Chapter 2

寄存器和地址寻址

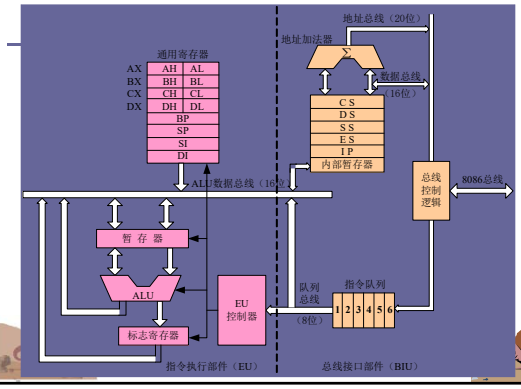
1、CPU概述

- 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。

Intel 8086 微处理器

- 16位数据
- 20位地址
- 物理寻址1Mbyte

8086CPU的编程结构

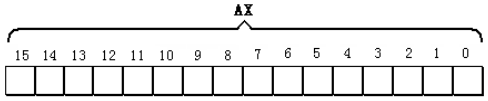


8086/8088 CPU的内部寄存器

- 通用寄存器
  - 数据寄存器
    - AX（AH，AL）
    - BX（BH，BL）
    - CX（CH，CL）
    - DX（DH，DL）
  - 指针寄存器
    - 堆栈指针寄存器 SP
    - 基址指针寄存器 BP
  - 变址寄存器
    - 源变址寄存器 SI
    - 目的变址寄存器 DI
- 专用寄存器
  - 控制寄存器
    - 指令指针 IP
    - 标志寄存器 FLAG（FH，FL）
  - 段寄存器
    - 代码段寄存器 CS
    - 堆栈段寄存器 SS
    - 数据段寄存器 DS
    - 附加段寄存器 ES

2.1 通用寄存器

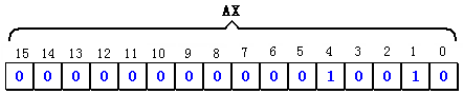
- 存放一般性数据的通用寄存器： AX、BX、CX、DX
- 8086CPU所有的寄存器都是16位的，可以存放两个字节。



16位寄存器的逻辑结构

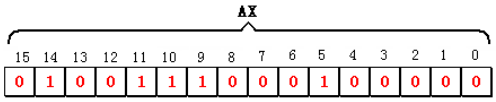
16位数据在寄存器中的存放情况

- 数据：18
- 二进制表示：10010
- 在寄存器AX中的存储：



16位数据在寄存器中的存放情况

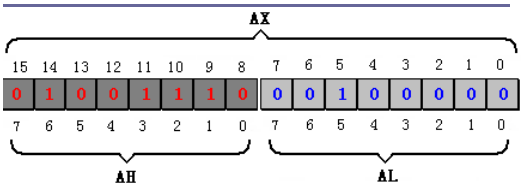
- 数据：20000
- 二进制表示：0100111000100000
- 在寄存器AX中的存储：



2.1 通用寄存器

- 为保证和上一代CPU保持兼容性，这四个通用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。

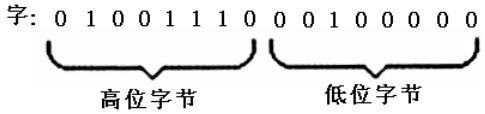
以AX为例：



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）

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。



3、数值数据

- 定点数：小数点位置固定
  - 8086是通用处理器，只处理定点数
- 浮点数：小数点位置可变，比定点数表示的数的范围大，精度高
  - 8087协处理器，与8086配套的浮点运算部件
  - 浮点计算、三角函数、指数函数、对数函数的运算

数值数据

- 1、无符号数值数据的表示形式
  - 二进制数：以“B”结尾，如00001101B；
  - 八进制数：以“O”结尾，如725O；
  - 十六进制数：以“H”结尾，如0A12H；
  - 十进制数：以“D”结尾或无任何字母作结尾，如10D、10。
- 注意：十六进制数据不能以字母开头，前面加‘0’

数值数据

- 2、有符号数值数据的表示形式
- 80X86采用n位二进制补码表示
  - 原码：将最高位作为符号位(0表示正，1表示负)，其它数字位用数值本身的绝对值表示。
    - 数字（**−6**）在8位计算机中的原码表示为：**1**000 0110
  - 反码：如果是正数，则表示方法和原码一样；如果是负数，则保留符号位1，然后将这个数字的原码按照每位取反。
    - 数字（**−6**）在8位计算机中的反码表示为：1111 1001

- 补码：如果是整数，则表示方法和原码一样；如果是负数，则将数字的反码加上1。
- 补码的设计目的
  - 使符号位能与有效值部分一起参加运算，从而简化运算规则
  - 使减法运算转换为加法运算，进一步简化计算机中运算器的线路设计

字符数据

- ASCII码 （美国信息标准交换码）

  字符数据是以ASCII码形式存放在内存中的。

  例如 “1” 就是 31H

  “A” 就是 41H
- BCD码：利用二进制形式来表示十进制数。

  压缩BCD（组合BCD） 一个字节存放两个十进制数位

  非压缩BCD（非组合BCD） 一个字节存放一个十进制数位

数值的表示范围

表 1.3 无符号数的表示范围

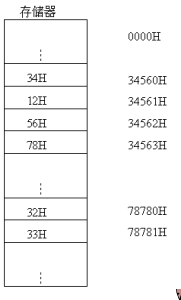
n	十进制数形式		十六进制数形式	
	最大值	最小值	最大值	最小值
8 位	255	0	0FFH	0
16 位	65535	0	0FFFFH	0
32 位	4294967295	0	0FFFFFFFFH	0

表 1.2 有符号数的表示范围

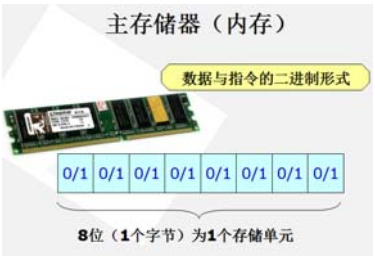
n	十进制数形式		二进制数形式		补码形式	
	最大值	最小值	最大值	最小值	最大值	最小值
8 位	+127	−128	2^7-1	-2^7	7FH	80H
16 位	+32767	−32768	$2^{15}-1$	-2^{15}	7FFFH	8000H
32 位	+2147483647	−2147483648	$2^{31}-1$	-2^{31}	7FFFFFFFH	80000000H

4、存储器单元的地址和内容

- 存储单元地址 ：8086系统中，为了标识和存取每一个存储单元，给每个存储单元规定一个编号，这就是存储单元地址。
  - [x]：地址
- 存储单元的内容 ：一个存储单元中存放的信息称为该存储单元的内容。
  - (x) ：内容



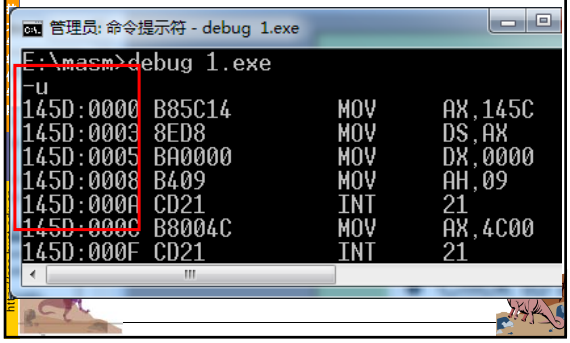
物理地址的形成

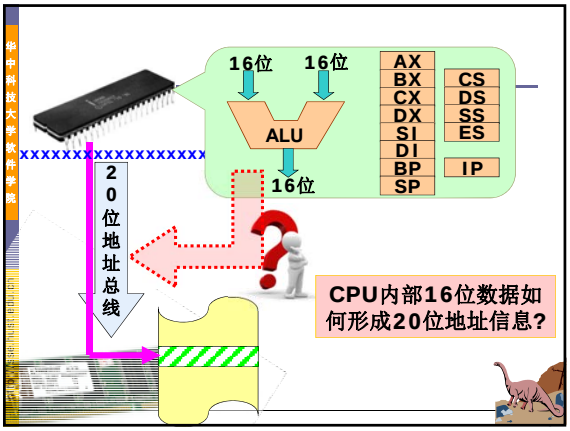


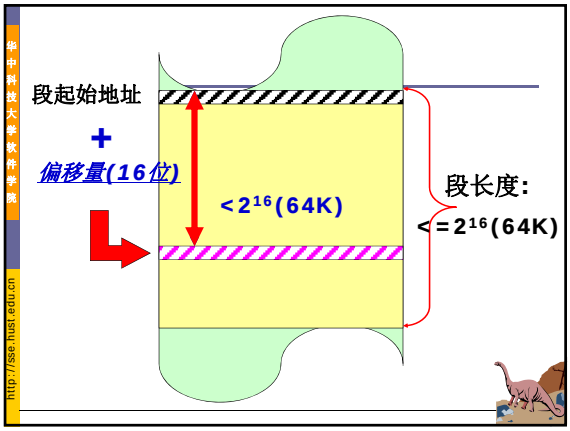
地址	内容
00000H	1 1 0 0 1 1 1 1
00001H	1 0 0 1 1 1 0 1
00002H	1 1 1 0 0 1 1 1
.....	.....
10002H	1 1 0 1 1 0 1 1
10003H	0 1 0 0 0 0 0 1
10004H	1 1 1 1 1 0 0 1
10005H	1 1 0 0 0 1 0 1
.....	.....
FFFFFH	0 0 0 0 1 1 0 1

物理地址：存储单元在存储器中的实际地址

逻辑地址





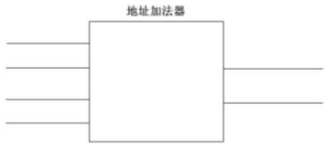


8086CPU给出物理地址的方法

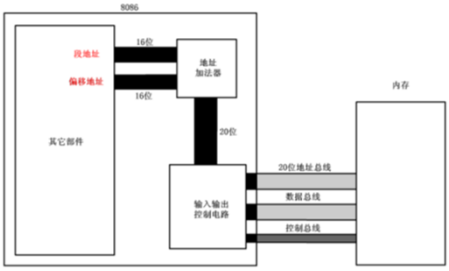
- 用逻辑地址表示物理地址
- 逻辑地址的表示形式
  - 段地（首）址（16位）：偏移地址（16位）
- 物理地址和逻辑地址的关系

“段首地址×16+偏移地址=物理地址”

“段首地址×16+偏移地址=物理地址”



地址加法器的工作过程



8086CPU给出物理地址的方法

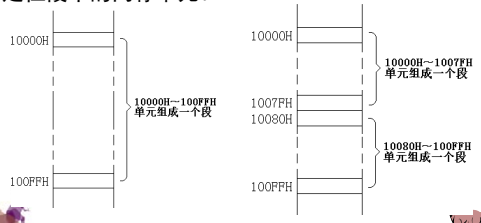
- （1）观察下面的地址？

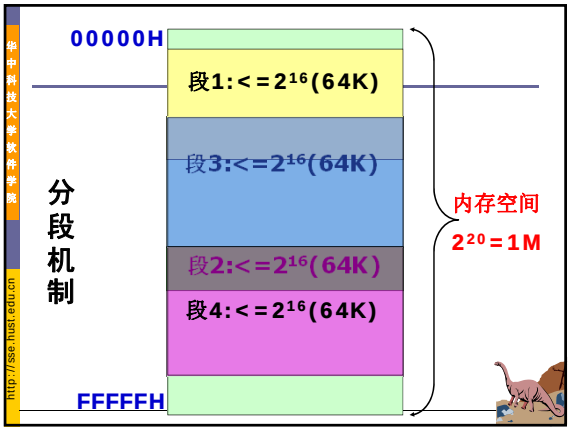
物理地址	段地址	偏移地址
21F60H	2000H	1F60H
	2100H	0F60H
	21F0H	0060H
	21F6H	0000H
	1F00H	2F60H

- 用不同的段地址和偏移地址形成同一个物理地址。

4.1 段的概念

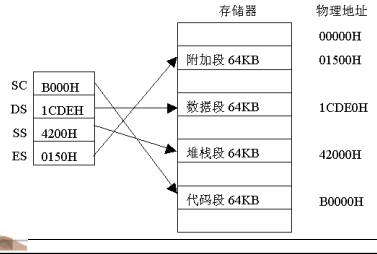
- 将若干地址连续的内存单元看作一个段，用段地址×16定位段的起始地址（段基地址），用偏移地址定位段中的内存单元。





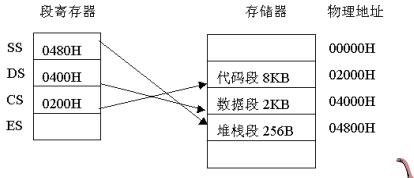
例1：各段独立分配

- 设CS=B000H、DS=1CDEH、SS=4200H、ES=0150H，它们分别为代码段、数据段、堆栈段和附加段的段首址。自每个段首址开始，各段均占64KB的范围，各段之间互不重叠。



例2：各段重叠分配

- 设CS=0200H、DS=0400H、SS=0480H，这样代码段、数据段和堆栈段的物理首地址分别为02000H、04000H和04800H。其中代码段占8KB地址空间，数据段占2KB，堆栈段占256B，SP=0100H。

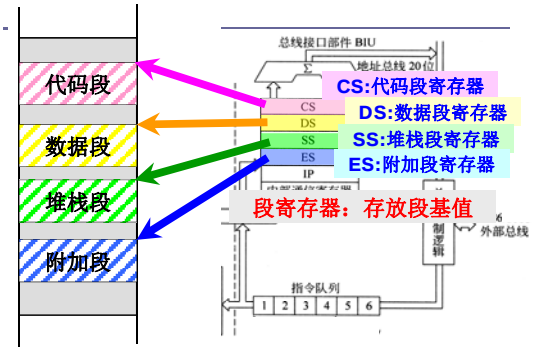


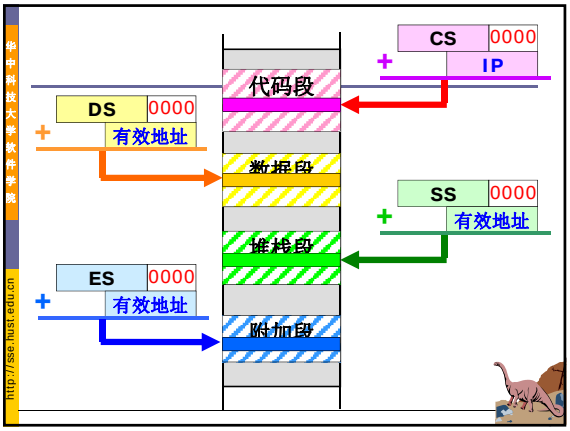
各段重叠存储单元分配图

需要注意

- （1）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；
- （2）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。
- （3）CPU可以用不同的段地址和偏移地址形成同一个物理地址。
- （4）按照这两个条件，8086CPU的1M字节地址空间最多可划分成64K个逻辑段，最少也要划分成16个逻辑段。逻辑段与逻辑段可以相连，也可以不连，还可以重叠。

4.2 段寄存器



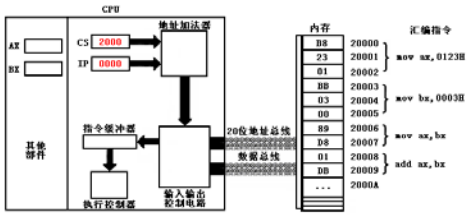


4.3 CS和IP

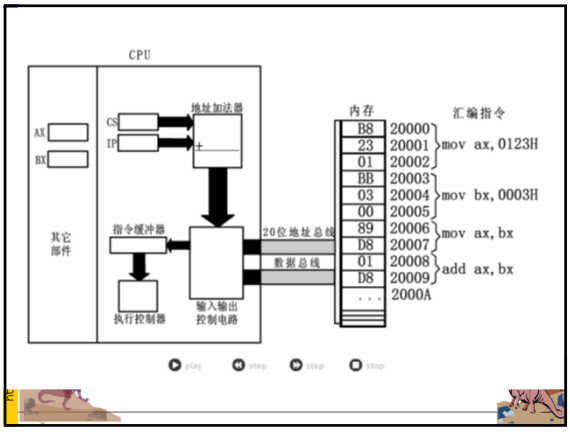
- 内存中指令和数据没有任何区别，都是二进制信息，CPU在工作的时候把有的信息看作指令，有的信息看作数据。
- CPU根据什么将内存中的信息看作指令？
- CS和IP是8086CPU中最关键的寄存器，它们指示了CPU当前要读取指令的地址。

CS为代码段寄存器；
IP为指令指针寄存器。

8086PC读取和执行指令相关部件



- 8086PC读取和执行指令演示
- 8086PC工作过程的简要描述



8086PC启动时的CS和IP

- 在 8086CPU 加电启动或复位后（ 即 CPU 刚开始工作时）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机开机后执行的第一条指令。

修改CS、IP的指令

- CPU从何处执行指令是由CS、IP中的内容决定的，程序员可以通过改变CS、IP中的内容来控制CPU执行目标指令。
- 如何改变CS、IP的值？

修改CS、IP的指令

- 同时修改CS、IP的内容：
  jmp 段地址：偏移地址
  jmp 2AE3:3
  jmp 3:0B16
  功能：用指令中给出的段地址修改CS，偏移地址修改IP。
- 仅修改IP的内容：
  jmp 某一合法寄存器
  jmp ax （类似于 mov IP,ax）
  jmp bx
  功能：用寄存器中的值修改IP。

如何修改CS、IP？

- 仅修改IP的内容：
  jmp 某一合法寄存器
  jmp ax （类似于 mov IP,ax）
  jmp bx
  功能：用寄存器中的值修改IP。

问题分析

- 内存中存放的机器码和对应汇编指令情况：
  （初始：CS=2000H，IP=0000H）

地址	内存中的机器码	对应的汇编指令
10000H	DB	mov ax,0123H
10001H	23	
10002H	01	
10003H	B8	mov ax,0000
10004H	00	
10005H	00	
10006H	8B	mov bx,ax
10007H	D8	
10008H	FF	jmp bx
10009H	E3	

地址	内存中的机器码	对应的汇编指令
20000H	B8	mov ax,6622H
20001H	22	
20002H	66	
20003H	EA	jmp 1000:3
20004H	03	
20005H	00	
20006H	00	
20007H	10	
20008H	89	mov cx,ax
20009H	C1	

问题分析结果：

最终指令的执行顺序

- （1）mov ax, 6622
- （2）jmp 1000:3
- （3）mov ax, 0000
- （4）mov bx, ax
- （5）jmp bx
- （6）mov ax, 0123H
- （7）转到第（3）步执行

5 代码段

- 如何使得代码段中的指令被执行？

```
mov ax,0000     (B8 00 00)
add  ax,0123H   (05 23 01)
mov bx,ax       (8B D8)
jmp  bx         (FF E3)
```

- 长度为 10 字节的字节的指令，保存在123B0H~123B9H的内存单元中
- 将CS:IP指向所定义的代码段中的第一条指令的首地址。

  CS = 123BH，IP = 0000H

```
.386
DATA    SEGMENT  USE16
BUF1    DB  20, 39, 50, -20, 0, -12
BUF2    DB  6 DUP(0)
DATA    ENDS

STACK   SEGMENT USE16
        DB 200  DUP(0)
STACK ENDS

CODE    SEGMENT USE16
ASSUME  CS:CODE, SS:STACK, DS:DATA

START:  MOV  AX, DATA
        MOV  DS, AX
        ……
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
        END  START
```

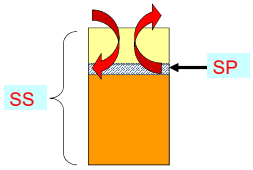
5、数据段

- 可以根据需要，将一组长度为N（N<=64k）、地址连续、起始地址为16倍数的内存单元当做专门存储数据的空间，称为数据段

```
.386
DATA    SEGMENT  USE16
BUF1    DB  20, 39, 50, -20, 0, -12
BUF2    DB  6 DUP(0)
DATA    ENDS

STACK   SEGMENT USE16
        DB 200  DUP(0)
STACK ENDS

CODE    SEGMENT USE16
ASSUME  CS:CODE, SS:STACK, DS:DATA

START:  MOV  AX, DATA
        MOV  DS, AX
        ……
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
        END  START
```

6、堆栈

- 特殊存储区域：堆栈段
- “先进后出”
- 一端固定（栈底）：SS
- 一端活动（栈顶）：SP

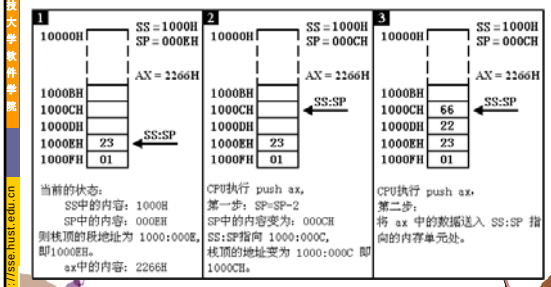



```
.386
DATA    SEGMENT  USE16
BUF1    DB  20, 39, 50, -20, 0, -12
BUF2    DB  6 DUP(0)
DATA    ENDS

STACK   SEGMENT USE16
        DB 200  DUP(0)
STACK ENDS

CODE    SEGMENT USE16
ASSUME  CS:CODE, SS:STACK, DS:DATA

START:  MOV  AX, DATA
        MOV  DS, AX
        ……
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
        END  START
```

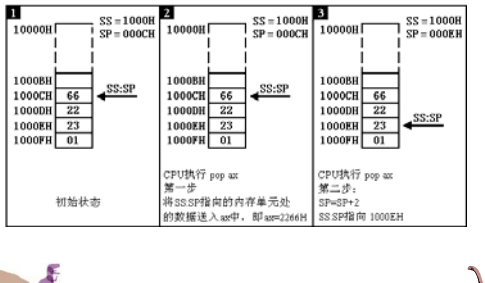
堆栈的操作

- 入栈PUSH
  - 1）SP=SP-2；
  - 2）向SS:SP指向的字单元中送入数据。
- 出栈POP
  - 1）从SS:SP指向的字单元中读取数据；
  - 2）SP=SP+2。
- 入栈和出栈都是以字为单位进行，用两个字节单元存放，高地址单元放高 8 位，低地址单元放低8 位。

Push 指令的执行过程

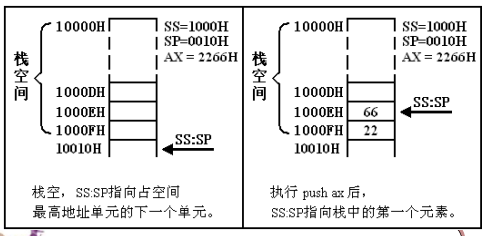


pop 指令的执行过程



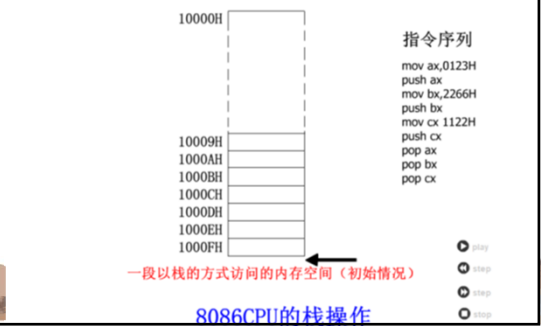
栈空的状态

- SP = 0010H



入栈和出栈的顺序

- 用栈来暂存以后需要恢复的寄存器的内容时 ，寄存器出栈的



栈顶超界的问题

- 当 栈满的时候再使用push指令入栈，（演示）
  栈空的时候再使用pop指令出栈，（演示）
  都将发生栈顶超界。
- 8086CPU不保证对栈的操作不会超界，需要我们自己管理。

段的使用：

- 必须将段首地址置入DS或SS中，而CS由系统自动置入。
- 每段的大小 ≤ 64KB，每个段在主存中的具体位置由程序或操作系统确定；

```
386
DATA    SEGMENT  USE16
BUF1    DB  20, 39, 50, -20, 0, -12
BUF2    DB  6 DUP(0)
DATA    ENDS

STACK   SEGMENT  USE16
        DB 200  DUP(0)
STACK ENDS

CODE    SEGMENT USE16
ASSUME  CS:CODE, SS:STACK, DS:DATA

START:  MOV  AX, DATA
        MOV  DS, AX
        ……
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
        END  START
```

7 汇编语言源程序的一般结构

- 分段式结构
  - 数据段
  - 堆栈段
  - 代码段
  - 附加段

段

```
;  ------------------
段名1    SEGMENT     ;  一个段的开始
         语句1
         语句2
         ……
段名1    ENDS        ;  一个段的结束
段名2    SEGMENT     ;  另一个段的开始
         ……
段名2    ENDS        ;  另一个段的结束
         ……          ;  其它段
         END         ;  源代码结束
```

源程序的基本框架

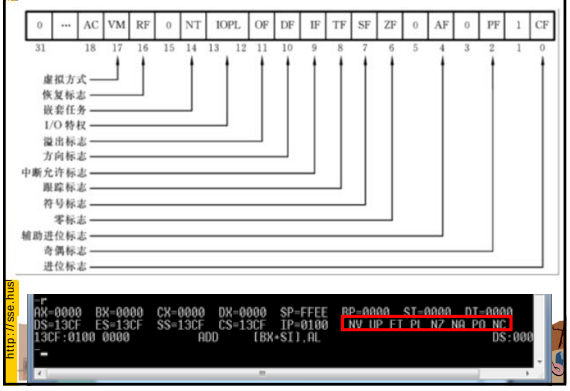
```
DATA    SEGMNET                    ┐
；根据需要定义一些变量、数据或单元   │ 数据段
DATA    ENDS                       ┘
EDATA   SEGMNET                    ┐
；根据需要定义一些变量、数据或单元   │ 附加段
EDATA   ENDS                       ┘
STACK   SEGMENT                    ┐
  ；根据需要定义堆栈大小           │ 堆栈段
STACK   ENDS                       ┘
```

```
CODE    SEGMNET                                  ┐
        ASSUME CS：CODE，DS：DATA，…             │
MAIN    PROC FAR                                 │
BEGIN:  MOV  AX，STACK                           │
        MOV  SS，AX                              │ 代码段
        MOV  SP，TOP ;TOP为栈顶的EA  } ；设置自己的堆栈。
        PUSH DS                                  │
        MOV  AX，0                               │
        PUSH AX                                  │
        MOV  AX，DATA                            │
        MOV  DS，AX                  } 对段寄存器进行赋值。
        …                                        │
        …；实现具体功能的语句，随程序各异        │
        RET                                      │
MAIN    ENDP                                     │
CODE    ENDS                                     │
        END  BEGIN                               ┘
```

专用寄存器SP、BP、SI、DI

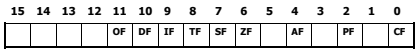
- 只能以字（16位）的形式使用，用于段内寻址时提供偏移地址。
- 堆栈指针寄存器SP：存放堆栈栈顶偏移地址，和堆栈段寄存器SS一起确定堆栈段栈顶物理地址。
- 基址指针寄存器BP：
  - 存放堆栈段中某一单元的偏移地址，堆栈段寄存器SS存放堆栈段段首址。
  - 在间接寻址中用作基址寄存器。
- 源变址寄存器SI和目的变址寄存器DI
  - 与数据段寄存器DS连用，确定数据段中某一存储单元的地址。
  - 在串操作指令中，SI与DS连用，DI与ES连用
  - 也可当数据寄存器

标志寄存器



标志寄存器

- 状态标志：表示前面的操作执行后，算术逻辑部件处于怎样一种状态。
  - 是否产生了进位，是否发生了溢出等等
  - 程序中，可以通过对某个状态标志的决定后面的走向及操作。



- 控制标志(3位)：
  - 每一位控制标志都对一种特定的功能起控制作用。可以通过专门的指令对其进行“置位”（Set）或“复位”（Reset）。

80386 CPU的寄存器

- 对于80386以后的80X86CPU的寄存器是32位的，在8086CPU的通用寄存器的左边（高位）扩展了EAX、EBX、ECX、EDX、ESP、EBP、EDI、ESI等寄存器，使它们分别能保存8位，16位和32位的数据。
- 从80386起，在总线接口部件中增加了两个附加的数据段寄存器FS和GS。