



## 计算机组成原理 第八章 CPU 的结构和功能



系统结构研究所

(一) **CPU**的功能和基本结构

(二) 指令执行过程

(三) 数据通路的功能和基本结构

(四) 控制器的功能和工作原理

1. 硬布线控制器

2. 微程序控制器

微程序、微指令和微命令；微指令格式；微命令的编码方式；微地址的形成方式

(五) 指令流水线

1. 指令流水线的基本概念

2. 指令流水线的基本实现

3. 超标量和动态流水线的基本概念

# Contents

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

指令控制

分析指令

操作控制

执行指令，发出各种操作命令

时间控制

控制程序输入及结果的输出

总线管理

处理中断

处理异常情况和特殊请求

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

8.1

### 1. CPU 与系统总线

指令控制

PC IR

操作控制

时间控制



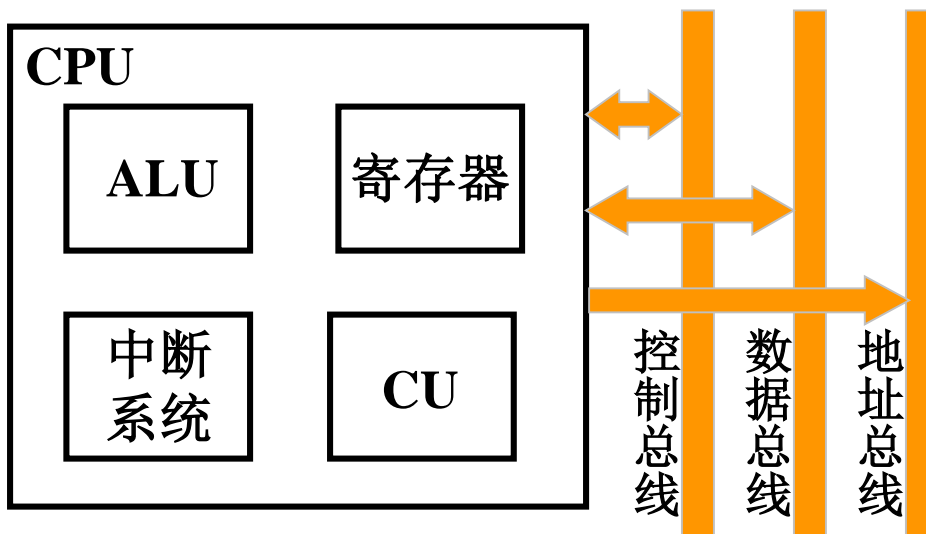
CU 时序电路

数据加工

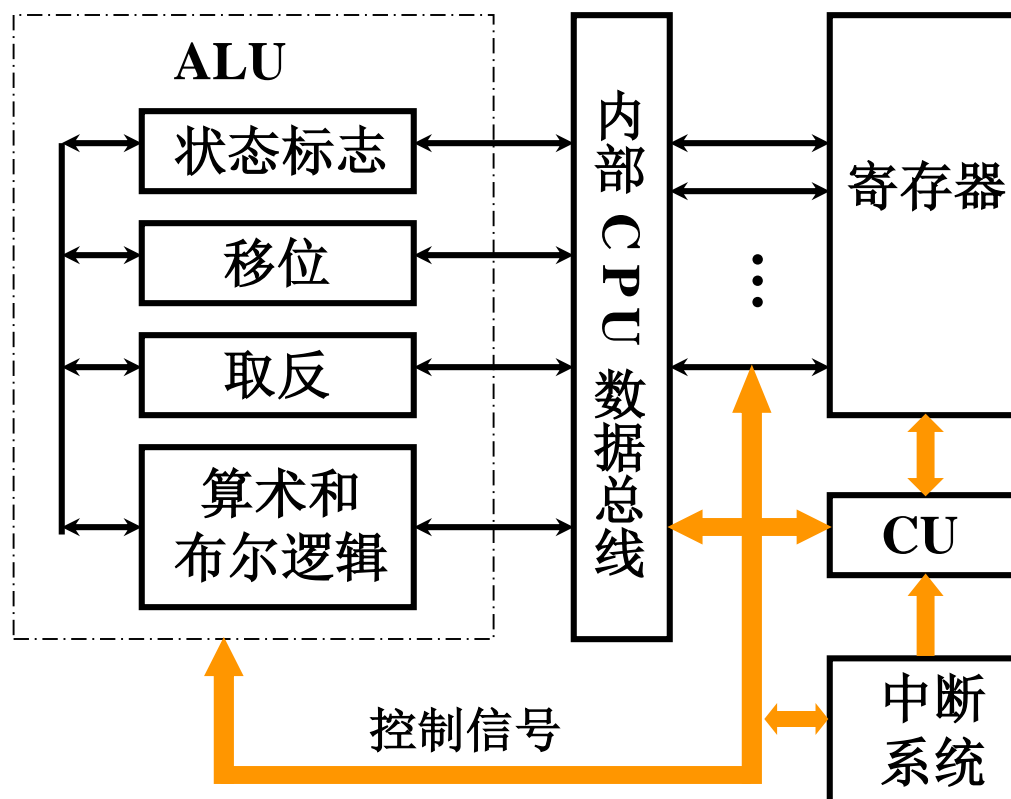
ALU 寄存器

处理中断

中断系统



## 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

### (1) 控制寄存器

$PC \rightarrow MAR \rightarrow M \rightarrow MDR \rightarrow IR$

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



### 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

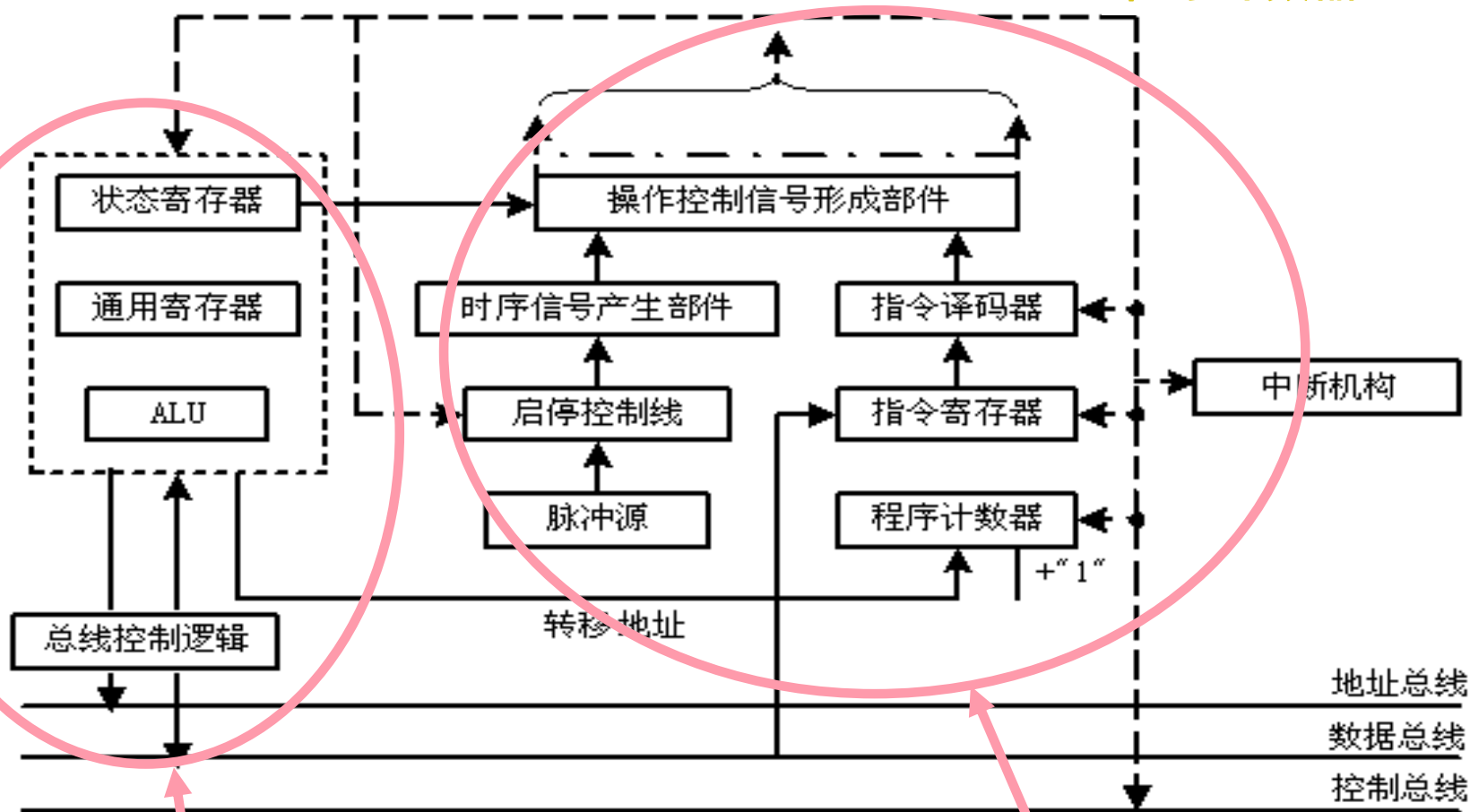
参见 8.4 节

## 五、ALU

参见 第 6 章

# CPU基本组成原理图

指令寄存器---IR  
程序计数器---PC



执行部件

控制部件

CPU 由 执行部件 和 控制部件 组成  
CPU 包含 数据通路 和 控制器

控制器 由 指令译码器 和 控制信号形成部件 组成



- PC、IR
- 指令译码器ID：对操作进行译码，向控制器提供操作的特定信号。
- 时序发生器：产生各种时序信号
  - 脉冲源：通常由石英晶体振荡器构成。产生一定频率的脉冲信号作为整个机器的时钟脉冲，是机器周期和工作脉冲的基准信号。在机器刚加电时，产生总清信号(reset)
  - 启停线路：保证可靠地送出或封锁完整的时钟脉冲，控制时序信号的发生或停止，从而启动机器工作或停机。



- 时序控制信号形成部件：当机器启动后，在CLK时钟作用下，根据当前正在执行的指令的需要，产生相应的时序控制信号，并根据被控功能部件的反馈信号调整时序控制信号。
- 状态寄存器（PSR）：存放PSW,PSW表明了系统基本状态，是控制程序执行的重要依据。

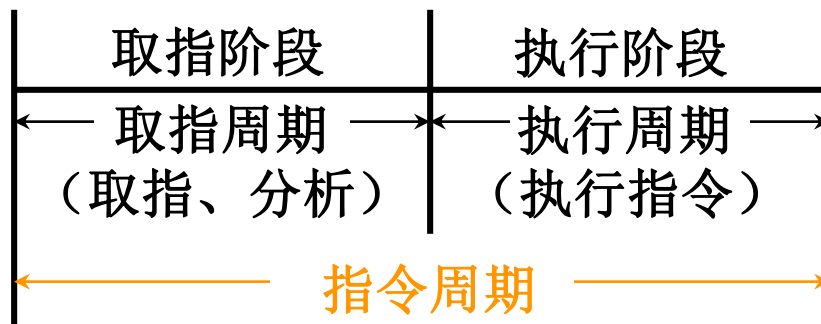
## 8.2 指令周期

### 一、指令周期的基本概念

#### 1. 指令周期

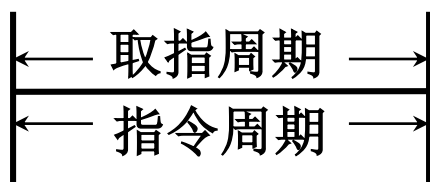
取出并执行一条指令所需的全部时间

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

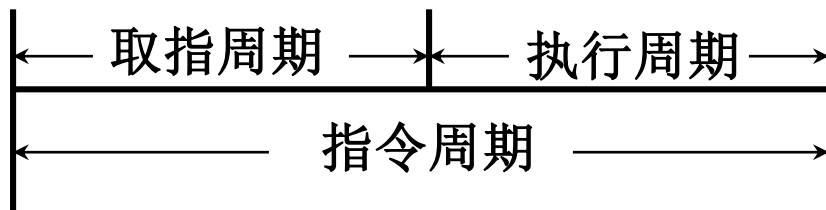


## 2. 每条指令的指令周期不同

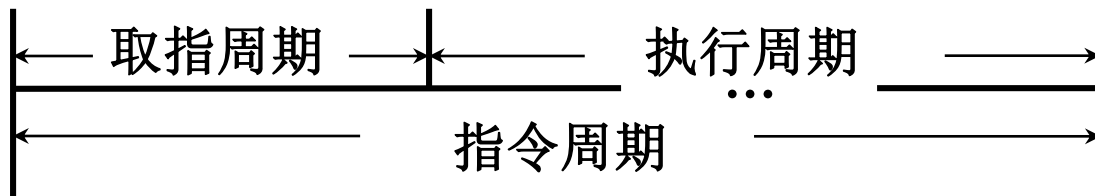
# 8.2



**NOP**



**ADD mem**

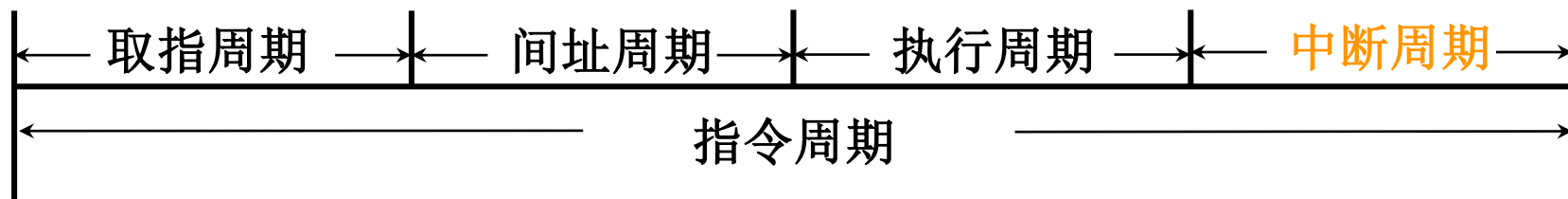


**MUL mem**

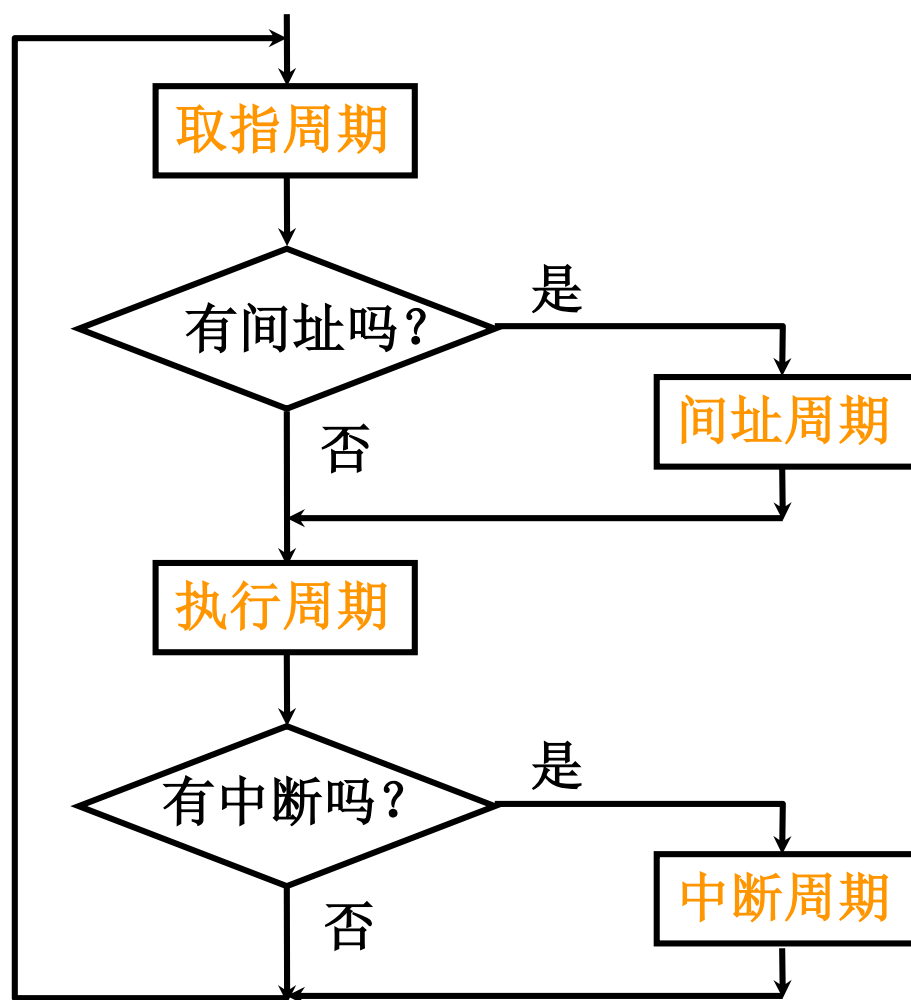
### 3. 具有间接寻址的指令周期



### 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

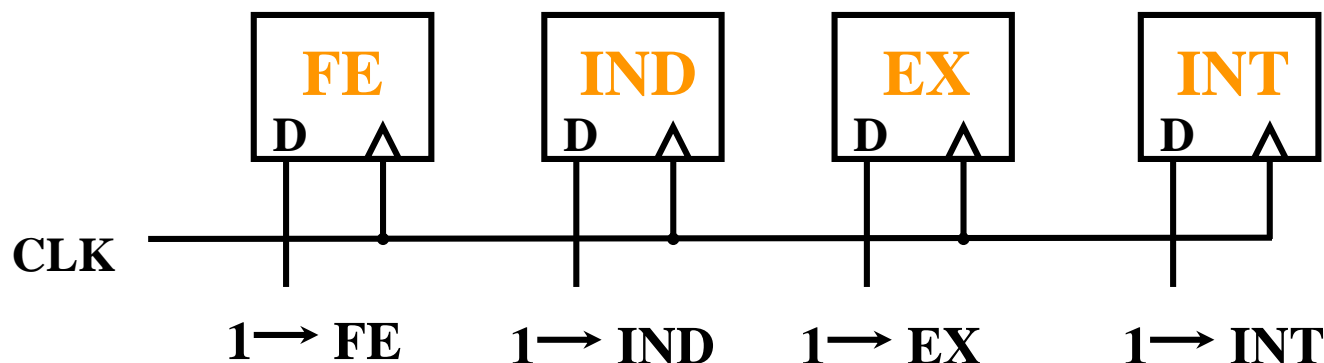
取 操作数

执行周期

存 程序断点

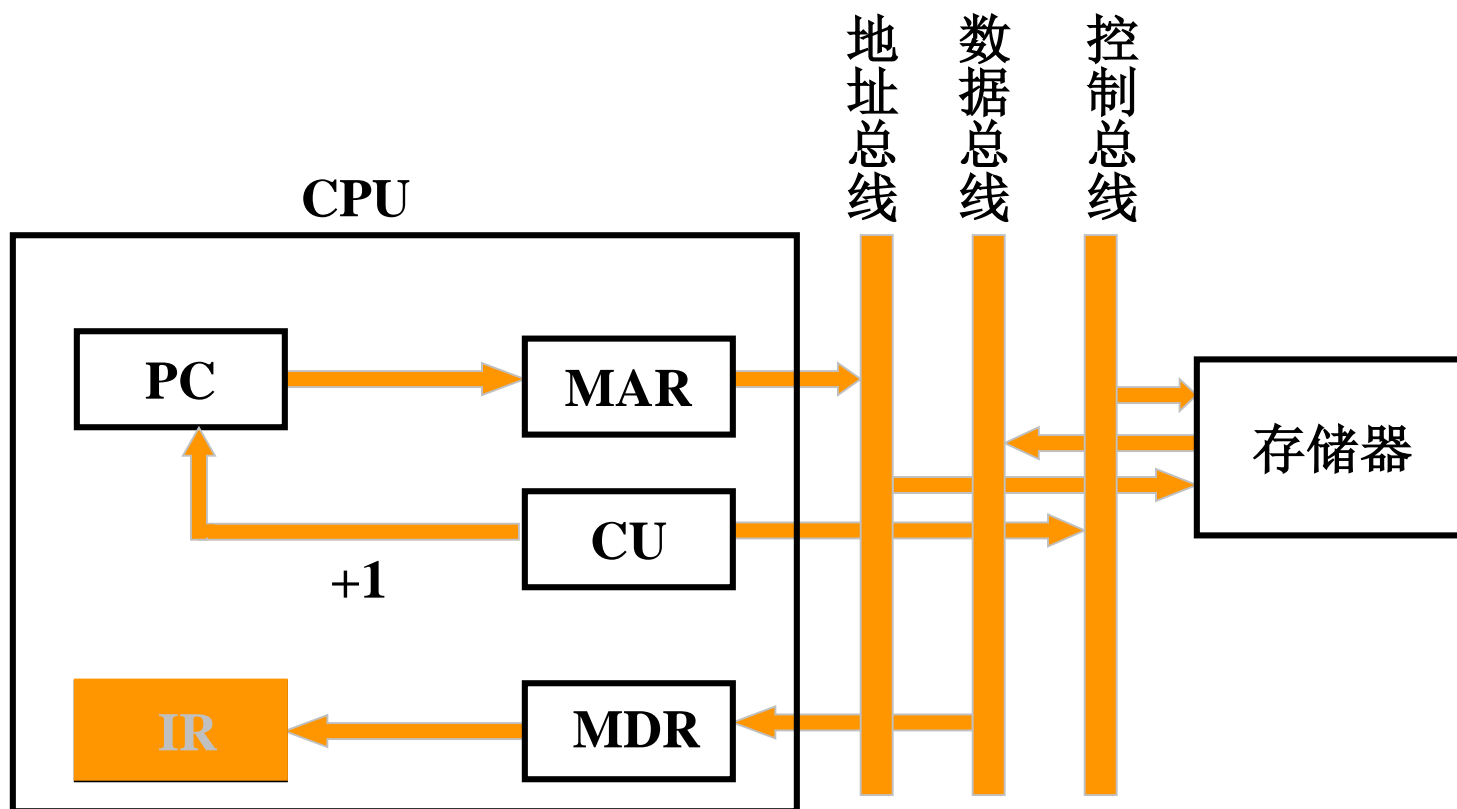
中断周期

CPU 的  
4个工作周期



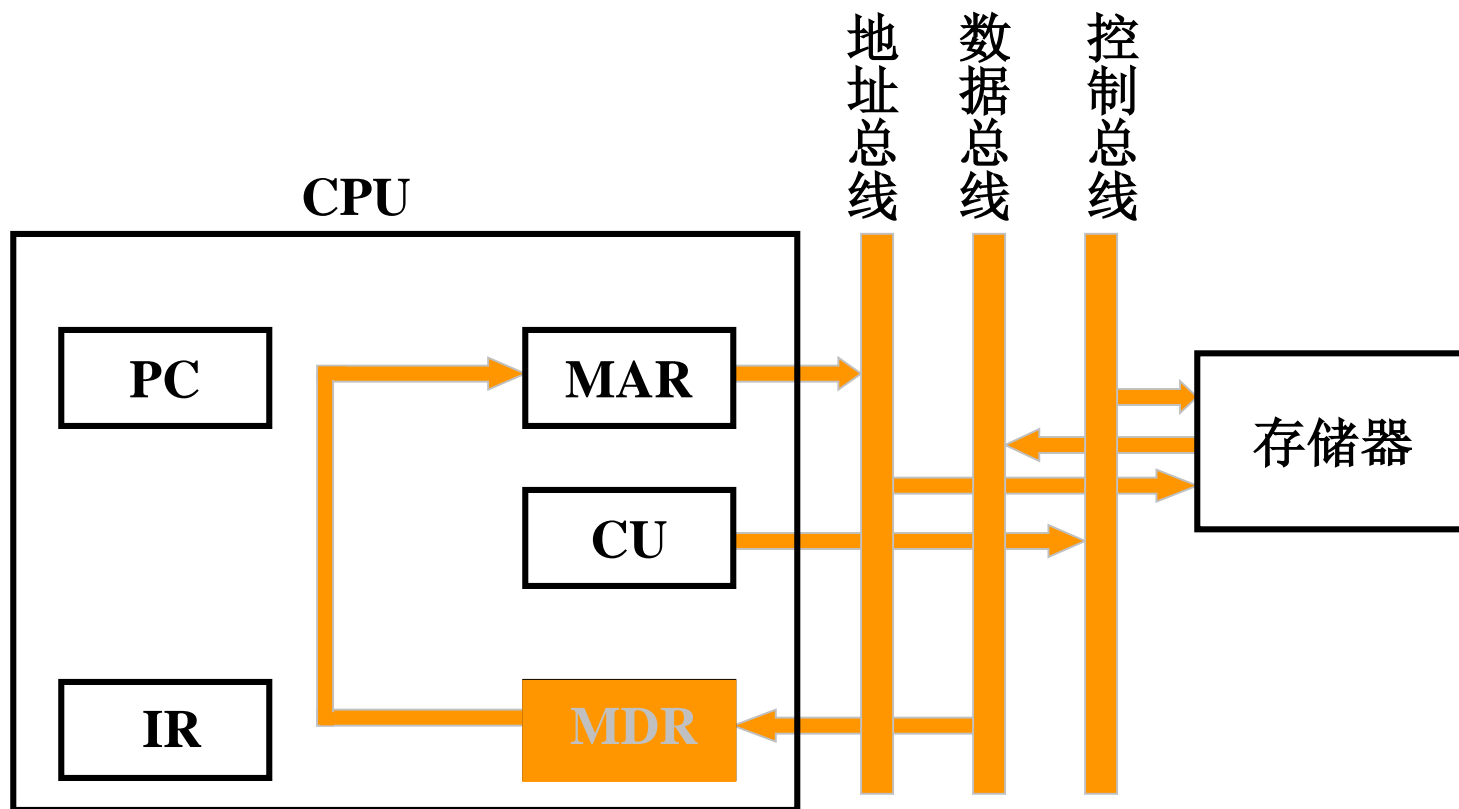
## 二、指令周期的数据流

### 1. 取指周期数据流



## 2. 间址周期数据流

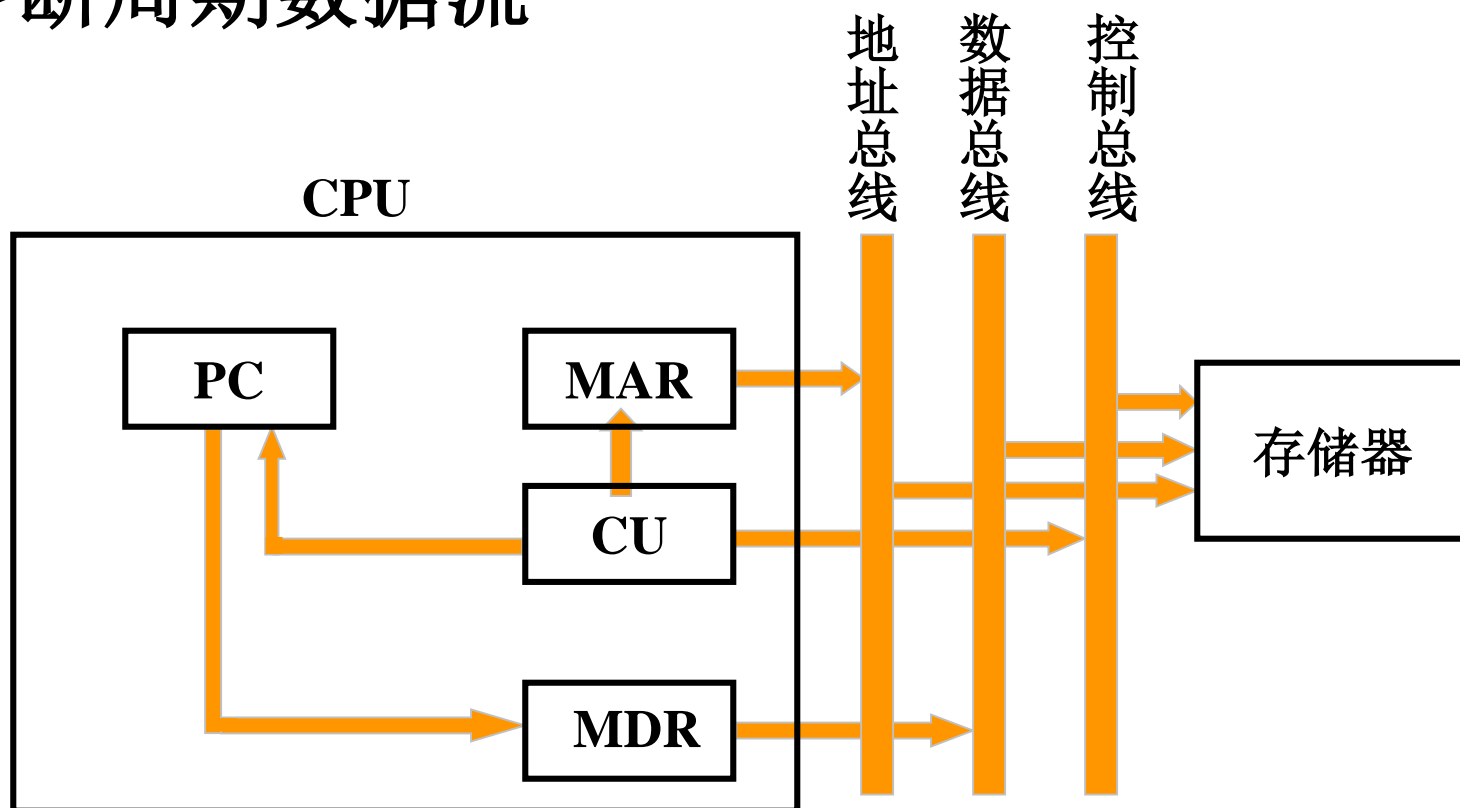
# 8.2



### 3. 执行周期数据流

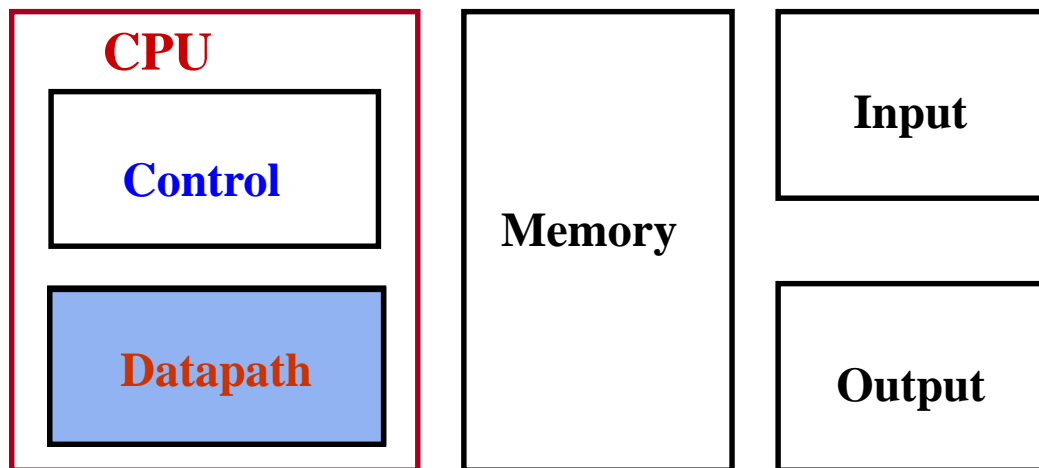
不同指令的执行周期数据流不同

### 4. 中断周期数据流



### 三、数据通路

❖ 计算机的五大组成部分：



❖ 什么是数据通路（DataPath）？

- 指令执行过程中，数据所经过的路径，包括路径中的部件。它是**指令的执行部件**。

❖ 控制器（Control）的功能是什么？

- 对指令进行译码，生成指令对应的控制信号，控制数据通路的动作。能对执行部件发出控制信号，是**指令的控制部件**。

# 数据通路的基本结构

## ❖ 数据通路由两类部件组成

- 组合逻辑元件（也称操作元件）
- 存储元件（也称状态元件）

## ❖ 元件间的连接方式

- 总线连接方式
- 分散连接方式

## ❖ 数据通路如何构成？

- 由“操作元件”和“存储元件”通过总线方式或分散方式连接而成

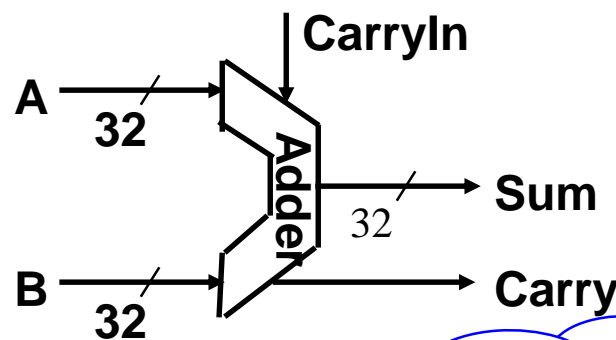
## ❖ 数据通路的功能是什么？

- 进行数据存储、处理、传送

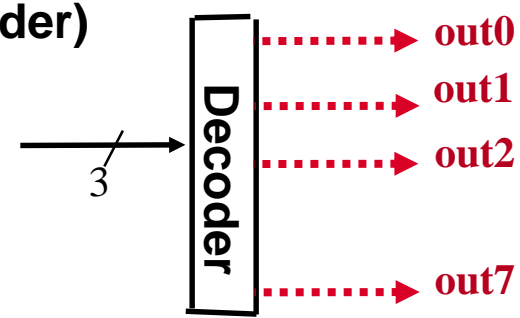
因此，数据通路是由操作元件和存储元件通过总线方式或分散方式连接而成的进行数据存储、处理、传送的路径。

.....➡ 控制信号

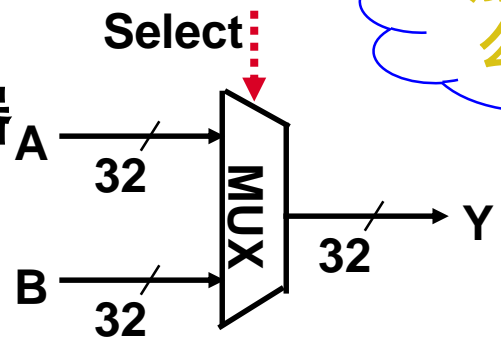
# ❖ 加法器 (Adder)



译码器 (Decoder)



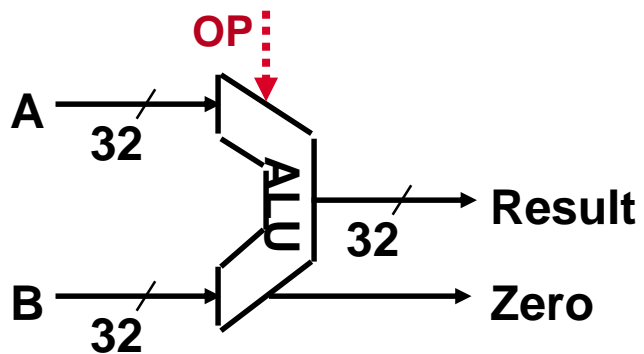
# 多路选择器 (MUX)



加法器需要什么控制信号?

何时要用到adder, ALU, MUX or Decoder?

# 算逻部件 (ALU)



组合逻辑元件的特点:

其输出只取决于当前的输入。即: 输入一样, 其输出也一样

定时: 所有输入到达后, 经过一定的逻辑门延时, 输出端改变, 并保持到下次改变, 不需要时钟信号来定时

# 状态元件：时序逻辑电路

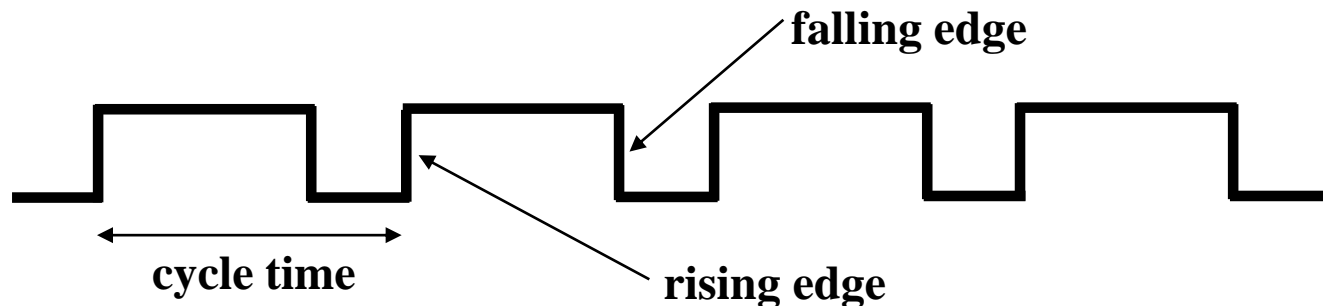
## ❖ 状态（存储）元件的特点：

- 具有存储功能，在**时钟控制**下输入被写到电路中，直到下个时钟到达
- 输入端状态由时钟决定何时被写入，输出端状态随时可以读出

## ❖ 定时方式：规定信号何时写入状态元件或何时从状态元件读出

### ■ 边沿触发（edge-triggered）方式：

- 状态单元中的值只在时钟边沿改变。每个时钟周期改变一次。
  - 上升沿（rising edge）触发：在时钟正跳变时进行读/写。
  - 下降沿（falling edge）触发：在时钟负跳变时进行读/写。

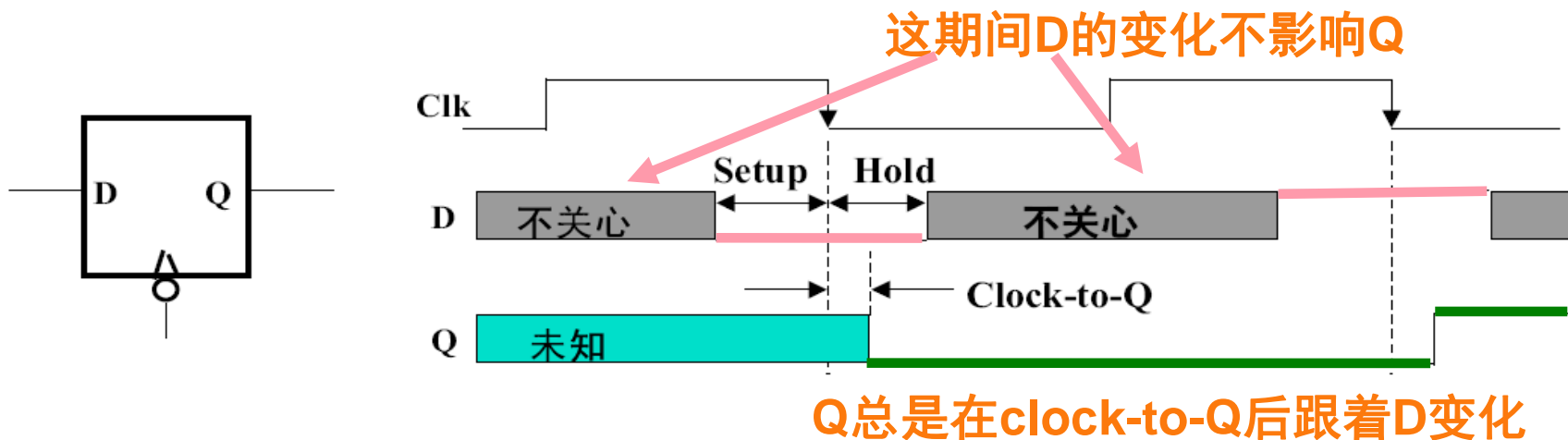


## ❖ 最简单的状态单元（回顾：数字逻辑电路课程内容）：

- D触发器：一个时钟输入、一个状态输入、一个状态输出



# 存储元件中何时状态被改变？



- ° 建立时间（**Setup Time**）：在触发时钟边沿 **之前** 输入必须稳定
- ° 保持时间（**Hold Time**）：在触发时钟边沿 **之后** 输入必须保持
- ° **Clock-to-Q time: ( Latch Prop - 锁存延迟 )**
  - 在触发时钟边沿，输出并不能立即变化

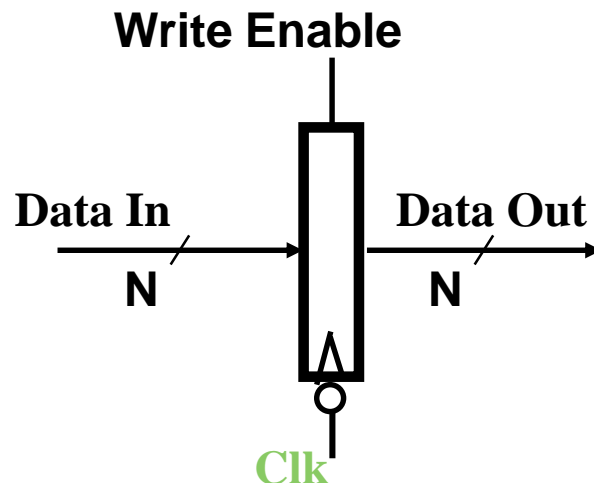
**切记：状态单元的输入信息总是在一个时钟边沿到达后的“Clk-to-Q”时才被写入到单元中，此时的输出才反映新的状态值**

**数据通路中的状态元件有两种：寄存器(组) + 存储器**

# 存储元件：寄存器和寄存器组

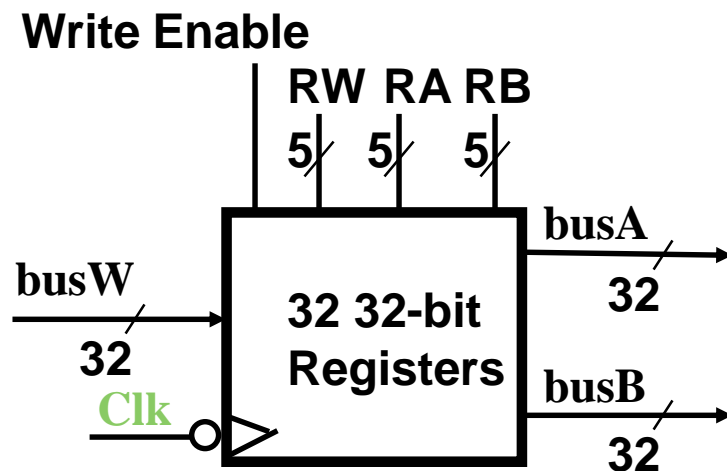
## ❖ 寄存器 (Register)

- 有一个写使能 (Write Enable-WE) 信号
  - 0: 时钟边沿到来时, 输出不变
  - 1: 时钟边沿到来时, 输出开始变为输入
- 若每个时钟边沿都写入, 则不需WE信号

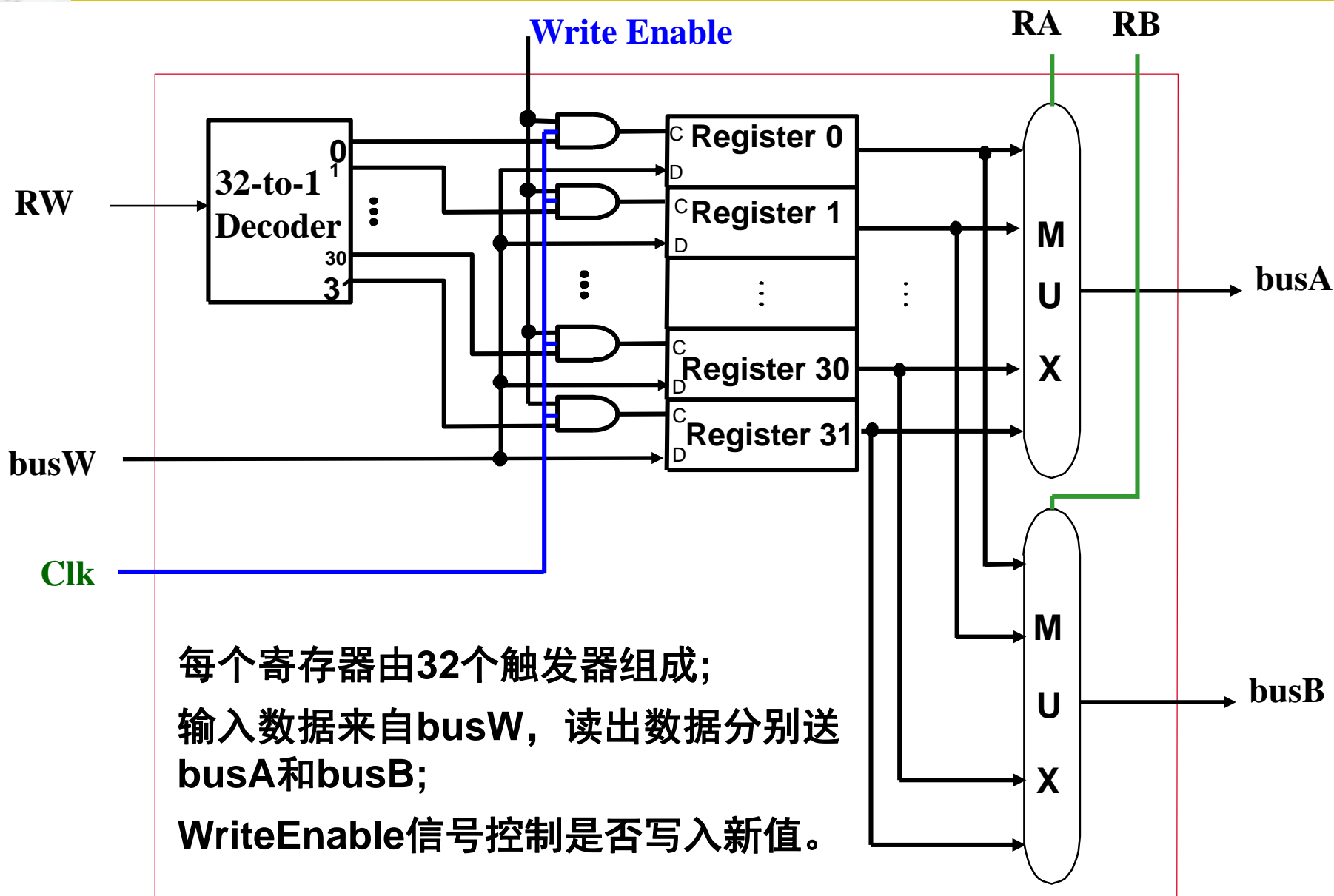


## ❖ 寄存器组 (Register File)

- 两个读口 (组合逻辑操作): busA和busB分别由RA和RB给出地址。地址RA或RB有效后, 经一个“取数时间 (AccessTime)”, busA和busB有效。
- 一个写口 (时序逻辑操作): 写使能为1的情况下, 时钟边沿到来时, busW传来的值开始被写入RW指定的寄存器中。



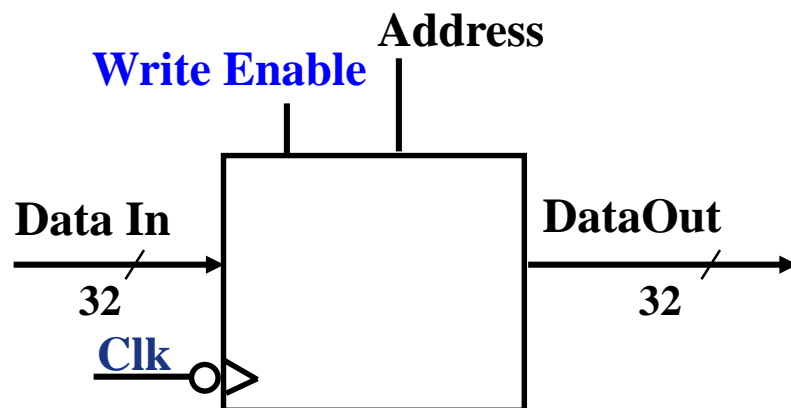
# 寄存器组的内部结构



# 存储元件：理想存储器

## ❖ 理想存储器（idealized memory）

- Data Out: 32位读出数据
- Data In: 32位写入数据
- Address: 读写公用一个32位地址
- 读操作（组合逻辑操作）：地址Address有效后，经一个“取数时间AccessTime”，Data Out上数据有效。
- 写操作（时序逻辑操作）：写使能为1的情况下，时钟Clk边沿到来时，Data In传来的值开始被写入Address指定的存储单元中。



为简化数据通路操作说明，把存储器简化为带时钟信号Clk的理想模型。

# 数据通路与时序控制

## ❖ 同步系统(Synchronous system)

- 所有动作有专门时序信号来定时
- 由时序信号规定何时发出什么动作

例如，指令执行过程每一步都有控制信号控制，由定时信号确定控制信号何时发出、作用时间多长

## ❖ 什么是时序信号？

- 同步系统用于同步控制的定时信号，如时钟信号

## ❖ 什么叫指令周期？

- 取并执行一条指令的时间
- 每条指令的指令周期肯定一样吗？

## ❖ 早期计算机的三级时序系统

- 机器周期 - 节拍 - 脉冲
- 指令周期可分为取指令、读操作数、执行并写结果等多个基本工作周期，称为机器周期。
- 机器周期有取指令、存储器读、存储器写、中断响应等不同类型

# 多级时序系统

## 1. 机器周期(CPU周期)

### (1) 机器周期的概念

所有指令执行过程中的一个基准时间

### (2) 确定机器周期需考虑的因素

每条指令的执行 步骤

每一步骤 所需的 时间

### (3) 基准时间的确定

- 以完成 最复杂 指令功能的时间 为准
- 以 访问一次存储器 的时间 为基准

若指令字长 = 存储字长      取指周期 = 机器周期

## 2. 时钟周期（节拍、状态）

- 一个机器周期内可完成若干个微操作
- 每个微操作需一定的时间，以时钟信号来控制产生每一个微操作命令
- 时钟信号控制节拍发生器，产生节拍，每个节拍宽度对应一个时钟周期
- 将一个机器周期分成若干个时间相等的时间段（节拍、状态、时钟周期）
- 时钟周期是控制计算机操作的最小单位时间
- 用时钟周期控制产生一个或几个微操作命令



# 多级时序系统

- 指令周期是从取指令、分析指令到执行完该指令所需的时间。
- 不同的指令，其指令周期长短可以不同。
- 在时序系统中通常不为指令周期设置时间标志信号，因而也不将其作为时序的一级

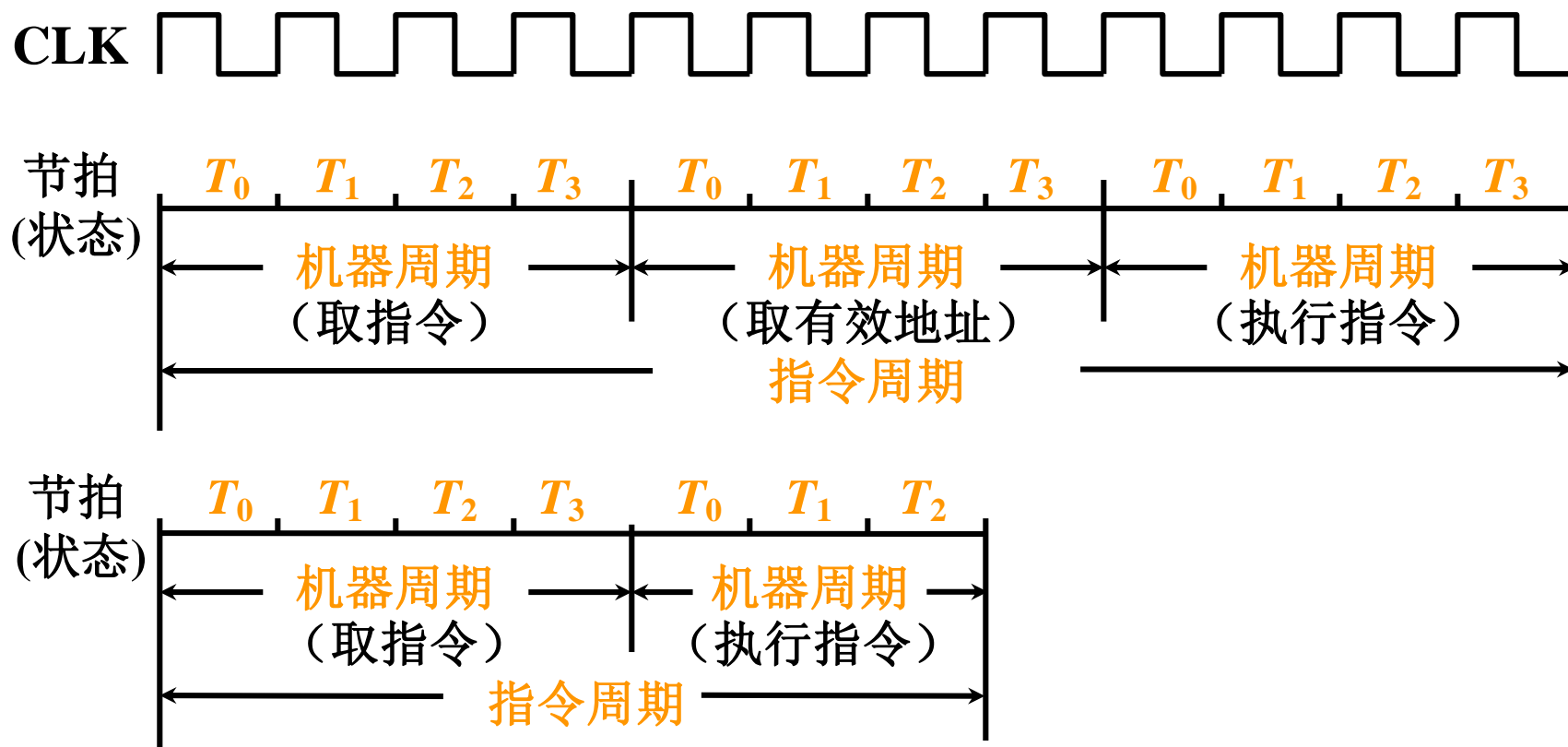


# 多级时序系统

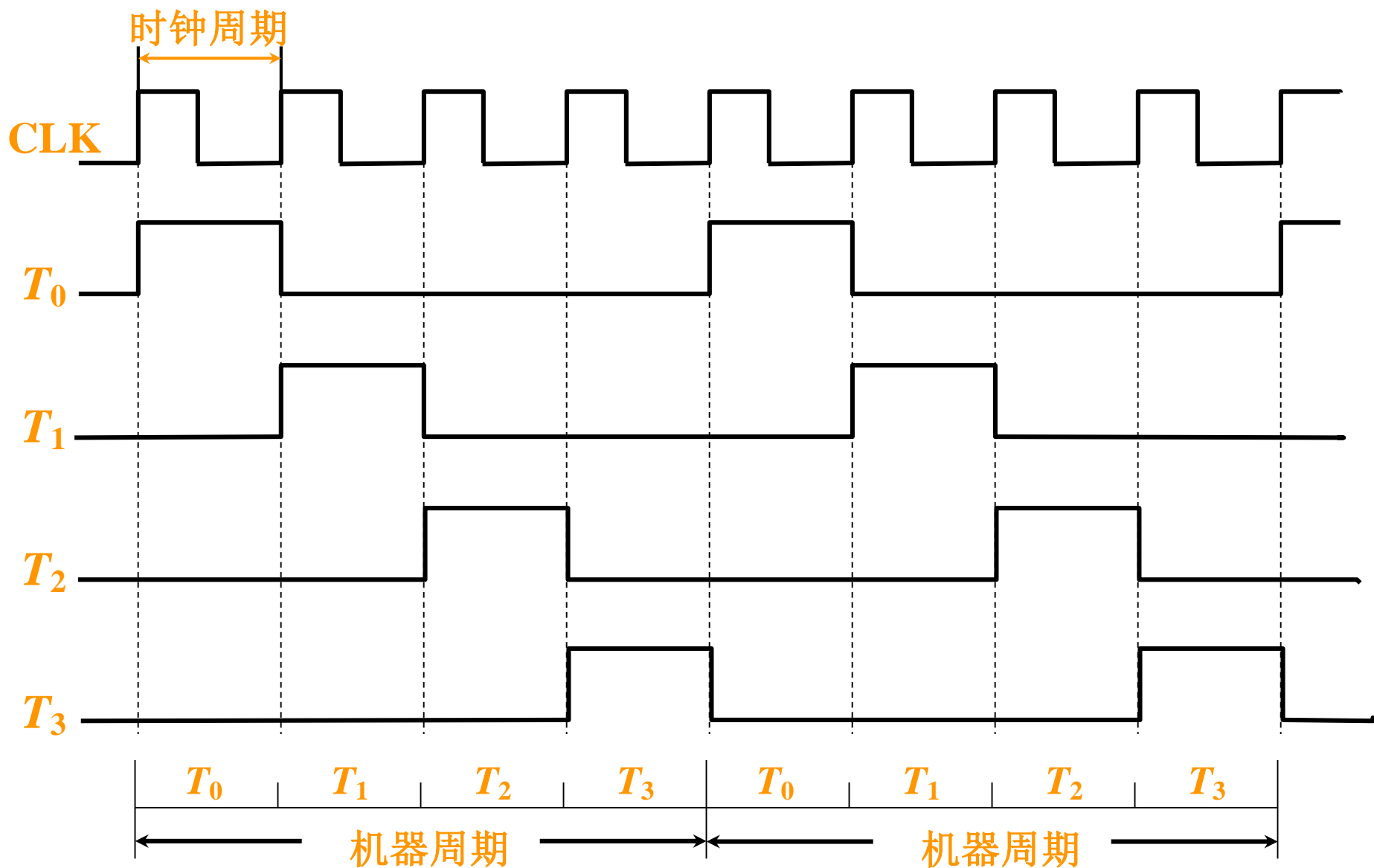
机器周期、节拍（状态）组成多级时序系统

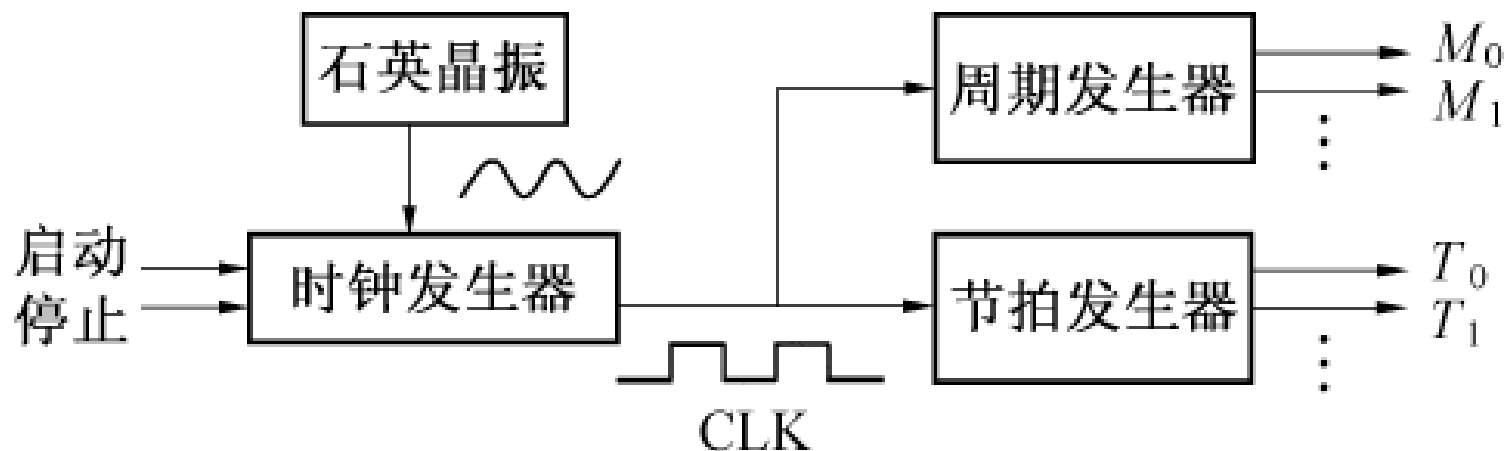
一个指令周期包含若干个机器周期

一个机器周期包含若干个时钟周期

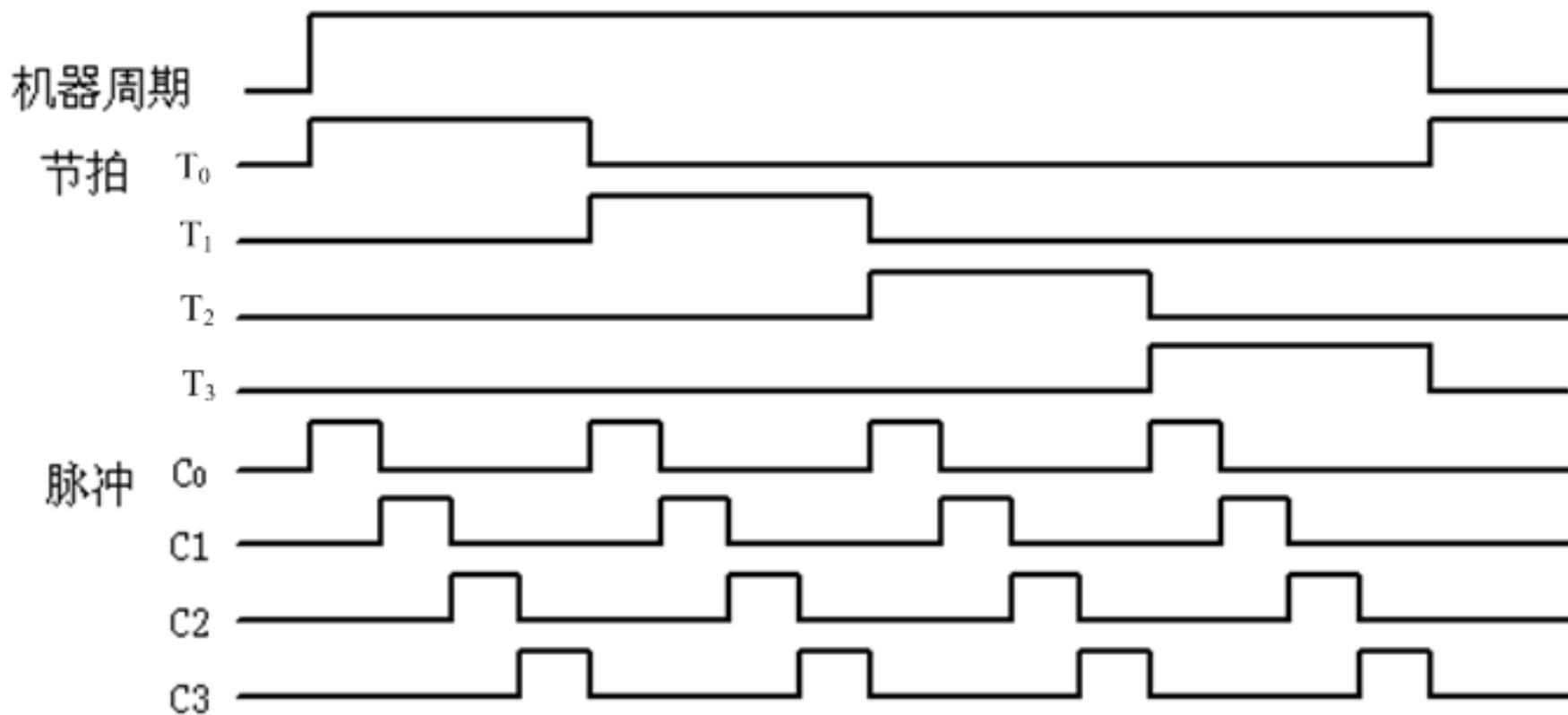


# 时钟周期（节拍、状态）

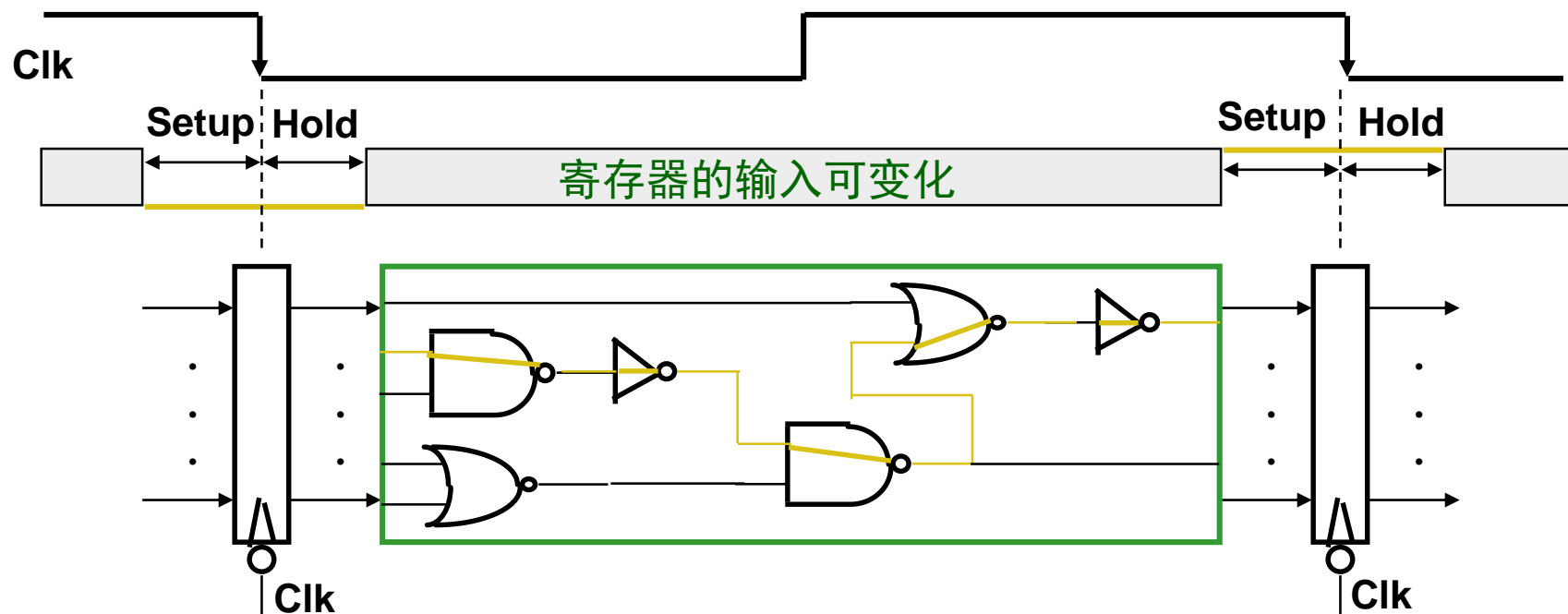




- ❖ 主振电路中的石英晶体振荡器产生一系列正弦信号，经时钟发生器整形分频得到时钟脉冲信号，成为机器的主频脉冲。主频脉冲再经过周期发生器和节拍发生器产生周期电位和节拍电位。



现代计算机已不再采用三级时序系统，机器周期的概念已逐渐消失。整个数据通路中的定时信号就是时钟，一个时钟周期就是一个节拍。



数据通路由 “... + 状态元件 + 操作元件( 组合电路) + 状态元件 + ...” 组成

只有状态元件能存储信息，所有操作元件都须从状态单元接收输入，并将输出写入状态单元中。其输入为前一时钟生成的数据，输出为当前时钟所用的数据

- ❖ 假定采用下降沿触发（负跳变）方式（也可以是上升沿方式）
  - 所有状态单元在下降沿写入信息，经过Latch Prop (clk-to-Q) 后输出有效
  - $\text{Cycle Time} = \text{Latch Prop} + \text{Longest Delay Path} + \text{Setup} + \text{Clock Skew(最大偏移)}$
- ❖ 约束条件： $(\text{Latch Prop} + \text{Shortest Delay Path} - \text{Clock Skew}) > \text{Hold Time}$

# 机器速度与机器主频的关系

- 机器的 主频  $f$  越快 机器的 速度也越快
- 在机器周期所含时钟周期数 相同 的前提下，  
两机 平均指令执行速度之比 等于 两机主频之比

$$\frac{\text{MIPS}_1}{\text{MIPS}_2} = \frac{f_1}{f_2}$$

- 机器速度 不仅与 主频有关 ， 还与机器周期中所含时钟周期（主频的倒数） 数 以及指令周期中所含的 机器周期数有关
- 机器运行速度还与字长和计算机体系结构有关。  
字长越长，单位时间内完成的数据运算就越多，运算速度越快  
体系结构：存储器采用分级结构，处理器采用流水结构、多机结构等

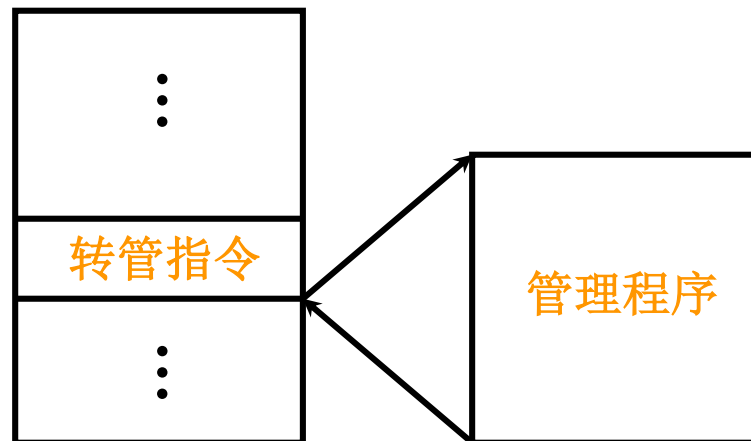
## 8.4 中断系统

### 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

## 1. 按中断来源分：

（1）内中断：发生在主机内部的中断称为内中断

（2）外中断：由主机外部事件引起的中断称为外中断



## 2. 按中断服务程序入口地址的获取方式分

- (1) **向量中断**：外部设备在提出中断请求的同时，通过**硬件**自动形成**中断服务程序入口地址**。**CPU**响应中断后，将根据向量地址直接转入相应中断服务程序。这种具有产生向量地址的中断称为向量中断。
- (2) **非向量中断**：非向量中断在产生中断的同时不能直接提供中断服务程序入口地址，而只产生一个**中断查询程序的入口地址**。需要通过中断查询程序确定中断源和中断服务程序的入口地址。



### 3. 按中断源位置分：

- 硬件中断
- 软件中断

### 4. 按是否可屏蔽分：**CPU**根据需要可以禁止响应某些中断源的中断请求。

- 可屏蔽中断：**CPU**可以禁止响应的中断
- 不可屏蔽中断：**CPU**必须响应的中断。

## 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

### 1. 中断请求标记 INTR

一个请求源 一个 INTR 中断请求触发器

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

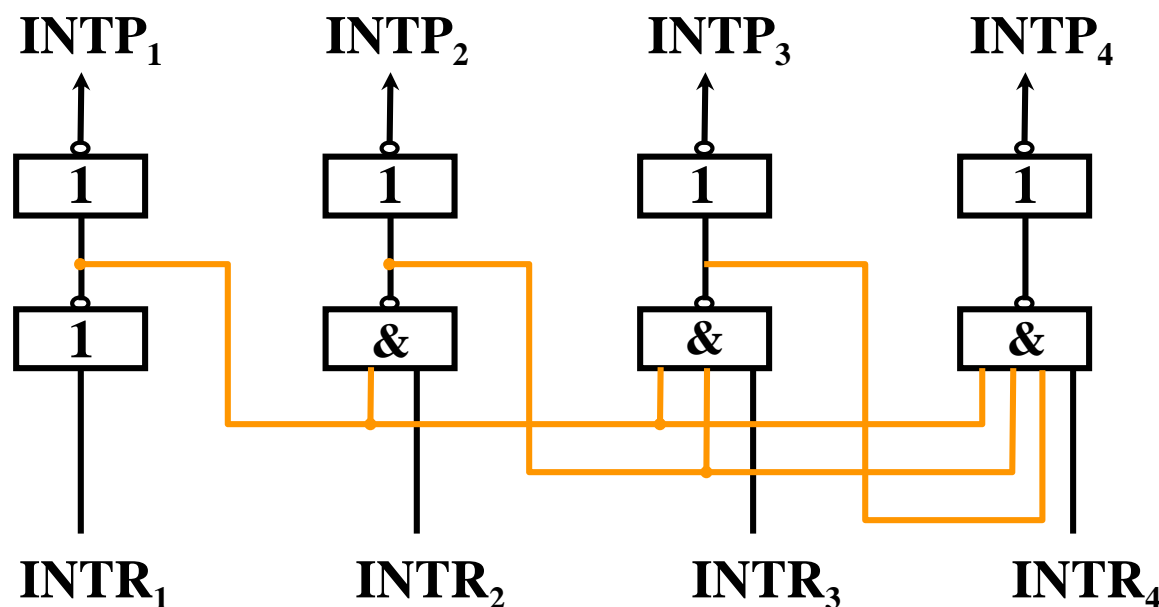
## 2. 中断判优逻辑

### (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

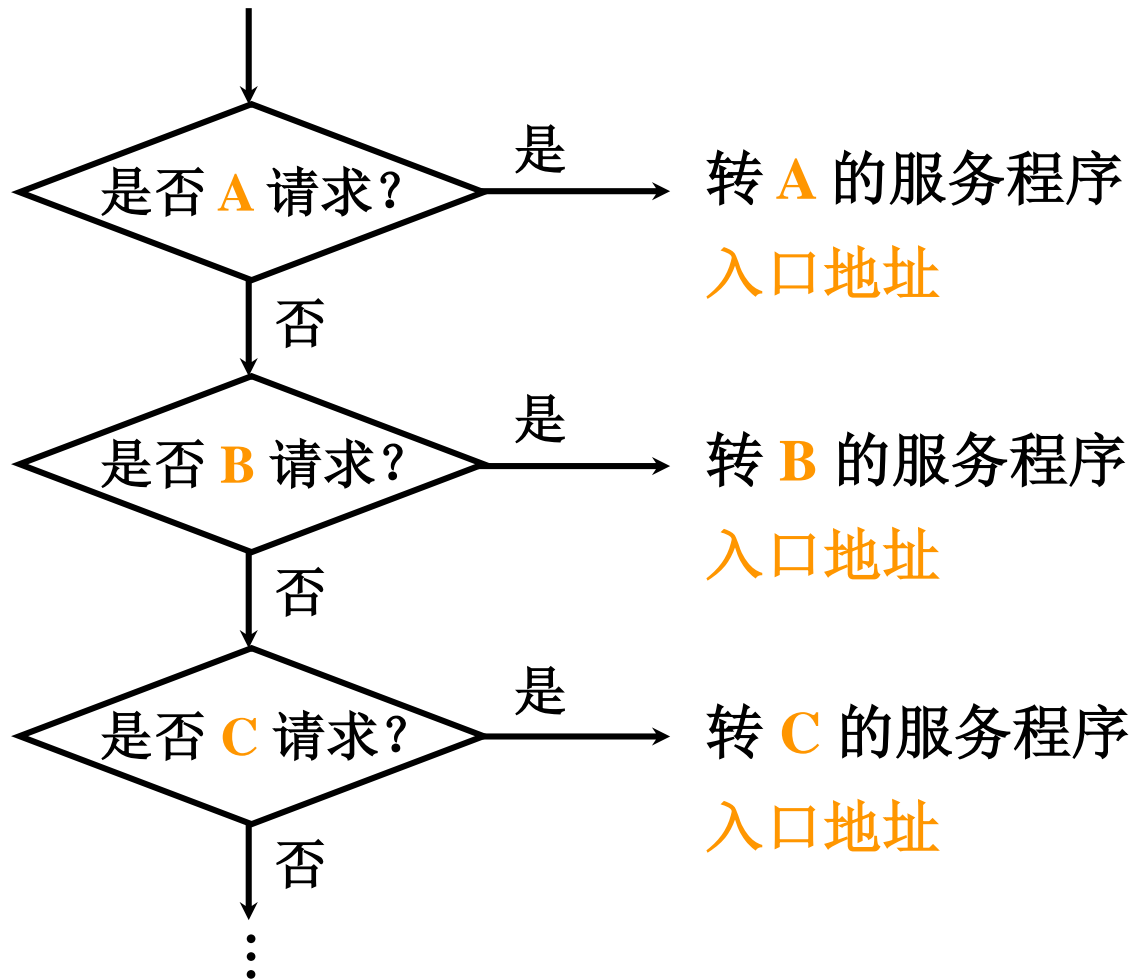
② 集中 在 CPU 内



$INTR_1$ 、 $INTR_2$ 、 $INTR_3$ 、 $INTR_4$  优先级 按 降序 排列

## (2) 软件实现（程序查询）

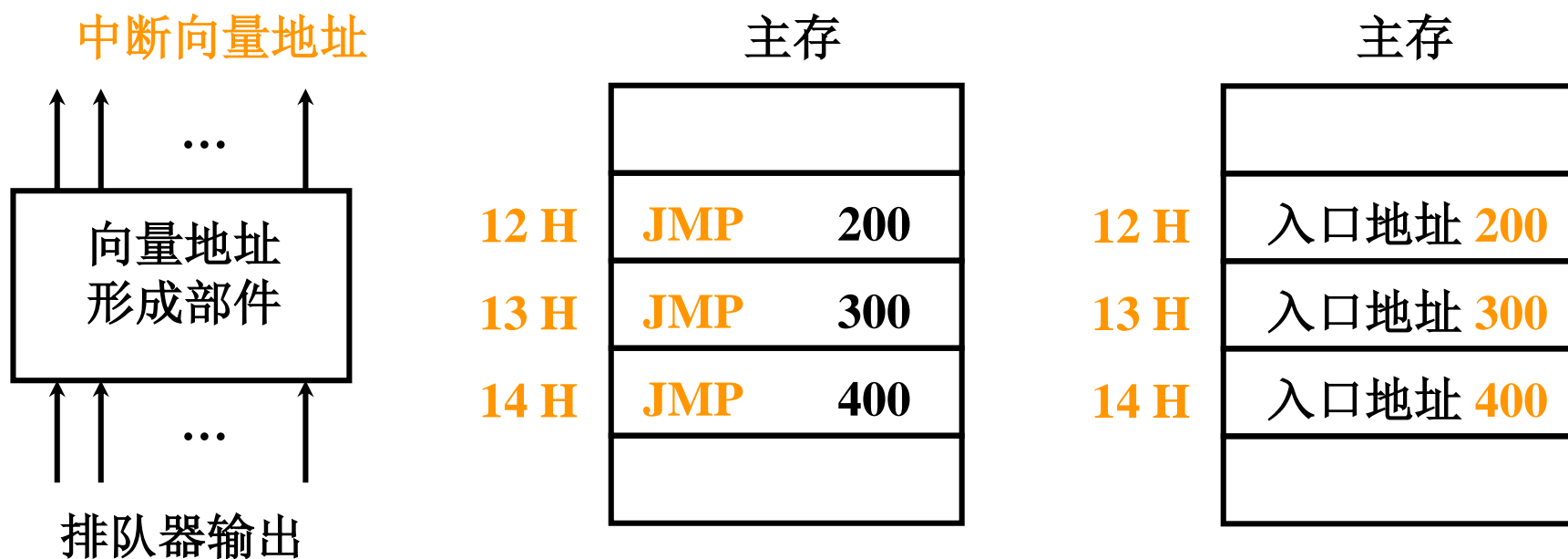
A、B、C 优先级按 降序 排列



# 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

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



允许中断触发器 **EINT = 1**（可被开中断指令置1也可被关中断指令置0）

## 指令执行周期结束时刻由CPU 发查询信号



## (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 → PC（硬件向量法）

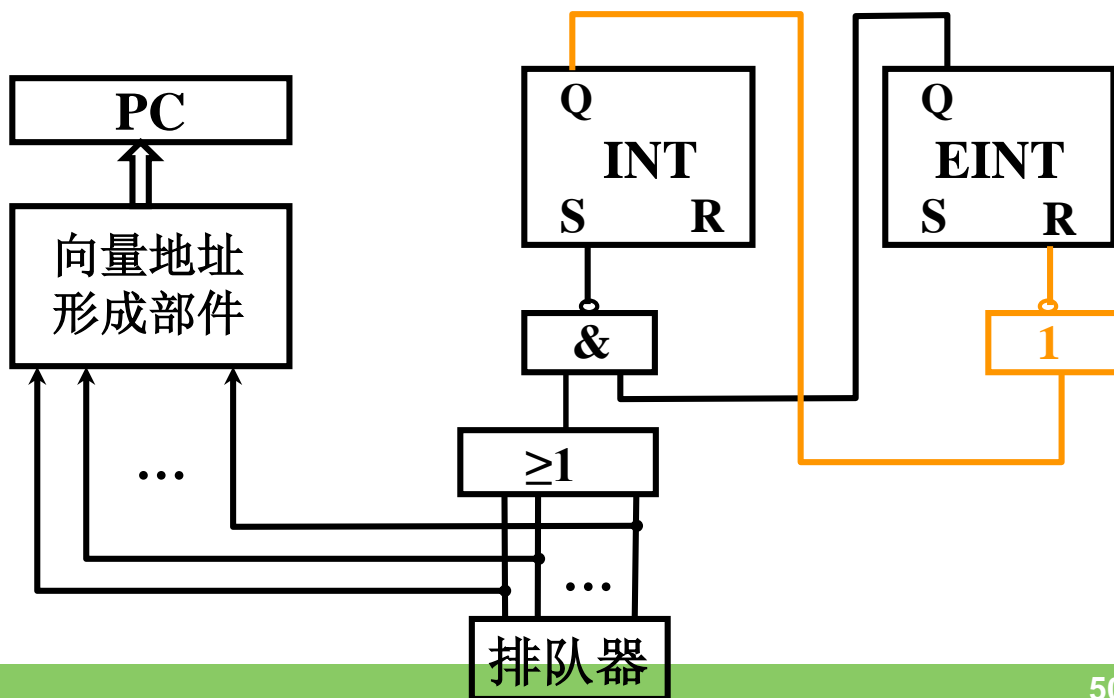
中断识别程序入口地址  $M \rightarrow PC$  (软件查询法)

### (3) 硬件关中断

## INT 中断标记

## EINT 允许中断

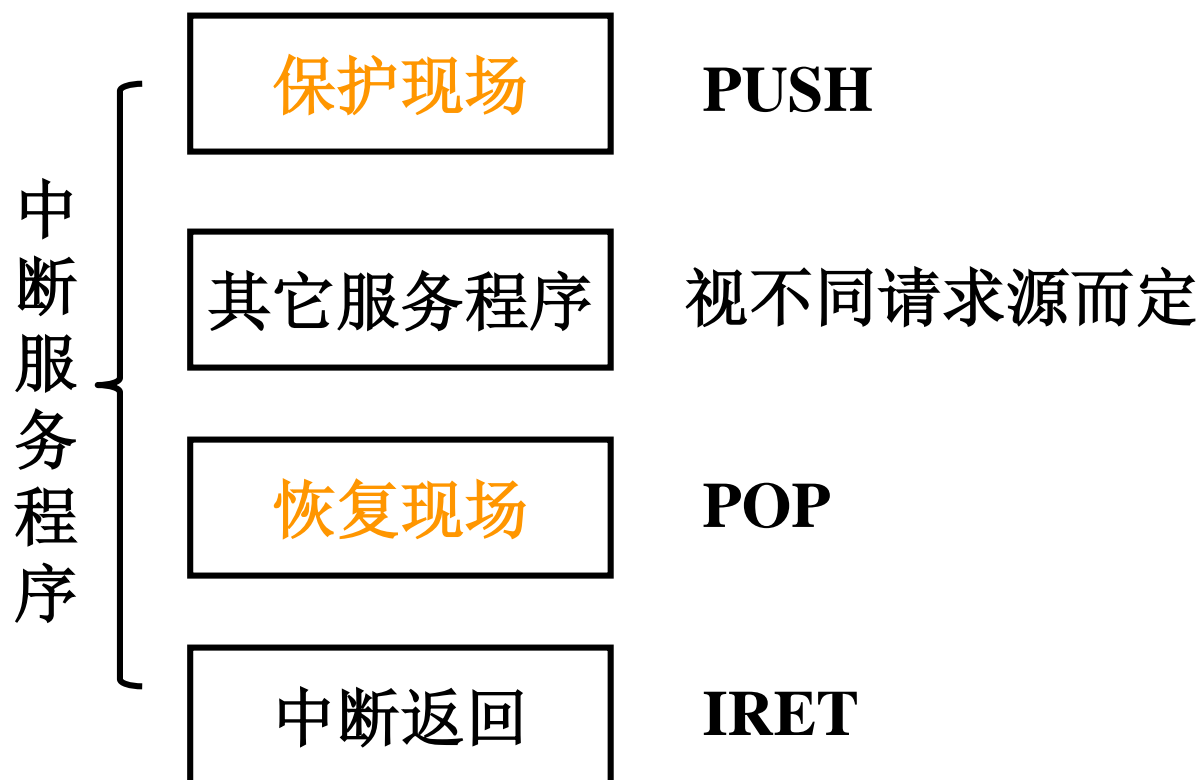
## R-S 触发器



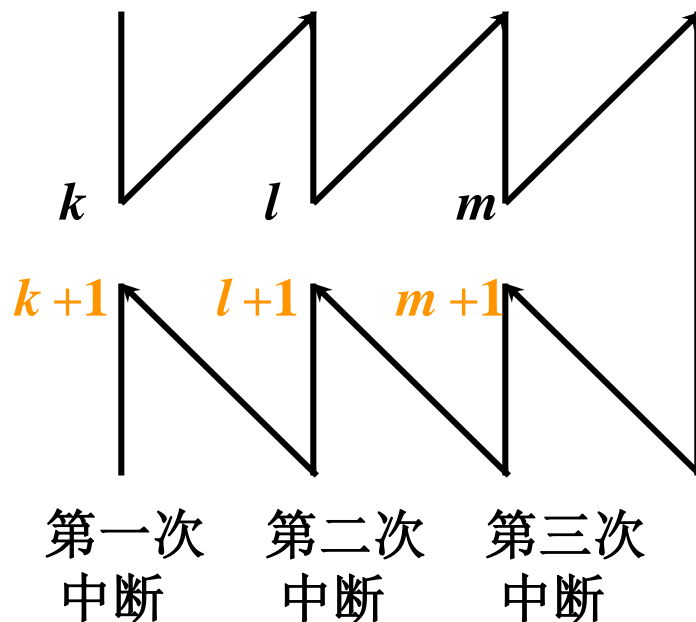
# 五、保护现场和恢复现场

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



### 1. 多重中断的概念



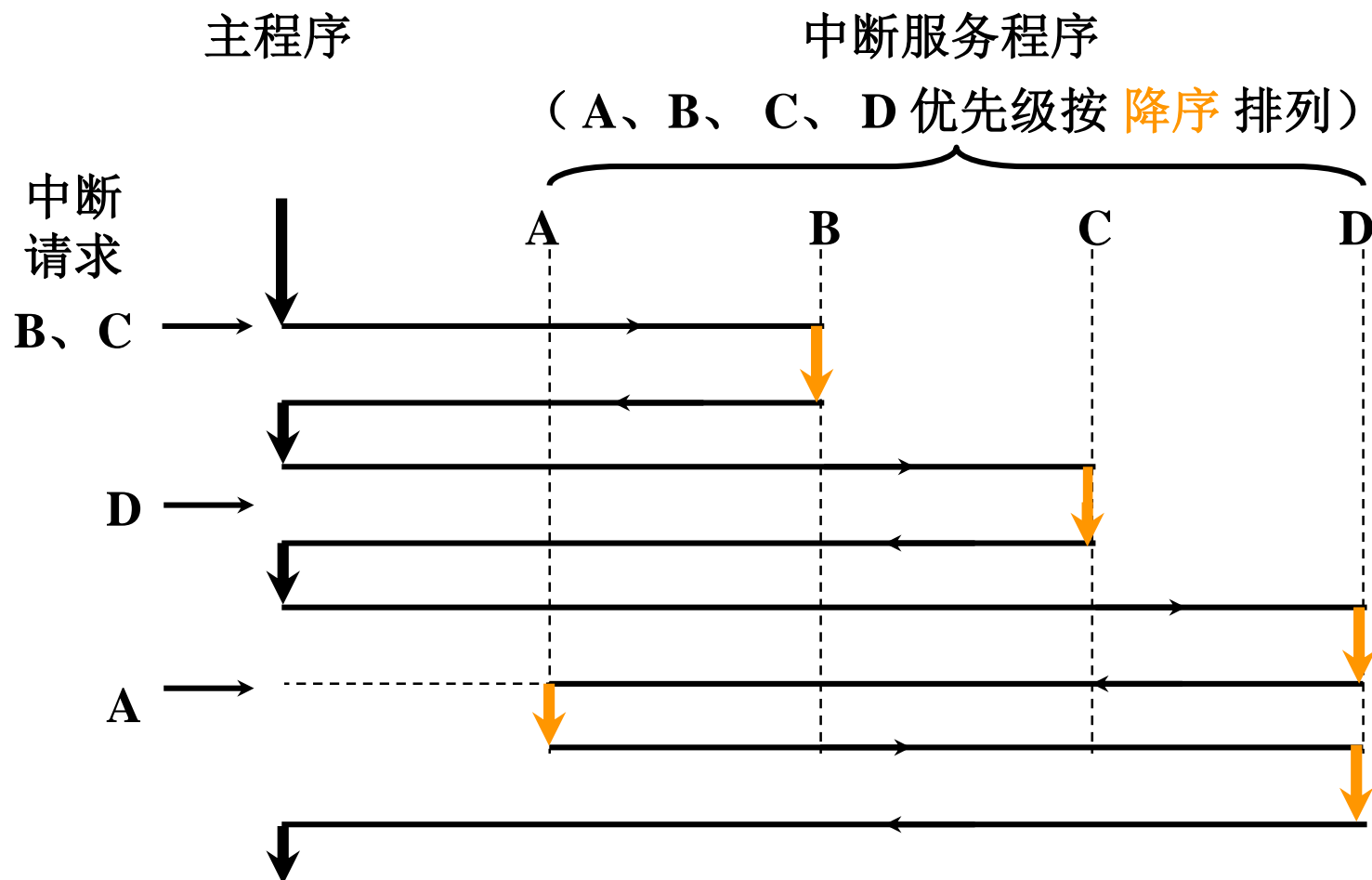
程序断点  $k+1$ ,  $l+1$ ,  $m+1$

## 2. 实现多重中断的条件

## 8.4

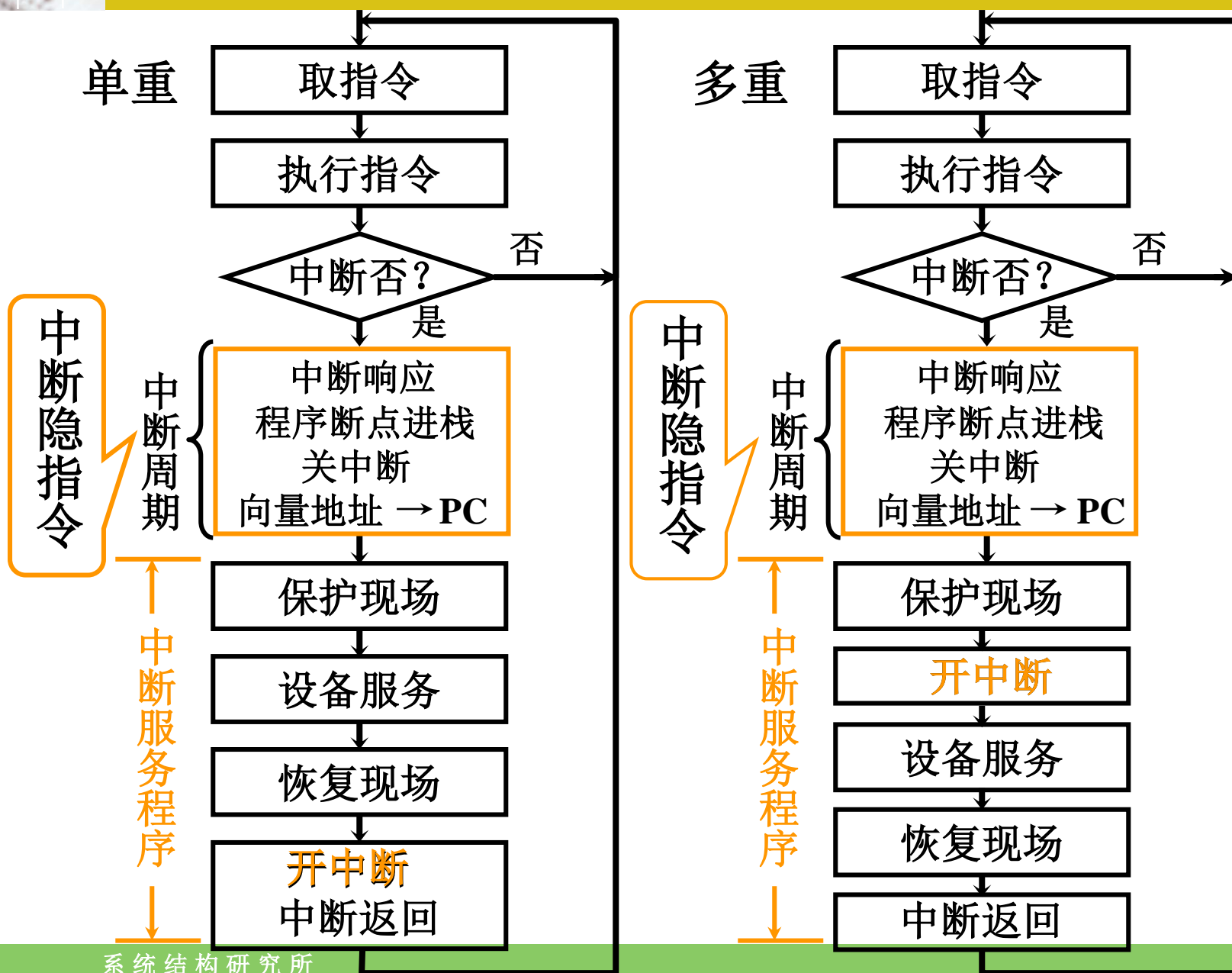
(1) 提前 设置 开中断 指令

(2) 优先级别高 的中断源 有权中断优先级别低 的中断源



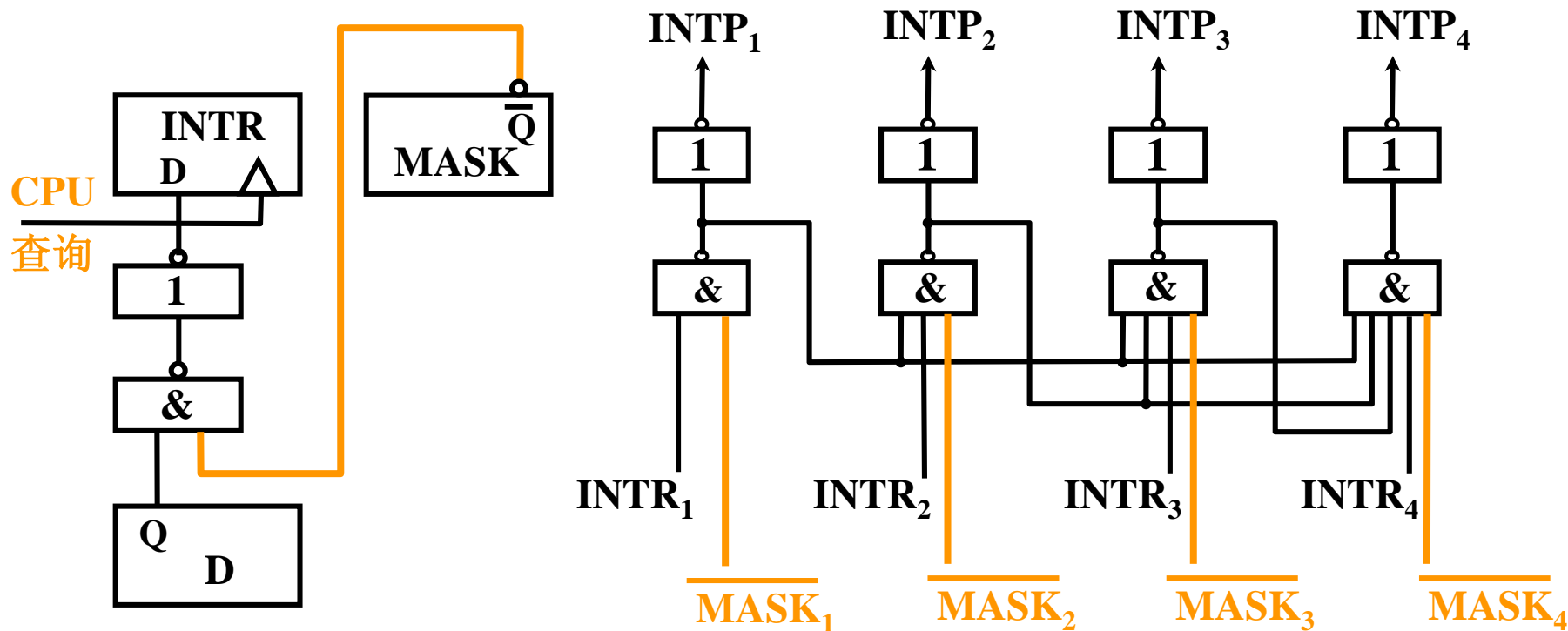
# 单重中断和多重中断的服务程序流程

5.5



# 3. 屏蔽技术

## (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

## 8.4

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

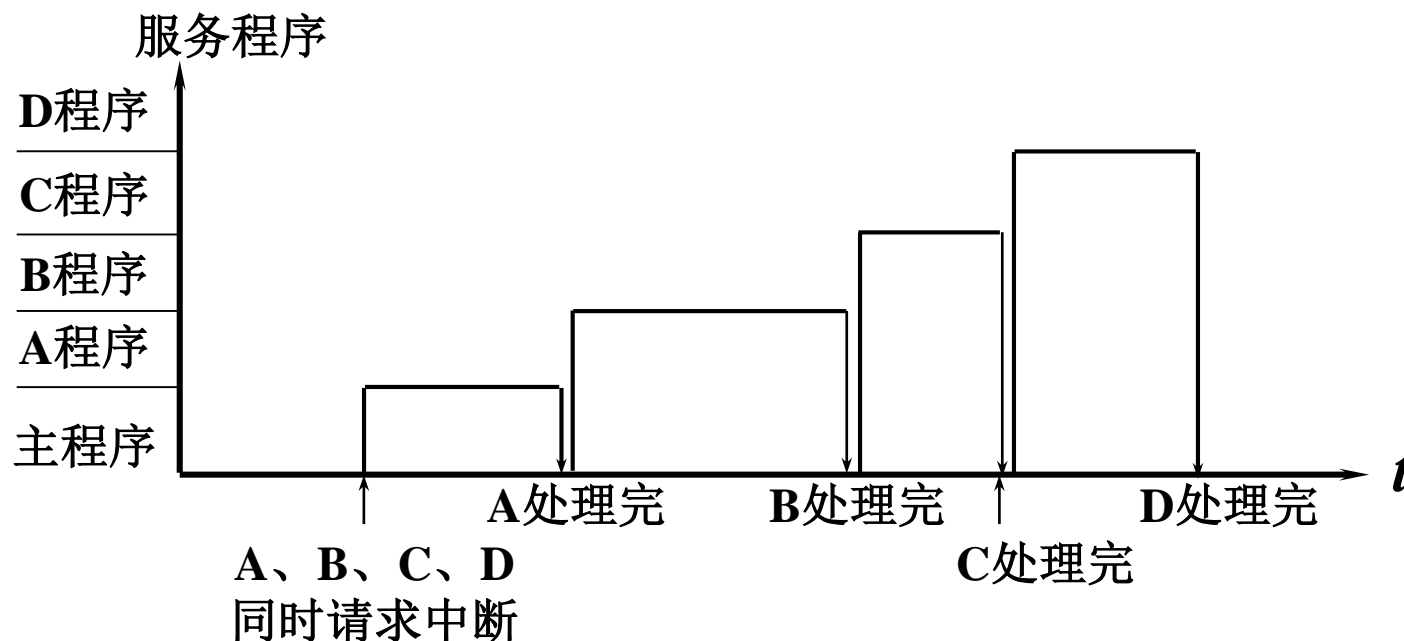
处理优先级      可改变（通过重新设置屏蔽字）

中断源	原屏蔽字	新屏蔽字
<b>A</b>	<b>1 1 1 1</b>	<b>1 1 1 1</b>
<b>B</b>	<b>0 1 1 1</b>	<b>0 1 0 0</b>
<b>C</b>	<b>0 0 1 1</b>	<b>0 1 1 0</b>
<b>D</b>	<b>0 0 0 1</b>	<b>0 1 1 1</b>

响应优先级 **A→B→C→D** 降序排列

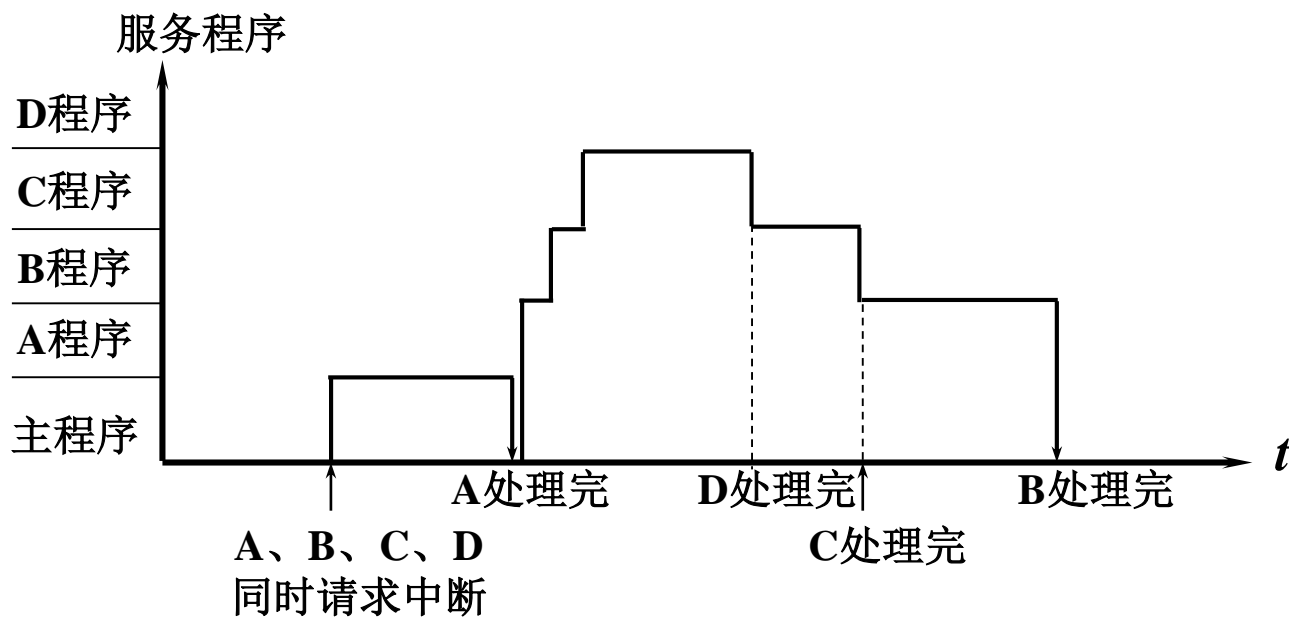
处理优先级 **A→D→C→B** 降序排列

### (3) 屏蔽技术可改变处理优先等级



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



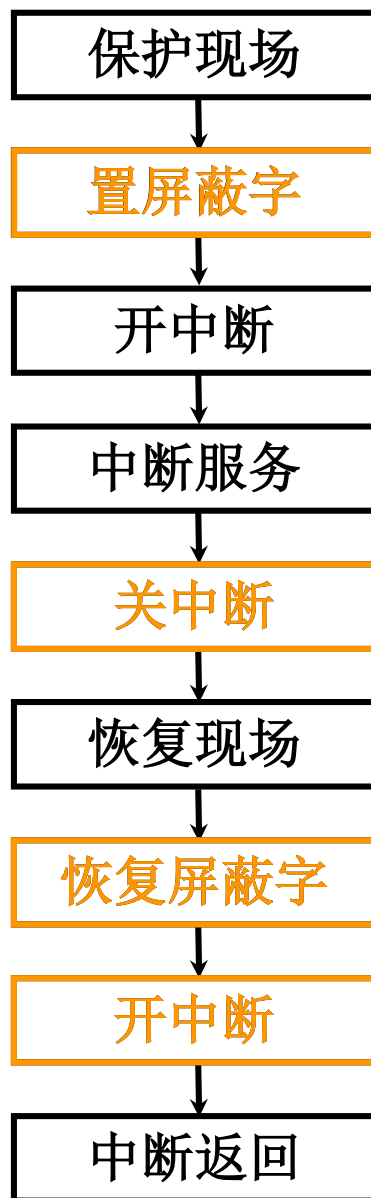
CPU 执行程序轨迹（新屏蔽字）

### (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



## (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 $\longrightarrow$ MDR

断点  $\rightarrow$  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?

# 例 题

1. 某机有五个中断源，按中断响应的优先顺序由高到低为L0,L1,L2,L3,L4，现要求优先顺序改为L4,L2,L3,L0,L1，写出各中断源的屏蔽字。

中断源	屏蔽字				
	0	1	2	3	4
L0					
L1					
L2					
L3					
L4					

中断源	屏蔽字				
	0	1	2	3	4
L0	1	1	0	0	0
L1	0	1	0	0	0
L2	1	1	1	1	0
L3	1	1	0	1	0
L4	1	1	1	1	1

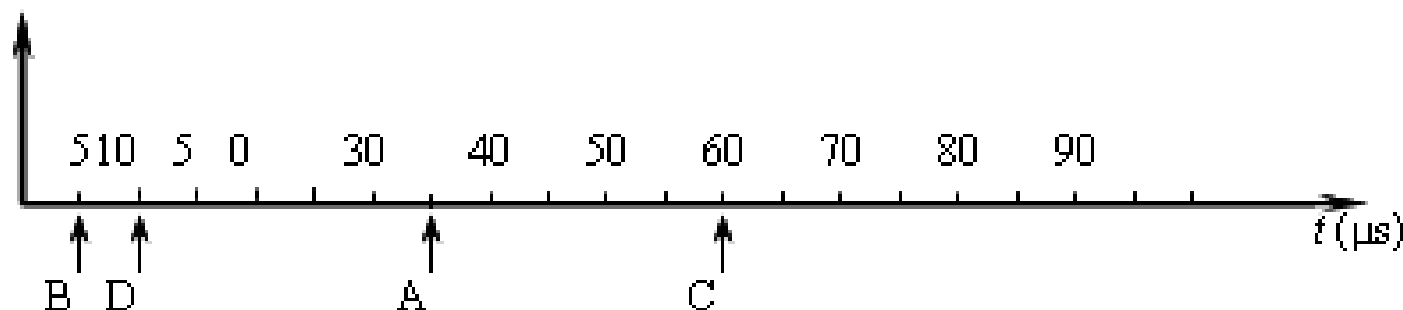
## 例 题

2. 设某机有四个中断源A、B、C、D，其硬件排队优先次序为 $A > B > C > D$ ，现要求将中断处理次序改为 $D > A > C > B$ 。

(1) 写出每个中断源对应的屏蔽字。

(2) 按下图时间轴给出的四个中断源的请求时刻，画出CPU执行程序的轨迹。设每个中断源的中断服务程序时间均为 $20\mu\text{s}$ 。

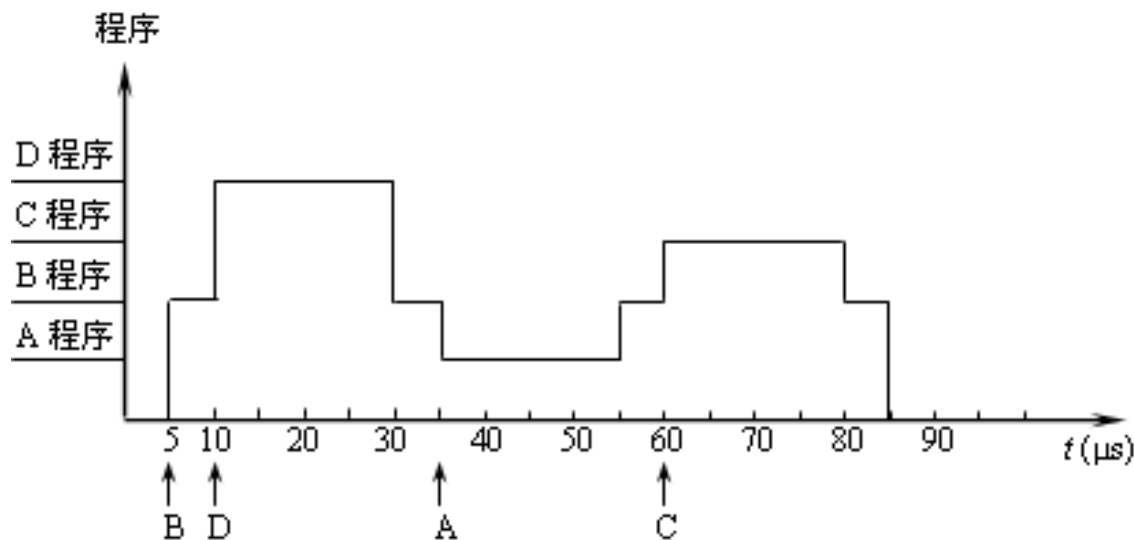
程序



答：（1）在中断处理次序  
改为 $D > A > C > B$ 后，  
每个中断源新的屏蔽字如  
表所示。

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

（2）根据新的处理次序，  
CPU执行程序的轨迹如图所示



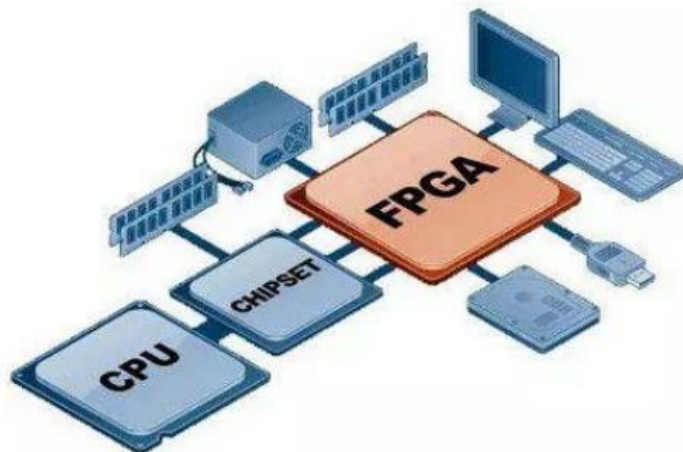




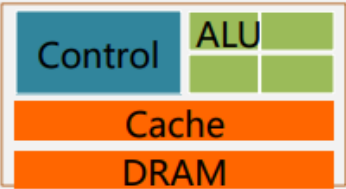

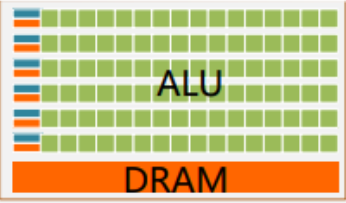

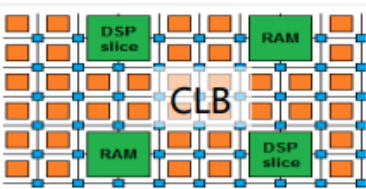

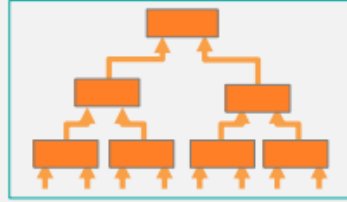

## 什么是异构计算

异构计算（Heterogeneous Computing），又称为异质运算，主要是指使用不同类型指令集和体系架构的计算单元组成系统的计算方式。常见的计算单元类别包括CPU、GPU、DSP、ASIC、FPGA等。

——维基百科



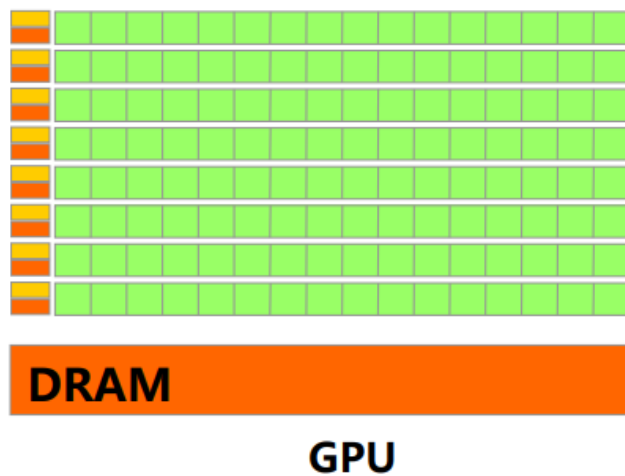
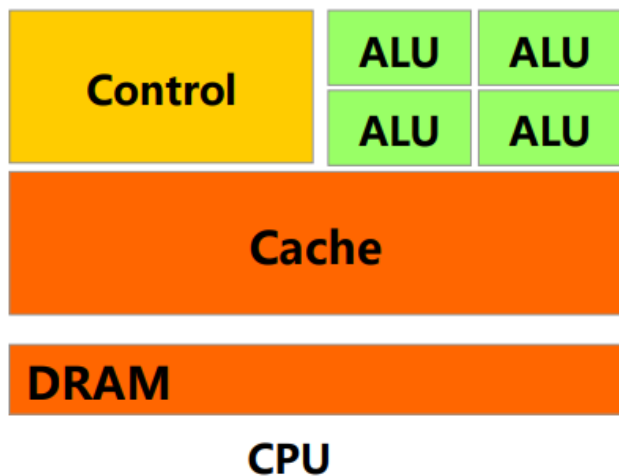
# 计算单元特性对比

	硬件抽象	计算模式	特性	Perf/W
灵活性 ↑ CPU			通用编程，上市时间短。	1x
GPU			适用大批量数据并行计算，高性能，高能耗。	10x
FPGA			适用于不规则并行计算，实时性好，能效比高。	20x
效能 ↓ ASIC			专用电路，高性能，低能耗。	>20x



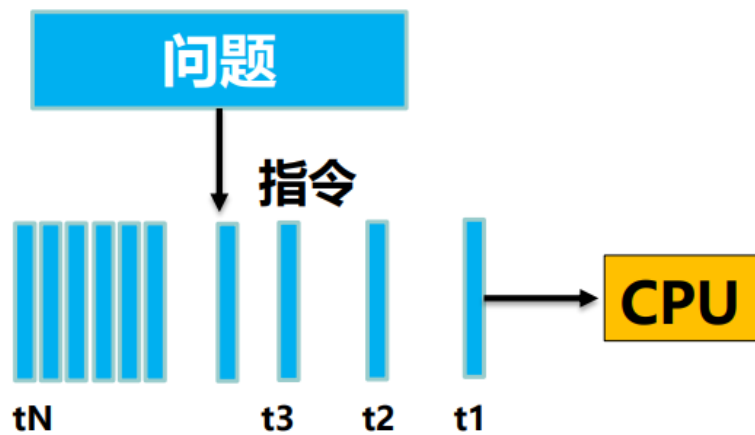
# 什么是GPU

- GPU (Graphics Processing Unit) 是图形处理器又称显示核心、视觉处理器、显示芯片，是一种专门在个人电脑、工作站、游戏机和一些移动设备上运行图像运算工作的微处理器。
- GPGPU (General-Purpose computing on Graphics Processing Units) 即图形处理单元上的通用计算，利用处理图形任务的图形处理器来计算原本由中央处理器处理的通用计算任务。
- GPU与CPU的硬件逻辑架构对比

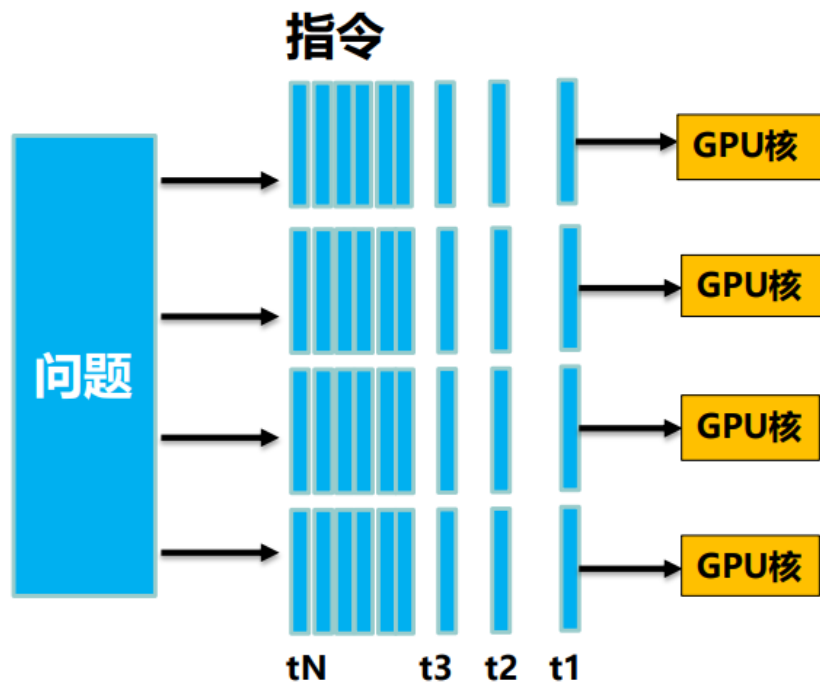




## GPU指令执行模型



CPU指令执行模型



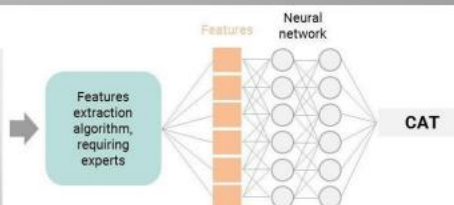
GPU指令执行模型



## GPU 加速应用场景



2006年，多伦多大学教授  
Geoffrey Hinton发表论文开启  
深度学习。



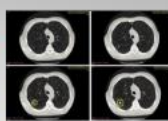
2012年6月，谷歌Brain项目用16,000个CPU  
从10万个Youtube视频截取图片通过深度学  
习学会什么是“猫”。



2016年3月，谷歌AlphaGo以  
4:1战胜李世石。



智能金融



智能医疗



平安城市



智慧交通



机器翻译



语音识别

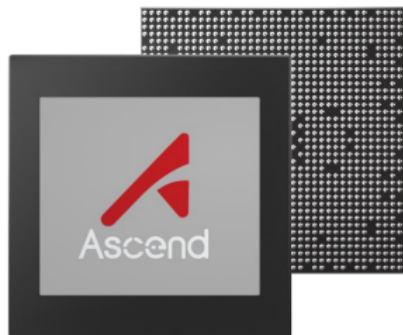


自动驾驶



## 什么是ASIC

- ASIC是一种专用芯片，与传统的通用芯片有一定的差异。是为了某种特定的需求而专门定制的芯片。ASIC芯片的计算能力和计算效率都可以根据算法需要进行定制。
- 优点：体积小、功耗低、计算性能高、计算效率高、芯片出货量越大成本越低。
- 缺点：算法是固定的，一旦算法变化就可能无法使用。





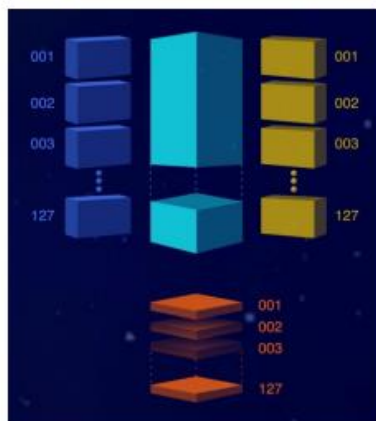
# 华为达芬奇架构

传统处理器：标量计算



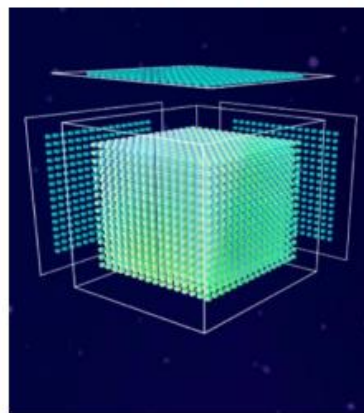
0.003 TOPS/W

传统加速卡：向量计算



0.3 TOPS/W

达芬奇架构：张量计算



3 TOPS/W

华为独创 3D Cube  
16\*16\*16 三维弹性立方体



# 昇腾系列AI处理器



Ascend 310

Ascend-Mini

架构: 达芬奇

半精度 (FP16): 8 TeraFLOPS

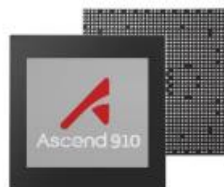
整数精度 (INT8) : 16 TeraOPS

16 通道全高清视频解码器 – H.264/265

1 通道全高清视频编码器 – H.264/265

功耗: 8W

12nm



Ascend 910

Ascend-Max

架构: 达芬奇

半精度 (FP16): 256 TeraFLOPS

整数精度 (INT8): 512 TeraOPS

128 通道全高清视频解码器 – H.264/265

最大功耗: 350W

7nm





## 异构计算总结



CPU

- 主频高，核数有限；
- 逻辑控制&算术运算单元、大量缓存；
- 管理和调度任务。



GPU

- 高并发：几千核并行；
- 强浮点能力；
- 高显存带宽：芯片内置HBM存储。



FPGA

- 主频低但集成大量计算单元；
- 流水线并行和数据并行；
- 硬件编程和加速，特定应用IP核。



ASIC

- 针对某一场景优化的专用处理单元；
- 硬件基本不可编程；
- 多个IP集成，高性价比和能效比。



# *Computer Organization*



# Thank You !

系统结构研究所