第二章　80X86计算机组织

2.1 80X86微处理器概况

2.2 基于微处理器的计算机系统构成

2.3 中央处理机

2.4 存储器

2.5 外部设备

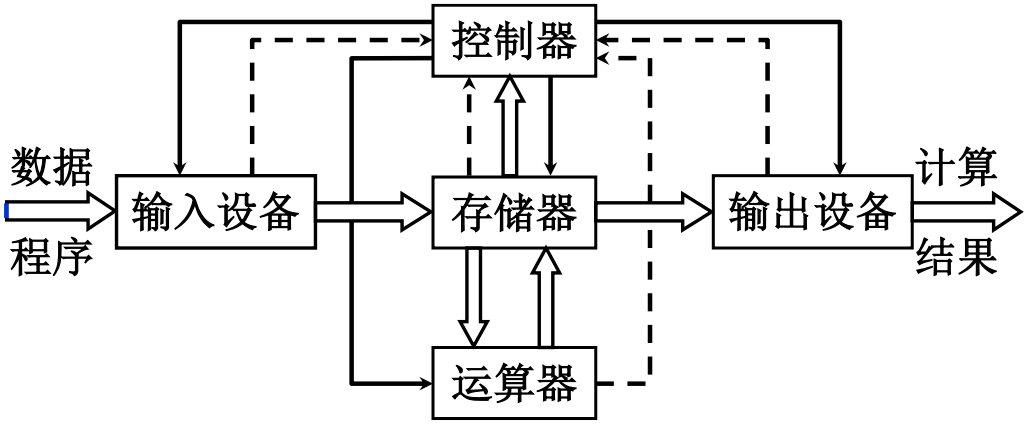
本 章 目 标

- 了解80X86微处理器
- 熟悉基于微处理器的计算机系统构成
- 熟练掌握80X86CPU的寄存器组功能和作用
- 掌握存储器地址的分段表示及其物理地址的计算
- 熟悉段应用规定

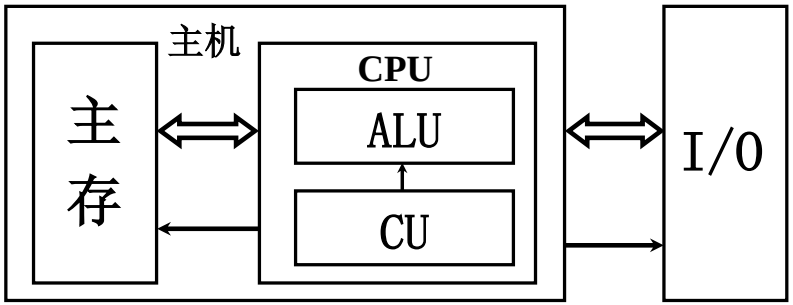
计算机结构

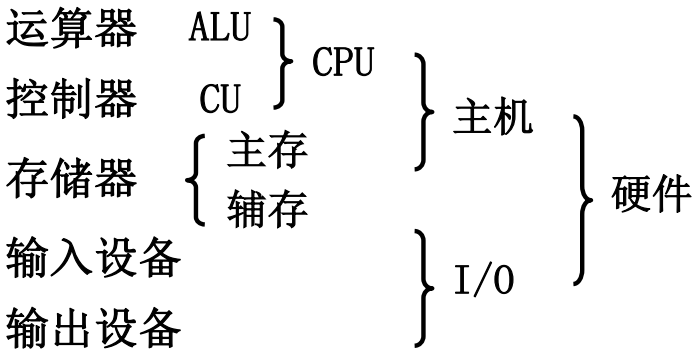
- **以存储器为中心的计算机结构**

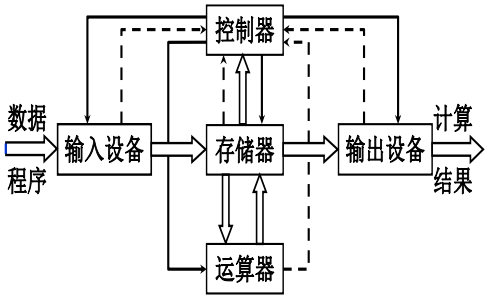
计算机系统主要由存储器、运算器+控制器、输入设备和输出设备构成



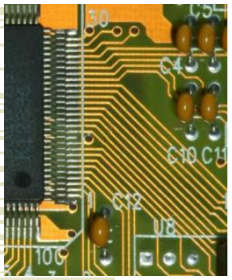
- **现代计算机硬件组成**

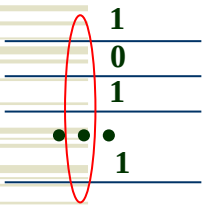




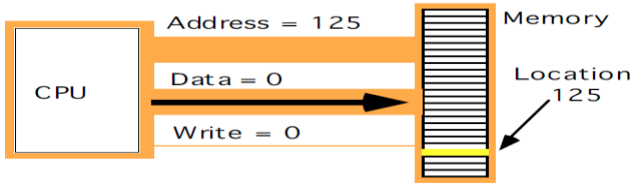


面向总线的体系结构

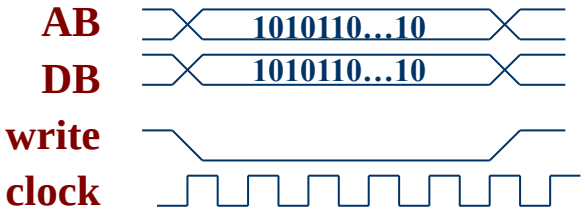


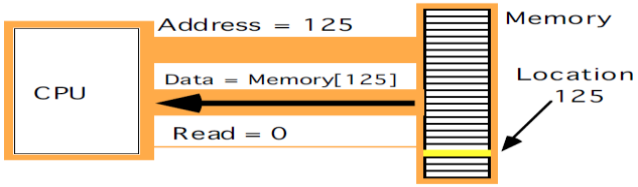


- 计算机五大部件互连方式
  - 分散连接：各部件之间使用单独的连线
  - 总线连接：各部件连接到一组公共信息传输线
  - 从分散连接→总线连接 （I/O设备与主机之间灵活连接）
- 系统总线
  - 数据总线（Data Bus）
    - 各部件之间数据传输
    - 内部数据总线宽度：CPU芯片内部数据传送的宽度（位数）
    - 外部数据总线宽度：CPU与外部交换数据时的数据宽度
  - 地址总线（Address Bus）
    - 传输地址 （Which memory location or I/O devices?）
  - 控制总线（Control Bus）
    - 控制CPU和其他部件的通信方式等 （Is sending or receiving?）

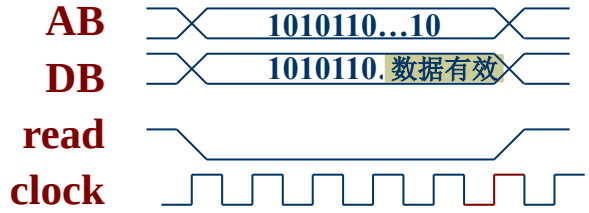


内存写操作　MOV [125], AX





内存读操作　MOV AX, [125]

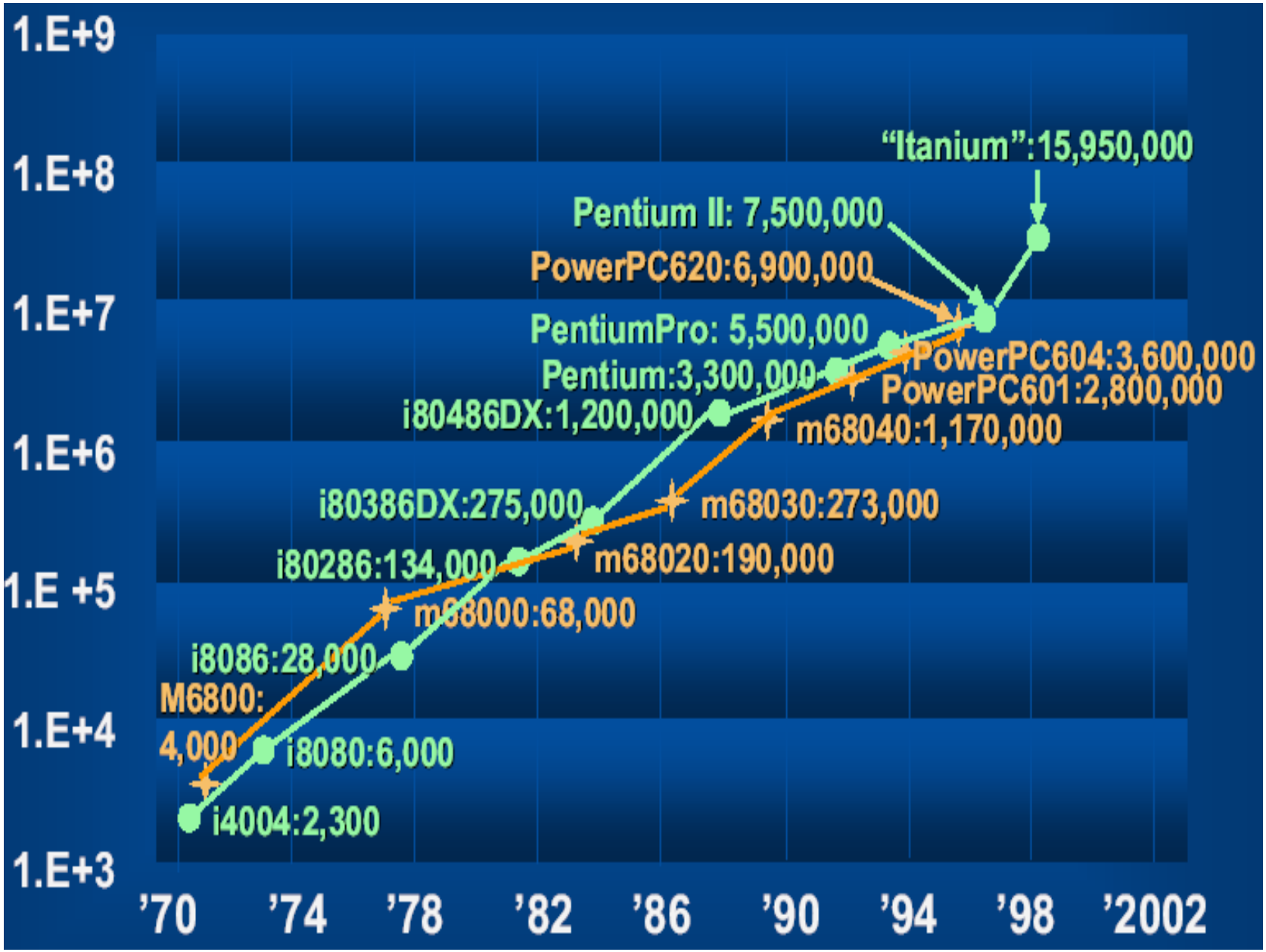


2.1 80X86微处理器

- 20世纪70年代初期，由于大规模集成电路技术的发展 ，已经开始把运算器和控制器集成在一块芯片上，构成中央处理器CPU（Central Processing Unit），80X86是由Intel公司开发的微处理器系列
  - 由80X86微处理器芯片构成的微机称为X86微机
  - 另外还有AMD公司微处理器系列、IBM公司POWERPC、SUN公司SPARC等
  - 各种CPU系列有自己的机器指令集，互不兼容
    - 高级语言：C,PASCAL,FORTRAN,JAVA等高级语言，与机器指令不对应，但对各CPU兼容，因为通过编译转换为对应的机器指令
    - 汇编语言：与机器指令一一对应，对各CPU不兼容，但汇编语言程序设计方法通用，助记符、程序结构大体相同
- 本课程以80X86微处理器为例讲解汇编语言程序设计基本概念、原理、方法；简单介绍ARM处理器相关知识（华为鲲鹏920）

Intel公司处理器系列

- 1971年，设计了第一片4位微处理器Intel 4004, 随之又设计生产了8位微处理器8008
- 1973年推出了8080；1974年基于8080的个人计算机（PC）问世
- 1977年Intel推出了8085
- 自此之后，又陆续推出了8086、80286、80386、80486、Pentium、XEON、EMT’64、Itanium2、多核等80X86系列微处理器

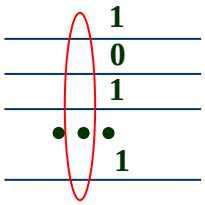

1.E+9
1.E+8
1.E+7
1.E+6
1.E +5
1.E+4
1.E+3
"Itanium":15,950,000
Pentium II: 7,500,000
PowerPC620:6,900,000
PentiumPro: 5,500,000
PowerPC604:3,600,000
Pentium:3,300,000
PowerPC601:2,800,000
i80486DX:1,200,000
m68040:1,170,000
i80386DX:275,000
m68030:273,000
i80286:134,000
m68020:190,000
m68000:68,000
i8086:28,000
M6800:
4,000
i8080:6,000
i4004:2,300
'70
'74
'78
'82
'86
'90
'94
'98
'2002

80X86微处理器概况

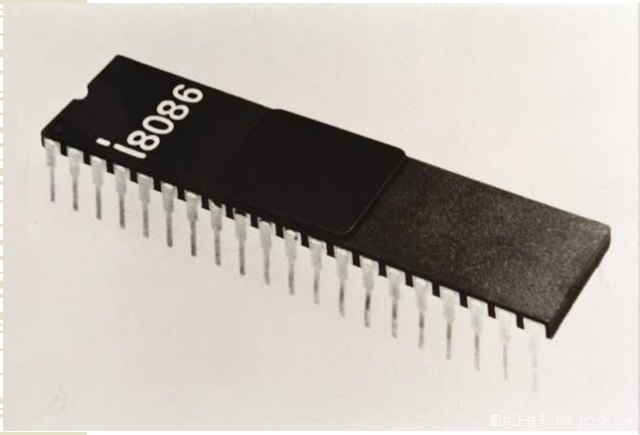
这里仅列出发布年份、字长、主频、地址总线宽度、寻址空间和高速缓冲

型号	发布年份	字长	主频(MHz)	地址总线宽度	寻址空间	高速缓存
8086	1978	16	4.77	20	1M	无
8088	1979	16	4.77	20	1M	无
80286	1982	16	6~20	24	16M	无
80386	1986	32	12.5~33	32	4G	有
80486	1989	32	25~100	32	4G	8KB
Pentium (586)	1993	32	60~166	32	4G	8KB数 8KB指令
PRO(P6)	1995	32	150~200	36	64G	8KB数据 8KB指令 256KB二级Cache
PII	1997	32	233~333	36	64G	32KB 512JB二级Cache, 有独立封装和独立总线

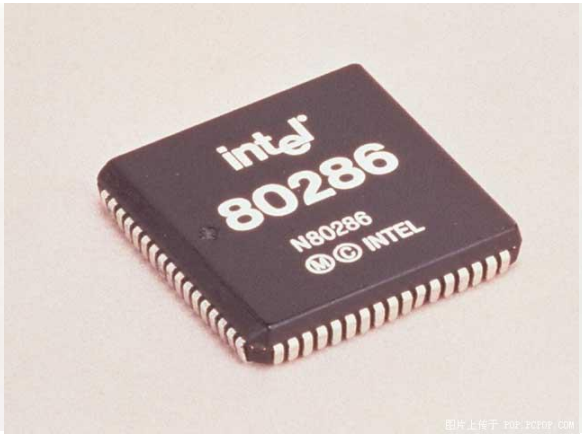
字长增加有利于提高计算精度和速度



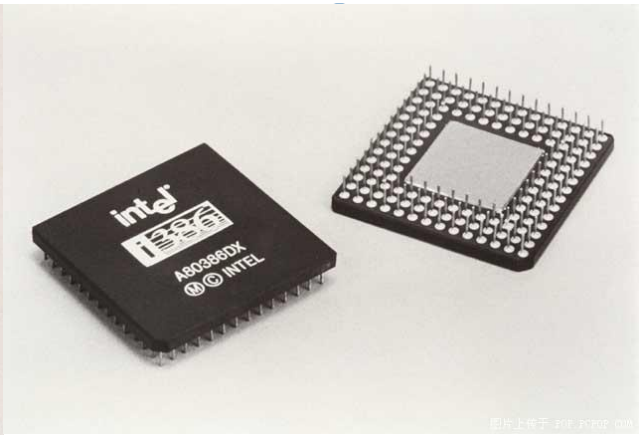
CPU和内存之间有多少根地址信号线，2^{20}=1M



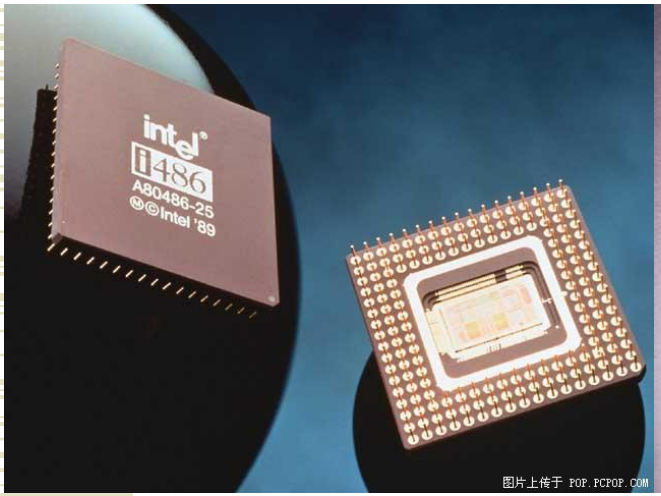
8086 (1978)



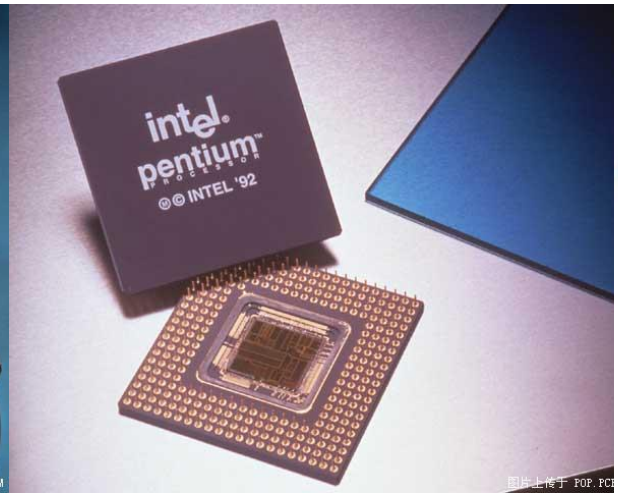
80286 (1982)



80386 (1985)



80486 (1989)



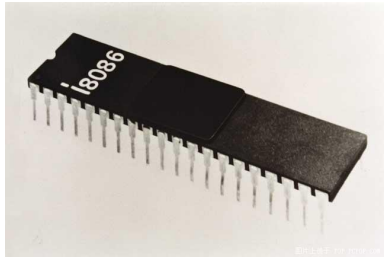
Pentium (1993)

Pentium IV (2000)

- 80286开始：增加了保护模式，可提供虚拟存储管理和多任务管理
  - 虚拟存储管理：提供更大的存储空间，允许用户运行程序空间大于实际主存储器空间
  - 多任务管理：允许多个用户或多个任务同时使用计算机
- 80386开始：又增加了虚拟8086工作模式
  - 虚拟86工作模式：一台机器可同时模拟多个8086处理器的工作，多用户可以完全安全隔离等
  - 硬件支持多任务转换，提高效率

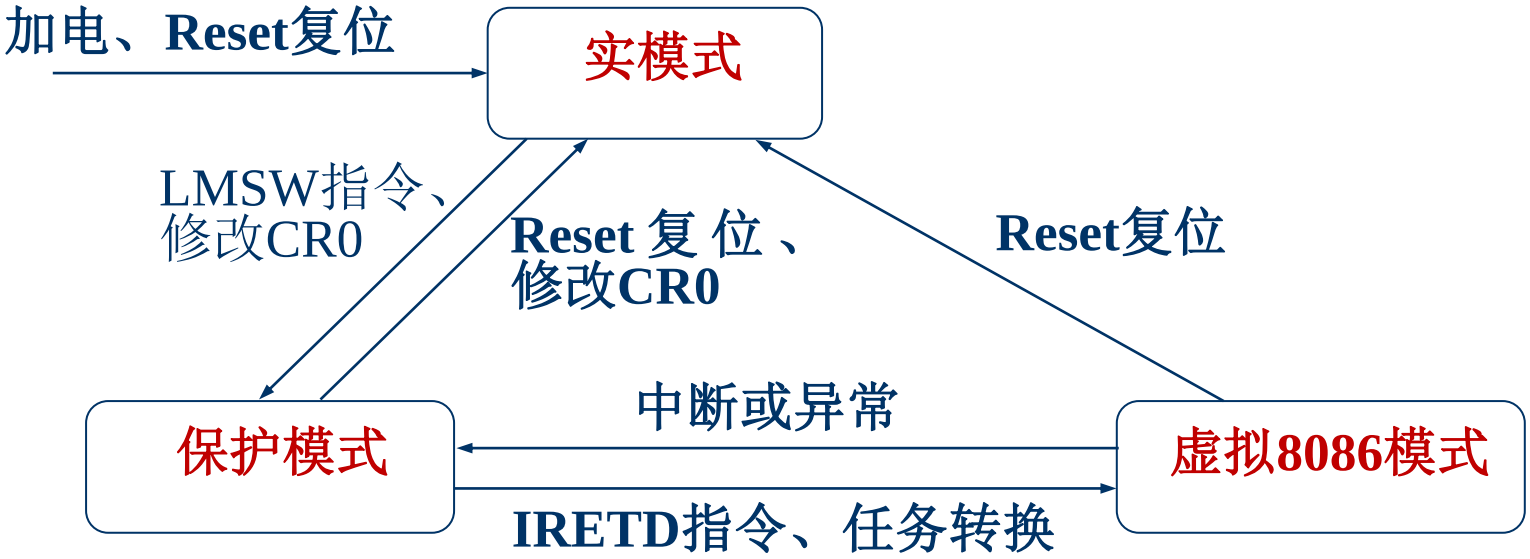
CPU+浮点运算协处理器
如8086、8087独立芯片

- 80486开始：把协处理器集成到CPU芯片中，提高浮点处理速度
- 字长增加有利于提高计算精度



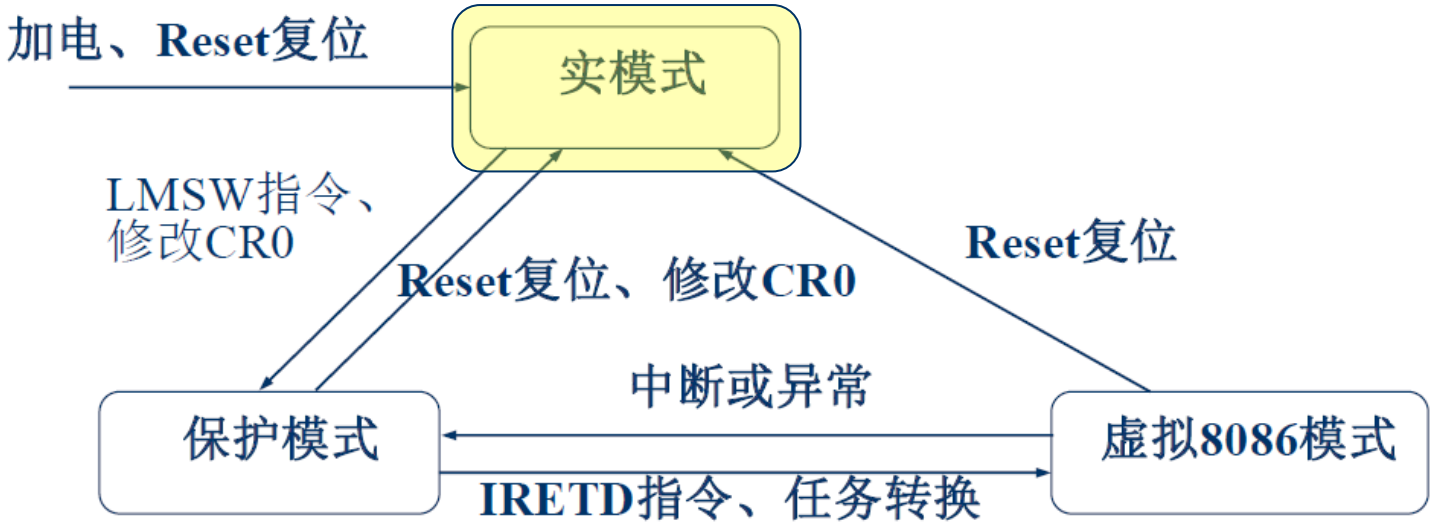
3种运行模式关系

- 8086，CPU只具有1种运行模式：实模式
- 80286开始，CPU具有2种运行模式：实模式、保护模式
- 80386开始，CPU具有3种运行模式：实模式、保护模式和虚拟8086模式



(1) 实模式

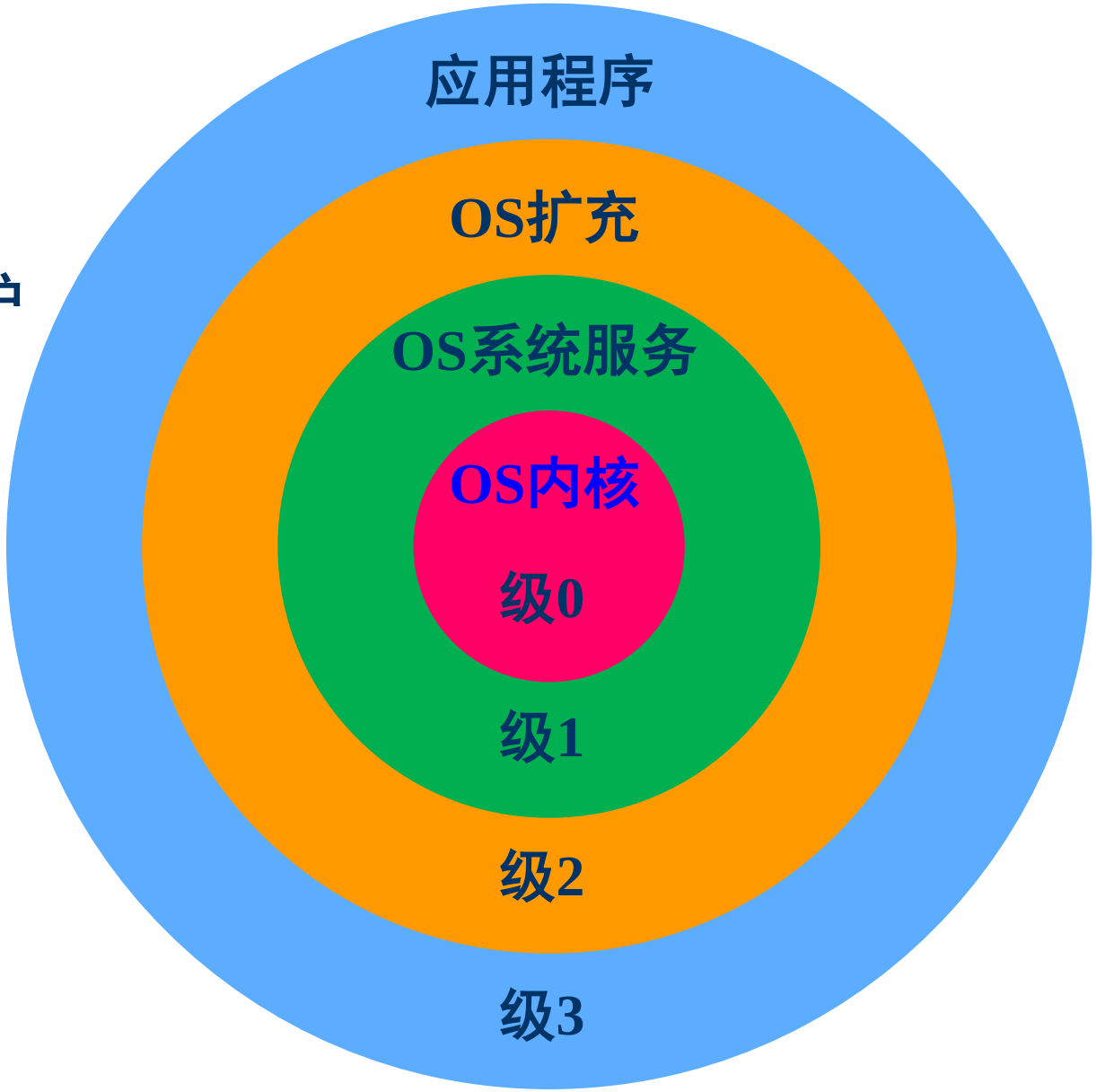
- CPU复位（Reset）或加电（Power On）时，处理器以实模式工作
- 在实模式下，80X86内存寻址方式和8086相同，由16位段寄存器和16位偏移地址形成20位的内存物理地址
- **在实模式下，所有的段都是可以读、写和可执行的**



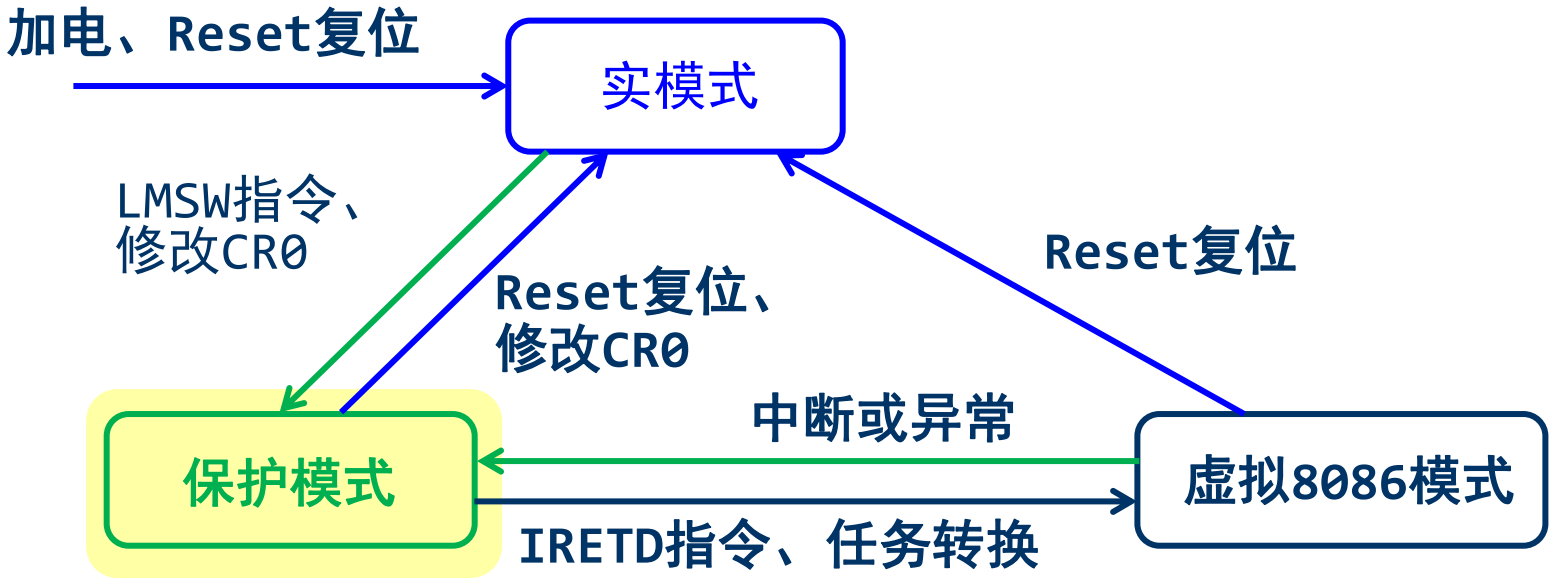
(2) 保护模式

- 在保护模式下，CPU提供了多任务、内存分段分页管理和特权级保护等功能
  - 这些功能是Windows/Linux等现代操作系统的基石
  - 如果没有CPU的支持，操作系统许多功能根本无法实现
    - 在实模式下，应用程序可以执行任何的CPU指令，读写所有的内存，DOS操作系统就不能控制应用程序的行为，应用程序可以做任何事情，没有任何限制。
    - 而在保护模式下，通过设置特权级和内存的分段分页，应用程序只能读写属于它自己的内存空间，而不能破坏其他应用程序和操作系统

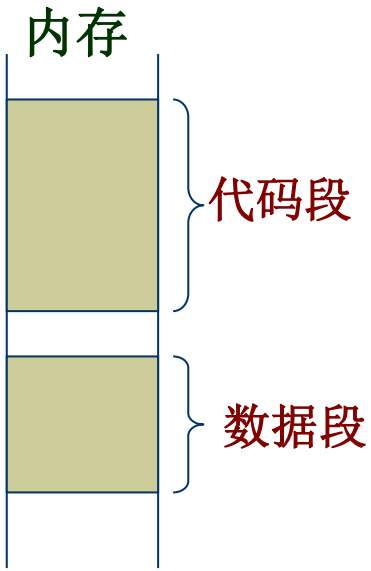
Pentium的存储保护包括特权级保护和存储区域保护

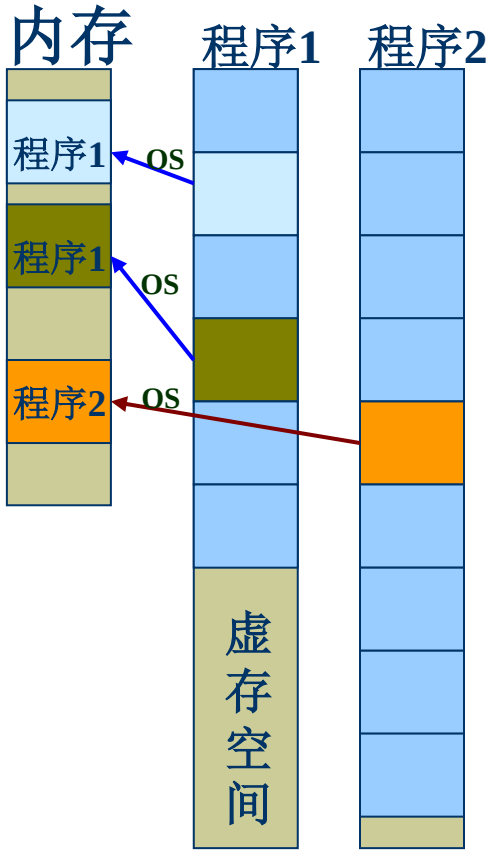


- 实模式下没有特权级的概念，相当于所有的指令都工作在操作系统的特权级0，即最高的特权级。它可以执行所有特权指令，包括读写控制寄存器CR0等。
  - Windows/Linux操作系统就是通过在实模式下初始化控制寄存器、GDTR、LDTR、IDTR、TR等寄存器以及页表，然后再通过置CR0的保护模式位（PE位）为1而进入保护模式的
- 实模式下不支持硬件上的多任务切换，所有的指令都在同一个环境下执行

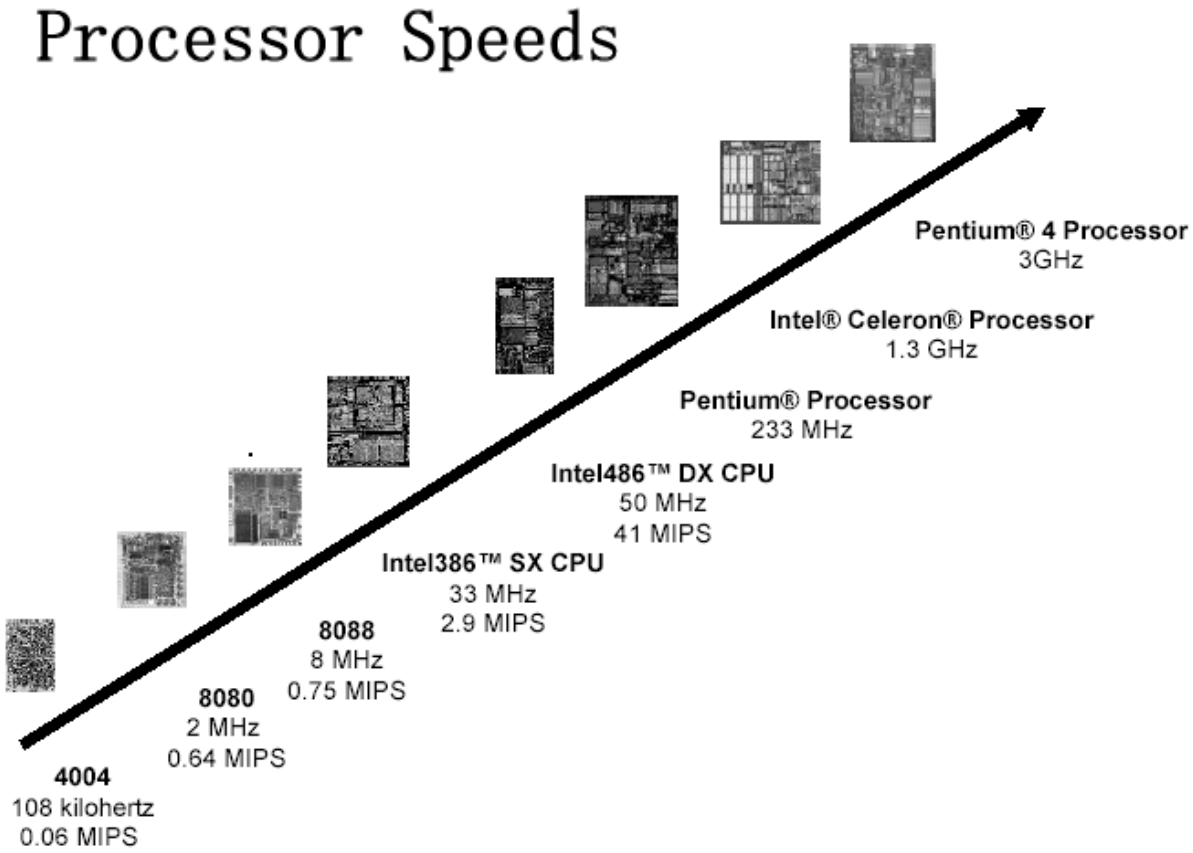


- 保护模式下提供的主要功能有：
  - 段的大小可以设置为2^{32}=4GB，段内的偏移量为32位
  - 特权级保护
  - 支持内存分页机制，支持虚拟内存
  - 支持多任务



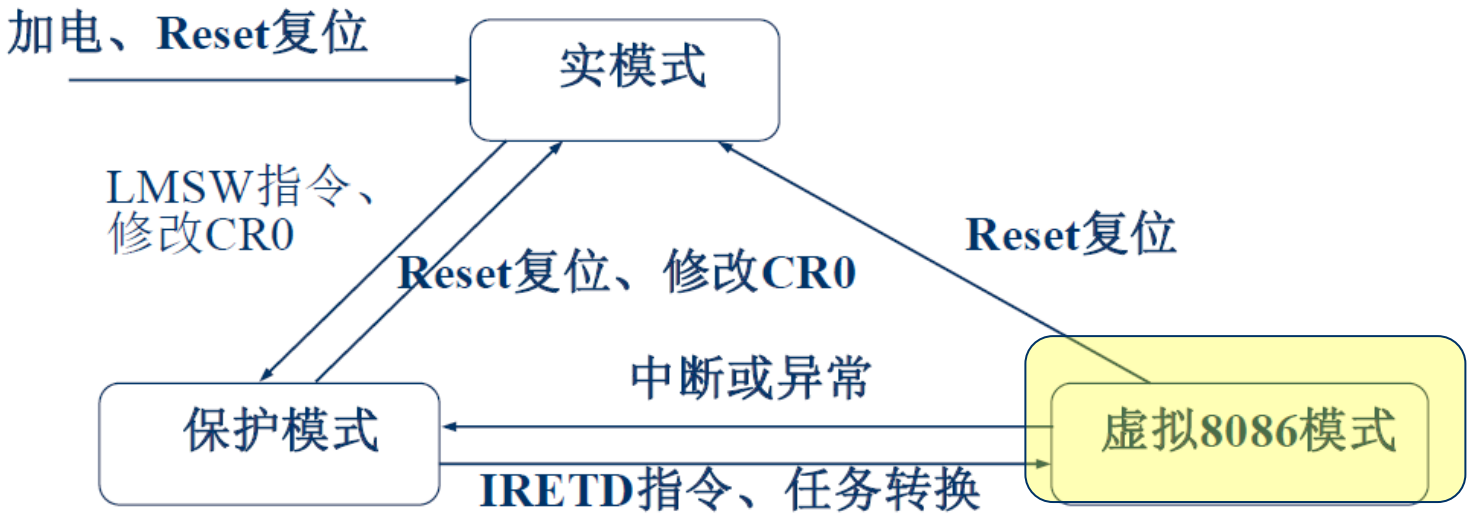


- Intel 推出的80x86系列处理器的性能和功能越来越强。但是，从汇编语言程序设计人员面对这些CPU的体系结构角度来看，8086的实模式和80386的保护模式到目前为止一直适用。
- **本课程介绍的实模式编程以8086为例说明，保护模式编程以80386为例说明**



(3) 虚拟8086模式

- CPU可以同时支持多个真正的CPU任务和多个虚拟86任务
- 以任务形式在保护模式下执行
- CPU支持任务切换和内存分页

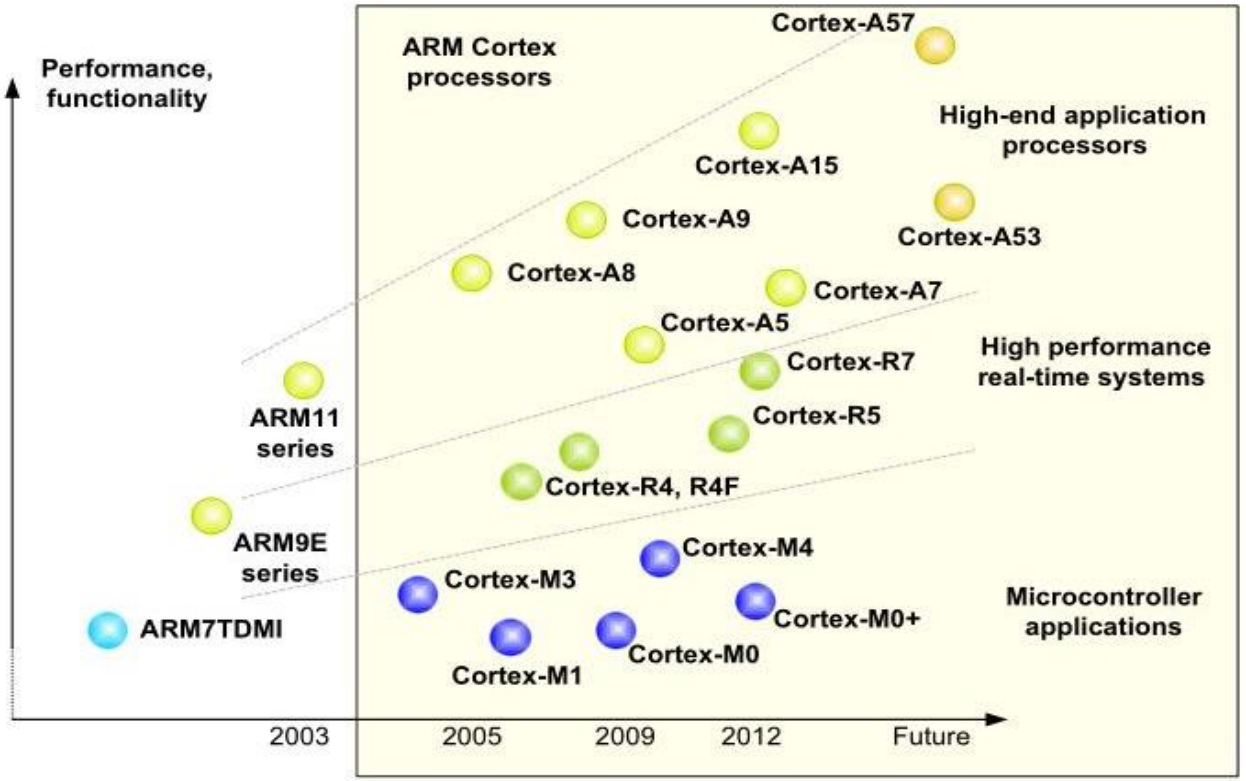


2.1.2 ARM处理器

ARM发展史

- 1985年，Roger Wilson和Steve Furber设计了第一代32位、6MHz处理器，简称ARM (Acorn RISC Machine)
- 1990年11月27日，Acorn公司正式改组为ARM计算机公司。由于缺乏资金，做出了一个意义深远的决定：
  - 自己不制造芯片，只将芯片的设计方案授权给其他公司，由别人生产
  - 反而使得ARM技术得到了广泛推广应用

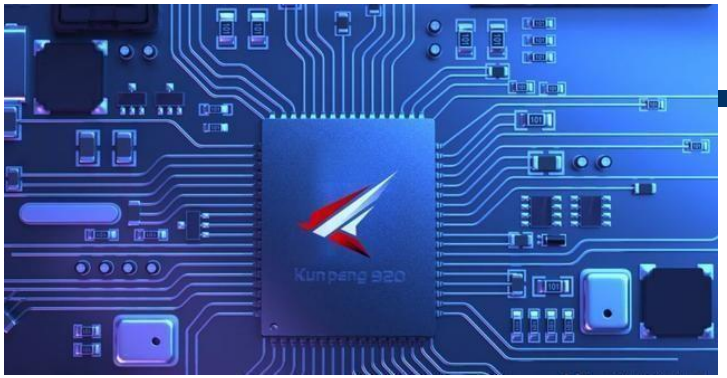
- 21世纪后，由于手机快速发展，出货量呈现爆炸式增长，ARM处理器占领了全球手机市场。
- 2004年，Cortex系列诞生，从此该公司不再用数字为处理器命名。它分为A、R和M三类，旨在为各种不同的市场提供服务。
- 2015年，ARM基于ARMv8架构推出了一种面向企业级市场的新平台标准。



Cortex处理器三大方向发展进程

ARM处理器芯片

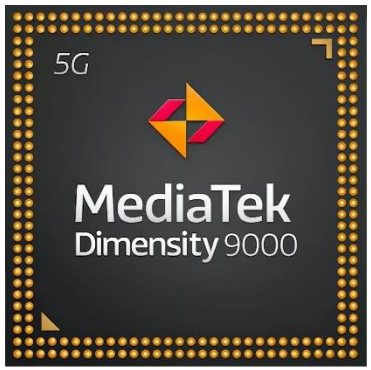
富岳A64FX



鲲鹏920

苹果M1

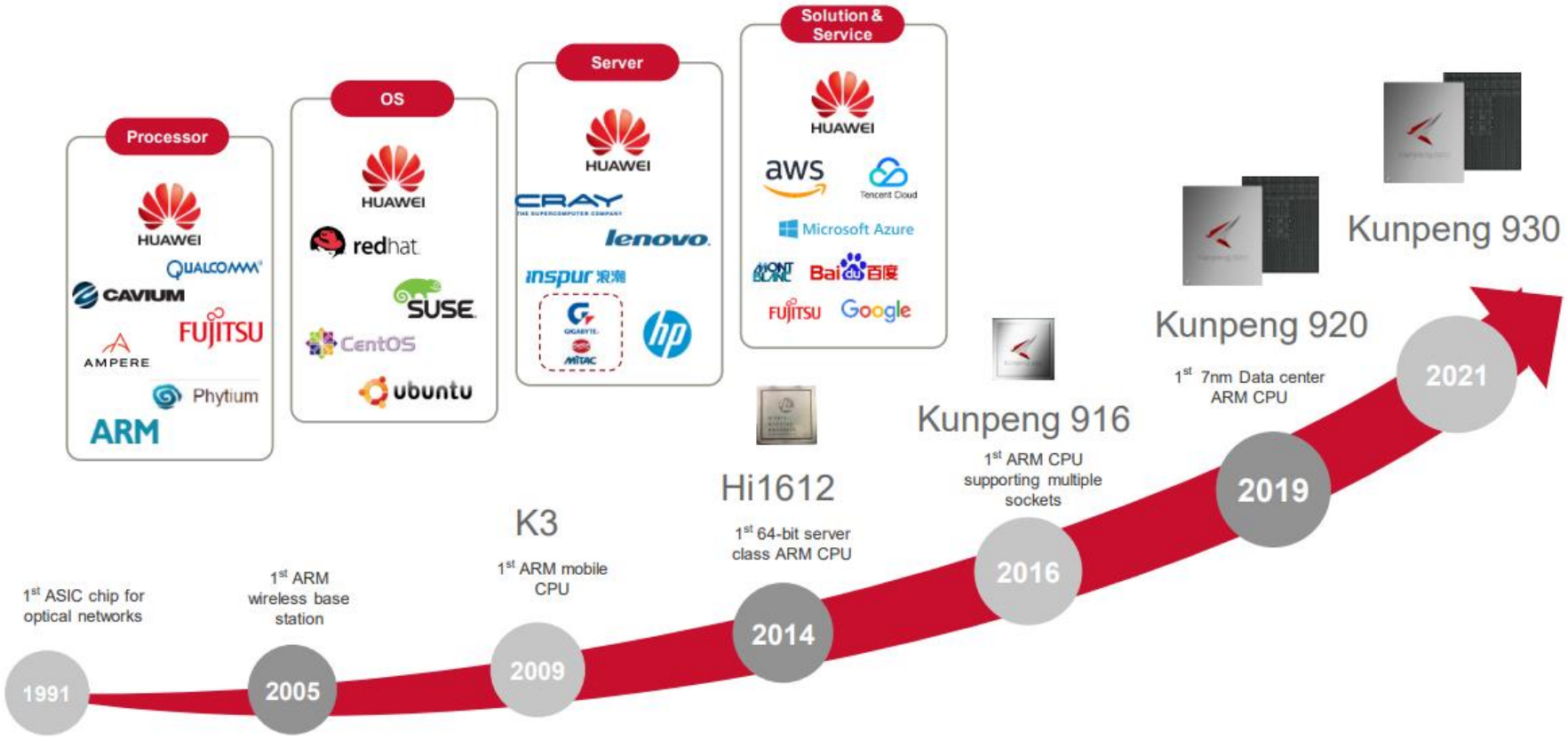
高通骁龙888



天玑2000

Exynos9820

华为对ARM的开发与应用



鲲鹏920处理器

- 基于ARMv8指令集，高性能服务器处理器
- 采用7nm工艺
- 最多64核
- 支持8通道DDR4内存及PCIe 4.0协议

Kunpeng 920

ARM-based CPU with the industry's highest performance

Processor	Kunpeng 920
Core	ARM v8.2 architecture, TaiShan core 2.6/3.0 GHz, 32/48/64 cores per socket
Cache	L1: 64 KB instruction cache and 64 KB data cache L2: 512 KB private per core L3: 24–64 MB shared for all (1 MB/core)
Memory	8 DDR4 channels per socket, up to 3200 MHz
Coherent Interconnect	Coherent SMP interface for 2S & 4S 3 ports, up to 240 Gbit/s per port
I/O	40 PCIe Gen 4.0 lanes 2 x 100GE, RoCEv2/RoCEv1, CCIX x4 USB 3.0, x16 SAS 3.0, x2 SATA 3.0
Package	60 mm x 75 mm, BGA
Process	7 nm
Power	TDP: 120/150/180/200W

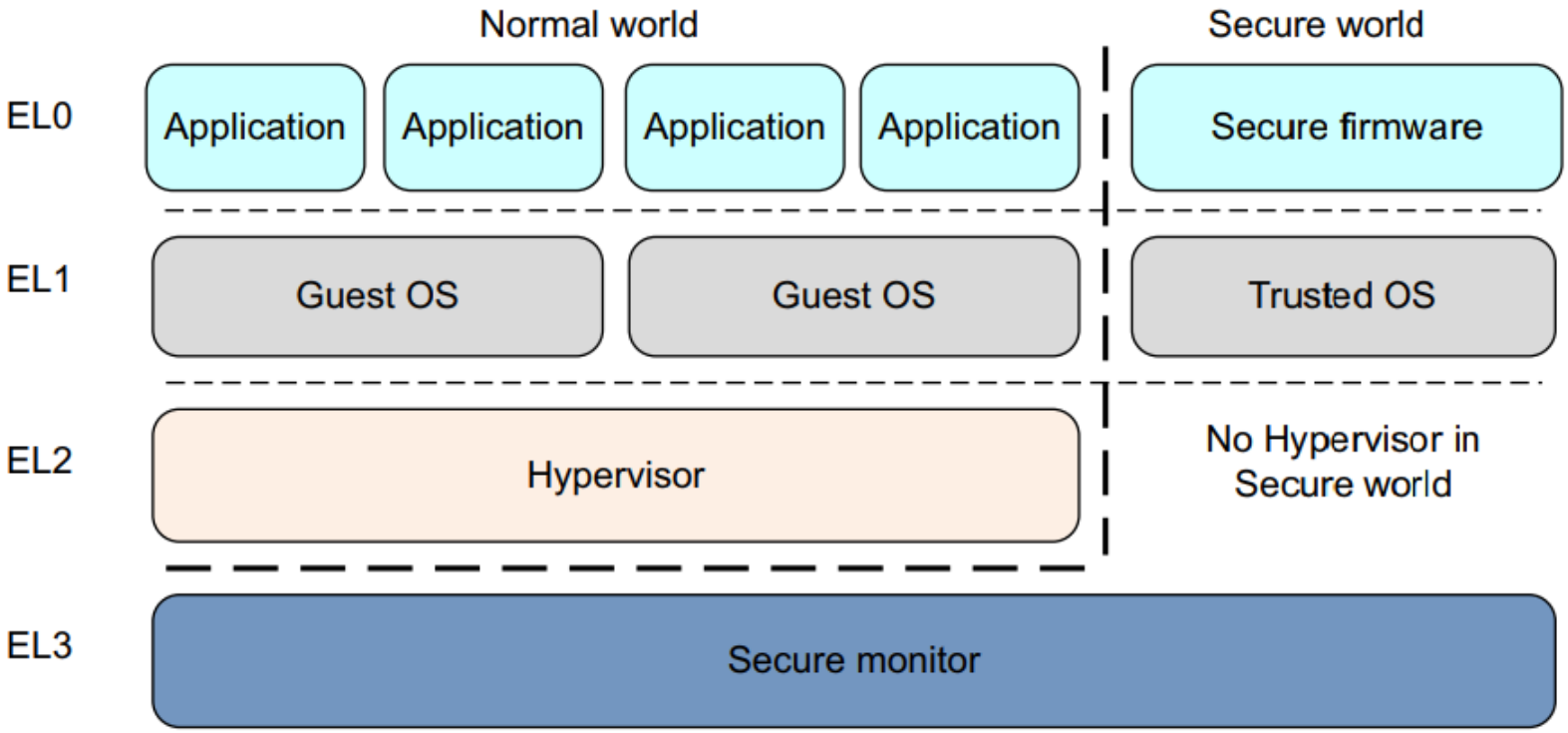
ARM处理器异常级别

- 异常（exception）和特权（privilege）是在ARMv8-A中定义的两个概念。
- Armv8-A通过实现不同级别的特权来实现这种分离。当前特权级别只能在处理器接受异常或从异常返回时更改。
- 在Armv8-A体系结构中，这些特权级别被称为异常级别（exception level，以下简称EL）。
- 每个异常级别都有编号，权限级别越高，编号越高。
- 通常情况下，一个软件，例如应用程序，操作系统内核或是虚拟机监视器仅占据一个异常级别。
- 一个例外是in-kernel监视器例如KVM，它跨越了EL2和EL1。

EL0：通常用户程序
EL1：操作系统内核
EL2：虚拟机监视器
EL3：低级固件

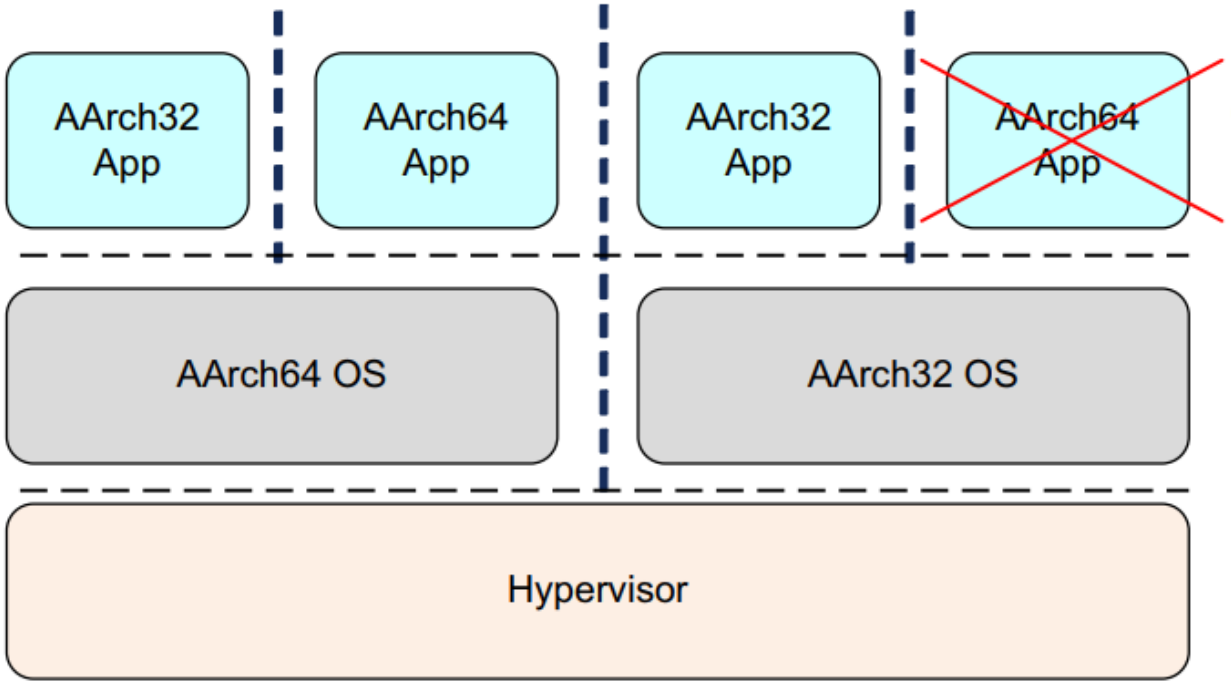
ARM处理器安全状态

- ARMv8-A提供了两个安全状态
  - Non-secure state被称为Normal world，
  - secure state被称为Secure World。



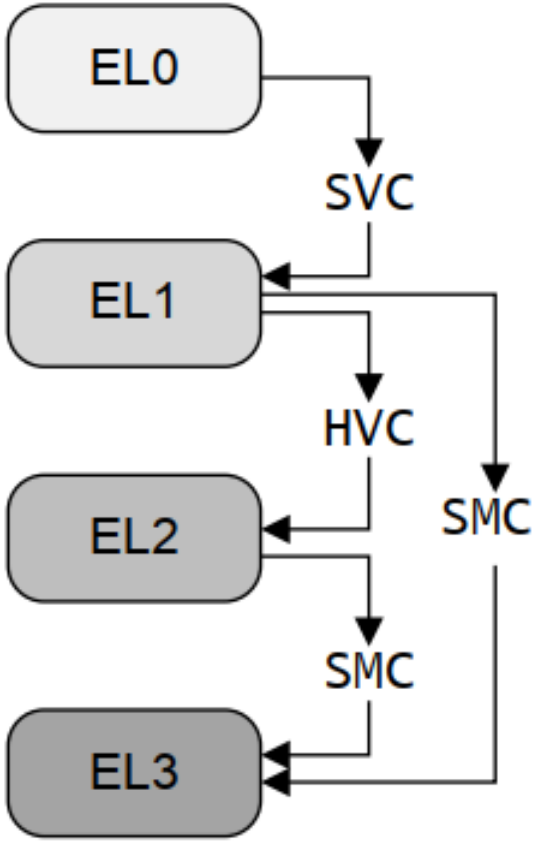
ARM处理器执行状态

- ARMv8-A有两种可用的执行状态：
  - AArch32：32位执行状态。此状态下的操作与Armv7-A兼容。有两个可用的指令集：T32和A32。标准寄存器宽度为32位。
  - AArch64：64位执行状态。有一个可用的指令集：A64。标准寄存器宽度为64位。

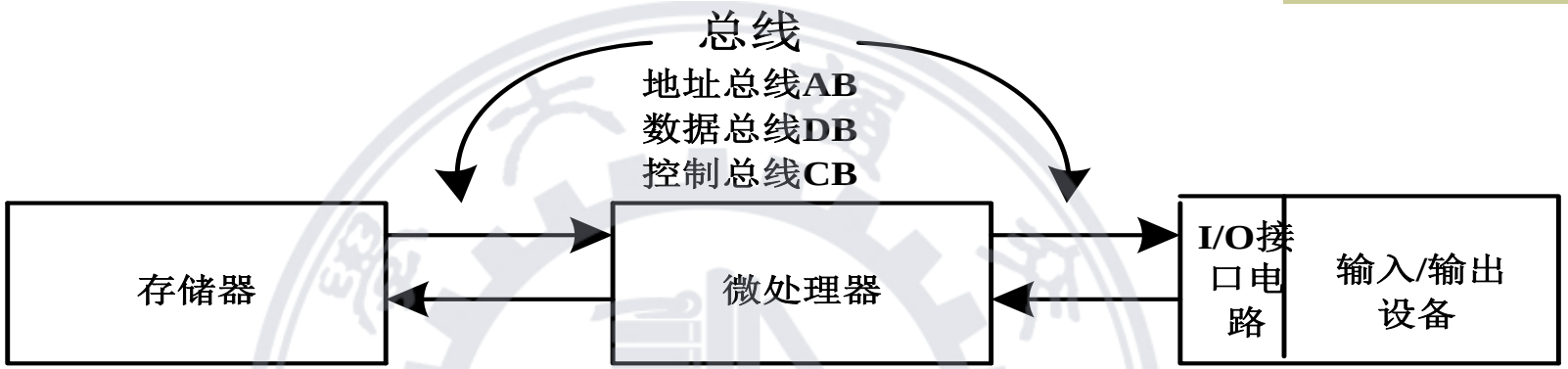


ARM异常基本切换

- 异常事件
  - Aborts：指令/数据中止
  - Reset
  - 异常生成指令
    - SVC：The Supervisor Call，用户程序调用系统内核服务
    - HVC：The Hypervisor Call，内核调用虚拟机监视服务
    - SMC：The Secure monitor Call，非安全模式调用安全服务
  - 中断
    - IRQ：中断请求
    - FIQ：快速中断请求
    - Serror：系统错误



2.2 基于微处理器的计算机系统构成



动态RAM
静态RAM
高速缓存
只读存储器ROM
等

8086
8088
80186
80286
80386
80486
Pentium
Pentium Pro
Pentium Pro mmx

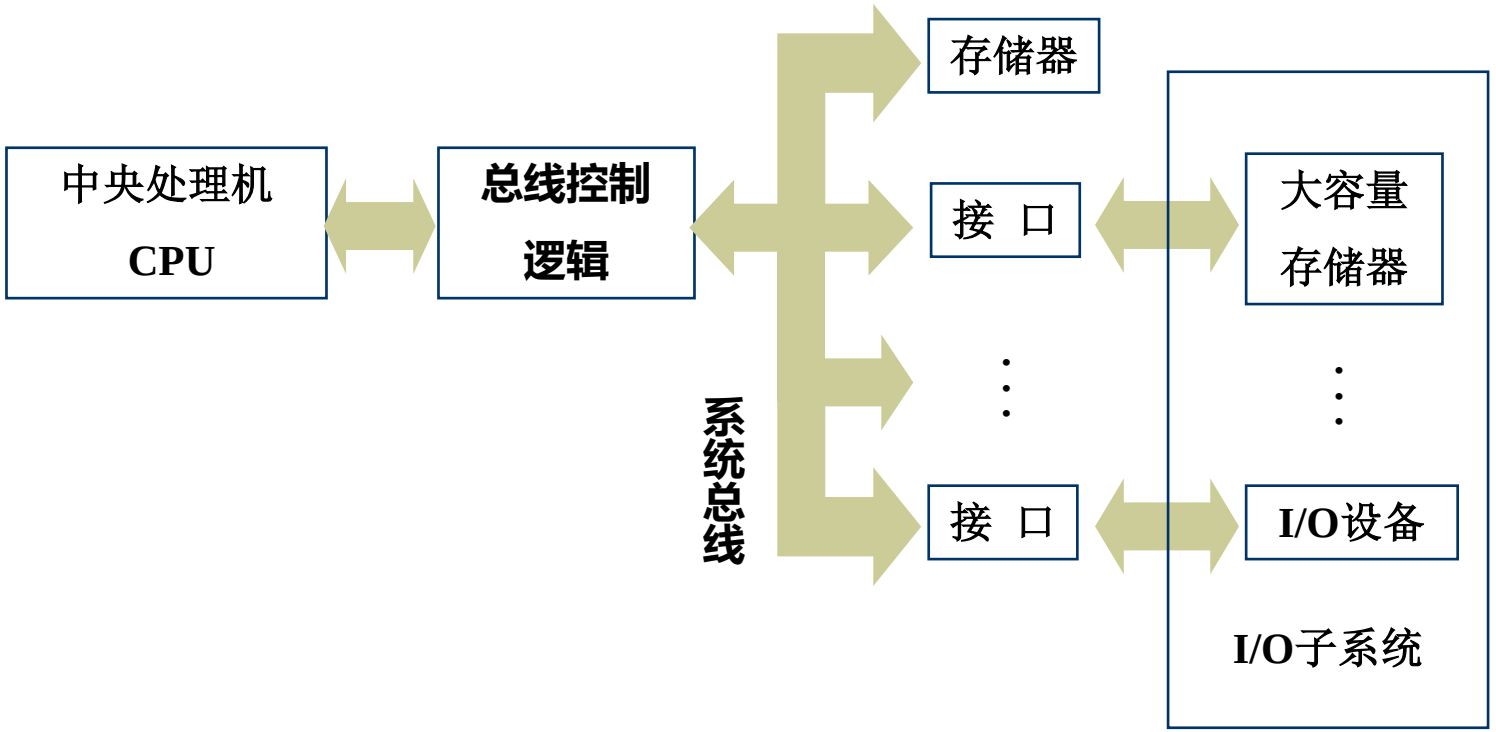
键盘
显示器
打印机
串行通信设备
软盘驱动器
硬盘驱动器
鼠标
CD-ROM驱动器
绘图仪
磁带机

基于微处理器的个人计算机方框图

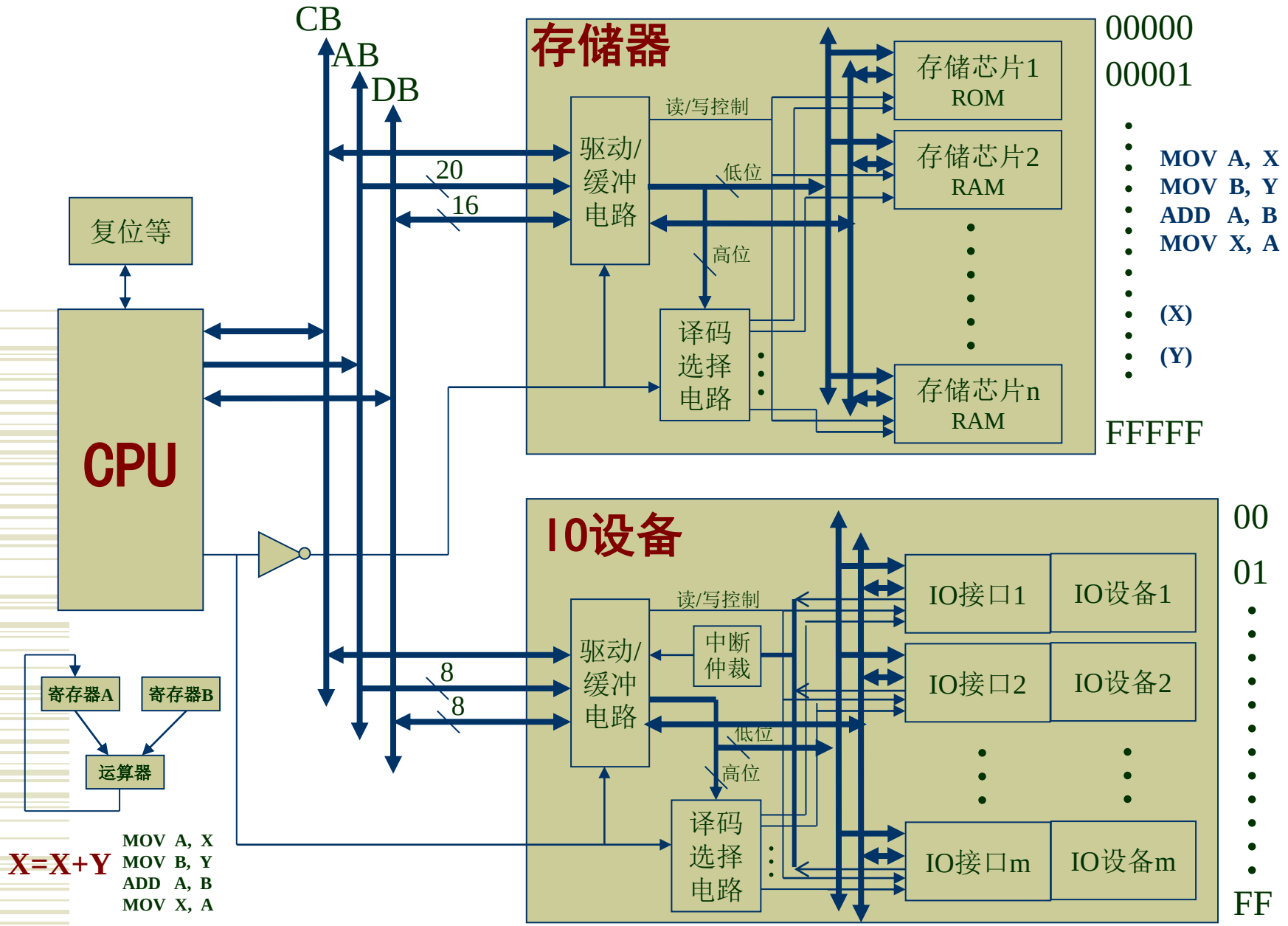
2.2.1 硬件

硬件包括：电路、插件板、机柜等。

典型的计算机结构硬件包括：微处理器、存储器、I/O接口电路及输入输出设备。



邮政编号：710049 → 71：西安市，0049：西安交通大学



2.2.2 软件

软件是为了运行、管理和维护计算机而编制的各种程序的总和。软件可分为系统软件和用户软件两大类

- **系统软件**：由计算机生产厂家提供给用户的一组程序，包括：操作系统、系统程序（编译、汇编、连接等）
  - 核心是操作系统OS
  - C等编译器
  - 汇编语言工具软件
    - MASM.EXE　　TASM.EXE
    - LINK.EXE　　TLINK.EXE
    - DEBUG.EXE
- **用户软件**：用户自行编制的各种程序，包括：用户程序、用户程序库

2.3 中央处理机

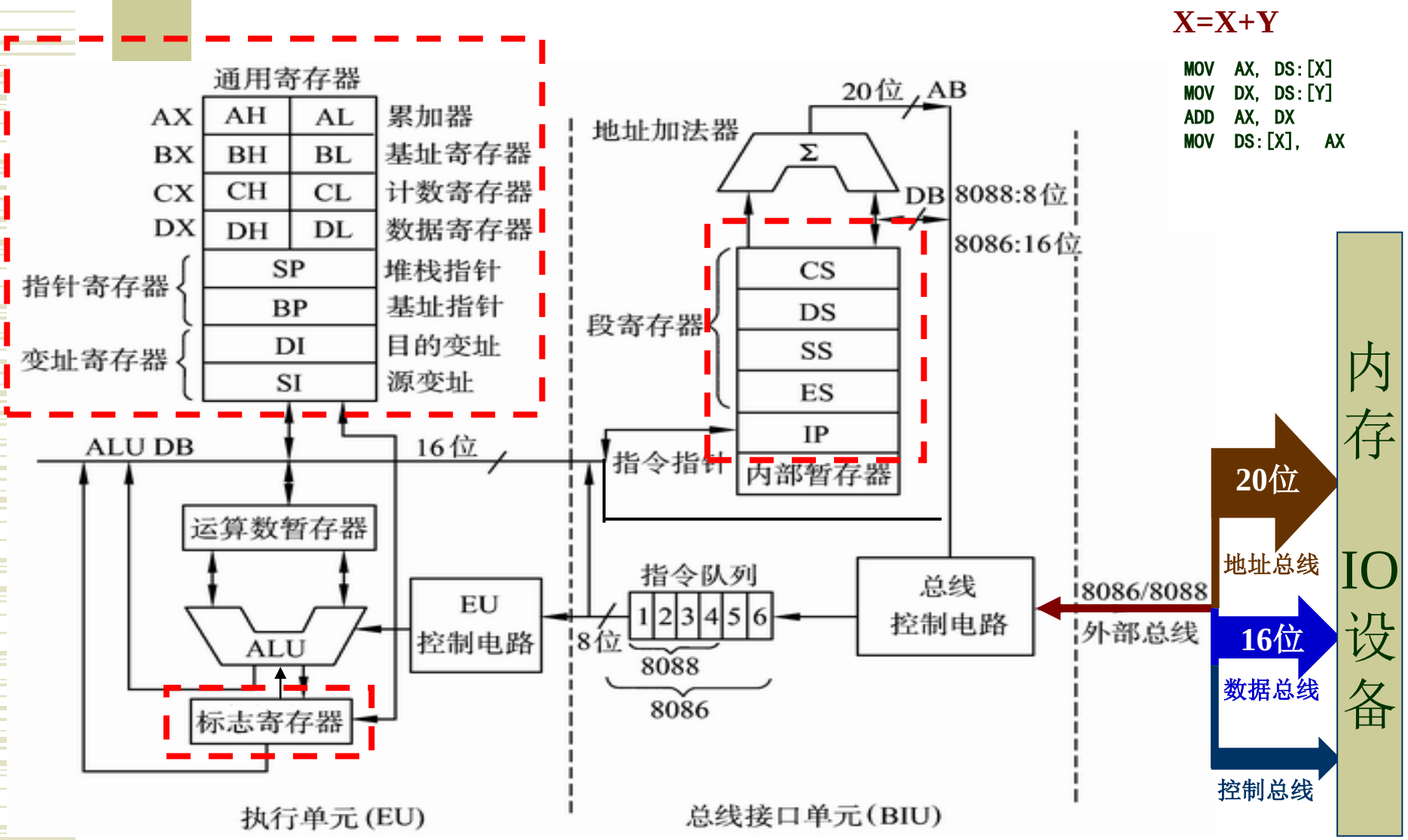
2.3.1 中央处理机（CPU）的组成

CPU的任务是执行存放在存储器里的指令序列，完成用户指定的功能

CPU组成：

1. 算术逻辑部件ALU
2. 控制逻辑EU
3. 工作寄存器（☆必须熟记）

8088/8086CPU的内部结构框图

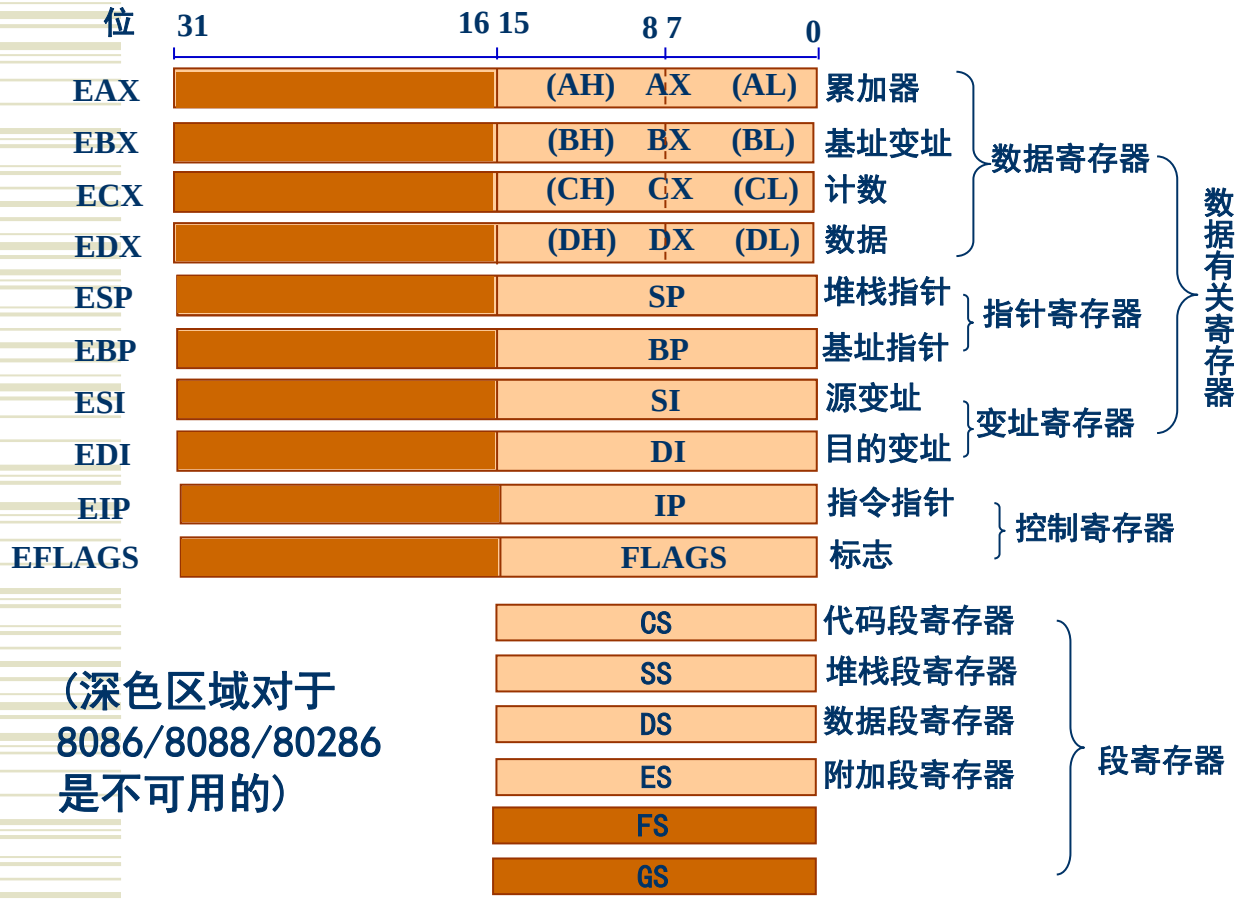


基本概念--寄存器 (Register)

- 处理器中临时数据的存储单元
- 存放运算过程中所需要的或所得到的信息（地址、数据、中间结果）
- 访问速度比内存快

2.3.2 80X86的寄存器组

80X86 程序可见寄存器组：



(深色区域对于8086/8088/80286是不可用的)

- 程序可见寄存器组包括多个8位、16位和32位寄存器，如图所示。深色部分只对80386（含80386）以上CPU有效。

1. 通用寄存器

AX、BX、CX、DX、SP、BP、SI、DI

80386以上CPU:EAX、EBX、ECX、EDX、ESP、EBP、ESI、EDI

2. 专用寄存器

SP、IP、FLAGS

80386以上CPU: ESP、EIP

3. 段寄存器

CS、DS、ES、SS、*FS*、 *GS*

以8086/8088的标志为主介绍

- 对于汇编程序员，CPU中的主要部件是寄存器
- 寄存器是CPU中程序员可以用指令读写的部件

8086/8088的寄存器组

✔ 数据寄存器（4个16位）

	高8位	低8位
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

- 暂存计算过程中所用到的：操作数、结果或其他信息
- 4个16位寄存器：
  AX、BX、CX、DX
- 8个8位寄存器：
  AH、AL、BH、BL、CH、CL、DH、DL

专用目的

AX：累加器，乘除指令中存放操作数，I/O指令使用其与外设传送信息

BX：基址寄存器

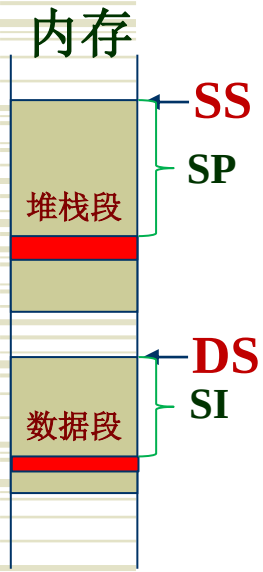
CX：计数器（移位指令、循环指令、串处理指令）

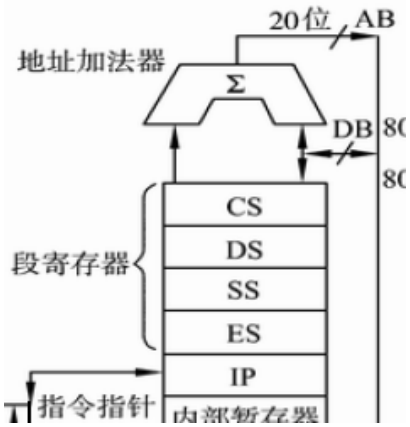
DX：双字长运算（和AX组合）存放高位字，I/O操作存放I/O端口地址

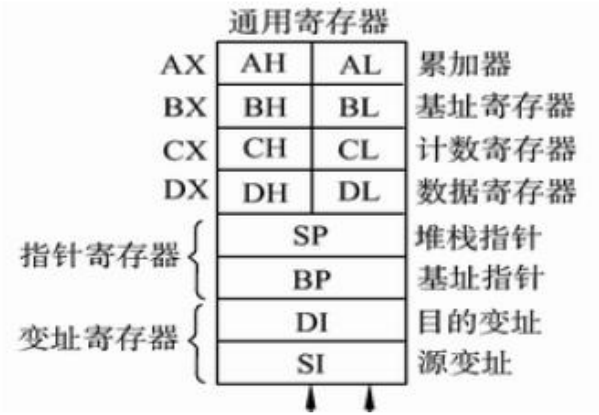
注：386以上增加四个32位寄存器：EAX、EBX、ECX、EDX。

指针及变址寄存器（4个，16位）

- **堆栈指针寄存器：SP**
  - 存放当前堆栈段栈顶偏移量
  - 总是与SS堆栈段寄存器配合存取堆栈中的数据
- **基址指针寄存器：BP**
  - 存放地址的偏移量（或数据）
  - 若存放偏移量时，缺省情况与SS配合
- **变址寄存器：SI**
  - 存放串数据的源地址偏移量（或数据）
  - 若存放偏移量时，缺省情况与DS配合
- **变址寄存器：DI**
  - 存放串数据的目的地址偏移量（或数据）
  - 若存放偏移量时，缺省情况与DS配合

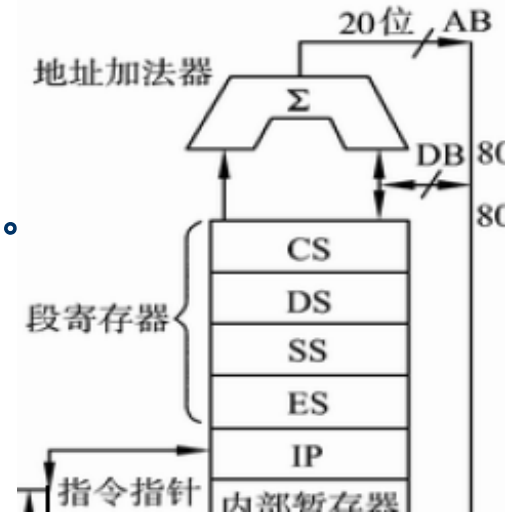




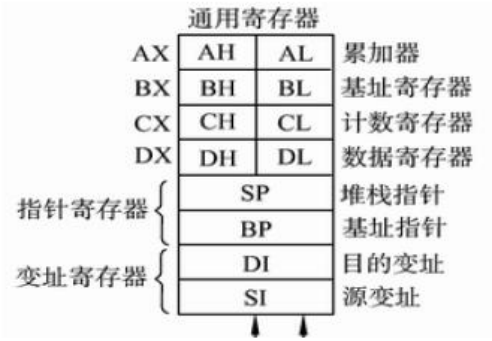


- 注： ESP、EBP、ESI、EDI（32位）只有386以上使用
  实模式使用SP、PB、SI、DI，保护模式使用ESP、EBP、ESI、EDI

段寄存器（4个，16位）



- 存储器采用分段管理方法组织；存储单元物理地址可以用段基址和偏移量计算获得；一个程序可以由多个段组成。
- **功能：** 段寄存器存放段基址。在实模式下存放段基地址，在保护模式下存放段选择子
- **代码段寄存器：CS**
  - 存放当前正在运行的程序代码段基地址（开始地址）
- **堆栈段寄存器：SS**
  - 指定堆栈段位置，存放堆栈段的基地址
  - 堆栈段是在内存开辟的一块特殊区域，其中的数据访问原则是后进先出（LIFO），SP指向栈顶，SS指向堆栈段基地址
- **数据段寄存器：DS**
  - 指定当前运行程序所使用的数据段基地址
- **附加数据段寄存器：ES**
  - 指定当前运行程序所使用的附加数据段基地址

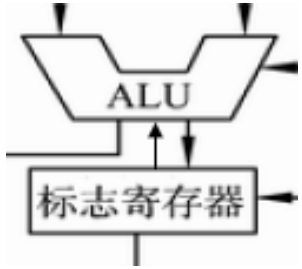


- **注：**
  - 段寄存器FS和GS指定当前运行程序的另外两个存放数据的存储段，只对80386以上；
  - 在默认情况下使用DS所指向段的数据，若要引用其他段中的数据，通常需要显式说明

控制寄存器（2个）

- **指令指针寄存器：IP**
  - 存放代码段中的指令地址偏移量，始终指向下一条即将执行的指令的首地址，控制器根据指令字长自动增加
  - 总是与CS段寄存器配合指出下一条要执行指令的地址
  - 实模式使用IP，保护模式使用EIP （386以上）
- **标志寄存器：FLAGS**

（PSW 程序状态字寄存器）



31…22	21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
										OF	DF	IF	TF	SF	ZF		AF		PF		CF	8086/8088
								NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80286
·····					RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80386
·····				AC	RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80486
·····	ID	VIP	VIF	AC	RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80586

标志寄存器FLAGS

记录正在运行程序的当前状态、控制和访问权限等

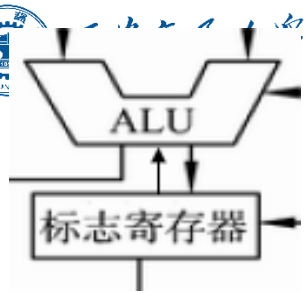
31…22	21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
										OF	DF	IF	TF	SF	ZF		AF		PF		CF	8086/8088
								NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80286
·····					RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80386
·····				AC	RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80486
·····	ID	VIP	VIF	AC	RF	VM		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	80586

PSW (Program Status Word):

条件码标志 – 记录程序运行结果的状态信息，用作后续转移控制条件

控制标志 – 用以控制程序的执行

系统标志 – 用于I/O、可屏蔽中断、程序调试、任务切换和系统工作方式等的控制



8088/8086标志寄存器：FLAGS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

条件码（状态）标志

（记录程序中运行结果的状态信息）

- OF 溢出标志
- SF 符号标志
- ZF 零标志
- CF 进位标志
- AF 辅助进位标志
- PF 奇偶标志

规则：事件成立置1，事件不成立清0

控制标志

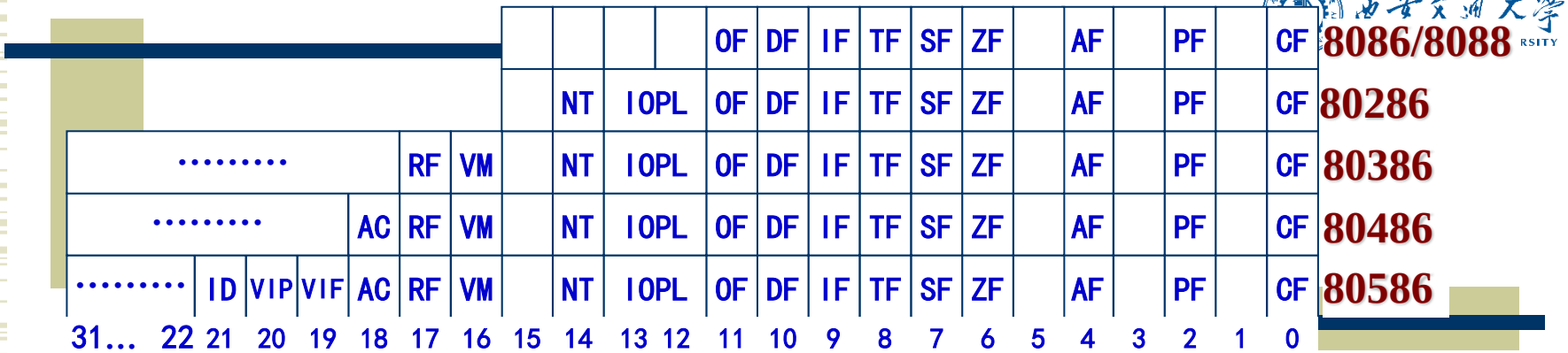
控制标志控制处理器的操作，要通过专门的指令才能使控制标志发生变化

- DF 方向标志

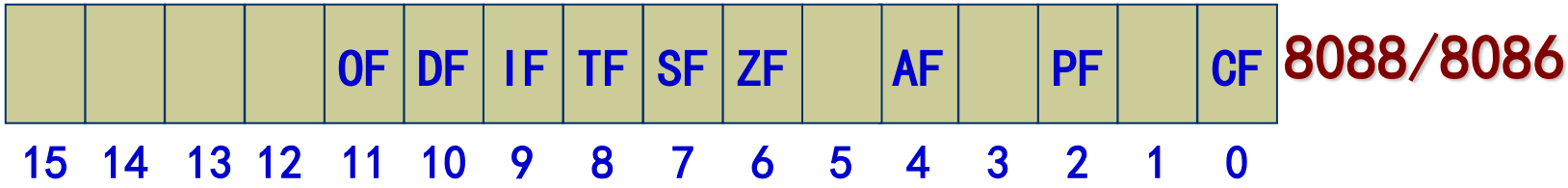
系统标志

用于I/O、可屏蔽中断、程序调试、任务切换和系统工作方式等的控制

- IF 中断标志
- TF 陷阱标志



FLAGS寄存器



- OF(Overflow Flag)溢出标志：由运算结果自动设置

  所谓溢出是指字节（字）运算结果超过了所能表数的范围

  字节运算带符号数范围：-128 ~ +127

  字 运 算带符号数范围：-32768 ~ +32767

  溢出时，标志OF=1，否则OF=0

FLAGS寄存器

8088/8086

- SF(Sign Flag)符号标志：由运算结果自动设置

  SF的值与运算结果的最高位相同

  运算结果为负，SF=1；运算结果为正，SF=0

- ZF(Zero Flag)零标志：由运算结果自动设置

  运算结果为零时，ZF=1；否则，ZF=0。

FLAGS寄存器

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

8088/8086

- **CF(Carry Flag)进位标志：由运算结果自动设置**

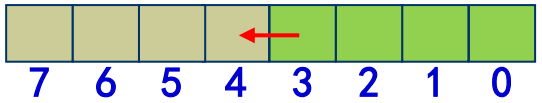
  在运算结果中，若最高有效位产生进位或借位，则CF=1；否则，CF=0

- **AF(Auxiliary Carry Flag)辅助进位标志：由运算结果自动设置**

  记录运算结果中，低半字节（最低4位）向高半字节（即D3向D4）的进位情况。

  若D3向D4有进位或借位，AF=1；否则，AF=0

  只有在执行十进制运算指令时才关心此位



FLAGS寄存器

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

8088/8086

- PF(Parity Flag) 奇偶标志：由运算结果自动设置

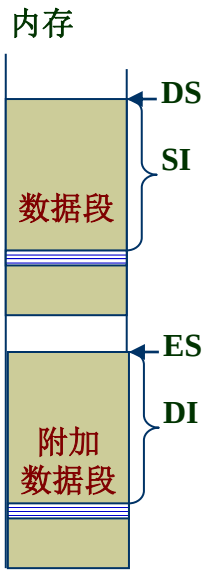
  若运算结果的低8位中，“1”的个数为偶数，则PF=1；否则，PF=0

0	1	1	0	1	0	1	0
7	6	5	4	3	2	1	0

- DF(Direction Flag)方向标志：由指令设置

  用于控制串操作，指示串操作时操作数地址的增减方向

  - DF为1时，串操作后使变址寄存器SI、DI自动减量
  - DF为0时，串操作后，使SI、DI自动增量



FLAGS寄存器

8088/8086

- IF(Interrupt Flag)中断标志：由指令设置

  IF只对外部可屏蔽中断请求（INTR）起作用

  若IF＝1，允许CPU响应INTR

  若IF＝0，　禁止响应INTR

- TF(Trap Flag)陷阱标志：由指令设置

  用于程序调试

  若TF=1，CPU处于单步运行方式

  若TF=0，CPU处于正常工作方式

标志位分类

➢ 条件（状态）标志

OF、SF、ZF、AF、CF和PF，其值取决于一个操作完成后，算数逻辑部件ALU所处的状态

➢ 控制标志和系统标志

DF、IF和TF，其值是通过指令人为设置的，用以控制程序的执行

Debug下演示标志位符号

```
C:\>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0100   NV UP EI PL NZ NA PO NC
0B39:0100 40            INC     AX
-r ax
AX 0000
:1111
-r
AX=1111  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0100   NV UP EI PL NZ NA PO NC
0B39:0100 40            INC     AX
```

第几位	标志	描述	类别	=1（符号）	=0（符号）
0	CF	Carry flag	状态标志	CY(Carry)	NC(No Carry)
2	PF	Parity flag	状态标志	PE(Parity Even)	PO(Parity Odd)
4	AF	Adjust flag	状态标志	AC(Auxiliary Carry)	NA(No Auxiliary Carry)
6	ZF	Zero flag	状态标志	ZR(Zero)	NZ(Not Zero)
7	SF	Sign flag	状态标志	NG(Negative)	PL(Positive)
8	TF	Trap flag (single step)	控制标志	——	——
9	IF	Interrupt enable flag	控制标志	EI (Enable Interrupt)	DI(Disable Interrupt)
10	DF	Direction flag	控制标志	DN(Down)	UP(Up)
11	OF	Overflow flag	状态标志	OV(Overflow)	NV(Not Overflow)

程序调试时，将标志位1或0用符号表示，便于阅读、理解

例： MOV AX, 1

MOV BX, 2

ADD AX, BX

指令执行后，AX=3, OF=0, CF=0, ZF=0, SF=0

例： MOV AX, FFFFH

MOV BX, 1

ADD AX, BX

指令执行后，AX=0, OF=0, CF=1, ZF=1, SF=0

2.3.3 ARM处理器的寄存器组

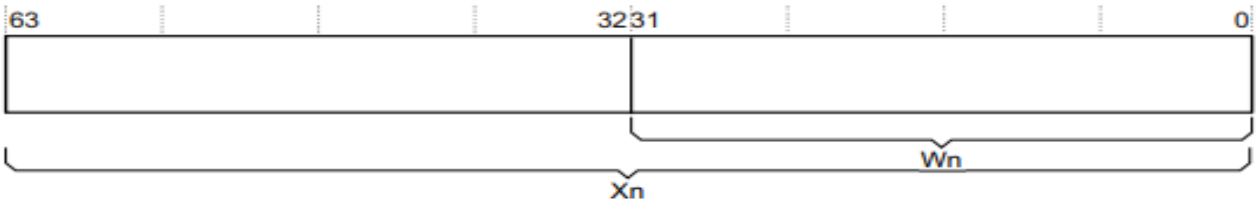
- 2011年发布了ARMv8 64位架构，有两种执行状态，AArch32和AArch64
  - 在运行中可以无缝地在32位和64位代码两种模式间切换
- ARM64位AArch64处理器寄存器类型
  - 31个通用寄存器（X0~X30）
  - 专用寄存器

31个通用寄存器

X0/W0
X1/W1
X2/W2
X3/W3
X4/W4
X5/W5
X6/W6
X7/W7
X8/W8
X9/W9
X10/W10
X11/W11
X12/W12
X13/W13
X14/W14
X15/W15
X16/W16
X17/W17
X18/W18
X19/W19
X20/W20
X21/W21
X22/W22
X23/W23
X24/W24
X25/W25
X26/W26
X27/W27
X28/W28
X29/W29
X30/W30

FP

LR

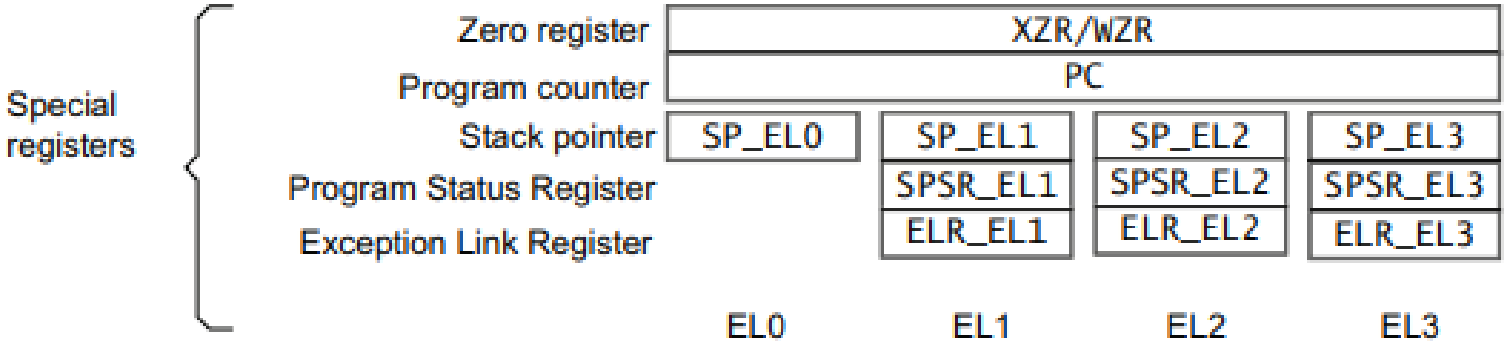


- AArch64执行状态时提供31个64位的通用寄存器，可在任何时间和异常级别访问
  - 64位寄存器 X0-X30，32位形式对应W0-W30
  - 32位Wn寄存器是64位Xn寄存器低半部分
- 从Wn寄存器读取数据时，忽略相应Xn寄存器的高32位并保持不变
- 写入Wn寄存器会将Xn寄存器的高32位设置为零

  eg：写入 0xFFFFFFFF时，W0将X0寄存器设置为 0x00000000FFFFFFFF

专用寄存器

- 零寄存器ZR（Zero register）：用作源数据寄存器时读为零，用作目标数据寄存器时丢弃结果（用于优化程序性能）
- 程序计数寄存器PC(Program counter)：机器指令地址指针
- 堆栈指针寄存器SP(Stack pointer)：堆栈指针
- 程序状态寄存器PSR(Program Status Register)：处理器状态位、控制位、程序模式
- 异常链接寄存器ELR(Exception Link Register)：保存异常（中断）处理完后的返回地址



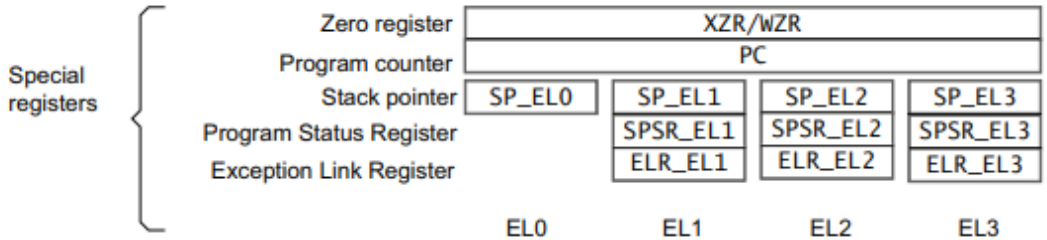
堆栈指针（SP）

- AAarch64架构中，堆栈指针选择与异常级别分离
  - 默认情况下，发生异常会选择目标异常级别的64位堆栈指针SP_EL*n*。例如，EL1选择SP_EL1
  - EL0 只能访问 SP_EL0
  - EL0以外异常级别时，可以使用与该异常级别关联的其它堆栈指针SP_ELn
- 大多数指令都不能使用SP。只有某些形式的算术指令，如ADD指令，可以读写当前堆栈指针来调整堆栈指针

```
·ADD SP, SP, #256 // SP=SP+256
```

Exception level	Options
EL0	EL0*t*
EL1	EL1*t*, EL1*h*
EL2	EL2*t*, EL2*h*
EL3	EL3*t*, EL3*h*

- t后缀表示SP_EL0堆栈指针被选择，
- h后缀表示SP_ELn堆栈指针被选择。



Arm中使用了许多与传统CPU不同的概念和设计思想

处理器状态

- 在AArch64中，传统的处理器状态字段可以被独立访问，它们的集合统称为处理器状态(PSTATE, Processor State)
  - 条件标志：N，Z，C，V
  - 中断掩码：D，A，I，F
  - 执行状态：nRW（32位/64位）
  - **堆栈指针**：SP（SP_EL0/SP_ELn）
  - 软件单步：SS（Software Step）
  - **非法执行**：IL
- PSTATE所有字段都可以在EL1或更高异常级别模式访问，但**在EL0中只能访问{N，Z，C，V}字段**。

名称	描述
N	负状态标志
Z	零条件标志
C	进位状态标志
V	溢出状态标志
D/A/I/F	异常屏蔽位
SS	软件单步位
IL	非法执行状态位
EL (2)	异常级。
nRW	执行状态 0 = 64 位 1 = 32 位
SP	堆栈指针选择器 0 = SP_EL0 1 = SP_ELn

程序状态寄存器

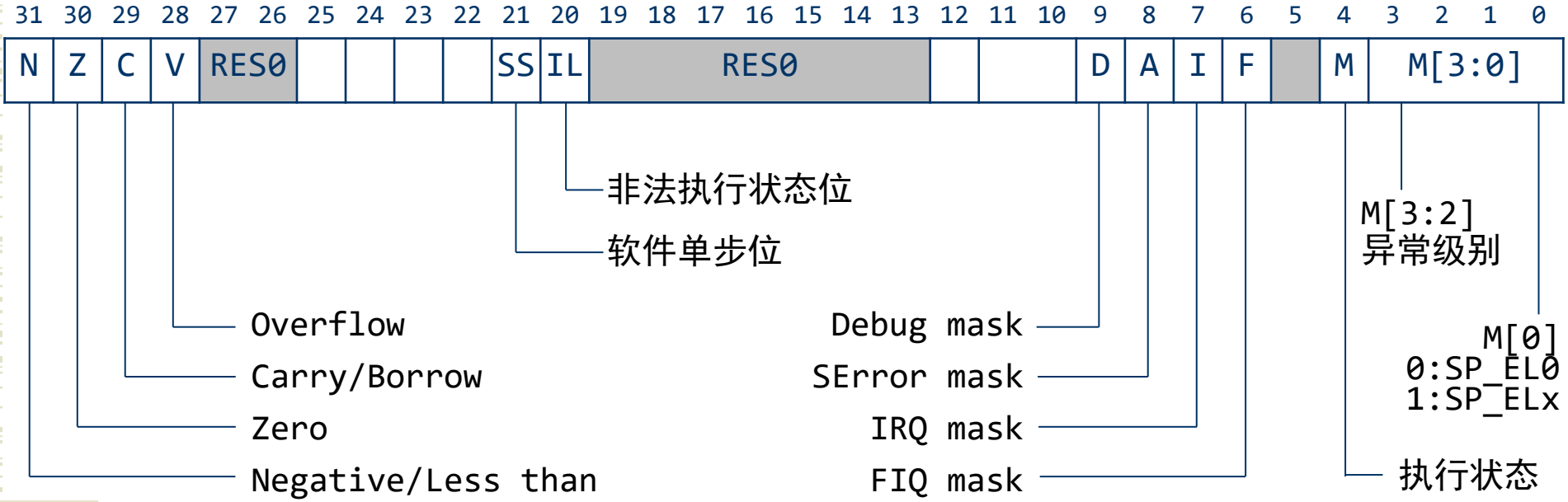
标志位	含义
N	负数标志位：两个补码带符号数运算时，N=1结果为负数，N=0结果为正数或零
Z	零标志位：Z=1运算结果为0，Z=0运算结果不为0
C	进位标志位： • 加法指令(包括CMN)：C=1运算结果有进位（无符号数溢出），C=0运算结果无进位 • 减法指令(包括CMP)：C=0运算有借位（无符号数溢出），C=1无借位 • 移位操作指令，C为移出值的最后一位 • 其他非加/减运算指令,C一般不受影响
V	溢出标志位： 1．加/减法运算指令，当操作数和运算结果为二进制补码表示的有符号数时，V=1表示符号位溢出 2．其他的非加/减法运算指令，V通常不改变
I/F/M/M[3:0]	控制位：当发生异常处理时，这些位（低8位）的值将发生相应的变化。在特权模式下,也可以通过软件来修改这些位

程序状态寄存器

- SPSR（Saved Program Status Register）：
  - 发生异常或中断请求处理时， PSTATE会被自动保存在SPSR
  - 异常返回时，SPSR用来恢复PSTATE

```
//Copies X0 to SPSR_EL1
·MSR SPSR_EL1, X0
```

- 程序状态寄存器各个位含义：



2.4 存储器

➢ 存储器是用来存放程序、数据、中间结果和最终结果的记忆装置

2.4.1 存储单元的地址和内容（☆）

- 计算机存储信息的基本单位是一个二进制位
- 8086字长为16位，地址长度20位
- 80386以上机的字长为32位，地址长度32位以上

➢16位二进制数能表示的地址？
➢20位地址如何表示？
➢存储单元中的地址和内容？

存储器地址和内容示意图

地址	内容
00000H	
00001H	
00002H	
00003H	
…	…
03080H	78H
03081H	56H
03082H	34H
03083H	12H
…	…
FFFFFH	

- 存储器以字节(8bit)为单位存储信息(数据等)
- 每个字节单元有一个地址

  从0编号，顺序加1
- 地址也用二进制数表示

  无符号整数，写成十六进制
- 16位二进制数可表示2^{16}=64K个地址

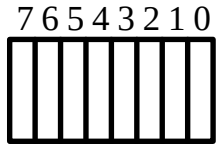
  0000H ～ FFFFH,
- 字长16位，一个字要占用相继的两个字节
  - 低位字节存入低地址，高位字节存入高地址
  - 以偶地址访问（读/写）存储器
  - 字单元地址用它的低地址来表示

    (03080H)=5678H

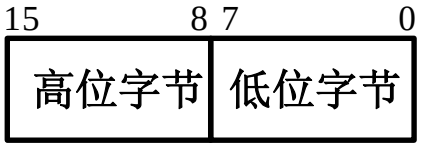
    ((0004H))=2F1EH （书P25）
- 双字长32位时？

  (03080H)=12345678H

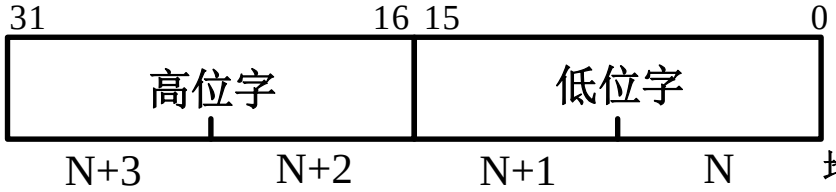
- 存储器仅是按地址以字节为单位存储二进制编码的器件。
  - 如何组织管理由具体CPU决定；
  - 这些二进制编码是程序、数据？ 由对它们的具体操作确定
- 指令如何表示存储单元的地址（虚拟地址、逻辑地址）和内容，
- CPU如何将虚拟地址转换为20位的物理地址读写数据，实现分段管理？



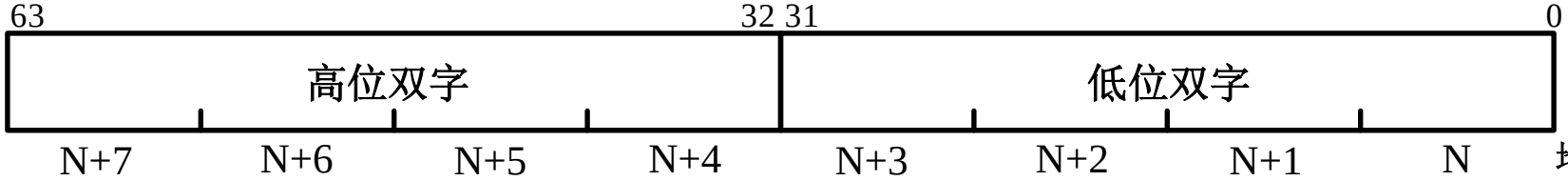
(1) 字节



(2) 字



(3) 双字



(4) 4字

例：存储器单元地址和内容

字节

7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	1
0	0	1	0	0	1	1	0
0	0	0	1	1	1	1	0
1	1	0	1	0	1	1	1

00000H (00000H)=9FH

00001H (00001H)=26H

00002H (00002H)=1EH

00003H (00003H)=D7H

(00000H)=269FH

(00002H)=D71EH

读写1个字，需要访问两次存储器

字

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1

高位字节 **低位字节**

存储系统中的存储部件层次与关系

每级存储器的性能参数可以表示为T_i，S_i，C_i。存储系统的性能可表示为：$T_i<T_{i+1}$；$S_i<S_{i+1}$；$C_i>C_{i+1}$。

X86机存储器管理方式

- X86机的存储器逻辑上采用分段管理的方法
  - X86CPU的相关地址寄存器16位，可管理访问2^{16}=64K空间
  - X86机的存储器物理地址20位，可使用空间2^{20}=1024K=1M

“...程序员在编制程序时要把存储器划分成段...”

- 存储器采用分段管理后，一个内存单元地址要用段基地址和偏移量两个逻辑地址来描述，表示为:

段基址:偏移量

段基址和偏移量的限定、物理地址的形成视CPU工作模式决定

2.4.2 实模式存储器寻址（☆）

➢ 存储器地址的分段

- ✔ 存储器有20根地址线　2^{20}=1024K=1M=1048576

  地址范围　00000H ～ FFFFFH

- ✔ 每段最大2^{16}=64K空间
- ✔ 小段：每16个字节为一小段，共有64K个小段

小段的首地址

⇩

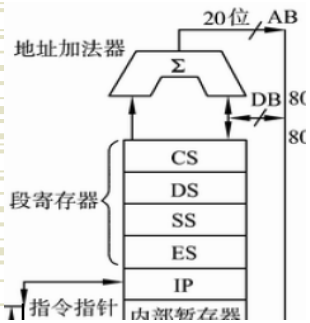
00000H ～ 0000FH
00010H ～ 0001FH
00020H ～ 0002FH
…
FFFF0H ～ FFFFFH

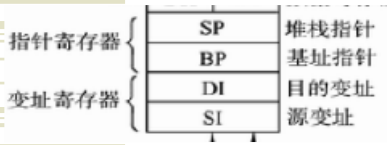
- 存储器分段：段起始地址必须是某一小段的首地址，
  段的大小可以是64K范围内的任意字节

物理地址：每个存储单元的唯一的20位地址

段基地址：段起始地址(20位)=10H × 段寄存器(16位)

偏移地址：段内相对于段起始地址的偏移量（16位），
偏移量又称为**有效地址（EA）**





物理地址 = 16d × 段寄存器 + 偏移地址

= 10H × 段寄存器 + 偏移地址

16 位 段 地 址 0000

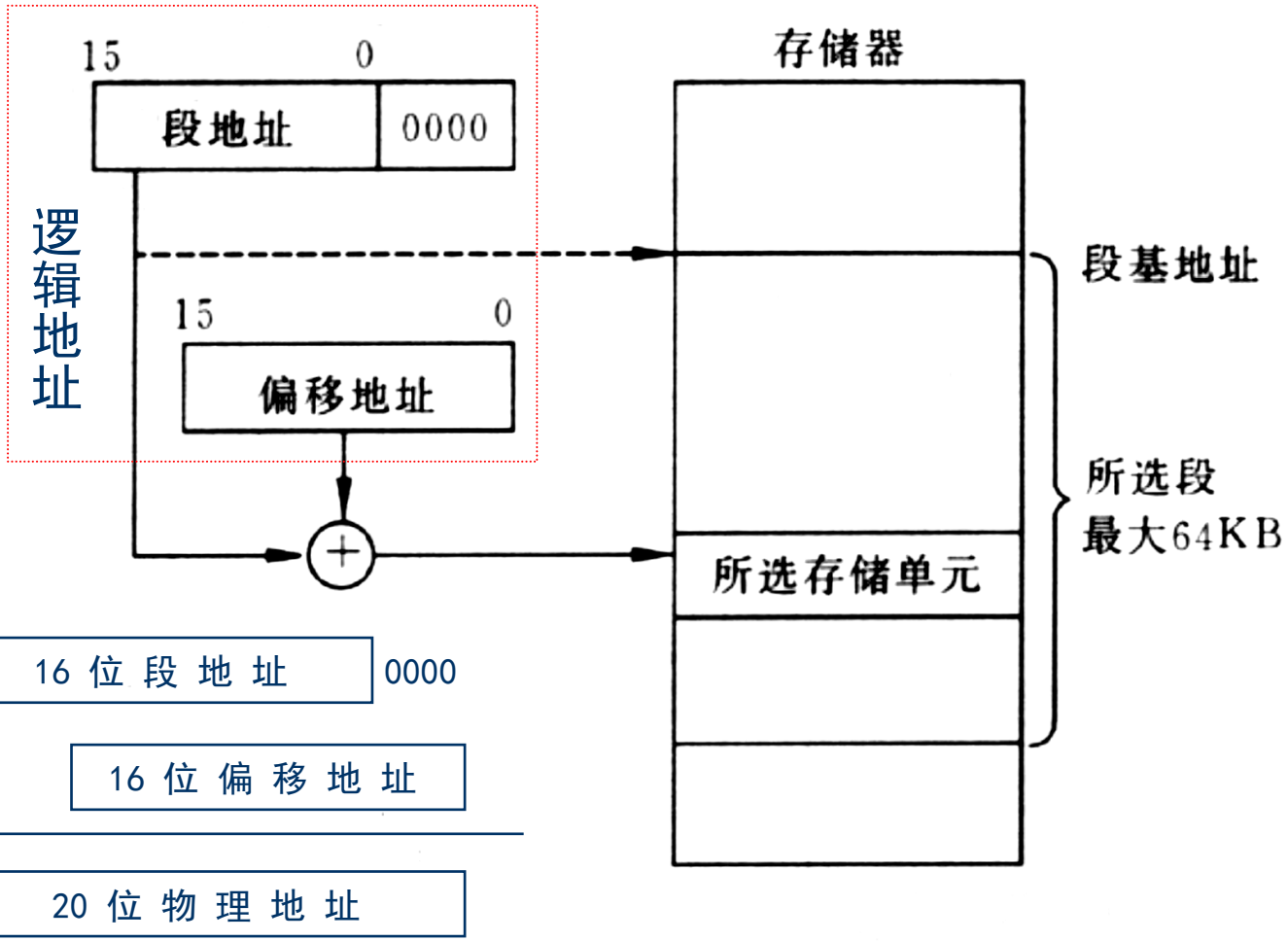
\+

16 位 偏 移 地 址

20 位 物 理 地 址

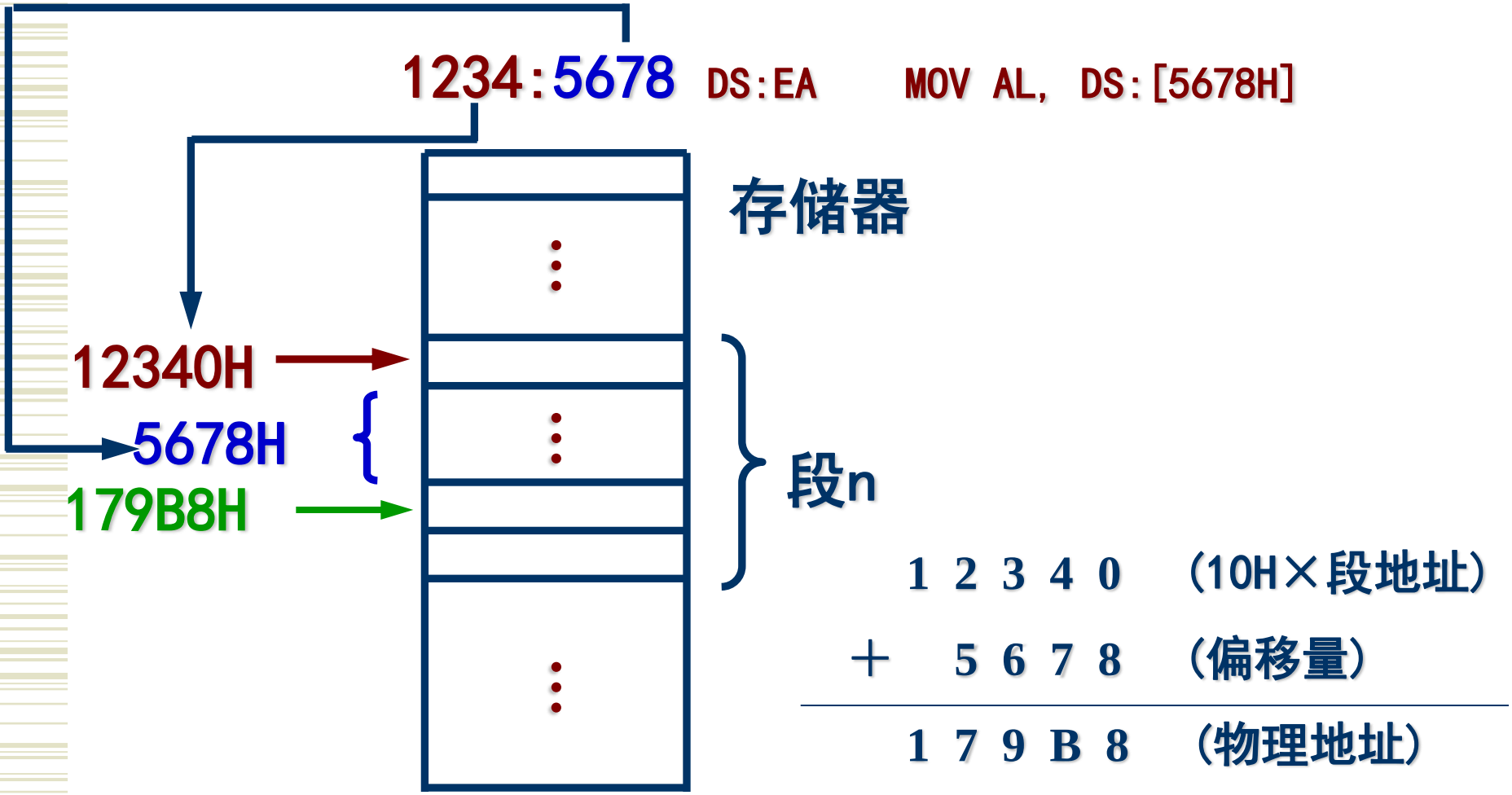
再看看小段、段寄存器、与地址有关寄存器长度想想有什么关系？

为什么段起始地址必须是某一小段的首地址？

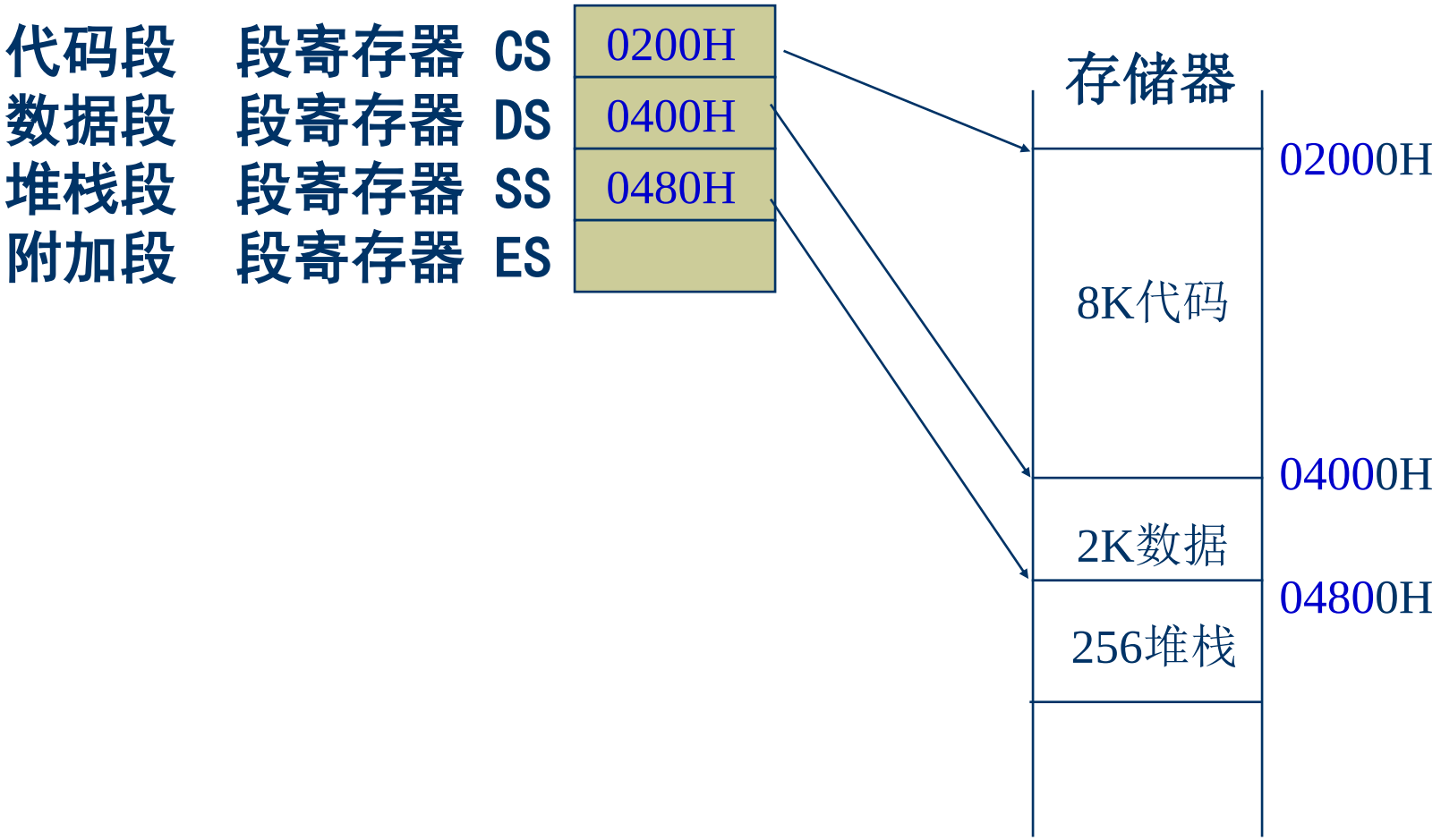


存储单元中的物理地址是唯一的
指令中给出的逻辑地址各种各样

例：某内存单元的地址用十六进制数表示为1234:5678，则其物理地址为12340H＋5678H＝179B8H。

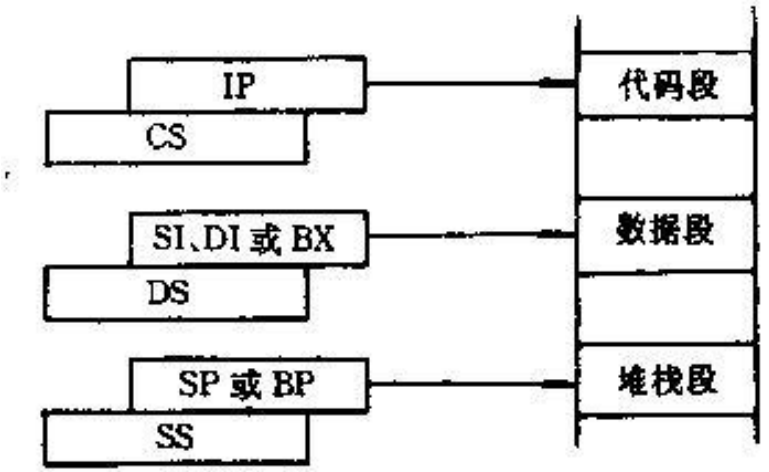


- X86处理器中有4个专门存放段地址的段寄存器（16位）



- 注意段寄存器与偏移地址的缺省组合(P28)及段地址的用途

```
MOV AL, [5678H] = MOV AL, DS:[5678H]
MOV AL, [SI] = MOV AL, DS:[SI]
MOV AL, [SP] = MOV AL, SS:[SP]
```

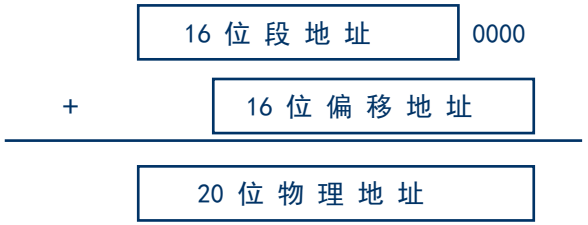


- 汇编指令中给出逻辑地址

  段地址：偏移地址

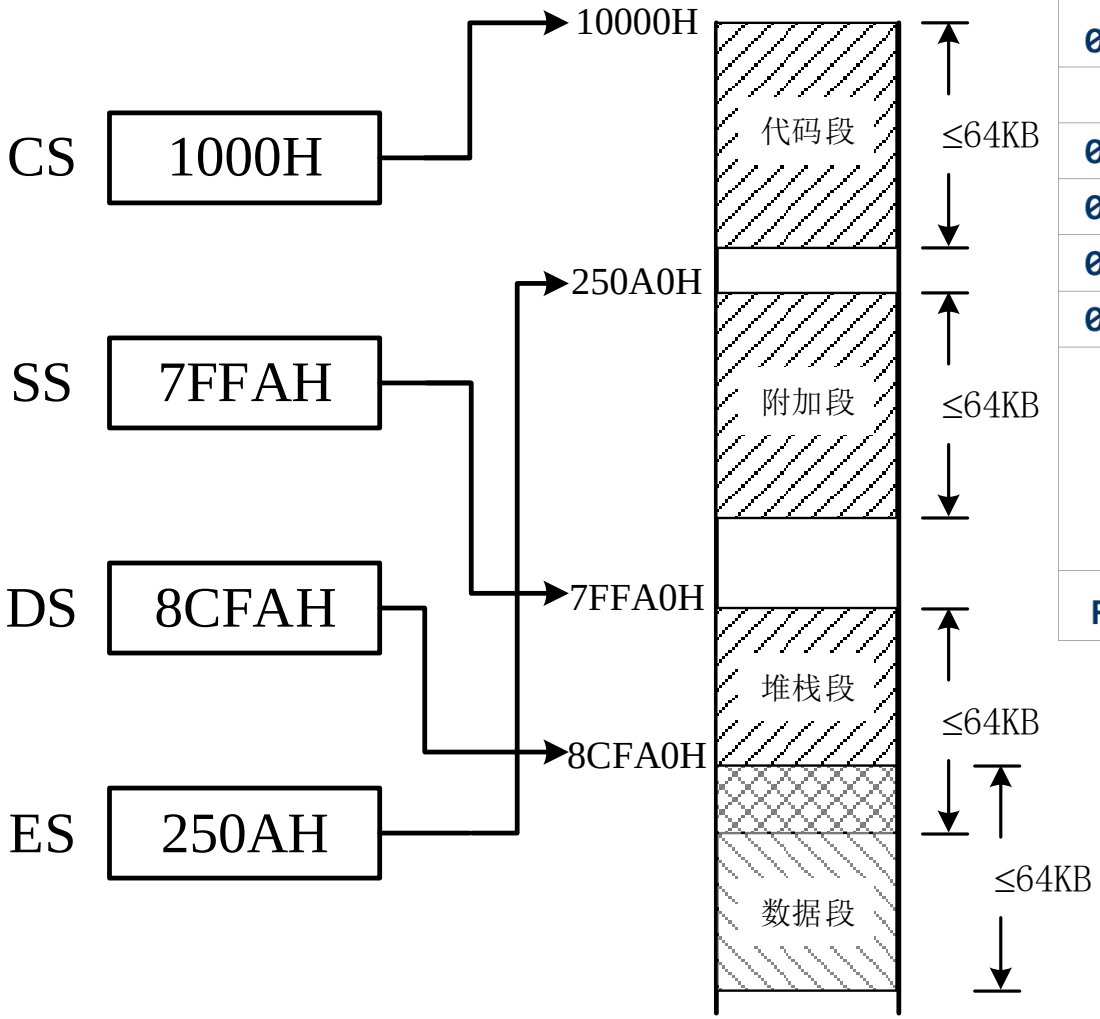
- 机器自动计算物理地址

  实模式与保护模式计算方法不一样



- 逻辑地址：机器指令给出的地址

- 各段之间关系：
  - 相邻
  - 相离
  - 重叠



地址	内容
00000H	
00001H	
00002H	
00003H	
…	…
03080H	78H
03081H	56H
03082H	34H
03083H	12H
…	…
FFFFFH	

以字节为单位编址

2.4.3 保护模式

- 80386以后的微机（80386、80486和Pentium）在性能上比8086、80286有了质的飞跃。它们不仅支持实模式，而且支持保护模式。
- 在实模式下，80386只相当于一台高速8086，编程方法与8086相似
- 只有在保护模式下，才能发挥80386的真正作用。在保护模式下，80386可寻址物理地址空间高达2^{32}=4G，支持2^{48}=64T的虚拟存储器，支持多任务和保护机制。

引出保护模式的存储器寻址，主要原因如下：

1. 解决如何寻址的问题（4GB或更多的地址空间）
2. 多用户共享cpu和mm，使微机系统能支持多任务

微机广泛使用要求系统能提供多任务处理功能，即多个应用程序能在同一台计算机上同时运行，而且它们之间必须相互隔离，使一个应用程序中的缺陷和故障不会破坏系统，也不会影响其他应用程序的运行。防止一个进程以非法方式侵权访问存储区域

3. 在系统支持多任务功能的同时，系统也支持了虚拟存储器特性。虚拟存储器可支持程序员编写的程序具有比主存储器所能提供的更大的空间

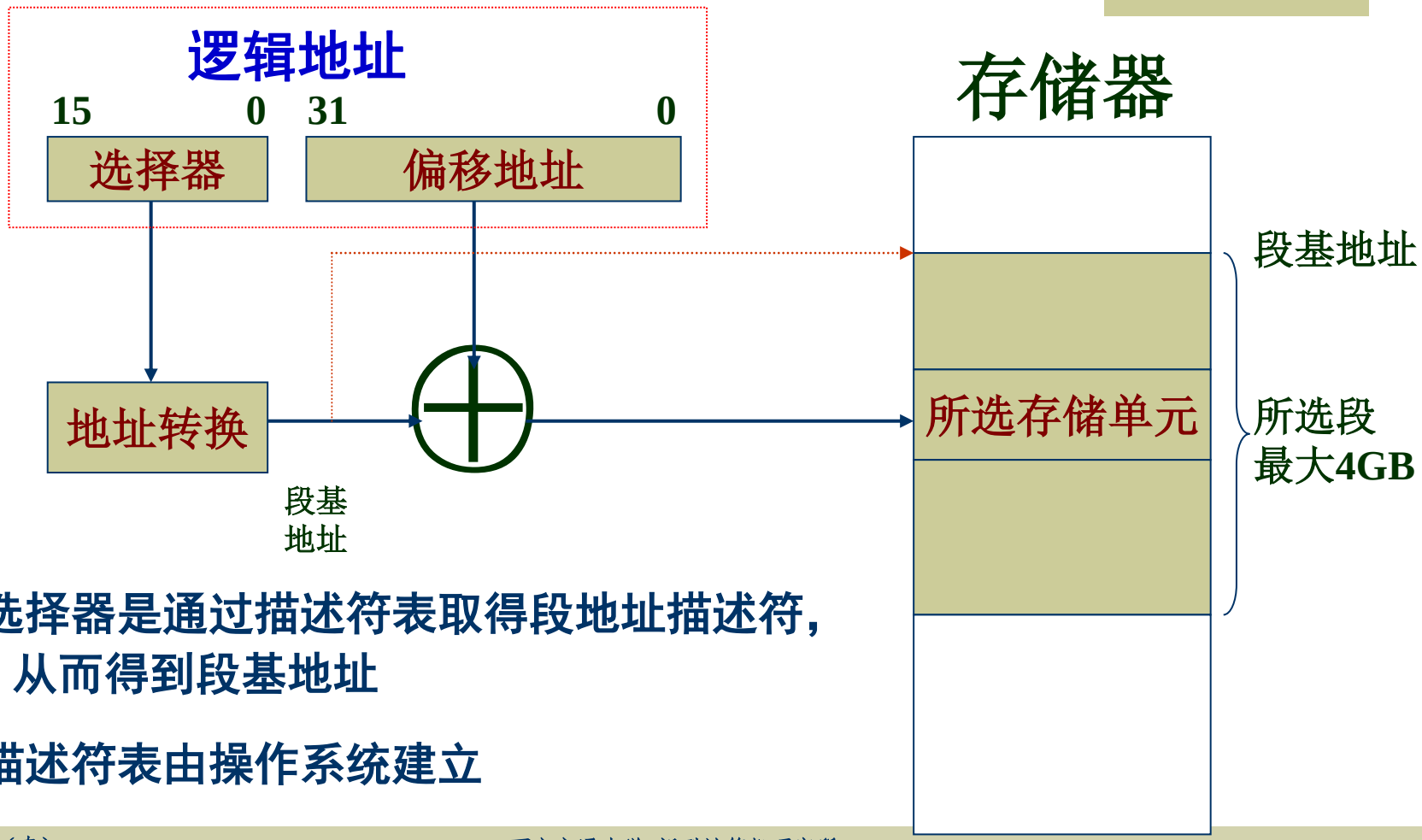
保护模式下存储器寻址

- 讨论保护模式下如何实现存储器寻址，主要内容如下：
  1. 逻辑地址
  2. 描述符
  3. 选择器和描述符表
  4. 程序不可见寄存器

1. 逻辑地址

- 在保护模式存储器寻址中，仍然要求程序员在程序中指定逻辑地址，只是机器采用另一种比较复杂的或者说间接的方法来求得相应的物理地址
- 在保护模式下，逻辑地址由选择器和偏移地址两部分组成
  - 选择器存放在段寄存器中，但它不能直接表示段基地址，而由控制器通过一定的方法取得段基地址，再和偏移量相加，从而求得所选存储单元的物理地址
  - 段基地址由操作系统从磁盘装入指令代码和数据段时自动分配设置，存放在存储器内的描述符表中

保护模式存储器寻址过程



- ✓ 选择器是通过描述符表取得段地址描述符，从而得到段基地址
- ✓ 描述符表由操作系统建立

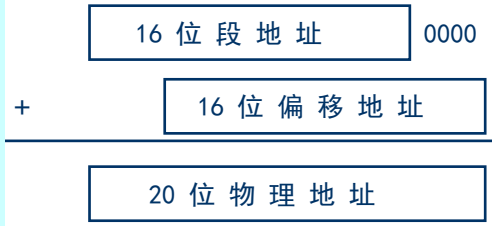
实模式下：

指令给出的逻辑地址=段寄存器：偏移地址

段寄存器左移4位，获得20位段起始地址

加上段内偏移地址

获得20位存储单元物理地址



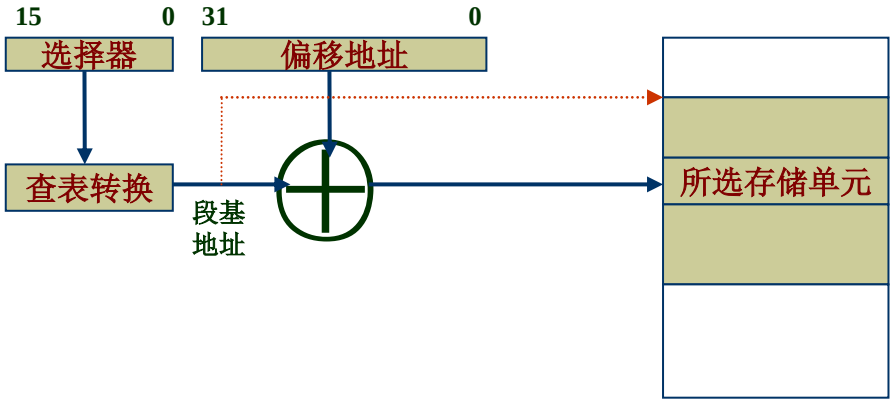
保护模式下：

指令给出的逻辑地址=段寄存器：偏移地址

查段描述符表，获得24、32位段起始地址

加上段内偏移地址

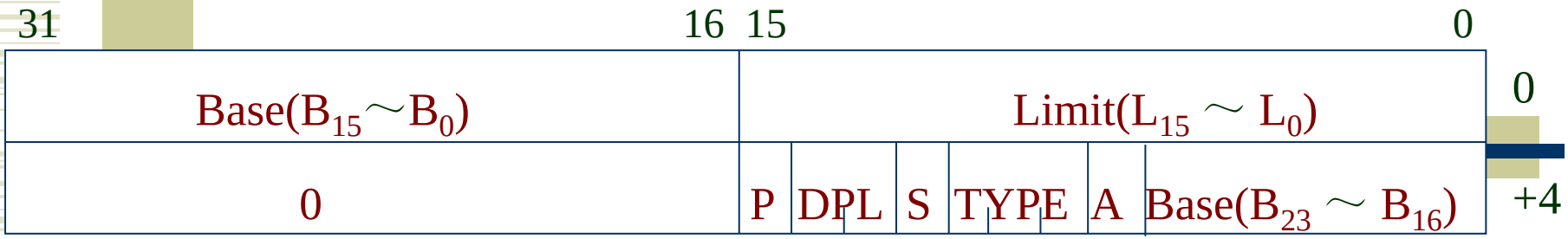
获得24、32位存储单元物理地址



2. 描述符（段的描述符）

- 描述符长度和组成
  - 8个字节
  - 由段基地址、界限、访问权限、附加字段4部分组成
- 用途：用来说明段在存储器中的位置、段的大小、控制和状态信息
- 描述符存放在存储器中，由操作系统从磁盘装入指令代码和数据段时自动分配设置，另外操作系统还要修改选择器内容

80286描述符



Base：段基地址，24位；Limit：段长度，2^{16}=64KB；
P：存在位；DPL：特权级（0-3）；　　S：1 系统段，0 应用程序段；
TYPE（E,ED/C,W/R）：E 可执行位，E=0不可执行段（数据段），E=1可执行代码段；ED/C 地址扩展方向位，ED=0，向上扩展，ED=1，向下扩展，W/R 可读/写位；
A：已访问位

80386/80486/Pentium描述符

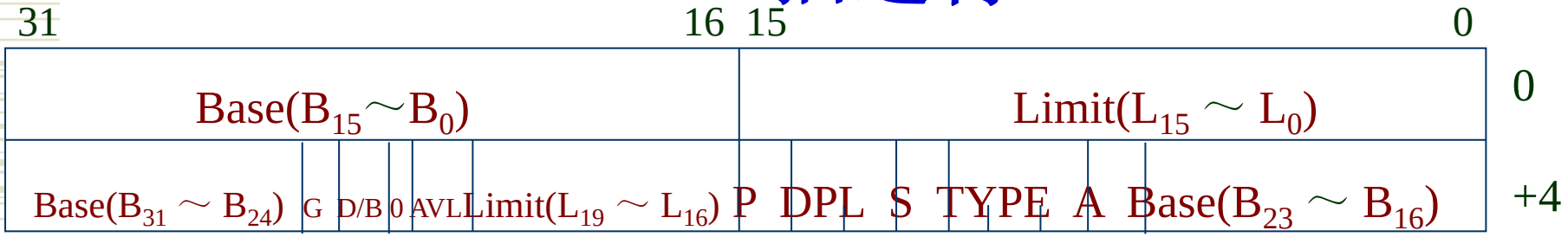
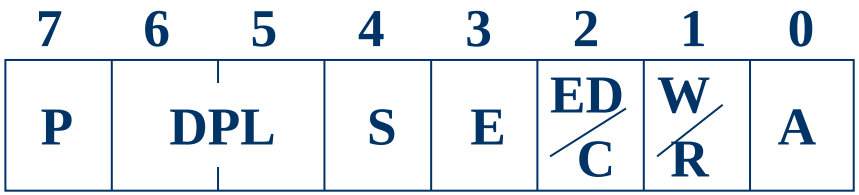


图2.11　描述符格式

访问权限字节



P位：存在位

P=0，段不在内存

P=1，段在内存中

DPL：段特权级

取值0~3，允许访问该段的最低特权级

S位：段描述符

S=1 系统段

S=0 应用程序段

A位：已访问位

A=0，段尚未被访问

A=1，段已被访问

E位：可执行位

E=0，不可执行代码段(数据段)

ED=0，段向上扩展为数据段

ED=1，段向下扩展为堆栈段

W=0，数据段只读

W=1，数据段可写

E=1，可执行代码段(代码段)

C=0，忽略描述符特权级

C=1，遵循描述符特权级

R=0，代码段不可读，即只执行

R=1，代码段可读

G位(粒度位)：

G=0, 段的长度以字节为单位

段长最大1M字节

G=1,段的长度以页(4K字节)为长度单位

段长最大1M×4K=4G字节

D位：D=0，16位指令方式

D=1，32位指令方式

AVL位：AVL=0，程序不可使用本段

AVL=1，程序可以使用本段

3. 选择器和描述符表

选择器存放在段寄存器中，16位长，格式位：

INDEX：所选描述符在描述符表中的偏移地址；

TI：指定描述符表

TI=0 从全局描述符表 GDT 中取描述符

TI=1 从局部描述符表 LDT 中取描述符

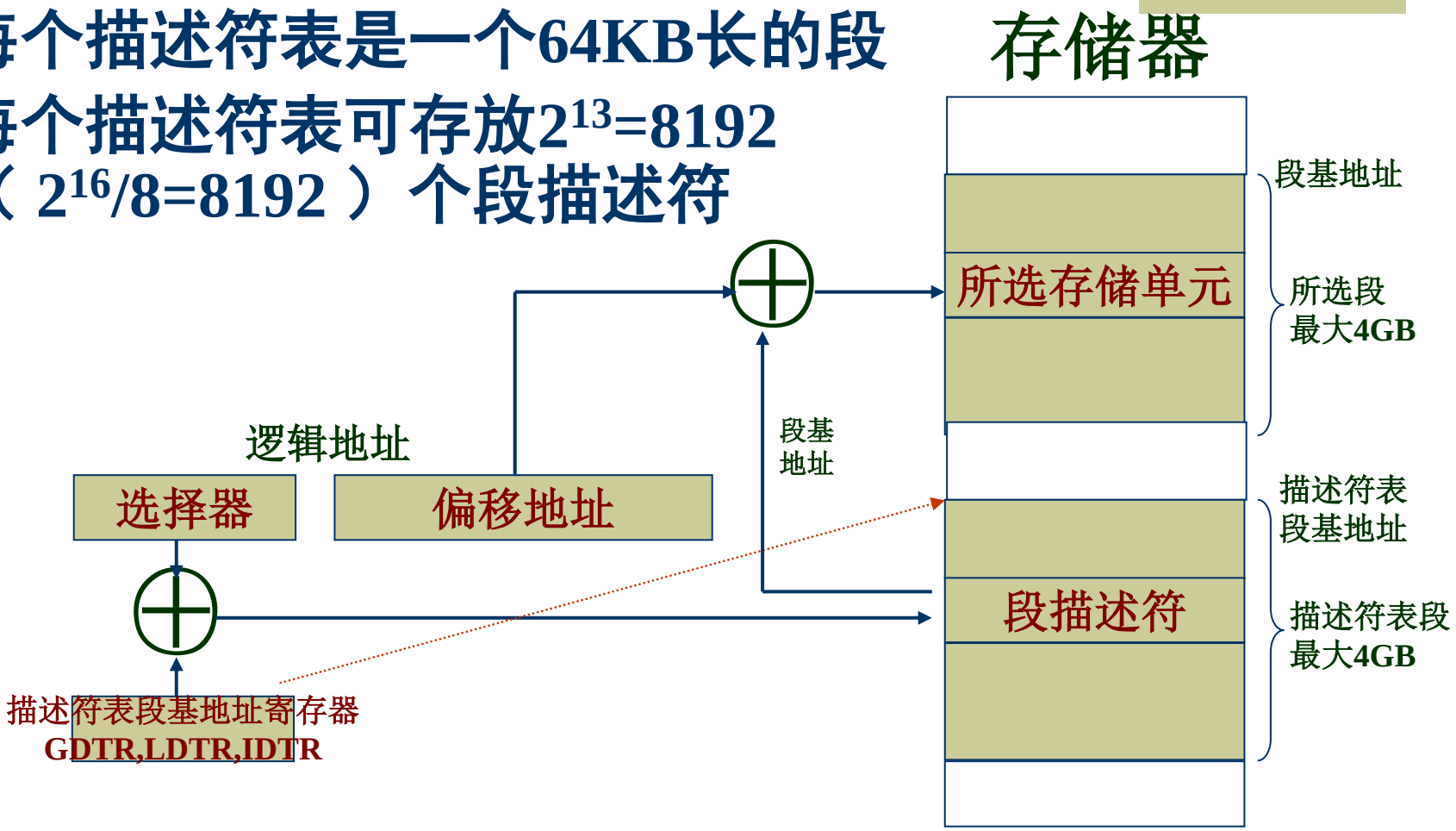
RPL：请求访问特权级，RPL≤DPL 才可以访问该段，取到描述符

描述符在描述符表中，一共有3类描述符表：

- 全局描述符表（GDT）【整个系统只有一个】
  - 其中的描述符指定的段可以用于所有的程序（如操作系统所用段）
- 局部描述符表（LDT） 【可以有多个】
  - 所指定的段通常只用于一个用户程序
- 中断描述符表（IDT） 【整个系统只有一个】

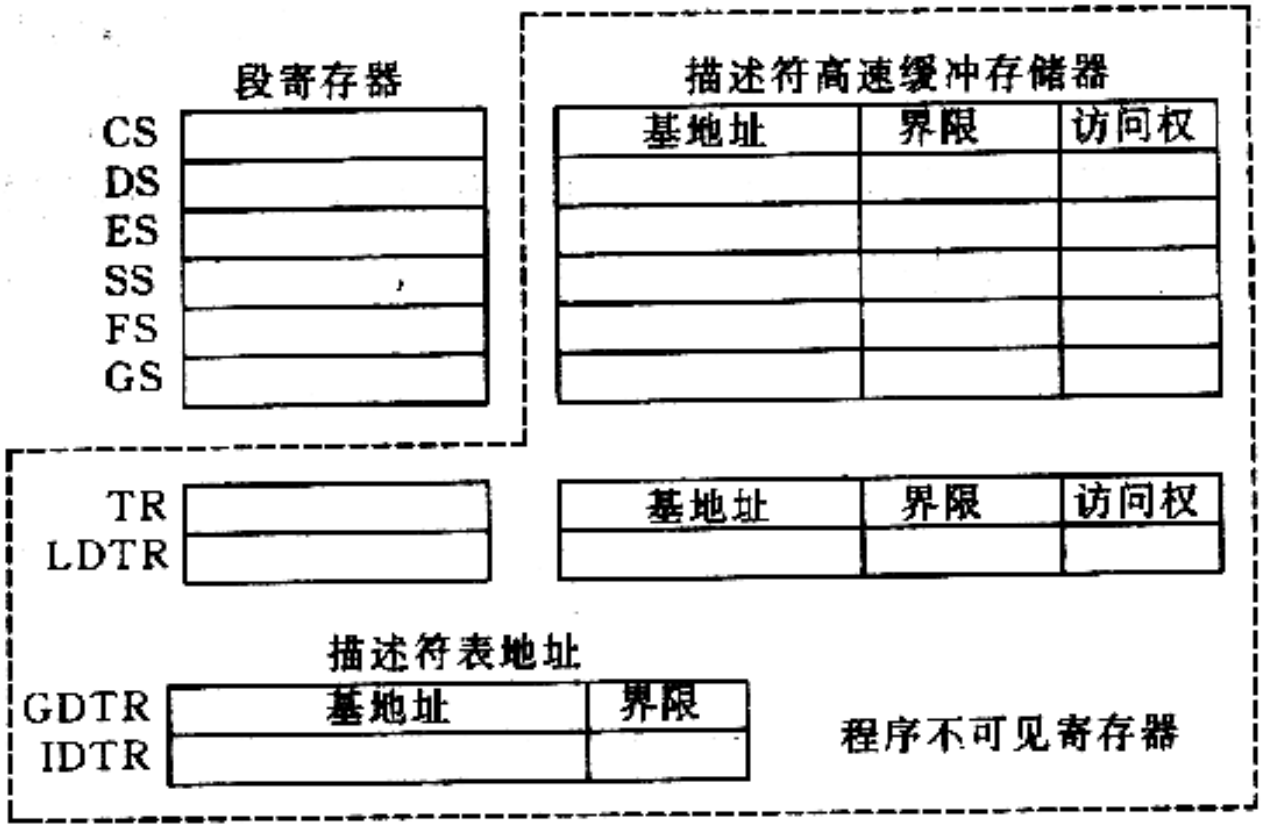
保护模式存储器寻址过程

- 描述符表存放在存储器中
- 每个描述符表是一个64KB长的段
- 每个描述符表可存放2^{13}=8192（$2^{16}/8$=8192）个段描述符



4. 程序不可见寄存器

- 指不能由用户程序访问而是只能由操作系统或硬件管理的寄存器
  - 如 GDTR, LDTR, IDTR 等



5. 描述符高速缓冲存储器

- 保护模式存储器寻址过程中多次访问主存，效率很低
  - 解决办法，采用cache原理，将常用（正在使用段）的描述符放在描述符高速缓冲存储器中
  - 描述符高速缓冲存储器在CPU中，访问速度与寄存器一样
  - 描述符高速缓冲存储器容量很小

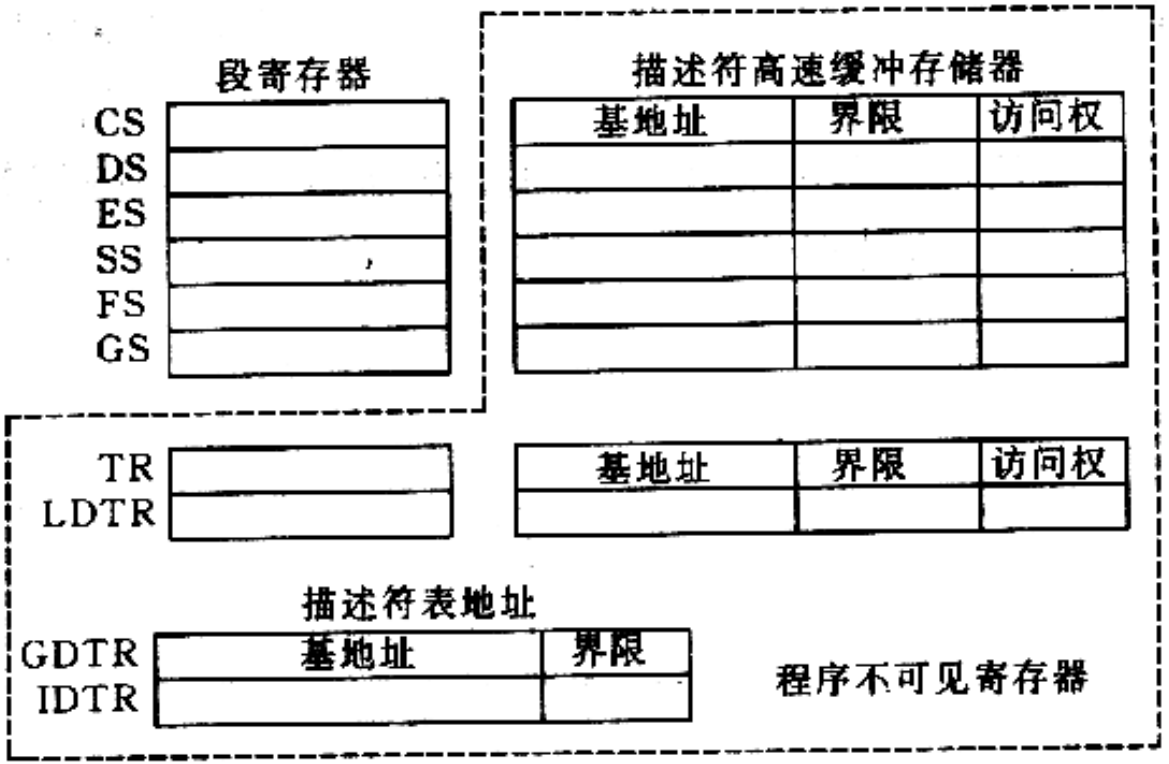


图 2.13　80x86 的程序不可见寄存器

2.4.2 ARM存储单元的地址和内容

- ARM体系结构中，字节的长度是8位，半字的长度是16位，字的长度为32位。
- ARM信息存储有大端格式和小端格式两种
  - 大部分采用小端模式，少数情况采用大端模式
  - 大端模式：数据高位保存在内存低地址单元，数据低位保存在内存高地址单元。
  - 小端模式：数据高位保存在内存高地址单元，数据低位保存在内存低地址单元。

X86采用小端模式

大端模式

地址	内容
4000	78
4001	56
4002	34
4003	12

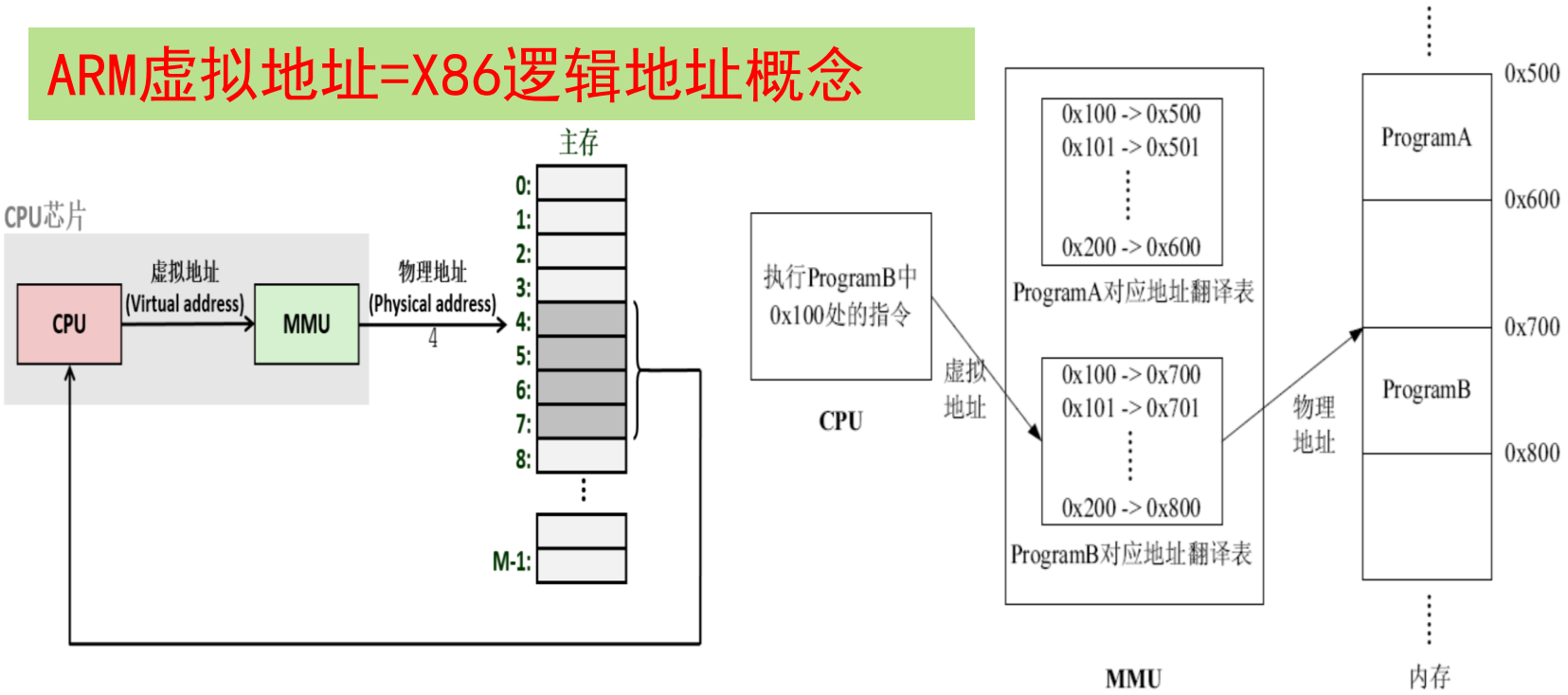
小端模式

地址	内容
4000	12
4001	34
4002	56
4003	78

ARM内存管理由MMU完成

MMU(Memory Management Unit)负责虚拟地址映射为物理地址，以及提供硬件机制的内存访问授权、多任务多进程操作系统支持

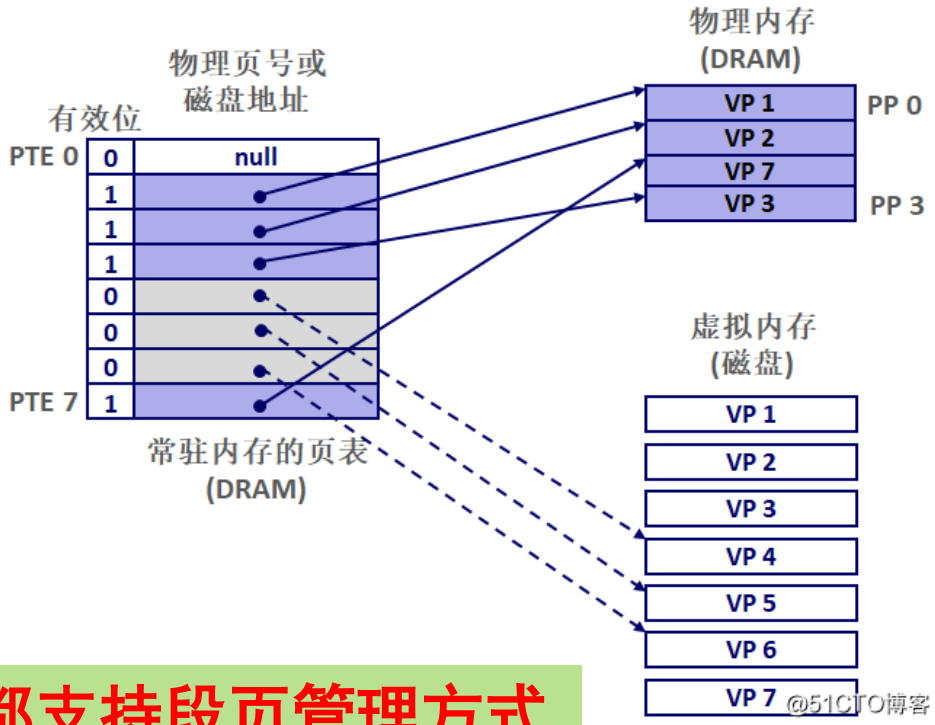
ARM虚拟地址=X86逻辑地址概念



ARM存储器管理方式

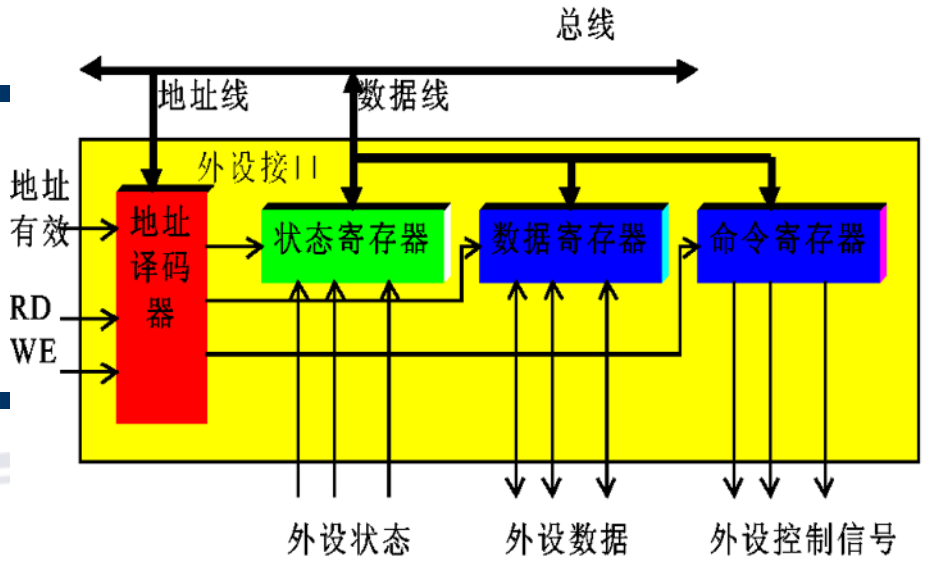
AArch64架构处理器根据不同应用特点，支持不同的存储粒度，包括：64KB、16KB和4KB。常用的是4KB页面。

虚拟地址及其对应的物理地址存储在页表中，每个页表项存储了一段虚拟地址对应的物理地址及其访问权限，或者下级页表的地址。



Intel的80386以后处理器都支持段页管理方式

2.5 外部设备



2.5.1 外部设备简介

输入、输出设备，大容量的外存储器。

外部设备

2.5.2 外设接口寄存器

- CPU对外部设备控制通过外设接口寄存器
  1. 数据寄存器：数据传送
  2. 状态寄存器：查看外部设备状态
  3. 命令寄存器：控制外部设备工作
- 外设接口寄存器访问（编址方式）

  单独编址：用专用指令访问，如output, input

  统一编址：用访存指令，如mov等

硬件设计好后，CPU对接口寄存器按地址读/写就可以对外设控制/数据传送

作 业

思源学堂→课后作业→第1次课后作业

要求：

（1）独立完成，严禁抄袭

（2）以PDF文档在思源学堂提交

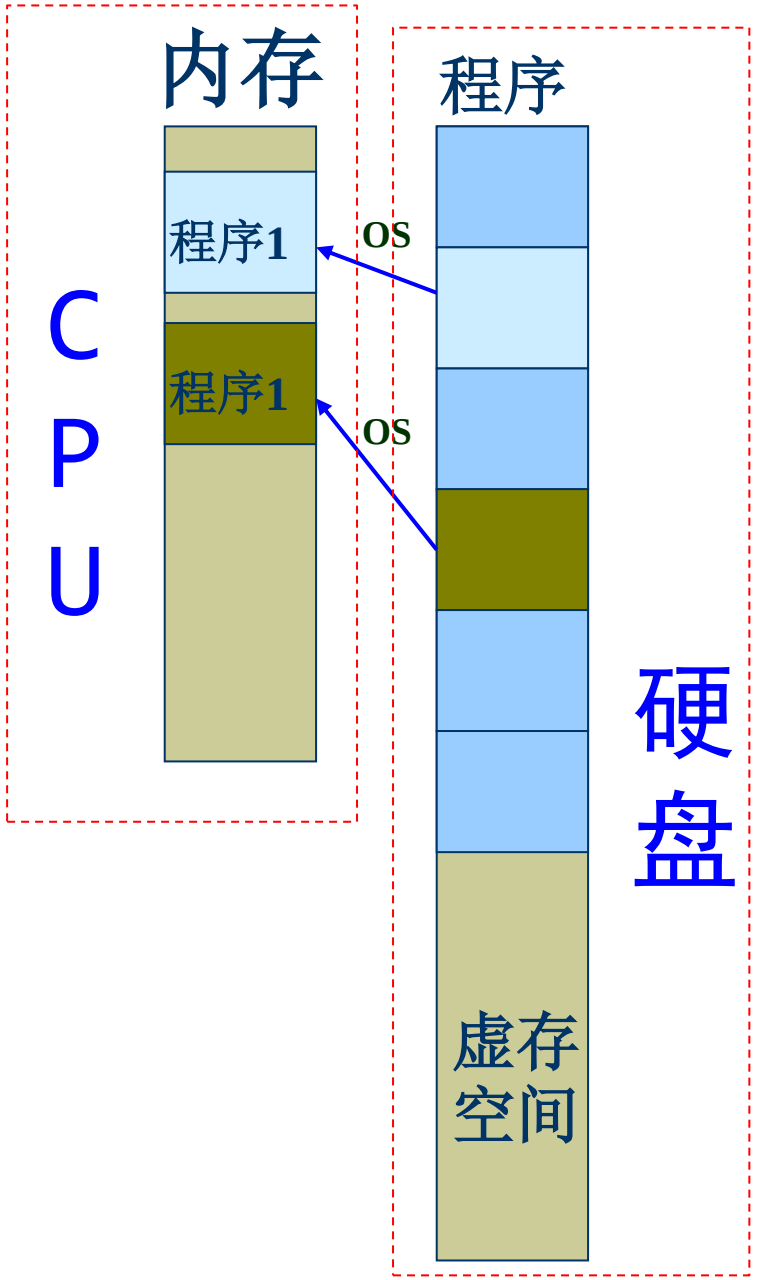
（3）截至日期：3月19日23:59

谢谢！

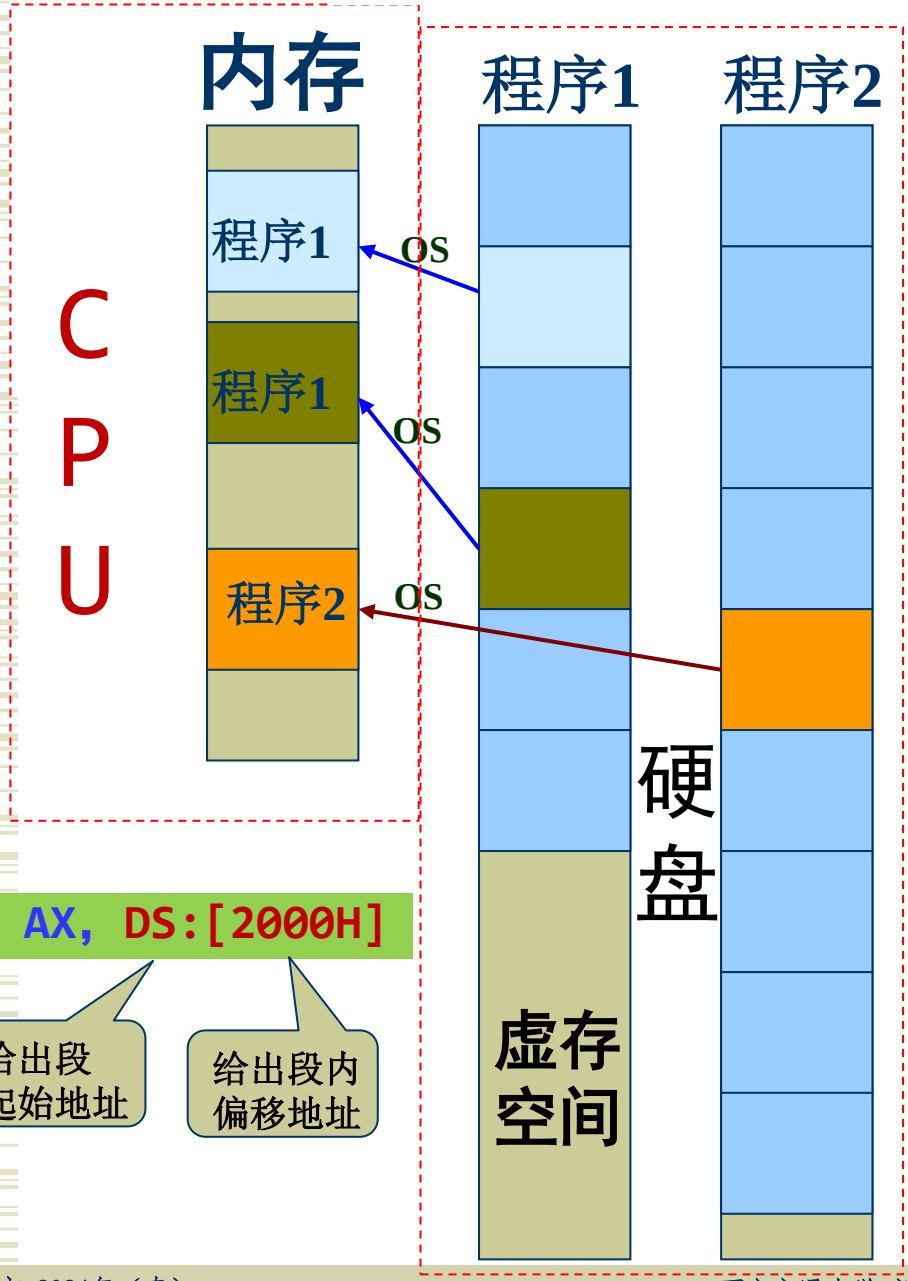
对内存管理由OS完成

- 虚拟存储空间设在硬盘上
- 内存物理空间可比虚拟存储空间小
- 程序执行时
  - 操作系统先分配一个内存段
  - 将程序从虚拟存储空间装入内存执行
  - 内存中程序/数据只是虚拟存储中程序/数据一小段副本
  - 管理过程对应用程序员透明

因此，对内存进行分段管理使用，这样计算机可支持多用户多任务，并且可提高内存的利用率。



对内存采用分段管理优点



- 内存空间可以比虚存空间小
  - 虚存空间=寻址空间，奔腾64G
  - 内存空间可以很小
- 程序可以按虚存空间编写
- 分段管理
  - 正在执行的代码段和数据段装入内存即可
  - 多个程序可以同时使用内存
  - 提高内存利用率
  - 可以运行比内存大的程序
  - 操作系统装入/替换，对用户透明

①资源利用率，②多任务，③保护隔离