第二章

并发进程

方　钰

主 要 内 容

2.1 进程的基本概念

2.2 UNIX的进程

2.3 中断的基本概念及UNIX中断处理

2.4 进程通信

什么是中断?

一种设备控制方式

一种外设的数据传输方式

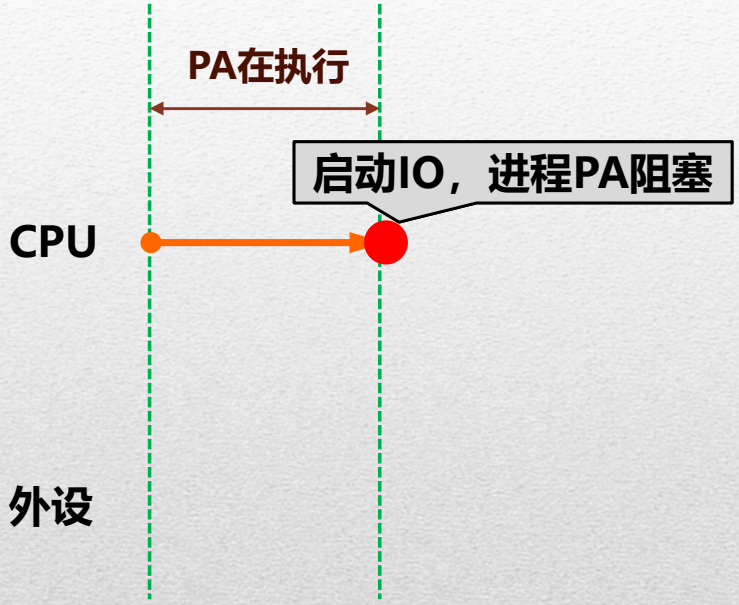
CPU不可能时刻查询外设的工作状态。。。

什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。

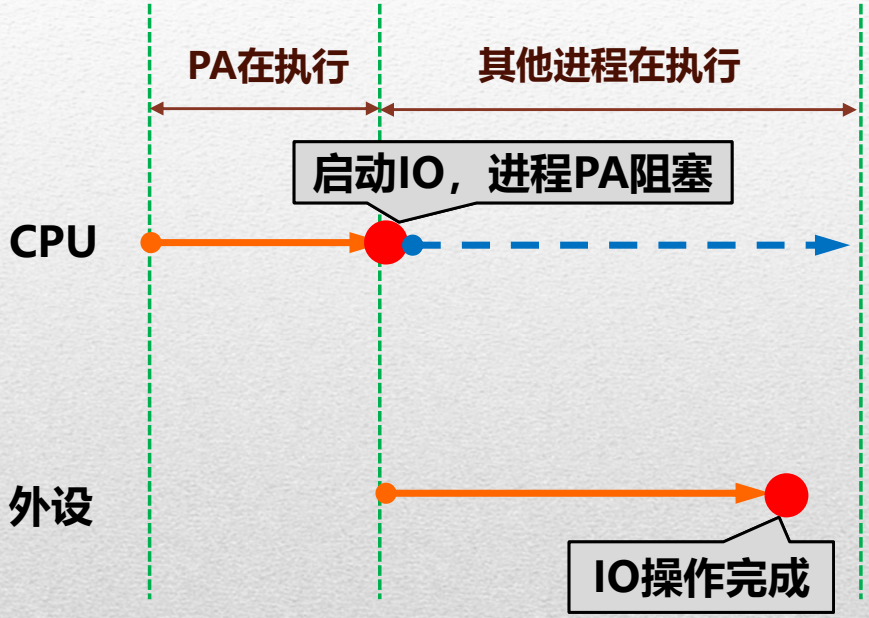


什么是中断?

一种设备控制方式

一种外设的数据传输方式

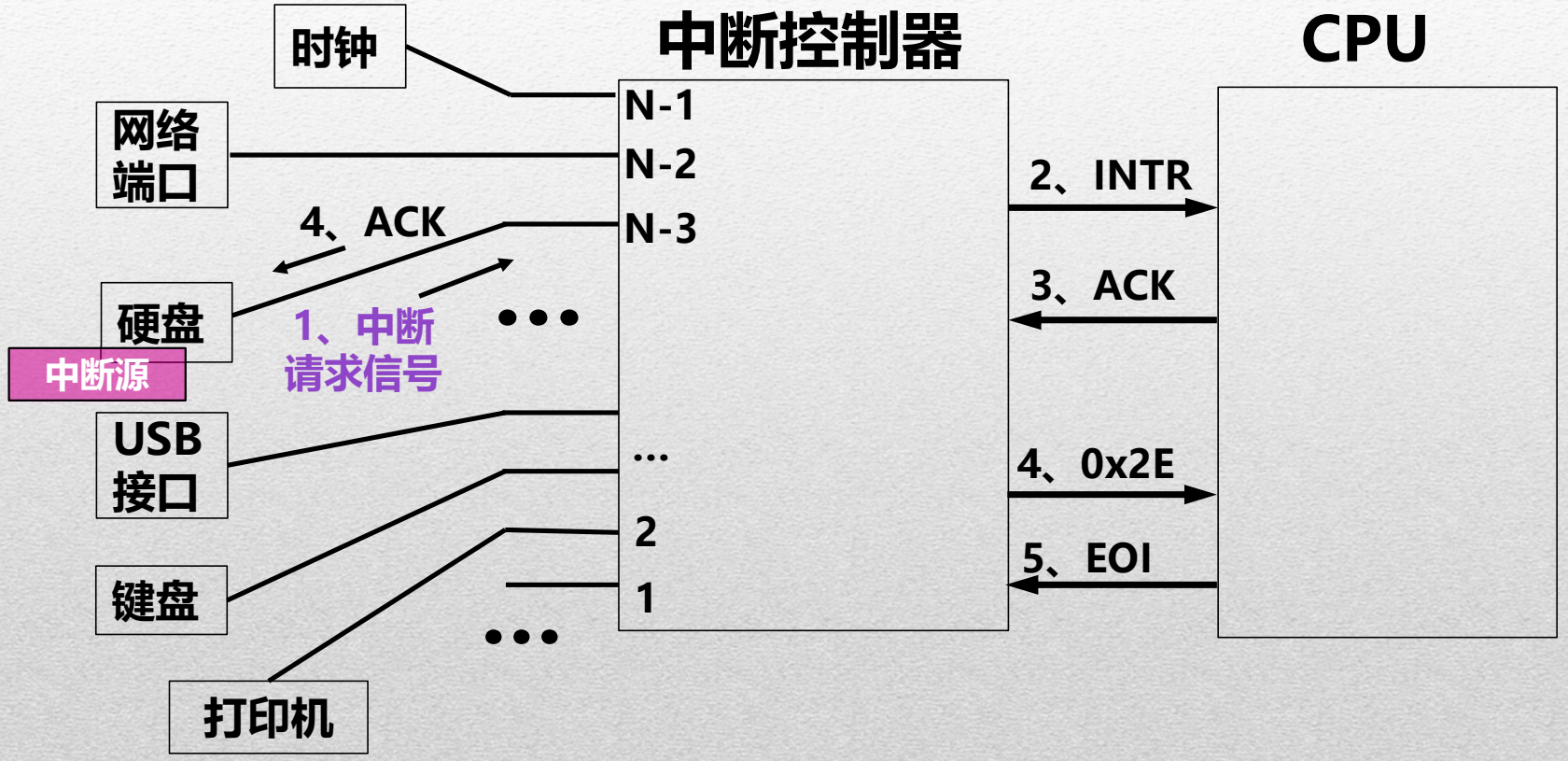
CPU不可能时刻查询外设的工作状态。。。



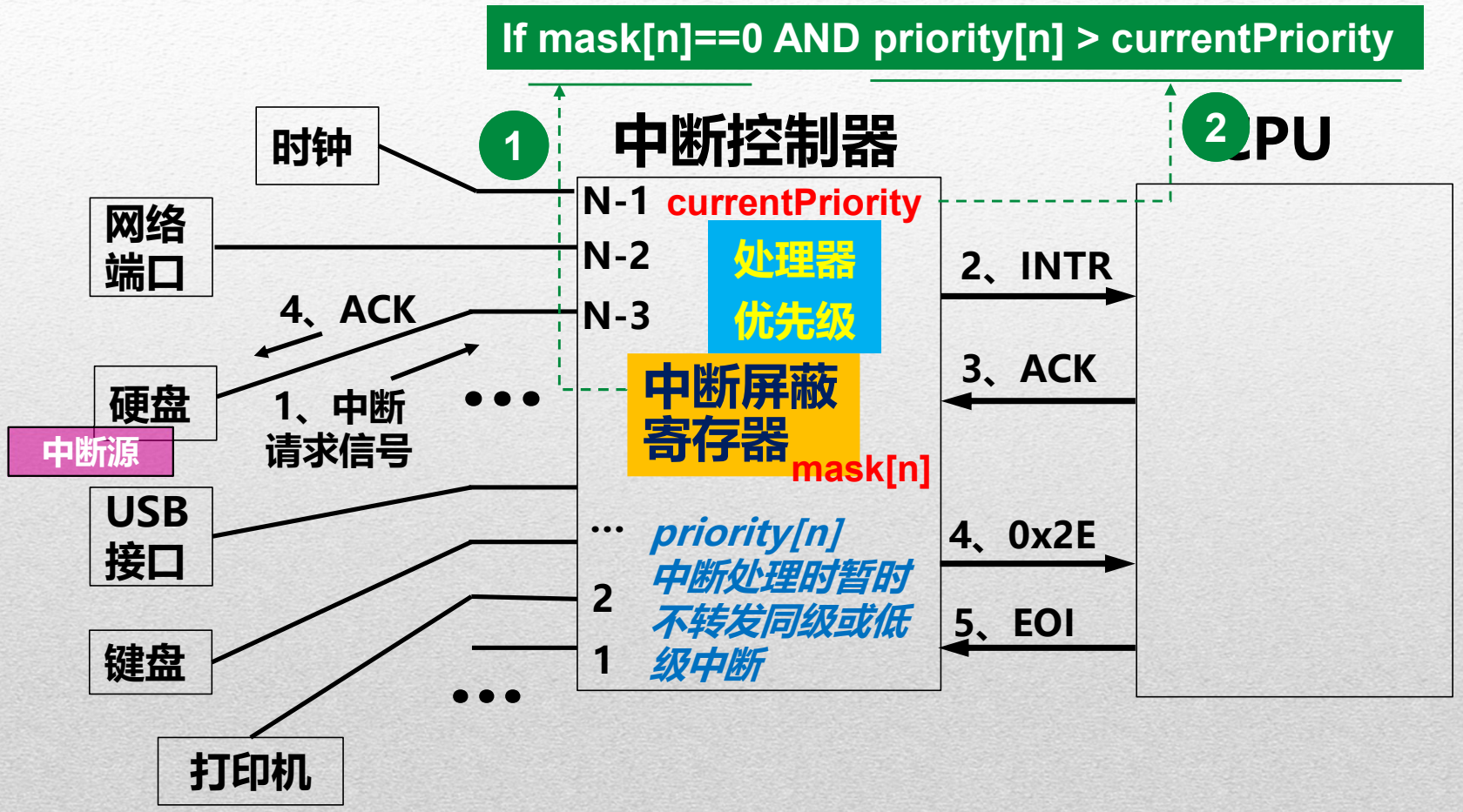
Q1>. CPU怎么知道外设的操作结束了?

中断的硬件机构

1. I/O结束后，设备向中断控制器发出中断请求。中断控制器决定是否向CPU转发该请求。



中断的硬件机构


中断仲裁
If mask[n]==0 AND priority[n] > currentPriority
时钟
1
中断控制器
2
CPU
网络端口
N-1
currentPriority
N-2
处理器优先级
2、INTR
4、ACK
N-3
硬盘
3、ACK
1、中断请求信号
中断源
中断屏蔽寄存器
mask[n]
USB接口
…
priority[n]
中断处理时暂时不转发同级或低级中断
4、0x2E
2
键盘
1
5、EOI
打印机

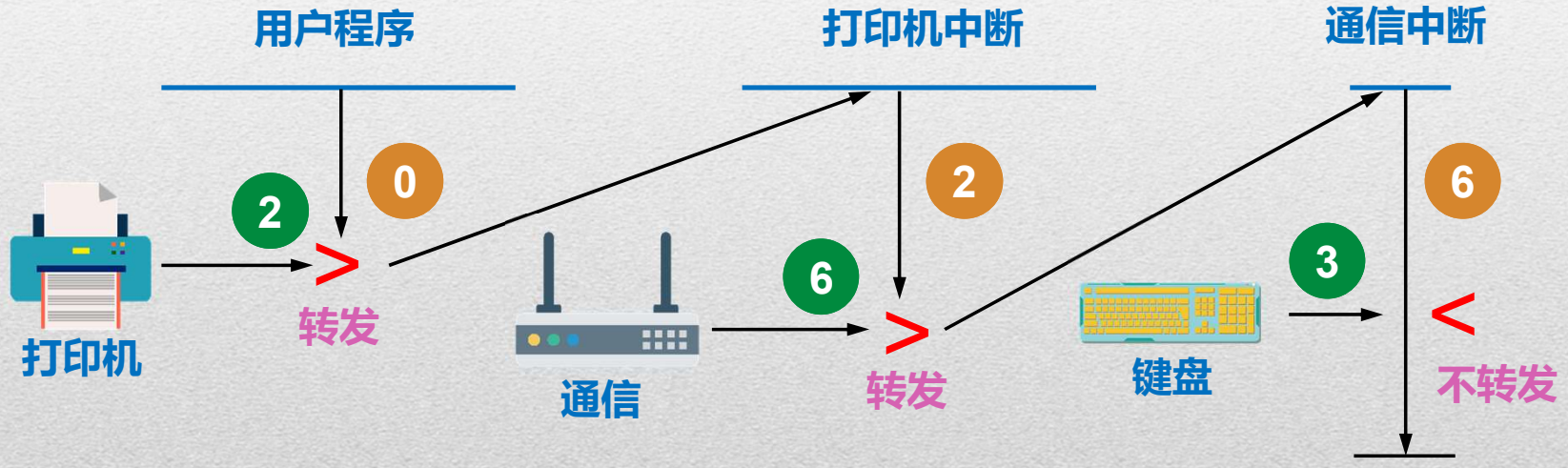
中断优先级

对不同的中断源，按重要性、紧迫程度分不同等级，并用一个正整数表示

处理机优先级

反映CPU正在执行的中断处理的优先级

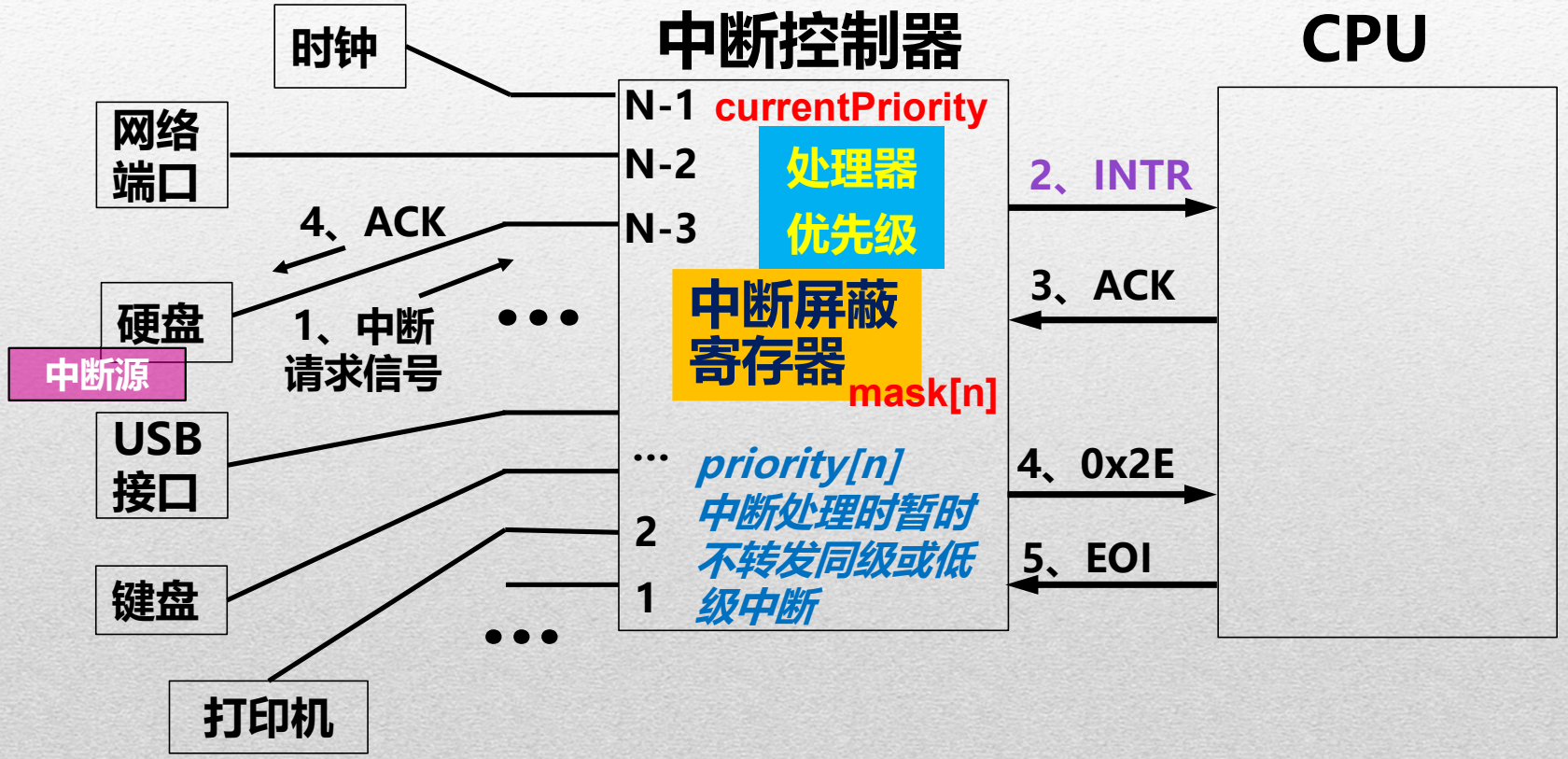
未执行中断服务子程序时：0



中断控制器只向CPU转发更高级的中断请求

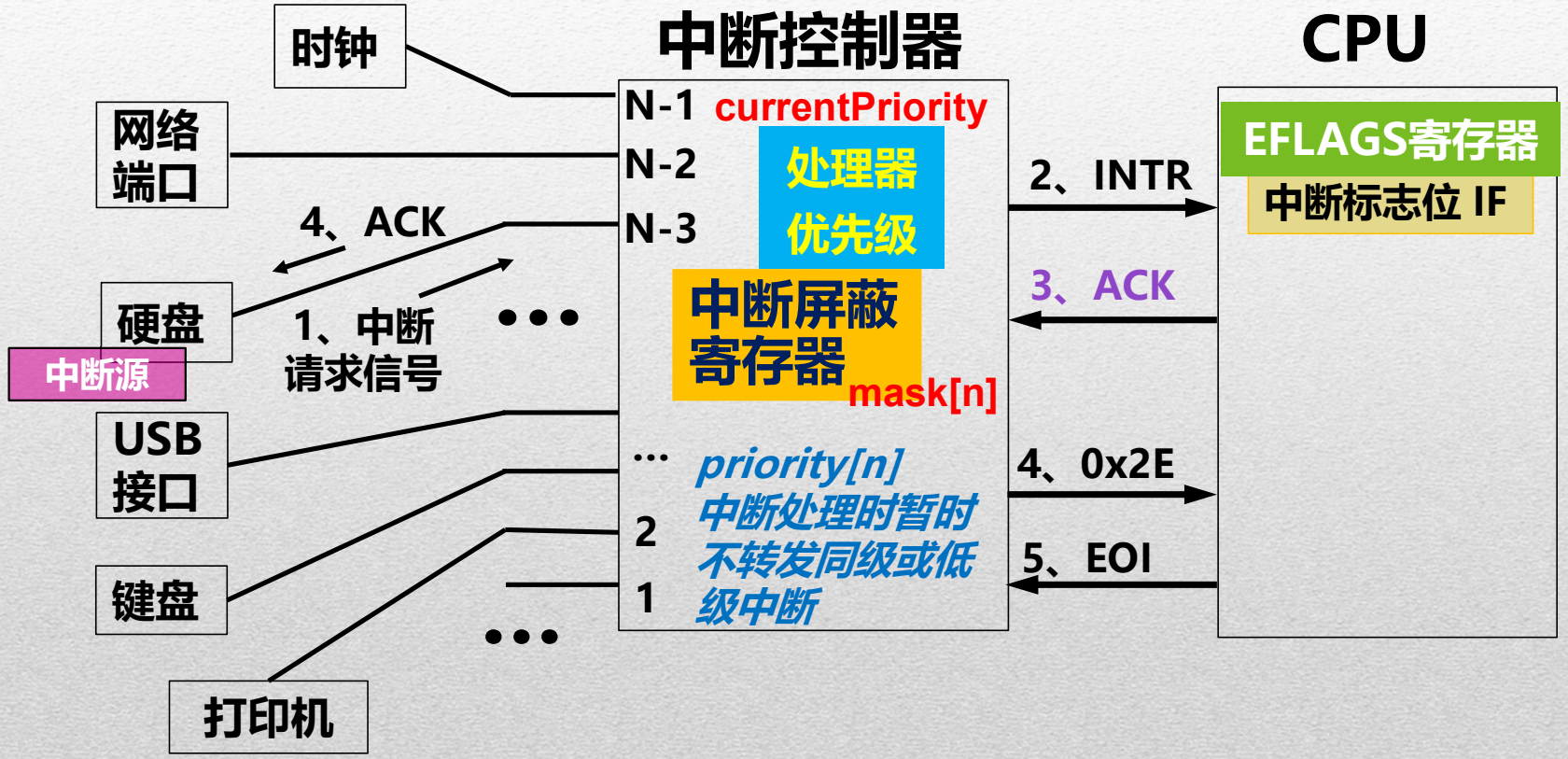
中断的硬件机构

2. 仲裁通过，中断控制器置INTR连线为高电平通知CPU系统发生了中断。



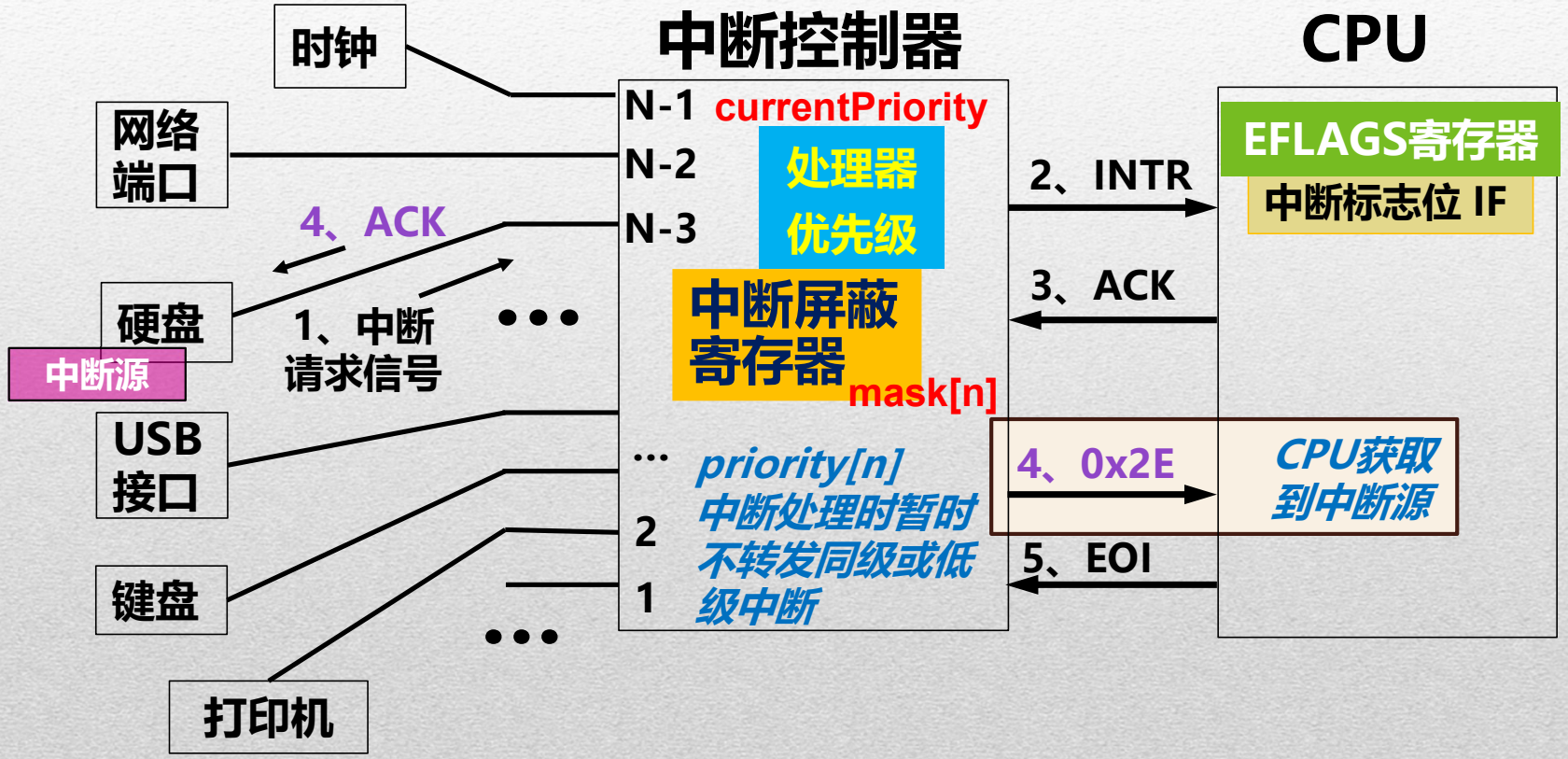
中断的硬件机构

3. 处理器在每条指令执行完毕后，立即检查INTR管脚。如果有中断请求，则立即回送ACK信号。



中断的硬件机构

4. 得到ACK信号的中断控制器送中断号给CPU。清除INTR请求信号，向外设发送ACK信号表示中断请求已被处理。

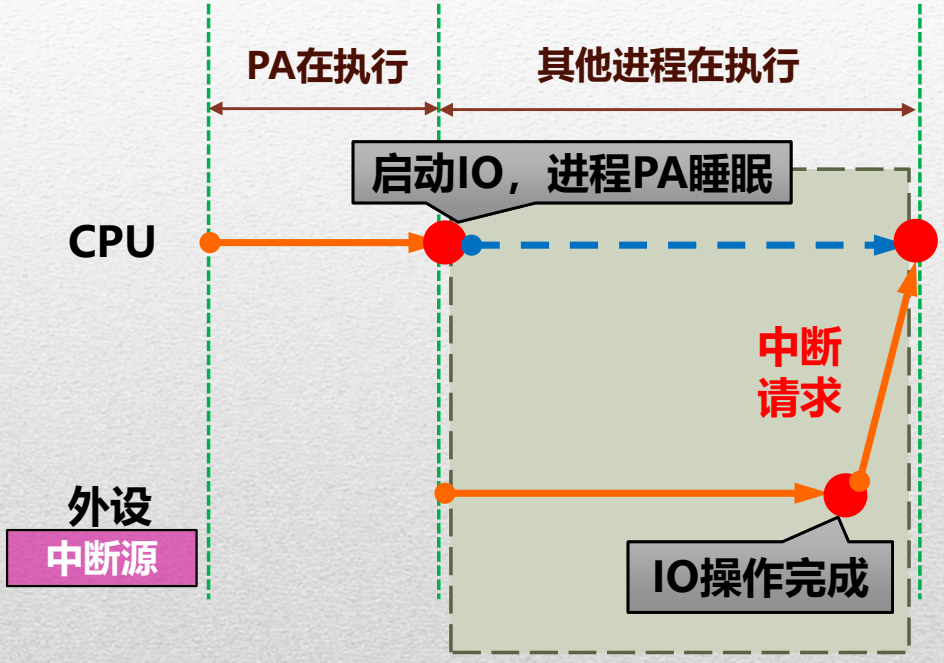


什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。



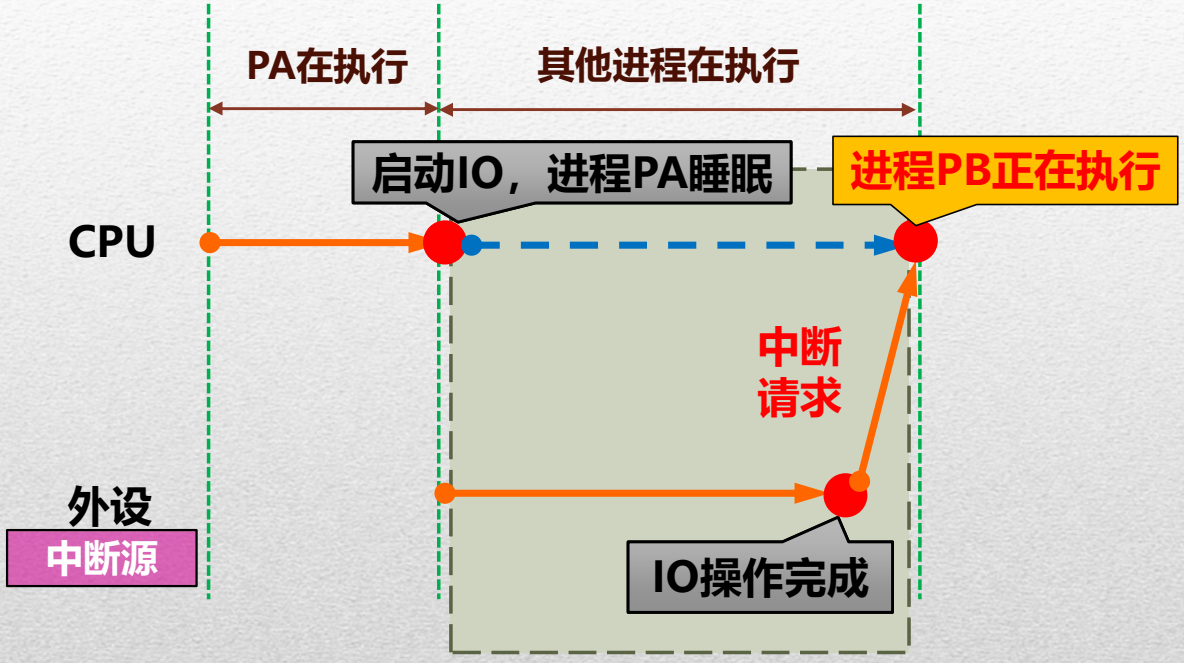
1. 保证了CPU和设备之间的并行操作

什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。



1. 保证了CPU和设备之间的并行操作

Q2>.CPU接收到中断请求后会发生什么?

设想一下如果你的工作被打断，你怎么办？

先

再

断点

CPU

通用寄存器

变量，中间计算结果，栈指针……

EAX：用来存放函数的返回值　　堆栈指针

EBX、ECX、EDX、ESI、EDI、ESP、EBP

专用寄存器

EIP：程序计数器　EFLAGS：处理机状态字

CS、SS、DS、ES、FS、GS、SS：段寄存器

……

CPU工作现场

你觉得进程PB现在该怎么办？

先记录自己当前的工作 → 转去执行中断处理 → 继续执行之前的工作

断点　　断点

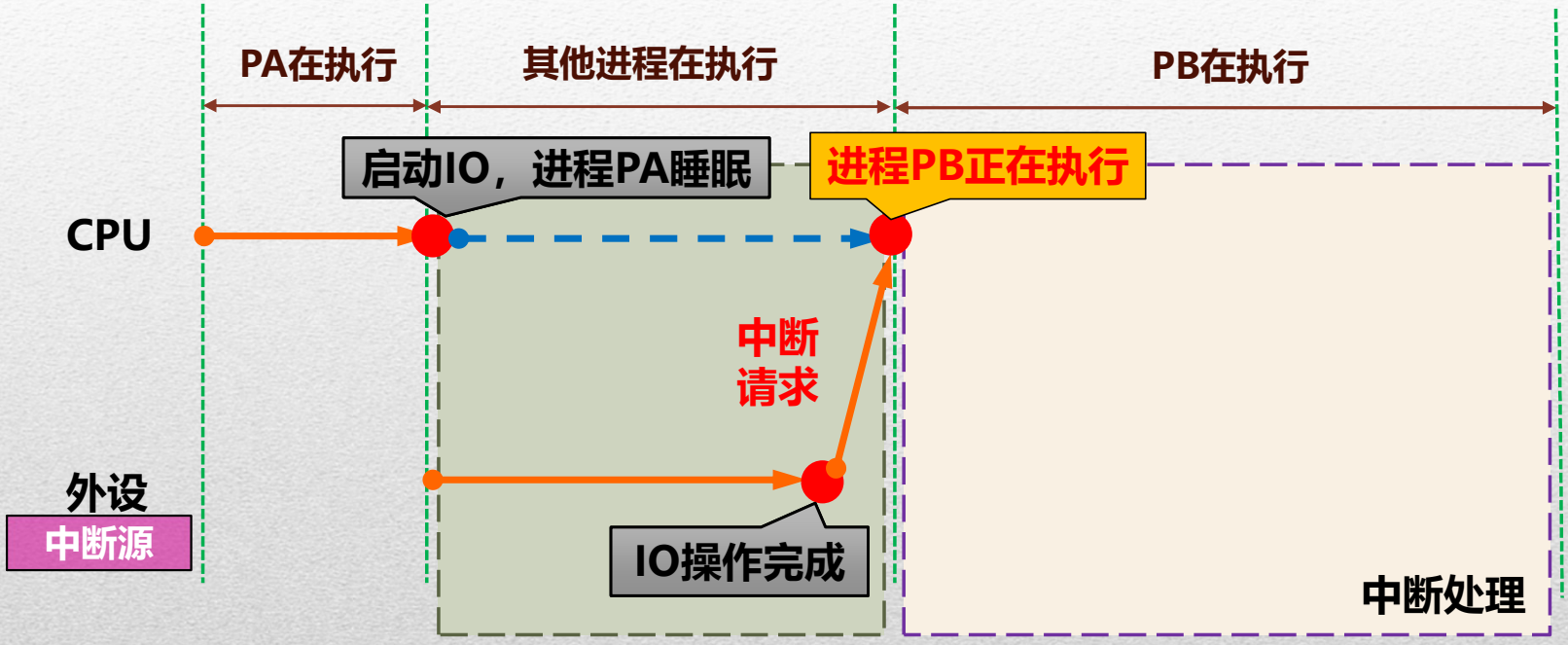
现场保护　　现场恢复

什么是中断?

一种设备控制方式

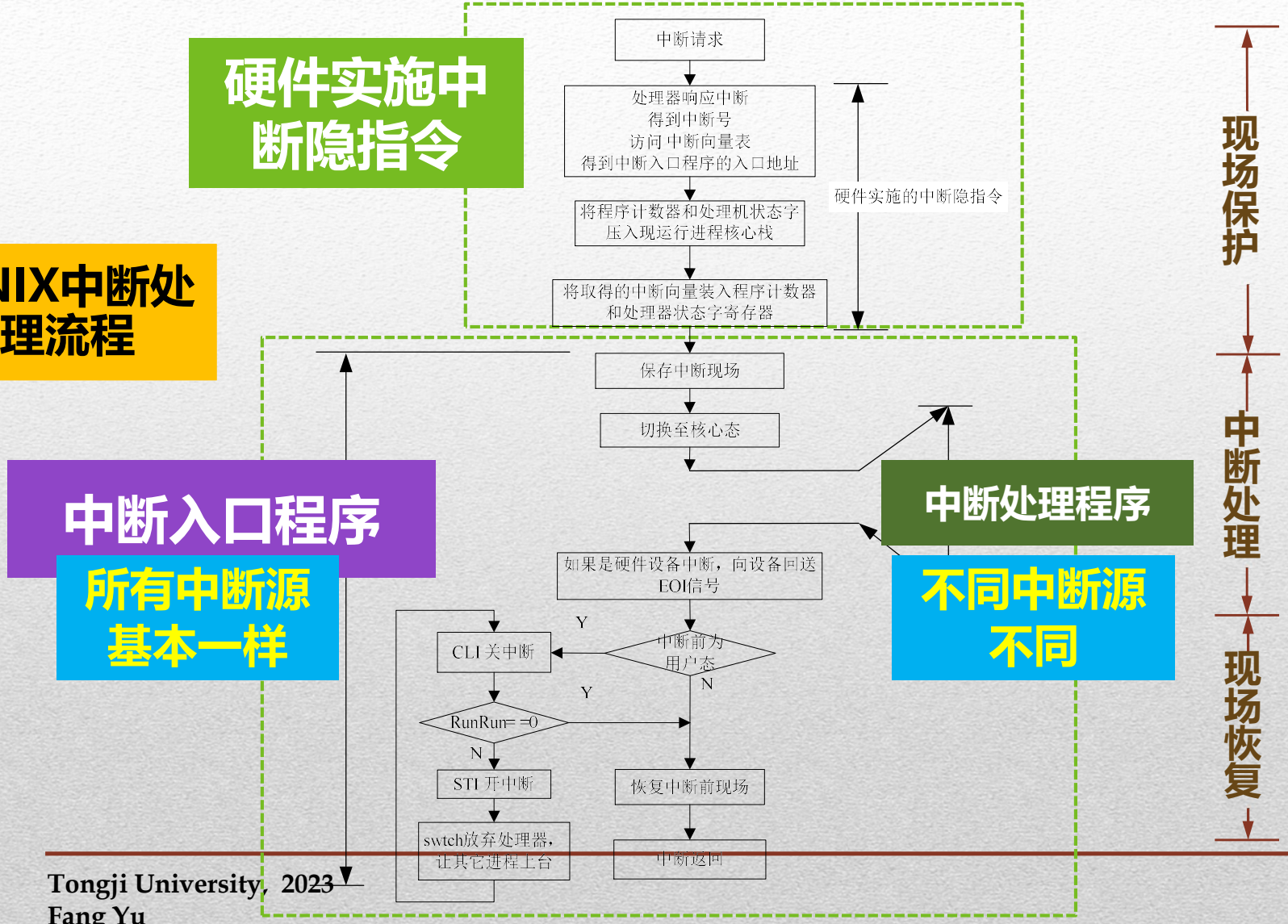
一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。



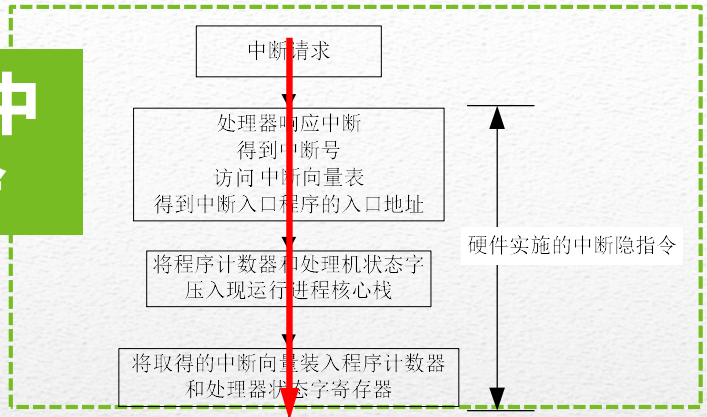
1. 保证了CPU和设备之间的并行操作

UNIX中断处理流程


硬件实施中断隐指令
中断入口程序
所有中断源基本一样
中断处理程序
不同中断源不同
中断请求
处理器响应中断
得到中断号
访问 中断向量表
得到中断入口程序的入口地址
将程序计数器和处理机状态字压入现运行进程核心栈
将取得的中断向量装入程序计数器和处理器状态字寄存器
硬件实施的中断隐指令
保存中断现场
切换至核心态
如果是硬件设备中断，向设备回送EOI信号
中断前为用户态
Y
N
CLI 关中断
Y
RunRun==0
N
STI 开中断
swtch放弃处理器，让其它进程上台
恢复中断前现场
中断返回
现场保护
中断处理
现场恢复

UNIX中断处理流程

硬件实施中断隐指令



执行中断隐指令

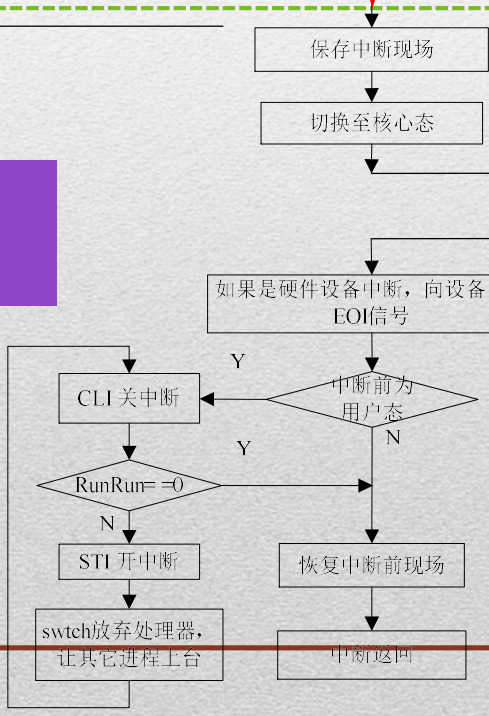
跳转到中断入口程序

中断入口程序

所有中断源基本一样

中断处理程序

不同中断源不同



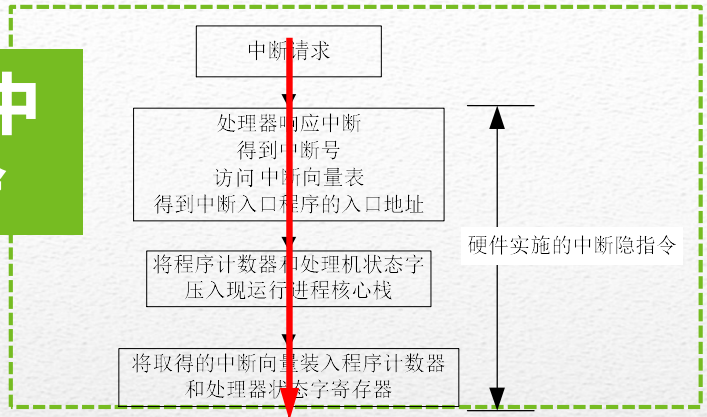
现场保护

中断处理

现场恢复

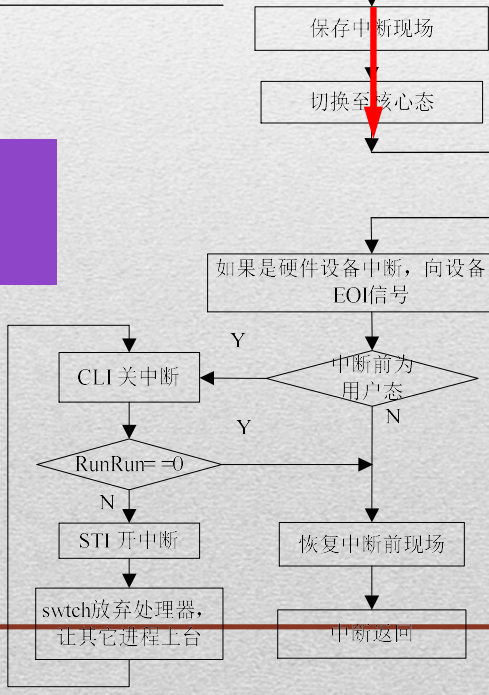
UNIX中断处理流程

硬件实施中断隐指令



中断入口程序

所有中断源基本一样



中断处理程序

不同中断源不同

现场保护

中断处理

现场恢复

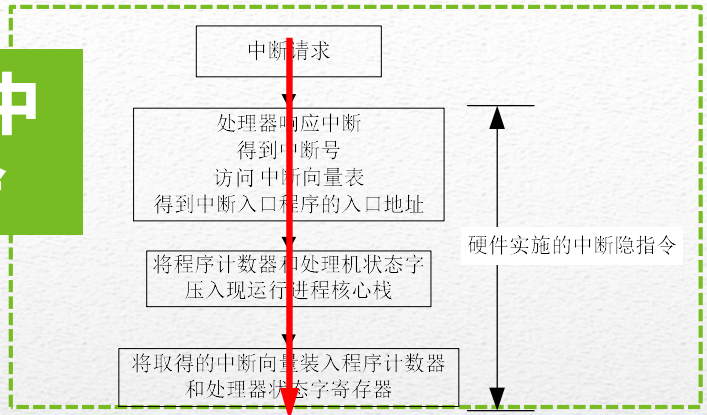
执行中断隐指令

跳转到中断入口程序

执行中断入口程序的前半部分

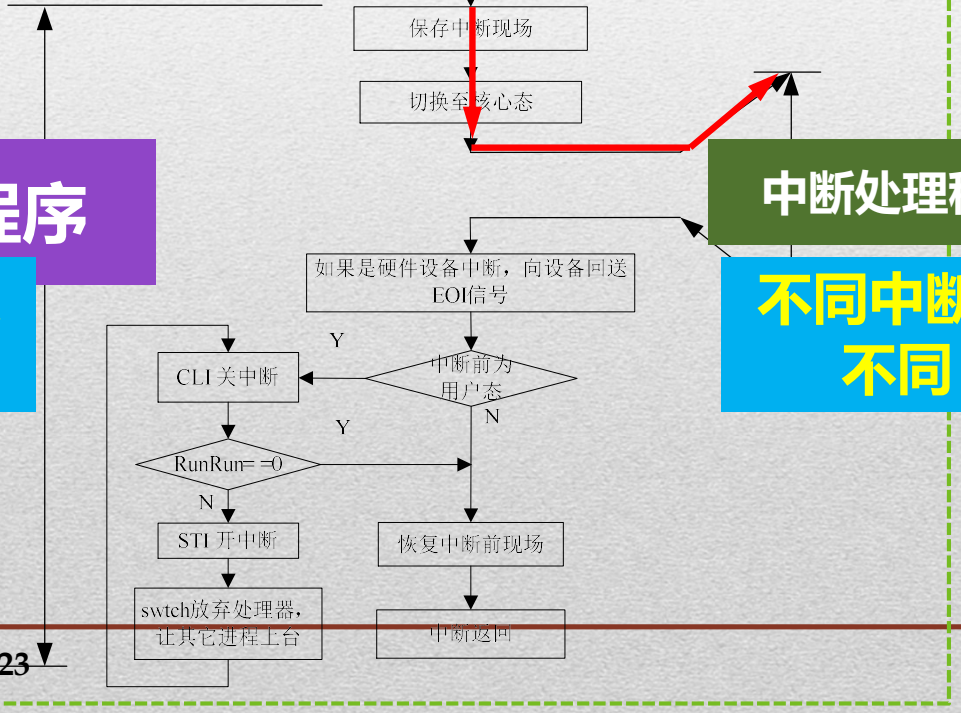
UNIX中断处理流程

硬件实施中断隐指令



中断入口程序

所有中断源基本一样



中断处理程序

不同中断源不同

现场保护

中断处理

现场恢复

执行中断隐指令

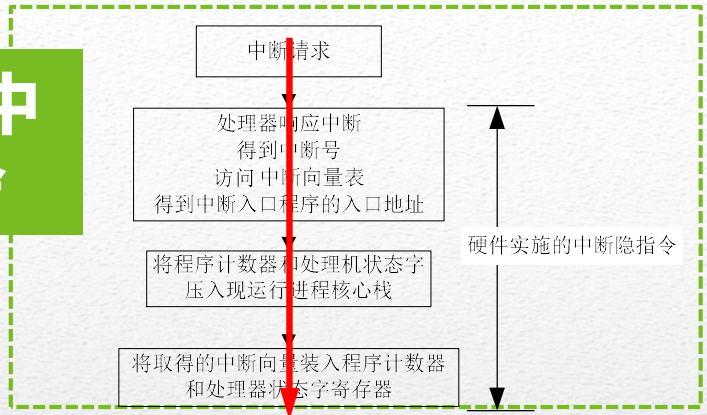
跳转到中断入口程序

执行中断入口程序的前半部分

跳转到中断处理程序

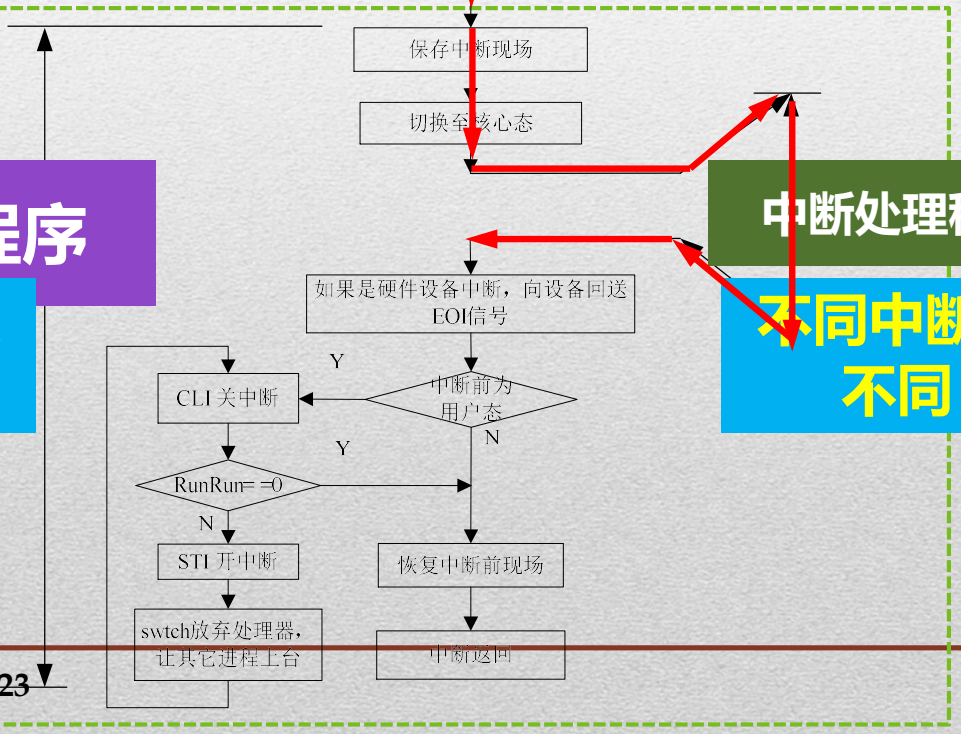
UNIX中断处理流程

硬件实施中断隐指令



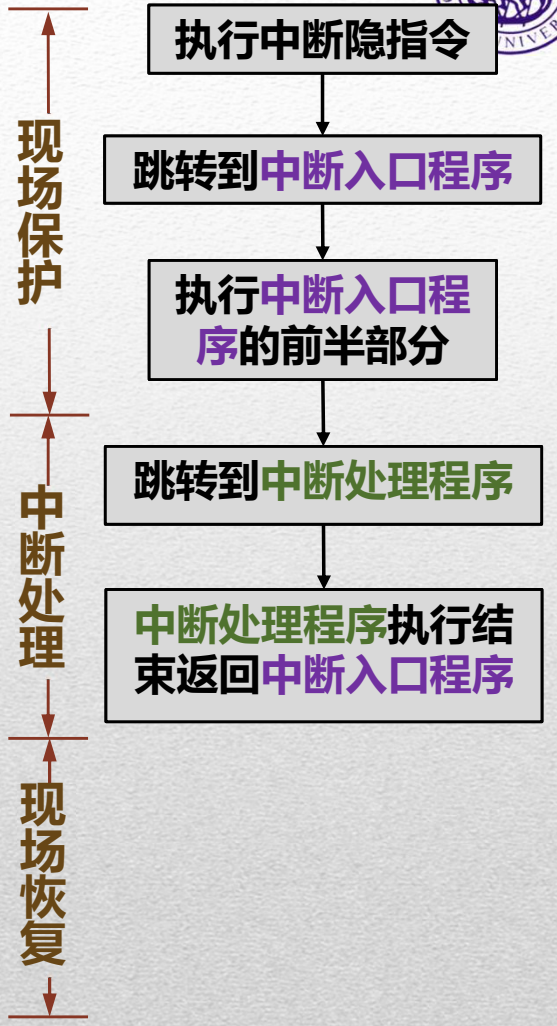
中断入口程序

所有中断源基本一样



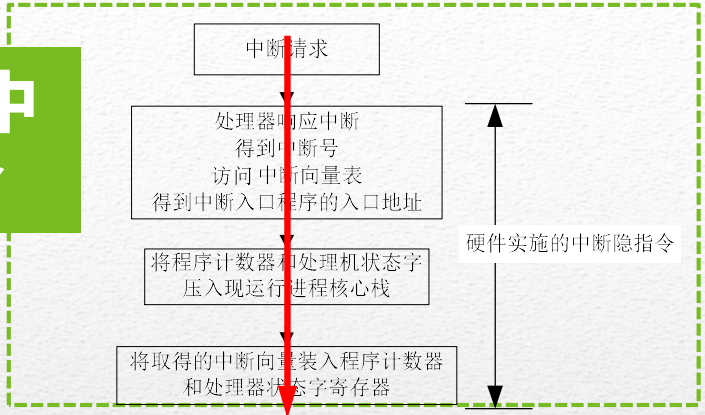
中断处理程序

不同中断源不同



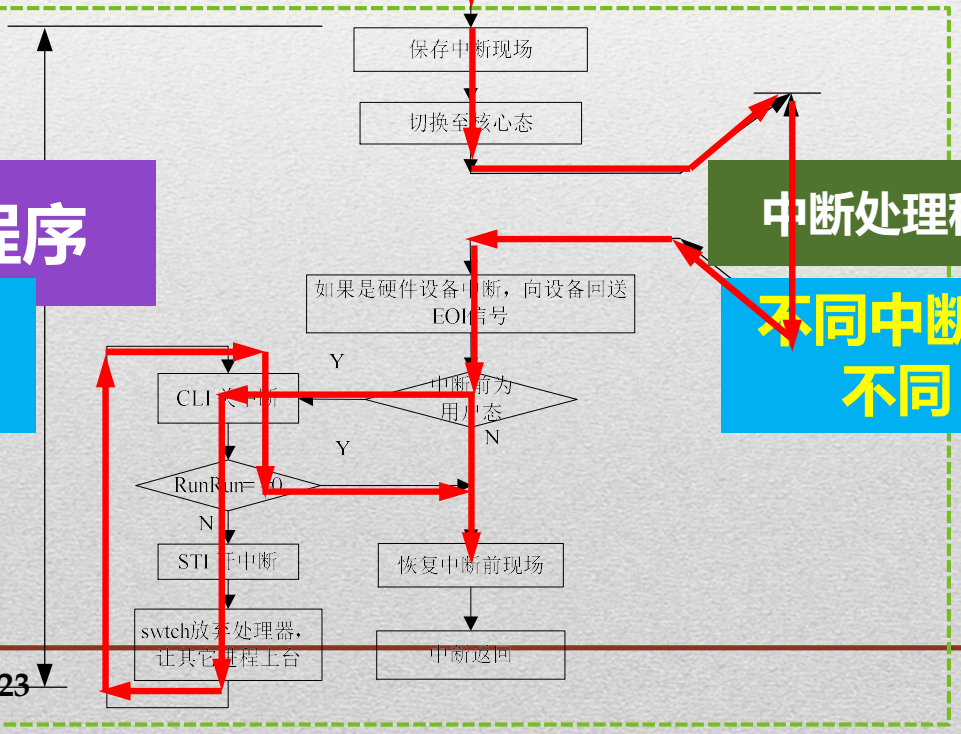
UNIX中断处理流程

硬件实施中断隐指令



中断入口程序

所有中断源基本一样



中断处理程序

不同中断源不同

现场保护

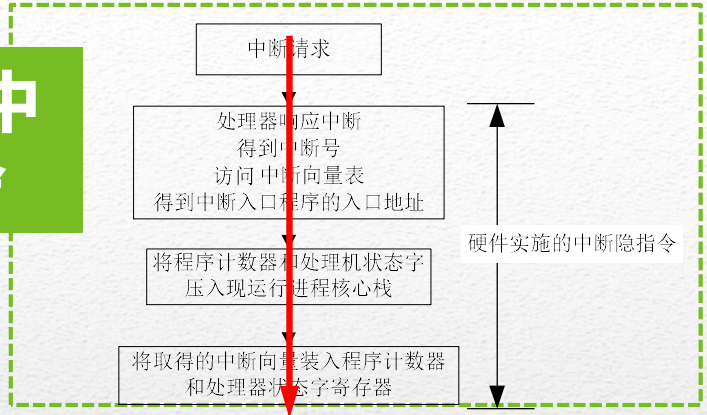
中断处理

现场恢复

1. 执行中断隐指令
2. 跳转到中断入口程序
3. 执行中断入口程序的前半部分
4. 跳转到中断处理程序
5. 中断处理程序执行结束返回中断入口程序
6. 执行中断入口程序的后半部分

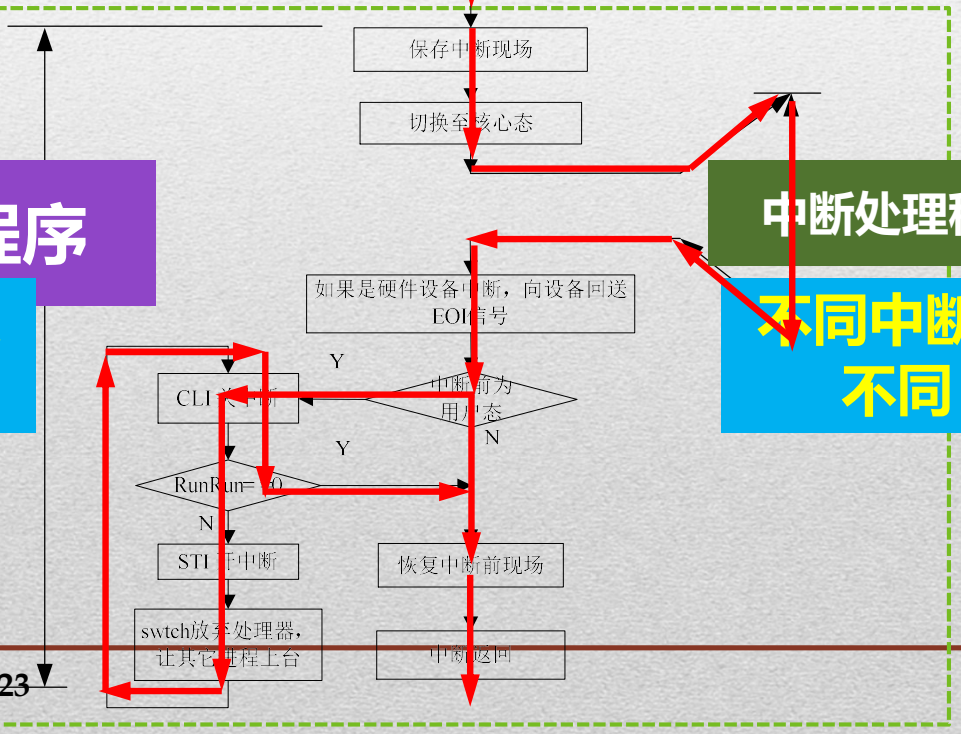
UNIX中断处理流程

硬件实施中断隐指令



中断入口程序

所有中断源基本一样



中断处理程序

不同中断源不同

现场保护

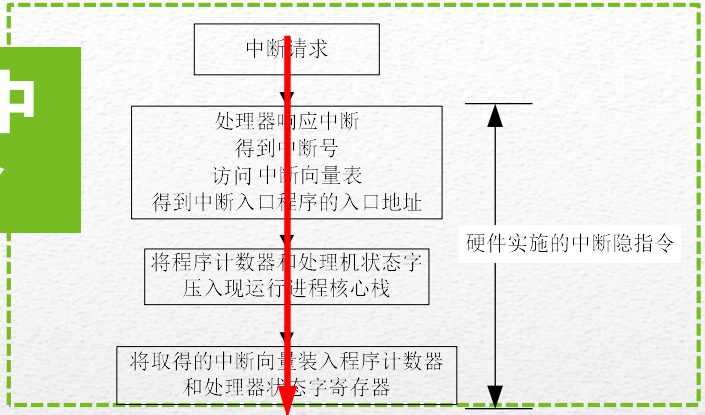
中断处理

现场恢复

1. 执行中断隐指令
2. 跳转到中断入口程序
3. 执行中断入口程序的前半部分
4. 跳转到中断处理程序
5. 中断处理程序执行结束返回中断入口程序
6. 执行中断入口程序的后半部分
7. 中断返回

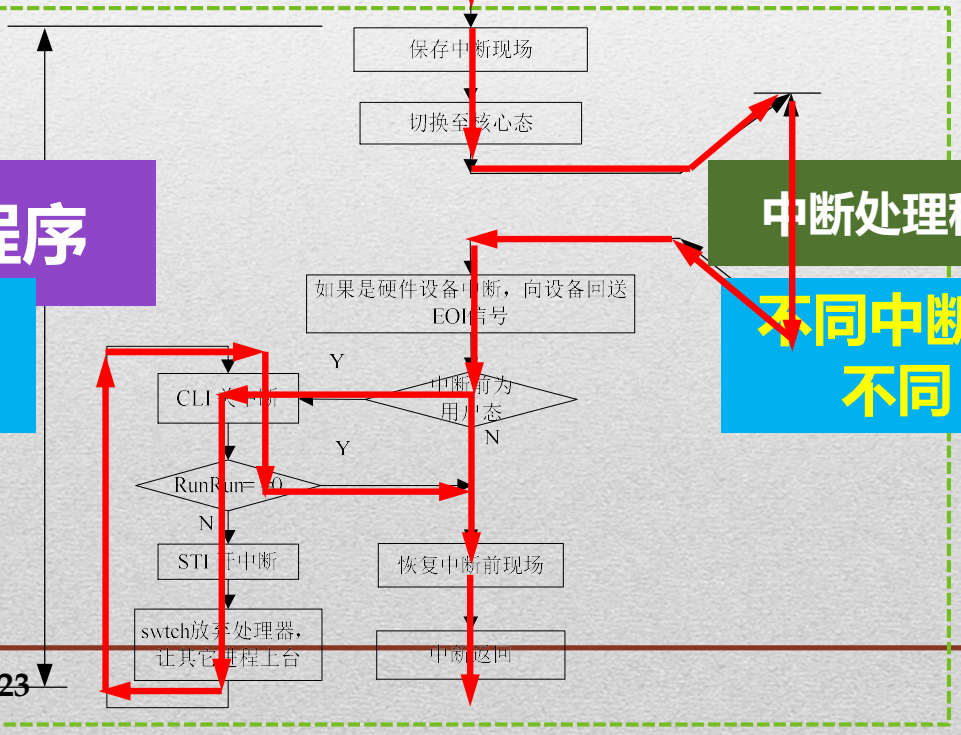
UNIX中断处理流程

硬件实施中断隐指令



中断入口程序

所有中断源基本一样



中断处理程序

不同中断源不同

现场保护

中断处理

现场恢复

1. 执行中断隐指令
2. 跳转到中断入口程序
3. 执行中断入口程序的前半部分
4. 跳转到中断处理程序
5. 中断处理程序执行结束返回中断入口程序
6. 执行中断入口程序的后半部分
7. 中断返回

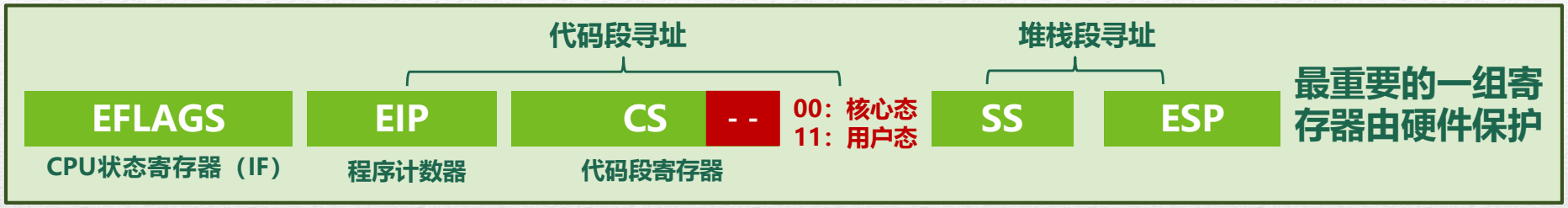
i386硬件实施的中断隐指令

1. 关中断

i386硬件实施的中断隐指令

1. 关中断

2. 实施硬件现场保护（中断前核心寄存器的值）

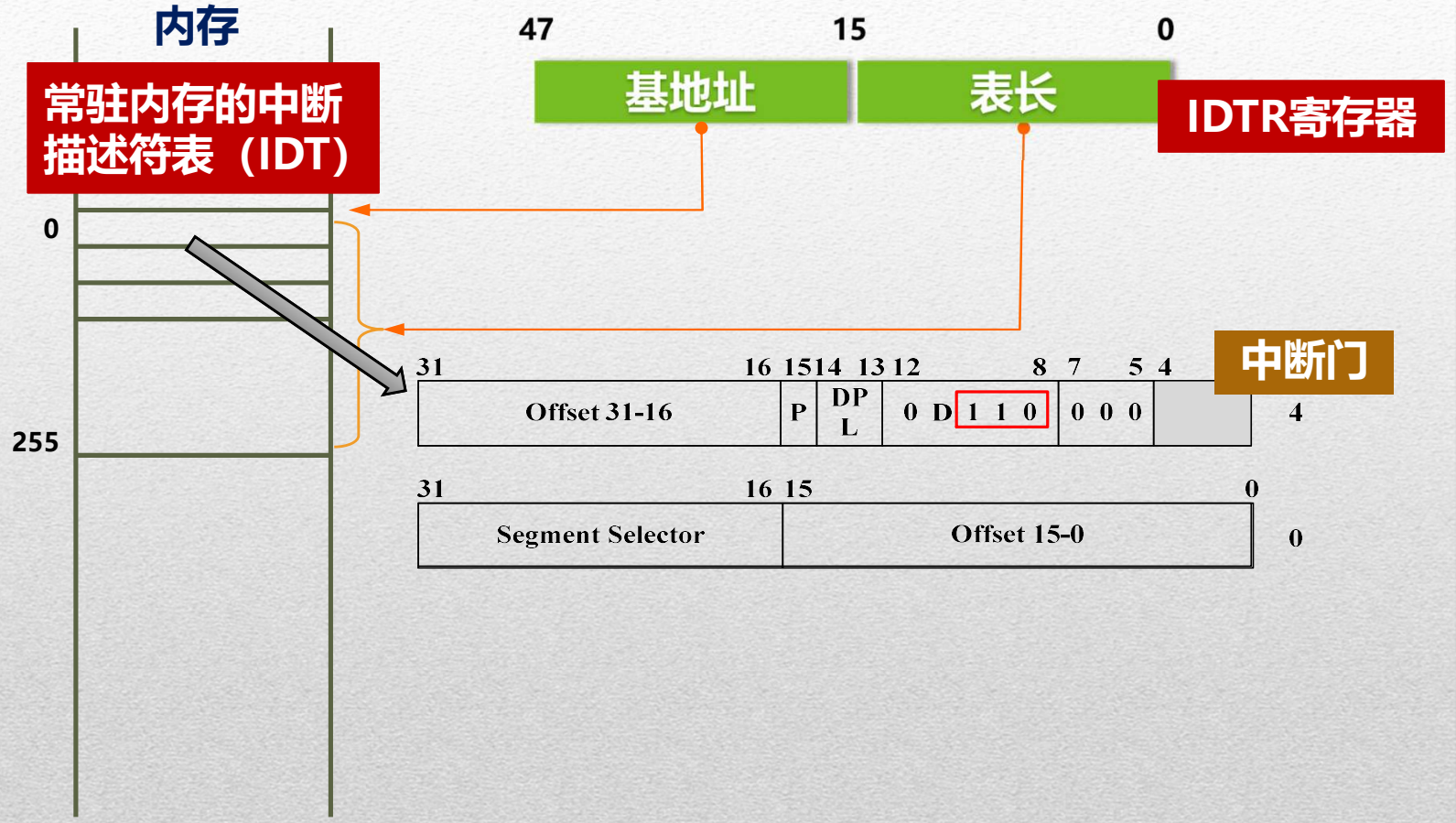


i386硬件实施的中断隐指令

3. 查询并装入中断向量（门）

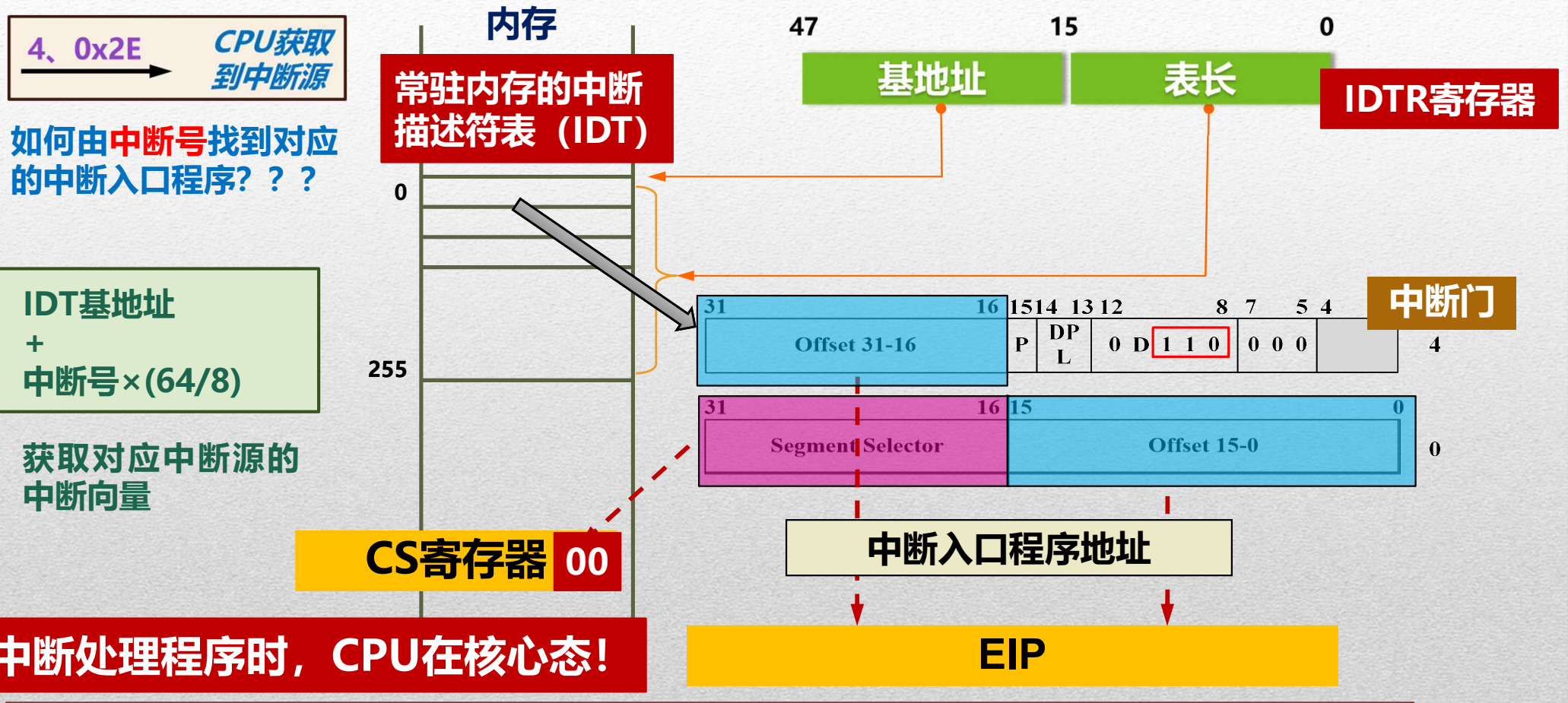
4、0x2E → CPU获取到中断源

如何由中断号找到对应的中断入口程序？？？



i386硬件实施的中断隐指令

3. 查询并装入中断向量（门）



UNIX中进程的两种执行状态

用户态
User Mode

执行用户程序，
提供用户功能

核心态
Kernel Mode

执行内核程序，
提供系统功能

可以在一定时机相互转换

内核不是与用户进程平行运行的孤立的进程集合。

中断

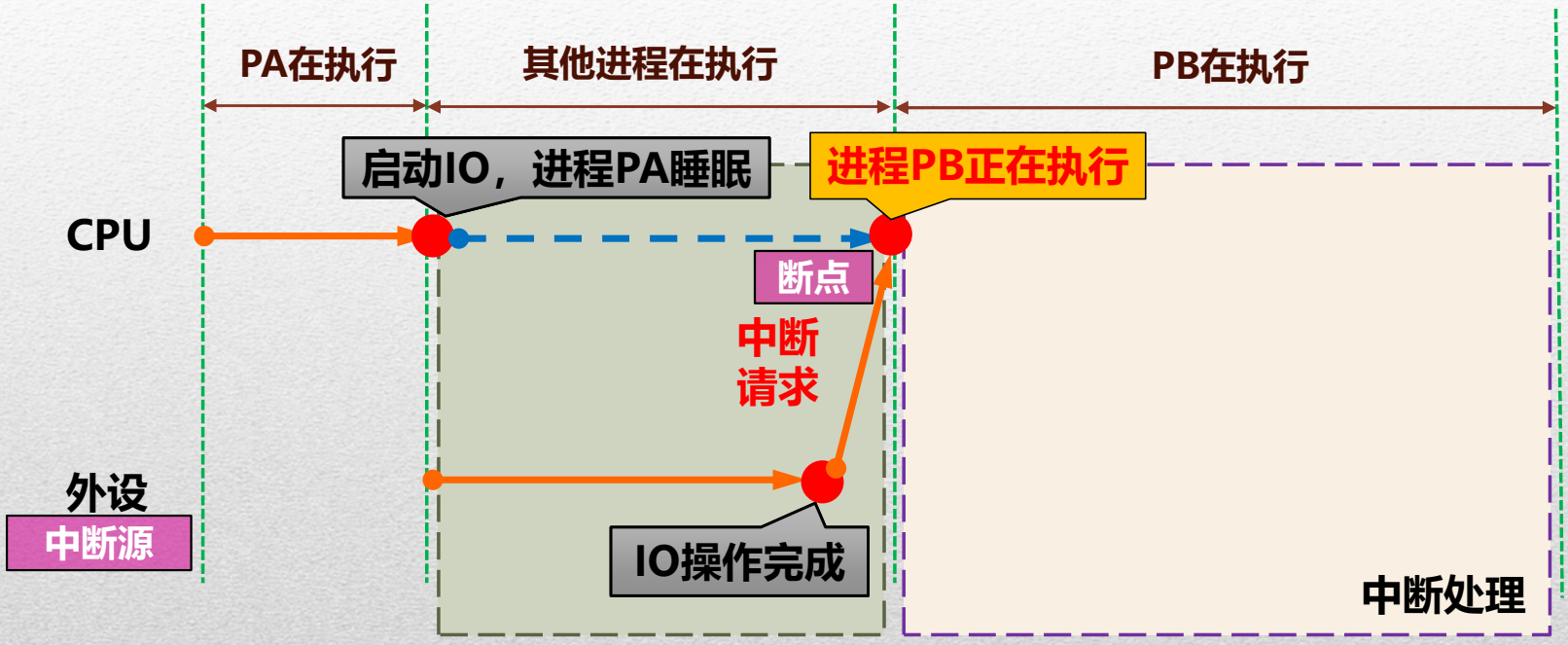
在核心态下执行内核代码的进程完成了内核功能！

什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。



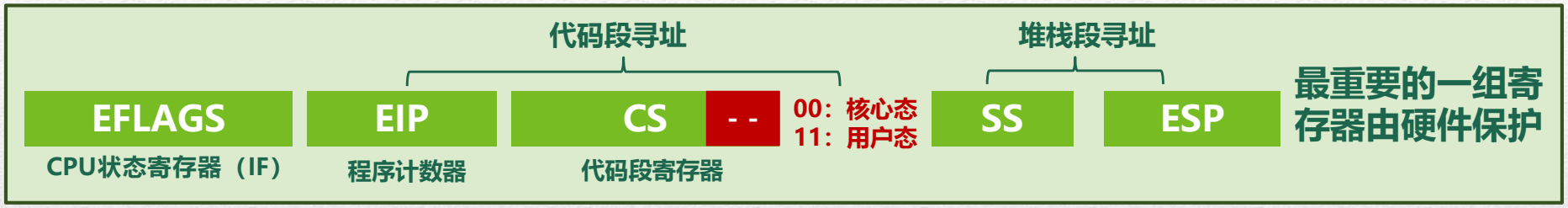
1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

i386硬件实施的中断隐指令

1. 关中断

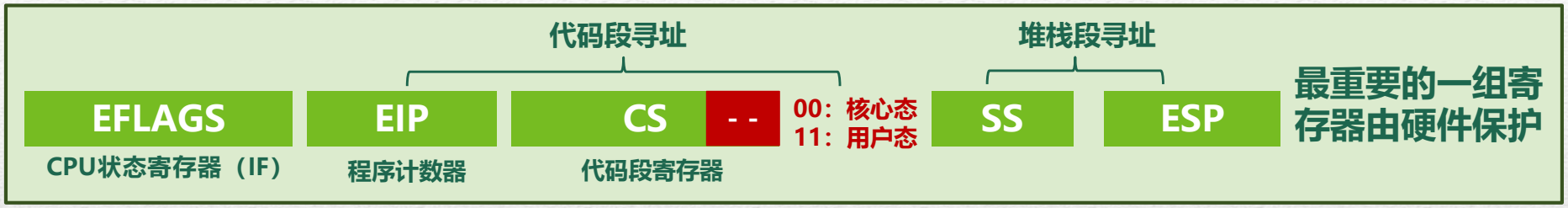
2. 实施硬件现场保护（中断前核心寄存器的值）

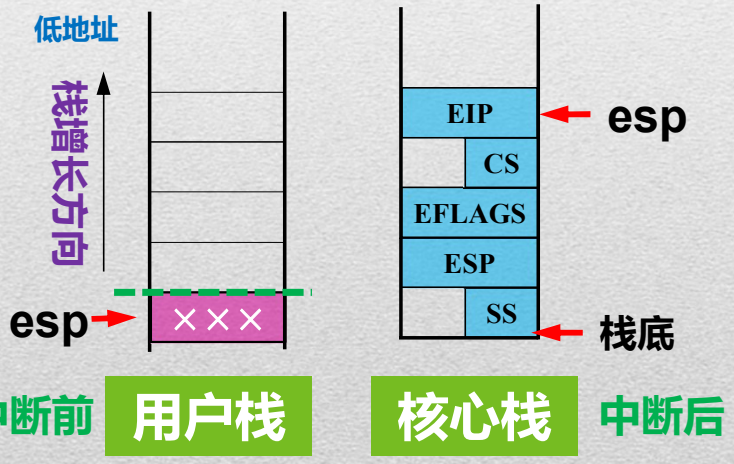


i386硬件实施的中断隐指令

1. 关中断

2. 实施硬件现场保护（中断前核心寄存器的值）



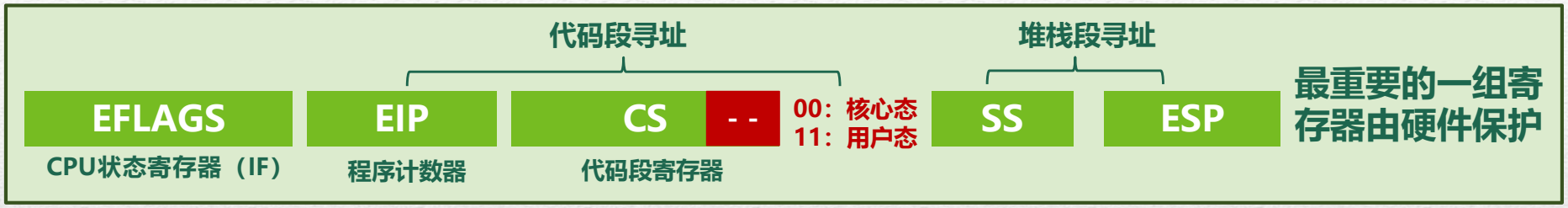


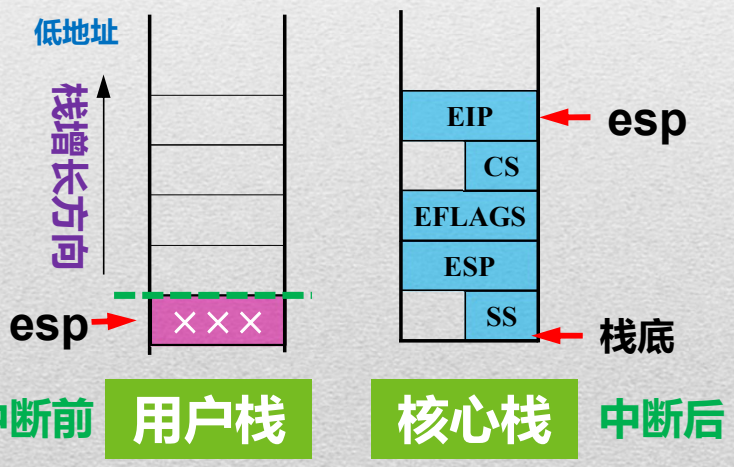
中断发生时现运行进程在用户态运行

i386硬件实施的中断隐指令

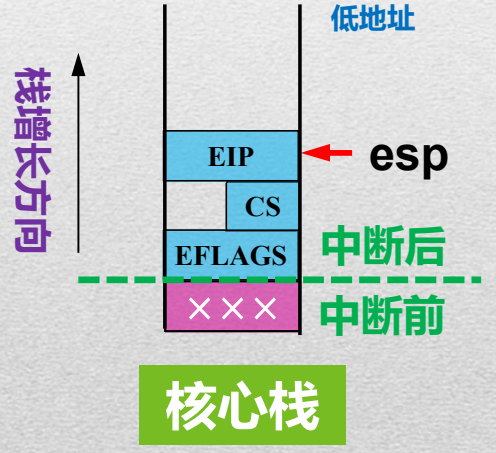
1. 关中断

2. 实施硬件现场保护（中断前核心寄存器的值）





中断发生时现运行进程在用户态运行

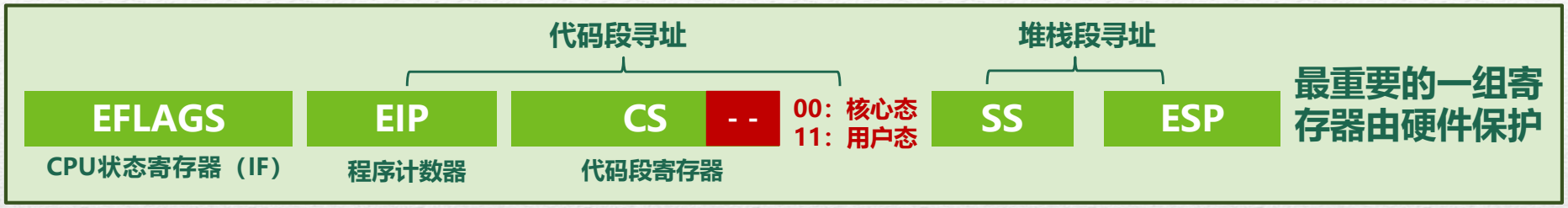


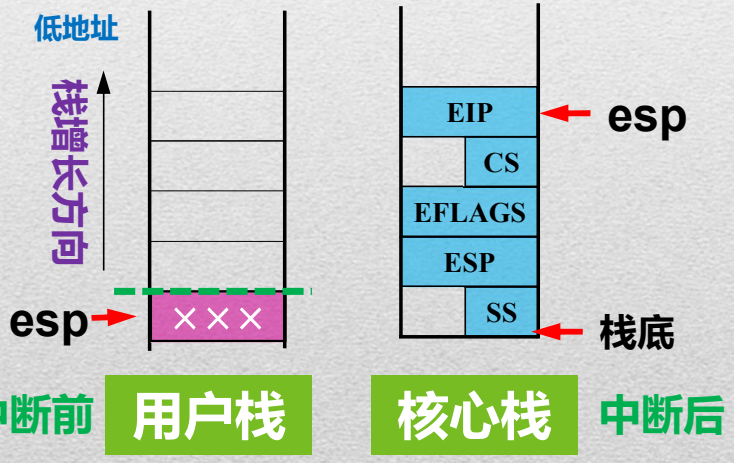
中断发生时现运行进程在核心态运行

i386硬件实施的中断隐指令

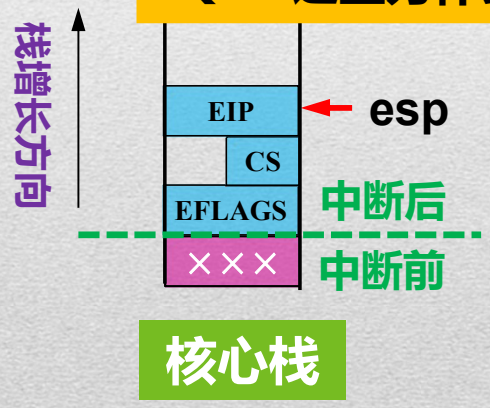
1. 关中断

2. 实施硬件现场保护（中断前核心寄存器的值）





中断发生时现运行进程在用户态运行



中断发生时现运行进程在核心态运行

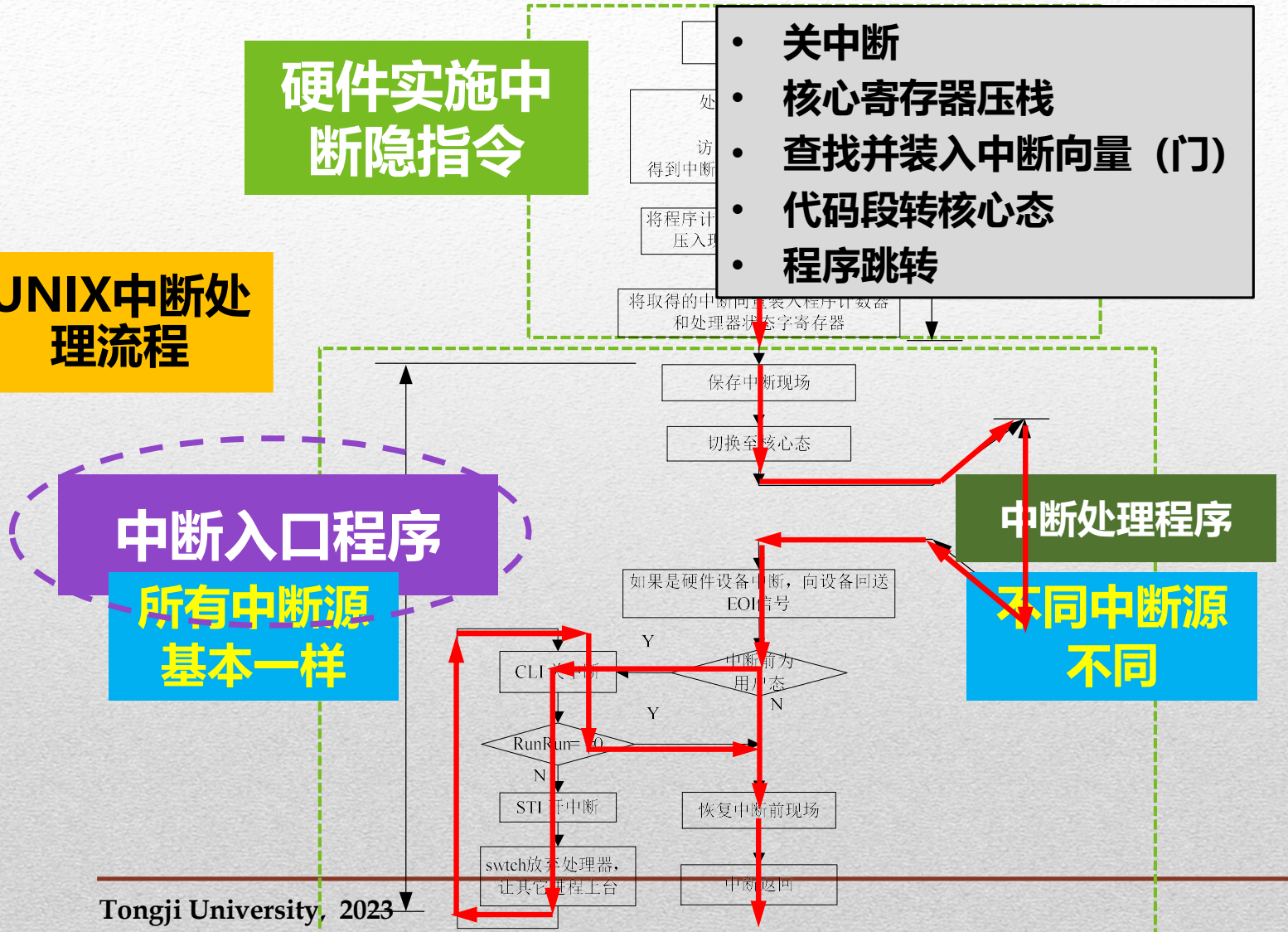
Q3>.这里为什么要关中断？什么时候开？

A3>. 不同设备的中断处理子程序中，可以根据情况选择不同的时机开中断

硬件实施中断隐指令

- 关中断
- 核心寄存器压栈
- 查找并装入中断向量（门）
- 代码段转核心态
- 程序跳转

UNIX中断处理流程



中断入口程序

所有中断源基本一样

中断处理程序

不同中断源不同

现场保护

中断处理

中断返回

执行中断隐指令

跳转到中断入口程序

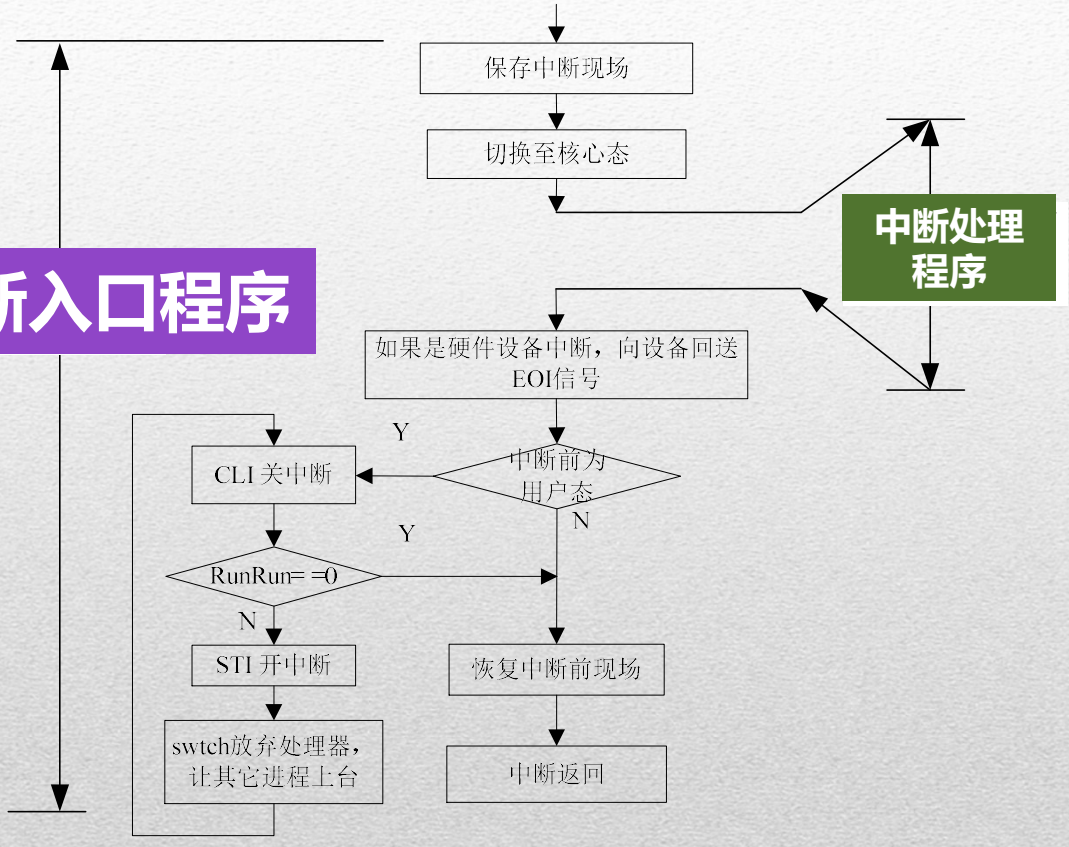
执行中断入口程序的前半部分

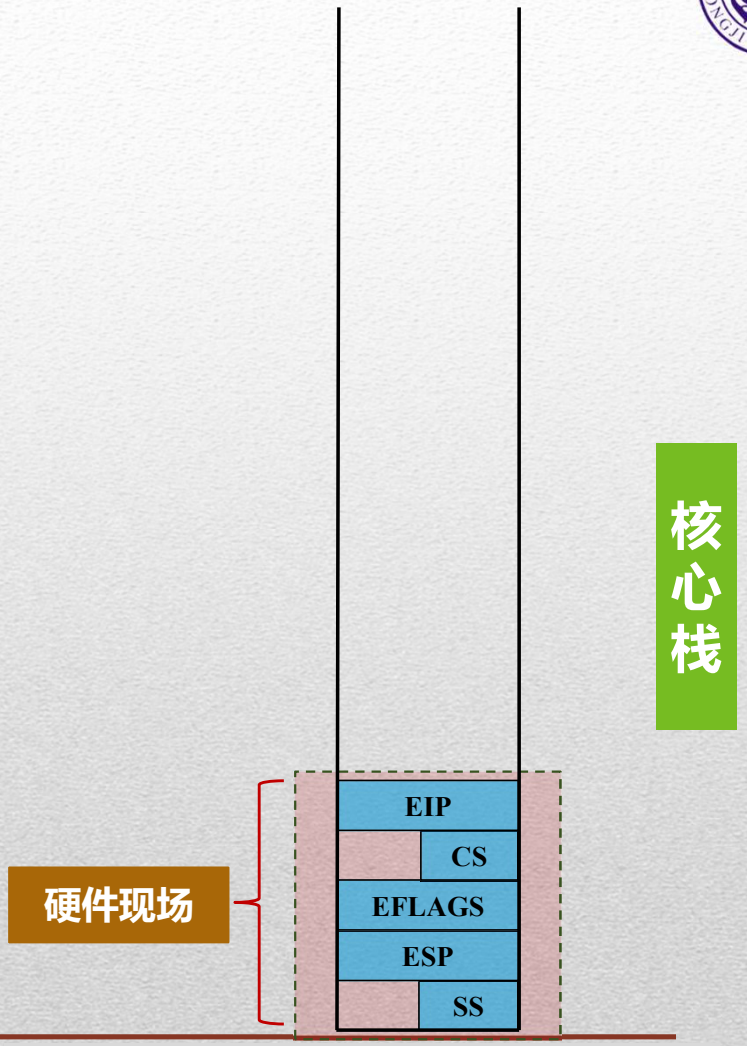
跳转到中断处理程序。执行结束后，返回中断入口程序

执行中断入口程序的后半部分

中断返回

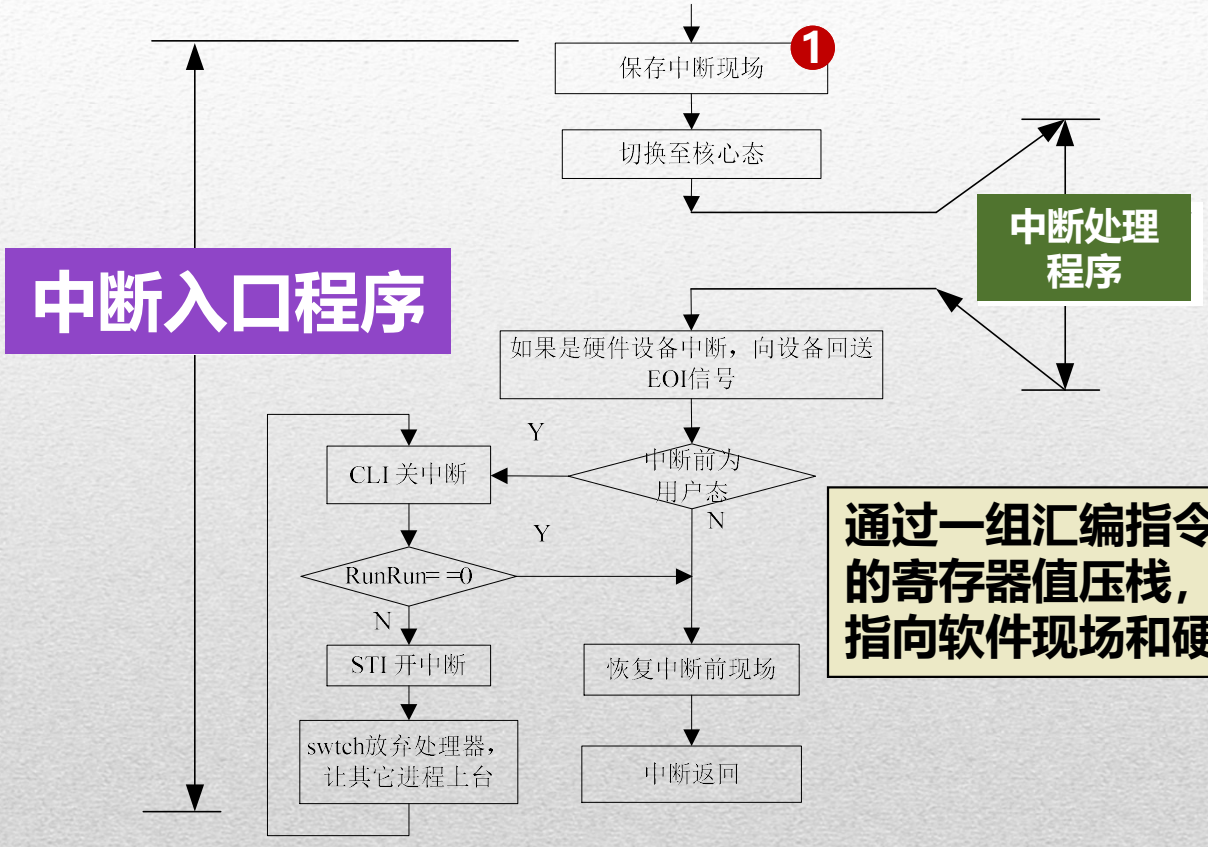
执行中断处理程序前：



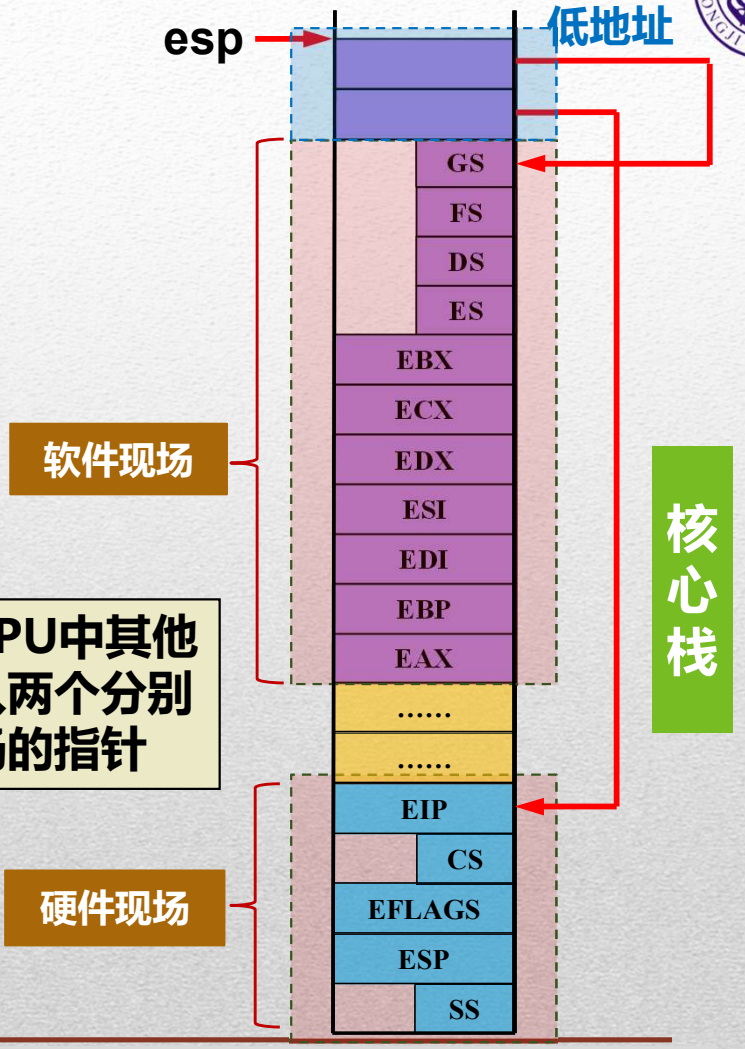


中断发生时现运行进程在用户态运行

（1）：继续完成现场保护的工作

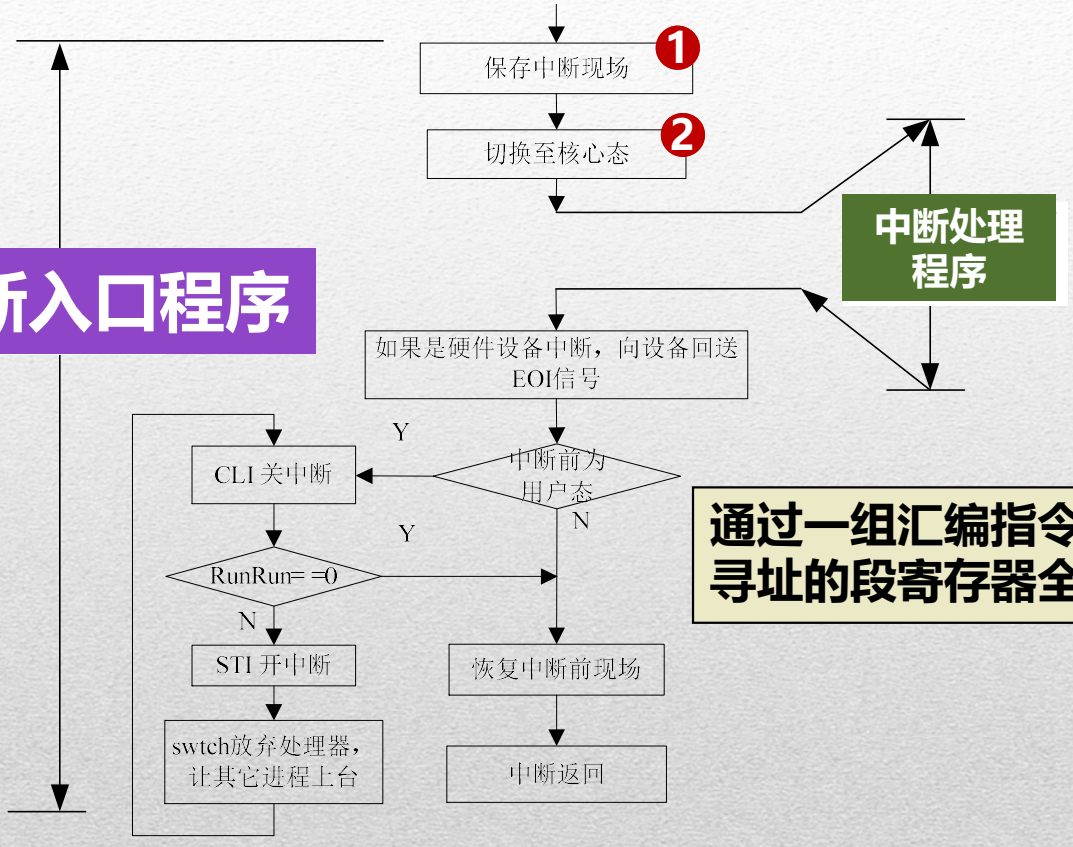


通过一组汇编指令，将CPU中其他的寄存器值压栈，并压入两个分别指向软件现场和硬件现场的指针

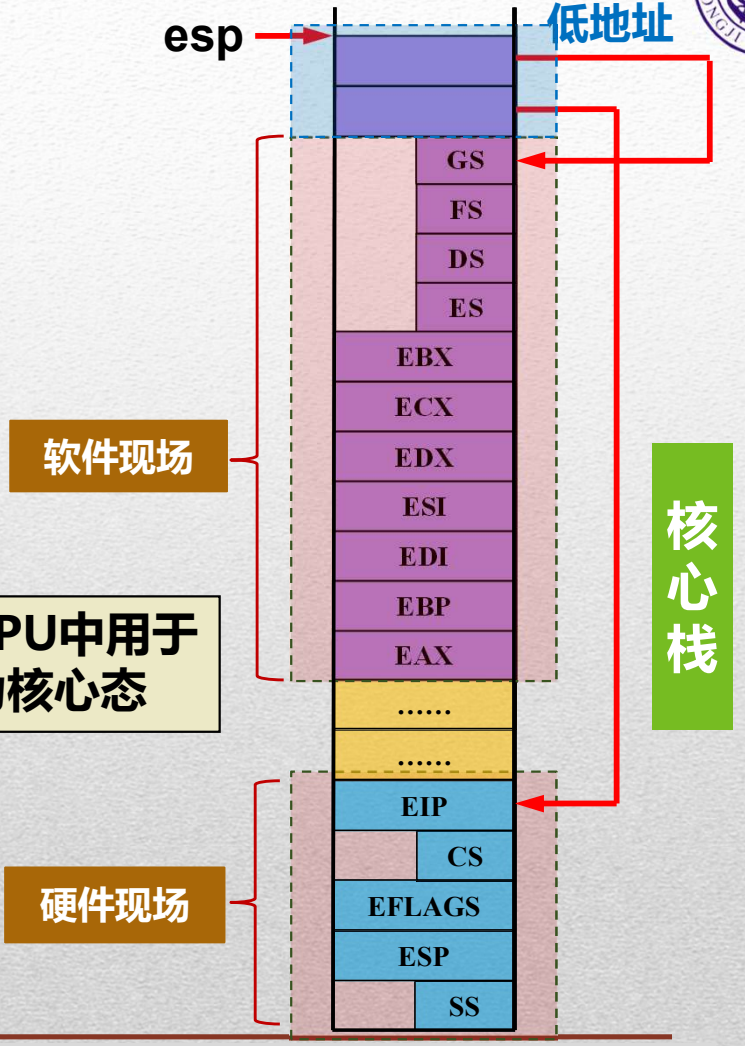


中断发生时现运行进程在用户态运行

（2）：内存管理转入核心态

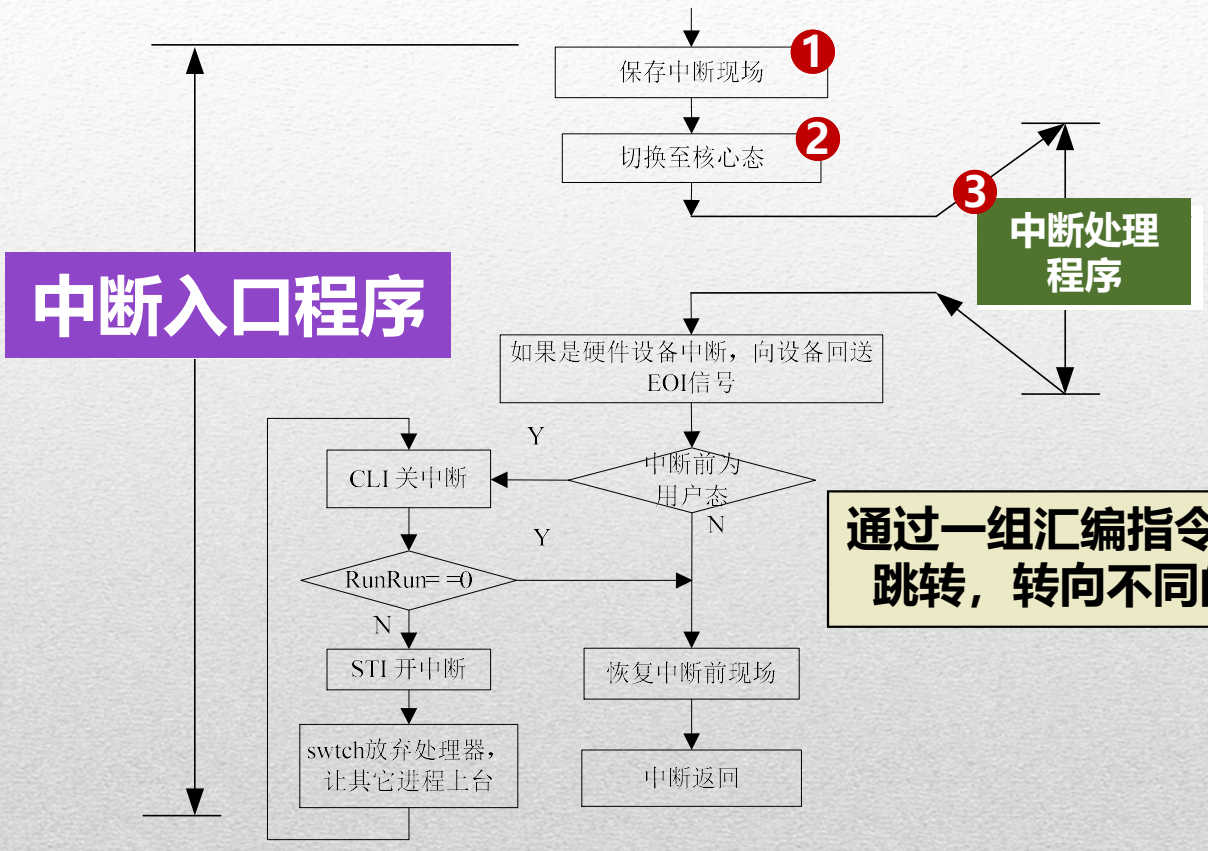


通过一组汇编指令，将CPU中用于寻址的段寄存器全部改为核心态

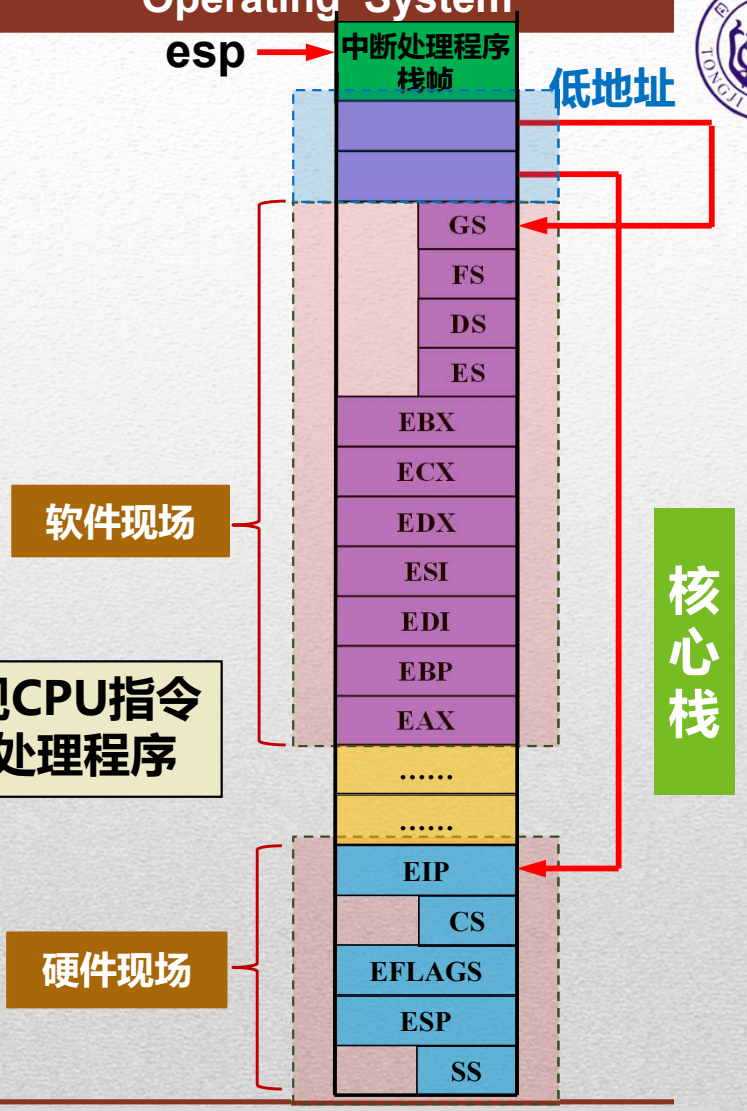


中断发生时现运行进程在用户态运行

（3）：调用中断处理子程序

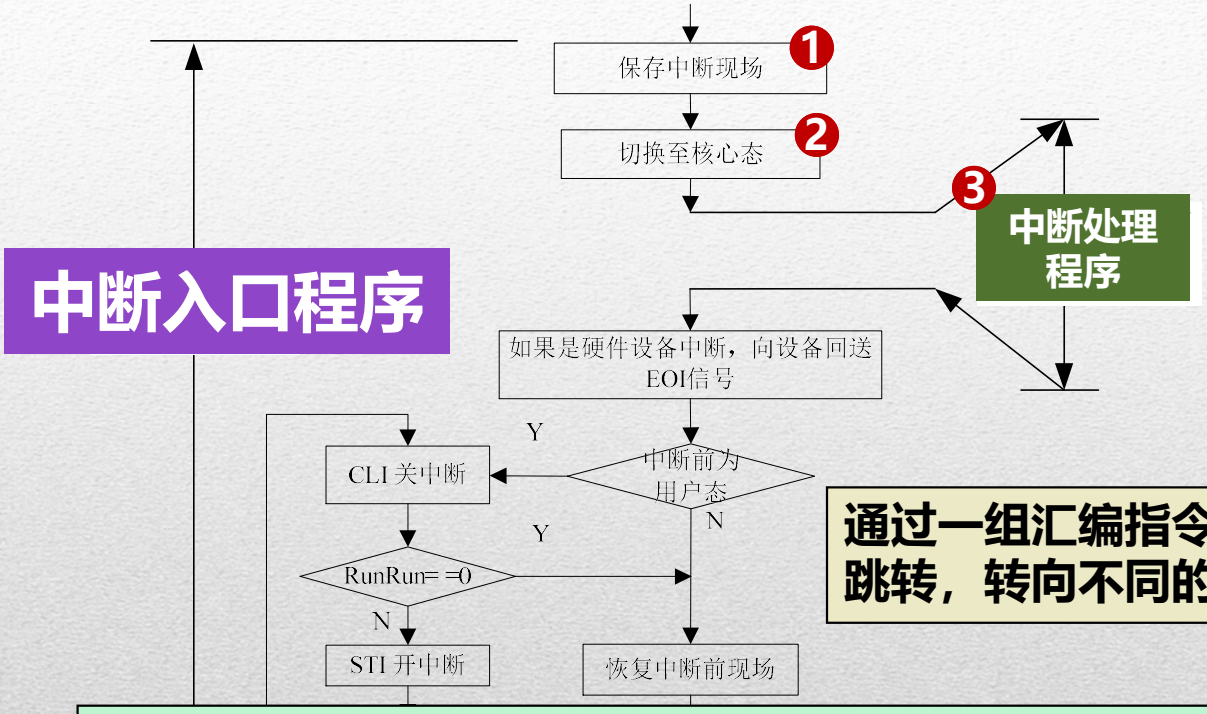


通过一组汇编指令，实现CPU指令跳转，转向不同的中断处理程序



中断发生时现运行进程在用户态运行

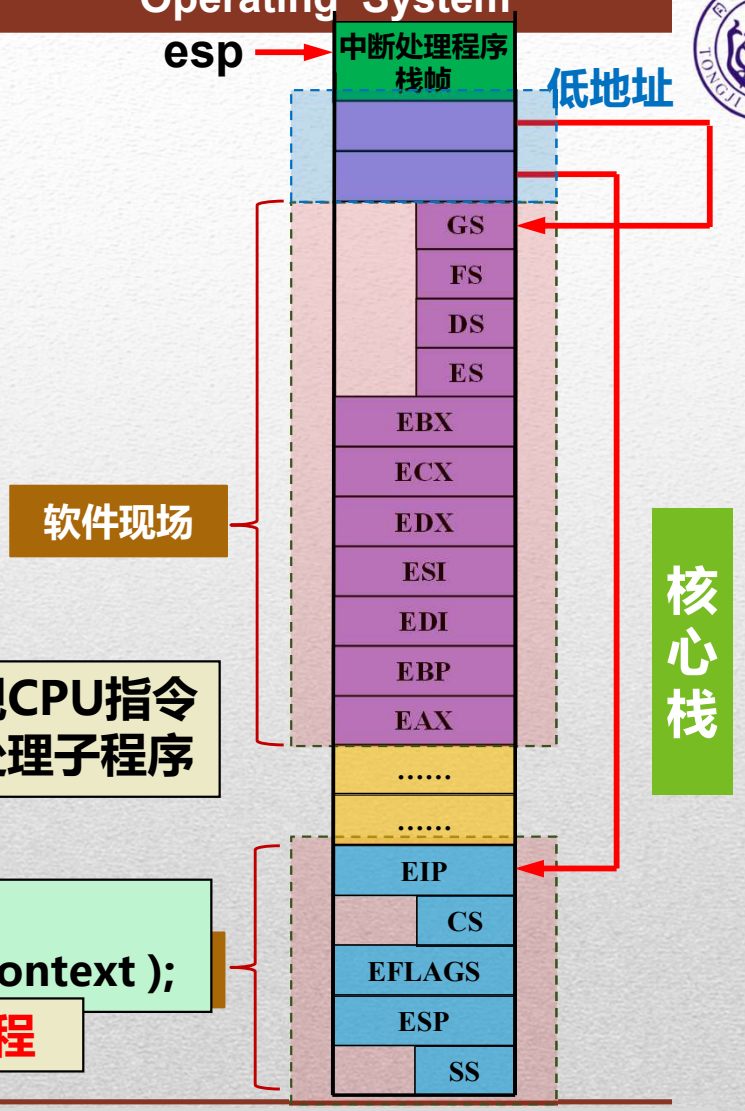
（3）：调用中断处理子程序



通过一组汇编指令，实现CPU指令跳转，转向不同的中断处理子程序

以时钟中断为例：
void Time::Clock(struct pt_regs* regs, struct pt_context* context);

中断处理子程序中会唤醒等待这次I/O操作而睡眠的进程



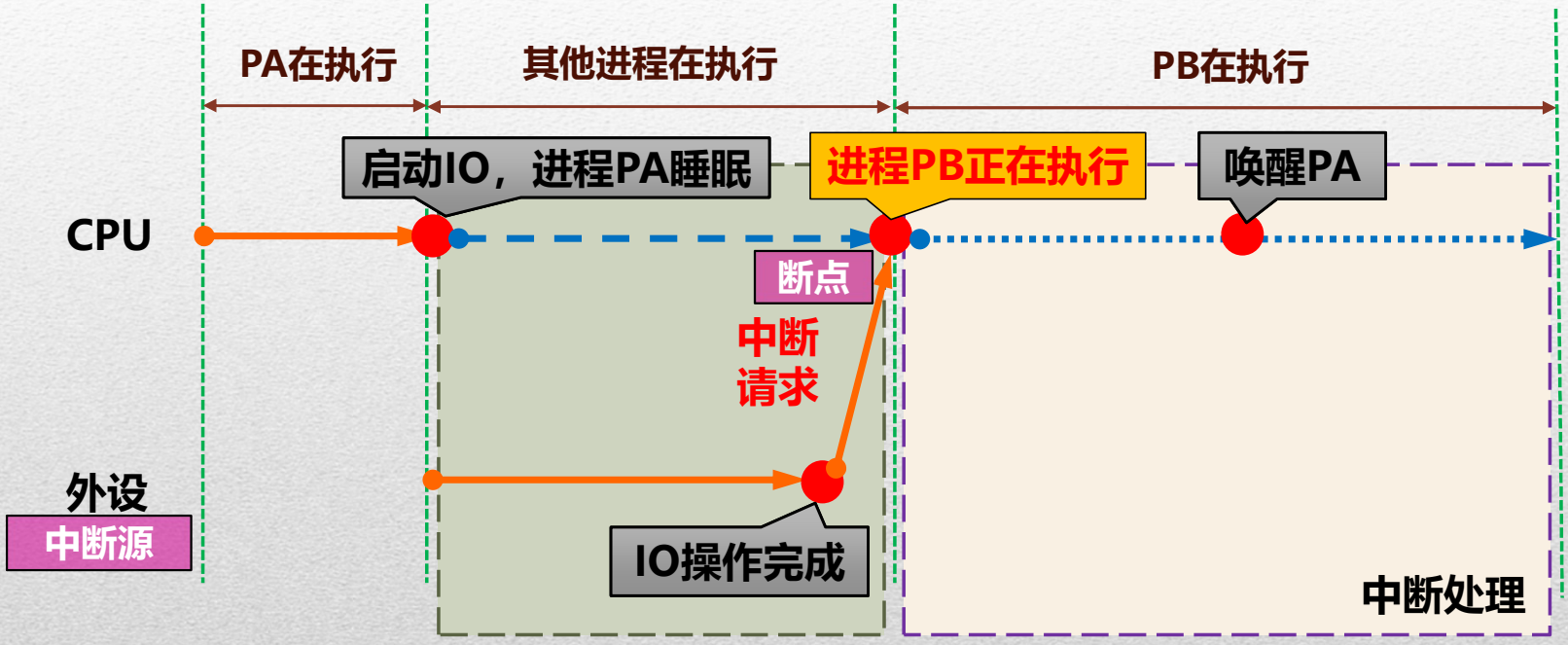
中断发生时现运行进程在用户态运行

什么是中断?

一种设备控制方式

一种外设的数据传输方式

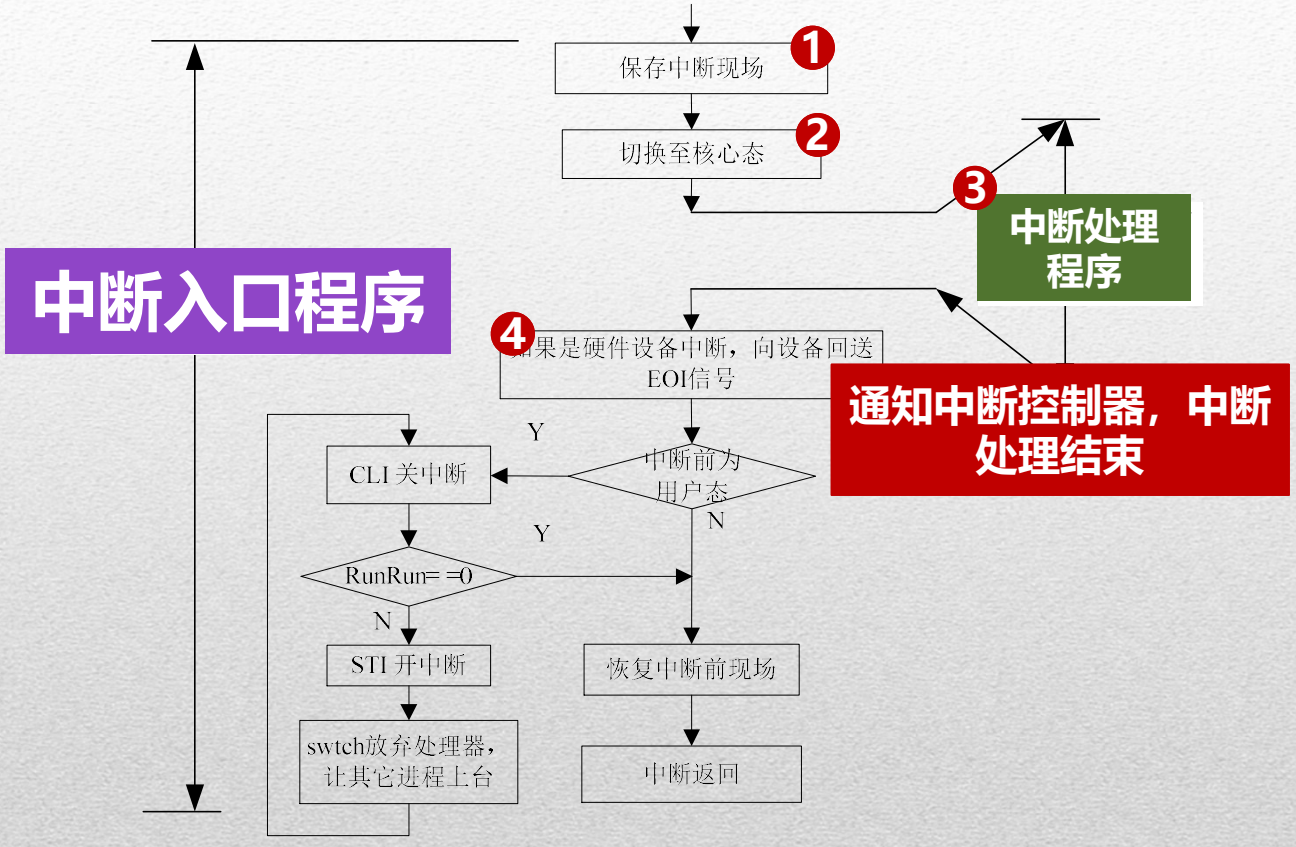
CPU不可能时刻查询外设的工作状态。。。

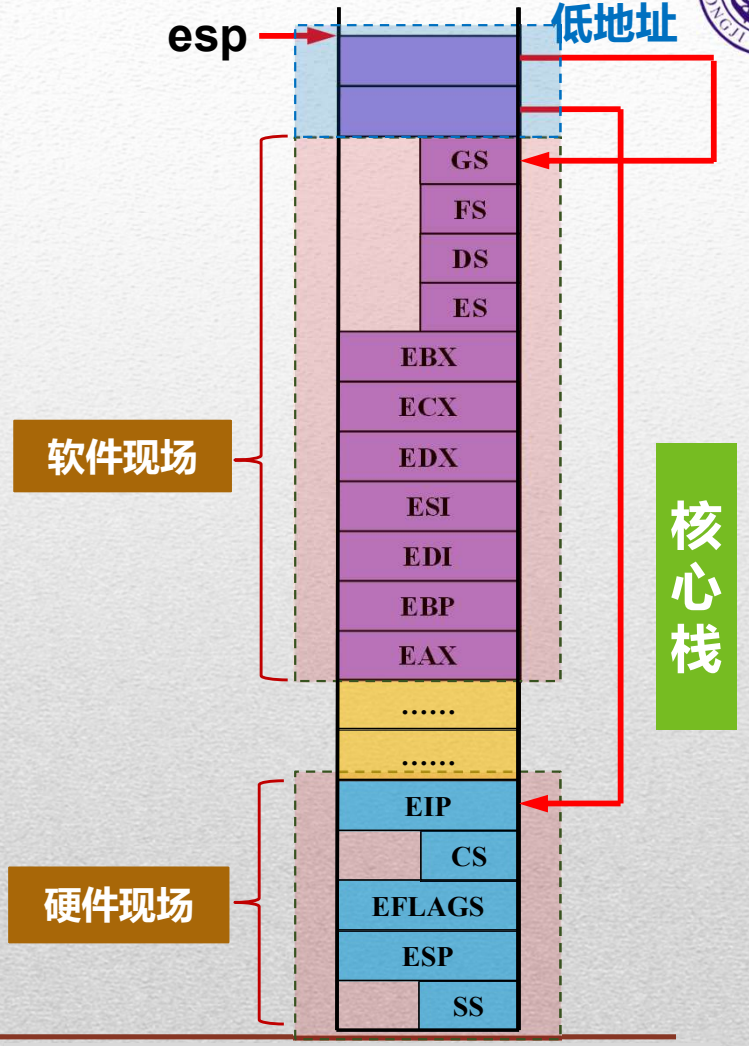


1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

（4）：发送EOI指令

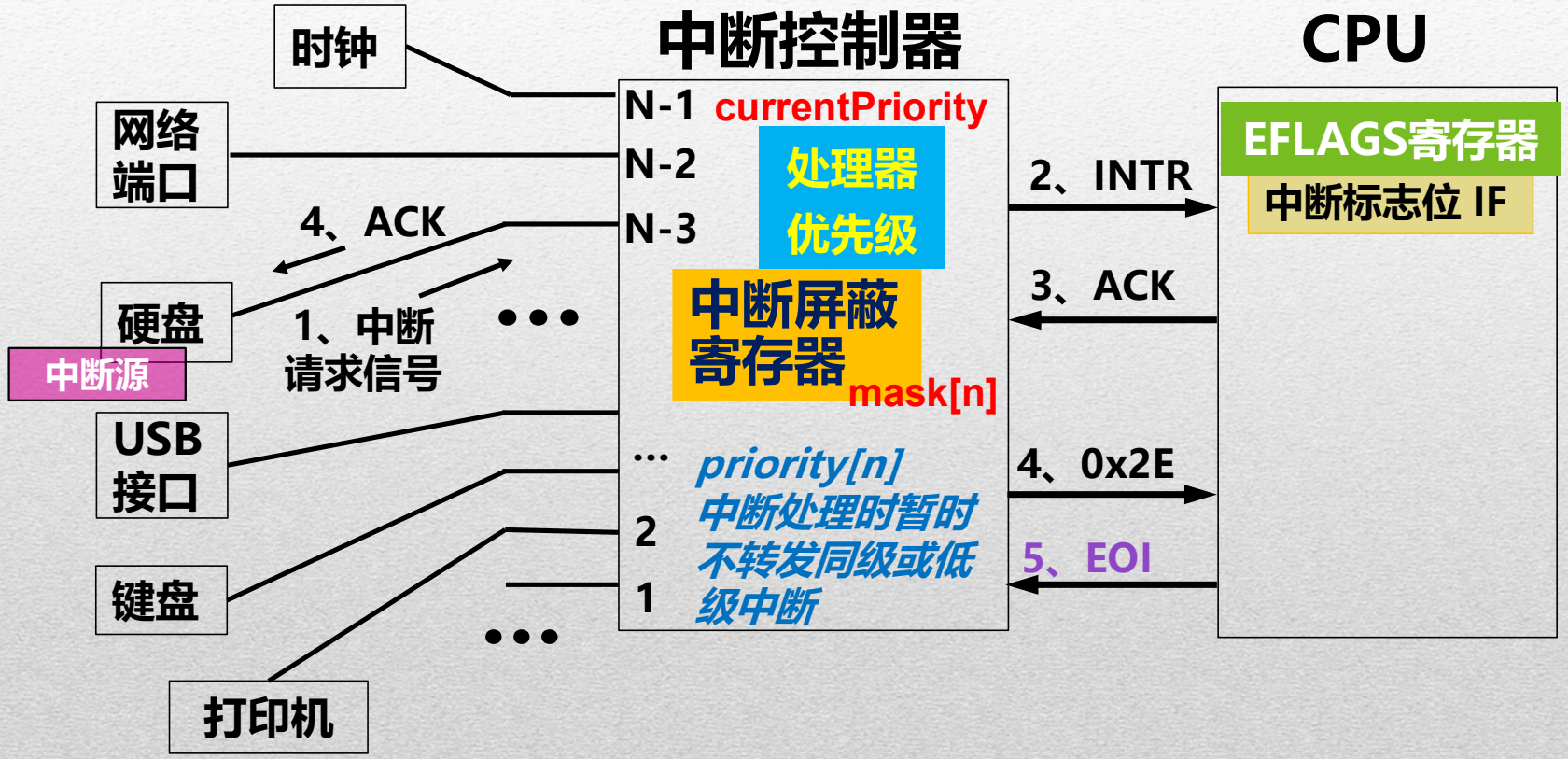




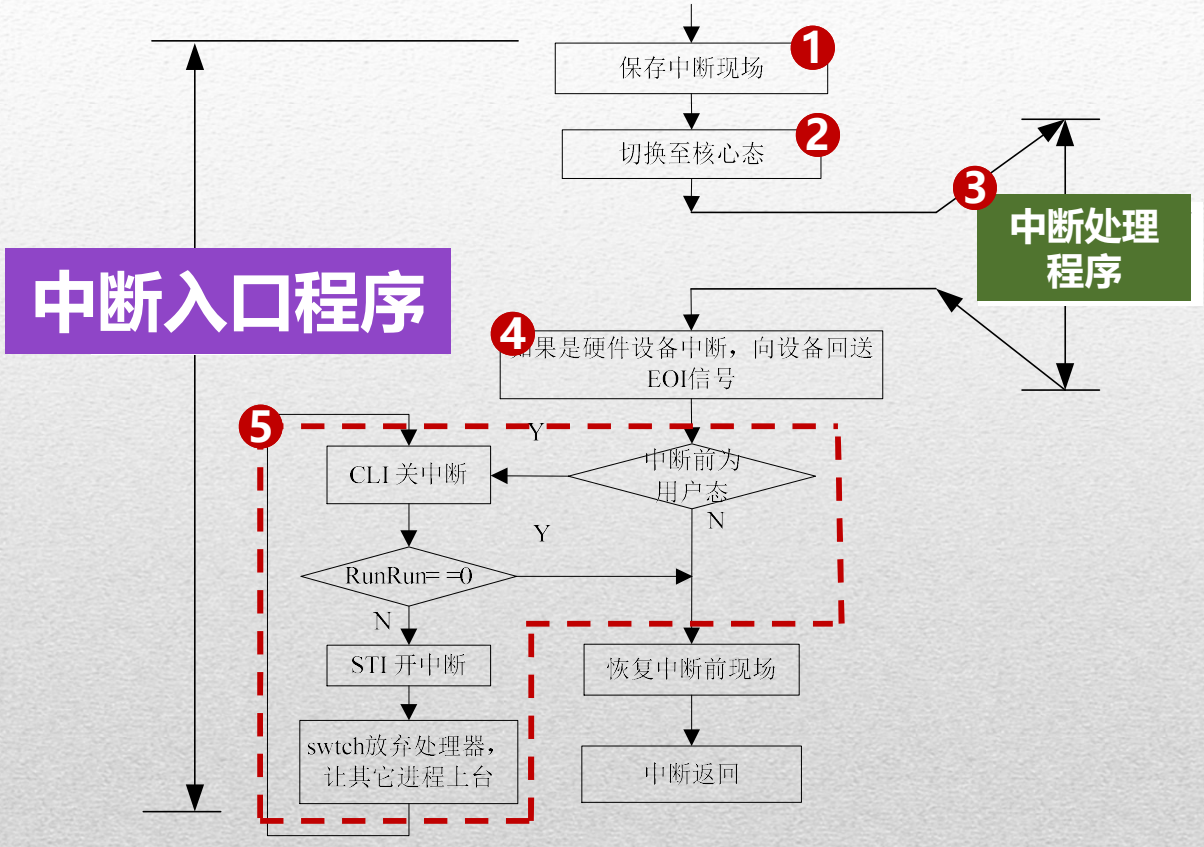
中断发生时现运行进程在用户态运行

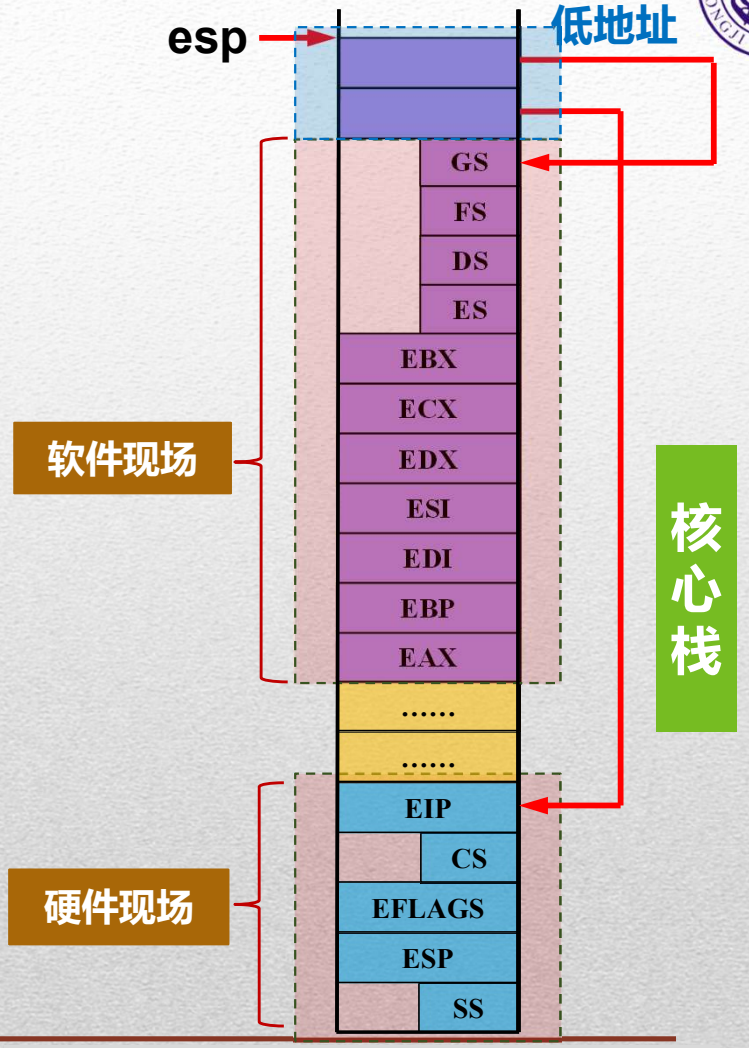
中断的硬件机构

4. 得到ACK信号的中断控制器送中断号给CPU。清除INTR请求信号，向外设发送ACK信号表示中断请求已被处理。



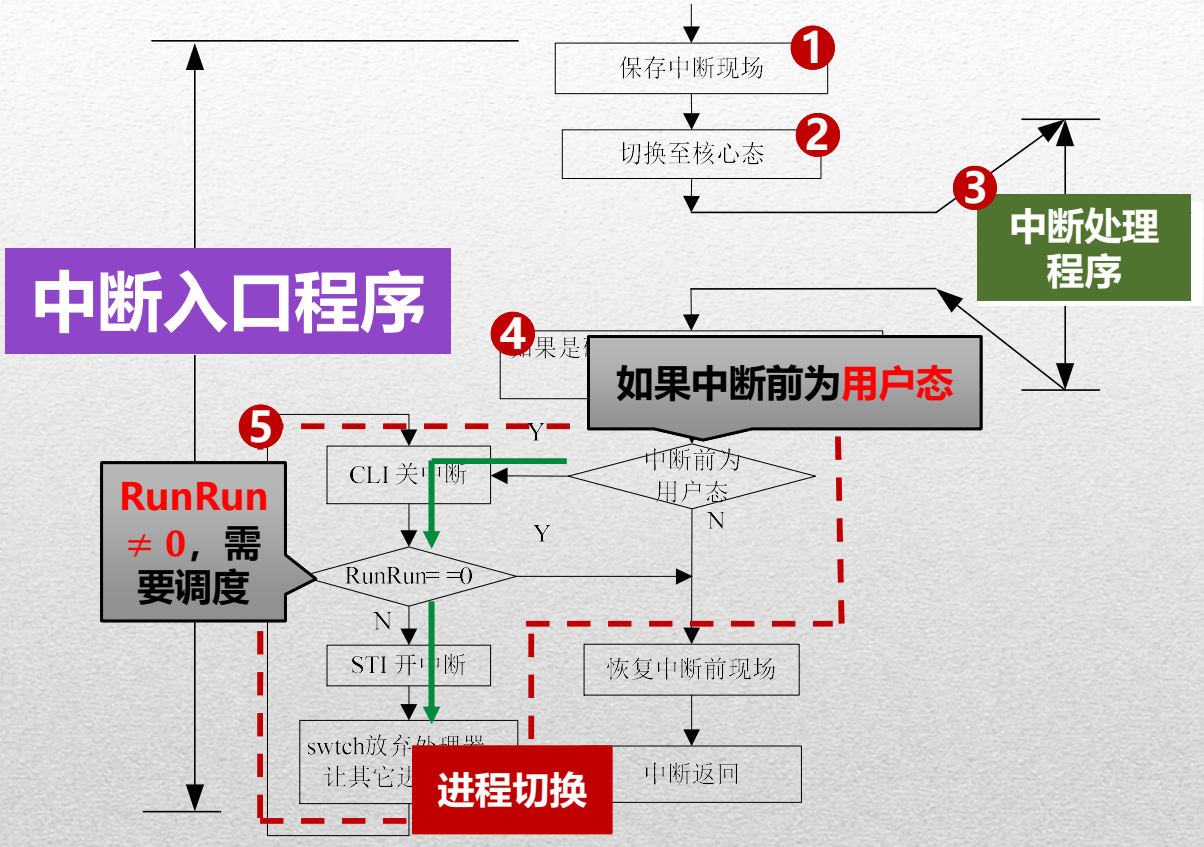
（5）：例行调度

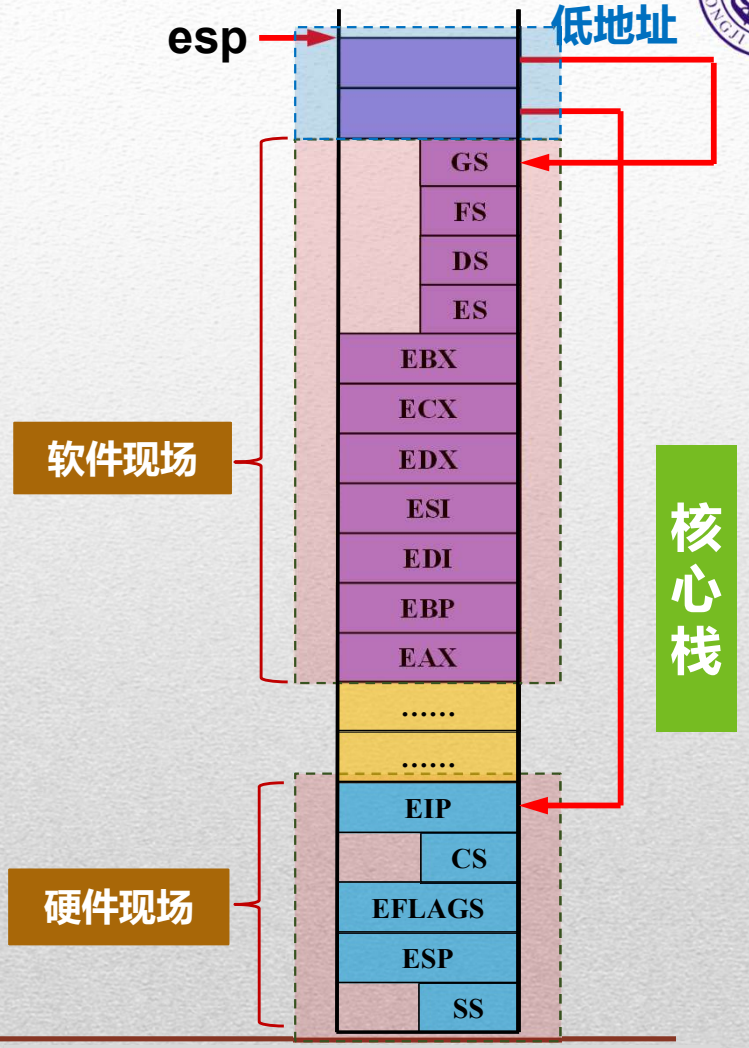




中断发生时现运行进程在用户态运行

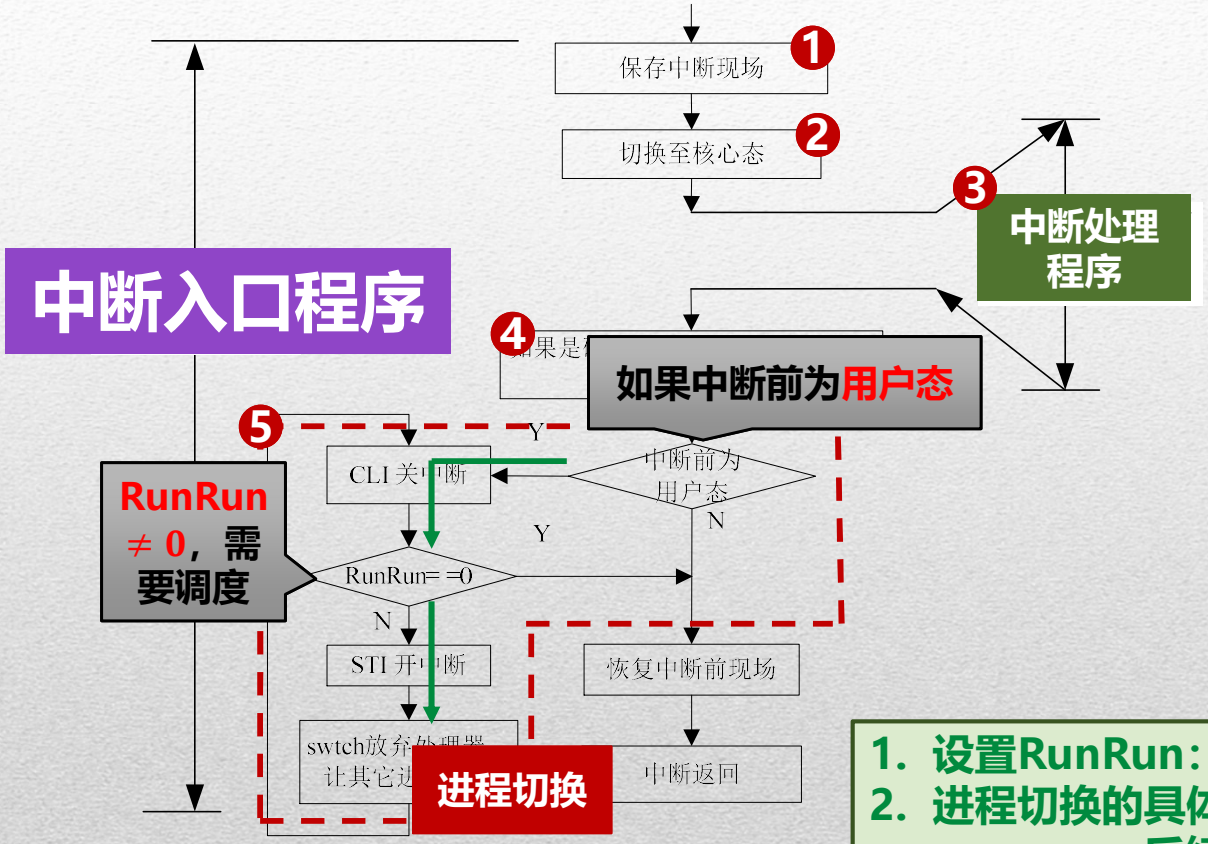
（5）：例行调度

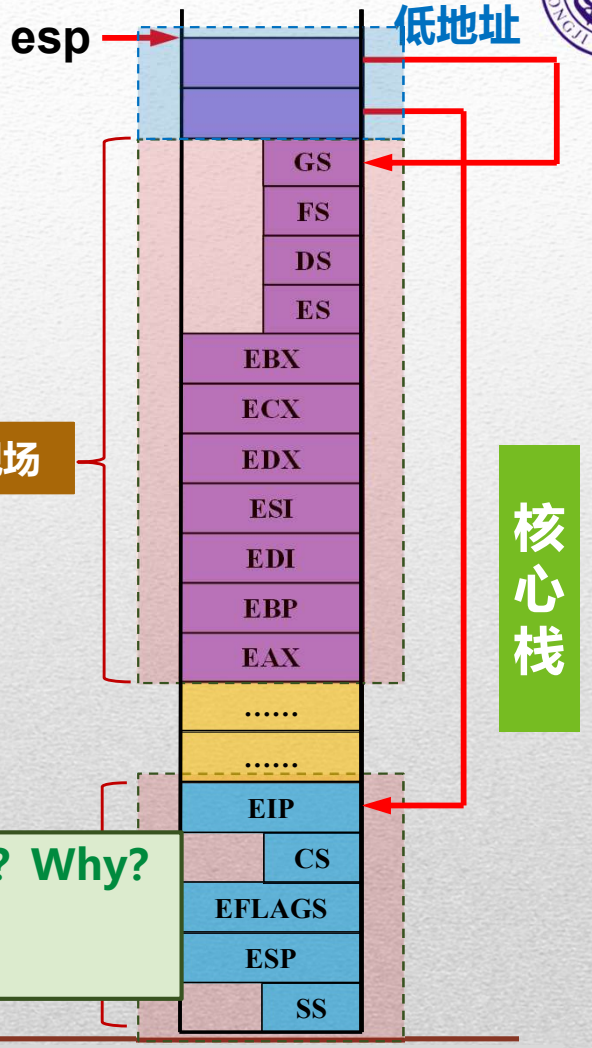




中断发生时现运行进程在用户态运行

（5）：例行调度

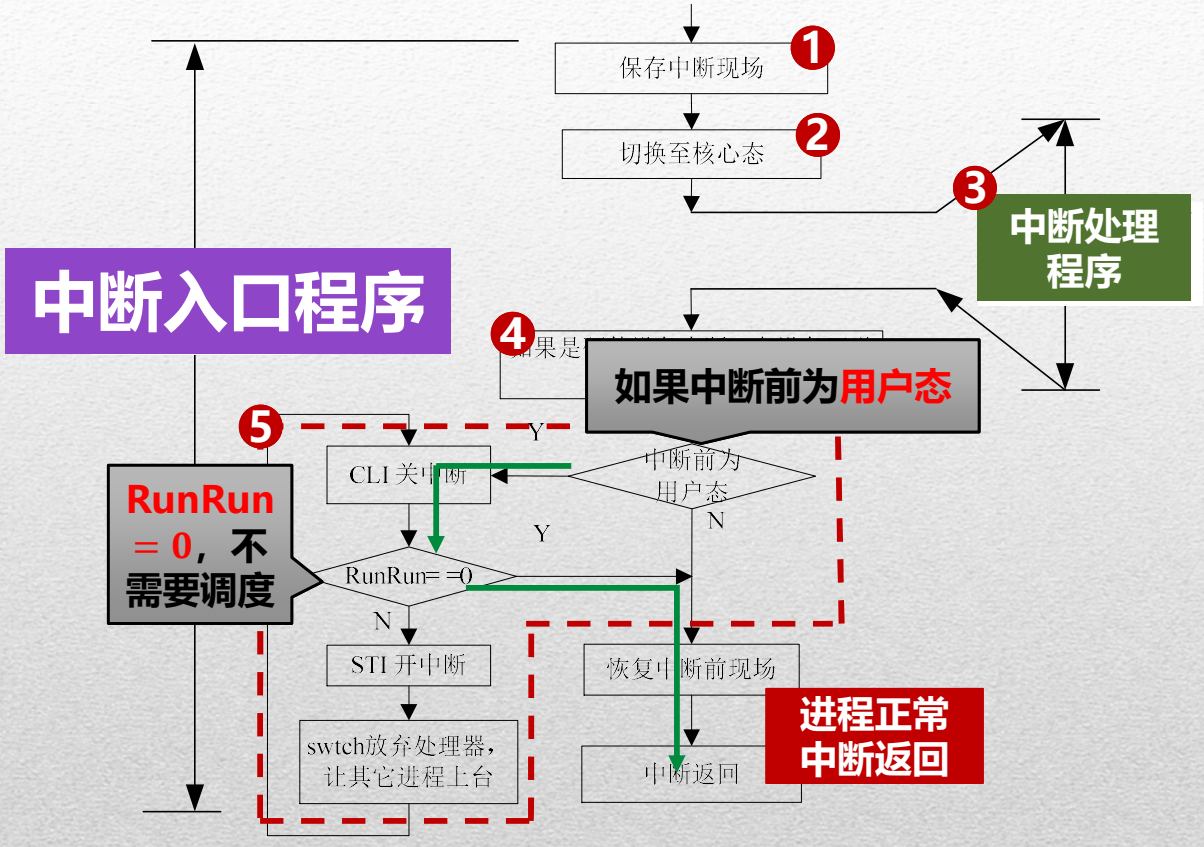


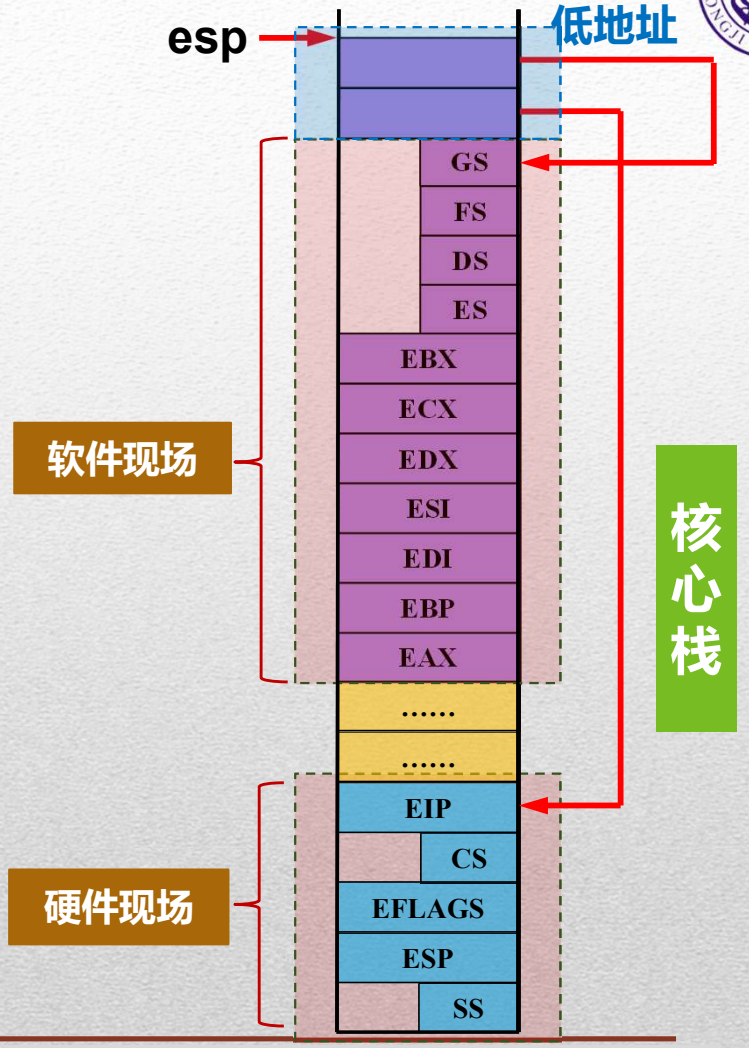


1. 设置RunRun：Who? When? Why?
2. 进程切换的具体过程
   后续章节讲解

中断发生时现运行进程在用户态运行

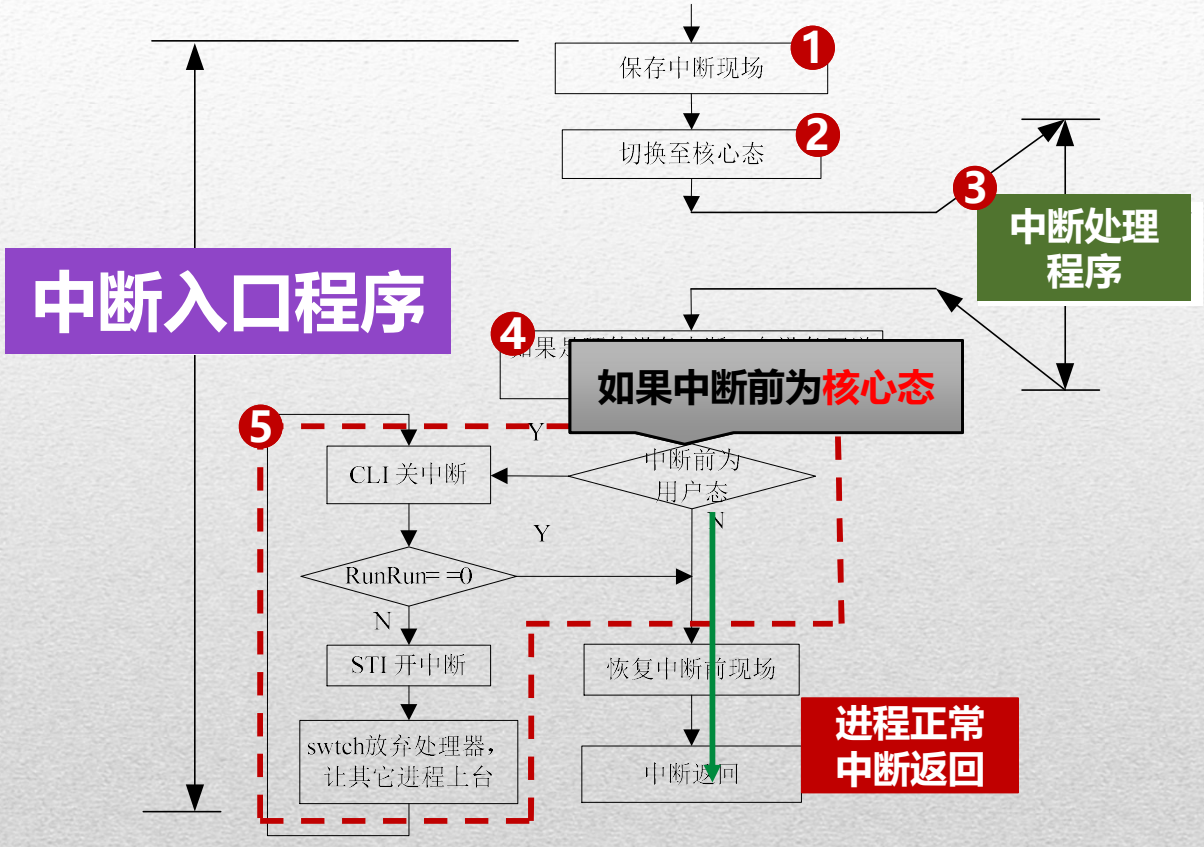
（5）：例行调度

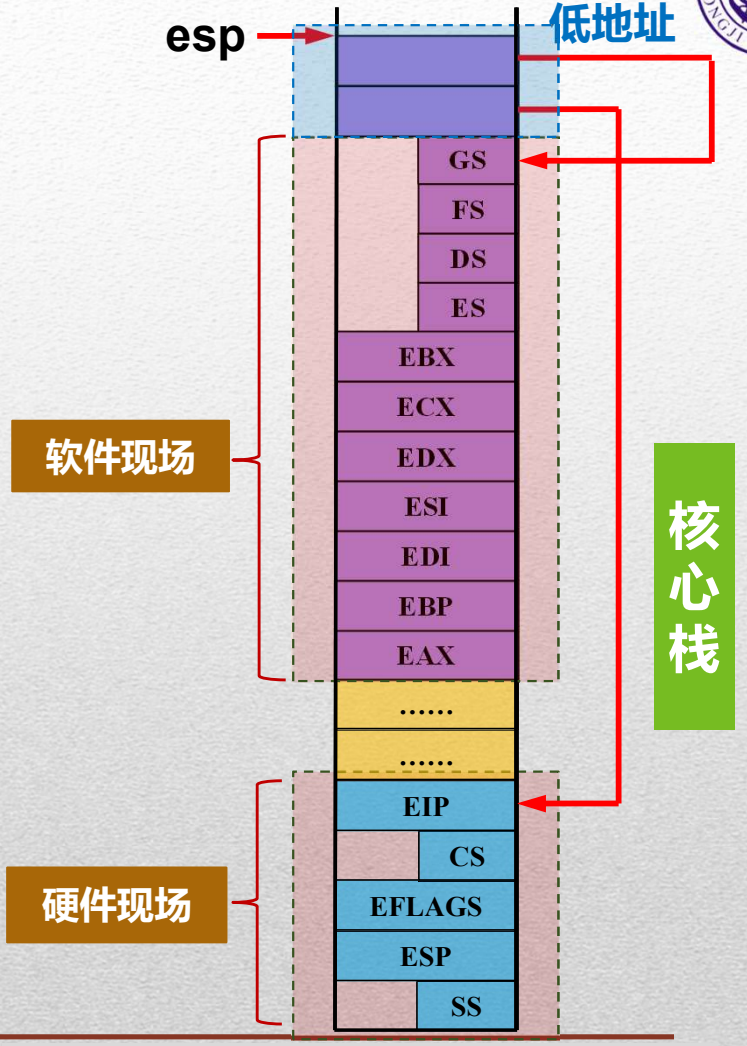




中断发生时现运行进程在用户态运行

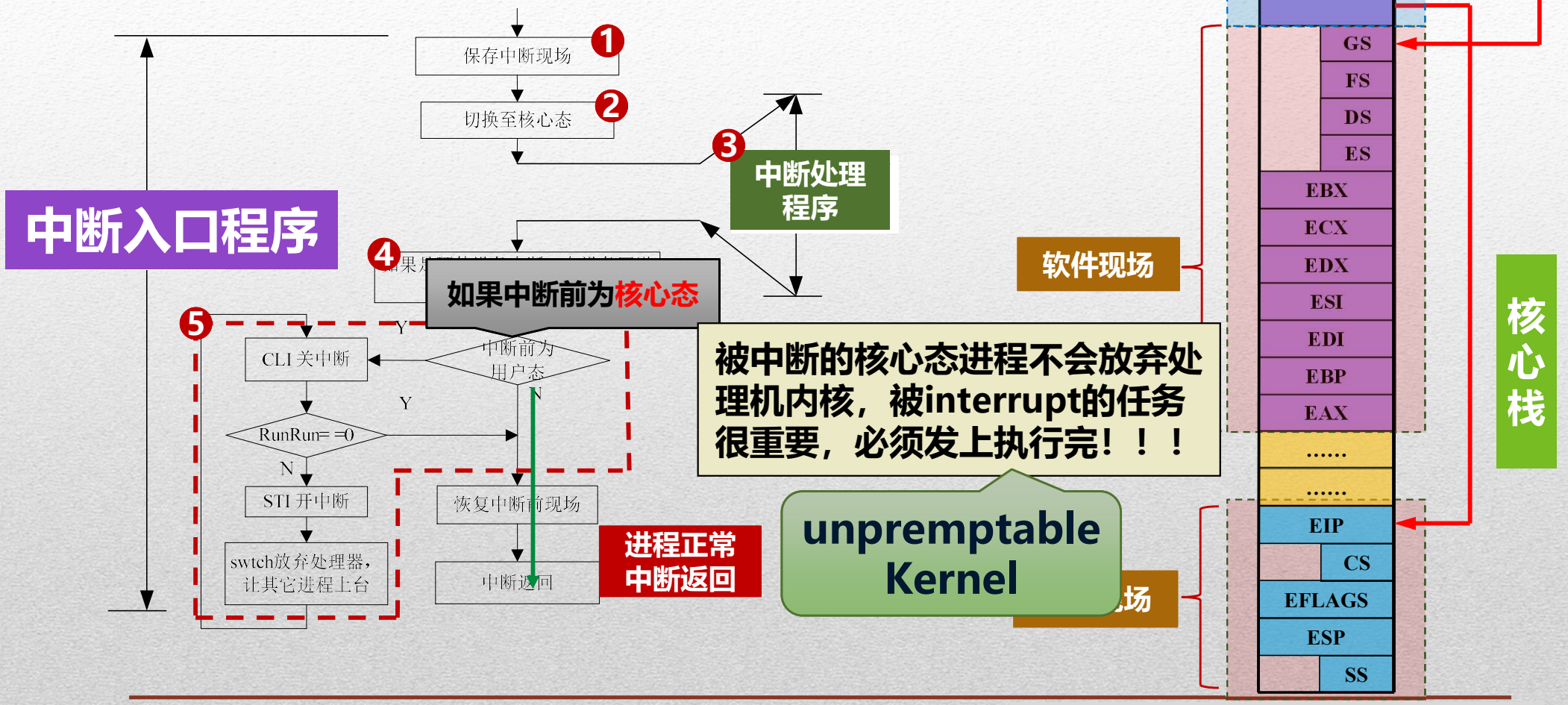
（5）：例行调度





中断发生时现运行进程在用户态运行

（5）：例行调度

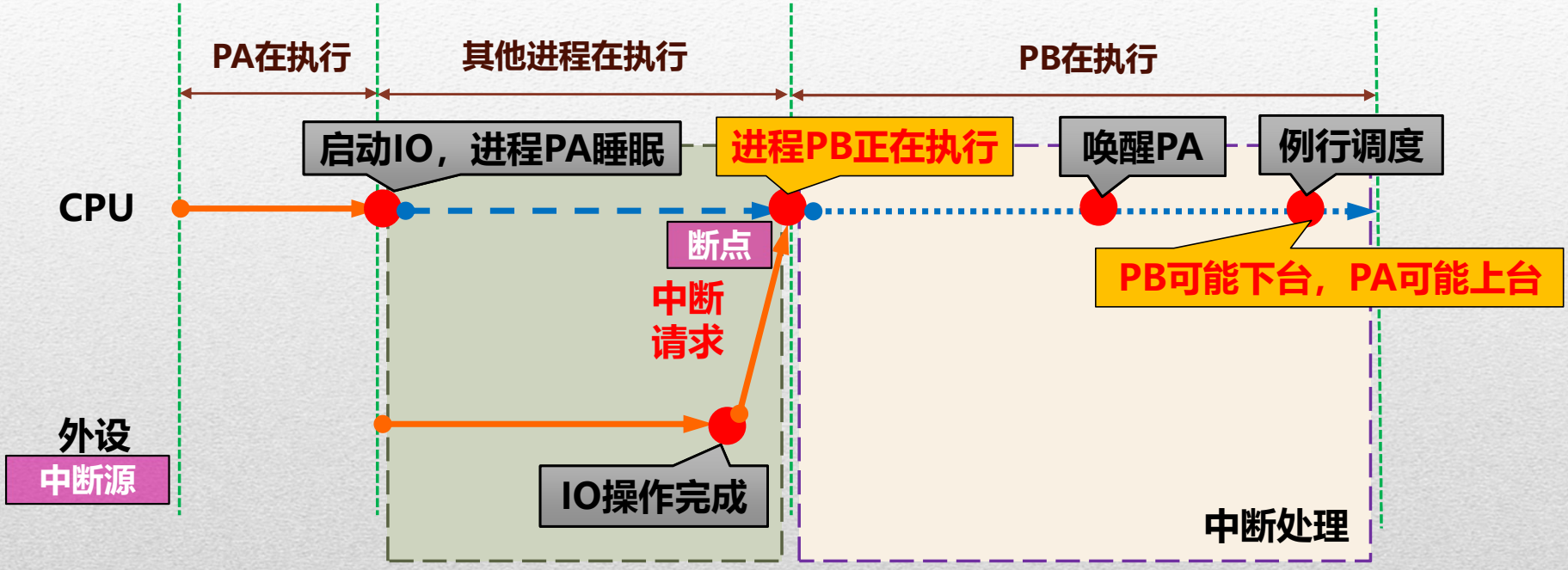

中断入口程序
保存中断现场
1
切换至核心态
2
3
中断处理程序
4
如果中断前为核心态
5
Y
中断前为用户态
N
CLI 关中断
Y
RunRun= =0
N
STI 开中断
swtch放弃处理器，让其它进程上台
恢复中断前现场
中断返回
进程正常中断返回
被中断的核心态进程不会放弃处理机内核，被interrupt的任务很重要，必须发上执行完！！！
unpremptable Kernel
esp
低地址
GS
FS
DS
ES
EBX
ECX
EDX
ESI
EDI
EBP
EAX
.....
.....
EIP
CS
EFLAGS
ESP
SS
软件现场
核心栈
中断发生时现运行进程在用户态运行

什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。

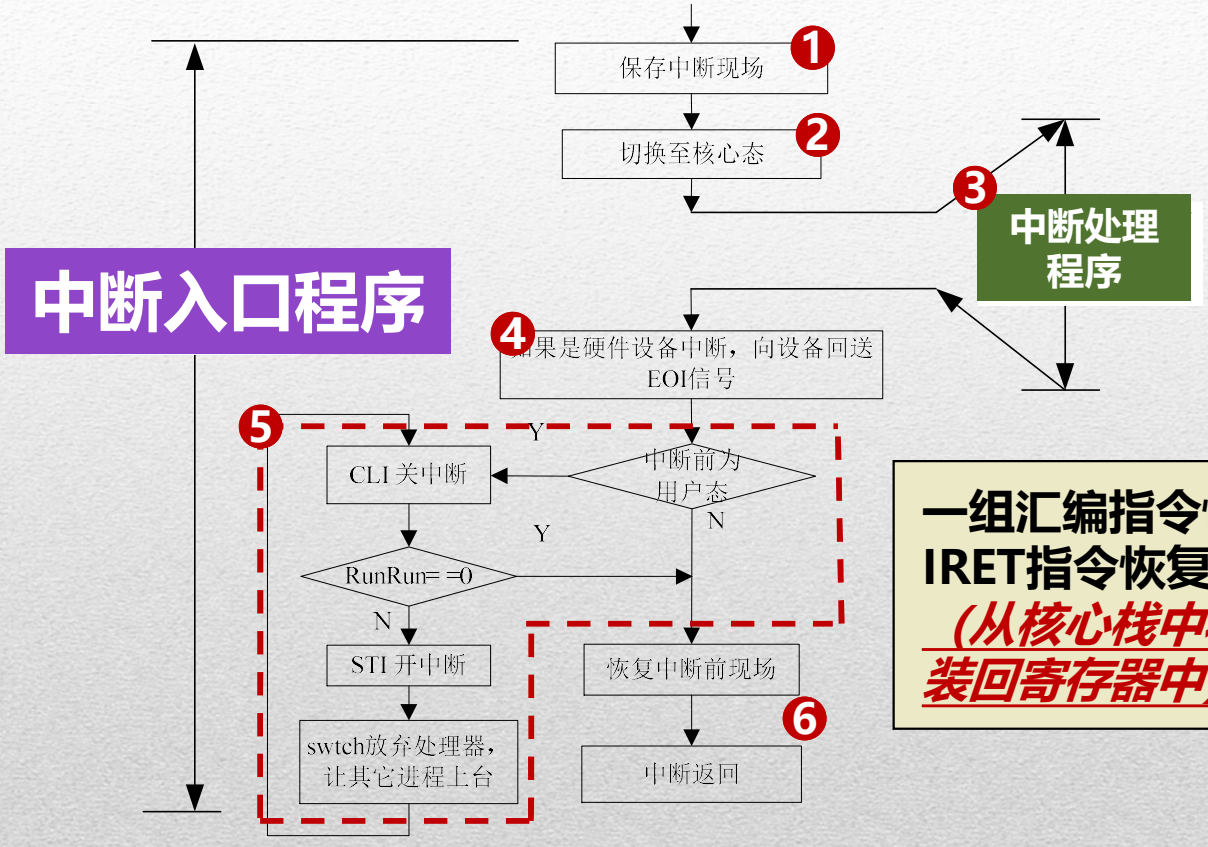


1. 保证了CPU和设备之间的并行操作

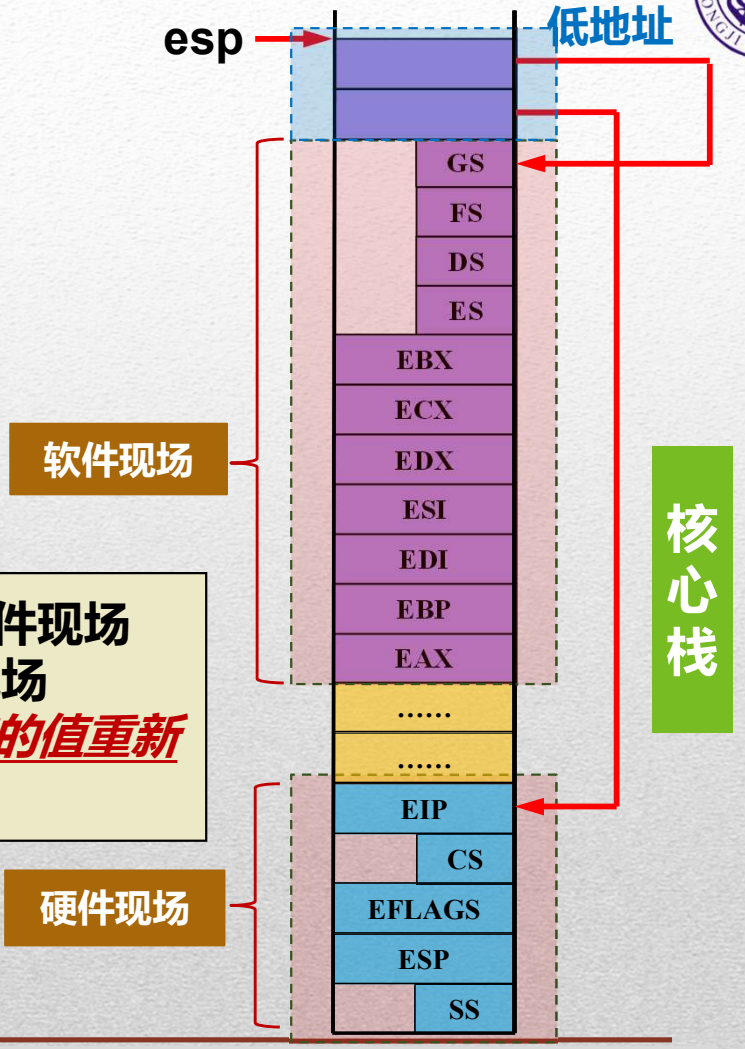
2. 提供了进程执行内核代码的机会

3. 多道程序并发的硬件基础

（6）：恢复现场和中断返回

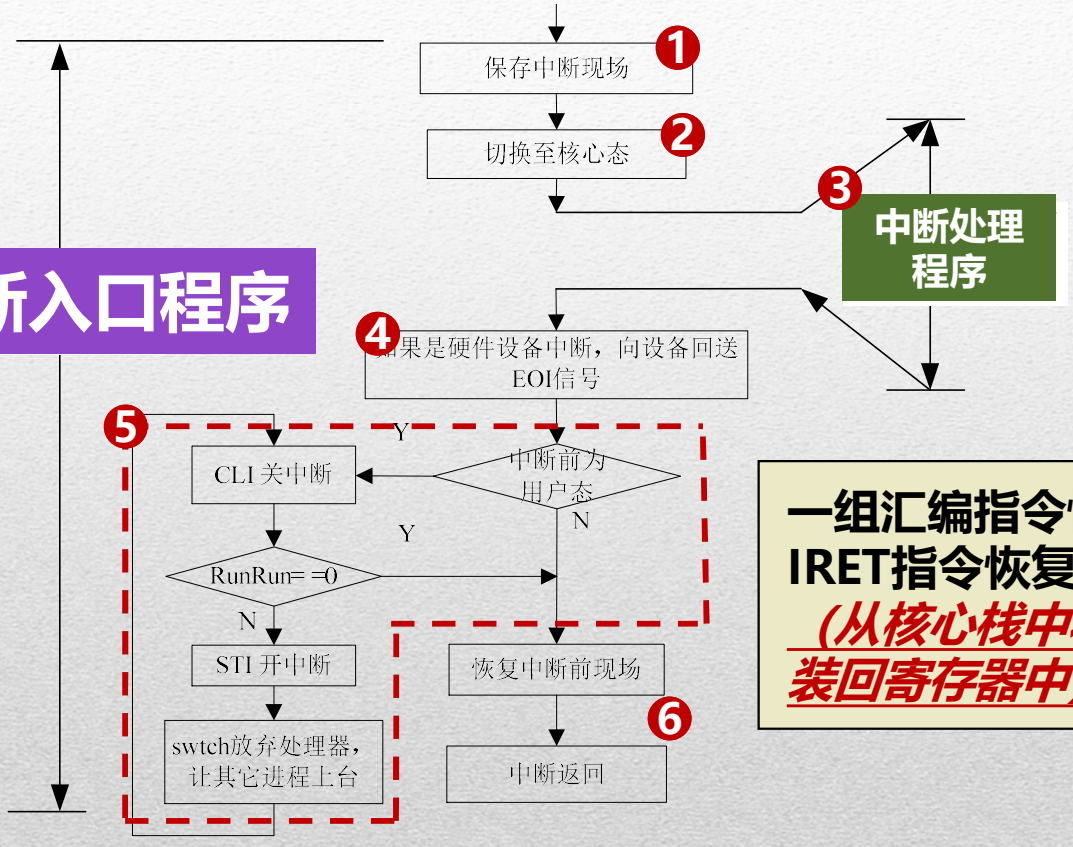


一组汇编指令恢复软件现场
IRET指令恢复硬件现场
（从核心栈中将相应的值重新装回寄存器中）



中断发生时现运行进程在用户态运行

（6）：恢复现场和中断返回

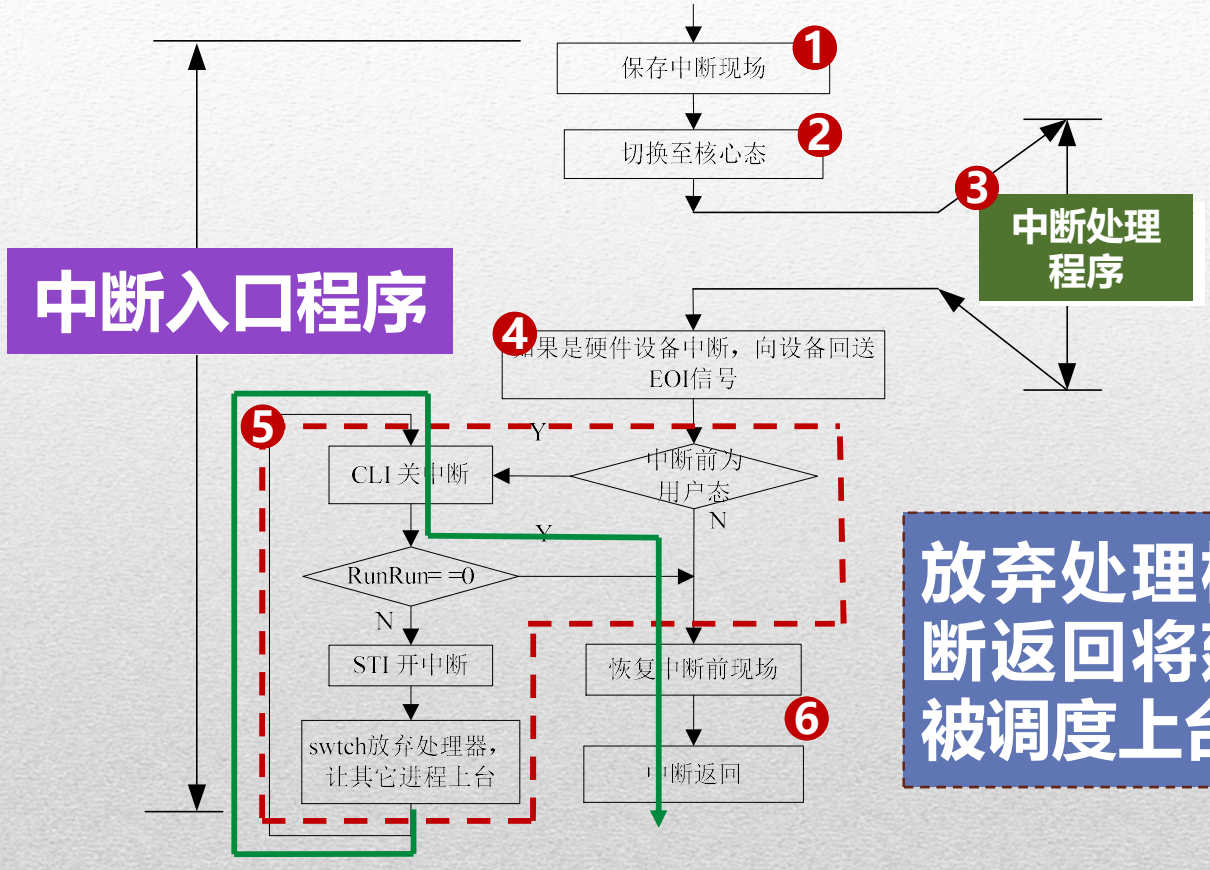


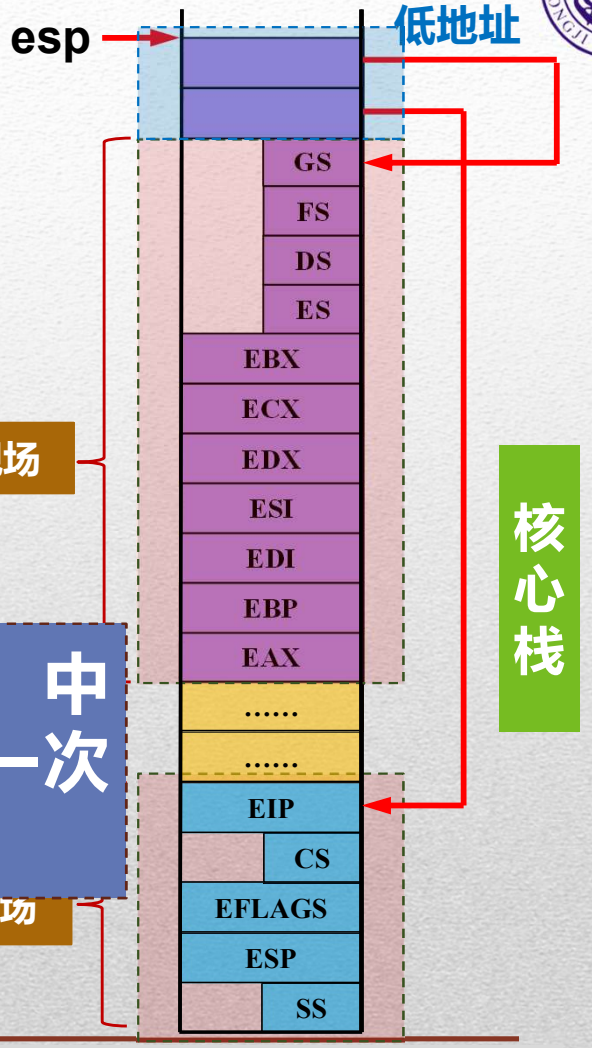
一组汇编指令恢复软件现场
IRET指令恢复硬件现场
（从核心栈中将相应的值重新装回寄存器中）

核心栈

中断发生时现运行进程在用户态运行

（6）：恢复现场和中断返回





放弃处理机的进程，中断返回将延迟至下一次被调度上台

中断发生时现运行进程在用户态运行

什么是中断?

一种设备控制方式

一种外设的数据传输方式

CPU不可能时刻查询外设的工作状态。。。

1. 保证了CPU和设备之间的并行操作
2. 提供了进程执行内核代码的机会
3. 多道程序并发的硬件基础

如何看待中断?

站在进程PA的角度：		站在进程PB的角度：	
执行	我要用外设	我正在执行自己的程序	执行
	CPU太快不等我	中断来了	被选为班长
阻塞	先把CPU让给别人	被迫执行内核代码	
	中断处理中我被叫醒	叫醒一个和我无关的进程	
就绪	等下一次上台的机会	中断结束可能被剥夺CPU	就绪

如何看待中断?

1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

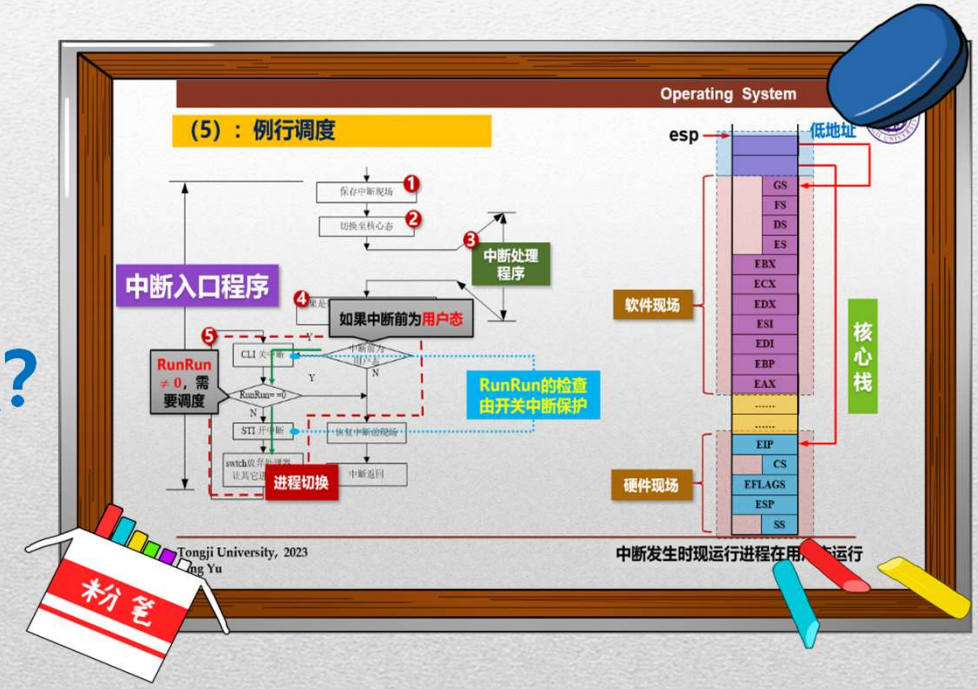
3. 多道程序并发的硬件基础

CPU

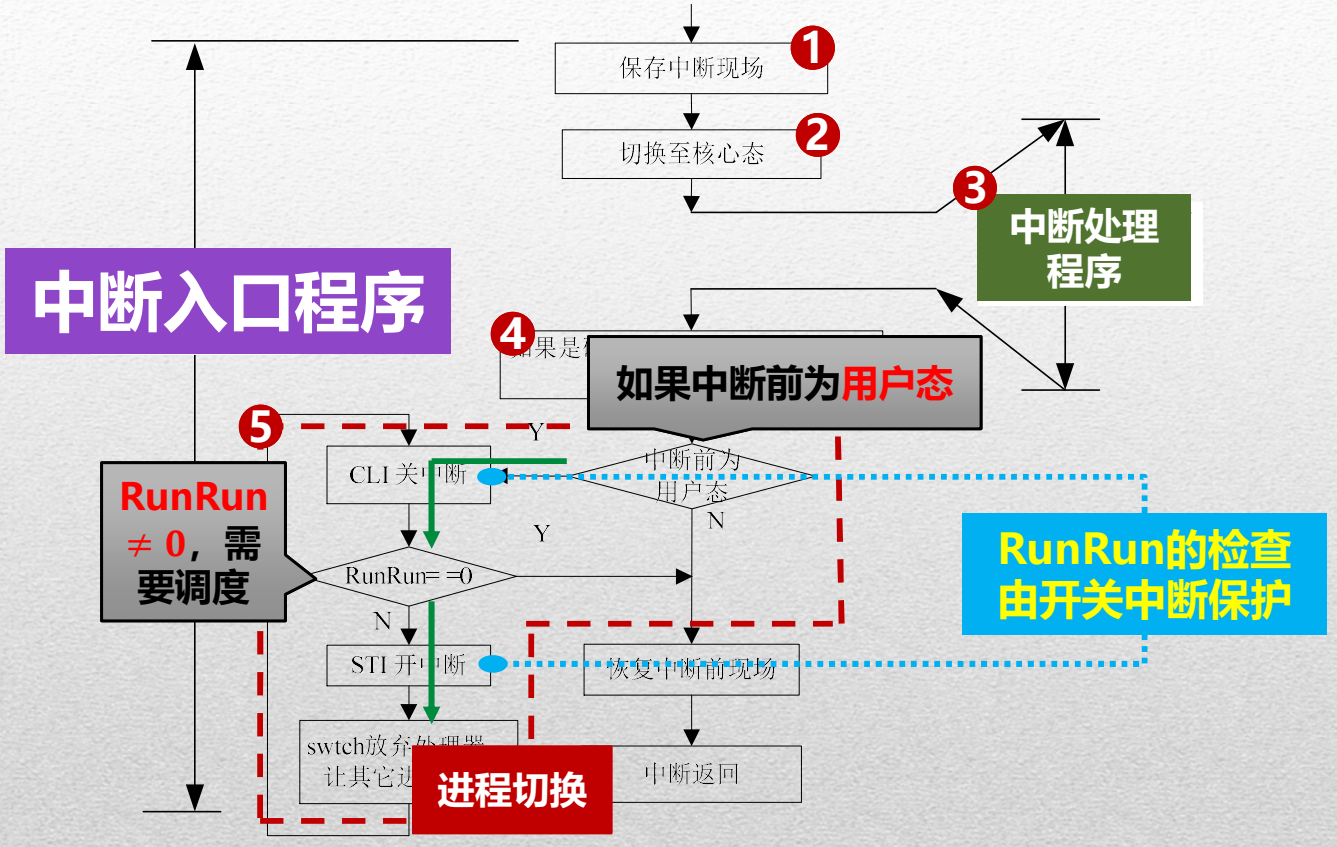
EFLAGS寄存器

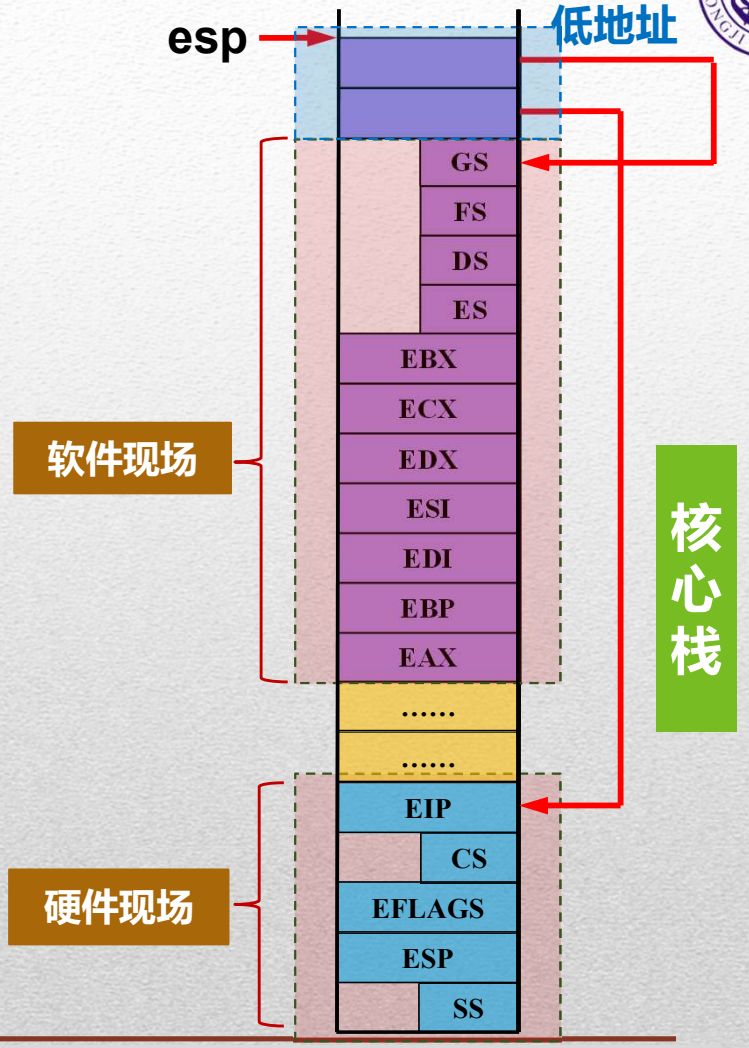
中断标志位 IF

Q3：如果中断关闭会发生什么?



（5）：例行调度





中断发生时现运行进程在用户态运行

如何看待中断?

1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

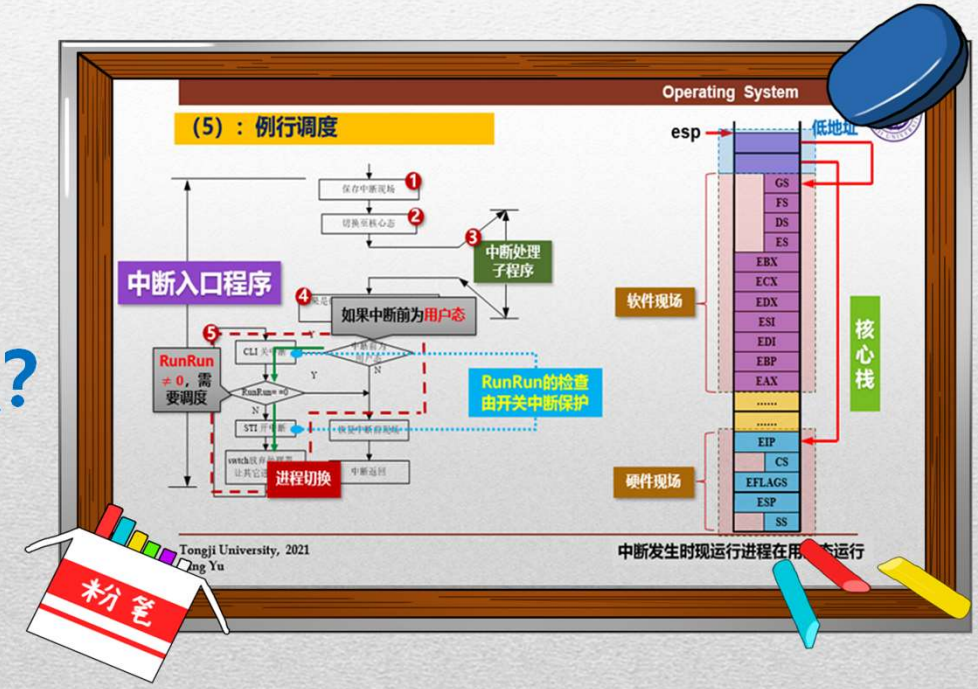
3. 多道程序并发的硬件基础

CPU

EFLAGS寄存器

中断标志位 IF

Q3：如果中断关闭会发生什么?



本节小结：

1　中断的基本概念

2　UNIX中断的处理过程

E03：并发进程（UNIX进程与中断）