第2章 相关硬件基础

目录

2.1中央处理器

2.1.1单处理器系统和多处理器系统

2.1.2寄存器

2.1.3特权指令与非特权指令

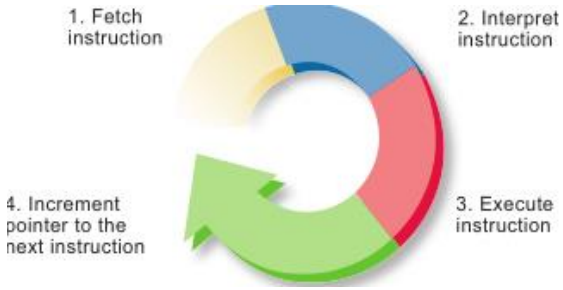
2.1.4处理器状态

2.1.5程序状态字寄存器

2.1.6 x86总体架构

2.1.1单处理器和多处理器系统

- 计算机系统的核心是中央处理器
  - 中央处理器的主要部件：
    - 控制器（控制单元，control unit）
    - 运算器（算数逻辑单元，Arithmetic/Logic Unit ）
    - 寄存器 （Register）
    - 高速缓存（cache）
  - 中央处理器的任务：执行指令
  - 中央处理器的工作流程：
    - 取指→译码→取操作数→执行指令

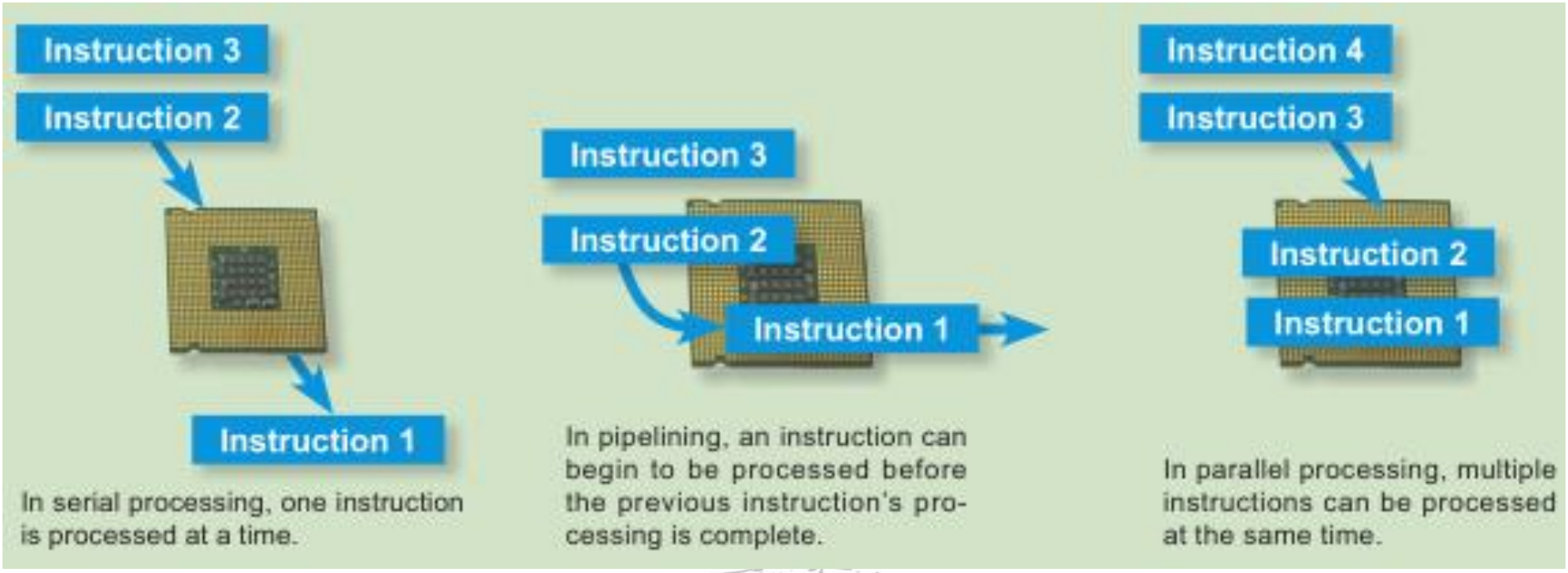


影响中央处理器性能的因素

- 影响中央处理器性能的因素：
  - 主频（clock rate）、字长（word size）、指令集（instruction set）、高速缓存（cache size）、核心数量（number of core）、 FSB 速度（ front side bus speed)、处理技术（processing techniques）等。

处理技术

- 串行 serial processing
- 流水线 pipelining processing
- 并行 parallel processing



2.1.1单处理器和多处理器系统

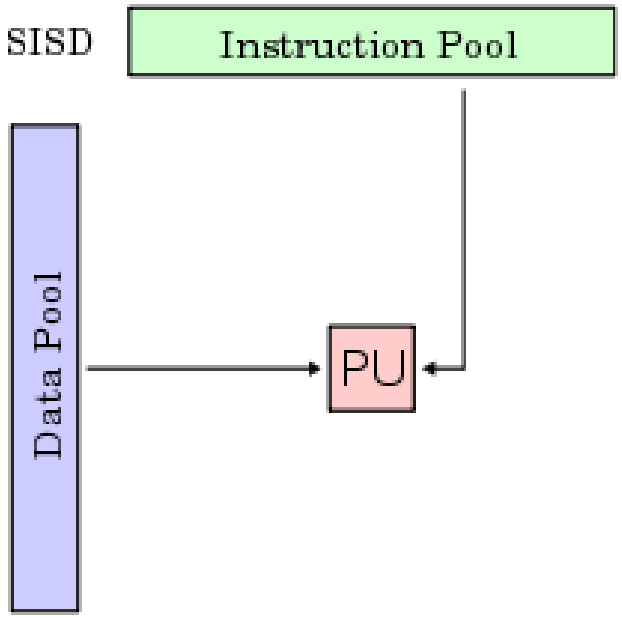
- 两类系统：
  - 单处理器系统：一个计算机系统只包括一个运算处理器。
  - 多处理器系统：一个计算机系统有多个运算处理器。
    - 对称多处理系统
    - 非对称多处理系统

计算机系统结构分类

- Flynn于1972年提出了计算平台的Flynn分类法，主要根据指令流和数据流来分类，共分为四种类型的计算平台。
  - 单指令流单数据流(SISD)
  - 单指令流多数据流(SIMD)
  - 多指令流单数据流(MISD)
  - 多指令流多数据流(MIMD)

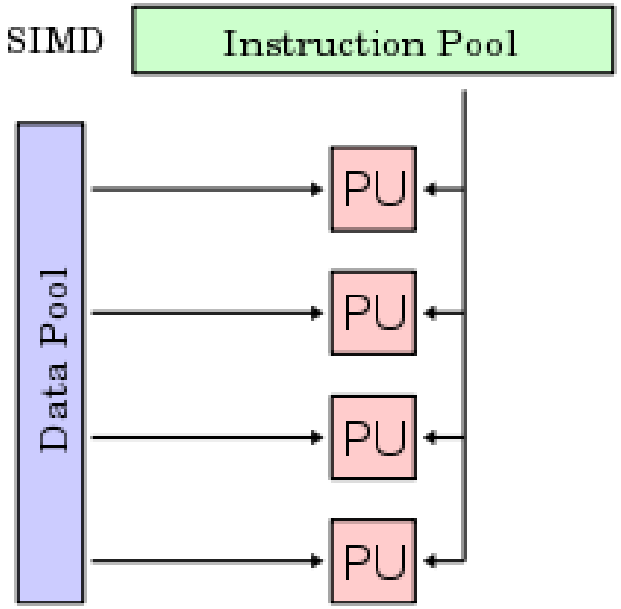
计算机系统结构分类

- 单指令流单数据流(SISD)：一个处理器在一个存储器中的数据上执行单条指令流。
- SISD其实就是传统的顺序执行的单处理器计算机。



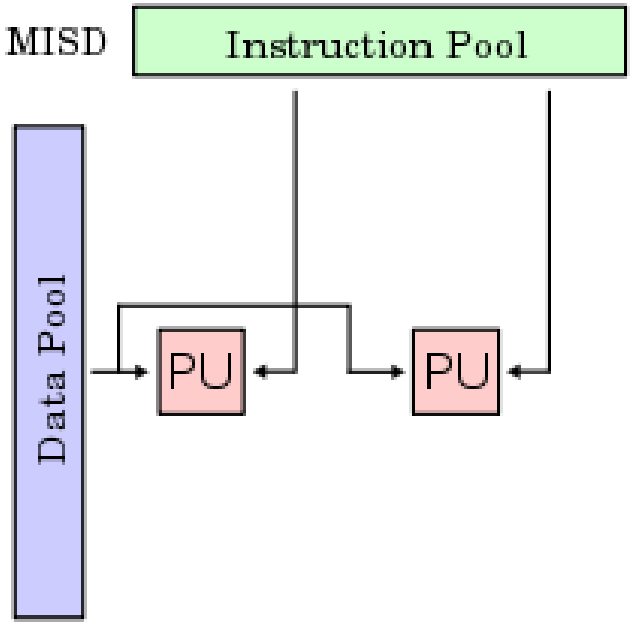
计算机系统结构分类

- 单指令流多数据流(SIMD)：单条指令流控制**多个处理单元**同时执行，每个处理单元包括处理器和相关的数据存储，**一条指令控制**了不同的处理器对**不同的数据进行操作。**
- 向量机和阵列机是这类计算机系统的代表



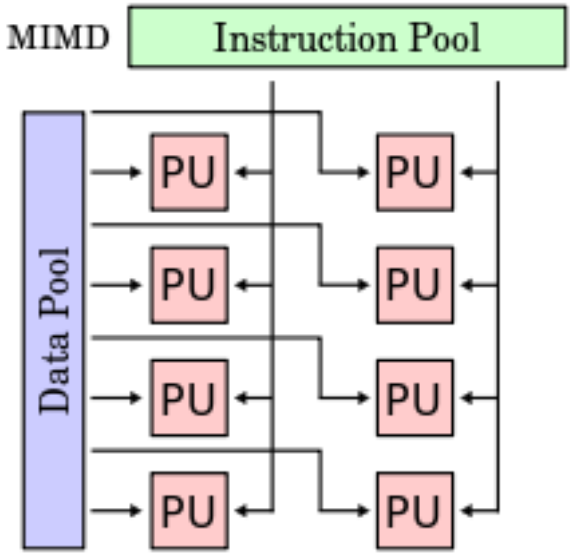
计算机系统结构分类

- 多指令流单数据流(MISD)：一个**数据流**被传送给**一组处理器**，通过处理器上不同指令操作最终得到处理结果
- 这类系统实际上很少见到



计算机系统结构分类

- 多指令流多数据流(MIMD)：多个处理器对各自不同的数据集同时执行不同的指令流。
- MIMD是能实现作业、任务、指令等各级全面并行的多机系统。如多核处理器、多处理机属于MIMD。
- 可以把MIMD系统划分为共享内存紧密耦合系统和内存分布松散耦合系统两类（共享内存型还可再分为MSP(Master/Slave)和SMP(Symmetric)）



计算机系统结构分类

体系结构类型	结构	关键特性	代表
单指令流单数据流 SISD	控制部分：一个 处 理 器：一个 主存模块：一个		单处理器系统
单指令流多数据流 SIMD	控制部分：一个 处 理 器：多个 主存模块：多个	各处理器以异步的形式执行同一条指令	并行处理机 阵列处理机 超级向量处理机
多指令流单数据流 MISD	控制部分：多个 处 理 器：一个 主存模块：多个	被证明不可能，至少是不实际	目前没有，有文献称流水线计算机为此类
多指令流多数据流 MIMD	控制部分：多个 处 理 器：多个 主存模块：多个	能够实现作业、任务、指令等各级全面并行	多处理机系统 多计算机

Flynn分类法四类计算机结构比较图

多处理器和多核处理器

- 多处理器指在一个体系结构上放置多个(单核)CPU芯片，而多核则指在同一块CPU芯片上放置多个核（core），即执行单元。
- 多CPU和多核的区别是后者更加紧凑，成本更低、功耗更小。
- 多CPU结构和多核结构可统称为多核结构。

多处理器和多核处理器

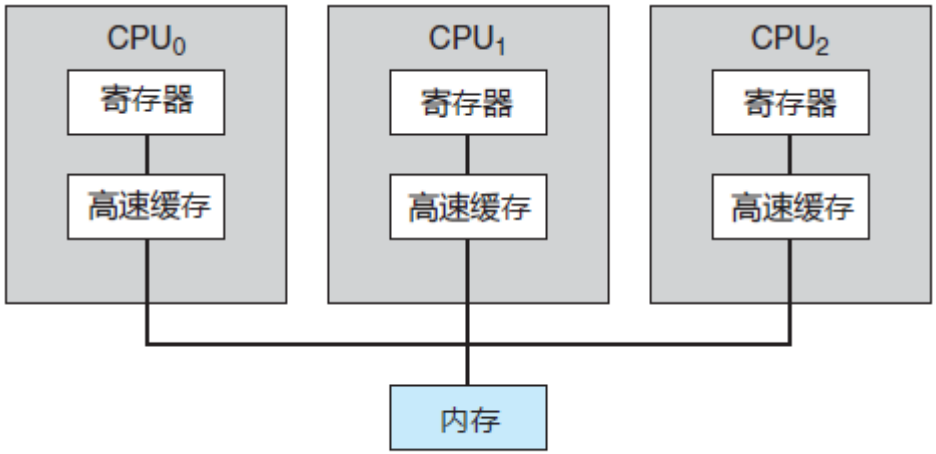


图 1 对称多处理的体系结构

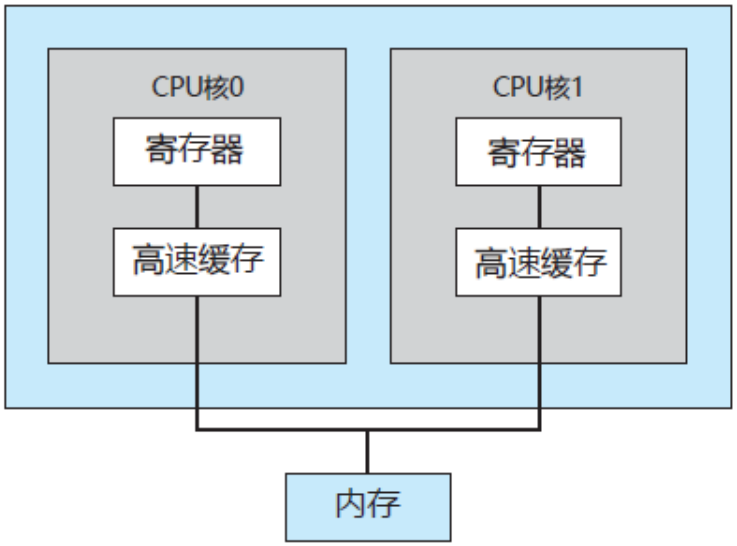
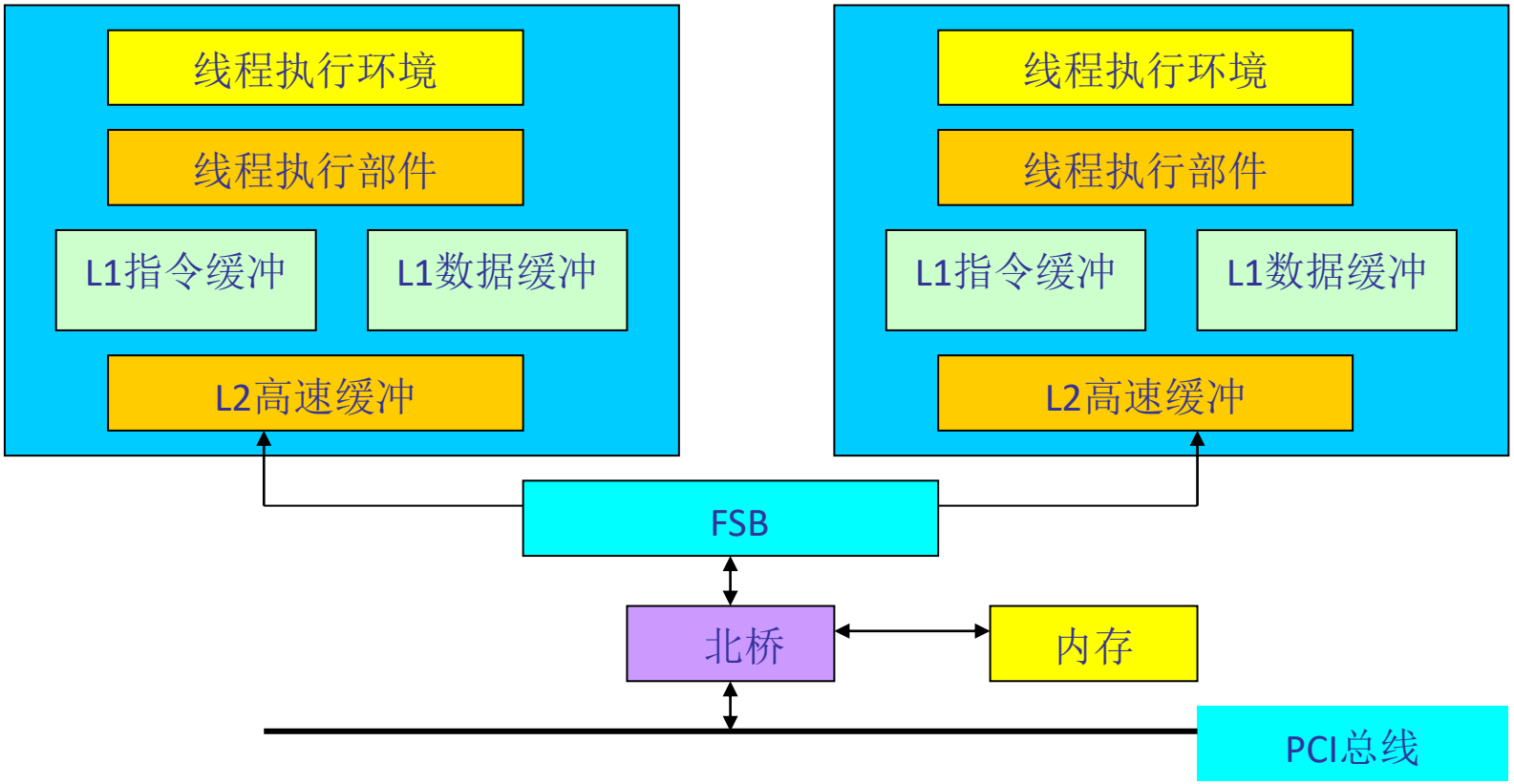


图 2 采用双核芯片设计

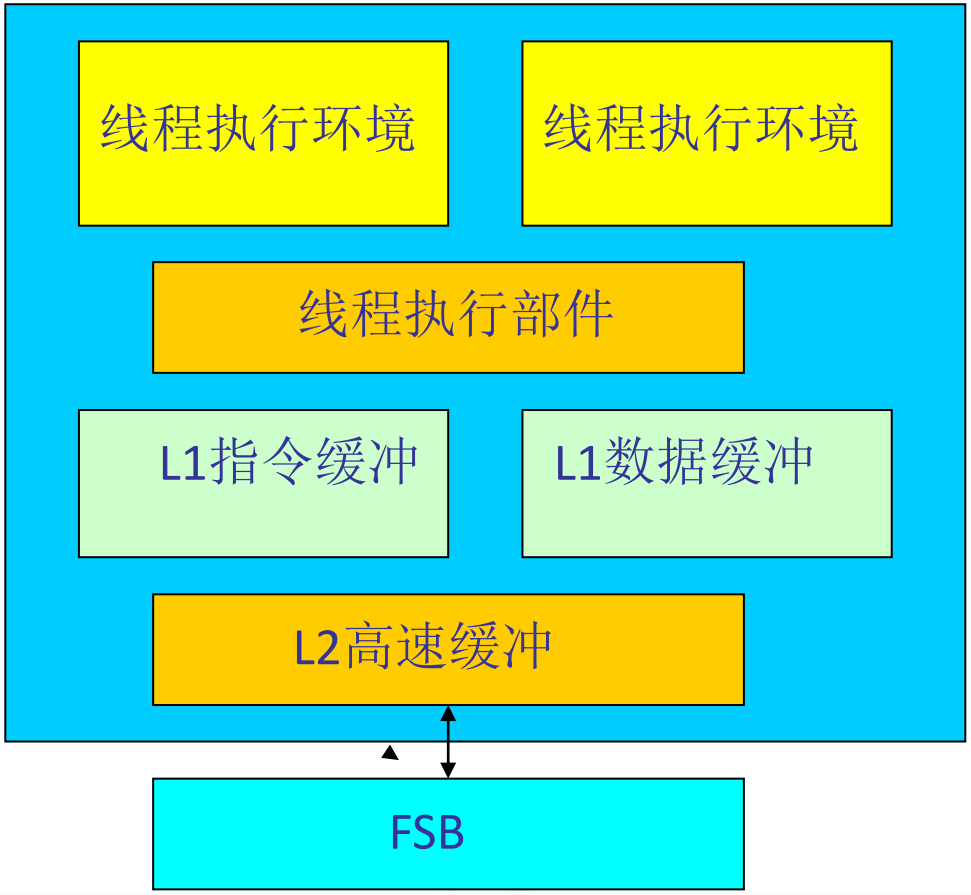
多核处理器与超线程

- (1)多处理器结构



多核处理器与超线程

- (2)超线程结构

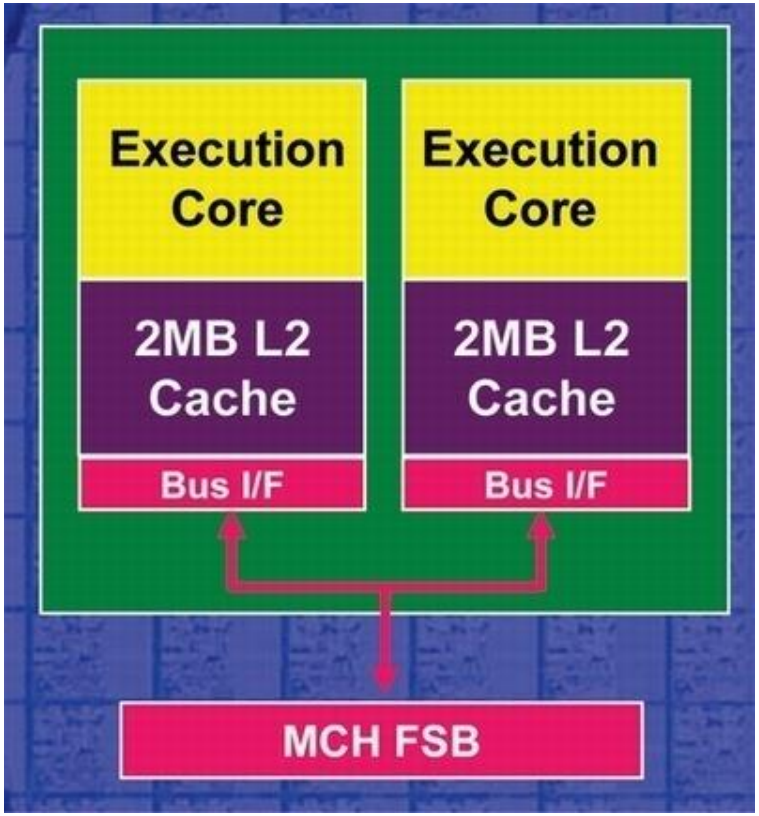


多核处理器与超线程

- 超线程
  - 是在一个CPU上同时执行多个程序，让应用程序使用芯片不同部分。要在处理器上多加入一个逻辑处理单元指针，在单个处理器芯片内部集成多个线程执行环境(包括一套完整的寄存器组，用于保存线程运行时所需的全部状态信息)。线程执行环境共享一套执行部件和所有高速缓存。从宏观上看，单个CPU芯片上存在多个并行的逻辑处理器。

多核处理器与超线程

- (3)多核结构

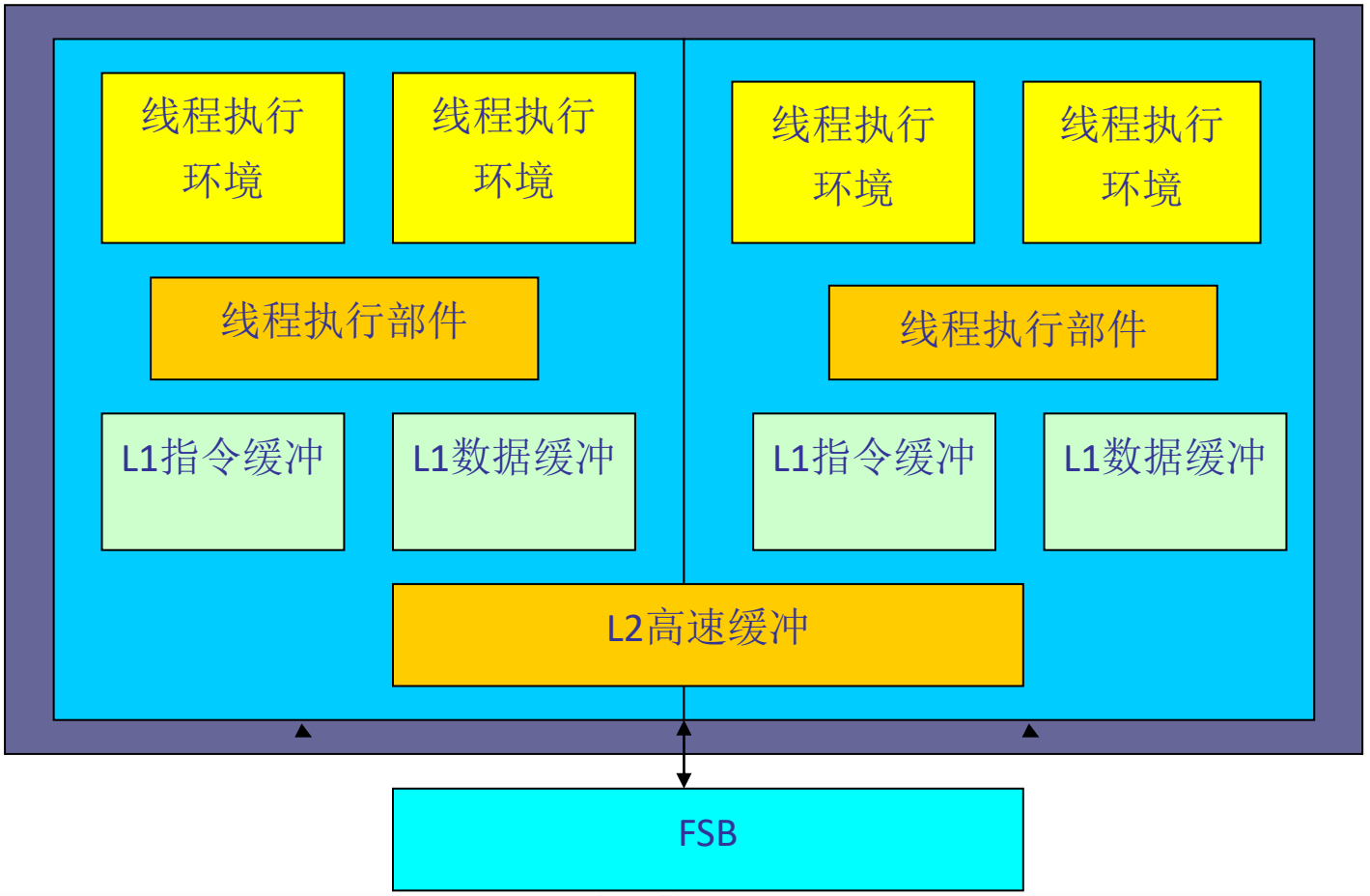


多核处理器与超线程

- 多核技术
  - 是在单个CPU芯片内部集成多套执行核
  - 每个执行核具有一套相对完整的执行部件、寄存器组和一级高速缓存等逻辑单元；各个内核之间还可能会共享二级高速缓存和FSB总线控制器等逻辑单元。

多核处理器与超线程

- (4)多核超线程结构



操作系统与多核处理器

- 操作系统与多核处理器的关系
  - 处理器通信支持
  - 进程/线程数据共享支持
  - 存储器层次及管理
  - 程序并行执行模型支持
  - 同步支持
  - 调度及优化
  - 能耗管理

2.1.2寄存器

- 计算机系统的处理器包括一组寄存器，其个数根据机型的不同而不同，它们构成了一级存储，比主存容量小，但访问速度快。
- 这组寄存器所存储的信息与程序的执行有很大关系，构成了处理器现场。

寄存器

- 按照功能分类：x86结构为例
  - 通用寄存器-- EAX，EBX，ECX和EDX
  - 指针及变址寄存器--ESP，EBP，ESI及EDI
  - 段选择符寄存器--CS、DS、SS、ES 、FS、GS
  - 指令指针寄存器和标志寄存器--EIP、EFLAGS
  - 控制寄存器--CR0，CR1，CR2和CR3
  - 外部设备使用的寄存器

2.1.3 特权指令与非特权指令

机器指令的集合称指令系统

① 数据处理类指令
- 执行算术和逻辑运算

② 转移类指令
- 改变指令的执行序列

③ 数据传送类指令
- 处理器的寄存器间、寄存器与主存单元间、主存单元间数据交换

④ 移位与字符串指令

⑤ I/O类指令

特权指令与非特权指令

- 从资源管理和控制程序执行的角度出发，必须把指令系统中的指令分作两部分：特权指令和非特权指令。

- 特权指令是指只能提供给操作系统的核心程序使用的指令。

特权指令做了一些什么?

- 设置定时器
- 读取时钟
- 清除内存
- 发起陷入指令
- 关中断
- 修改设备状态信息
- 用户与内核态切换
- 访问I/O设备

中央处理器如何判定特权指令是否能够执行呢?

2.1.4 处理器状态

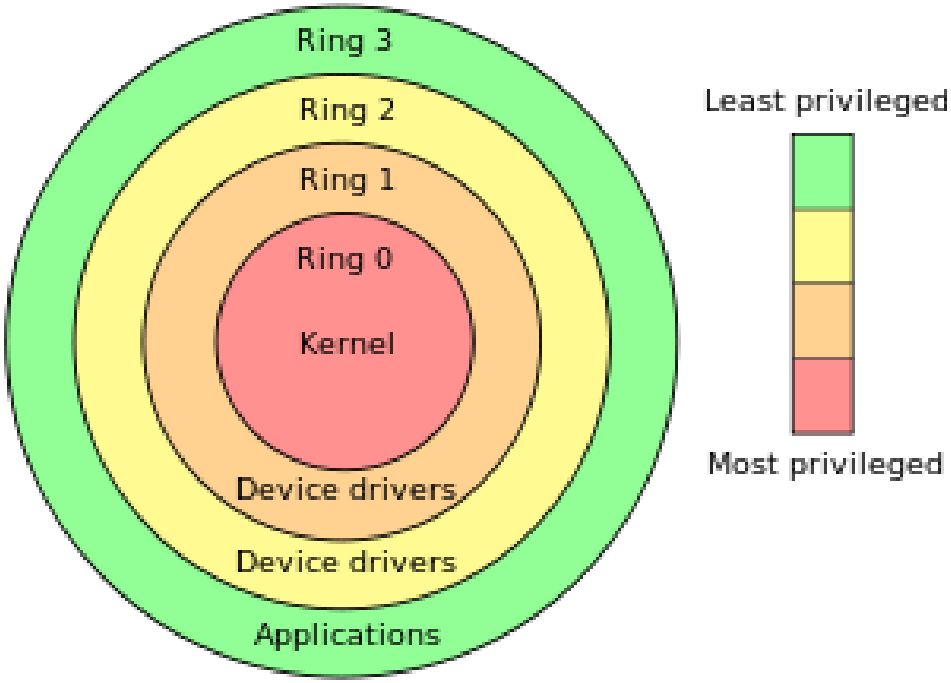
- 处理器状态标志和设置处理器成不同状态:
  - 管理状态（特权状态、系统模式、特态或管态）
  - 用户状态（目标状态、用户模式、常态或目态）
- 处理器处于管理状态时
  - 程序可以执行全部指令，使用所有资源，具有改变处理器状态的能力；
- 处理器处于用户状态时
  - 程序只能执行非特权指令

- 操作系统内核工作在管理态，用户程序工作在用户态
  - 特权指令只允许OS内核执行

- CPU通常用受保护的控制寄存器来区分管理态和用户态

OS保护

- Intel Pentium的处理器状态有四种，支持4个保护级别，0级权限最高，3级权限最低



- 内核态运行在Ring0
- 用户态运行在Ring3

内存保护

- OS必须做到不同进程间的内存隔离
- OS内核占用的内存不能被用户态的进程破坏
- 硬件支持的内存保护
  - 基址与限址寄存器
  - 页表寄存器/页属性/页保护
  - 段描述符/段属性
    - 大小、基址、特权级、代码/数据
  - 通过特权级检查，实现内存的隔离与保护

处理器状态的切换

相互保护隔离了，怎么实现相互联系呢？

- 引起处理器状态切换的原因（用户态→核心态）：
  - 程序请求操作系统服务，执行系统调用
  - 程序运行产生中断或者异常事件，程序被中断，转向中断处理程序或异常处理程序
- 状态切换步骤
  - 保存中断处理器现场
  - 根据中断号设置程序计数器
  - 交换PSW，转向中断处理程序

2.1.5 程序状态字寄存器

- 计算机如何知道当前处于何种工作状态？这时能否执行特权指令？通常操作系统都引入程序状态字PSW（Program Status Word）来区别不同的处理器工作状态
- PSW用来控制指令执行顺序并保留和指示与程序有关的系统状态，**主要作用是实现程序状态的保护和恢复**
- 每个程序都有一个与其执行相关的PSW，每个处理器都设置一个PSW寄存器。程序占有处理器执行，它的PSW将占有PSW寄存器

程序状态字寄存器

- PSW寄存器包括以下内容：
  - 程序基本状态：
    - 程序计数器；
    - 条件码；
    - 处理器状态位。
  - 中断码。保存程序执行时当前发生的中断事件。
  - 中断屏蔽位。指明程序执行中发生中断事件时，是否响应出现的中断事件。

2.1.6 x86总体架构

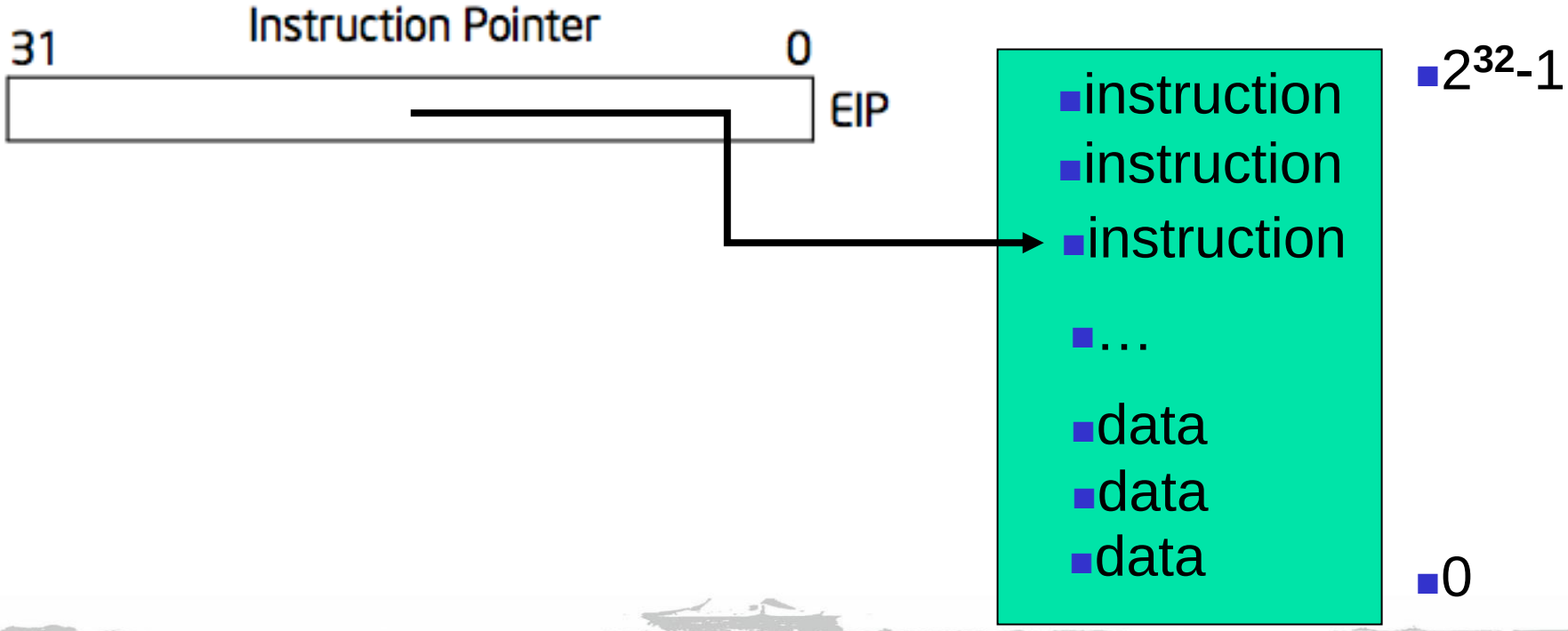
- 通用寄存器
- 8, 16, and 32 bit versions
- 堆栈平衡目的：%EBP, %ESP

General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

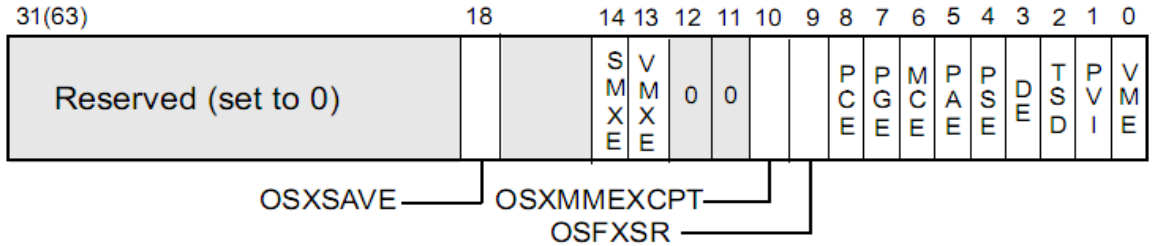
2.1.6 x86总体架构

- **EIP** 执行指令在内存中的地址
- 指令长度不是固定的
- EIP通常可以被 CALL, RET, JMP等控制

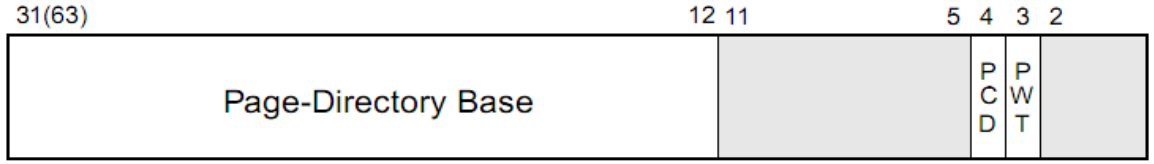


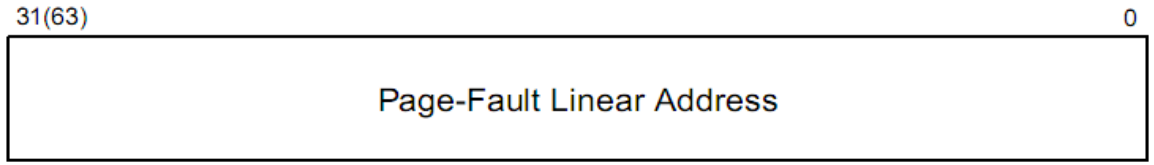
2.1.6 x86总体架构

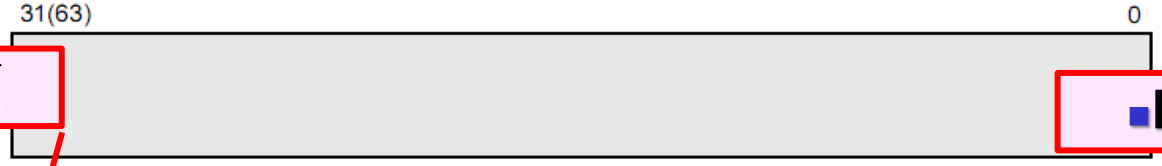
- 控制寄存器



■页目录

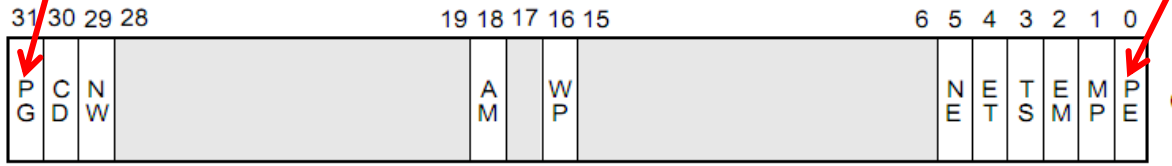






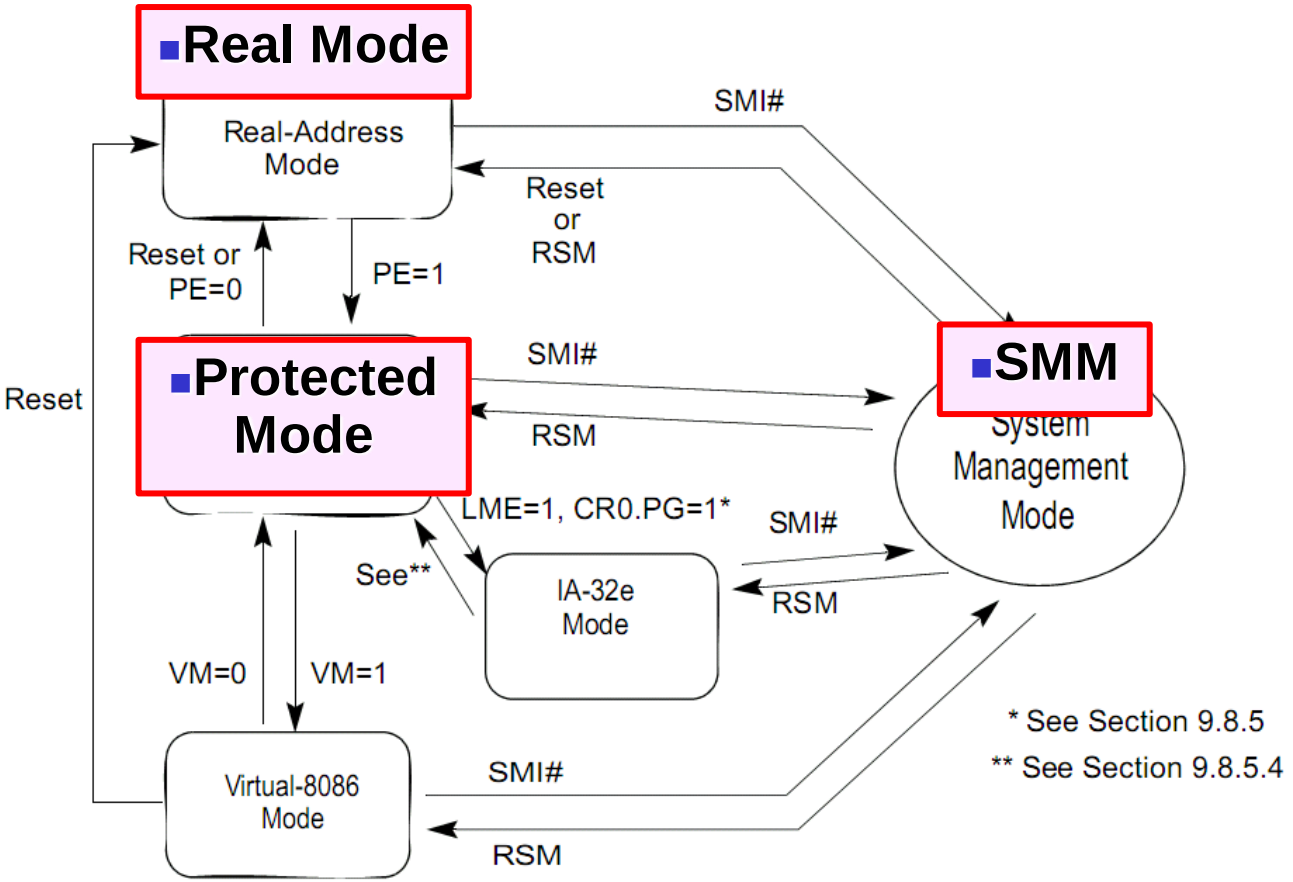
■PG: 分页

■PE: 保护模式



2.1.6 x86总体架构

- 实模式
- 保护模式
- SMM模式



2.1.6 x86总体架构

- 8086: 16-bits 处理器
  - 可寻址空间: 64KB

- 地址总线 20-bits
  - 最大寻址空间: 1MB
  - 16位如何与20位匹配
    - 引入段寄存器，分两次读进地址：段基址+偏移
    - 实模式寻址：physical addr = 16 * segment + offset
    - CS: code segment, for EIP
    - SS: stack segment, for SP and BP
    - DS: data segment for load/store via other registers
    - ES: another data segment, destination for string ops
    - *e.g.* `CS=f000 IP=fff0 => ADDR: ffff0`

2.1.6 x86总体架构

- 80386: 32-bit data and bus addresses
  - 保护模式

- 向前兼容性
  - 8086是16位的处理器+20位的地址总线
  - 为了兼容……

  - 从实模式启动，切换到保护模式
  - 一直沿用至今

2.1.6 x86总体架构

- 与内存管理相关的寄存器
  - GDTR (全局描述符寄存器)
    - 段属性
  - IDTR (中断描述符寄存器)
    - 中断响应程序的地址
    - 保护级别
  - TR (任务寄存器)
    - 与进程相关
    - 用户进程切换时的状态保存地址
    - TSS

2.1.6 x86总体架构

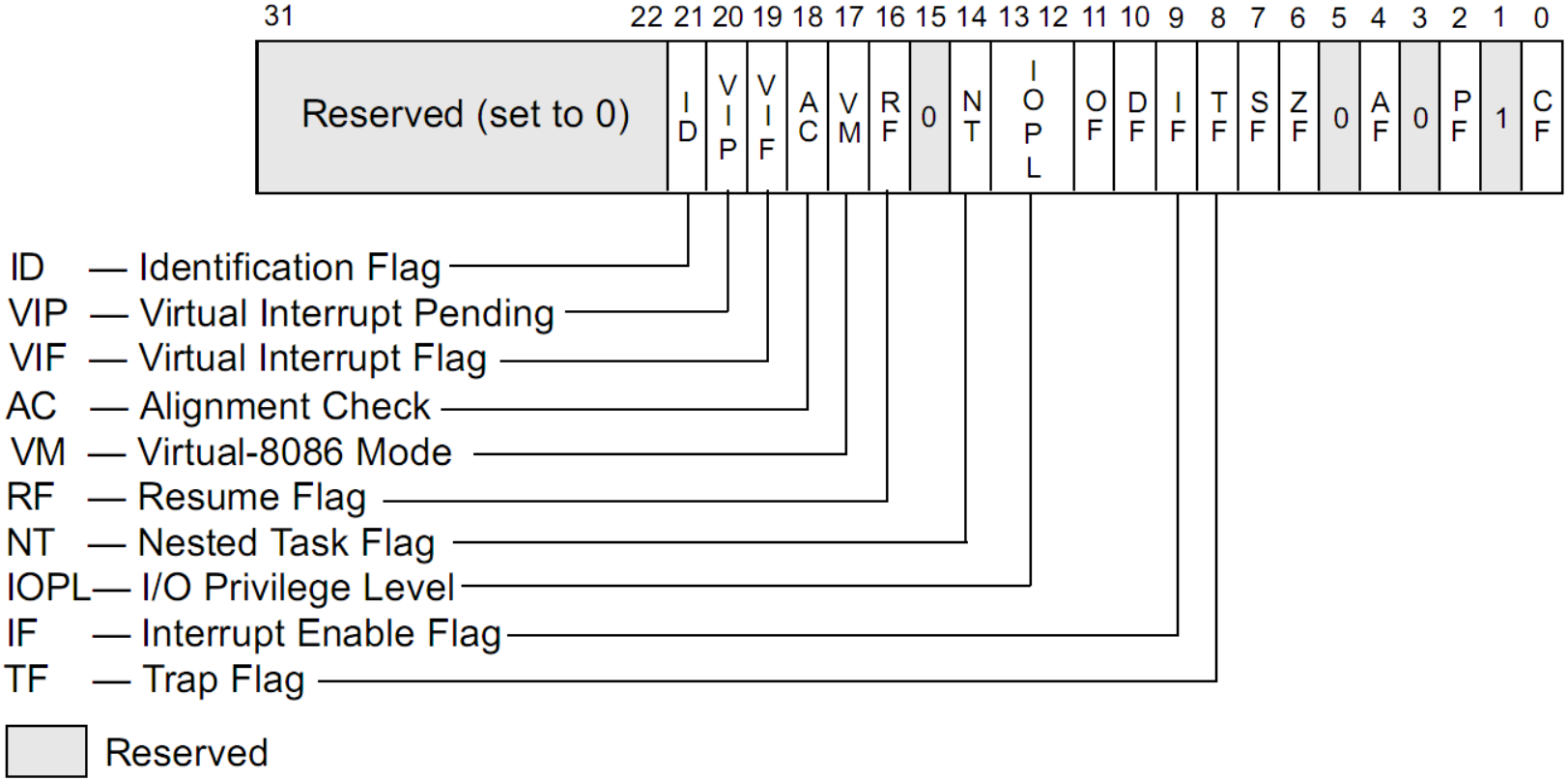
- 程序状态字寄存器PSW包括以下内容：
  - 程序基本状态：
    - 程序计数器；
    - 条件码；
    - 处理器状态位。
  - 中断码。保存程序执行时当前发生的中断事件。
  - 中断屏蔽位。指明程序执行中发生中断事件时，是否响应出现的中断事件。

2.1.6 x86总体架构

- Intel Pentium中，PSW由标志寄存器EFLAGS和指令指针寄存器EIP组成，均为32位。
- EFLAGS的低16位称FLAGS，标志可划分为三组：
  - 状态标志：CF, PF, AF, ZF, OF, SF
  - 控制标志：VM, TF, IF(中断允许）等
  - 系统标志：IOPL（控制I/O访问）

2.1.6 x86总体架构

- System Flags in the EFLAGS



2.2 中断处理

2.2.1中断的概念

2.2.2中断分类

2.2.3中断、异常和系统调用

2.2.4 中断优先级和多重中断

2.2.5 同步与原子性

2.2 中断处理

- OS是一堆功能各异的代码组合，在不同中断（事件）的驱动下，完成硬件资源的操作和管理

- 典型中断
  - 来自硬件设备的处理请求
  - 程序运行产生异常事件
    - 出错
    - 程序请求操作系统服务

- **当事件发生时，处理器通常会立即终止当前任务，触发状态切换**
- OS为每一类事件定义了一组响应程序
  - 中断处理
  - 异常处理

- 状态切换的途径
  - 均通过中断机制发生
  - 中断是用户态到核心态转换的仅有途径

2.2.1 中断的概念

- 中断是指程序执行过程中，当发生某个事件时，中止CPU上现行程序的运行，引出处理该事件的程序执行的过程。

- 中断通常也定义为一个能够改变CPU执行的指令序列的事件；OS是由中断（事件）驱动的

- 在物理上，中断与CPU内外部硬件电路产生的电信号相关

2.2.2中断分类

- 同步中断：指令执行时由CPU控制单元产生的
  - 只有在一条指令执行终止后CPU才会发出中断
- 异步中断：由其它硬件设备依据CPU时钟信号随机产生的

注：

- 中断的分类与处理器的体系结构有关、IBM中大型机、Intel微处理器等手册中对中断的分类均不相同；
- 从中断的产生及处理机理上，大致相同；
- 在Intel 处理器手册中，同步中断称为异常，异步中断称为中断
- 也有时候，中断和异常会统称为中断

中断分类

- 异步中断（中断）由定时器和I/O设备产生
- 同步中断（异常）由程序错误产生，或者由内核必须处理的异常条件产生
  - 更细粒度的分类
    - 异常：由程序错误产生的异常
    - 系统调用：请求系统服务

中断、异常、系统调用三者有什么区别和联系？

2.2.3 中断、异常和系统调用

- 中断和异常希望解决的问题
  - 当外设连接计算机时，会出现什么现象？（中断）
  - 当应用程序出现非预期行为时，会出现什么现象？（异常）
- 系统调用希望解决的问题
  - 用户应用程序是如何得到系统服务？

中断、异常和系统调用

- 产生原因不同
  - 中断：来自硬件设备的处理请求
  - 异常
    - 非法指令或其它意想不到的原因导致当前指令执行失效
    - 如内存错误、除零错误等的处理请求
  - 系统调用
    - 应用程序主动地向操作系统发生的服务请求
    - 是正在运行的程序所期待的事件，是由执行“访管指令”所引起的。

中断、异常和系统调用

- 中断：
  - 由与现行指令无关的中断信号触发的(异步的)，
  - 中断的发生与CPU处在用户模式或内核模式无关（系统不能确定中断发生的时间）
  - 一般来说，中断处理程序提供的服务不是为当前进程所需的；
- 异常（包括系统调用）：
  - 由处理器正在执行现行指令而引起的，
  - 异常处理程序提供的服务是为当前进程所用的。

中断、异常和系统调用

- 尽管中断、异常和系统调用产生的原因并不相同，但是在实现机制上，系统调用通常通过一种特殊的异常来实现；同时，异常与中断通常又都是通过相同的中断机制来实现

异常的分类

- Linux中异常
  - 处理器可探测的异常
    - 出错fault：一般性错误，可以修复
    - 陷入trap：用于调试
    - 停止abort：致命的、不可恢复错误
  - 编程异常programmed exception
    - 系统调用，进程自愿进入核心
- 一般分类：Fault、Trap、Abort

出错（Fault）和陷入(Trap)的区别

- 原因不同：
  - Fault是错误引起的，它可能能被故障处理程序修正。如缺页异常是一种fault
  - Trap是有意的异常，是执行一条指令的结果。可以应用在调试中。最主要的用途是在应用程序和内核间提供一个接口，即系统调用。
- 返回行为不同：
  - Fault：发生时保存指向触发异常的那条指令，返回时重新执行那条指令
  - Trap：发生时保存指向触发异常的那条指令的下一条指令，返回到下一条指令执行

异常及异常处理

- (1) 由硬件检测并处理的异常
  - 典型错误
    - Page Fault
    - 不合法的内存访问
    - 除0
- (2) 硬件自身出错产生的异常
  - 通常在检测错误后会重新执行该指令
  - PC、regs, mode等运行状态会由CPU保存

异常及异常处理

- (3) 有些异常的处理是修复异常，并返回上下文重新执行该指令
  - 比如Page Fault
    - OS会调用页面置换程序，修复页面错误
    - 恢复上下文，重新执行出错的指令

- (4) 有些异常是由OS通知相应进程
  - 比如Unix 系统中的信号
  - 直接“杀死”相应进程

系统调用及其处理

- 系统调用时提供给用户态程序执行特权操作的接口

- 系统调用的响应
  - 触发事件，切换至内核态
  - 传递系统调用的参数
  - 保存状态以便完成系统调用后恢复执行

  - 形如：Int 0x80/sysenter

系统调用及其处理

- 系统调用的参数传递
  - 系统调用号，保存在eax中
  - 其它的参数传递
    - ebx, ecx, edx, esi, edi, ebp

中断及中断处理

- 中断（异步中断）
  - 定时器
  - I/O 硬件中断装置

定时器

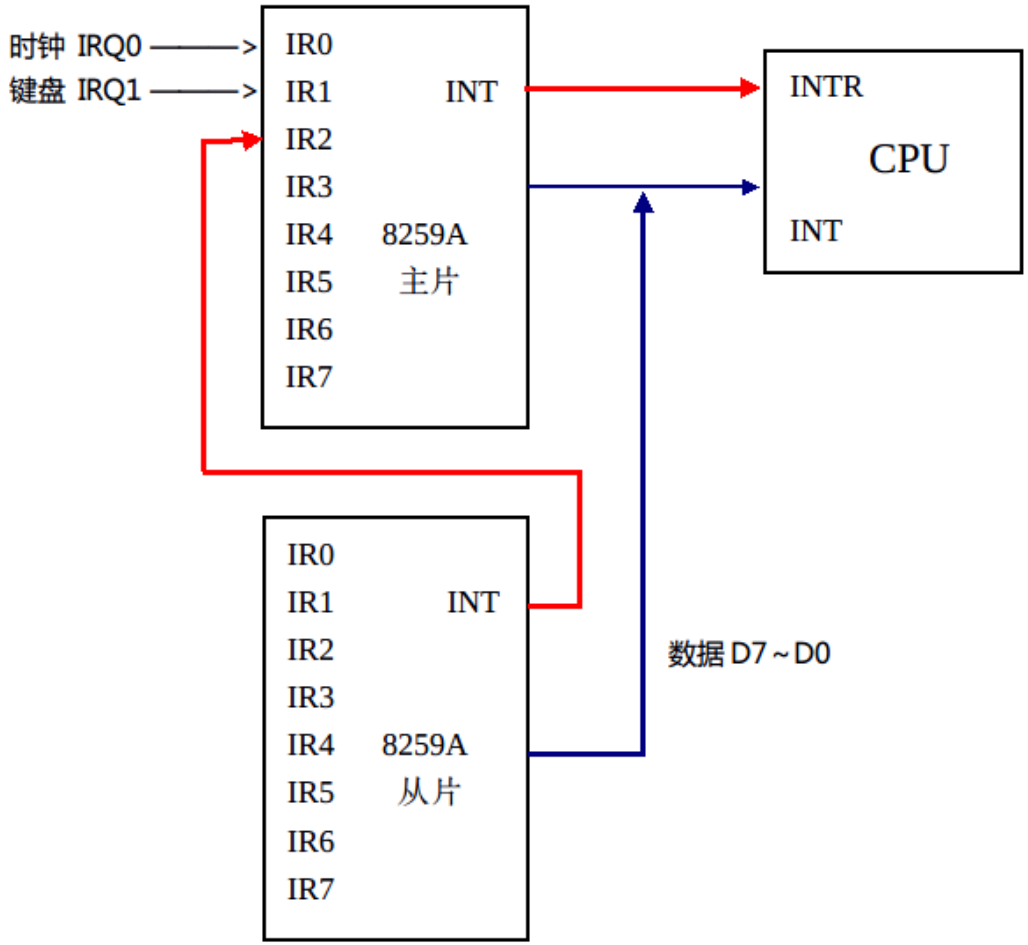
- OS回收控制的重要方式
  - 定时器会定时产生中断
  - 设定定时器是特权操作
  - 分时OS的调度
- 能够避免死循环
  - 程序出错或恶意程序的非法独占CPU
- 在时间相关的函数中被广泛使用 (e.g., sleep())

中断装置

- 发现中断源并产生中断的硬件称中断装置
- 所有计算机系统都采用硬件和软件结合的方法实现中断处理
- 中断装置主要做以下三件事：
  - 发现中断源：当发现多个中断源，按照中断优先级别来先后响应
  - 保护现场：暂停当前程序运行，将中断点的PSW保存至核心栈，使得处理程序可运行
  - 启动处理中断事件的程序

中断装置

- 可编程中断控制器8259A



中断装置

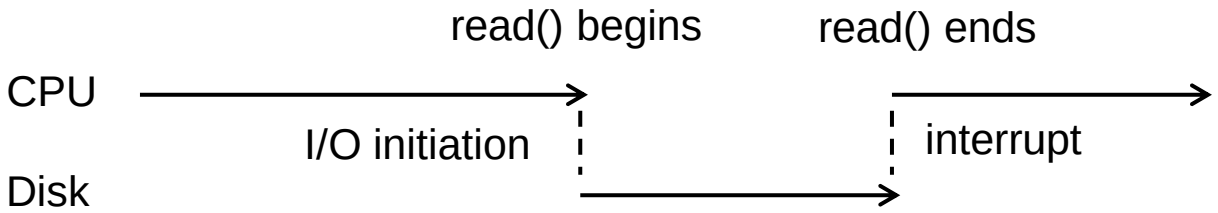
- CPU响应中断的大致过程
  - 检测到中断或异常后，自动保存CPU状态
    - 依据TR(Task Register)，定位当前进程的内存空间
    - 将CPU状态保存在相应内存地址
  - 依据IDTR寄存器，定位到IDT内存地址
  - 依据中断向量号在IDT中检索对应的中断处理程序
  - 跳转至中断处理程序

中断处理程序

- 处理中断事件的程序称为中断处理程序。它的主要任务是处理中断事件和恢复正常操作
- 不同中断源对应不同中断处理程序，故**快速找到中断处理程序的入口地址**是一个关键问题
- 中断处理程序主要做四项工作：
  - 保护未被硬件保护的一些必需的处理状态
    - CPU自动保存的只有EIP/EFLAGS/ESP/SS/CS
  - 识别各个中断源，分析产生中断的原因
  - 处理发生的中断事件
  - 恢复正常操作

I/O中断及其处理

- 中断是异步I/O的基础
  - OS 启动I/O
  - I/O设备开始工作
  - 完成后通过中断信号通知CPU
  - OS通过中断向量表来处理中断



小结：中断和异常

	类别	原因	同步/异步	返回行为
中断	中断（interrupt）	来自I/O设备或其他硬件部件	异步	总是返回到下一条指令
异常	陷入(trap)	有意识安排的	同步	返回到下一条指令
	故障（fault）	可恢复的错误	同步	返回到当前指令
	终止（abort）	不可恢复的错误	同步	不会返回

2.2.4 中断优先级和多重中断

1. 中断的优先级
2. 中断的屏蔽
3. 多重中断事件的处理

中断优先级

- 计算机执行的每一瞬间，可能有几个中断事件同时发生，中断装置如何来响应同时发生的中断呢?
  - 它按照预定顺序来响应，这个预定顺序称中断的优先级，首先响应优先级高的中断事件。

中断的屏蔽

- 主机可允许或禁止某类中断的响应，如允许或禁止所有的I/O中断、外部中断、及某些程序性中断。
- 有些中断是不能被禁止的，例如，计算机中的自愿性访管中断就不能被禁止。

多重中断事件的处理

- 中断正在进行处理期间，这时CPU又响应了新的中断事件，于是暂时停止正在运行的中断处理程序，转去执行新的中断处理程序，这就叫多重中断（又称中断嵌套）。
- 中断可以嵌套，异常中可以发生中断，但中断中不可以发生异常

2.2.5 同步与原子性

- 中断随时发生，并会打断当前的任务执行
- OS必须保证中断前后的同步
- 必须保证现场保护的原子性
  - 关中断以保证指令序列的原子性
  - 原子指令，如读/写内存

2.3 操作系统的硬件支持

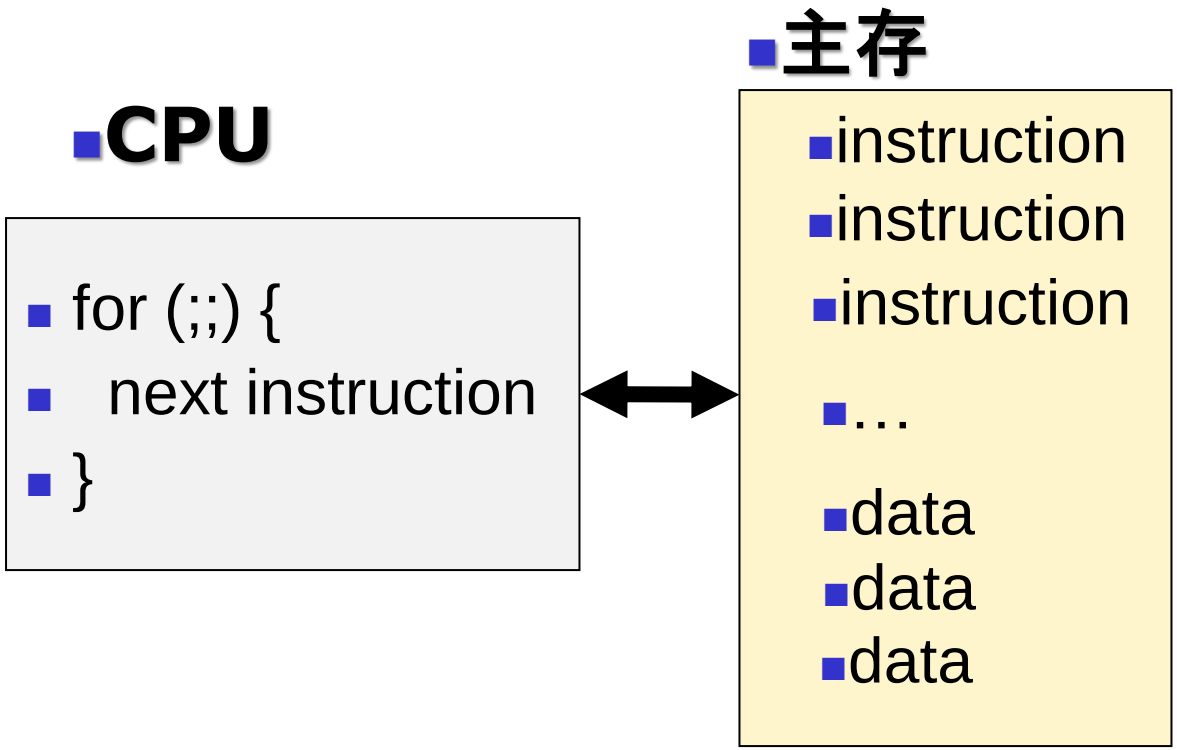
- OS能获得的典型硬件支持
  - 保护：保护硬件资源的受限访问
    - 双模式（内核态和用户态）
    - CPU提供的特权指令和非特权指令
    - 内存保护（硬件支持的分段和分页）
  - 支持并响应计算机系统的各类软硬件事件，如设备变化、程序的出错等异常
    - 中断和异常机制
    - 系统调用
    - 定时器
  - 支持多任务的并发与调度
    - 中断发生时的状态切换
    - I/O的控制
    - 同步与互斥

2.3 操作系统的硬件支持

OS提供的服务	硬件特征
硬件资源保护和操作系统保护	操作系统的双模式 CPU的标记位 特权指令/非特权指令
并发时的中断	中断向量
系统调用	可编程异常指令
I/O设备管理	I/O中断
任务调度	定时器
同步	原子指令

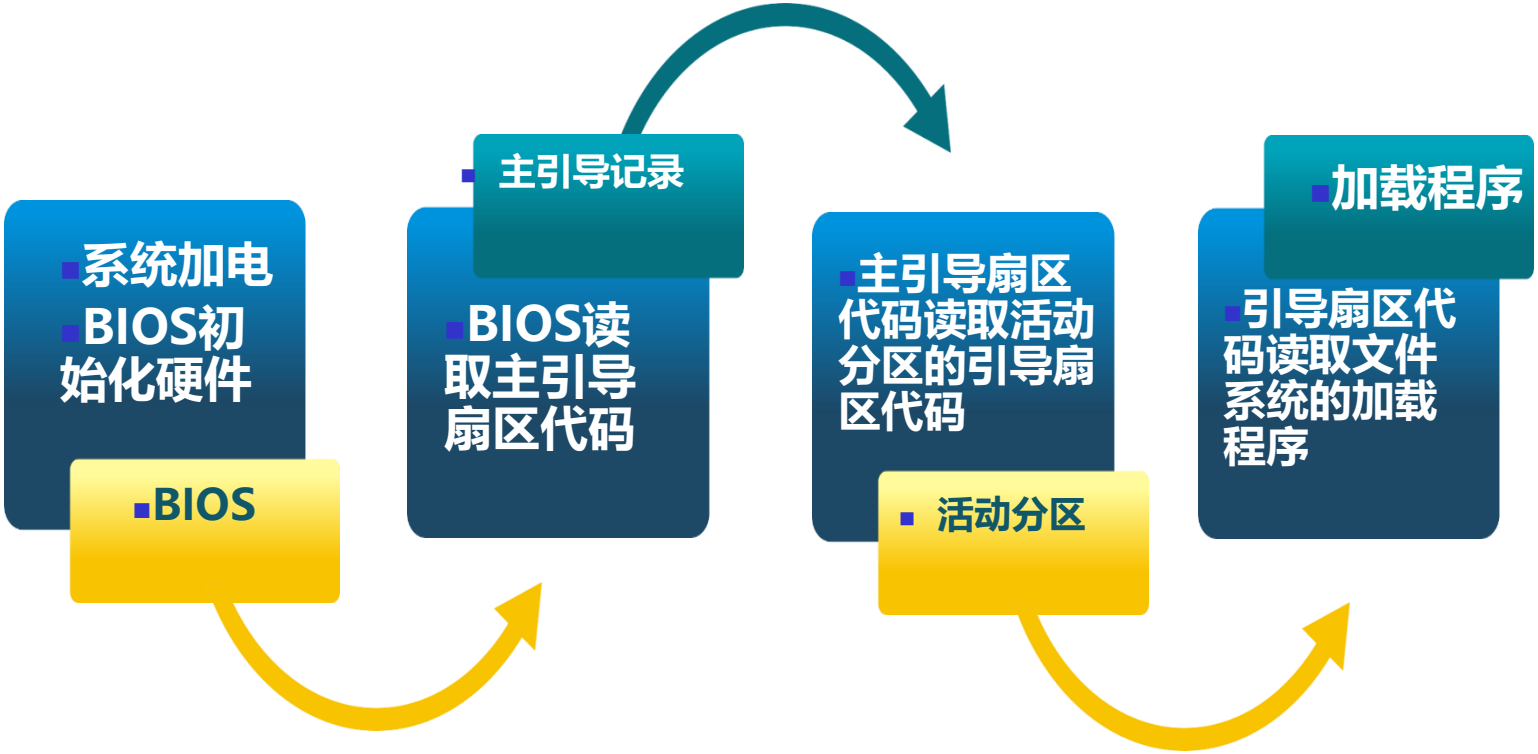
2.4 计算机的启动

- **CPU** 解释并执行指令
- 内存同时保存指令和数据

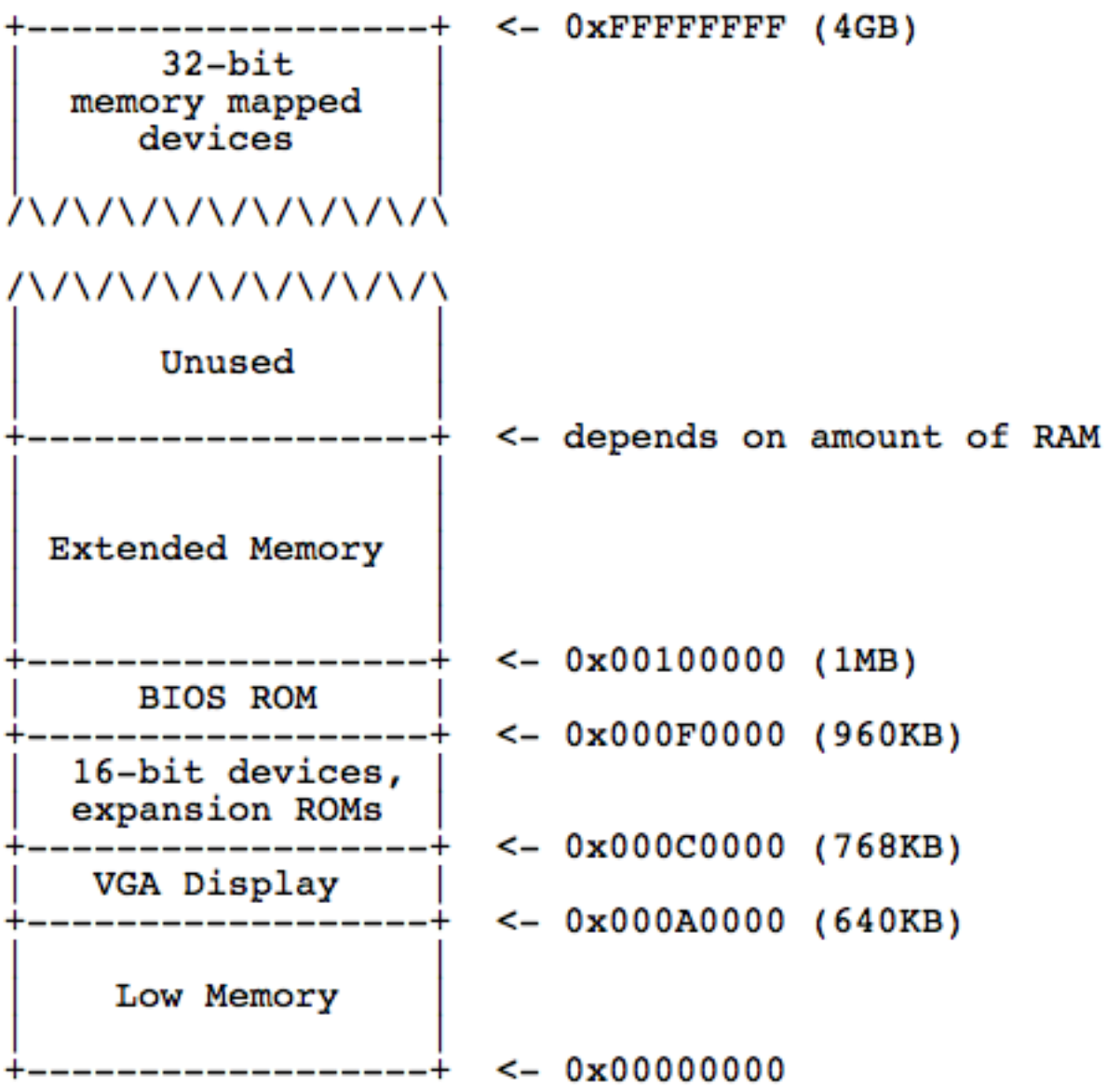


- 计算机是如何将硬件交于操作系统管理的?

2.4 计算机的启动

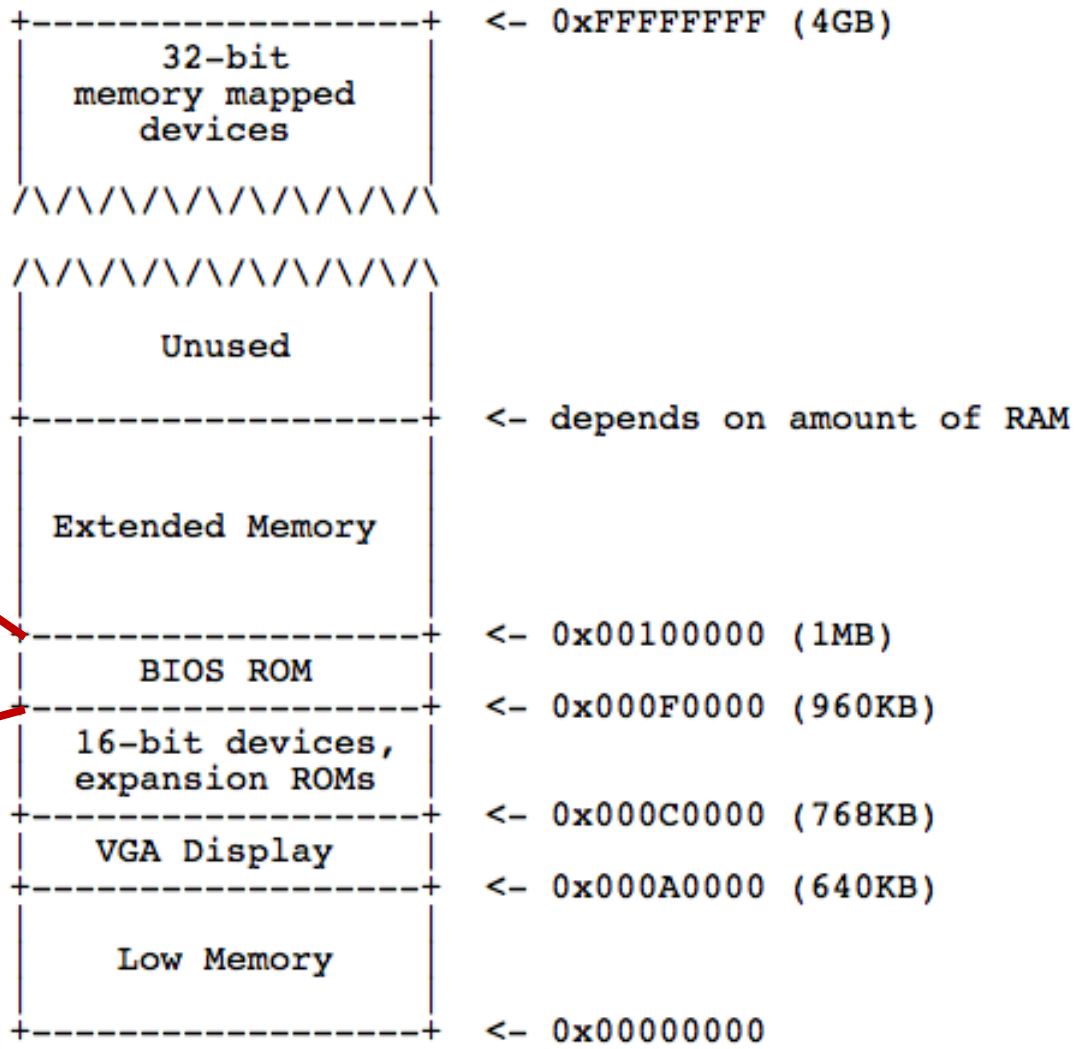

系统加电
BIOS初始化硬件
BIOS
主引导记录
BIOS读取主引导扇区代码
主引导扇区代码读取活动分区的引导扇区代码
活动分区
加载程序
引导扇区代码读取文件系统的加载程序

内存布局


0xFFFFFFFF (4GB)
32-bit
memory mapped
devices
Unused
depends on amount of RAM
Extended Memory
0x00100000 (1MB)
BIOS ROM
0x000F0000 (960KB)
16-bit devices,
expansion ROMs
0x000C0000 (768KB)
VGA Display
0x000A0000 (640KB)
Low Memory
0x00000000

启动时计算机内存和磁盘布局

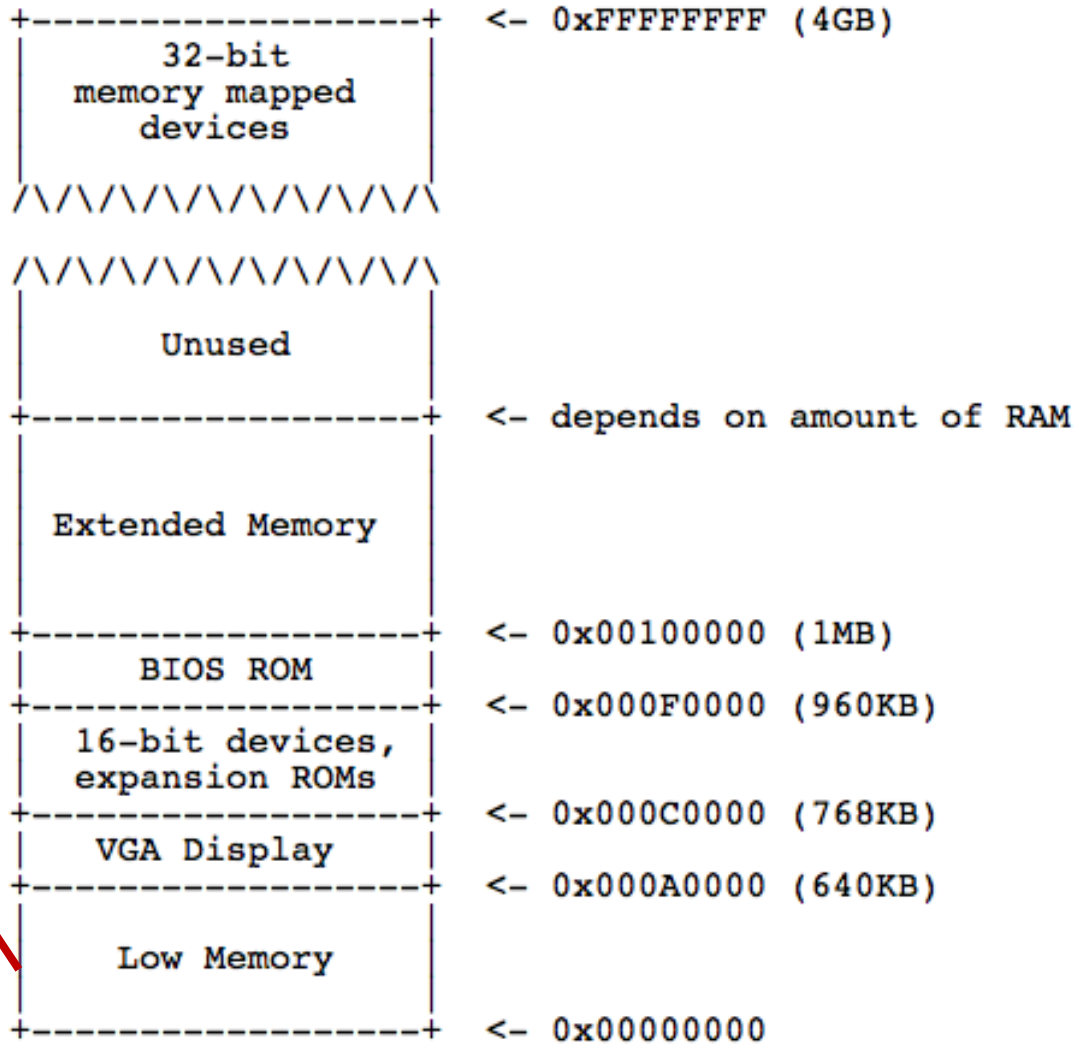
- CS: 0xF000, IP: 0xFFF0
- （CS：代码段寄存器；IP：指令指针寄存器）
- 系统处于实模式
- PC = 16*CS+IP
- EIP：0xFFFF0，指向最后的16个字节
- 20位地址空间：1MB



- 基本输入输出的程序
- 系统设置信息
- 开机后自检程序
- 系统自启动程序等

2.4 计算机的启动

- 主引导扇区：512字节，以”55AA”结尾
- BIOS自检完成后，将引导扇区的程序(bootloader)从主引导扇区加载到**0x7c00**
- 跳转到0x7c00 处



2.4 计算机的启动

- Bootloader完成系统初始化
- OS内核的加载

2.4 计算机的启动

- 系统启动规范
  - BIOS
    - 固化到计算机主板上的程序
    - 包括系统设置、自检程序和系统自启动程序
  - UEFI
    - 传统BIOS弊端：赶不上硬件发展（如不支持2TB硬盘）
    - 统一可扩展固件接口
    - 在所有平台上一致的操作系统启动服务