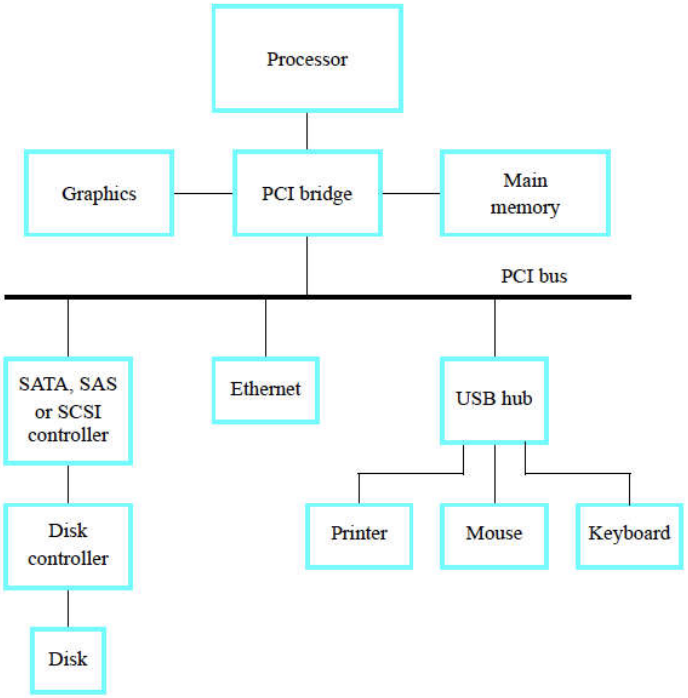
计算机组成原理

第五章 输入输出系统

llxx@ustc.edu.cn
wjluo@ustc.edu.cn

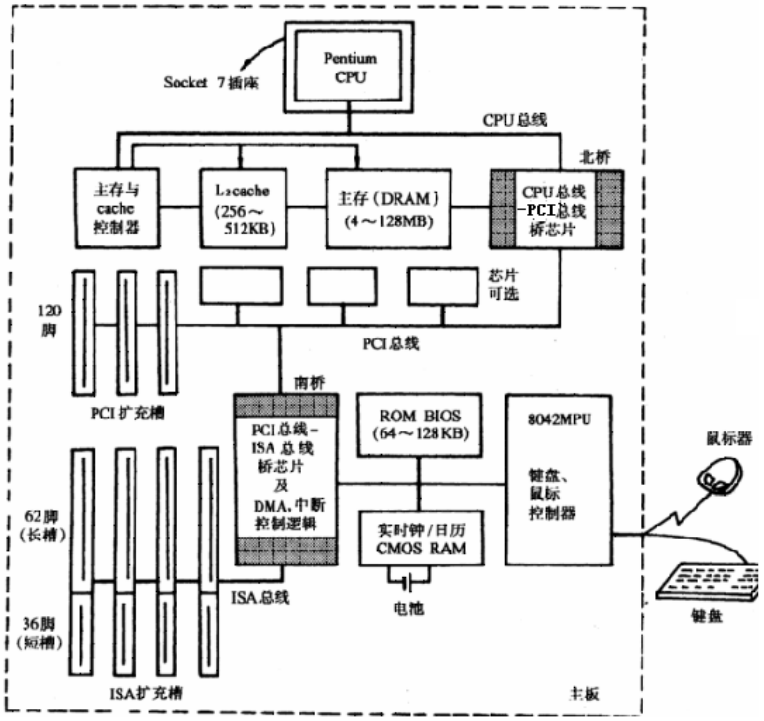
本章内容



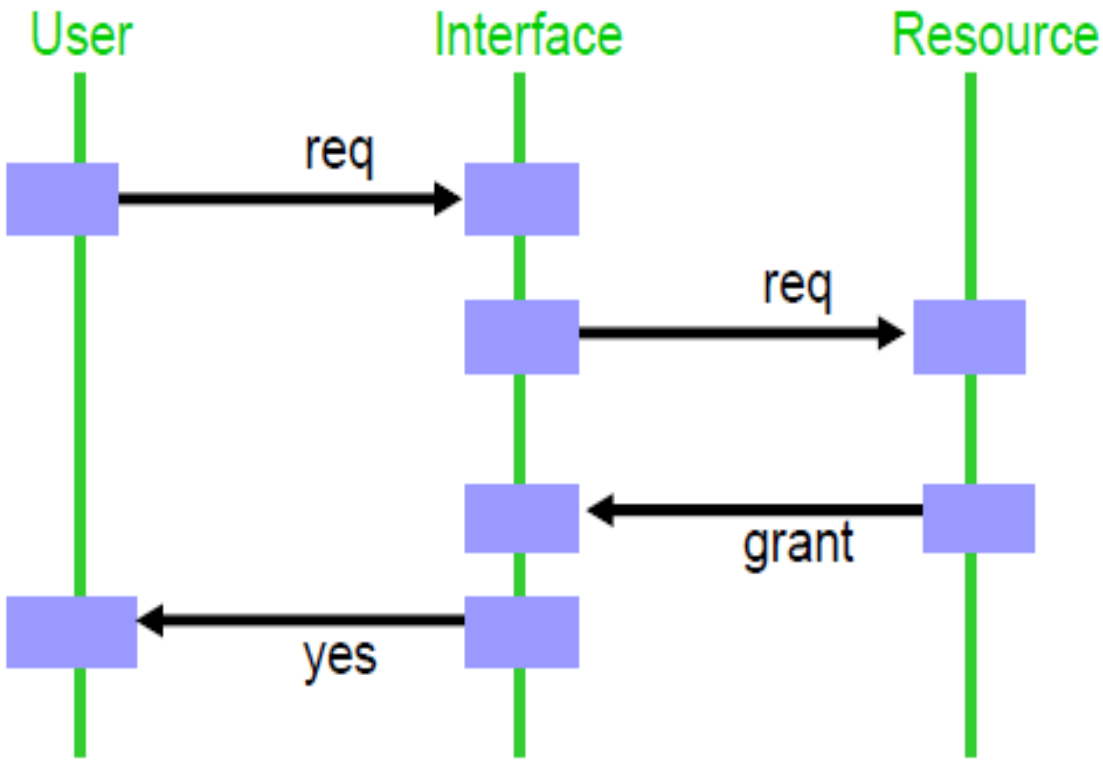
- **I/O系统的工作原理**
  - **I/O系统的构成**
  - **数据传输方式：“通信协议”**
    - 编址方式
    - 数据传送方式
    - 传输同步方式
    - 数据传输控制方式：程序查询、中断、DMA
    - …
- **I/O设备的工作原理**
  - **键盘、显示器、打印机等**

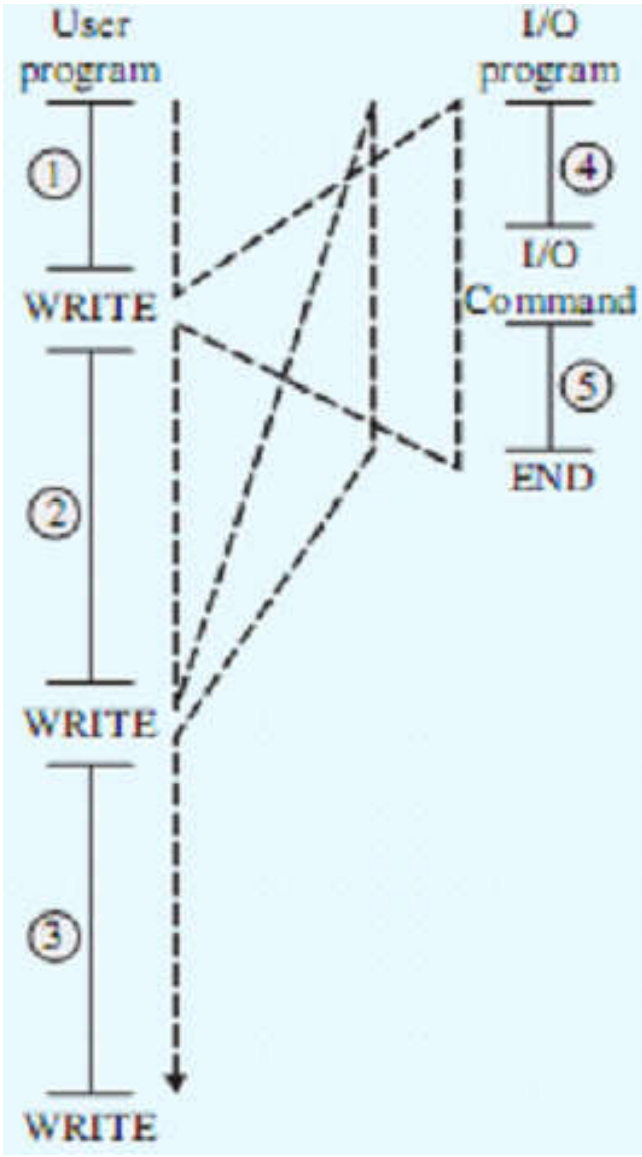
I/O：getchar()，putchar()？

```
#include <stdio.h>
int main(void)
{   int ch;

    printf("Input a character:");

    /* read a character from
    the standard input stream */
    ch = getchar();
     putchar(ch);

    return 0;
}
```



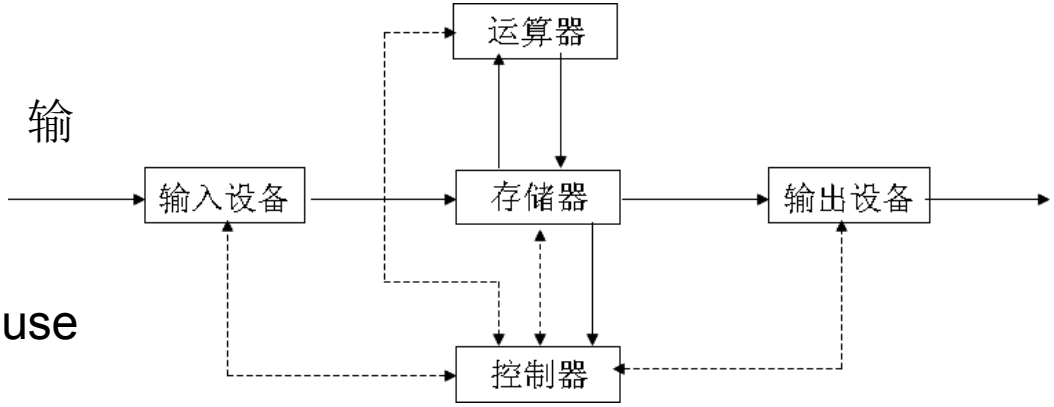
Input/Output：Interactive，Reactive

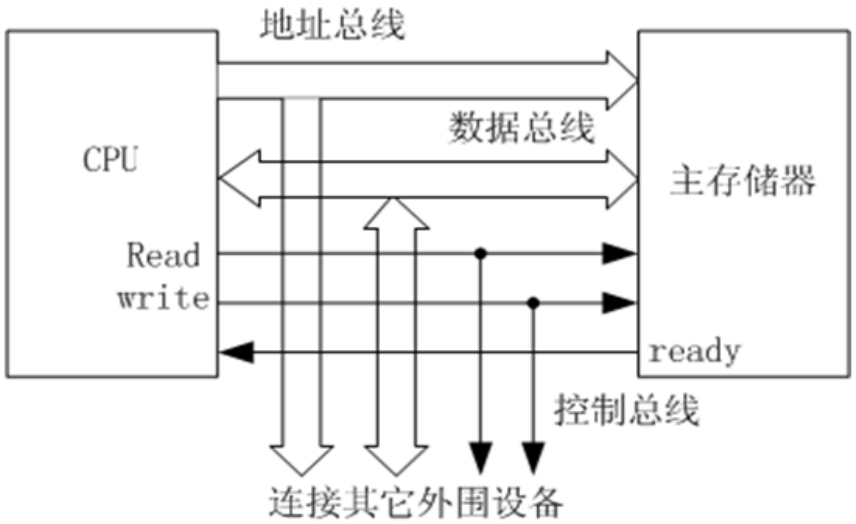




I/O的本质：内存与外设间数据交换

- **现代计算机组成设备**
  - **Von Neumann系统的组成**
    - 运算器、控制器、存储器、输入设备、输出设备
  - **Peripheral device**
    - 输入设备：keyboard、mouse、touchscreen、scanner、digital camera、microphone
    - 输出设备：displayer、printer
  - **Connectivity**
    - Network
  - **以存储器为中心的体系**
- **如何访问I/O设备？**
  - **组成（软件、硬件、接口）**
  - **过程控制**

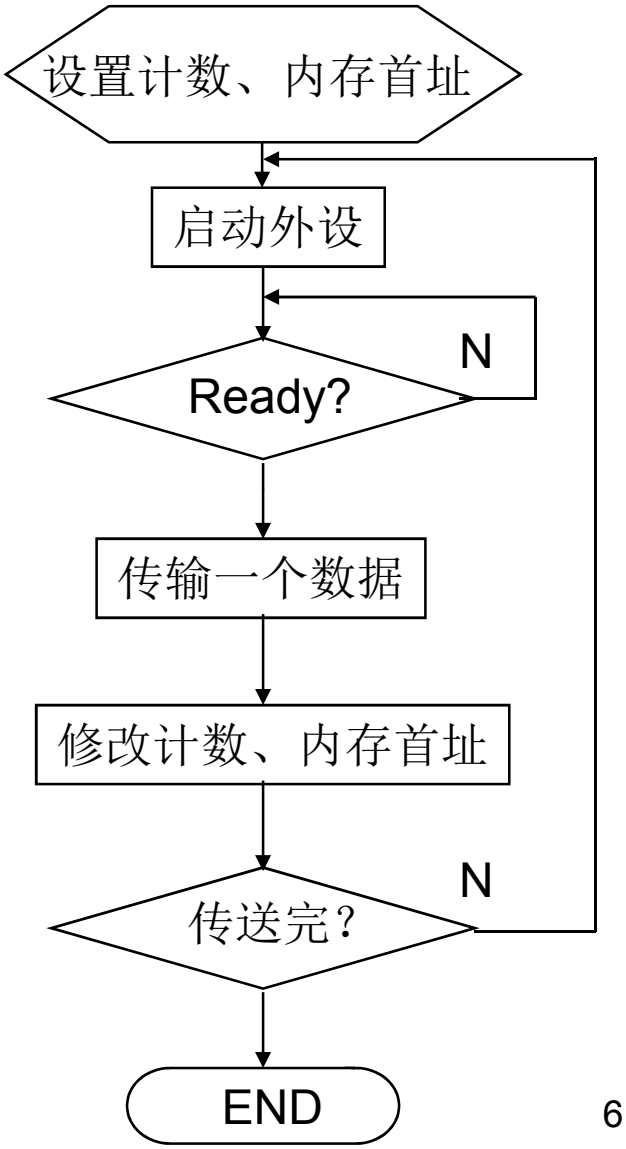




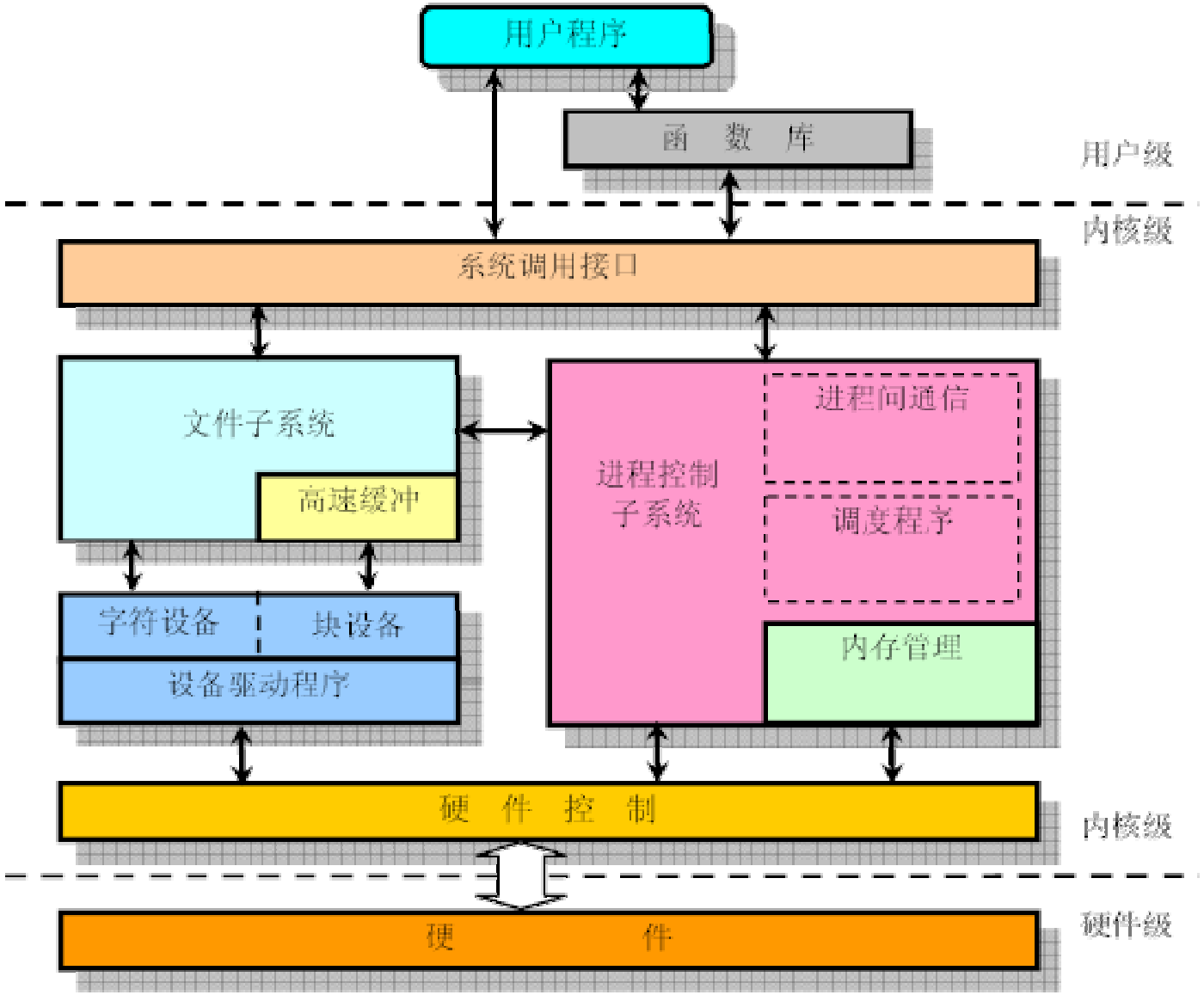
I/O系统组成

- **软件：**
  - **软件的主要任务：**
    - 将数据输入至主机
    - 将运算结果输出给用户
    - 实现I/O系统和主机协同工作
  - **应用软件**
  - **操作系统**
  - **设备驱动程序**

- **硬件：种类多样化。**

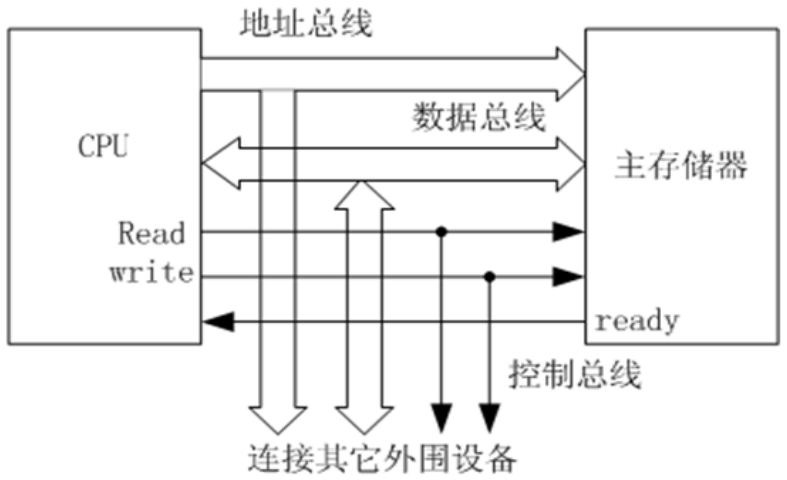


基于Linux的计算机系统组成

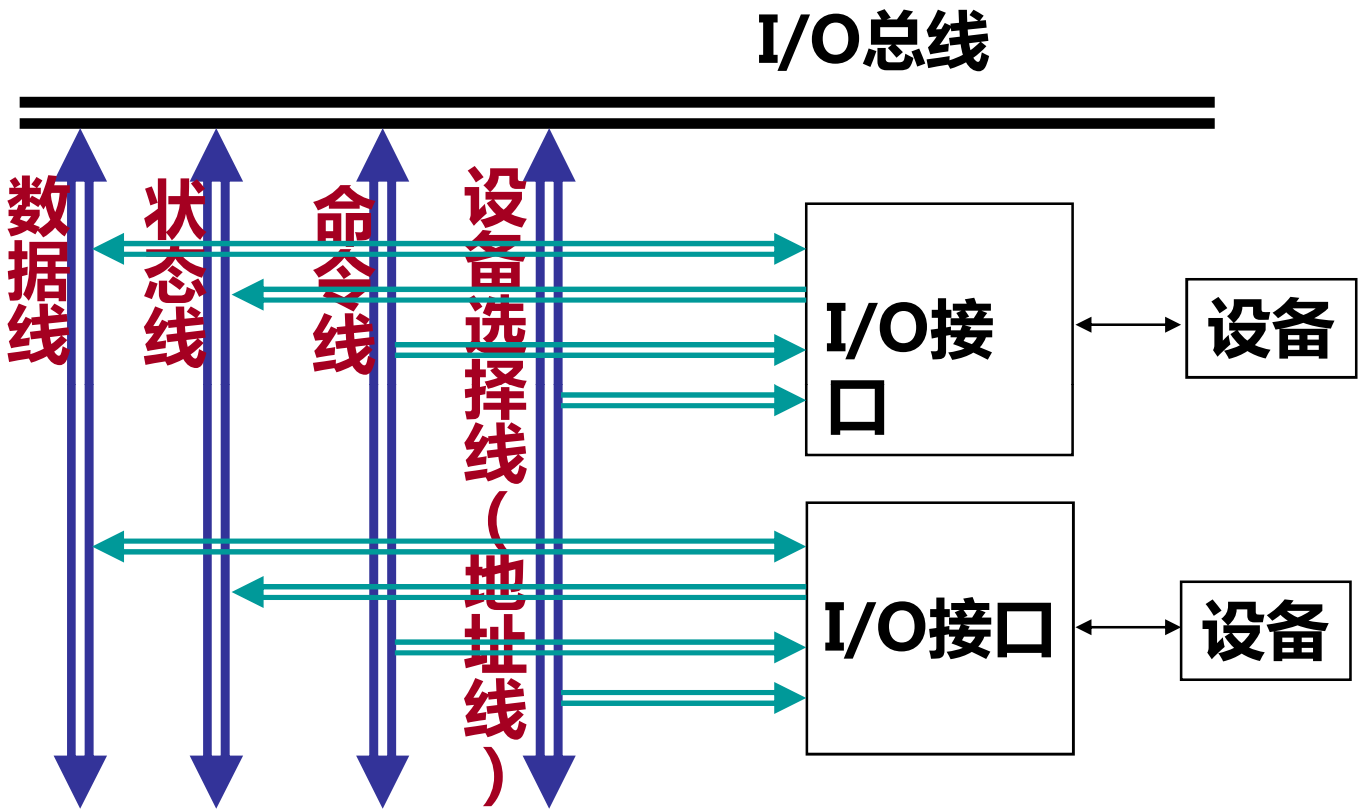


I/O接口的概念

- “接口”：两个系统或两个部件之间的交接部分
  - 硬件接口：连接电路
  - 软件接口：逻辑边界（数据结构）
- I/O接口
  - 指主机与外设之间设置的硬件电路及相应的软件控制
- 设置I/O接口的理由：模块化、标准化
  1. 电平转换
  2. 设备选择
  3. 数据缓冲
  4. 在CPU和外设之间传送命令
  5. 监视设备状态(错误处理)

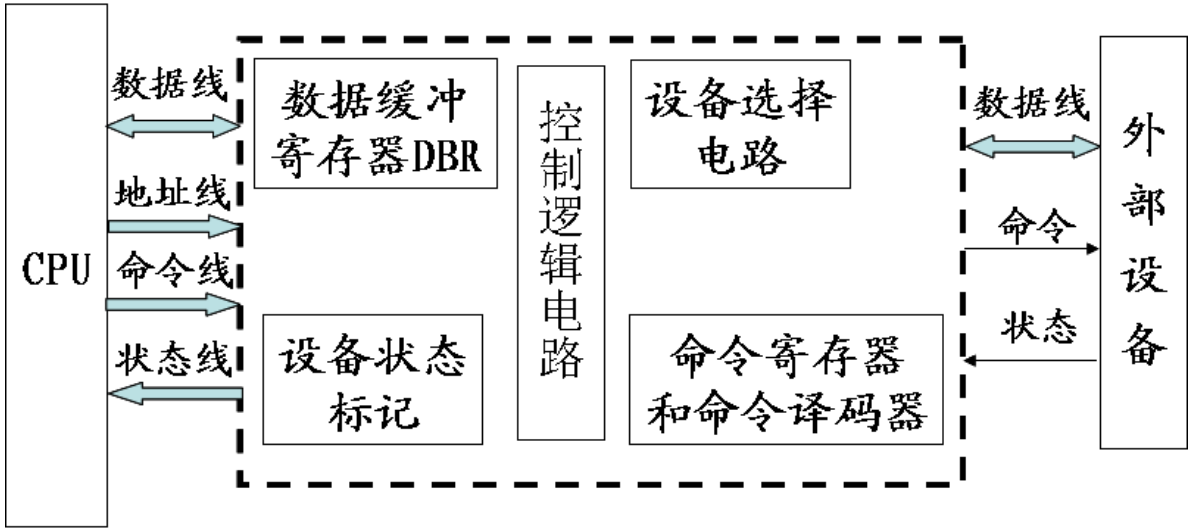


总线连接方式的I/O接口电路



I/O接口的功能与组成

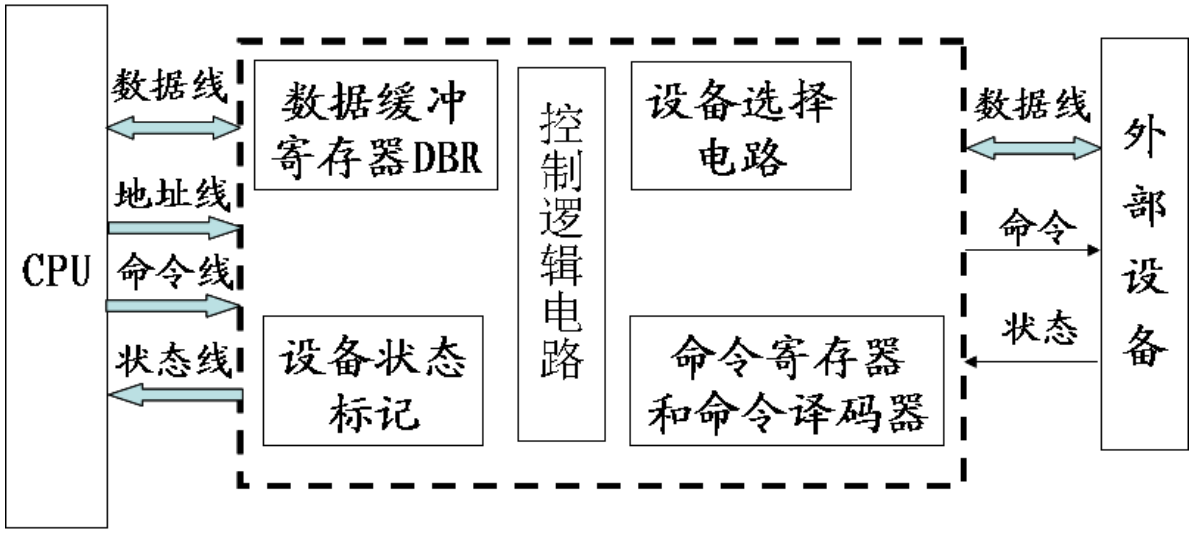
1. 选址（设备选择）
2. 传送命令
   - 命令寄存器，命令译码器
3. 传送数据
   - 数据缓冲寄存器
4. 反映I/O设备工作状态
   - 暂停、准备就绪、正在准备等状态



接口（interface）和端口（port）

- 端口 = 寄存器组（数据、控制、状态…）

- 接口 = N个端口 + 控制逻辑

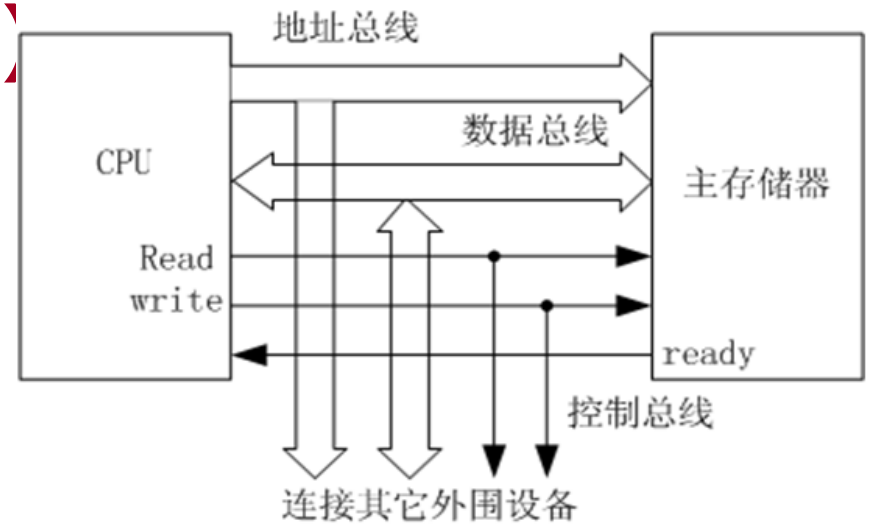


接口的类型

- 按数据传送方式分，有并行接口和串行接口。
  - 并行接口：一个字节或一个字的所有位同时传送，如Intel 8255。
  - 串行接口：一位一位传送，如Intel 8251。
- 按功能选择的灵活性分，有可编程接口和不可编程接口。
  - 可编程接口：可用程序来改变或选择接口的功能和操作方式(如Intel 8255、Intel 8251)。
  - 不可编程接口：不能用程序来改变其功能，但可通过硬连线路逻辑来实现不同的功能(如并行接口芯片Intel 8212)
- 按通用性分类，有通用接口和专用接口。
  - 通用接口：可供多种外设使用，如Intel 8255、8212。
  - 专用接口：为某类外设或某种用途专门设计的，如Intel 8279可编程键盘/显示器接口；Intel 8275可编程CRT控制器接口等。
- 按数据传送的控制方式分类，有程序型接口和DMA式接口
  - 程序型式接口：用于连接速度较慢的设备，如键盘、打印机等，如Intel 8259。
  - DMA式接口：用于连接高速I/O设备，如磁盘，常用Intel 8237。

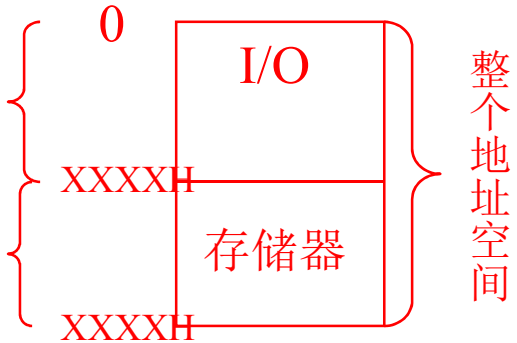
I/O设备与主机的联系方式

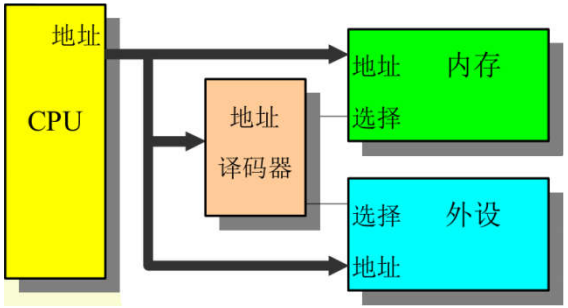
1. I/O端口的编址方式
2. 设备寻址：每台设备都有一个设备号，I/O指令的设备码字段直接指出设备号。
3. 数据传送方式（串行 / 并行）
4. 同步方式（联络方式）
5. 连接方式（拓扑结构）

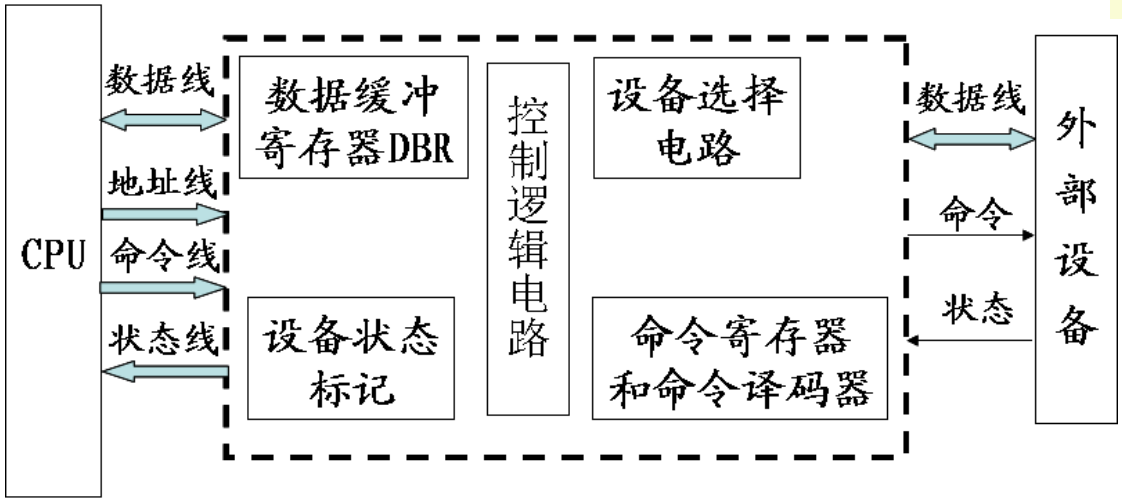


I/O编址方式—统一编址

- **存储器映射方式**
  - **在主存储器的地址空间中划出某一区域专门作为外设地址区使用**
    - 外设寄存器的地址包含在主存储器的地址空间内。
    - 划给外设的这部分区域不能配置存储器芯片
    - 使用通用的MOV或访存指令也可以访问I/O接口。
  - **需占用小部分存储空间。**
  - Intel MCS-51、MIPS、ARM等采用

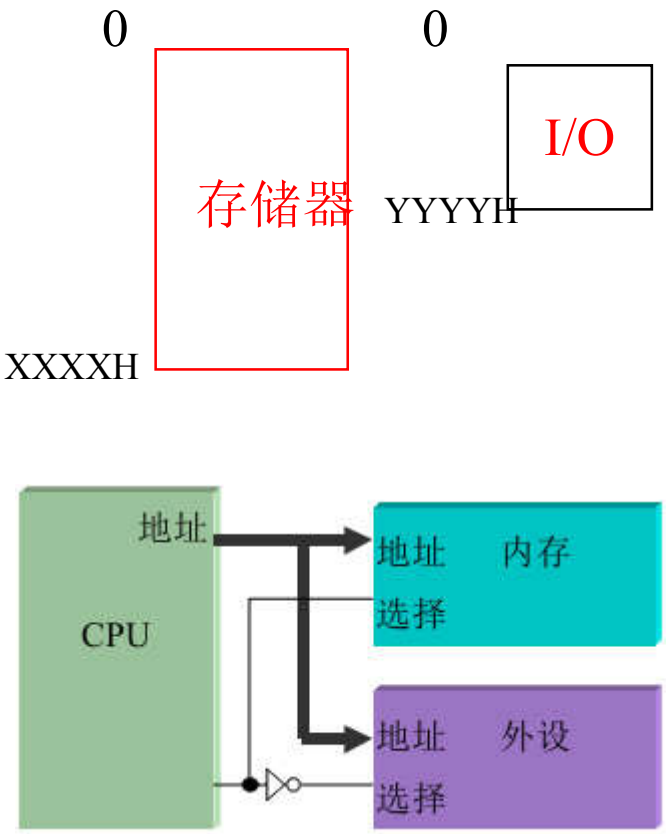




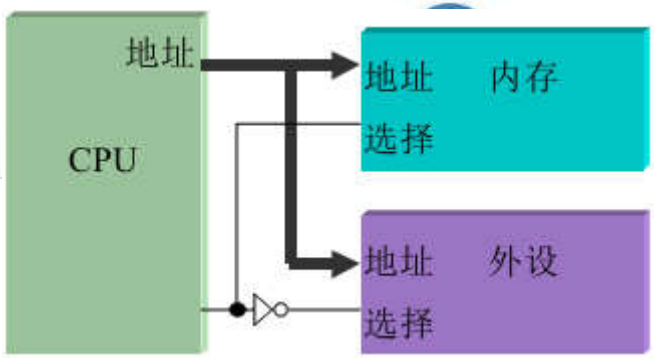


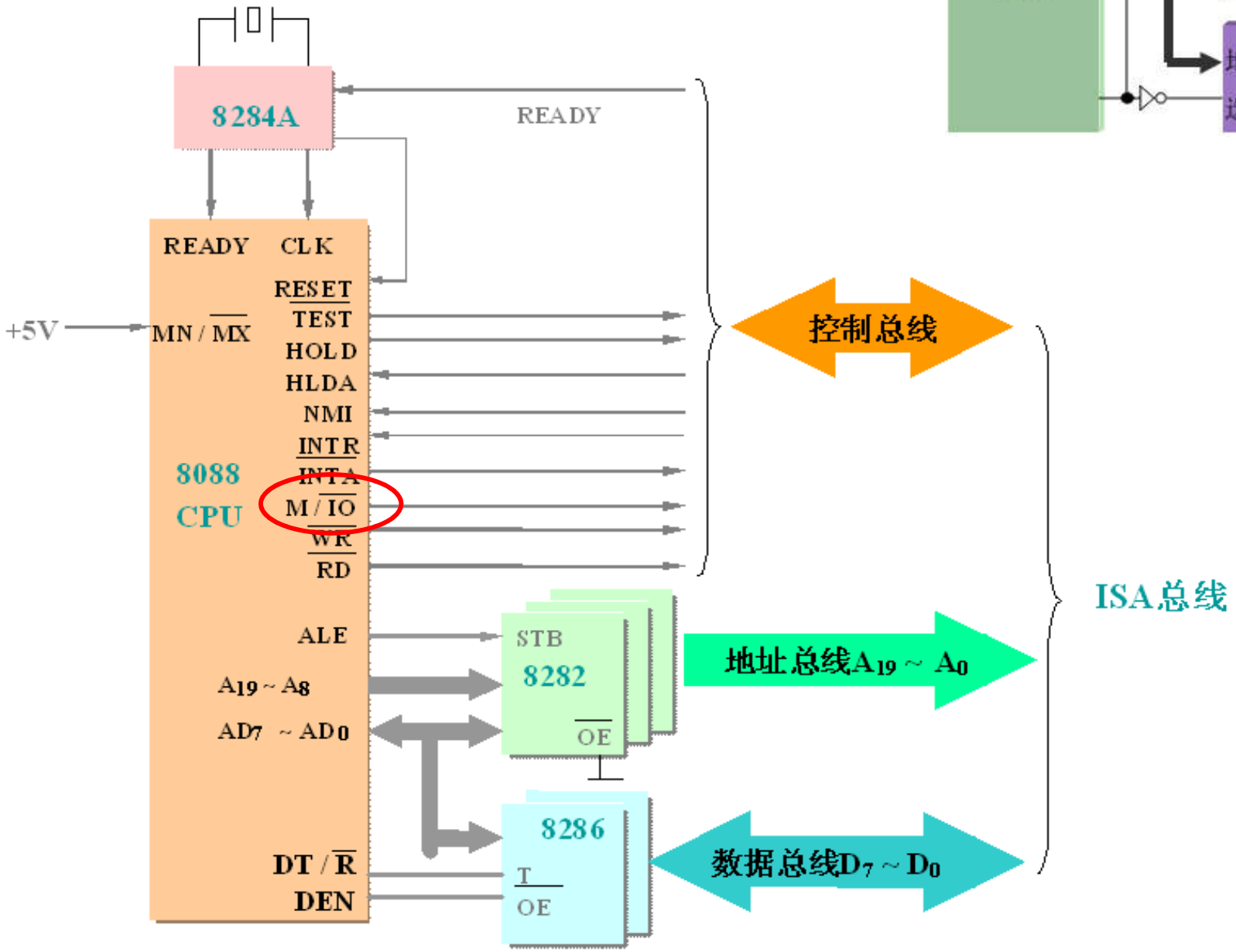
I/O编址方式—独立编址

- **I/O端口和存储器分开编址（I/O Mapped I/O）**
  - **指令系统中分别设立面向存储器的指令和面向I/O操作的指令（IN指令和OUT指令）。**
- **优点：不占用主存空间**
- **缺点：需专门的I/O指令，其寻址方式较简单，编程灵活性稍差。**
- 80x86采用I/O端口独立编址



独立编址I/O寻址


地址
CPU
地址
选择
内存
地址
选择
外设


8284A
READY
READY
CLK
RESET
TEST
HOLD
HLDA
NMI
INTR
INTA
M / IO
WR
RD
+5V
MN / MX
8088
CPU
ALE
A19 ~ A8
AD7 ~ AD0
DT / R
DEN
STB
8282
OE
8286
T
OE
控制总线
地址总线A19 ~ A0
数据总线D7 ~ D0
ISA总线

同步方式

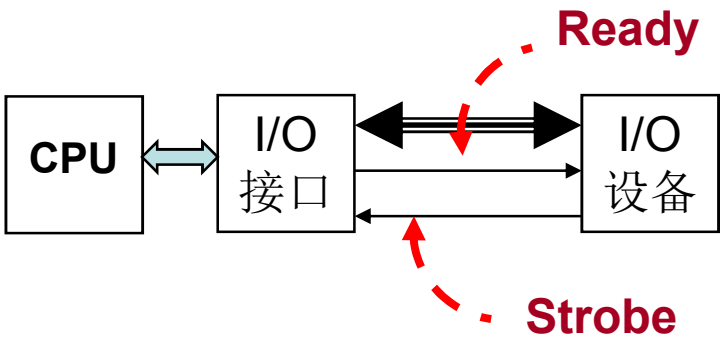
- **主机较外设速度快，必须同步**

1. 立即响应方式：

- 外设处于等待状态，CPU的I/O指令一到，立即响应
  - 如指示灯的亮与灭。

2. 异步方式： Handshaking protocol

- 应答式（acknowledge）
  - Ready：就绪
  - Strobe：选通
- 速度慢

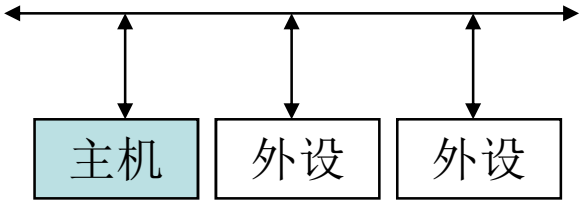


3. 同步式：主机与外设采用同步时标

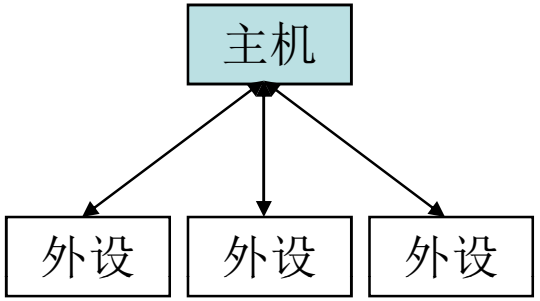
- 并行，速度快

外设与主机的连接方式

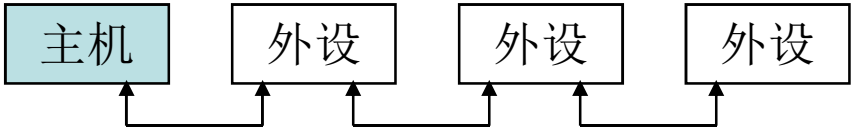
- **拓扑结构(topology)**



总线型(BUS)－IDE硬盘



星型，又叫辐射式－USB



级联型(Cascade)－SCSI

数据传输控制方式

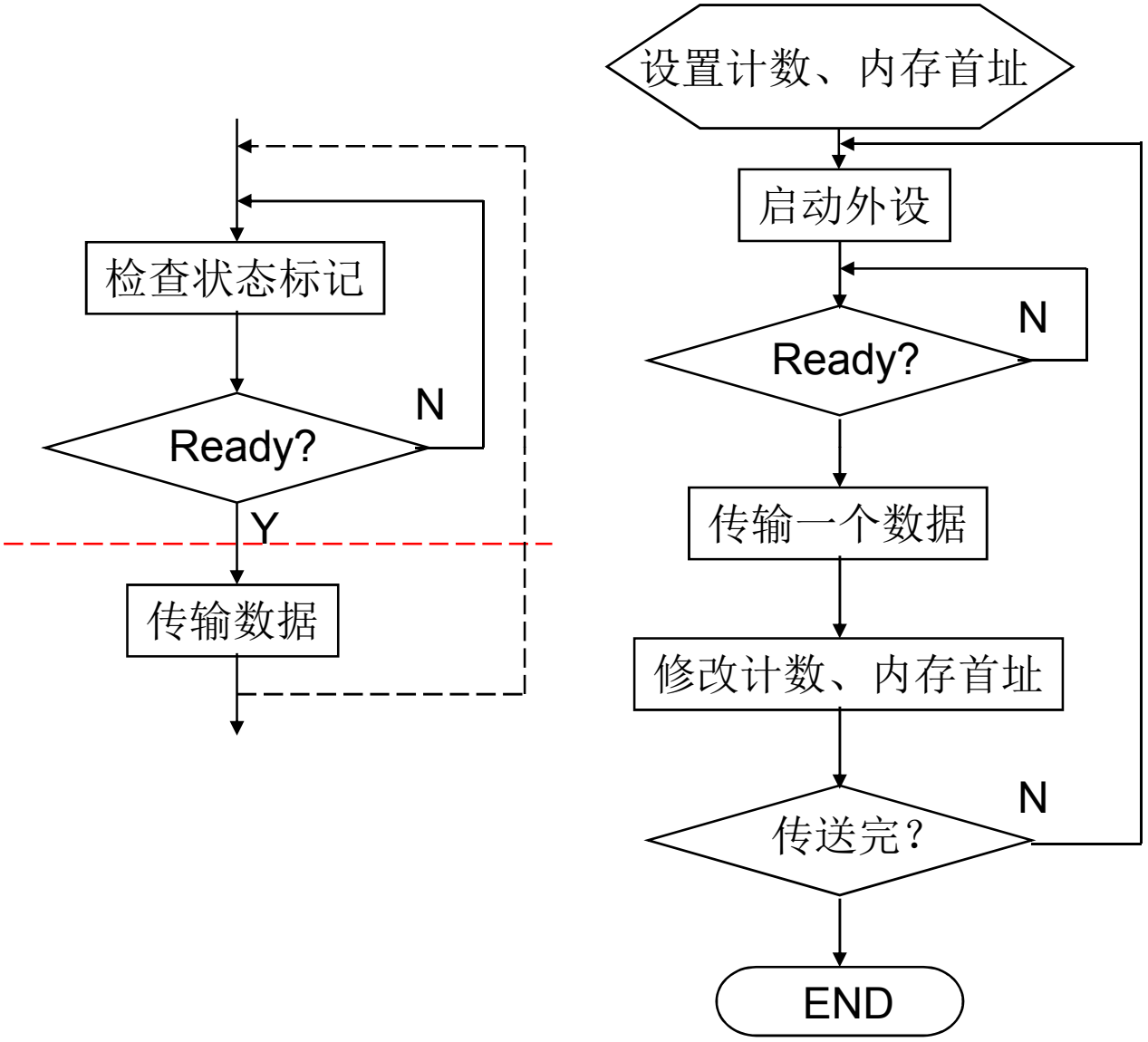
1. **程序查询方式**
2. **程序中断方式**

} CPU控制传递过程

3. **直接内存访问方式（Direct Memory Access）**

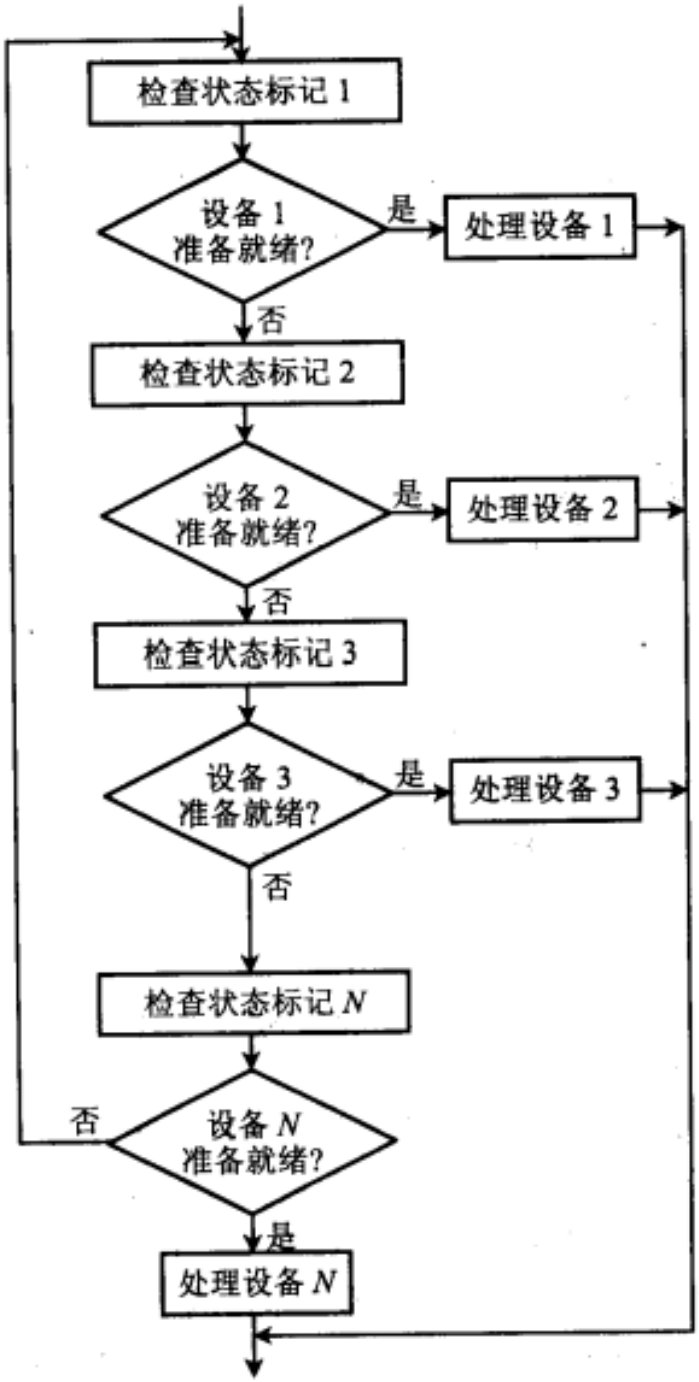
5.4 程序查询方式

程序查询方式



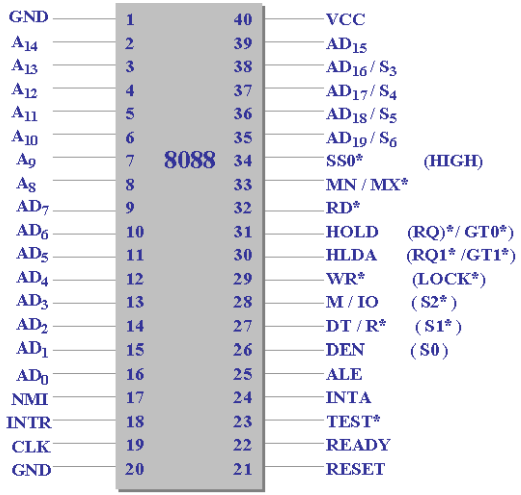
- 由CPU控制数据传输的过程
- 处理器等待，效率低

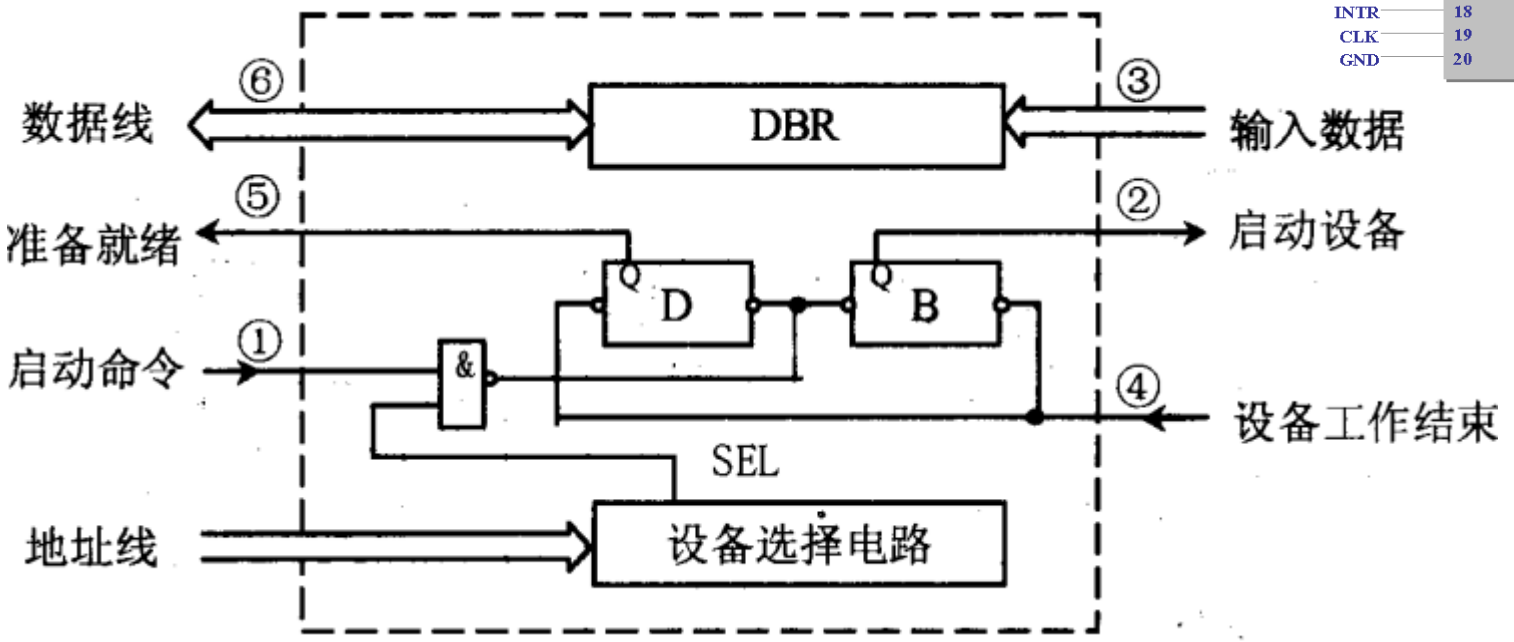
多个设备的查询流程示意



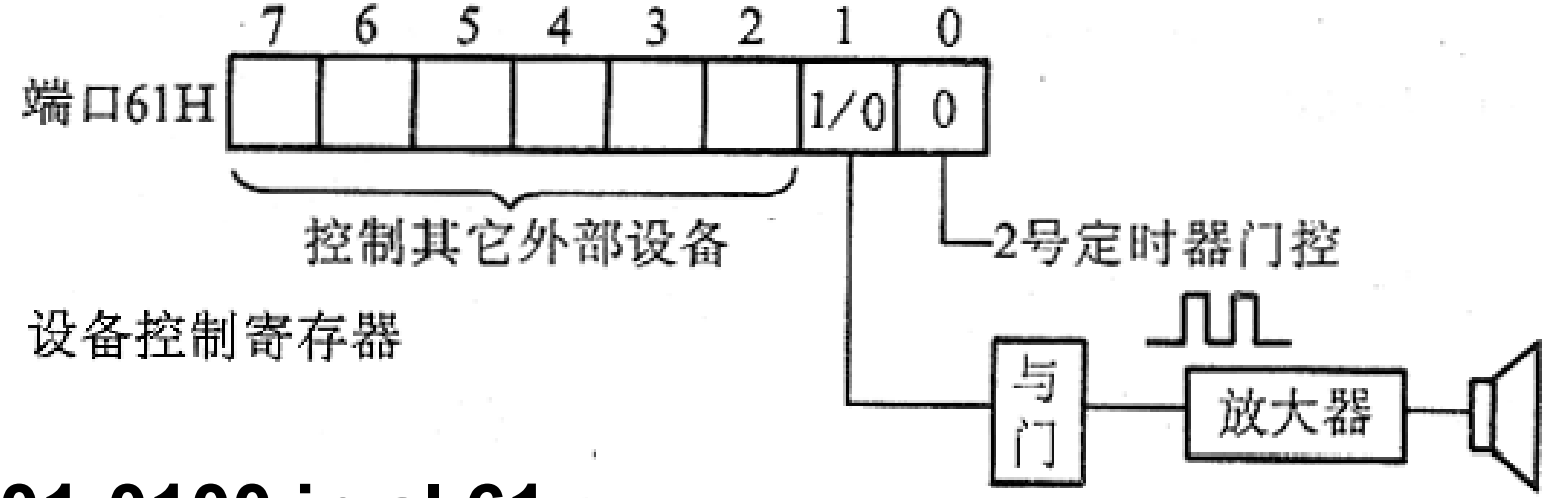
程序查询方式的接口电路

- DBR：数据缓冲器，用于存放欲传送的数据。
- D为完成触发器，B为工作触发器。





例：发声程序设计



0B01:0100 in al,61

0B01:0102 xor al,2

0B01:0104 out 61,al

0B01:0106 mov cx,400

0B01:0109 loop 109

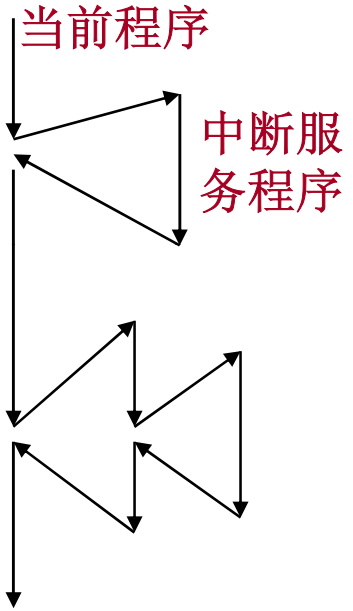
0B01:010B jmp 100

5.5 程序中断方式

（§5.5、§8.4）

中断概念

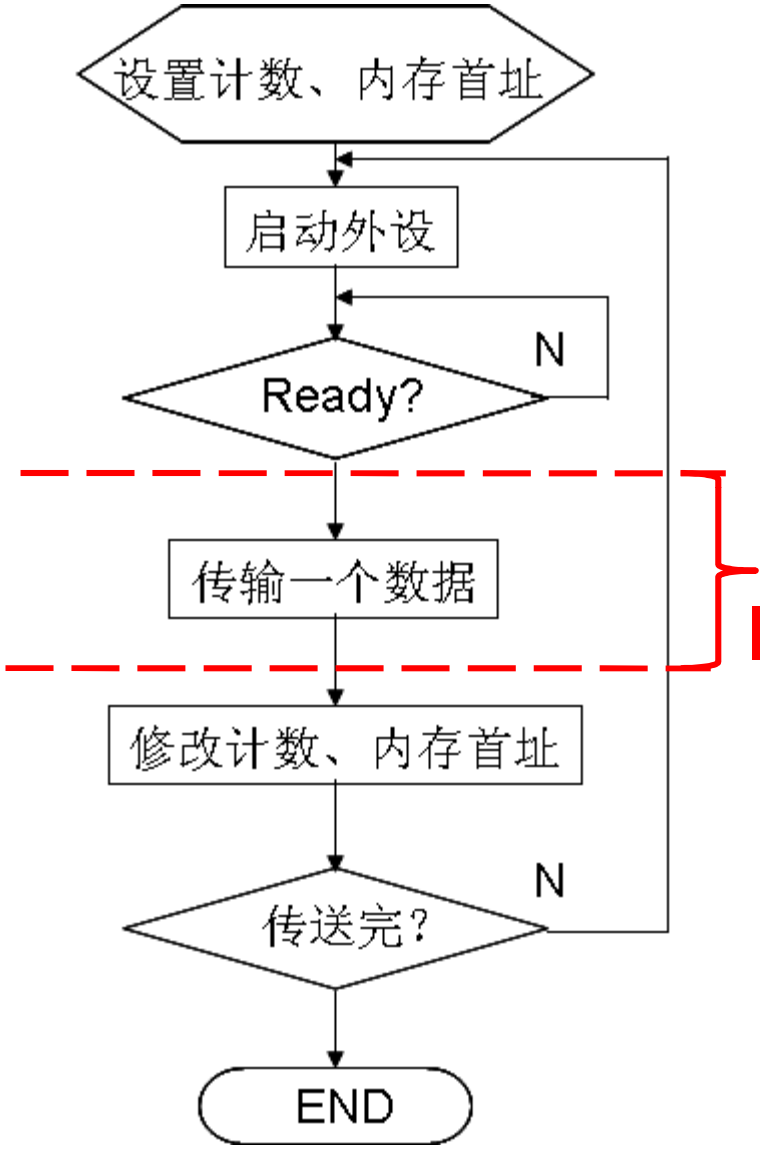
- 中断的概念
  - 暂停当前程序的执行，转而执行其他程序，在它们执行完成后再恢复被中断程序的执行。
  - 允许一个处理器“同时”执行多个任务
  - 中断服务程序（ISR）



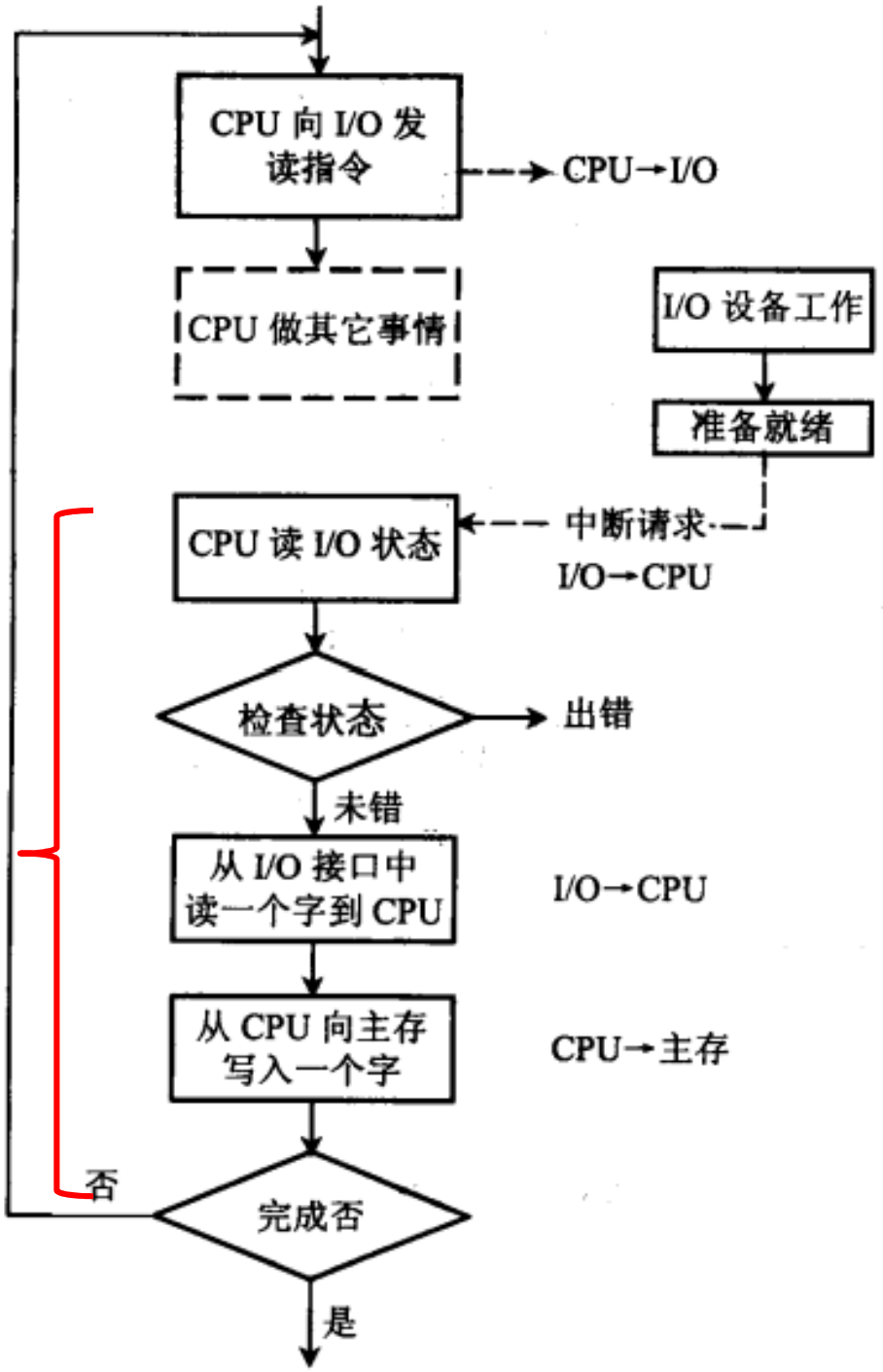
- 中断的产生
  - 为了提高计算机的整机效率
  - 为了应付突发事件
  - 为了实时控制的需要

- 为了实现中断，计算机系统中必须有相应的中断系统或中断机构。

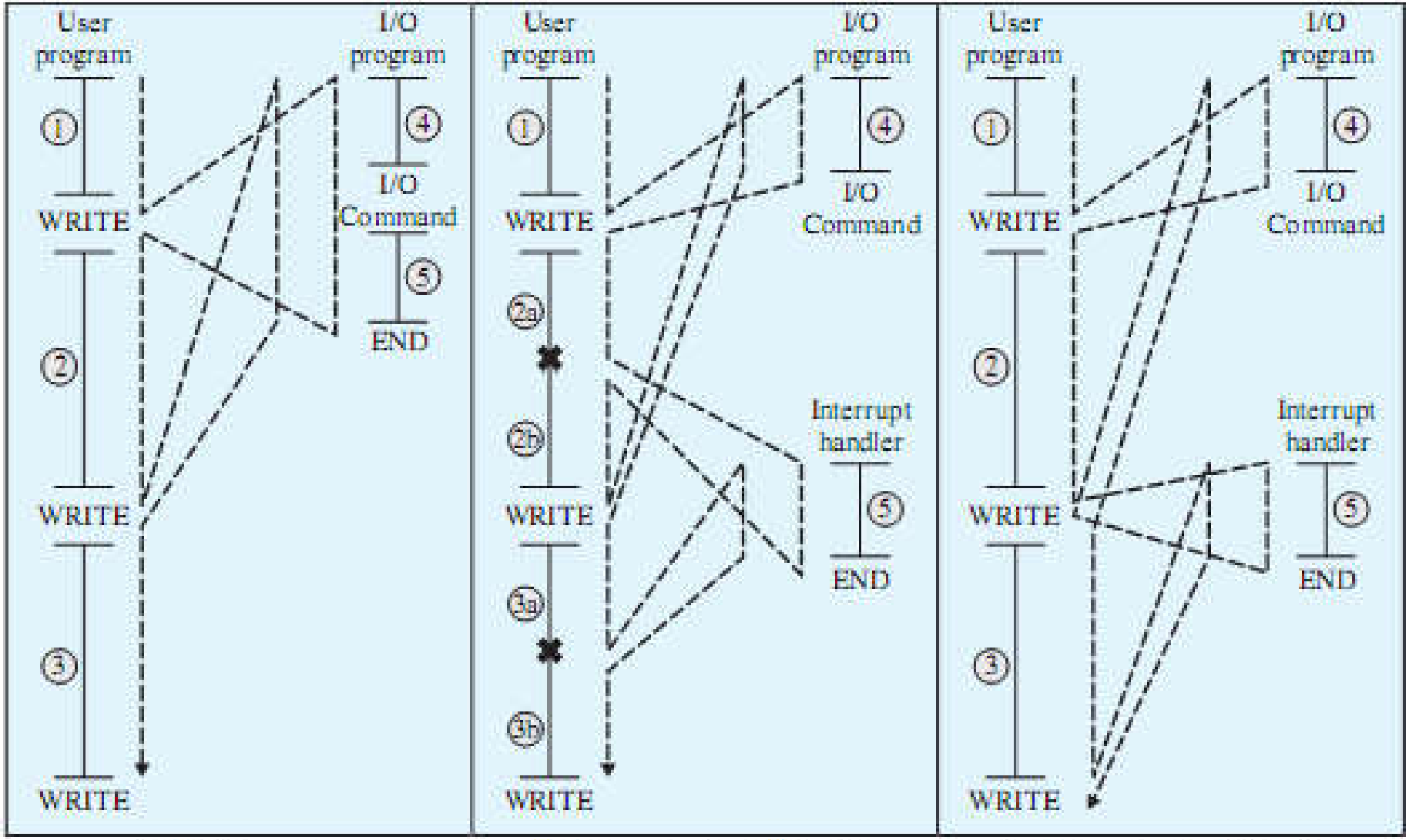
中断I/O过程


设置计数、内存首址
启动外设
N
Ready?
传输一个数据
修改计数、内存首址
N
传送完?
END

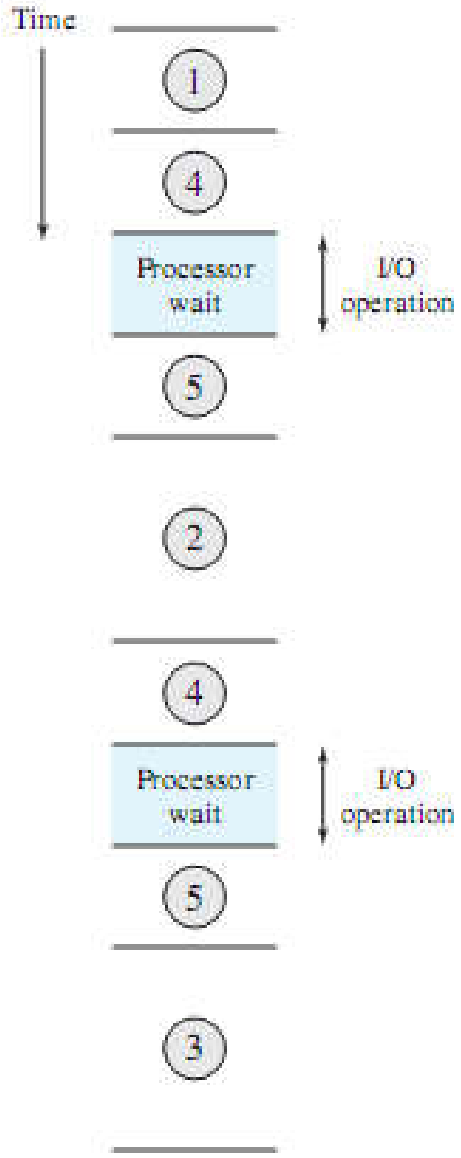
ISR


CPU 向 I/O 发读指令
CPU→I/O
CPU 做其它事情
I/O 设备工作
准备就绪
CPU 读 I/O 状态
中断请求
I/O→CPU
检查状态
出错
未错
从 I/O 接口中读一个字到 CPU
I/O→CPU
从 CPU 向主存写入一个字
CPU→主存
否
完成否
是

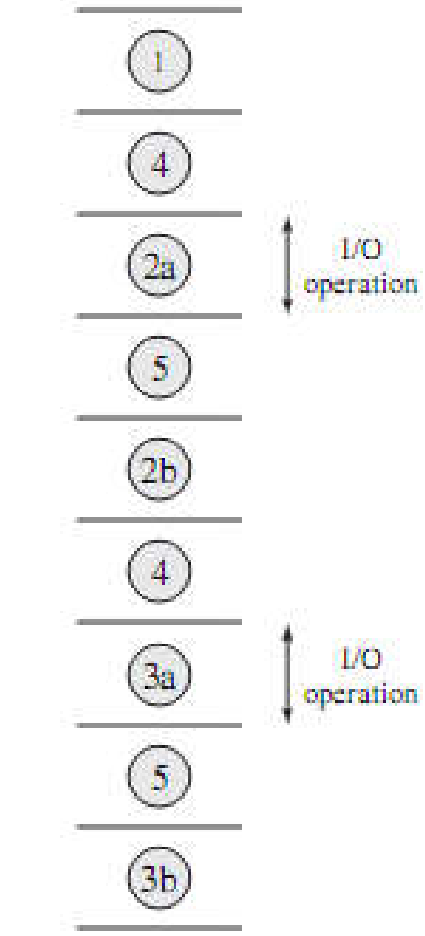
Program Flow of Control without and with Interrupts



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

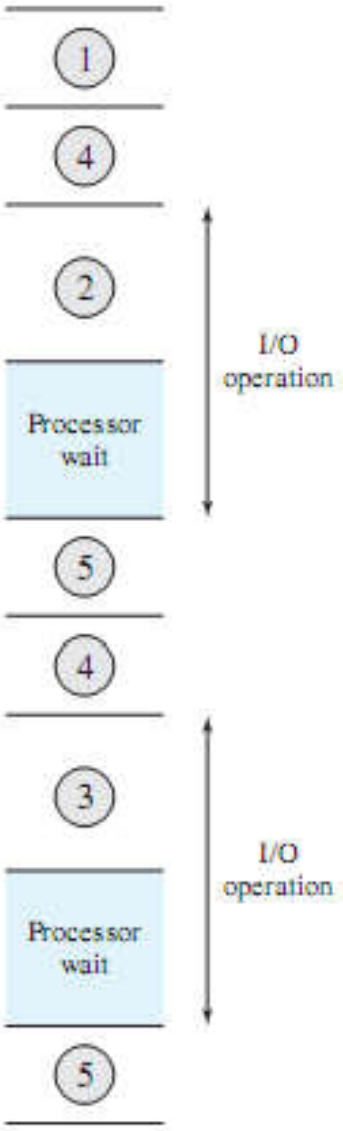


(a) Without interrupts
(circled numbers refer
to numbers in Figure 1.5a)



(b) With interrupts
(circled numbers refer
to numbers in Figure 1.5b)

Short I/O Wait



(b) With interrupts
(circled numbers refer
to numbers in Figure 1.5c)

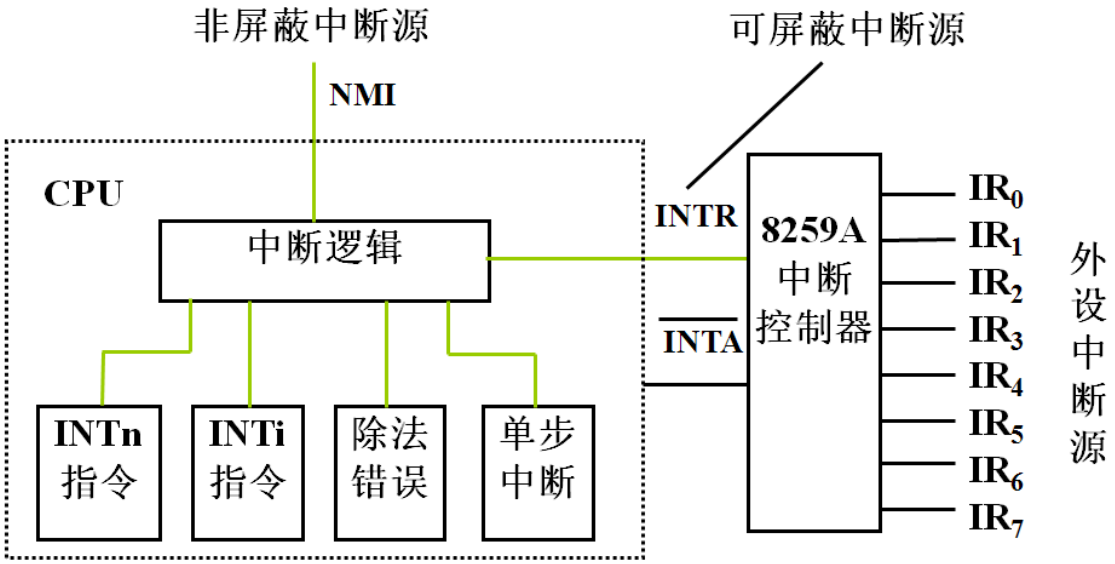
Long I/O Wait

中断I/O用途

- CPU与I/O设备并行工作
  - 键盘输入响应，打印机输出
- 硬件故障处理：故障-->中断-->自动恢复
  - 掉电后自动保存当前状态
- 人机通信
  - 随机干预机器工作，如死循环处理（Ctrl - Break）
- 多任务切换
  - 时钟中断
- 实时处理
  - 对随机事件的快速响应
- 多处理机通信
  - CPU与FPU

中断分类

- 内部中断，外部中断
- 硬中断，软中断（指令）
- 可屏蔽中断、不可屏蔽中断（NMI）
- 中断、异常、陷阱
- 精确中断，不精确中断



中断系统需解决的问题

1. 如何确定中断源
2. 出现多个中断时，中断响应的顺序
3. 中断响应的条件、时机
   - 条件：处理器允许中断、该中断未被屏蔽
   - 时机：指令周期结束检查是否有中断请求
4. 现场保护与恢复
5. 中断服务程序的入口地址
6. 中断处理过程中出现中断如何处理？

- **中断控制器**

中断请求标记（中断源识别）

- 中断请求触发器（INTR）：用于保存各中断源的请求
  - 每个中断源一位
  - 可以在CPU内部，也可以在外部（8259）

1	2	3	4	5			n
掉电	过热	内存读写校验错	阶上溢	非法除法		光电输入机	打印输出机

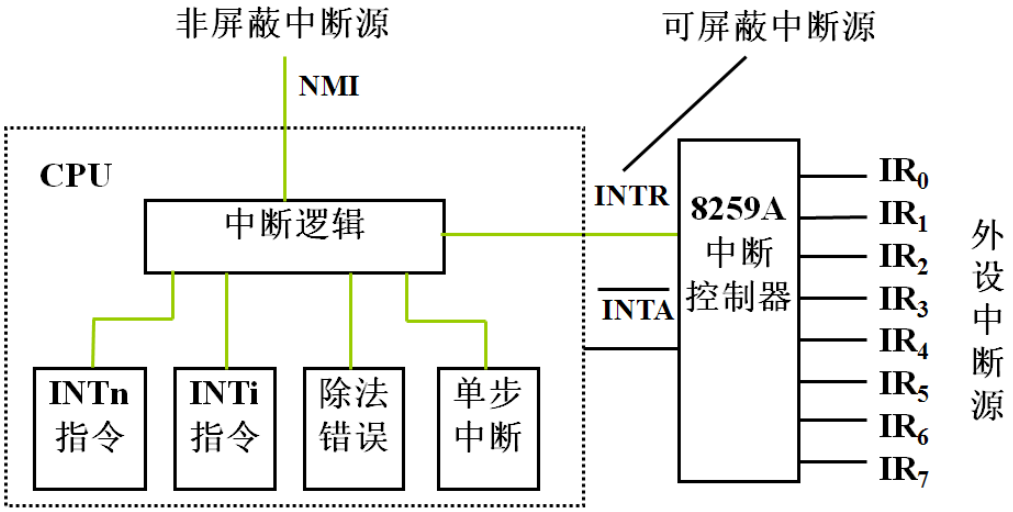
中断请求标记寄存器

中断判优逻辑

- **中断优先级（priority）**
  - **各个中断源的优先顺序是根据该中断若得不到及时响应，致使机器工作出错的严重程度而定的。**
  - **8086/8088 除软件中断外，内部“非屏蔽中断”、“可屏蔽中断”均设立有优先级**
    - 内部：溢出、断点、单步、除0
    - 内部中断（除单步外）的优先级高于非屏蔽中断，非屏蔽中断高于可屏蔽中断，单步中断优先级最低
- **中断判优可用硬件实现，也可用软件实现。**
  - **硬件排队**
  - **软件排队**

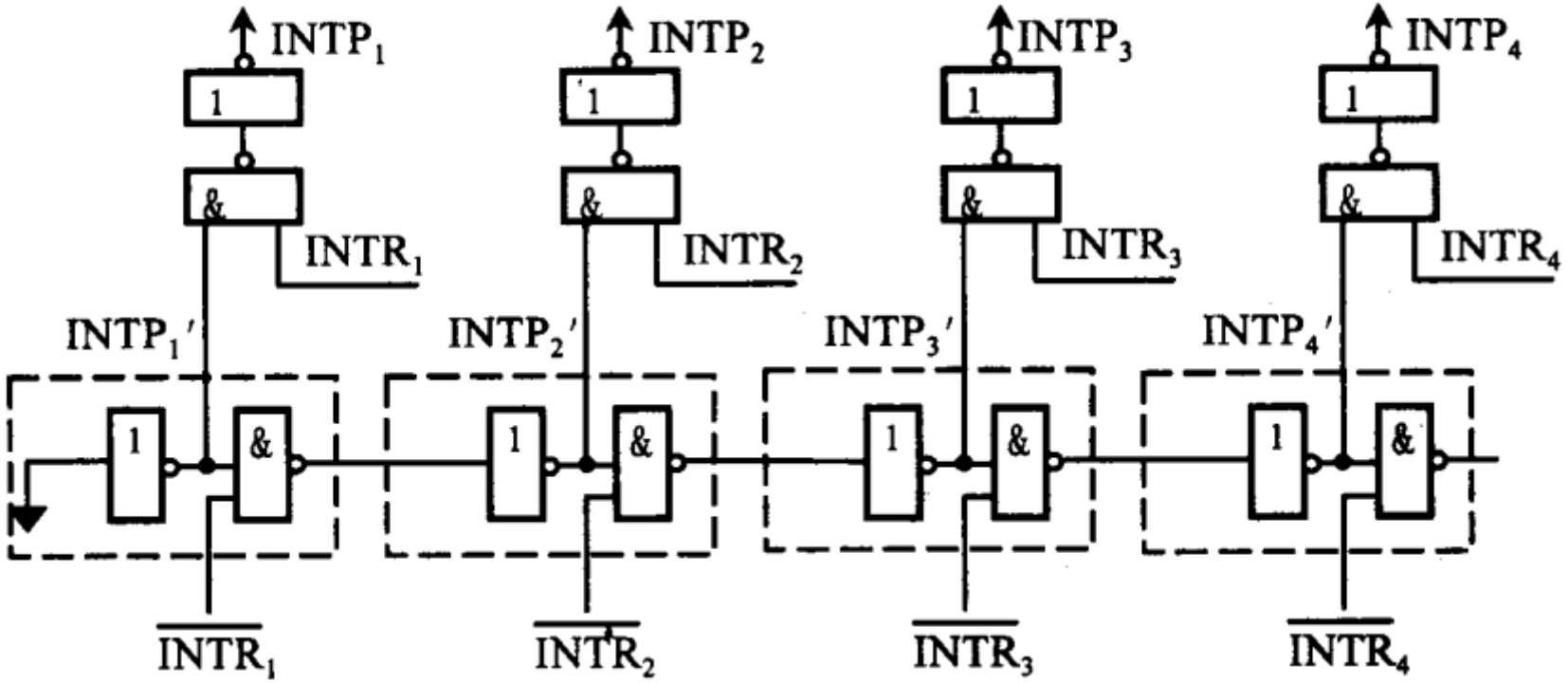
中断判优逻辑—硬件排队

- **硬件排队分两种情况**
  - **第一种，链式排队器**
    - 对应中断请求触发器分散在各个接口电路中的情况。
  - **第二种，集中式**
    - 将排队器设在CPU内（或中断控制器中）。



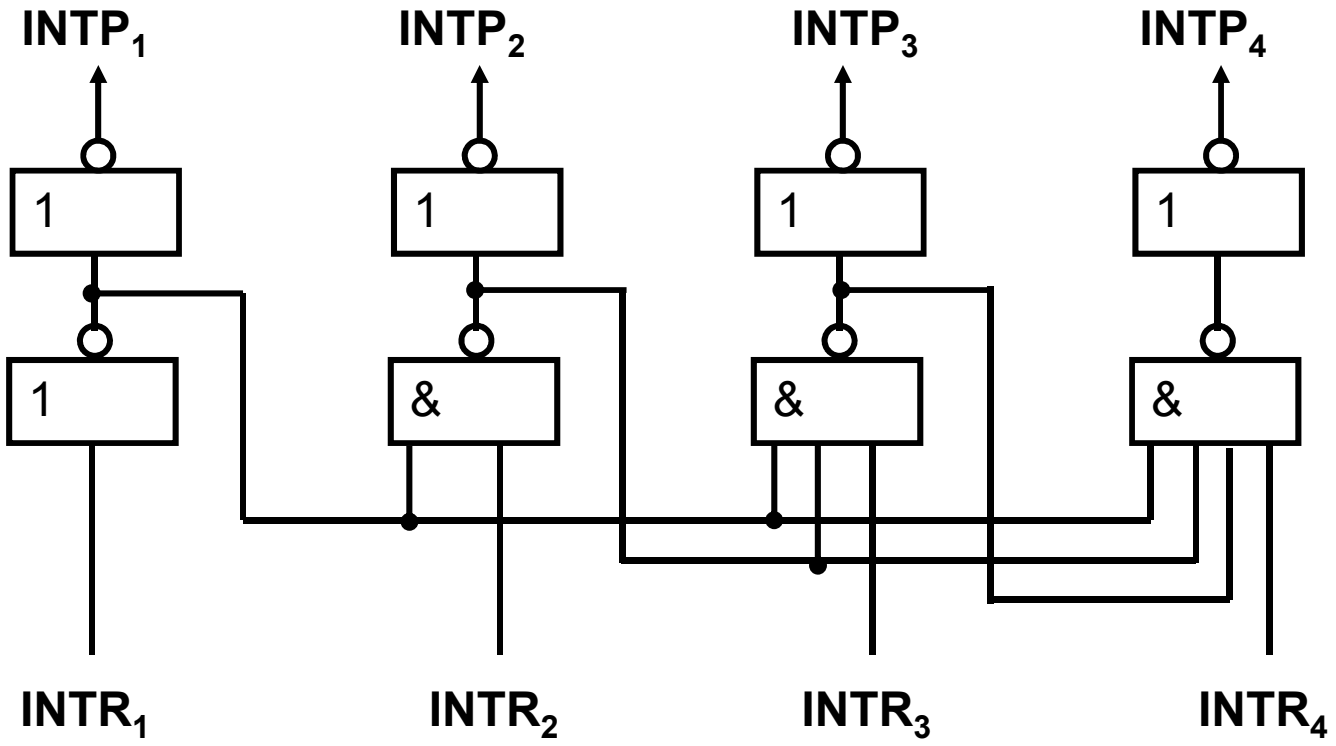
中断判优逻辑—硬件排队

第一种叫**链式排队器**，对应中断请求触发器分散在各个接口电路中的情况。



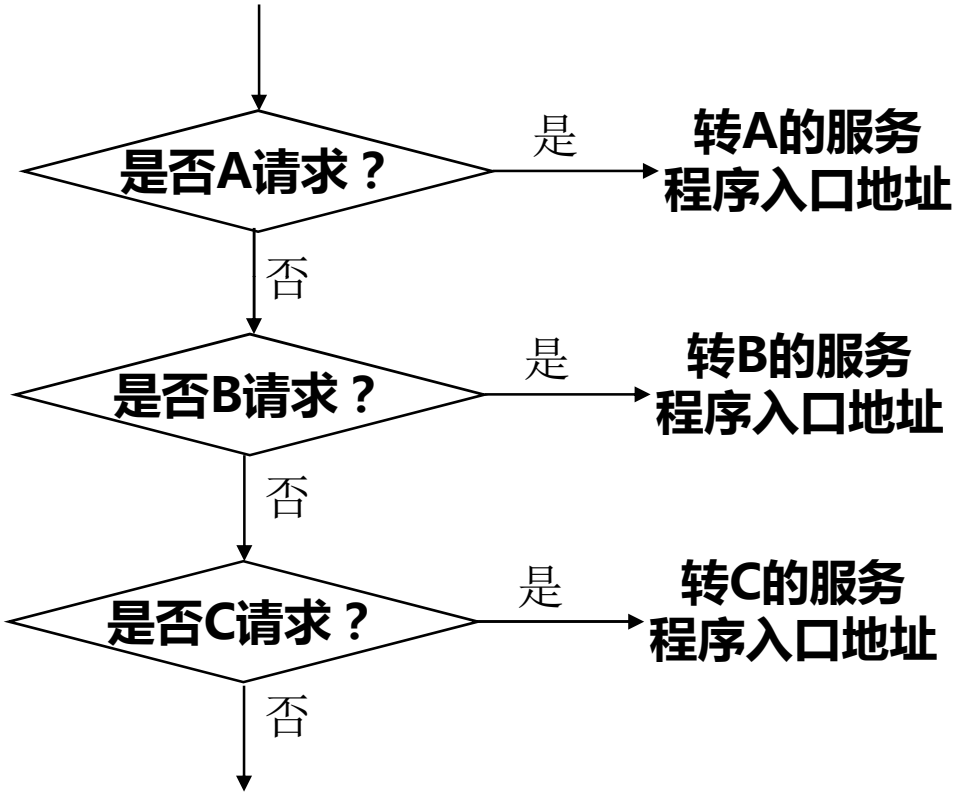
中断判优逻辑—硬件排队

- 第二种是将排队器设在CPU/INTC内。



中断判优逻辑—软件排队

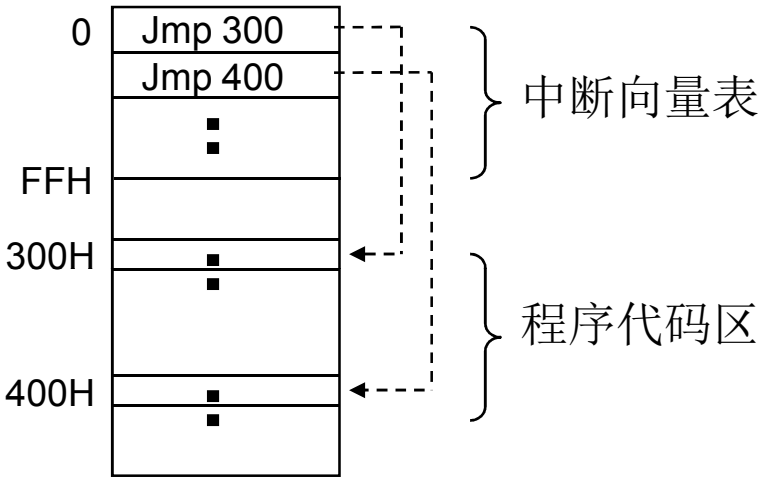
- 软件排队通过编写查询程序实现。



中断服务程序入口地址

- 不同的中断（类型）有各自的服务程序。

- **硬件向量法（中断向量法）**
  - 中断向量：设备接口向CPU提供的自身标识
    - CPU根据中断向量检索中断向量表，得到中断服务程序的入口地址
  - 向量地址，ISR入口地址？



- **软件查询法：用软件寻找中断服务程序入口地址的方法。**
  - 由程序员（或系统）事先确定中断源对应的入口地址，不涉及硬件设备，但查询时间比较长。

中断响应的条件和时间

- **条件：处理器允许中断、该中断未被屏蔽。**
  - **程序状态寄存器PSW、中断屏蔽寄存器IMR**

31	30	29	28	27 ... 7	6	5	4	3 2 1 0
C	N	Z	V	保留位	0	0	IE	mode

- **时机：指令周期结束检查是否有中断请求。**
  - **中断周期：两个机器周期**
    - 中断查询
    - 读中断向量
    - 保存断点（npc），关中断
  - **查询断点：（ IIxx？ ）**
    - 某些计算机中的有些执行时间很长，若CPU的查询信号一律安排在指令周期结束时刻，又可能因CPU发现中断请求过迟而出错。
    - 为此，在指令执行过程中设置若干个查询段点，CPU在每个“查询断点”时刻发出中断查询信号，以便及时发现和响应中断请求。

中断隐指令

- **中断隐指令：在机器指令系统中没有的指令，它是CPU在中断周期内由硬件自动完成的一条指令。**

- **中断隐指令的主要功能：**
  - **中断响应**
    - 保护程序断点：将当前程序计数器PC的内容（程序段点）保存到存储器中，或者存在存储器的特定单元（如0号地址），或者入栈。
    - 寻找中断服务程序的入口地址
    - 关中断：确保CPU响应中断后的所需作的一序列操作不至于有受到新的中断请求的干扰。

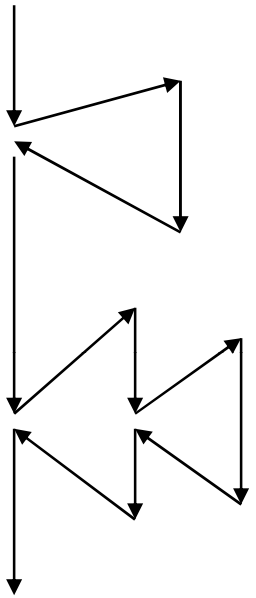
保护现场和恢复现场

- **保护现场：**
  - 程序断点—中断隐指令完成
  - CPU内部寄存器内容—用户（或系统）用机器指令编程实现

- **恢复现场：**在中断返回前，必须将寄存器的内容恢复到中断处理前的状态。
  - 由中断服务程序完成。

多重中断

- 在中断服务过程中出现高级别中断

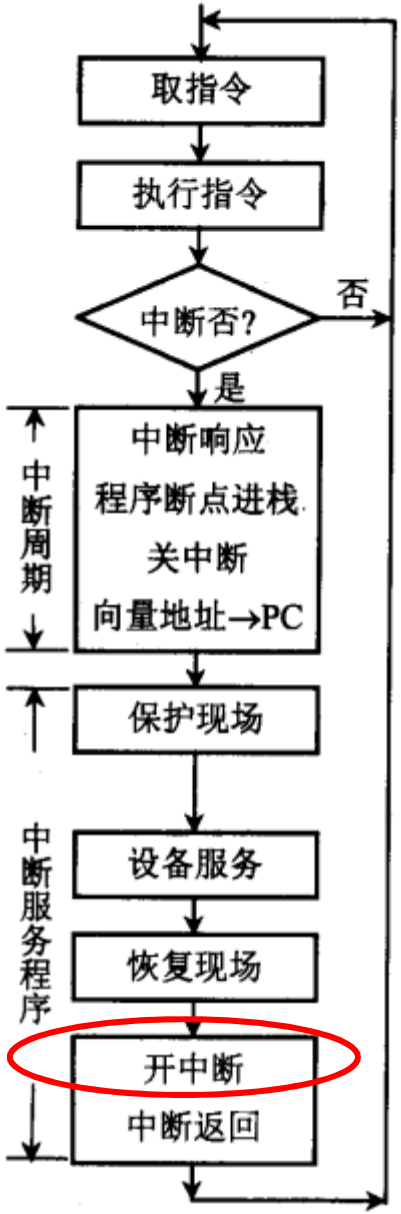


- 中断嵌套
  - 同级、低级不嵌套

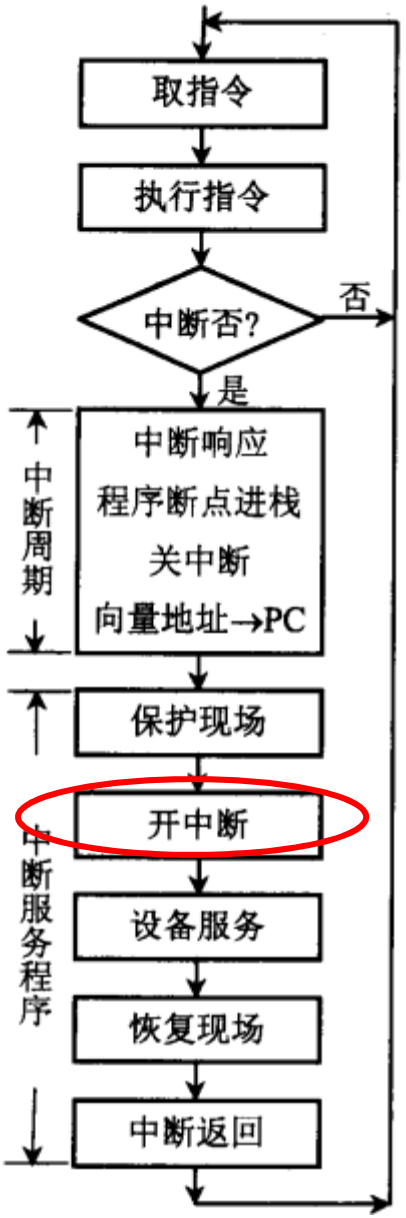
- 多重中断断点保护
  - 利用栈先进后出的特点
  - 若将断点保存在特定存储单元内（例如约定为0号单元），为保证多重中断的断点不会被覆盖，需在执行“开中断”指令之前先将0号地址单元的内容转存至别的地址单元。

实现多重中断的条件

- 提前设置“开中断”指令
  - 开中断的位置不一样。
- 优先级别高的中断源有权中断优先级别低的中断源。



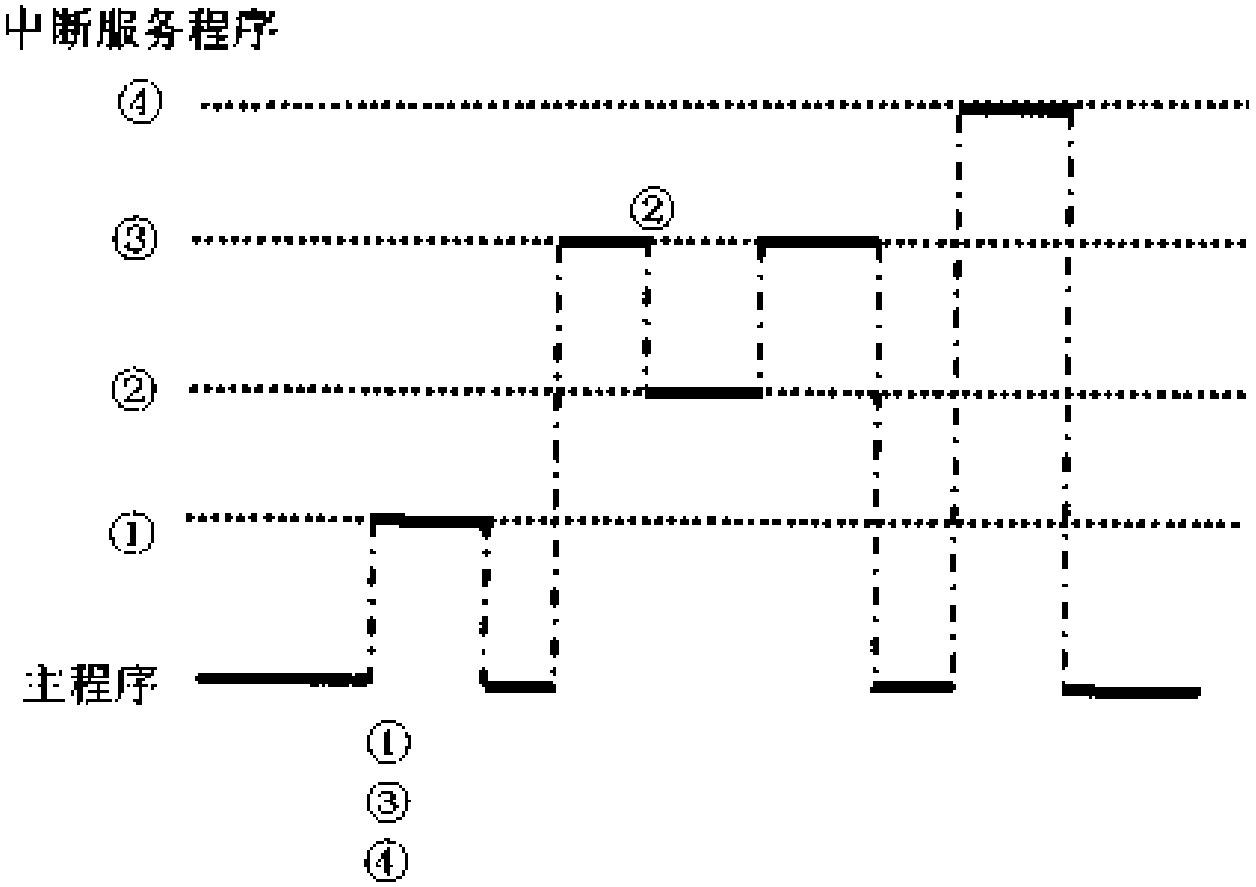
单重中断



多重中断

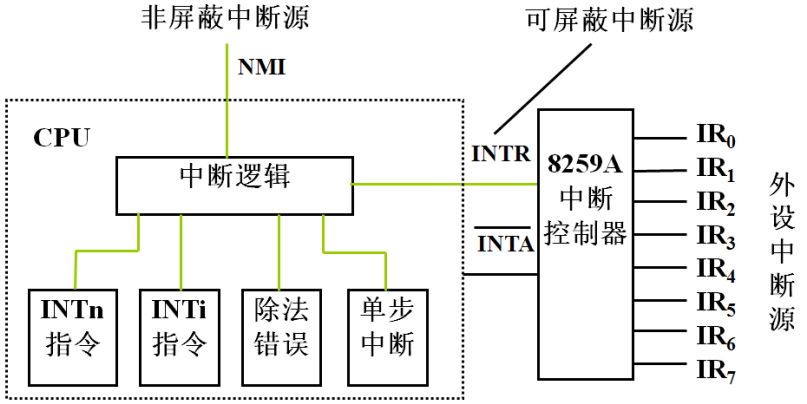
实现多重中断的条件

2. 优先级别高的中断源有权中断优先级别低的中断源。

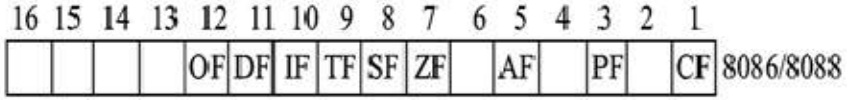


处理优先级为1>2>3>4时CPU运动轨迹

中断屏蔽技术



- **禁止响应某些中断**
  - **可屏蔽中断 vs. 不可屏蔽中断**
    - 可以屏蔽外部中断，不能屏蔽软中断
- **中断允许触发器**
  - **PSW中的一位，可以禁止CPU响应所有可屏蔽中断**

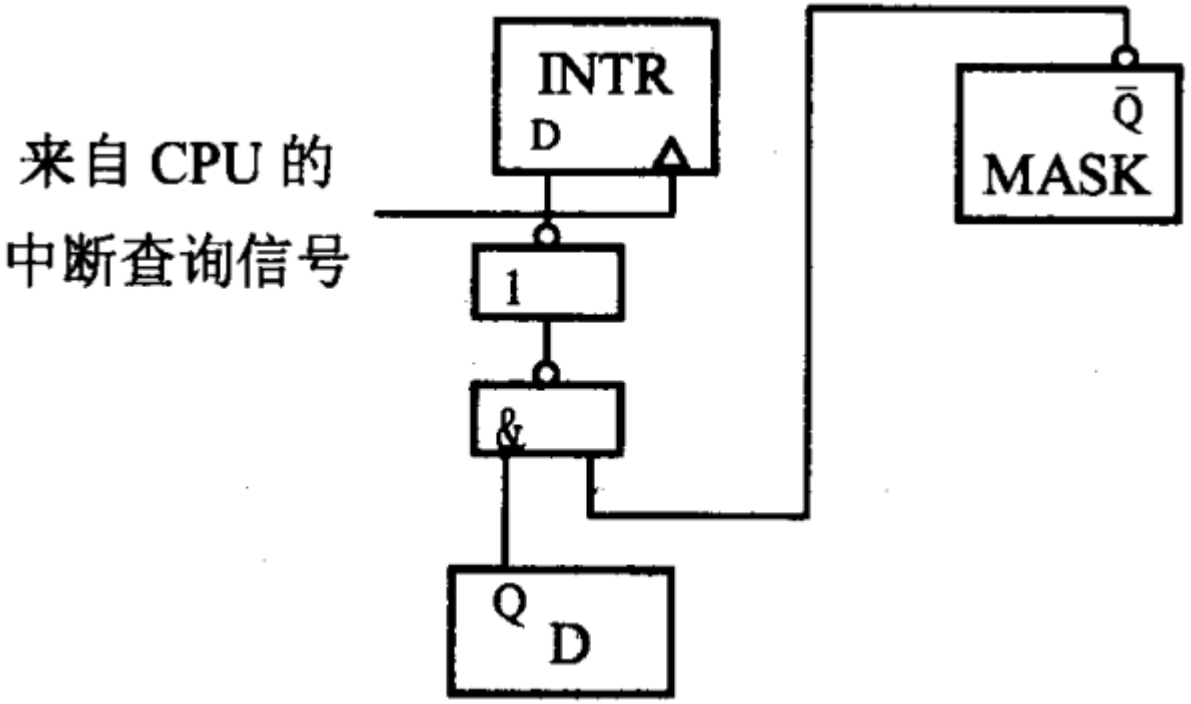


- **中断屏蔽寄存器（IMR）**
  - **位于中断接口中，每个中断源一位（每位称作中断屏蔽触发器）**
  - **屏蔽字：中断屏蔽寄存器的内容**

- **中断响应条件：处理器允许中断、该中断未被屏蔽**

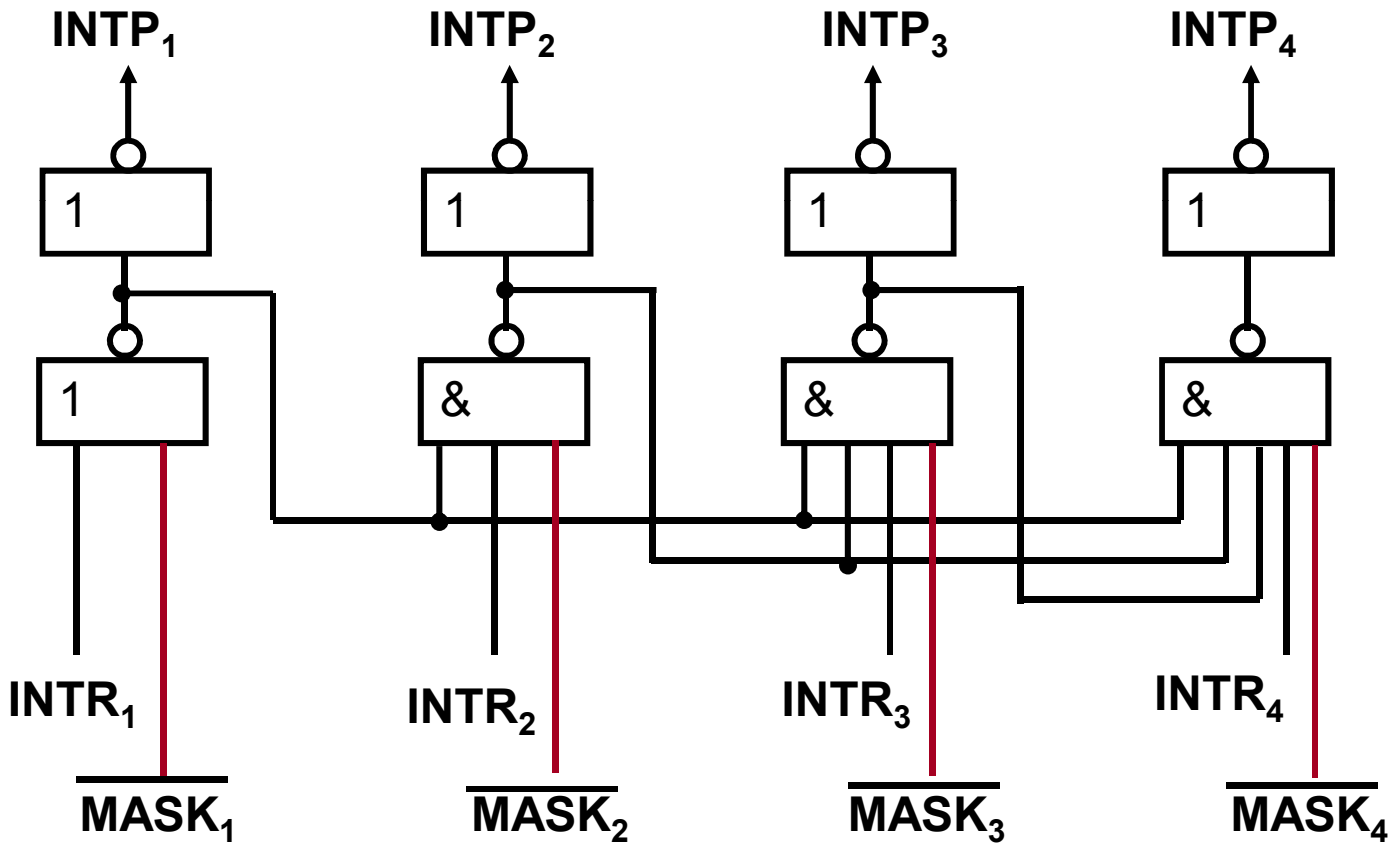
中断屏蔽技术

- **中断屏蔽触发器和屏蔽字**
  - **中断接口电路中完成触发器D、中断请求触发器INTR和屏蔽触发器MASK三者之间的关系。**



中断屏蔽技术

- **中断屏蔽触发器和屏蔽字**
  - 具有屏蔽功能的排队器

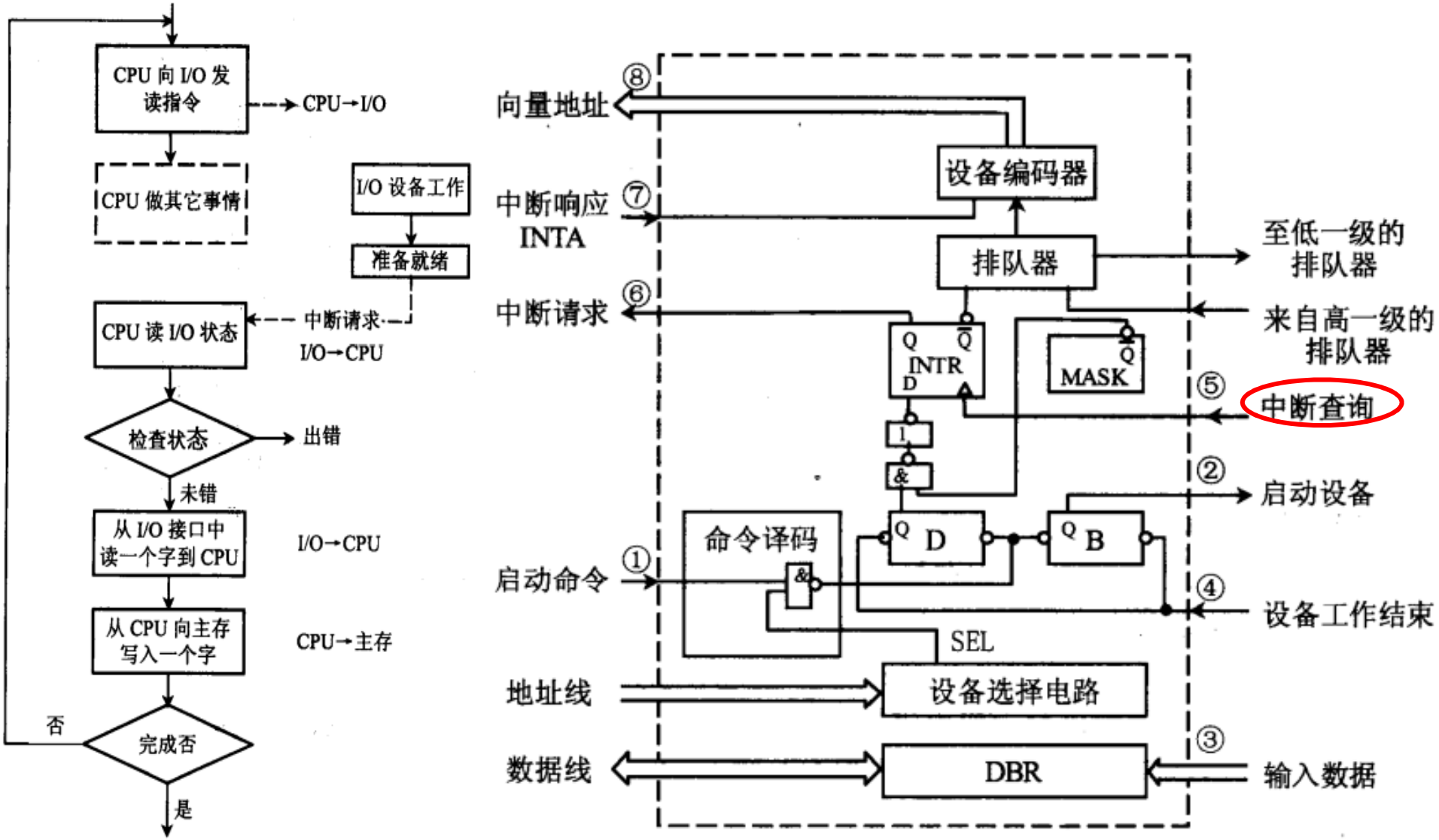


中断屏蔽技术

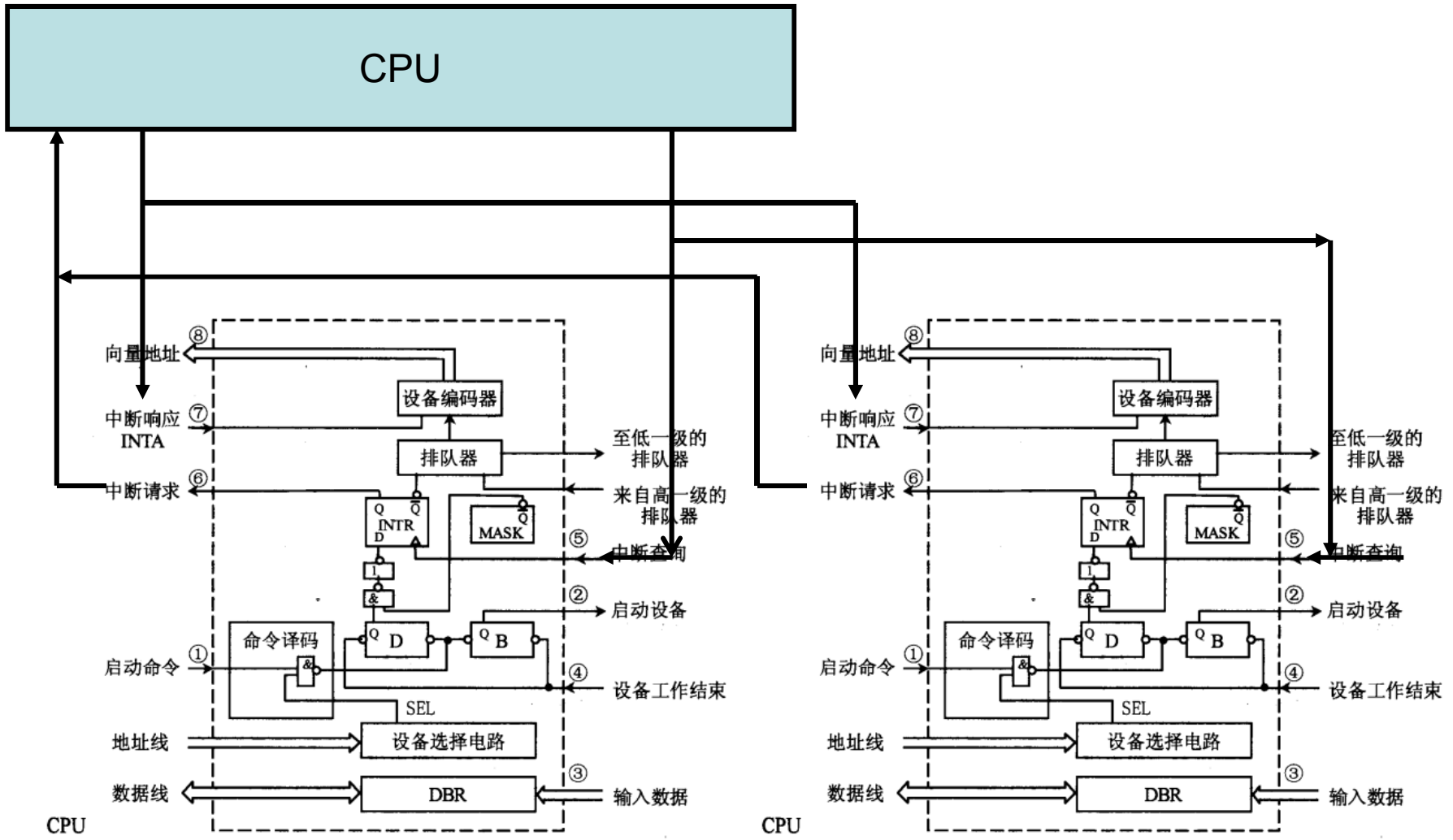
- **中断屏蔽字可改变优先等级**

- **例如：5级中断高于6级中断，但若在中断服务程序中预先设置屏蔽字为000010111111：**
  - **这样，当5、6级中断同时发生时，5级中断被屏蔽，6级中断未被屏蔽，因此，可以优先响应6级中断。**
  - **当处理完6级中断后，再设屏蔽字为000001111111，CPU才能响应5级中断源的请求。**

程序中断方式接口电路的基本组成（分布式）



如何构成系统？


CPU
向量地址
⑧
中断响应
INTA
⑦
中断请求
⑥
设备编码器
排队器
至低一级的
排队器
来自高一级的
排队器
INTR
MASK
⑤
中断查询
②
启动设备
命令译码
D
B
启动命令
①
④
设备工作结束
SEL
地址线
设备选择电路
数据线
DBR
③
输入数据
CPU
CPU

中断响应过程

- 响应过程
  - 发INTA，识别中断源（读中断类型码）
  - 清中断、关中断
  - 保存断点：PC，标志寄存器
  - 得到中断向量，转ISR
  - 保存现场：通用寄存器……
    - 保存的位置：主存，堆栈
  - 【开中断】
  - 执行中断服务
  - 【关中断】
  - 恢复现场
  - IRET
  - 开中断
  - 中断返回
- 软硬件协同完成
  - 中断隐指令、IRET

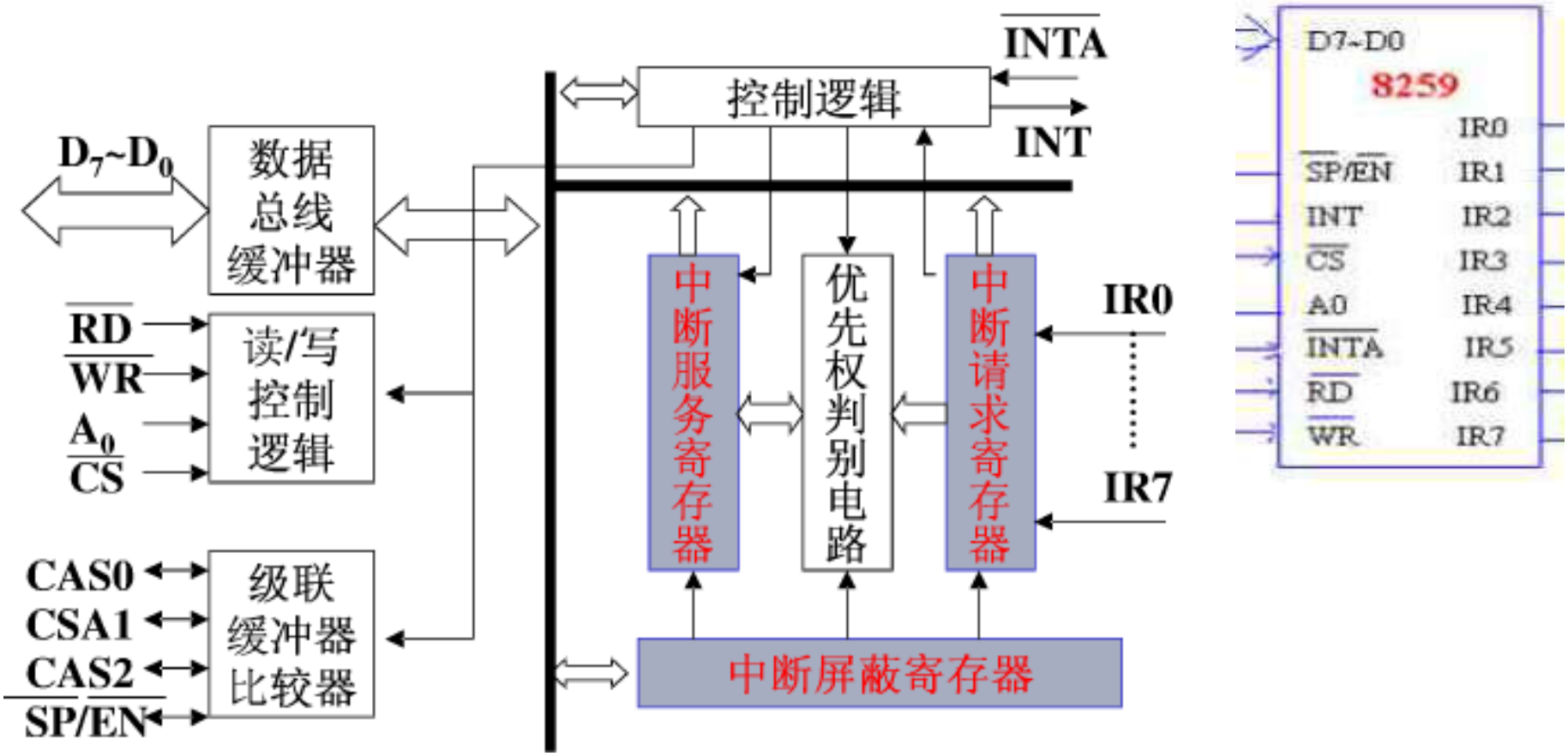
允许多重中断

中断隐指令（中断周期）

ISR

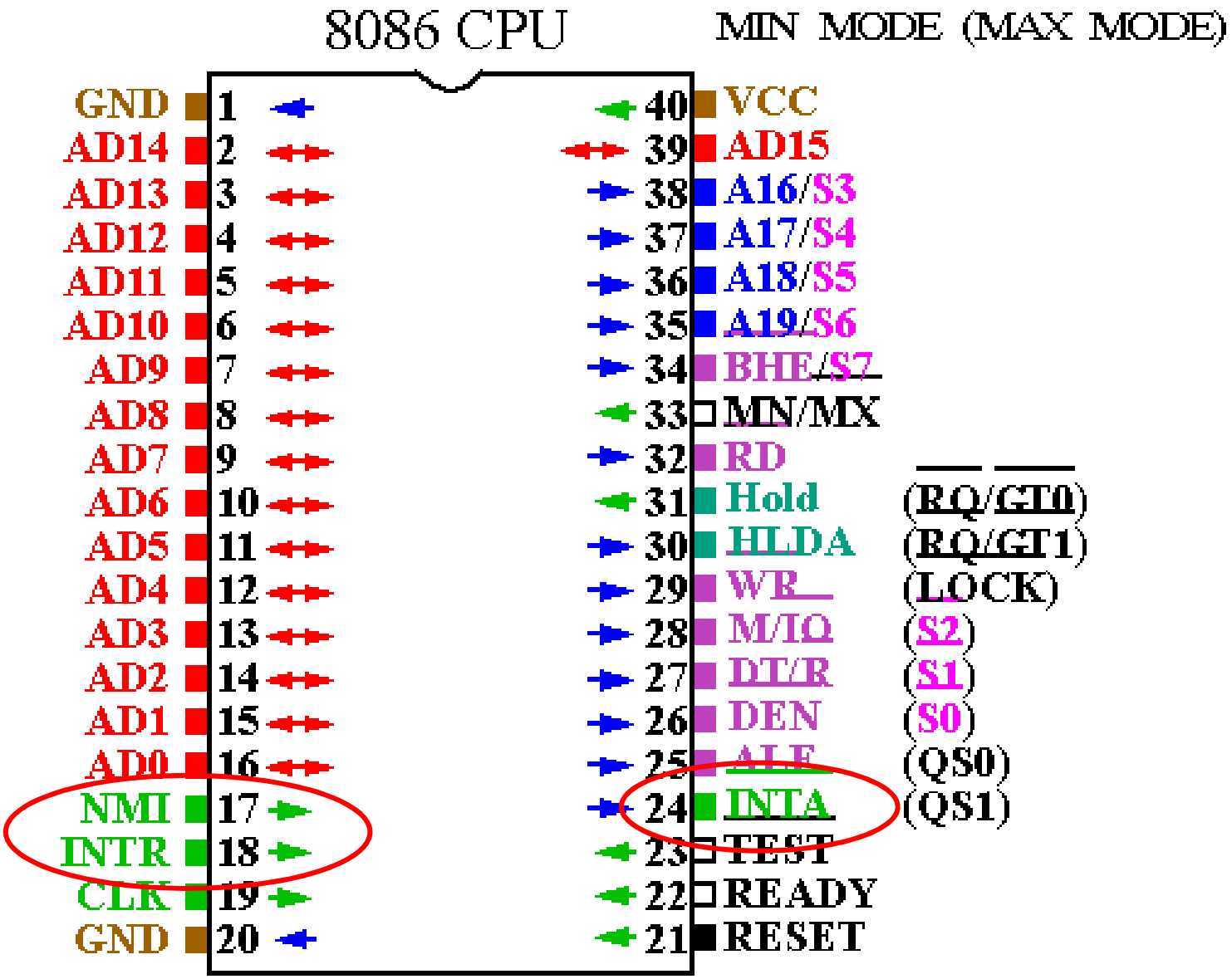
中断隐指令

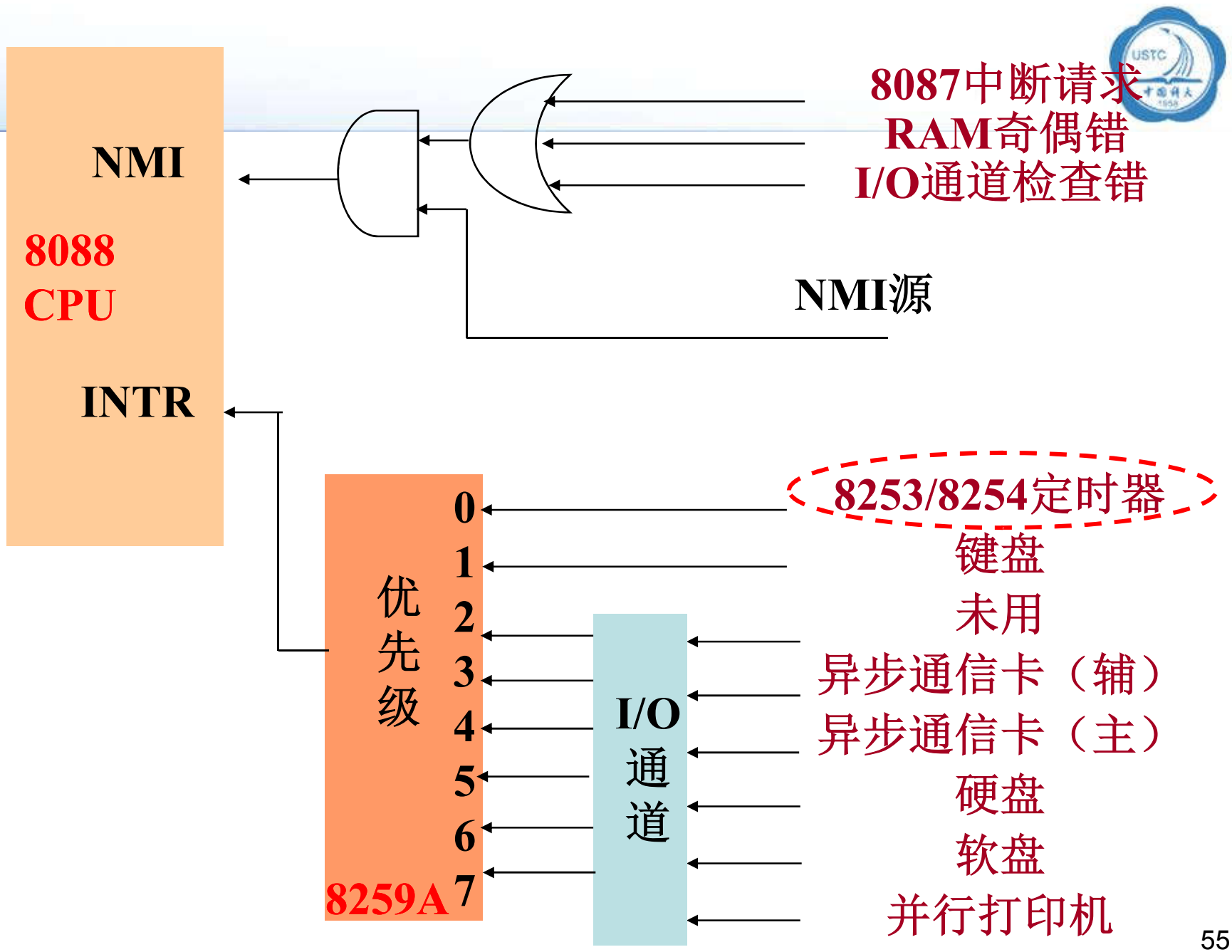
8259



中断服务寄存器(ISR)：用于保存所有正在服务的中断源。

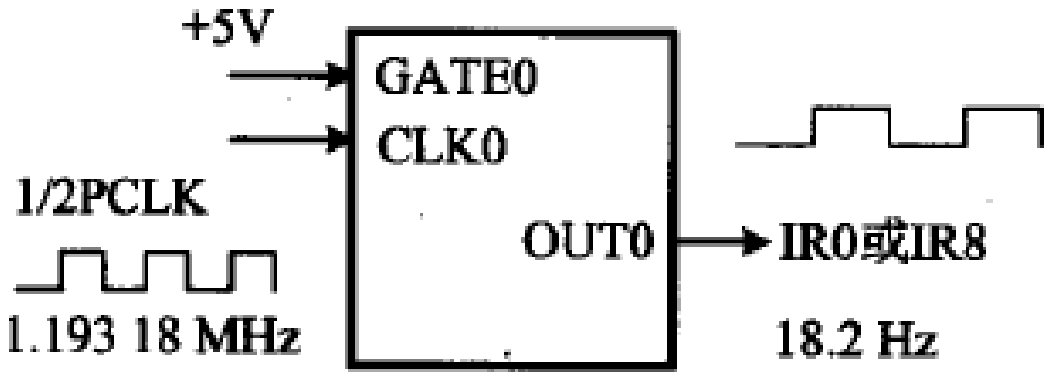
例：8086 Pinout



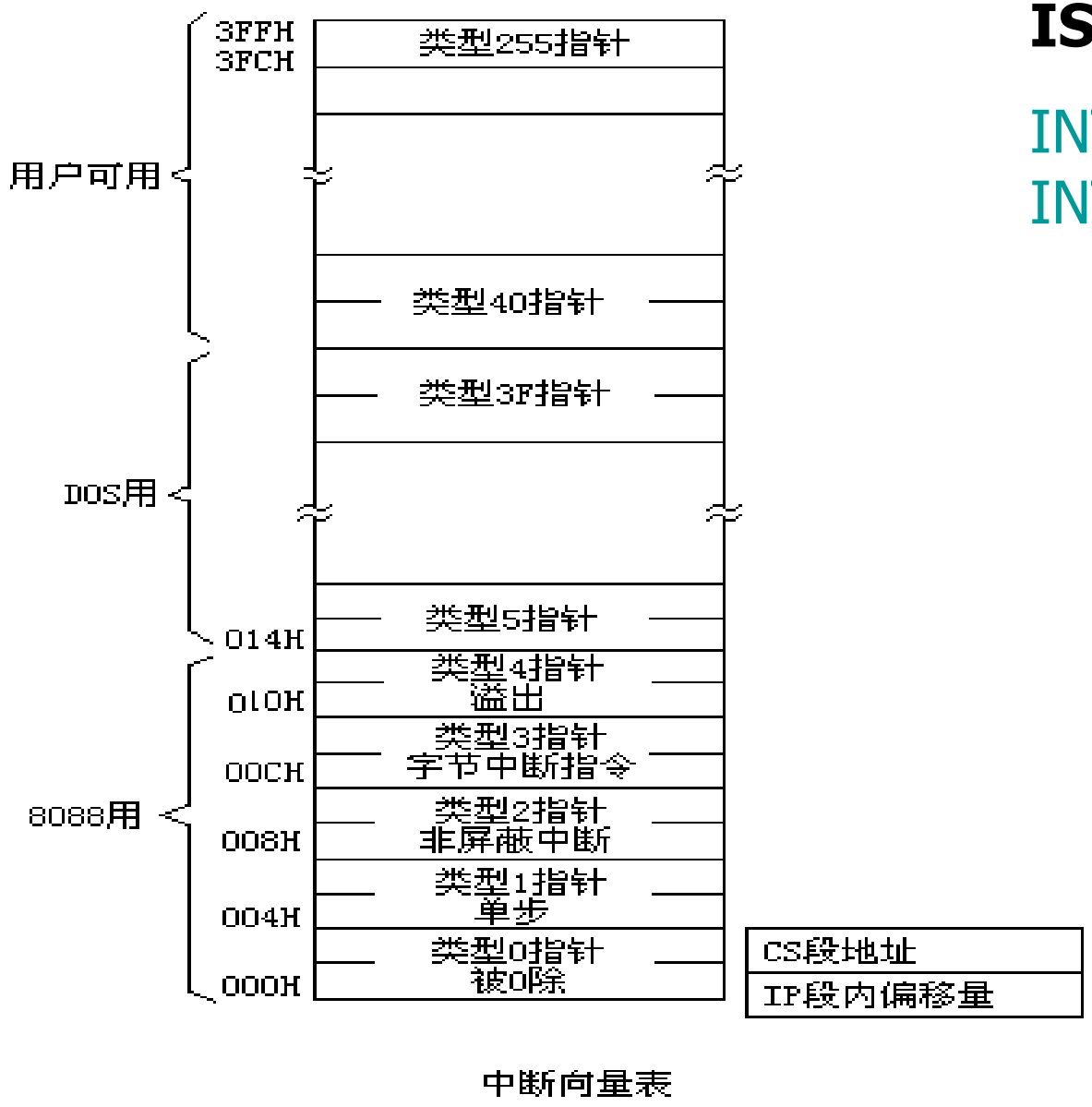

NMI
8088
CPU
INTR
8087中断请求
RAM奇偶错
I/O通道检查错
NMI源
优先级
0
1
2
3
4
5
6
7
8259A
I/O通道
8253/8254定时器
键盘
未用
异步通信卡（辅）
异步通信卡（主）
硬盘
软盘
并行打印机

8254可编程间隔定时器（PIT）

- **3个计时通道**
  - **每个通道都有一个递减计数器，最大值是10000h**
  - **通道0：负责更新系统时钟**
    - 每个时钟滴答（计数器减到0）产生一次时钟中断
      - 时钟中断频率为1193181 / 65536＝18.2HZ
    - 送往8259 IRQ0，产生周期性的时钟tick信号
  - **通道1：用于控制DMAC对RAM的刷新**
  - **通道2：产生方波信号，连接到PC机的扬声器。**



8086中断服务程序

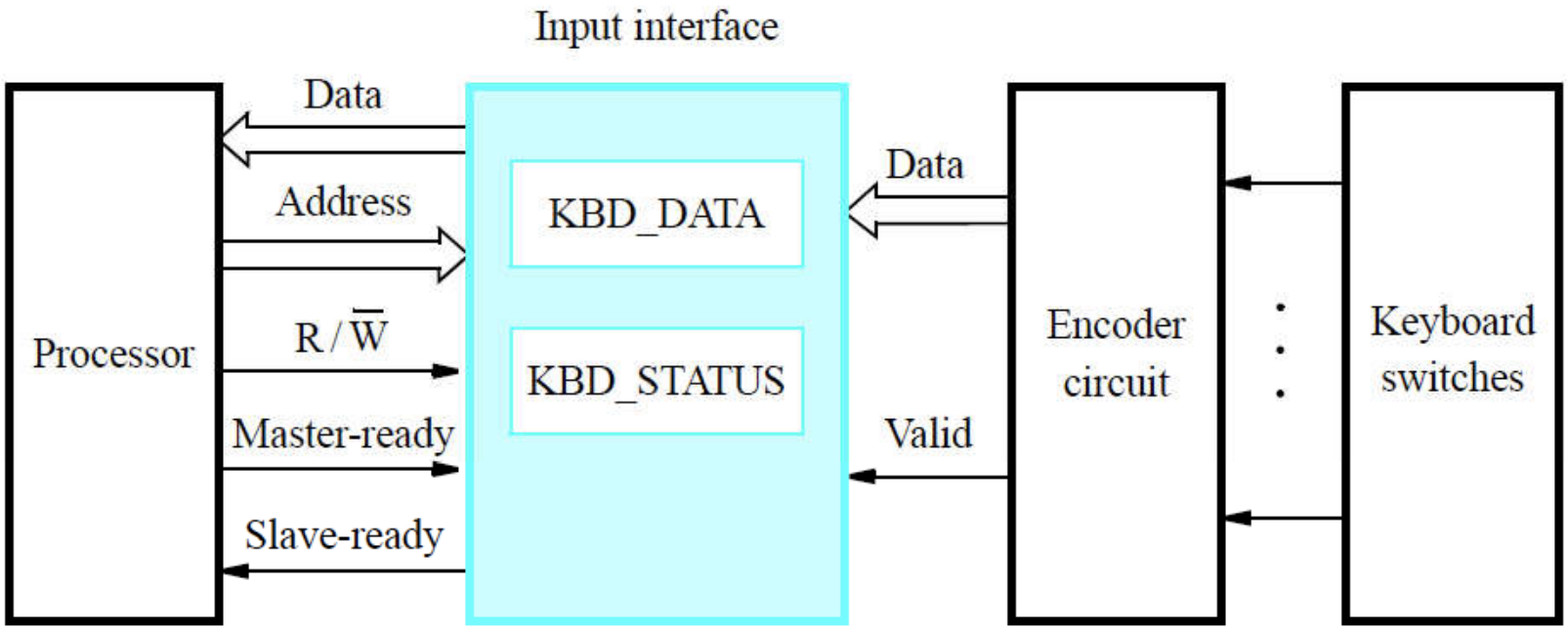


中断向量表

ISR框架结构

```
INTHANDLE PROC FAR
INTHAND:
    PUSH AX
    PUSH BX
    PUSH CX
    …
    POP CX
    POP BX
    POP AX
    IRET
```

Keyboard-to-processor connection



键盘工作原理

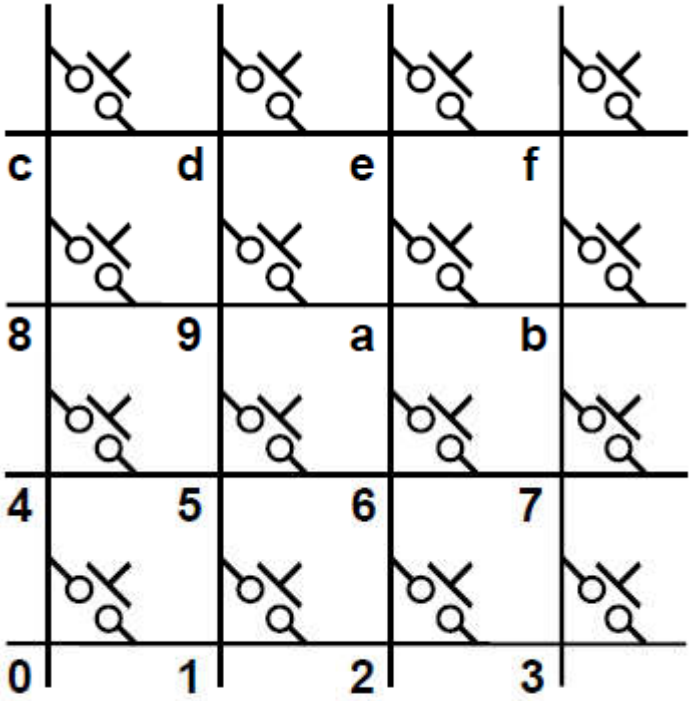
- 由一组排列成阵列的按键开关组成
  - 按下一个键，产生一个位置码
  - 位置码转换成字符码，送入主机
- 读键过程
  - 按键
  - 识别（查出按下的是哪一个键）
  - 将此键翻译成ASCII码

- 编码键盘与非编码键盘
  - 编码键盘法：采用硬件电路确认哪个键按下的方法。
  - 非编码键盘法：采用软件确认哪个键按下的方法。它利用简单的硬件和一套专用的键盘编码程序来判断按键的位置，然后由CPU将位置码经查表程序转换成相应的编码信息。

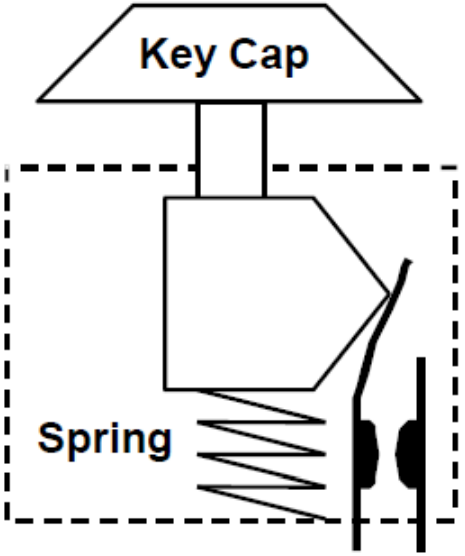
硬件：编码键盘，直接得到*字符码*（ASCII）
软件：非编码键盘，读位置码—>查表—>字符码

- 消抖方法（按键时产生的机械抖动容易造成误判）：硬件电路，软件

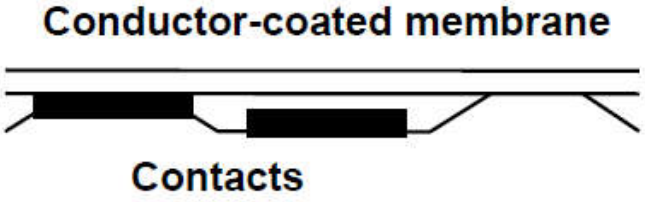
Keyboard



Logical arrangement of keys

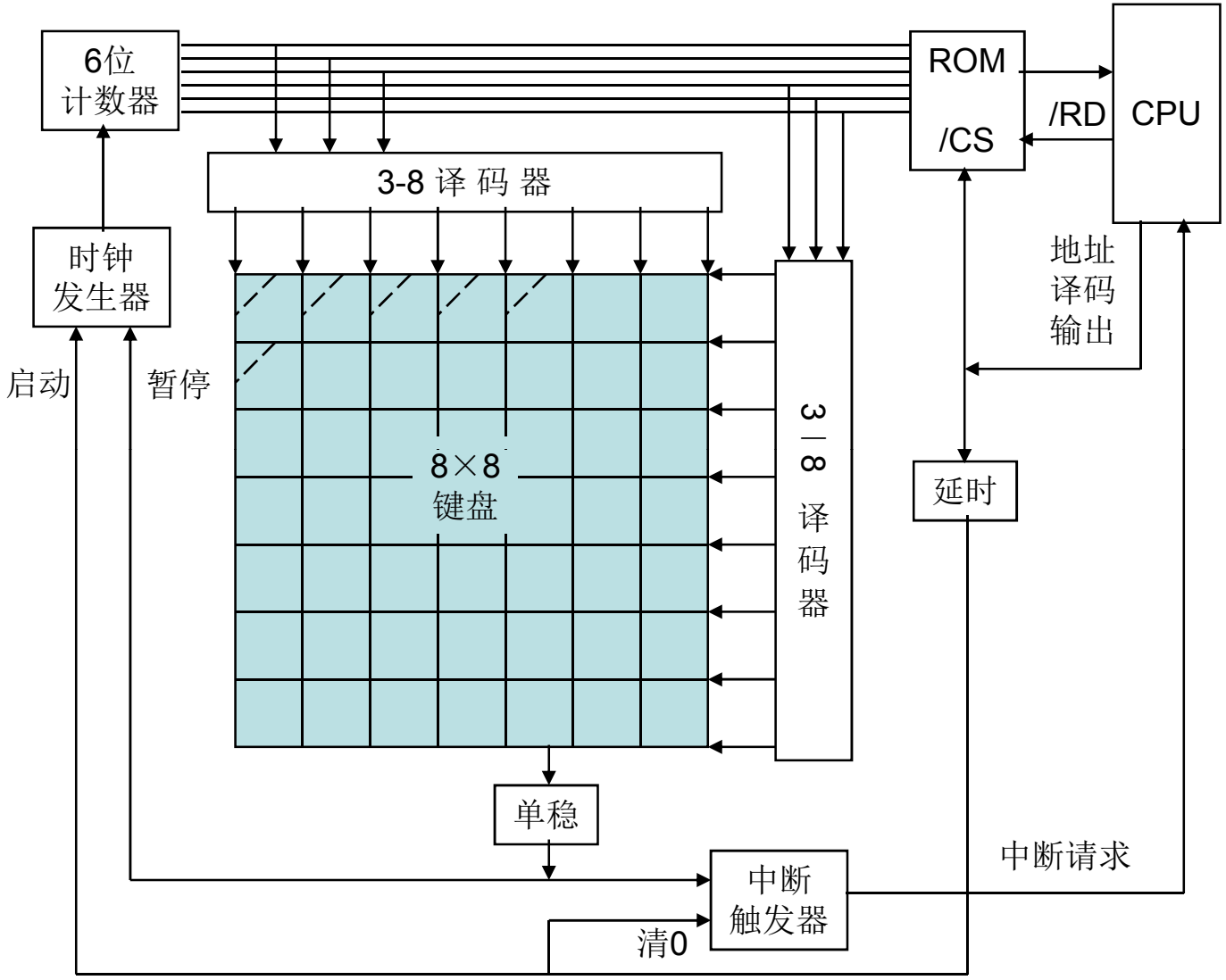


Mechanical switch



Membrane switch

编码键盘原理图



要点：

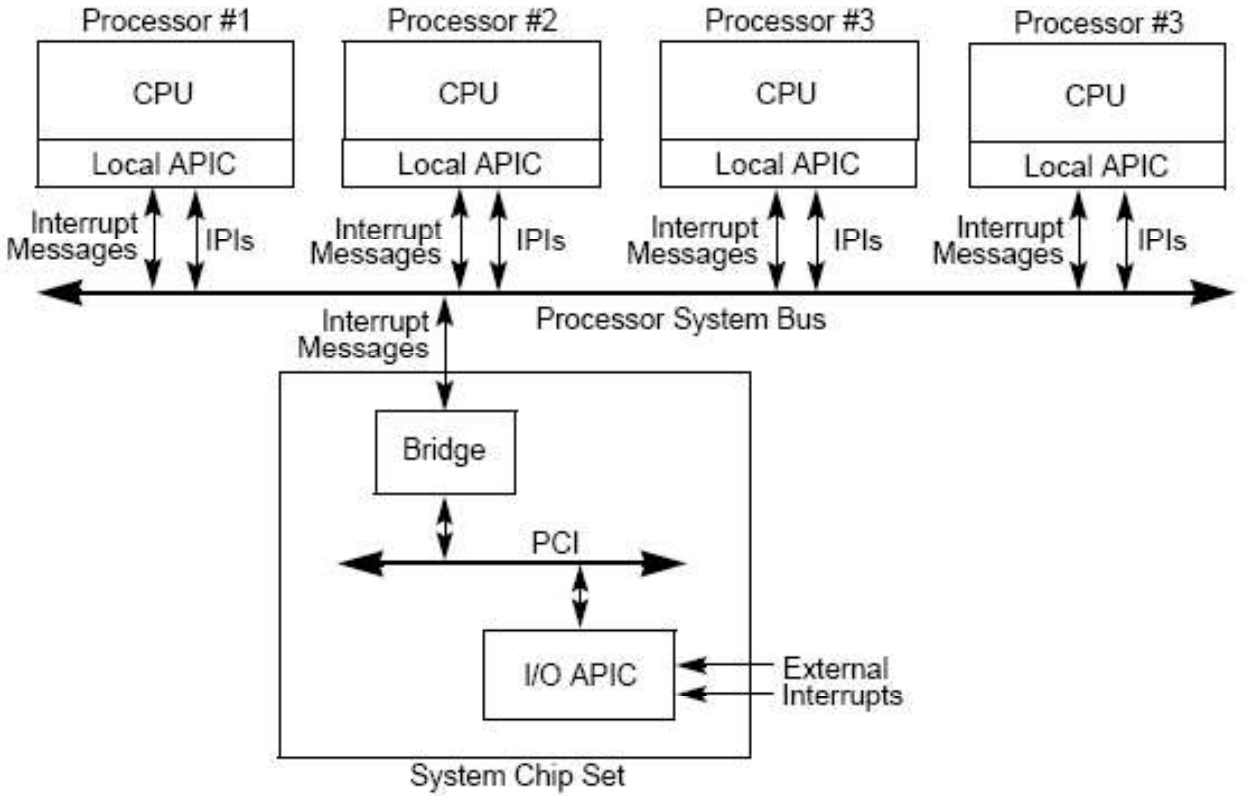
ROM中存储各个按键的字符码(ASCII)

计数器循环计数扫描键盘得到当前按键的位置码

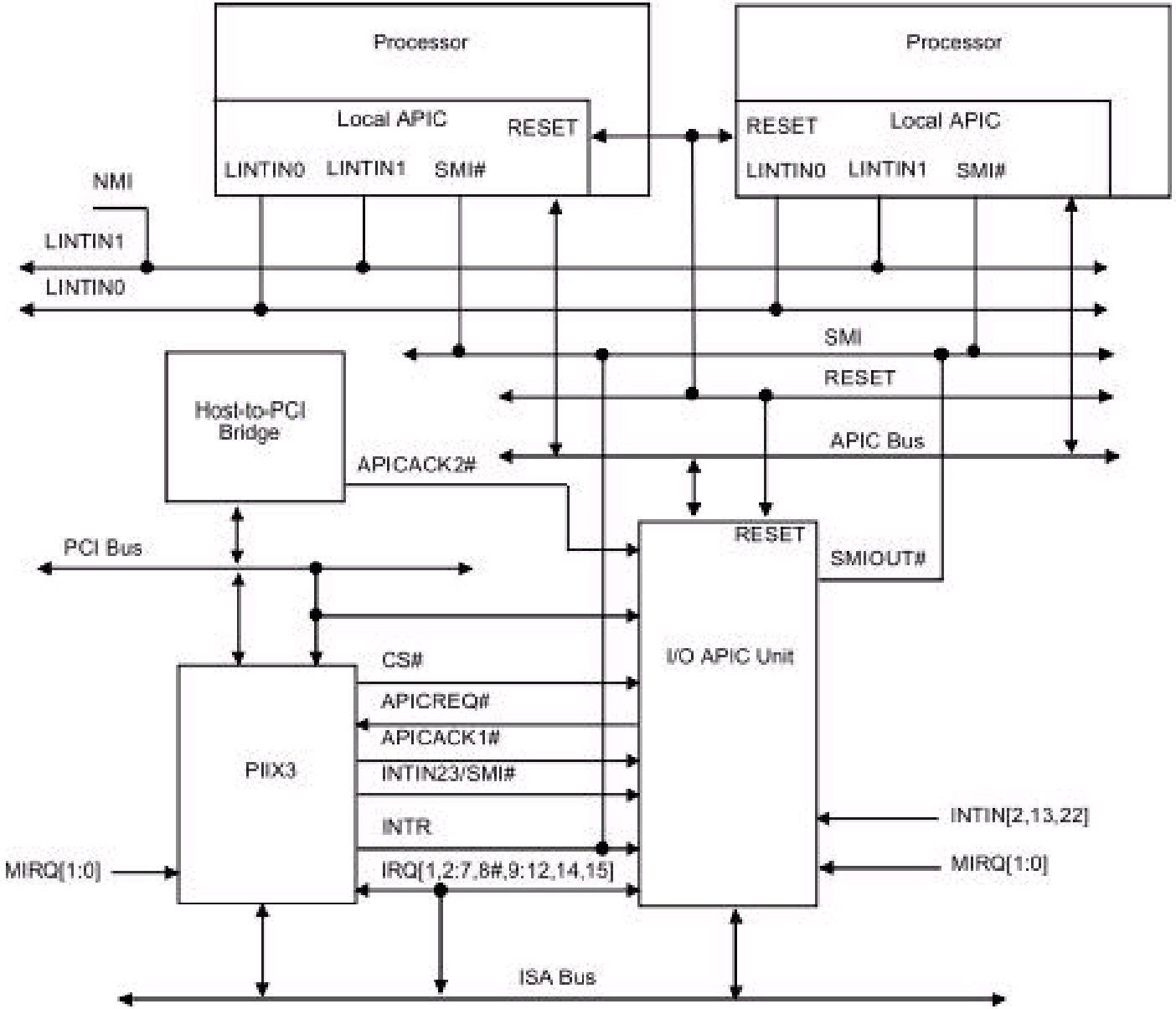
按键按下时停止计数，并产生中断请求，CPU读字符码

多核高级可编程中断控制器 APIC

- **由Local APIC和I/O APIC构成，负责传送IPI消息**
  - **通过中断命令寄存器（ICR）来接收和发送IPI消息**
    - 外部中断：必须将外部中断处理分发给一组核处理。
    - 核间中断：用于核间通信，须将某核的中断请求分发给能够提供服务的核。



例：APIC


Processor
Local APIC
RESET
LINTIN0
LINTIN1
SMI#
Processor
RESET
Local APIC
LINTIN0
LINTIN1
SMI#
NMI
LINTIN1
LINTIN0
SMI
RESET
APIC Bus
Host-to-PCI Bridge
APICACK2#
PCI Bus
RESET
SMIOUT#
I/O APIC Unit
CS#
APICREQ#
APICACK1#
INTIN23/SMI#
INTR
IRQ[1,2:7,8#,9:12,14,15]
PIIX3
MIRQ[1:0]
INTIN[2,13,22]
MIRQ[1:0]
ISA Bus

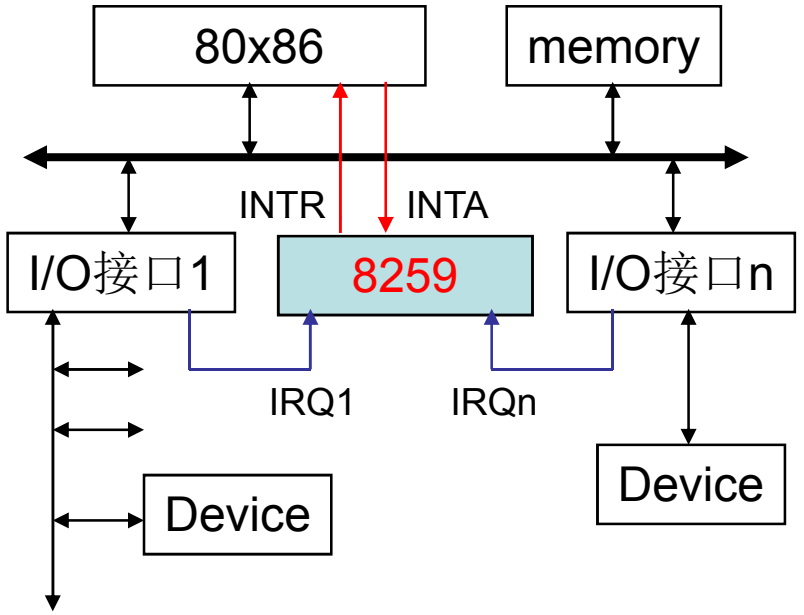
SMP IRQ Affinity


```
# cat /proc/interrupts
           CPU0       CPU1
  0:  918926335          0    IO-APIC-edge  timer
  1:          2          0    IO-APIC-edge  i8042
  8:          0          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 12:          4          0    IO-APIC-edge  i8042
 14:    8248017          0    IO-APIC-edge  ide0
 50:        194          0   IO-APIC-level  ohci_hcd:usb2
 58:      31673          0   IO-APIC-level  sata_nv
 90:    1070374          0         PCI-MSI  eth0
233:         10          0   IO-APIC-level  ehci_hcd:usb1
NMI:       5077       2032
LOC:  918809969  918809894
ERR:          0
MIS:          0
```

```
# /etc/init.d/irqbalance stop
# echo "2" > /proc/irq/90/smp_affinity
# cat /proc/interrupts
           CPU0       CPU1
  0:  922506515          0    IO-APIC-edge  timer
  1:          2          0    IO-APIC-edge  i8042
  8:          0          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 12:          4          0    IO-APIC-edge  i8042
 14:    8280147          0    IO-APIC-edge  ide0
 50:        194          0   IO-APIC-level  ohci_hcd:usb2
 58:      31907          0   IO-APIC-level  sata_nv
 90:    1073399        145         PCI-MSI  eth0
233:         10          0   IO-APIC-level  ehci_hcd:usb1
NMI:       5093       2043
LOC:  922389696  922389621
ERR:          0
MIS:          0
```

中断小结—基本概念

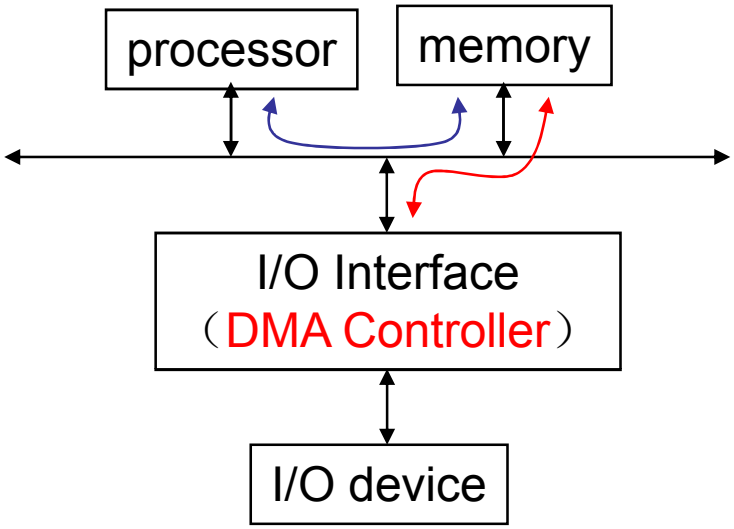
- 基本概念
  - 中断、中断类型、中断优先级、中断屏蔽、中断嵌套、中断服务、中断向量...
  - 中断的响应条件、时机
  - 中断处理过程
    - 中断向量、保存现场、中断服务、恢复现场、中断返回
- 如果中断频繁，则效率低
  - 高速、批量数据传输不适用
  - 某些任务必须使用（如调度）
  - ISR必须短小
- 同步？异步？
  - 与指令周期的关系
  - 响应的及时性
    - 软中断，内部中断，异常：同步
    - 硬中断，外部中断：异步（与指令周期的关系，低优先级不能中断高优先级）

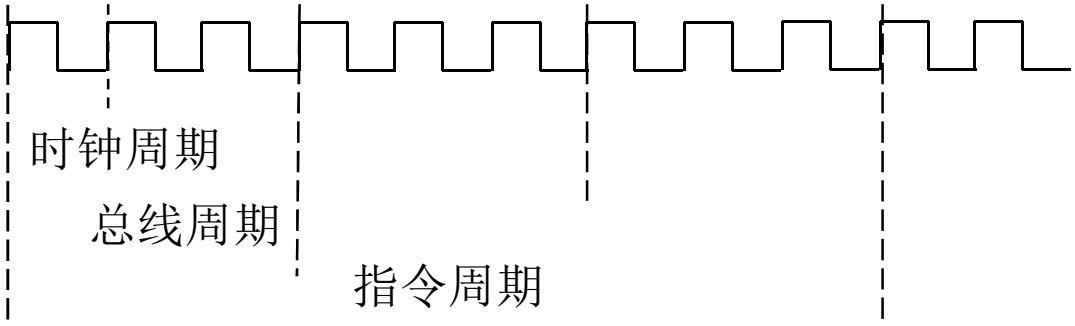


5.6 DMA方式

DMA方式

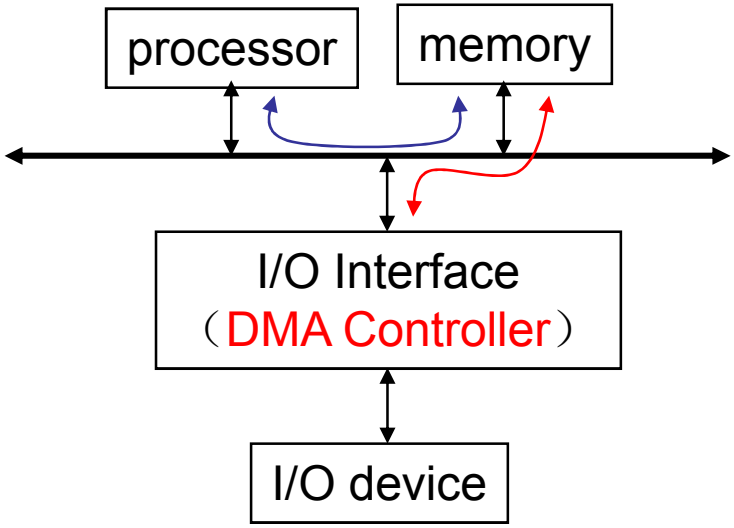
- 进一步提高效率，避免在I/O时占用CPU
- 需要解决访存时总线冲突问题
  - CPU在总线周期结束时让出总线
    - 时钟周期
    - 总线周期
    - 指令周期
- 批量数据传输





DMA概念

- 用途：高速外设直接与主存进行数据传输
  - CPU不用暂停现行程序而为I/O服务
- 关键问题：DMA接口与CPU之间共享主存，存在总线争用、访存冲突问题
  - CPU暂停方式
  - 周期窃取方式
  - 交替访问方式



CPU暂停方式

- CPU暂停方式过程
  - DMA向CPU申请总线
  - CPU暂停
  - DMA传输
  - DMA释放总线
  - CPU继续
- 并未真正实现并发

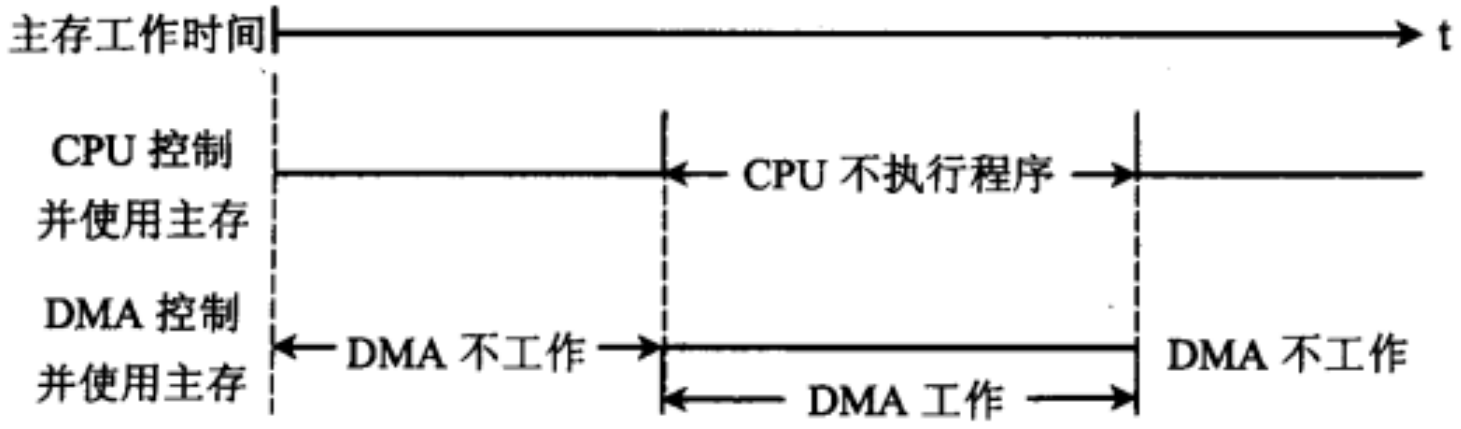
- 回顾：总线主设备（Master）、从设备（Slaver）
  - 总线仲裁

周期窃取/周期挪用（cycle stealing）

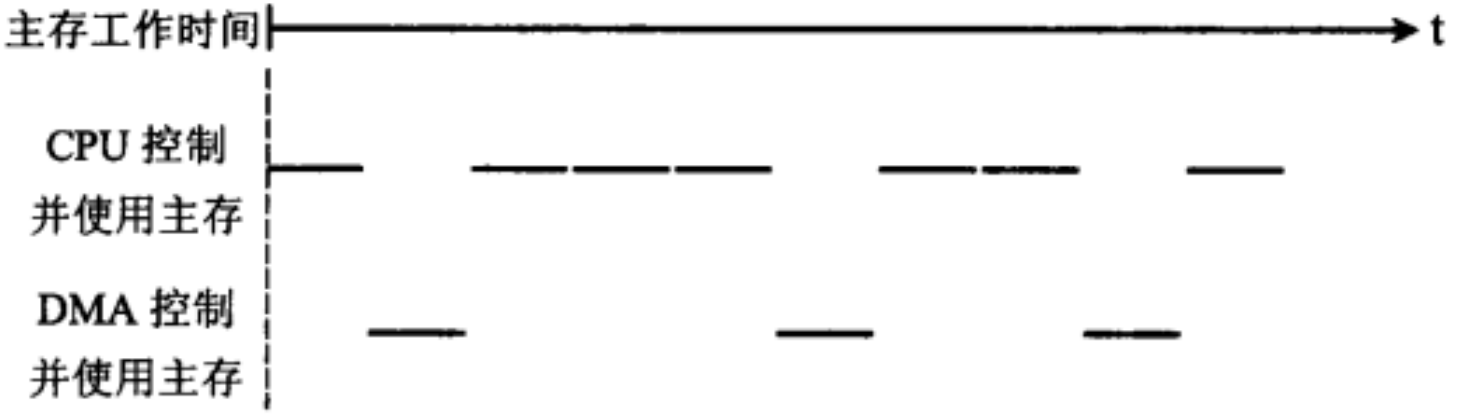
- 当I/O设备发出DMA请求时，I/O设备便挪用或窃取总线占用权一个或几个主存周期。
- 有三种情况
  - CPU不访存（复杂指令mul）：DMA使用
  - CPU正在访存：DMA等待CPU完成（总线周期结束），然后获得总线使用权
  - CPU与DMA同时发生：DMA优先，窃取一到二个存取周期（否则数据丢失）

- 应该指出，I/O设备每挪用一个主存周期都要申请总线控制权、建立总线控制权和归还总线控制权。因此，尽管传送一个字对主存而言之占用一个主存周期，但是对DMA接口而言，实质上要占2～5个主存周期。

CPU暂停方式和周期挪用示意



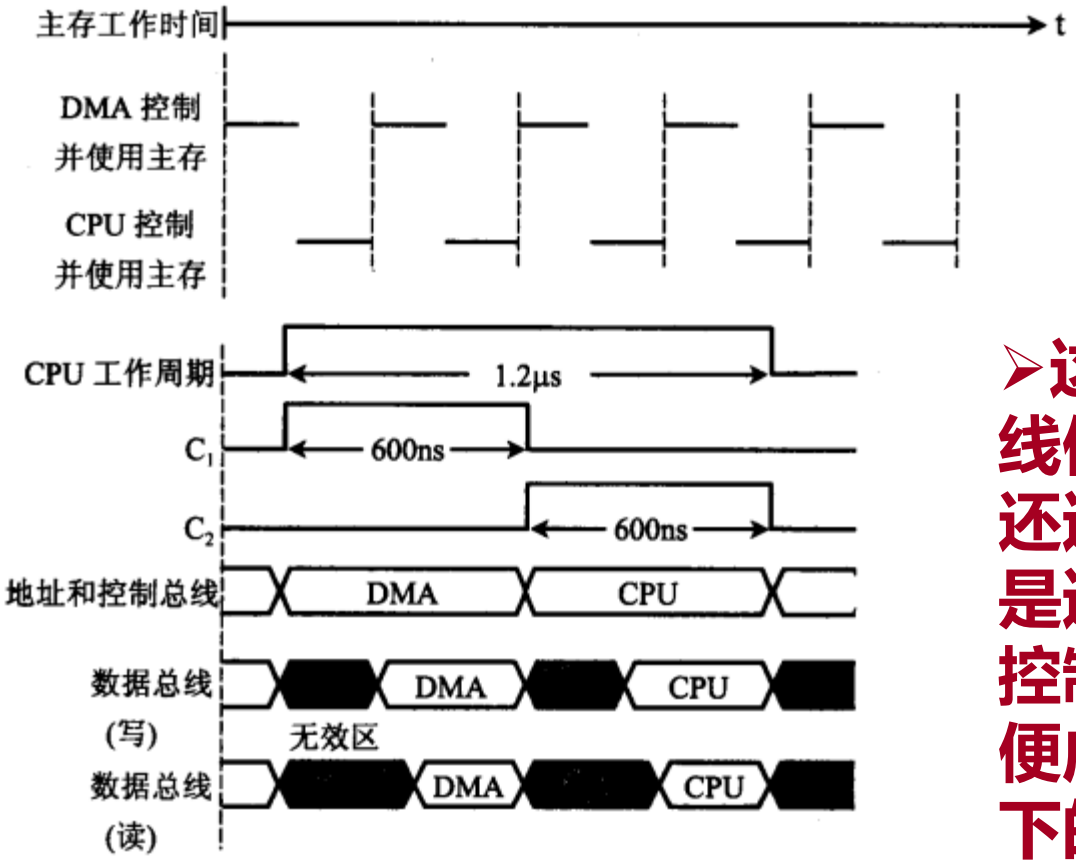
(a) 停止 CPU 访问主存



(b) 周期挪用

交替访问——周期扩展

- 将CPU工作周期延长（或者CPU的工作周期本身比主存周期长得多），分成两段。下图为交替访问方式示意图。



(c) DMA 与 CPU 交替访问

➢这种方式不需要总线使用权的建立和归还过程，总线使用权是通过C_1和C_2分别控制的。实际上总线便成了C_1和C_2控制下的多路转换器。

DMA接口的功能

① 向CPU申请DMA传送；

② 在CPU允许DMA工作时，处理总线控制权的转交，避免因进入DMA工作而影响CPU正常活动或引起总线竞争；

③ 在DMA期间管理系统总线，控制数据传送；

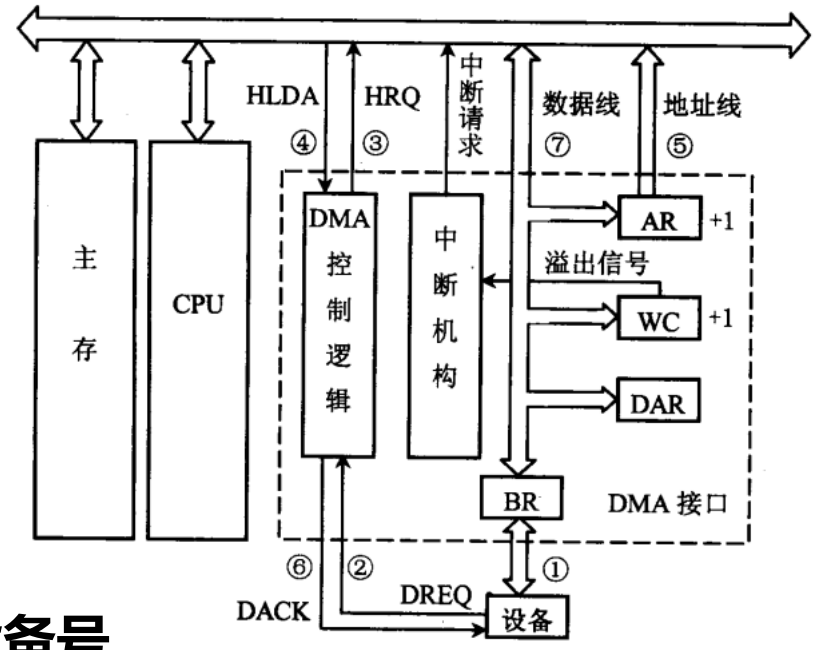
④ 确定数据传送的起始地址和数据长度，修正数据传送过程中的数据地址和数据长度；

⑤ 在数据块传送结束时，给出DMA操作完成的信号。

简单的DMA接口的基本组成

➢DMA接口也称作DMA控制器(DMAC)

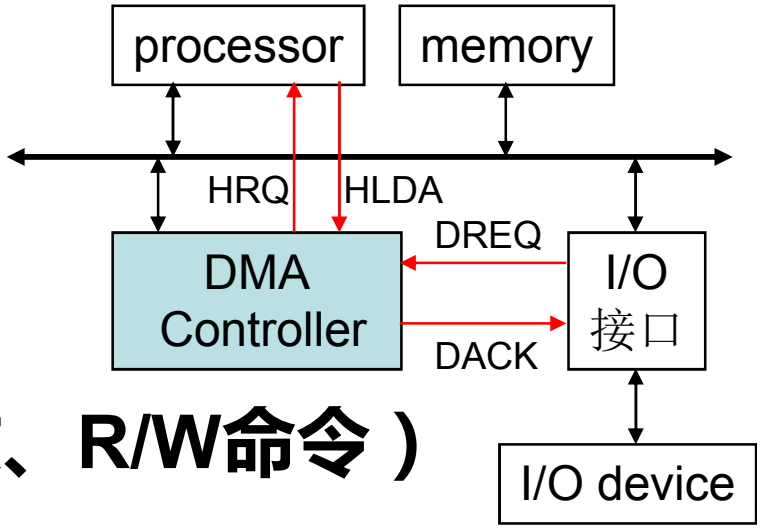
- 主存地址寄存器（Address Register）
  - 每传输一个数据（字节、字），地址加1
- 字计数器（word counter）
- 数据缓存寄存器（Buffer Reg）
  - 完成数据格式转换等
  - 数据传输不一定经过DMAC
- DMA控制逻辑
  - DREQ-DACK、HRQ-HLDA
  - 优先级控制
- 设备地址寄存器（DAR）
  - 外设中数据块地址或当前设备的设备号
- 中断逻辑——不一定在DMAC中
  - 请求CPU进行DMA“后处理”
  - DMA出错处理



DMA工作过程

- 预处理
  - 初始化（数据传输方向、AR、WC、DAR）
  - 启动设备开始传输
- 数据传输
  - 外设准备好数据
  - 外设通过DMAC申请总线使用权
    - DREQ → HRQ → HLDA → DACK
  - DMAC接管总线，控制完成数据传输
  - 释放总线
    - /HLDA
- 后处理
  - 请求中断服务，对读入的数据进行处理

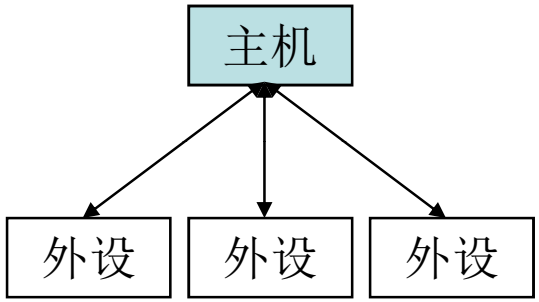
DMA控制器（DMAC）

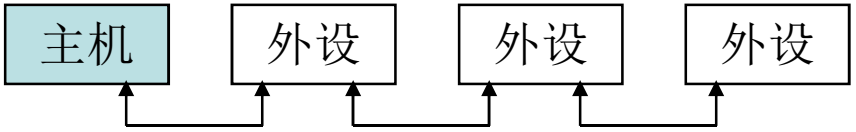


- 功能
  - 申请总线
  - 控制总线
  - 控制传输过程（地址、计数、R/W命令）
  - 释放总线
- I/O
  - 内存与外设、内存与内存、外设与外设
- 数据大小
  - 单字节、块传输、请求传输

DMA系统连接方式

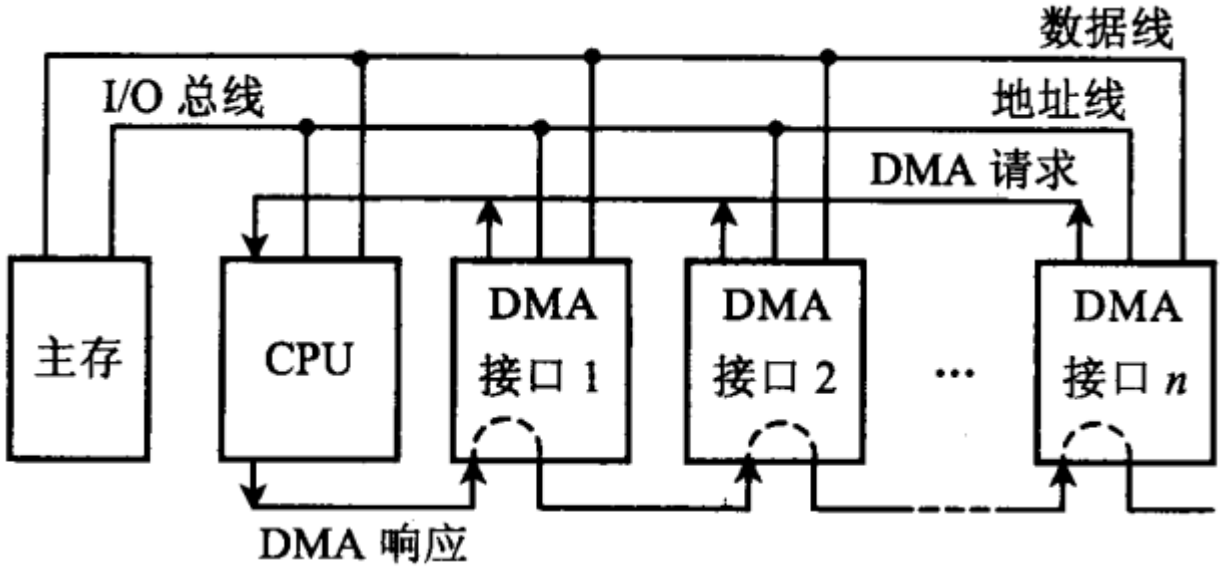
- **系统中存在多个DMA通道和多个I/O设备，如何连接？**
  - **优先级问题**
- **DMA ←→ CPU：多个DMAC**
  - **公共请求方式：链式/级联**
  - **独立请求方式：星型连接**
- **DMA ←→ 设备：**
  - **一个DMAC多个设备**
  - **选择型：**
    - DMAC“软”选择响应设备
  - **多路型：“硬”选择响应设备**
    - 链式多路型
    - 独立请求多路型
- **分布式？集中式？**
  - **分布式：公共请求，链式多路型**
  - **集中式：独立请求，选择型**





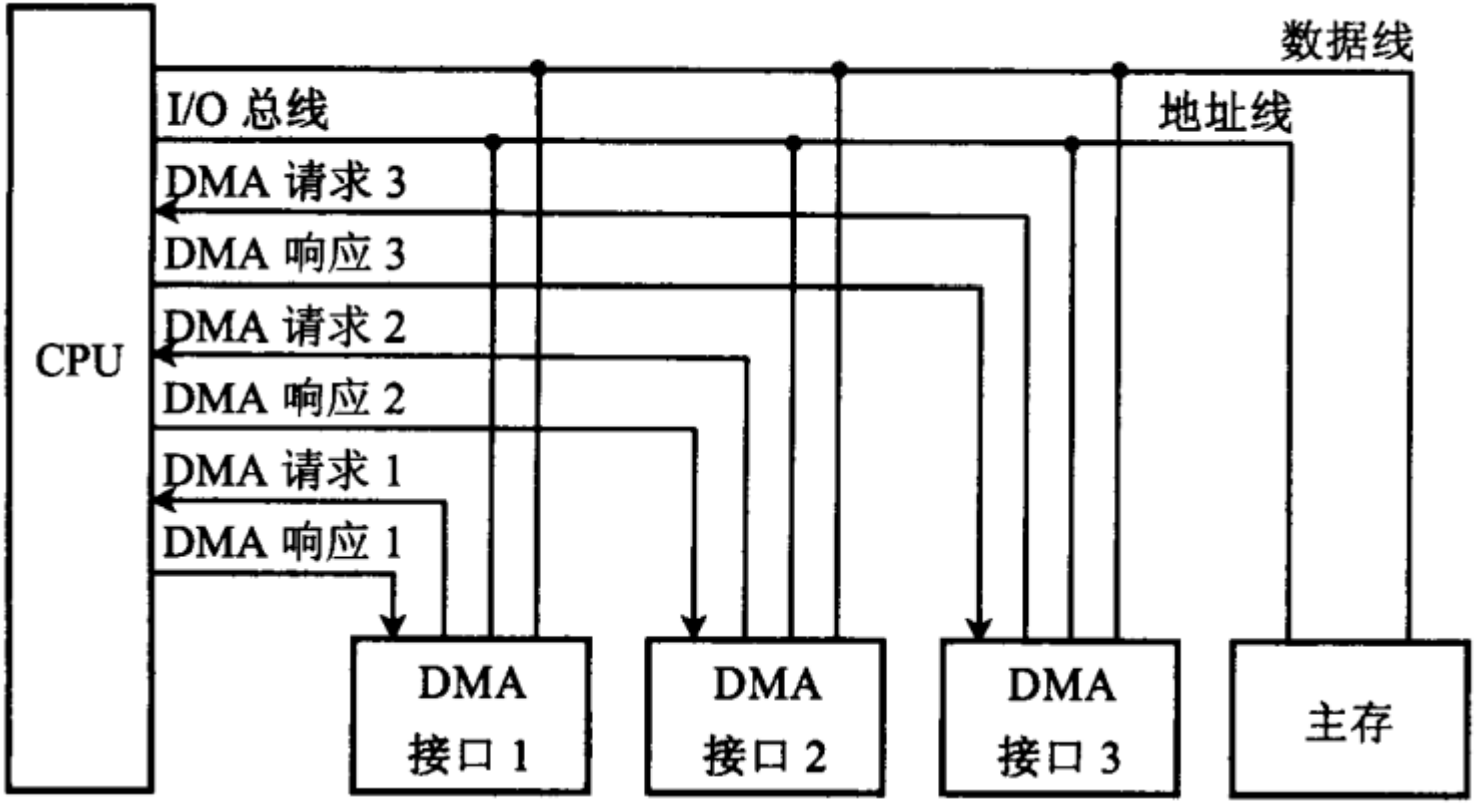
DMA ←→ CPU

- 公共请求方式



DMA ←→ CPU

- 独立请求方式



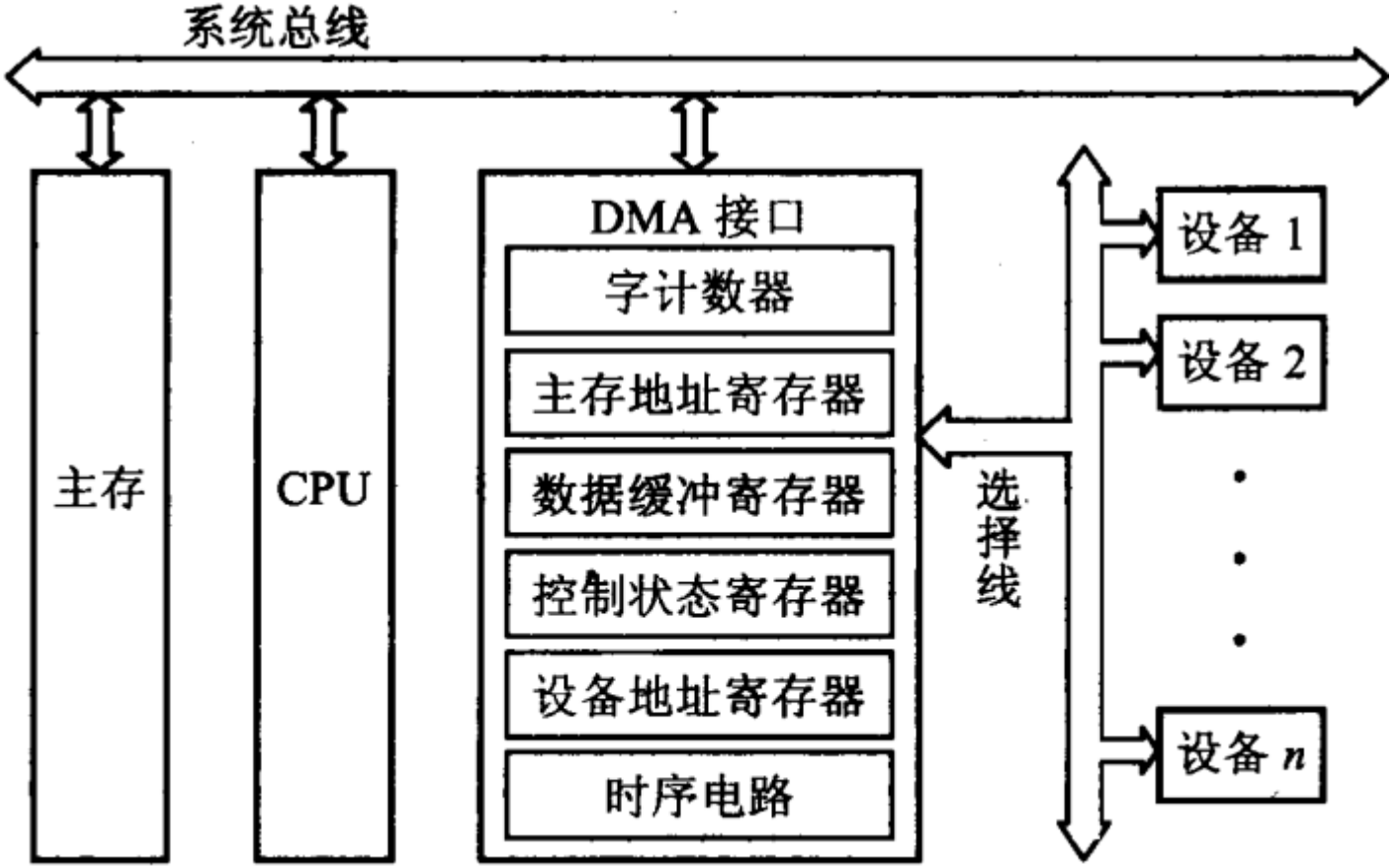
DMA接口的类型（与设备接口）

- **一个DMAC多个设备**
  - **选择型： DMAC“软”选择响应设备**
  - **多路型：“硬”选择响应设备**
    - 链式多路型
    - 独立请求多路型

- **分布式？集中式？**
  - **分布式：公共请求，链式多路型**
  - **集中式：独立请求，选择型**

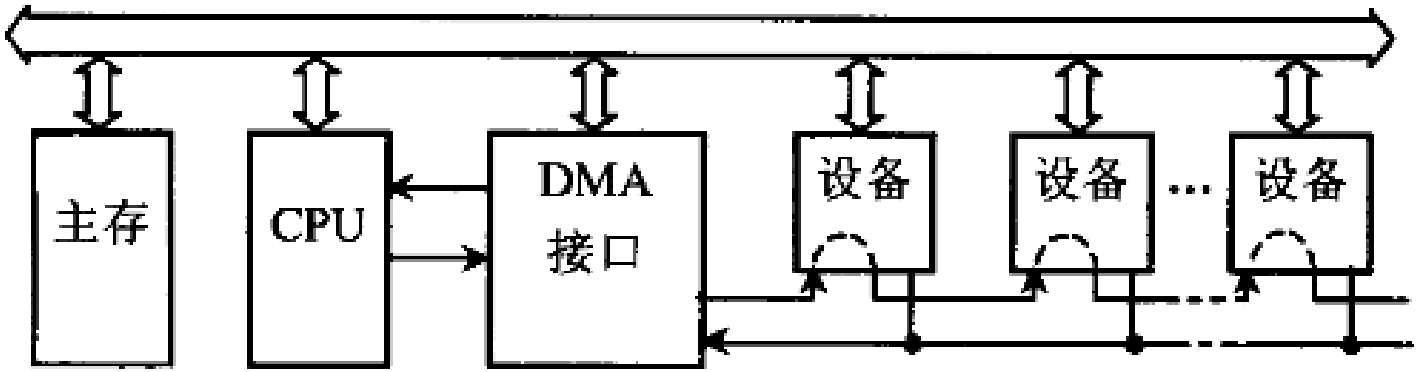
DMA接口的类型

- **选择型DMA接口的逻辑框图**

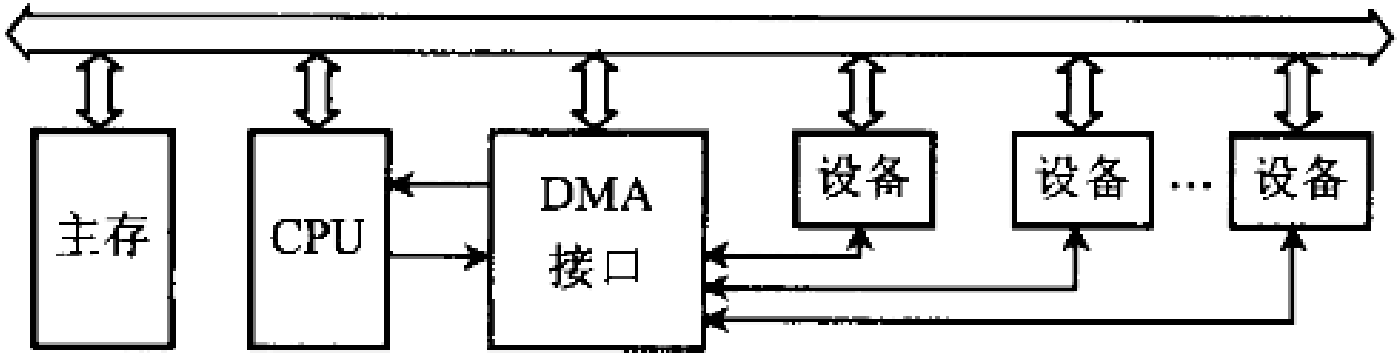


DMA接口的类型

- **多路型DMA接口的逻辑框图**



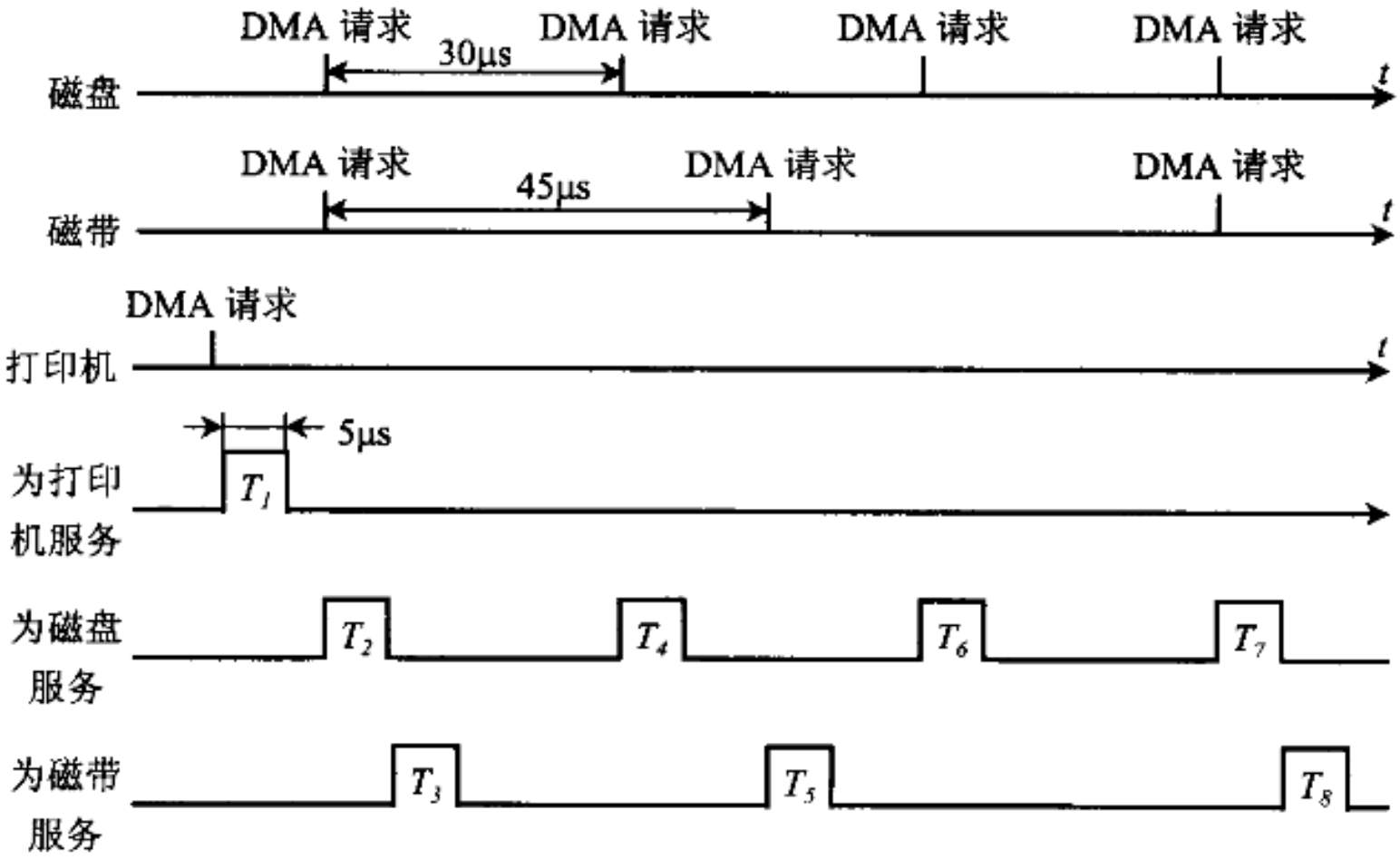
(a) 链式多路型 DMA 接口



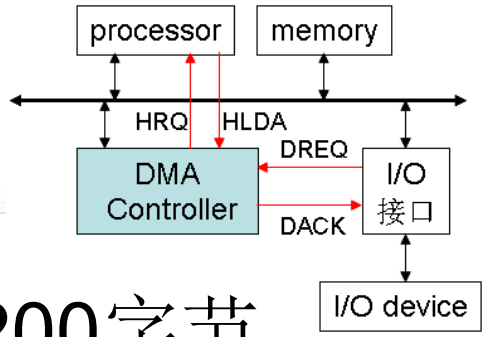
(b) 独立请求多路型 DMA 接口

DMA接口的类型

- 多路型DMA接口的工作原理



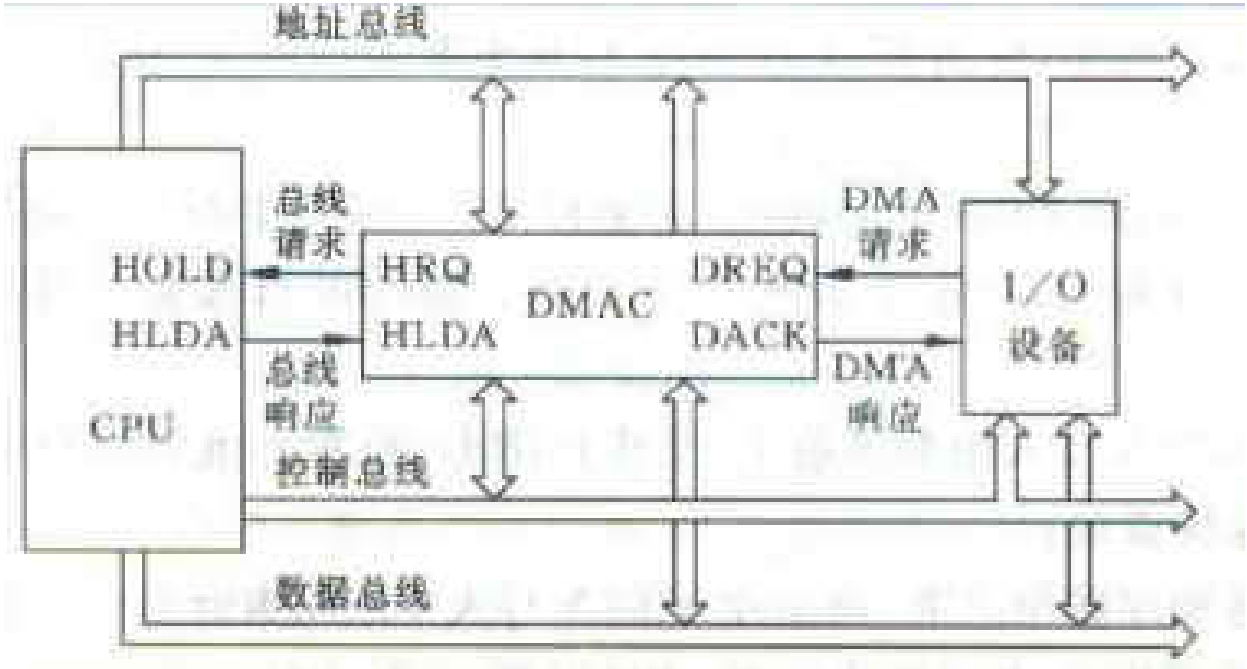
例



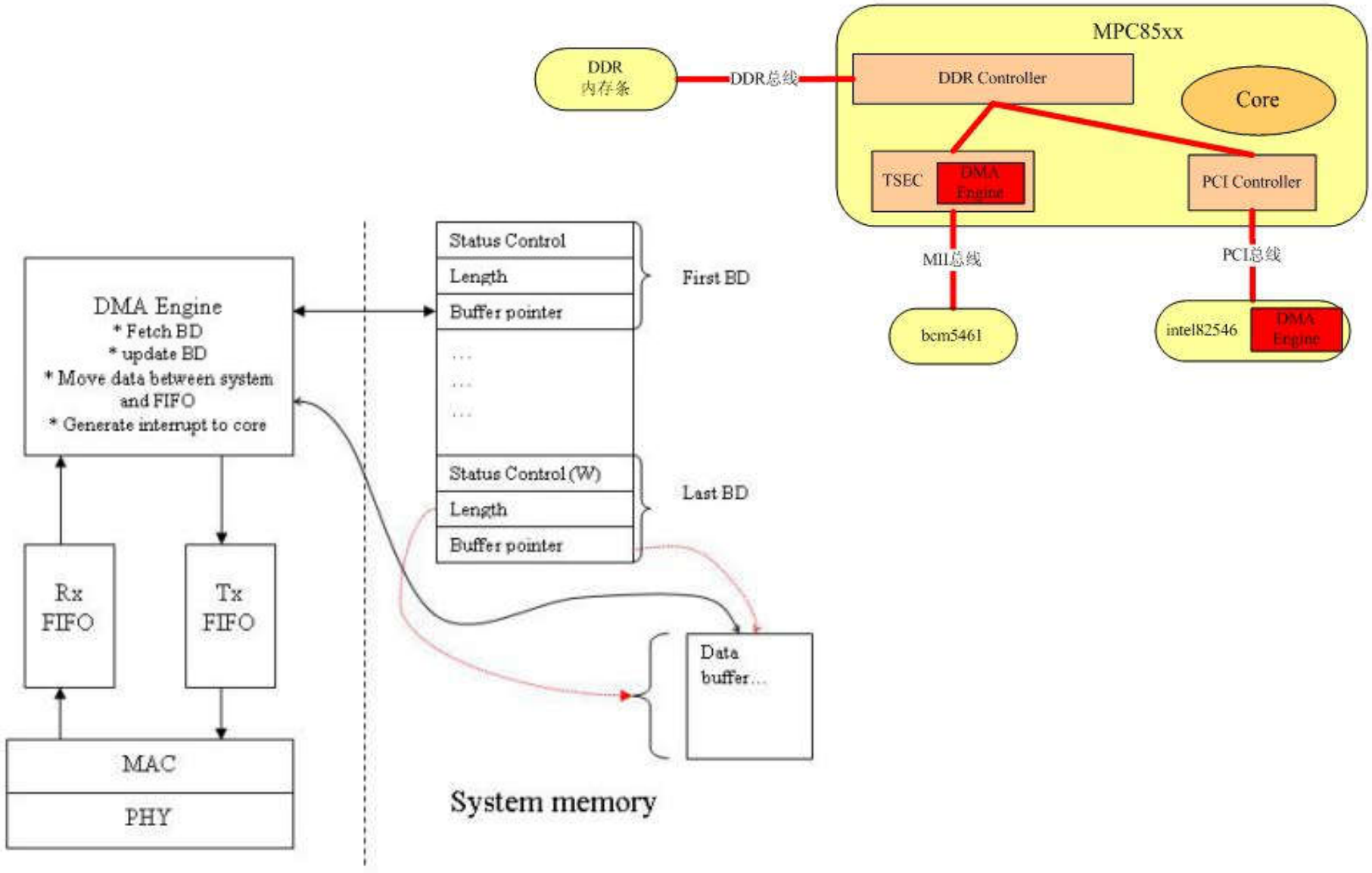
- 用DMA方式将串行通信口接收到的200字节数据存入以BUFFER为首地址内存区域
  - 初始化
    - DMAC初始化：向DMAC写入内存首地址，传输计数（200），传输方向（外设→内存），控制命令（允许DMA传输）等
    - 串行通信接口初始化：设置串行通信的参数，允许串行输入等
  - 数据传输
    - 串口每输入一个数据，自动进行DMA传输
    - 最后一个数据传输结束后，DMAC发出传输结束信号EOP。
  - 后处理
    - CPU可以通过查询知道传输已经结束，也可以利用EOP信号申请中断，在中断服务程序里进行后处理。

IDE接口

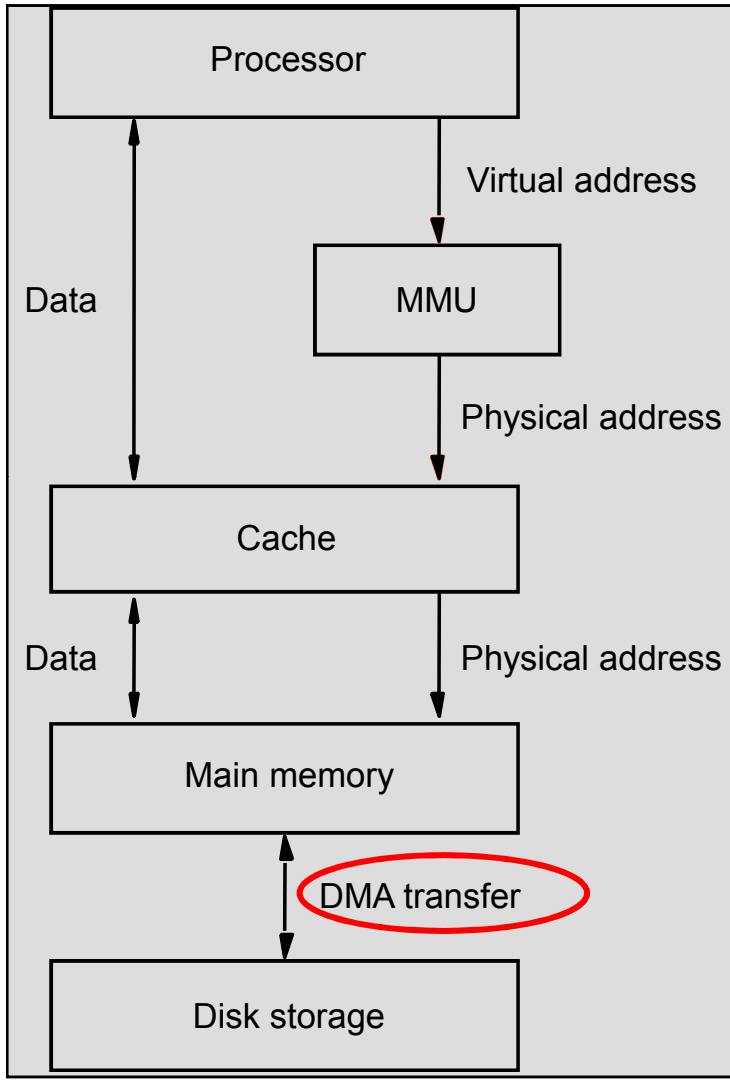
- PIO 模式： 3.3MB/s~16.6MB/s(PIO mode 0~4)
- DMA模式
  - Single-Word DMA
  - Multi-Word DMA，最大传输速率16.6MB/s
  - Ultra DMA Mode，133MB/秒



网络数据通信：网卡的DMA Engine



Virtual memory organization



- *Memory management unit (MMU) translates virtual addresses into physical addresses.*
- *If the desired data or instructions are in the main memory they are fetched as described previously.*
- *If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.*
- *MMU causes the operating system to bring the data from the secondary storage into the main memory.*

DMA与cache一致性

- 方式一：禁止DMA目标地址范围内的cache功能
  - 设置这些内存页uncached
- 方式二：Flush与Invalidate指令操作

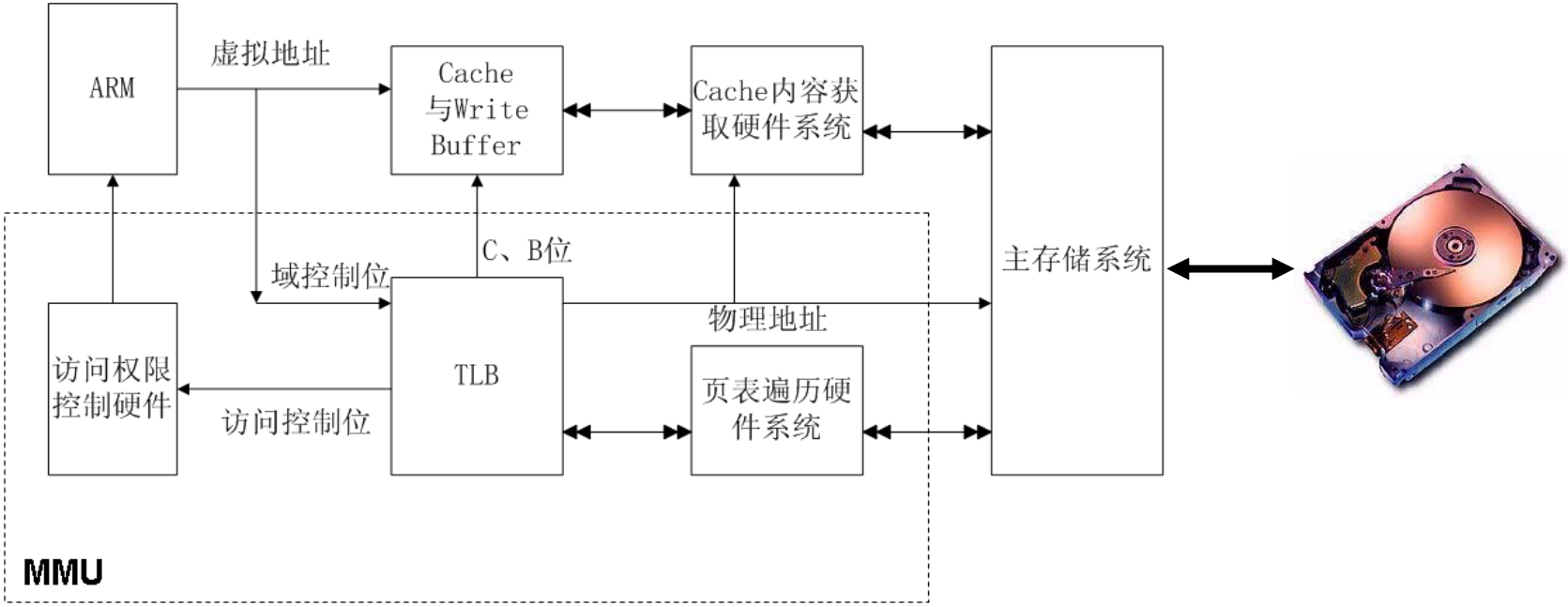


图 3.31　使用 cache Flush 操作情况示意图

图 3.32　使用 Cache Invalidate 操作情况示意图

DMA小结

- 用途
  - 批量高速数据传输
  - DRAM刷新控制
- 关键问题：总线争用
  - CPU暂停方式：x86系统采用
  - 周期窃取方式：本书称“广泛采用”？
  - 交替访问方式
- 工作过程：预处理、数据传输、后处理
- DMAC的功能
- 数据传输的DMA方式与INT方式的比较

中断控制方式与DMA方式有何异同?

DMA是程序中断传送技术的发展，在硬件逻辑机构的支持下，以更快的速度，更简便的形式传送数据，二者的区别为:

(1)中断方式通过程序实现数据传送，而DMA方式直接靠硬件来实现。

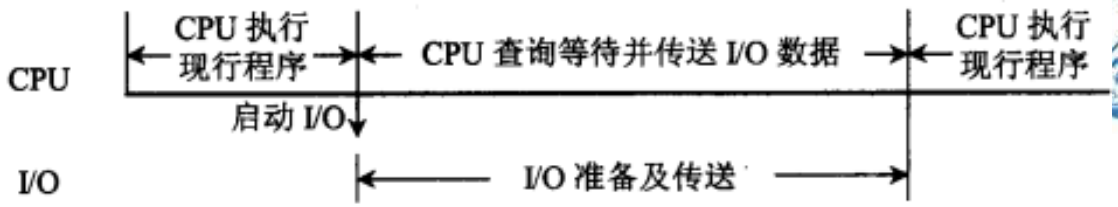
(2)CPU对中断的响应是在执行完一条指令之后，而对DMA的响应则可以在指令执行过程中的任何两个存储周期之间。

(3)中断方式具有数据传送和处理异常事件的能力，而DMA只能进行数据传送。

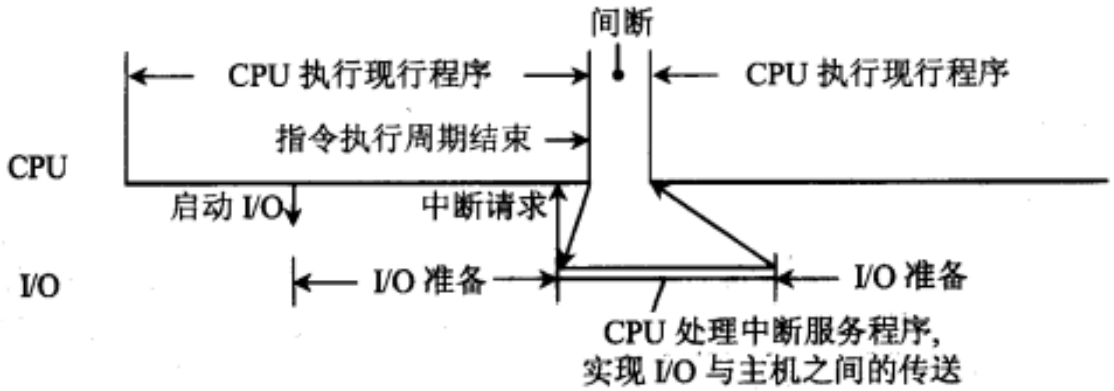
(4)中断方式必须切换程序，要进行CPU现场的保护和恢复，DMA仅挪用了一个存储周期，不改变CPU现场。

(5)DMA请求的优先权比中断请求高，CPU优先响应DMA请求,是为了避免DMA所连接的高速外设丢失数据。

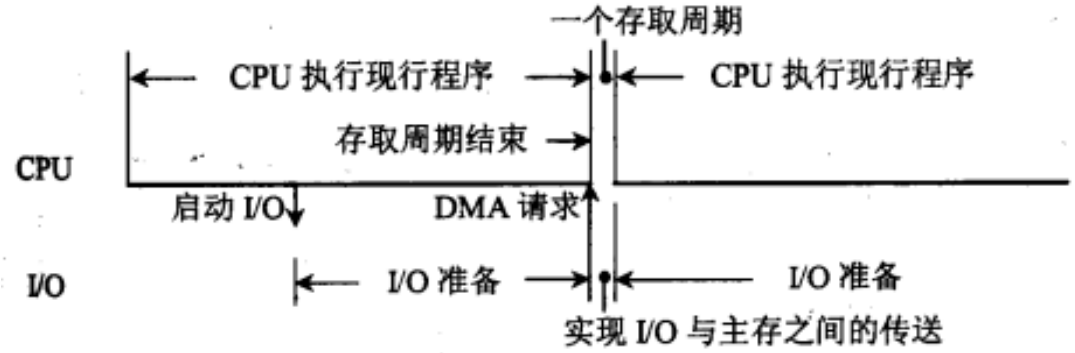
三种方式的CPU工作效率比较



(a) 程序查询方式

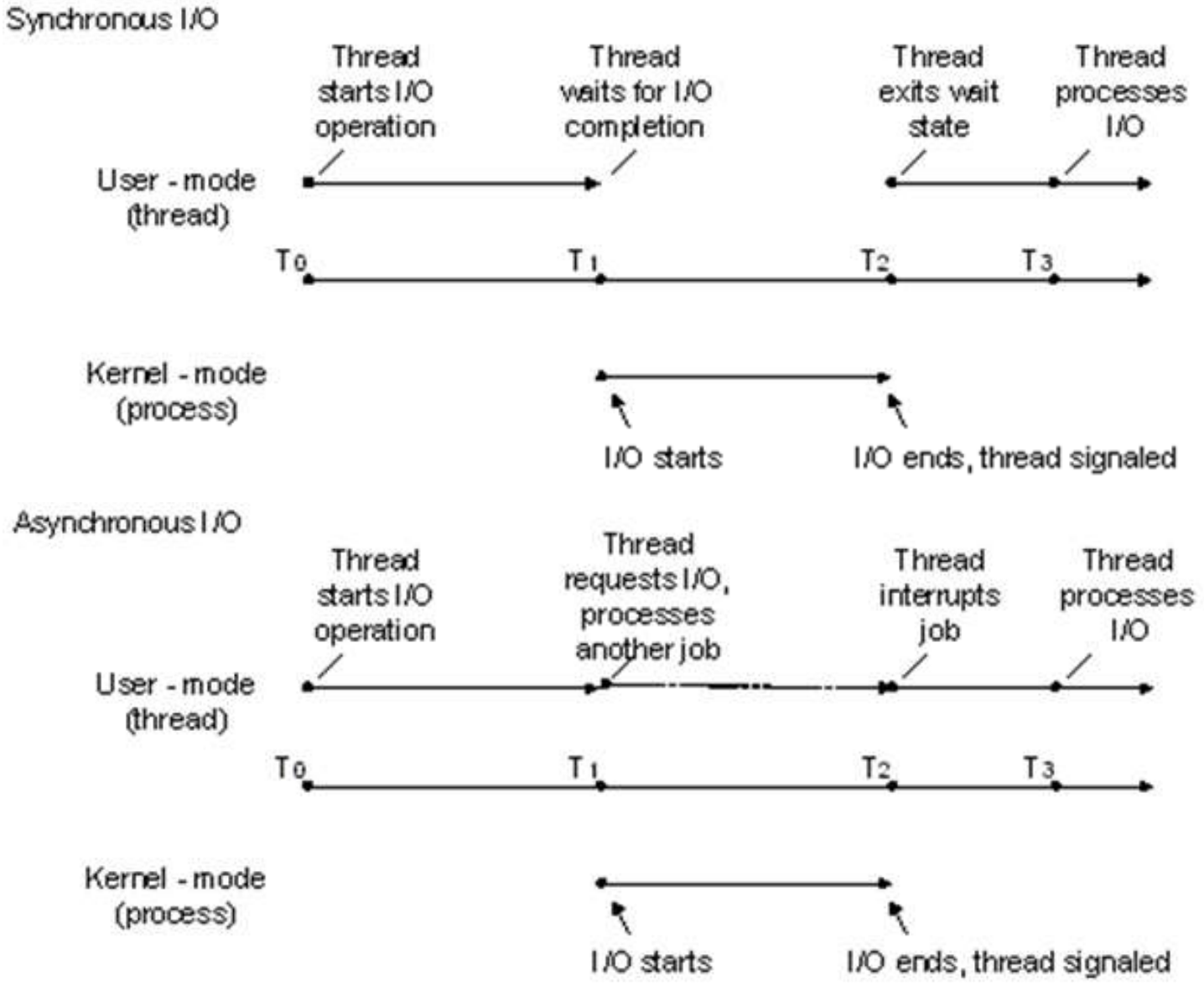


(b) 程序中断方式

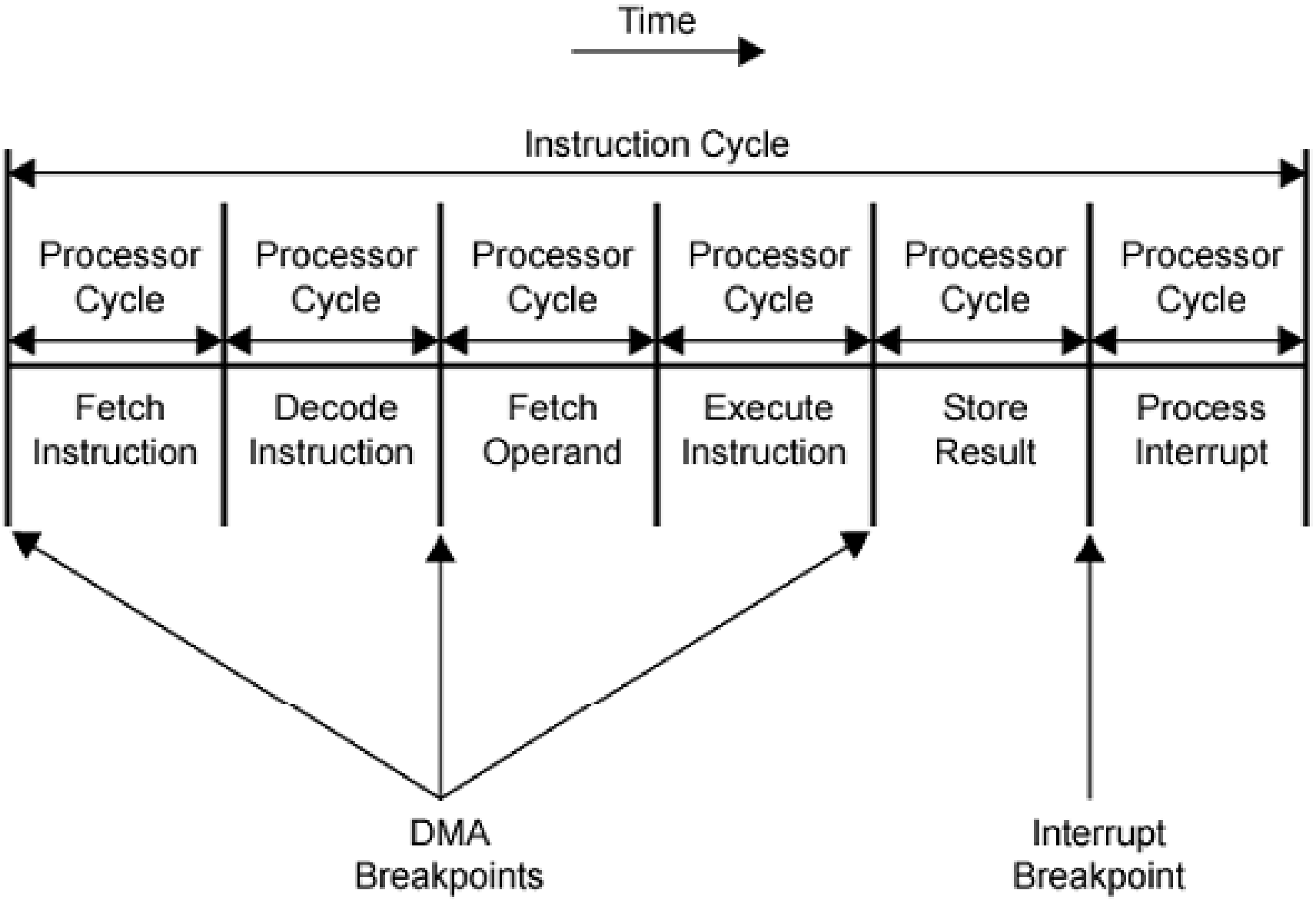


(c) DMA 方式

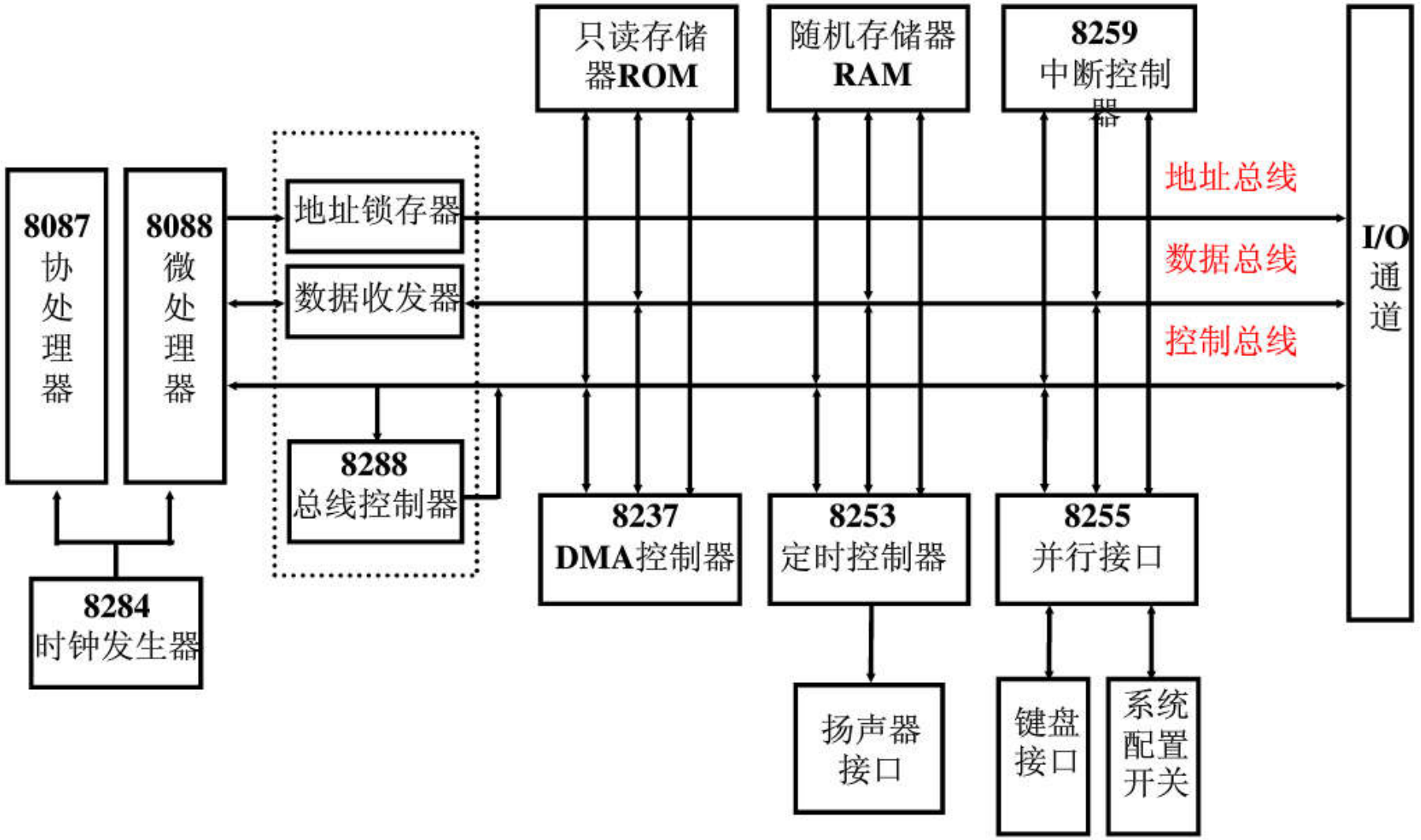
two types of I/O synchronization



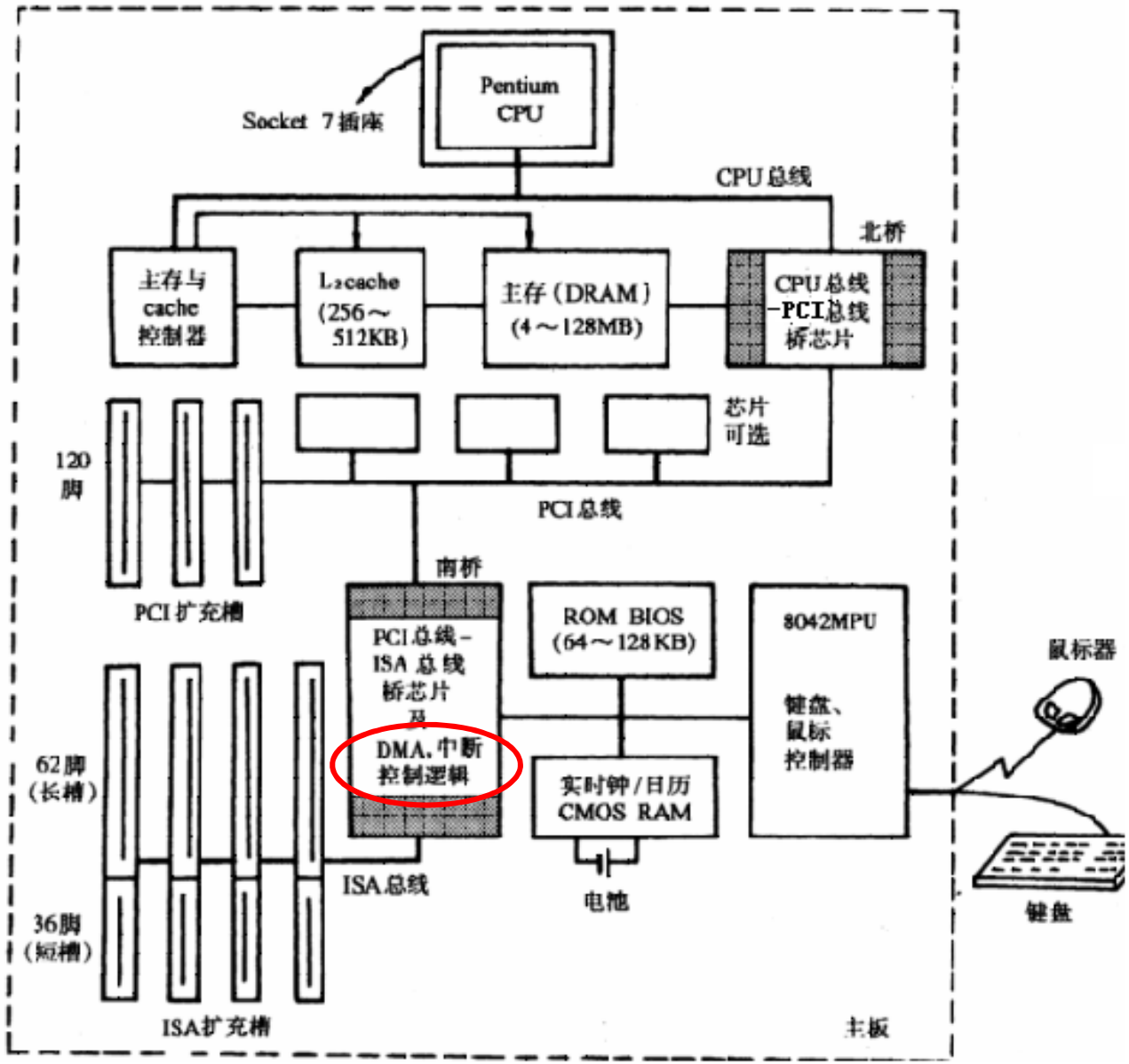
DMA and Interrupt Breakpoints



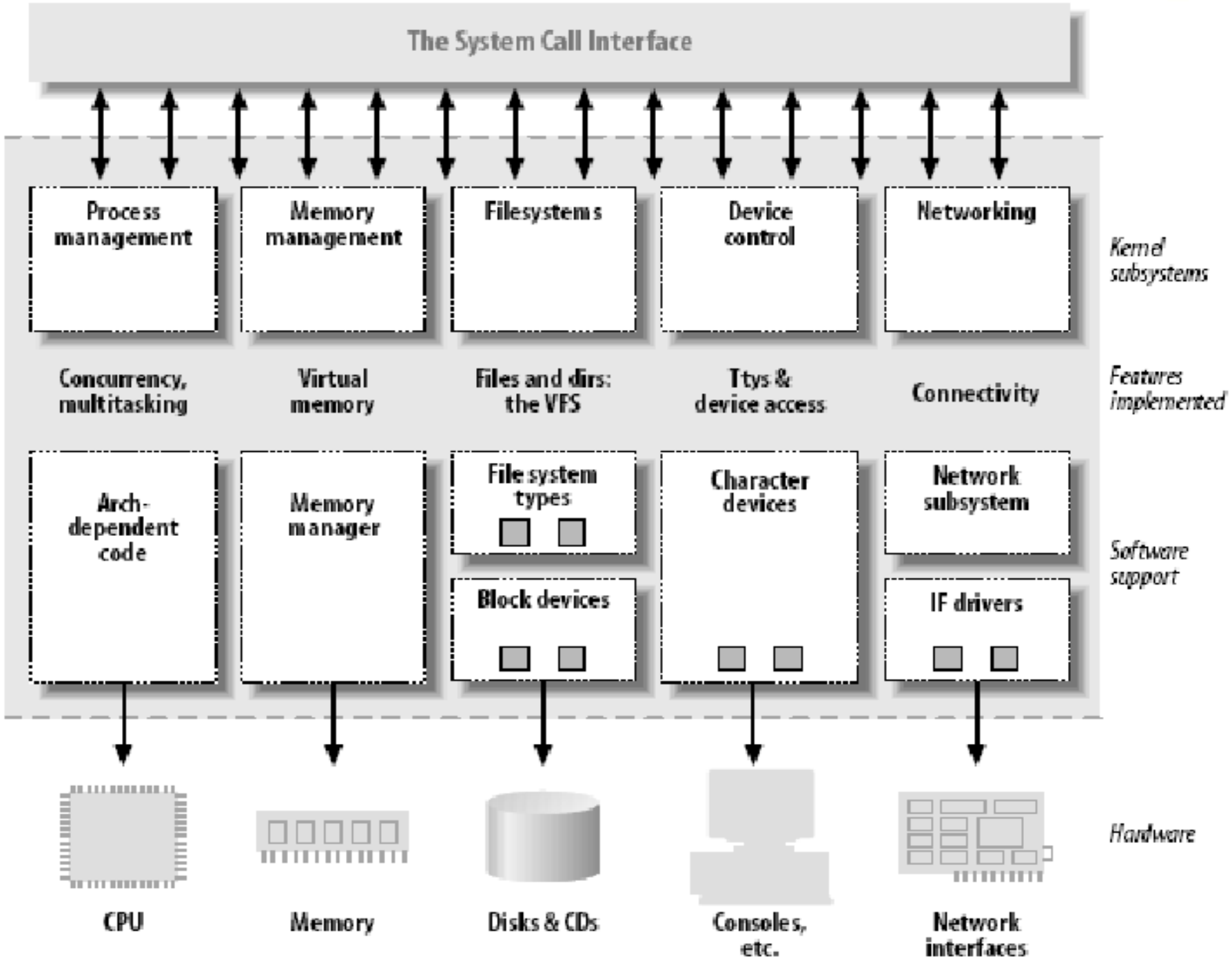
微机系统中的I/O控制器



Pentium主板结构（p427）

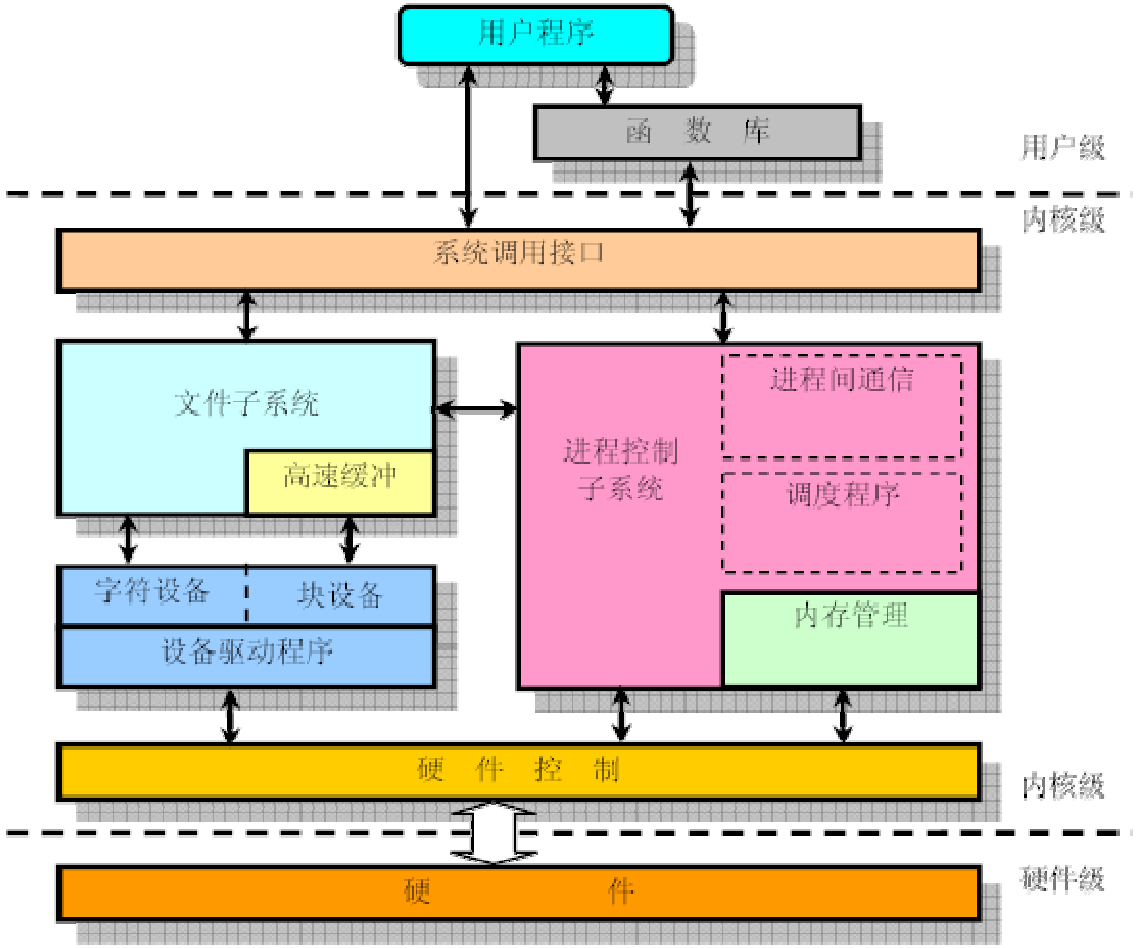


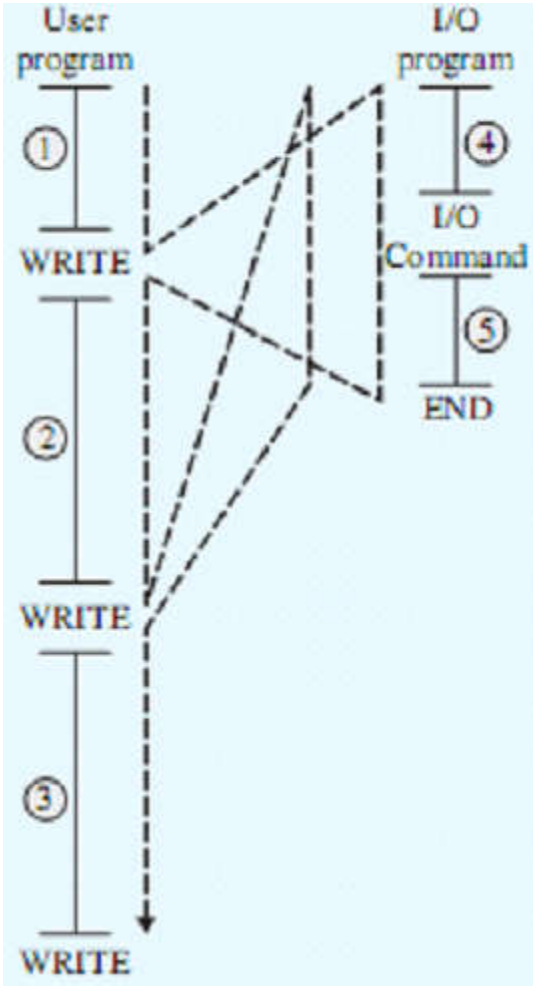
软硬件接口：设备驱动


The System Call Interface
Process management
Memory management
Filesystems
Device control
Networking
Kernel subsystems
Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
Ttys & device access
Connectivity
Features implemented
Arch-dependent code
Memory manager
File system types
Block devices
Character devices
Network subsystem
IF drivers
Software support
CPU
Memory
Disks & CDs
Consoles, etc.
Network interfaces
Hardware

设备驱动程序

- 操作特定设备的程序：字符设备、块设备
- 将所有设备映射成“文件”

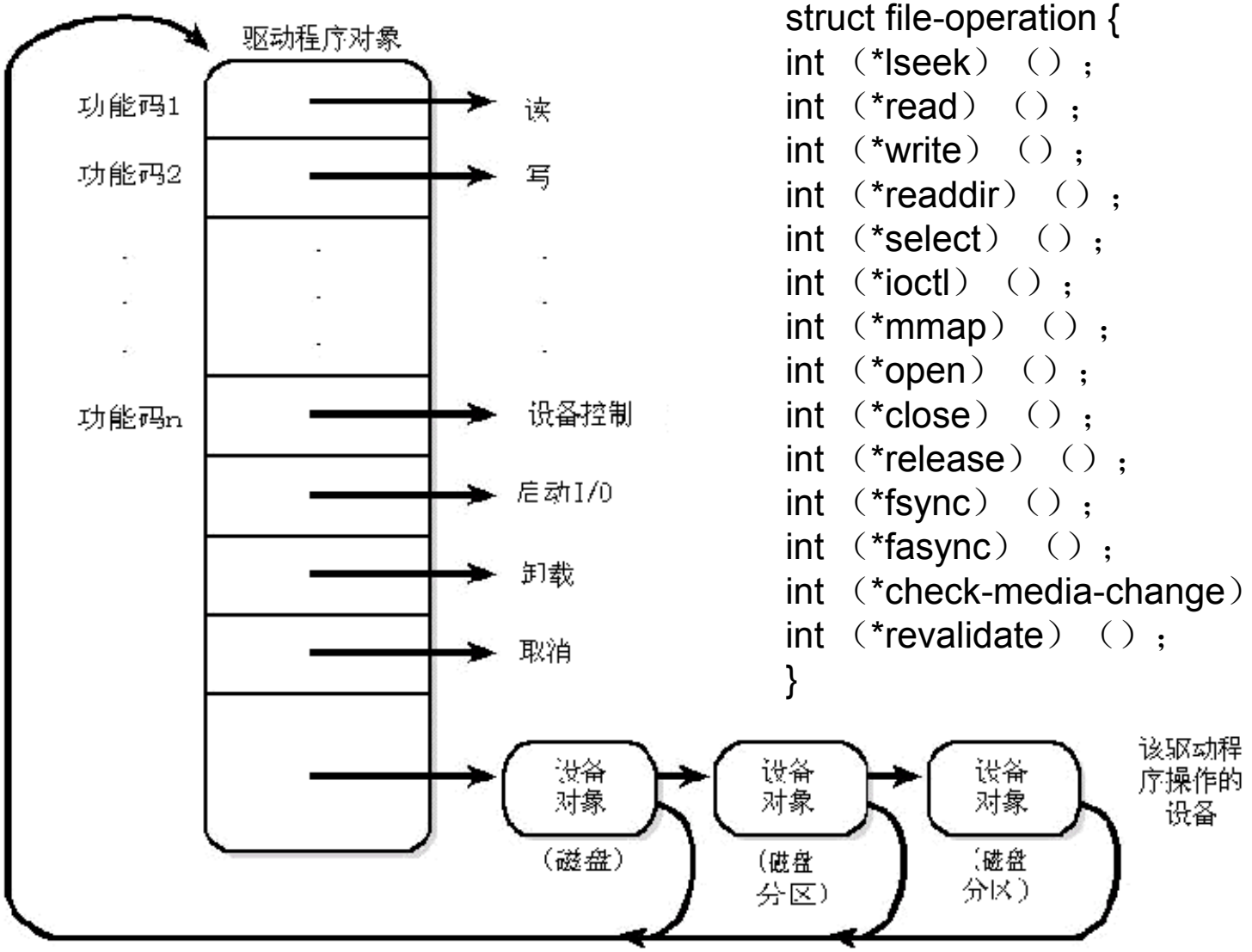




device driver的功能

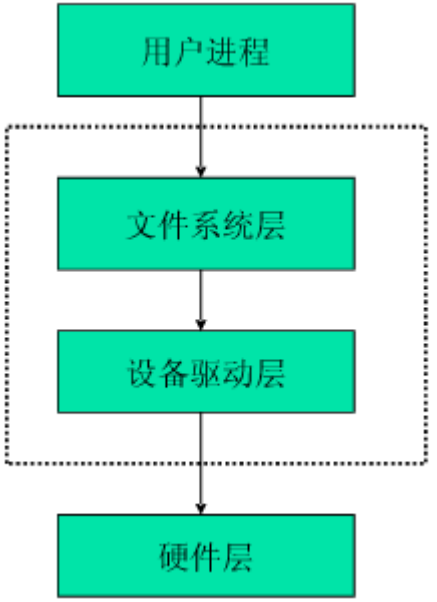
- **设备管理**
  - **Hardware Startup**, initialization of the hardware upon power-on or reset.
  - **Hardware Shutdown**, configuring hardware into its power-off state.
  - **Hardware Install**, allowing other software to install new hardware on-the-fly.
  - **Hardware Uninstall**, allowing other software to remove installed hardware on-the-fly.
  - **Hardware Disable**, allowing other software to disable hardware on-the-fly.
  - **Hardware Enable**, allowing other software to enable hardware on-the-fly.
- **读写操作**
  - **Hardware Read**, allowing other software to read data from hardware.
  - **Hardware Write**, allowing other software to write data to hardware.
- **并发控制**
  - **Hardware Acquire**, allowing other software to gain singular (locking) access to hardware.
  - **Hardware Release**, allowing other software to free (unlock) hardware.

驱动程序和设备




```
struct file-operation {
int （*lseek） （）；
int （*read） （）；
int （*write） （）；
int （*readdir） （）；
int （*select） （）；
int （*ioctl） （）；
int （*mmap） （）；
int （*open） （）；
int （*close） （）；
int （*release） （）；
int （*fsync） （）；
int （*fasync） （）；
int （*check-media-change） （）；
int （*revalidate） （）；
}
```

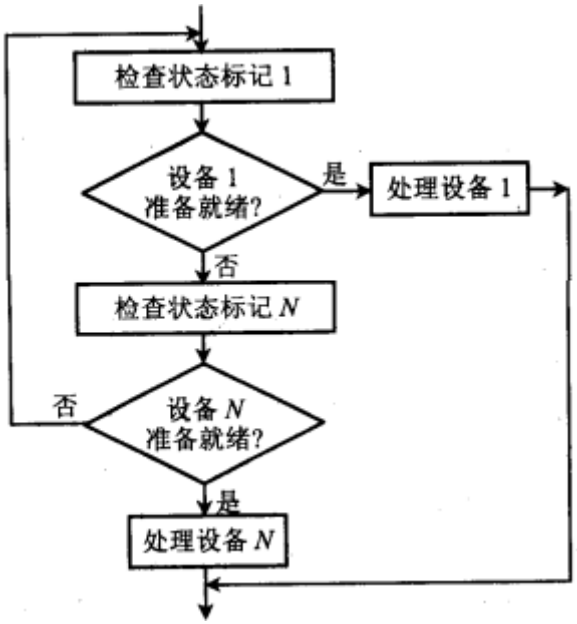
例：LED应用程序

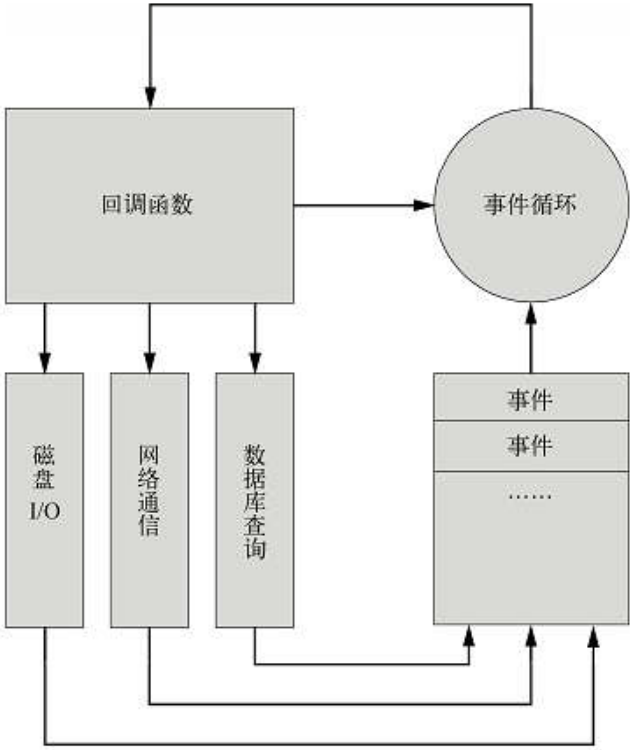



```
int main(void)
{    int fd;
     char led_on = 0x01;                    //待显示的数据

     fd = open("/dev/led/0", O_RDWR);  //打开led设备
     if(fd==-1)   {
       printf("can not open device\n");
       exit(1);
     }
     write(fd, &led_on, 1);                 //LED开
     close(fd);                             //关闭设备文件
     return 0;
}
```

I/O应用程序编程模型

- 多I/O并发：磁盘、网络、外设…
- 单线程I/O
  - 程序查询方式：顺序轮询式
  - 阻塞式I/O？
- 多线程
  - 每个线程负责一个I/O任务
  - 同步式I/O（阻塞式）
  - 线程切换开销大
- 事件驱动
  - 单线程
  - 异步I/O：事件队列循环

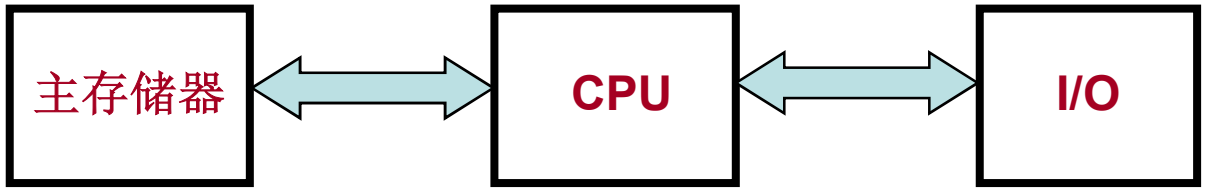




输入输出系统的发展

- 如何控制输入输出的过程
  - 如何解放CPU？
- 四个阶段（四种类型）
  - 以CPU为中心的阶段(早期阶段)
  - 接口模块和DMA阶段
  - 具有通道结构的阶段
  - 具有I/O处理机的阶段

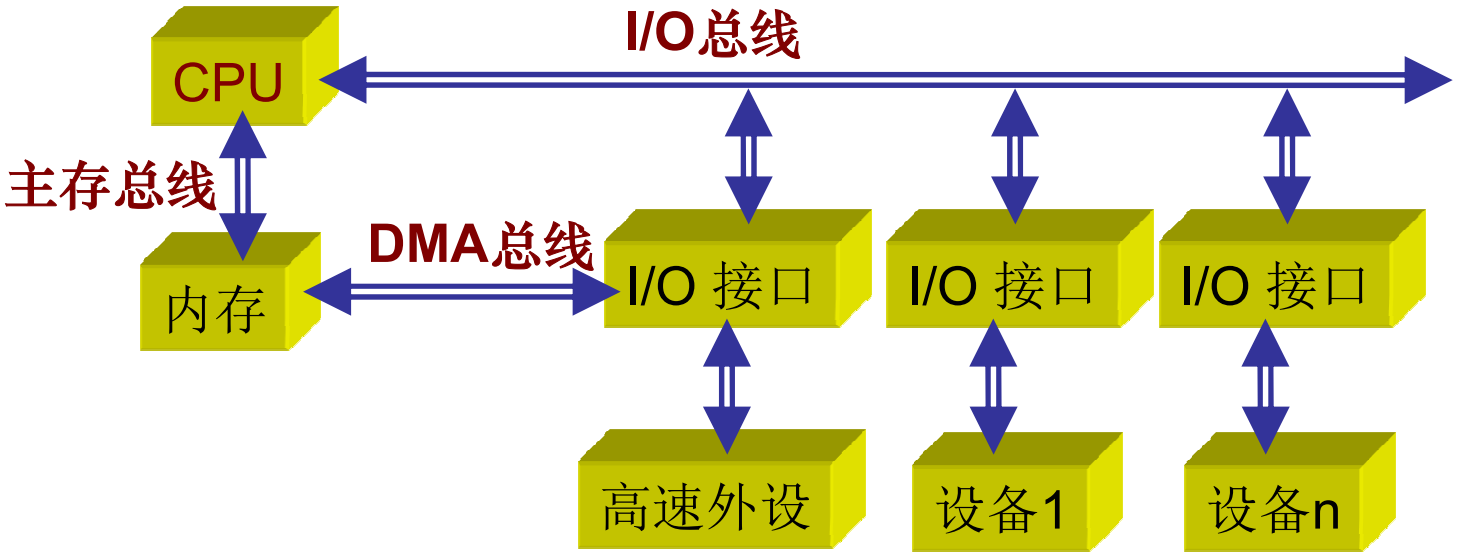
早期阶段



- I/O设备较少，I/O设备与主机交换信息都通过CPU
  - 这种交换方式延续了很长时间。
- 当时的I/O设备具有如下几个特点：
  - 每个I/O设备都必须配有一套独立的逻辑电路与CPU相连，**线路复杂**。
  - 输入输出过程穿插在CPU执行程序之中进行，当I/O与主机交换信息时，**CPU不得不停止各种运算**。
  - 每个I/O设备的逻辑控制电路与CPU的控制器紧密构成一个不可分割的整体，**增减或更换I/O设备十分困难**。

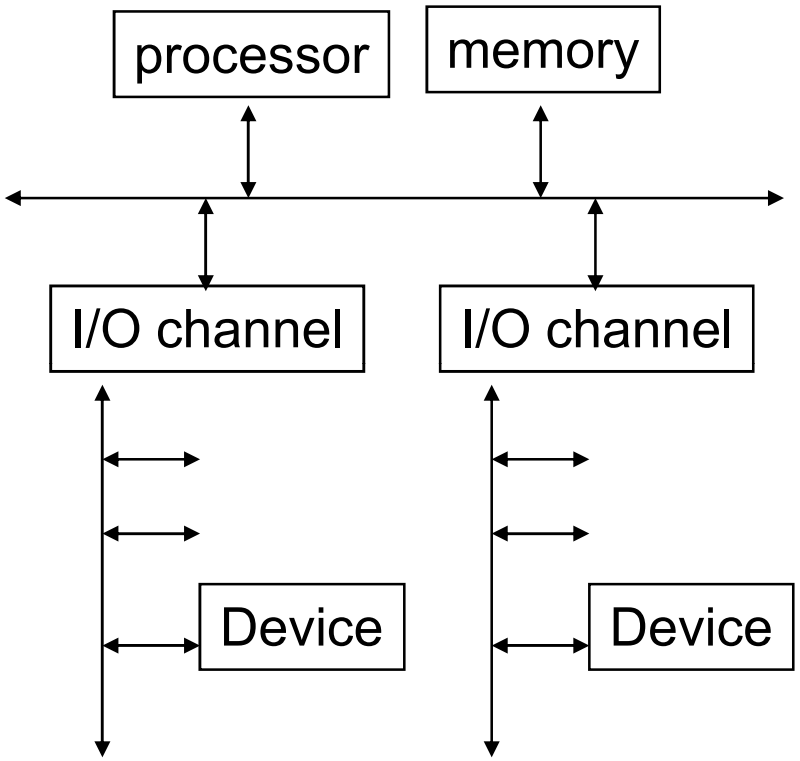
接口模块和DMA阶段

- 接口模块：控制、缓冲。。。
  - I/O设备与CPU可按并行方式工作
  - 在主机与I/O交换信息时，CPU要中断现行程序。
- DMA
  - I/O和主存之间增加一条数据通路。



具有通道结构的阶段

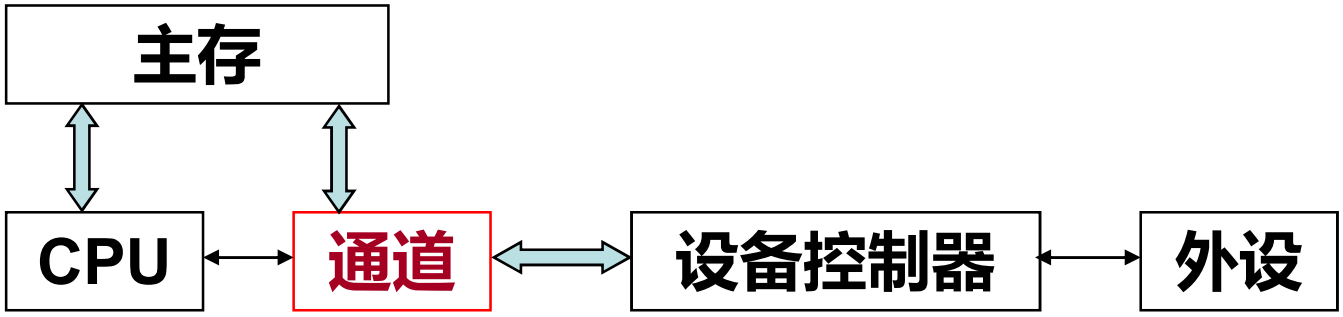
- 对**大型系统**，设备多，数据传输频繁，DMA造成的总线冲突仍然影响CPU的效率。
- 解决办法：采用I/O通道方式进行数据交换。
  - 执行专用指令，完成数据交换
  - **专用处理器IOP**，受主CPU控制（启停等）



具有通道结构的阶段

- **通道与DMA的区别**
  - **对CPU而言，通道比DMA具有更强的独立处理I/O的能力。**
    - DMA方式是通过DMA控制器控制总线，在外设和主存之间直接实现I/O传送；而通道通过执行通道程序进行I/O操作的管理。
  - **DMA控制器通常只控制一台或多台同类的高速设备，而通道可控制多台同类或不同类的设备。**

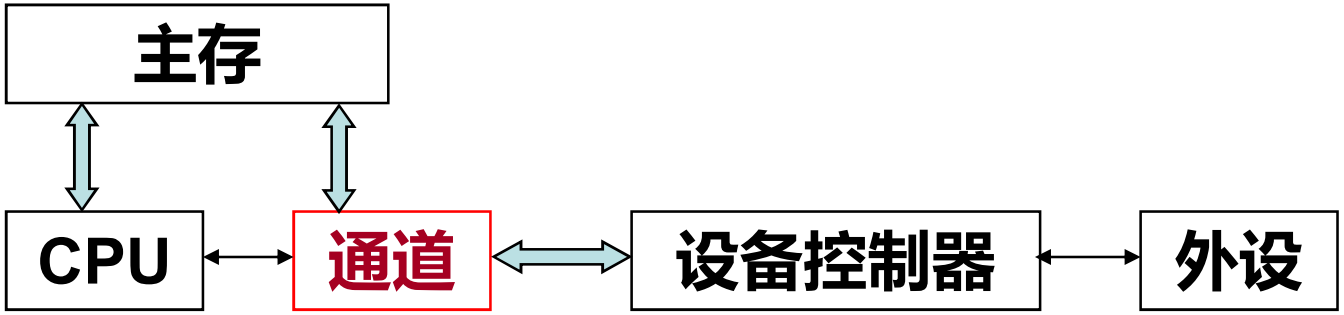
- **通道的位置**

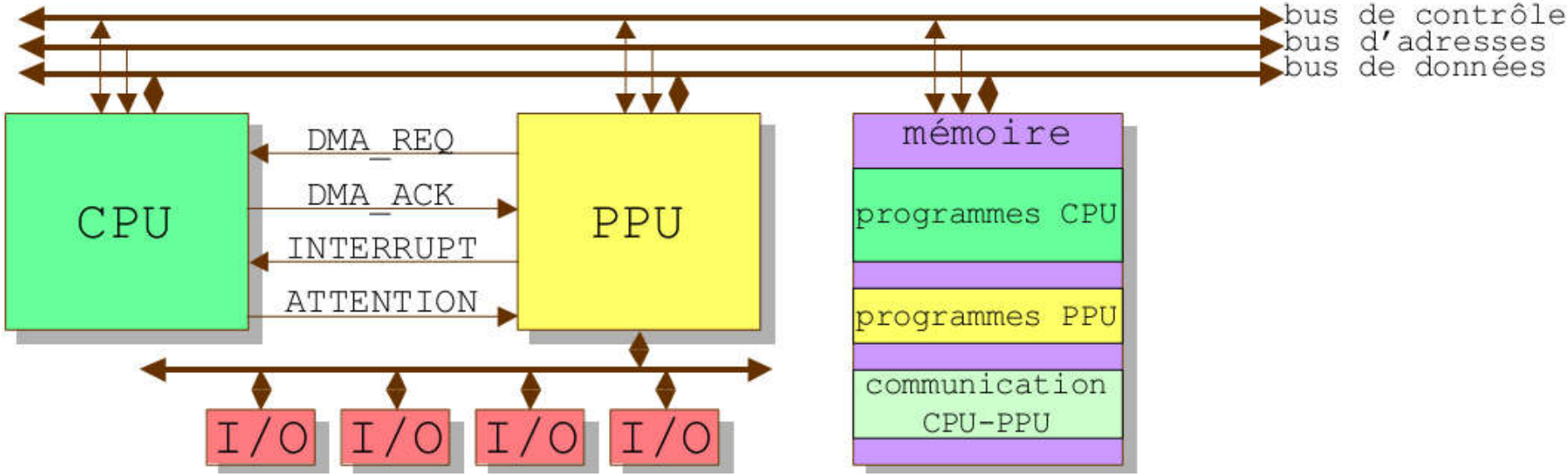


通道的工作过程

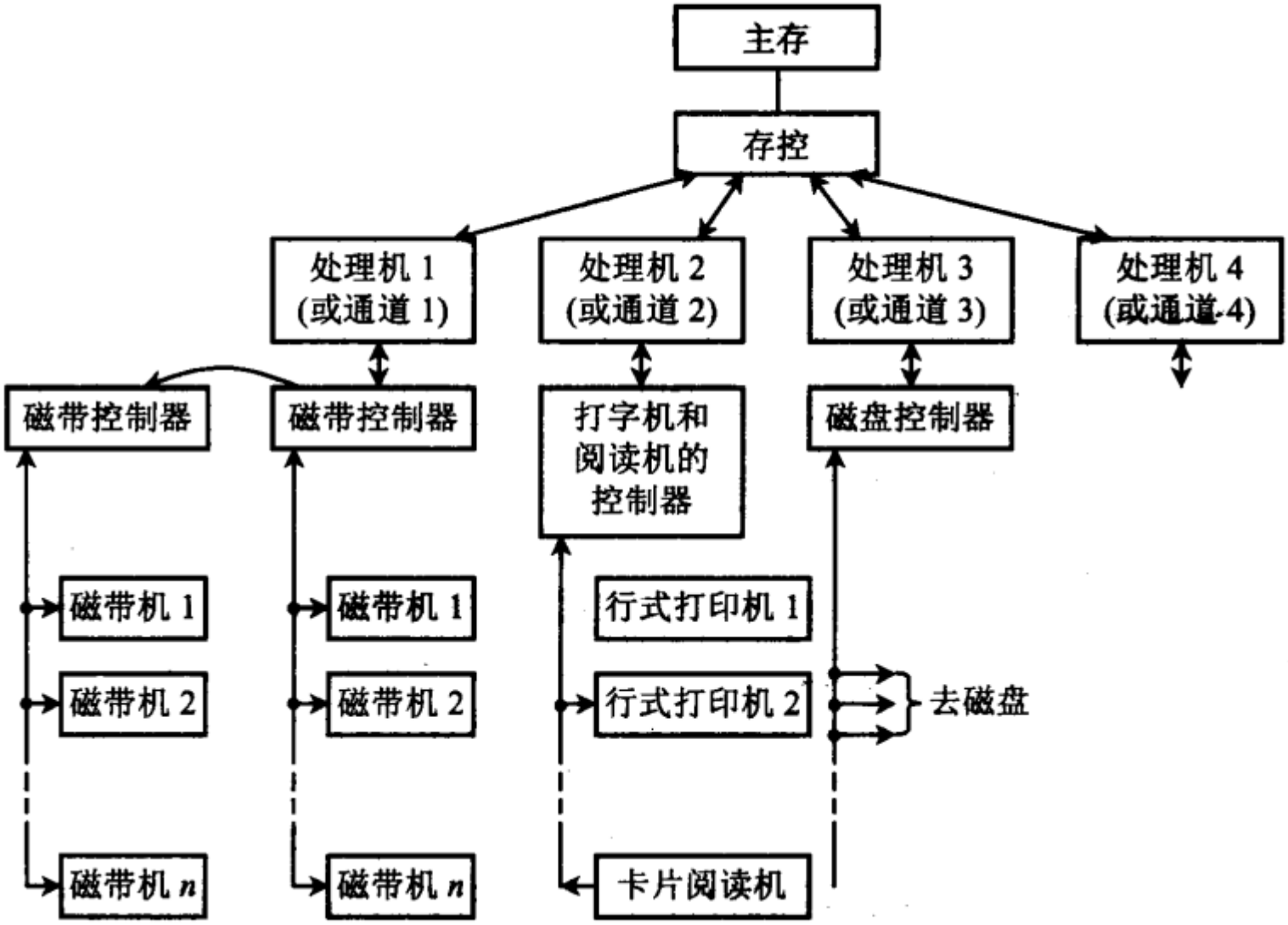
- **CPU对通道的控制通过如下两个方式进行。**
  - **CPU执行I/O指令**
    - 当需要进行I/O操作时，CPU按约定的格式准备好命令和数据，编制好通道程序，然后通过执行I/O指令(如：START I/O，TEST I/O ，HALT I/O等)来启动通道。
    - I/O指令应给出通道开始工作所需的全部参数，如：通道执行何种操作，在哪个通道和设备上进行操作等。
    - CPU启动道后，通道和外部设备将独立进行工作。
  - **处理来自通道的中断请求。**
    - 当通道和外设发生异常或结束处理时，通道采用“中断”方式向处理器报告。

通道架构PPU：peripheral processing units





具有通道的I/O硬件


主存
存控
处理机 1
(或通道 1)
处理机 2
(或通道 2)
处理机 3
(或通道 3)
处理机 4
(或通道 4)
磁带控制器
磁带控制器
打字机和
阅读机的
控制器
磁盘控制器
磁带机 1
磁带机 2
磁带机 n
磁带机 1
磁带机 2
磁带机 n
行式打印机 1
行式打印机 2
卡片阅读机
去磁盘

具有I/O处理机的阶段

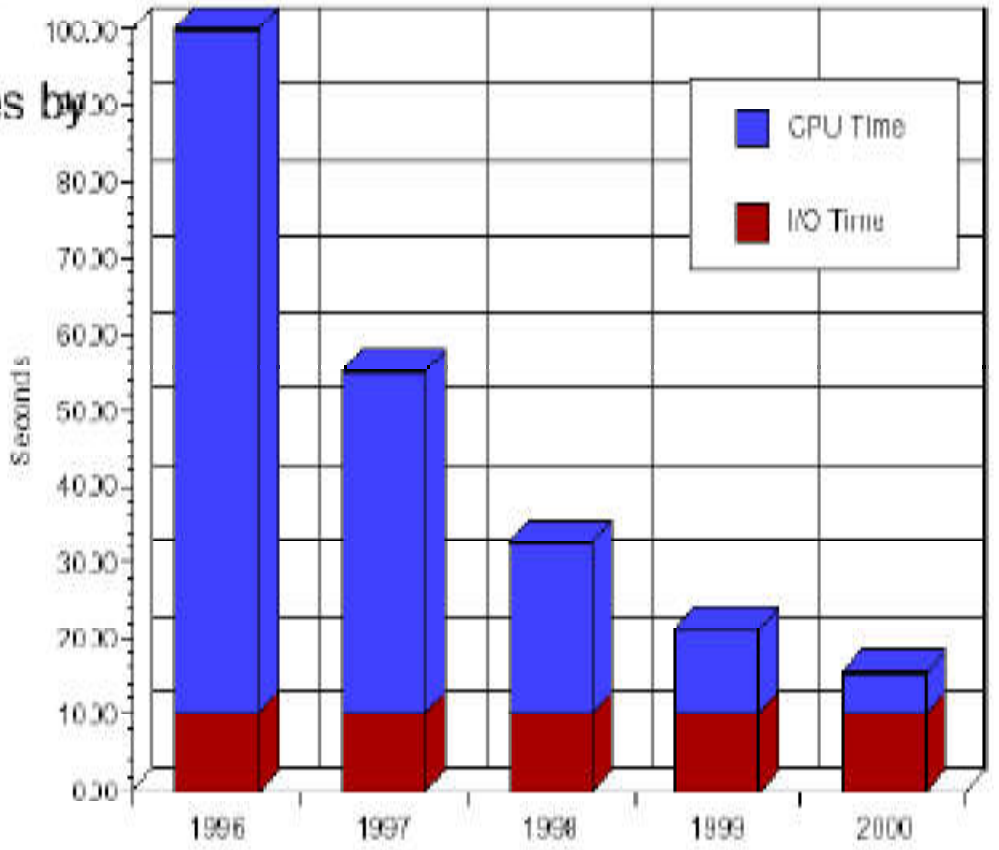
- **I/O处理机（IOP）**
  - **又叫外围处理机（Peripheral Processor Unit或PPU）**
  - **独立于主机工作**
    - 不仅可完成I/O通道要完成的I/O控制，还可完成码制转换、格式处理、数据块检错纠错等操作。

几种方式比较（见Stallings 6.7）

- **主要差异是I/O设备与CPU的连接方式和控制方式**
  - **CPU直接控制外设：简单的微控制器仍然采用**
  - **加入I/O控制器，CPU脱离具体外设操作**
  - **加入中断机制，CPU不再等待I/O**
  - **加入DMA机制，CPU与I/O并行**
  - **将I/O模块转换成专用的I/O处理器**
    - I/O处理器有自己的指令集，
    - CPU通过让I/O处理器执行专用的程序而完成一系列的I/O操作
  - **将I/O模块转换成专用的I/O处理机，独立管理多个外设**

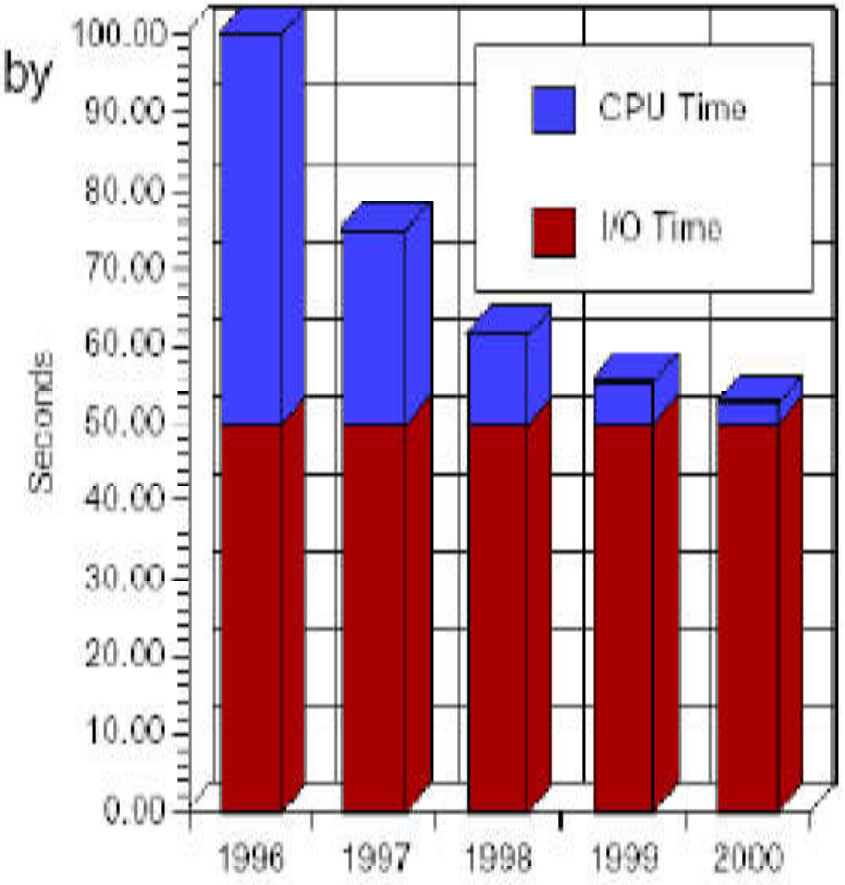
Application Performance

- 1996 - 1997
  - CPU performance improves by
    - N = 400/200 = 2
  - program performance improves by
    - N = 100/55 = 1.81
- 1997 - 1998
  - CPU performance - factor of 2
  - program performance
    - N = 55/32.5 = 1.7
- 1998 - 1999
  - CPU performance - factor of 2
  - program performance
    - N = 32.5 / 21.25 = 1.53
- 1999 - 2000
  - CPU Performance - factor of 2
  - program performance
    - N = 21.25 / 15.6 = 1.36

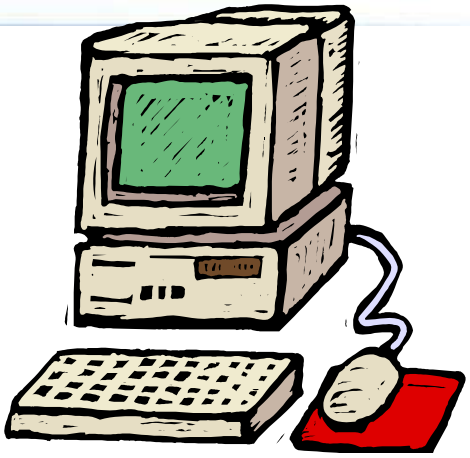


Performance for Web Surfing

- Assume 50 seconds CPU & 50 seconds I/O
- 1996 - 1997
  - CPU performance improves by
    - N = 400/200 = 2
  - program performance improves by
    - N = 100/75 = 1.33
- 1997 - 1998
  - CPU performance - factor of 2
  - program performance
    - N = 75/62.5= 1.2
- 1998 - 1999
  - CPU performance -f actor of 2
  - program performance
    - N = 62.5/56.5 = 1.11

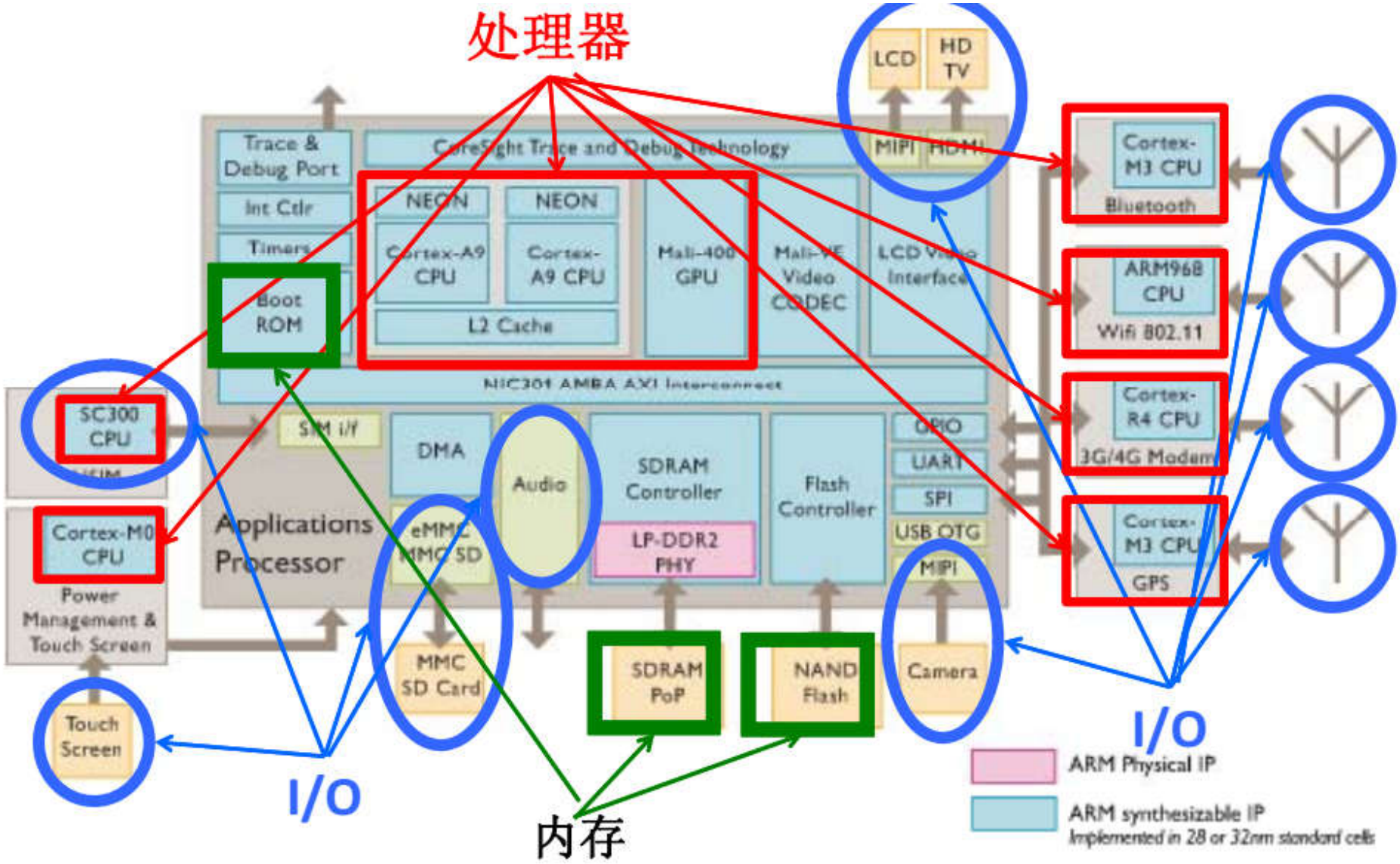


5.2 外部设备



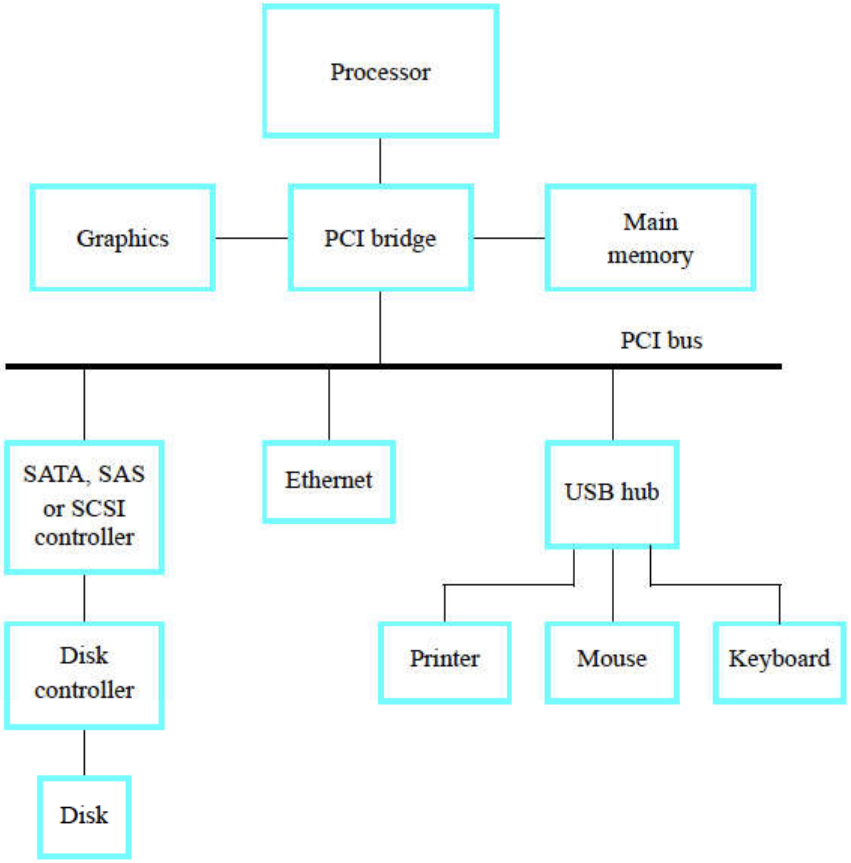
1. 输入设备：键盘、鼠标、触摸屏、光笔、画笔与图形板、图像输入设备
2. 输出设备：显示器、打印机
3. 其他外部设备：终端、A/D和D/A转换器、汉字处理设备
4. 多媒体技术
5. 新型人机交互界面

iPhone

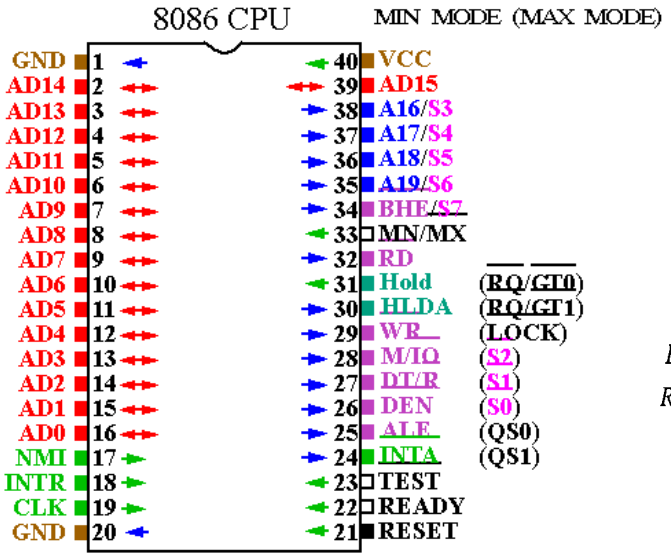

处理器
LCD
HD TV
MIPI
HDMI
Trace & Debug Port
CoreSight Trace and Debug Technology
Int Ctlr
Timers
Boot ROM
NEON
NEON
Cortex-A9 CPU
Cortex-A9 CPU
L2 Cache
Mali-400 GPU
Mali-VE Video CODEC
LCD Video Interface
NIC301 AMBA AXI Interconnect
SC300 CPU
SIM i/f
DMA
Audio
SDRAM Controller
LP-DDR2 PHY
Flash Controller
GPIO
UART
SPI
USB OTG
MIPI
Applications Processor
eMMC MMC SD
Cortex-M0 CPU
Power Management & Touch Screen
Touch Screen
MMC SD Card
SDRAM PoP
NAND Flash
Camera
Cortex-M3 CPU
Bluetooth
ARM968 CPU
Wifi 802.11
Cortex-R4 CPU
3G/4G Modem
Cortex-M3 CPU
GPS
I/O
I/O
内存
ARM Physical IP
ARM synthesizable IP
Implemented in 28 or 32nm standard cells

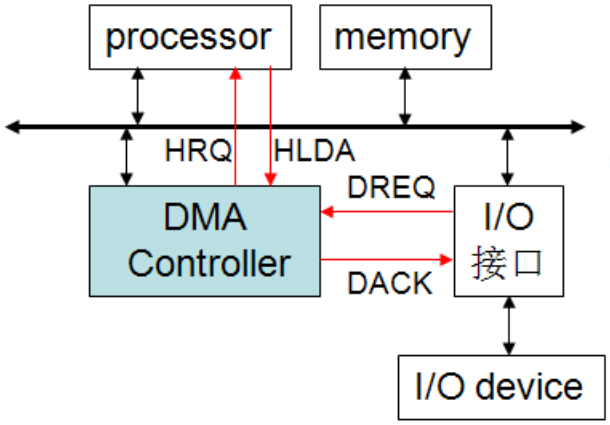
小结

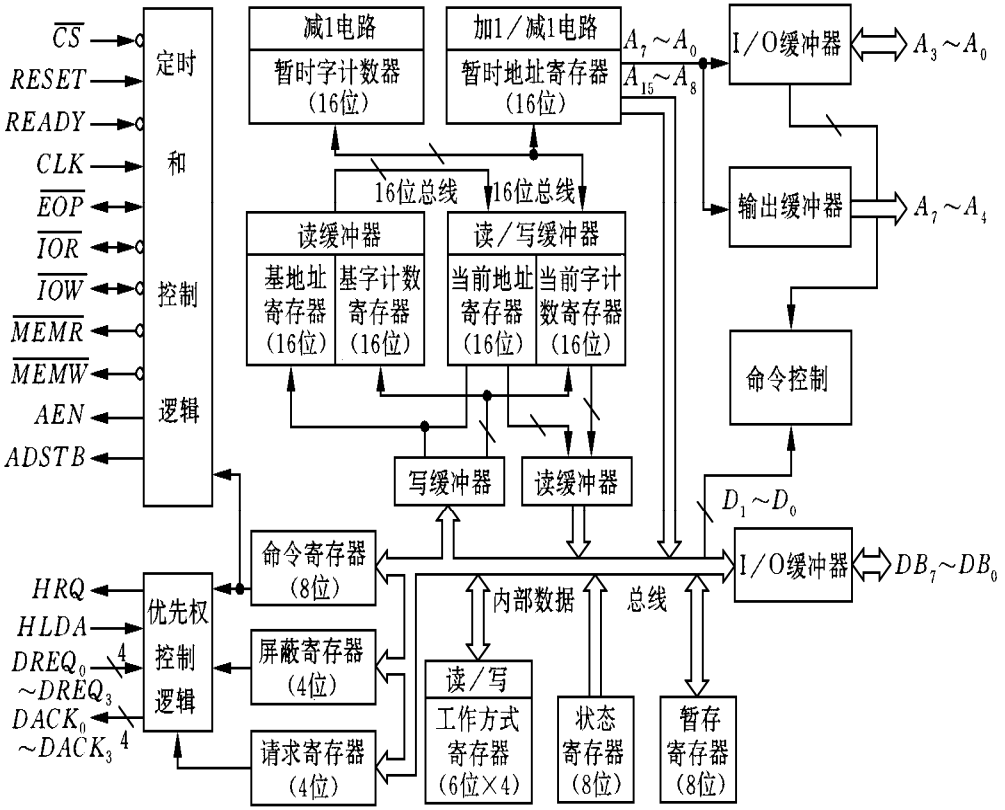
- 内容
  - 外设原理
  - I/O系统组成
  - I/O接口的基本特征
    - 编址方式
    - 同步方式
    - 控制方式
- 要求
  - 基本工作原理、流程、结构框图
- 解放CPU：降低CPU利用率！
- 作业
  - 5.4、5.8、5.11、8.23
  - 仿真（可选）



系统互连

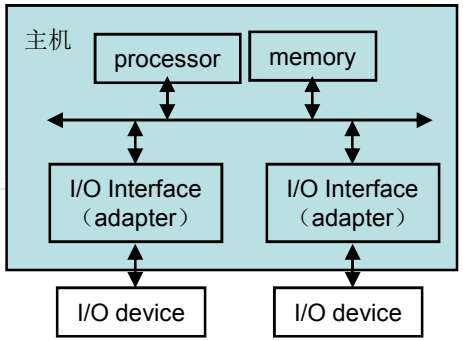


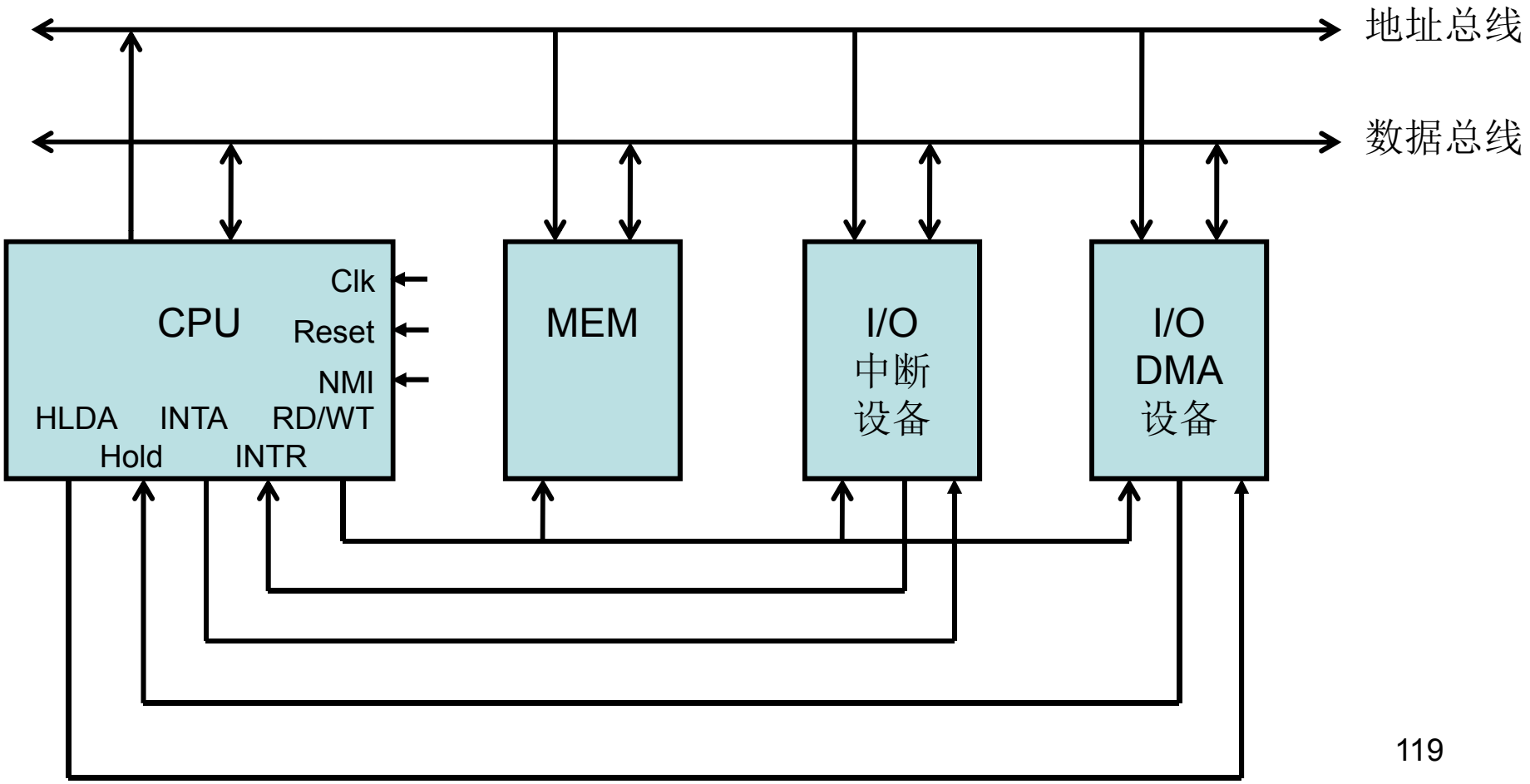




DMAC8237

Big Picture仿真：层次化





休息是为了走更远的路！