

汇编语言程序设计

lesson 1

课程说明

- 课堂讲授16学时，实验课16学时。
- 成绩=平时40%+考试60%
- 本课程**忌讳死记硬背，重视理解！**
- 教材：王爽，《汇编语言（第3版）》
（基于x86-16,32位汇编）
- 课堂纪律

注意:请自行复习以下内容

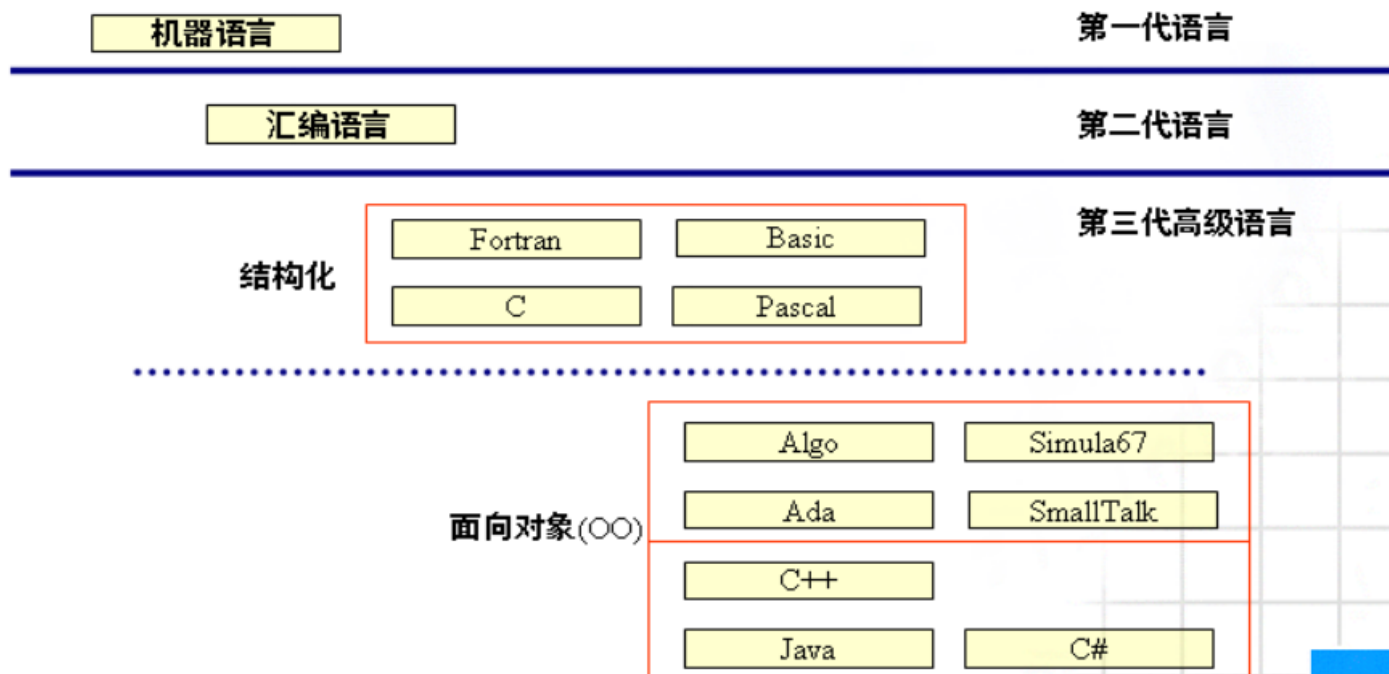
- 进制转换:
 - 二进制
 - 十进制
 - 十六进制
 - 八进制
- 基本逻辑运算

第1章 基础知识

- 1.1 机器语言
- 1.2 汇编语言
- 1.3 存储器
- 1.4 地址总线
- 1.5 数据总线
- 1.6 控制总线
- 1.7 其他器件
- 1.8 内存地址空间

引言

■ 程序设计语言的发展



引言

- 什么是汇编语言？为什么要学汇编语言？
- 汇编语言是直接硬件之上工作的编程语言。汇编课程的研究重点放在如何利用硬件系统的编程结构和指令集有效灵活的控制系统进行工作。

引言

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int i=5,j=5,q,p;
```

```
    p=(i++)+(i++)+(i++);
```

```
    q=(++j)+(++j)+(++j);
```

```
    printf(“%d\n,&d\n,%d\n,%d\n”,p,q,i,j);
```

```
}
```

问： $q=6+7+8=21$?

有的编译器中，q会
得出等于22

引言

0040150D mov ecx,dword ptr [ebp-8]

00401060 add ecx,1

00401063 mov dword ptr [ebp-8],ecx

00401066 mov edx,dword ptr [ebp-8]

00401069 add edx,1

0040106C mov dword ptr [ebp-8],edx

0040106F mov eax,dword ptr [ebp-8]

00401072 add eax,dword ptr [ebp-8]

00401075 mov ecx,dword ptr [ebp-8]

00401078 add ecx,1

0040107B mov dword ptr [ebp-8],ecx

0040107E add eax,dword ptr [ebp-8]

00401081 mov dword ptr [ebp-0ch],eax

上一段程序反编译后

引言

- 1.学习和使用汇编语言可以从根本上认识、理解计算机的工作过程。
- 通过用汇编语言编制程序可以更清楚地了解计算机是如何完成各种复杂的工作。在此基础上，程序设计人员能更充分地利用机器硬件的全部功能，发挥机器的长处。
- 2. 在计算机系统中，某些功能必须用汇编语言程序来实现。
- 比如：机器自检、系统初始化、实际的输入输出设备的操作等。

1.1 机器语言

- 机器语言是机器指令的集合，是一台机器可以正确执行的命令。
- 程序员们将 0、1 数字编程的程序代码打在纸带或卡片上，1打孔，0不打孔，再将程序通过纸带机或卡片机输入计算机，进行运算。
- 示例
应用8086CPU完成运算：

$$S = 768 + 12288 - 1280$$

1.1 机器语言

- $S = 768 + 12288 - 1280$

- 机器码:

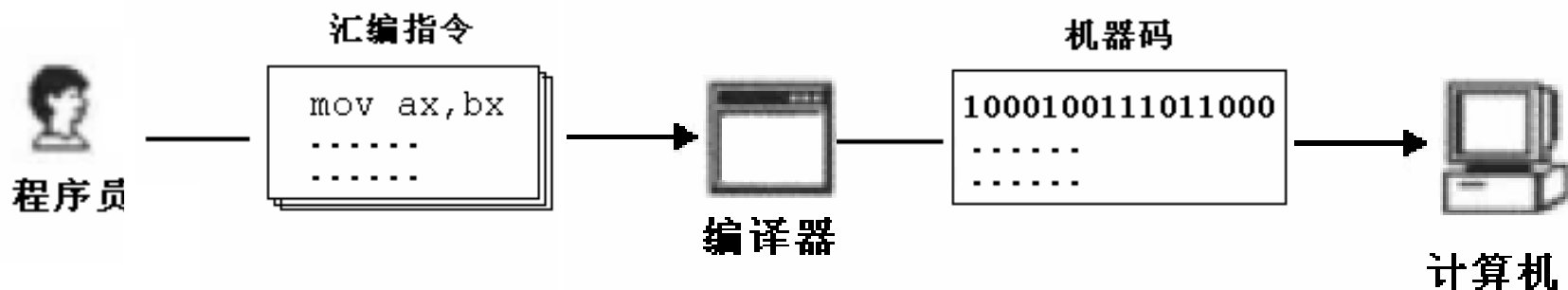
```
10110000000000000000000011  
0000010100000000000110000  
00101101000000000000000101
```

- 假如将程序错写成以下这样，请找错误:

```
10110000000000000000000011  
0000010100000000000110000  
00010110100000000000000101
```

1.2 汇编语言

- 汇编指令是机器指令的助记符。



1.2 汇编语言

- 汇编语言由以下**3**类组成：
 - 1、汇编指令（机器码的助记符）
 - 2、伪指令（由编译器执行）
 - 3、其它符号（由编译器识别）
- 汇编语言的核心是汇编指令，它决定了汇编语言的特性。

1.2 汇编语言

- 1、机器相关性
- 2、高速度和高效率
- 3、编写和调试的复杂性

1.3 存储器

- **CPU** 是计算机的核心部件，它控制整个计算机的运作并进行运算，想让**CPU** 工作，就必须向它提供指令和数据。
- 指令和数据在存储器中存放，本课程关注的是平时所说的内存，其作用仅次于**CPU**。
- 磁盘不同于内存，磁盘上的数据或程序如果不读到内存中，就无法被**CPU** 使用。
- 本课程一个重要的内容是**CPU**如何操作内存。

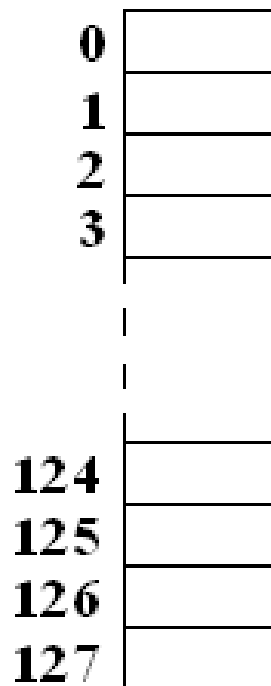
1.3 存储器

- 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；

- 例如：

一个存储器有128个存储单元，
编号从0~127。

如右图示：



1.3 存储器

- 对于大容量的存储器一般还用以下单位来计量容量（以下用**B**来代表Byte）：
 - $1\text{KB}=1024\text{B}$
 - $1\text{MB}=1024\text{KB}$
 - $1\text{GB}=1024\text{MB}$
 - $1\text{TB}=1024\text{GB}$
- 磁盘的容量单位同内存的一样，实际上以上单位是计算机中常用的计量单位。

1.3 存储器

- CPU要想进行数据的读写，必须和外部器件（标准的说法是芯片）进行三类信息的交互：
 - 存储单元的地址（地址信息）
 - 器件的选择，读或写命令（控制信息）
 - 读或写的数据（数据信息）

1.3 存储器

- 在计算机中专门有连接**CPU**和其他芯片的导线，**CPU**是通过导线把电信号传送到存储芯片，通常称为总线（**Bus**）。
 - 物理上：一根根导线的集合；
 - 逻辑上划分为：
 - 地址总线
 - 数据总线
 - 控制总线

1.3 存储器

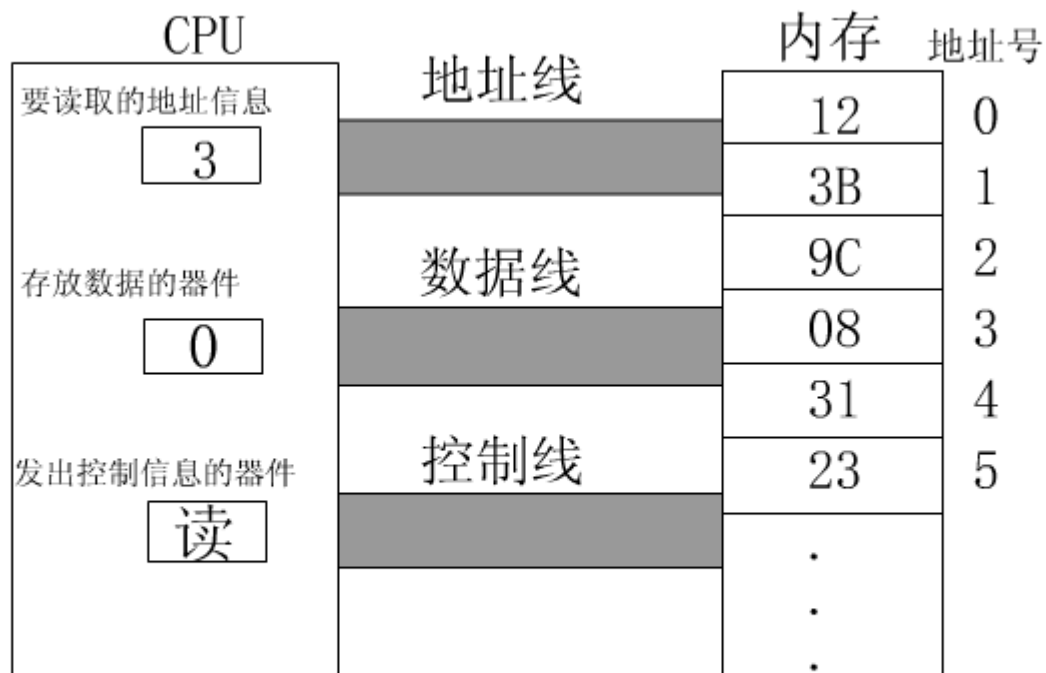
- 总线在逻辑上划分的图示：



1.3 存储器

- CPU在内存中读或写的数据演示：
 - 读演示
 - 写演示
- 从上面我们知道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？

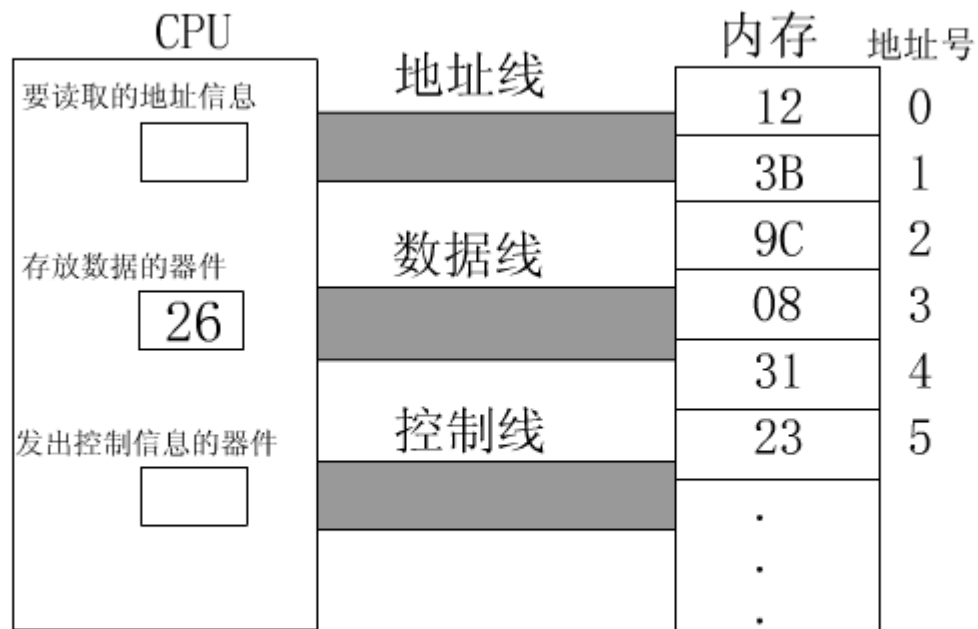
1.3 存储器



CPU从内存中3号单元处读取数据的过程

▶ play ◀ step ▶ step ◻ stop

1.3 存储器



CPU向内存中3号单元写入数据26的过程

▶ play ◀◀ step ▶▶ step ◻ stop

1.3 存储器

- 对于8086CPU，下面的机器码能够完成从3号单元读数据：
 - 机器码： 1010000000000001100000000
 - 含义： 从3号单元读取数据送入寄存器AX
 - CPU接收这条机器码后将完成上面所述的读写工作。

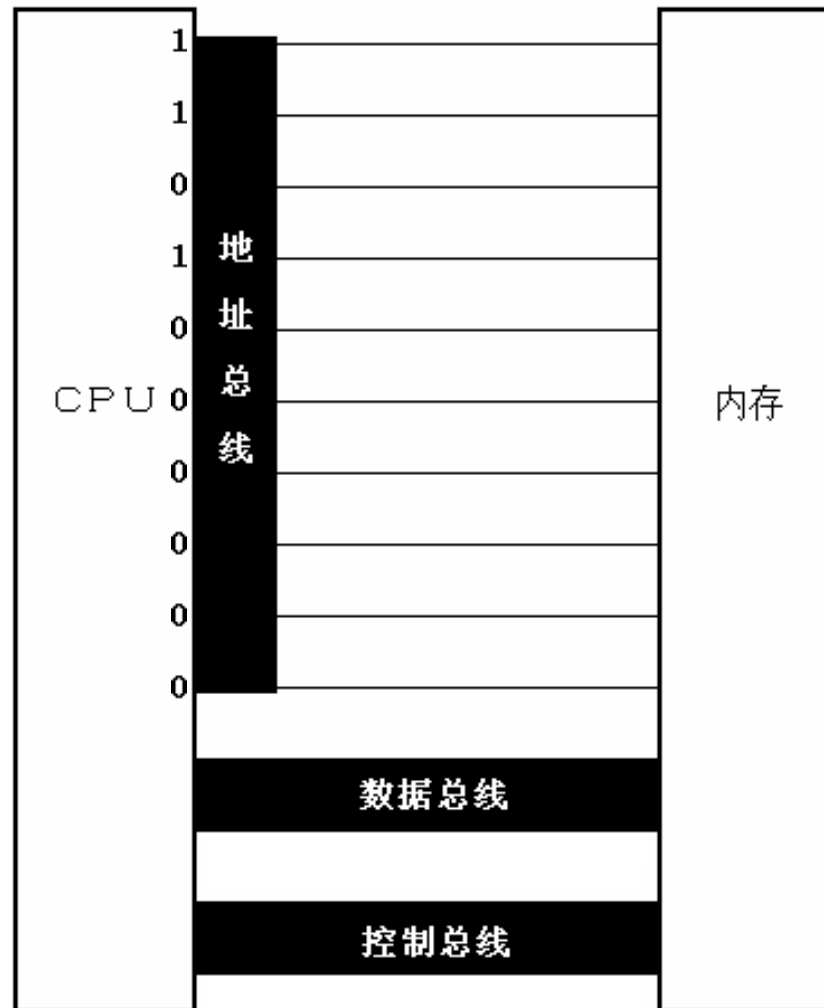
1.3 存储器

- 机器码难于记忆，用汇编指令来表示，情况如下：
 - 机器码：1010000000000001100000000
 - 对应的汇编指令：MOV AX,[3]
 - 含义：传送3号单元的内容到AX

1.4 地址总线

- CPU是通过地址总线来指定存储单元的。
- 地址总线上能传送多少个不同的信息，CPU就可以对多少个存储单元进行寻址。
- 地址总线发送地址信息演示

1.4 地址总线



1.4 地址总线

- 一个CPU有N根地址总线，则可以说这个CPU的地址总线的宽度为N。
- 这样的CPU最多可以寻找 2^N 个内存单元。

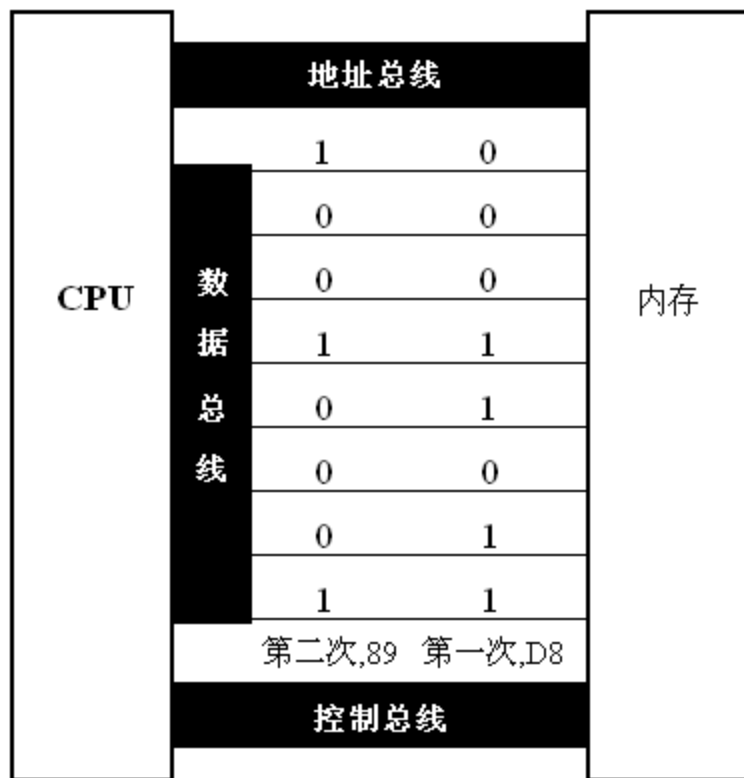
1.5 数据总线

- CPU与内存或其它器件之间的数据传送是通过数据总线来进行的。
- 数据总线的宽度决定了CPU和外界的数据传送速度。

1.5 数据总线

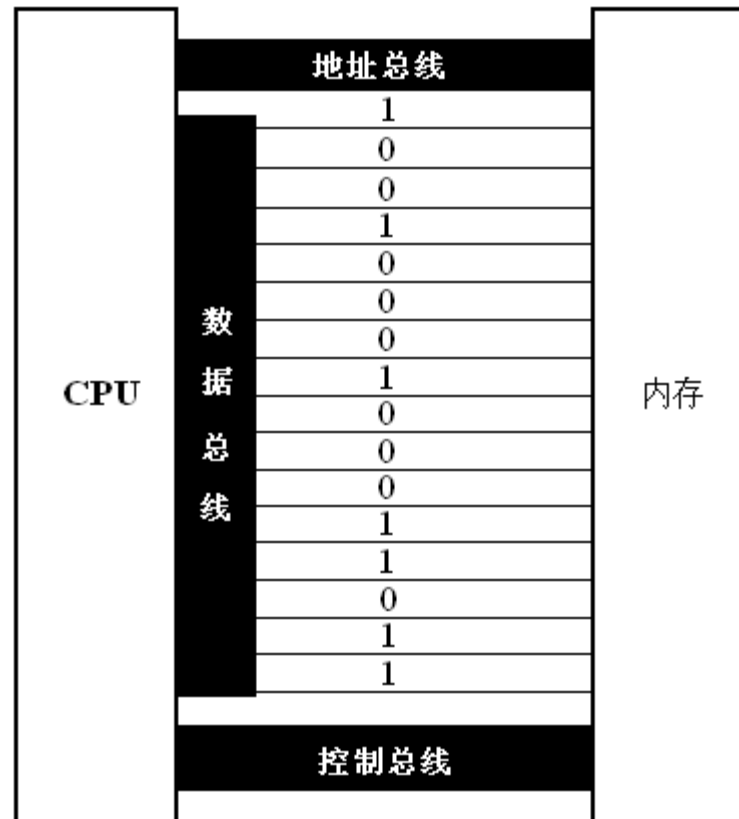
- 我们来分别看一下它们向内存中写入数据**89D8H**时，是如何通过数据总线传送数据的：
 - 8088CPU数据总线上的数据传送情况

1.5 数据总线



8位数据总线上传送的信息

1.5 数据总线



16位数据总线上传送的信息

1.6 控制总线

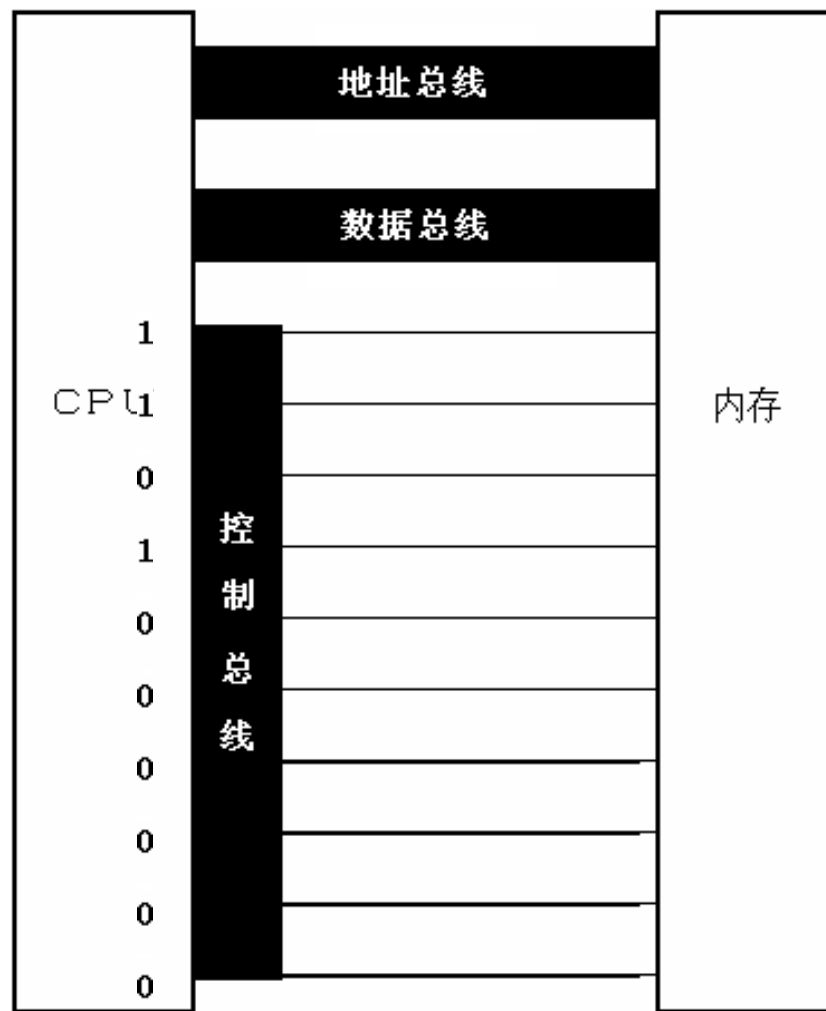
- **CPU**对外部器件的控制是通过控制总线来进行的。在这里控制总线是个总称，控制总线是一些不同控制线的集合。

- 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。

所以，控制总线的宽度决定了**CPU**对外部器件的控制能力。

- 控制总线上发送的控制信息

1.6 控制总线



1.7 其他器件——主板

- 在每一台**PC**机中，都有一个主板，主板上
有核心器件和一些主要器件。
- 这些器件通过总线（地址总线、数据总线、
控制总线）相连。

1.7 其他器件——接口卡

- 计算机系统中，所有可用程序控制其工作的设备，必须受到**CPU**的控制。
- **CPU**对外部设备不能直接控制，如显示器、音箱、打印机等。直接控制这些设备进行工作的是插在扩展插槽上的接口卡。

1.7 其他器件——各类存储器芯片

- 从读写属性上看分为两类：
随机存储器（**RAM**）和只读存储器（**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**

1.7 其他器件——各类存储器芯片

- 装有BIOS的ROM

BIOS: Basic Input/Output System, 基本输入输出系统。

BIOS是由主板和各类接口卡（如：显卡、网卡等）厂商提供的软件系统，可以通过它利用该硬件设备进行最基本的输入输出。在主板和某些接口卡上插有存储相应**BIOS的ROM**。

- PC机中各类存储器的逻辑连接情况

主板

RAM(主存储器)

ROM(装有系统BIOS)

CPU

RAM(主存储器)

内存条

扩展插槽

总线

RAM(显存)

ROM(装有显卡BIOS)

显卡

扩展插槽

显示器

ROM(装有网卡BIOS)

网卡

扩展插槽

其他器件

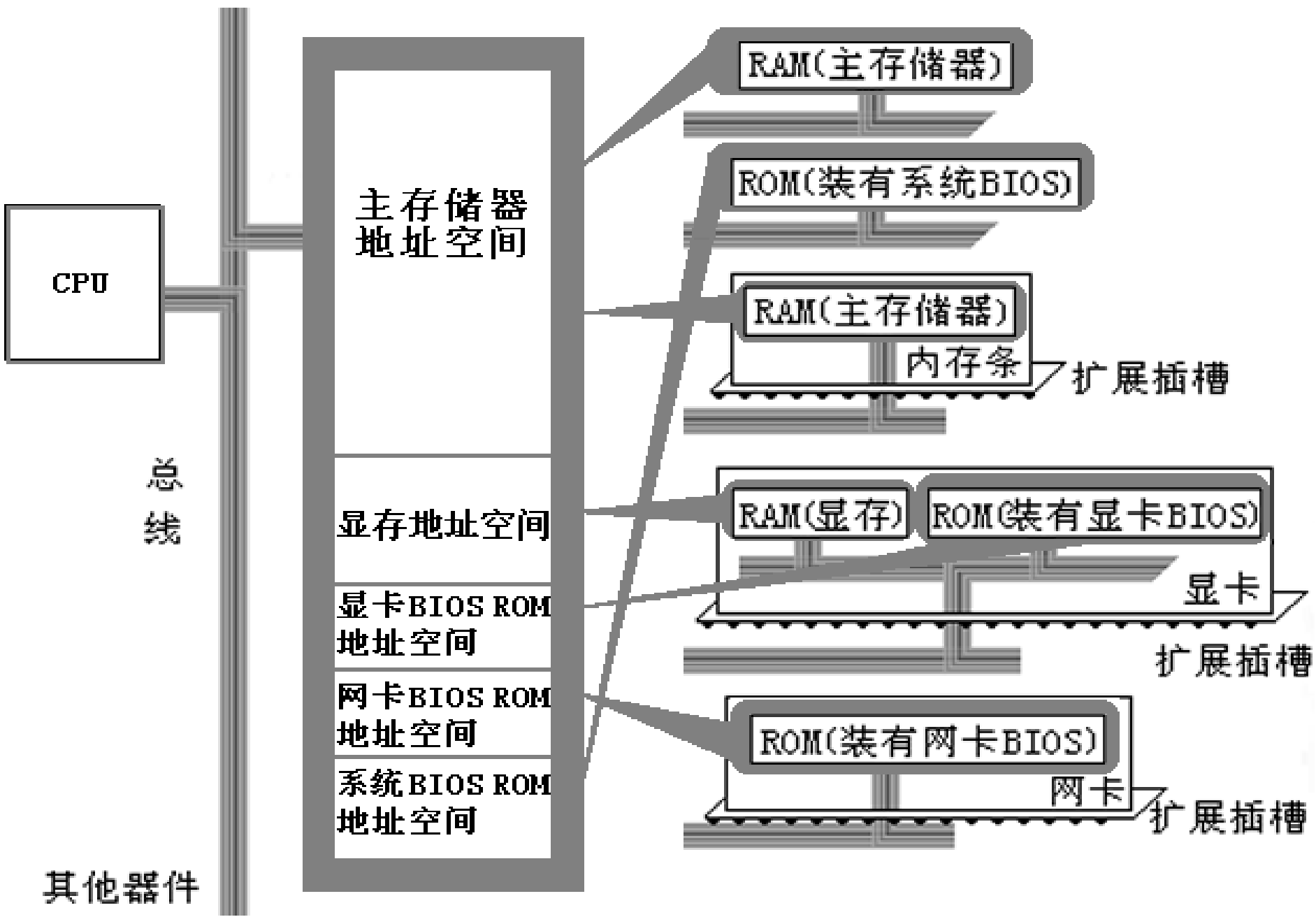


1.8 内存地址空间

- 上述的那些存储器在物理上是独立的器件。
- 但是它们在以下两点上相同：
 - 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。

1.8 内存地址空间

- 将各类存储器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。



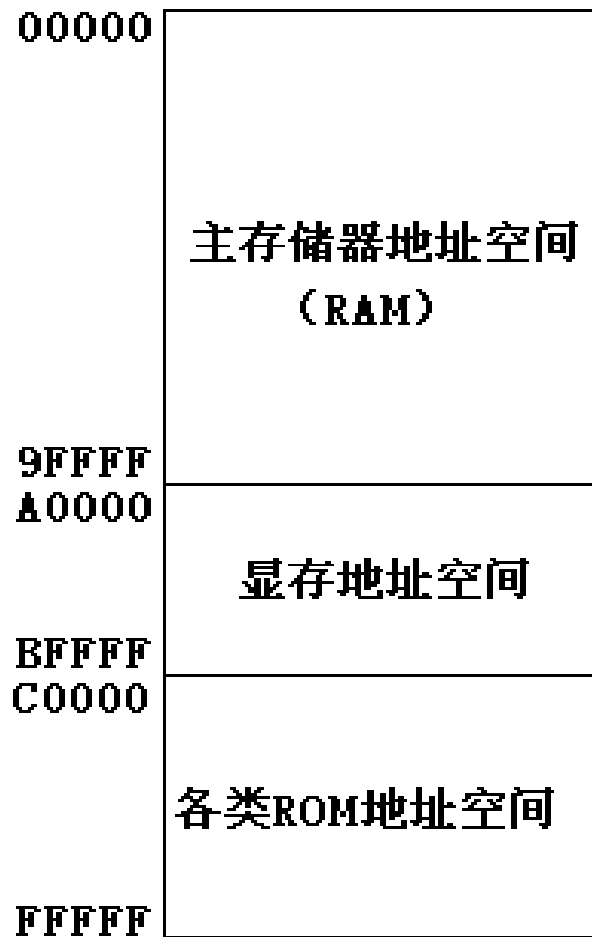
1.8 内存地址空间

- 假设，上图中的内存空间地址段分配如下：
 - 地址0~7FFFFH的32KB空间为主随机存储器的地址空间；
 - 地址8000H~9FFFFH的8KB空间为显存地址空间；
 - 地址A000H~FFFFH的24KB空间为各个ROM的地址空间。

1.8 内存地址空间

- 不同的计算机系统的内存地址空间分配情况是不同的。
- 8086PC机内存地址空间分配的基本情况

8086PC机的内存地址空间分配



1.8 内存地址空间

■ 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。

第2章 寄存器

- 2.1 通用寄存器
- 2.2 字在寄存器中的存储
- 2.3 几条汇编指令
- 2.4 物理地址
- 2.5 8086CPU给出物理地址的方法
- 2.6 段的概念

CPU概述

- 一个典型的**CPU**由运算器、控制器、寄存器等器件组成，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。
- 外部总线实现**CPU**和主板上其它器件的联系。

寄存器概述

- 8086CPU有14个寄存器 它们的名称为：
AX、BX、CX、DX、SI、DI、SP、BP、
IP、CS、SS、DS、ES、FLAGS（又称PSW，
program status word）。

寄存器概述-80x86程序可见寄存器

80x86的程序可见
寄存器组:

通用寄存器

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

数据寄存器

指针寄存器

变址寄存器

专用寄存器

| | | |
|--------|--|-------|
| ESP | | SP |
| EIP | | IP |
| EFLAGS | | FLAGS |

控制寄存器

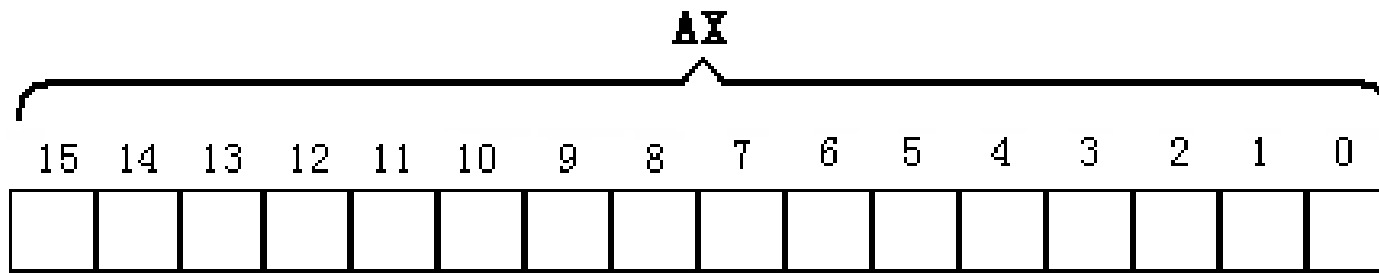
段寄存器

| |
|----|
| CS |
| DS |
| SS |
| ES |
| FS |
| GS |

2.1 通用寄存器

- 8086CPU所有的寄存器都是16位的，可以存放两个字节。
- AX、BX、CX、DX 通常用来存放一般性数据被称为通用寄存器。
- 下面以AX为例，我们看一下寄存器的逻辑结构。

2.1 通用寄存器

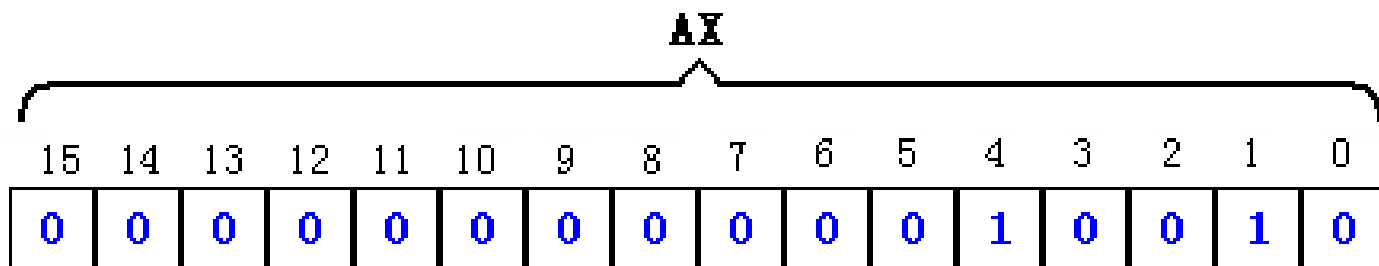


16位寄存器的逻辑结构

- 一个16位寄存器可以存储一个16位的数据。（数据的存放情况）

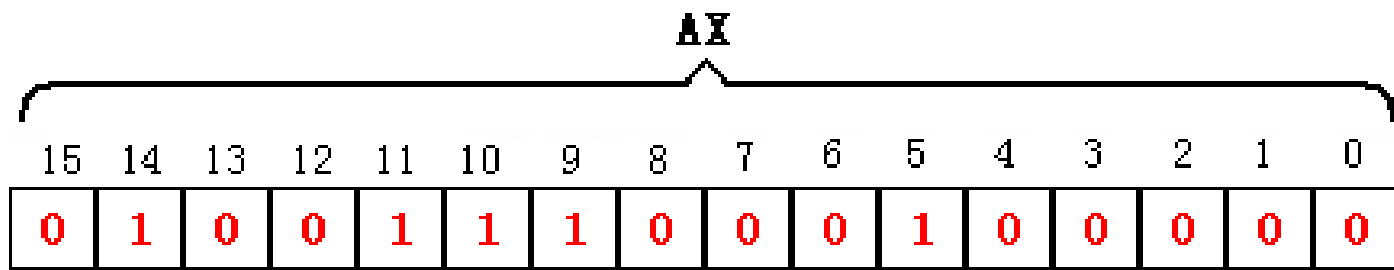
2.1 通用寄存器

- 数据：18
- 二进制表示：10010
- 在寄存器AX中的存储：



2.1 通用寄存器

- 数据：20000
- 二进制表示：0100111000100000
- 在寄存器AX中的存储：



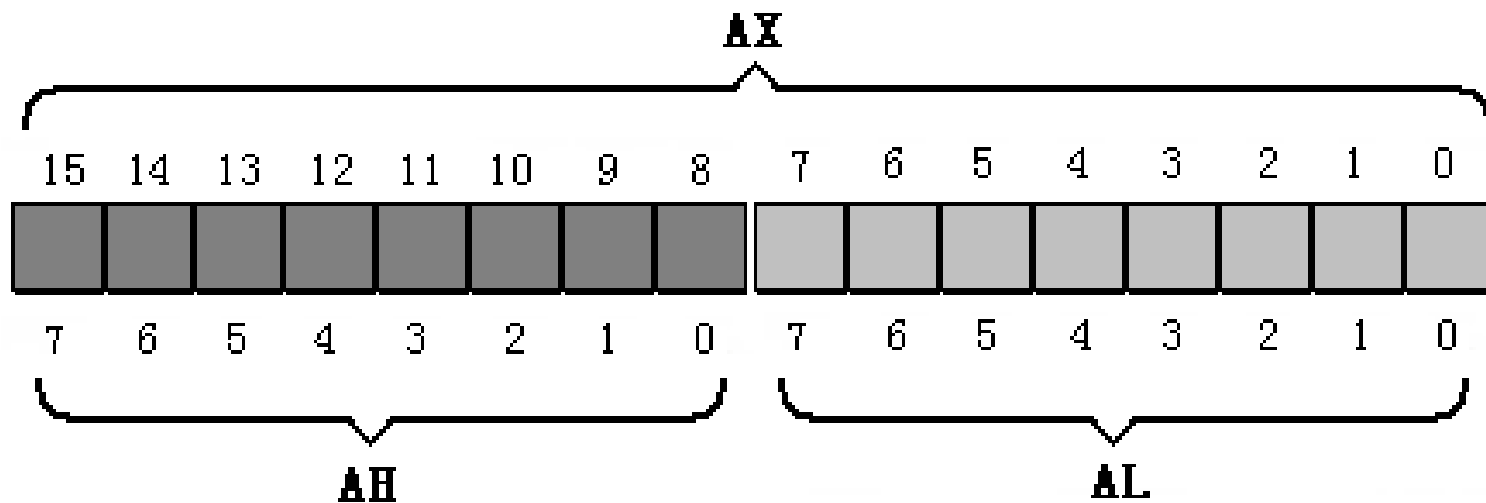
- 一个16位寄存器所能存储的数据的最大值为多少？
答案： $2^{16}-1$

2.1 通用寄存器

- 8086上一代CPU中的寄存器都是8位的；
- 为保证兼容性，这四个寄存器都可以分为两个独立的8位寄存器使用。
 - AX可以分为AH和AL；
 - BX可以分为BH和BL；
 - CX可以分为CH和CL；
 - DX可以分为DH和DL。
- 8086CPU的8位寄存器存储逻辑

2.1 通用寄存器

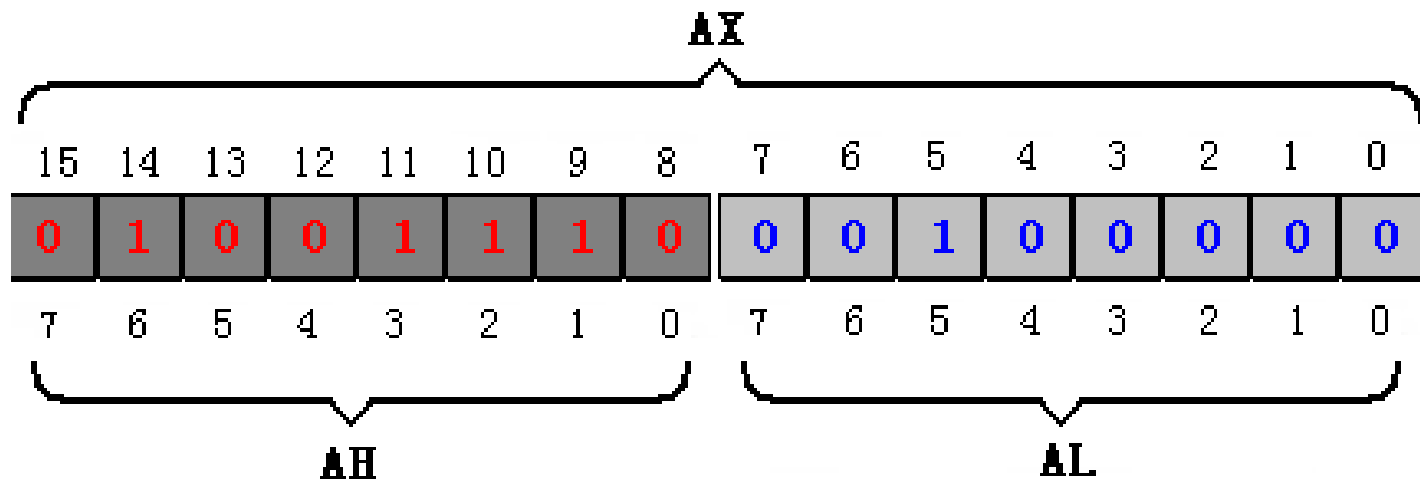
- 以AX为例，8086CPU的16位寄存器分为两个8位寄存器的情况：



2.1 通用寄存器

- AX的低8位（0位~7位）构成了AL寄存器，高8位（8位~15位）构成了AH寄存器。
- AH和AL寄存器是可以独立使用的8位寄存器。
- 8086CPU的8位寄存器数据存储情况

2.1 通用寄存器



| 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) |

2.1 通用寄存器


80x86 32位通用寄存器

| 32位 | 16位 | 8位（高） | 8位（低） |
|-----|-----|-------|-------|
| EAX | AX | AH | AL |
| EBX | BX | BH | BL |
| ECX | CX | CH | CL |
| EDX | DX | DH | DL |

2.2 字在寄存器中的存储

- 一个字可以存在一个**16**位寄存器中，这个字的高位字节和低位字节自然就存在这个寄存器的高**8**位寄存器和低**8**位寄存器中。

字: 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0



高位字节 低位字节

- 一个**8**位寄存器所能存储的数据的最大值是多少？
答案： 2^8-1 。

2.3 几条汇编指令

| 汇编指令 | 控制 CPU 完成的操作 | 用高级语言的语法描述 |
|-----------|----------------------------|------------|
| mov ax,18 | 将 18 送入寄存器 AX | AX=18 |
| mov ah,78 | 将 78 送入寄存器 AH | AH=78 |
| add ax,8 | 将寄存器 AX 中的数值加上 8 | AX=AX+8 |
| mov ax,bx | 将寄存器 BX 中的数据送入寄存器 AX | AX=BX |
| add ax,bx | 将 AX 和 BX 中的数值相加，结果存在 AX 中 | AX=AX+BX |

汇编指令不区分大小写!

2.3 几条汇编指令

这里遇到进位丢失问题，但是 CPU 不是真的不丢弃这个进位值，这个问题会在后面的课程中讨论。

- CPU 执行下表中的程序段的每条指令后，对寄存器中的数据进行改变。

程序段中指令执行情况之一（原AX中的值：0000H，原BX中的值：0000H）

| 程序段中的指令 | 指令执行后AX中的数据 | 指令执行后BX中的数据 |
|----------------------------|-------------|-------------|
| <code>mov ax, 4E20H</code> | 4E20H | 0000H |
| <code>add ax, 1406H</code> | 6226H | 0000H |
| <code>mov bx, 2000H</code> | 6226H | 2000H |
| <code>add ax, bx</code> | 8226H | 2000H |
| <code>mov bx, ax</code> | 8226H | 8226H |
| <code>add ax, bx</code> | ? | 8226H |

2.3 几条汇编指令

程序段中指令执行情况之二（原AX中的值：0000H，原BX中的值：0000H）

| 程序段中的指令 | 指令执行后AX中的数据 | 指令执行后BX中的数据 |
|----------------------------|-------------|-------------|
| <code>mov ax, 001AH</code> | 001AH | 0000H |
| <code>mov bx, 0026H</code> | 001AH | 0026H |
| <code>add al, bl</code> | 0040H | 0026H |
| <code>add ah, bl</code> | 2640H | 0026H |
| <code>add bh, al</code> | 2640H | 4026H |
| <code>mov ah, 0</code> | 0040H | 4026H |
| <code>add al, 85H</code> | 00C5H | 4026H |
| <code>add al, 93H</code> | ? | 4026H |

2.4 物理地址

- 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。

2.5 8086CPU给出物理地址的方法

- 8086CPU是16位结构：
 - 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。

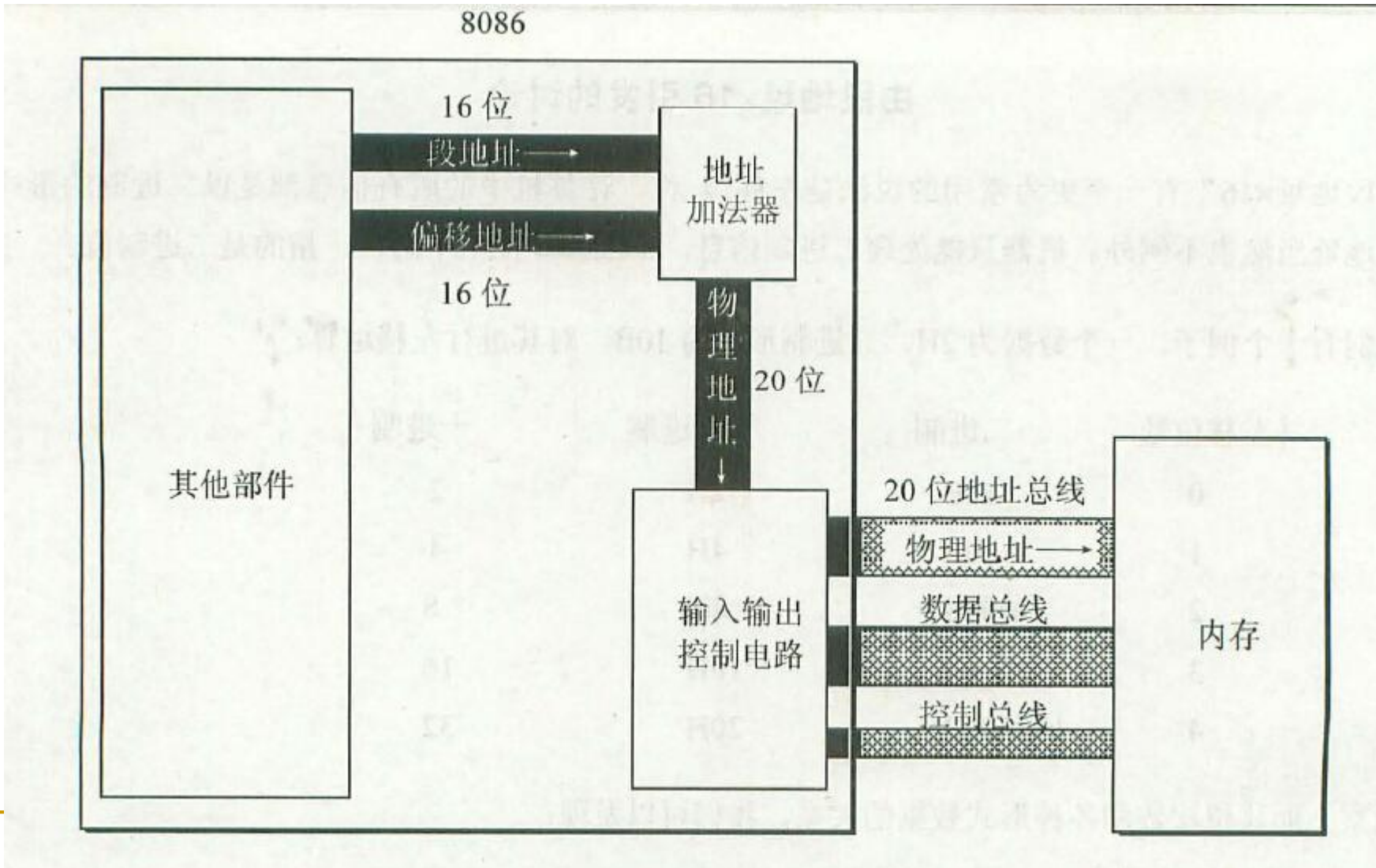
2.5 8086CPU给出物理地址的方法

- 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。

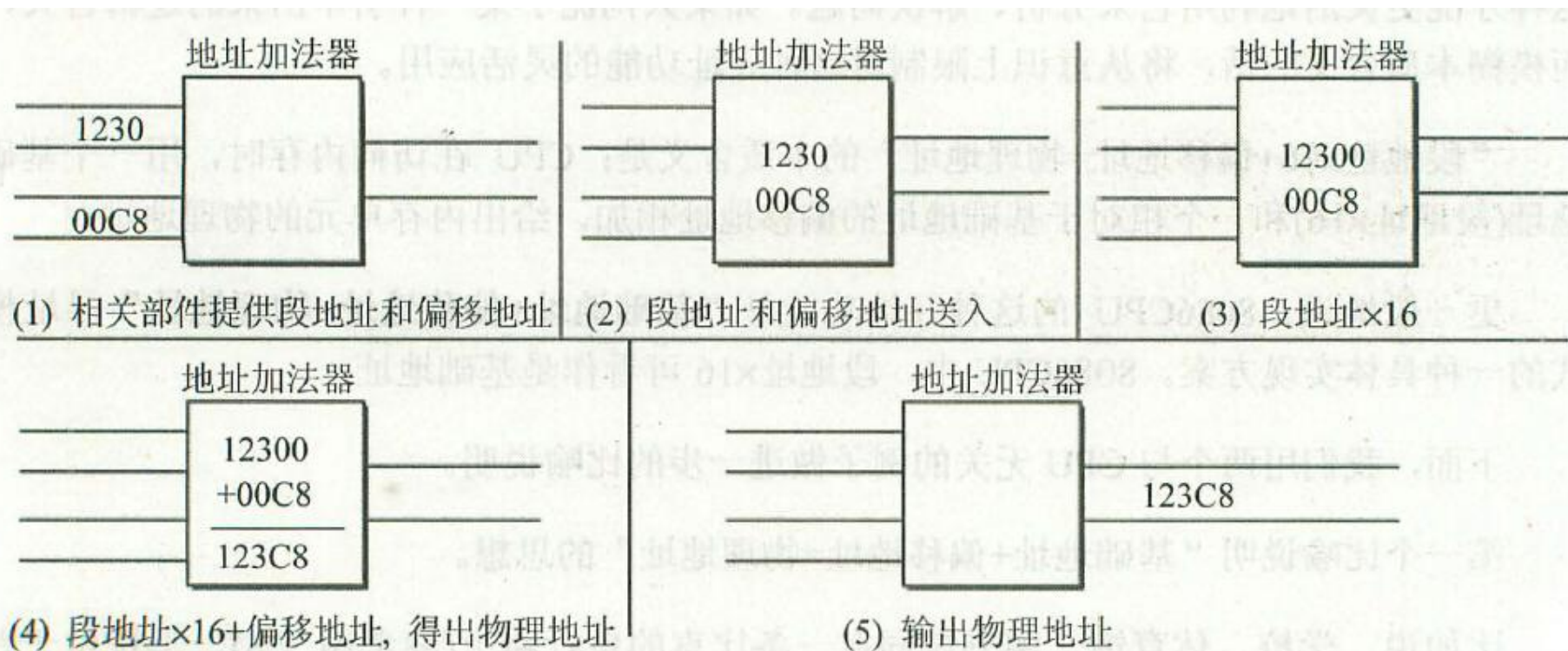
2.5 8086CPU给出物理地址的方法

- **实模式存储器寻址：**8086CPU采用一种在内部用两个16位地址合成的方法来形成一个20位的物理地址。
- 80x86中8086/8088只能在实模式下工作，其他微处理器可在实模式或保护模式下工作。
- 8086CPU相关部件的逻辑结构

在8086CPU内部用两个16位地址合成的方法来形成一个20位的物理地址



地址加法器



由段地址 $\times 16$ 引发的讨论

由段地址 $\times 16$ 引发的讨论

| 移位位数 | 二进制 | 十六进制 | 十进制 |
|------|---------|------|-----|
| 0 | 10B | 2H | 2 |
| 1 | 100B | 4H | 4 |
| 2 | 1000B | 8H | 8 |
| 3 | 10000B | 10H | 16 |
| 4 | 100000B | 20H | 32 |

- 观察移位次数和各种形式数据的关系：
 - (1) 一个数据的二进制形式左移1位，相当于该数据乘以2；
 - (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；
 - (3) 地址加法器如何完成段地址 $\times 16$ 的运算？
以二进制形式存放的段地址左移4位。

2.6 段的概念

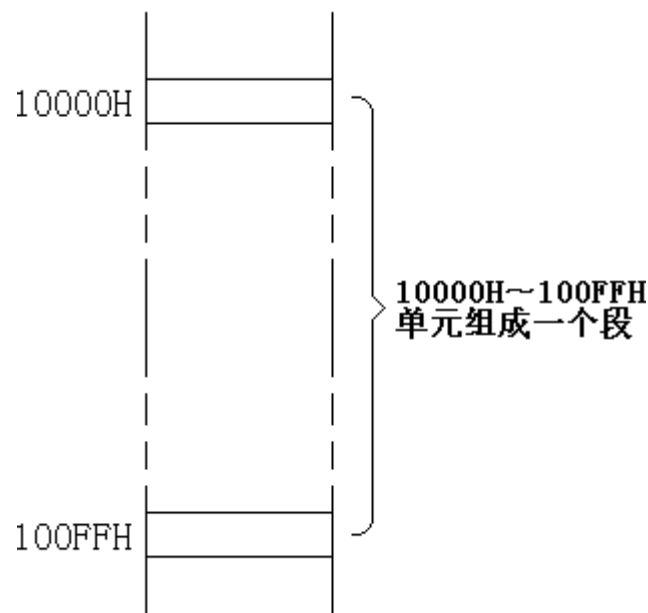
- 错误认识：

- 内存被划分成了一个一个的段，每一个段有一个段地址。

- 其实：

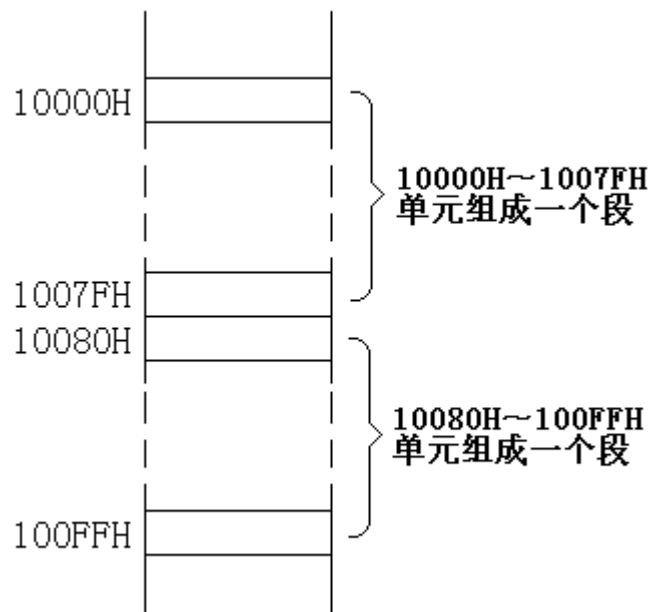
- 内存并没有分段，段的划分来自于CPU，由于8086CPU用“ $(\text{段地址} \times 16) + \text{偏移地址} = \text{物理地址}$ ”的方式给出内存单元的物理地址，使得我们可以用分段的方式来管理内存。

2.6 段的概念



- 我们可以认为：地址10000H~100FFH的内存单元组成一个段，该段的起始地址（基础地址）为10000H，段地址为1000H，大小为100H。

2.6 段的概念



- 我们也可以认为地址`10000H~1007FH`、`10080H~100FFH` 的内存单元组成两个段，它们的起始地址（基础地址）为`10000H`和`10080H`，段地址为：`1000H` 和`1008H`，大小都为`80H`。

2.6 段的概念

- 以后，在编程时可以根据需要，将若干地址连续的内存单元看作一个段，用段地址 $\times 16$ 定位段的起始地址（基础地址），用偏移地址定位段中的内存单元。
- 内存单元地址小结

内存单元地址小结

- CPU访问内存单元时，必须向内存提供内存单元的物理地址。
- 8086CPU在内部用段地址和偏移地址移位相加的方法形成最终的物理地址。
- 思考两个问题

内存单元地址小结

- (1) 观察下面的地址，有什么发现？

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

- 结论：CPU可以用不同的段地址和偏移地址形成同一个物理地址。

内存单元地址小结

- (2) 如果给定一个段地址，仅通过变化偏移地址来进行寻址，最多可以定位多少内存单元？
 - 结论：偏移地址**16**位，变化范围为**0~FFFFH**，仅用偏移地址来寻址最多可寻**64K**个内存单元。
 - 比如：给定段地址**1000H**，用偏移地址寻址，CPU的寻址范围为：**10000H~1FFFFH**。

内存单元地址小结

- 在8086PC机中，存储单元的地址用两个元素来描述。即段地址和偏移地址。
- “数据在21F60H内存单元中。”对于8086PC机的两种描述：
 - （a）数据存在内存2000:1F60单元中；
 - （b）数据存在内存的2000段中的1F60H单元中。
- 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。

小结