汇编语言程序设计

Assembly Language Programming

主讲：徐娟

计算机与信息学院　　分布智能与物联网研究所

E-mail: xujuan@hfut.edu.cn,

Mobile: 18055100485

第一章 基础知识

1.1 数的表示

1.2 微型计算机（PC）系统

1.3 Intel 80x86系列微处理器

1.4 8086微处理器

1.5 8086的寻址方式

§1 数的表示

- 数制
- 数制之间的转换
- 二进制运算
- 计算机中数的表示
- BCD码
- 字符编码

1.1 数　制

- **十进制：基数为10，逢十进一**

$12.34D = 1\times10^{1} + 2\times10^{0} + 3\times10^{-1} + 4\times10^{-2}$

- **二进制：基数为2，逢二进一**

$1101B = 1\times2^{3} + 1\times2^{2} + 1\times2^{0} = 13_{10}$

- **十六进制：基数为16，逢十六进一**

$9\quad 1\quad 8\quad 7\ H$

$= 9\times16^{3} + 1\times16^{2} + 8\times16^{1} + 7\times16^{0}$

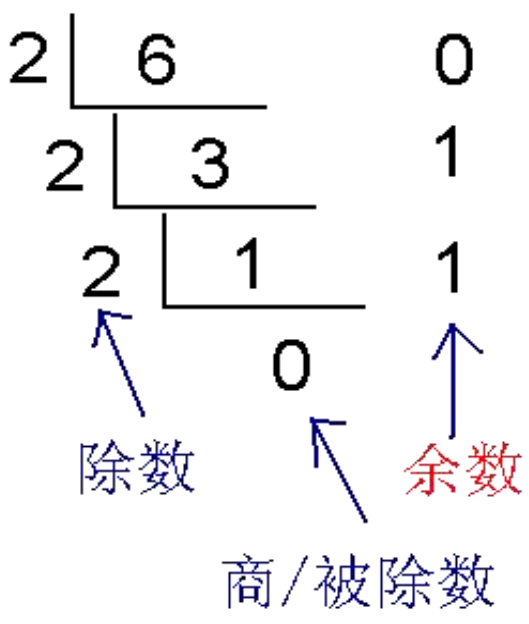
- **八进制：基数为8，逢八进一**

1.1 数　制

数　制	基 数	数　码
二进制 Binary	2	0,1
八进制 Octal	8	0,1,2,3,4,5,6,7
十进制 Decimal	10	0,1,2,3,4,5,6,7,8,9
十六进制 Hexadecimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F

1.2 数制之间的转换

- **二进制 ⇄ 十进制**
  - → 1011B = 11D
  - ← 除法：
    - 整数部分除2；
    - 小数部分乘2。



步骤演化：

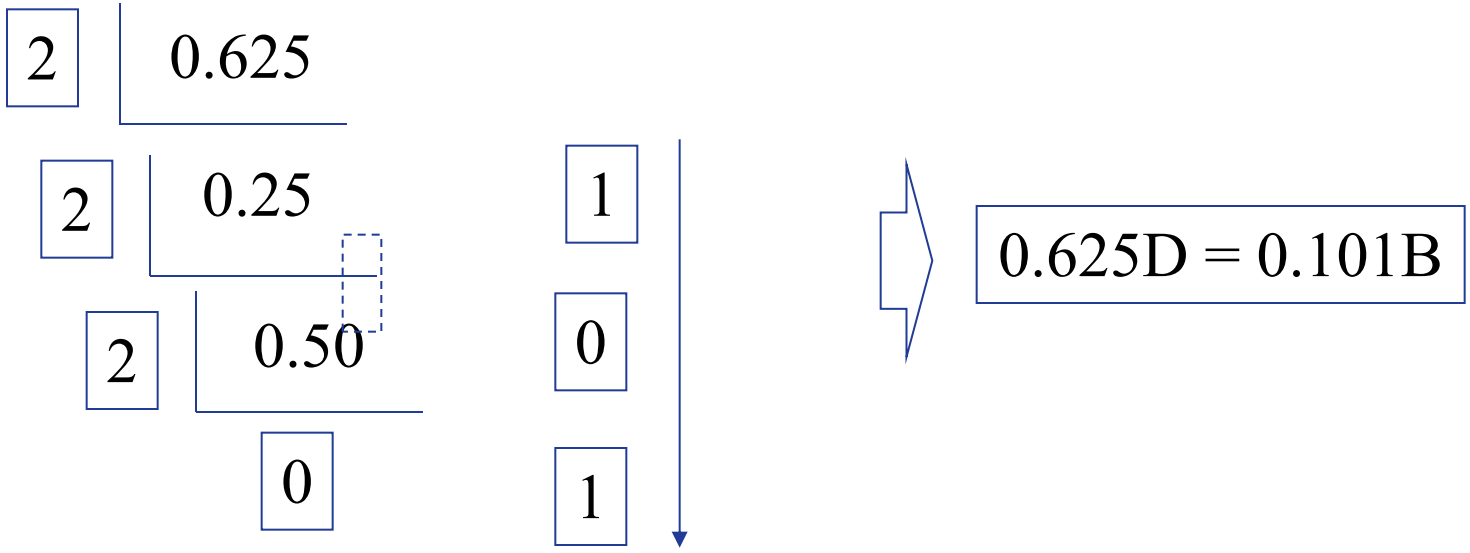
2 | 6 0
3

① 6 ÷ 2，商2余0

2 | 6 0
2 | 3 1
1

② 3 ÷ 2，商1余1

1.2 数制之间的转换



- 小数可能不能用二进制来表示完全，如0. 6

1.2 数制之间的转换

❖二进制 ⇄ 十六进制

→ 0011 0101 1011 1111

↓ ↓ ↓ ↓

3 5 B F

← A 1 9 C

↓ ↓ ↓ ↓

1010 0001 1001 1100

∴ A19CH = 1010,0001,1001,1100B

4位二进制对应一位16进制

1.2 数制之间的转换

- 十六进制 ⇄ 十进制

→ BF3CH = $11\times16^3 + 15\times16^2 + 3\times16^1 + 12\times16^0$

← 降幂法　除法

- 常用数

0—00H　128—80H　255—FFH　256—100H

32767—7FFFH　65535—FFFFH

1.3 二进制运算

算术运算

二进制

加法规则

0+0=0

0+1=1

1+0=1

1+1=0 （进位1）

乘法规则

0×0=0

0×1=0

1×0=0

1×1=1

十六进制

```
  05C3H          3D25H
+ 3D25H        - 05C3H
------         ------
  42E8H          3762H
```

1.3 二进制运算

逻辑运算（按位bit操作）

“与”运算（AND）

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

“或”运算（OR）

A	B	A∨B
0	0	0
0	1	1
1	0	1
1	1	1

“非”运算（NOT）

A	～A
0	1
1	0

“异或”运算（XOR）

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

1.3 二进制运算

例： X=00FFH　　Y=5555H，求Z=X ⊕ Y=?

X= 0000　0000　1111　1111 B

⊕ Y= 0101　0101　0101　0101 B

∴ Z=55AAH

1.4 计算机中数的表示

- 肯定是用二进制
- 无符号数：直接用“二进制”来表示。
- 有符号数：原码、反码、补码。
- 浮点数：（尾数）规格化+（指数）移码

补 码

定义

- 最高有效位为符号位：0—正；1—负
- 正数的补码为它本身（二进制值）；
- 负数的补码 = $2^n-|x|$， n为机器的字长。

$2^n-|x|$ 等价于 对$|x|$取反+1

例

- $[46]_{补码}$ = 0010 1110

$[-46]_{补码} = 2^8 - |-46| = 1,0000,0000 - 0010\ 1110 = 11010010$

$[-46]_{补码}$

$|-46| = 46 = 0010\ 1110 \rightarrow 1101\ 0001+1=1101\ 0010$

补码的特性

- 补码的运算

$$[N_1+N_2]_{补}=[N_1]_{补}+[N_2]_{补}$$

$$[N_1-N_2]_{补}=[N_1]_{补}+[-N_2]_{补}$$

$$[N]_{补补}=[N]$$

补码的加、减运算都可以转换成加法运算，

运算时符号位参加运算。

符号位进位丢弃，结果为负数再取补码。

举 例

❖ 补码运算

例： $N_1=-0.\ \ 1100$　　$N_2=-0.0010$

$[N_1]_{补}=1.0100$　　$[N_2]_{补}=1.1110$　　$[-N_2]_{补}=0.0010$

$$\begin{array}{rl} 1.0100 & [N_1]_{补} \\ +\ \ 1.1110 & [N_2]_{补} \\ \hline (1)\,1.0010 & [N_1+N_2]_{补} \end{array}$$

丢弃 ←

取补： 1.1110　$[N_1+N_2]_{补补}$

真值 ：-0.1110　$[N_1+N_2]$

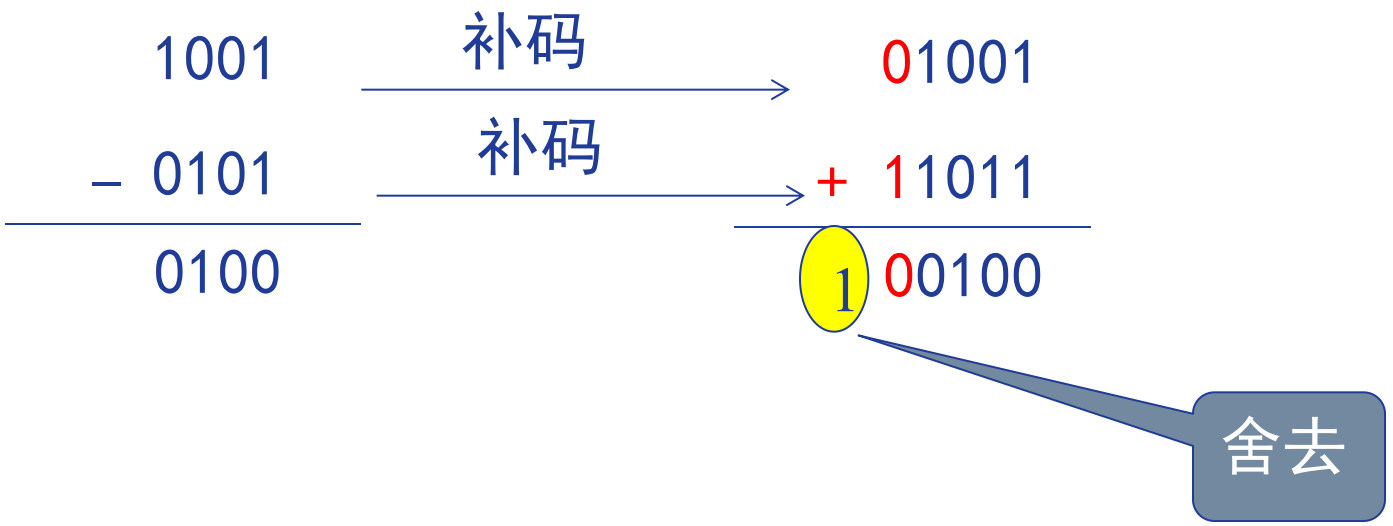
$$\begin{array}{rl} 1.0100 & [N_1]_{补} \\ +\ \ 0.0010 & [-N_2]_{补} \\ \hline 1.0110 & [N_1-N_2]_{补} \end{array}$$

取补： 1.1010　$[N_1-N_2]_{补补}$

真值 ：-0.1010　$[N_1-N_2]$

举 例

例：用二进制补码运算求出（1001）$_2$－（0101）$_2$



数 的 范 围

- 无符号数：8位(0--255） 16位（0--65535）
- 有符号数：8位（-128--127） 16位（-32768--32767）
- n位补码表示数范围： $-2^{n-1} \leq N \leq 2^{n-1}-1$

补码的表数范围

一个带符号数在不同位数下，其二进制补码表示可能是不同的

十进制	二进制	十六进制	十进制	十六进制
	n=8			n=16
+127	0111 1111	7F	+32767	7FFF
+126	0111 1110	7E	+32766	7FFE
…	…	…	…	…
+2	0000 0010	02	+2	0002
+1	0000 0001	01	+1	0001
0	0000 0000	00	0	0000
-1	1111 1111	FF	-1	FFFF
-2	1111 1110	FE	-2	FFFE
…	…	…	…	…
-126	1000 0010	82	-32766	8002
-127	1000 0001	81	-32767	8001
-128	1000 0000	80	-32768	8000

补码:C语言

- 其实C语言中unsigned和signed两个关键字就是来区别无符号数和有符号数，缺省的是signed类型，也就是有符号数。
- 实例：
  - char i = -1;
  - short j = -1;
  - int k = -1

1.5 BCD (Binary-Coded Data) 码

❖ 二进制编码的十进制：

- Packed BCD：用4位二进制表示一个十进制数码
  - 0000-----0　　0001-----1
  - 0001, 0010, 0011, 0100 = 1234
- Unpacked BCD：用8位二进制表示一个十进制数码
  - ****0000----0　　****0001----1
  - ****, 0001, ****, 0010, ****, 0011, ****, 0100 = 1234

1.6 字符编码

- ASCII：英文，7 bits（128个代码）
- 常用字符的ASCII码。
  - 数字'0'～'9'：30H～39H
  - 字母'A'～'Z'：41H～5AH
  - 字母'a'～'z'：61H～7AH
  - 空格：20H
  - 回车CR：0DH——控制光标回到当前行的最左端
  - 换行LF：0AH——移动光标到下一行，而所在列不变
  - 空字符：0

注意回车与换行的差别

1.6 字符编码

- GB:国标码，是我国于1981年公布的国家标准，作为信息交换用汉字编码的字符(GB2312-80),包括6763个简体字以及其他字符。
- GBK: GB的扩展，包括Unicode中的20902个汉字，也称汉字大字符集。
- BIG5:大五碼，包括13,060個繁體字，也是香港比較多人使用的標準。
- UNICODE: 16位二进制 = 65536 汉字：20902个
- UTF-8: 用4字节表示，2^{32} = 42亿….

第一章 基础知识

1.1 数的表示

1.2 微型计算机（PC）系统

1.3 Intel 80x86系列微处理器

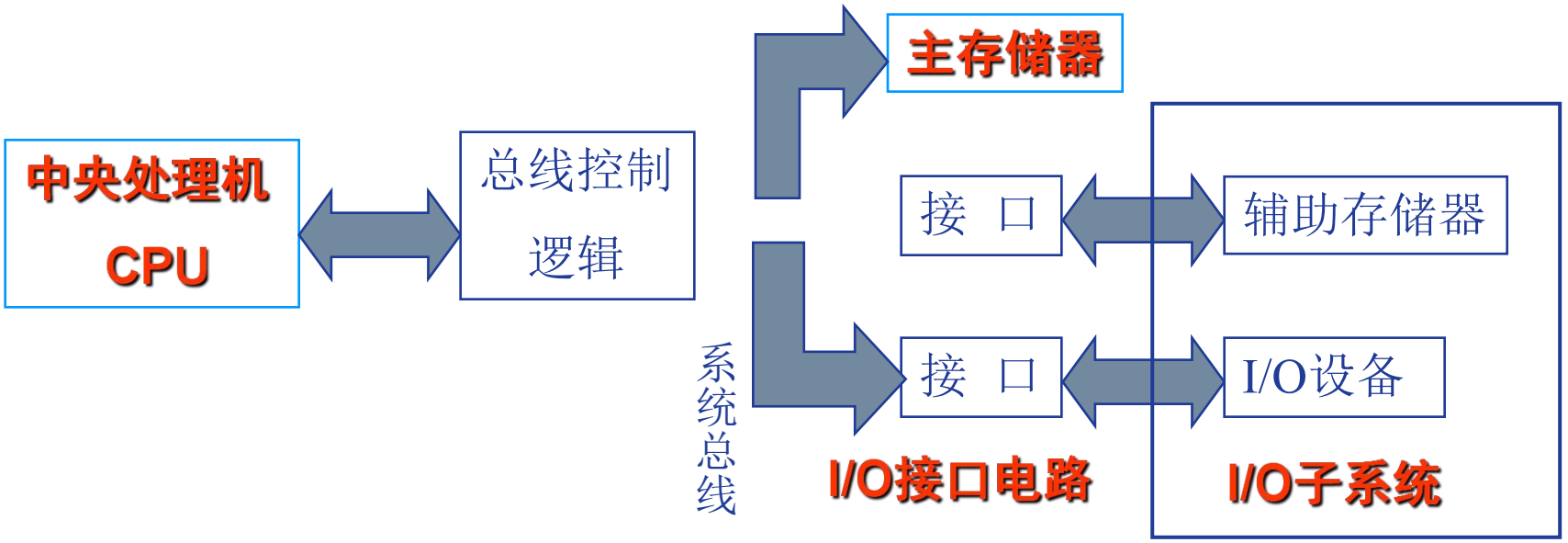
1.4 8086微处理器

1.5 8086的寻址方式

2 微型计算机(PC)系统

2 微型计算机(PC)系统

- PC的硬件：主机、键盘、鼠标、显示器
- 主板的组成：CPU、存储器、外围芯片组、扩张插槽等
- 扩张插槽上有：RAM内存+接口卡



汇编语言程序员看到的硬件

- **中央处理单元 CPU（Intel 80x86）**

  对汇编语言程序员，最关心其中的寄存器组

- **存储器（主存储器）**

  呈现给汇编语言程序员的，是存储器地址

- **外部设备（接口电路）**

  汇编语言程序员看到的是端口（I/O地址）

内存（存储器）

- 内存是存放指令和数据的部件，由若干内存单元构成。
- 指令和数据是应用上的概念，在内存中都是二进制数，没有区别。

$2^2=4$　$2^4=16$　$2^8=256$　$2^{10}=1024$　$2^{20}=1048576$

存储器用以下单位来计量容量

1个二进制位：bit（比特）

8个二进制位：Byte（字节） 1Byte=8bit　$D_7 \sim D_0$

2个字节：Word（字） 1Word=2Byte=16bit　$D_{15} \sim D_0$

1个双字：DWord = 2 Word　$D_{31} \sim D_0$

1KB= 2^{10}=1024B　（Kilo）

1MB=1024KB= 2^{20}　（Mega）

1GB=1024MB= 2^{30}　（Giga）

1TB=1024GB = 2^{40}　（Tera）

B：Byte

b：bit

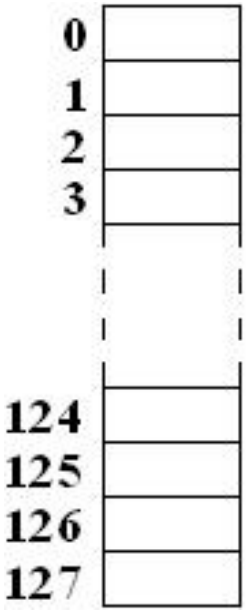
网络速度：10Mbps

文件大小：10MB

内存（存储器）

- 与内存储器相关概念：单元，地址，内容，字长
  - 把内存储器视为一个存放信息的大仓库，而一个大仓库又分成若干个小的存储间，每一个房间称为一个单元；
  - 为了区别这些单元，给每个单元编号，这个编号称为地址；
  - 单元内部存放着信息称为单元的内容。
  - 单元信息的长度成为字长。
- 比喻：宿舍楼 ---宿舍（号）---学生---学生个数

存储器被划分为若干个存储单元， 每个存储单元从0开始顺序编号；编号＝地址



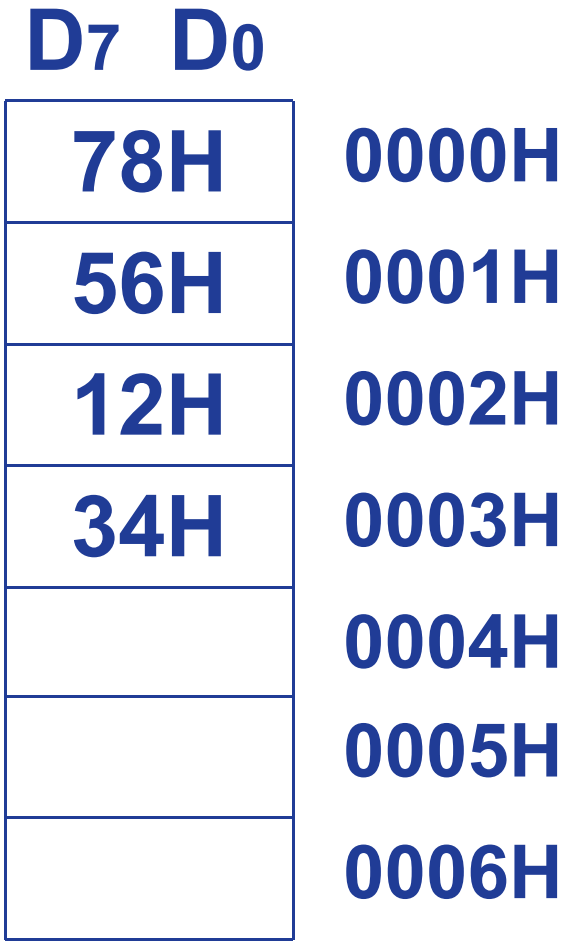
数据的存储格式

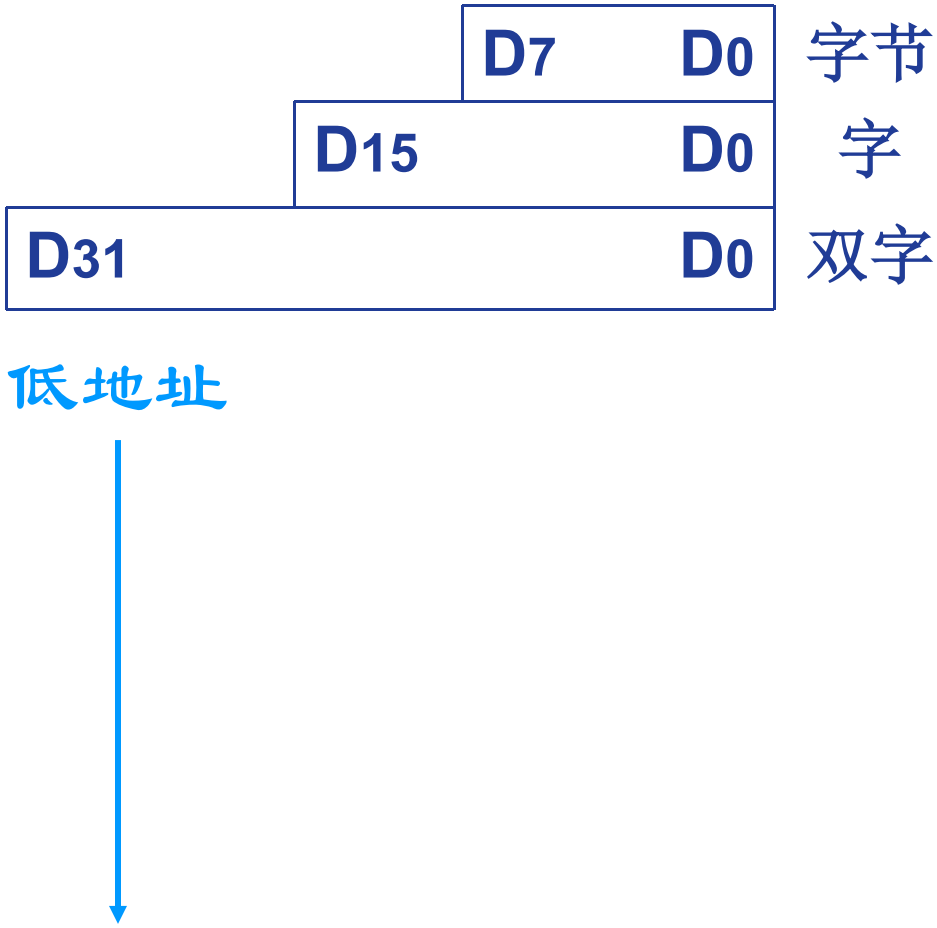
- 80x86的内存以字节编址：每个内存单元有唯一的地址，可存放1个字节。
- 内存单元的2个要素：地址（编号）与值（内容）。
  - （100H）=34H　　或者　　[0002H]＝34H
  - 地址用无符号整数来表示（编程用十六进制表示）
- 多字节数据在存储器中占连续的多个存储单元；
  - 低字节在低地址单元，高字节在高地址单元；
  - 字的地址由其低地址来表示。双字也类似。（小端方式）
  - 同一地址可以看作是字节、字或双字单元的地址。

数据的存储格式

2号“字”单元的内容为：[0002H] = 3412H

0号“双字”单元的内容为：[0000H] = 34125678H





CPU

- 组成：
  - 算术逻辑部件、控制部件和寄存器组。
- CPU的作用：
  - 控制指令的执行。
  - 执行算术与逻辑运算。
- 对汇编语言程序员来说，CPU通过寄存器完成一条指令的取指和执行功能。

- CPU的内部总线实现CPU内部各个器件之间的联系。
- 外部总线（系统总线）实现CPU和主板上其它器件的联系。

I/O 子系统

I/O子系统

- 通过接口电路与微机系统连接
- I/O接口电路由若干接口寄存器组成，需要用编号区别各个寄存器：数据寄存器、状态寄存器、命令寄存器
- I/O端口是I/O地址的通俗说法，是接口电路中寄存器的编号。

  汇编语言程序员看到的，是端口
- 8086计算机采用16位表示I/O端口，系统通过这些端口与外设进行通信
- Intel 8086支持64K个8位端口
- I/O地址可以表示为：0000H ～ FFFFH

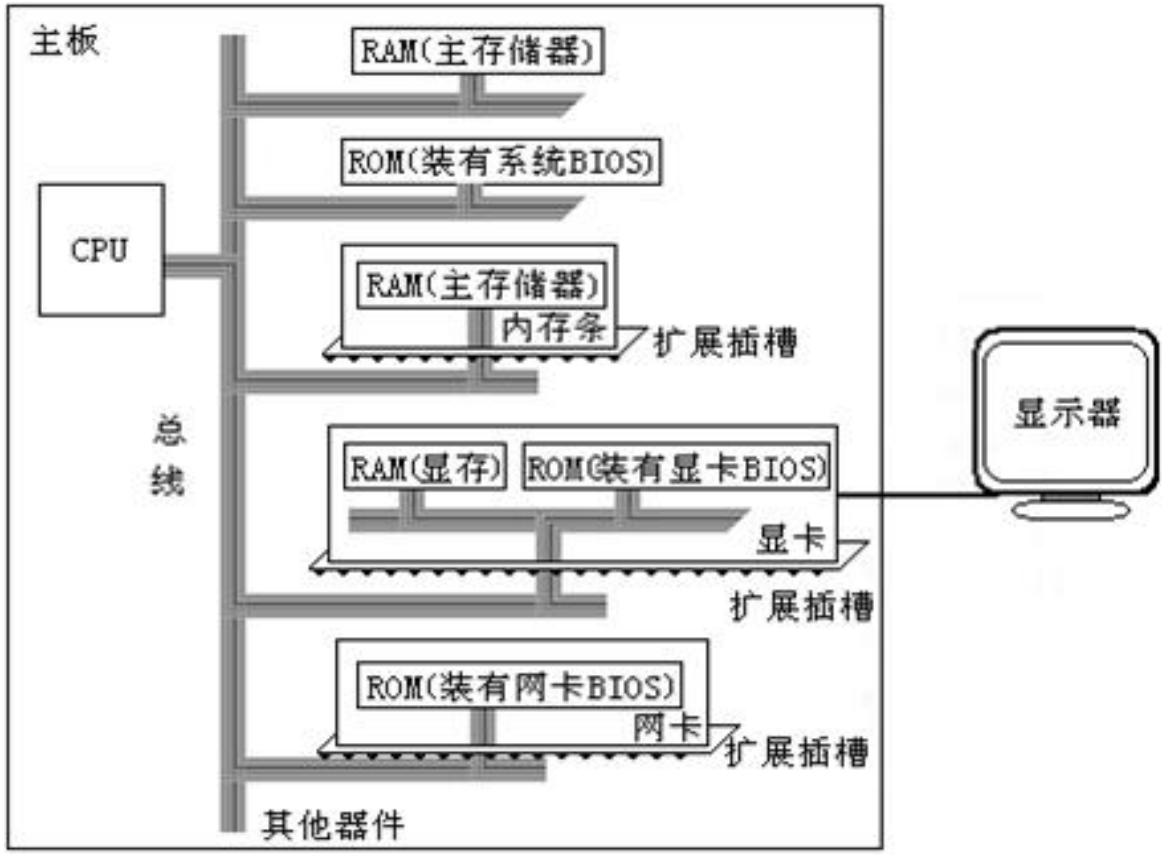
各类存储器芯片

- **从读写属性上看分为两类：**

  随机存储器（RAM）

  只读存储器（ROM）

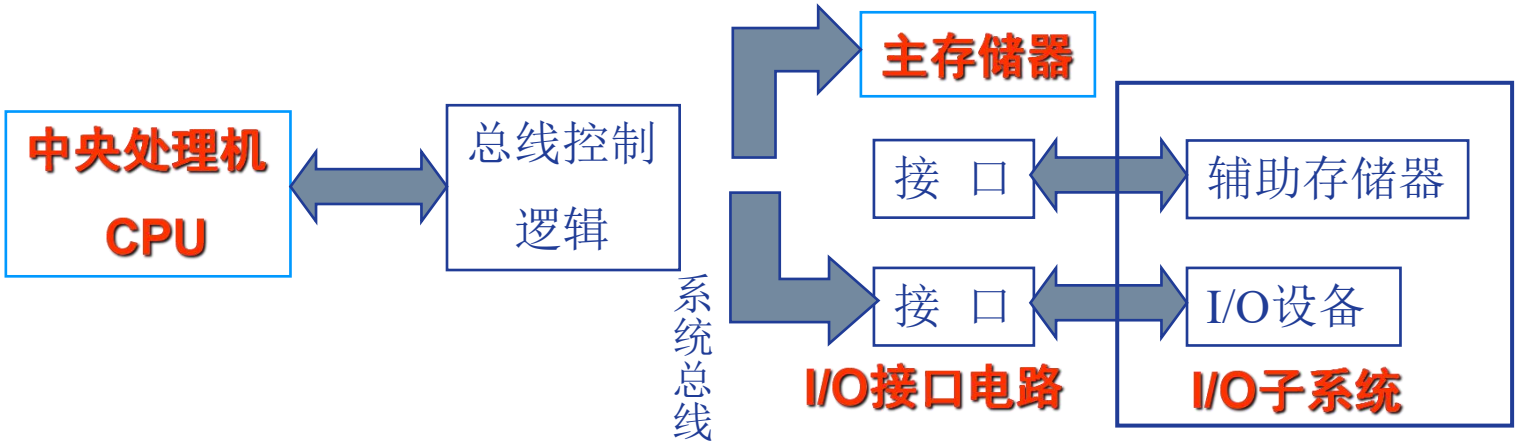
- **从功能和连接上分类：**
  - 随机存储器RAM
  - 装有BIOS的ROM
  - 接口卡上的RAM



PC机中各类存储器的连接情况

系统总线

- 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



地址总线

❖ 地址总线：

- 地址总线用来指出数据的地址（内存或I/O）。

  CPU是通过地址总线来指定存储单元的。

- 地址总线的位数决定了最大可编址的内存与I/O空间。

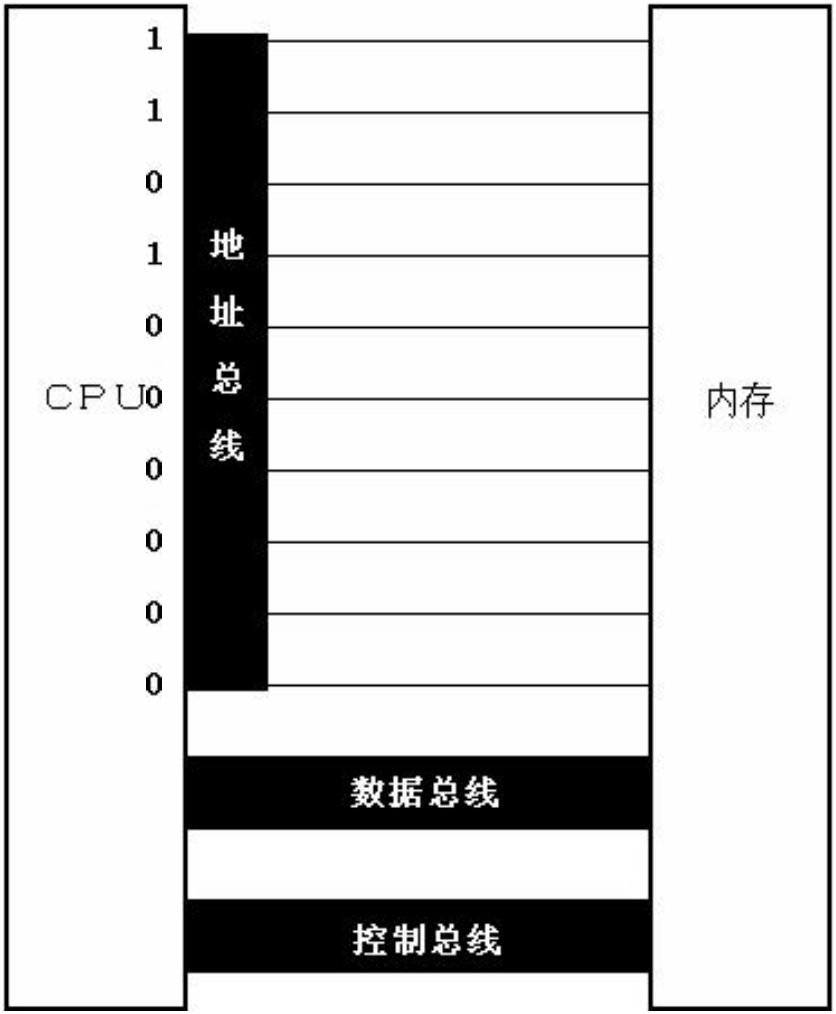
  对于N位地址总线，CPU可以提供2^N个不同地址：0～2^N -1。

  也就是说地址总线的宽度决定了CPU的寻址能力；

- 地址总线由内存与I/O子系统共享使用（I/O只用低16位）。

地址总线

地址总线发送地址信息

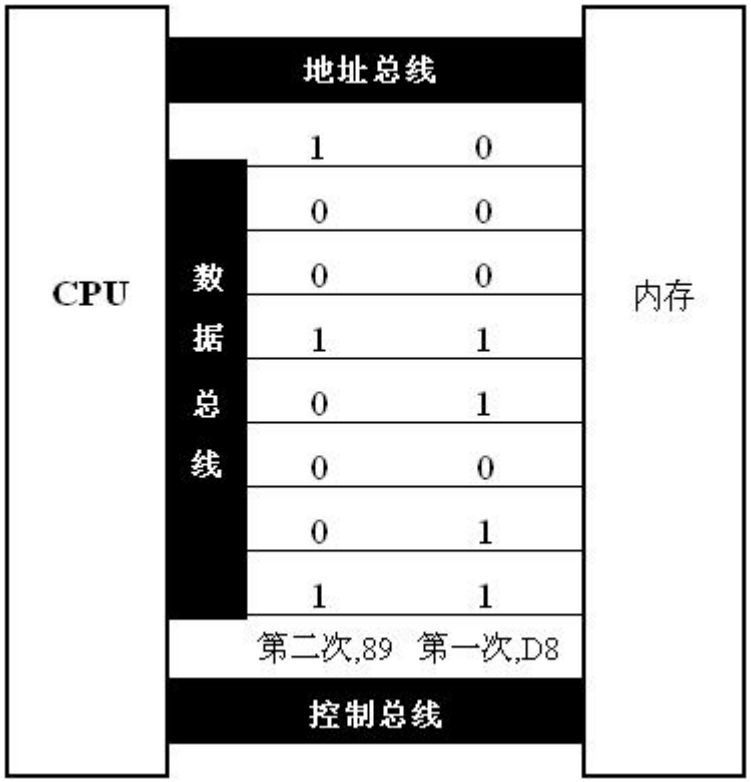


数据总线

- 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”。
- 数据总线的宽度决定了CPU和外界的数据传送速度。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。

数据总线

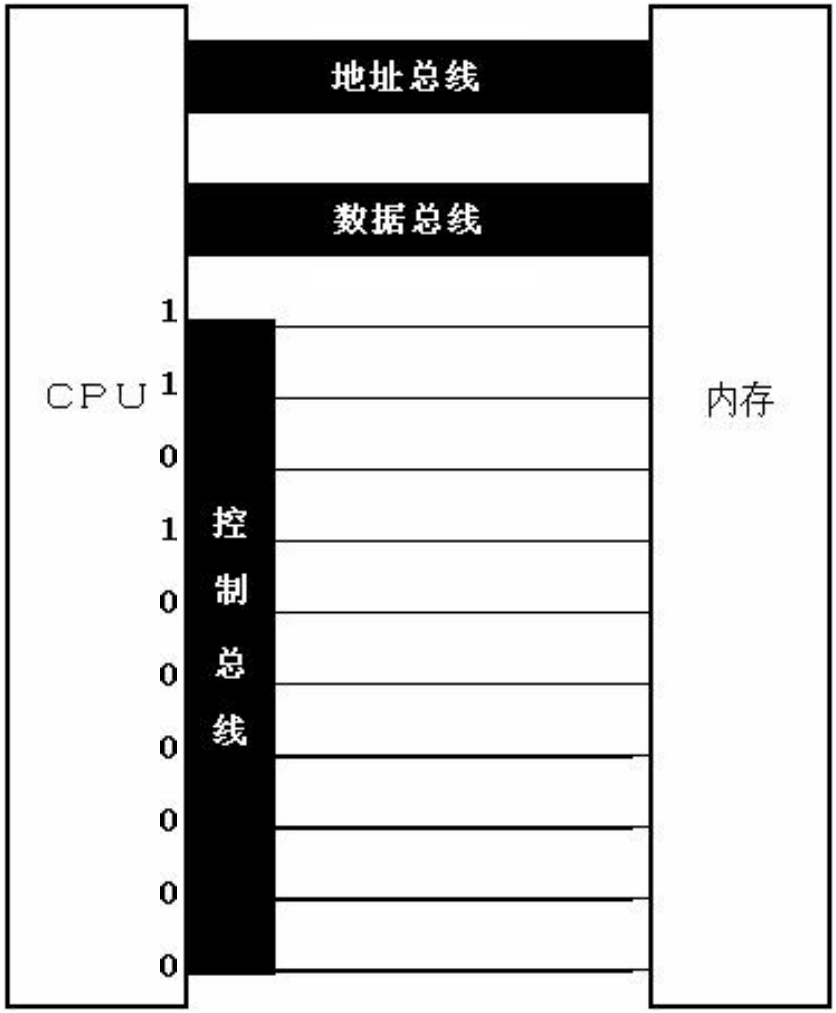
8位数据总线上传送的信息



16位数据总线上传送的信息

控制总线

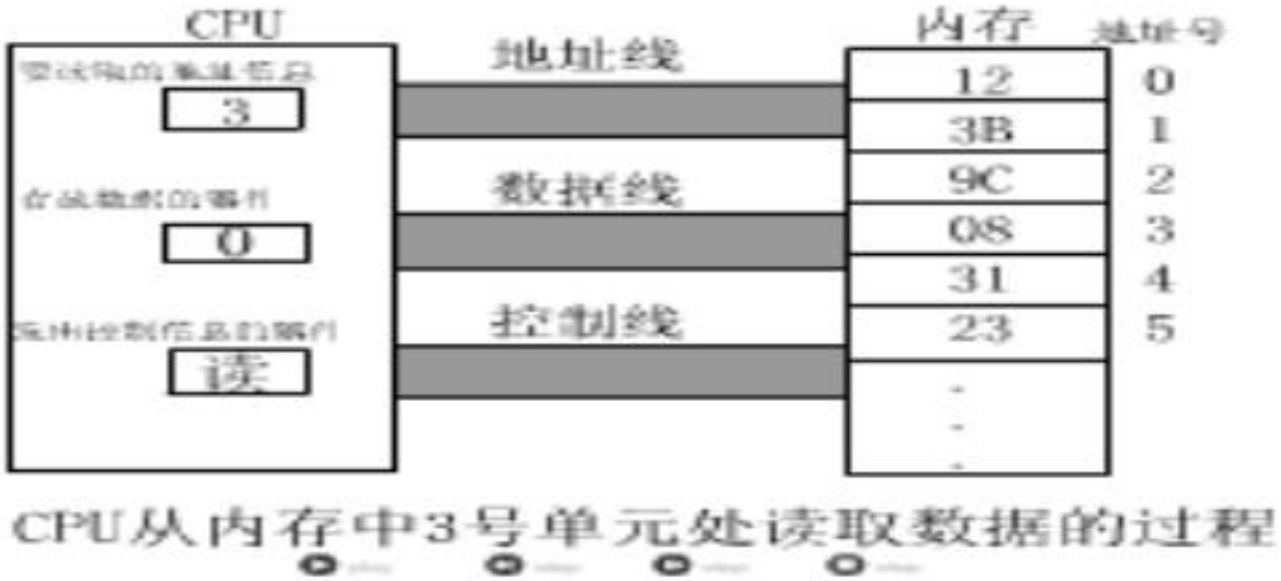
- 控制总线用来控制CPU与内存和I/O设备之间的数据传送方式（如传送方向）。
- 有多少根控制总线，就意味着CPU提供了对外部器件的多少种控制。
- 控制总线的宽度决定了CPU对外部器件的控制能力。



CPU对存储器的读写

CPU对存储器的读写必须通过三类总线

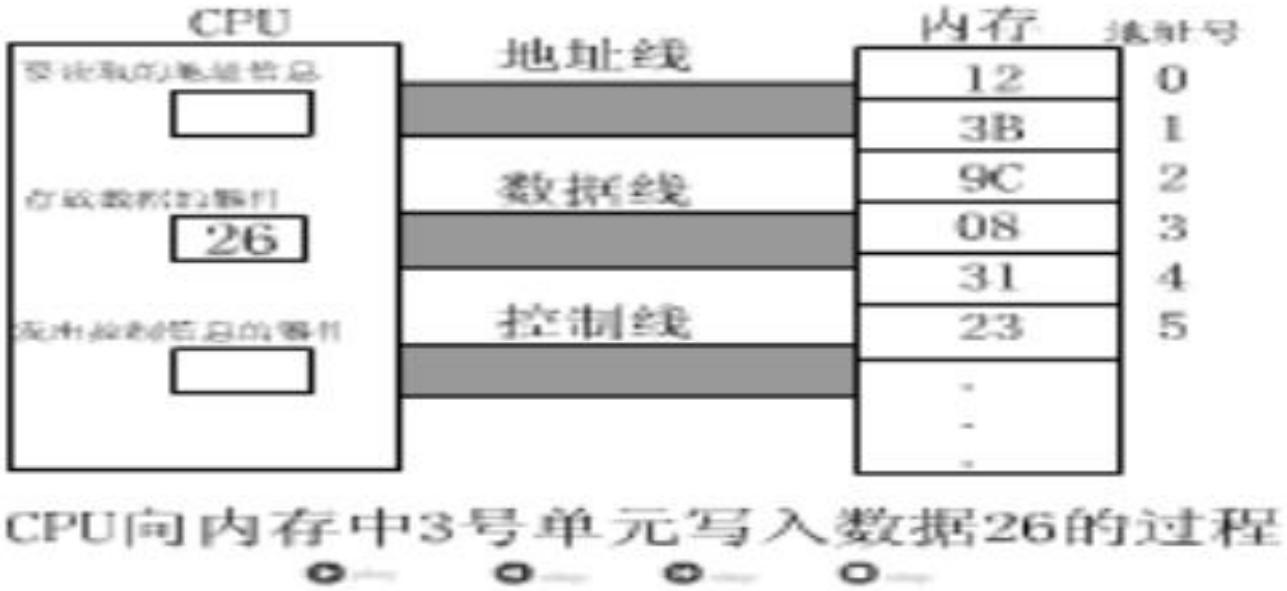
- 读取数据



CPU从内存中3号单元处读取数据的过程

CPU对存储器的读写

写入数据



CPU向内存中3号单元写入数据26的过程

以上知道CPU是如何进行数据读写的。可是我们如何命令计算机进行数据的读写呢？ 汇编语言编程通过指令实现

内存地址空间

- 上述存储器在物理上是独立的器件。但以下两点相同：

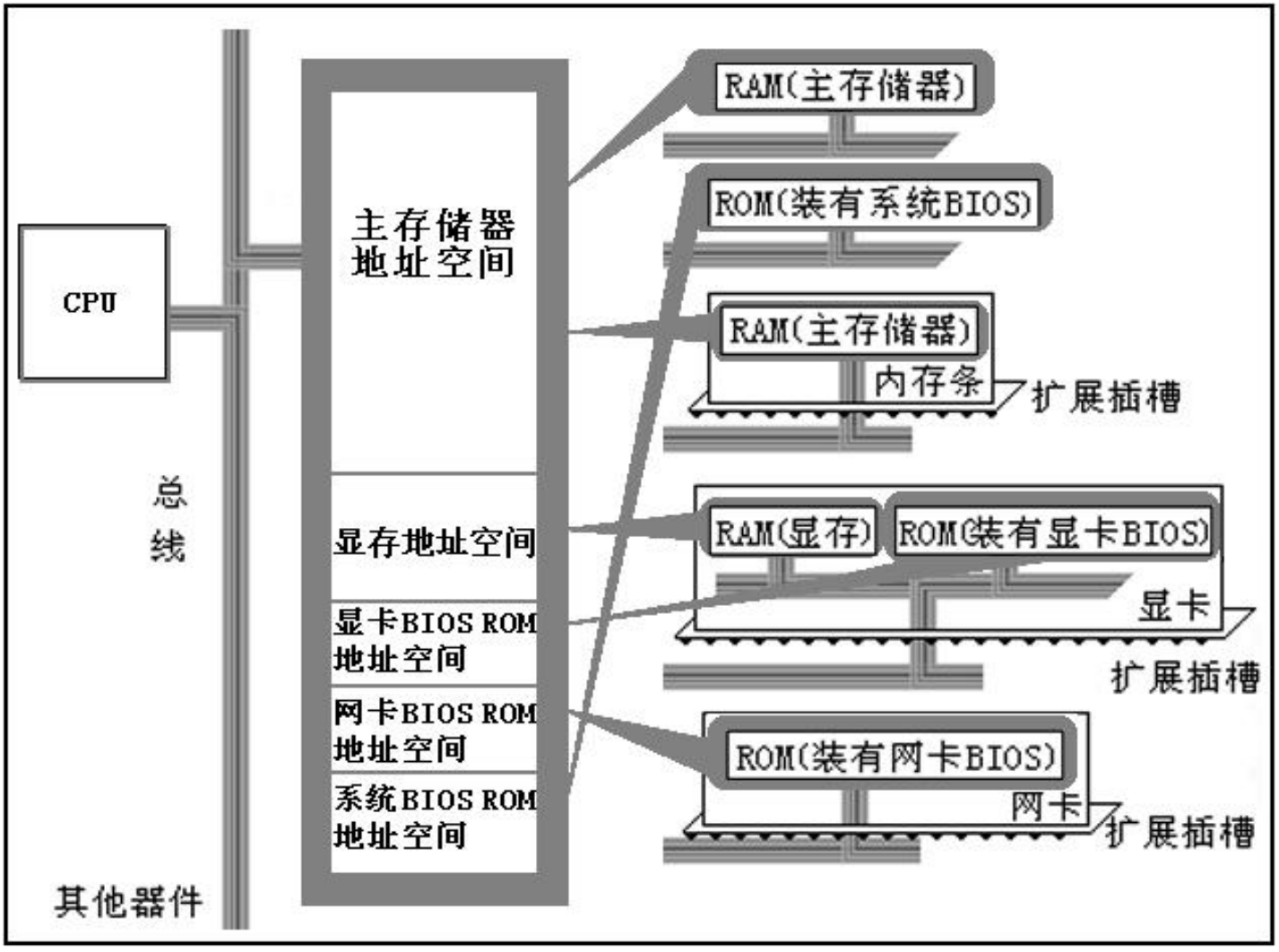
1、都和CPU的总线相连。

2、CPU对它们进行读或写的时候都通过控制线发出内存读写命令。

- **CPU操作这些存储器时把它们看作一个逻辑存储器：**
  - 所有的物理存储器被看作一个由若干存储单元组成的逻辑存储器；
  - 每个物理存储器在这个逻辑存储器中占有一个地址段，即一段地址空间；
  - CPU在这段地址空间中读写数据，实际上就是在相对应的物理存储器中读写数据。

内存地址空间

- 将各类存储器看作一个逻辑存储器



将各类存储器看作一个逻辑存储器

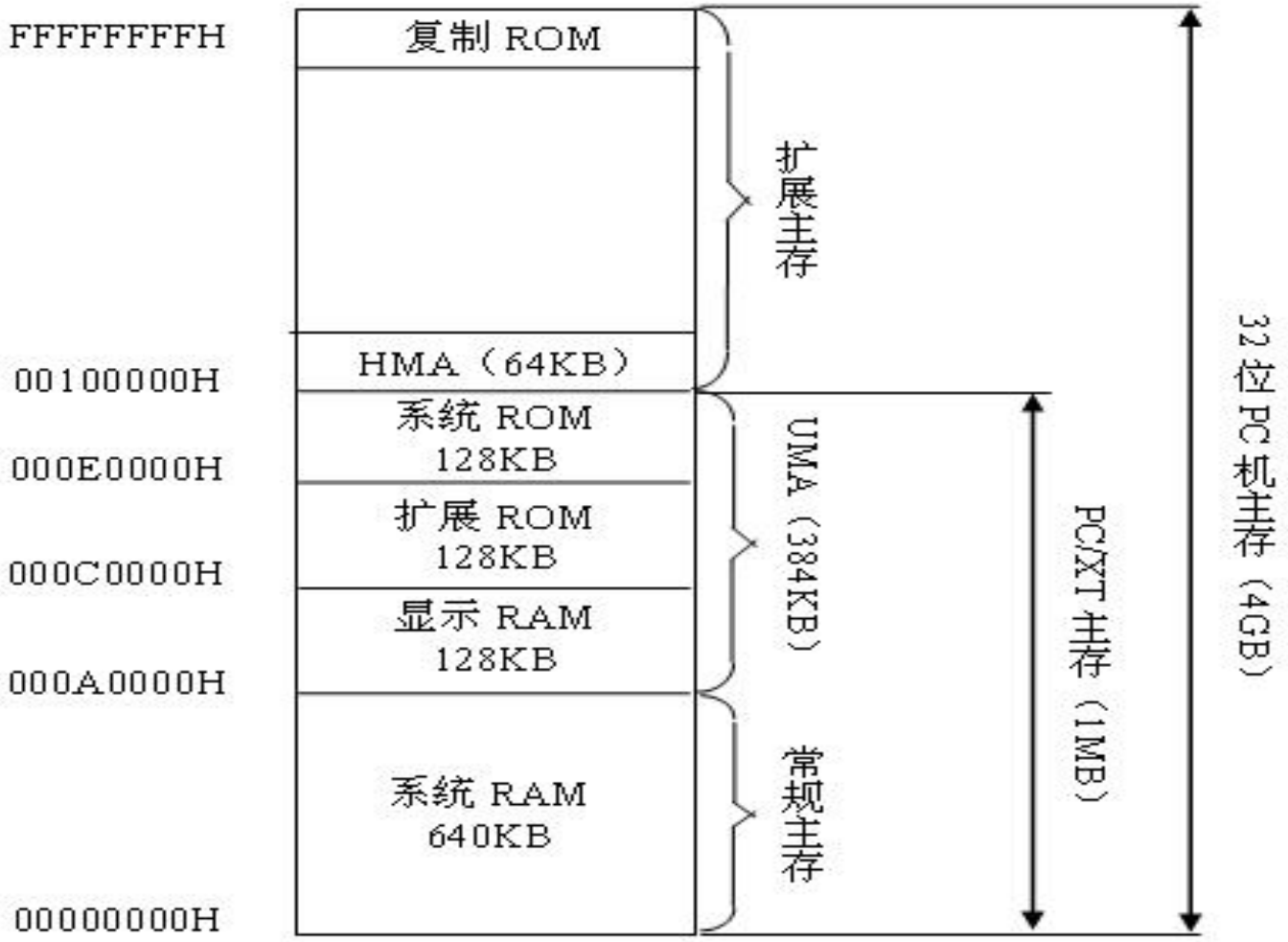
内存地址空间

- 最终运行程序的是CPU，我们用汇编编程的时候，必须要从CPU角度考虑问题。
- 对CPU来讲，系统中的所有存储器中的存储单元都处于一个统一的逻辑存储器中，它的容量受CPU寻址能力的限制。

  这个逻辑存储器即是我们所说的内存地址空间。
- 一个CPU的地址线宽度为10，那么可以寻址1024个内存单元，这1024个可寻到的内存单元就构成这个CPU的内存地址空间。

内存地址空间

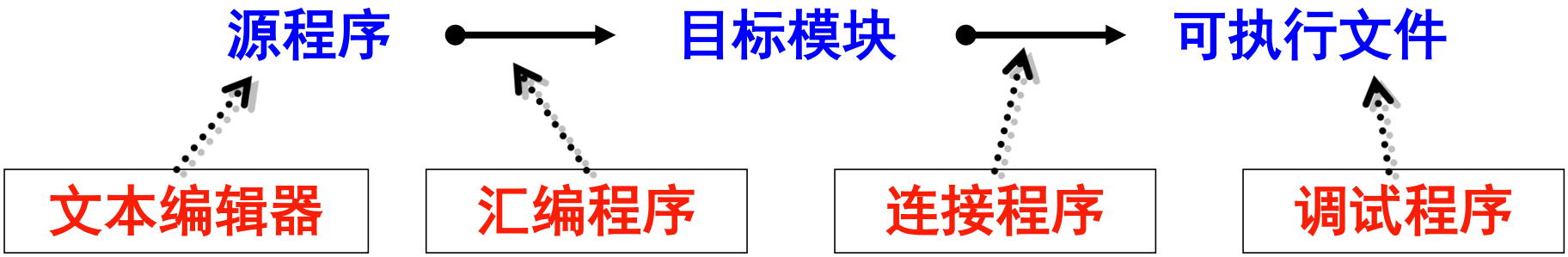
- 不同计算机系统的内存地址空间分配情况是不同的。



32位8086微处理器的内存地址空间

微机的软件

- **系统软件：DOS平台**
  - MS-DOS 6.22实地址方式
  - Windows的MS-DOS模拟环境(64位系统下Dosbox)
- **应用软件：开发汇编语言程序涉及**
  - 文本编辑器：Notepad++
  - 汇编（编译）程序：MASM 6.15
  - 连接程序：Link.exe
  - 调试程序：Debug
  - 集成化开发环境:Visual Studio



- **工具介绍**
- **参照视频：**汇编语言从0开始 重制版 自学必备(配套王爽汇编语言第三版或第四版)

2.1-2.4

https://www.bilibili.com/video/BV1mt411R7Xv/?spm_id_from=333.999.0.0&vd_source=67493d02f55cc832a7436a674b79e7d8

第一章 基础知识

1.1 数的表示

1.2 微型计算机（PC）系统

1.3 Intel 80x86系列微处理器

1.4 8086微处理器

1.5 8086的寻址方式

§3 Intel 80x86系列微处理器

Intel 8086

Intel 64处理器

酷睿多核系列

奔腾多核系列

奔腾4

IA-32处理器

奔腾III

奔腾II

奔腾

80486

80386

80286

8086

4004

16位80x86处理器

2.1 16位80x86处理器

- 16位结构处理器
- 8086/8088指令系统提供16位基本指令集
- 80186/80188增加若干条实用指令
- 8086的工作方式是实方式（Real Mode）
- 80286增加保护方式（Protected Mode）
- 80286引入了系统指令
  - 为操作系统等核心程序提供处理器控制功能

指令系统、指令集（Instruction Set）

2.2 IA-32处理器

- **80386引入英特尔32位指令集结构ISA**
  - 兼容原16位80286指令系统
  - 全面升级为32位
  - 提供虚拟8086工作方式（Virtual 8086 Mode）
- **80486集成浮点处理单元支持浮点指令**
- **Pentium系列**
  - 陆续增加若干整数指令、完善浮点指令
  - 增加一系列多媒体指令（SIMD指令）

IA-32（Intel Architecture-32）

2.3 Intel 64处理器

- 引入64位英特尔指令集结构
  - 兼容32位指令系统
  - 新增64位工作方式
- 继续丰富多媒体指令
- 处理器集成多核（Multi-core）技术

Many core

处理器进入多核时代

第一章 基础知识

1.1 数的表示

1.2 微型计算机（PC）系统

1.3 Intel 80x86系列微处理器

1.4 8086微处理器

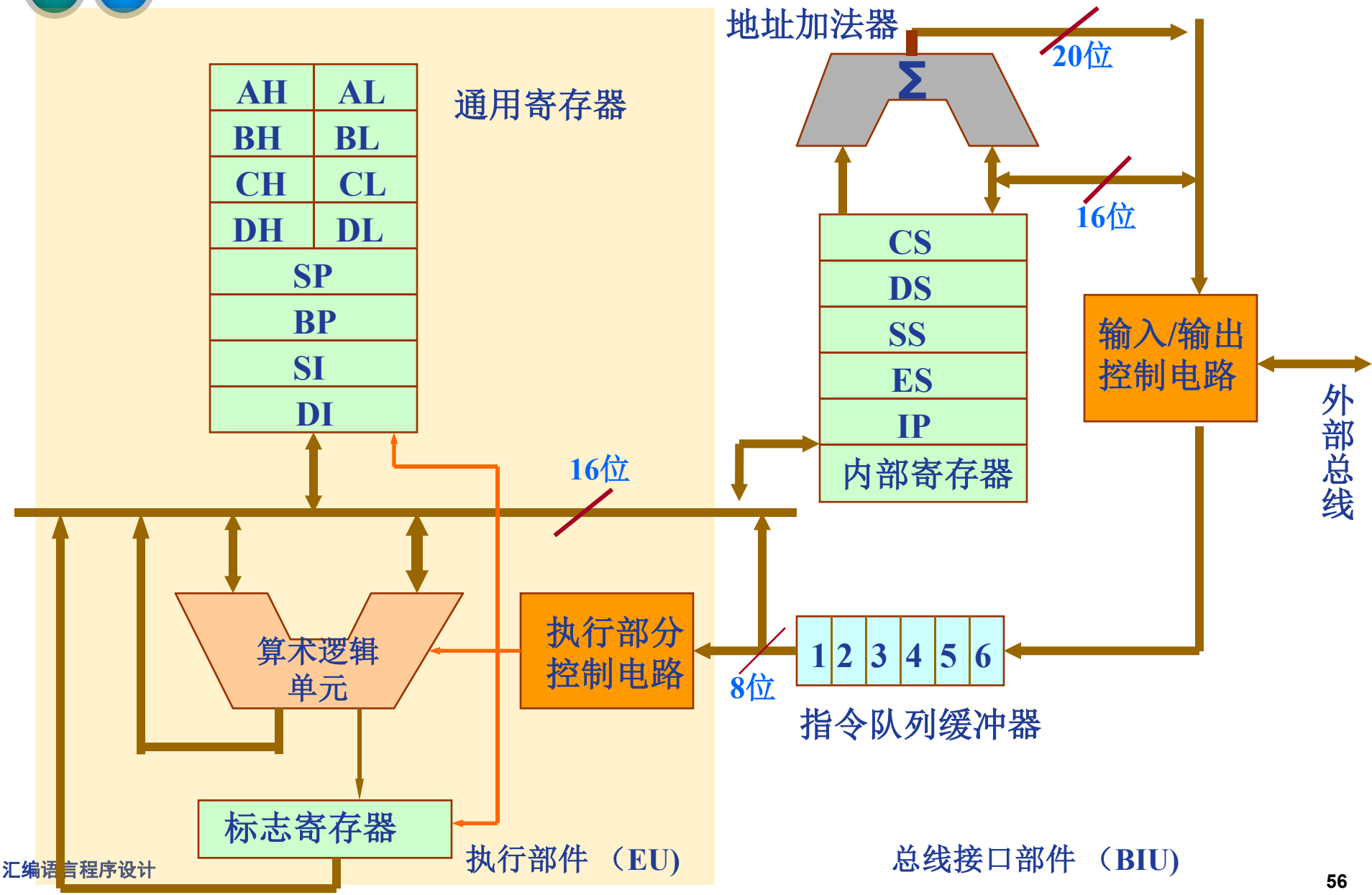
1.5 8086的寻址方式

4.1 8086的内部结构

- **组成：**
  - 算术逻辑部件、控制部件和寄存器组。
- **CPU的作用：**
  - 执行算术与逻辑运算。
  - 控制指令的执行。

- 这些器件靠内部总线相连。内部总线实现CPU内部各个器件之间的联系。
- 外部总线（系统总线）实现CPU和主板上其它器件的联系。

4.1 8086的内部结构



4.1 8086的内部结构

- 执行单元EU

  负责指令的译码、执行和数据的运算

- 总线接口单元BIU

管理8086与外部总线的接口，负责CPU对存储器和外设进行访问

- 对汇编语言程序员来说，CPU通过寄存器完成指令的取指和执行功能。

4.2 8086 存储器组织

存储器的分段技术

- CPU访问内存单元时要给出内存单元的地址。所有的内存单元构成的存储空间是一个一维的线性空间。
- 每一个内存单元在这个空间中都有唯一的地址，这个唯一的地址称为物理地址。

4.2 8086 存储器组织

- **16位结构描述了一个CPU具有以下特征：**
  - 1、运算器一次最多可以处理16位的数据。
  - 2、寄存器的最大宽度为16位。
  - 3、寄存器和运算器之间的通路是16位的。

- 8086有20位地址总线，可传送20位地址，寻址能力为1M。
- 8086内部为16位结构，它只能传送16位的地址，表现出的寻址能力却只有64K。

如何给出20位的物理地址？

分段技术

地址加法器合成物理地址的方法：用两个16位地址合成一个20位的物理地址。

物理地址=段地址×16+偏移地址

分段技术

- 一张可以容纳 4 位数据的纸条——2826这个数据；
- 没有能容纳4位数据的纸条，仅有两张可以容纳3位数据的纸条。

可以写下四位数据的纸条

2	8	2	6

两张可以写下3位数据的纸条

2	0	0

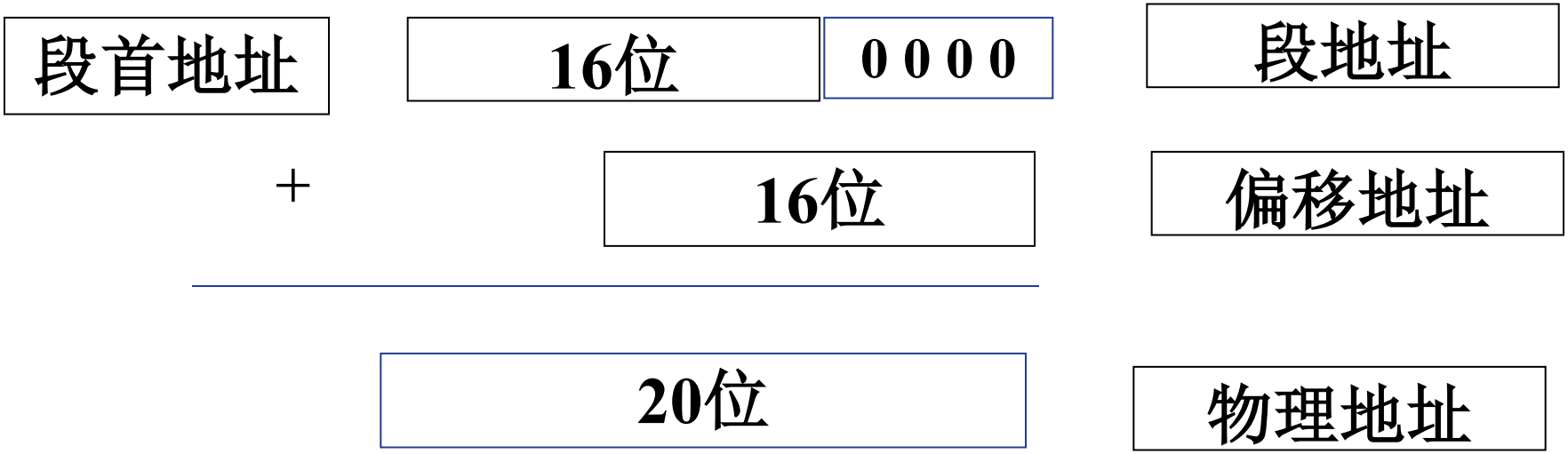
8	2	6

分段技术

- 将存储器分成若干个逻辑段
- 段首地址必须为：****0H。其有效地址“****H”存放在段寄存器中，称为段地址。
- 段中某一个单元相对于段首的距离称为偏移地址，偏移地址存放在偏移地址寄存器中。
- 段的长度不超过2^{16}=64K。

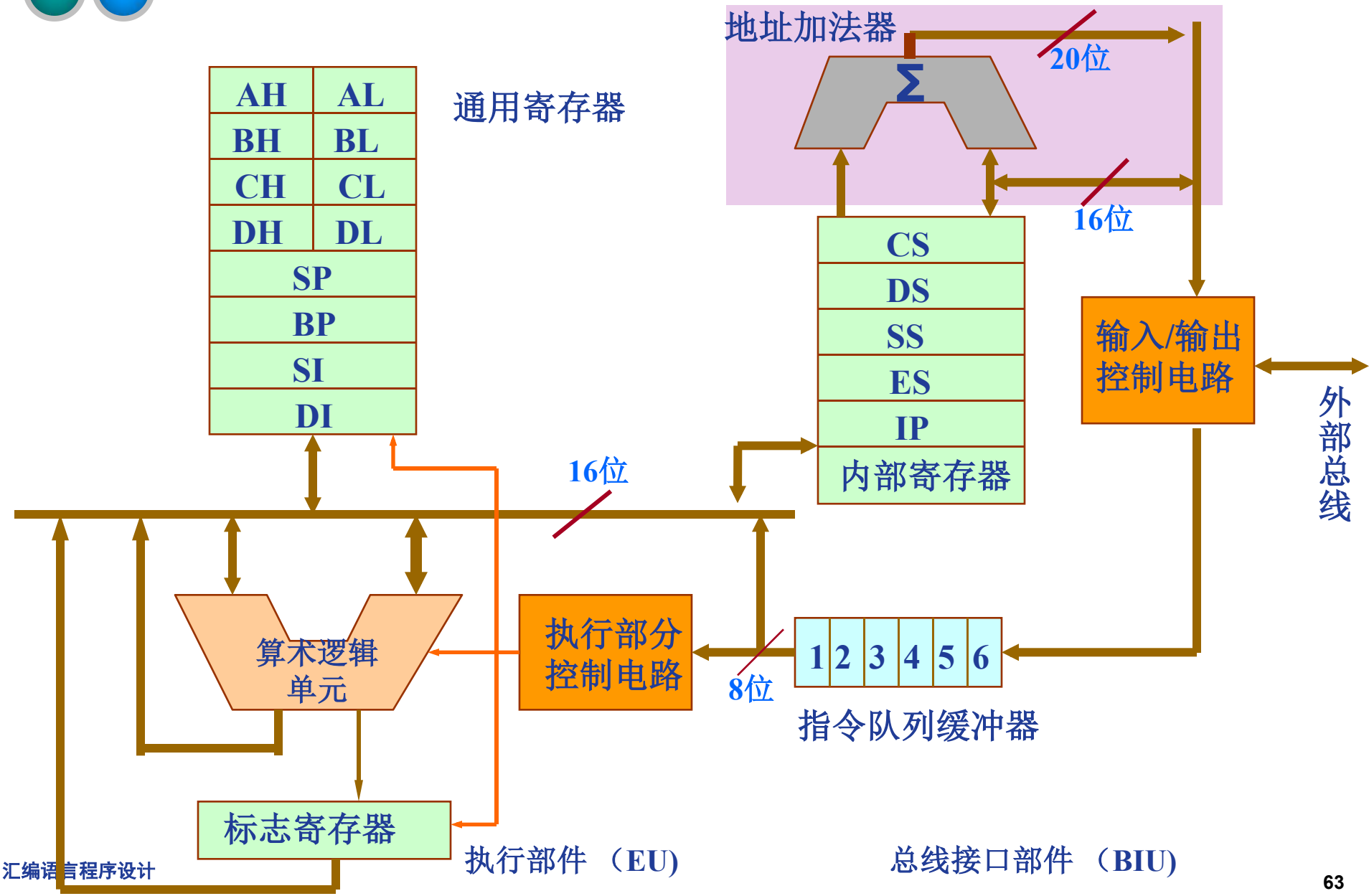
分段技术

- 地址加法器如何完成段地址×16的运算？
  - 一个数据的二进制形式左移1位，相当于该数据乘以2；
  - 一个数据的二进制形式左移N位，相当于该数据乘以2的N次方；
  - 以二进制形式存放的段地址左移4位。



- **物理地址 = 10H ×段地址 + 偏移地址**

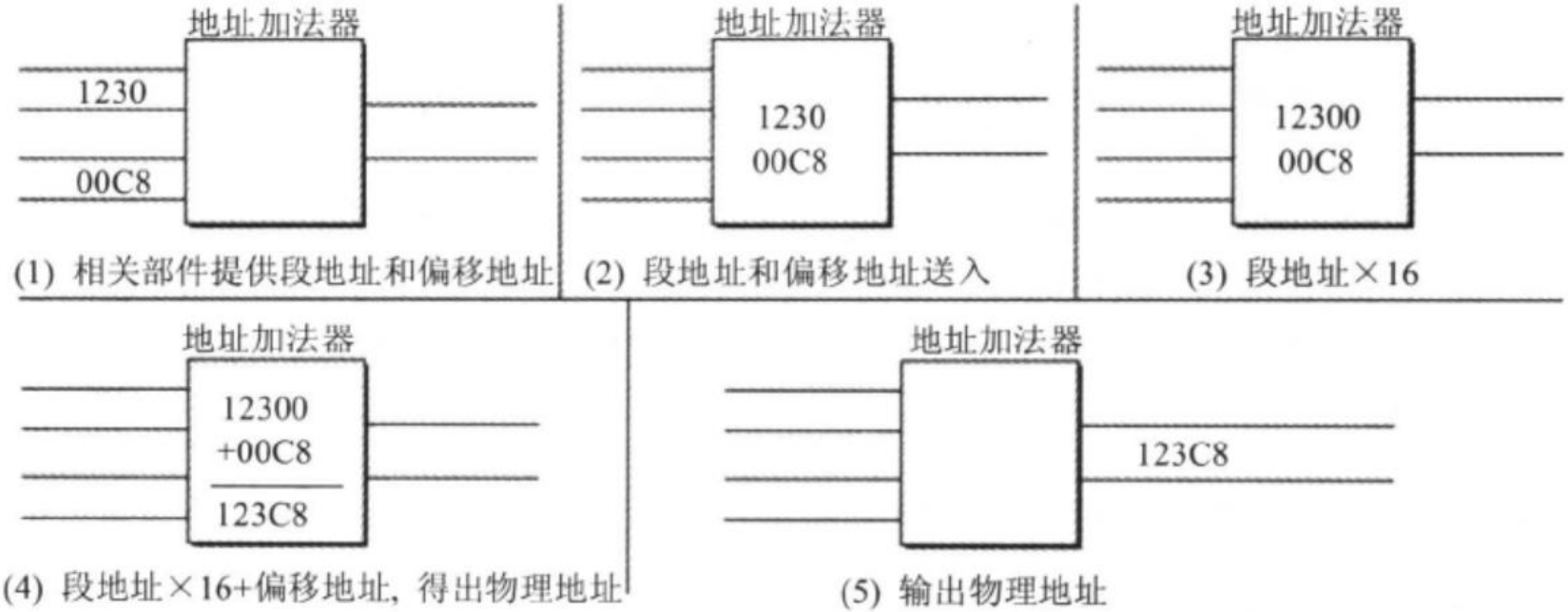
8086的内部结构


地址加法器
Σ
20位
16位
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
通用寄存器
CS
DS
SS
ES
IP
内部寄存器
输入/输出
控制电路
外部总线
16位
算术逻辑
单元
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
执行部件 （EU)
总线接口部件 （BIU)

CPU形成物理地址的过程

8086CPU给出物理地址的方法

- ALU完成加法，地址加法器
- 段地址一般在程序开始时预定
- 访问某一个内存单元，程序中只需要给出16位偏移地址



分段技术小结

- 内存并没有分段，段的划分来自于CPU，由于8086CPU用“（段地址×16）+偏移地址=物理地址”的方式给出内存单元的物理地址，使得我们可以用分段的方式来管理内存。
- 段地址×16 必然是 16的倍数，所以一个段的起始地址也一定是16的倍数；可根据需要，将地址连续、起始地址为16的倍数的一组内存单元定义为一个段。
- 偏移地址为16位，16 位地址的寻址能力为 64K，所以一个段的长度最大为64K。也就是说给定一个段地址，仅通过变化偏移地址来进行寻址，最多可以定位64K个内存单元。 0~FFFFH
- CPU访问内存单元时，必须向内存提供内存单元的最终物理地址=段地址+偏移地址
- CPU可以用不同的段地址

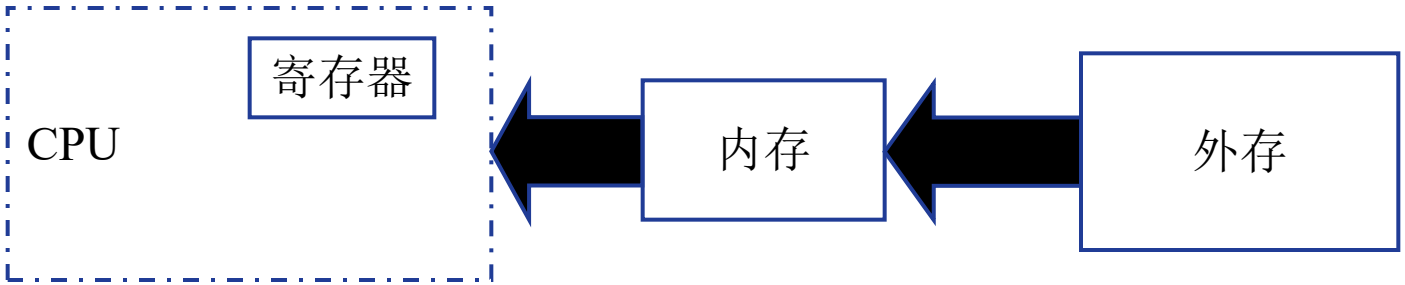
和偏移地址形成同一个物理地址。

物理地址	段地址	偏移地址
21F60H	2000H	1F60H
	2100H	0F60H
	21F0H	0060H
	21F6H	0000H
	1F00H	2F60H

4.3 8086 的寄存器组

对汇编语言程序员来说，关心的是CPU的寄存器。

- 8086 / 8088中共有14个16位寄存器
- 寄存器在CPU内部，所以访问速度快。但容量小

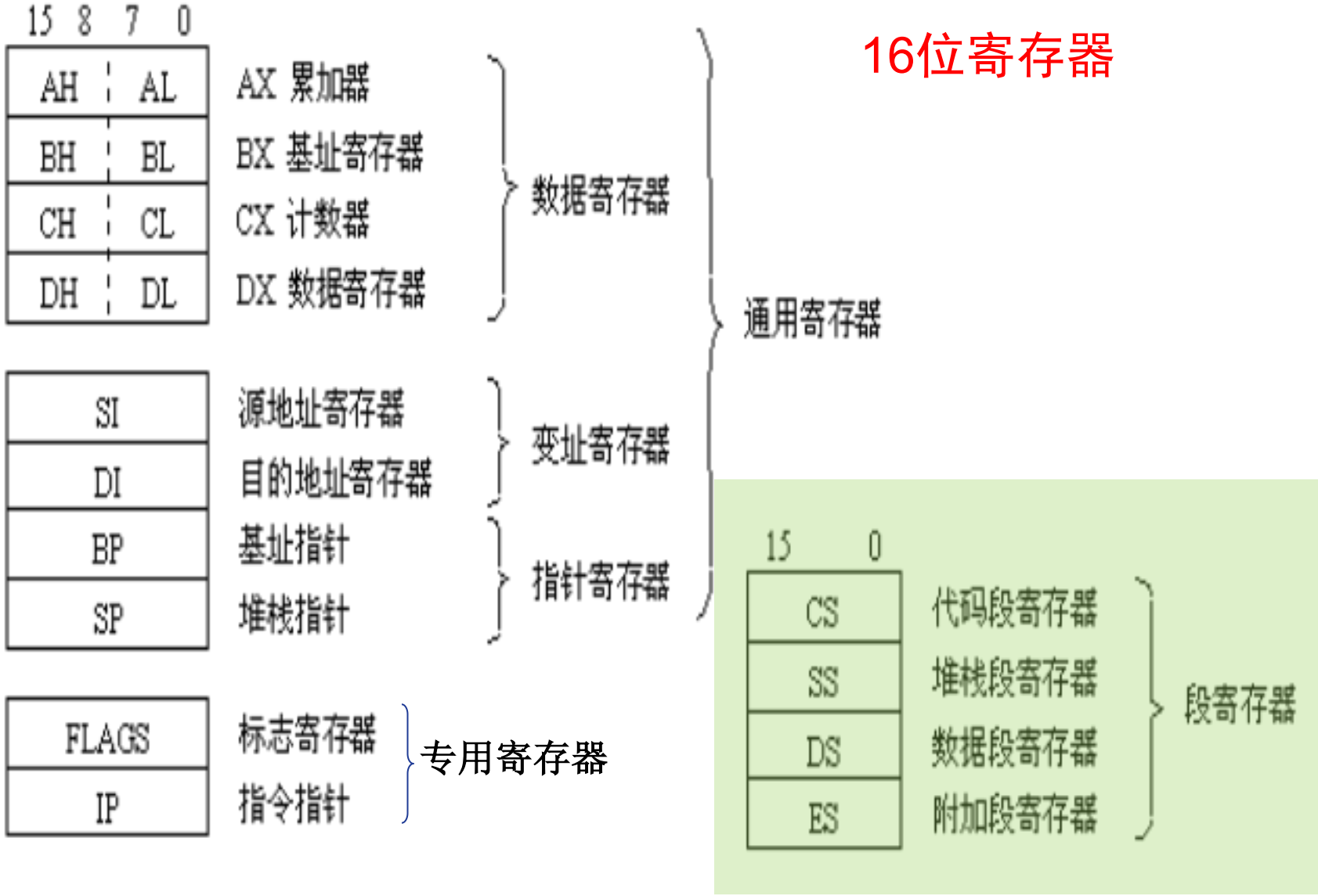


4.3 8086 的寄存器组

寄存器与存储器的比较：

寄 存 器	存 储 器
在CPU内部 访问速度快 容量小，成本高 用名字表示 没有地址	在CPU外部 访问速度慢 容量大，成本低 用地址表示 地址可用各种方式形成

4.3 8086 的寄存器组

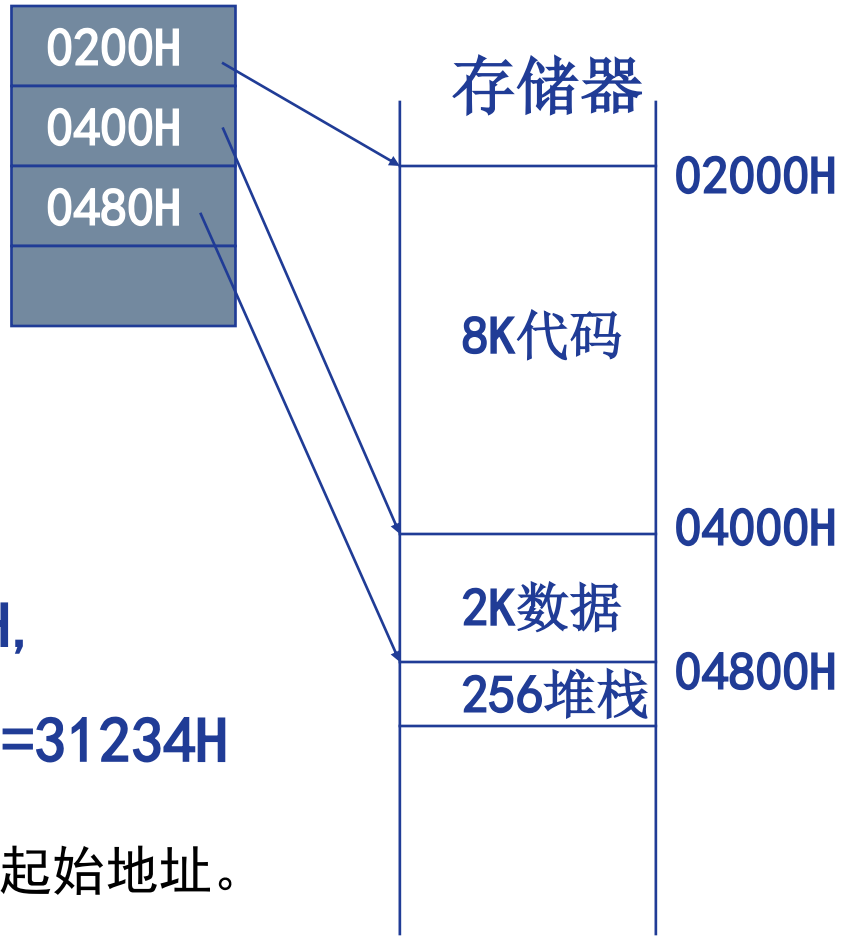


段寄存器:CS/DS/SS/ES

4个专门存放段地址的段寄存器（16位）

代码段　段寄存器 CS
数据段　段寄存器 DS
堆栈段　段寄存器 SS
附加段　段寄存器 ES

逻辑段



例：(DS)=3000H,　偏移=1234H,

物理地址= 10H × (DS) + 偏移 =31234H

- 段寄存器用来确定该段在内存中的起始地址。
- 用途特定，不可分开使用。

段寄存器:CS/DS/SS/ES

- **CPU几种典型的操作**
  - 取 指 令：指令单元地址＝（CS）×10H＋IP
  - 堆栈操作：堆栈数据地址＝（SS）×10H＋偏移
  - 内存数据：内存数据地址＝（DS）×10H＋偏移

CS+IP（Instruction Pointer）

- CS+IP是8086CPU中最关键的寄存器，它们指示了CPU当前要读取指令的地址。
- 指令指针寄存器IP，指示代码段中指令的偏移地址
- 它与代码段寄存器CS联用，确定下一条指令的物理地址
- IP寄存器是一个专用寄存器，程序一般不可直接使用。

8086CPU读取和执行指令演示

CS+IP

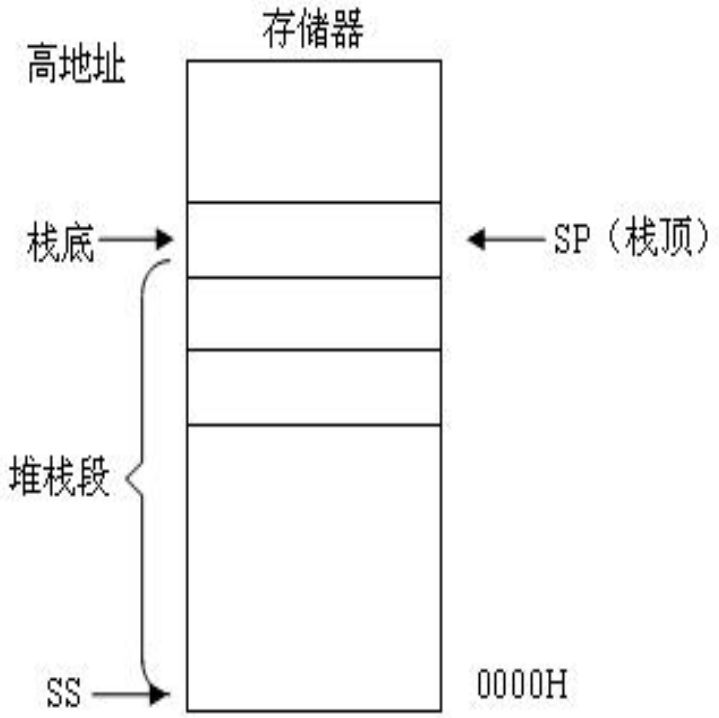
SS+指针寄存器:SP/BP

- 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
- 指针寄存器用于寻址内存堆栈内的数据
  - SP（Stack Pointer）为堆栈指针寄存器，指示栈顶的偏移地址
  - SP 不能再用于其他目的，具有专用目的
  - BP（Base Pointer）为基址指针寄存器，表示数据在堆栈段中的基地址

SS+SP(BP)

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



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
- CPU利用DS:偏移地址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:偏移地址存取附加段中的数据
- 串操作指令将附加段ES作为其目的操作数的存放区域

如何分配各个逻辑段

- 程序的指令序列必须安排在代码段CS
- 程序使用的堆栈一定在堆栈段SS
- 程序中的数据默认是安排在数据段DS，也经常安排在附加段ES，尤其是串操作的目的区必须是附加段ES
- 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

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

如何分配各个逻辑段

```
assume cs:code,ds:data,ss:stack
data segment
    dw 0123H,0456H,0789H,0abcH,0defh,0fedh,0cbah,0987H
data ends
stack segment
    dw 0,0,0,0,0,0,0,0
stack ends

code segment

start:  mov ax,stack
        mov ss,ax
        mov sp,16
        mov ax,data
        mov ds,ax
        push ds:[0]
        push ds:[2]
        pop ds:[2]
        pop ds:[0]
        mov ax,4c00h
        int 21h
code ends
end start
```

如何分配各个逻辑段

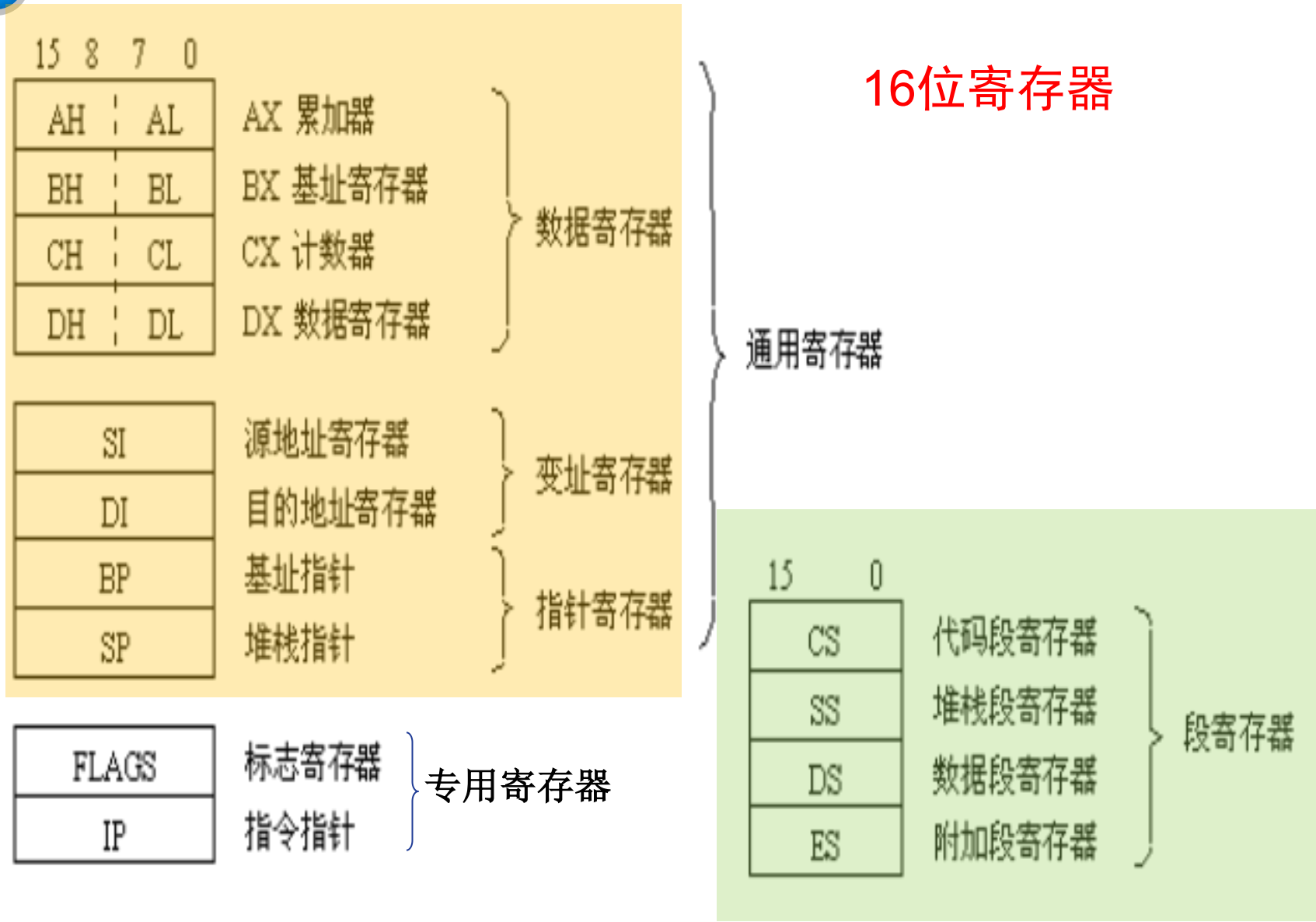
CPU如何处理我们定义的各种段？
依靠程序中的汇编指令和CS:IP\SS:SP\DS等寄存器的设置来确定。

① CPU 执行程序，程序返回前，data 段中的数据为多少？

② CPU 执行程序，程序返回前，cs=_____、ss=_____、ds=_____。

③ 设程序加载后，code 段的段地址为 X，则 data 段的段地址为_____，stack 段的段地址为_____。

4.3 8086 的寄存器组



数据寄存器：AX/BX/CX/DX

- 存放任何数据信息。暂存计算的中间结果，数据中转站。
- 每个寄存器又有它们各自的专用目的：

AX（Accumulator）——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；

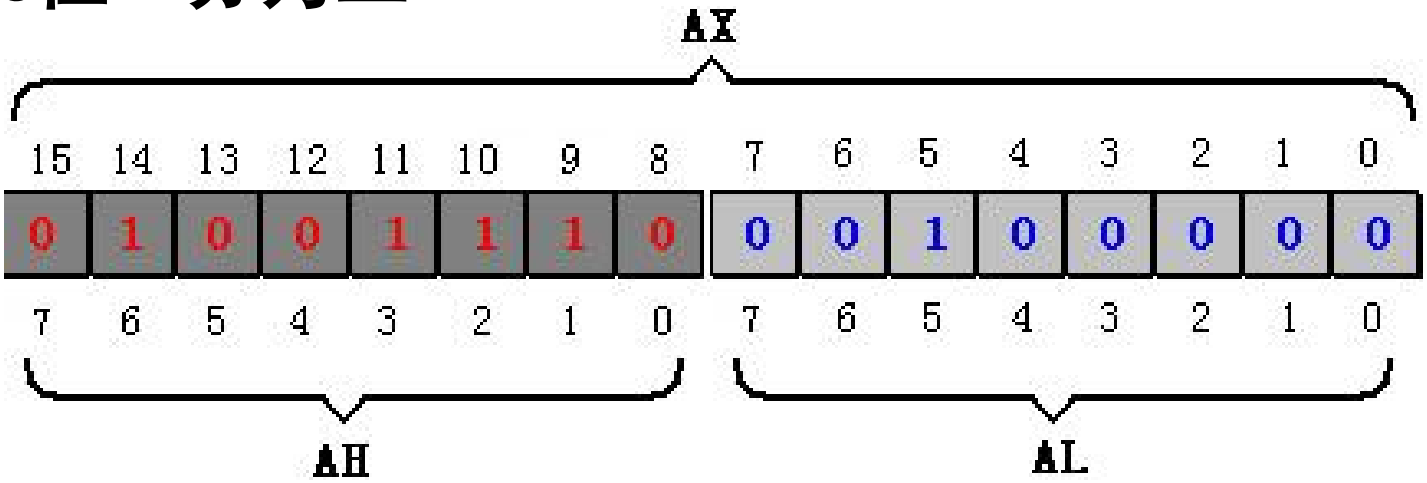
BX（Base）——基址寄存器，常用做存放存储器地址；

CX（Count）——计数器，作为循环和串操作等指令中的隐含计数器；

DX（Data）——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。

数据寄存器：AX/BX/CX/DX

- 16位一分为二



AH和AL寄存器是可以独立使用的8位寄存器。

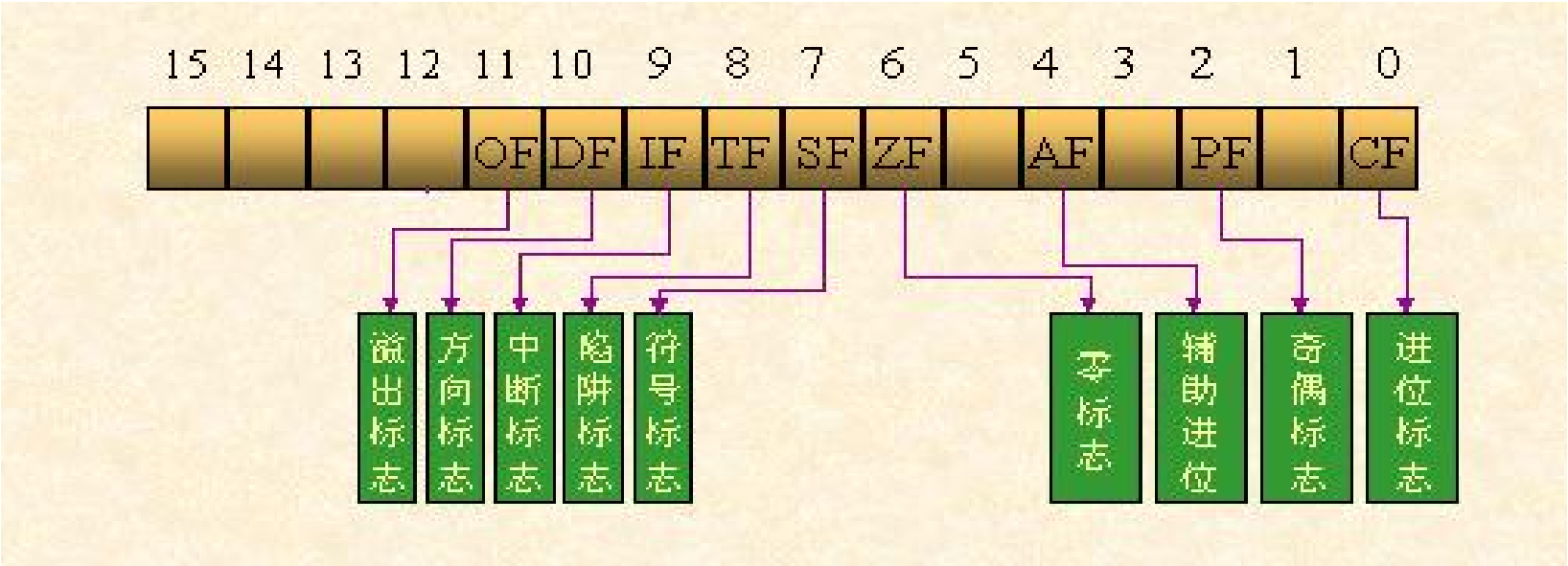
寄存器	寄存器中的数据	所表示的值
AX	0100111000100000	20000（4E20H）
AH	01001110	78（4EH）
AL	00100000	32（20H）

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
- 串操作类指令中，SI（Source Index）是源变址寄存器
- 串操作类指令中，DI（Destination Index）是目的变址寄存器
- 16位，不可拆分使用

标志寄存器

- 标志寄存器F(FLAGS)，又称程序状态字寄存器PSW
  - 状态标志位——由CPU根据当前程序运行结果的状态自动完成的。一般用作转移指令的转移控制条件。CF, OF, ZF, SF, PF, AF
  - 控制标志位——用以存放控制CPU工作方式的标志信息。由程序设置。DF, IF, TF



进位标志CF（Carry Flag）

- 进行无符号数运算的时候，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。

  例如：

  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

  例如：

  3AH + 7CH＝B6H，结果不是零：

  ZF = 0

  86H + 7AH＝（1）00H，结果是零(为什么？)：

  ZF = 1

  注意：ZF为1表示的结果是0

符号标志SF（Sign Flag）

CPU对有符号数运算结果的一种记录，记录数据的正负

- 结果为负，SF = 1；
- 结果为正，SF = 0。

例如：

```
mov al,10000001B
add al,1
```

执行后，结果为10000010B，SF=1，

表示：如果指令进行的是有符号数运算，那么结果为负；

注意：

- 在我们将数据当作有符号数来运算的时候，可以通过它来得知结果的正负。
- 如果我们将数据当作无符号数来运算，SF的值则没有意义，虽然相关的指令影响了它的值。

奇偶标志PF （Parity Flag）

- 当运算结果最低字节中“1”的个数为偶数时，PF = 1；否则PF = 0。

例如：

3AH + 7CH＝B6H＝10110110B，

结果中有5个1，是奇数：PF = 0

注意：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。

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。

例如：

3AH + 7CH＝B6H

D_3有进位：AF = 1

这个标志主要由处理器内部使用，用于十进制算术运算指令中，用户一般不必关心。（类似于进位标志）

溢出标志OF（Overflow Flag）

- 有符号数运算的结果有溢出，则OF=1；否则 OF＝0。
- 只是对有符号数而言。对无符号数而言，OF＝1并不意味着结果出错。

例如：

```
mov al,98
add al,99
```

执行后将产生溢出。

因为add al,99 进行的有符号数运算是：

(al)=(al)+99=98+99=197

而结果197超出了机器所能表示的8位有符号数的范围：-128~127。

程序运行结果是多少？

什么是溢出

- 处理器内部以补码表示有符号数
  - 8个二进制位能够表达的整数范围是：+127 ~ -128
  - 16位表达的范围是：+32767 ~ -32768
- 如果运算结果超出这个范围，就是产生了溢出
- 有溢出，说明有符号数的运算结果不正确
- 无符号数有溢出吗？
  - FFH + 01H = 00H, CF = 1, 进位

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CF表示无符号数运算结果是否超出范围，运算结果仍然正确；
  - 可恢复的错误。
- 溢出标志OF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。
  - 不可恢复错误。

溢出和进位的对比

例1：7FH + 01H＝80H

无符号数运算：127＋1＝128，　范围内，无进位

有符号数运算：　127＋1＝128 ，范围外，有溢出

例2：FFH + 01H＝（1）00H

无符号数运算：255＋1＝256，范围外，有进位

有符号数运算：－1＋1＝0 ，　范围内，无溢出

如何运用溢出和进位

- 处理器对两个操作数进行运算时，并不知道操作数是有符号数还是无符号数，所以全部设置，按各自规则。
- 应该利用哪个标志，则由程序员来决定。
  - 将参加运算的操作数是无符号数，就注意CF；
  - 将参加运算的操作数是有符号数，则注意OF。
- 我怎么知道是什么数？
  - 除了你没人知道，☺

如何运用溢出和进位

例： **MOV AX, 1**　　OF, CF, ZF, SF

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

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，串操作的存储器地址自动增加；
  - 设置DF＝1，串操作的存储器地址自动减少。

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。

陷阱标志TF（Trap Flag）

- **用于控制处理器是否进入单步操作方式：**
  - 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。**
  - 利用单步中断可对程序进行逐条指令的调试。
  - 这种逐条指令调试程序的方法就是单步调试。

第一章 基础知识

1.1 数的表示

1.2 微型计算机（PC）系统

1.3 Intel 80x86系列微处理器

1.4 8086微处理器

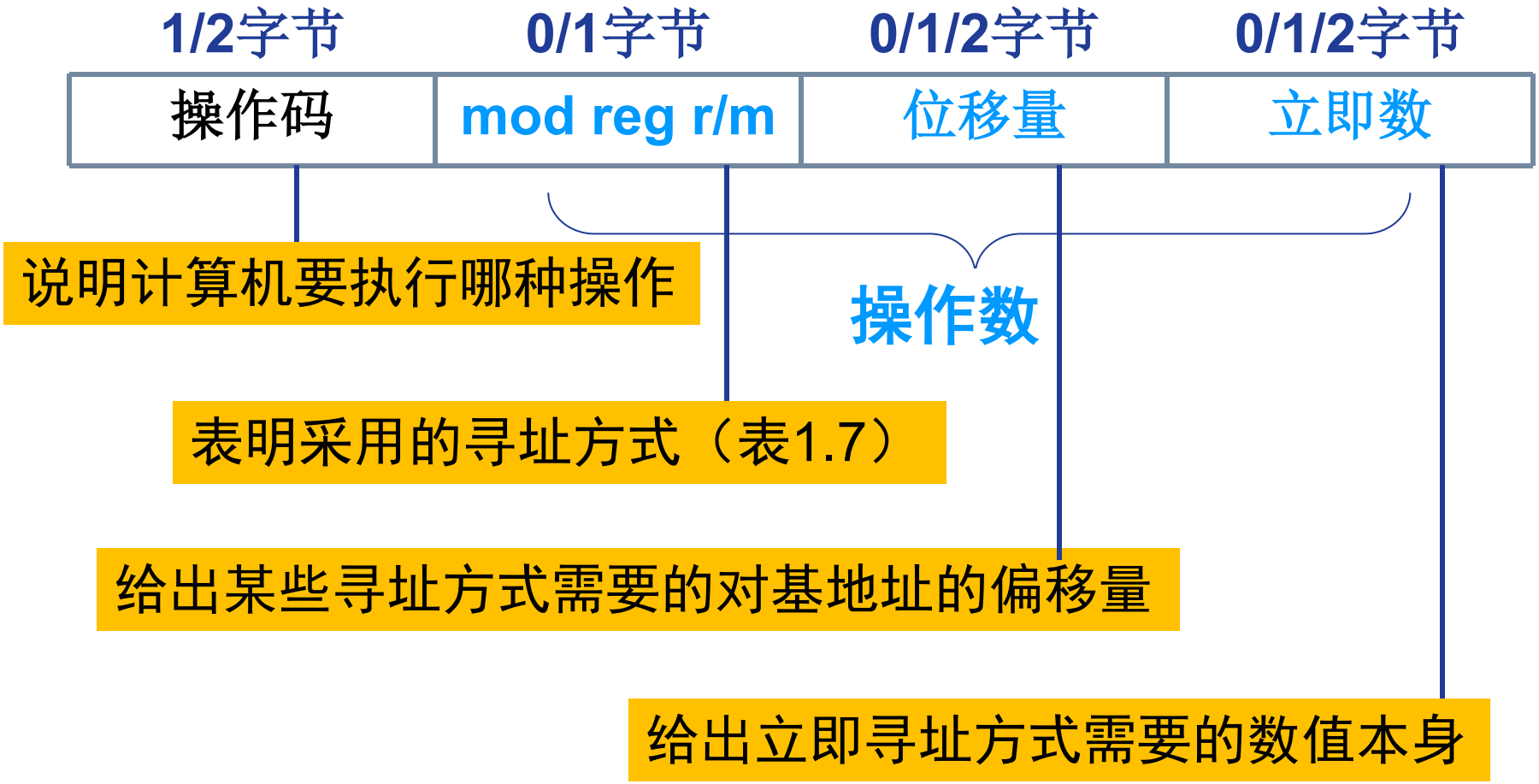
1.5 8086的寻址方式

§5 8086 寻址方式

- 8086机器码格式
- 指令格式
- 寻址方式

5.1 8086 机器码格式

- 机器码格式：将指令以2进制数0和1进行编码的形式



标准机器代码示例

操作码	操作数

mov al,05 ；机器代码是B0 05

- 前一个字节B0是操作码（含一个操作数AL），后一个字节05是立即数

mov ax,0102H ；机器代码是B8 02 01

- 前一个字节B8是操作码（含一个操作数AX），后两个字节02 01是16位立即数（低字节02在低地址）

5.2 指令格式

❖ 指令的一般格式

【标号：】	指令助记符	操作数1	操作数2	【；注释】

- 有些指令不需要操作数，通常的指令都有一个或两个操作数，个别指令有3个甚至4个操作数

- 标号表示该指令在主存中的逻辑地址
- 每个指令助记符就代表一种指令
- 目的和源操作数表示参与操作的对象
- 注释是对该指令或程序段功能的说明

5.2 指令格式

- 每种指令的操作码：
  - 用一个唯一的助记符表示（指令功能的英文缩写）
  - 对应着机器指令的一个二进制编码
- 指令中的操作数：
  - 立即操作数：一个具体的数值
  - 寄存器操作数：存放数据的寄存器
  - 内存操作数：指明数据在主存位置的存储器地址。

    通常为有效地址EA，段地址在某个段寄存器中。

5.3 8086寻址方式

- 立即寻址方式
- 寄存器寻址方式
- 内存操作数寻址方式

格式： MOV dest,src　　; dest←src

- **功能：将源操作数src传送至目的操作数dest**

立即寻址方式(immediate addressing)

- 操作数在指令中给出,作为指令机器码的一部分存储

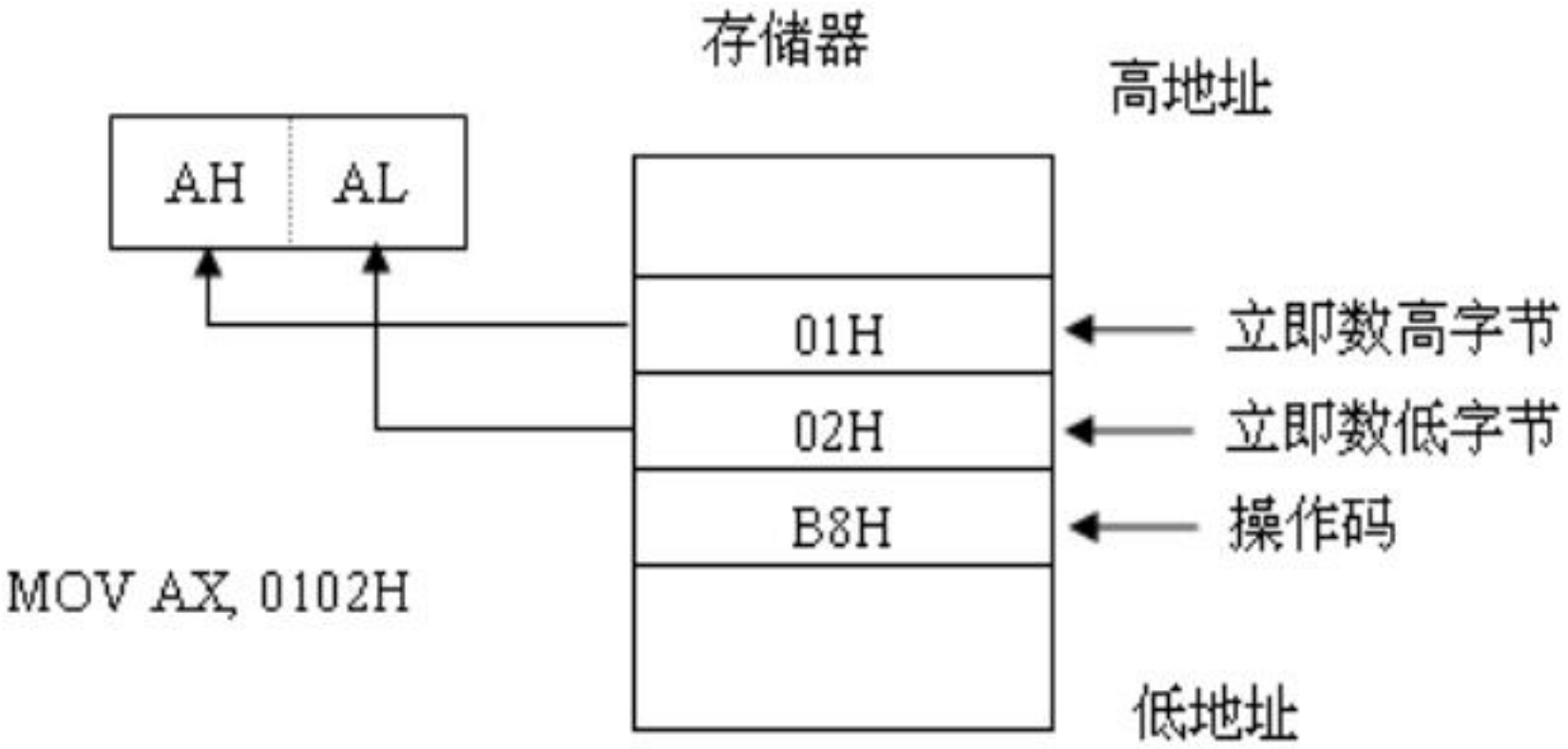
```
MOV   AL, 34H          ; 机器码：B034
MOV   AX, 0034H        ; 机器码：B83400
```

- 使用场合：常数，8位和16位。
- 立即数寻址方式常用来给寄存器赋值

```
MOV  AL,05H            ; AL←05H
MOV  AX,0102H          ; AX←0102H
```

立即寻址方式(immediate addressing)

MOV AX,0102H ；AX←0102H



寄存器寻址方式(register addressing)

- 操作数在指定的数据寄存器中

```
MOV   AX, BX
MOV   AL, BH
```

- 注意可用寄存器：
  - 8位寄存器r8：AH、AL、BH、BL、CH、CL、DH、DL
  - 16位寄存器r16：AX、BX、CX、DX、SI、DI、BP、SP
  - 4个段寄存器seg：CS、DS、SS、ES
  - CS/IP不能用作目的操作数：
    - MOV CS/IP, AX （X）

内存操作数寻址方式

指令中给出操作数的主存地址信息（偏移地址，称之为有效地址EA），而段地址在默认的或用段超越前缀指定的段寄存器中

- 直接寻址方式(direct addressing)
- 寄存器间接寻址方式(register indirect)
- 寄存器相对寻址方式(register relative)
- 基址变址寻址方式(based indexed..)
- 相对基址变址方式(relative based indexed..)

段超越

- 隐式段地址——8086/8088指令系统对存储单元的访问，其段地址都是从系统事先约定好的段寄存器中获取；
- 规则为：除串操作指令外，若出现BP（SP），默认在SS中，否则所有的操作都默认在DS中。
- （显式段地址）段超越——不是按照系统的约定，而是在指令中显式指定某一段寄存器作为存储器操作数的段地址。

直接寻址方式(direct addressing)

- 内存操作数的偏移地址由指令直接给出

```
MOV   AX, [2000H]              ; AX←DS:[2000H]
```

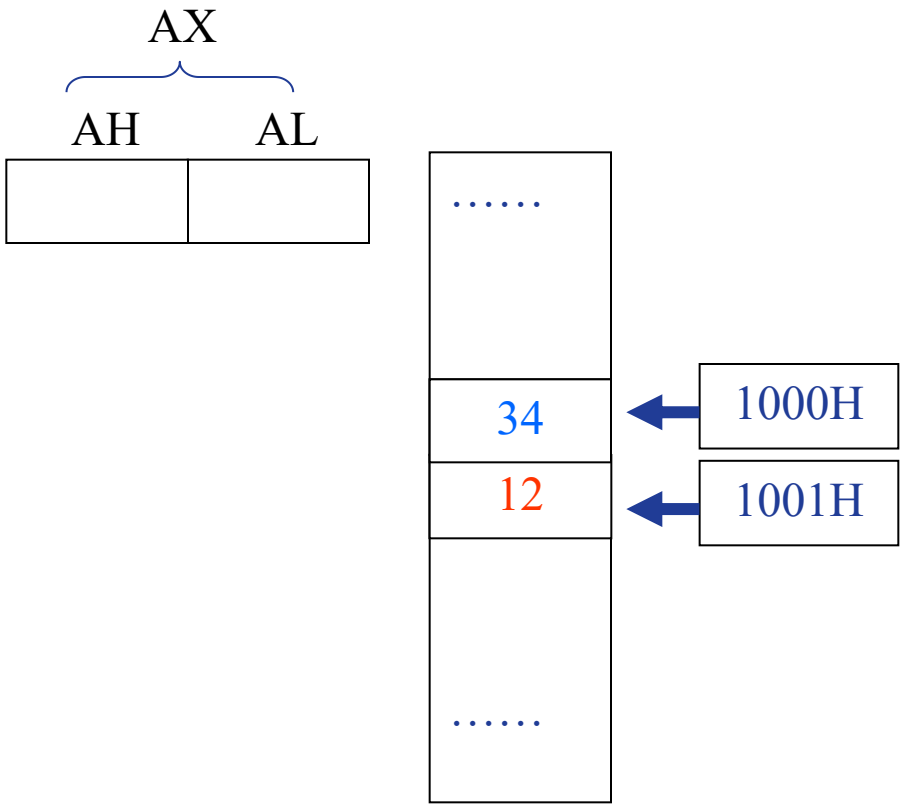
注意：

- 隐含的段为数据段 DS
- 物理地址 = 16 * (DS) +偏移地址

看动画片？：直接寻址过程

比　较

- **比较1**
  - MOV　AL, [1000H]
  - AL=34H;
  - MOV　AX, [1000H]
  - AX=1234H
- **比较2**
  - MOV　AX, 1000H
  - AX=1000H;
  - MOV　AX, [1000H]
  - AX=1234H



寄存器间接寻址方式(register indirect)

- 指定某个地址寄存器（SI、DI、BX、BP）的内容作为内存操作数的偏移地址

```
MOV AX, [BX]          ; AX←DS:[BX]
MOV [BP], AL
```

- 使用场合：表格、字符串、缓冲区处理
- 注意段地址规则：
  - BX, SI, DI →(DS)
  - BP →(SS)

看动画片：寄存器间接寻址过程

寄存器相对寻址方式(register relative)

- 指令中指定地址寄存器（SI、DI、BX、BP）与一个位移量相加作为内存操作数的偏移地址

看动画片

$$\text{偏移地址} = \begin{cases} (BX) \\ (BP) \\ (SI) \\ (DI) \end{cases} + \begin{cases} 8\text{位} \\ 16\text{位} \end{cases} \text{位移量}$$

```
MOV AX,[SI+2]          ; AX←DS:[SI+02H]
MOV AX,[BP+06H]        ; AX←SS:[BP+06H]
```

- BX/SI/DI对应段地址默认在DS，BP默认在SS；可用段超越前缀
- 使用场合：适于表格、字符串、缓冲区的处理；
- 一维数组方式(DATA[DI] ＝ [DATA + DI])

基址变址寻址方式(based indexed..)

- 指定基址寄存器(BX,BP)、变址寄存器(SI,DI)内容相加作为内存操作数的地址。

  看动画片

```
MOV [BX+DI],DX

MOV AL, [BP+SI]  ; AL←SS:[BP+SI]
```

- BX对应段地址默认在DS，BP默认在SS；可用段超越前缀
- 使用场合：适于数组、字符串、表格的处理，更加灵活
- 注意：必须是一个基址寄存器和一个变址寄存器的组合
  - MOV AX, [BX][BP] (X)
  - MOV AX, [SI][DI] (X)

相对基址变址方式(relative based indexed.)

- 指定基址寄存器(BX,BP) 、变址寄存器(SI,DI) 、位移量之和作为内存操作数的地址

$$\text{偏移地址} = \begin{cases}(BX)\\(BP)\end{cases} + \begin{cases}(SI)\\(DI)\end{cases} + \begin{cases}8\text{位}\\16\text{位}\end{cases} \text{位移量}$$

```
MOV AL, [SI+BX+2]          MOV [BX+DI-16H], DX
MOV AL, 2[SI+BX]           ; AL←DS:[BX+SI+02H]
```

- BX对应段地址默认在DS，BP默认在SS；可用段超越前缀
- 使用场合：适于二维数组的寻址

看动画片

(Buffer[BX][SI] = [Buffer+BX+SI])

寻址方式的多种表示方式

- 位移量可用符号表示：

```
MOV AX,[SI+COUNT]
; COUNT是事先定义的变量或常量（就是数值）
MOV AX,[BX+SI+WNUM]     ; WNUM是变量或常量
```

- 同一寻址方式可以写成不同的形式：

```
MOV AX,[BX][SI]                 ; MOV AX,[BX+SI]
MOV AX,COUNT[SI]                ; MOV AX,[SI+COUNT]
MOV AX,WNUM[BX][SI]
; 等同于   MOV AX,WNUM[BX+SI]
; 等同于   MOV AX,[BX+SI+WNUM]
```

第1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
4. 掌握8086的寻址方式

习题

1.7、1.8、1.11、1.19、1.24

课后训练

1、在debug中观察AX的值？

mov ax，001AH

mov bx，001AH

add al, bl

add ah, bl

add bh, al

mov ah, 0

add al, 85H

add al, 93H

如何使用debug命令？