

# 计算机组成原理

翁睿

哈尔滨工业大学

# 关于DMA 周期窃取过程的一点补充

- **宏观** 看起来好像通过DMA一次性连续传送了一批数据
- **微观** 每与主存间传送一个字，设备发一次 DMA请求

## ➤ 一个字的传输过程分析如下

1. **设备准备数据**：设备接收到CPU发送的启动命令后，开始准备数据。
2. **设备发送DMA请求**：数据准备好后，设备通过DREQ控制线向DMA接口发出DMA请求。
3. **DMA接口申请总线**：DMA接口收到DMA请求后，立即将HRQ(HOLD)信号置“1”，向CPU申请总线控制权。

必须让出，因为DMA比CPU优先极高
4. **总线授权**：CPU在每个机器周期结束后，响应总线使用申请并让出总线控制权，并发出总线授权信号HLDA通知DMA接口可以使用总线。
5. **数据传输**：DMA接口成功的偷走了一个访存周期，一个字的数据在设备和主存之间按约定的方向传送。

其中，3、4步骤虽然经过了CPU，但CPU所做的事只相当于：

**HLDA=(HOLD && (CPU当前不访存))**，这个过程几乎不耗时。

# 基本寻址方式及其优缺点总结

# 7.3

假设：A=地址字段值，R=寄存器编号，  
EA=有效地址，(X)=X中的内容



方式	算法	主要优点	主要缺点
立即	操作数=A	指令执行速度快	操作数范围有限
直接	EA=A	有效地址计算简单	地址范围有限
间接	EA=(A)	有效地址范围大	多次存储器访问
寄存器	操作数=(R)	指令执行快，指令短	地址范围有限
寄间接	EA=(R)	地址范围大	额外存储器访问
偏移	EA=A+(R)	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

偏移寻址：将直接方式和寄存器间接方式结合起来。

具体包括：相对 / 基址 / 变址三种

# 第8章 CPU 的结构和功能

## 8.1 CPU 的结构

## 8.2 指令周期

## 8.3 指令流水

## 8.4 中断系统

# 8.1 CPU 的结构

## 一、CPU 的功能

### 1. 控制器的功能

取指令

分析指令

执行指令，发出各种操作命令

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

### 2. 运算器的功能

实现算术运算和逻辑运算

## 二、CPU 结构框图

### 8.1

#### 1. CPU 与系统总线

指令控制

PC IR

操作控制

CU 时序电路

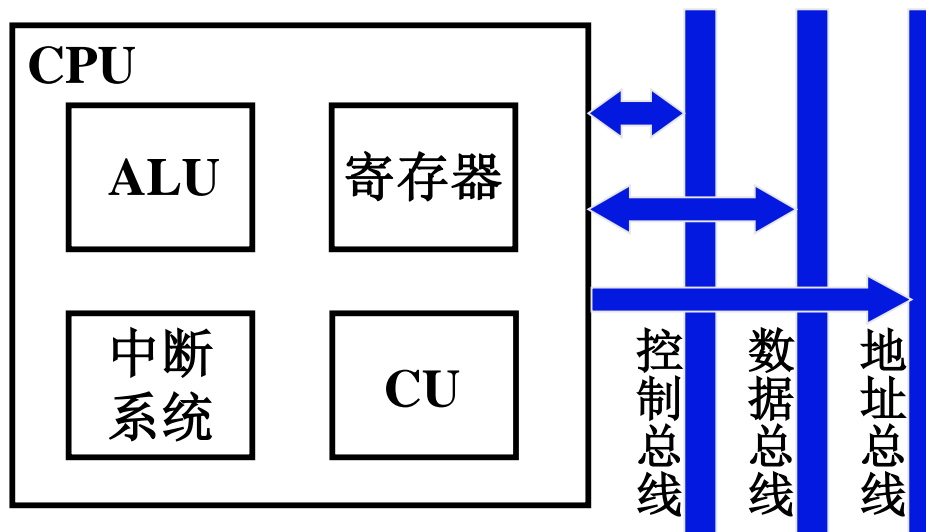
时间控制

数据加工

ALU 寄存器

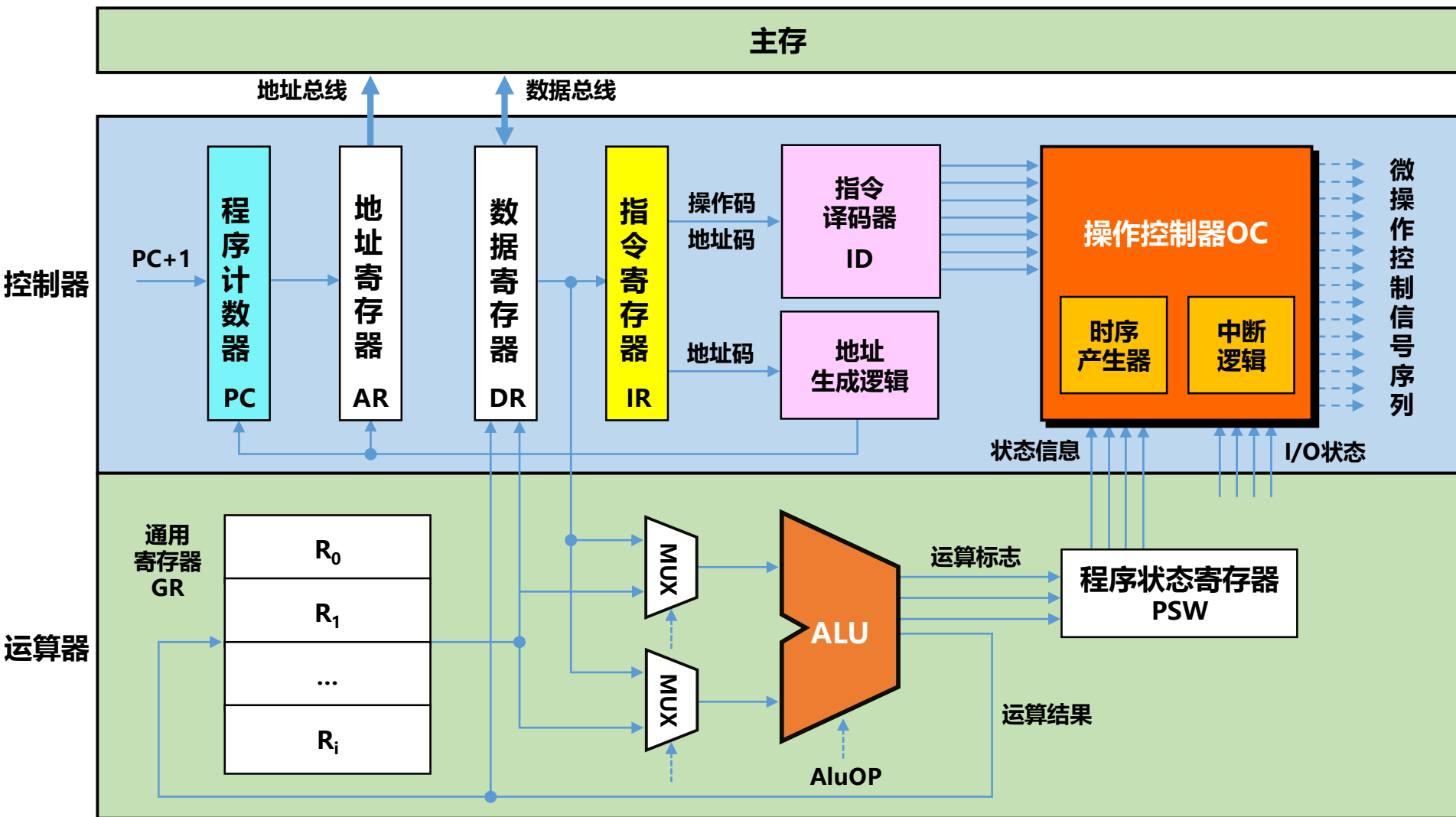
处理中断

中断系统



# 简单CPU模型

## 8.1



# 三、CPU 的寄存器

## 8.1

### 1. 用户可见寄存器

- (1) 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
- (2) 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字长数据
- (3) 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    段基址    栈指针
- (4) 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



## 2. 控制和状态寄存器

### (1) 控制寄存器

**PC** → **MAR** → **M** → **MDR** → **IR**

控制 CPU 操作

其中 **MAR**、**MDR**、**IR**

用户不可见

**PC**

用户可见

### (2) 状态寄存器

状态寄存器

存放条件码

**PSW** 寄存器

存放程序状态字

## 3. 举例

**Z8000**

**8086**

**MC 68000**

## 四、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

参见 第 10 章

### 2. 中断系统

参见 本章 8.4 节

## 五、ALU

参见 第 6 章

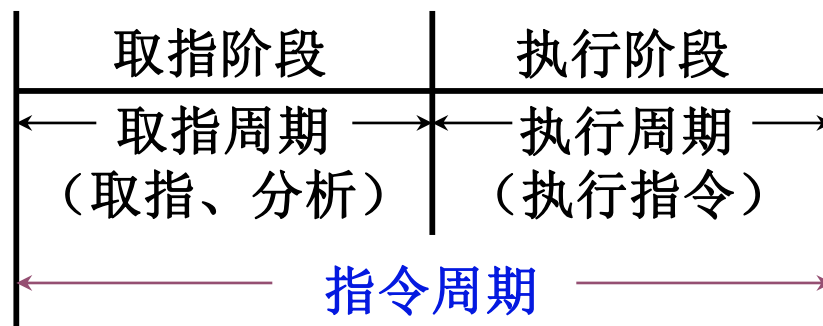
## 8.2 指令周期

### 一、指令周期的基本概念

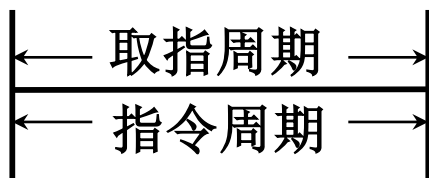
#### 1. 指令周期

取出并执行一条指令所需的全部时间

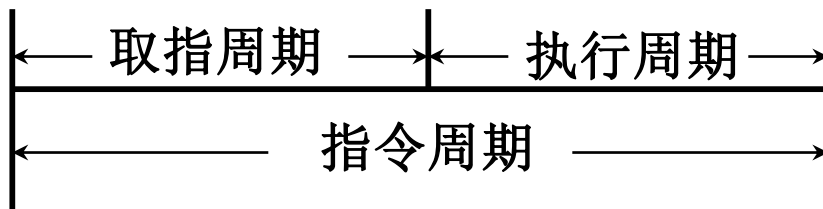
完成一条指令 { 取指、分析      取指周期  
                                执行              执行周期



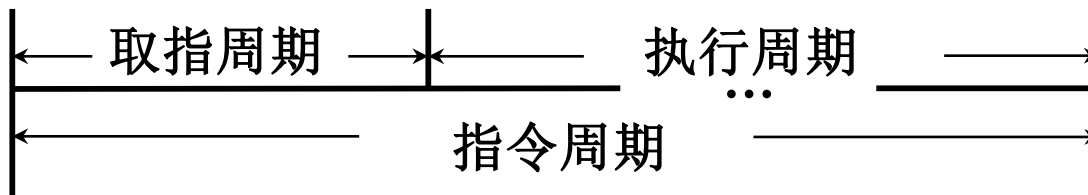
## 2. 每条指令的指令周期不同



**NOP**

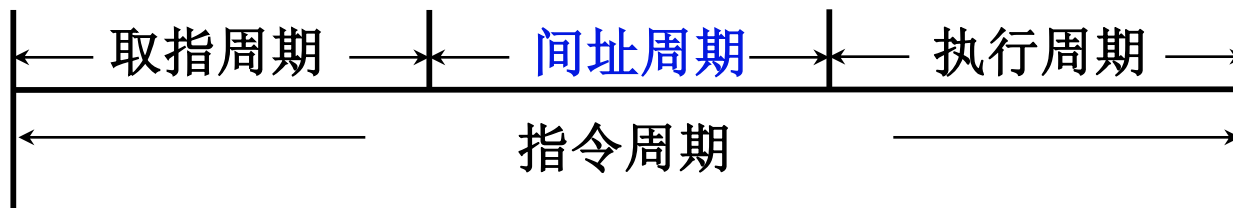


**ADD mem**

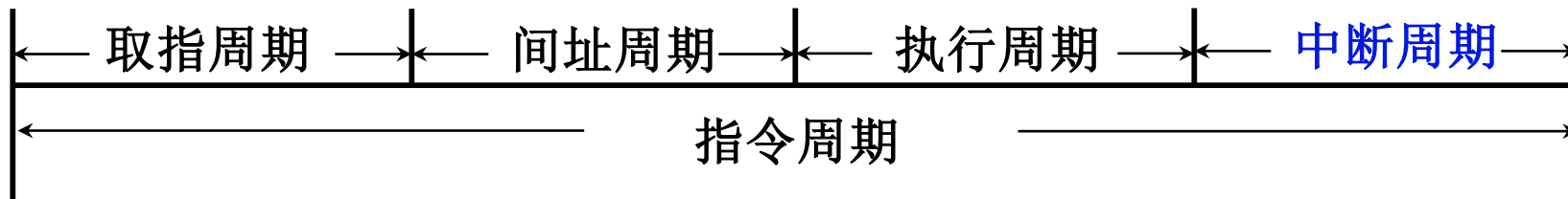


**MUL mem**

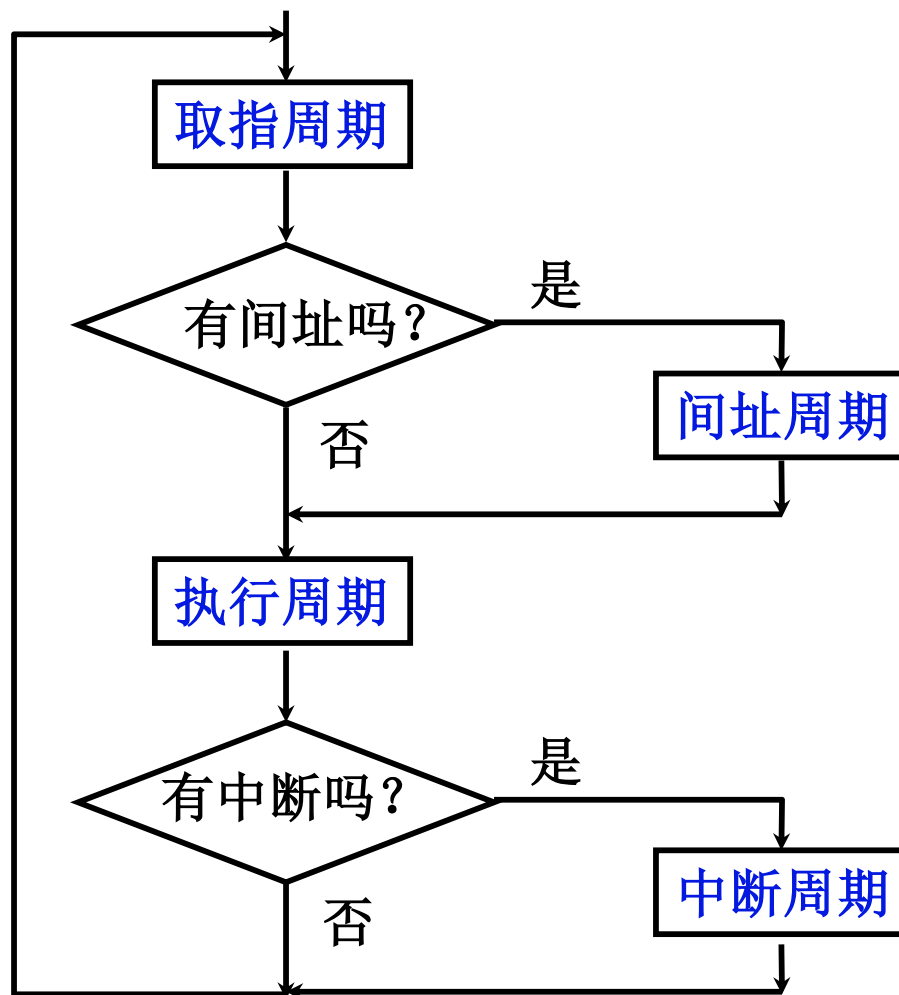
### 3. 具有间接寻址的指令周期



### 4. 带有中断周期的指令周期



## 5. 指令周期流程



## 6. CPU 工作周期的标志

CPU 访存有四种性质

取 指令

取指周期

取 地址

间址周期

取 操作数

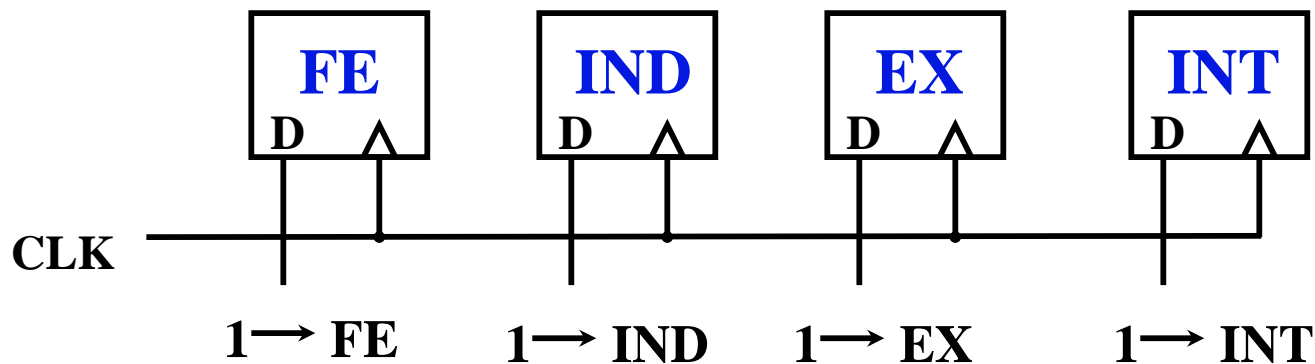
执行周期

存 程序断点

中断周期

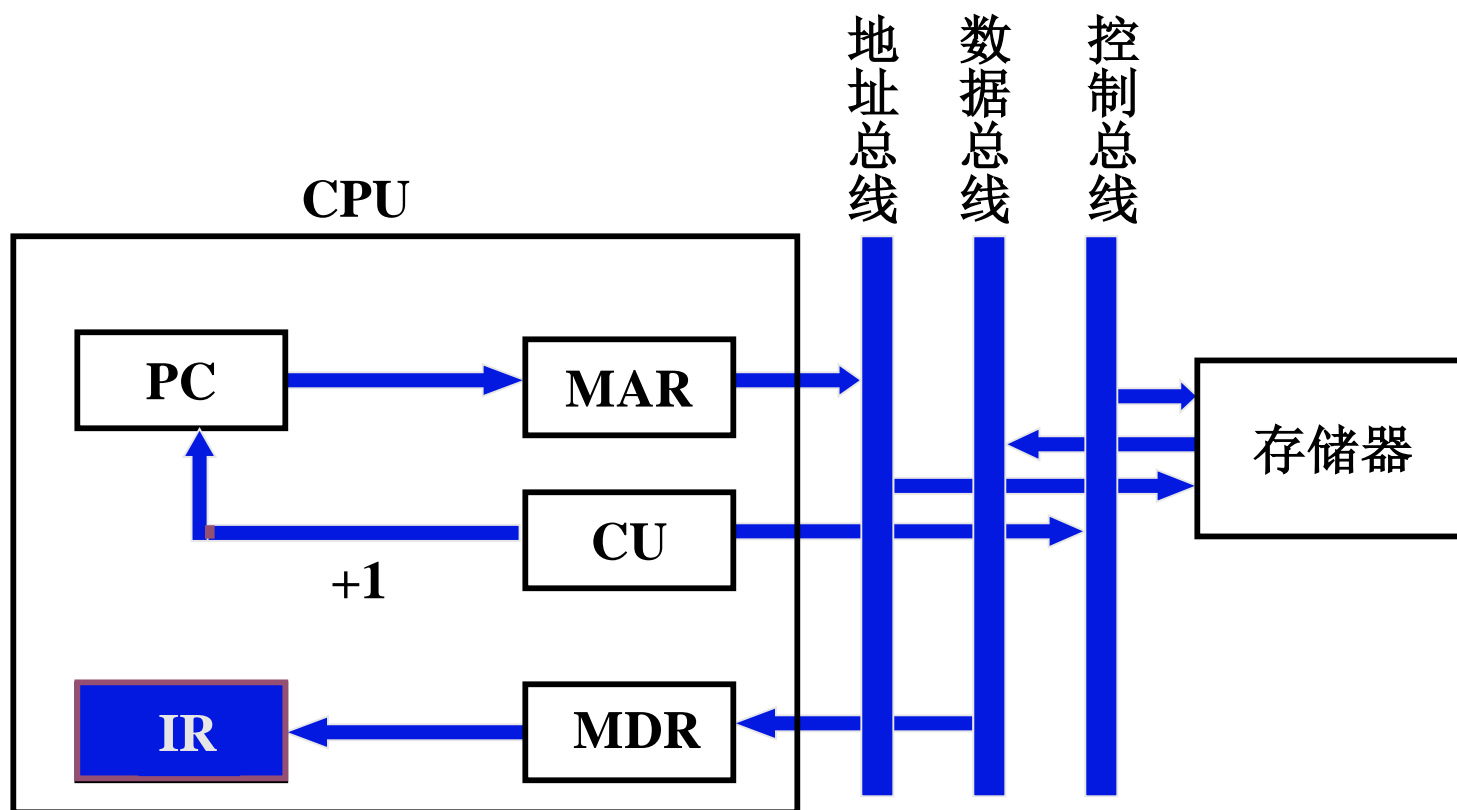
CPU 的

4个工作周期



## 二、指令周期的数据流

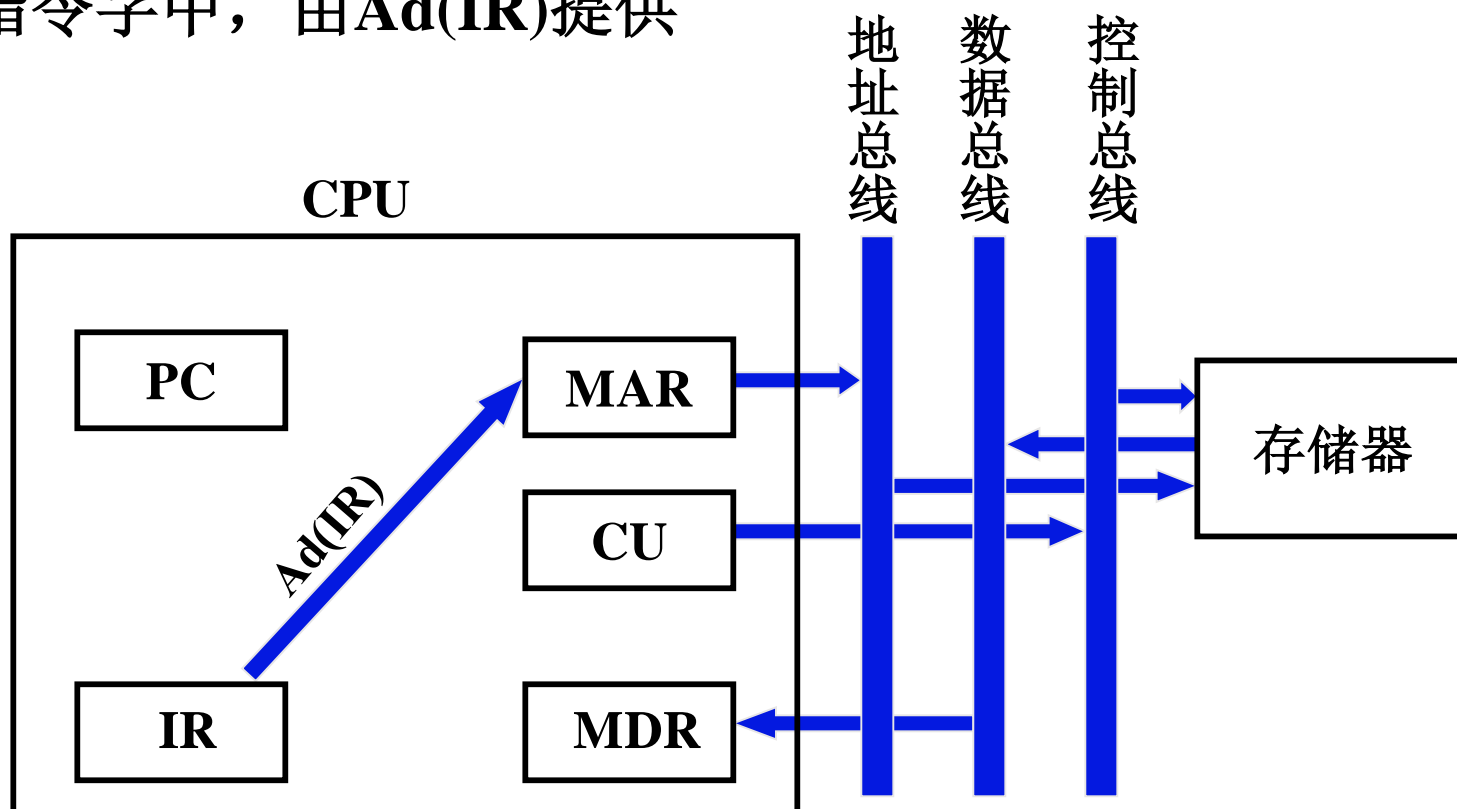
### 1. 取指周期数据流





## 2. 间址周期数据流

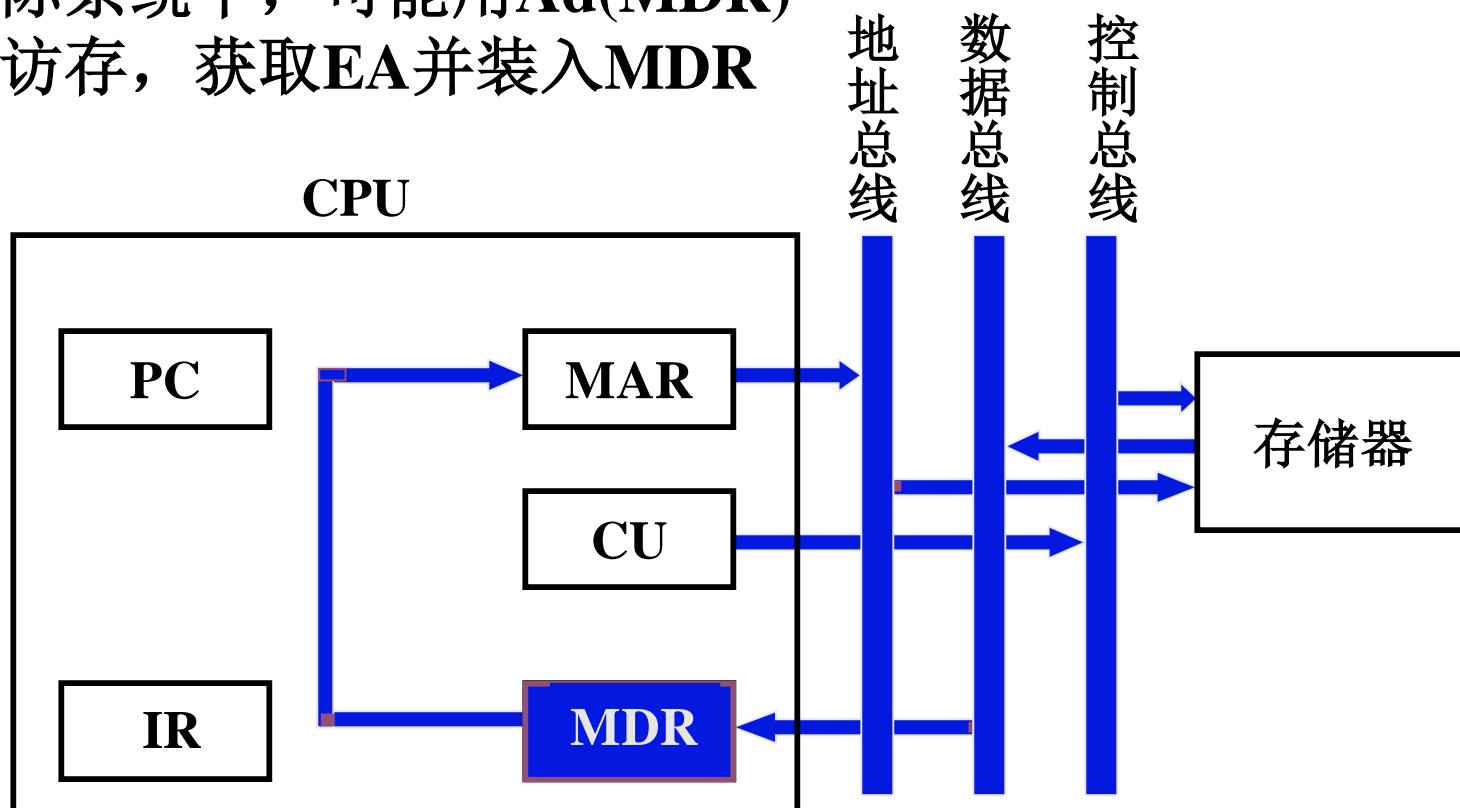
以一次间址为例，此时访存地址在指令字中，由Ad(IR)提供



## 2. 间址周期数据流

因为取指周期结束时，MDR 的值刚刚送入 IR，此时  $Ad(MDR)$  就是  $Ad(IR)$

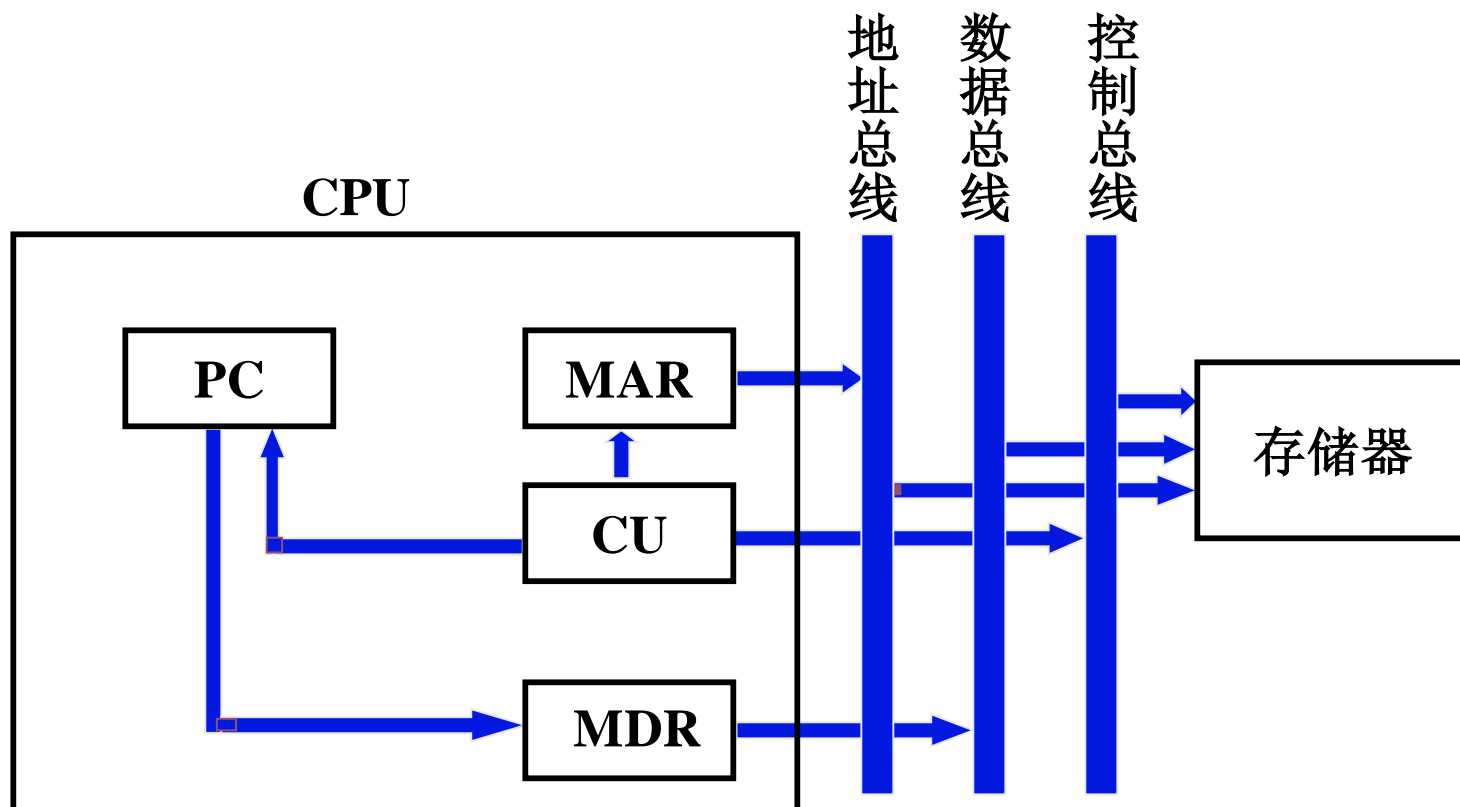
在实际系统中，可能用  $Ad(MDR)$  进行访存，获取 EA 并装入 MDR



### 3. 执行周期数据流

不同指令 的执行周期数据流 不同

### 4. 中断周期数据流



## 8.3 指令流水

### 一、如何提高机器速度

#### 1. 提高访存速度

高速芯片

Cache

多体并行

#### 2. 提高 I/O 和主机之间的传送速度

中断

DMA

通道

I/O 处理机

多总线

#### 3. 提高运算器速度

高速芯片

改进算法

快速进位链

#### • 提高整机处理能力

高速器件

改进系统结构，开发系统的并行性

## 二、系统的并行性

## 8.3

### 1. 并行的概念

并行 { **并发** 两个或两个以上事件在 **同一时间段** 发生  
**同时** 两个或两个以上事件在 **同一时刻** 发生

**时间上互相重叠**

### 2. 并行性的等级

过程级（程序、进程）	<b>粗粒度</b>	软件实现
指令级（指令之间） （指令内部）	<b>细粒度</b>	硬件实现

# 三、指令流水原理

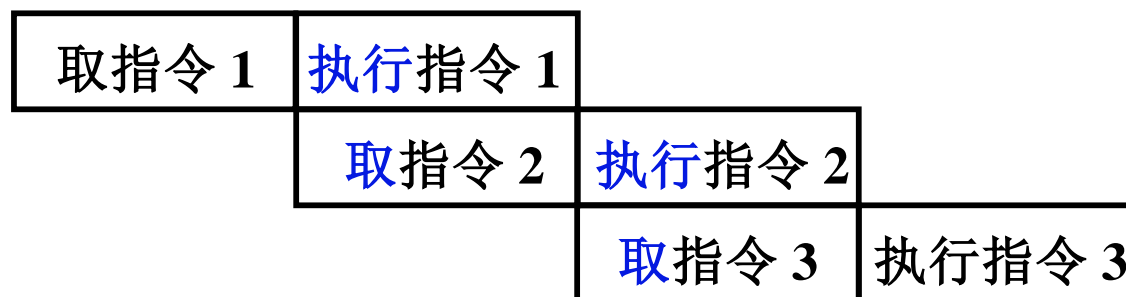
## 8.3

### 1. 指令的串行执行



取指令      取指令部件      完成      总有一个部件 空闲  
执行指令      执行指令部件      完成

### 2. 指令的二级流水

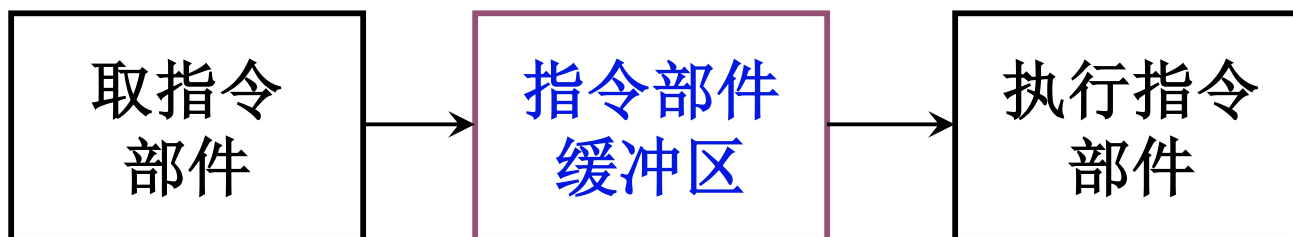


若 取指 和 执行 阶段时间上 完全重叠  
指令周期 减半    速度提高 1 倍

### 3. 影响指令流水效率加倍的因素

## 8.3

#### (1) 执行时间 > 取指时间



#### (2) 条件转移指令 对指令流水的影响

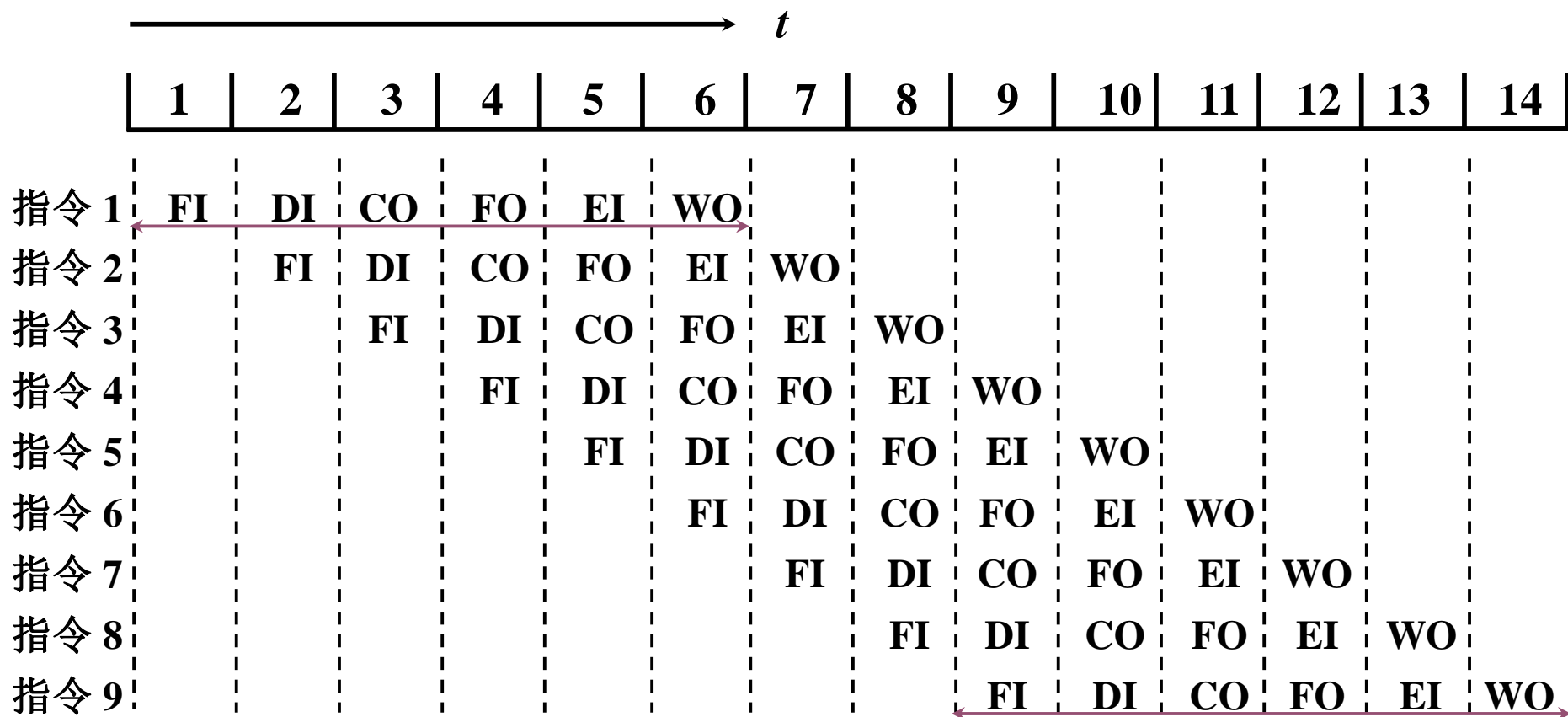
必须等 上条 指令执行结束，才能确定 下条 指令的地址，  
造成时间损失

解决办法 ？

猜测法

## 4. 指令的六级流水

8.3



完成 一条指令

串行执行

六级流水

6 个时间单位

$6 \times 9 = 54$  个时间单位

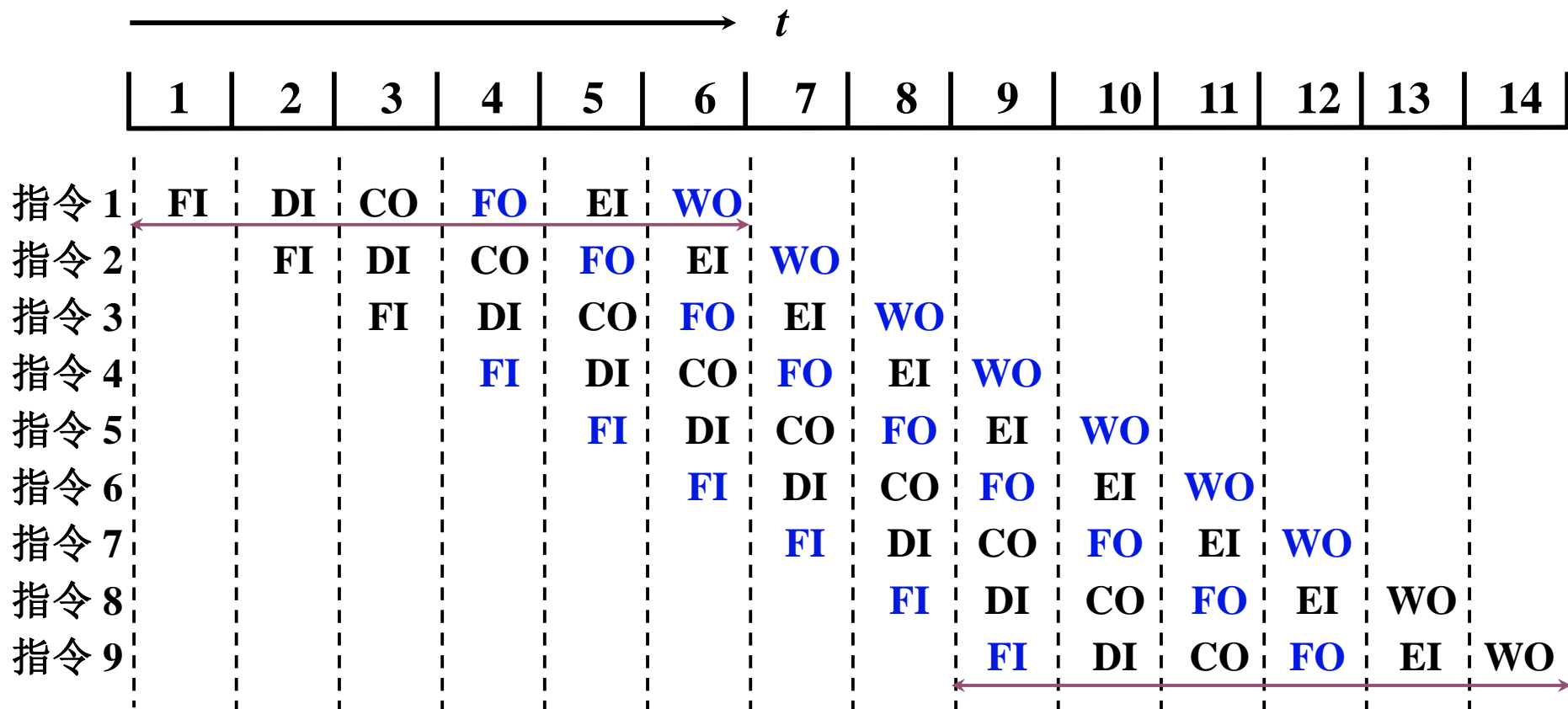
14 个时间单位



### 三、影响指令流水线性能的因素

## 8.3

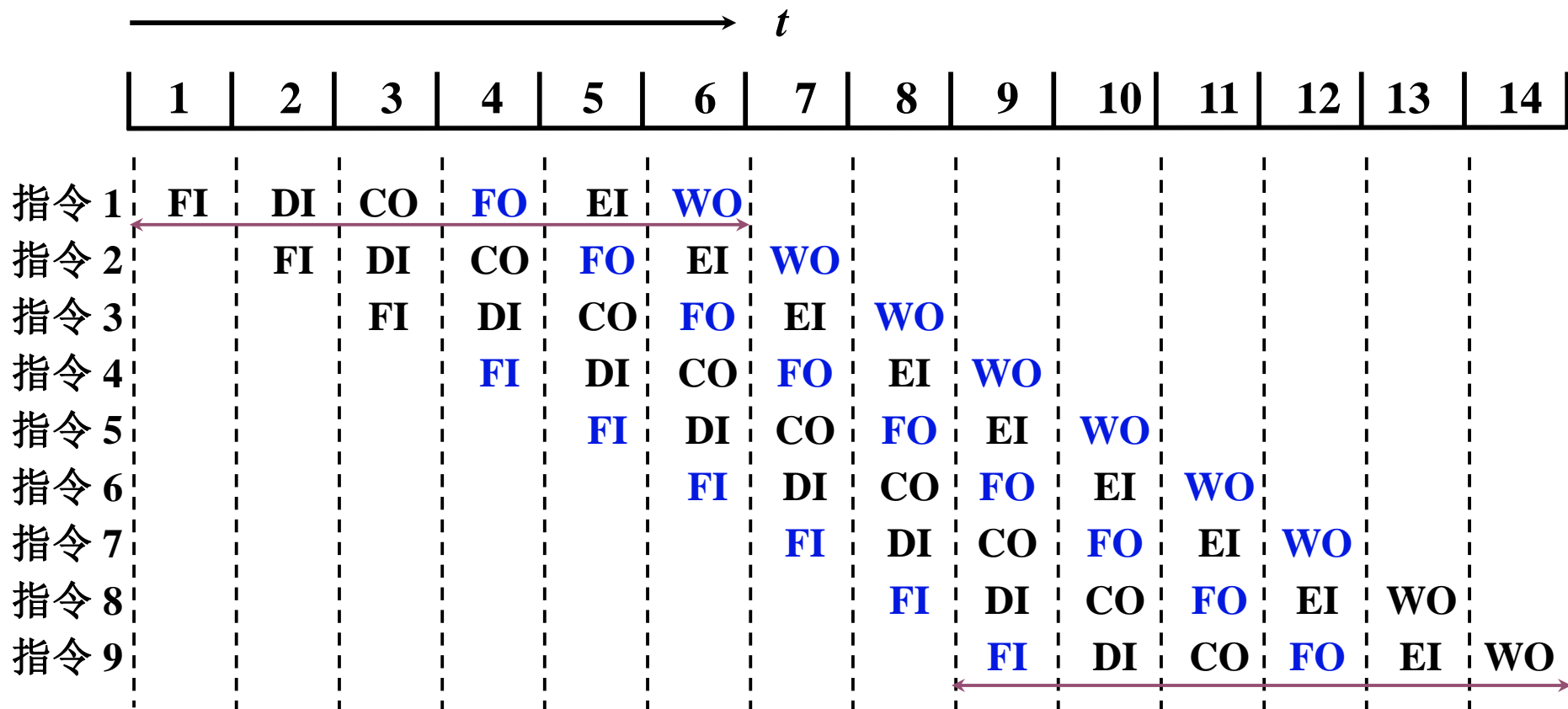
#### 1. 结构相关 不同指令争用同一功能部件产生资源冲突



### 三、影响指令流水线性能的因素

## 8.3

#### 1. 结构相关 不同指令争用同一功能部件产生资源冲突



#### 解决办法

- 停顿等待
- 指令存储器和数据存储器分开
- 指令预取技术（适用于访存周期短的情况）

## 2. 数据相关

## 8.3

不同指令因重叠操作，可能改变操作数的 读/写 访问顺序

- 写后读相关 (RAW)

SUB  $R_1$ ,  $R_2$ ,  $R_3$  ;  $(R_2) - (R_3) \rightarrow R_1$

ADD  $R_4$ ,  $R_5$ ,  $R_1$  ;  $(R_5) + (R_1) \rightarrow R_4$

- 读后写相关 (WAR)

STA M,  $R_2$  ;  $(R_2) \rightarrow M$  存储单元

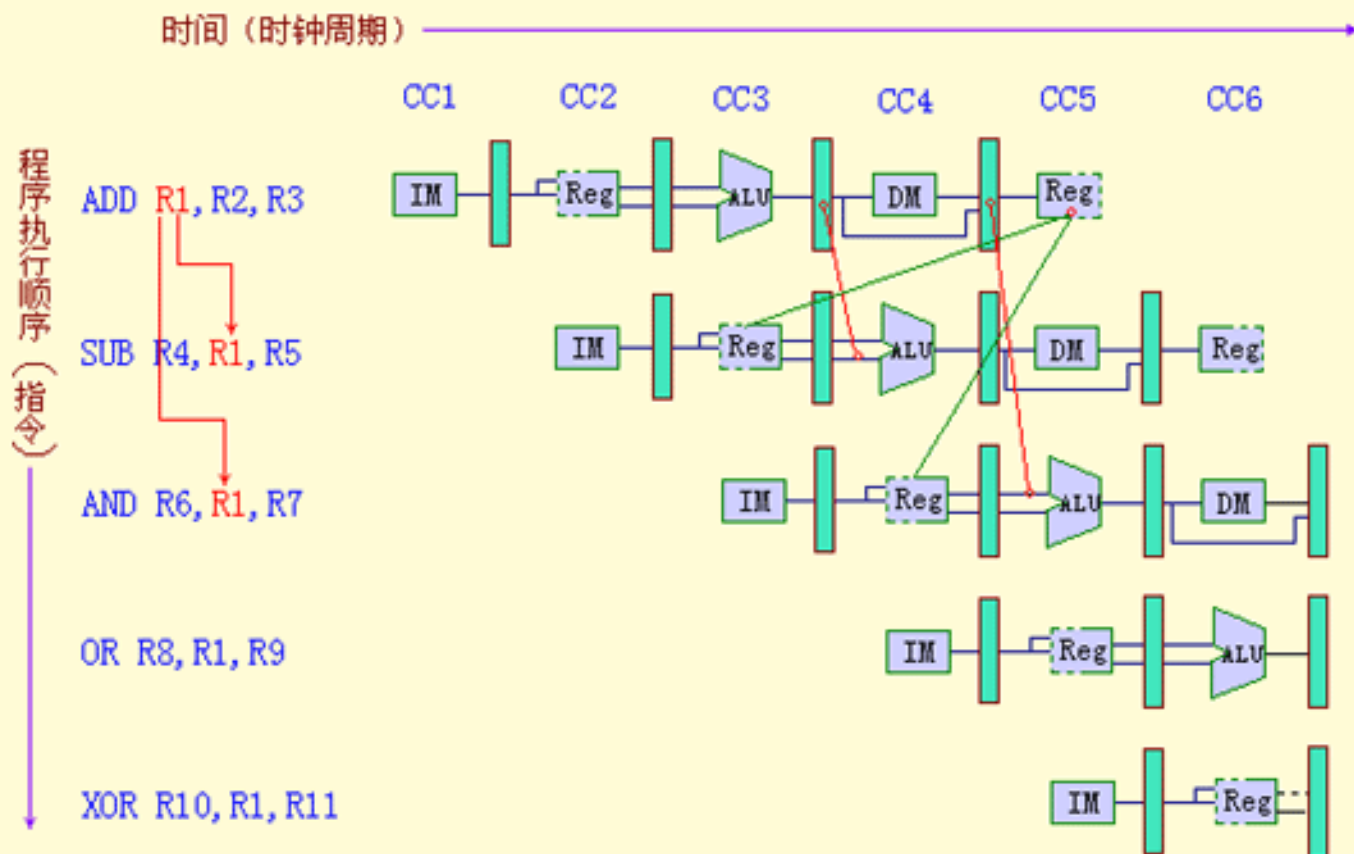
ADD  $R_2$ ,  $R_4$ ,  $R_5$  ;  $(R_4) + (R_5) \rightarrow R_2$

- 写后写相关 (WAW)

MUL  $R_3$ ,  $R_2$ ,  $R_1$  ;  $(R_2) \times (R_1) \rightarrow R_3$

SUB  $R_3$ ,  $R_4$ ,  $R_5$  ;  $(R_4) - (R_5) \rightarrow R_3$


## 采用定向技术消除数据相关



### 3. 控制相关

## 8.3

由转移指令引起



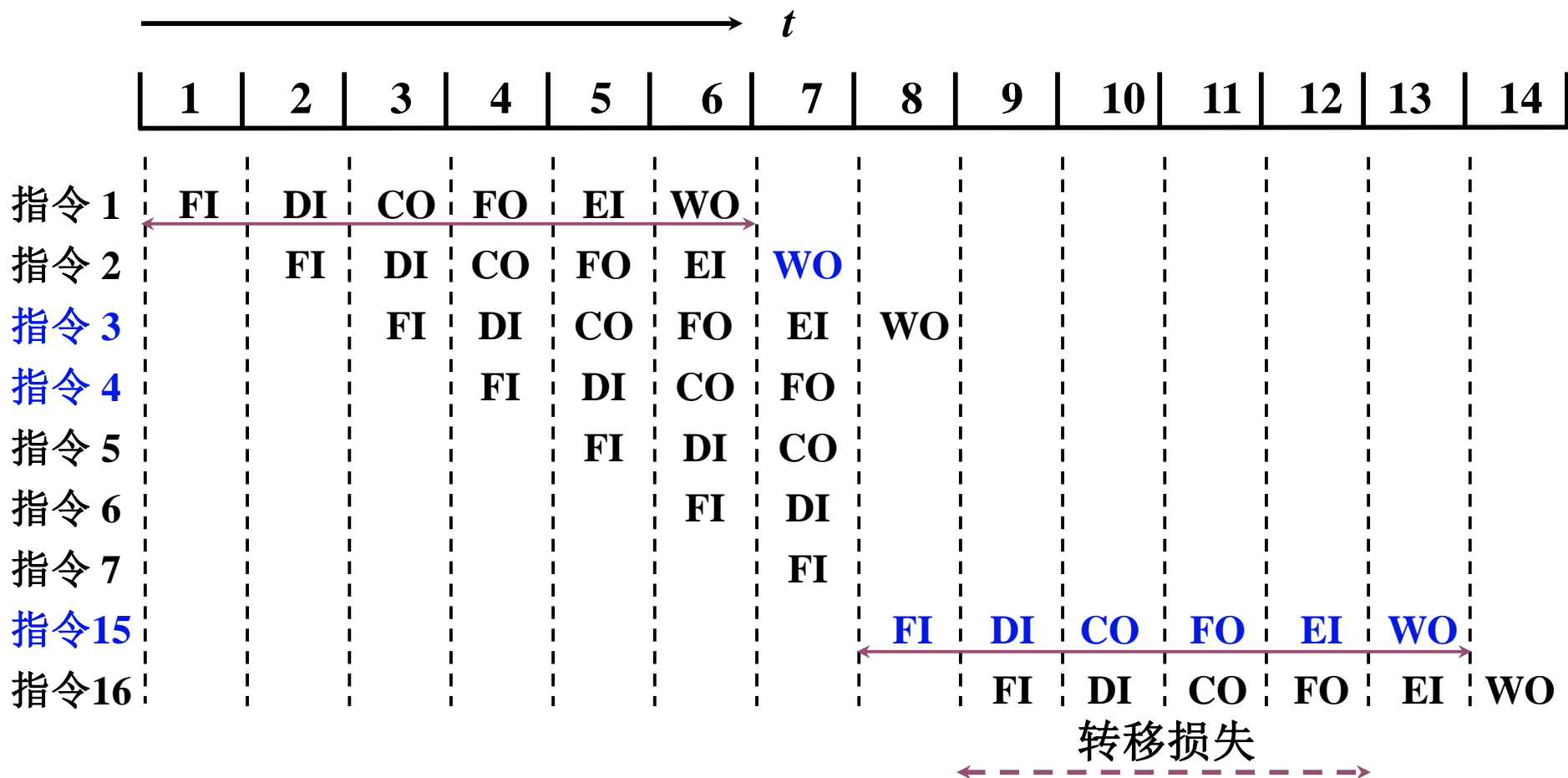
LDA # 0  
LDX # 0  
M ADD X, D  
INX  
CPX # N  
BNE M  
DIV # N  
STA ANS

**BNE** 指令必须等  
**CPX** 指令的结果  
才能判断出  
是转移  
还是顺序执行

### 3. 控制相关

8.3

设 指令3 是转移指令



## 四、流水线性能

## 8.3

### 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

设  $m$  段的流水线各段时间为  $\Delta t$

- 最大吞吐率

$$T_{pmax} = \frac{1}{\Delta t}$$

- 实际吞吐率

连续处理  $n$  条指令的吞吐率为

$$T_p = \frac{n}{m \cdot \Delta t + (n-1) \cdot \Delta t}$$

## 2. 加速比 $S_p$

$m$  段的流水线的速度与等功能的非流水线的速度之比

设流水线各段时间为  $\Delta t$

完成  $n$  条指令在  $m$  段流水线上共需

$$T = m \cdot \Delta t + (n-1) \cdot \Delta t$$

完成  $n$  条指令在等效的非流水线上共需

$$T' = nm \cdot \Delta t$$

则

$$S_p = \frac{nm \cdot \Delta t}{m \cdot \Delta t + (n-1) \cdot \Delta t} = \frac{nm}{m + n - 1}$$

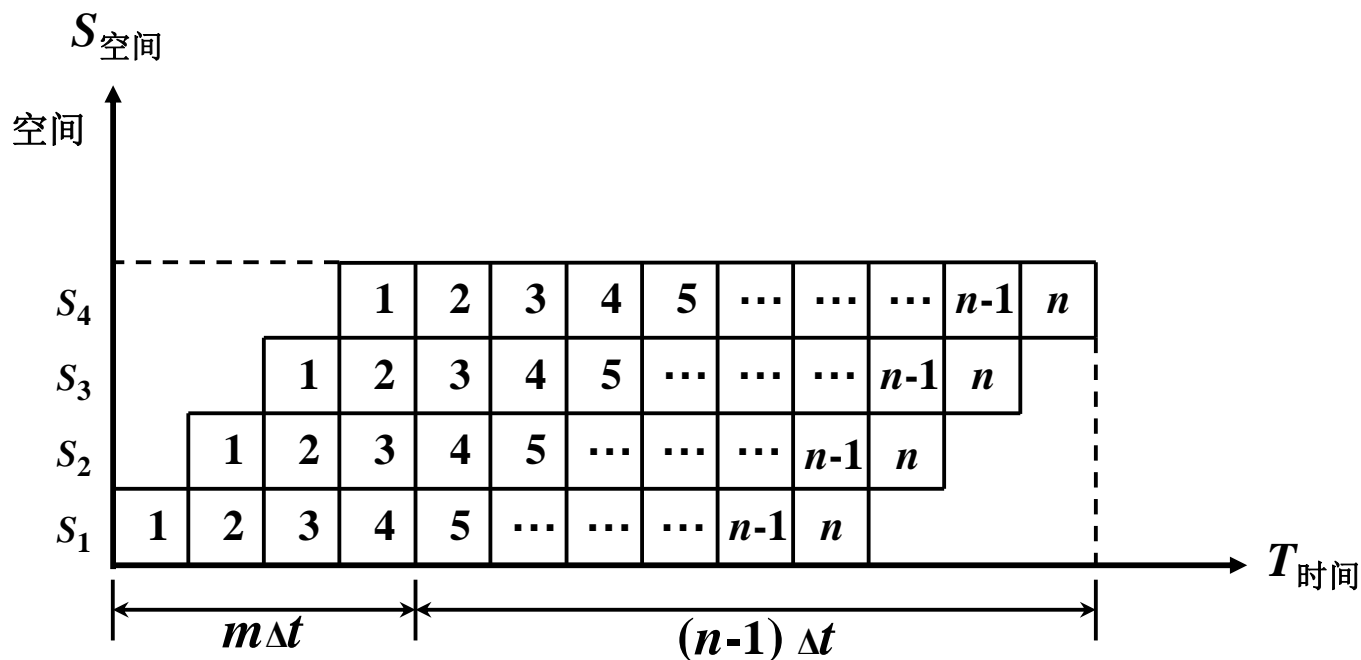


### 3. 效率

## 8.3

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



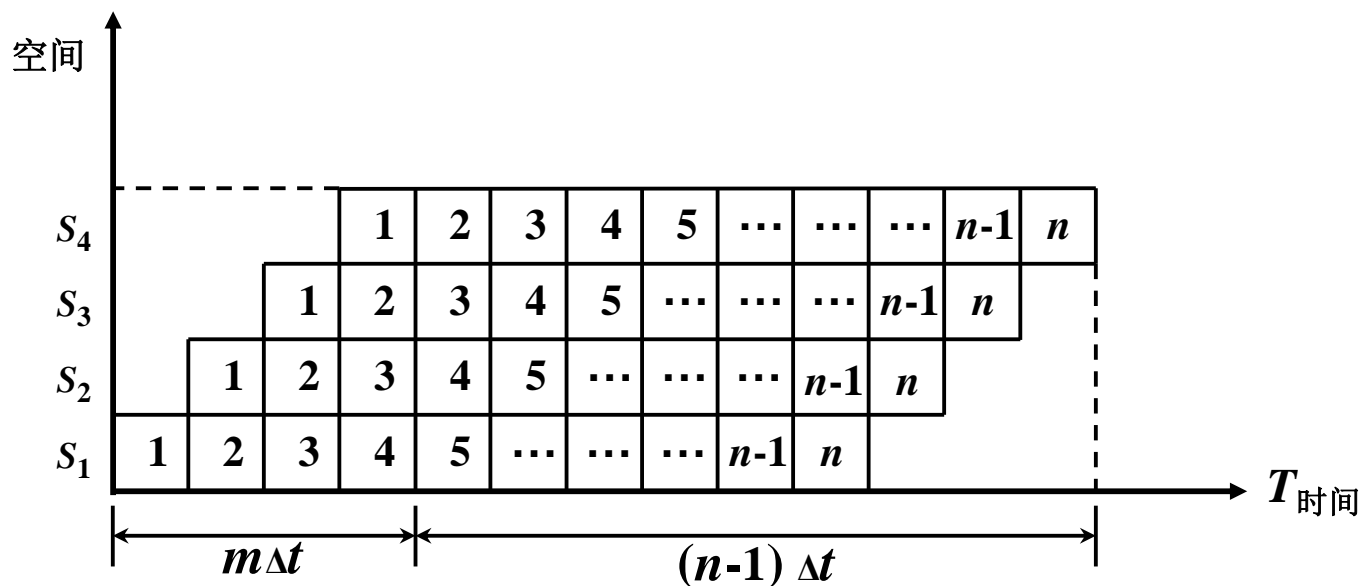
### 3. 效率

## 8.3

流水线中各功能段的 **利用率**

效率 =  $\frac{\text{流水线各段处于工作时间的时空区}}{\text{流水线中各段总的时空区}}$

$$= \frac{mn\Delta t}{m(m+n-1)\Delta t}$$



# 五、流水线的多发技术

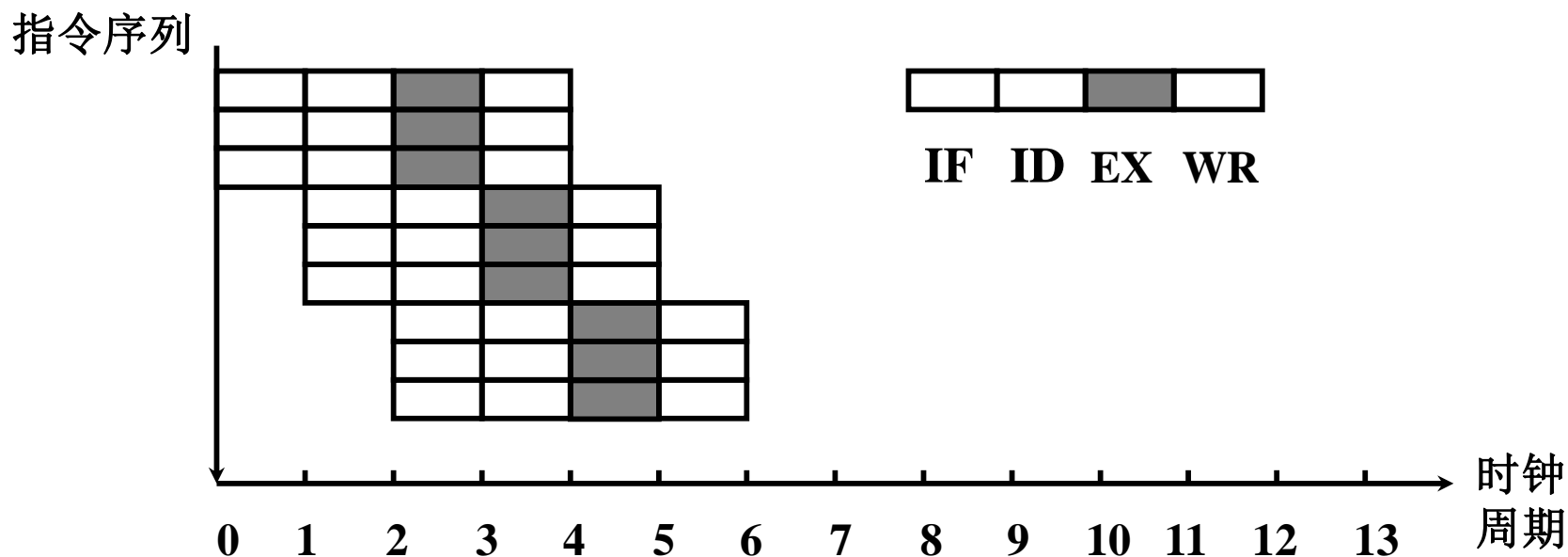
## 8.3

### 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，把可并行执行的指令搭配起来



## 2. 超流水线技术

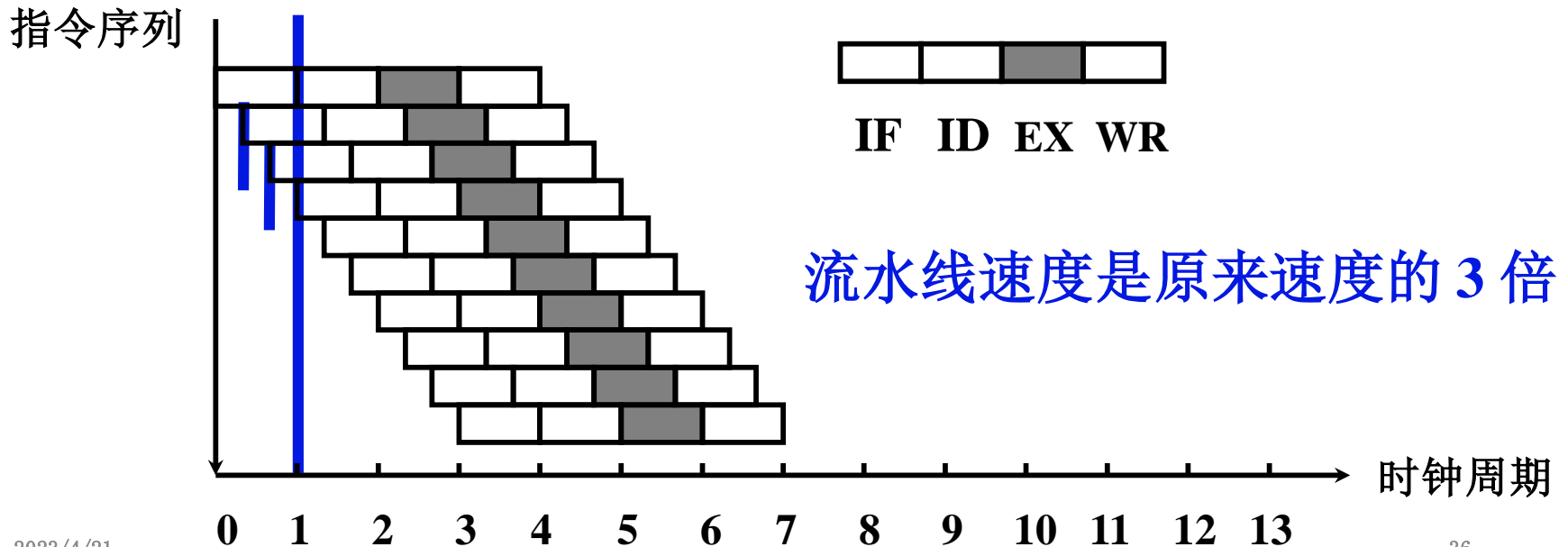
## 8.3

- 在一个时钟周期内再分段（3段）

在一个时钟周期内一个功能部件使用多次（3次）

- 不能调整指令的执行顺序

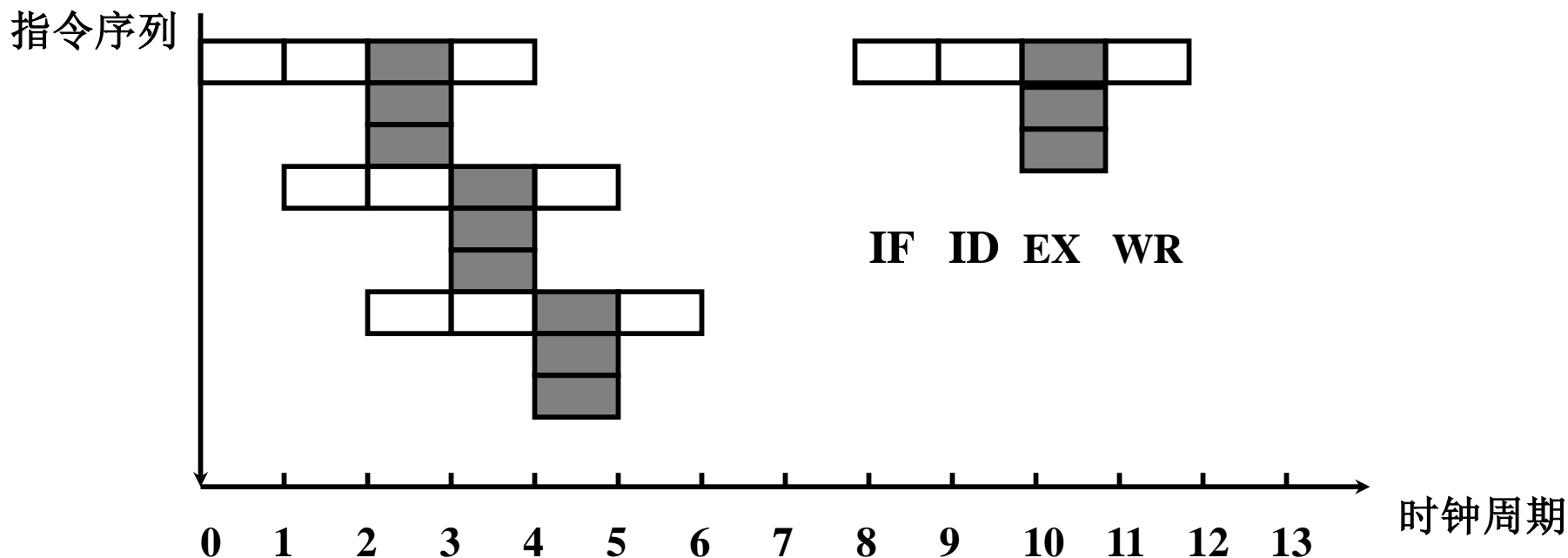
靠编译程序解决优化问题



### 3. 超长指令字技术

## 8.3

- 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 **多个处理部件**

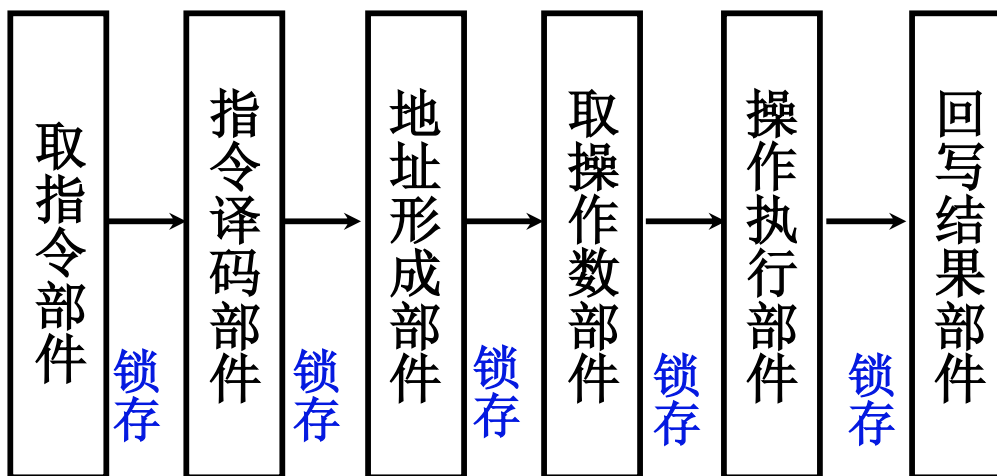


# 六、流水线结构

## 8.3

### 1. 指令流水线结构

完成一条指令分 6 段，每段需一个时钟周期



若 流水线不出现断流

1 个时钟周期出 1 结果

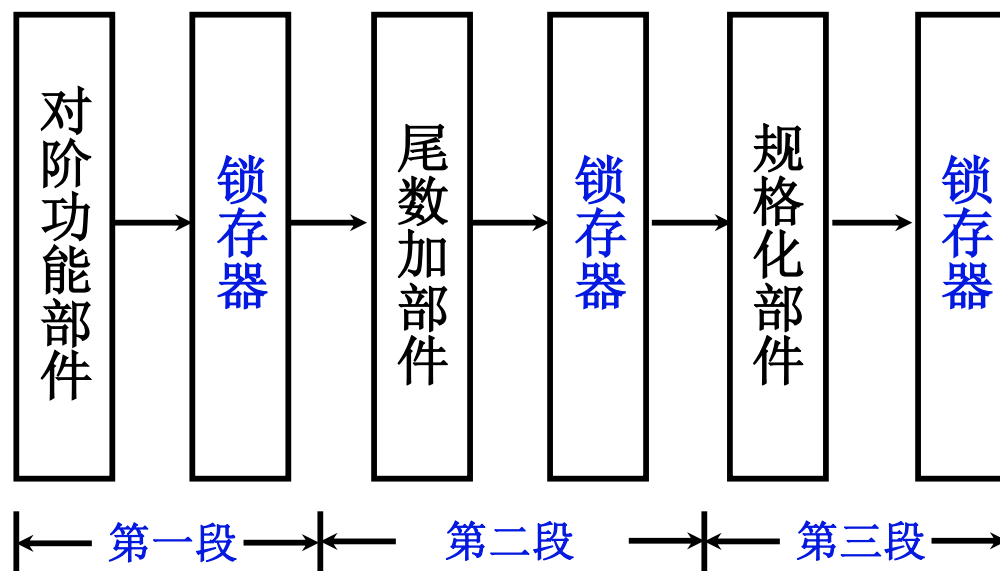
不采用流水技术

6 个时钟周期出 1 结果

理想情况下，6 级流水 的速度是不采用流水技术的 6 倍

## 2. 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



分段原则 每段 操作时间 尽量 一致

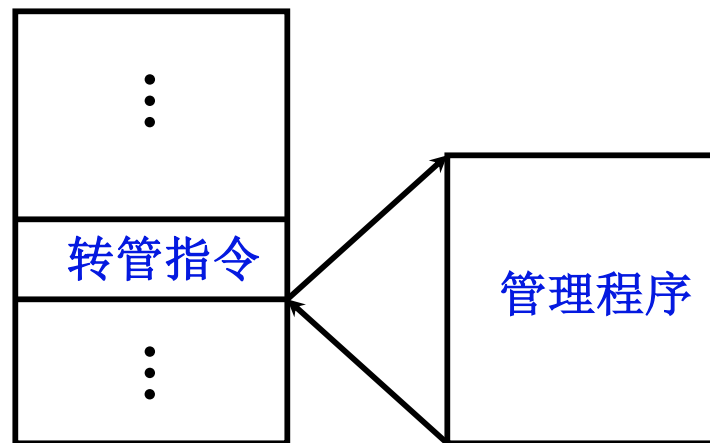
# 8.4 中断系统

## 一、概述

### 1. 引起中断的各种因素

#### (1) 人为设置的中断

如 转管指令



(2) 程序性事故 溢出、操作码不能识别、除法非法

(3) 硬件故障

(4) I/O 设备

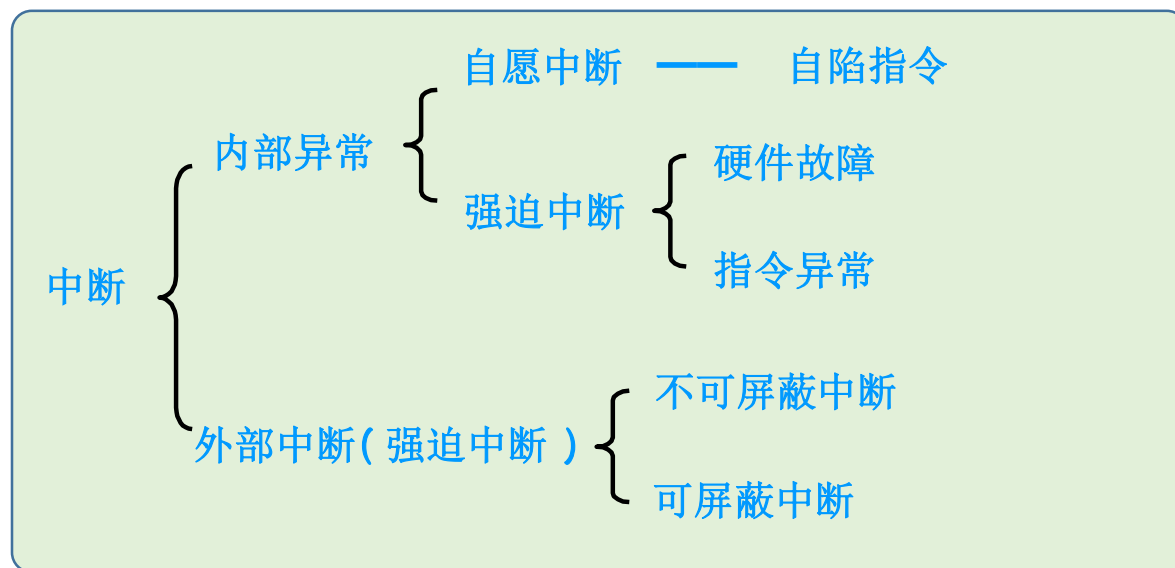
(5) 外部事件 用 键盘中断 现行程序



# 中断的分类与作用

- 中断技术赋予计算机应变能力，将有序的运行和无序的事件统一起来，大大增强了系统的处理能力

- 主机外设并行工作
- 程序调试
- 故障处理
- 实时处理
- 人机交互



### • 内部异常

#### • 故障 (Fault)

- 由指令执行引起的异常，如未定义指令、越权指令、段故障、缺页故障、存储保护违例、数据未对齐、除数为零、浮点溢出、整数溢出等。
- 可恢复的故障，指令需恢复执行，不可恢复故障，进程被终止

#### • 自陷 (Trap)

- 是一种事先安排的“异常”事件，通过在程序中显式的调用自陷指令触发自陷异常，用于在用户态下调用操作系统内核程序，如系统调用、条件陷阱指令。

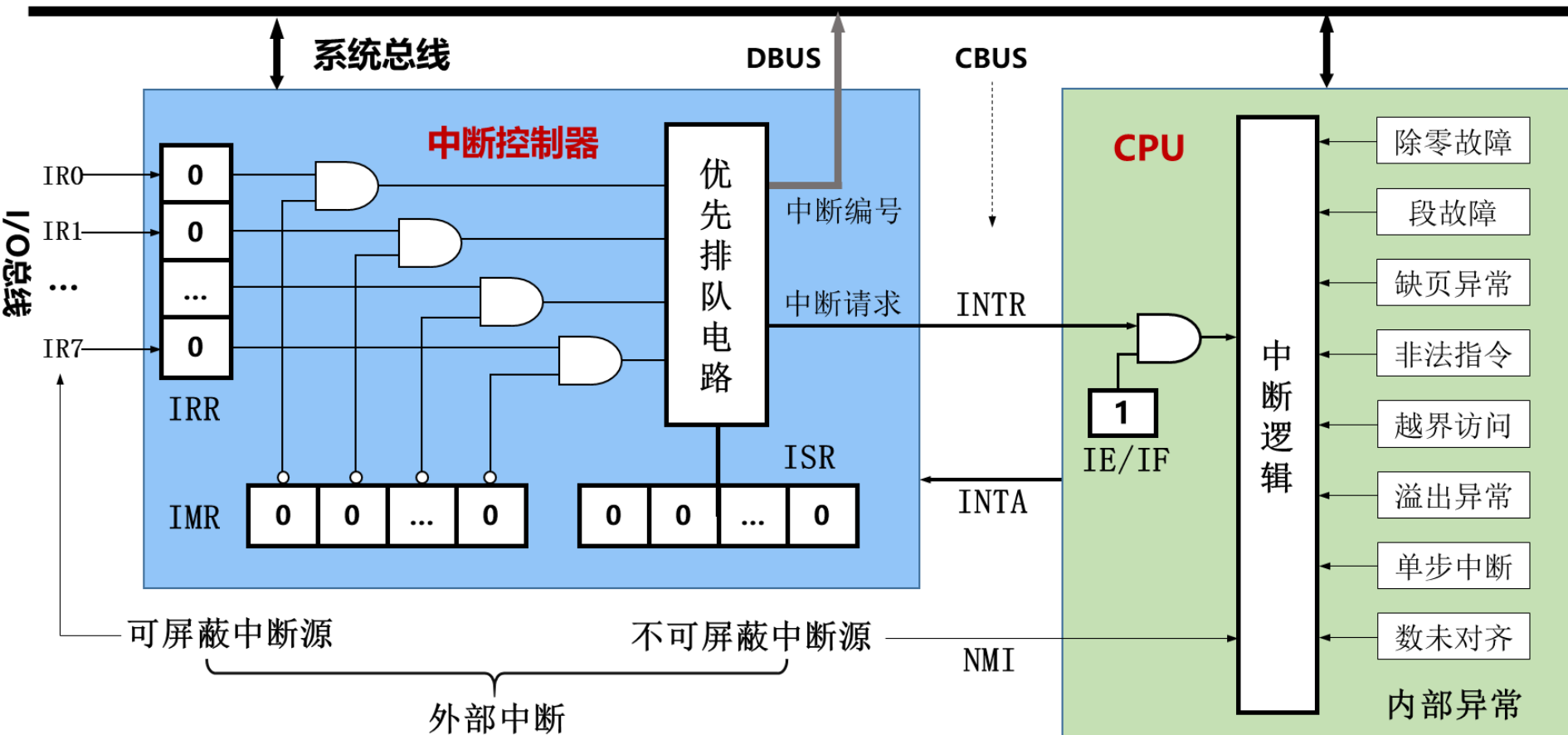
#### • 终止 (Abort)

- 随机出现的使得 CPU 无法继续执行的硬件故障，和具体指令无关。如机器校验错、总线错误、异常处理中再次异常的双错等，此时当前程序无法继续执行，只能终止执行，由异常服务处理程序来重启系统

### • 外部中断

# 中断硬件接口

## 8.4



## 2. 中断系统需解决的问题

- (1) 各中断源 如何 向 CPU 提出请求？
- (2) 各中断源 同时 提出 请求 怎么办？
- (3) CPU 什么 条件、什么 时间、以什么 方式 响应中断？
- (4) 如何 保护现场？
- (5) 如何 寻找入口地址？
- (6) 如何 恢复现场，如何 返回？
- (7) 处理中断的过程中又 出现新的中断 怎么办？

硬件 + 软件

## 二、中断请求标记和中断判优逻辑

## 8.4

### 1. 中断请求标记 **INTR**

一个请求源 一个 **INTR** 中断请求标记触发器

多个**INTR** 组成 中断请求标记寄存器

1	2	3	4	5			<i>n</i>
掉电	过热	主存读写校验错	阶上溢	非法除法		键盘输入	打印机输出

**INTR** 分散 在各个中断源的 接口电路中

**INTR** 集中在 **CPU** 的中断系统 内

## 2. 中断判优逻辑

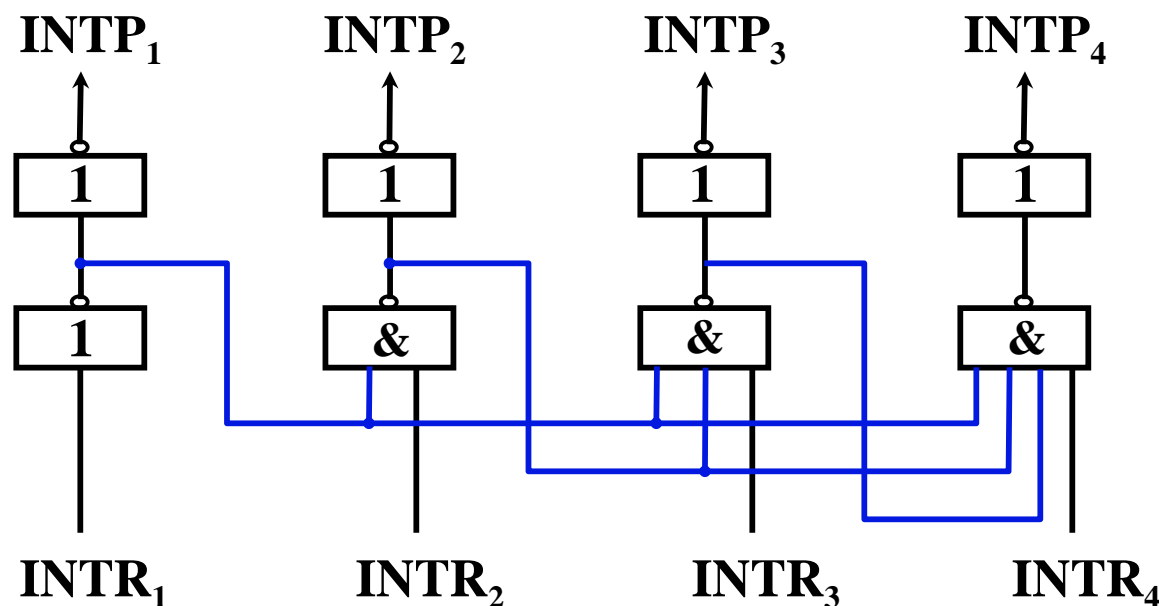
## 8.4

### (1) 硬件实现（排队器）

① 分散 在各个中断源的 接口电路中 链式排队器

参见 第五章

② 集中 在 CPU 内

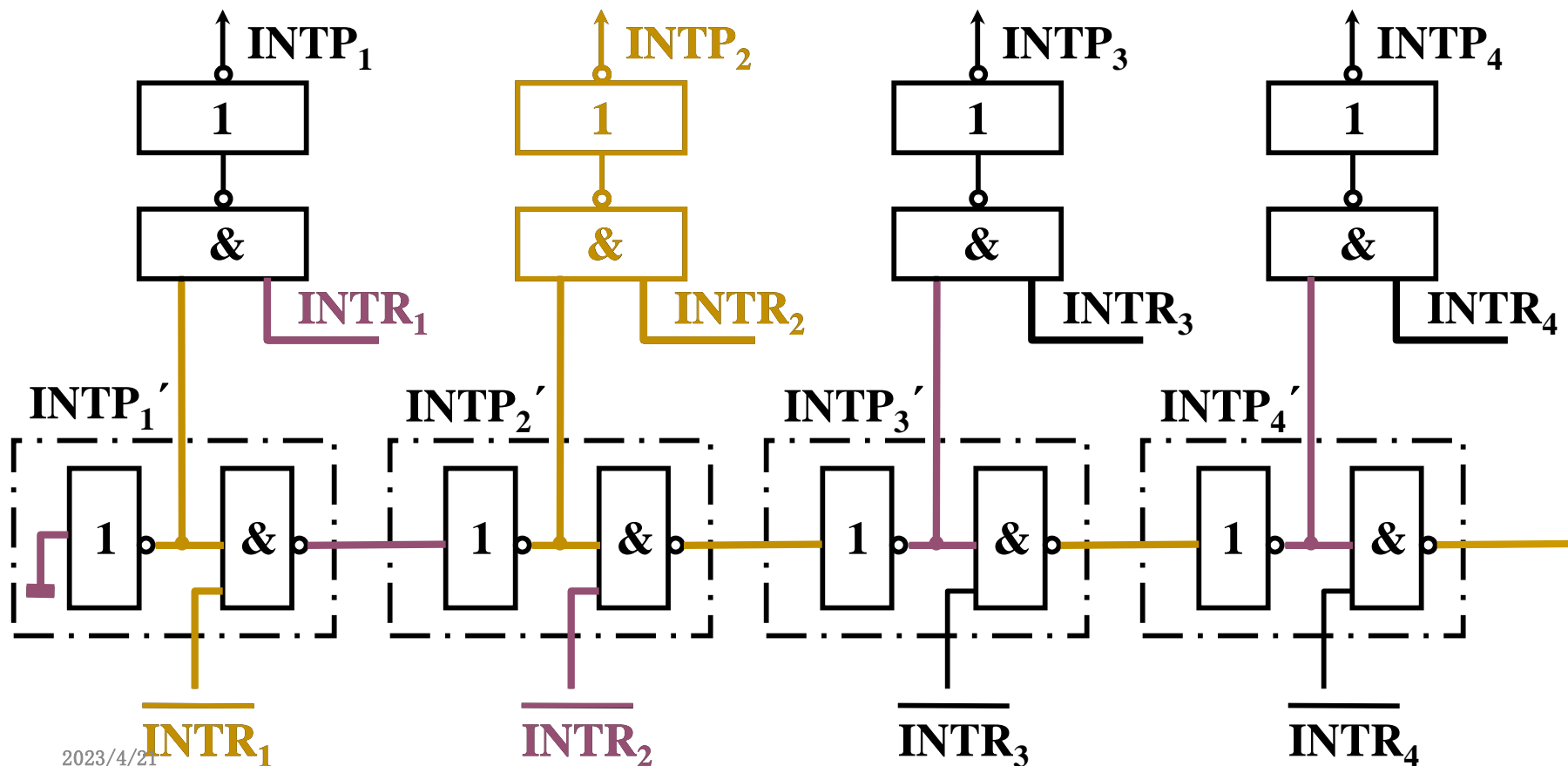


$INTR_1$ 、 $INTR_2$ 、 $INTR_3$ 、 $INTR_4$  优先级按降序排列

# 排队器

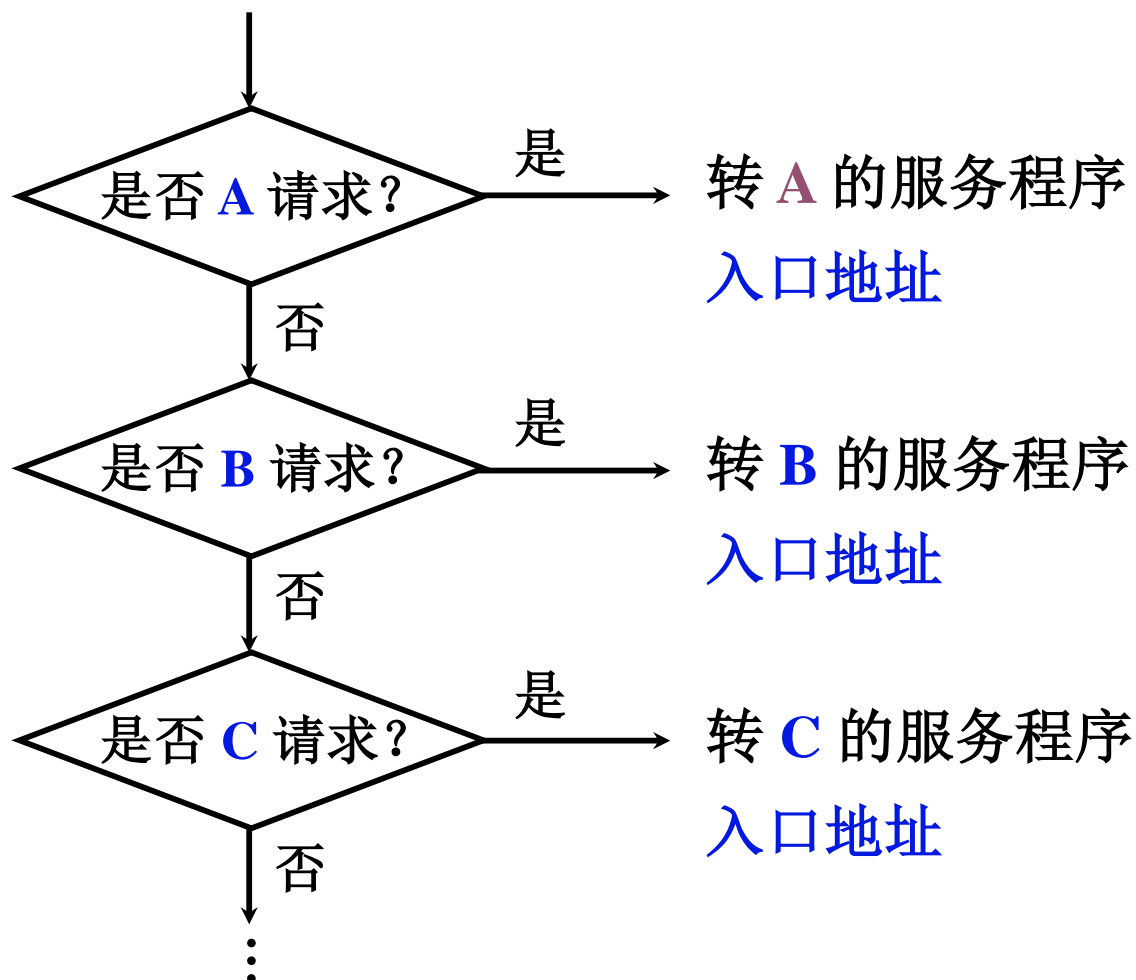
## 5.5

硬件 在 CPU 内或在接口电路中（链式排队器）



## (2) 软件实现（程序查询）

A、B、C 优先级按 降序 排列

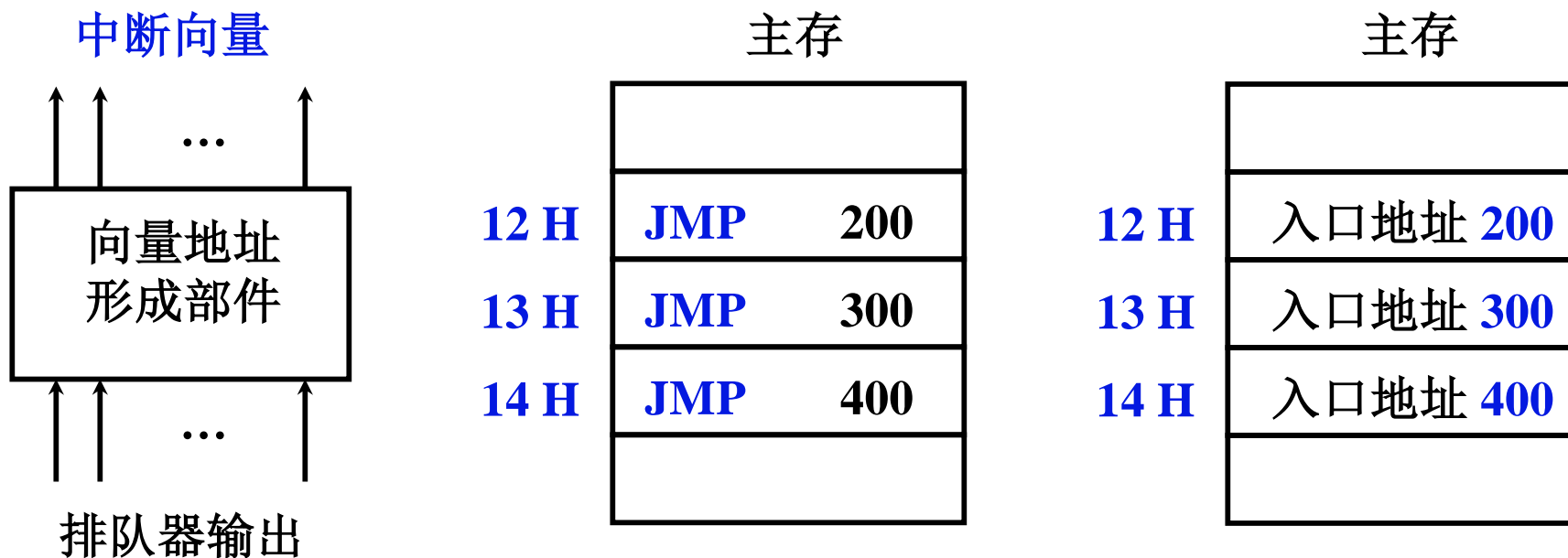




# 三、中断服务程序入口地址的寻找

## 8.4

### 1. 硬件向量法



向量地址 12H、13H、14H

入口地址 200、300、400

## 2. 软件查询法

八个中断源 1, 2, ... 8 按 降序 排列

中断识别程序（入口地址 **M**）

地 址	指 令	说 明
<b>M</b>	<b>SKP</b> DZ 1 <sup>#</sup>	1 <sup>#</sup> D = 0 跳（D为完成触发器）
	<b>JMP</b> 1 <sup>#</sup> SR	1 <sup>#</sup> D = 1 转1 <sup>#</sup> 服务程序
	<b>SKP</b> DZ 2 <sup>#</sup>	2 <sup>#</sup> D = 0 跳
	<b>JMP</b> 2 <sup>#</sup> SR	2 <sup>#</sup> D = 1 转2 <sup>#</sup> 服务程序
	⋮	
	<b>SKP</b> DZ 8 <sup>#</sup>	8 <sup>#</sup> D = 0 跳
	<b>JMP</b> 8 <sup>#</sup> SR	8 <sup>#</sup> D = 1 转8 <sup>#</sup> 服务程序

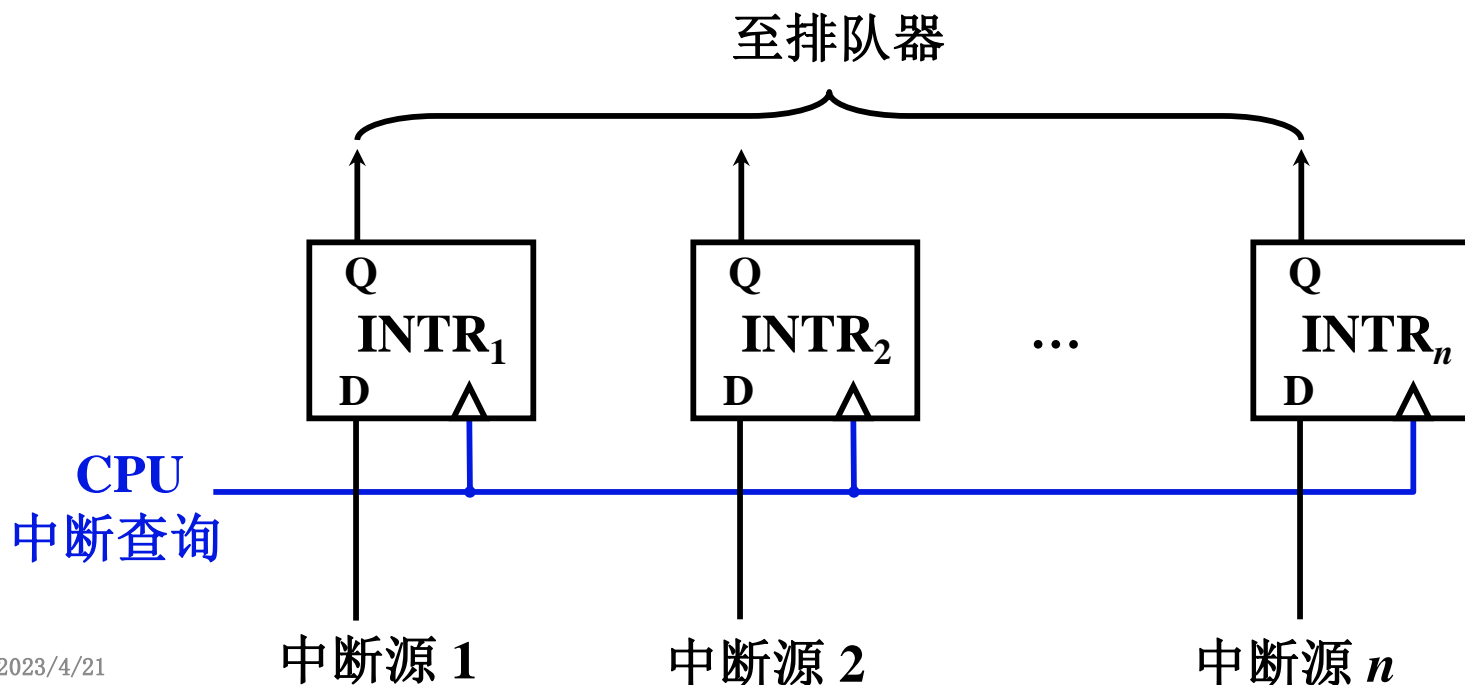
## 四、中断响应

### 1. 响应中断的条件

允许中断触发器  $EINT = 1$

### 2. 响应中断的时间

指令执行周期结束时刻由CPU发查询信号



### 3. 中断隐指令

8.4

#### (1) 保护程序断点

断点存于 特定地址（0 号地址）内      断点 进栈

#### (2) 寻找服务程序入口地址

向量地址  $\rightarrow$  PC （硬件向量法）

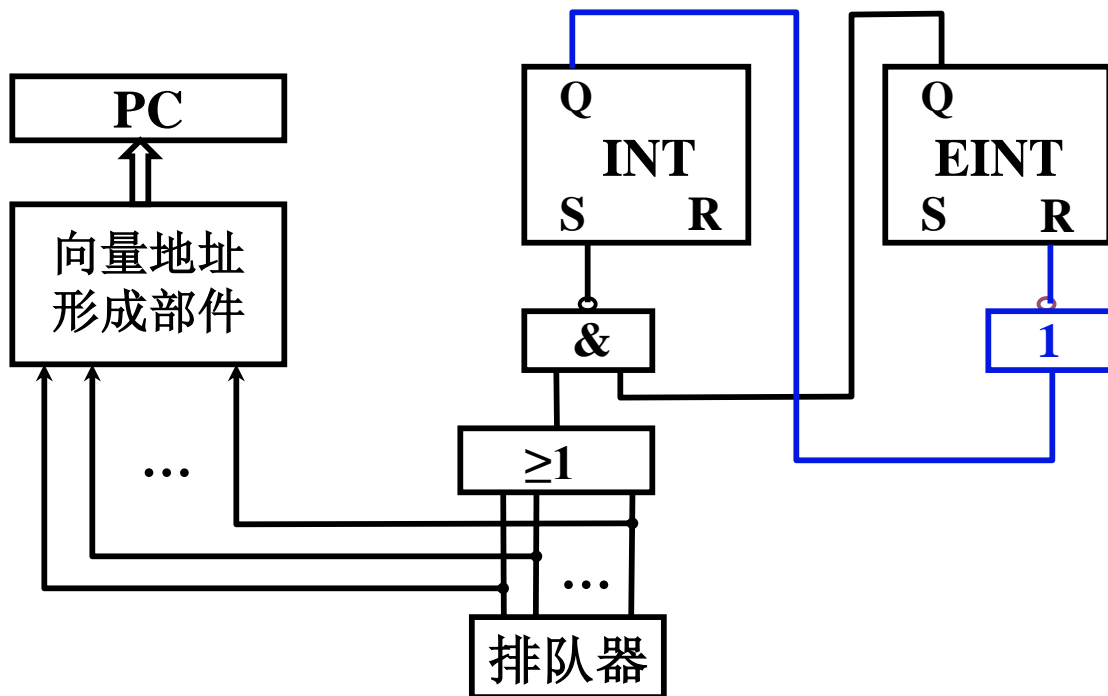
中断识别程序 入口地址  $M \rightarrow$  PC （软件查询法）

#### (3) 硬件 关中断

INT 中断标记

EINT 允许中断

R-S 触发器



# 五、保护现场和恢复现场

8.4

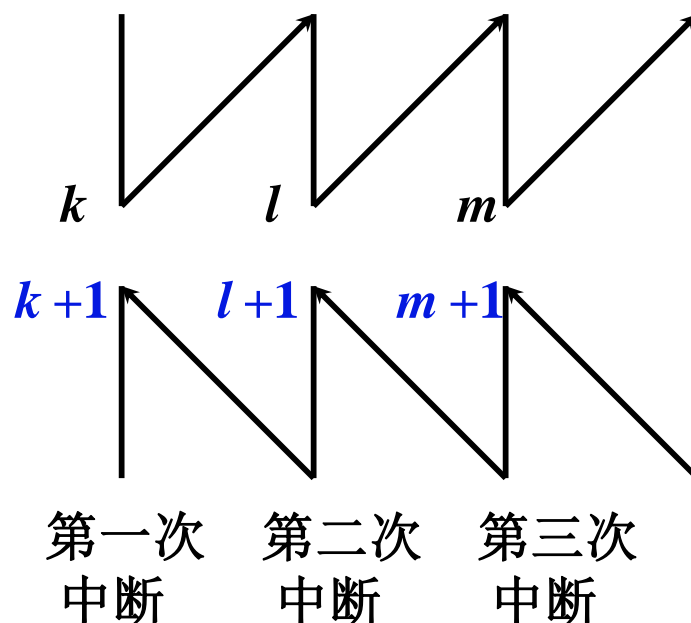
1. 保护现场 { 断点 中断隐指令 完成  
寄存器 内容 中断服务程序 完成
2. 恢复现场 中断服务程序 完成



# 六、中断屏蔽技术

## 1. 多重中断的概念

- 高优先级中断请求能否中断运行中的程序呢？
- 系统硬件、软件开销的权衡
  - ✓ 单重中断
    - 所有中断源均属同一级，离CPU近的优先级高
    - CPU处理某个中断时，不响应其他中断
  - ✓ 多重中断
    - 优先级高的中断可以打断优先级低的中断服务程序
    - 中断嵌套

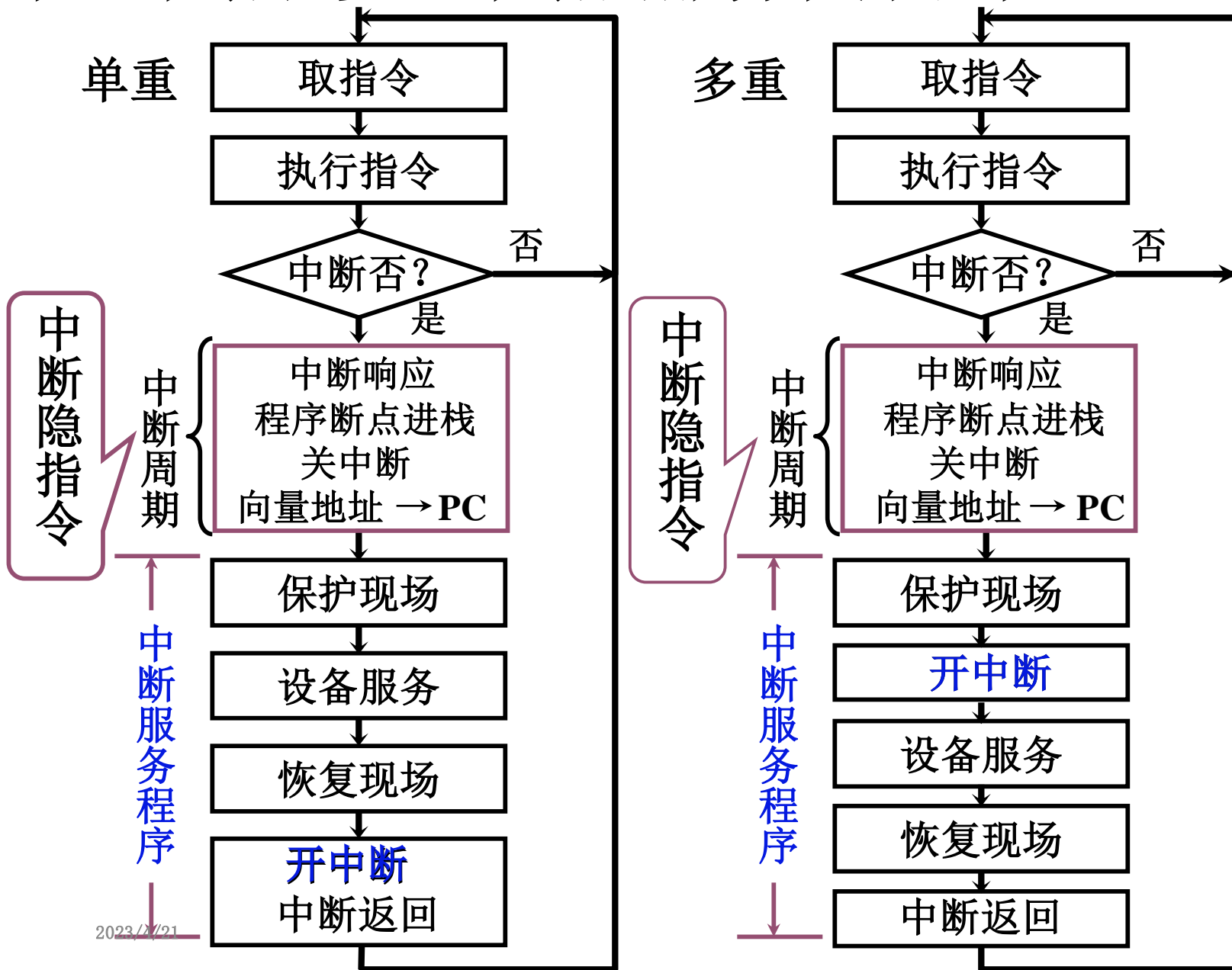


程序断点  $k+1$ ,  $l+1$ ,  $m+1$

# 划分优先级的一般规律

- 硬件故障中断属于最高级，其次是程序错误中断
- 非屏蔽中断优于可屏蔽中断
- DMA请求优先于I/O设备传送的中断请求
- 高速设备优于低速设备
- 输入设备的中断优于输出设备
- 实时设备优先于普通设备

# 单重中断和多重中断的服务程序流程



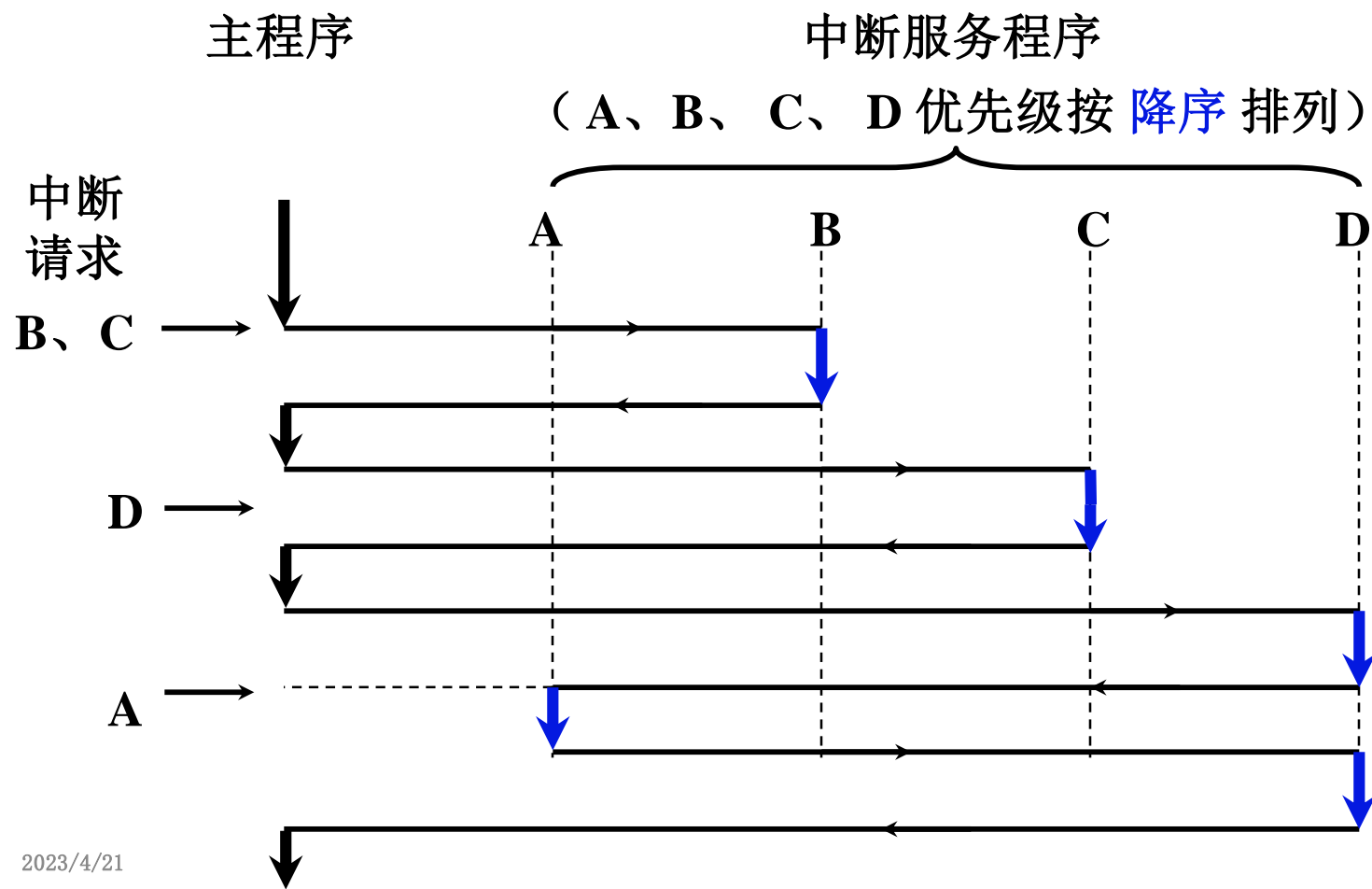


## 2. 实现多重中断的条件

8.4

(1) 提前 设置 开中断 指令

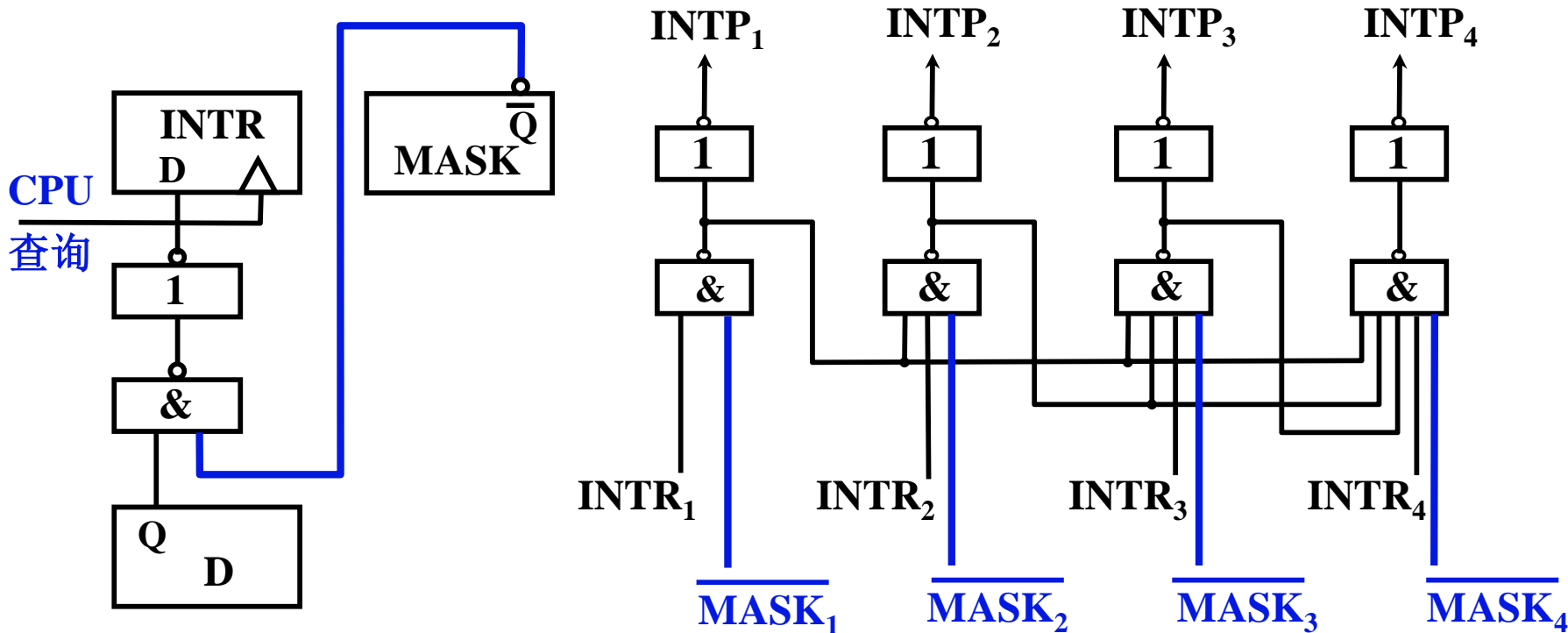
(2) 优先级别高 的中断源 有权中断优先级别低 的中断源



# 3. 屏蔽技术

## 8.4

### (1) 屏蔽触发器的作用



**MASK = 0**（未屏蔽）

**INTR** 能被置“1”

**MASK<sub>i</sub> = 1**（屏蔽）

**INTP<sub>i</sub> = 0**（不能被排队选中）

## (2) 屏蔽字

16个中断源 1, 2, 3, ..., 16 按 降序 排列

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

### (3) 屏蔽技术可改变处理优先等级

## 8.4

响应优先级      不可改变

处理优先级      可改变（通过重新设置屏蔽字）

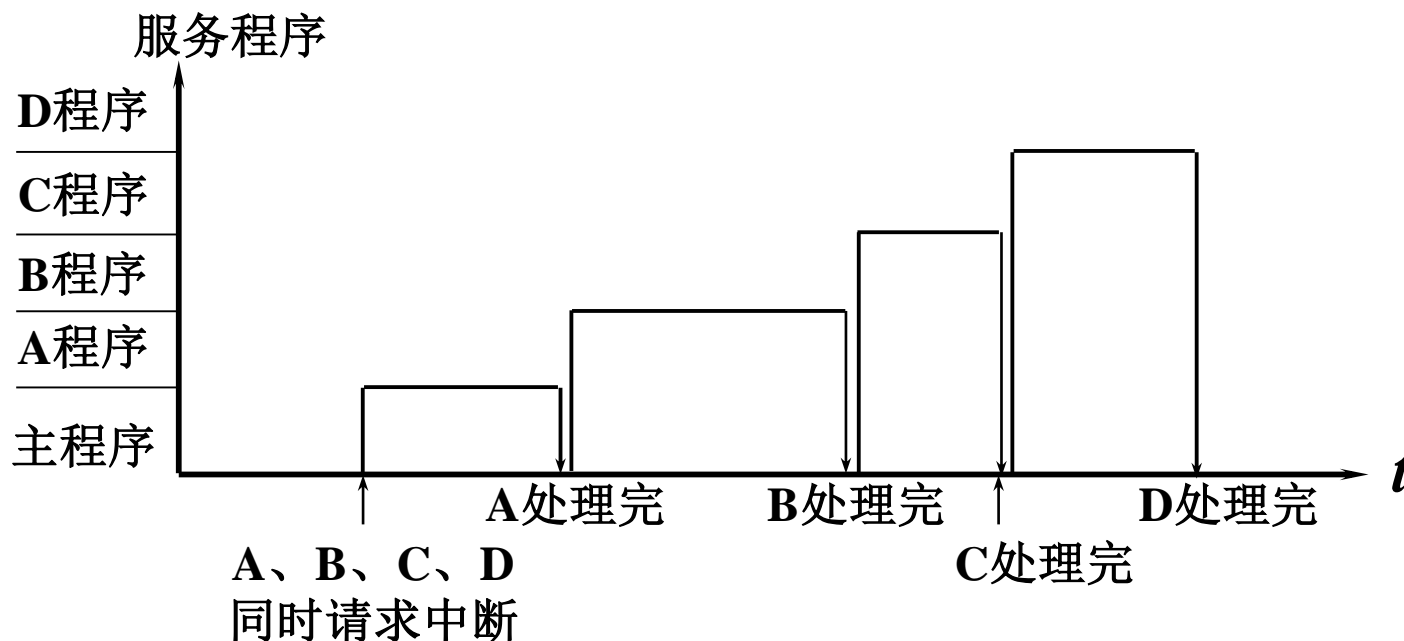
中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1

响应优先级 **A→B→C→D** 降序排列

处理优先级 **A→D→C→B** 降序排列

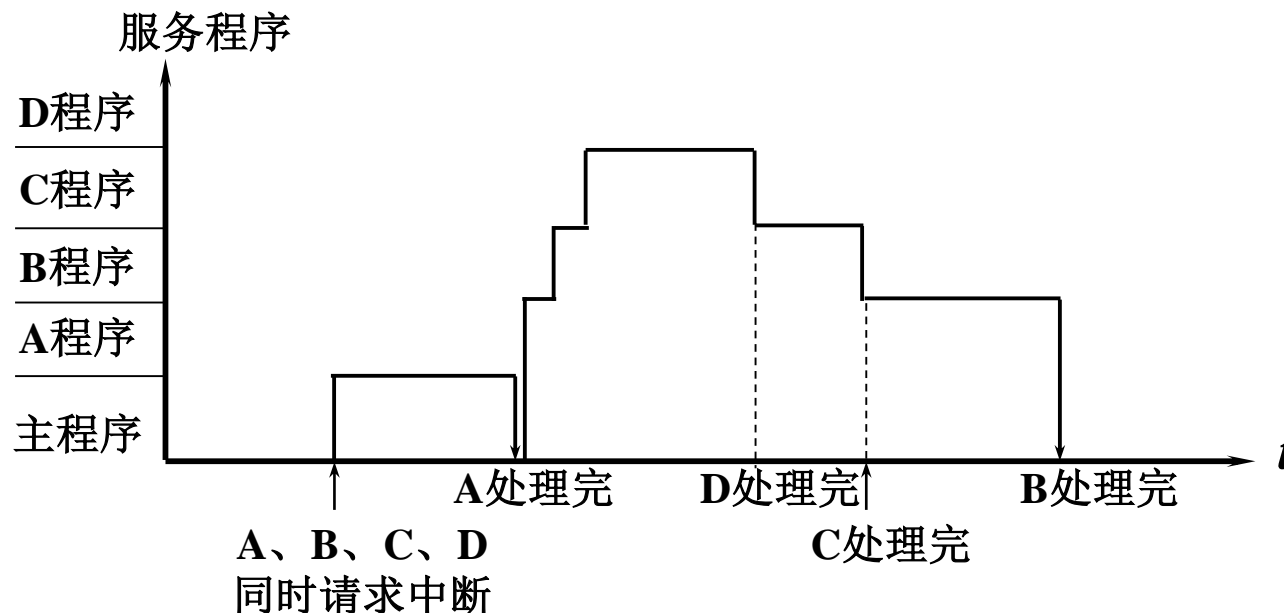
### (3) 屏蔽技术可改变处理优先等级

8.4



CPU 执行程序轨迹（原屏蔽字）

### (3) 屏蔽技术可改变处理优先等级

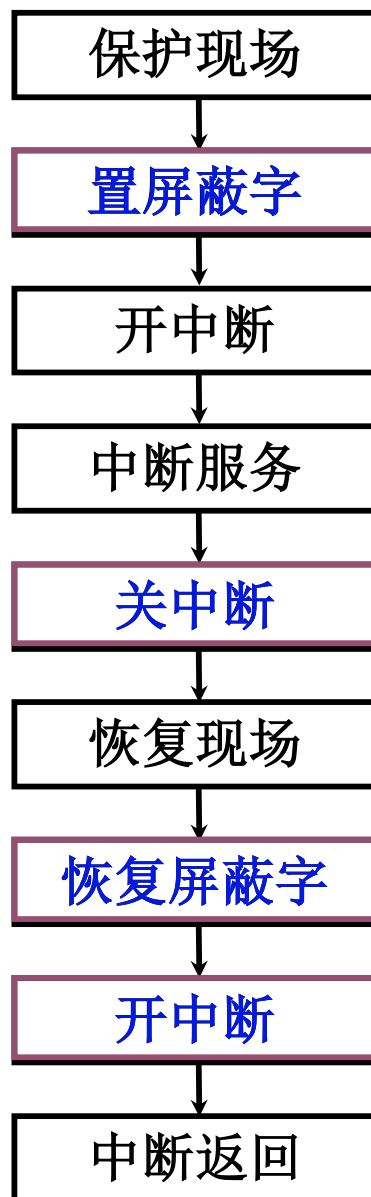


CPU 执行程序轨迹（新屏蔽字）

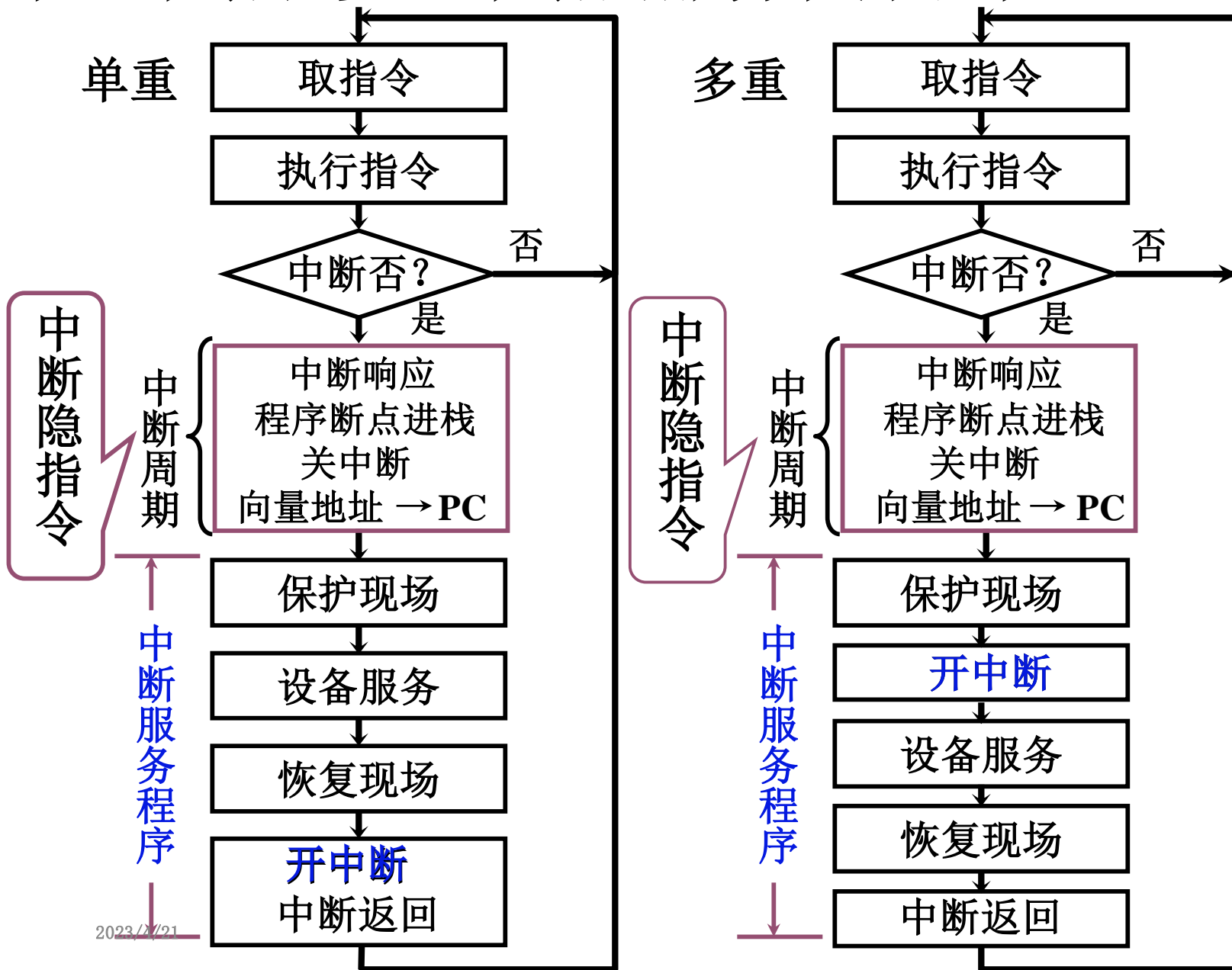
### (4) 屏蔽技术的其他作用

可以 **人为地屏蔽** 某个中断源的请求  
便于程序控制

## (5) 新屏蔽字的设置



# 单重中断和多重中断的服务程序流程





## 4. 多重中断的断点保护

(1) 断点进栈                      中断隐指令 完成

(2) 断点存入 “0” 地址      中断隐指令 完成

中断周期       $0 \rightarrow \text{MAR}$

命令存储器写

$\text{PC} \rightarrow \text{MDR}$       断点  $\rightarrow \text{MDR}$

$(\text{MDR}) \rightarrow \text{存入存储器}$

三次中断，三个断点都存入 “0” 地址

？ 如何保证断点不丢失？

### (3) 程序断点存入 “ 0 ” 地址的断点保护8.4

地 址	内 容	说 明
0	××××	存程序断点
5	JMP SERVE	5 为向量地址
SERVE	STA SAVE	保护现场
	⋮	
置屏蔽字	LDA 0	} 0 地址内容转存
	STA RETURN	
	ENI	开中断
	⋮	} 其他服务内容
	LDA SAVE	
	JMP @ RETURN	恢复现场
SAVE	××××	间址返回
RETURN	××××	存放 ACC 内容
		转存 0 地址内容

# 总结：中断系统需解决的问题

## 8.4

- (1) 各中断源 如何 向 CPU 提出请求？
- (2) 各中断源 同时 提出 请求 怎么办？
- (3) CPU 什么 条件、什么 时间、以什么 方式 响应中断？
- (4) 如何 保护现场？
- (5) 如何 寻找入口地址？
- (6) 如何 恢复现场，如何 返回？
- (7) 处理中断的过程中又 出现新的中断 怎么办？

硬件 + 软件