**汇编语言**

笔记

冰糖雪狸

2020

目录

[一. 基础 2](#_Toc46318136)

[二. 汇编指令放在什么地方 3](#_Toc46318137)

[三. 内存编号为什么从0开始 4](#_Toc46318138)

[四. 地址线 数据线 控制线 5](#_Toc46318139)

[五. 加深对内存的认识 6](#_Toc46318140)

[六. 承上启下 7](#_Toc46318141)

[七. 第一章总结 8](#_Toc46318142)

[八. AX BX CX DX寄存器 9](#_Toc46318143)

[九. 寄存器运算检测 10](#_Toc46318144)

[十. 地址寄存器 – 表示内存地址信息的寄存器 11](#_Toc46318145)

[十一. 检测点2.2 12](#_Toc46318146)

[十二. CPU如何区分指令和数据 13](#_Toc46318147)

[十三. IP寄存器和指令 14](#_Toc46318148)

[十四. 检测点2.3 15](#_Toc46318149)

[十五. Debug调试工具使用总结 16](#_Toc46318150)

[十六. 承上启下 17](#_Toc46318151)

[十七. 寄存器（内存的访问） 18](#_Toc46318152)

[十八. 字节型数据、字型数据小结 20](#_Toc46318153)

[十九. 栈 21](#_Toc46318154)

[二十. 承上启下 22](#_Toc46318155)

[二十一. 源程序、编译和链接 24](#_Toc46318156)

[二十二. LOOP循环指令 25](#_Toc46318157)

[二十三. segment段空间 26](#_Toc46318158)

# 基础

计算机编程语言是一套**表示**系统

汇编语言 – 低级语言

C/C++/Python – 高级语言

汇编语言是一种观测手段，从机器的角度思考问题

计算机是由指令驱动的，指令告诉计算机这里怎么做，那里怎么做

机器指令 010101010111010 - 表示高低电平

CPU – 中央处理器

Test：

DOSBOX – debug -u

【相同的相同的一串16进制数字：内存编号】 【十六进制数字】 【一些单词缩写 + 一些数字】

在DOBBOX 中：

编号信息 – 机器之类 – 汇编指令

机器指令 | 汇编指令

-------------------------------------------------------

74 03 JZ 0105

A = 10

B = 11

C = 12

D = 13

E = 14

F = 15

0111 0100 0000 0011

-------------------------------------------------------

E9 97 00 JMP019C

1110 1001 1001 0111 0000 000 0

-------------------------------------------------------

B8 05 00 MOV AX,0005

1011 0000 0101 0000 0000

-------------------------------------------------------

MOV - move - 移动

将0005移动到 一个叫AX的寄存器中

**总结：**

1. 汇编指令 – 通过编译器翻译成机器指令（机器码），机器指令就是一串0和1
2. 伪指令 – 告诉编译器，怎么翻译
3. 符号体系 - +-\*/ 编译器管

**题目：**

1. 什么是机器指令 0101010001110
2. 机器指令由谁执行 CPU
3. 机器指令和汇编指令有什么关系 通过编译器 汇编指令->机器指令
4. 什么是编译器 翻译软件
5. 什么是伪指令 告诉编译器，怎么翻译

# 二. 汇编指令放在什么地方

1. 汇编指令放在那里？

绝大多数指令和数据都是存放在内存条中

DOSBOX -debug -u -d

u指令和d指令 解释不同

u – 机器指令和汇编指令

d – 数据

同一串十六进制数字产生了两种解释 -> 计算机怎么区分的？

**内存的最小单元是什么？**

74***H*** 03***H*** E9***H*** – 一个字节 = 2个16进制位 = 8个2进制位 **0110 0111**

**↑-这里的H代表 16进制**

**B – 二进制 – Binary**

**O – 八进制 - Octonary**

**D – 十进制 - Decimal**

**H – 十六进制 – Hex**

***比如：0100***

***🡺 4个 2进制位***

***🡺 1个 16进制位***

【Byte – 字节】

1Byte = 8bit 1bit = 1个2进制位 01

1KB = 1024Byte k = kilo b = Byte

1MB = 1024KB m = million B = Byte

1GB = 1024MB g = giga(千兆，亿)

**最小寻址单位：**

最小寻址单位是指特定的计算机硬件机构所支持的最小数据访问块大小。以个人电脑为例，**内存机构的最小寻址单位为1个字节（1 Byte）即8个bit。**也就是说，你**无法单独访问1 bit的信息或者任意小于1字节的信息**。个人电脑中的硬盘部分最小访问单位为4KB（依厂商不同而有所区别，较早的硬盘该单位比较小），这就是通常所讲的“硬盘按块寻址”，一块既指4KB的数据。

测试：

1. 机器指令（汇编指令）和数据存放在哪里？ 内存
2. 机器指令（汇编指令）是以什么形式存放在内存中的？ 二进制 （Debug将2进制->16进制，方便阅读）
3. 内存（存储单元）它的编号是怎么计算的？ 从0开始
4. 内存的计量单位？ 在上面

# 三. 内存编号为什么从0开始

CPU中存放了一定的指令和数据

其他存放在内存中，绝大部分存放在内存条中（内存条是内存的一部分）

指令和数据在内存中是没有区别的

CPU和内存条在主板上通过电路连接，他们之间的联系需要通过这些电路

CPU要从内存条中读取和写入数据怎么做？（写入 – 放回内存）

需要什么信息：

内存编号信息 地址线 - 内存地址

数据信息 🡪 数据线

读写信息（是读还是写） 控制线

电路能表示高低电平状态，0和1

如果只有一根地址线，要寻找地址为3的内存能找到吗？

找不到，因为物理上限制住了。

【寻址能力】地址线的大小，决定了能找到的**最大的**内存地址

n根地址线就能表示寻找能力为【0 ~ (**2n – 1**) 】

测试：

1. 内存编号为什么从0开始？ 物理上的限制，电路只能表示0或者1
2. 什么叫寻址能力 地址线的数量，表示了能传几个0和1，决定了能找到的最大内存地址

# 四. 地址线 数据线 控制线

地址线 – 决定了CPU的寻址能力

数据线 – 决定了CPU和其他部件 进行数据传送时 一次性能够传送多少数据的能力

控制线 – 决定了CPU对其他部件的控制能力（主板上不止有内存条，还有其他硬件）

检测：

1 Byte = 8 bit

1 KB = 1024 Byte 1 MB = 1024 \* 1024 Byte = 2^10 \* 2^10 Byte = 2^20 Byte

1 MB = 1024 KB 1 GB = 1024 \* 1024 \* 1024 Byte = 2^30 Byte

1 GB = 1024 MB

检测：

* 1个CPU的寻址能力为 8 KB，那么他的地址**总线宽度**为 2^10 \* 2^3 = 2^13🡺13**根**
* 1KB的存储器有\_\_\_1024\_\_\_存储单元的编号从\_0\_到\_\_\_\_1023\_\_\_
* 1KB的存储器可以存储\_\_\_\_2^13\_\_\_\_\_\_bit，\_\_\_\_1024\_\_\_\_\_\_Byte
* 1GB，1MB，1KB 分别是\_2^30，2^20，2^10 Byte
* 1个CPU的寻址能力为8KB, 那么它的地址总线的宽度为**13**  
  解答：8KB=(1024 x 8)Byte=(2^10 x 2^3)Byte=2^13Byte  
  **1Byte为一个存储单元**，**一根地址总线寻址能力为2^1个存储单元**，因此地址总线宽度为13.   
  **注: 寻址就是用一个数字表示一个字节存储单元的地址，而不是表示这个存储单元中的内容，所以不用乘以8**
* 1KB的存储器有1024个存储单元,存储单元的编号从0到1023。  
  解答：1个存储单元存储1Byte，1KB=1024Byte，也就是1024个存储单元。存储单元编号从0开始。
* 1KB的存储器可以存储2^13个bit,1024个Byte。  
  解答：1Byte=8bit,1KB=1024x8bit=2^10x2^3=2^13个bit。
* 8080,8088,80286,80386的地址总线宽度为16根，20根，24根，32根 则他们的寻址能力分别为：**64 KB, 1MB, 16MB, 4GB**

1KB=1024B=(2^10)B 1MB=1024KB=(2^20)B 1GB=1024MB=(2^30)B  
16根为(2^16)B=(2^6)KB=64KB  
20根为(2^20)B=(2^10)KB=(2^0)MB=1MB  
24根为(2^24B)=(2^14)KB=(2^4)MB=16MB  
32根为(2^32)B=(3^22)KB=(3^12)MB=(2^2)GB=4GB

* 8080, 8088, 8086, 80286, 80386的数据总线宽度为8根，8根，16根，16根，32根 则他们依次可以传送的数据为 1B, 1B, 2B, 2B, 4B

**解答：1根数据总线传输一个内存二进制位信息(bit)，即1bit，8bit构成1Byte。  
8根传输8bit=1Byte、16根传输16bit=2Byte、32根传输32bit=4Byte。**

* 从内存中读取1024字节的数据，8086至少要读512次，80386至少要读取 256次

解答：1024字节即1024Byte，根据第6题，8086一次读取2Byte数据，因此要读1024/2=512次，80386一次读取4Byte，因此要读1024/4=256次

* 在存储器中，数据和程序（指令）以 **二进制** 形式存放

解答：因为存储器是直接与CPU交流，而CPU只认识0和1的二进制数，因此是以二进制形式存储

# 五. 加深对内存的认识

e B800:400 回车

1空格 1空格

2空格 2空格

…

**e 修改内存地址中的内容**

* 内存地址是不是内存条中的内存地址？ 显然不是
* 计算机有很多部件。内存条是计算机的一个部件，显卡（显存）插在主板上
* CPU和计算机各个部件的关系？ 内存条 显卡（显存）-> 给他们编号
* 什么是RAM内存？ 允许读取和写入 断电丢失指令和数据
* 什么是ROM内存？ 只允许读取 断电指令和数据还存在 通过内存地址访问 启动计算机上面

CPU

内存空间

内存条（假如编号是100-200）

显存（假如编号是200-300）

ROM（假如编号是300-400）

假如修改200-300编号内容，则会修改显存上的数据

假如修改300-400编号的内容，**不允许！！ROM只读！**

鼠标（端口号）

键盘

端

口

* CPU也是通过内存地址访问鼠标键盘吗？

不是，是通过【端口 - port】，鼠标和键盘里都有一块芯片，同样能储存指令和数据

通过端口号 60H就是端口号

input out 和端口有关 读取-写入 控制线读写信息

CPU可以通过主板上的电路读到所有数据

CPU就像人类的大脑

主板就像人的骨骼

主板上的电路是什么？ 附加在骨骼上的神经

大脑得到反应，CPU得到数据

# 六. 承上启下

内存编号为什么是 073F:02CE 这样的形式？

内存编号为什么是从0开始？

端口号：CPU也给鼠标键盘，麦克风，音响 编了号，通过这些编号可以读取设备状态

CPU通过内存地址访问内存空间

RAM – 断电丢失指令和数据 允许读取和写入

ROM – 断电存在 只允许写入

GPU – 图形处理芯片 很久以前没有这玩意 现在对图形要求越来越高，需要分工，GPU专门对图形处理

GPU有专门的编程语言去编程 需要比较强的物理和数学知识 🡺 果断放弃

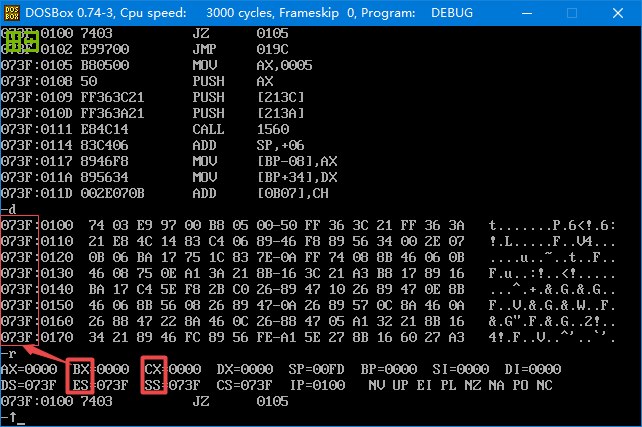
从B800:0000 400 开始当做显存就可以

汇编语言是针对CPU的，针对CPU 的地址线，数据线，控制线

CPU中一定有可以存放 地址信息 数据信息的地方🡺寄存器

汇编程序员就是通过汇编语中的汇编指令去修改寄存器中的内容，从而控制CPU，从而控制整个计算机

MOV AX,0005 AX-数据寄存器



DS ES SS CS 都是冒号左边的一种地址信息 IP比较像

# 七. 第一章总结

# 八. AX BX CX DX寄存器

在计算机中AX BX CX DX 寄存器比较特殊，因为他们可以各自分割为2个8位寄存器

1 Byte = 8 bit

2 Byte = 16 bit

他们是16位寄存器：0000 0000 0000 0000

AH AL

存储范围：0000 0000 0000 0000 ~ 1111 1111 1111 1111 或者 0~FFFF***H*** 或者0~65535

各自分割为2个8位寄存器：

AX = AH + AL AH构成高8位寄存器 AH寄存器 H = High

BX = BH + BL AL构成低8位寄存器 AL寄存器 L = Low

CX = CH + CL

DX = DH + DL

为什么或这样？

为了兼容，比如8086CPU有8位寄存器，为了保证以前编写的程序稍加修改就可以运行在8086CPU上

最小内存单元？

1字节 – 1Byte – 8 bit

CPU从内存中读取一个字节 8bit字节型数据 8位数据🡺8位寄存器中

数据线呢？

16根数据线 数据线的宽度决定了CPU一次性能够读取多长的数据

8086CPU一次性可以处理2种尺寸的数据

字节型数据 1Byte 8bit 🡺 8位寄存器中

字型数据 2Byte 16bit 🡺16位寄存器中（2字节）

字型数据：一个字节是高位字节（AH, BH, CH, DH）

一个字节是低位字节（AL, BL, CL, DL）

**数据与寄存器之间要保证一致性！！**

8位寄存器 🡺 8位寄存器 🡺 进行8位运算

8位数据 🡺 8位寄存器

16位数据 🡺 16位寄存器 🡺 进行16位运算

Debug中 ：

-a 输入命令 -r 逐条 -t 执行 **并且只认16进制数字**

**寄存器是互相独立的，在运算过程中是不会影响到其他寄存器的，如果加法运算超出了寄存器能够存储的范围，那么超出的部分会存到其他的地方去！AL就是AL，AH就是AH，不会相互影响！**

练习: 16位加法 16位加法 8位加法

**mov ax, 18h mov bx, ax mov ax, 0h mov ax, 4e20h**

**mov ah, 78h add ax, bx mov ax, 93h add ax, 1406h**

**add ax, 8h add al, 85h mov bx, 2000h**

**add ax, bx**

# 九. 寄存器运算检测

写出每条指令执行后寄存器中的值：

十进制

MOV AX, 62627 AX = F4A3 H

MOV AH, 31H AX = 31A3 H

MOV AL, 23H AX = 3123 H

ADD AX, AX AX = 6246 H

MOV BX, 826CH BX = 826C H

MOV CX, AX CX = 6246 H

MOV AX, BX AX = 826C H

ADD AX, BX AX = 1 04D8 H

MOV AL, BH AX = 0482 H

MOV AH, BL AX = 6C82 H

ADD AH, AH AX = D882 H

ADD AL, 6 AX = D888 H

ADD AL, AL AX = D810 H

MOV AX, CX AX = 6246 H

只能使用目前学过的汇编指令，最大使用4条命令，编程计算2的4次方：2\*2\*2\*2 = 16

MOV AX, 2

ADD AX, AX

ADD AX, AX

ADD AX, AX

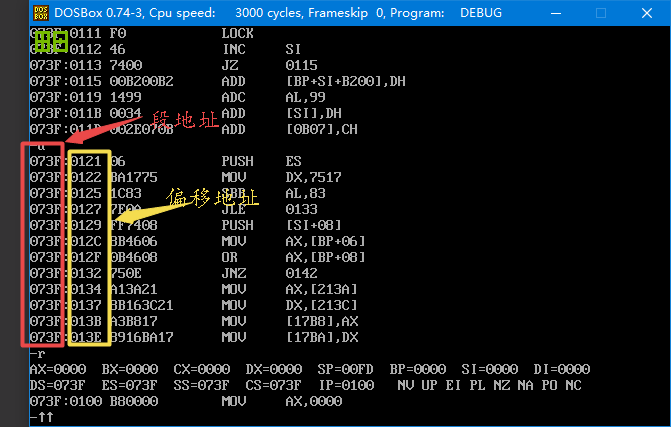
MOV AL, 2

ADD AL, AL

ADD AL ,AL

ADD AL, AL

# 十. 地址寄存器 – 表示内存地址信息的寄存器



存放段地址的：段地址寄存器

存放偏移地址的：偏移地址寄存器

段地址寄存器 : 偏移地址寄存器

DS SP

ES BP

SS SI

CS DI

IP

BX

8086CPU 有20根地址线 0000 0000 0000 0000 0000 ~ 1111 1111 1111 111 1111 0~FFFFF H

地址线的数量决定了CPU的寻址能力

地址加法器 🡺 地址的计算方式

计算方法：

**物理地址 = 段地址 x 16(10H) + 偏移地址**

**基础地址 = 段地址 x 16(10H)**

**物理地址 = 基础地址 + 偏移地址**

# 十一. 检测点2.2

**物理地址 = 段地址 \* 10H + 偏移地址**

段地址为0001H，仅通过变化偏移地址寻址，CPU的寻址范围为 10H 到 1000FH

解答：0001H \* 10H = 10H 🡸 基础地址

10H + (0 ~ FFFF)H = 物理地址

有一个数据存放在内存20000H单元中，现给段地址为SA，若想用偏移地址找到此单元，则SA应满足的条件为：最小 **(20000H – FFFFH) /10H = 1000 H(错误)， 正确结果：1000H + 1H = 1001H**，最大20000H/10H =2000H （提示：当段地址为多少时，CPU无论怎么改变偏移地址，都无法找到20000H）



# 十二. CPU如何区分指令和数据

u指令 🡪 将某个内存地址 开始的字节 全部当作 指令

d指令 🡪 将某个内存地址 开始的字节 全部当作 数据

CPU 中怎么表示内存地址的？

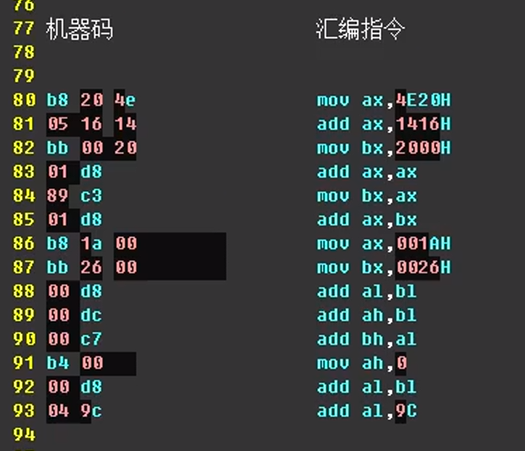
通过段地址寄存器和偏移地址寄存器

**8086CPU中，在任意时刻，CPU将CS:IP所指向的全部当作指令来执行**

在内存中，指令和数据是没有任何区别的。二者都是二进制信息，CPU只有在工作的时候才将有的信息当作指令，有的信息当作数据。

CPU根据什么将内存汇总的信息当作指令？

**CPU将CS：IP开始指向的内存单元中的内容当作指令**



将上面的机器码用 e 指令输入到2000:0这个位置

【内存也是内存，翻译成汇编指令 被当作指令来执行】

指令和数据在内存中有区别吗？ 没有

**CPU中的CS段地址和IP偏移地址寄存器组合的时候从中读取内容，当作指令执行！！**

**使用：-r CS 指令可以修改CS指向的地址**

**-r IP 修改IP指向的地址**

**总结：**

**CPU 任意时刻将 段寄存器CS 和 偏移地址寄存器 IP 组合出来的地址 从中读取出来的内容当作指令**

# 十三. IP寄存器和指令

关系：指令是有长度的 一条指令可以由多个字节构成

**指令的执行过程：**

1. **CPU从CS：IP所指向的内存单元中读取指令，存放到 指令缓存器 中**
2. **IP = IP + 所读指令的长度，从而指向下一条指令**
3. **执行 指令缓存器 中的内容，回到步骤1**

**汇编指令：jmp – jump的简写**

**jmp 🡺 转移指令，可以修改CS和IP这两个寄存器，决定了CPU从哪里读取指令**

-a jmp 2000:0 🡪 将CS：IP替换为2000:0

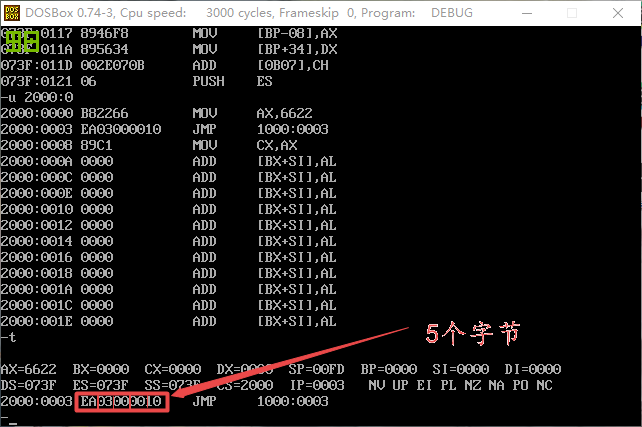
-a jmp 寄存器，比如jmp ip, ax 🡪 将**IP**中的内容替换为ax中的内容

-a jmp 寄存器，比如jmp bx 🡪 将**IP**中的内容替换为bx中的内容

Call 转移指令 – 会把当前地址暂存起来，类似于PUSH？ 存疑

Ret 转移指令 – 取回call暂存的指令到IP中，类似于POP？ 存疑

**【上机测试】测试：**

假设CS = 2000H，IP = 0000 写出下列指令的执行过程

A 2000:0 CS = 2000 IP = 0

MOV AX, 6622H CS = 2000 IP = 0 + 3 = 3D

JMP 1000:3 CS = 2000 IP = 3 + 5 = 8D

MOV CX, AX

A 1000:0 CS = 1000 IP = 3D

MOV AX, 0123H

MOV AX, 0 CS = 1000 IP = 3 + 3 = 6D

MOV BX, AX CS = 1000 IP = 6 + 2 = 8D

JMP BX CS = 1000 IP = 8 + 2 = 10D

MOV CX, 0

# 十四. 检测点2.3

下面3条指令执行后，CPU几次修改IP？都是在什么时候修改的？最后IP中的值是多少？

MOV AX, BX 🡪 存放到指令缓存器中 🡪 **修改1次**，指向“SUB AX, AX” 🡪 执行 AX = BX

SUB AX, AX【减法】🡪 存放到指令缓存器中 🡪 **修改一次**，指向“JUM AX” 🡪执行 AX = AX – AX

JUM AX 🡪 存放到指令缓存器中 🡪 **修改一次** 🡪 **执行 IP = 0**

修改了4次 IP，最后IP = 0

# 十五. Debug调试工具使用总结

**r 查看和改变寄存器中的内容，将CS:IP所指向的机器码，翻译成汇编指令**

 -r CS 🡨 改变CS中的内容

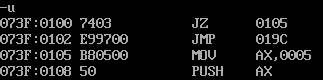
 -r 🡨 查看寄存器中的内容

**d 查看内存中的内容 【内存地址 - 内存地址中相应的内容 - ASCII码】🡨一次为128节**

-d 1000:0 🡪 查看以1000:0开始的128个字节

 -d 1000:8 F 🡪 查看0到F之间的内容

-d 1030:3 2F 🡪 查看0到2F之间的内容

**u 将内存中的机器指令翻译成汇编指令**

-u 1000:0 🡪 从1000:0开始的机器码翻译成汇编指令

**a 以汇编指令的格式在内存中写入一条汇编指令**

-a 🡪 在当前地址上写入（最终写入的是机器码）

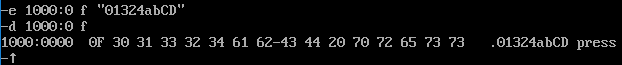
-a 🡪 2000:0 在2000:0地址上写入（最终写入的是机器码）

**t 执行当前CS:IP所指向的机器指令**

**e 改写内存中的内容，类似于a指令**

-e 2000:0 🡪 一个一个的修改

-e 2000:0 “01234abCD” 🡪 直接修改成引号里对应的ASCII码



**p 当执行loop循环时，跳过loop循环**

# 十六. 承上启下

**寄存器：**

数据寄存器

地址寄存器 🡺 段地址寄存器：偏移地址寄存器

标志位寄存器

**通用寄存器：**

AX

BX 也可以被当作位偏移地址寄存器

CX 也有其他作用

DX AX DX 用来处理数据的

他们都是16位寄存器，可以分割为两个独立的8位寄存器

AX = AH +AL

BX = BH + BL

CX = CH + CL

DX = DH + DL

要注意保证数据与寄存器，寄存器与寄存器位数的一致性，编译器通过寄存器的位数来判断是存放几位

**指令：**

MOV 移动指令

ADD 运算指令 运算时如果超过寄存器最大值，存放到了其他地方，**并不是消失了**

**【本章重点】CPU是如何区分指令和数据的？**

CS：IP这两个寄存器，在任意时刻CPU将段地址寄存器CS和偏移地址寄存器IP组合出来的地址中读取的内容，全部当作指令来执行

指令是有长度的，跟IP寄存器有关

**可以修改CS：IP这两个寄存器的指令：**

转移指令： jmp 2000:0

Jmp 寄存器格式 类似于：mov ip, ax(但不能这么写)

Call xxxxx 指令执行的过程 🡺 改变了的IP保存了起来

**指令和数据 存放在内存中是没有任何区别的，如何产生不同意义的呢？**

是汇编程序员通过修改寄存器中的内容，告诉编译器数据在哪里，指令在哪里

# 十七. 寄存器（内存的访问）

***通过观察和思考去猜测设计者为何如此设计***

有几种数据长度？

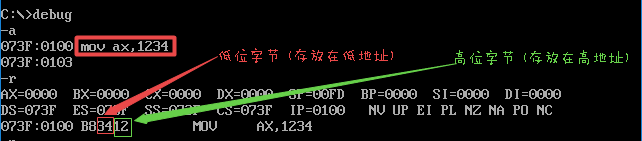
字节型数据和字型数据(2Byte)

在内存中这些数据是如何存放的？

一个字型数据存放在内存单元中可以由**2个连续**的内存地址的内存单元组成

高地址内存单元存放 🡪 字型数据的高位字节

低地址内存单元存放 🡪 字型数据的低位字节



问题3.1

1. 20H
2. 4EH
3. 12H
4. 00H

地址0中存放的 字节型数据 是多少？ 20H

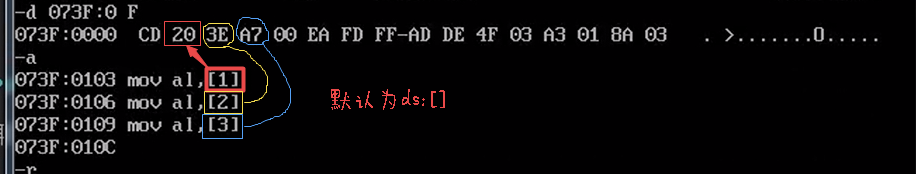
地址0中存放的 字型数据 是多少？ 4E20H 4E是高位字节 20是低位字节

地址2中存放的 字节型数据 是多少？ 12H

地址2中存放的 字型数据 是多少？ 0012H 00是高位字节 12是低位字节

**地址1中存放的 字型数据 是多少？ 124EH 12是高位字节 4E是低位字节**

**测试1：**



Mov al, ds:[0] 分析：al – CPU中的AL寄存器

Mov al, ds:[1] ds – 内存地址 【data segment – 数据段地址寄存器】

Mov al, ds:[2] []内的 – 偏移地址

Mov al, ds:[3] [] – 得到里面的内容

[number] – 段地址与偏移地址组合得到的物理地址

Mov al, ds:[0] 🡪 从内存中读取了多长的数据？ **一个字节8位寄存器**

Mov bx, 1000H

Mov ds, bs

Mov ax, ds:[0]

Mov ax, ds:[1]

Mov ax, ds:[2]

Mov ax, ds:[3]

Mov ds:[0], ax

Mov ds:[1], ax

Mov ds:[0], al

Mov ds:[1], ah

**测试2：**

写出下面的指令执行后，寄存器ax, bx, cx 中的值

Mov ax, 1000H

Mov ds, ax

物理地址 内容 | AX BX CX

Mov ax, ds:[0] 10000H 23H | 1123H

Mov bx, ds:[2] 10001H 11H | 6622H

Mov cx, ds:[1] 10002H 22H | 2211H

Add bx, ds:[1] 10003H 66H | 8833H

Add cx, ds:[2] | 8833H

运算时一定要确定寄存器的长度

测试3：

写出下面的指令执行后，内存中的值

Mov ax, 1000H

Mov ds, ax 🡨 确定了段地址

物理地址 内容 | ds: 23 11 22 11

Mov ax, 2C34H 10000H 23H | ds: 34 2C 22 11

Mov ds:[0], ax 10001H 11H | ds: 34 2C 22 11

Mov bx, ds:[0] 10002H 22H | BX: 2C34H

Sub bx, ds:[2] 10003H 11H | BX: 1B12H **Sub: 逗号左边的值-右边的，结果给左边 bx = bx – ds:[2]**

Mov ds:[2], bx | ds: 23 11 12 1B

# 十八. 字节型数据、字型数据小结

***低地址存低位字节，高地址存高位字节***

***寻址时用低地址作为基址***

* 字节型数据，字型数据在内存中存放
* 字型数据在内存中存放时，需要2个地址连续的内存单元存放
* 高位字节 存放在 高地址
* 低位字节 存放在 低地址
* 在使用 mov add sub 时要注意数据和寄存器的长度
* ds es ss cs – 段地址寄存器 【ds - 数据段地址寄存器】【cs – 指令有关】
* 通过 ds:[number] 可以确定一个物理地址
  + Ds – 段地址
  + :[number] – 偏移地址
* Cs:ip – 指令从哪里来
* Ds – 数据从哪里来
* 通过修改寄存器中的内容 🡪 控制CPU 🡪 从而控制整个计算机

格式：

Mov 寄存器，数据 mov ax, 8

Mov 寄存器，寄存器 mov ax, bx

Mov 寄存器，**内存单元** mov ax, **ds:[0]**

Mov **内存单元**，寄存器 mov **ds:[0]**, bx

Mov **段寄存器**，寄存器 mov **ds**, ax

add/sub寄存器，数据 add/sub ax, 0

add/sub寄存器，寄存器 add/sub bx, ax

add/sub寄存器，**内存单元**  add/sub bx, **[2]**

add /sub**内存单元**，寄存器 add/sub **[0]**, ax

# 十九. 栈

栈 – 是一段连续的内存单元，也就是一段**连续的内存地址**

入栈 – push 🡪 将16位寄存器/内存中的字型数据 🡪 栈顶标记上面

出栈 – pop 🡪 将栈顶标记所标识的字型数据 🡪 16位寄存器/内存中

栈顶标志 🡪 内存地址

在**8086CPU中，任意时刻将段地址寄存器SS和偏移地址寄存器SP所组合出来的内存地址当作栈顶标记！**

数据从哪里来？ 1. 寄存器 2. 内存

数据的类型？ 1. 字节型数据 2. 字型数据

**入栈和出栈操作的是字型数据【2Byte】- 16位寄存器**

-----------------------------------------------------------------------------------

举例：

Push ax 1. 修改SP寄存器中的数值 **SP = SP – 2**

2. 将 AX 中字型数据 放到 SS:SP所组合出来的**内存地址**中 入栈

Pop bx 1. SS:SP所组合出来的内存地址中的**字型数据**传给bx

2. 修改栈顶标记 **SP = SP + 2** 🡨 成为新栈顶标记 出栈

-----------------------------------------------------------------------------------

栈的设置：

起始地址 + 你所设定的栈的大小的字节数【最好是16 的倍数，防止稀奇古怪的问题出现】

0000 + 16（10H） = 10H SS:SP所组合出来的栈顶标记

栈的越界：

1. 在汇编中CPU总把SS:SP指向的内存地址作为栈顶标志，所以无法判断是否越界
2. 要根据自己的需求合理设置栈的大小
3. 如果SS:SP = 1000:0000 ，继续pop会变成1000:FFFD
4. 栈一定不要越界，否则还可能破坏其他寄存器中的数据，从而造成一连串的未知的错误！！！

栈的极限大小：

SP寄存器的变化范围： 0~FFFFH 32768个字型数据（64kb）

SS = 2000H SP = 0 设置了可以存放32768个字型数据的箱子

如果一直push，当SP又 = 0是，如果再push会 覆盖栈中原来的内容

栈的作用：

1. 临时性保存数据
2. 进行数据交换
3. 实质上就是一种mov移动指令，通过SS:SP

# 二十. 承上启下

在内存中存放（自定义）数据的段 🡪 数据段

在内存中存放（自定义）指令的段 🡪 指令段/代码段

将一段内存定义为栈空间，存放临时性数据 🡪 栈段

如何让CPU按照我们的安排去访问这些内存段呢？数据、指令从哪里来，临时性数据存放到哪里去？

* 对于 数据段 🡪 ds寄存器 [0], [1], [2] … mov add sub 指令去访问这些内存单元。CPU就会将我们定义的数据段的内容当作数据来访问
* 对于 代码 🡪 通过修改CS:IP这两个寄存器的值去指向我们定义的代码段，这样CPU就将执行我们定义的代码段中的指令
* 对于 栈段 🡪 通过修改SS:IP 这两个寄存器去决定栈顶标记在哪里。 这样CPU在执行栈的操作比如PUSH, POP就会将我们定义的栈段当作栈空间使用，进行临时性数据的存放或者取出。
* 不管我们如何安排，CPU将内存中的某段内容当作 指令 是因为CS:IP指向那里
* 不管我们如何安排，CPU将内存中的某段内容当作 栈空间 是因为 SS:IP指向那里
* 我们一定要清楚我们是如何安排内存的。以及如何让CPU按照我们的安排去行事

小汇总：

* DS 🡺 数据段
* CS:IP 🡺 指令段
* SS:SP 🡺 栈段

汇编指令 🡪 被编译器翻译成010001的机器指令，机器码由CPU执行

伪指令 🡪 由编译器执行

符号体系 🡪 由编译器执行

编译和连接：

* 汇编指令 🡪 被编译器翻译成010001的机器指令，机器码由CPU执行
* 伪指令 🡪 由编译器执行
* 符号体系 🡪 由编译器执行
* 系统要运行exe文件的话，需要给他分配一段内存
* 系统怎么知道要分配多大内存给这个程序的？也就是EXE文件？
  + exe文件中除了拥有整个程序，还包括了一些信息 ↓
  + 文件有多大 – 程序在哪里 – 描述信息
  + 系统就是根据这些描述信息对寄存器进行相关的设置
* **start** – **伪指令**告诉编译器 🡪 程序的入口在哪里，然后系统通过这个描述文件中的内容去设置CS:IP，当然还有其他的一些内存（这些信息都被记录在了exe文件中）
  + data segment 🡪 告诉编译器 data段从这里开始
  + data end**s** 🡪 告诉编译器 data段从这里结束
* 段的名字 可以随便取以方便我们阅读
  + add、sub 是汇编指令，但乘法运算在汇编语言中肯定是有对应的汇编指令的

程序返回的功能：

* 系统在加载程序时给程序分配内存，设置寄存器
* 程序结束后这些内存和寄存器都需要归还给系统，不得永久占用，内存是有限的

程序的跟踪（在DOXBox中）：

* debug + 程序名
* cx = 程序的长度
* p - 执行int指令，而不是普通的-t指令
* q - 退出
* PSP区 – 从DS:0开始的256个字节 🡪 系统会将程序加载到内存中，**PSP区是用来系统和内存之间进行通信的**

# 二十一. 源程序、编译和链接

* 规则：

mov ax, **0**FFFFH 🡨 首个字符若为字母，则前面必须加上一个0！

* 10 和 10H和的区别
  + 10为十进制中的10，但10H为16进制中的10
* exe文件执行的过程，事实上就是内存分配的过程
* 安排自己的段（segment）：代码段，数据段，栈段

# 二十二. LOOP循环指令

* LOOP循环指令 – 循环指令 – 按照次数来跳转
* JMP跳转 – 跳转指令
* LOOP指令的执行次数（循环次数）保存在cx寄存器中
* LOOP循环指令的两个步骤：

1. cx = cx – 1
2. 判断cx中的值，不为0则跳转（jmp）到 标号（内存地址）位置，继续执行

等于0则执行下面的指令

* p指令可以直接跳过LOOP指令
* g + 某条指令的内存地址：执行到某条指令

# 二十三. segment段空间

* 一个段（segment）实际占用多少空间？
  + 在8086CPU中只能是**16**的倍数
  + 一个segment最少占用16字节
  + 假如定义了一个内存段：
    - datasg segment

dw 2, 4, 5 🡨占用16字节

datasg ends

* + - datasg segment

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

dw 8, 9, 10, 11 🡨 占用16 \* 2 = 32字节

datasg ends

# 转移指令

转移指令 – 可以修改IP或者同时修改CS和IP

jmp 无条件转移指令

loop 条件转移指令，根据CX寄存器中的值