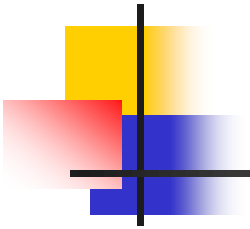
计算机组成原理
(第八讲)

厦门大学信息学院软件工程系　曾文华

2021年5月26日

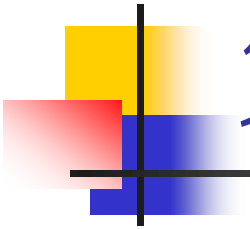


第3篇　中央处理器

第 6 章　计算机的运算方法

第 7 章　指 令 系 统

第 8 章　CPU 的结构和功能



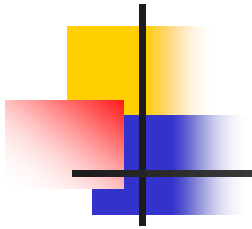
第 8 章　CPU 的结构和功能

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

二、CPU 结构框图

三、 CPU 的寄存器

四、 控制单元 CU 和中断系统

五、ALU

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，以及移位运算

二、CPU 结构框图

1. CPU 与系统总线

程序计数器　指令寄存器

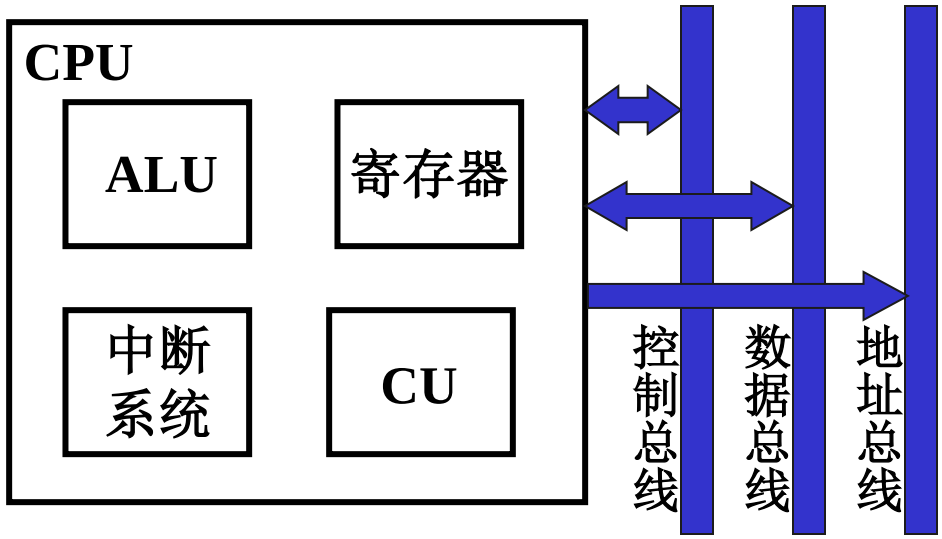
指令控制　PC　IR

控制单元

操作控制
时间控制　} CU　时序电路

数据加工　ALU　寄存器

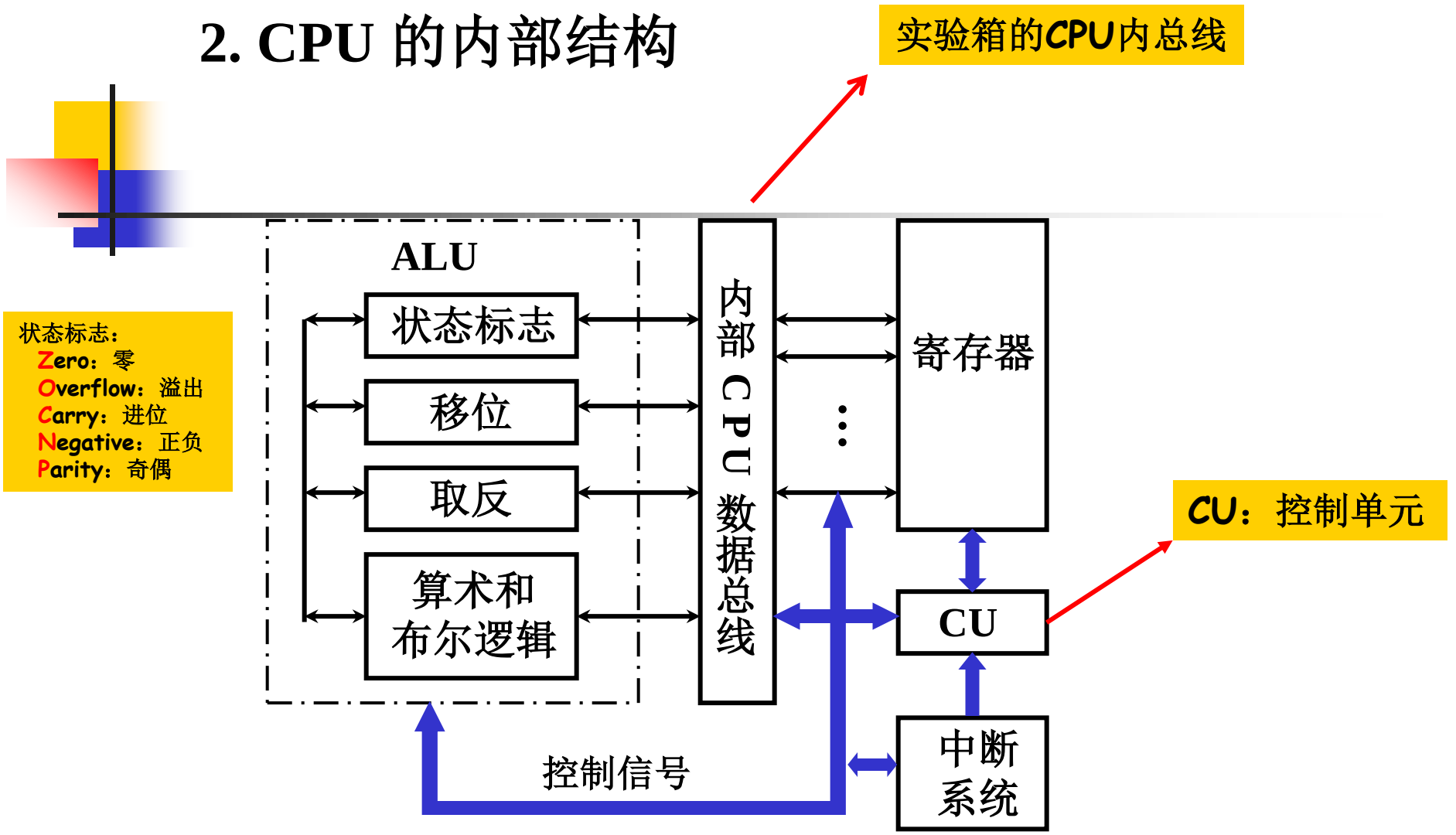
处理中断　中断系统



系统总线：

- ■数据总线
- ■地址总线
- ■控制总线

2. CPU 的内部结构


实验箱的CPU内总线
ALU
状态标志
移位
取反
算术和
布尔逻辑
内部CPU数据总线
寄存器
CU
中断
系统
控制信号
CU：控制单元
状态标志：
Zero：零
Overflow：溢出
Carry：进位
Negative：正负
Parity：奇偶

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

DS　　SP

(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

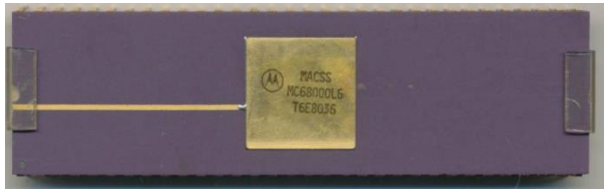
状态寄存器　　存放条件码

PSW 寄存器　　存放程序状态字(Program State Word)

3. 举例　　Z8000　　8086　　MC 68000

Z80000　　80386

- 图8.3：Z8000、8086、MC68000三种微处理器的寄存器组织
  - Z8000：Zilog公司于1979年推出的16位CPU。
  - 8086：Intel公司于1978年推出的16位CPU。
  - MC68000：Motorola公司1979年推出的16位CPU。



- 图8.4：Z80000、80386两种32位微处理器寄存器组织
  - Z80000：Zilog公司于1986年推出的32位CPU。
  - 80386：Intel公司于1985年推出的32位CPU。

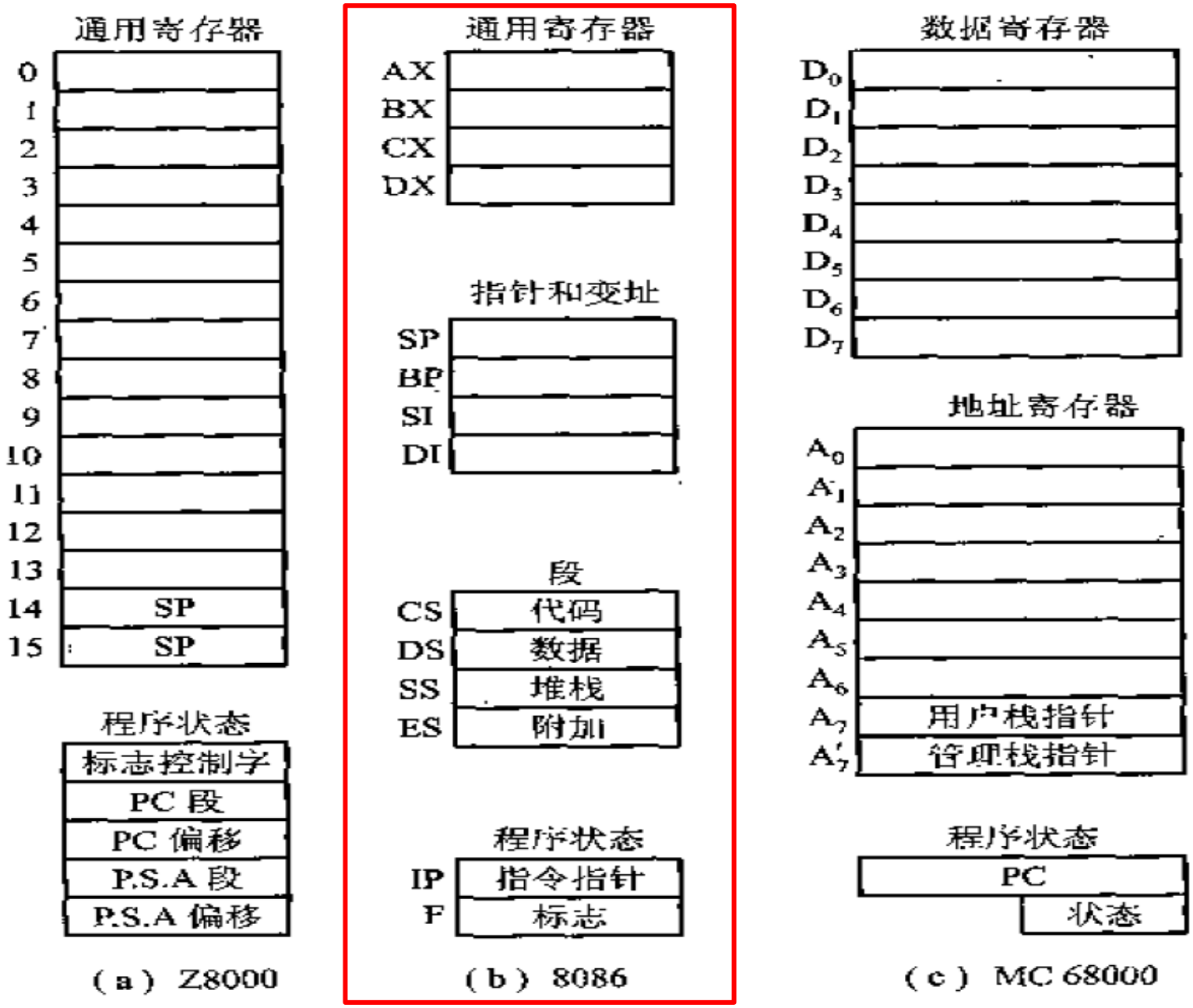


图 8.3　三种微处理器的寄存器组织

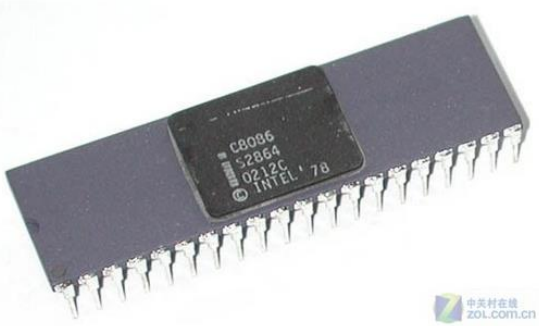
8086 CPU 的寄存器

- 通用寄存器：**AX、BX、CX、DX**

- **SP**：堆栈指针寄存器
- **BP**：基址指针寄存器
- **SI**：源变址寄存器
- **DI**：目的变址寄存器

- **CS**：代码段
- **DS**：数据段
- **SS**：堆栈段
- **ES**：附加段

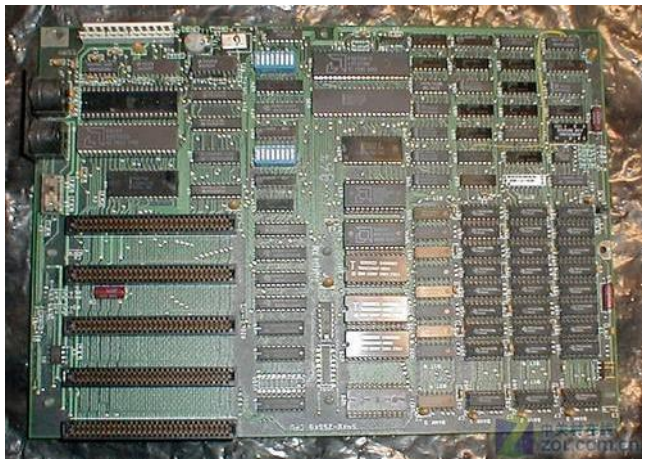
- **IP**：指令指针
- **F**：标志寄存器



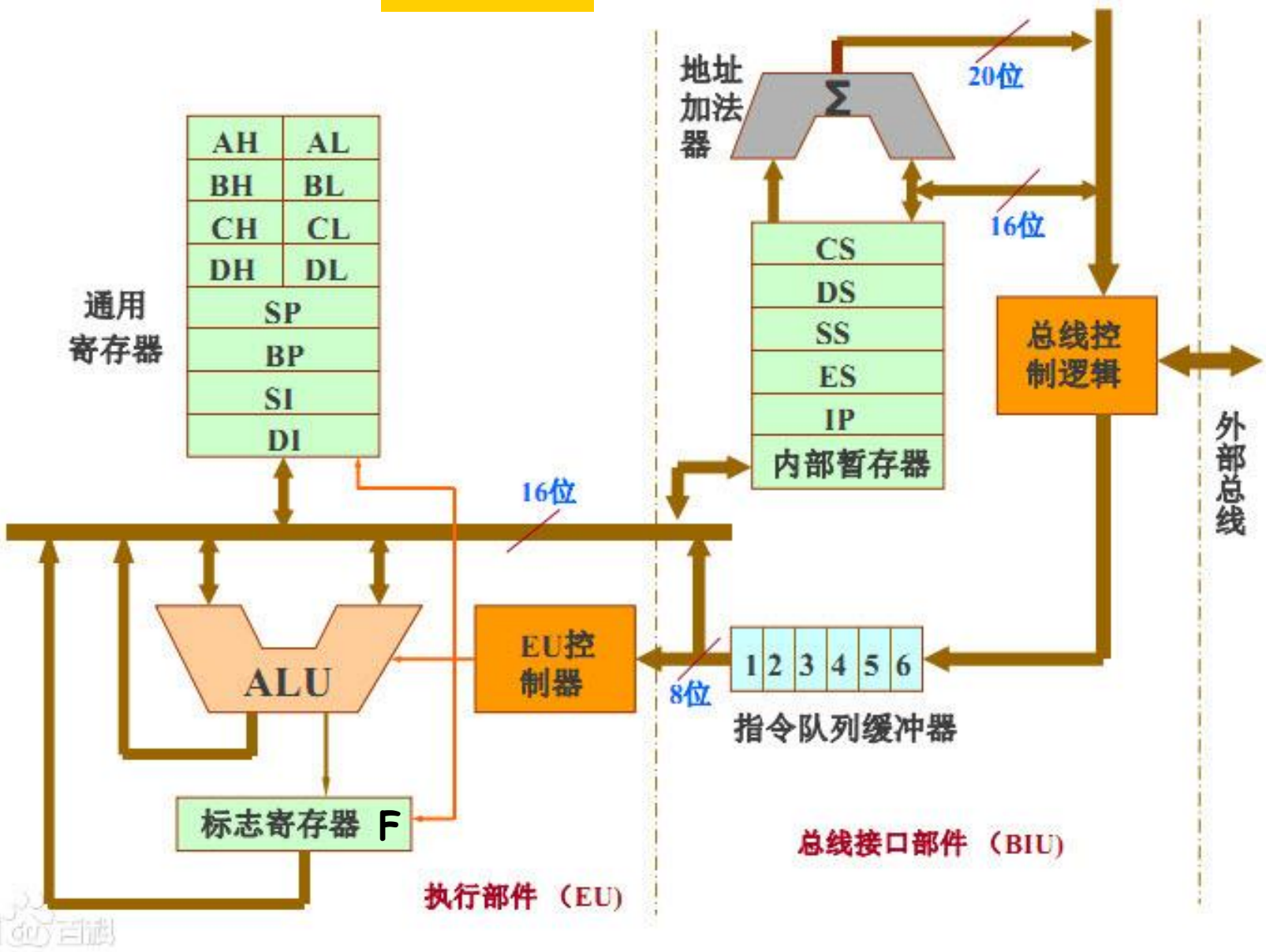
8086 CPU

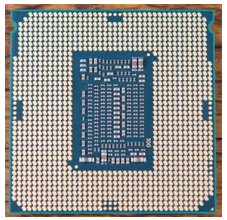
图 4.2.1 8088/8086CPU 管脚图

- **Intel 8086是一个由Intel于1978年所设计的16位微处理器芯片，是x86架构的鼻祖。不久之后，Intel 就推出了 Intel 8088 (一个拥有8位外部数据总线的微处理器)。它是以8080和8085的设计为基础，拥有类似的寄存器组，但是地址总线扩充为20位。总线接口单元(Bus Interface Unit)透过6字节预存(prefetch) 的队列(queue)位指令给执行单元(Execution Unit)，所以取指令和执行是同步的，8086 CPU有20条地址线，可直接寻址1MB的存储空间，每一个存储单元可以存放一个字节（8位）二进制信息。**



8086 CPU


地址
加法
器
Σ
20位
16位
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
通用
寄存器
CS
DS
SS
ES
IP
内部暂存器
总线控
制逻辑
外
部
总
线
16位
ALU
EU控
制器
1
2
3
4
5
6
8位
指令队列缓冲器
标志寄存器 F
总线接口部件 （BIU）
执行部件 （EU）



纪念8086的40周年！Intel 推出酷睿 i7-8086K 处理器

- **Intel**作为**PC**行业毫无疑问的龙头企业，在如今的全球科技行业拥有着极强的话语权。而这家“巨无霸”的崛起还要追溯到**1978**年一颗名为“**8086**”的处理器。正是这颗只有**4**万个晶体管，时钟频率仅**4.77MHz~10MHz**的**16**位处理器开启了**X86**架构处理器的辉煌历史。在整整**40**年后的**2018**年，**Intel**推出了一颗**6**核心**12**线程，出厂最高睿频高达**5GHz**的限量版处理器来纪念当年的历史时刻。这就是我们今天要和大家分享的这颗具有特殊意义的酷睿**i7-8086K**处理器。

泡泡网 PCPOP.COM	Core i7-8086K
架构	Coffee Lake
制程工艺	14nm++
物理核心	6
线程数	12
默认频率	4.0GHz
最高睿频	5.0GHz
三级缓存	12MB
TDP	95W

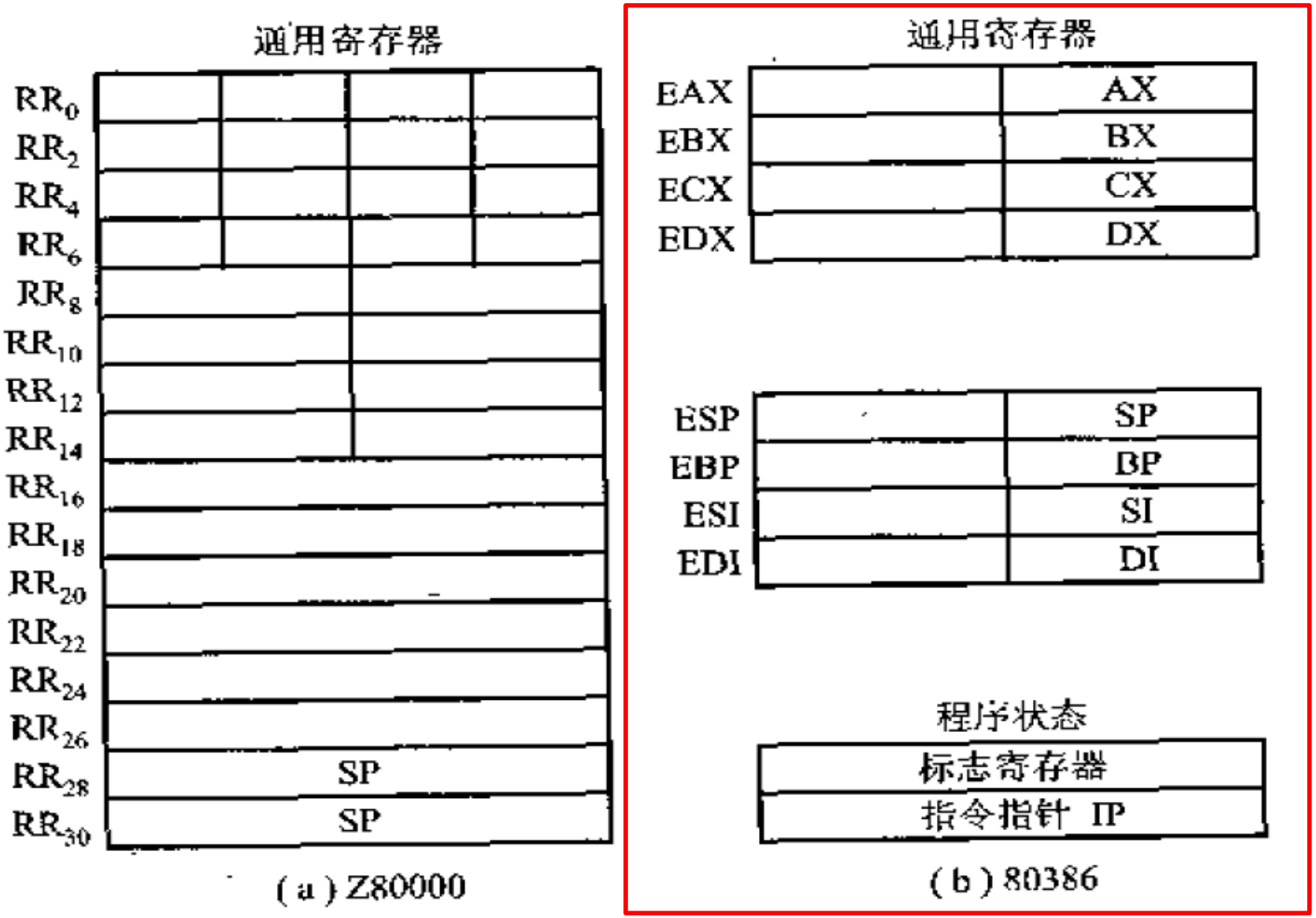


图 8.4　两种 32 位微处理器寄存器组织

80386 CPU的寄存器

8个	通用寄存器(EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI)
6个	段寄存器(CS、SS、DS、ES、FS、GS)
2个	指令指针寄存器和标志寄存器(EIP、EFLAGS)
4个	系统表寄存器(GDTR、IDTR、LDTR、TR)
5个	控制寄存器(CR0、CR1、CR2、CR3、CR4)
8个	调试寄存器(DR0、DR1、DR2、DR3、DR4、DR5、DR6、DR7)
2个	测试寄存器(TR6、TR7)

共35个32位寄存器

	31 16	15 87 0		31 16	15 0
EAX		(AH)AX(AL)	ESP		SP
EBX		(BH)BX(BL)	EBP		BP
ECX		(CH)CX(CL)	ESI		SI
EDX		(DH)DX(DL)	EDI		DI

15 0	31 0 19 0 11 0
CS 选择器	CS 描述符高速缓存寄存器
SS 选择器	SS 描述符高速缓存寄存器
DS 选择器	DS 描述符高速缓存寄存器
ES 选择器	ES 描述符高速缓存寄存器
FS 选择器	FS 描述符高速缓存寄存器
GS 选择器	GS 描述符高速缓存寄存器

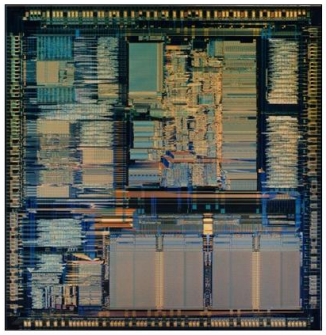
	31 16	15 0		31 16	15 0
EIP		IP	EFLAGS		FLAGS

	31 0	15 0
GDTR		
IDTR		

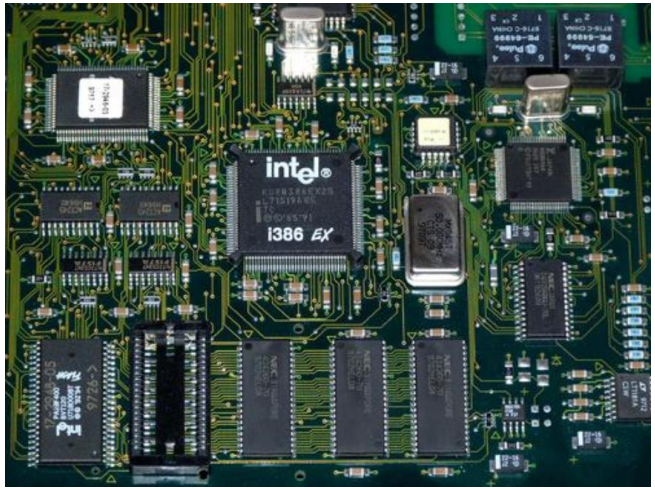
	15 0	31 0 19 0 11 0
LDTR	LDTR 选择器	LDTR 高速缓存
TR	TR 选择器	TR 高速缓存

80386 CPU的寄存器

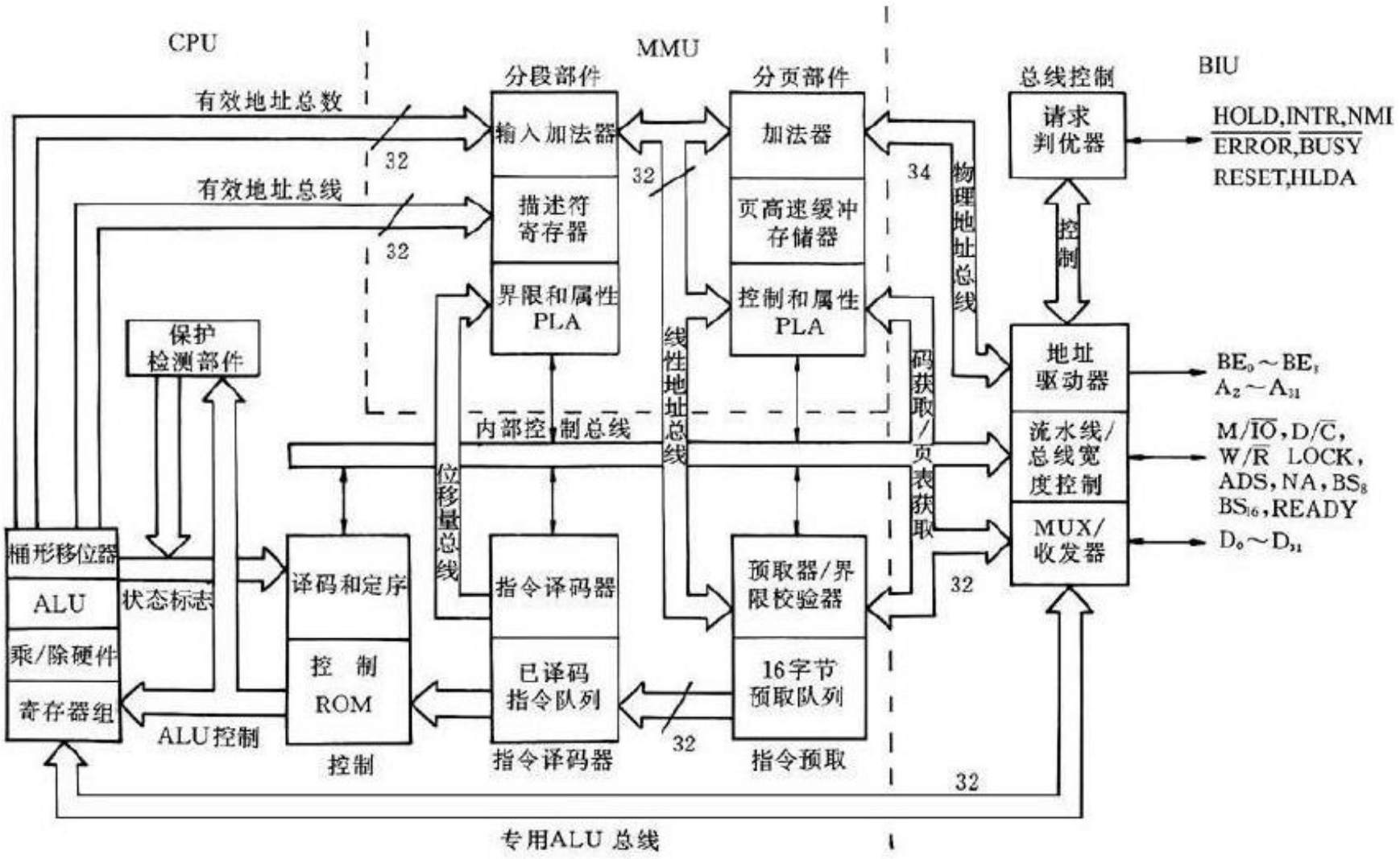
80386 CPU



- **INTEL 1985年推出的CPU芯片，它是80x86系列中的第一种32位微处理器，而且制造工艺也有了很大的进步，与80286相比，80386内部内含27.5万个晶体管，时钟频率为12.5MHz，后提高到20MHz，25MHz，33MHz。80386的内部和外部数据总线都是32位，地址总线也是32位，可寻址高达4GB内存。它除具有实模式和保护模式外，还增加了一种叫虚拟86的工作方式，可以通过同时模拟多个80x86处理器来提供多任务能力。除了标准的80386芯片，也就是80386DX外，出于不同的市场和应用考虑，INTEL又陆续推出了一些其它类型的80386芯片：80386SX、80386SL、80386DL等。**



80386 CPU


CPU
MMU
BIU
分段部件
分页部件
总线控制
有效地址总数
有效地址总线
32
输入加法器
描述符寄存器
界限和属性PLA
加法器
页高速缓冲存储器
控制和属性PLA
34
物理地址总线
请求判优器
HOLD,INTR,NMI
ERROR,BUSY
RESET,HLDA
控制
保护检测部件
内部控制总线
线性地址总线
码获取/页表获取
地址驱动器
BE0～BE3
A2～A31
流水线/总线宽度控制
M/IO,D/C,
W/R LOCK,
ADS,NA,BS8
BS16,READY
MUX/收发器
D0～D31
桶形移位器
ALU
乘/除硬件
寄存器组
状态标志
位移量总线
译码和定序
控制ROM
指令译码器
已译码指令队列
预取器/界限校验器
16字节预取队列
ALU控制
控制
指令译码器
指令预取
专用ALU总线

四、 控制单元 CU 和中断系统

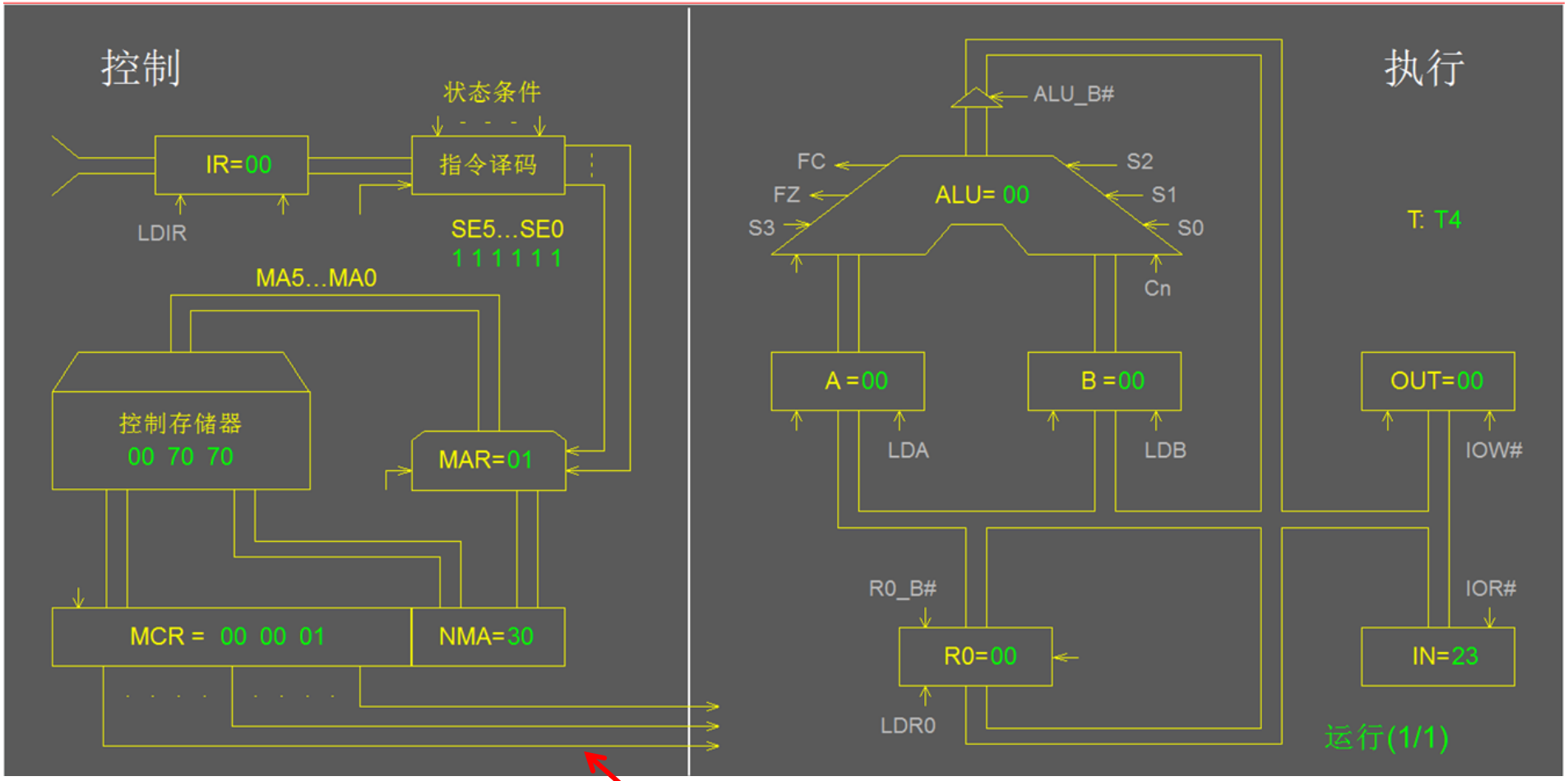
1. CU　产生全部指令的微操作命令序列

- 组合逻辑设计　　硬连线逻辑（硬布线逻辑）
- 微程序设计　　存储逻辑　　参见 第 4 篇

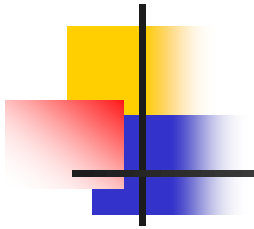
2. 中断系统　　参见 8.4 节　 5.5节

五、ALU　　参见 第 6 章

实验箱的控制器和运算器



微操作命令序列



8.2 指 令 周 期

一、 指令周期的基本概念

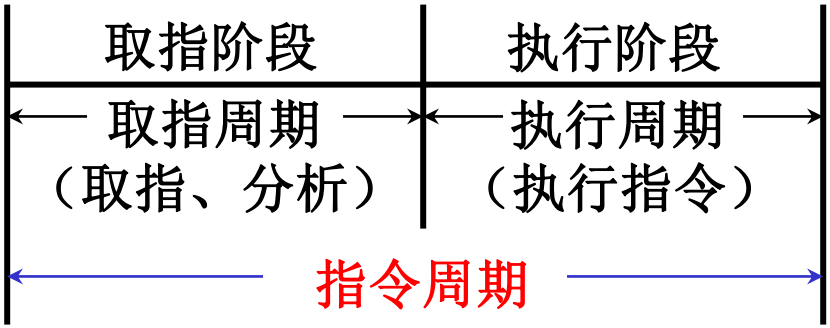
二、 指令周期的数据流

一、 指令周期的基本概念

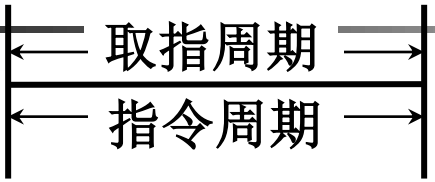
1．指令周期

取出并执行一条指令所需的全部时间

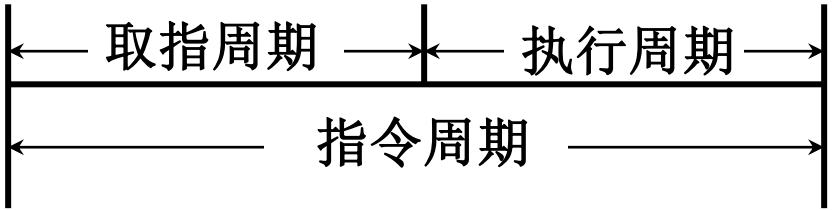
完成一条指令 { 取指、分析 —— 取指周期
执行 —— 执行周期



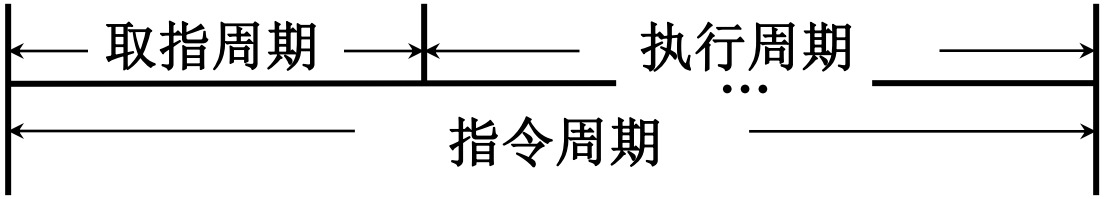
2. 每条指令的指令周期不同



NOP、无条件转移指令



ADD mem



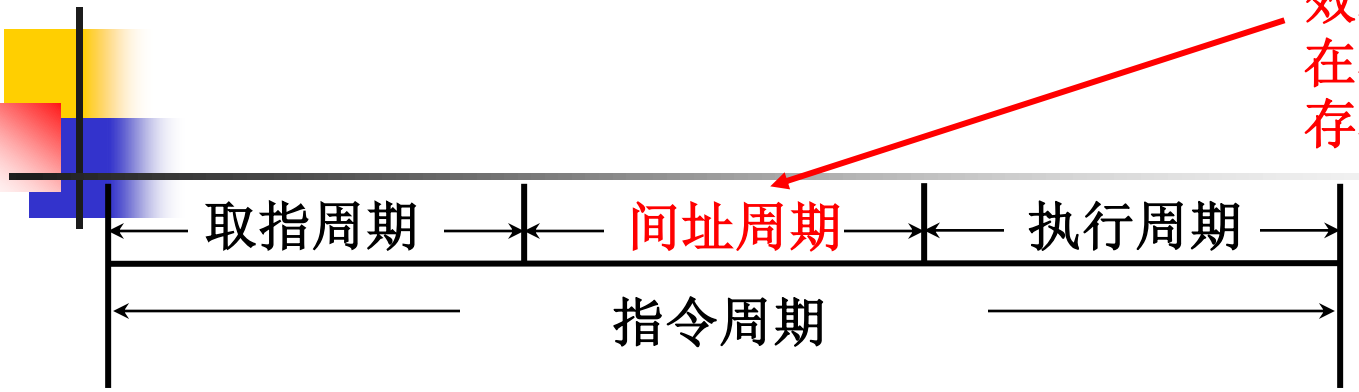
MUL mem

取指周期相同，执行周期不同

（假设指令的长度固定）

3. 具有间接寻址的指令周期

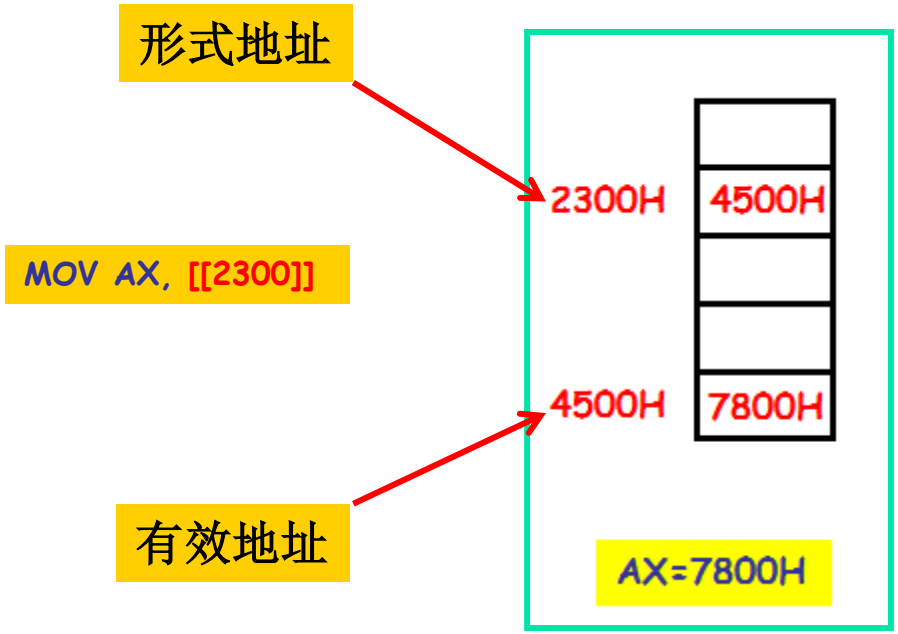
用于取操作数的有效地址（有效地址在存储器中，或寄存器中）



MOV AX, [[2300]]

MOV AX, [[[2300]]]

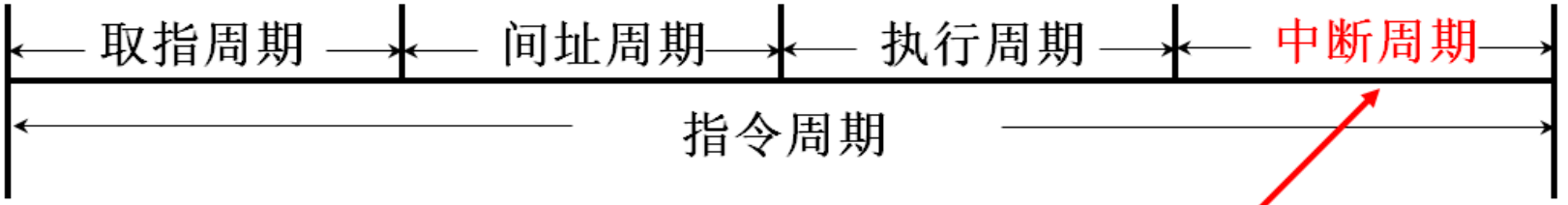
MOV AX, [BX]



间址周期：用于取操作数的有效地址，即得到4500H

执行周期：[4500H]->AX
即7800H -> AX

4. 带有中断周期的指令周期



CPU将程序断点
保存到存储器中

主程序

```
MOV    AL, 20H
INC    AL
```

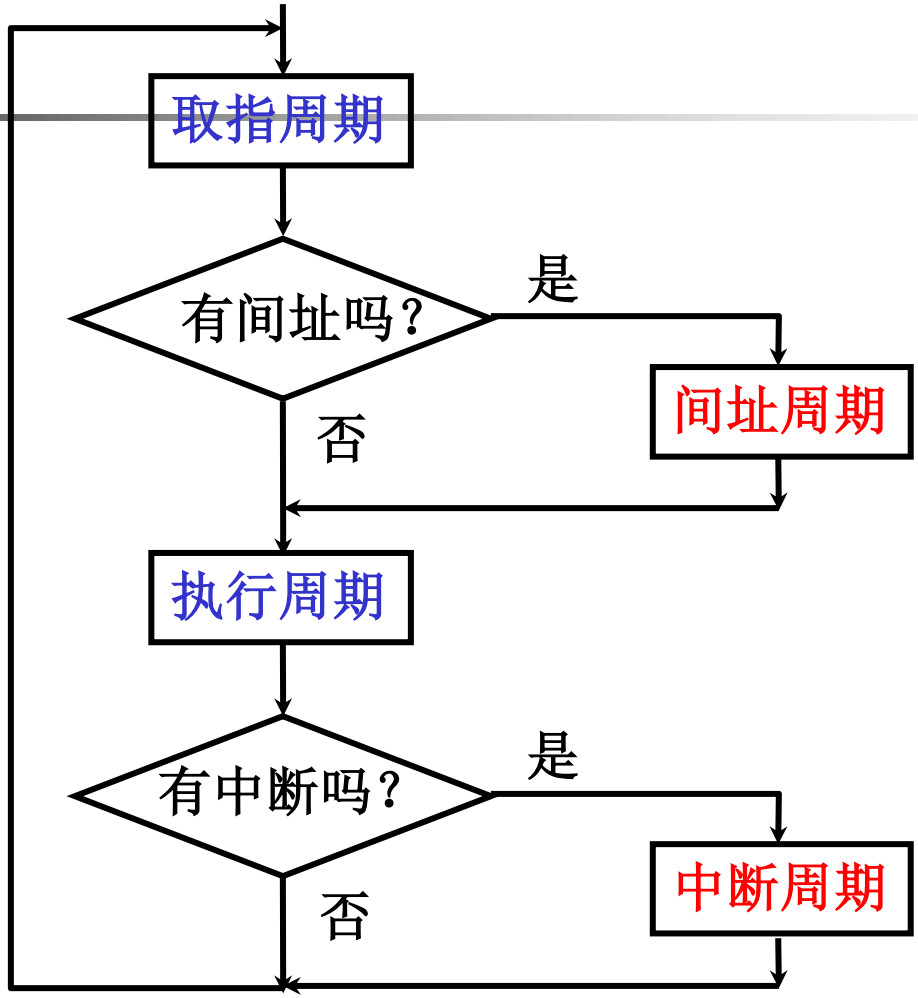
程序
断点

中断服务程序

```
SSS:   MOV CX, DX

       IRET
```

5. 指令周期流程



6. CPU 工作周期的标志

Fetch
Indirect
Execute
Interrupt

CPU 访存有四种性质：

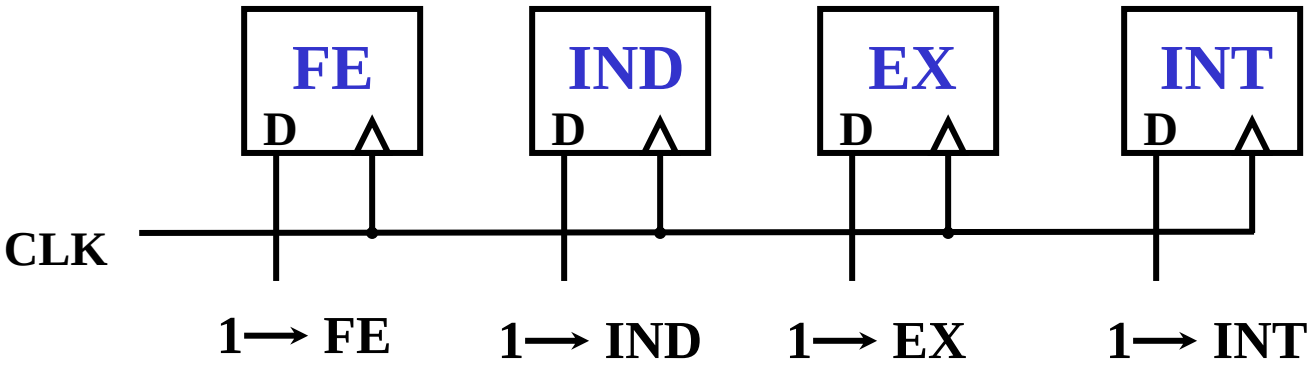
取 指令　　取指周期：FE

取 地址　　间址周期：IND

取 操作数　　执行周期：EX

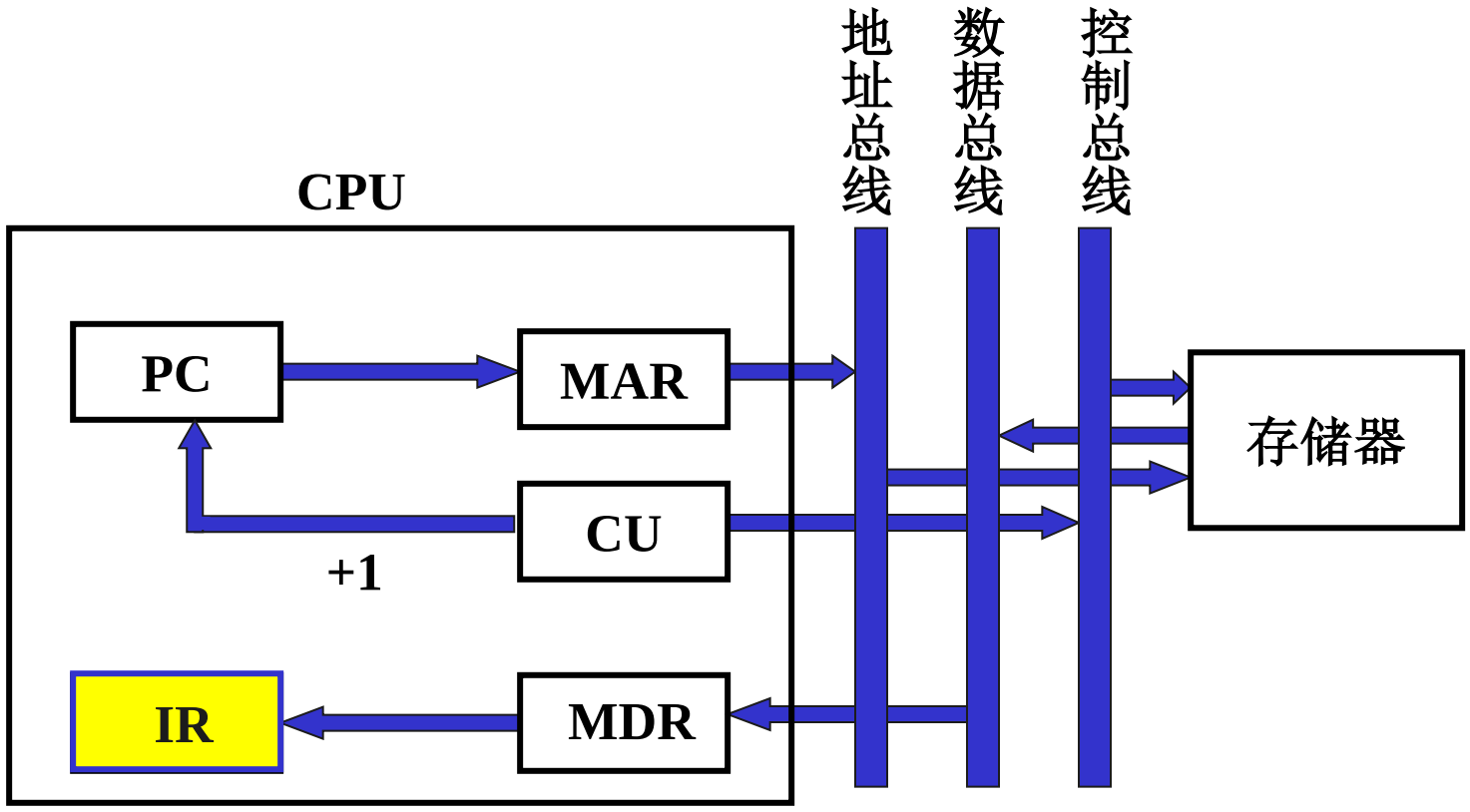
存 程序断点　　中断周期：INT

CPU 的
4个工作周期



二、 指令周期的数据流

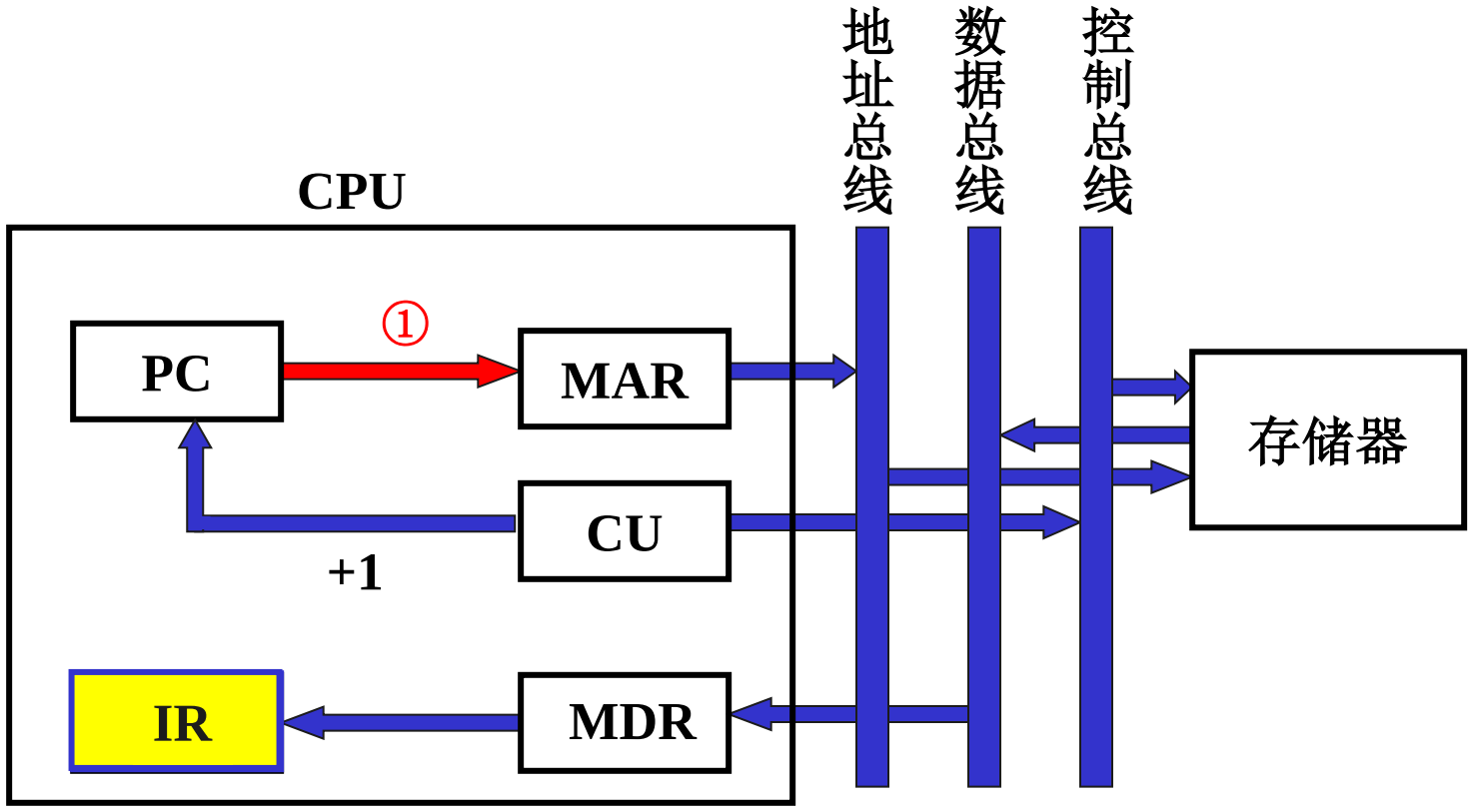
1. 取指周期数据流



取指周期的目的是：将指令的机器码从存储器中取到IR中

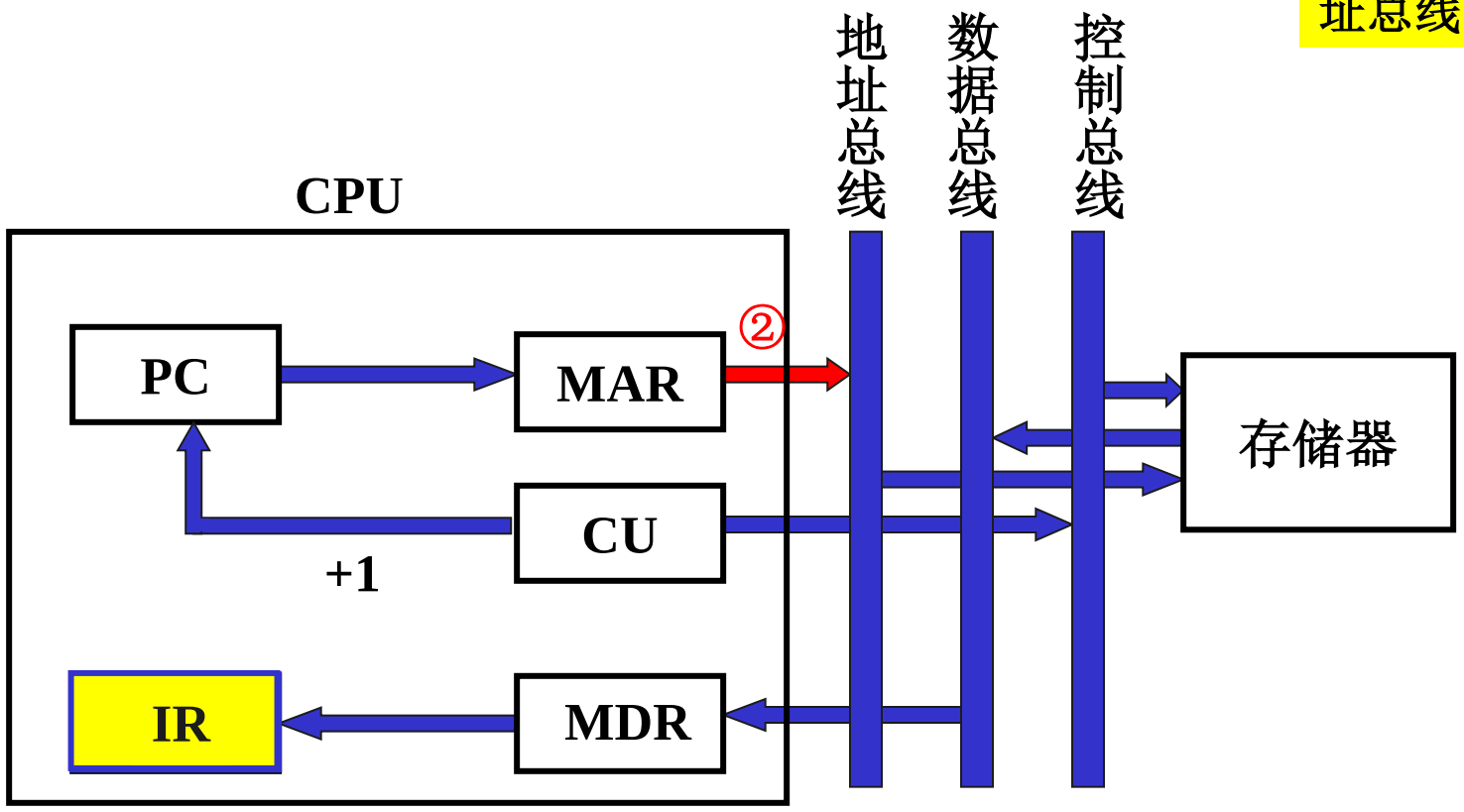
1. 取指周期数据流

①PC送MAR



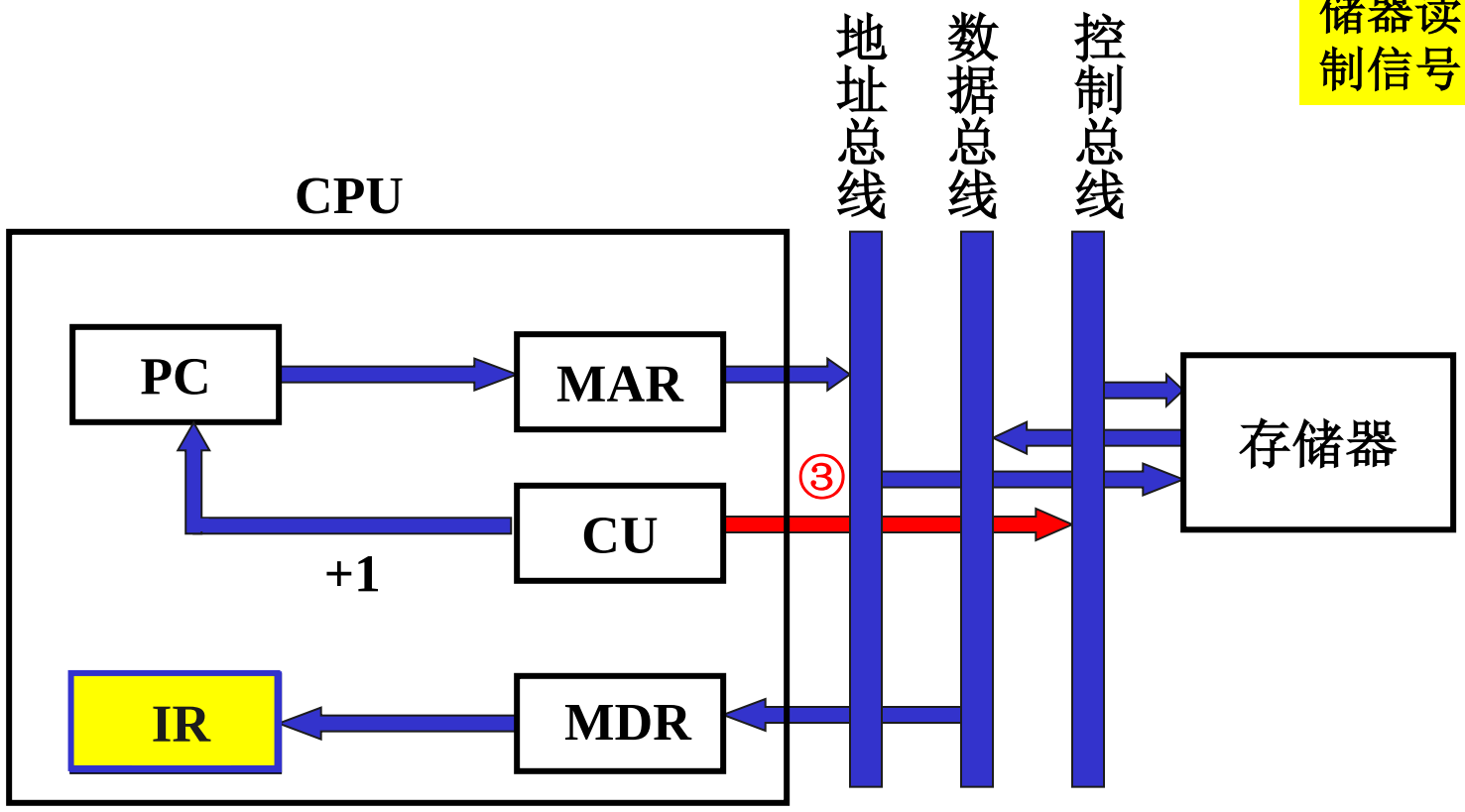
1. 取指周期数据流

②MAR送地址总线



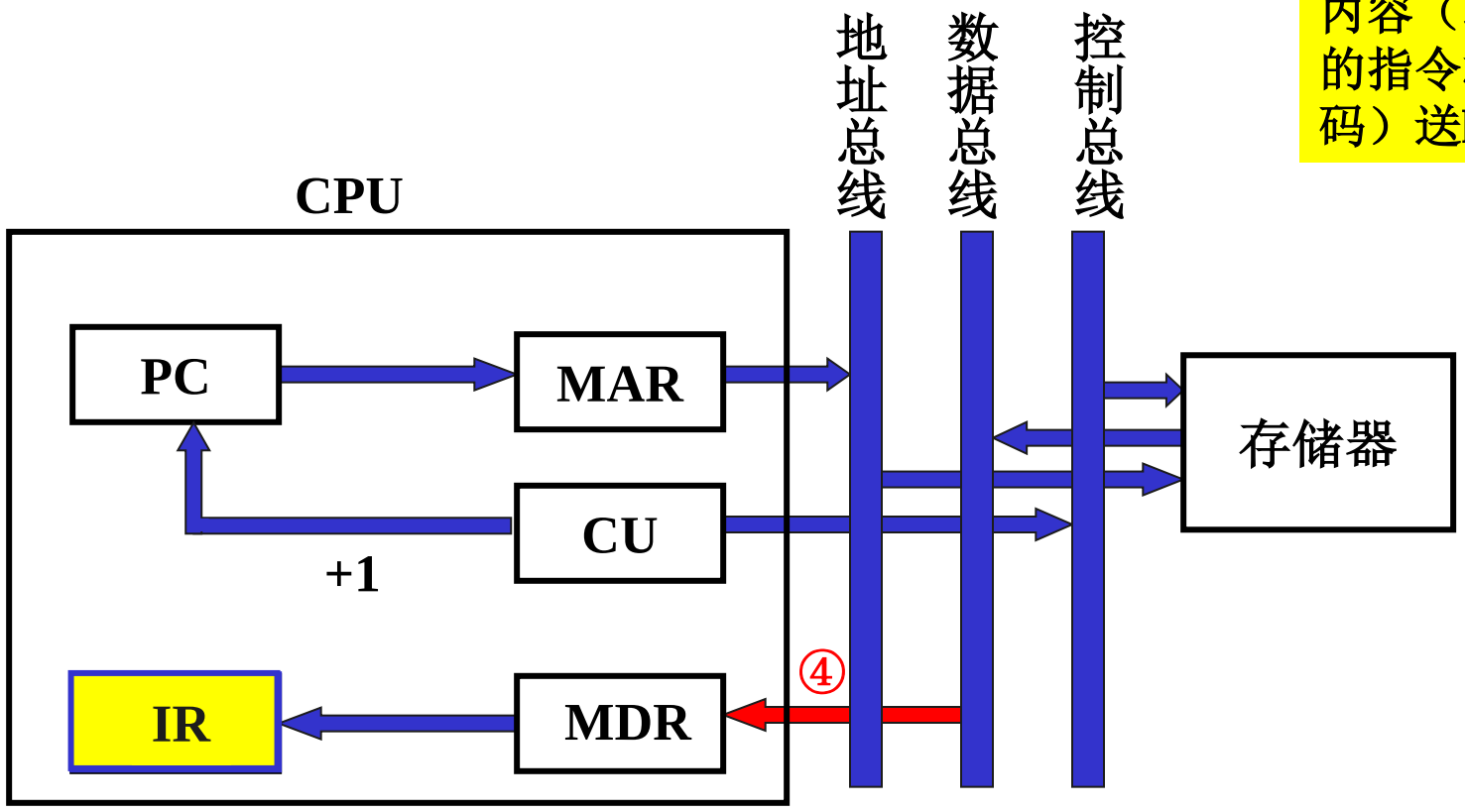
1. 取指周期数据流

③CU发出存储器读的控制信号



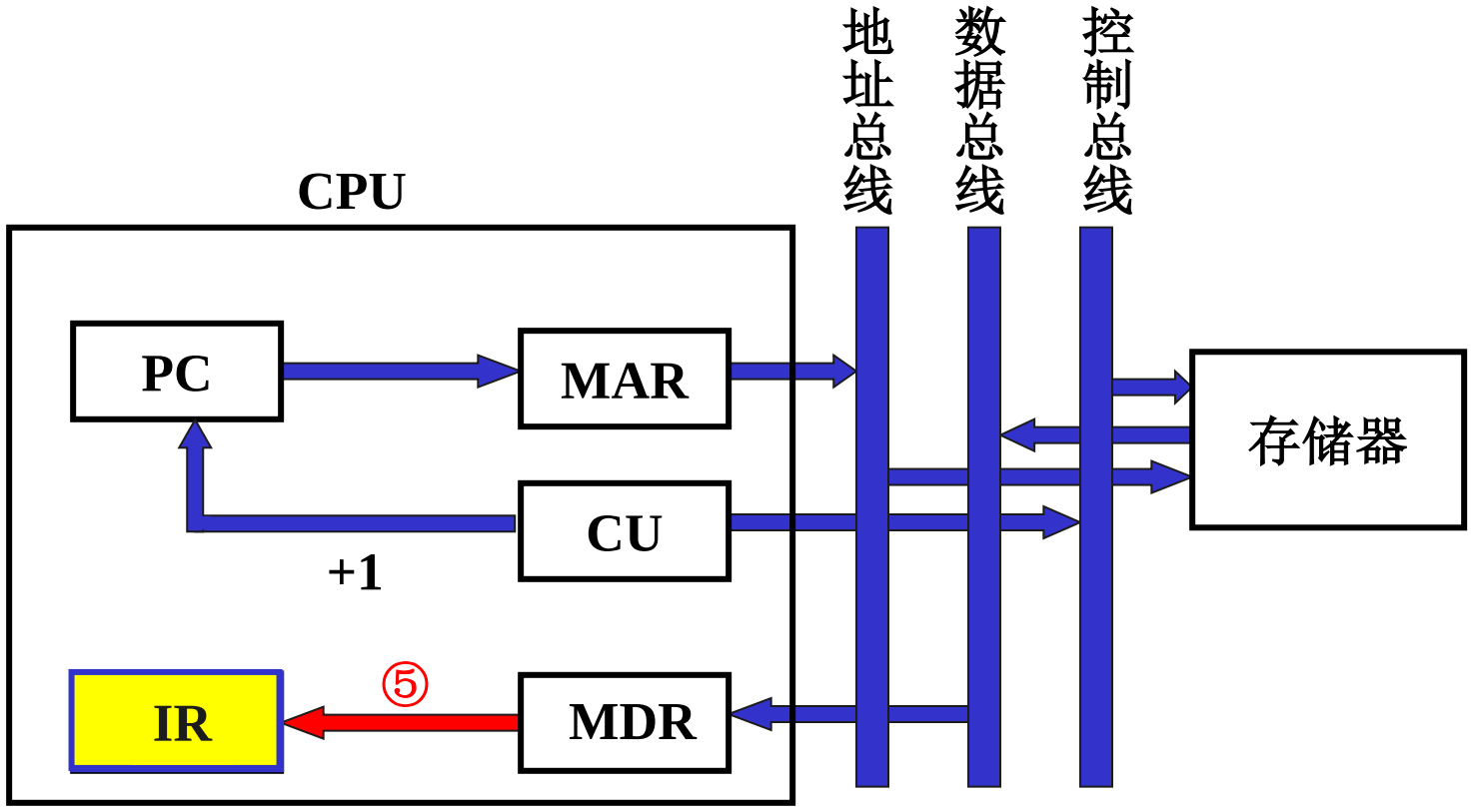
1. 取指周期数据流

④数据总线内容（取出的指令机器码）送MDR



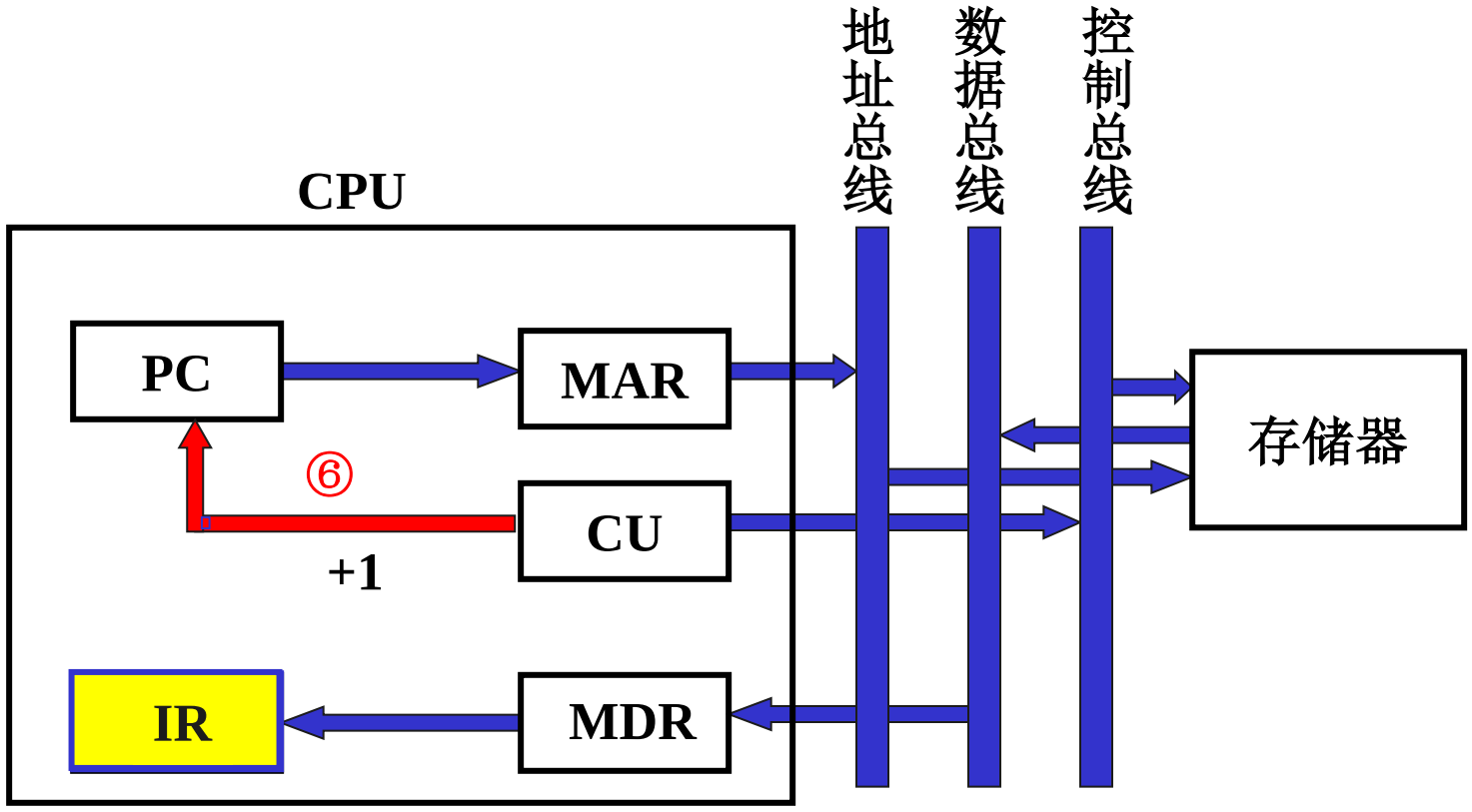
1. 取指周期数据流

⑤MDR送IR

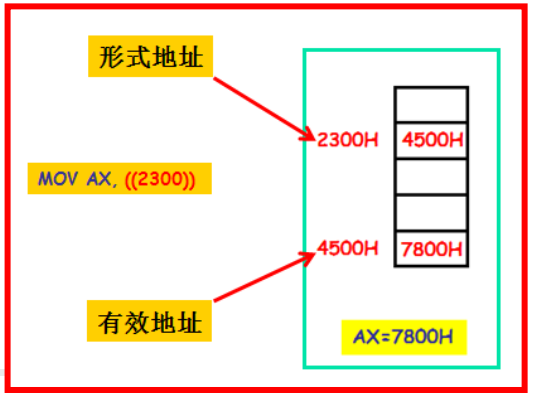


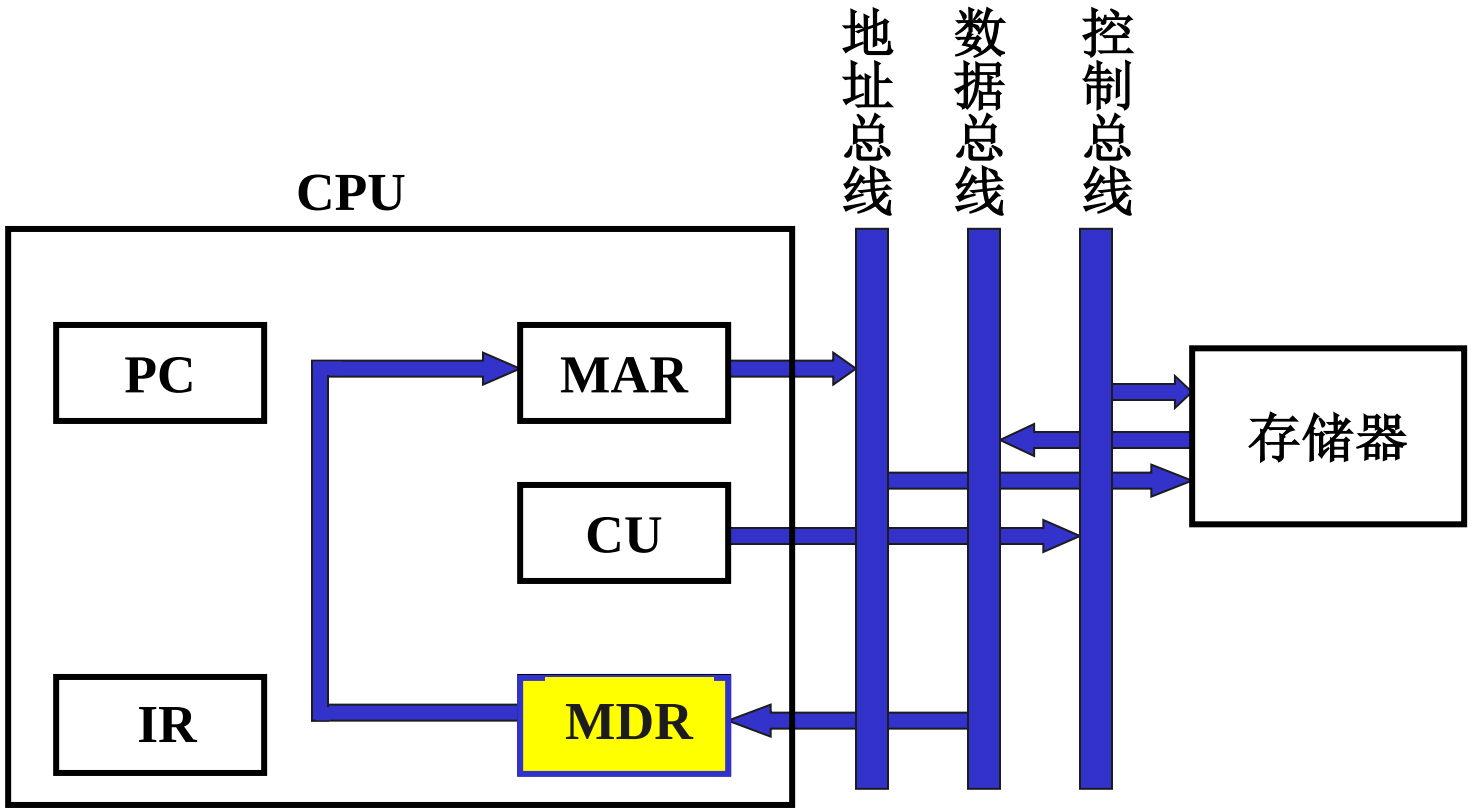
1. 取指周期数据流

⑥PC+1->PC



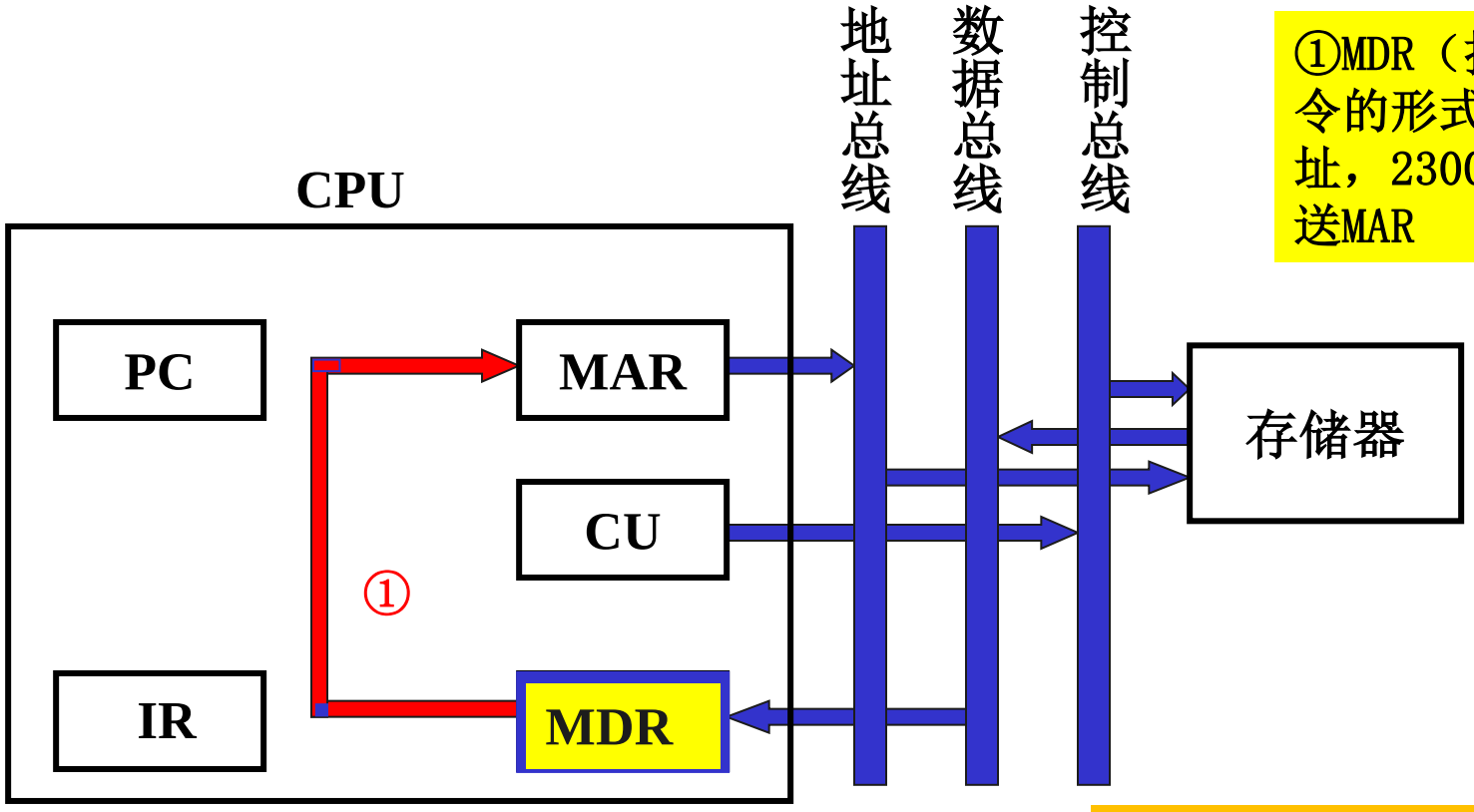
2. 间址周期数据流





间址周期的目的是：由形式地址得到有效地址

2. 间址周期数据流

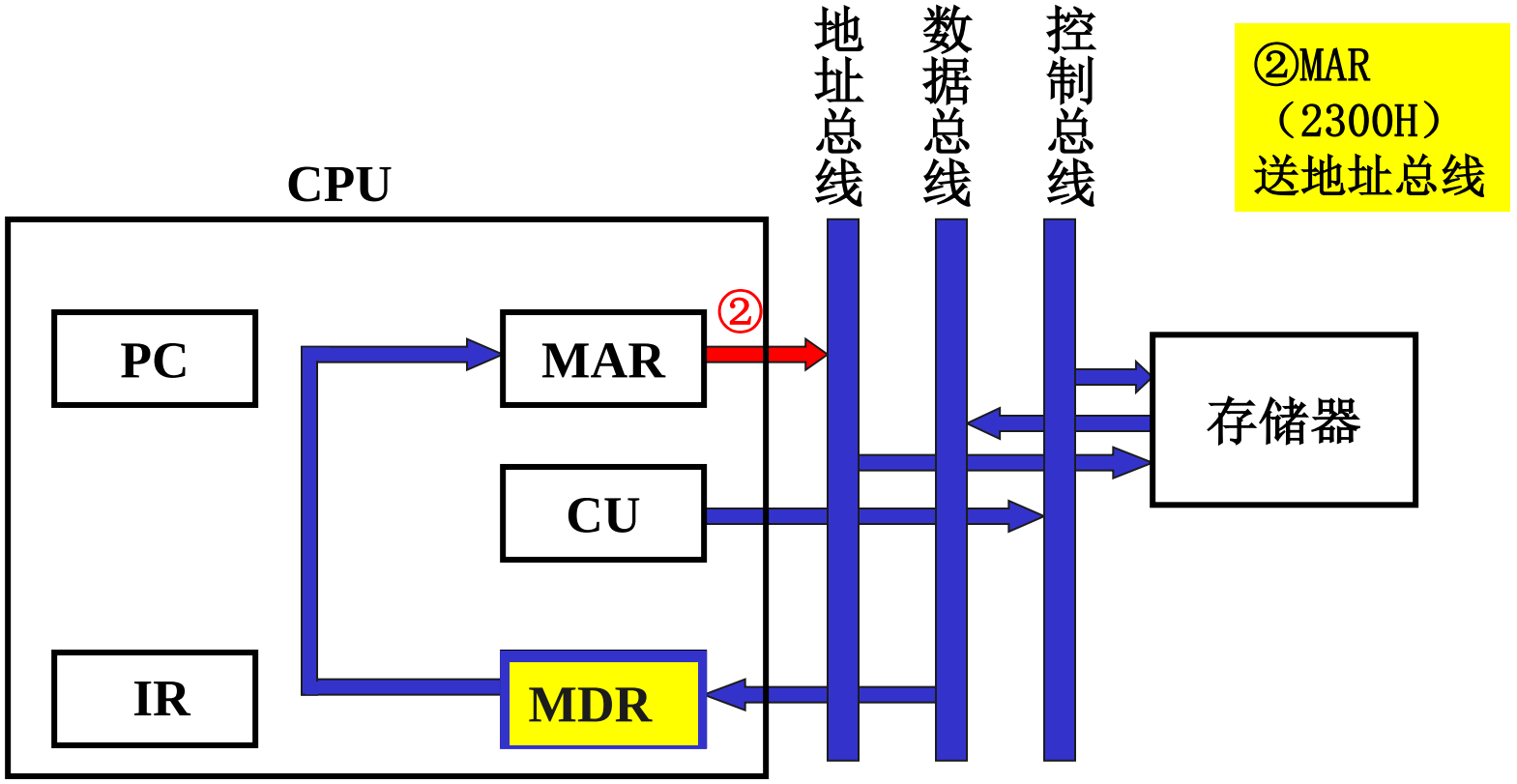


①MDR（指令的形式地址，2300H）送MAR

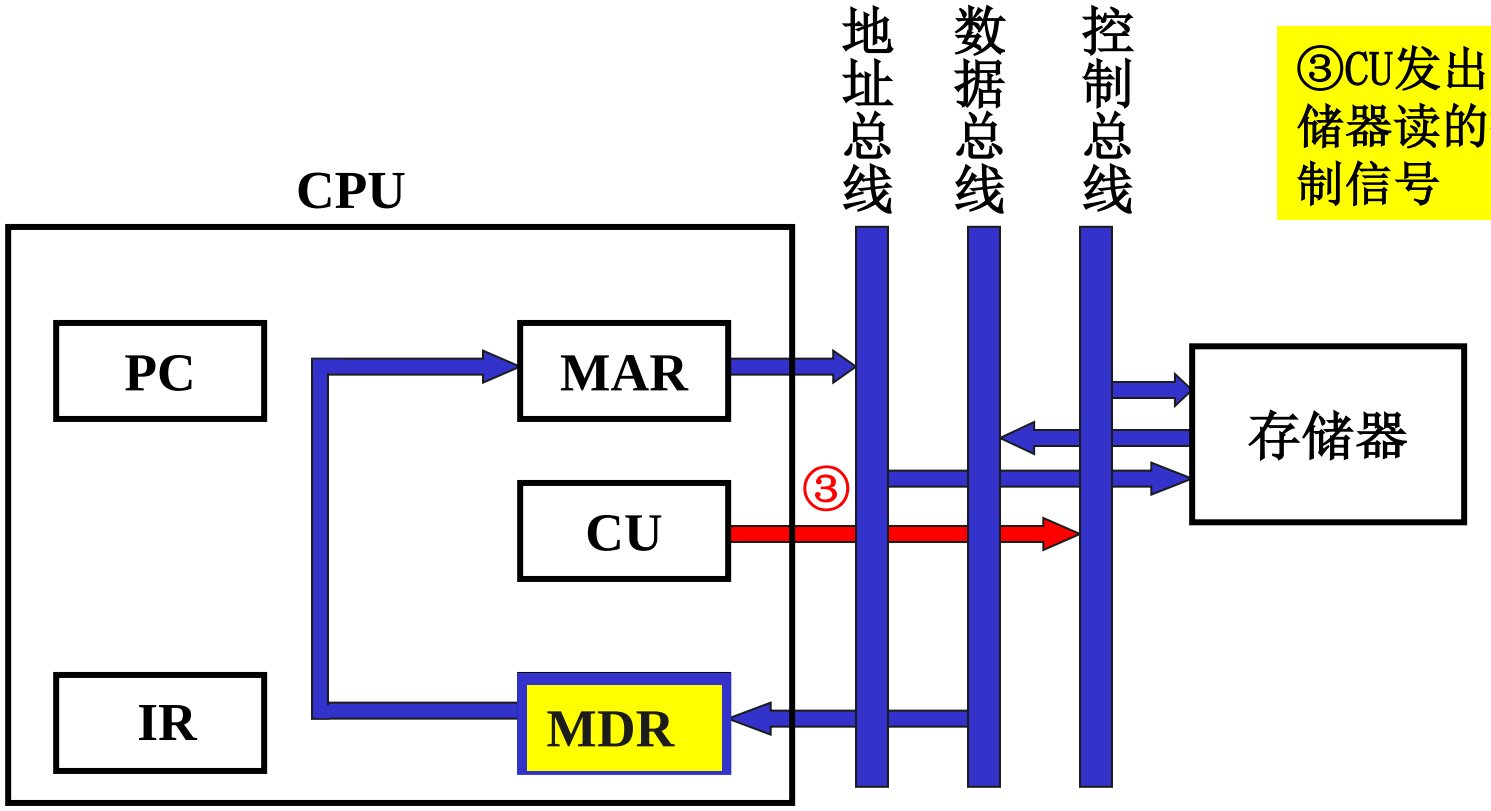
MOV AX, [[2300H]]

间接寻址时，取指周期阶段得到形式地址（如2300H），并放在MDR中

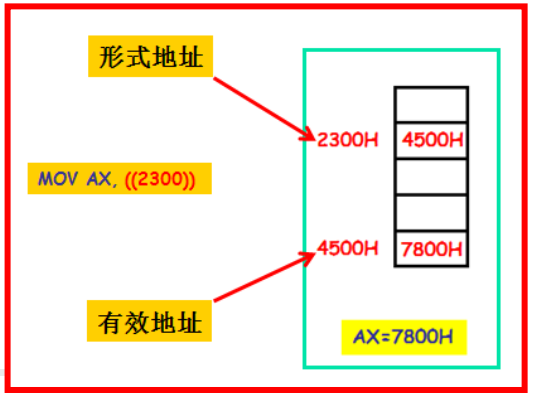
2. 间址周期数据流

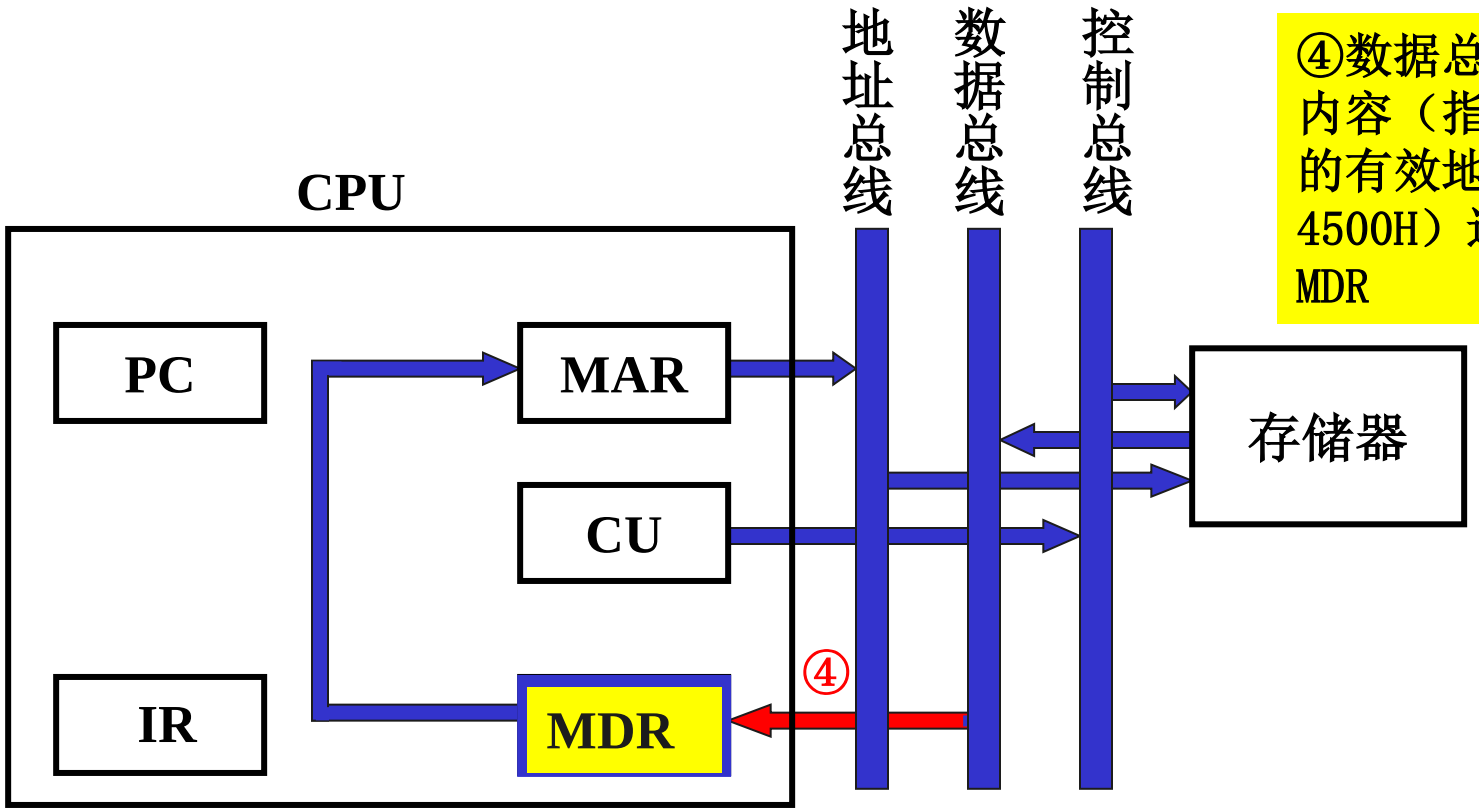


2. 间址周期数据流



2. 间址周期数据流





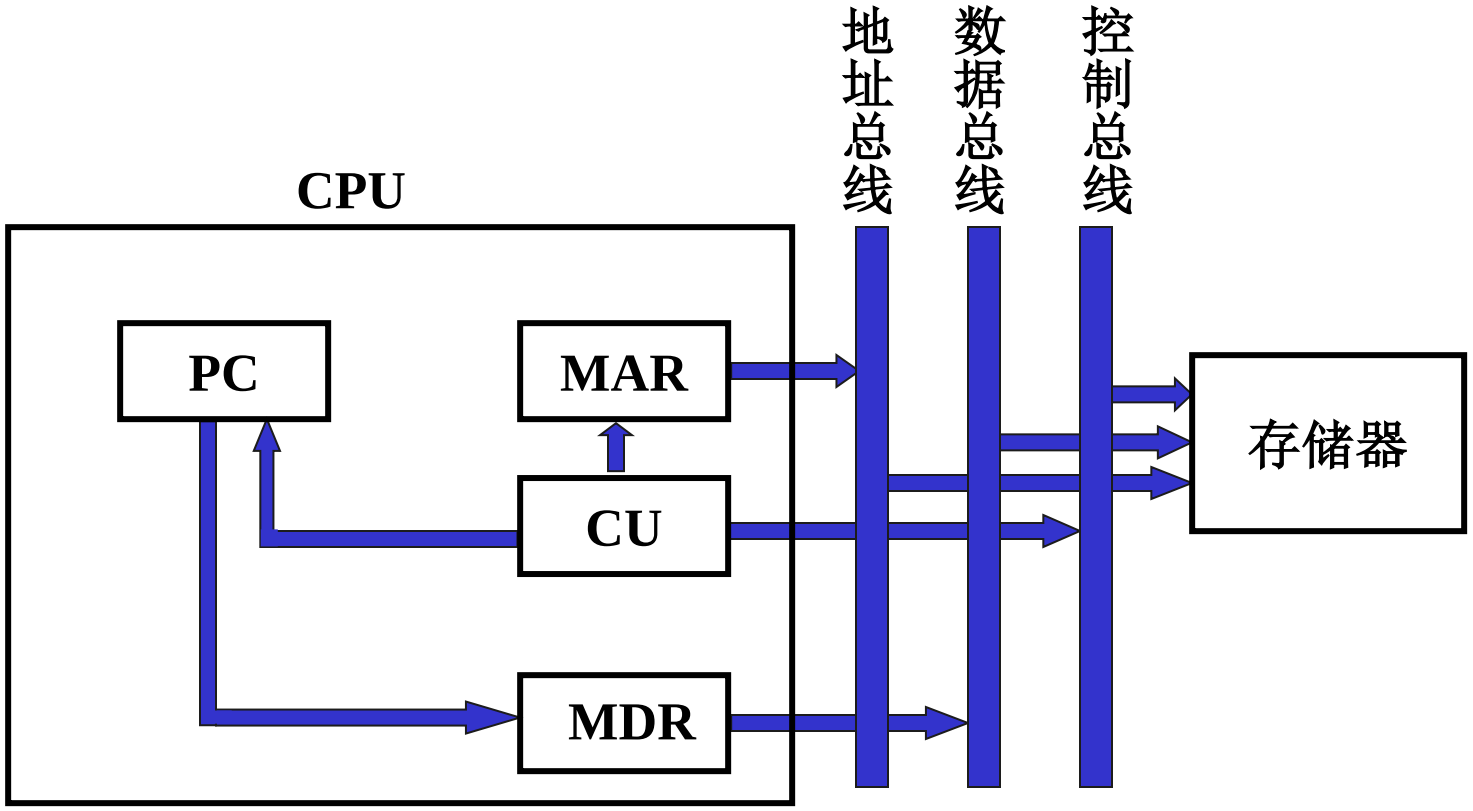
④数据总线内容（指令的有效地址，4500H）送MDR

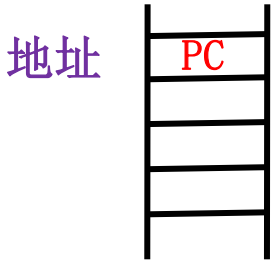
3. 执行周期数据流

不同指令的执行周期数据流不同

4 . 中断周期数据流

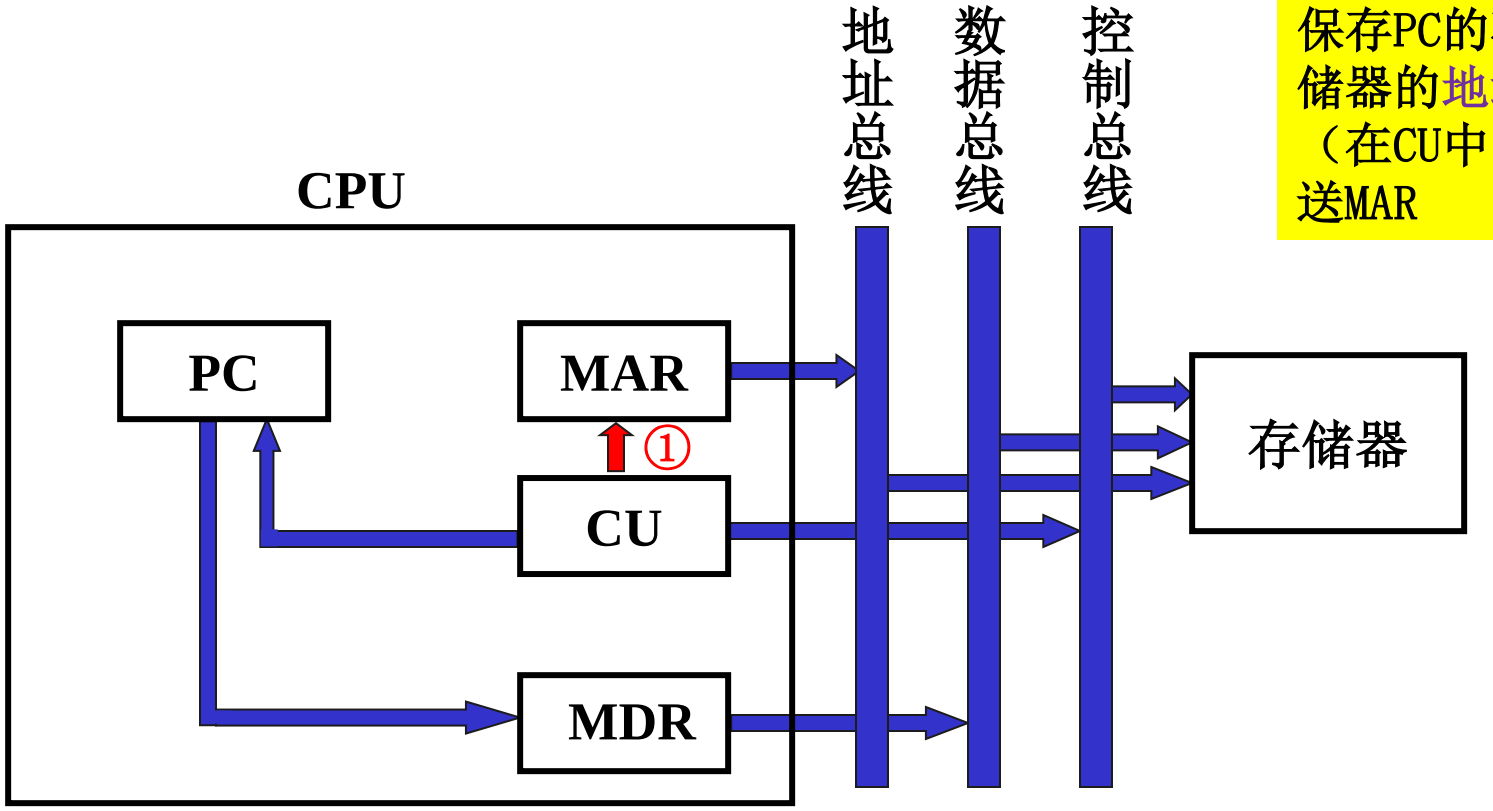
中断周期的目的是：（1）将PC的内容（中断结束后的返回地址）保护起来；（2）将中断服务程序入口地址（在CU中）送PC，从而使CPU开始执行中断服务程序



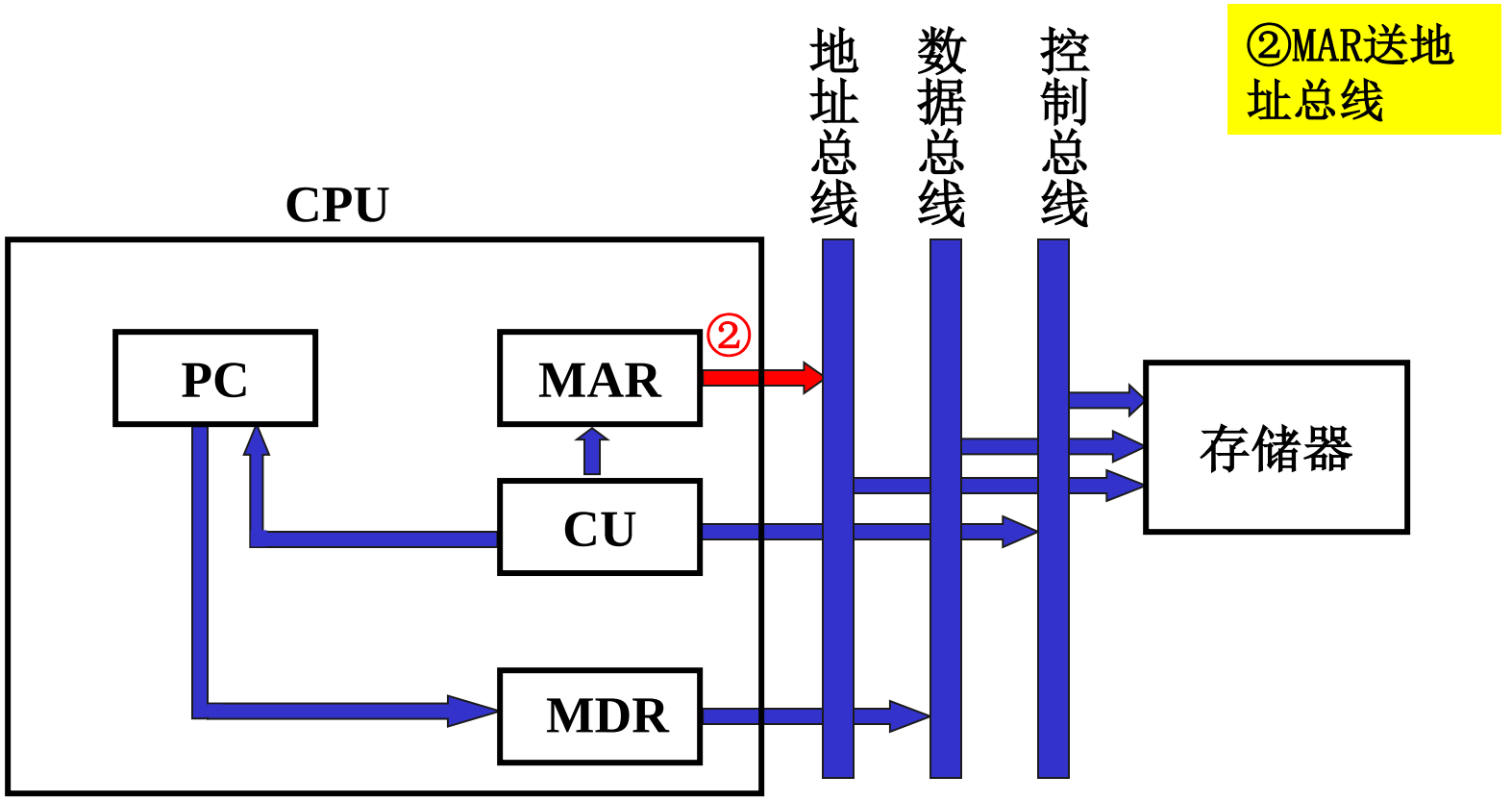


4．中断周期数据流

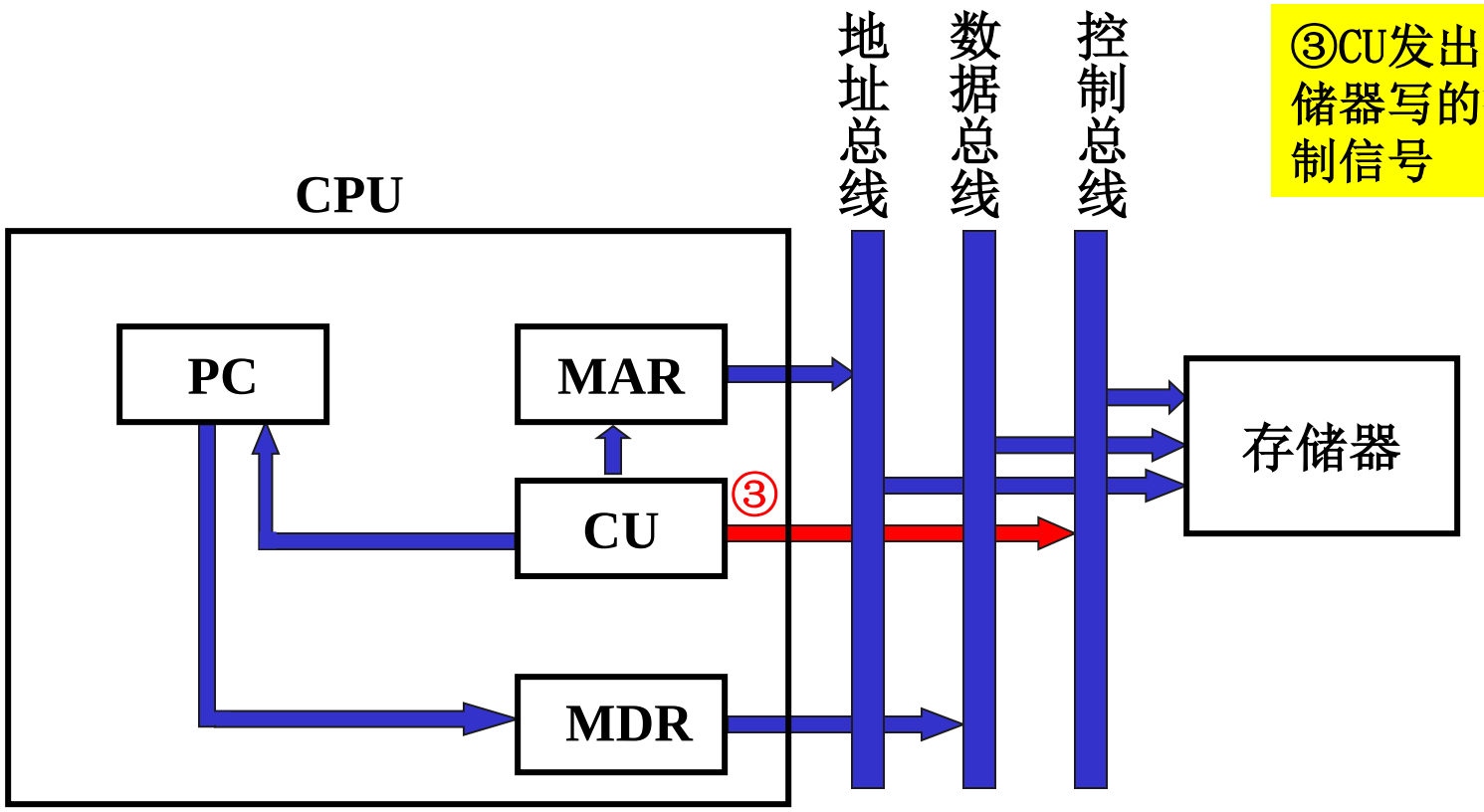
①CU将用于保存PC的存储器的地址（在CU中），送MAR



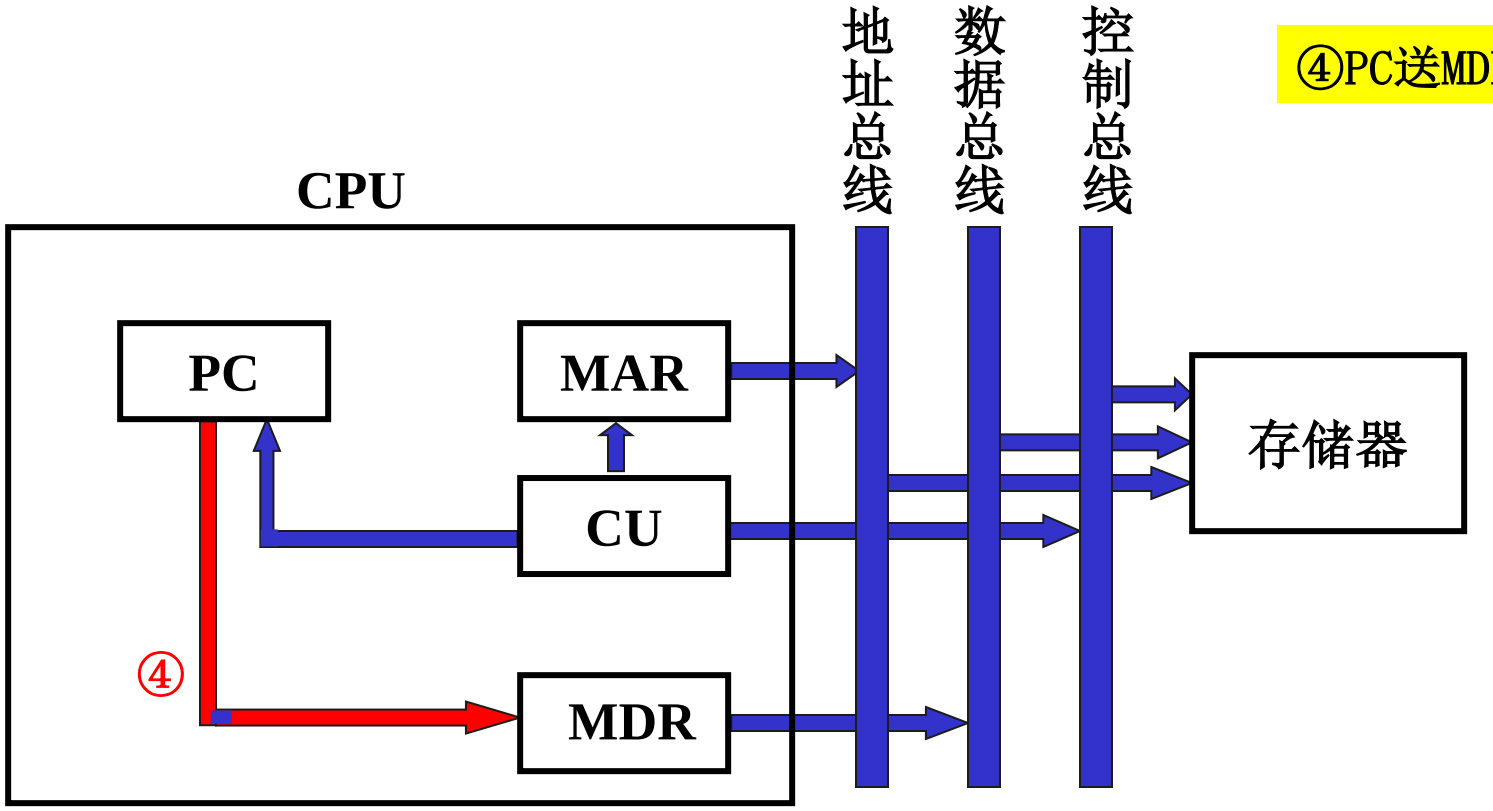
4．中断周期数据流



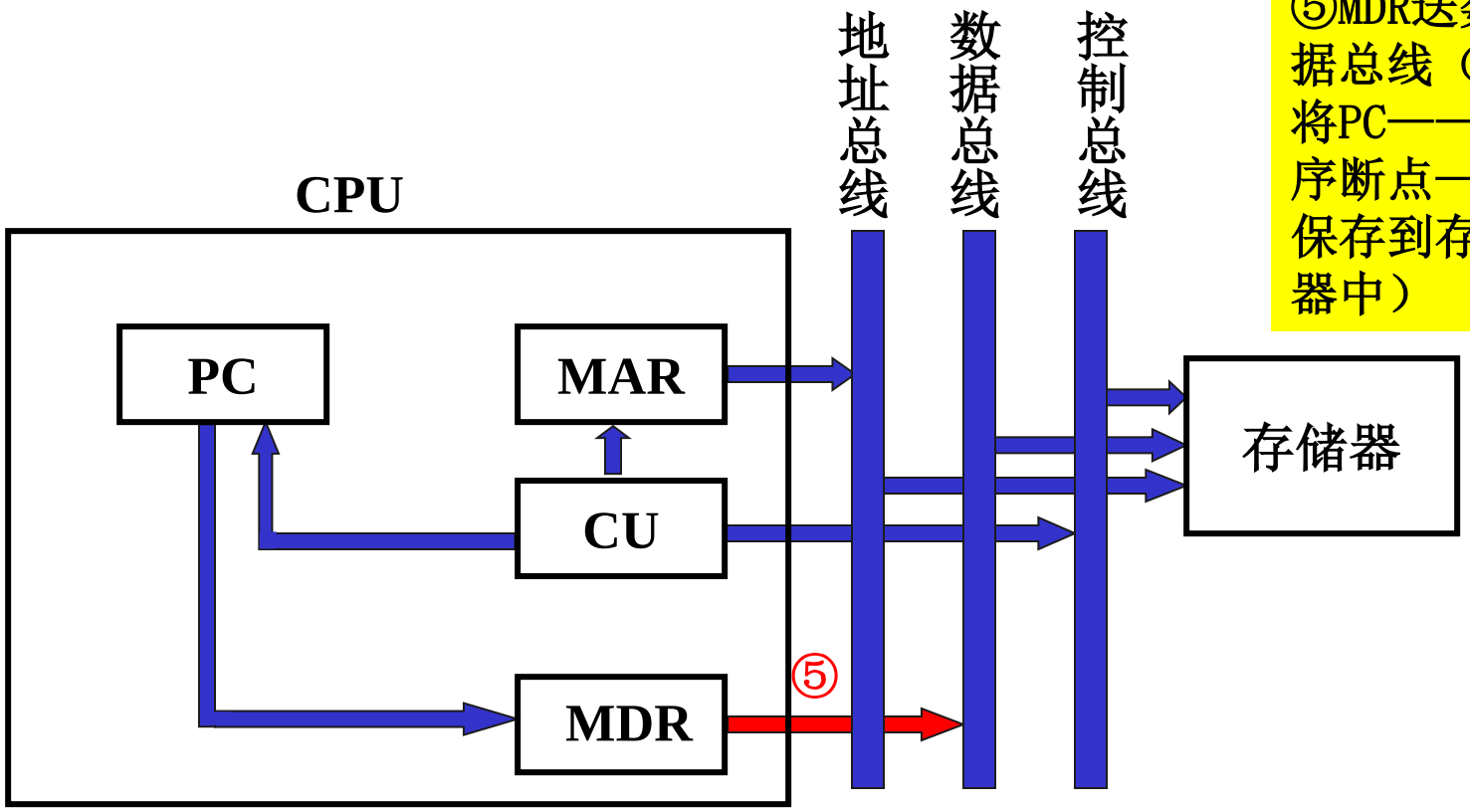
4．中断周期数据流



4．中断周期数据流



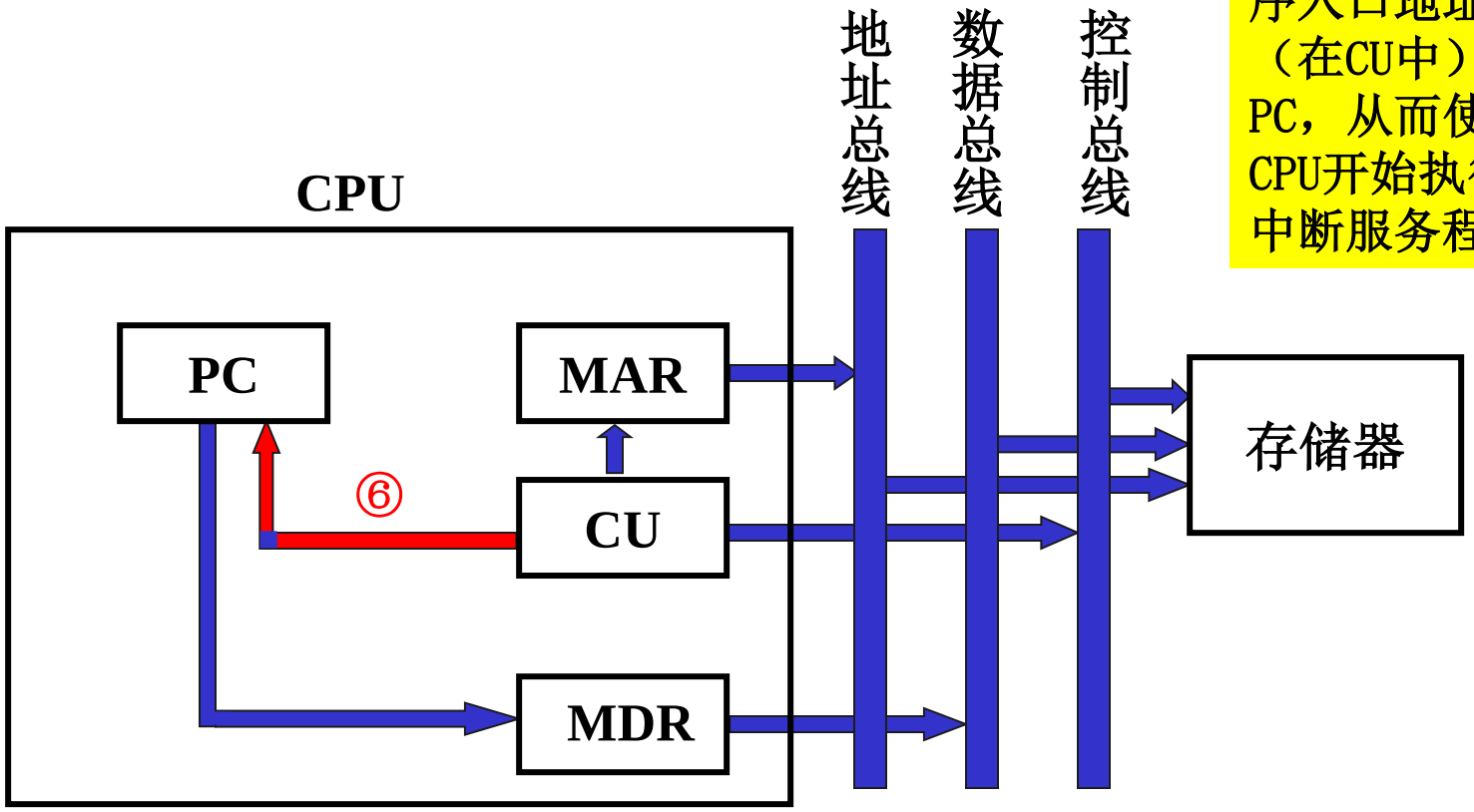
4．中断周期数据流



⑤MDR送数据总线（即将PC——程序断点——保存到存储器中）

4．中断周期数据流

⑥中断服务程序入口地址（在CU中）送PC，从而使CPU开始执行中断服务程序



8.3 指令流水

一、如何提高机器速度

二、系统的并行性

三、指令流水原理

四、影响指令流水线性能的因素

五、流水线性能

六、流水线的多发技术

七、流水线结构

一、如何提高机器速度

1. 提高访存速度

高速存储芯片　　Cache　　多体并行（第4章4.2.7小节）

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(6.5.2 小节)

4. 提高整机处理能力

①高速器件　　②改进系统的结构，开发系统的并行性

二、系统的并行性

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

2. 并行性的等级

过程级
- 程序级
- 进程级

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

- 并行性的等级：
  - 作业级或程序级
  - 任务级或进程级

  } 粗粒度、过程级

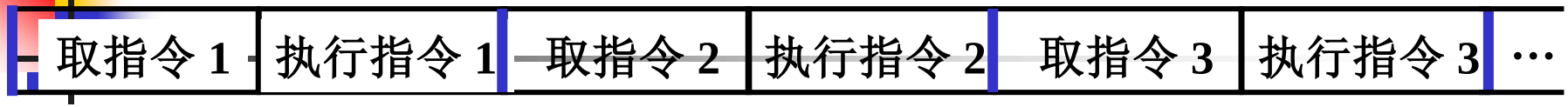
  - 指令之间级
  - 指令内部级

  } 细粒度、指令级

- 粗粒度并行性**(Coarse-grained Parallelism)**，一般用算法（软件）实现
- 细粒度并行性**(Fine-grained Parallelism)**，一般用硬件实现

三、指令流水原理

1. 指令的串行执行

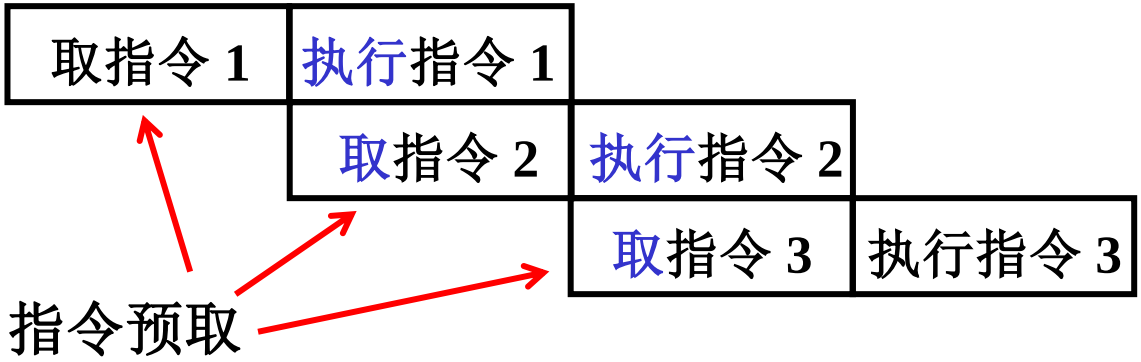


取指令　　取指令部件　　完成

执行指令　　执行指令部件　　完成

总有一个部件 空闲

2. 指令的二级流水

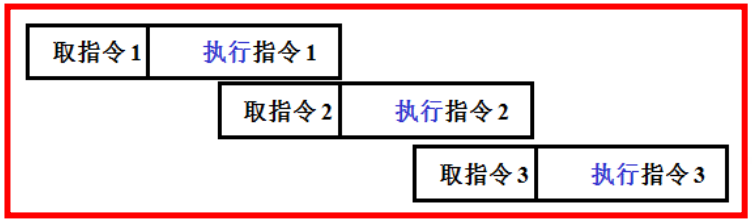


若 取指 和 执行 阶段时间上 完全重叠

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

(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

```
L:  MOV AL, 20H
    ……
    CMP AL,BL
    JZ  L
    MOV AL, 30H
```

解决办法 ？ 猜测法：猜条件不成立

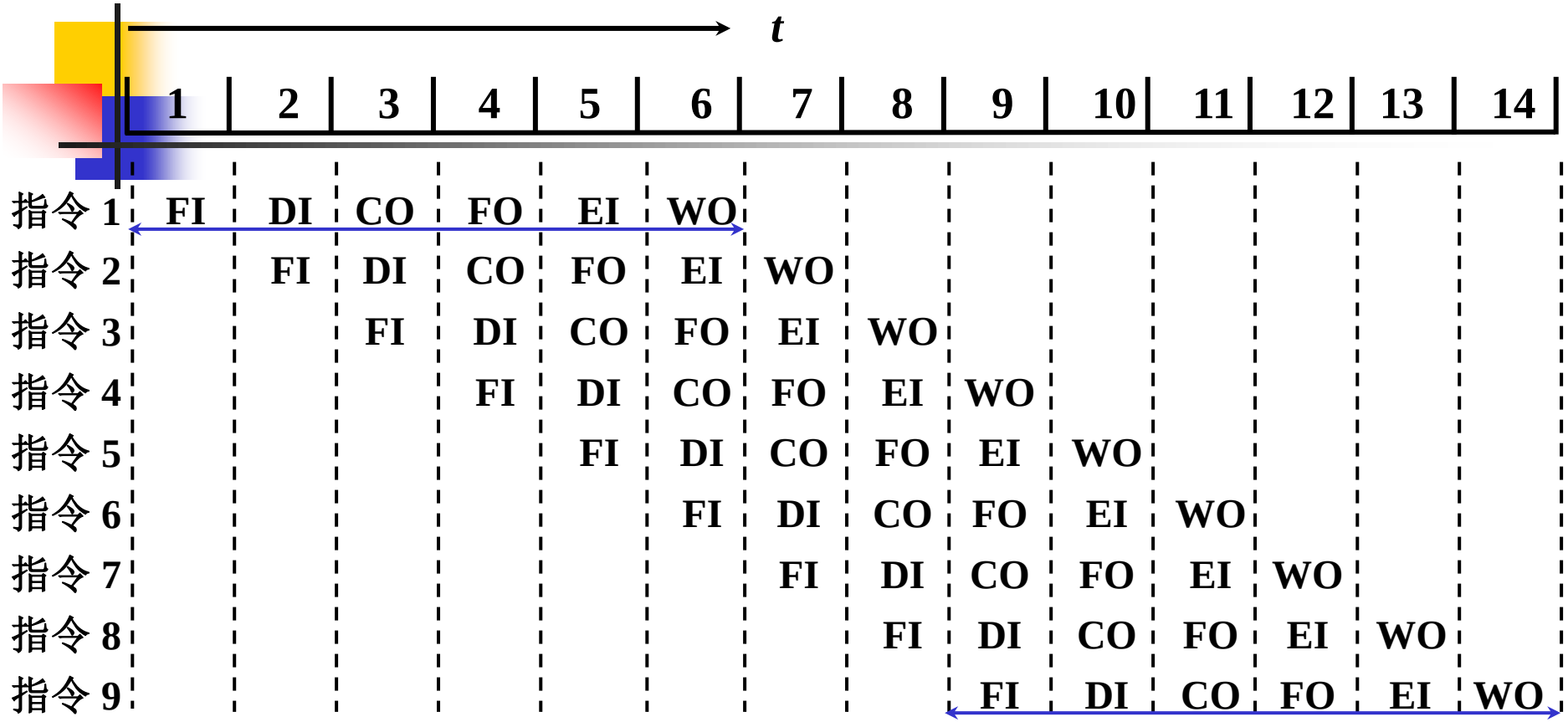
因此，指令的执行效率不可能提高1倍（二级流水）

4. 指令的六级流水

FI（Fetch Instruction）：取指　　DI（Decode Instruction）：指令译码

CO（Calculate Operands）：计算操作数地址　　FO（Fetch Operands）：取操作数

EI（Execute Instruction）：执行指令　　WO（Write Operands）：写操作数



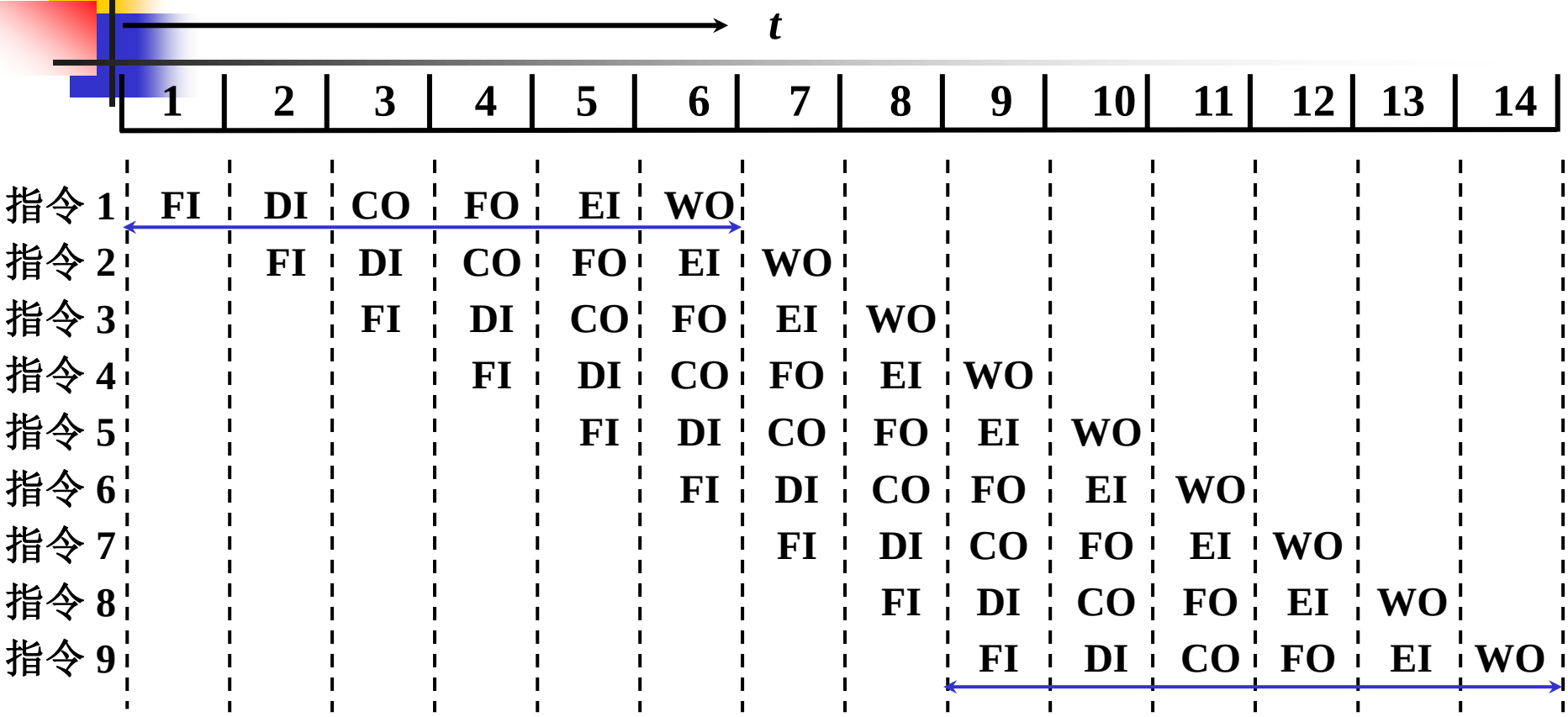
完成一条指令　　6 个时间单位

串行执行　　6 × 9 = 54 个时间单位

六级流水　　14 个时间单位　　54/14=3.86

四、影响指令流水线性能的因素

1. 结构相关　不同指令争用同一功能部件产生资源冲突



FI：取指

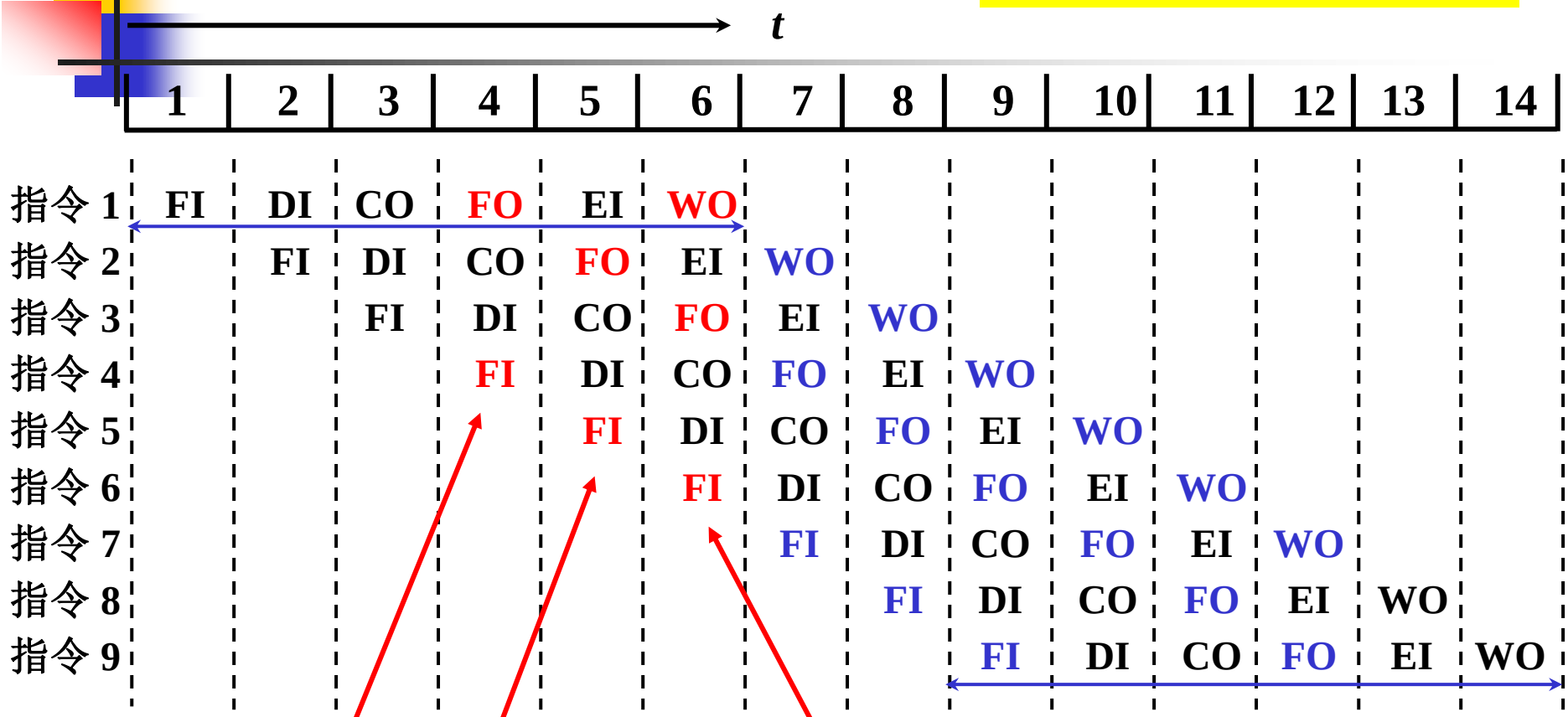
FO：取操作数

WO：写操作数

都要访问存储器

1. 结构相关

由于存在相关，要使流水线不断流，实现起来很困难



结构相关也称为资源相关

1. 结构相关（续）

因为程序的相近指令之间出现某种关联，使指令流水出现停顿，影响流水线效率

解决办法：

- 停顿
- 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）

1. 结构相关（续）

- 假设流水线由5段组成：（五级流水）
  1. IF：取指令
  2. ID：指令译码/读寄存器
  3. EX：执行/计算访存有效地址
  4. MEM：存储器访问
  5. WB：结果写回寄存器

表8.1

- 表8.1：不同类型指令（3类）在各流水段中所进行的操作
  1. ALU指令
  2. 取数/存数指令
  3. 转移指令

表 8.1　不同类型指令在各流水段中所进行的操作

流水段	指令		
	ALU	取/存	转移
IF	取指	取指	取指
ID	译码 读寄存器堆	译码 读寄存器堆	译码 读寄存器堆
EX	执行	计算访存有效地址	计算转移目标地址， 设置条件码
MEM	—	访存(读/写)	若条件成立，将转移 目标地址送 PC
WB	结果写回寄存器堆	将读出的数据写入寄存器堆	—

1. 结构相关（续）

- 表8.2：两条指令同时访存造成结构相关冲突
  - 在第4个时钟周期，第i条指令的MEM和第i+3条指令IF发生了访存冲突
- 表8.3：解决访存冲突的一种方案
  - 第i+3条指令停顿1个时钟周期
- 解决访存冲突的另一种方法：设置两个独立的存储器分别存放操作数和指令，以免取指令和取操作数同时进行时相互冲突，使取某条指令和取另一条指令的操作数实现时间上的重叠
- 还可以使用指令预取技术

为实现并行执行指令的流水线，在执行本条指令时，就同时预先从主存中取出下一条指令的代码，减少等待访存的时间，这种操作叫指令预取技术

表 8.2　两条指令同时访存造成结构相关冲突

指令	时钟周期							
	1	2	3	4	5	6	7	8
LOAD 指令	IF	ID	EX	MEM	WB			
指令 $i+1$		IF	ID	EX	MEM	WB		
指令 $i+2$			IF	ID	EX	MEM	WB	
指令 $i+3$				IF	ID	EX	MEM	WB
指令 $i+4$					IF	ID	EX	MEM

IF（取指令）和MEM（存储器访问）都要访问存储器

表 8.3　解决访存冲突的一种方案

指令	时钟周期								
	1	2	3	4	5	6	7	8	9
LOAD 指令	IF	ID	EX	MEM	WB				
指令 $i+1$		IF	ID	EX	MEM	WB			
指令 $i+2$			IF	ID	EX	MEM	WB		
指令 $i+3$				停顿	IF	ID	EX	MEM	WB
指令 $i+4$						IF	ID	EX	MEM

2. 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

- （1）写后读相关（RAW）　　先写后读

SUB　R_1，R_2，R_3　　；　（R_2）－（R_3）→ R_1

ADD　R_4，R_5，R_1　　；　（R_5）+（R_1）→ R_4

第1条指令还没有写完，第2条指令就读了！

这样导致第2条指令读出来的是错误的值（之前的值）！

2. 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

- （1）写后读相关（RAW）　　先写后读

  SUB　R_1，R_2，R_3　　；　（R_2）－（R_3）→ R_1

  ADD　R_4，R_5，R_1　　；　（R_5）+（R_1）→ R_4

- （2）读后写相关（WAR）　　先读后写

  STA　M，R_2　　；　（R_2）→ M 存储单元

  ADD　R_2，R_4，R_5　　；　（R_4）+（R_5）→ R_2

第1条指令还没有读完，第2条指令就写完了！

这样导致第1条指令读出来的是错误的值（第2条指令写后的值）！

2. 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

- （1）写后读相关（RAW）　　先写后读

  SUB　R_1，R_2，R_3　　；　（R_2）－（R_3）→ R_1

  ADD　R_4，R_5，R_1　　；　（R_5）+（R_1）→ R_4

- （2）读后写相关（WAR）　　先读后写

  STA　M，R_2　　；　（R_2）→ M 存储单元

  ADD　R_2，R_4，R_5　　；　（R_4）+（R_5）→ R_2

- （3）写后写相关（WAW）　　写后再写

  MUL　R_3，R_2，R_1　　；　（R_2）×（R_1）→ R_3

  SUB　R_3，R_4，R_5　　；　（R_4）－（R_5）→ R_3

第1条指令还没有写完，第2条指令就写完了！这样导致R3的值是第1条指令写的值！

2. 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

- （1）写后读相关（RAW）　　先写后读

  SUB　R_1，R_2，R_3　　；　（R_2）－（R_3）→ R_1

  ADD　R_4，R_5，R_1　　；　（R_5）+（R_1）→ R_4

- （2）读后写相关（WAR）　　先读后写

  STA　M，R_2　　；　（R_2）→ M 存储单元

  ADD　R_2，R_4，R_5　　；　（R_4）+（R_5）→ R_2

- （3）写后写相关（WAW）　　写后再写

  MUL　R_3，R_2，R_1　　；　（R_2）×（R_1）→ R_3

  SUB　R_3，R_4，R_5　　；　（R_4）－（R_5）→ R_3

解决办法　• 后推法　• 采用 旁路技术

2. 数据相关（续）

IF：取指令
ID：指令译码/读寄存器
EX：执行/计算访存有效地址
MEM：存储器访问
WB：结果写回寄存器

- ADD R1,R2,R3
- SUB R4,R1,R5

先写后读(RAW)相关

第1条指令还没有写完，第2条指令就读了

- 表8.4：ADD指令和SUB指令发生先写后读(RAW)的数据相关冲突

表 8.4 ADD 和 SUB 指令发生先写后读(RAW)的数据相关冲突

指令	时钟周期					
	1	2	3	4	5	6
ADD	IF	ID	EX	MEM	WB	
SUB		IF	ID	EX	MEM	WB

读 R_1　　写 R_1

2. 数据相关（续）

- ADD R1,R2,R3
- SUB R4,R1,R5
- AND R6,R1,R7
- OR R8,R1,R9
- XOR R10,R1,R11

先写后读(RAW)相关

- 表8.5：未对数据相关进行特殊处理的流水线
- 表8.6：对数据相关进行特殊处理的流水线（停顿3个时钟周期）

后推法

- 还可以采用定向技术（也称为旁路技术或相关专用通路技术），图8.16（文字的解释见P351下半段）

旁路技术

IF：取指令
ID：指令译码/读寄存器
EX：执行/计算访存有效地址
MEM：存储器访问
WB：结果写回寄存器

表 8.5　未对数据相关进行特殊处理的流水线

指令	时钟周期								
	1	2	3	4	5	6	7	8	9
ADD	IF	ID	EX	MEM	WB				
SUB		IF	ID	EX	MEM	WB			
AND			IF	ID	EX	MEM	WB		
OR				IF	ID	EX	MEM	WB	
XOR					IF	ID	EX	MEM	WB

表 8.6　对数据相关进行特殊处理的流水线

指令	时钟周期											
	1	2	3	4	5	6	7	8	9	10	11	12
ADD	IF	ID	EX	MEM	WB							
SUB		IF				ID	EX	MEM	WB			
AND			IF				ID	EX	MEM	WB		
OR				IF				ID	EX	MEM	WB	
XOR					IF				ID	EX	MEM	WB

停顿3个时钟周期
等第1条指令写完

后推法

旁路技术

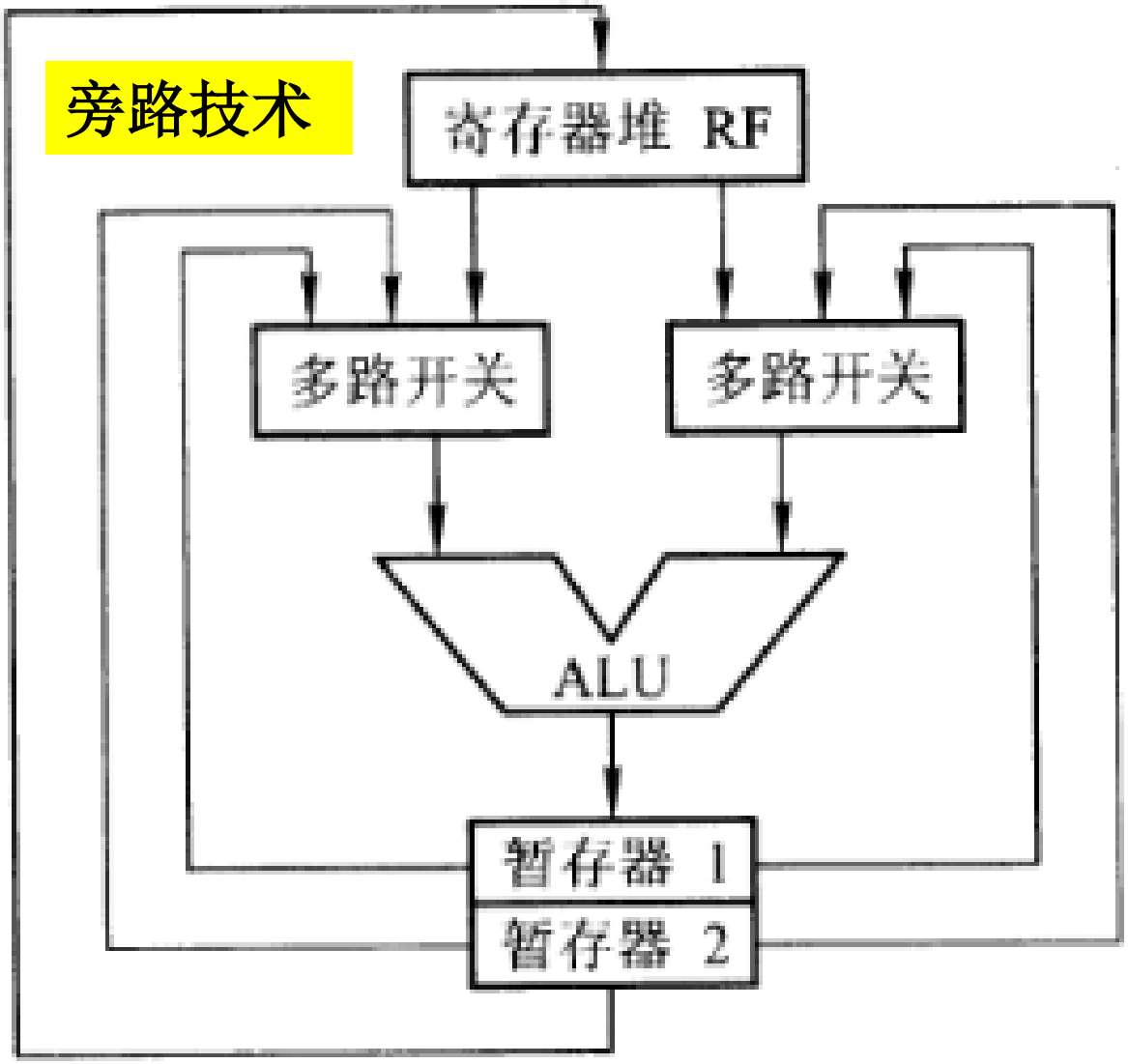


图 8.16　带有旁路技术的 ALU 部件

```
ADD  R1,R2,R3
SUB  R4,R1,R5
AND R6,R1,R7
OR   R8,R1,R9
XOR R10,R1,R11
```

不必等到ADD指令的执行结果送回寄存器（R1）后，再从该寄存器（R1）中取出结果，作为下一条指令（SUB指令）的源操作数；而是直接将（ADD指令）执行结果送到其他指令（SUB指令等）所需要的地方（通过暂存器和多路开关）

3. 控制相关

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

求N个数（存放在存储器中）平均值的程序

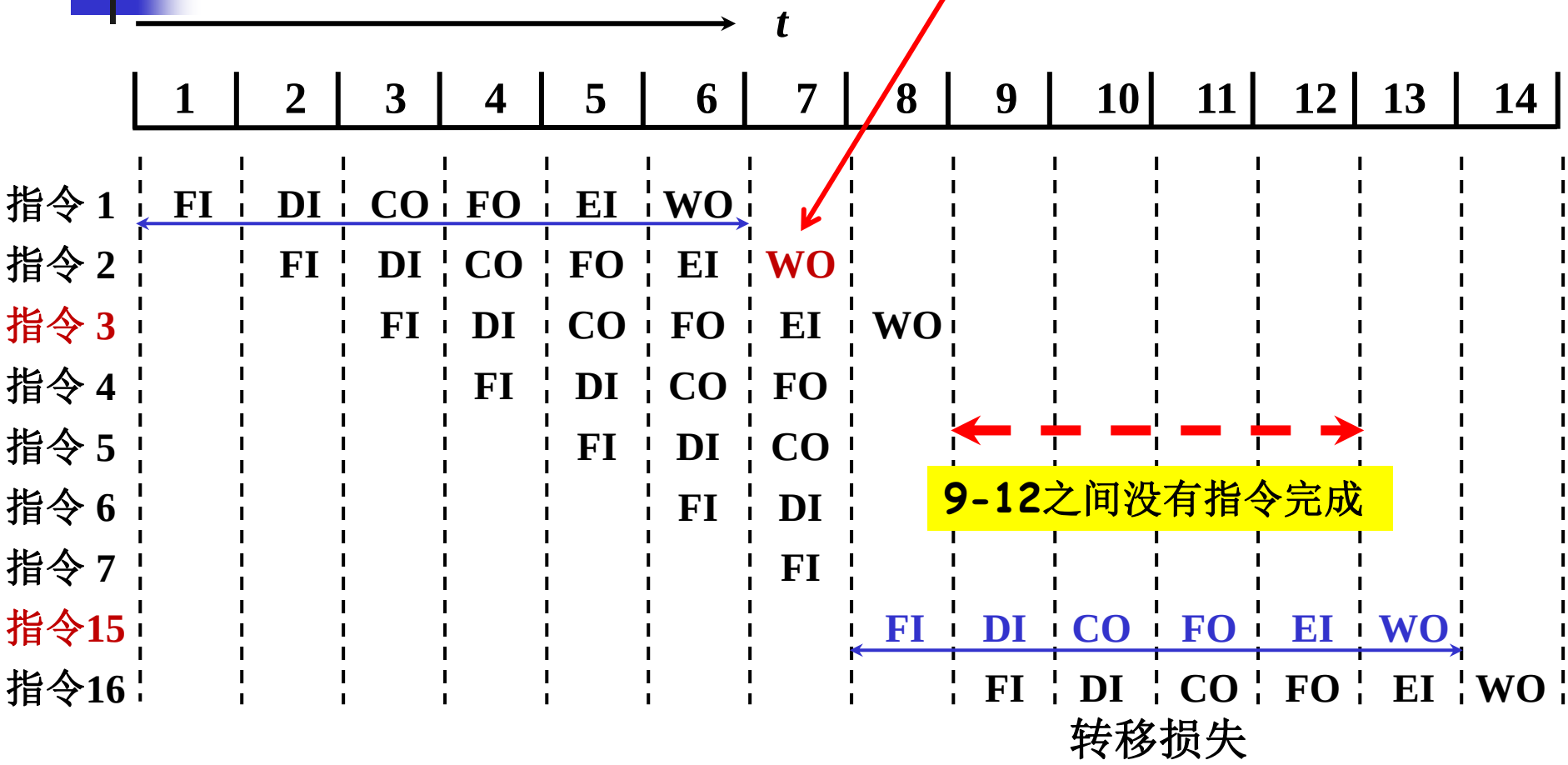
教材P316

3. 控制相关（续）

FI（Fetch Instruction）：取指　　DI（Decode Instruction）：指令译码
CO（Calculate Operands）：计算操作数地址　　FO（Fetch Operands）：取操作数
EI（Execute Instruction）：执行指令　　WO（Write Operands）：写操作数

设 指令3 是转移指令

指令3为条件转移指令，要根据指令2的结果，决定是否转移到指令15



五、流水线性能

1. 吞吐率 (Throughput Rate)

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

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

(9/14)=0.64 * T_{pmax}

- 实际吞吐率

连续处理 n 条指令的吞吐率为

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

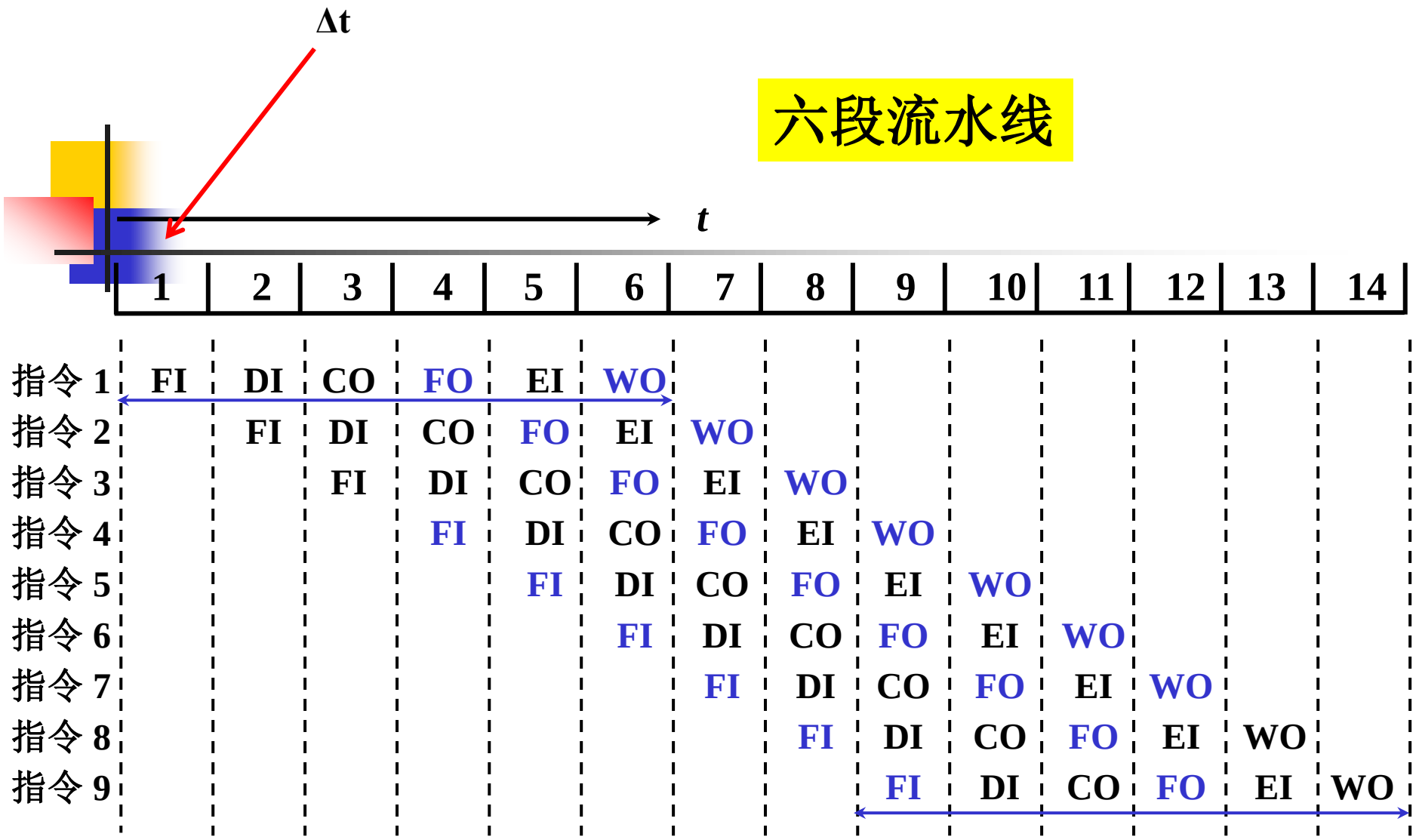
m(m级流水)=6

n(n条指令)=9

n=指令数=9

m=流水段(级)数=6

m+(n-1)=14


Δt
t
六段流水线
1 2 3 4 5 6 7 8 9 10 11 12 13 14
指令 1 FI DI CO FO EI WO
指令 2 FI DI CO FO EI WO
指令 3 FI DI CO FO EI WO
指令 4 FI DI CO FO EI WO
指令 5 FI DI CO FO EI WO
指令 6 FI DI CO FO EI WO
指令 7 FI DI CO FO EI WO
指令 8 FI DI CO FO EI WO
指令 9 FI DI CO FO EI WO
n=9 指令条数
m=6 流水段（级）数
m+(n-1)=14
Tp=(9/14)*Tpmax
Tp：实际吞吐率
Tpmax：最大吞吐率

2. 加速比 S_p (Speedup Ratio)

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

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

理想加速比=6

完成 n 条指令在等效的非流水线上共需

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

(6×9)/14=3.86

则

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

m(m级流水)=6

n(n条指令)=9

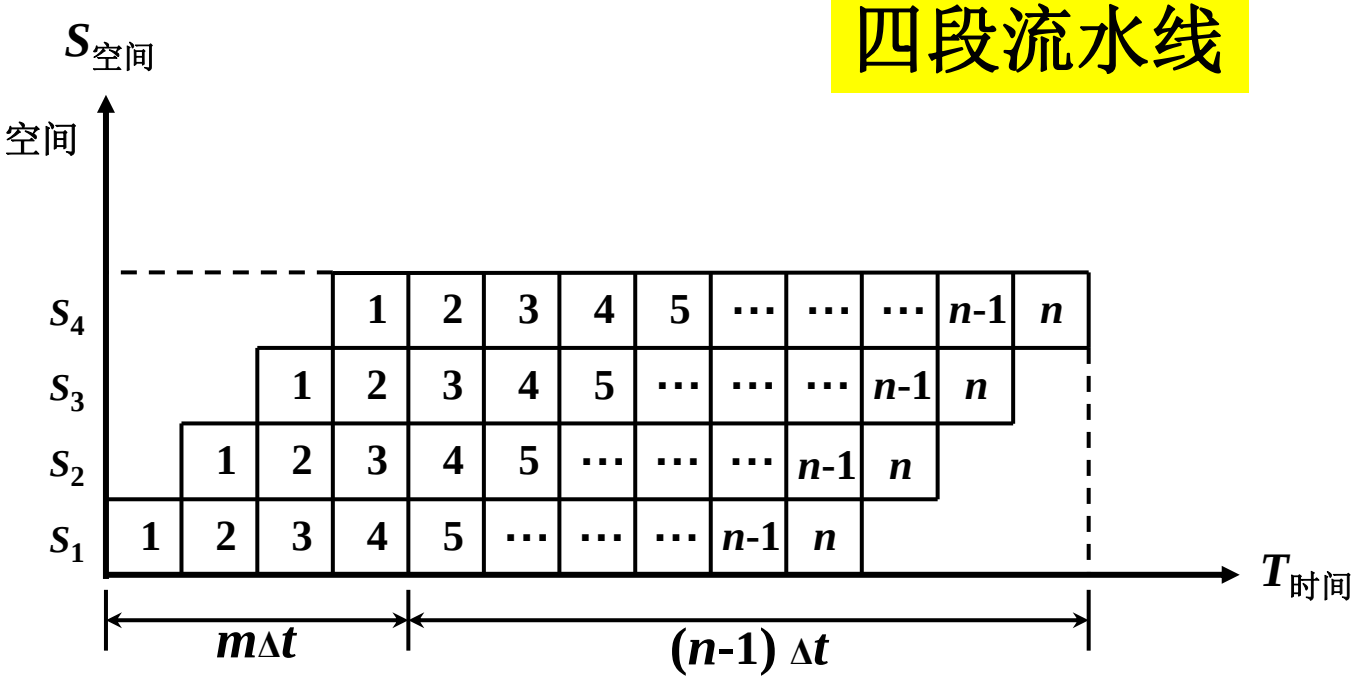
3. 效率 (Efficient)

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

四段流水线



3. 效率（续）

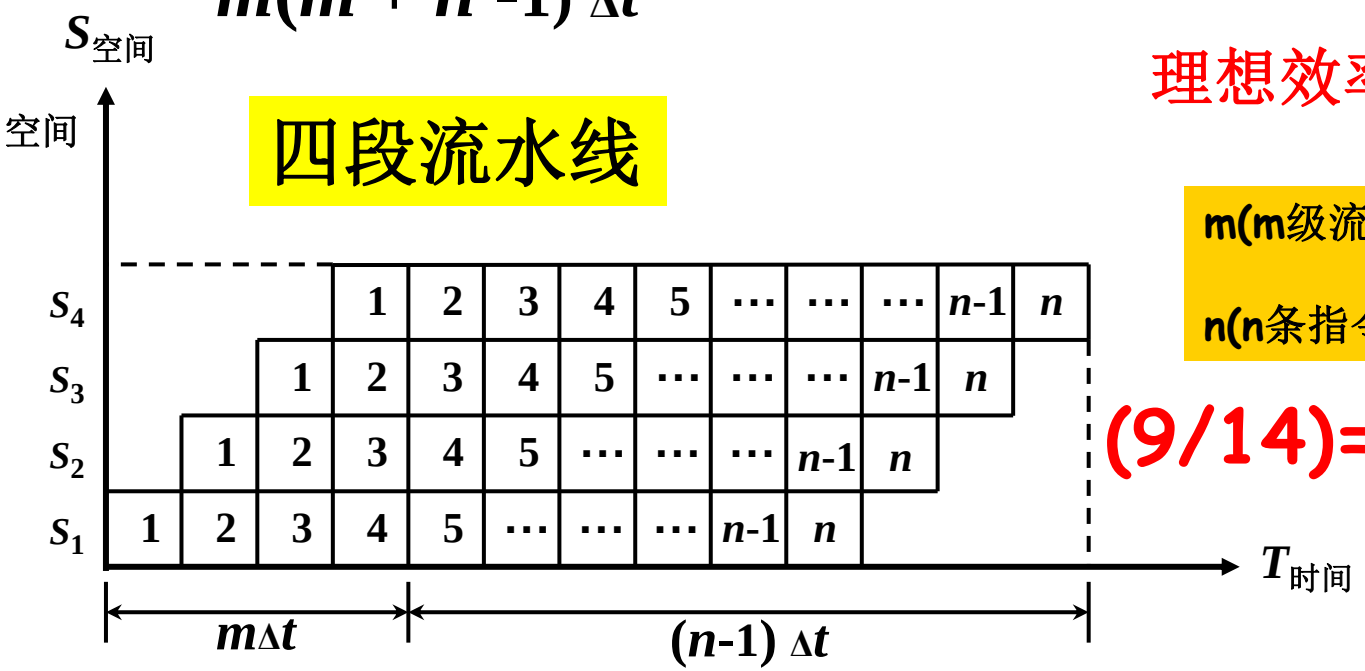
流水线中各功能段的 利用率

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

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

效率=n/(m+n-1)

理想效率=1



m(m级流水)=6

n(n条指令)=9

(9/14)=0.64

- **例8.1**：假设流水线分取指(IF)、译码(ID)、执行(EX)、写回(WR)4个过程段，共有10条指令连续输入此流水段

(1)画出指令周期流程

(2)画出非流水线时空图

(3)画出流水线时空图

(4)假设时钟周期为100ns，求流水线的实际吞吐率

(5)求该流水处理器的加速比

- **例8.1**：假设流水线分取指(IF)、译码(ID)、执行(EX)、写回(WR)4个过程段，共有10条指令连续输入此流水段

(1)画出指令周期流程

(2)画出非流水线时空图

(3)画出流水线时空图

(4)假设时钟周期为100ns，求流水线的实际吞吐率

(5)求该流水处理器的加速比

- 解：

$\Delta t = 100ns$

- (1)图8.19(a)
- (2)图8.19(b)
- (3)图8.19(c)

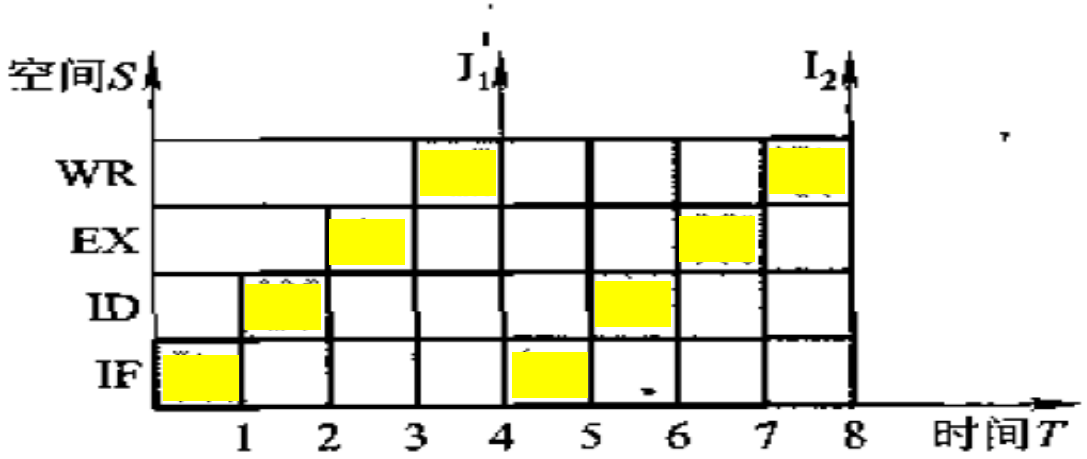
n(10条指令)=10　　m(4级流水)=4

- (4)实际吞吐率为：(10/13)X(1/100ns)=0.77X10^7条指令/秒

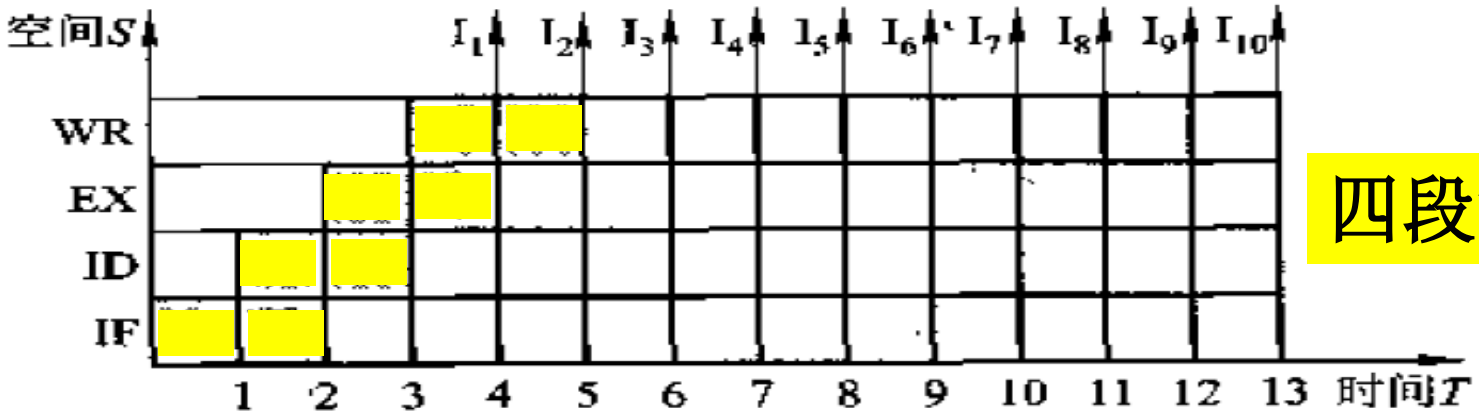
- (5)加速比：(4X10)/(4+(10-1))=40/13=3.08

效率 = n/(m+n-1) = 10/13 = 77%

（a） 指令周期流程



（b） 非流水线时空图



四段流水线

（c） 标准流水线时空图

图 8.19 例 8.1 答图

六、流水线的多发技术(3个技术)

1. 超标量技术（Superscalar）

```
MOV BL,8
ADD AX,1756H
ADD CL,4EH
```

- 每个时钟周期内可 并发多条独立指令

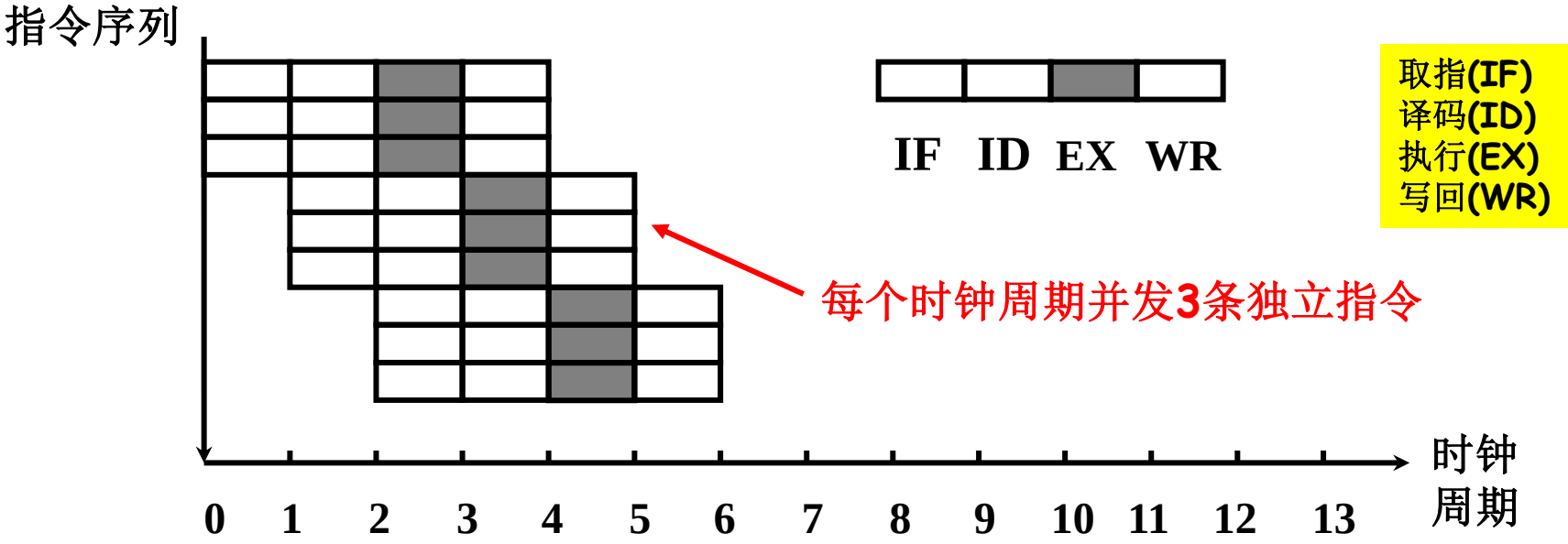
  配置多个功能部件

```
INC AX
ADD AX,BX
MOVDS,AX
```

X

- 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



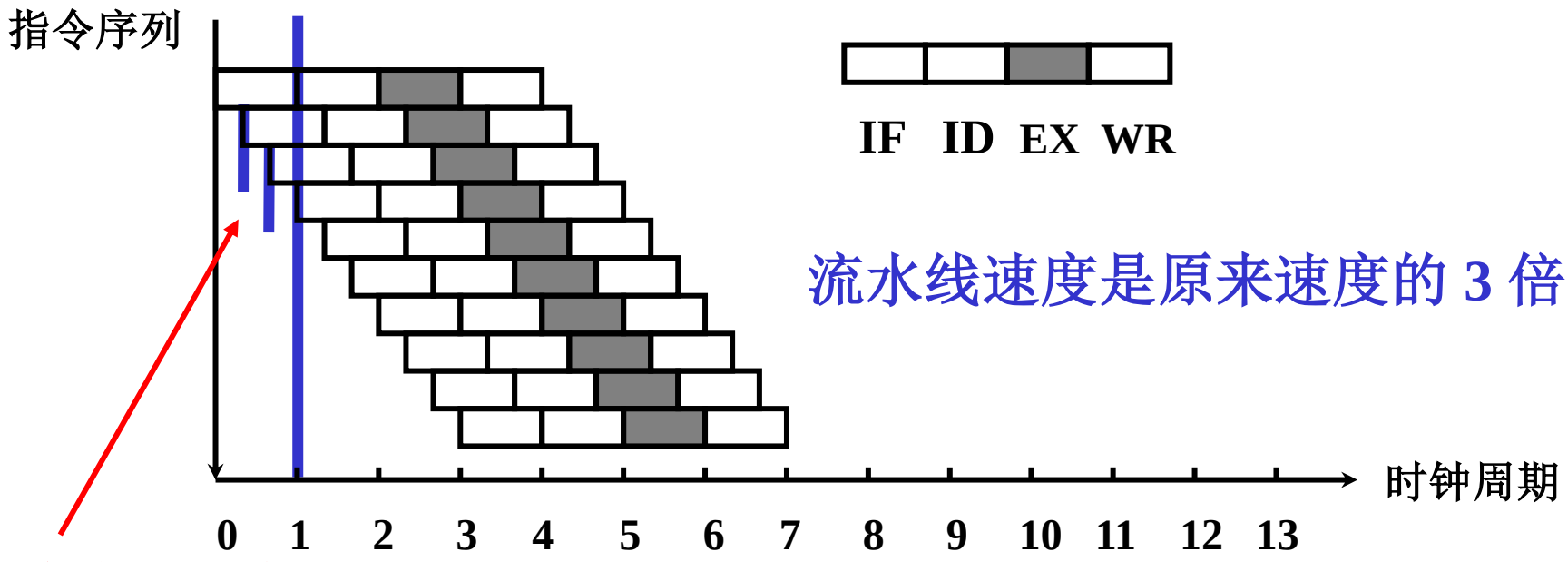
2. 超流水线技术（Superpiplined）

- 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（VLIW）

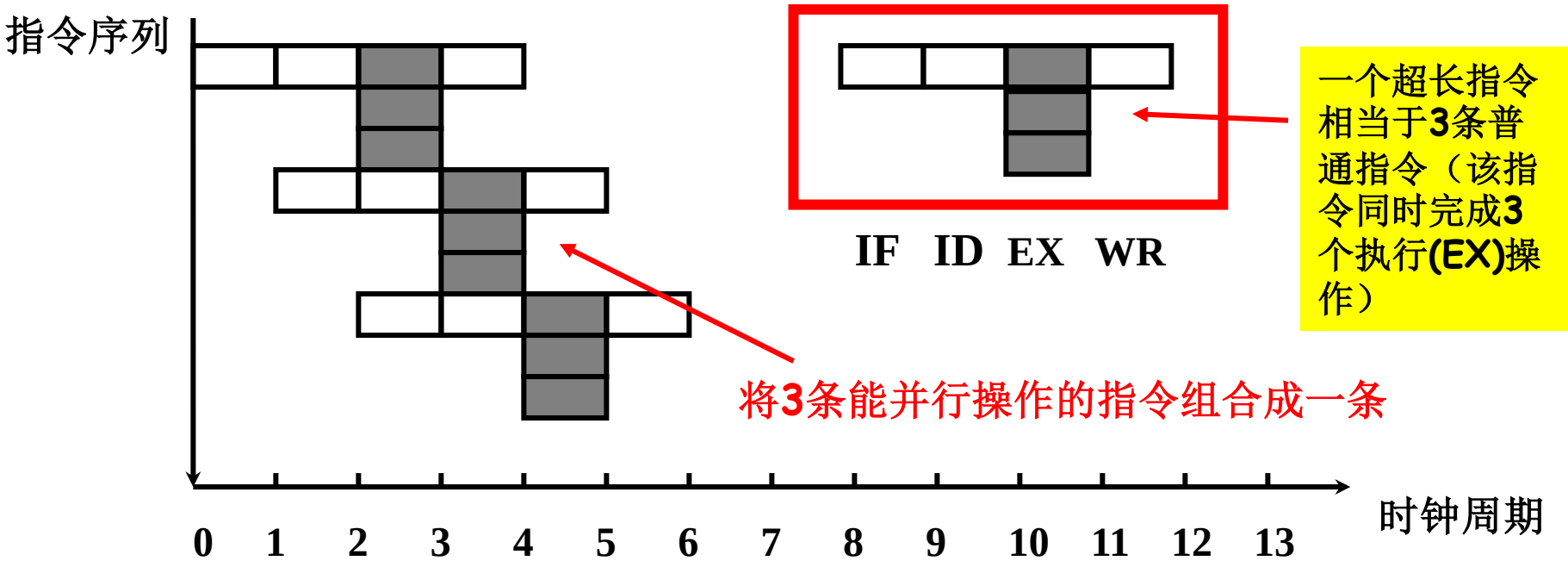
Very Long Instruction Word

- 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



七、流水线结构

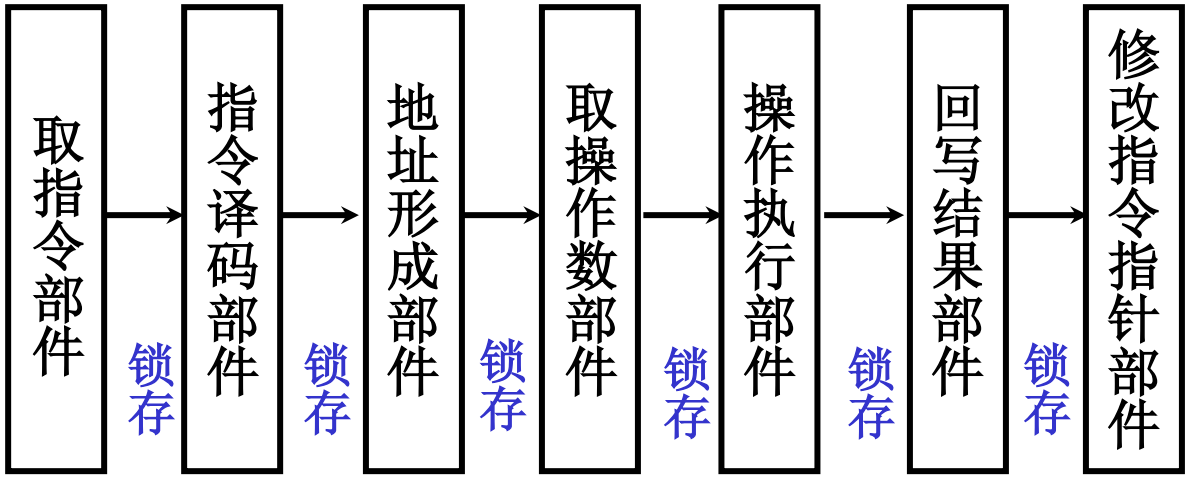
FI（Fetch Instruction）：取指　　DI（Decode Instruction）：指令译码
CO（Calculate Operands）：计算操作数地址　　FO（Fetch Operands）：取操作数
EI（Execute Instruction）：执行指令　　WO（Write Operands）：写操作数

IF：取指令
ID：指令译码/读寄存器
EX：执行/计算访存有效地址
MEM：存储器访问
WB：结果写回寄存器

取指(IF)
译码(ID)
执行(EX)
写回(WR)

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

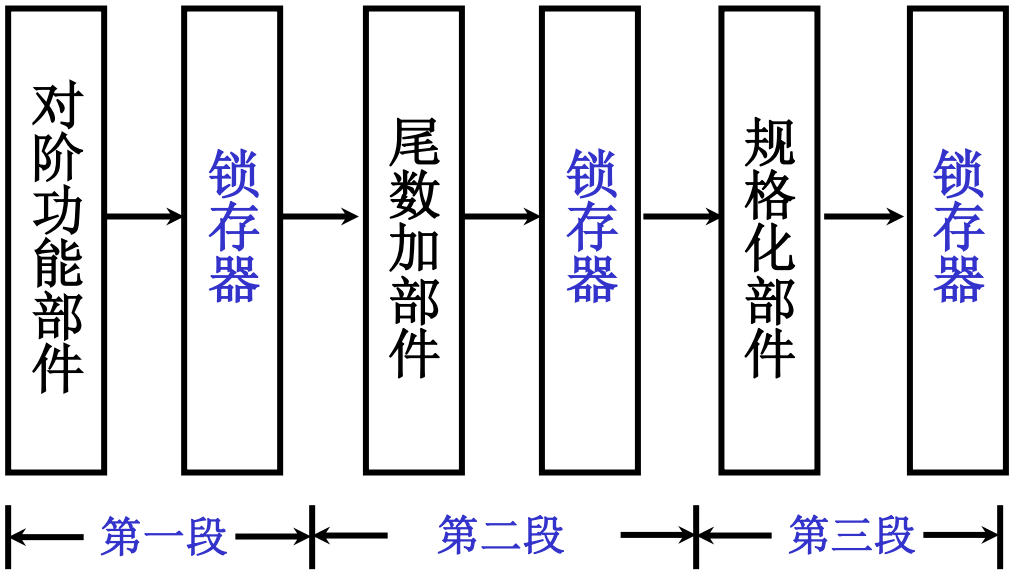
理想情况下，7 级流水 的速度是不采用流水技术的 7 倍

2. 运算流水线

1、对阶
2、尾数求和
3、规格化
4、舍入
5、溢出判断

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

5.5小节“程序中断方式”
主要是讲I/O的中断

8.4 中断系统

这里则是讲CPU的中断

一、概述

二、中断请求标记和中断判优逻辑

三、中断服务程序入口地址的寻找

四、中断响应

五、保护现场和恢复现场

六、中断屏蔽技术

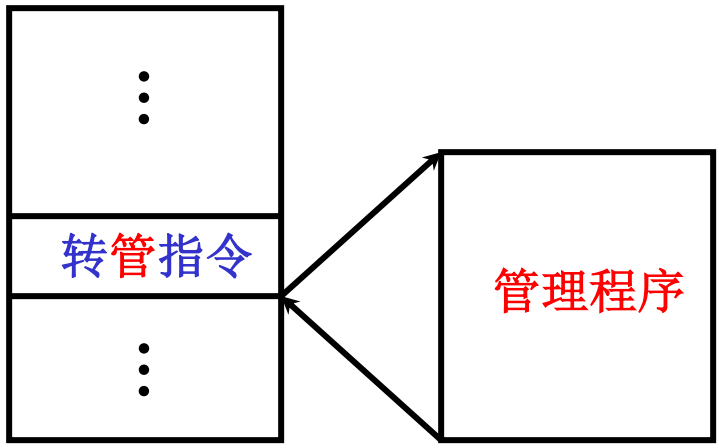
一、概述

1. 引起中断的各种因素

(1) 人为设置的中断

如　转管指令

INT　n　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？（优先级排队）

(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？（中断服务程序入口地址）

(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？(中断嵌套)

有些问题通过硬件解决，有些问题通过软件解决

二、中断请求标记和中断判优逻辑

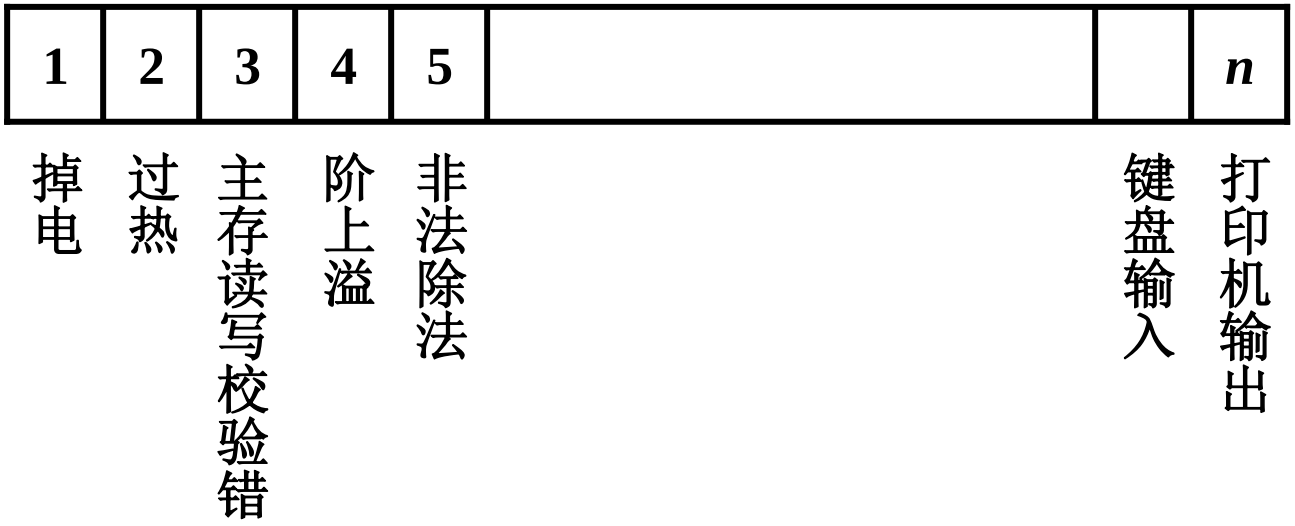
触发器是1位；

寄存器是n位、由多个触发器构成

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



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

参见 第五章 图5.38

(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

② 集中 在 CPU 内

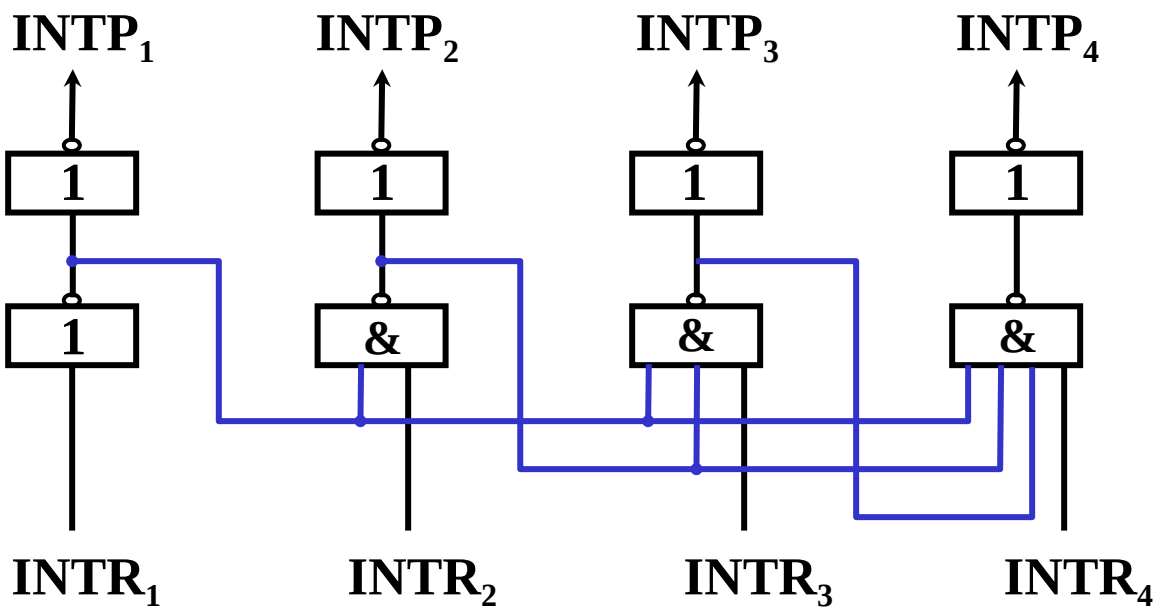


图8.25

$INTR_1$、$INTR_2$、$INTR_3$、$INTR_4$ 优先级 按 降序 排列（$INTR_1$最高）

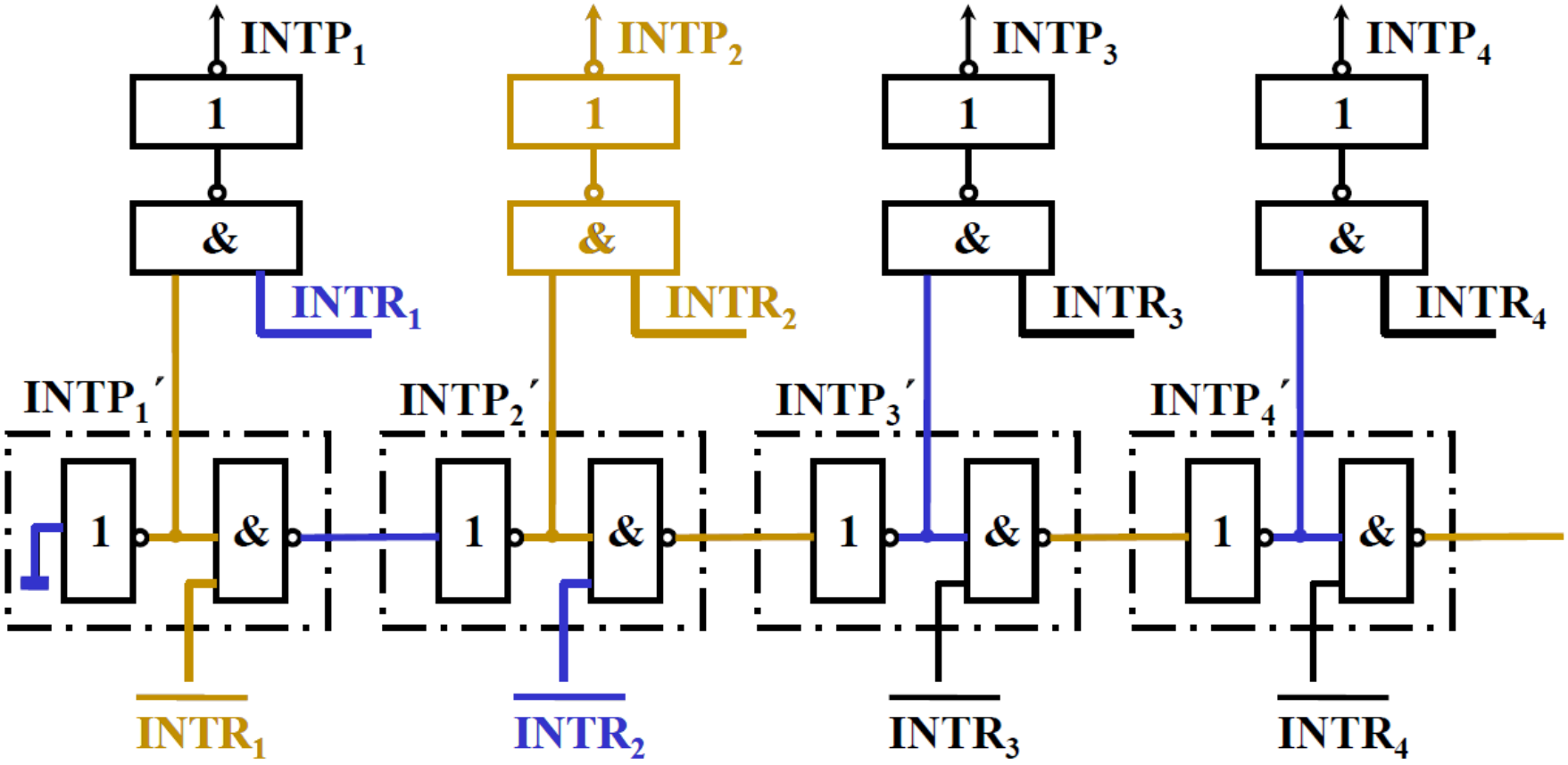
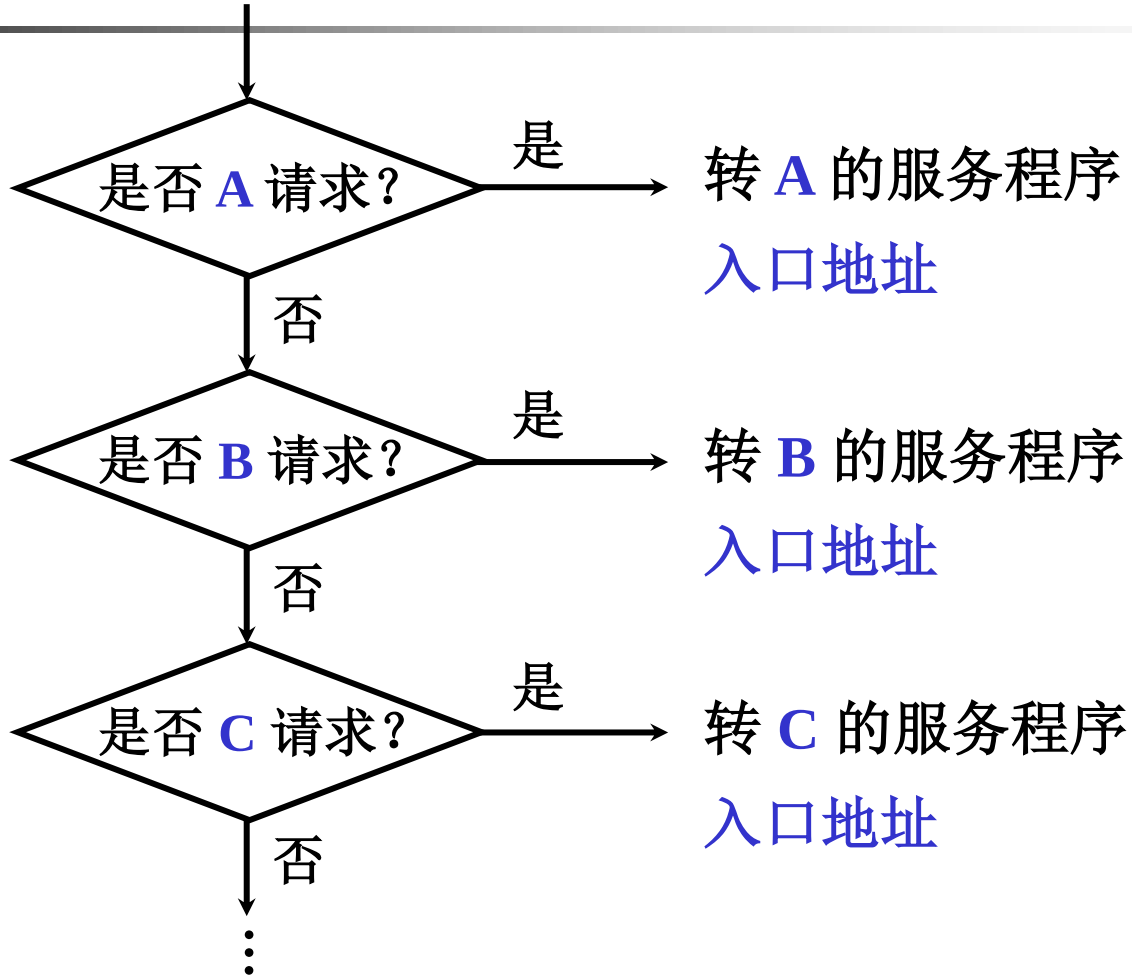

INTP1
INTP2
INTP3
INTP4
1
&
INTR1
INTR2
INTR3
INTR4
INTP1′
INTP2′
INTP3′
INTP4′

图5.38　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（A最高）



三、中断服务程序入口地址的寻找(2种方法)

1. 硬件向量法 (图5.39、图5.40)

12 H 13 H 14H

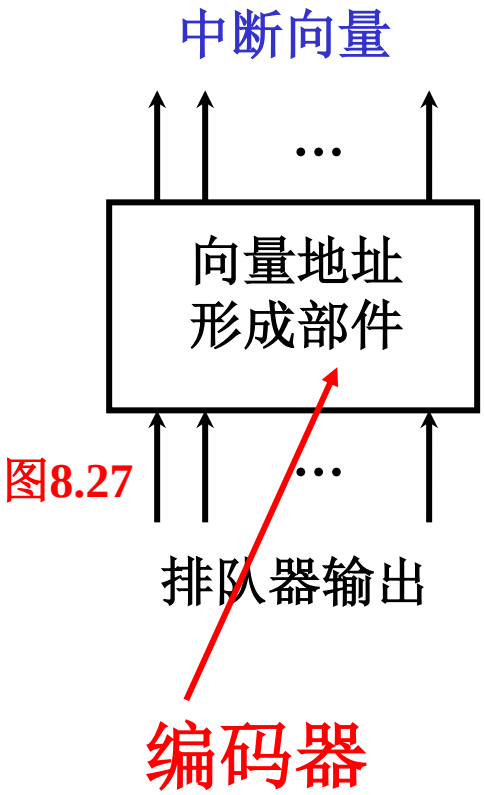
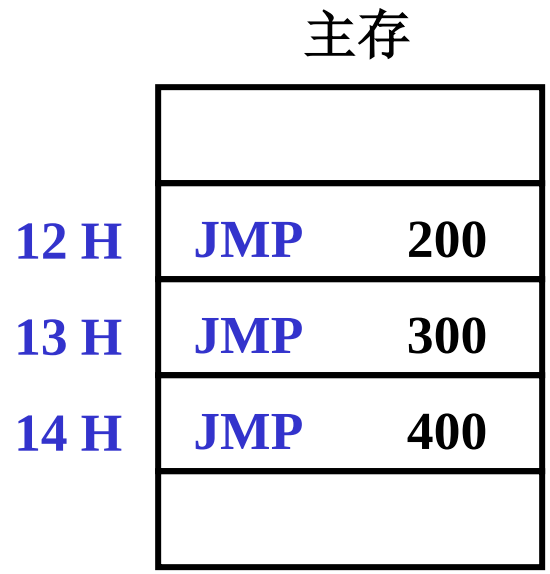


图8.27



存放一条无条件转移指令

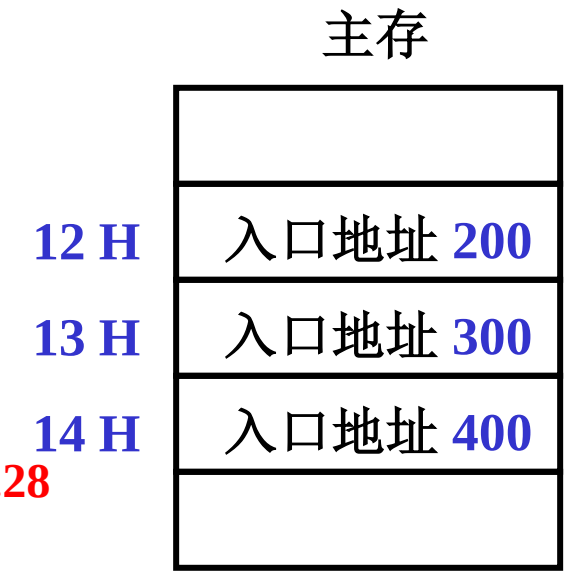


图8.28

向量地址表

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

四、中断响应

开中断指令使EINT=1

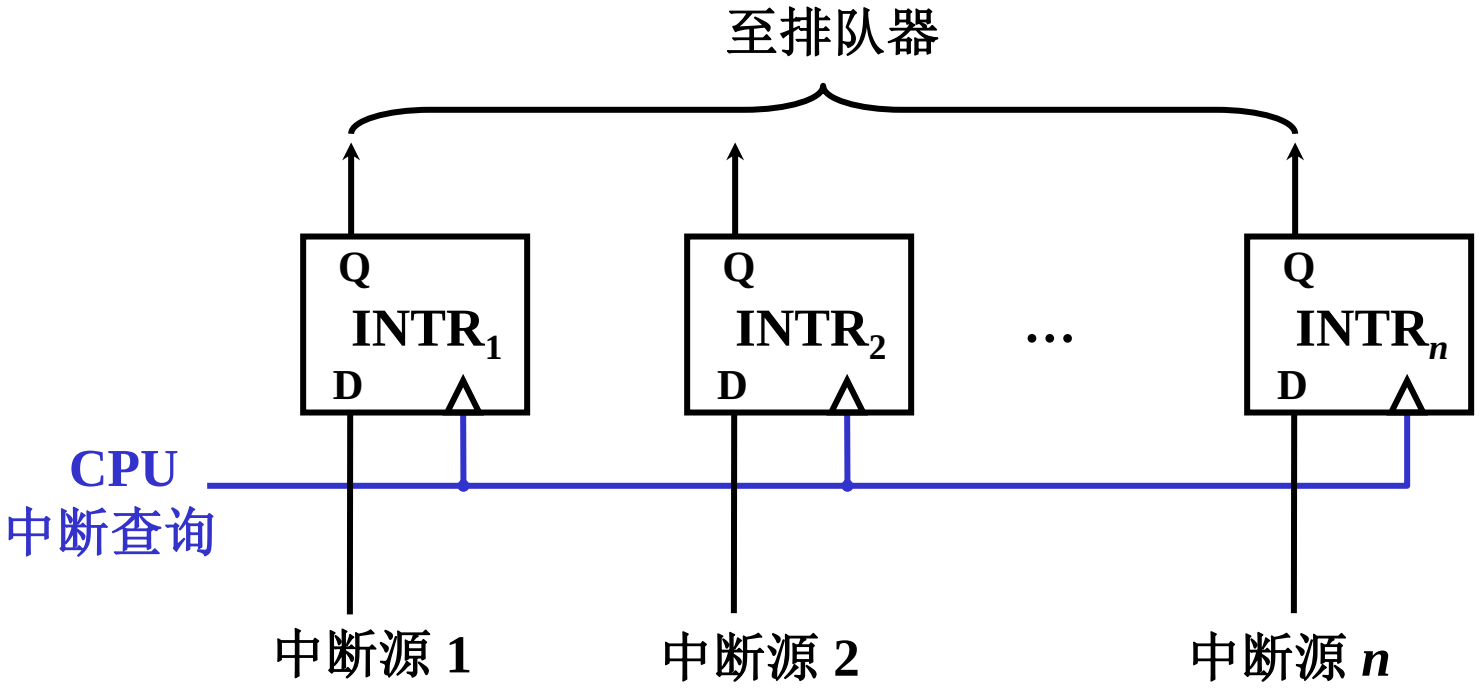
关中断指令使EINT=0

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(中断周期CPU自动完成的操作)

(1) 保护程序断点 （将当前的PC保存到存储器中，可以有2个地方）

断点存于 特定地址（ 0 号地址） 内　　或者断点进栈

(2) 寻找中断服务程序入口地址（2种方法）

向量地址 ⟶ PC　　执行转移指令 （硬件向量法）

中断识别程序 入口地址 M ⟶ PC　执行中断识别程序（软件查询法）

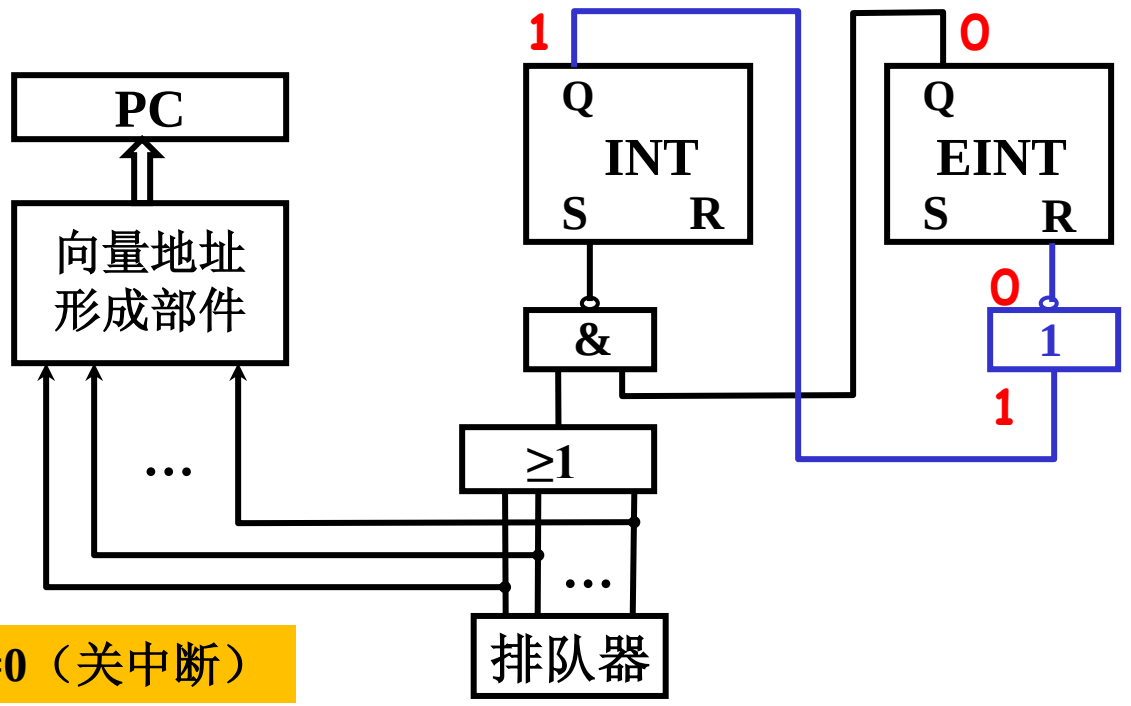
(3) 硬件 关中断

(自动关中断)

INT 中断标记触发器

EINT 允许中断触发器

R – S 触发器



INT=1 自动 使EINT=0（关中断）

五、保护现场和恢复现场

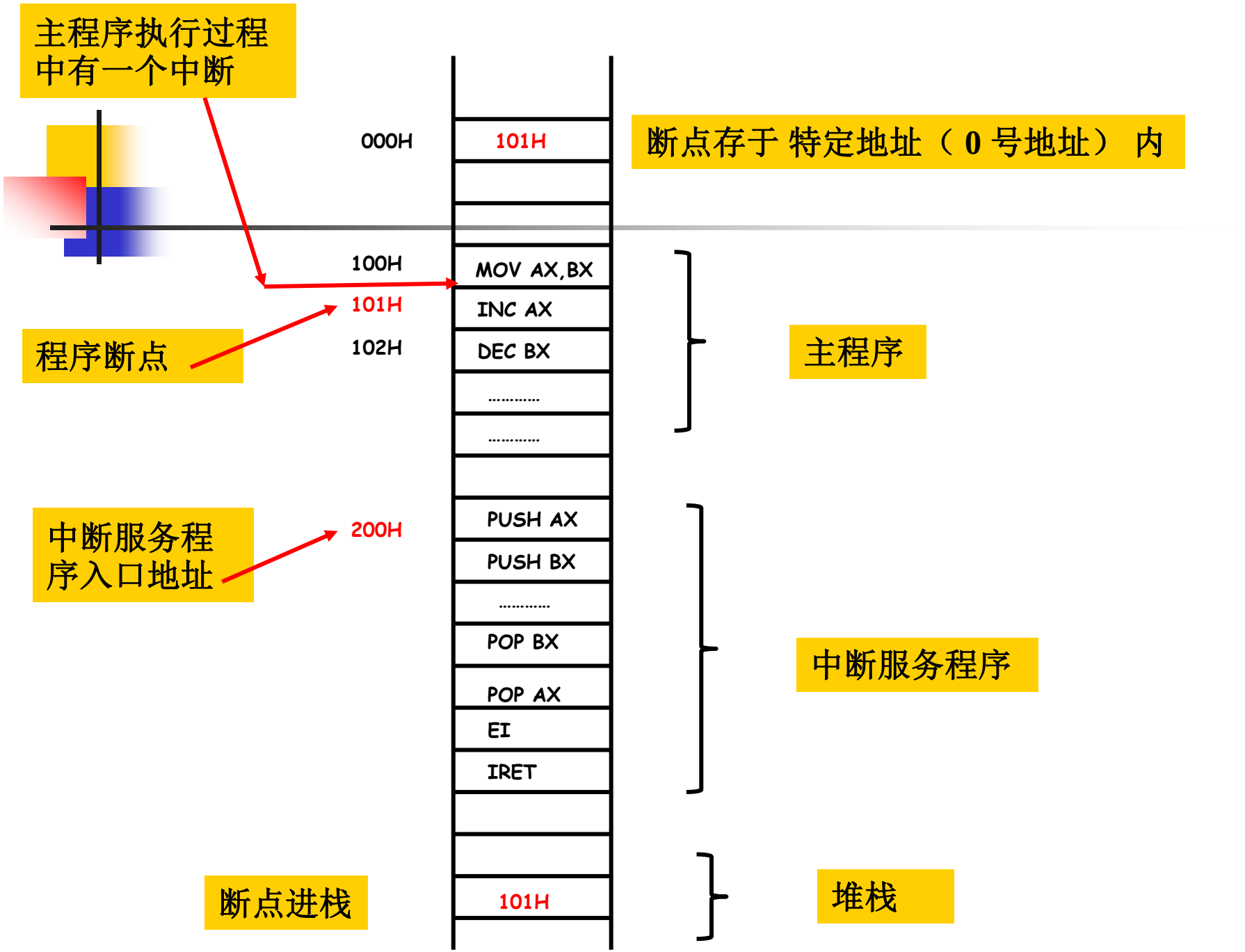
1. 保护现场

- 断点（PC）　　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

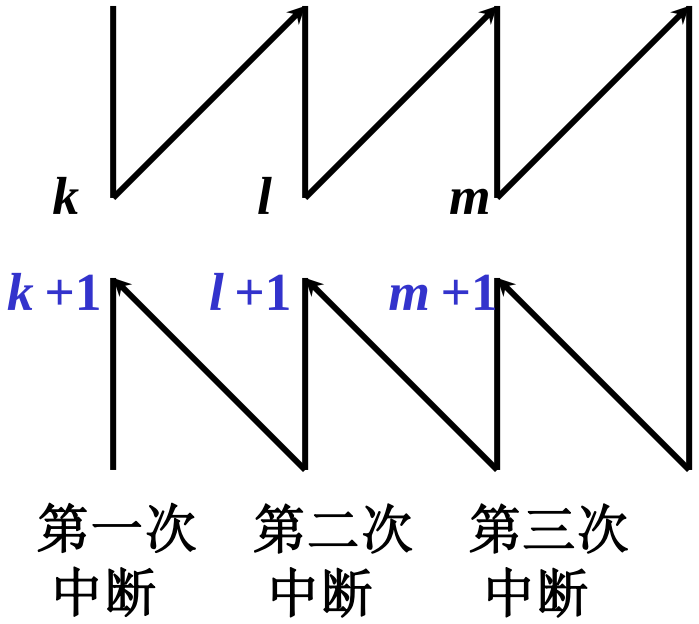
中断服务程序

- 保护现场　　PUSH　　保护寄存器的内容
- 其它服务程序　　视不同请求源而定
- 恢复现场　　POP　　恢复寄存器的内容
  - 开中断指令EI
- 中断返回　　IRET


主程序执行过程
中有一个中断
000H
101H
断点存于 特定地址（ 0 号地址） 内
100H
MOV AX,BX
101H
INC AX
程序断点
102H
DEC BX
.....
.....
主程序
中断服务程
序入口地址
200H
PUSH AX
PUSH BX
.....
POP BX
POP AX
EI
IRET
中断服务程序
断点进栈
101H
堆栈

六、中断屏蔽技术

1. 多重中断的概念



中断嵌套

程序断点 $k+1$ ， $l+1$ ， $m+1$

2. 实现多重中断的条件

(1) 提前 设置 开中断 指令（图5.43的右图）

(2) 优先级别高 的中断源 有权中断优先级别低 的中断源

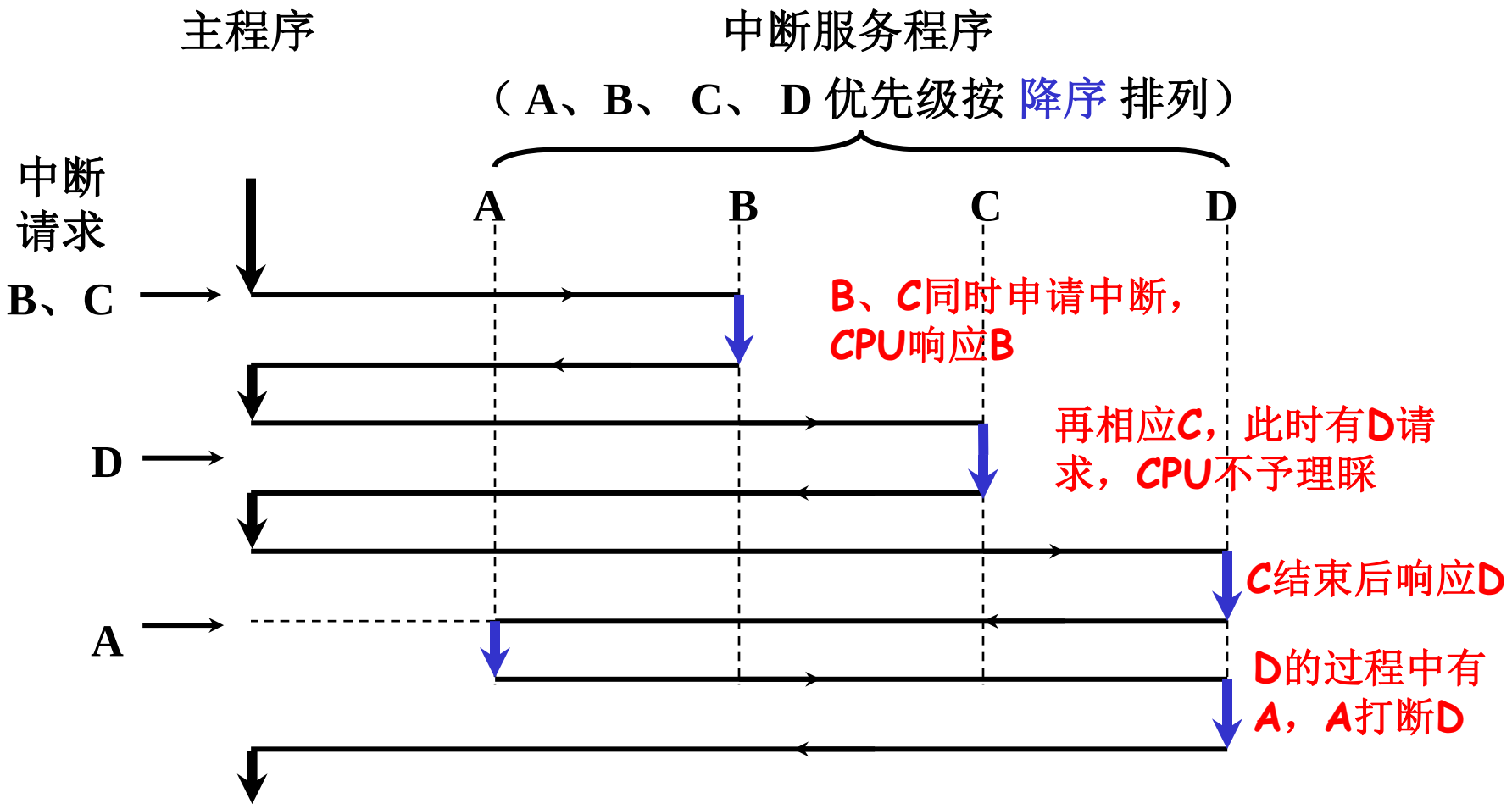
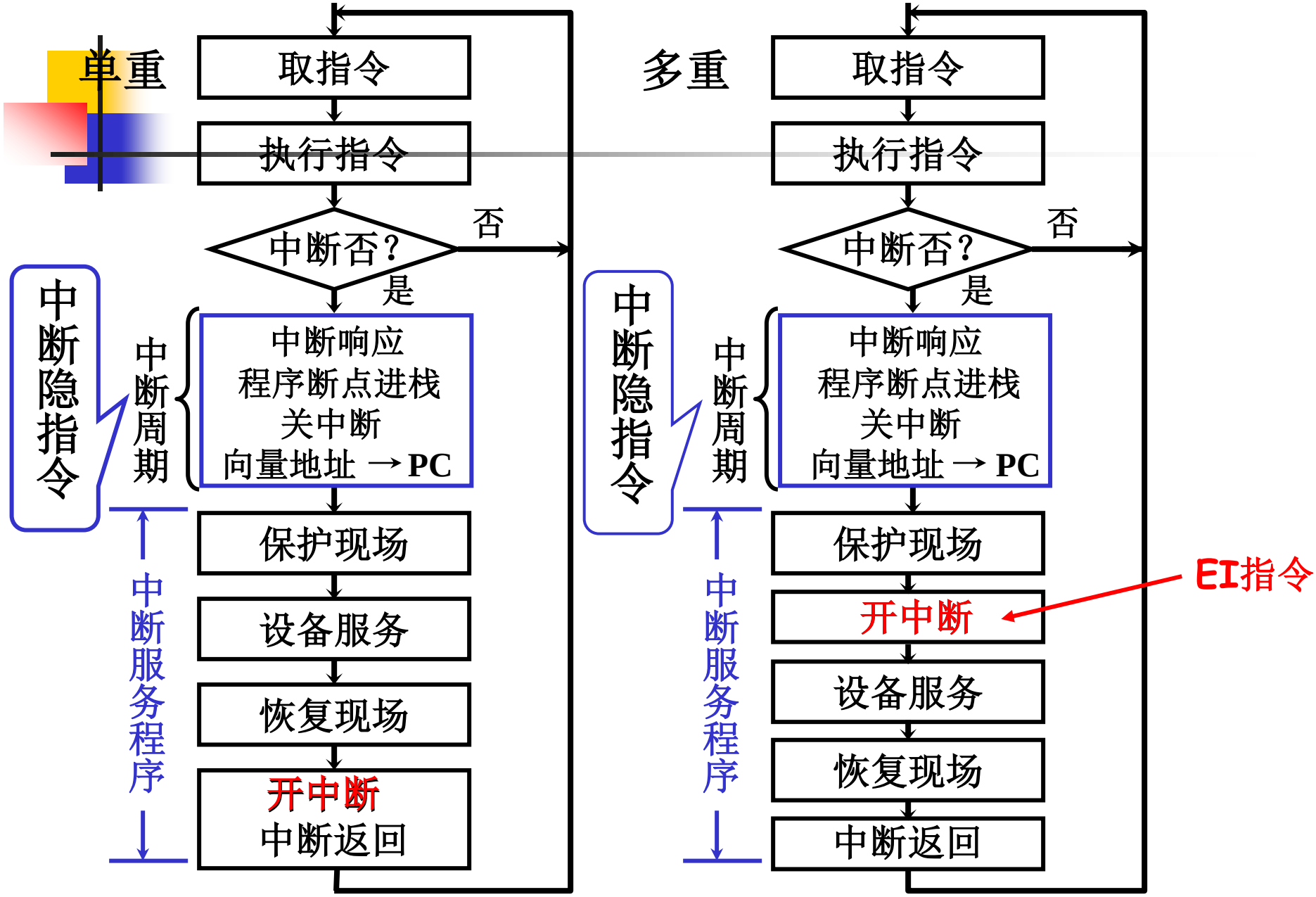


图5.43


单重
取指令
执行指令
中断否？
否
是
中断隐指令
中断周期
中断响应
程序断点进栈
关中断
向量地址 → PC
保护现场
设备服务
恢复现场
开中断
中断返回
中断服务程序
多重
取指令
执行指令
中断否？
否
是
中断隐指令
中断周期
中断响应
程序断点进栈
关中断
向量地址 → PC
保护现场
开中断
设备服务
恢复现场
中断返回
中断服务程序
EI指令

3. 屏蔽技术

(1) 屏蔽触发器的作用

在图8.25排队器的基础上，增加屏蔽功能

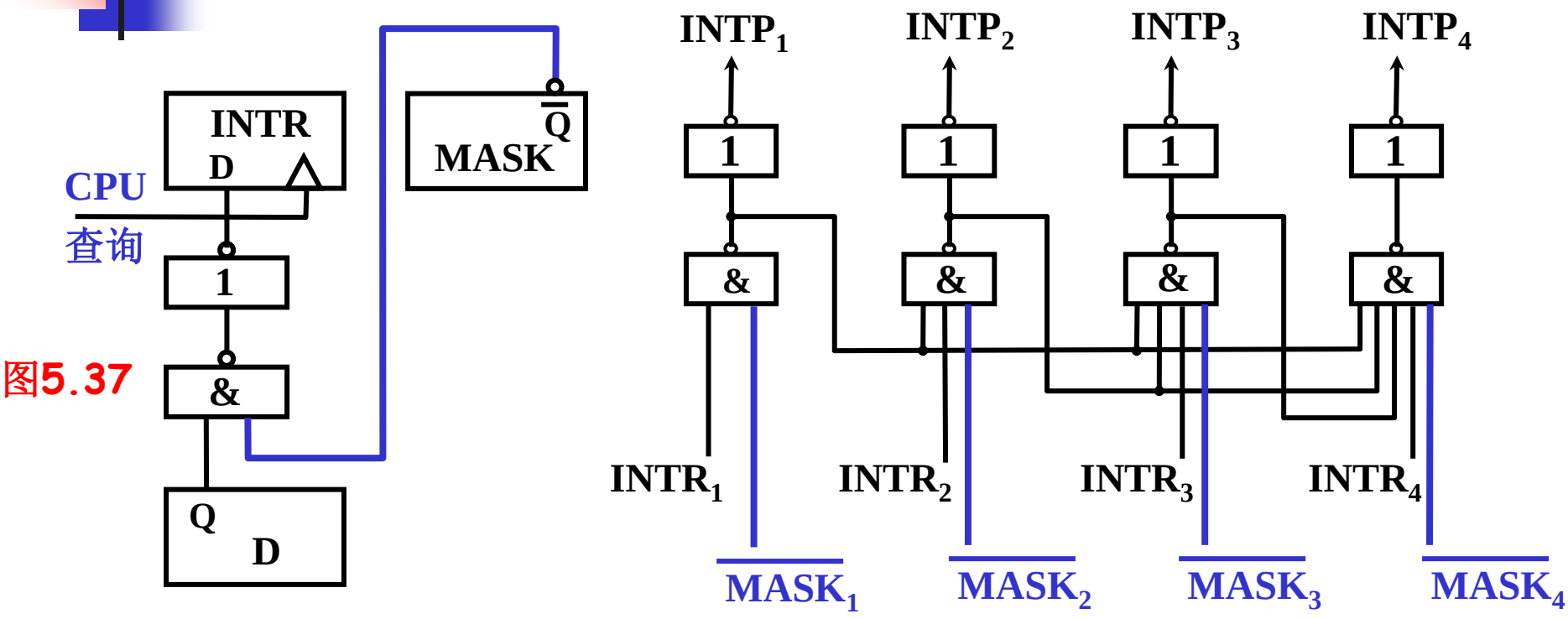


图5.37

MASK = 0（未屏蔽）　　$MASK_i$ = 1 （屏蔽）

INTR　能被置 “1”　　$INTP_i$ = 0 （不能被排队选中）

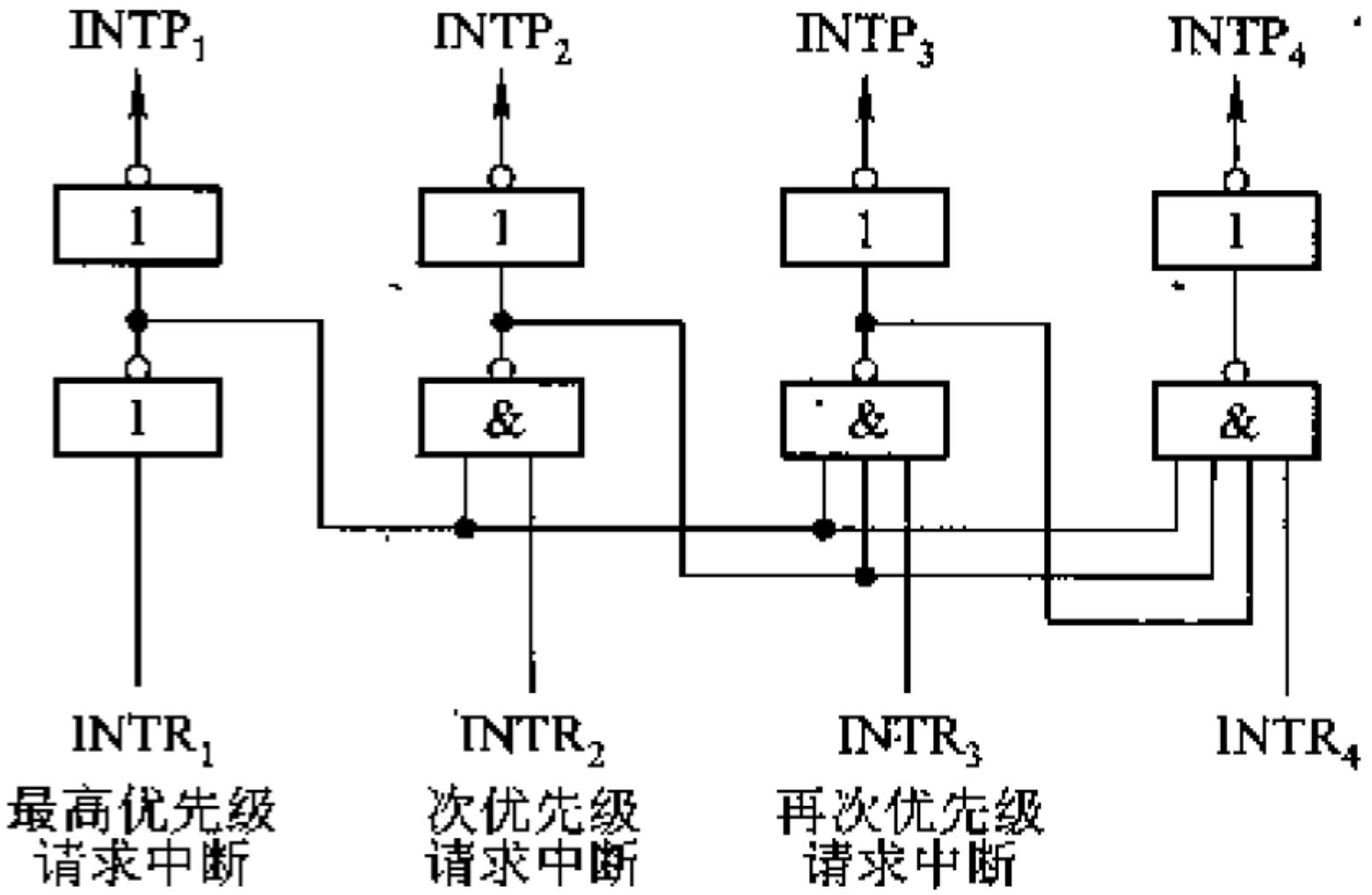


图 8.25　集中在 CPU 内的排队器

排队器

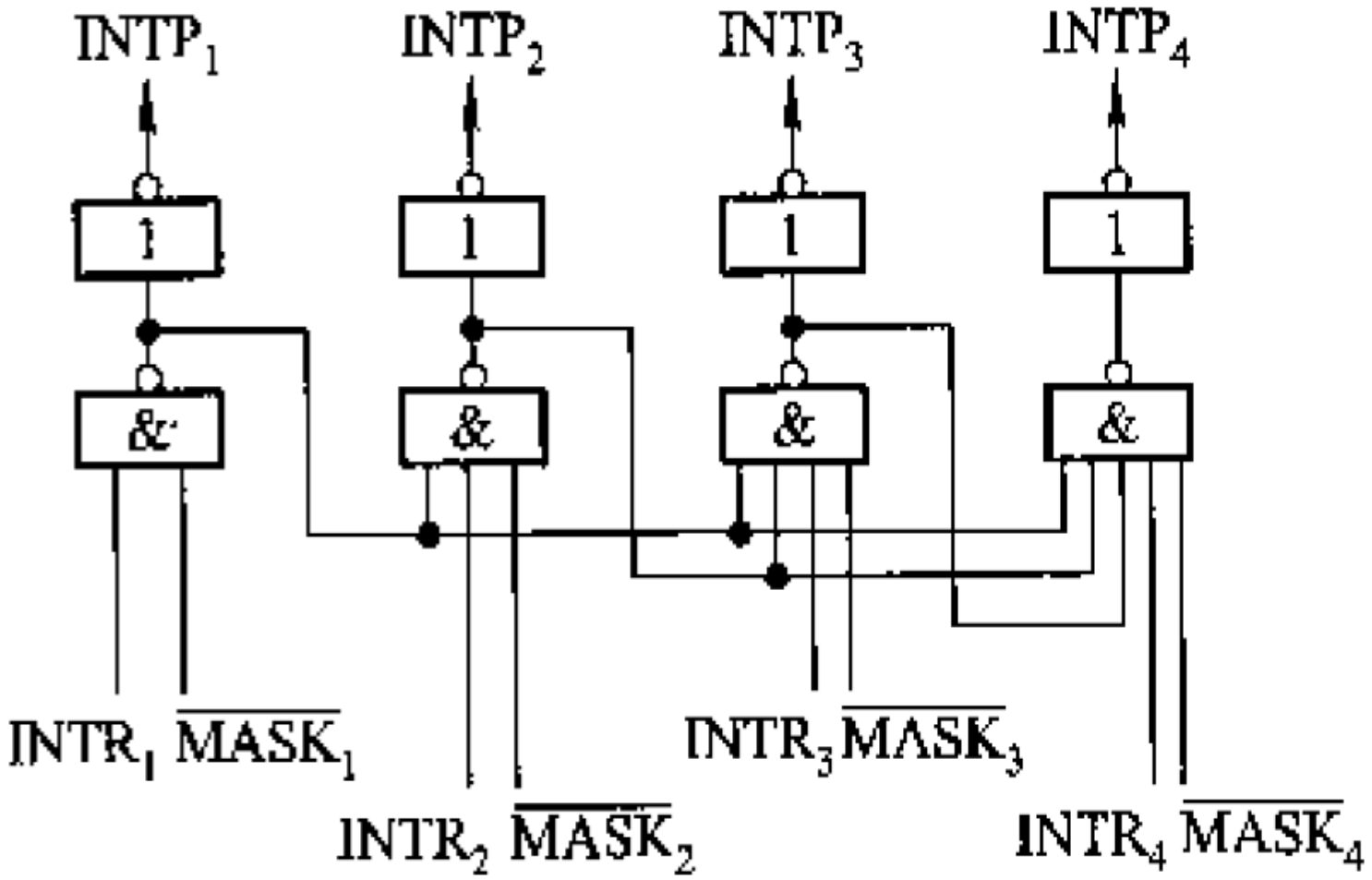


图 8.33　具有屏蔽功能的排队器

增加屏蔽功能

(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

(2) 屏蔽字

1级中断源的请求已被CPU响应，若在中断服务程序中设置一个全“1”的屏蔽字，则就不会再响应任何中断源

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

(2) 屏蔽字

2级中断源的请求已被CPU响应，若在中断服务程序中设置一个“011…1”的屏蔽字，则只有1级中断源可以中断2级中断源

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

(2) 屏蔽字

6级中断源的请求已被CPU响应，若在中断服务程序中设置一个“0000011111111111”的屏蔽字，则只有1-5级的中断源可以中断6级中断源

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

响应优先级　　不可改变

响应优先级是指CPU响应各中断源请求的优先次序（由排队器电路确定，不可改变）

处理优先级　　可改变（通过重新设置屏蔽字）

处理优先级是指CPU实际对各中断源请求的处理优先次序

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

① A不能被其他中断

④ B可以被A、C、D中断

③ C可以被A、D中断

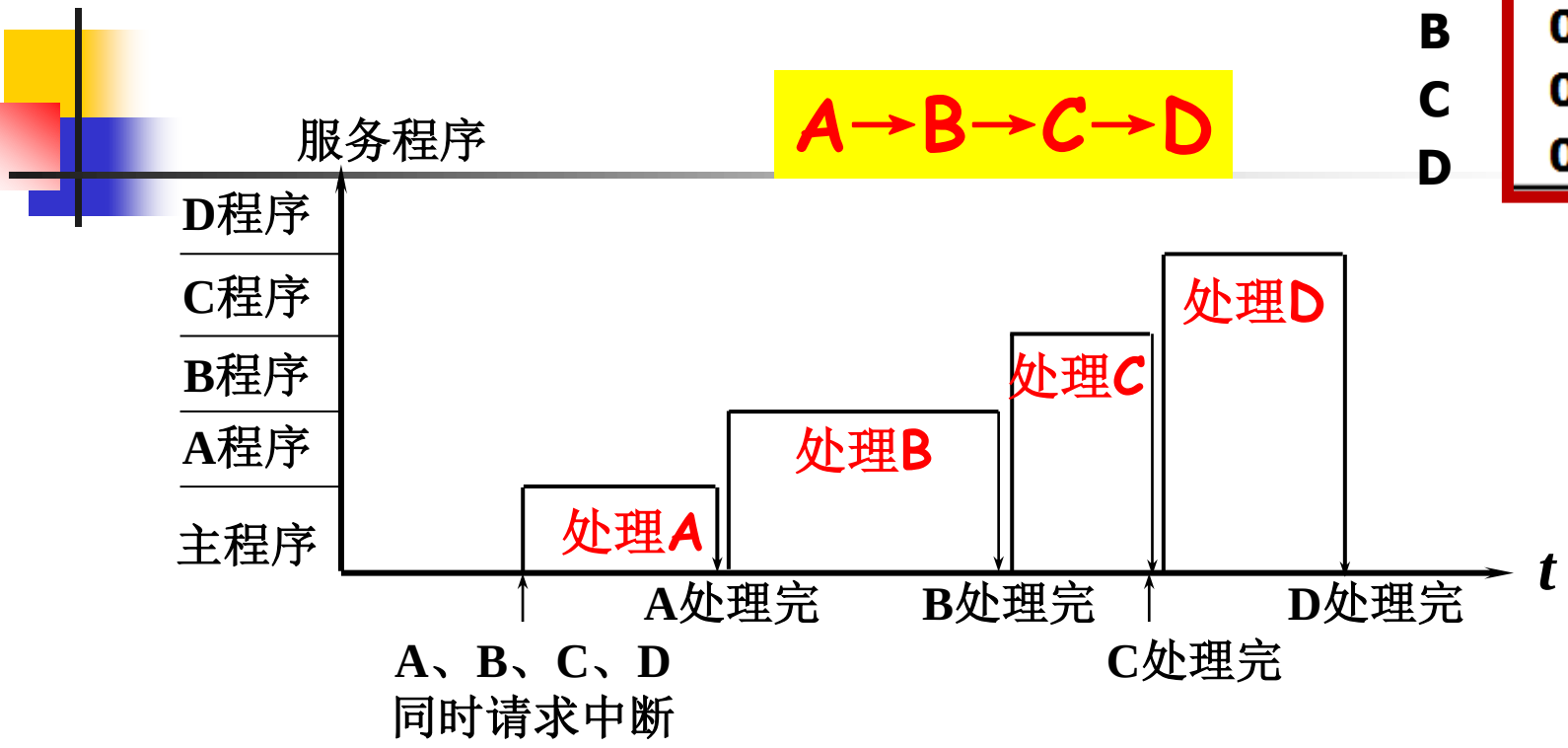
② D可以被A中断

本来　响应优先级 A→B→C→D 降序排列

现在　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（续）

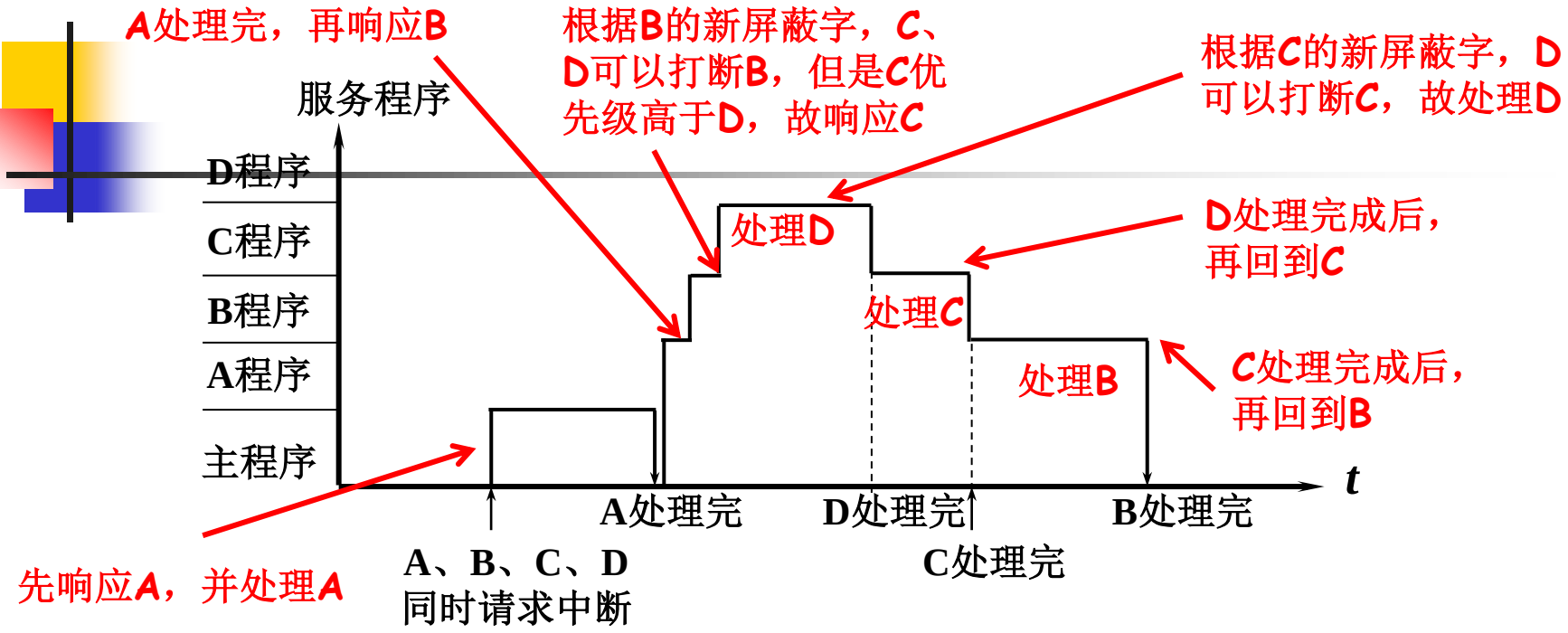
	原屏蔽字
A	1 1 1 1
B	0 1 1 1
C	0 0 1 1
D	0 0 0 1



CPU 执行程序轨迹（原屏蔽字）

A→D→C→B

(3) 屏蔽技术可改变处理优先等级(续)



CPU 执行程序轨迹（新屏蔽字）

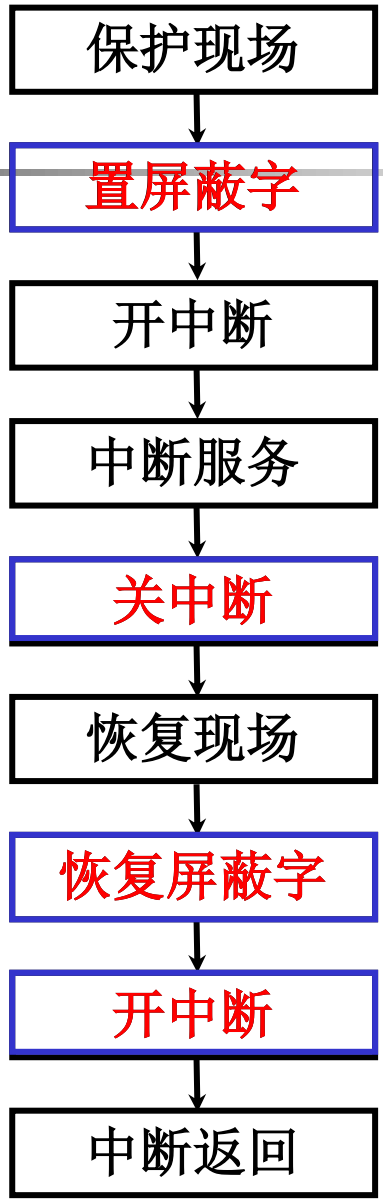
(4) 屏蔽技术的其他作用

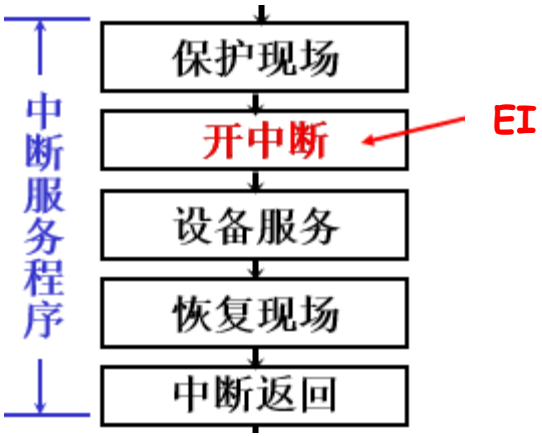
可以 人为地屏蔽 某个中断源的请求

便于程序控制

	新屏蔽字
A	1 1 1 1
B	0 1 0 0
C	0 1 1 0
D	0 1 1 1

(5) 新屏蔽字的设置





没有屏蔽字的中断服务程序

- 例8.2：设某计算机有4个中断源，其硬件排队器按照1->2->3->4降序排序，各中断服务程序所对应的屏蔽字如表8.9所示。

  (1)给出上述4个中断源的中断处理次序

  (2)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)4个中断源的处理次序为：3->1->4->2

  (2)如图8.37所示

表 8.9　例 8.2 各中断源对应的屏蔽字

中断源		屏蔽字 1	2	3	4
1	No.2	1	1	0	1
2	No.4	0	1	0	0
3	No.1	1	1	1	1
4	No.3	0	1	0	1

3 -> 1 -> 4 -> 2

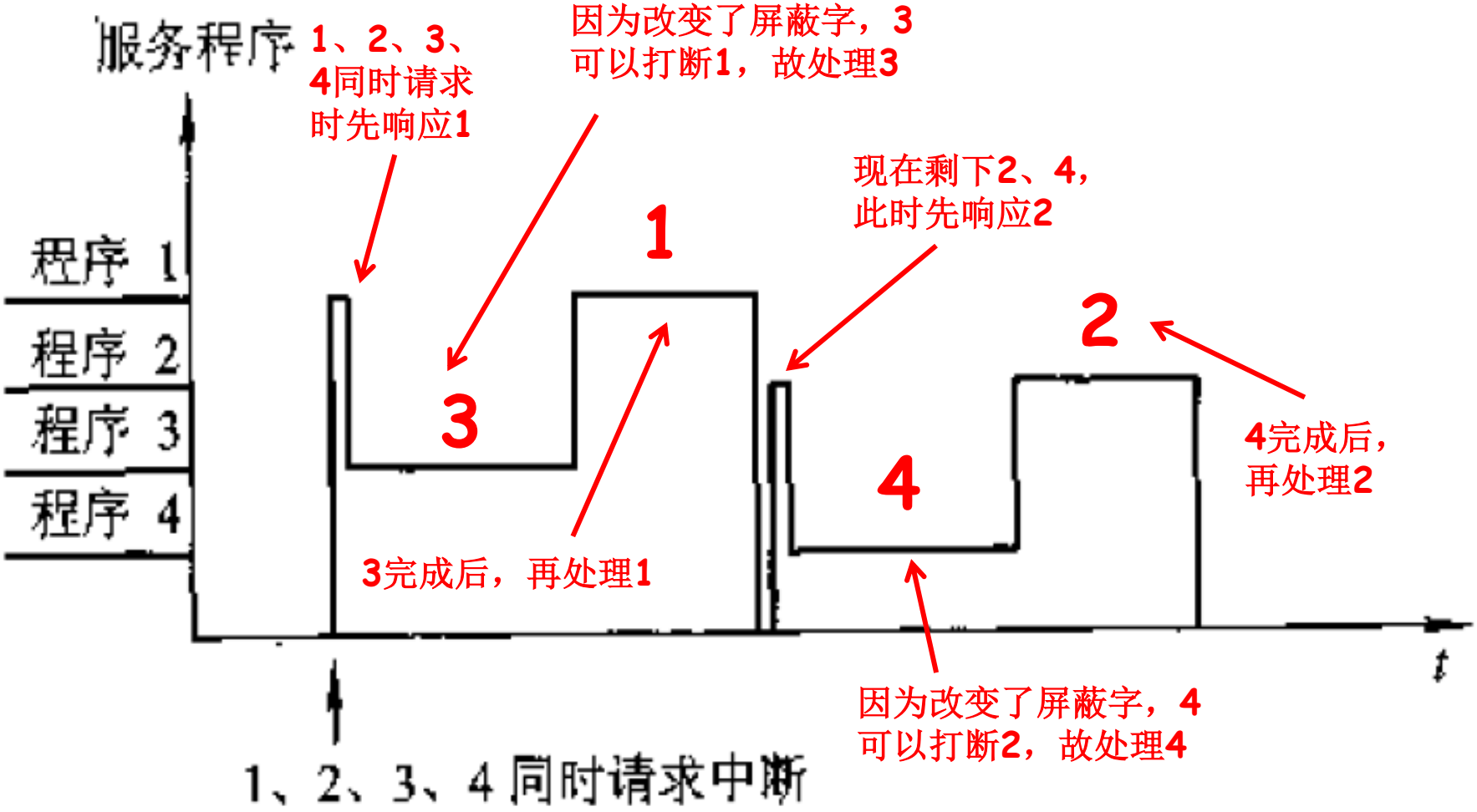


图 8.37　例 8.2 CPU 执行程序的轨迹

1 -> 2 -> 3 -> 4

3 -> 1 -> 4 -> 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 ⟶ MAR

命令存储器写

PC ⟶ MDR　　即：断点 ⟶ MDR

(MDR) ⟶ 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？

在中断服务程序中的开中断指令之前，必须先将0地址单元的内容转存到其他地址单元

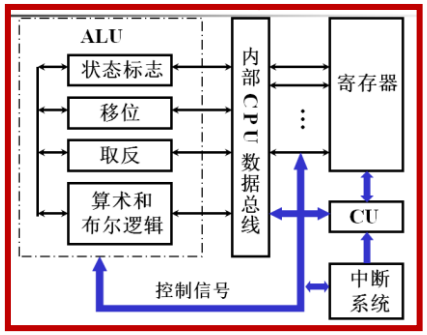
(3) 程序断点存入 “ 0 ” 地址的断点保护

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
	⋮	
	LDA　0	} 0 地址内容转存
	STA　RETURN	} 0 地址内容转存
置屏蔽字 →	ENI	开中断
	⋮	} 其他服务内容
	LDA　SAVE	恢复现场
	JMP　@　RETURN	间址返回
SAVE	××××	存放 ACC 内容
RETURN	××××	转存 0 地址内容

在中断服务程序中的开中断指令之前，必须先将0地址单元的内容转存到其他地址单元

本章小结

- **CPU的功能**
  - 控制器的功能
    - 取指令、分析指令、执行指令
  - 运算器的功能
- **CPU的内部结构**



- **CPU的寄存器**
  - 用户可见的寄存器
  - 控制和状态寄存器（用户不可见的寄存器）
- 控制器的设计方法
  - 组合逻辑设计（硬布线控制器）
  - 微程序设计（微程序控制器）

本章小结

- 指令周期
  - 取指周期
  - 间址周期
  - 执行周期
  - 中断周期

- 指令周期的数据流
  - 取指周期的数据流：6步
  - 间址周期的数据流：4步
  - 执行周期的数据流：不同指令的执行周期数据流不一样
  - 中断周期的数据流：6步

本章小结

- 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
- 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
- 影响指令流水线性能的因素
  - 结构相关：资源相关
  - 数据相关：
    - 写后读相关（RAW）　　先写后读
    - 读后写相关（WAR）　　先读后写
    - 写后写相关（WAW）　　写后再写
  - 控制相关：转移指令引起的

本章小结

- 流水线的性能
  - 吞吐率(Throughput Rate)
  - 加速比 S_p (Speedup Ratio)
  - 效率(Efficient)
- 流水线的多发技术
  - 超标量技术（Superscalar）
  - 超流水线技术（Superpiplined）
  - 超长指令字技术（VLIW）

本章小结

- **中断系统需解决的问题**（有些是通过硬件解决，有些是通过软件解决）
  - 各中断源如何向CPU提出请求？
  - 各中断源同时提出请求怎么办？（优先级排队）
  - CPU什么条件、什么 时间、以什么 方式响应中断？
  - 如何保护现场？
  - 如何寻找入口地址？（中断服务程序入口地址）
  - 如何恢复现场？如何返回？
  - 处理中断的过程中又 出现新的中断 怎么办？(中断嵌套)

本章小结

- 中断请求标记：INTR
- 中断判优逻辑
  - 硬件实现：链式排队器
  - 软件实现

图5.38：在I/O接口中
图8.25：在CPU中

- 中断服务程序入口地址的寻找
  - 硬件向量法
  - 软件查询法

图5.39：在I/O接口中
图8.27：在CPU中

本章小结

- 中断响应
  - 响应中断的 条件
  - 响应中断的 时间
  - 中断隐指令(中断周期CPU自动完成的操作)
    - 保护程序断点
    - 寻找中断服务程序入口地址
    - 硬件关中断 (自动关中断)

保护到地址为0的存储器中

保护到堆栈中

- 保护现场和恢复现场

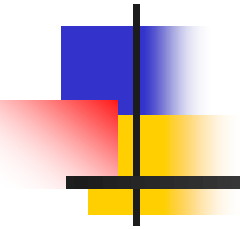
- 中断屏蔽技术：多重中断
  - 屏蔽字
  - 多重中断的断点保护

第14次作业——习题（P370-372）

- 8.1
- 8.2
- 8.3
- 8.4
- 8.7
- 8.8
- 8.10
- 8.15
- 8.17
- 8.18
- 8.21
- 8.24
- 8.25
- 8.26

关于作业的提交

- **1周内**必须提交（上传到学院的**FTP**服务器上），否则认为是迟交作业；如果期末仍然没有提交，则认为是未提交作业
  - 作业完成情况成绩=第**1**次作业提交情况*第**1**次作业评分+第**2**次作业提交情况*第**2**次作业评分+……+第**N**次作业提交情况*第**N**次作业评分
  - 作业评分：**A**（好）、**B**（中）、**C**（差）三挡
  - 作业提交情况：按时提交（**1.0**）、迟交（**0.5**）、未提交（**0.0**）
- 请采用电子版的格式（**Word**文档）上传到**FTP**服务器上，文件名取“学号+姓名+第**X**次作业**.doc**”
  - 例如：**1192019220364 2**+袁佳哲+第**14**次作业**.doc**
- 第**14**次作业提交的截止日期为：**2021年6月4日晚上24点**



The End

Thanks