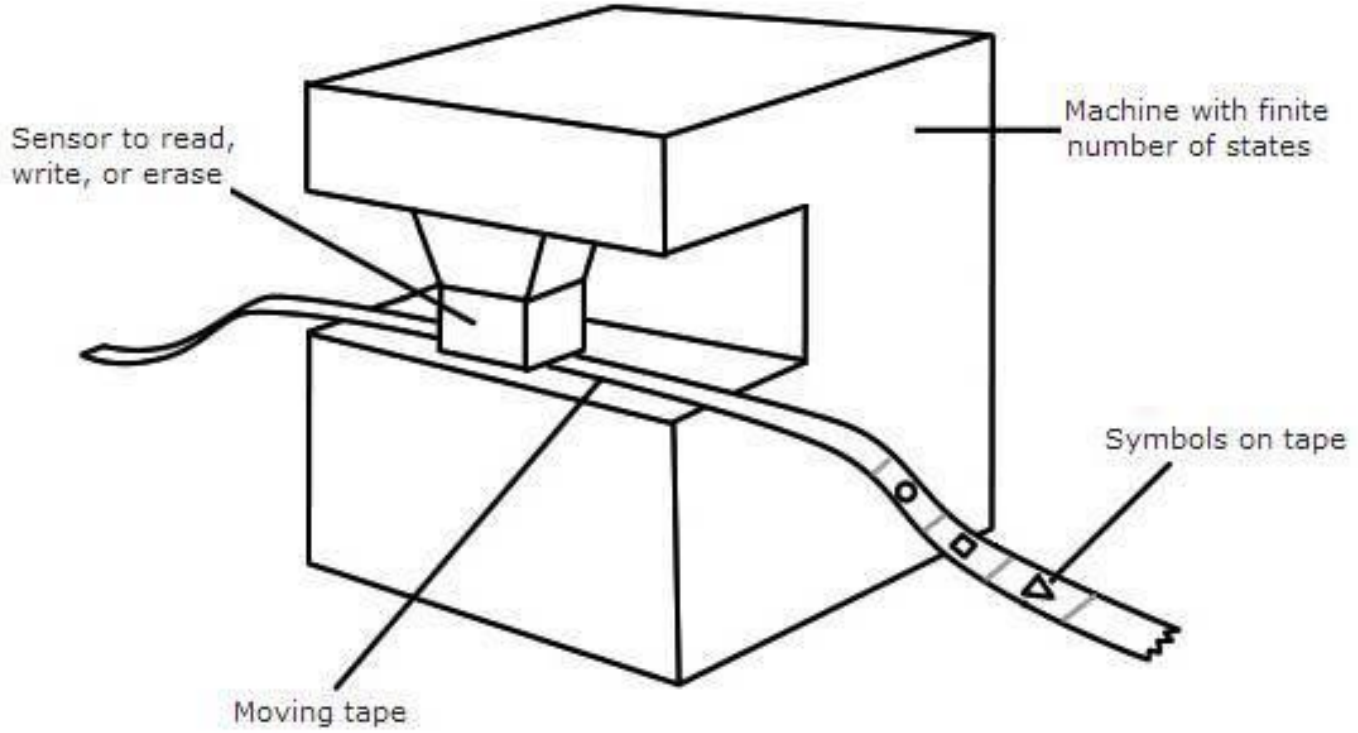
2.1 内存寻址概述

西安邮电大学

图灵机-计算力无限的理想机器



A Turing Machine

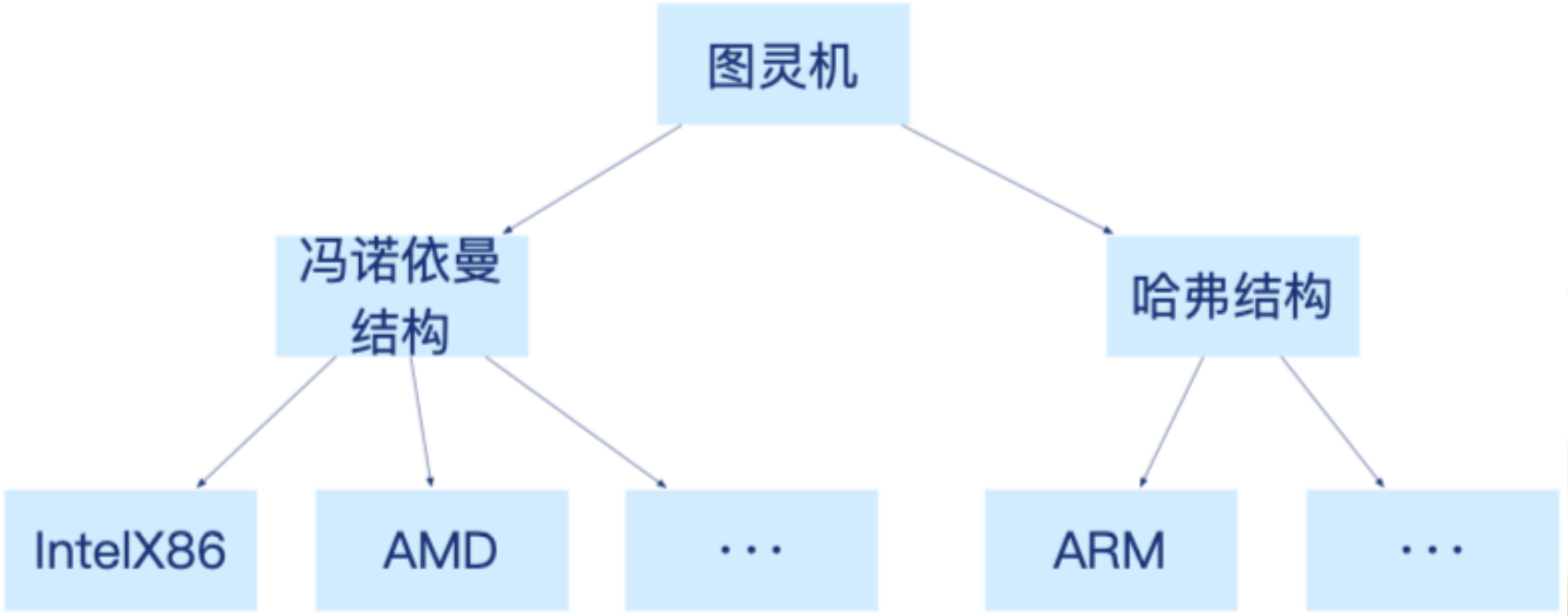
图灵机-计算力无限的理想机器

为什么首先讲内存寻址？首先从图灵机说起，图灵机是一种通用自动机器模型。
它的理念是，有一个二端无限沿伸的纸带作为存储装置，输入，输出和状态转移函数是一个机器的三要素，这三要素组合并变形可成为一切机器的原型，可解决一切图灵机能解决的问题。

冯·诺依曼体系结构

图灵机听起来是纸上谈兵，但它却是当代冯.诺依曼计算机体系的理论鼻祖。它带来的“数据连续存储和选择读取思想”是目前我们使用的几乎所有机器运行背后的灵魂。计算机体系结构中的核心问题之一就是如何有效地进行内存寻址，因为所有运算的前提都是先要从内存中取得数据，所以内存寻址技术从某种程度上代表了计算机技术。

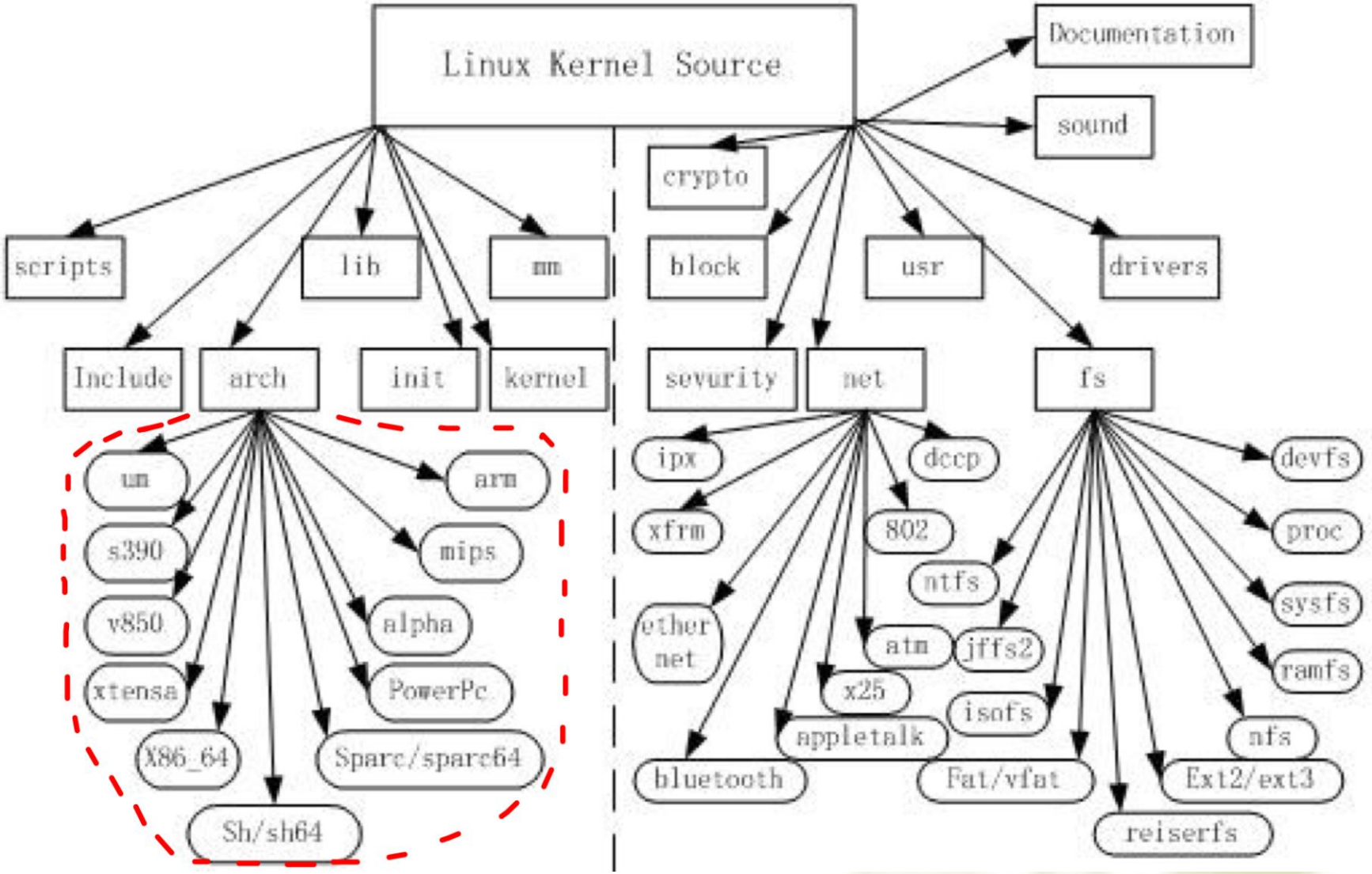
图灵机与冯·诺依曼体系结构



我们说图灵机是冯.诺依曼计算机体系的鼻祖，

而目前的大多数CPU体系结构的鼻祖又是**冯·诺依曼体系。**

Linux内核的可移植性设计



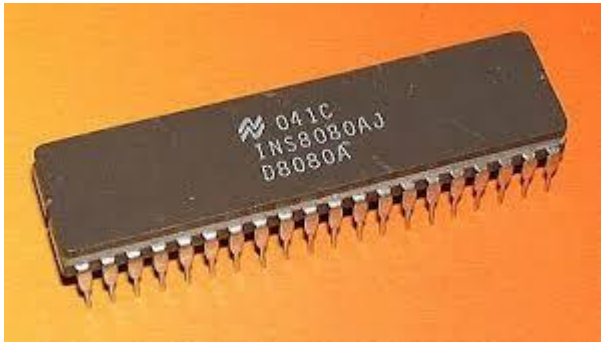
冯·诺依曼体系结构

- Linux在内核设计中，目前几乎支持所有主流的CPU架构，其设计理念遵循了分离体系结构相关代码的原则，在上图Linux支持的众多的CPU体系结构中，与体系结构相关的代码在专门的arch目录下，大家最熟悉的就是X86。因此，我们所介绍的内存寻址也是以此为背景，而且是以32位寻址为主。

X86内存寻址的不同时期

- 石器时期—8位
- 青铜时期—16位
- 白银时期—24位
- 黄金时期—32位，64位

石器时期—8位寻址



- 在微处理器的历史上，第一款微处理器芯片4004是由Intel推出的，只有4位；
- 在4004之后，intel推出了一款8位处理器叫8080；
- 那时没有段的概念，访问内存都要通过绝对地址，因此程序中的地址必须进行硬编码（也就是给出具体地址），而且也难以重定位。

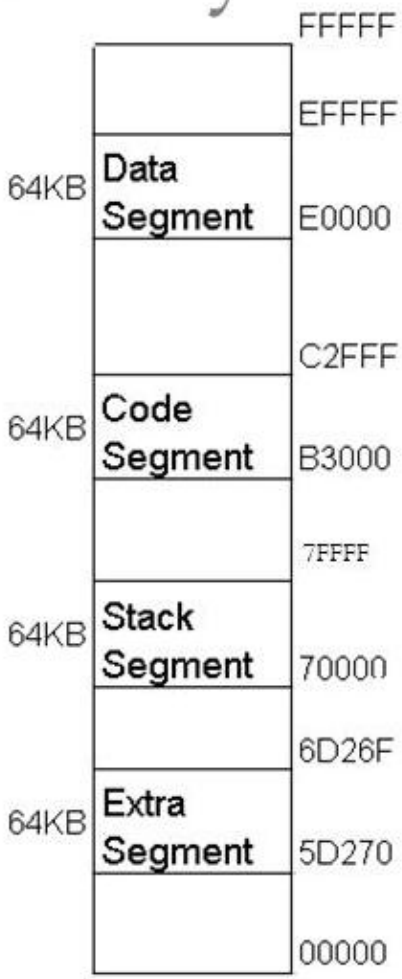
青铜时期—“段”的引入

Segmented Memory

Within the 1 MB of memory, the 8086 defines 4 64KB memory blocks.

DS: E000	CS: B300
SS: 7000	ES: 5D27

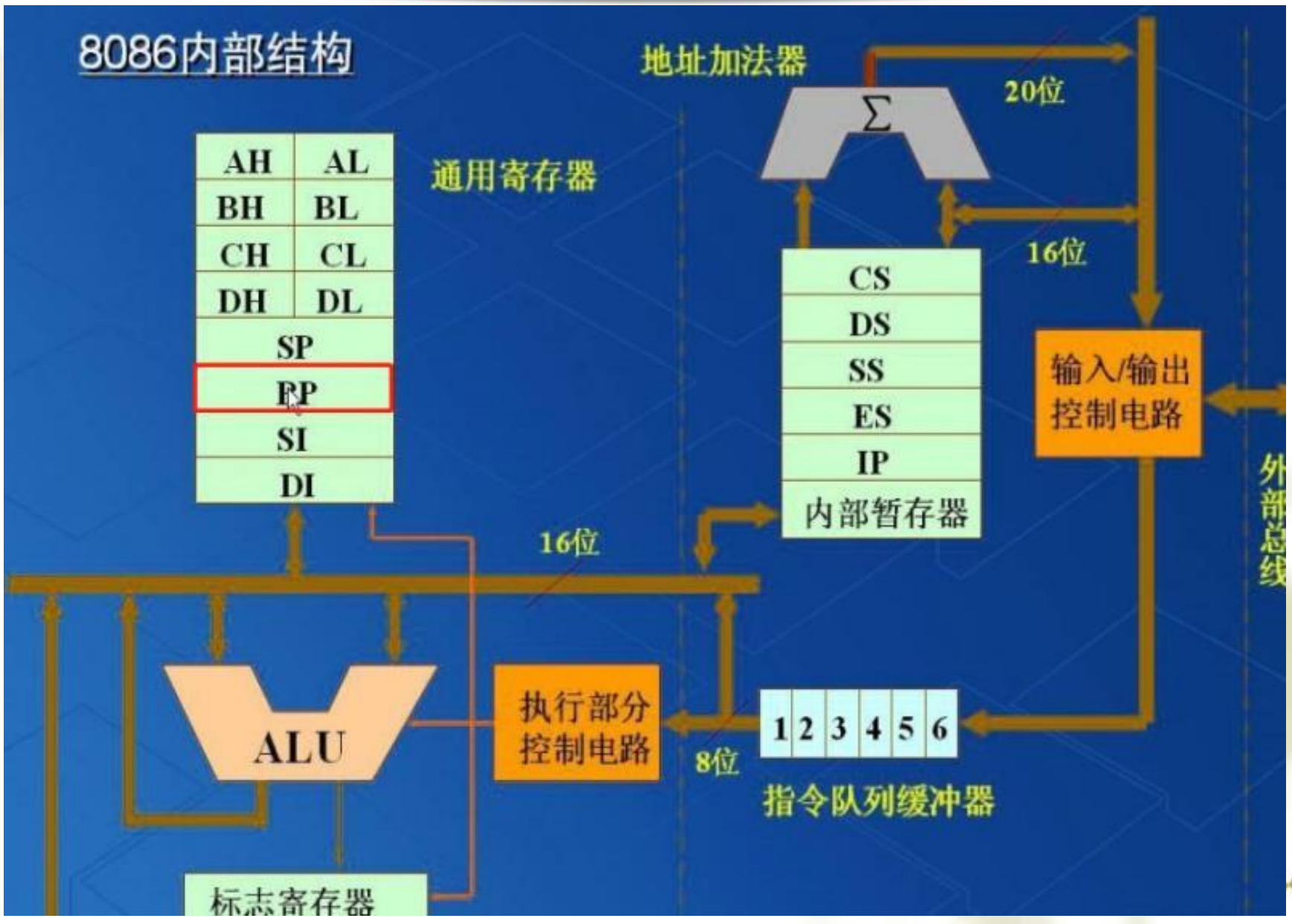
The segment registers point to location 0 of each segment. (The base address)



青铜时期—“段”的引入

- 8086处理器的目标定位1M，于是它的地址总线扩展到了20位，可是数据总线只有16位。也就是把1M大的空间分成数个64k的段来管理（化整为零了）。
- 段描述了一块有限的内存区域，区域的起始位置存在专门的寄存器（也就是段寄存器中）。

X86的寻址方式


8086内部结构
地址加法器
Σ
20位
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
通用寄存器
CS
DS
SS
ES
IP
内部暂存器
16位
输入/输出
控制电路
外部总线
16位
ALU
执行部分
控制电路
1 2 3 4 5 6
8位
指令队列缓冲器
标志寄存器

X86的寻址方式

- 把16位的段地址左移动4位后，再与16位的偏移量相加便可获得一个20位的内存地址，也就实现了从16位内存地址到20位实际地址的转换，或者叫“映射”，这种模式也叫“实模式”。

白银时期—保护模式的引入

- 286地址总线位数增加到了24位 。
- 从此开始引进了一个全新理念—保护模式
- 访问内存时不能直接从段寄存器中获得段的起始地址了，而需要经过额外转换和检查 。

黄金时期—内存寻址的飞跃

- 386是一个32位的CPU，其寻址能力达到4GB
- Intel选择了在段寄存器的基础上构筑保护模式，并且保留段寄存器16位，在保护模式下，它的段范围不再受限于64K，可以达到4G这真正解放了软件工程师，他们不必再费尽心思去压缩程序规模，软件功能也因此迅速提升
- 从80386以后，从32位到目前的64位，Intel的CPU经历了各种型号，但基本上属于同一种系统结构的改进与加强，而无本质的变化，所以我们把80386以后的处理器统称为x86

实模式和保护模式寄存器对比

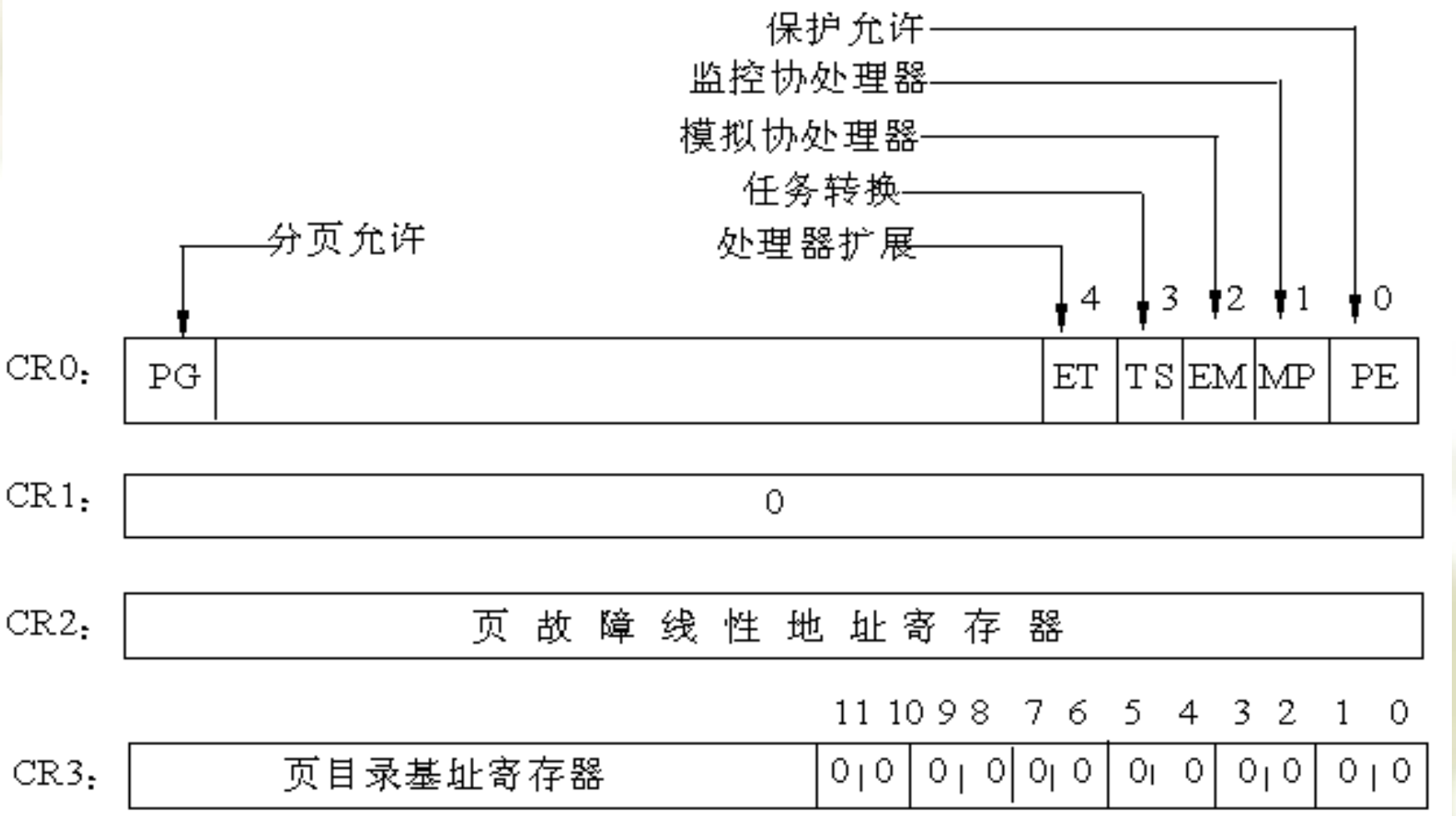
表 1-18086 和 86386 的寄存器的对比

	8086 的寄存器	80386 的寄存器
通用寄存器	AX、BX、CX、DX、SP、BP、DI、SI	EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP
段寄存器	CS、DS、SS、ES	CS、DS、SS、ES、FS、GS
段描述符寄存器	无	对程序员不可见
状态和控制寄存器	FLAGS 、IP	EFLAGS、EIP、CR0、CR1、CR2、CR3
系统地址寄存器	无	GDTR、IDTR、TR、LDTR
调试寄存器	无	DR0--DR7
测试寄存器	无	TR0--TR7

实模式和保护模式寄存器对比

- 保护模式下的寄存器有很大的变化，一些寄存器是专门属于操作系统使用的，比如用于分页的控制寄存器CR0~CR3，一般用户不能使用，一些寄存器是系统地址寄存器，还有7个调试寄存器和7个测试寄存器，都是保护模式所特有的。

保护模式下的页表寄存器



控制寄存器

PG	PE	方式
0	0	实模式，8080操作
0	1	保护模式，但不允许分页
1	0	出错
1	1	允许分页的保护模式

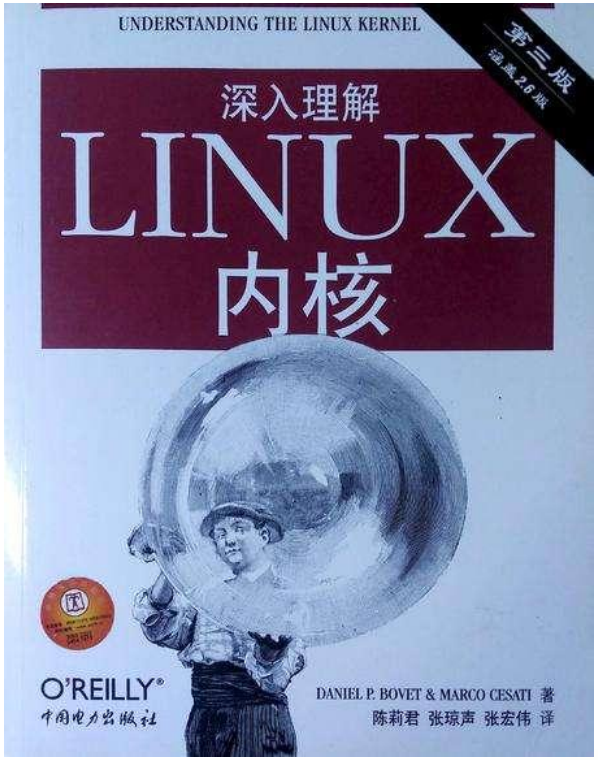
控制寄存器

- 这几个寄存器中保存全局性和任务无关的机器状态。
- CR0中包含了6个预定义标志，这里介绍内核中主要用到的0位和31位。0位是保护允许位PE(Protedted Enable)，用于启动保护模式，如果PE位置1，则保护模式启动，如果PE=0，则在实模式下运行。CR0的第31位是分页允许位(Paging Enable)，它表示芯片上的分页部件是否被允许工作。由PG位和PE位定义的操作方式如图所示。
- 其他几个寄存器暂时不详细介绍。

Linux内核中C和汇编语言

- GNU 的C语言 http://www.faqs.org/docs/learnc/
- 汇编使用的是AT&T的汇编格式，与Intel的汇编格式稍有差异
- 在C语言中可以嵌入汇编代码，叫GCC嵌入式汇编
- 具体请参见教材2.5节

参考资料



- 深入理解Linux内核第三版第二章
- Intel® 64 and IA-32 Architectures
- Software Developer Manuals

为什么要引入保护模式

那么问题来了，为什么要引入保护模式？保护模式到底保护什么？为什么能达到保护这些对象的目的？

谢谢大家！

THANK YOU