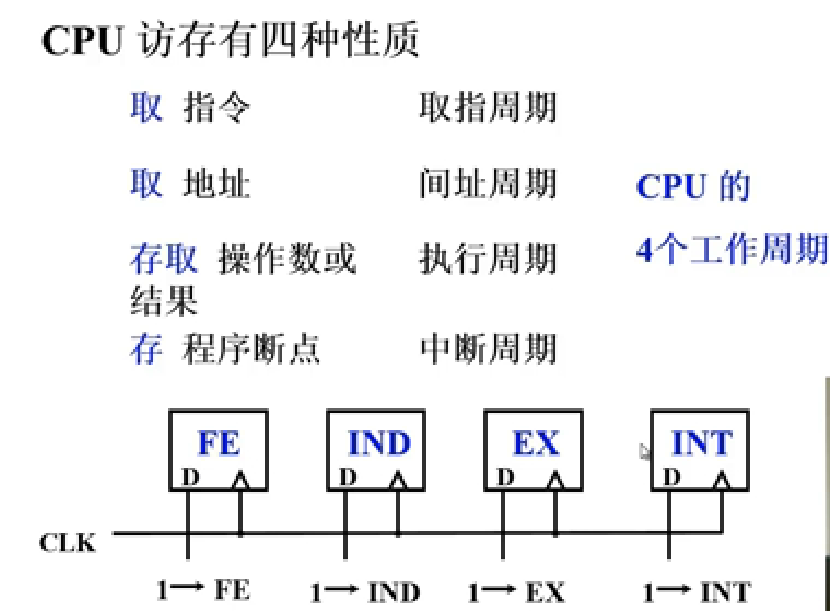
1. 指令周期的流程

完成一条指令的流程：



1. CPU工作周期的标志

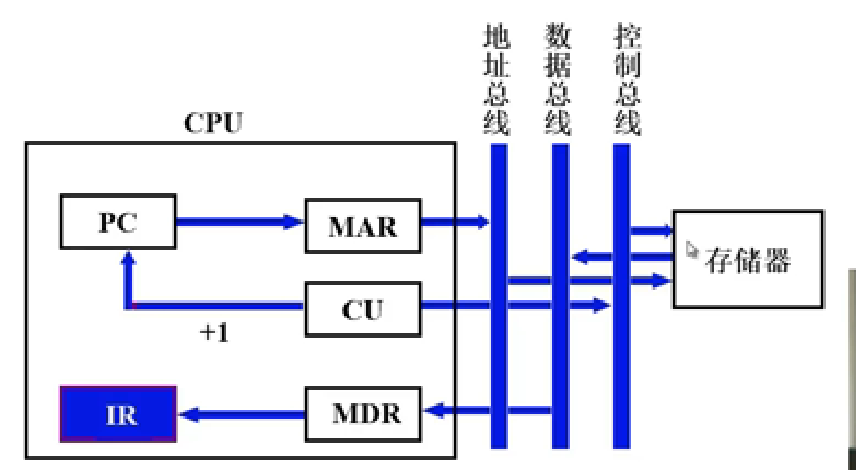
控制器在指令周期的不同阶段要发出不同的控制命令，控制器也需要指令当前指令执行到什么阶段——利用触发器；



### 指令周期的数据流

1. 取指周期数据流

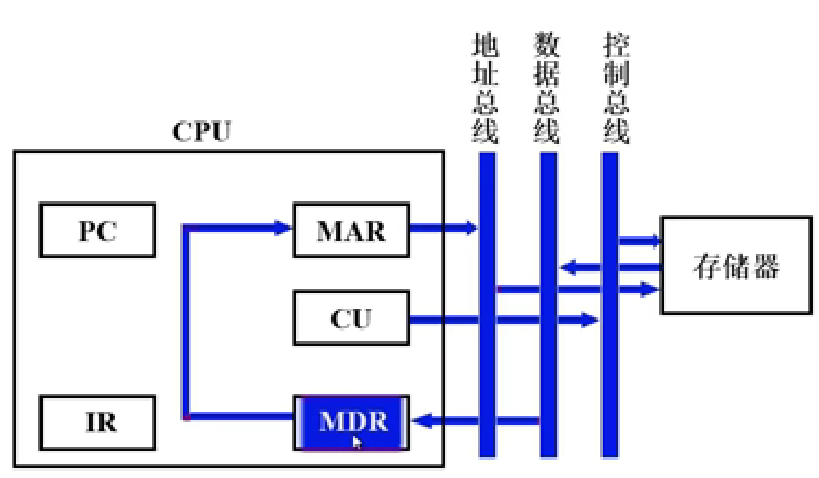
从PC开始：



1. 间指周期数据流

一旦取址周期结束，CU便检查IR中的内容，以确定其是否有间指操作；

取出来的指令在IR和MDR中都有，这里从MDR开始：



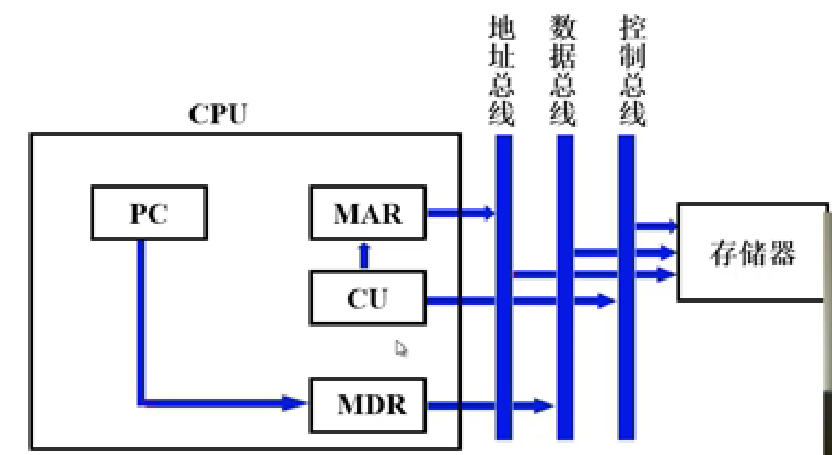
1. 执行周期数据流

不同指令的执行周期数据流不同；

1. 中断周期数据流

**保存断点：**

从CU开始，将断点（中断结束后该执行的指令）写入到存储器中;



**形成中断服务程序的入口地址：**由CU形成，将**入口地址给到PC**；

**硬件关中断**；

## 指令流水

### 如何提高机器速度

1. 提高访存速度

高速芯片 Cache 多体并行；

1. 提高I/O和主机之间的传送方式

中断 DMA 通道 I/O处理机 多总线；

1. 提高运算器的方法

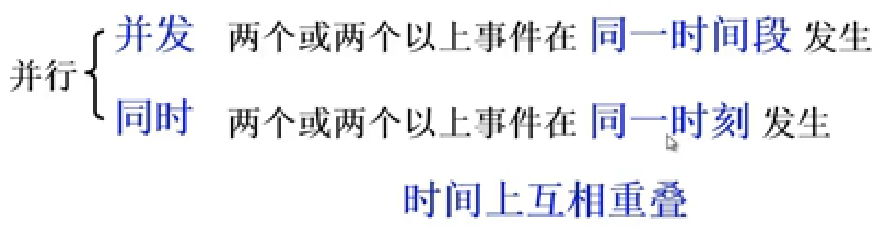
高速芯片 改进算法 快速进位链；

1. **提高整机处理能力**

**高速器件 改进体系机构，开发系统的并行性；**

### 系统的并行性

1. 并行的概念



流水线是同时性的；

1. 并线性的等级

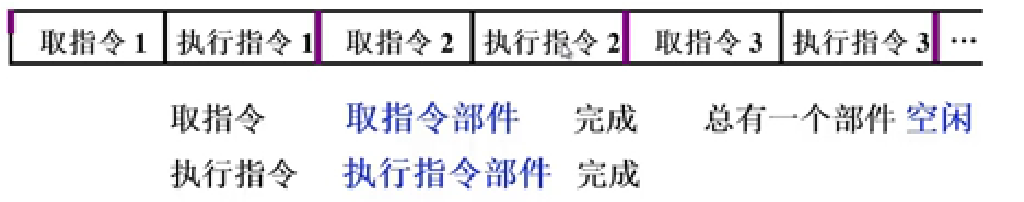
过程性（程序、进程）；——粗颗粒度——软件实现

指令级（指令之间、指令内部，ILT）：多条指令同时处于被解释或被执行状态；或指令内部的微操作并行；——细颗粒的——硬件实现

### 指令流水的原理

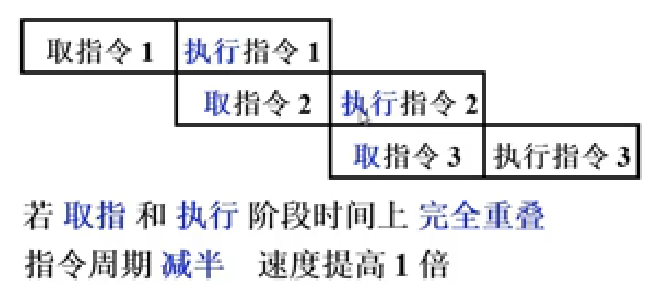
1. 指令的串行执行

假设只有取址和执行阶段；



存在取址部件和执行部件；串行执行时，总有一个部件空闲，则控制器的利用率很低；

1. 指令的二级流水



二级流水使取址部件和执行部件得到充分利用；

1. 影响指令流水效率加倍的因素

* 执行时间 > 取指时间



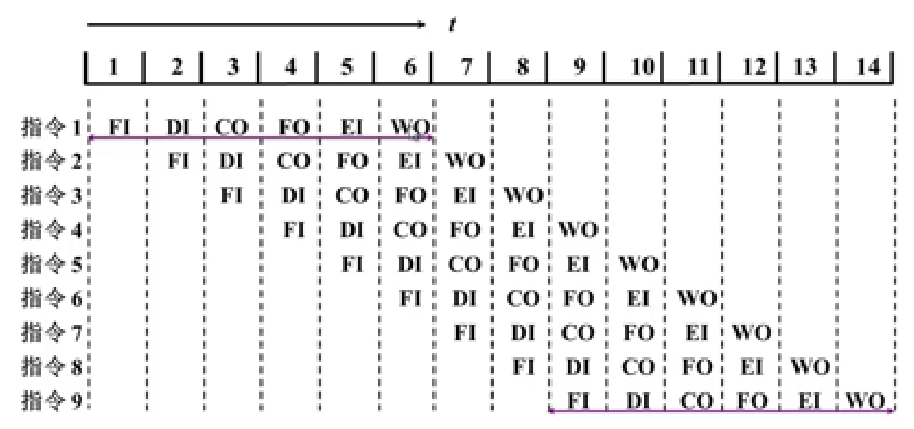
* **条件转移指令**对指令流水的影响

无条件转移指令对指令流水造成的影响较小，在译码阶段就可以确定下条指令的地址；

**条件转移指令**必须**等到上条指令执行结束**，才能获知条件是否成立，从而确定**下条**指令的地址，造成时间损失；

解决方法：分支预测等；

1. 六级流水线举例



FI：取址；

DI：指令译码；

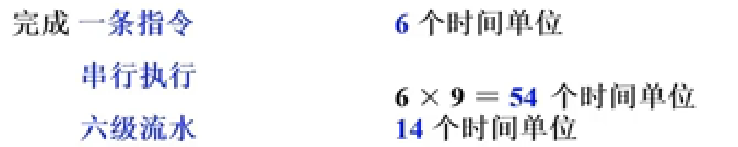
CO：形成操作数的地址；

FO：取操作数

EI：执行指令

WO：结果存储；

假设每个阶段消耗时间相同；



### 影响指令流水性能的因素

1. 结构相关

不同指令争用**同一功能部件**产生的资源冲突，降低流水线效率；



**解决访存冲突办法**：

* 停顿（加入气泡）；
* **指令存储器**和**数据存储器**分开——哈佛结构；
* 指令预取技术（适用于访存周期短的情况）；

1. 数据相关

不同指令因**重叠**操作，可能改变操作数的读/写**访问顺序**；

* 先写后读相关（RAW）



* 先读后写相关（WAR）



* 先写后写相关（WAR）



不管用什么并行技术执行程序，都要保证这个程序的执行结果和程序串行执行的**结果完全相同**；（程序串行执行时，有些指令存在先后顺序，并行执行时，不能打乱其顺序）

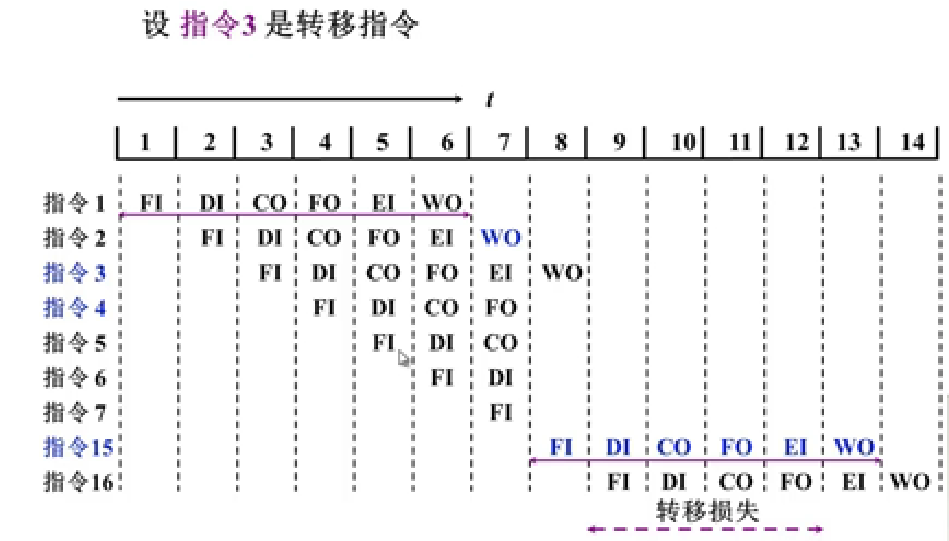
**解决办法：**

* 后推法；
* 采用旁路技术（相关专用通路）；

1. 控制相关

控制相关主要是由**转移指令**引起的转移损失；





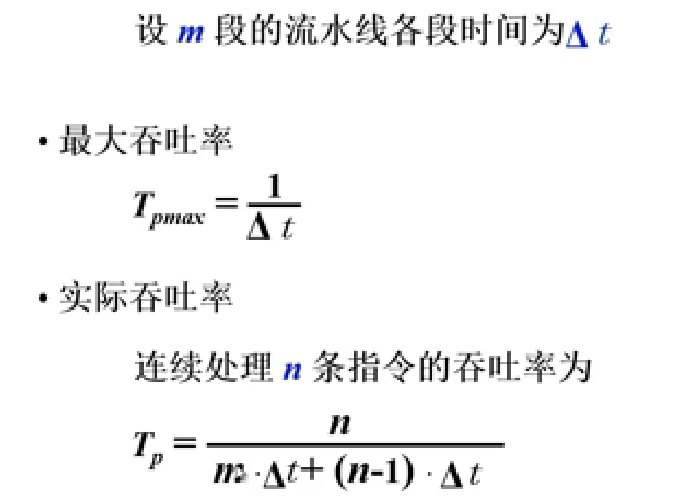
### 指令流水线性能

1. 吞吐率

单位时间内**流水线所完成指令**或**输出结果**的**数量**；

最大吞吐率：没有资源冲突、条件转移等，满负荷运行下的吞吐率；

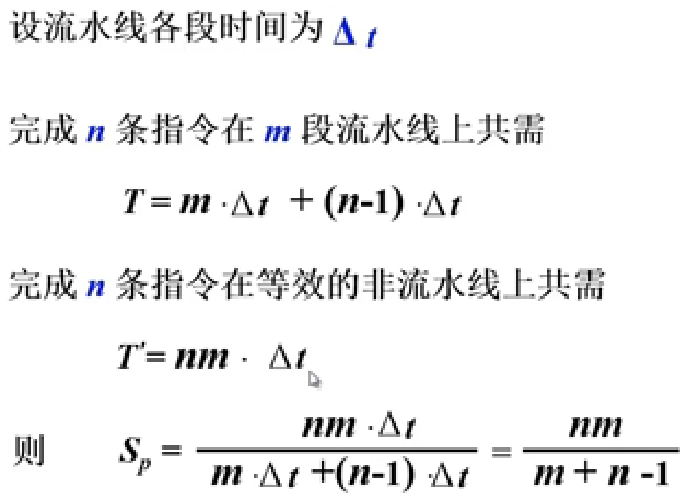
实际吞吐率：



1. 加速比Sp

采用某种技术和不采用某种技术的比值；

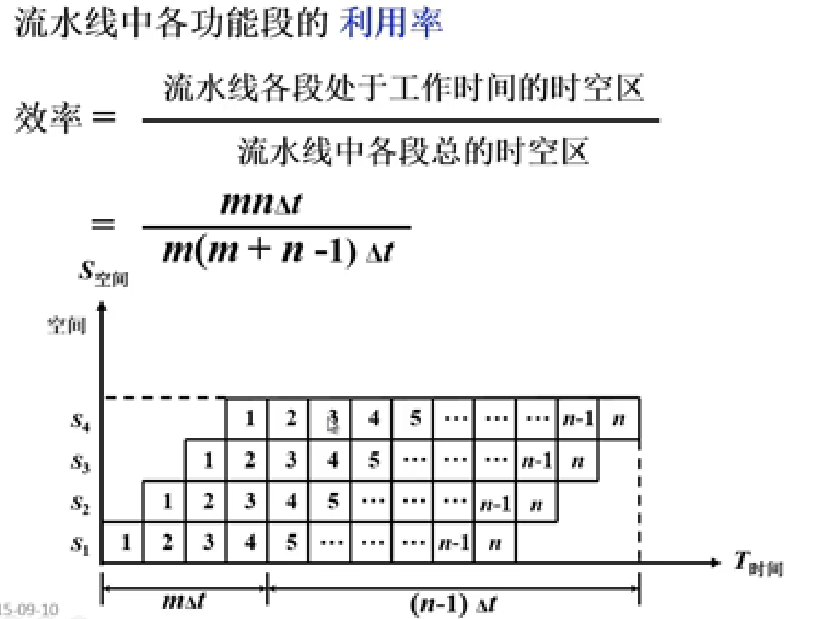
流水线的加速比是指m段流水线的速度与等功能的非流水线的速度之比。



1. 效率

流水线中各功能段的利用率；

由于流水线有建立时间和排空时间，因此各功能段的设备不可能一直处于工作状态；

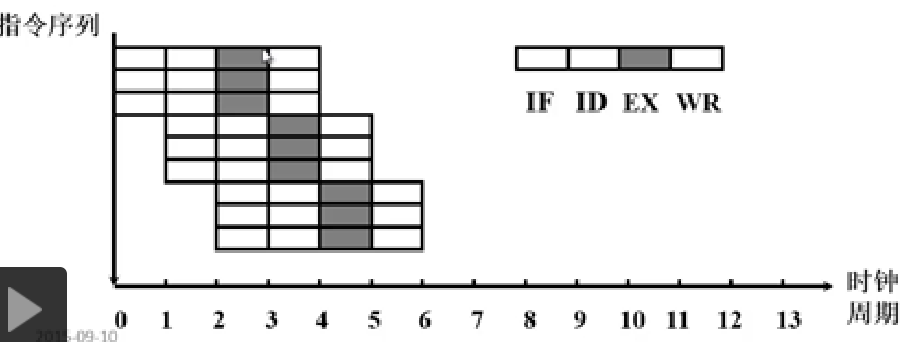


### 流水线的多发技术

每级流水段之间有锁存器；

1. 超标量技术

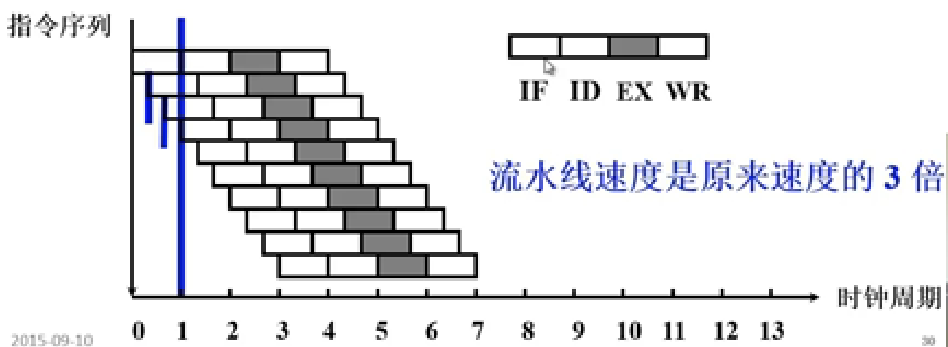
* 每个时钟周期内可**并发多条独立指令**，配置多个功能部件；
* **不能调整**指令的**执行顺序**，通过**编译优化**技术（静态），把可**并行执行**的指令搭配起来同时执行；



3条4级指令流水线

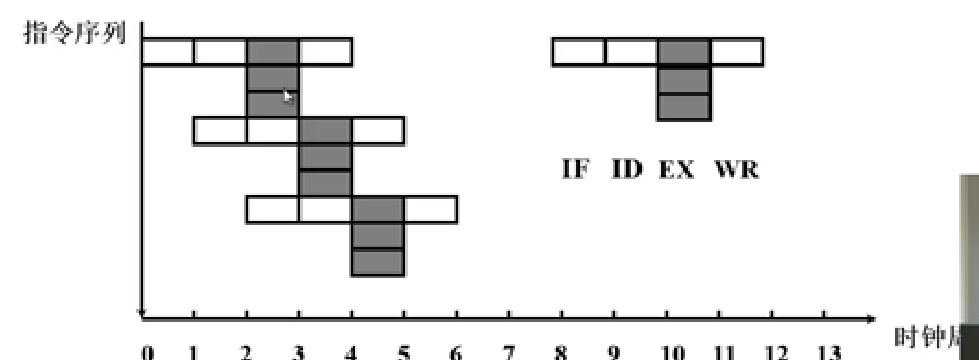
1. 超流水线技术

* 在一个时钟周期内在分段，在**一个时钟周期**内一个**功能部件使用多次**（3次）；
* 指令执行过程中，不能调整指令的执行顺序，靠编译程序解决优化问题；



1. 超长指令字技术

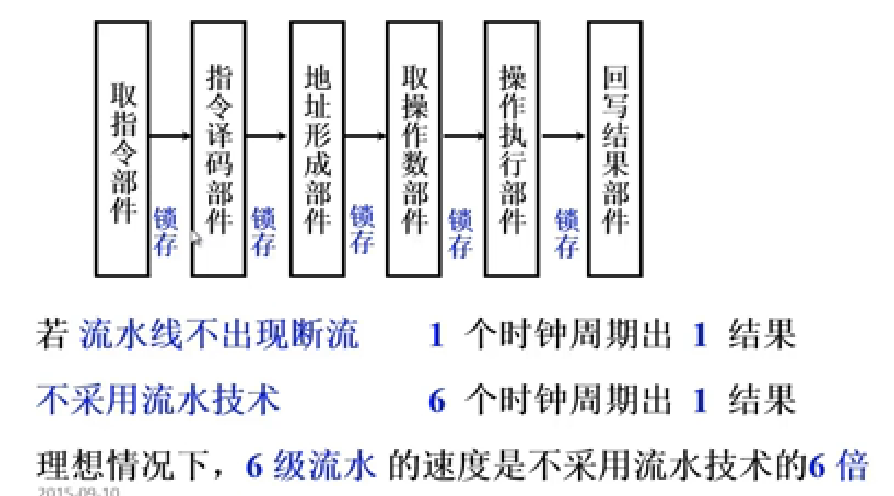
* 由编译程序挖掘出指令间潜在的并行性，将多条能并行操作的指令组合成一条具有多个操作码字段的超长指令字（可达几百位）；
* 采用多个处理执行部件；



### 流水线结构

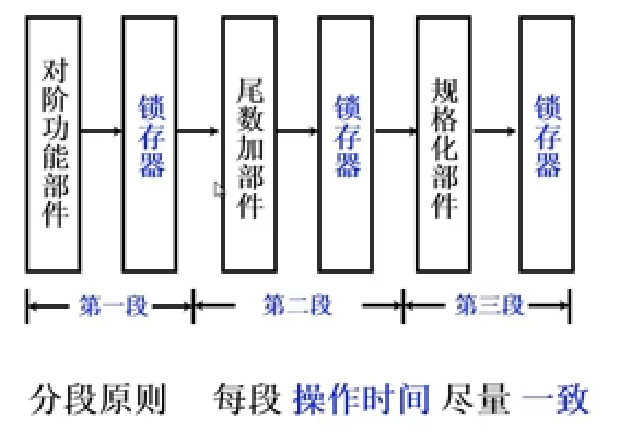
1. 指令流水线结构

完成一条指令分6段（不同结构不同），每段需要一个时钟周期；



1. 运算流水线

完成浮点加减运算可分对阶、尾数求和、规格化三段，所以也可以采用流水线方式；

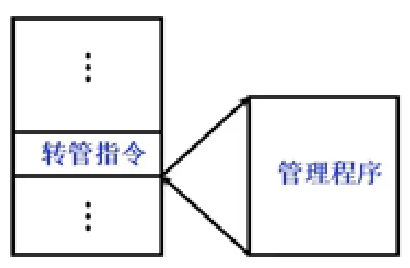


## 中断系统

### 概述

1. 引起中断的各种因素

* 人为设置的中断：如转管指令；
* 程序性事故：溢出、操作码不能识别、除法非法；
* 硬件故障；
* I/O设备输入输出；
* 外部事件：用键盘中断现行程序；



1. 中断系统需解决的问你题



硬件 + 软件 实现；

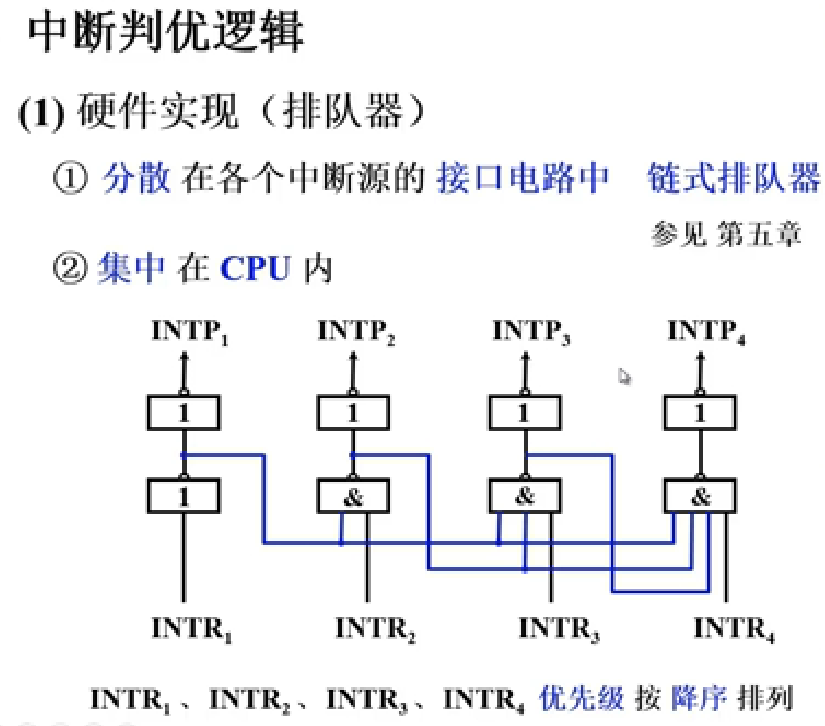
### 中断请求标记和中断判优逻辑

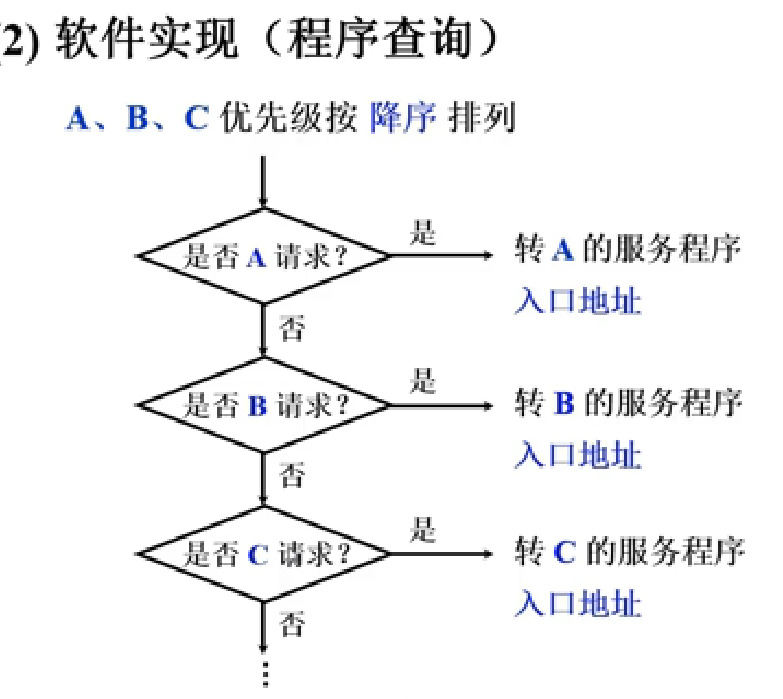
1. 中断源如何向CPU提出中断请求



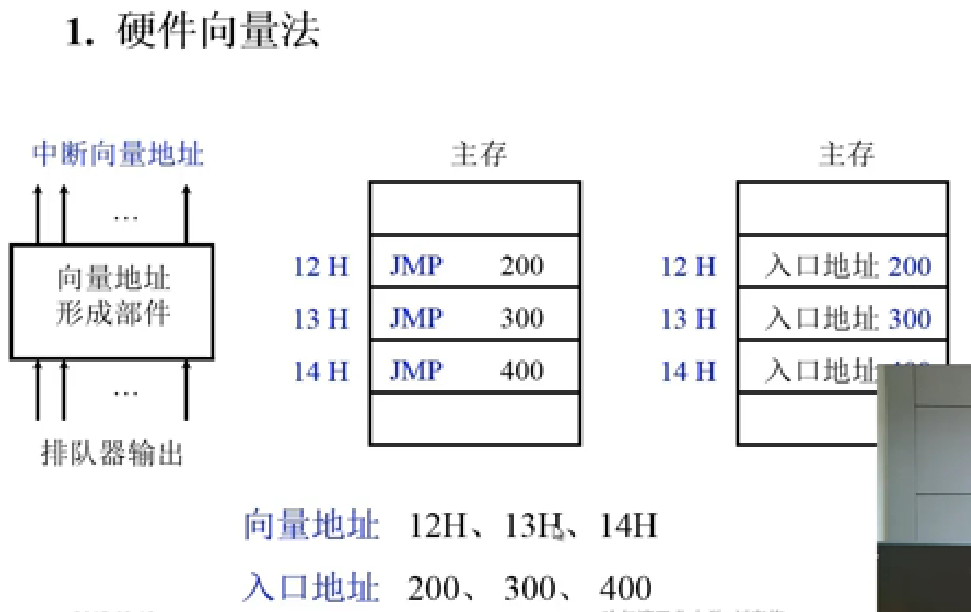
1. 中断源同时提出中断请求CPU怎么响应

按重要性分优先级；





### 中断服务程序入口地址的寻找



硬件向量法速度快，但灵活度低；



### 中断响应

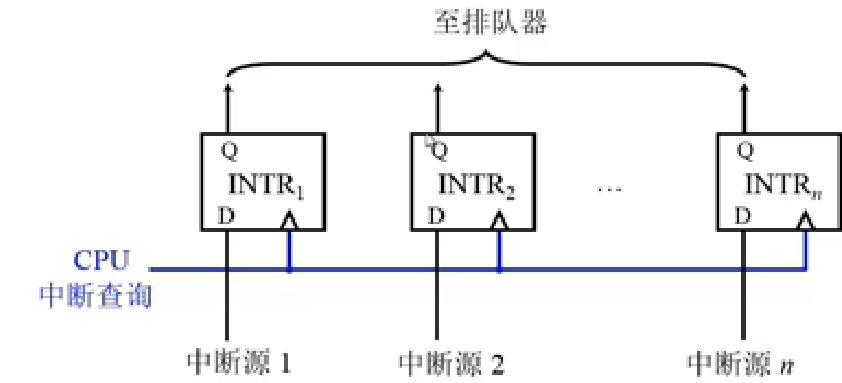
1. 响应中断的条件

允许中断触发器EINT = 1；

1. 响应中断的时间

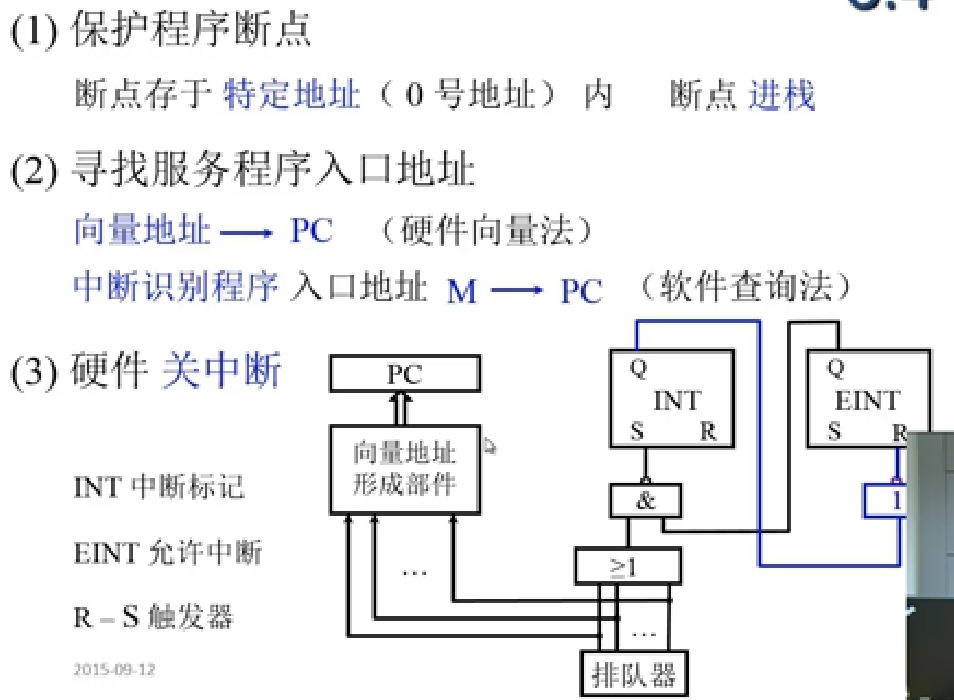
通常，CPU只有在执行执行结束以后才能响应中断

指令执行周期结束时刻由CPU发查询信号；



1. 中断隐指令——由计算机硬件实现

程序断点的保存、中断服务程序入口地址的形成、开/关全局中断的行为；



### 保存现场和恢复现场