汇编语言介绍

汇编语言是最接近机器语言的底层语言，常应用于嵌入式、反编译等领域，学习它能让我们更好的理解计算机是怎么去执行我们的命令的。汇编语言是直接在硬件之上工作的编程语言，首先要了解硬件系统的结构，才能有效的应用汇编语言对其编程。

学习汇编语言的目的：学会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，换句话说，并不是我们以后就要使用汇编语言编写程序，而是使用高级语言的时候，以汇编语言的角度去更有效的提升代码质量

机器语言

概念：是机器指令的集合，展开来讲就是一台机器可以正确执行的命令（0、1集合）。 如：指令：
01010000（push Ax）表示将Ax推进堆栈（电平脉冲）

早期的程序员将0、1数字编程的程序代码打在纸带上，打孔为1、不带孔为0；再将程序通过纸带机输入计算机进行运算，后来有了集体管、石英、以及硅材料。

汇编语言

汇编语言的主体是汇编指令； 汇编指令和机器指令的差别在于指令的表达方法上，汇编指令是机器指令便于记忆的书写格式，汇编指令是机器指令的助记符

```
机器指令：1000100111011000
操作：  寄存器BX的内容送到AX中
汇编指令：  Mov AX，Bx    # move的缩写Mov，但是注意是Bx移动到Ax，后往前
# 使用汇编语言这样的写法更接近人类语言，便于阅读和记忆
```

基础知识

寄存器

简单来讲是嵌入cpu中的存储器，使cpu可以存储数据的器件，一个cpu中有多个寄存器，AX是其中的一个寄存器的代号，BX是另一个寄存器的代号。注意：寄存器是更接近于cpu，区别于高级缓存和二级缓存。

如何让汇编语言转机器语言?

程序(C,C++) --> 汇编指令 --> 编译器 --> 机器码 --> 计算机

汇编语言组成

汇编语言由以下三类组成：1、汇编指令（机器码的助记符，对应机器码的动作）2、伪指令（由编译器执行，没有对应的机器码但是编译器认识，可以翻译成机器码，或者组合指令）3、其他符号（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。

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，要想让一个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，也就是平时说的内存。（注意计算机很多部件都有存储器都有存储器，如显卡里面的存储器，我们叫它显存，通过GPU将显存的数据读取出来，这里的GPU比cpu更快）；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

一台Pc机中内存的作用仅次于cpu，离开了内存，性能再好的cpu也无法工作。 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。

指令和数据

指令和数据是应用上的概念，在内存或磁盘上，指令和数据没有任何区别，都是二进制信息。 二进制信息：

- 1000100111011000 --> 89D8H(数据)：解读，4个字节表示一个1000:8;1101:D;1001:9;1000:8;H表示Hex十六进制，B表示二进制
- 1000100111011000 --> Mov AX,BX(指令程序)，和上面的二进制一样，但是可以表示指令
- 同一个指令到底表示什么，我们说了算，可以是数据/指令程序。

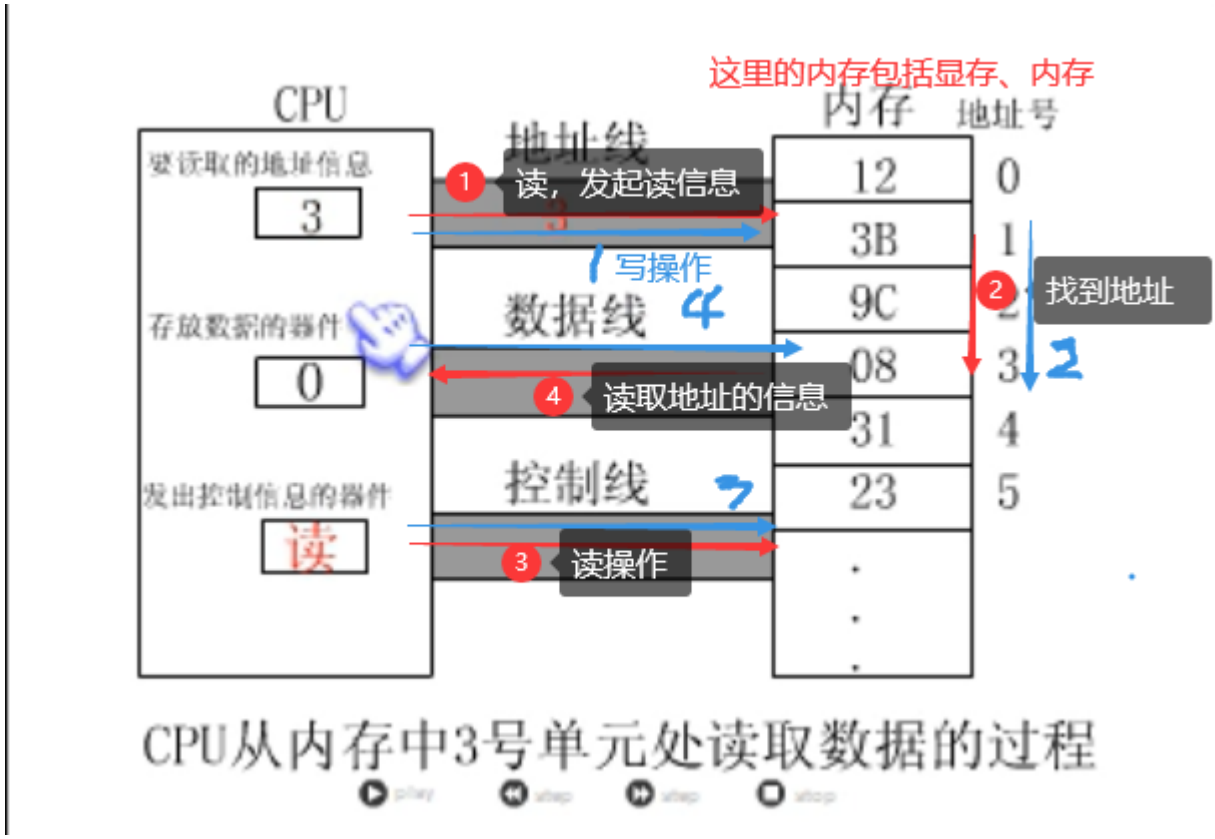
CPU对存储器的读写

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）

那么cpu是通过什么将地址、数据和控制信息传到存储芯片中的哪？电子计算机能处理、传输信息都是电信号，电信号当然要用导线传送。在计算机中专门有连接cpu和其他芯片的导线，通常称为总线（bus）。

- 物理上：一根根导线的集合；
- 逻辑上划分为：（cpu先通过地址总线找到目标地址，控制总线下达命令动作，数据总线操作数据，这就解释了同一指令不同形式，因为看这个二进制在那条总线上传递）
    - 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信息，cpu就可以对多少个存储单元进行寻址，也就是寻址能力，如现在的64位、32位cpu就是它的寻址能力
    - 数据总线：cpu与内存或其他器件之间的数据传递
    - 控制总线：cpu对外部器件的控制是通过控制总线来进行的



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的N次方个内存单元。

数据总线

数据总线的宽度决定了cpu和外界的数据传送速度，换句话说如果是16位的数据，8位数据总线宽度要传递两次。

控制总线

控制总线是一些不同控制线的集合，有多少条控制总线，就意味着cpu提供了对外部件的多少种控制，也就是说控制总线的宽度决定了cpu对外部件的控制能力。如有一根名为读信号输出控制线负责由cpu向外传送读信号，cpu向该控制线上输出低电平表示读取数据；有一根名为写信号输出控制总线负责由cpu向外传送写信号。

小结

- 汇编指令是机器指令的助记符，同时机器指令一一对应
- 每一种cpu都有自己的汇编指令集
- cpu可以直接使用的信息在存储器中存放
- 在存储器中指令和数据没有任何区别，都是二进制信息
- 存储单元从零开始顺序编号
- 一个存储单元可以存储8个bit，也就是8个二进制
- 每个cpu芯片都是有很多的管脚，这些管脚和总线相连，也可以说，这些管脚引出总线，一个cpu可以引出三种总线的宽度标志这这个cpu的不同方面的性能

内存地址空间基础

一个cpu的地址线宽度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。

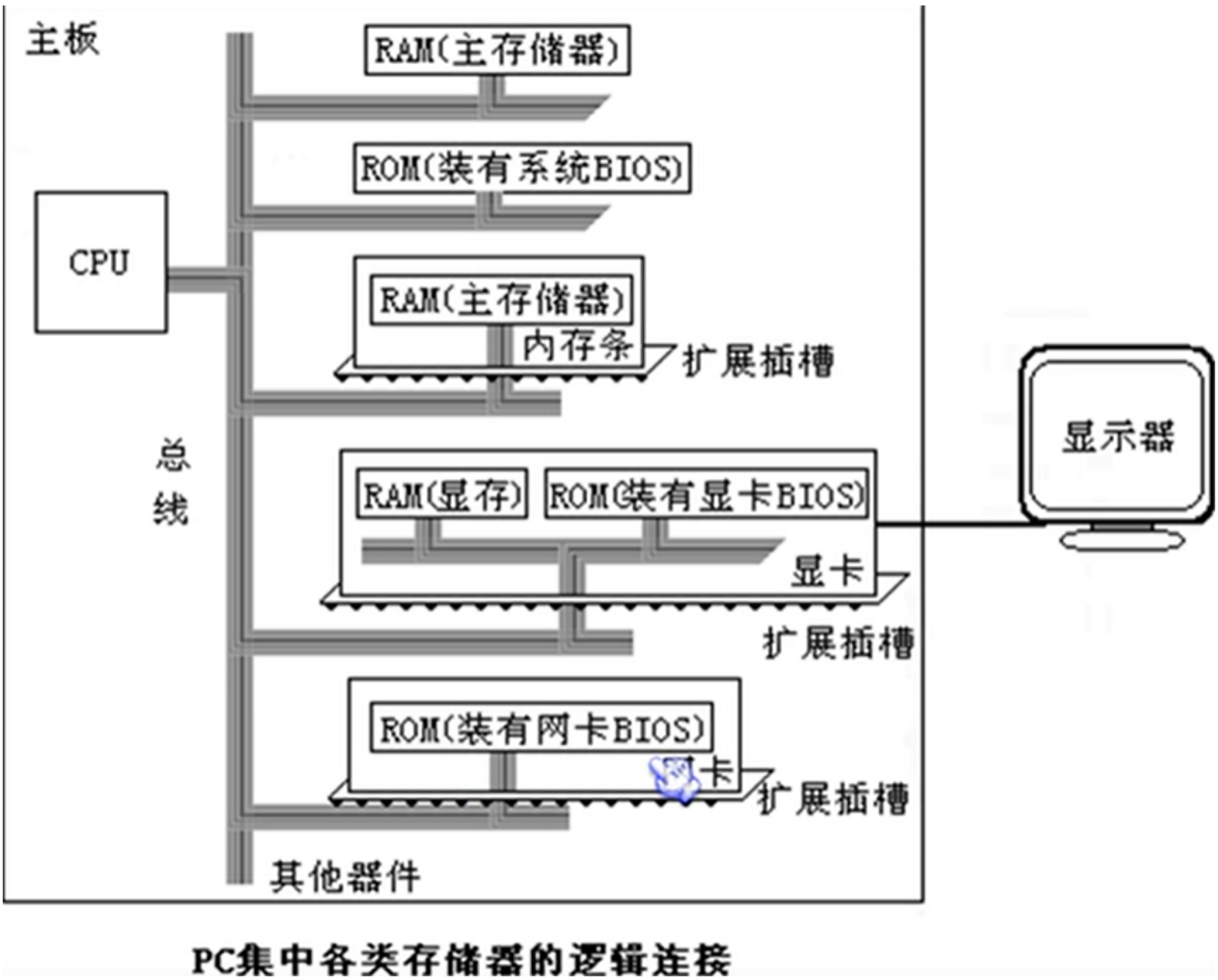
主板：每个pc机中都有主板，主板上有核心器件和一些主要器件，这些核心器件通过总线（地址、数据、控制总线）相连。

接口： 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控制

cpu对外部设备不能直接控制，如显示器、音响、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

各类存储芯片：

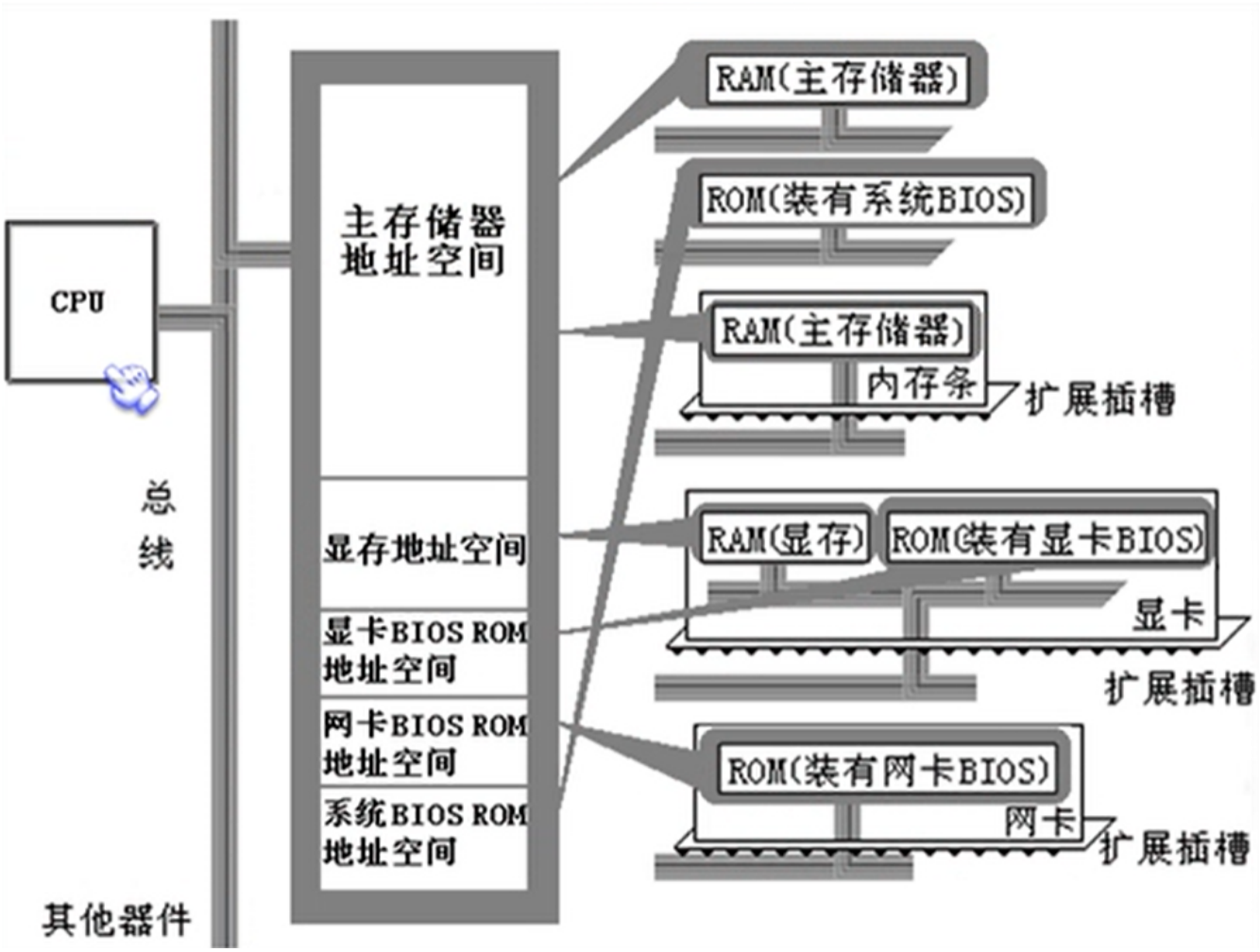
- 从读写属性上分为两类：随机存储器（RAM) 和只读存储器（ROM）
- 从功能和连接上分类：
  - 随机存储器: RAM，如：内存，断电即丢失
  - 装有BIOS的ROM，如：Basic Input/Output System，基本输入输出系统，它是由主板和各类接口卡（如显卡、网卡等）厂家提供的软件系统，可以通过它利用该硬件设备进行最基本的输入输出。
  - 接口卡上的RAM，如：显存



PC集中各类存储器的逻辑连接

上述的那些存储器在物理上是独立的器件，但是在以下两点上是相同的：1、都是和cpu的总线相连；2、cpu对他们进行读写的时候是通过控制线发出内存读写命令。

从cpu的角度看内存：所有的物理存储器看作一个有若干存储单元组成的**逻辑存储器**（一视同仁）；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在这个段地址空间中读写数据，实际上就是在对相应的物理存储器中进行读写数据



假设，上图中的内存空间地址段分配如下：（不同的cpu内存地址空间分配不同）

- 地址0~7FFFH的32KB空间为主随机存储器地址空间（内存）；
- 地址8000H~9FFFH的8KB空间为显存地址空间；
- 地址A000H~FFFFH的24KB地址空间为哥哥ROM的地址空间；

内存地址空间进阶

最终运行程序的是cpu，我们用汇编编程的时候，必循要从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，它的容量受限于cpu寻址能力的限制，这个逻辑存储器即时我们所说的内存地址空间。

CPU工作原理

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。区别：

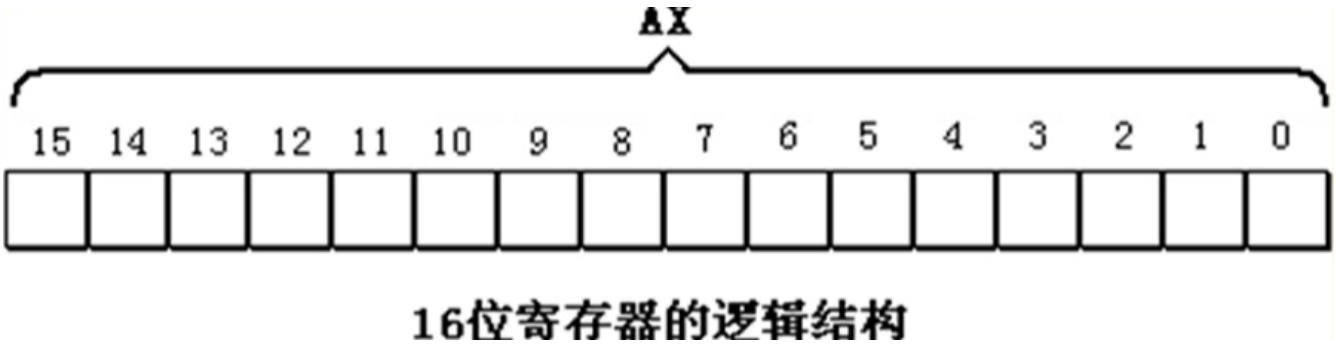
- 内部总线实现cpu内部各个器件之间的联系
- 外部总线实现cpu和主板上其他器件的联系

寄存器

8086cpu有14个寄存器他们的名称为：AX、BX、CX、DX、SI、DI、SP、BP、IP、CS、SS、DS、ES、PSW。其中8个是通用寄存器，现在的酷睿也是。所有寄存器都是16位的（也就是说最大能存储的数字是2^16-1），可以存储两个字节。

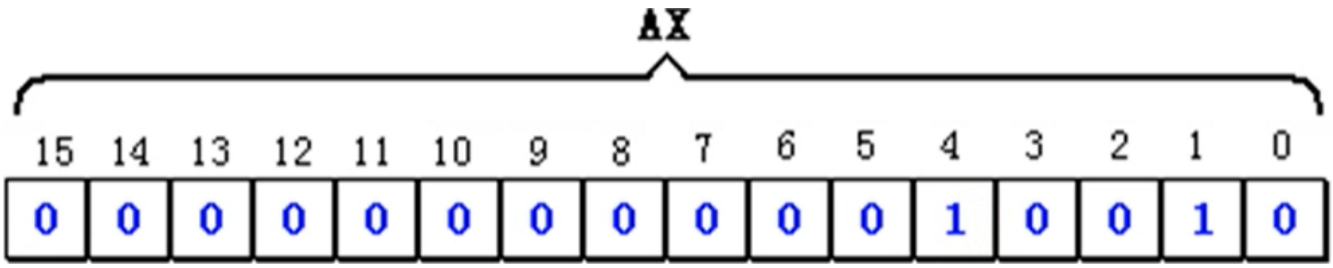
通用寄存器

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，看看它的逻辑结构：



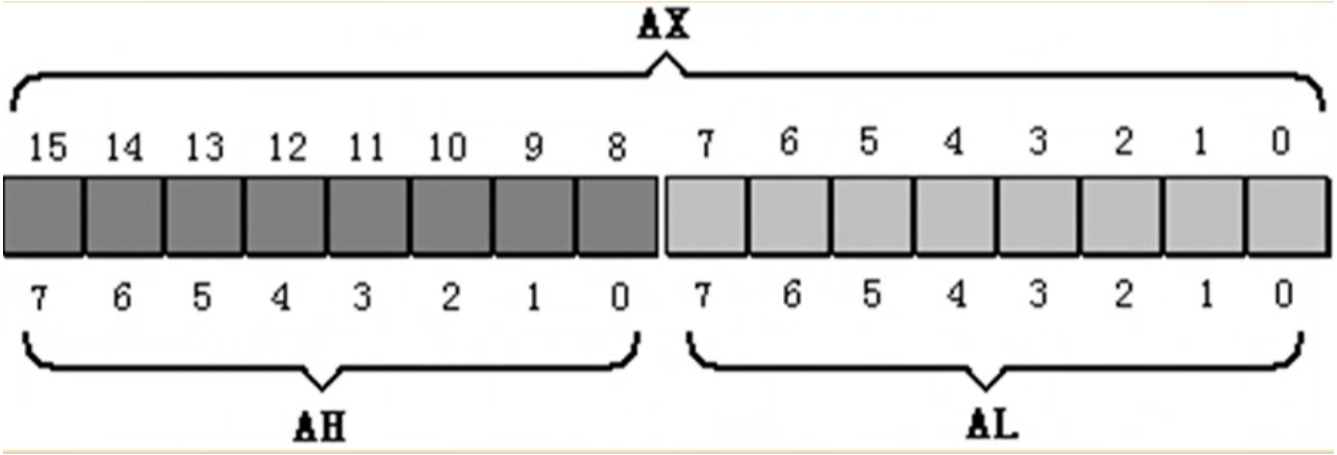
16位寄存器的逻辑结构

数据：18
二进制表示：10010
在寄存器AX中存储：



在8086上一代cpu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。（兼容）

- 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；



十六进制的出现：通常16位的二进制可读性很差，所以可以将其分为4份，一份最大的表示15用F表示；如：4（0100）、E（1110）

几条汇编指令

注意：汇编语言不区分大小写！！！

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8送入AX	AX = 18
mov ah,78	将78送入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,BX 中的内容相加，结果存在AX中	AX = AX + BX

汇编指令举例

例1：下面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。答：1044C

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
mov ax,4E20H	4E20H	0000H
add ax,1406H	6226H	0000H
mov bx,2000H	6226H	2000H
add ax,bx	8226H	2000H
mov bx,ax	8226H	8226H
add ax,bx	?	8226H

例2：下面CPU执行下表的程序后的结果？注意这里出现了AL,低位寄存器，按理说应该是0158H，但是由于是AL，所以我们只能看到后两位，多的会抛给AH（不是直接抛过去，而是通过中间寄存器），也就是说正确答案是0058H

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
mov ax,001AH	001AH	0000H
mov bx,0026H	001AH	0026H
add al,bl	0040H	0026H
add ah,bl	2640H	0026H
add bh,al	2640H	4026H
mov ah,0	0040H	4026H
add al,85H	00C5H	4026H
add al,93H	?	4026H

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。

我们将这个唯一的地址称为物理地址。

16位结构的cpu（可推出64位cpu）

概括的讲，16位结构描述了一个cpu具有以下的几个特征：

- 运算器一次最多可以处理16位数据；

- 寄存器的最大宽度位16位；
- 寄存器和运算器之间的通路是16位；

8086有20位地址总线，可传递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。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！！！（地址加法器）

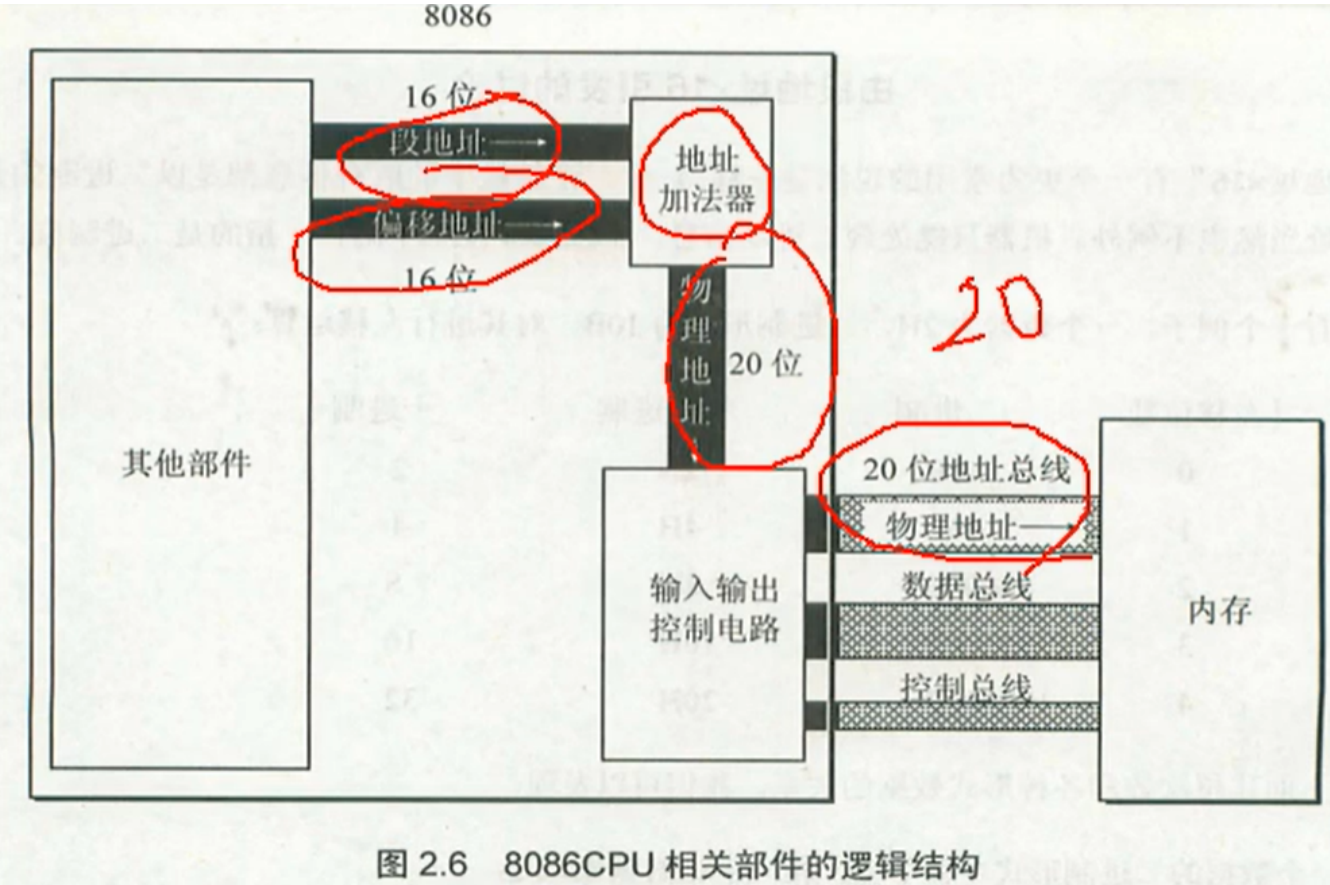


图 2.6　8086CPU 相关部件的逻辑结构

根据上图，我们可以知道：

- cpu中的相关部件提供两个16位的地址，一个称为段地址，另一个称为偏移地址；
- 段地址和偏移地址通过内部总线送入一个称为地址加法器的部件中；
- 地址加法器将两个16位地址合并成一个20位的地址
- 地址加法器合成物理地址的方法： **物理地址 = 段地址*16 + 偏移地址**

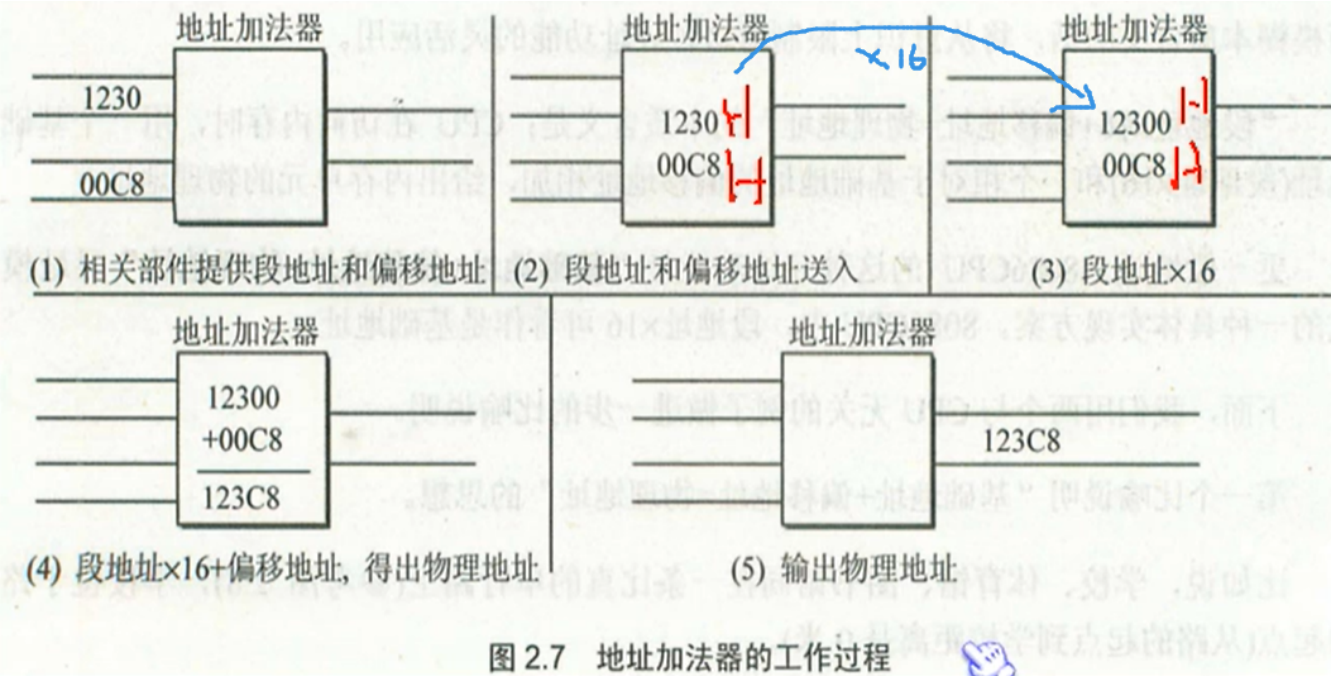


图 2.7　地址加法器的工作过程

经过分析：

1、一个数据的十六进制形式左移一位，相当于乘以16；
2、一个数据的十进制形式左移一位，相当于乘以10；
3、一个数据的x进制形式左移一位，相当于乘以x；

形象比喻：假如我们想告诉别人一个四位数：2826，但是我只有3个格子的纸条，那么我该怎样告诉别人？

先告诉别人，第一个纸条乘以10+第二张纸条！

段的概念

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，使得我们用段地址来管理内存。

小结

- cpu访问内存单元是，必须向内存提供物理地址；

- 8086cpu内部用段地址和偏移地址相加的方法形成最终的物理地址
- cpu可以用不同的段地址和偏移地址形成同一个物理地址
- 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2^16-1个内存单元,即0~FFFFH，仅用偏移地址来寻址可寻64K内存单元；如给定段地址1000H，用偏移地址寻址cpu的寻址范围为：10000H~1FFFFH
- 数据在21F60H内存单元中，对于8086pc机中的两种描述：
    - 数据存在内存2000：1F60单元中
    - 数据存在内存的2000段中的1F60H单元中

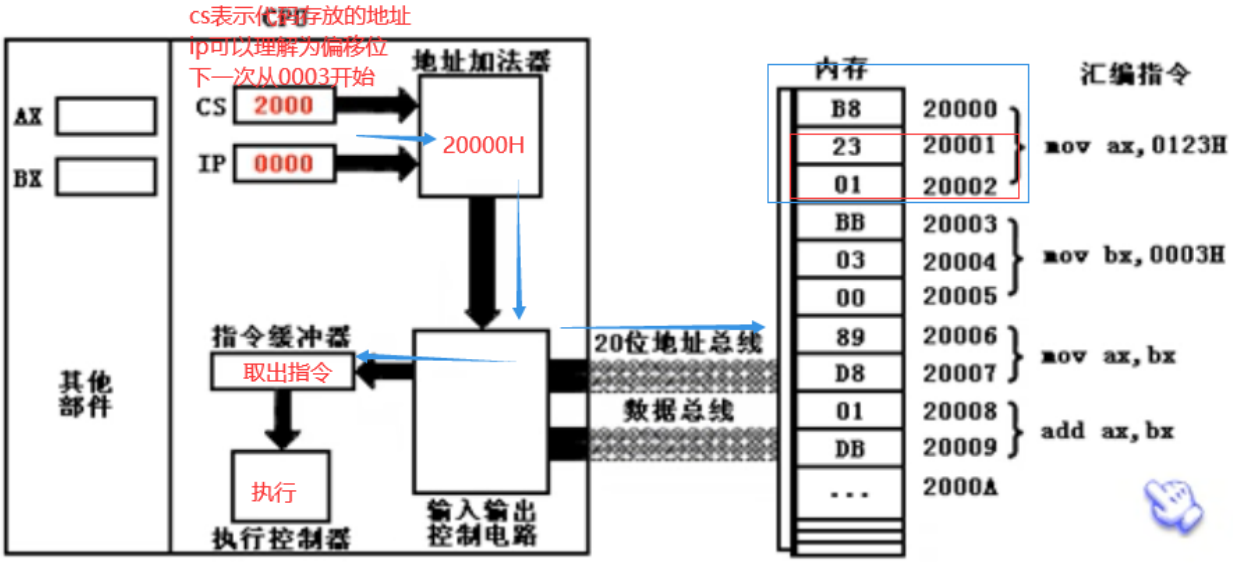
段寄存器

段寄存器就是提供段地址的；8086cpu有4个段寄存器：CS（代码段地址寄存器）、DS（数据段地址寄存器）、SS（堆栈段地址寄存器）、ES（傅家段地址寄存器）

当8086cpu要访问内存是，由这4个段寄存器提供内存单元的段地址。

CS和IP是8086cpu中最关键的寄存器，他们指示了cpu当前要**读取指令的地址。**

- CS为代码段地址寄存器
- IP为指令指针寄存器



8086pc工作过程的简述

- 从CS:IP指向内存单元读取指令，读取的指令进入指令缓冲器；
- IP = IP + 所读取指令的长度，从而指向下一条指令；IP自己维护了一个自增器
- 执行指令，跳转步骤1，重复

在8086cpu开始工作（开机）CS和IP被设置为CS=FFFFH，IP=0000H，即在8086pc机刚启动时，cpu从内存FFFF0H单元中读取指令执行，FFFF0H单元中的指令是8086pc机开机的第一条指令。

在任何时候，cpu将CS、IP中的内容当作指令的段地址和偏移地址，用他们合成指令的物理地址，到内存中读取指令代码，执行。如果说，内存中的一段信息曾被cpu执行过，那么它所在的内存单元必然被CS：IP指向过。

修改CS、IP的指令

在cpu中，程序员能够用指令读写的部件只有寄存器（用户态），程序员可以通过改变寄存器中的内容实现对cpu的控制，cpu从何处执行指令是由CS、IP中的内容决定的，可以通过改变CS、IP中的内容来控制cpu执行目标指令。

如何修改AX中的值?

```
mov 指令
如：mov ax，123
mov指令可以改变8086cpu的大部分寄存器的值，被称为传送指令，那能够通过mov指令改变CS、IP的值吗？不能！
```

mov指令不能用于设置CS、IP寄存器的值，8086cpu没有提供这样的功能。而是提供了另外的指令来改变他们的值：转移指令（jmp），功能：用指令中给出的段地址修改CS，偏移地址修改IP。

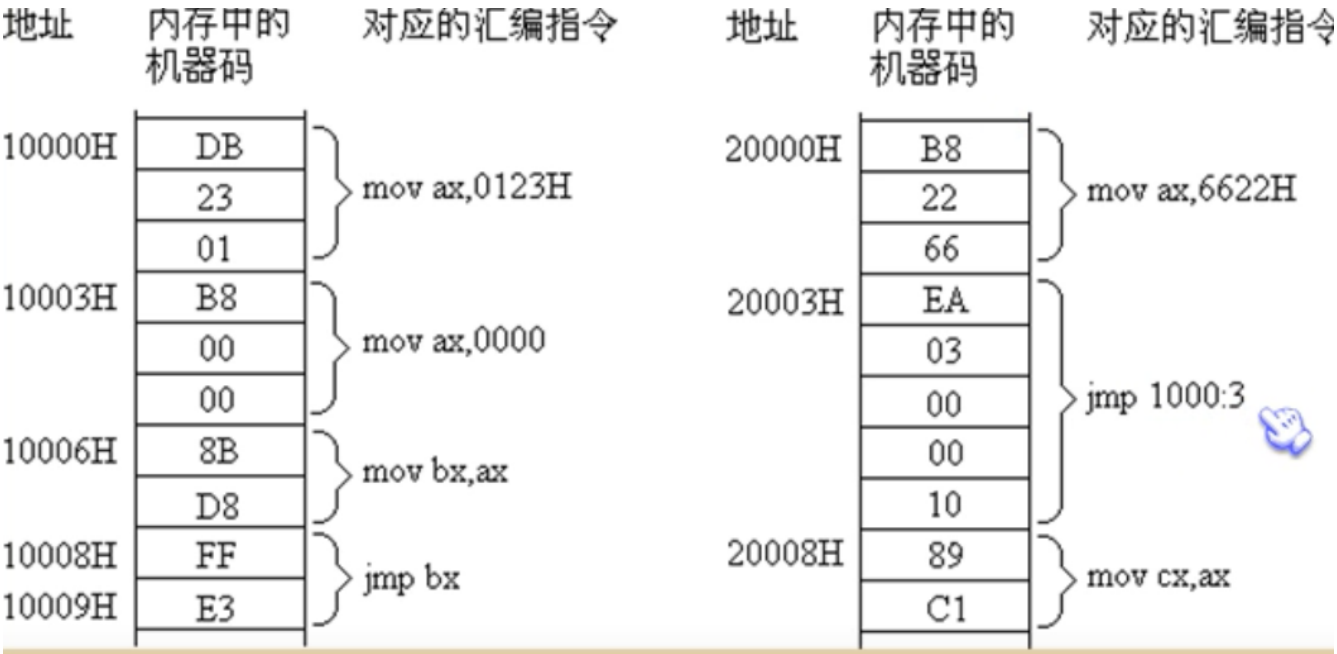
同时修改CS、IP寄存器的内容：

```
语法：jmp 段地址：偏移地址

jmp 2AE3：3    #表示cpu跳到2AE33H的地址上
jmp 3：0B16    #注意段地址3，表示0003H，所以正确跳转地址：0B46H
```

仅修改IP的内容：

```
语法：jmp 某一个合法寄存器，功能是用寄存器中的值修改IP
jmp ax  # 类似于 mov IP, ax
jmp bx

如：我们想IP跳转200H
mov ax, 200H
jmp ax
```

例题：内存中存放的机器码和对应的汇编指令如下：（初始值：CS=2000H，IP=0000H）



执行步骤如下：

```
1、mov ax, 6622H  # 将6622H赋值个ax
2、jmp 1000: 3    # 修改了CS、IP的值，进行cpu跳转寄存器，IP偏移3
3、mov ax,0000    # 将ax的值赋值为0000
4、mov bx, ax     # 将ax的值给bx
5、jmp bx         # 此时段地址没有变还是10000H，IP为0000，所以跳转到开头
6、mov ax, 0123H   # 将0123H给ax,
7、在往下走，一直死循环
```

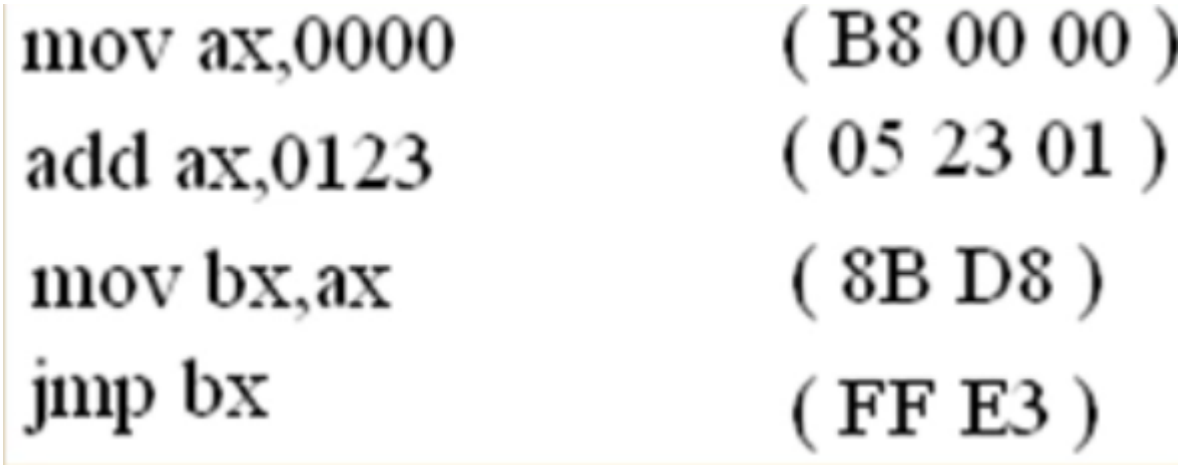
代码段

我们经常在高级预言中会使用定义一些函数时会称一个代码段，亦或者在Python中的缓存机制，小数据池等，表示在一个代码段的时候，我们可以随意访问这个代码段的所有变量等，这里我们就具体的来讲一讲代码段。

对于8086pc机来说，在编程时，**可以根据需要，将一组内存单元定义为一个段。**可以将长度为N（N<=64KB)的一组代码，存在一组地址连续、起始地址为16的倍数的内存单元中这段内存是用来存放代码的，从而定义了一个**代码段。**

需要注意的是：我们知道一个偏移地址最大容量是64KB，所以代码段的长度不能超过64KB

如：



这段长度为10字节的指令，存在从123B0H~123B9H的内存中，我们就可以称他是一个段地址为123BH，长度为10字节的代码段。

如何使得代码段的指令被执行?

将一段内存当作代码段，仅仅是我们在编程时的一种安排，cpu并不会由于这种安排，就自动将我们定义的代码段中的指令当作指令来运行。必须要CS、IP来进行指向才能取出、执行，换句话说，cpu只认被CS、IP指向的内存单元中的内容为指令。