

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

指令控制

操作控制

时间控制

处理中断

数据加工

2. 运算器的功能

实现算术运算和逻辑运算

二、CPU 结构框图

1. CPU 与系统总线

指令控制

PC IR

操作控制

CU 时序电路

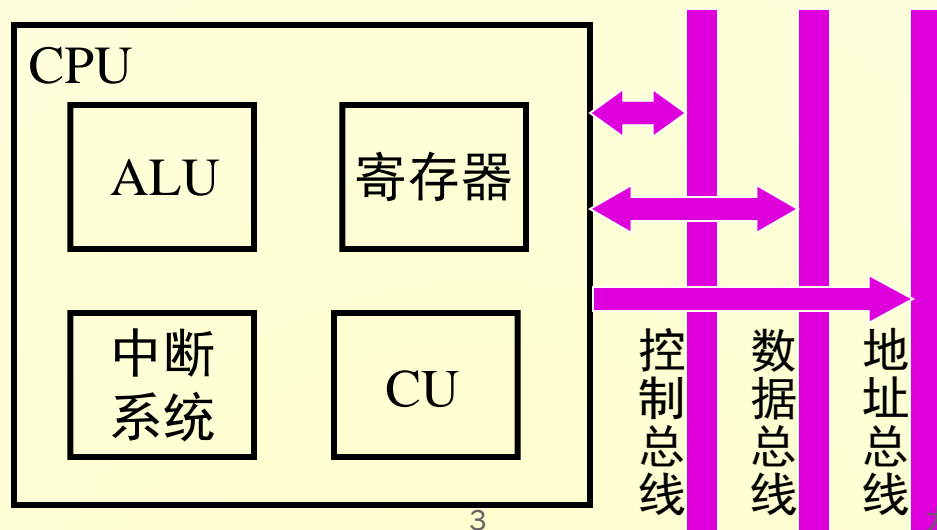
时间控制

数据加工

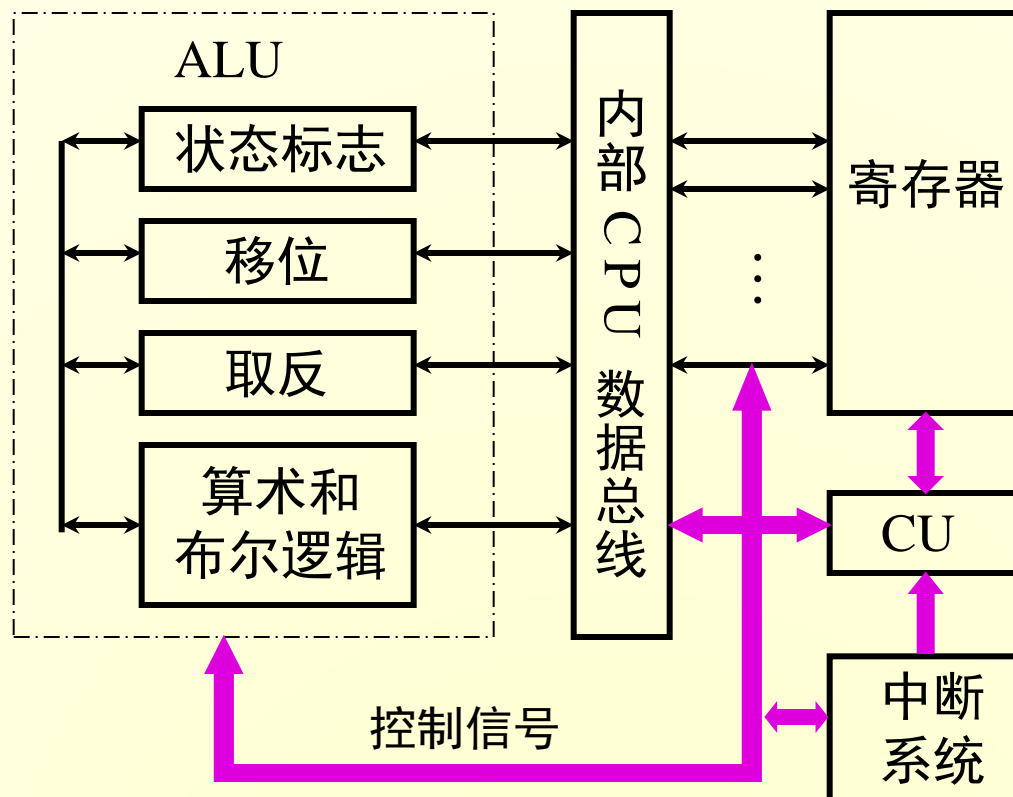
ALU 寄存器

处理中断

中断系统



2. CPU 的内部结构



三、CPU 的寄存器

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 段基值 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4篇

2. 中断系统

参见 8.4 节

五、ALU

参见 第6章

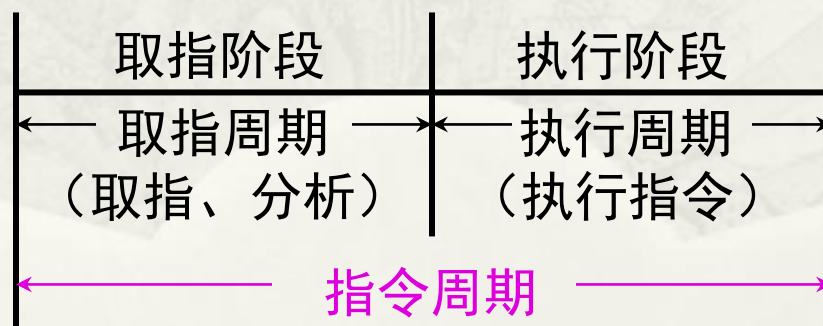
8.2 指令周期

一、指令周期的基本概念

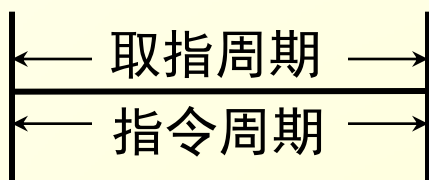
1. 指令周期

取出并执行一条指令所需的全部时间

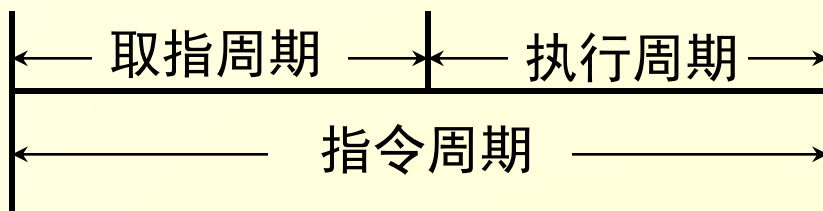
完成一条指令 { 取指、分析 取指周期
 执行 执行周期



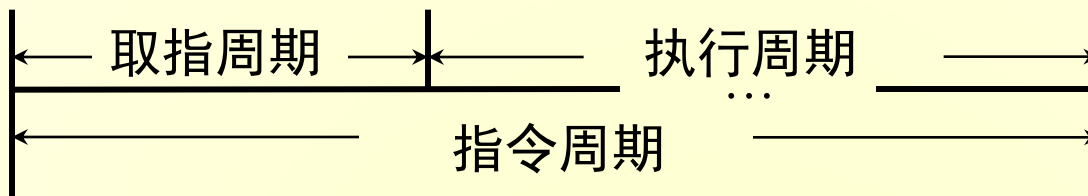
2. 每条指令的指令周期不同



NOP

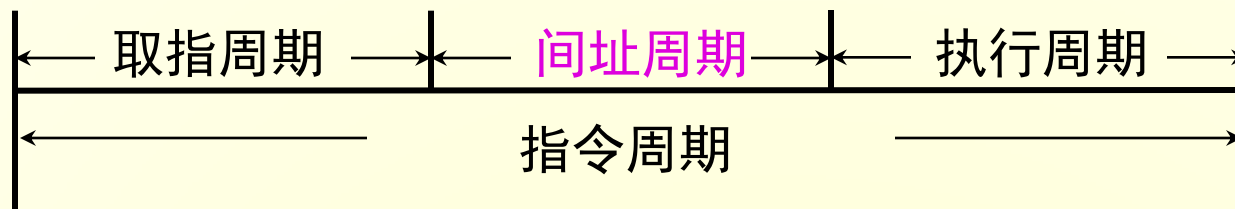


ADD mem

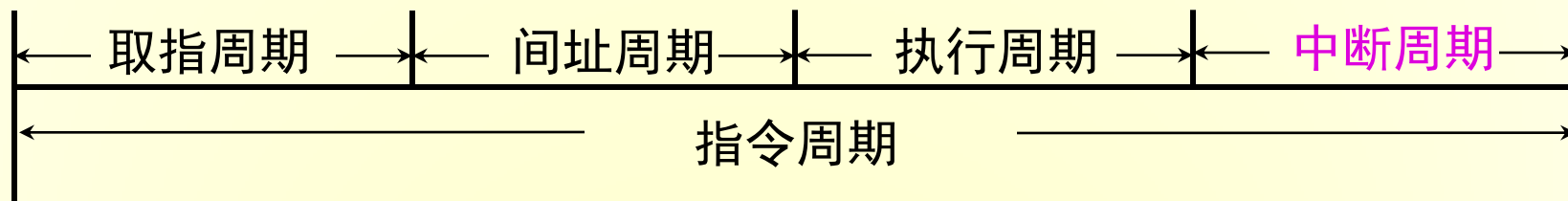


MUL mem

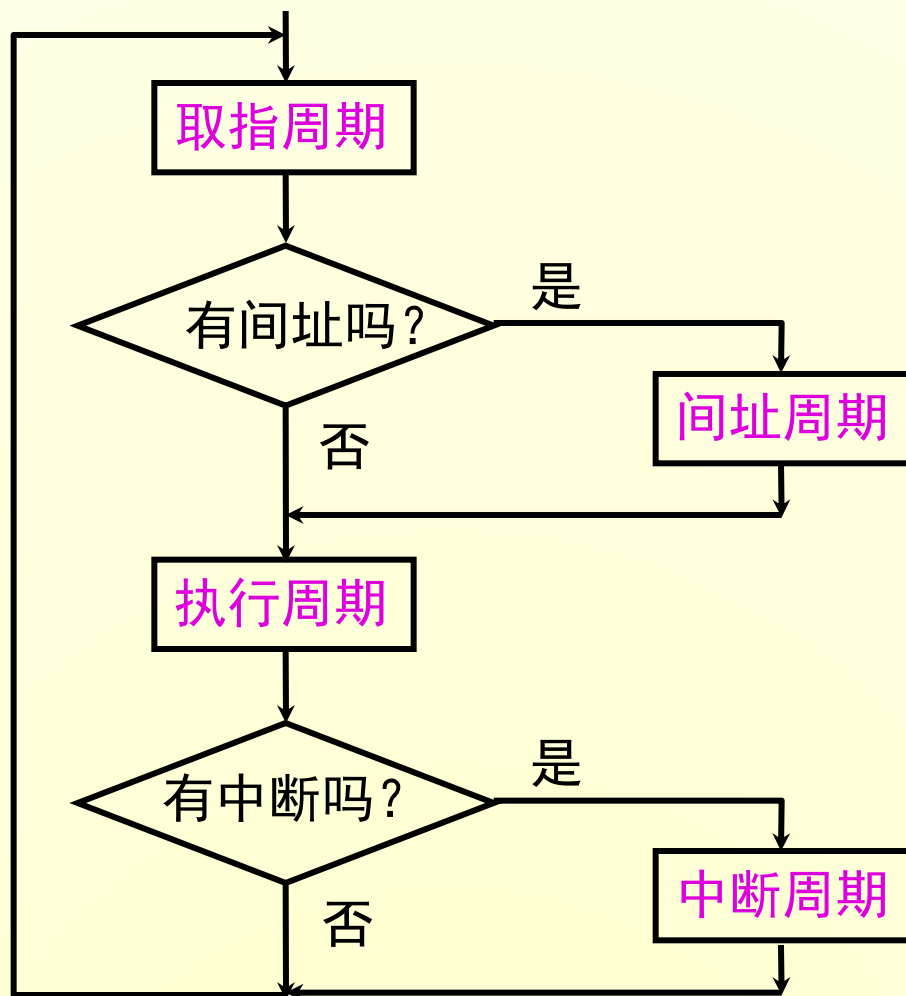
3. 具有间接寻址的指令周期



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

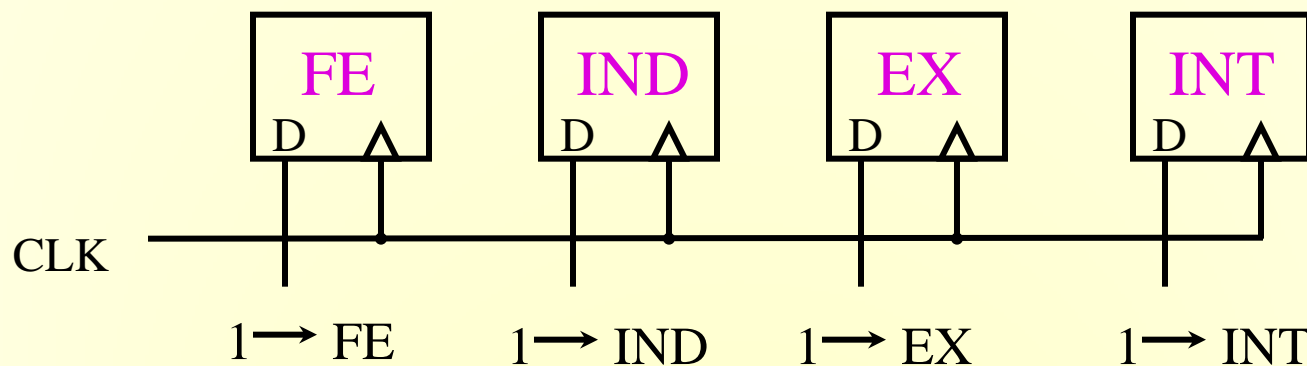
执行周期

存 程序断点

中断周期

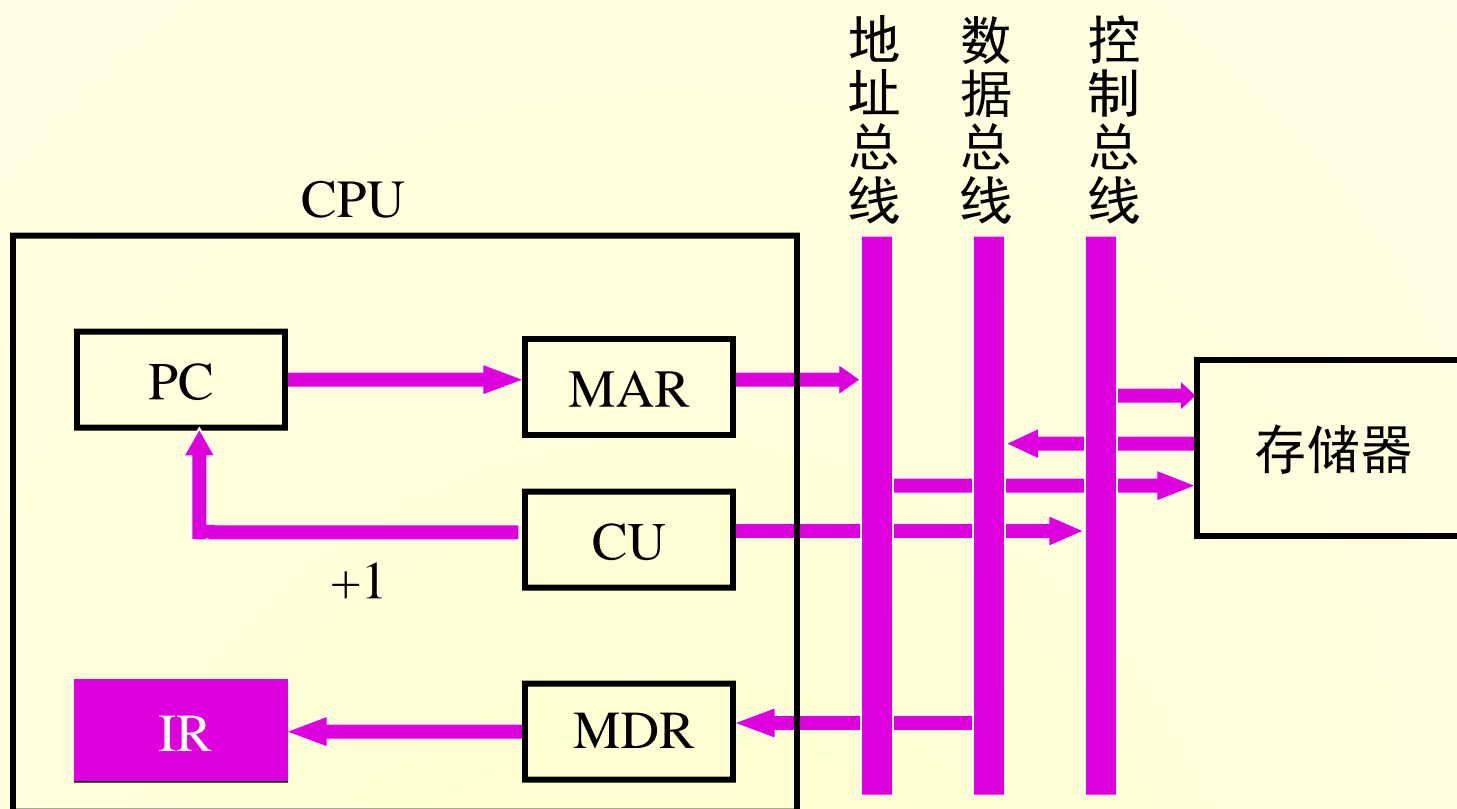
CPU 的

4个工作周期

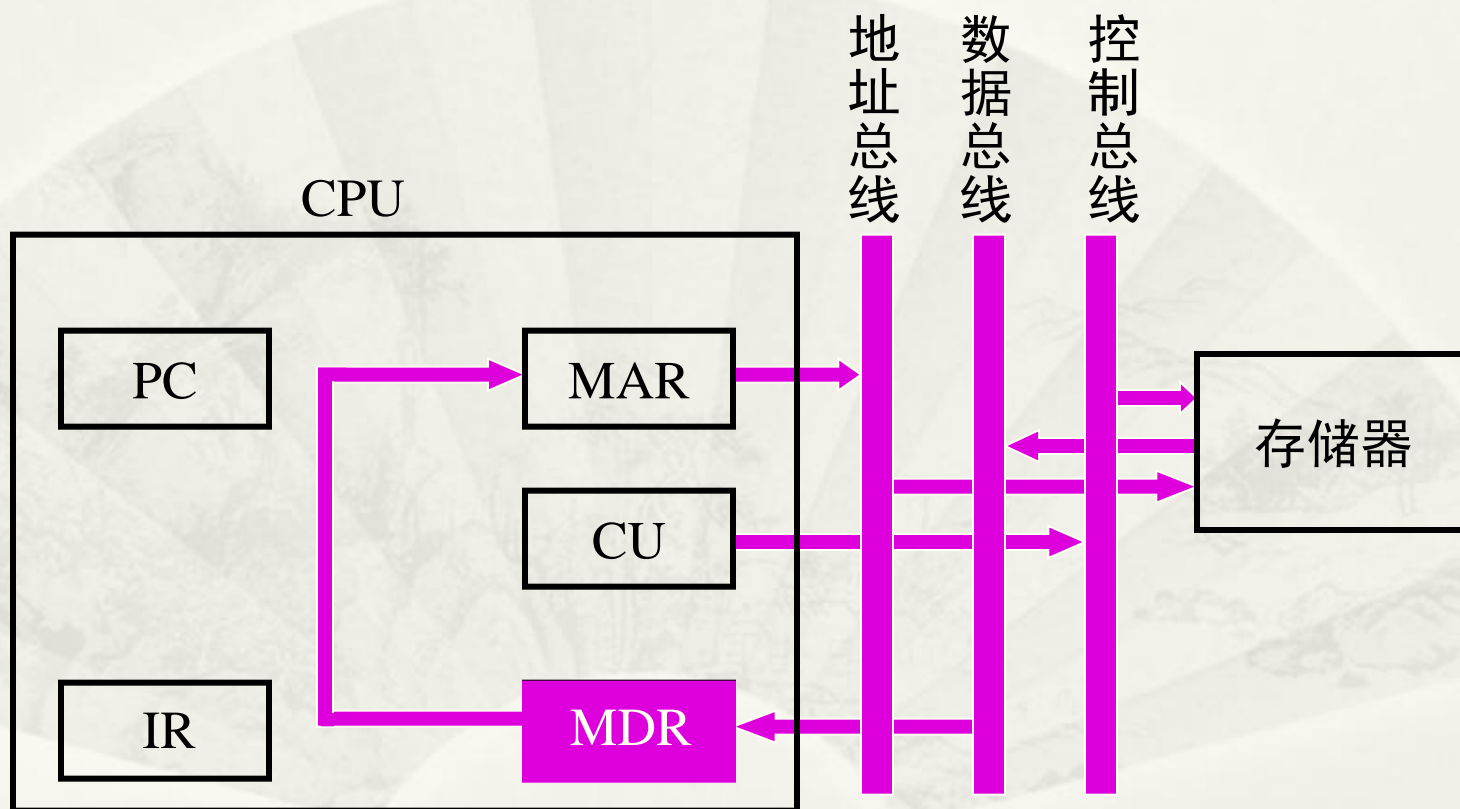


二、指令周期的数据流

1. 取指周期数据流



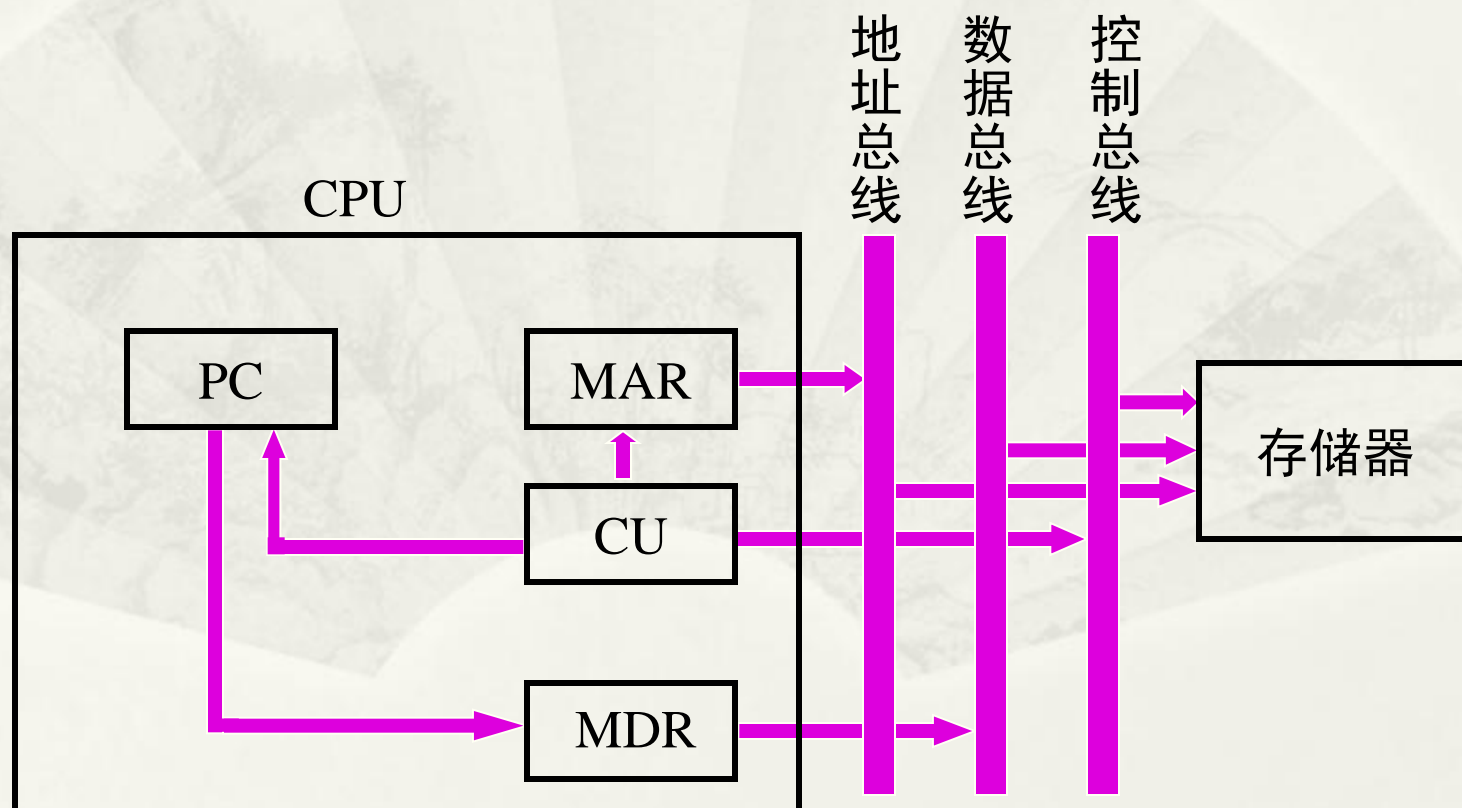
2. 间址周期数据流



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

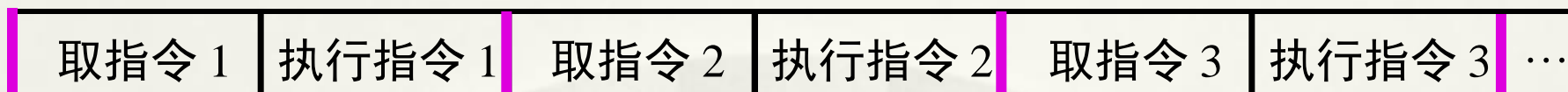
2. 并行性的等级

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

三、指令流水原理

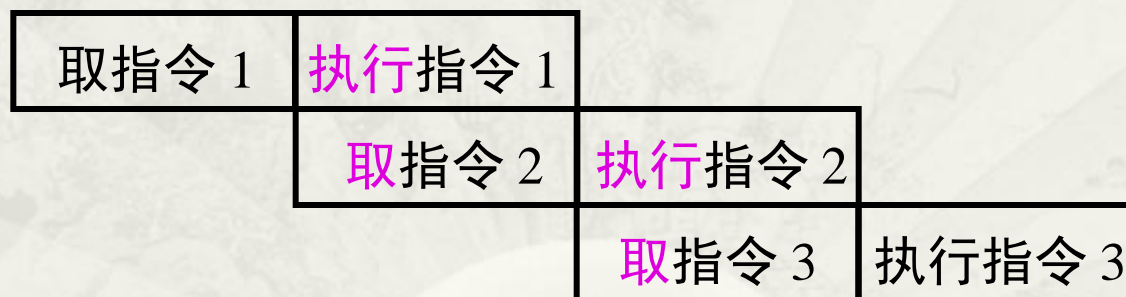
8.3

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



指令预取

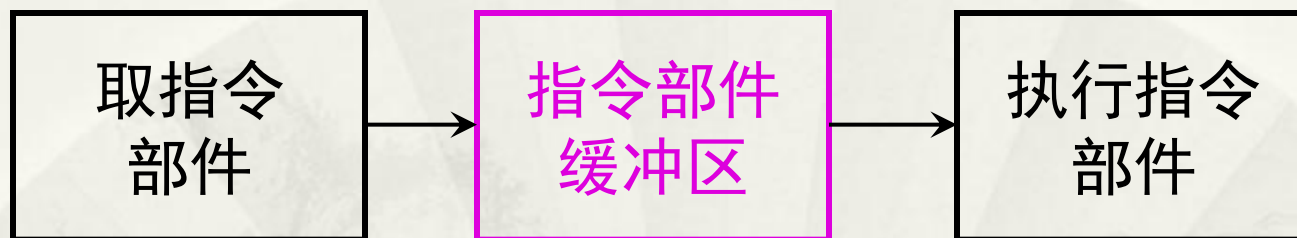
若 取指 和 执行 阶段时间上 完全重叠

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



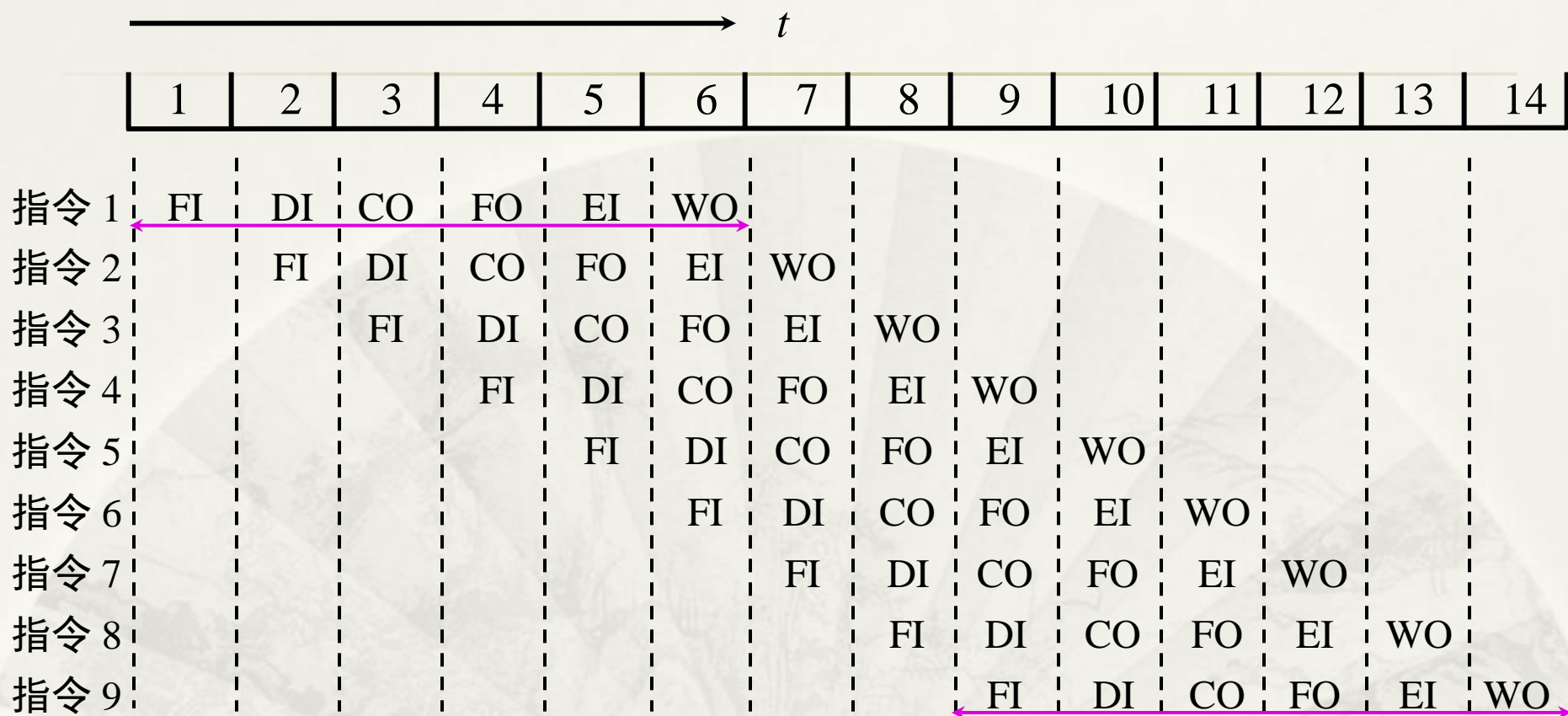
(2) 条件转移指令 对指令流水的影响

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

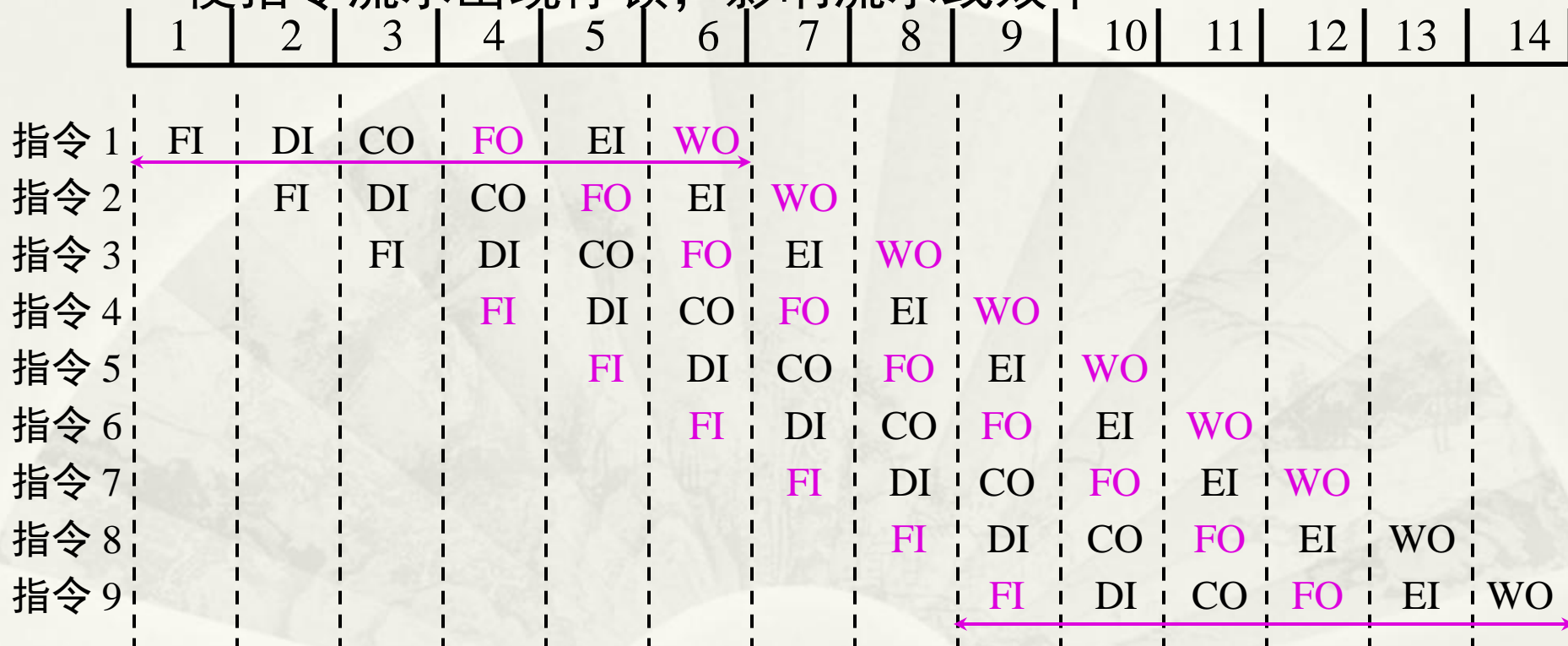
14 个时间单位

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

使指令流水出现停顿，影响流水线效率



解决办法

- 停顿
- 指令 1 与指令 4 冲突
- 指令 2 与指令 5 冲突
- 指令 1、指令 3、指令 6 冲突
- 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	;	$(R_4) - (R_5) \rightarrow R_3$

解决办法

- 后推法

- 采用 旁路技术

3. 控制相关

8.3

由转移指令引起

→ M

LDA	# 0
LDX	# 0
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 Δt

- 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 Δt

完成 n 条指令在 m 段流水线上共需

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

完成 n 条指令在等效的非流水线上共需

$$T' = nm \Delta t$$

则

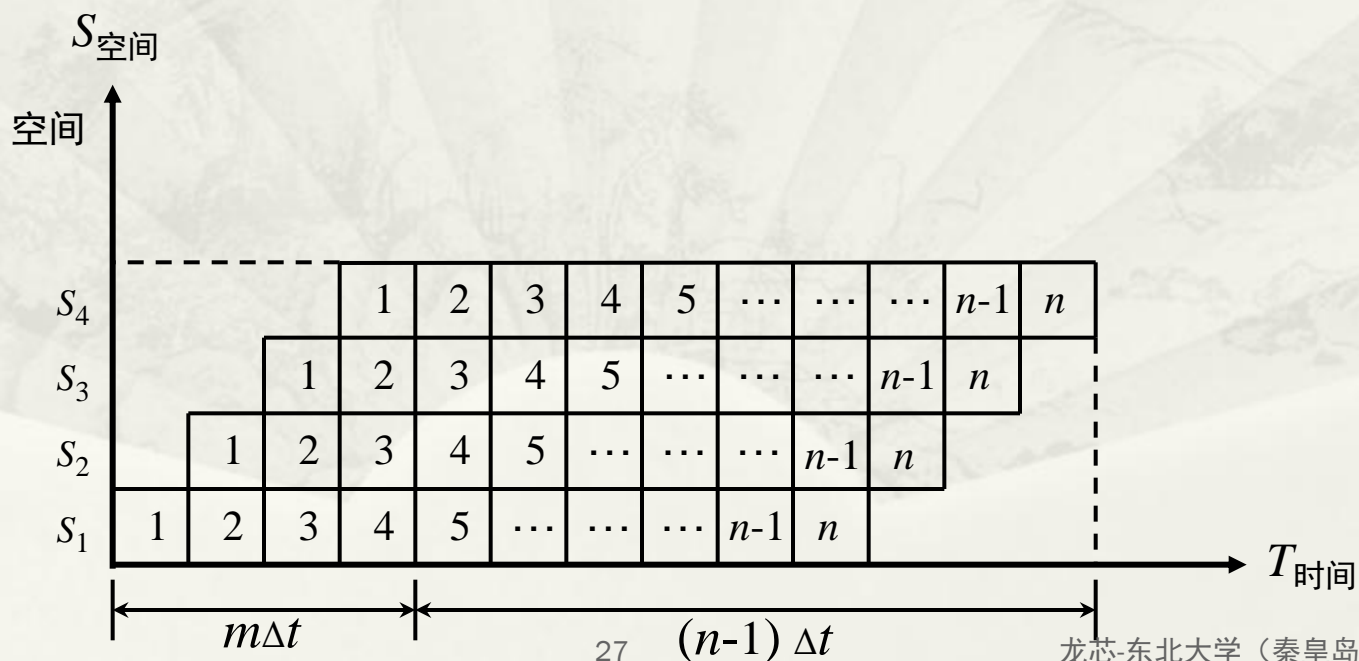
$$S_p = \frac{nm \Delta t}{m \Delta t + (n-1) \Delta t} = \frac{nm}{m + n - 1}$$

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



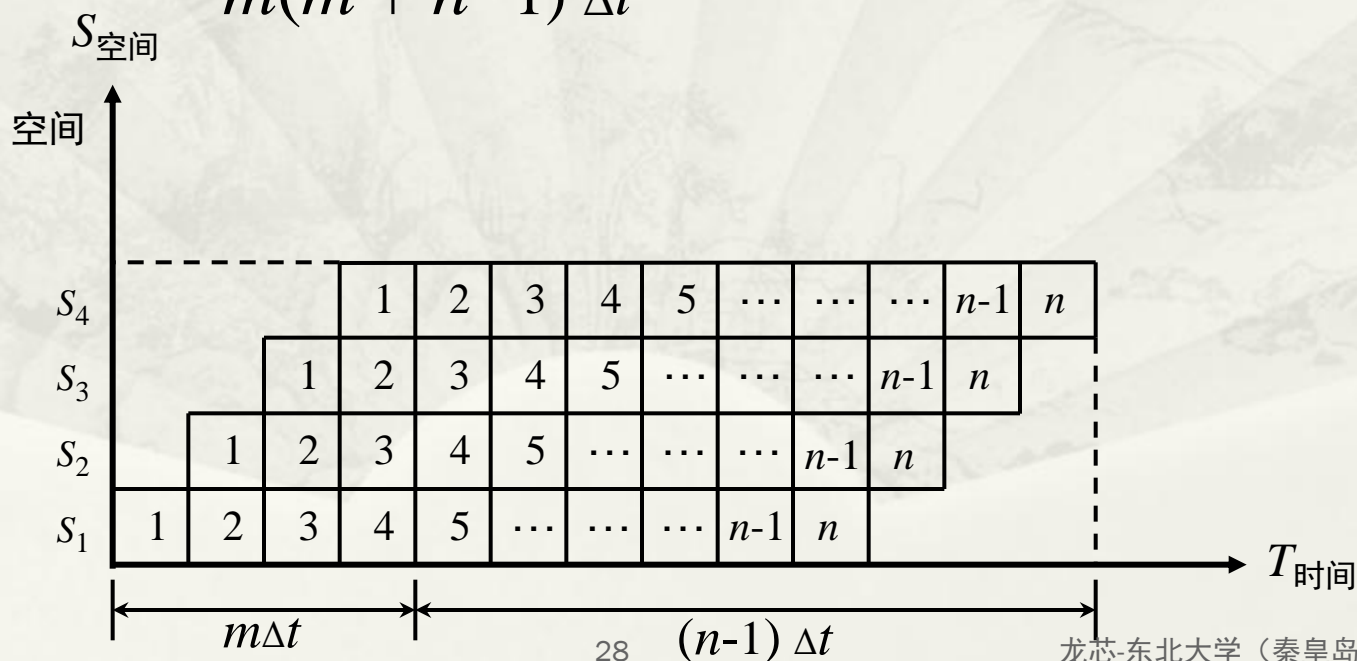
3. 效率

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

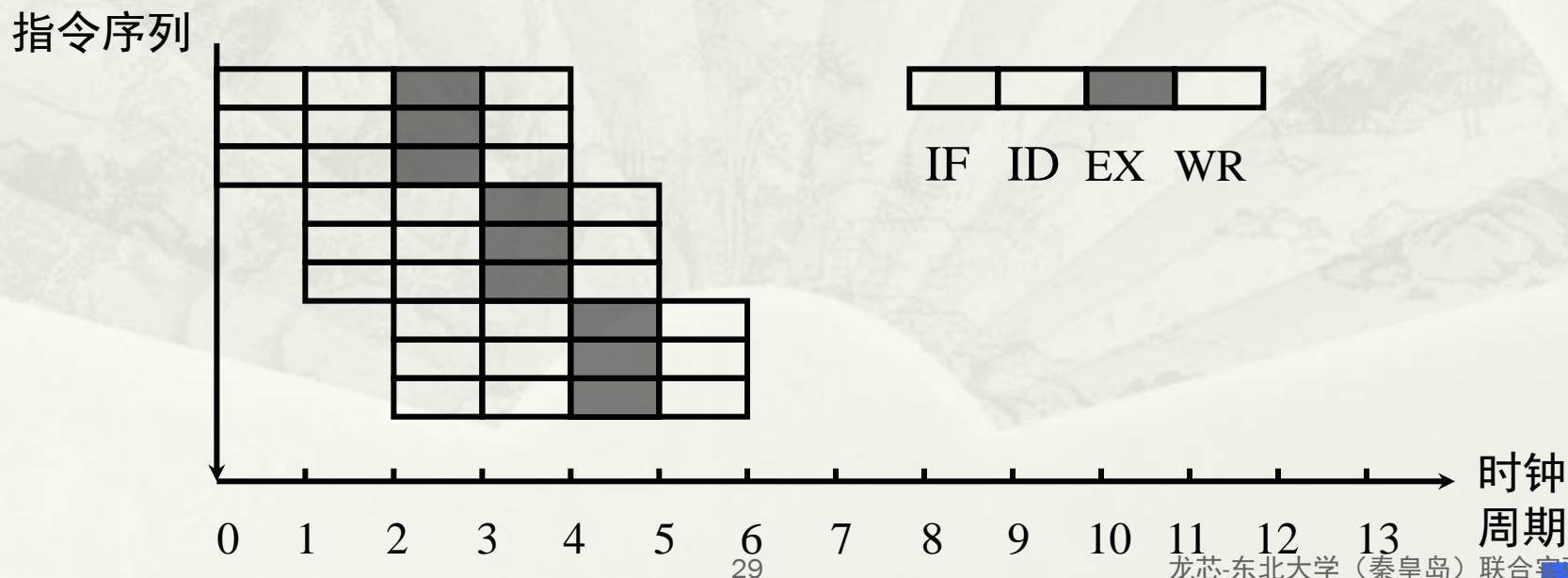
8.3

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

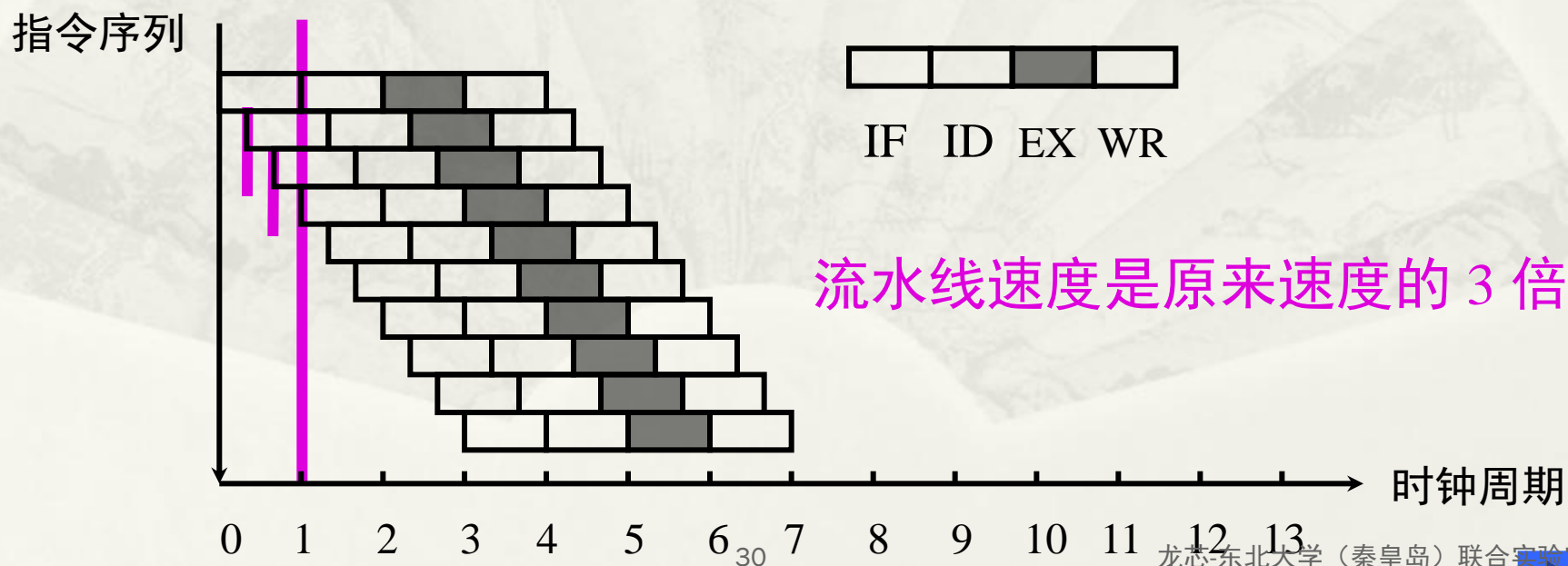
8.3

- 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

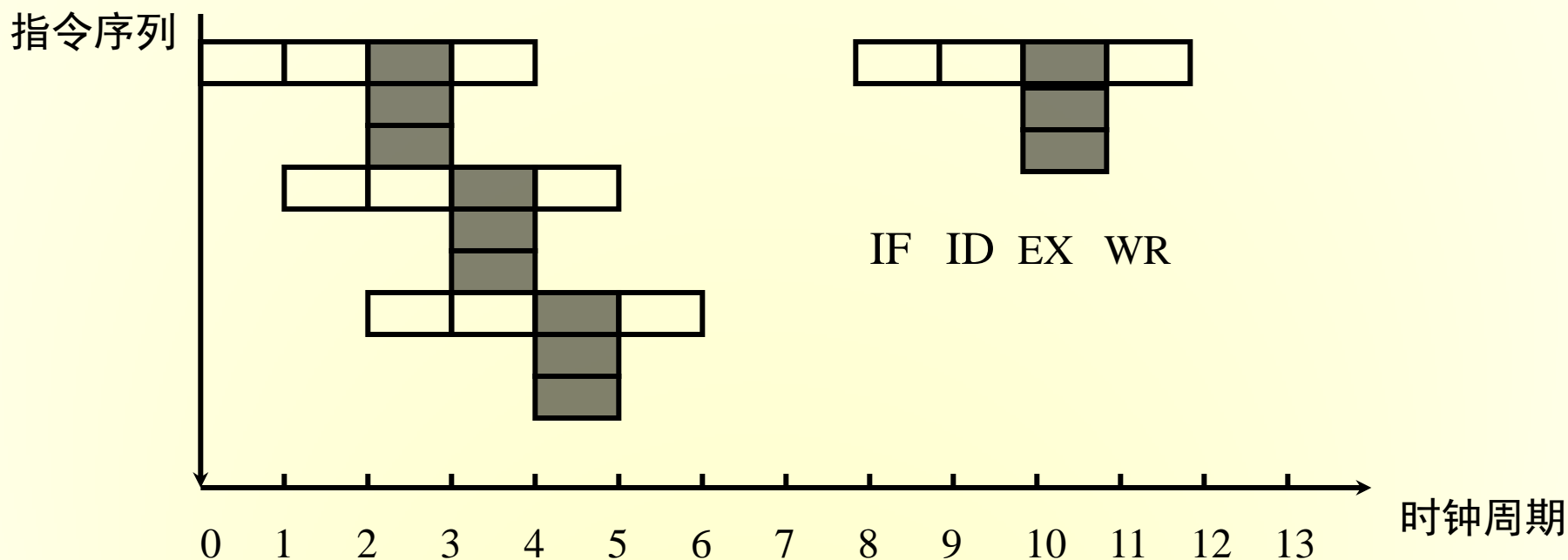
靠编译程序解决优化问题



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**

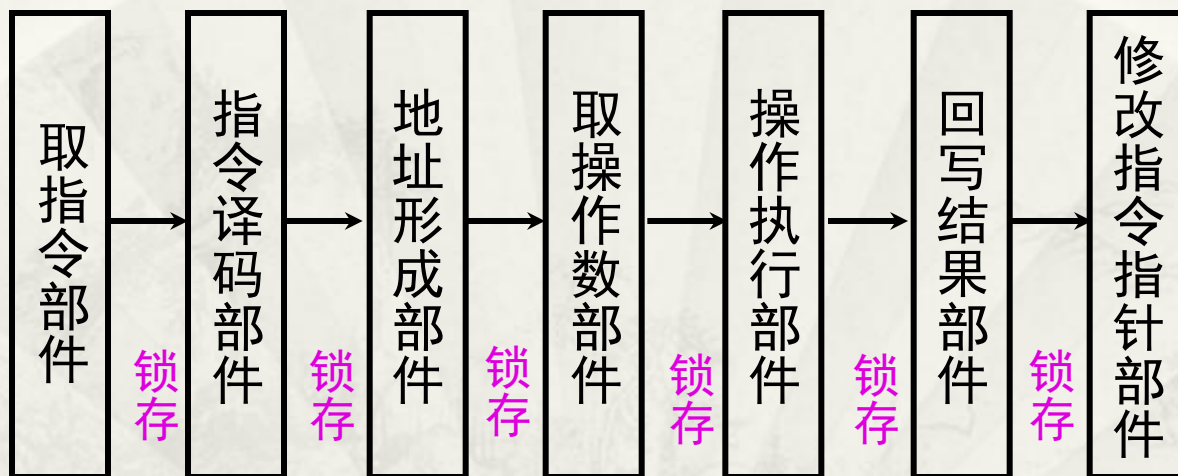


六、流水线结构

8.3

1. 指令流水线结构

完成一条指令分 7 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

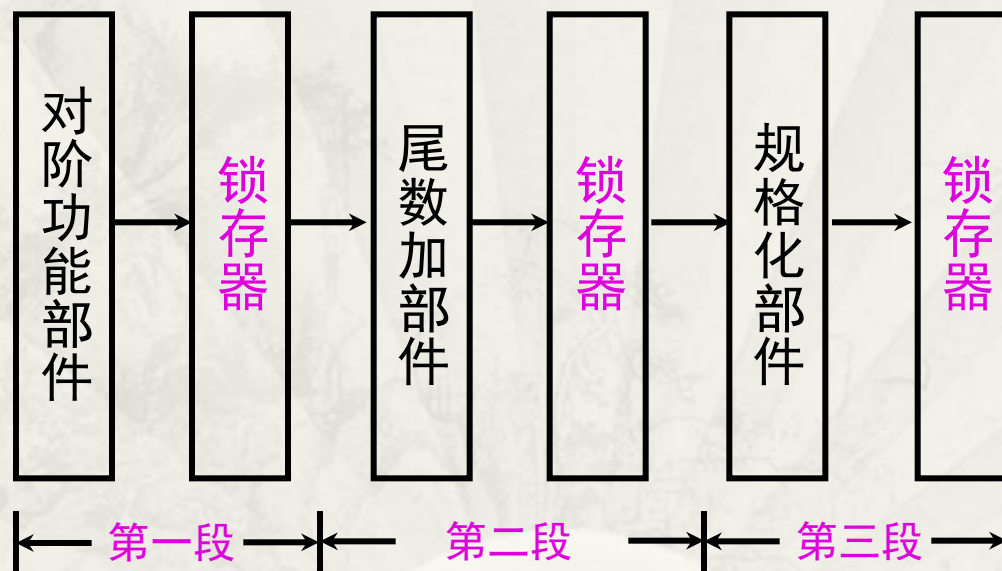
7 个时钟周期出 1 结果

理想情况下，7 级流水的速度是不采用流水技术的 7 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

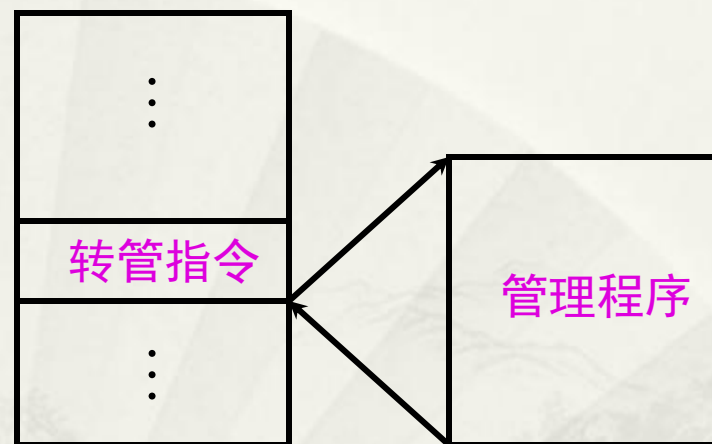
8.4 中断系统

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



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			n
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

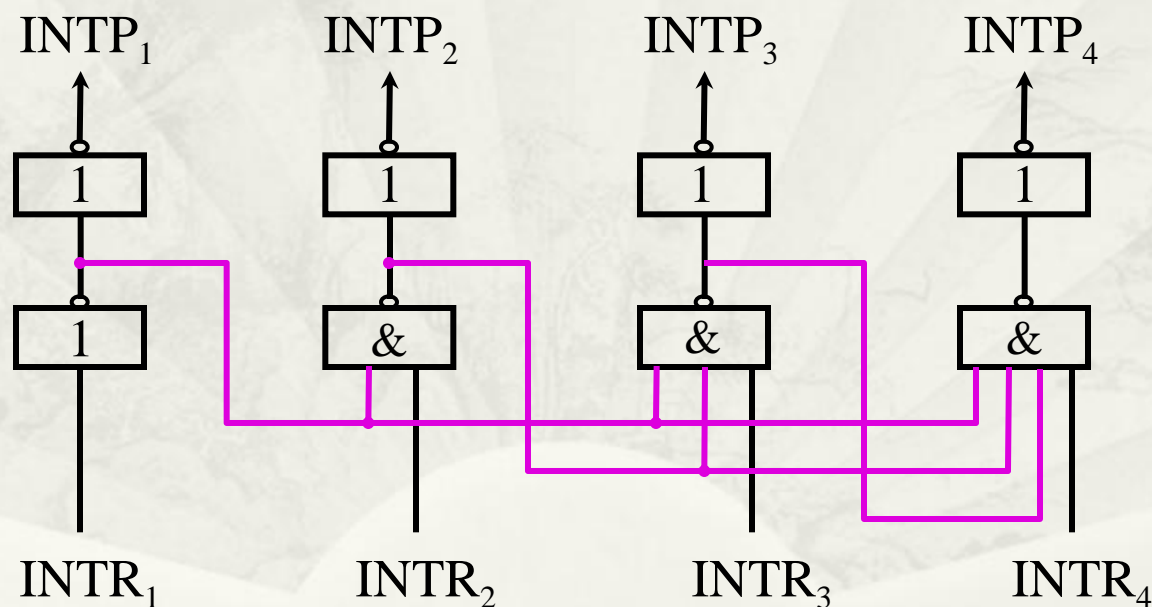
8.4

(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

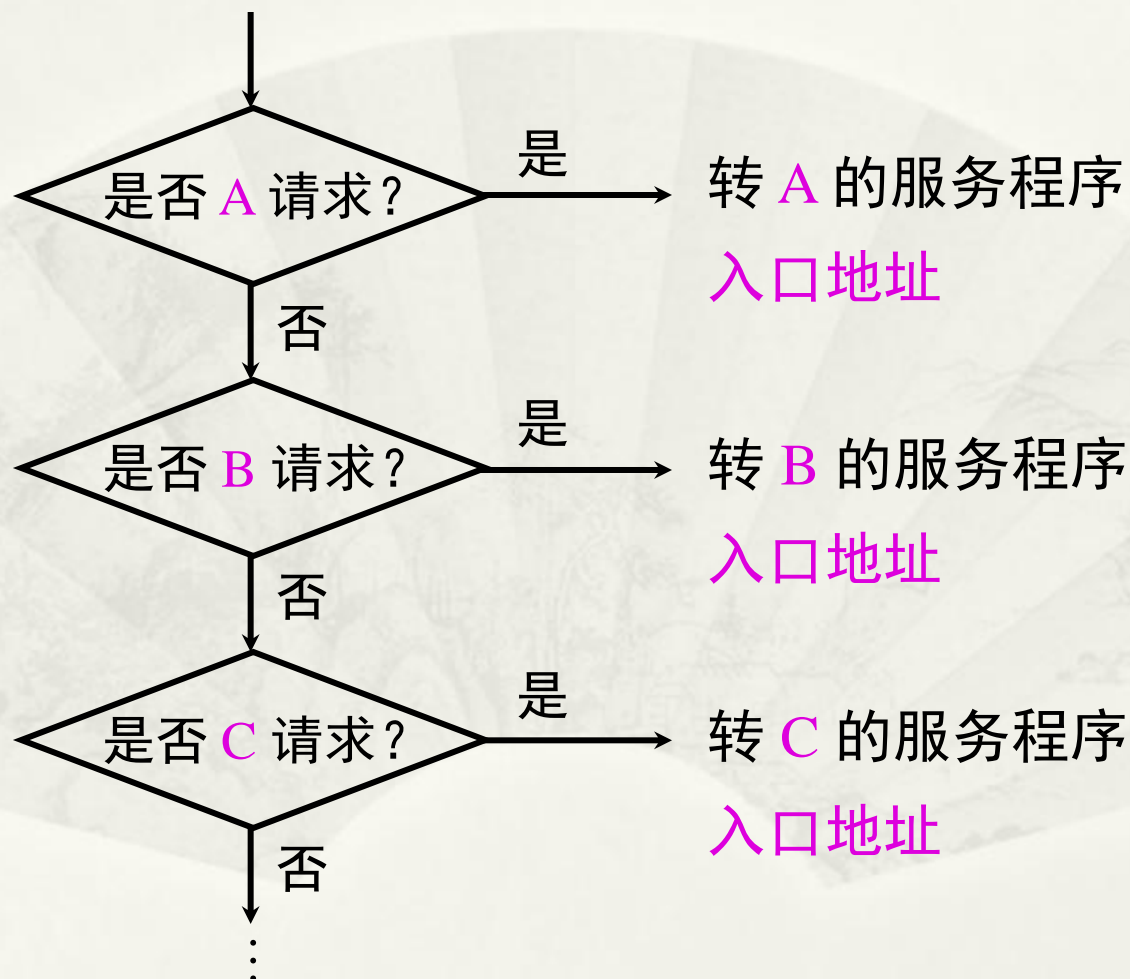
② 集中 在 CPU 内



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

(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

8.4

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
M	SKP DZ 1 [#]	1 [#] D = 0 跳 (D为完成触发器)
	JMP 1 [#] SR	1 [#] D = 1 转1 [#] 服务程序
	SKP DZ 2 [#]	2 [#] D = 0 跳
	JMP 2 [#] SR	2 [#] D = 1 转2 [#] 服务程序
	⋮	
	SKP DZ 8 [#]	8 [#] D = 0 跳
	JMP 8 [#] SR	8 [#] D = 1 转8 [#] 服务程序

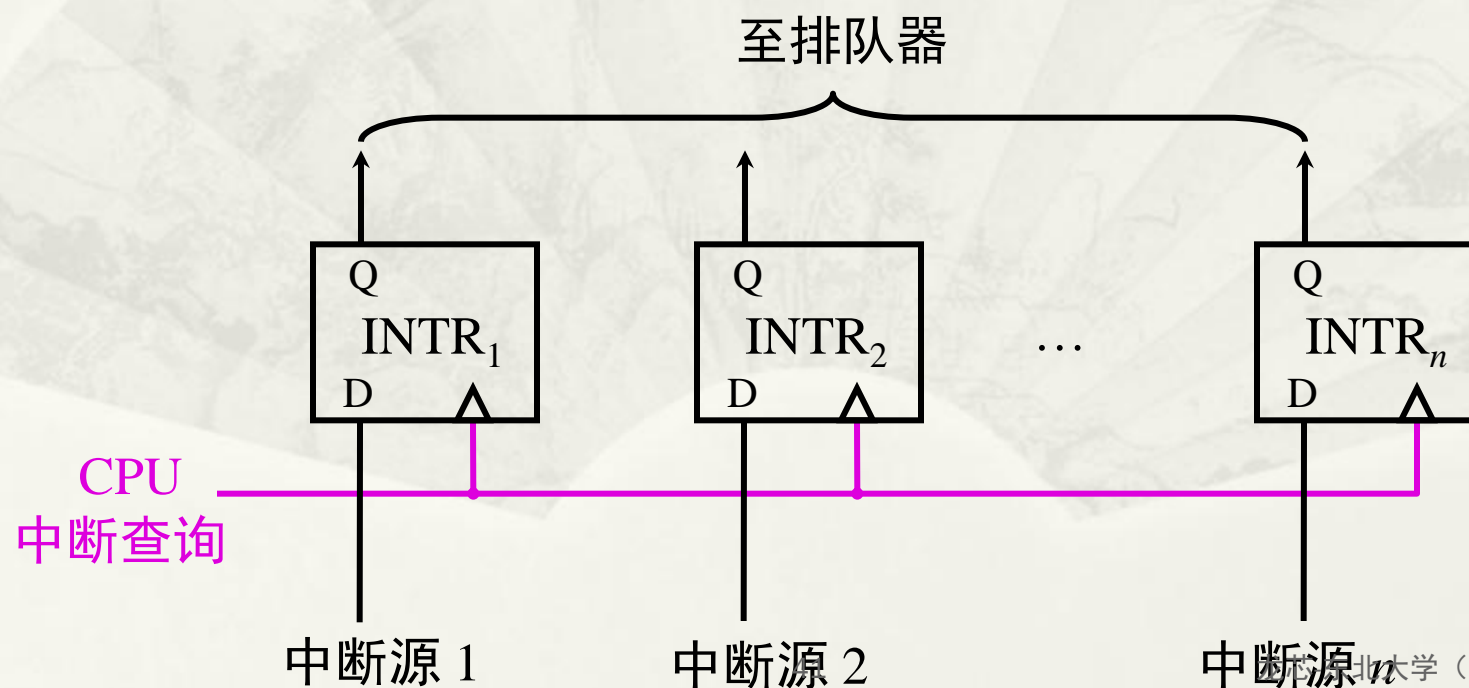
四、中断响应

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）

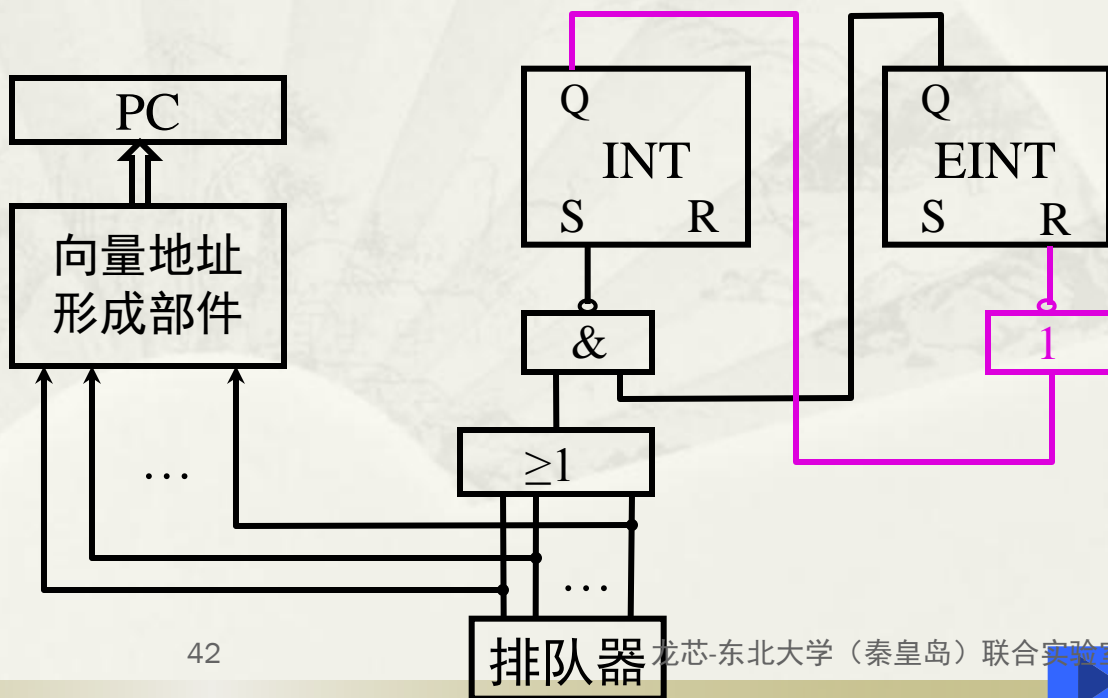
中断识别程序 入口地址 $M \rightarrow$ PC （软件查询法）

(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

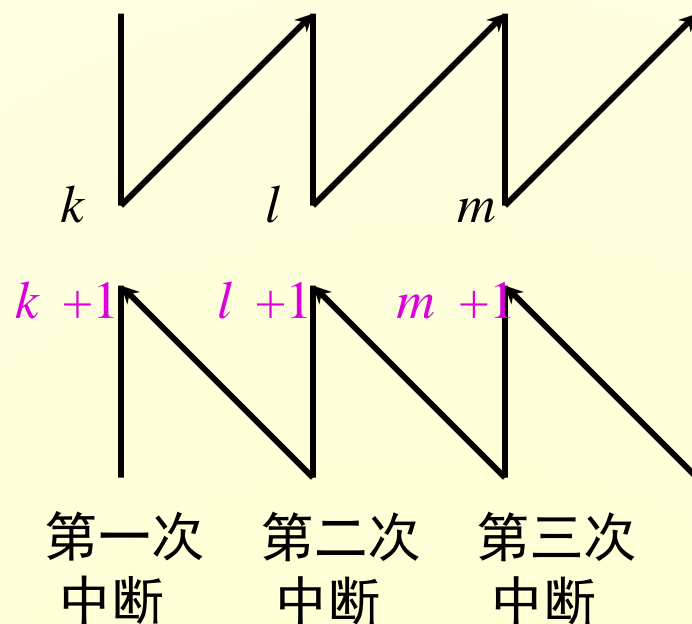
8.4

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



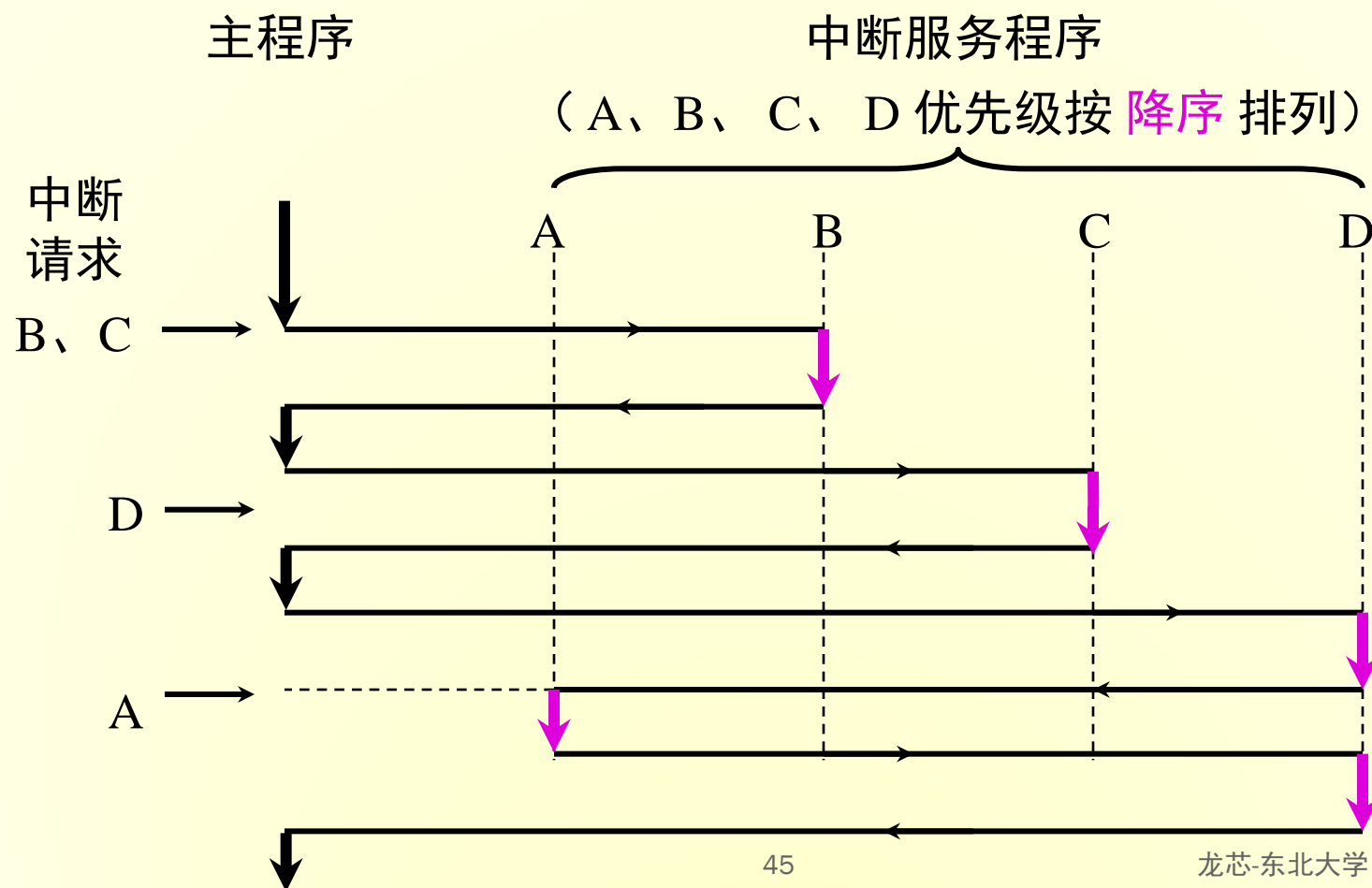
程序断点 $k+1$, $l+1$, $m+1$

2. 实现多重中断的条件

8.4

(1) 提前 设置 开中断 指令

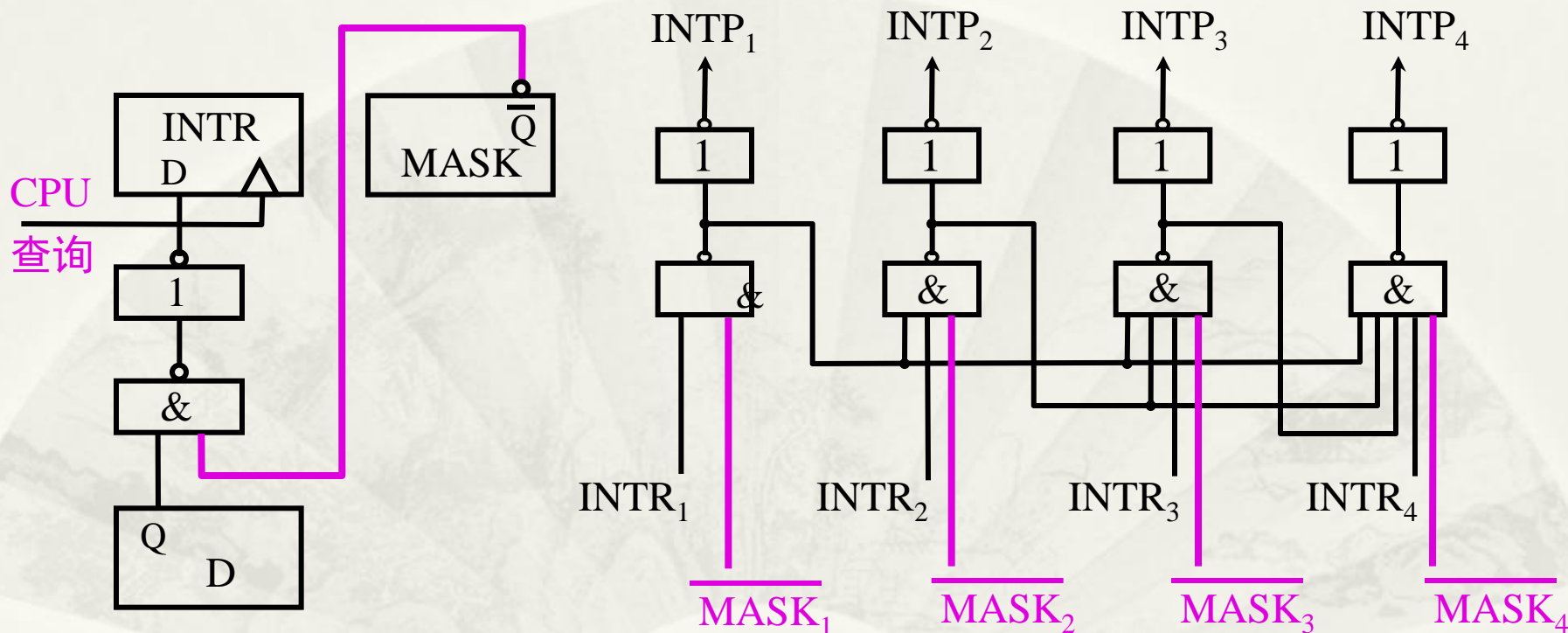
(2) 优先级别高 的中断源 有权中断优先级别低 的中断源



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”

$MASK_i = 1$ (屏蔽)

$INTP_i = 0$ (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）

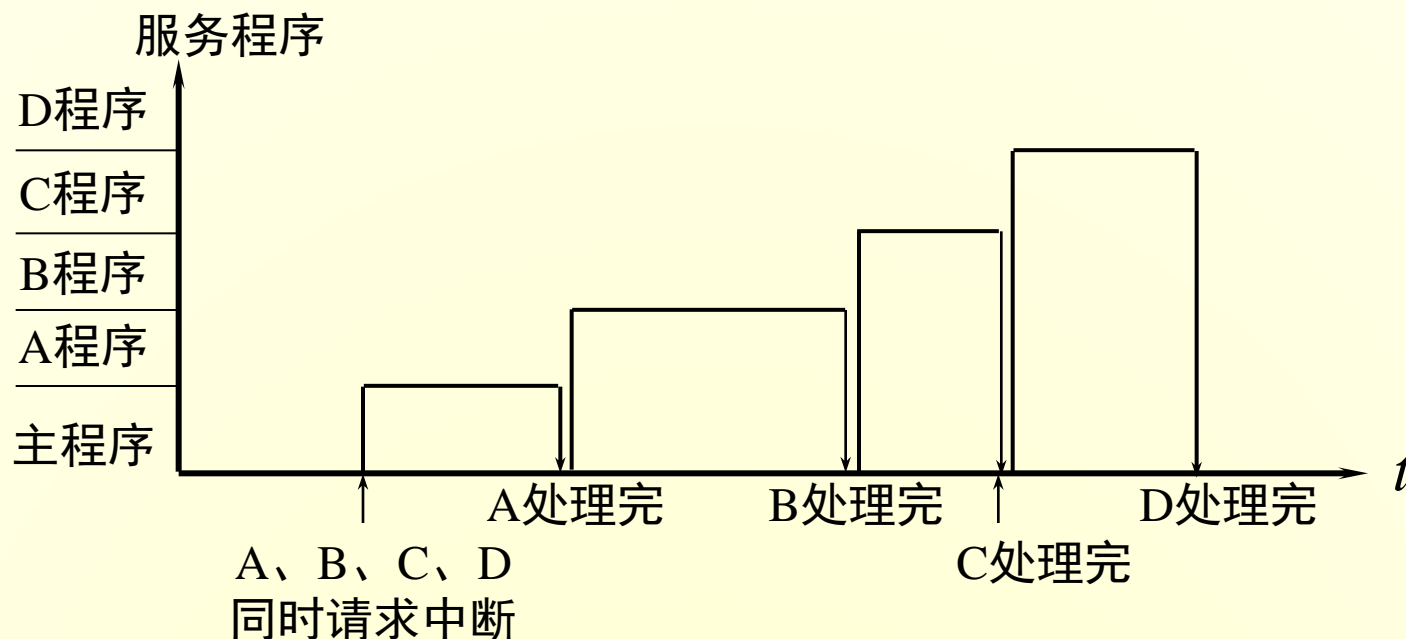
中断源	原屏蔽字	新屏蔽字
A	1 1 1 1	1 1 1 1
B	0 1 1 1	0 1 0 0
C	0 0 1 1	0 1 1 0
D	0 0 0 1	0 1 1 1

响应优先级 $A \rightarrow B \rightarrow C \rightarrow D$ 降序排列

处理优先级 $A \rightarrow D \rightarrow C \rightarrow B$ 降序排列

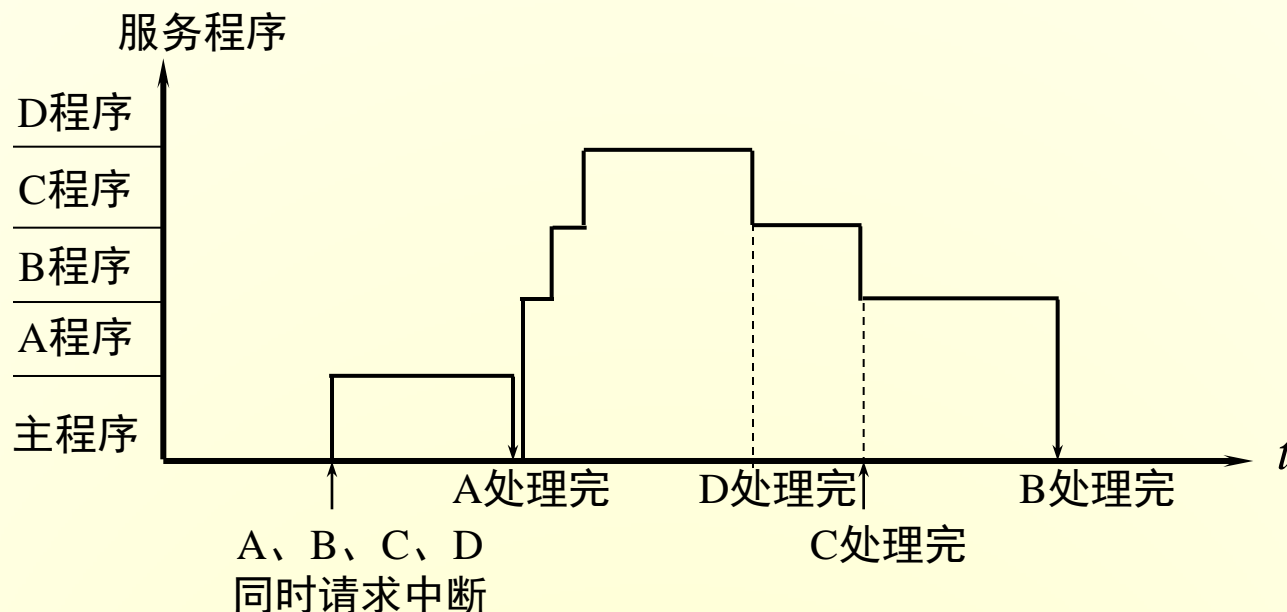
(3) 屏蔽技术可改变处理优先等级

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



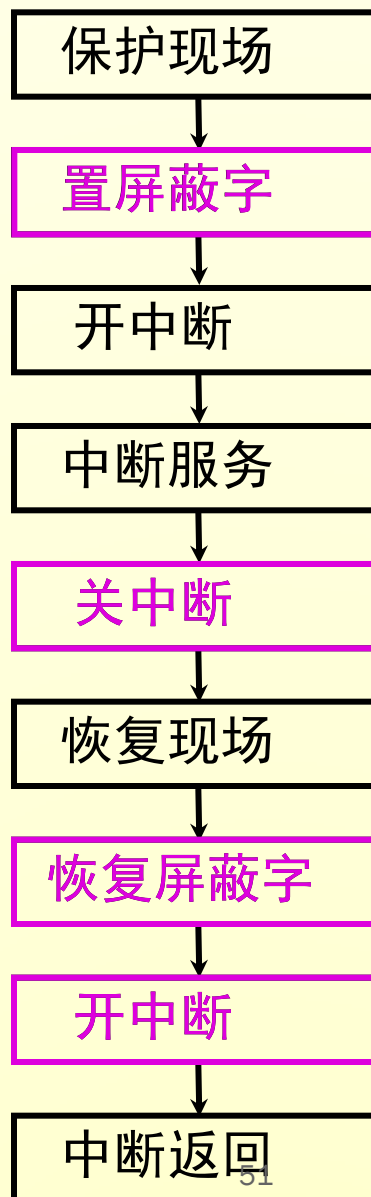
CPU 执行程序轨迹（新屏蔽字）

(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

8.4

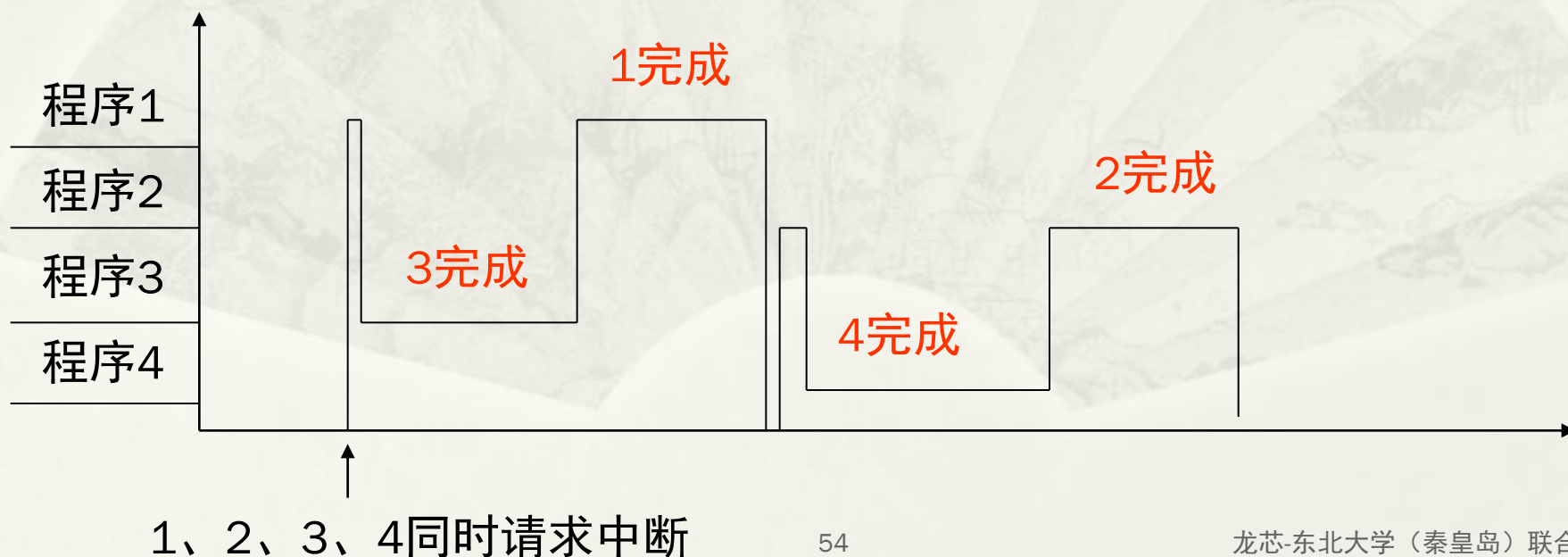


例8.2 设某机有4个中断源1, 2, 3, 4, 其硬件排队优先次序按1->2->3->4降序排列, 各中断源的服务程序中所对应的屏蔽字如下表:

中断源	屏蔽字			
	1	2	3	4
1	1	1	0	1
2	0	1	0	0
3	1	1	1	1
4	0	1	0	1

- * 给出上述4个中断源的中断处理次序
- * 若4个中断源同时有中断请求，画出CPU执行程序的轨迹。

- * 解（1）处理次序按照3->1->4->2降序排列。
- * (2)当4个中断源同时有请求时，由于硬件排队
的优先次序是1->2->3->4，故CPU先响应
1，执行1.屏蔽字1101，开中断后执行3的
程序，结束后回到1，1结束后，响应2，执
行2，0100屏蔽字，执行4.回到2.



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 MAR

命令存储器写

PC \longrightarrow MDR 断点 \longrightarrow MDR

(MDR) \rightarrow 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