

第2章 80X86计算机组织

§80x86计算机的基本结构（CPU、内存、I/O以及连接它们的总线）

§ 80x86 CPU的寄存器组织

§ 实模式的内存分段与编程要点

§ 标志位CF、OF、SF、ZF的含义及判断方法

80x86计算机或x86计算机

通常，将 Intel 公司生产的 8086/8088、80286、80386、80486、Pentium、Pentium Pro、Pentium II、Pentium III、Pentium 4 及其兼容的 CPU，统称为 80x86 CPU 或 x86 CPU

将基于这些 CPU 的计算机，称为 80x86 计算机或 x86 计算机

80x86 CPU的3种工作模式

1. 实模式

与8086兼容的工作模式，只有低20位地址线起作用，仅能寻址第一个1MB的内存空间。MS DOS运行在实模式下。

2. 保护模式

32位80x86 CPU的主要工作模式，提供对程序和数据进行安全保护的机制。在保护模式下，机器可提供**虚拟存储**的管理和**多任务**的管理机制。Windows 9x/NT/2000运行在保护模式下。

计算机可以运行程序空间大于主存储器空间的用户程序

允许多个用户可以同时在机器上工作

3. 虚拟8086模式

一台机器可同时模拟多个8086处理器的工作。在Windows 9x下，若打开一个MS DOS窗口，运行一个DOS应用程序，那么该程序就运行在虚拟8086模式下。

16/32位PC机

Ø 16位PC机是指采用16位80x86 CPU的IBM PC/XT/AT这三款个人微机或它们的兼容机。

Ø 32位PC机是指采用32位80x86 CPU而形成的微机，其基本结构仍然源于PC/AT机。

Ø 本课程采用16位个人计算机

微机系统组成

Ø 硬件 (Hardware)

n 控制器、运算器

n 存储器

n 输入设备和输出设备

n 总线

n 接口

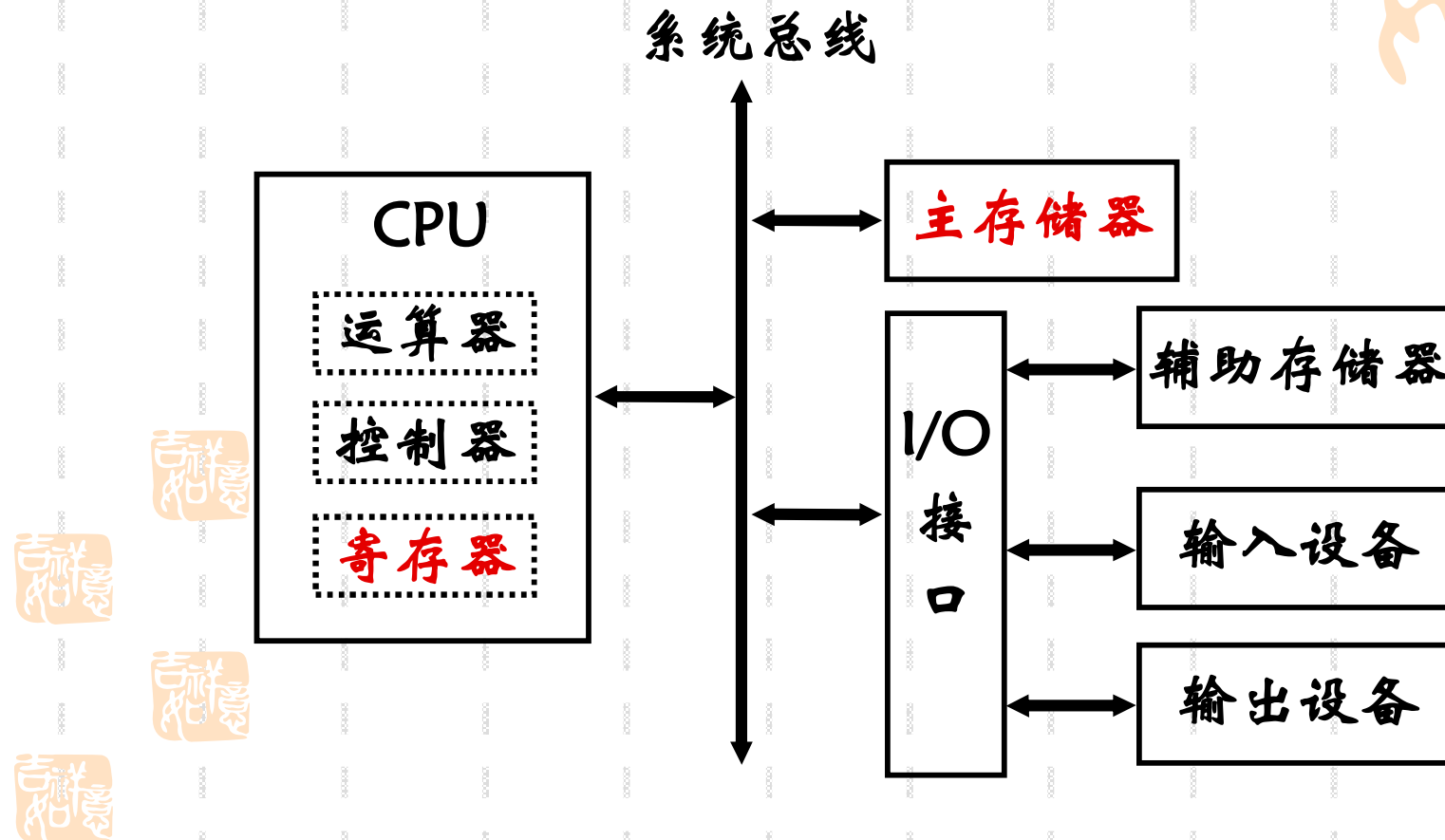
Ø 软件 (Software)

n 系统软件

n 应用软件



微机系统组成图



对汇编语言程序员，最关心CPU中的寄存器、存储器地址、端口 (I/O地址)

CPU



运算器： 执行算术与逻辑运算。

控制器： 控制指令的执行。

寄存器组： CPU内部的高速存储单元，它们为处理器提供各种操作所需要的数据或地址等信息。

汇编语言程序采用它们各自的符号名。例如，在Intel 8086/8088 CPU中有AX BX CX DX

主存储器（内存）

§内存是存放指令和数据的部件，由若干内存单元构成。为了区别每个单元，将它们编号，这个编号就是存储器地址。

§80x86的内存以字节编址：每个内存单元有唯一的地址，可存放1个字节。

§要正确理解内存单元的2个要素：地址（编号）与值（内容）。

主存储器（内存）



20000H

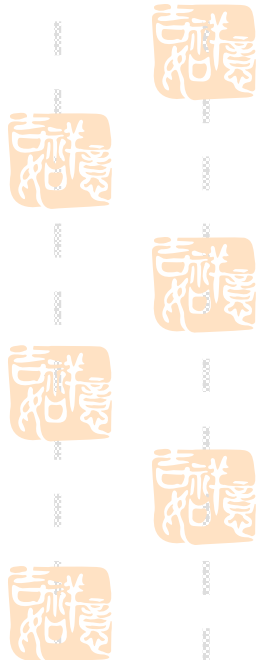
20001H

20
30
...

通常采用十六进制数来表达地址

Intel 8086 具有 1 兆字节
(1MB) 存储器容量

其存储器地址可以表示为：
00000H ~ FFFFFH



存储单元及其存储内容

每个存储单元存放一个字节的內容

D7 D0

34H
12H
56H
78H

00000H

00001H

00002H

00003H

00004H

00005H

00006H

[0002H] = 34H

多字节数据存放方式

多字节数据在存储器中占连续的多个存储单元：

n存放时，低字节存入低地址，高字节存入高地址；

n表达时，用它的低地址表示多字节数据占据的地址空间。

多字节数据存放方式示例

D7 D0

34H
12H
56H
78H

00000H

00001H

00002H

00003H

00004H

00005H

00006H

“字”单元的内容为：

$[0002H] = 1234H$

“双字”单元的内容为：

$[0002H] = 78561234H$

同一个存储器地址可以是
字节单元地址、字单元地址、
双字单元地址等等

数据的地址对齐

字单元安排在偶地址 (xxx0B)、双字单元安排在模4地址 (xx00B) 等, 被称为“地址对齐 (Align)”

对于不对齐地址的数据, 处理器访问时, 需要额外的访问存储器时间, 应该将数据的地址对齐

总线 (1)

总线是部件之间进行数据（电信号）交换的通道。

80x86计算机的系统总线分为3类：

§ 数据总线

§ 地址总线

§ 控制总线

总线 (2)

1. 数据总线

§ 数据总线是用来传递数据的，定义了CPU在每个内存周期所能存取数据的位数。

§ 80x86系列CPU的数据总线为8位、16位、32位或64位。这就是“为什么通常的数据存取是以8位、16位、32位或64位进行的”。

§ 数据总线越宽，处理能力越强。

§ 具有N位数据总线并不意味着CPU只能处理N位数据。

总线 (3)

2. 地址总线

§ 地址总线用来指出数据的地址（内存或I/O）。

§ 地址总线的位数决定了最大可编址的内存与I/O空间。

§ 对于N位地址总线，CPU可以提供 2^N 个不同地址： $0 \sim 2^N - 1$ 。

§ 地址总线由内存与I/O子系统共享使用（I/O只用低16位）。

3. 控制总线

控制总线用来控制CPU与内存和I/O设备之间的数据
传送方式（如传送方向）。
16

I/O 子系统

每个I/O设备必须通过专门的I/O接口电路与主机（CPU和内存）相连。

对程序员来说，I/O接口电路由接口寄存器组成，为了区别它们，各个寄存器进行了编号，形成I/O地址。

端口就是指I/O地址，是微机系统对I/O接口电路中与程序设计有关的寄存器的编号

通常采用十六进制数来表达端口

§80x86 的I/O端口为16位，故支持64K个8位端口

17

一个I/O地址可以表示为：0000H ~ FFFFH

微机的软件

Ø 系统软件：DOS平台


Ø 应用软件：开发汇编语言程序涉及

 n 文本编辑器

 n 汇编程序

 n 连接程序

 n 调试程序

 n 集成化开发环境



文本编辑器 (Editor)

- ❶ 文本编辑器用于编辑无任何格式的文档
- ❷ 程序设计时要采用文本编辑器编写源程序
- ❸ 常见的文本编辑软件有很多，如
 - nMS-DOS的EDIT全屏幕编辑器
 - nWindows的Notepad记事本
 - n程序开发系统中的程序编辑器，例如你熟悉的Turbo C编辑器
 - nMASM集成开发环境的编辑器
- ❹ 大家可以采用微机中任何一个文本编辑器编写汇编语言源程序

汇编程序 (Assembler)

Ø 汇编程序将汇编语言源程序翻译（称为“汇编”）成机器代码目标模块

Ø 汇编程序的主要功能是：

(1) 检查源程序

(2) 测出源程序中的语法错误，并给出出错信息

(3) 产生源程序的目标程序，并可给出列表文件（同时列出汇编语言和机器语言的文件，称为LST文件）

(4) 展开宏指令

汇编程序 (Assembler)

80x86CPU的汇编程序主要有微软的宏汇编程序MASM。

较著名的还有Borland公司的TASM，无实质差别



汇编语言程序与汇编程序是两个概念

连接程序 (Linker)

Ø 连接程序将汇编后的目标模块转换为可执行程序

Ø 每个程序开发环境都有连接程序

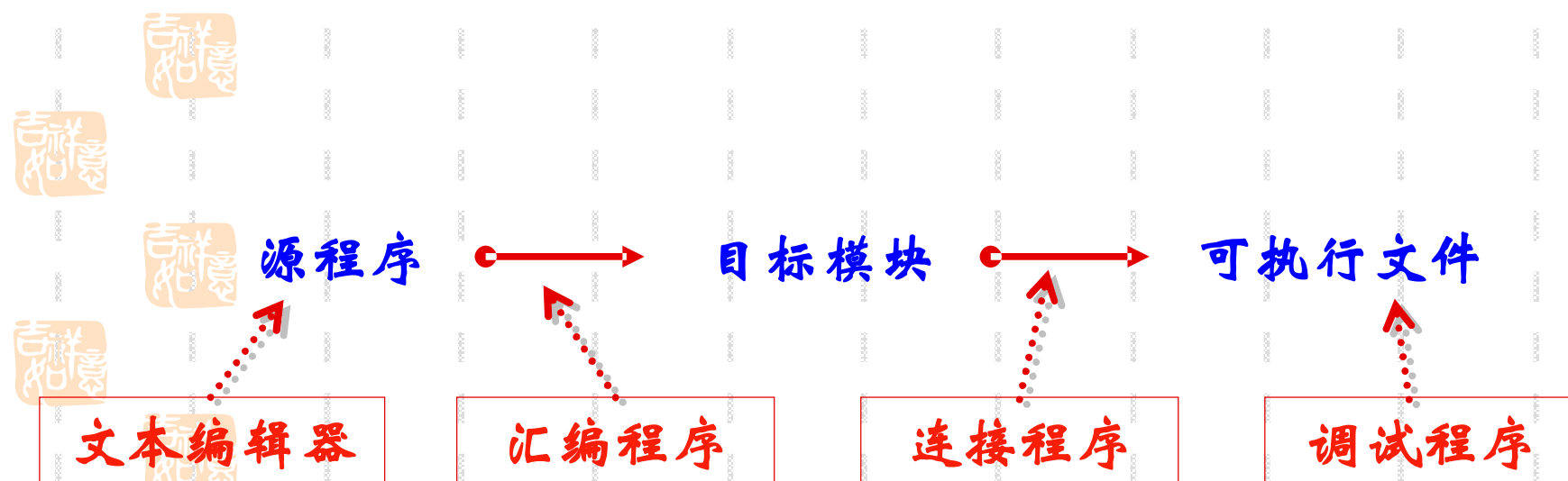
连接程序的文件名通常是: LINK.EXE



调试程序 (Debugger)

Ø 调试程序进行程序排错、分析等

Ø 常用DOS的DEBUG程序



集成化开发环境

集成化开发环境是进行程序设计所用到的各种软件的有机集合。其中，有文本编辑器，有语言翻译程序，有连接程序，还组合有调试程序等。

大型的程序设计项目往往要借助这种集成开发环境，也就是软件开发工具（包）。

8086微处理器



Ø 微处理器是微机的硬件核心

Ø 主要包含指令执行的运算和控制部件，还有多种寄存器

Ø 对程序员来说，微处理器抽象为以名称存储的寄存器



8086 的功能结构

8086 内部结构有两个功能模块，完成一条指令的取指和执行功能

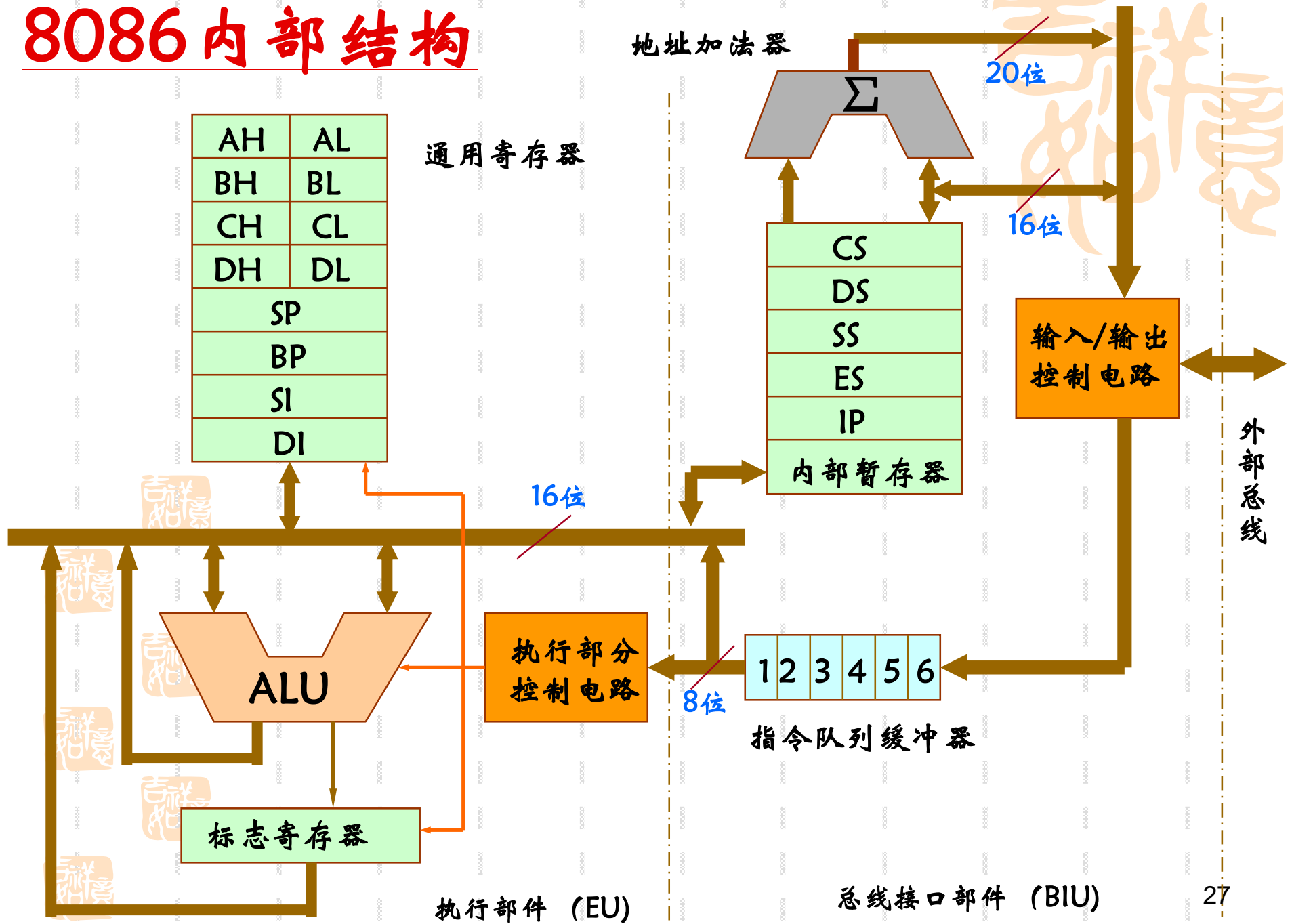
模块之一：总线接口单元BIU，主要负责读取指令和操作数

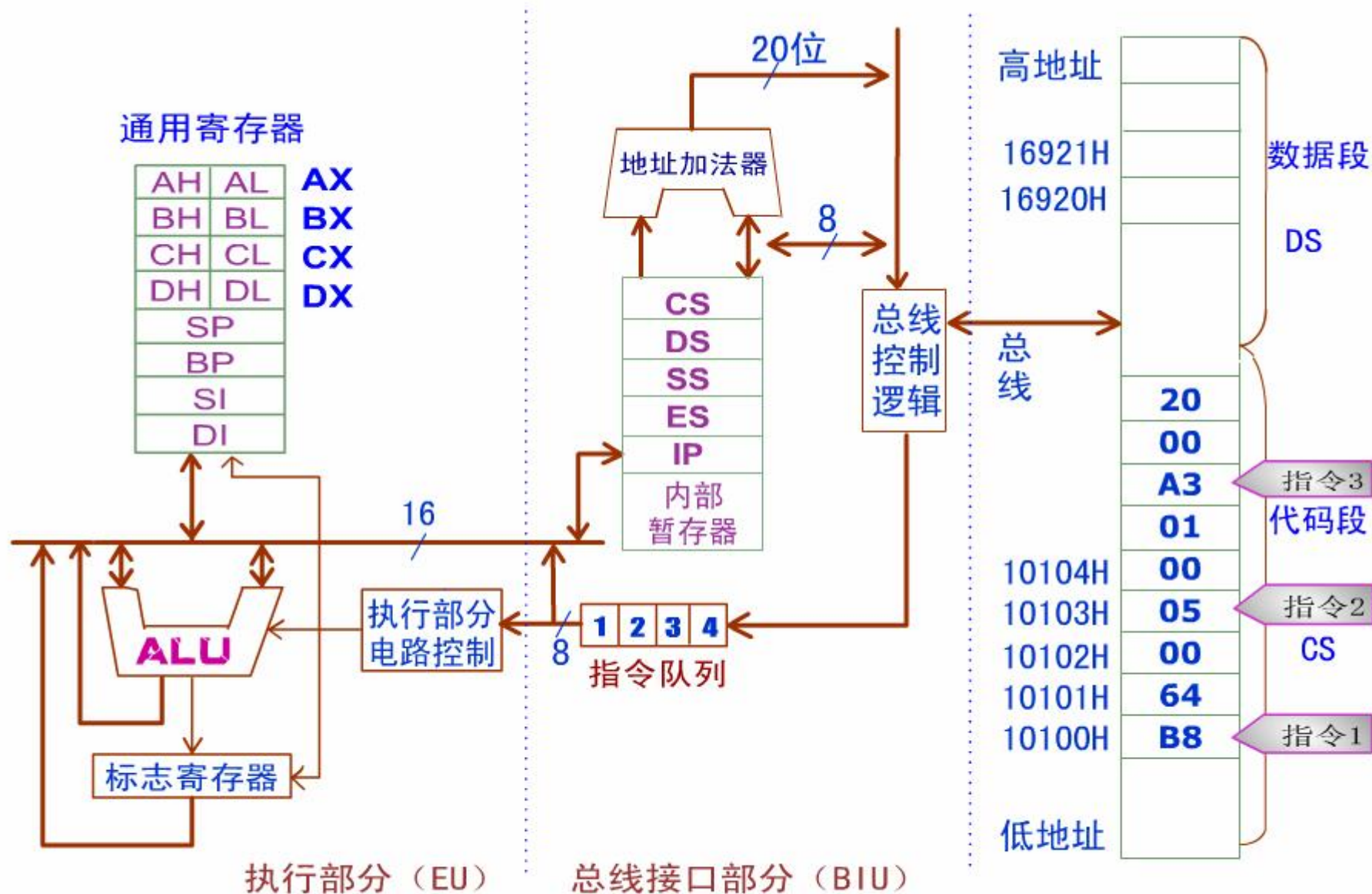
模块之二：执行单元EU，主要负责指令译码和执行

内部结构

指令执行

8086 内部结构





播放



停止

8088的指令执行 示例

8086的寄存器组

对汇编语言程序员来说，8086内部结构就是可编程的寄存器组

✓ 执行单元EU 8个通用寄存器

✓ 1个指令指针寄存器

✓ 1个标志寄存器

✓ 4个段寄存器

8086的通用寄存器

Ø 8086的16位通用寄存器是：

AX BX CX DX SI DI BP SP

Ø 其中前4个数据寄存器都还可以分成高8位和低8位两个独立的寄存器

Ø 8086的8位通用寄存器是：

AH BH CH DH AL BL CL DL

Ø 对其中某8位的操作，并不影响另外对应8位的数据

数据寄存器ax、bx、cx、dx

数据寄存器用来存放计算的结果和操作数，也可以存放地址

每个寄存器又有它们各自的专用目的

nAX——累加器，使用频度最高，用于算术、逻辑运算以及与外设传送信息等；

nBX——基址寄存器，常用做存放存储器地址；

nCX——计数器，作为循环和串操作等指令中的隐含计数器；

nDX——数据寄存器，常用来存放双字长数据的高16位，或存放外设端口地址。

变址寄存器si、di

变址寄存器常用于存储器寻址时提供地址

nSI是源变址寄存器

nDI是目的变址寄存器

串操作类指令中，SI和DI具有特别的功能



指针寄存器sp、bp

Ø 指针寄存器用于寻址内存堆栈内的数据

Ø SP为堆栈指针寄存器，指示栈顶的偏移地址

Ø SP不能再用于其他目的，具有专用目的

Ø BP为基址指针寄存器，表示数据在堆栈段中的基地址

Ø SP和BP寄存器与SS段寄存器联合使用以确定堆栈段中的存储单元地址

堆栈 (Stack)

堆栈是主存中一个特殊的区域

它采用先进后出FILO (First In Last Out) 或后进先出LIFO (Last In First Out) 的原则进行存取操作，而不是随机存取操作方式。

堆栈通常由处理器自动维持。在8086中，由堆栈段寄存器SS和堆栈指针寄存器SP共同指示

8086的寄存器组

对汇编语言程序员来说，8086内部结构就是可编程的寄存器组

✓ 执行单元EU 8个通用寄存器

✓ 1个指令指针寄存器

✓ 1个标志寄存器

✓ 4个段寄存器

指令指针IP

❶ 指令指针寄存器IP，指示代码段中指令的偏移地址

❷ 它与代码段寄存器CS联用，确定下一条指令的物理地址

❸ 计算机通过CS：IP寄存器来控制指令序列的执行流程

❹ IP寄存器是一个专用寄存器

8086的寄存器组

对汇编语言程序员来说，8086内部结构就是可编程的寄存器组

✓ 执行单元EU 8个通用寄存器

✓ 1个指令指针寄存器

✓ 1个标志寄存器

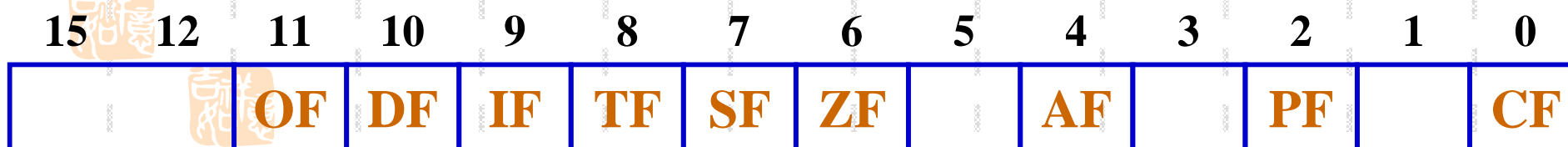
✓ 4个段寄存器

标志寄存器

❶ 标志 (Flag) 用于反映指令执行结果或控制指令执行形式

❷ 8086 处理器的各种标志形成了一个 16 位的标志寄存器 FLAGS (程序状态字 PSW 寄存器)

🎯 程序设计需要利用标志的状态



标志的分类

Ø 状态标志——用来记录程序运行结果的状态信息，许多指令的执行都将相应地设置它

CF ZF SF PF OF AF

Ø 控制标志——可由程序根据需要用指令设置，用于控制处理器执行指令的方式

DF IF TF

进位标志CF (Carry Flag)

当运算结果的最高有效位有进位（加法）或借位（减法）时，进位标志置1，即 $CF = 1$ ；否则 $CF = 0$ 。

$3AH + 7CH = B6H$ ，没有进位： $CF = 0$

$AAH + 7CH = (1) 26H$ ，有进位： $CF = 1$

零标志ZF (Zero Flag)

若运算结果为0，则 $ZF = 1$ ；

否则 $ZF = 0$

注意：ZF为1表示的结果是0

$3AH + 7CH = B6H$ ，结果不是零： $ZF = 0$

$84H + 7CH = (1) 00H$ ，结果是零： $ZF = 1$

符号标志SF (Sign Flag)

运算结果最高位为1，则 $SF = 1$ ；否则 $SF = 0$

有符号数据用最高有效位表示数据的符号
所以，最高有效位就是符号标志的状态

$3AH + 7CH = B6H$ ，最高位 $D_7 = 1$ ： $SF = 1$

$84H + 7CH = (1) 00H$ ，最高位 $D_7 = 0$ ： $SF = 0$

奇偶标志PF (Parity Flag)

当运算结果最低字节中“1”的个数为零或偶数时，PF = 1；否则PF = 0

PF标志仅反映最低8位中“1”的个数是偶或奇，即使是进行16位字操作

$$3AH + 7CH = B6H = 10110110B$$

结果中有5个1，是奇数：PF = 0

溢出标志OF (Overflow Flag)

若算术运算的结果有溢出，
则 $OF = 1$ ；否则 $OF = 0$

$3AH + 7CH = B6H$ ，产生溢出： $OF = 1$

$AAH + 7CH = (1) 26H$ ，没有溢出： $OF = 0$

溢出标志OF (Overflow Flag)

问题

什么是溢出？

溢出和进位有什么区别？

处理器怎么处理，程序员如何运用？

如何判断是否溢出？

什么是溢出

- ❑ 处理器内部以补码表示有符号数
- ❑ 8位表达的整数范围是：+127~-128
- ❑ 16位表达的范围是：+32767~-32768
- ❑ 如果运算结果超出这个范围，就产生了溢出
- ❑ 有溢出，说明有符号数的运算结果不正确

3AH+7CH=B6H，就是58+124=182，
已经超出-128~+127范围，产生溢出，故OF=1；
另一方面，补码B6H表达真值是-74，显然运算结果也
不正确

溢出和进位

❌ 溢出标志OF和进位标志CF是两个意义不同的标志

❌ 进位标志表示无符号数运算结果是否超出范围，运算结果仍然正确；

❌ 溢出标志表示有符号数运算结果是否超出范围，运算结果已经不正确。

请看例子

溢出和进位的对比



例1: $3AH + 7CH = B6H$

无符号数运算: $58 + 124 = 182$ 范围内, 无进位

有符号数运算: $58 + 124 = 182$ 范围外, 有溢出

例2: $AAH + 7CH = (1) 26H$

无符号数运算: $170 + 124 = 294$ 范围外, 有进位

有符号数运算: $-86 + 124 = 28$ 范围内, 无溢出

如何运用溢出和进位

❶ 处理器对两个操作数进行运算时，按照无符号数求得结果，并相应设置进位标志CF；同时，根据是否超出有符号数的范围设置溢出标志OF。

❷ 应该利用哪个标志，则由程序员来决定。也就是说，如果将参加运算的操作数认为是无符号数，就应该关心进位；认为是有符号数，则要注意是否溢出。

溢出的判断

❖ 判断运算结果是否溢出有一个简单的规则：

❖ 只有当两个相同符号数相加（包括不同符号数相减），而运算结果的符号与原数据符号相反时，产生溢出；因为，此时的运算结果显然不正确

❖ 其他情况下，则不会产生溢出

辅助进位标志AF (Auxiliary Carry Flag)

运算时 D_3 位（低半字节）有进位或借位时，
 $AF = 1$ ；否则 $AF = 0$ 。

 这个标志主要由处理器内部使用，用于十进制算术运算调整指令中，用户一般不必关心

$3AH + 7CH = B6H$ ， D_3 有进位： $AF = 1$

方向标志DF (Direction Flag)

用于串操作指令中，控制地址的变化方向：

● 设置 $DF = 0$ ，存储器地址自动增加；

● 设置 $DF = 1$ ，存储器地址自动减少。

CLD指令复位方向标志： $DF = 0$

STD指令置位方向标志： $DF = 1$

中断允许标志IF (Interrupt-enable Flag)

用于控制外部可屏蔽中断是否可以被处理器响应：

● 设置IF=1，则允许中断；

● 设置IF=0，则禁止中断。

CLI指令复位中断标志：IF=0

STI指令置位中断标志：IF=1

陷阱标志TF (Trap Flag)

用于控制处理器进入单步操作方式：

● 设置 $TF = 0$ ，处理器正常工作；

● 设置 $TF = 1$ ，处理器单步执行指令。

单步执行指令——处理器在每条指令执行结束时，便产生一个编号为1的内部中断

这种内部中断称为单步中断

所以TF也称为单步标志

n 利用单步中断可对程序进行逐条指令的调试

n 这种逐条指令调试程序的方法就是单步调试

8086的寄存器组

对汇编语言程序员来说，8086内部结构就是可编程的寄存器组

✓ 执行单元EU 8个通用寄存器

✓ 1个指令指针寄存器

✓ 1个标志寄存器

✓ 4个段寄存器

存储器分段



§ 8086的存储器地址为20位

§ ALU、IP、SP、BP、BX、SI、DI都只有16位



§ 8086将存储器分成若干段来表示



段地址与偏移地址



§ 段地址——段的起始地址的高16位地址。段内再由16位二进制数来寻址

§ 偏移地址——段内存储单元到段首址的字节距离



物理地址与逻辑地址

§ 物理地址——用20位二进制数表示。地址范围为00000h~FFFFFFh

唯一。

逻辑地址——段地址：偏移地址

不唯一

物理地址和逻辑地址的转换

- Ø 将逻辑地址中的段地址左移4位，加上偏移地址就得到20位物理地址
- Ø 一个物理地址可以有多个逻辑地址

逻辑地址 1460:100、1380:F00

物理地址 14700H 14700H

段地址左移4位

加上偏移地址

得到物理地址

14600H

+ 100H

14700H

13800H

+ F00H

14700H

段寄存器

Ø 8086有4个16位段寄存器

nCS (代码段) 指明代码段的起始地址

nSS (堆栈段) 指明堆栈段的起始地址

nDS (数据段) 指明数据段的起始地址

nES (附加段) 指明附加段的起始地址

Ø 每个段寄存器用来确定一个逻辑段的起始地址，每种逻辑段均有各自的用途

代码段 (Code Segment)

Ø 代码段用来存放程序的指令序列

✓ 代码段寄存器CS存放代码段的段地址

✓ 指令指针寄存器IP指示下条指令的偏移地址

Ø 处理器利用CS:IP取得下一条要执行的指令

堆栈段 (Stack Segment)

堆栈段确定堆栈所在的主存区域

堆栈段寄存器SS存放堆栈段的段地址

堆栈指针寄存器SP指示堆栈栈顶的偏移地址

处理器利用SS:SP操作堆栈顶的数据

数据段 (Data Segment)

❖ 数据段存放运行程序所用的数据

✓ 数据段寄存器DS存放数据段的段地址

✓ 各种主存寻址方式（有效地址EA）得到存储器中操作数的偏移地址

✓ 处理器利用DS:EA存取数据段中的数据

附加段 (Extra Segment)

○ 附加段是附加的数据段，也用于数据的保存：

✓ 附加段寄存器ES存放附加段的段地址

✓ 各种主存寻址方式（有效地址EA）得到存储器中操作数的偏移地址

处理器利用ES:EA存取附加段中的数据

串操作指令将附加段作为其目的操作数的存放区域₆₄

分段要求

Ø 8086对逻辑段要求:

n段地址低4位均为0

n每段最大不超过64KB

Ø 8086对逻辑段并不要求:

n必须是64KB

n各段之间可以重叠

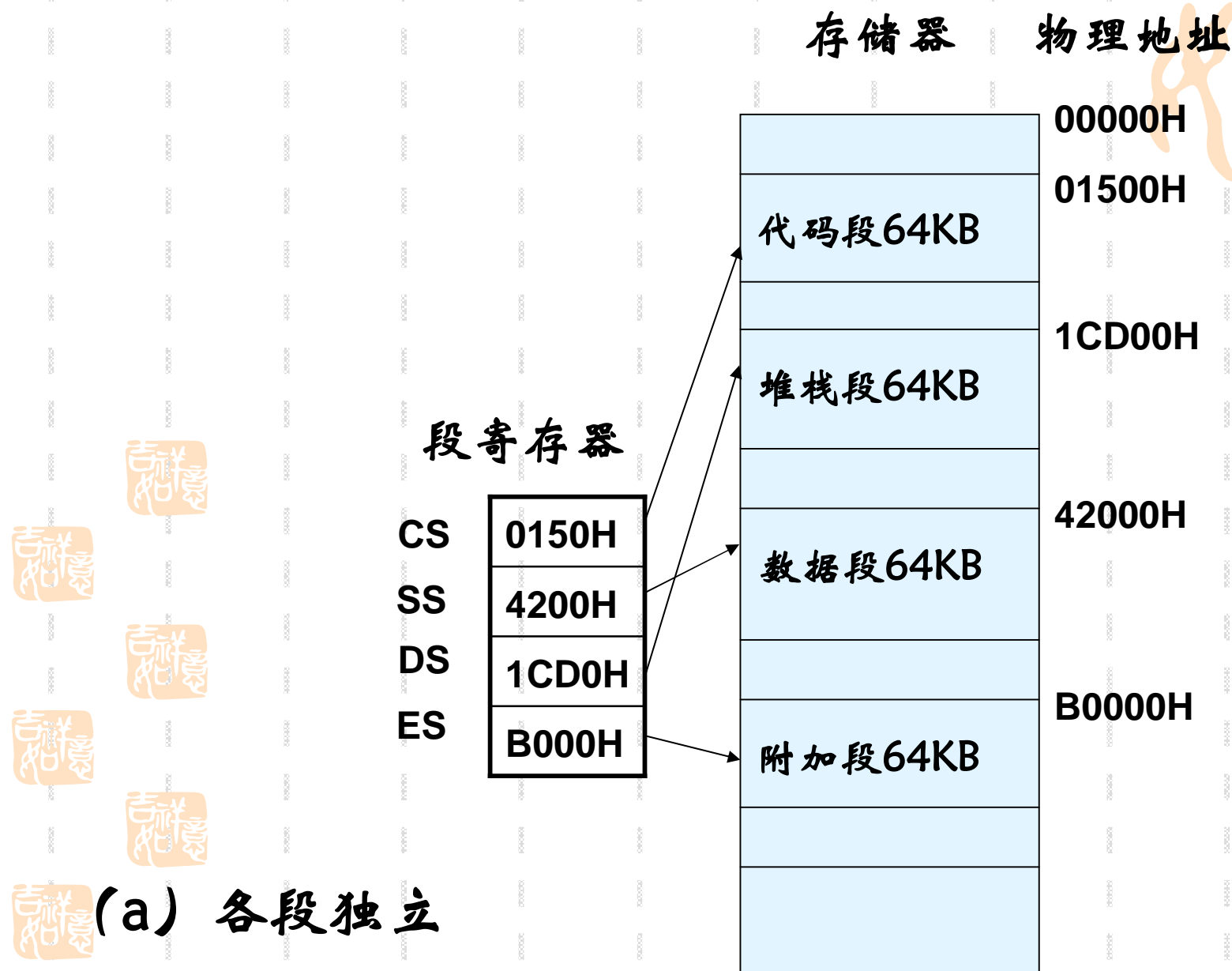
各段独立

各段重叠

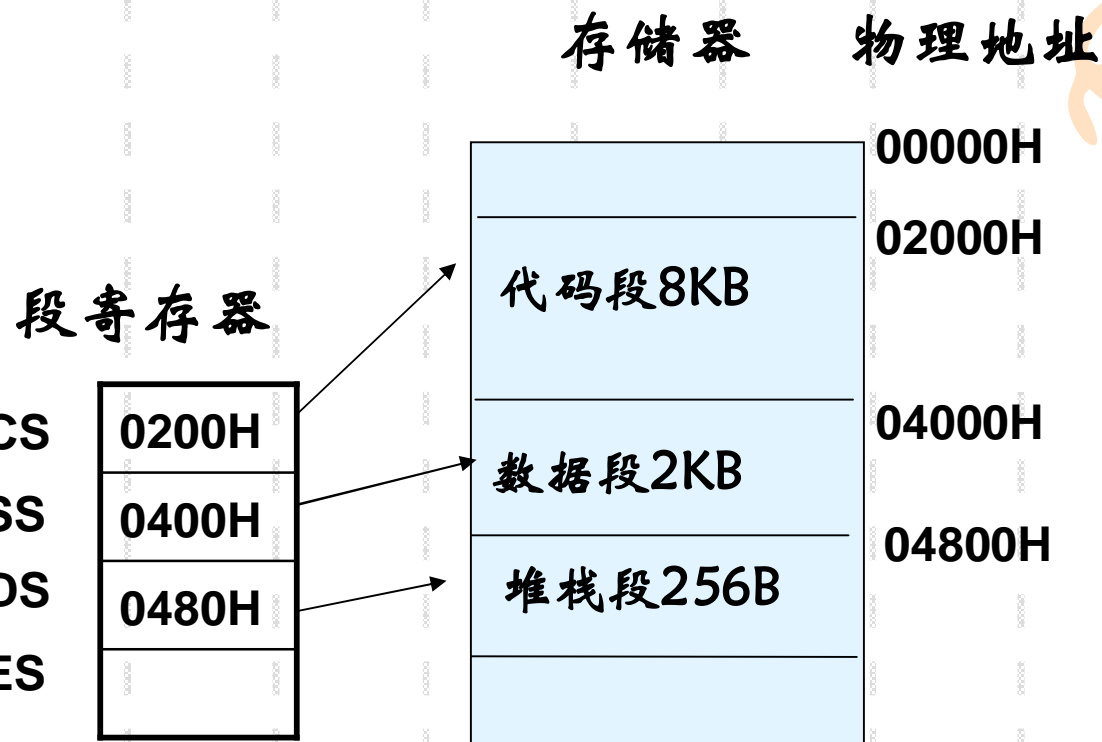
最多多少段?

最少多少段?

各个逻辑段独立



各个逻辑段重叠



(b) 各段重叠

1MB 空间的分段

Ø 1MB 空间最多能分成多少个段？

每隔16个存储单元就可以开始一个段，
所以1MB最多可以有：

$$2^{20} \div 16 = 2^{16} = 64K \text{ 个段}$$

Ø 1MB 空间最少能分成多少个段？

每隔64K个存储单元开始一个段，
所以1MB最少可以有：

$$2^{20} \div 2^{16} = 16 \text{ 个段}$$



如何分配各个逻辑段

演示

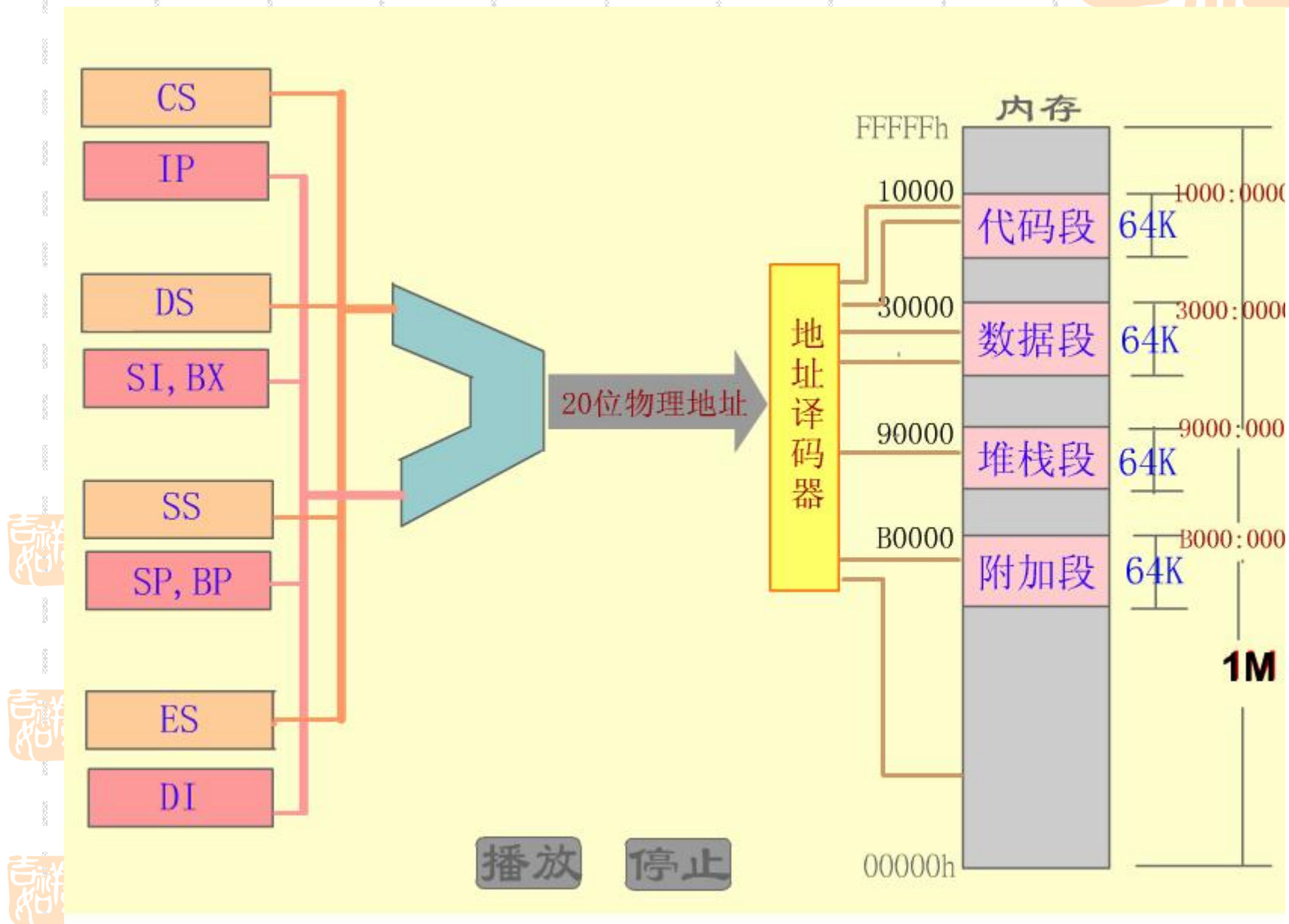
程序的指令序列必须安排在代码段

程序使用的堆栈一定在堆栈段

程序中的数据默认是安排在数据段，也经常安排在附加段，尤其是串操作的目的区必须是附加段

数据的存放比较灵活，实际上可以存放在任何一种逻辑段中

逻辑段分配



段超越前缀指令

示例

❌ 没有指明时，一般的数据访问在DS段；使用BP访问主存，则在SS段

❌ 默认的情况允许改变，需要使用段超越前缀指令；8086指令系统中有4个：

CS: ; 代码段超越，使用代码段的数据

SS: ; 堆栈段超越，使用堆栈段的数据

DS: ; 数据段超越，使用数据段的数据

ES: ; 附加段超越，使用附加段的数据

段超越的示例

Ø 没有段超越的指令实例：

MOV AX,[2000H] ; AX←DS:[2000H]

；从默认的DS数据段取出数据

Ø 采用段超越前缀的指令实例：

MOV AX,ES:[2000H] ; AX←ES:[2000H]

；从指定的ES附加段取出数据

不允许使用段超越的情况

§ 串处理指令的目的串必须用ES段

§ PUSH指令的目的和POP指令的源必须用SS段

§ 指令必须存放在CS段

段寄存器的使用规定

访问存储器的方式	默认	可超越	偏移地址
取指令	CS	无	IP
堆栈操作	SS	无	SP
一般数据访问	DS	CS ES SS	有效地址EA
BP基址的寻址方式	SS	CS ES DS	有效地址EA
串操作的源操作数	DS	CS ES SS	SI
串操作的目的操作数	ES	无	DI

8086微处理器总结

Ø 8086有8个8位通用寄存器、8个16位通用寄存器

Ø 8086有6个状态标志和3个控制标志

Ø 8086将1MB存储空间分段管理，有4个段寄存器，对应4种逻辑段

Ø 8086有4个段超越前缀指令，用于明确指定数据所在的逻辑段

第2章 教学要求

1. 了解微机系统的基本软硬件组成
2. 熟悉汇编语言的基本概念和应用特点
3. 掌握8086的寄存器组和存储器组织

习题 (p33)

2.4 2.8 2.9

