第一章　预备知识

第一章　预备知识

一、本章的学习内容：

本章学习汇编语言的预备知识：

1. 什么是汇编语言？
2. Intel 80X86微处理器中的寄存器组
3. 主存储器的编址方式及物理地址的形成方式
4. 数和符号在计算机中的表示方法
5. 标志寄存器
6. 汇编源程序的基本结构

第一章　预备知识

二、本章的学习重点：

1. 汇编语言、汇编程序的概念
2. 80X86寄存器组
3. 堆栈的概念、进栈、出栈指令
4. 实模式和保护模式下物理地址的形成
5. 整数和字符串的表示方法
6. CF、OF、ZF、SF标志位
7. 汇编源程序举例

第一章　预备知识

三、本章学习的难点：

1. 寄存器组各个寄存器的名称和用途
2. PUSH/POP指令、堆栈指示器的变化
3. 保护模式下物理地址的形成
4. 有符号数、无符号数及其运算对于标志寄存器的影响

1.1 机器语言与汇编语言

1. 机器语言

机器语言的特点:

(1) 由0和1组成的二进制码。

(2) 能为计算机识别并执行。

(3) 依赖于某一类型的机器。 不同类型的CPU都有自己特有的、一定数量的基本指令。

1.1 机器语言与汇编语言

机器指令(硬指令):指挥计算机完成某一基本操作的命令,是面向机器的。

操作种类　　操作数存放位置

例:

1011 1011　　BB　　←——→ 操作码

0011 0100　　34　　←——→ 操作数

0001 0010　　12

操作: 1234H → BX

1.1 机器语言与汇编语言

指令系统：每台计算机都规定了自己所特有的、一定数量的基本命令，这批指令的全体为计算机的指令系统

机器语言：机器指令的集合

机器语言程序：用机器语言编写的程序

1.1 机器语言与汇编语言

2. 汇编语言(改进的方案)

操作码	地址码

助记符　　变量或标号

汇编语言：一种用符号书写的、其主要操作与机器指令一一对应，并遵循一定语法规则的计算机语言。

1.1 机器语言与汇编语言

例2：刚才的例子若用汇编语言来书写：

操作码		地址码
BB		34 12
MOV	BX,	1234H

1.1 机器语言与汇编语言

汇编源程序

用汇编语言编写的程序——类似于高级语言编写的源程序

汇编程序

把汇编源程序翻译成目标程序的语言加工程序——类似于高级语言的编译程序

汇编

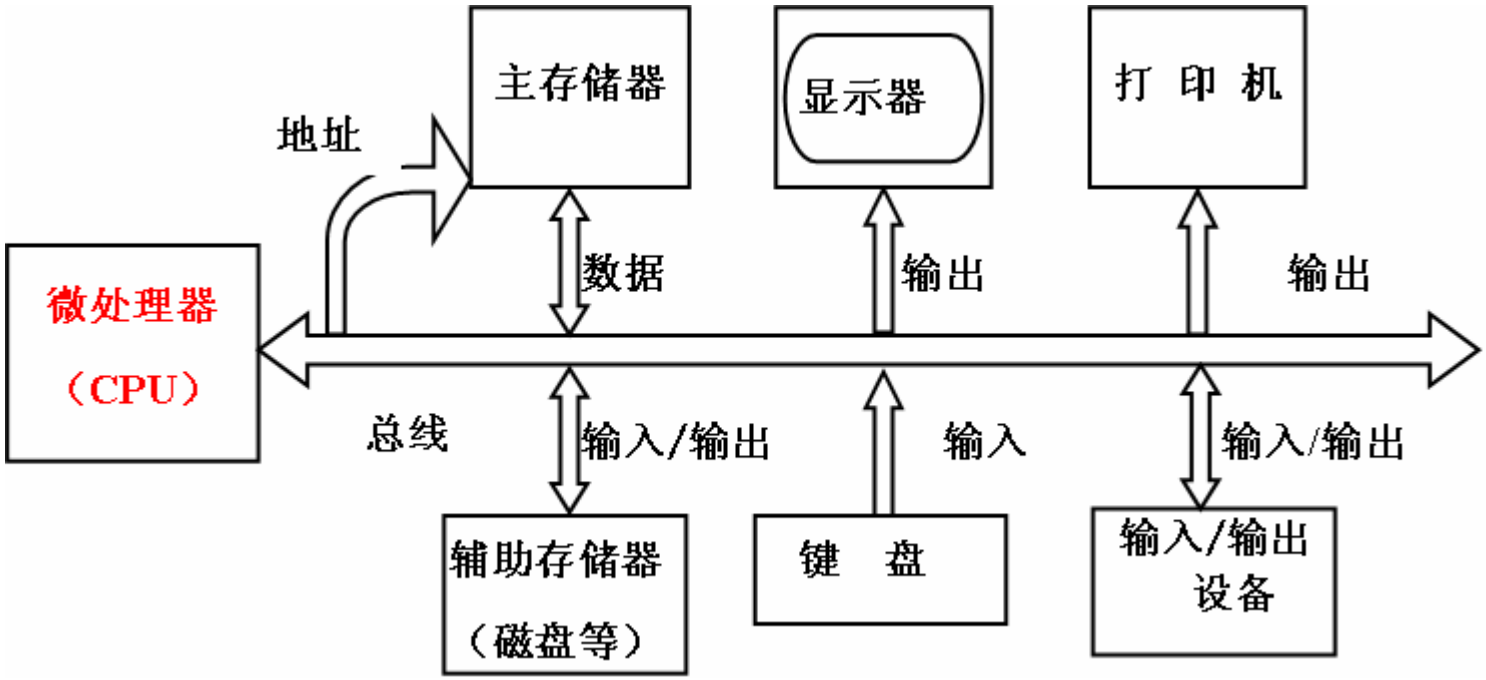
汇编程序进行翻译的过程 —— 类似于高级语言的编译过程

1.1 机器语言与汇编语言

机器语言、汇编语言和高级语言的比较

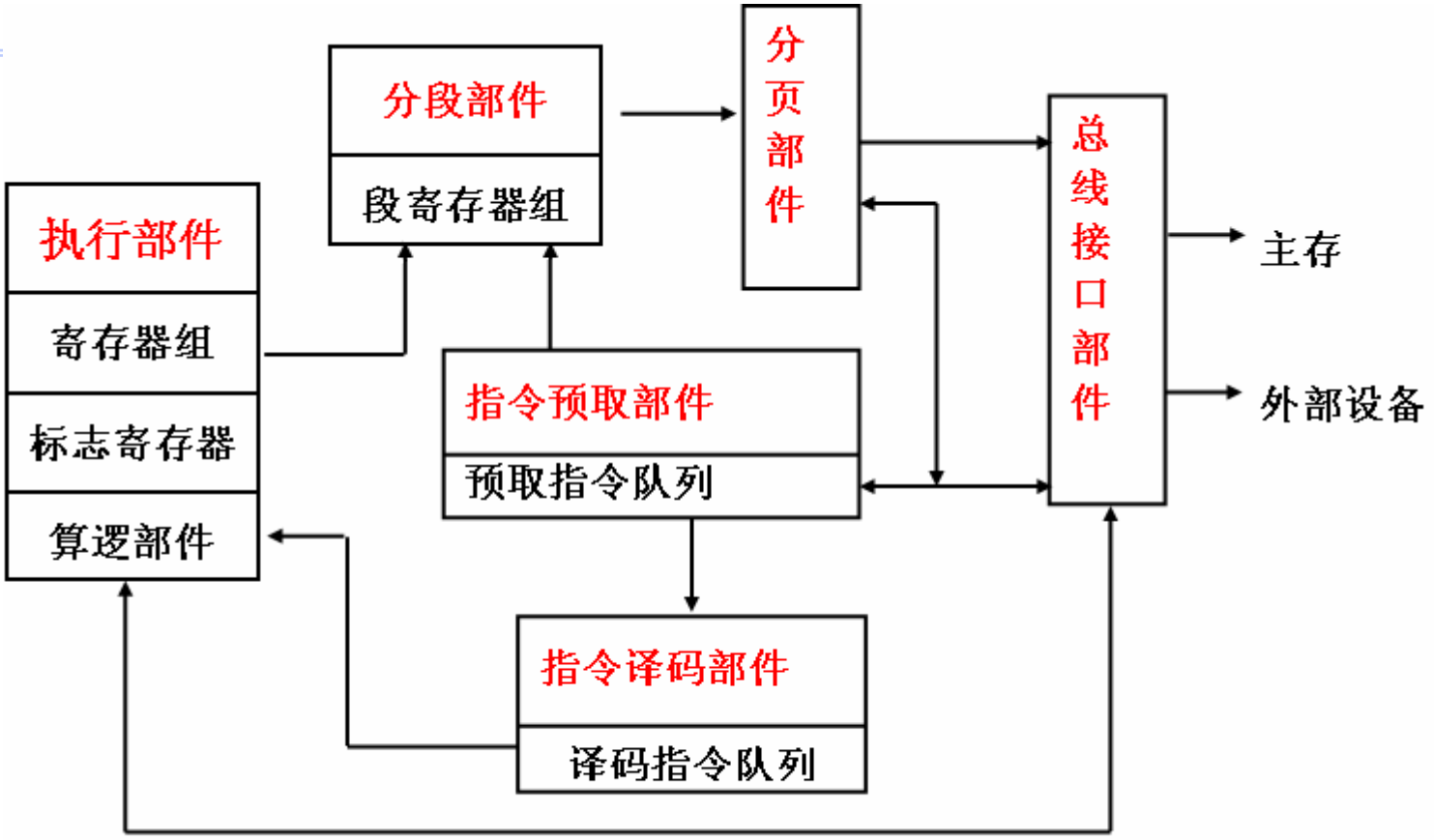
	机器语言	汇编语言	高级语言
计算机能否直接识别	能	不能	不能
易用性	差	中	好
占据空间	小	小	大
执行速度	快	快	慢
用途	特殊 加密／解密	系统核心 要求速度快，代码短的程序 直接操纵I/O 信息安全	一般性软件开发

1.2 Intel系列机简介



微型计算机的构成

1.2.2 80x86微处理器结构



80x86微处理器结构

1.2.2 80x86微处理器结构

1. 总线接口部件：是CPU与整个计算机系统之间的高速接口

   功能：接受所有的总线操作请求，并按优先权进行选择，最大限度地利用本身的资源为这些请求服务。

2. 执行部件：寄存器组、标志寄存器、算逻部件、控制部件等组成

   功能：从译码指令队列中取出指令并且执行

1.2.2 80x86微处理器结构

(1) 数据寄存器组(EAX，EBX，ECX，EDX)

EAX (累加器)Accumulator

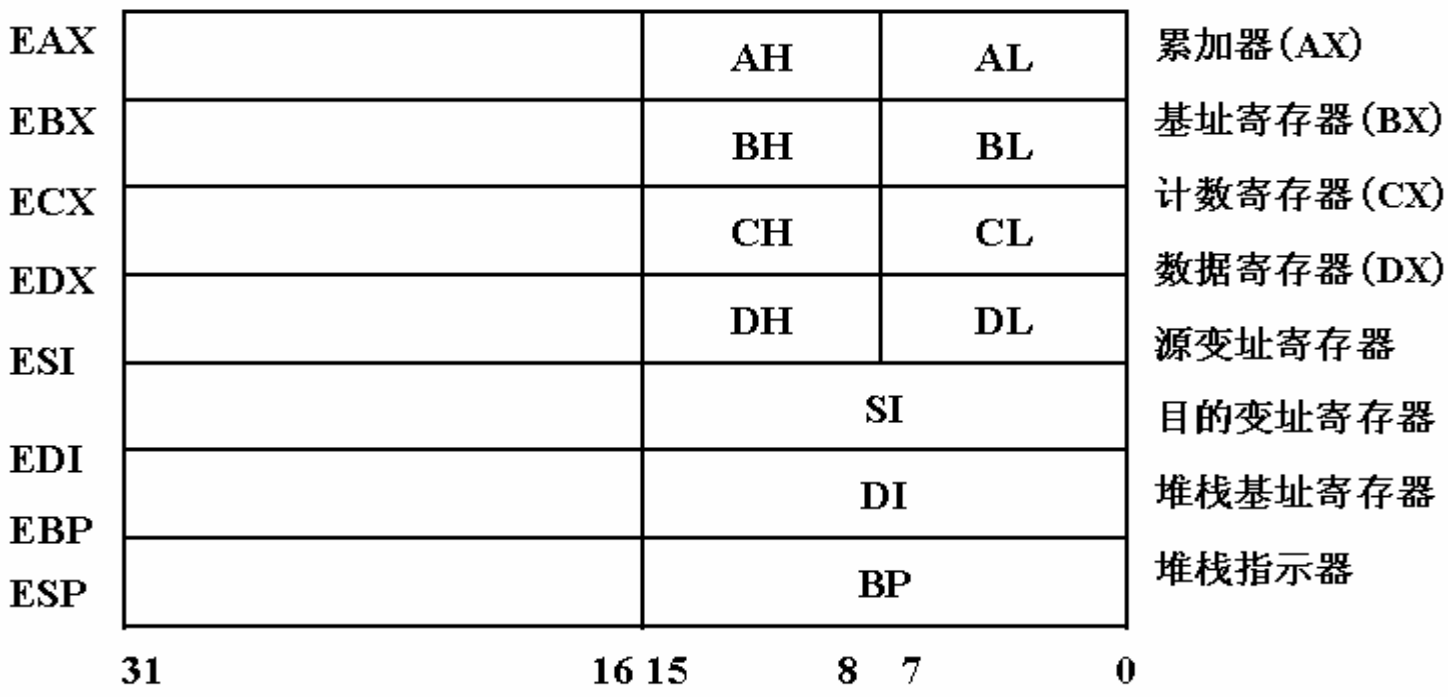
EBX (基址寄存器)Base

ECX (计数寄存器)Count

EDX (数据寄存器)Data

作用：用来保存操作数、运算结果或作指示器、变址寄存器，减少存取操作数所需要的访问总线和主存储器的时间，加快运行速度。

1.2.2 80x86微处理器结构



注意：

它们既可作32位、16位寄存器，也可作8位寄存器使用。

16位和8位的寄存器不能作指示器或变址寄存器。

1.2.2 80x86微处理器结构

(2) 指示器变址寄存器组(ESI、EDI、ESP、EBP)

作用：一般用来存放操作数的偏移地址，用作指示器或变址寄存器。

ESP（Stack Pointer），称为堆栈指示器，存放的是当前堆栈段中栈顶的偏移地址;

EBP(Base Pointer)，为对堆栈操作的基址寄存器;

ESI(Source Index)，称为源变址寄址器；字符串指令源操作数的指示器

EDI(Destination Index)，称为目的变址寄存器;字符串指令目的操作数的指示器

1.2.2 80x86微处理器结构

3. 指令预取部件和指令译码部件

指令预取部件: 通过总线接口部件，把将要执行的指令从主存中取出，送入指令排队机构中排队。

指令译码部件: 从指令预取部件中读出指令并译码，再送入译码指令队列排队供执行部件使用。

指令指示器: 它总是保存着下一条将要被CPU执行的指令的偏移地址(简称EA)，其值为该指令到所在段首址的字节距离。

1.2.2 80x86微处理器结构

4. 分段部件和分页部件

使用分段部件和分页部件实现虚拟存储空间映射到物理存储空间

程序员使用二维地址 段地址：段内偏移地址

分段部件

段地址：段内偏移地址 → 一维的线性的地址

分页部件

虚拟的线性的地址 → 主存储器的物理地址

1.2.2 80x86微处理器结构

段寄存器：保存段首地址

- 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
- 附加段寄存器 FS
- 附加段寄存器 GS

1.2.2 80x86微处理器结构

指令执行过程:

(1) 指令预取部件和指令译码部件

EIP → 指令的偏移地址

EIP增量，形成下一条指令的地址

(2) 分段部件和分页部件

CS : EIP → 指令的物理地址

(3) 总线接口部件

从主存中取指令 → 预取指令队列

1.2.2 80x86微处理器结构

(4) CPU按序从预取指令队列中取出指令 → 指令译码部件。

(5) 指令译码部件译码 → 执行部件执行指令;

(6) 执行过程中若需要取主存操作数 → 操作数偏移地址

(7) 分段部件和分页部件

偏移地址 , 段寄存器 → 操作数的物理地址

(8) 总线接口部件

从主存中取数据 → 执行部件

1.2.2 80x86的三种工作方式

1. 实方式（实际地址）

   操作相当于一个可进行32位快速运算的8086（内部32位、外部总线16位数据、20位地址）

2. 保护方式（虚地址）

   是80386设计目标全部达到的工作方式，通过对程序使用的存储区采用分段、分页的存储管理机制，达到分级使用互不干扰的保护目的。能为每个任务提供一台虚拟处理器，使每个任务单独执行，快速切换。

3. 虚拟8086方式

   保护方式下所提供的同时模拟多个8086处理器。

1.3 主存储器和物理地址的形成

存储器：用来存放程序和数据的装置，包括主存和外存。

主存储器：主机内部的半导体存储器，相对外存，容量小，速度快，成本高。

1.3.1 主存储器

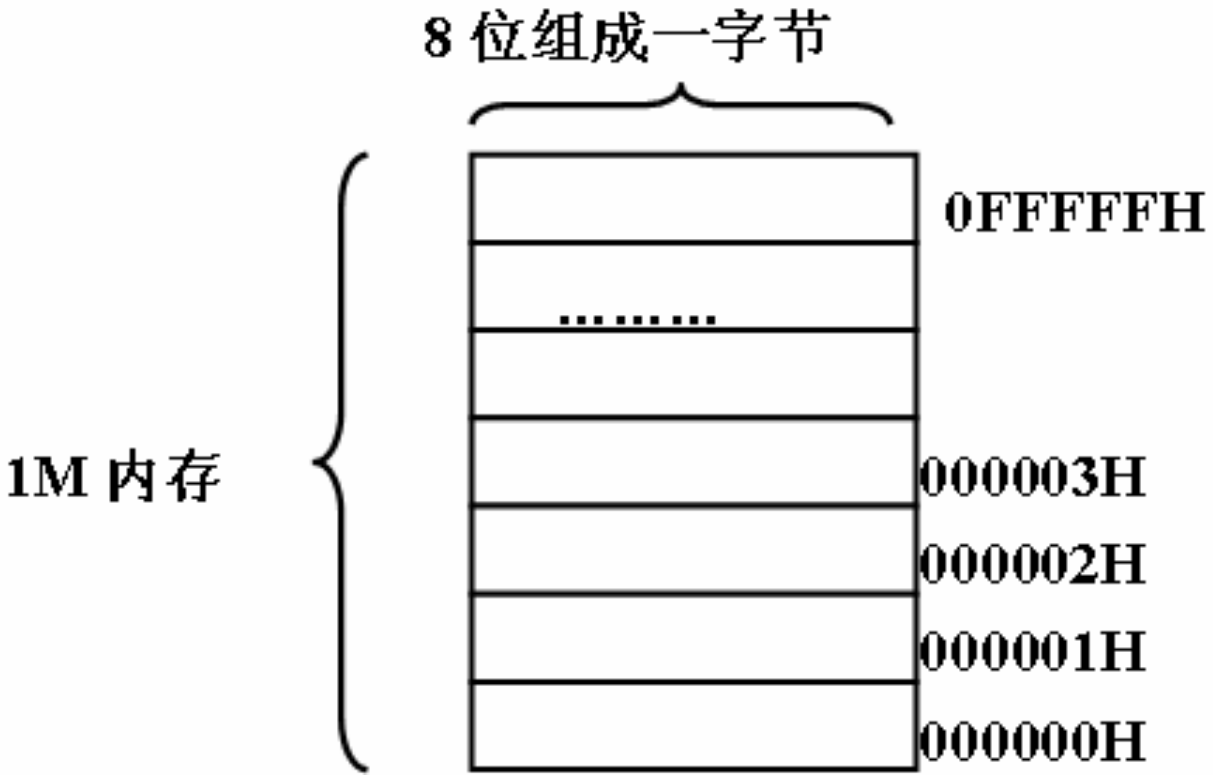
主存储器编址方式

主存储器的基本存贮单位是位，它能容纳一个二进制数字0或1。

8位组成一个字节，一个字节是一个存储单元。

为了区别不同的单元，每一字节都被指定一个编号，即为此单位的物理地址。

1.3.1 主存储器



80X86机是按8位字节编址的，即以字节作最小寻址单位。

1.3.1 主存储器

字：由2个字节组成，字地址由两字节中地址较小的一个即低字节的地址决定。存放的方式是低8位存放低字节，高8位存放高字节。

双字：双字的地址也由四个字节中的最低地址确定

<table>
<tr><td colspan="10">双字(32位)</td></tr>
<tr><td colspan="2">字(16位)</td><td colspan="8">字(16位)</td></tr>
<tr><td>字节(8位)</td><td>字节(8位)</td><td colspan="2">字节(8位)</td><td colspan="6">字节(8位)</td></tr>
<tr><td></td><td></td><td colspan="2"></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

1.3.1 主存储器

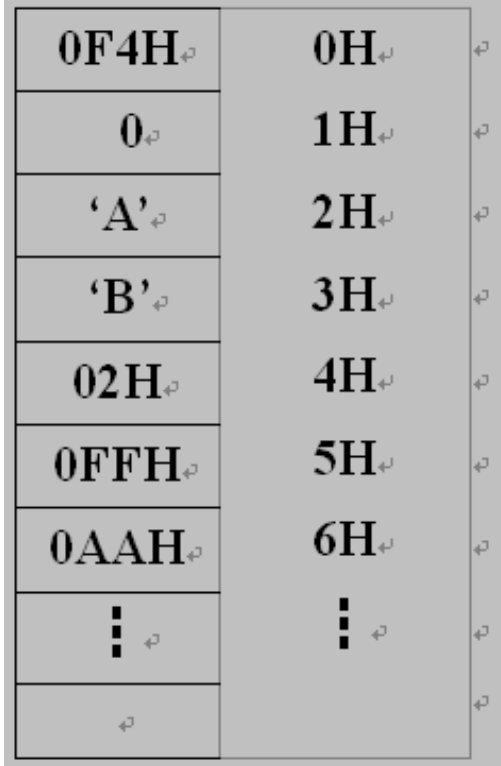
问题：一个字数据该怎样存入计算机主存呢？双字数据呢？

高-高 低-低

字：要占有连续的两个字节．16位中，低8位存放在低地址字节，高8位存放在相邻的高地址字节中。

双字：32位中，低16位存放在低地址字，高16位存放在相邻的高地址字地址中。

1.3.1 主存储器



0F4H	0H
0	1H
‘A’	2H
‘B’	3H
02H	4H
0FFH	5H
0AAH	6H
⋮	⋮

0 号字内容为：00F4H

2 号字节内容为：字符 A 的 ASCII 码 41H

2 号字的内容为：字符 A 和 B 的 ASCII 码 4241H

2 号双字的内容为：0FF024241H

5 号字节内容为：0FFH

4 号字的内容为：0FF02H

5 号字的内容为：0AAFFH

由此可见，字数据在主存中存放的形式是低 8 位存放在低字节地址，高 8 位存放在相邻的高字节地址中。

1.3.2 堆栈

从逻辑上来看：是内存中开辟的一片存贮区，这片存贮区采用的存贮方式是一端固定，一端活动，即只允许在一端插入或删除(访问可任意)。

堆栈中数据的存取原则：“先进后出”，

堆栈中的数据也称元素或栈项。

元素进栈称压入，出栈称弹出。

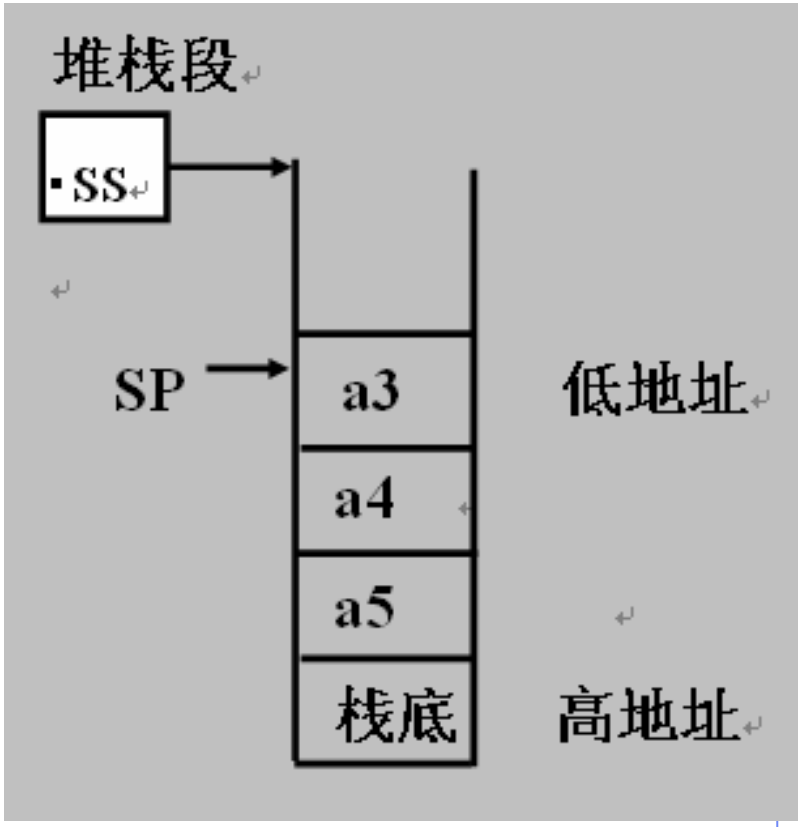
1.3.2 堆栈

从硬件的观点看堆栈必须由一片存贮单元和一个指示器组成。

固定端叫栈底。

栈指针用来指示栈顶进栈和出栈时偏移地址的变化

指针所指示的最后存入信息的单元叫栈顶，所有信息的存取都在栈顶进行，栈指针总是指向栈顶的。



1.3.2 堆栈

问题：为什么要用堆栈？

程序中经常用到子程序或处理中断，此时，主程序需要把子程序或中断程序将用到的寄存器内容保护起来，以便子程序或中断返回后，主程序能够从调用点或中断点处继续执行。

此外，堆栈还经常用于：

保护和恢复现场

主程序和子程序之间传递参数

子程序中的局部变量

1.3.2 堆栈

1. 进栈指令PUSH

格式：PUSH　OPS

功能：将立即数、寄存器、段寄存器或存贮器中的一个字/双字数据压入堆栈。

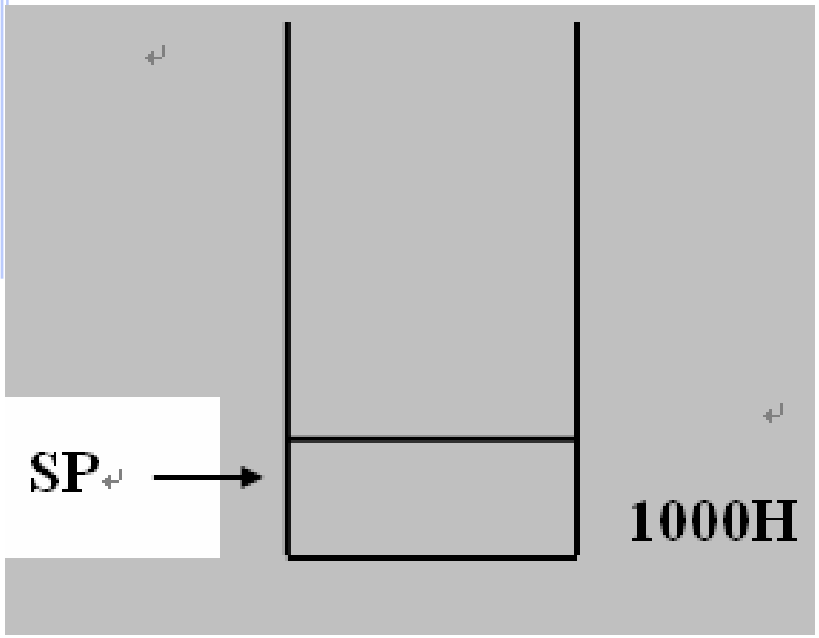
例：PUSH　AX

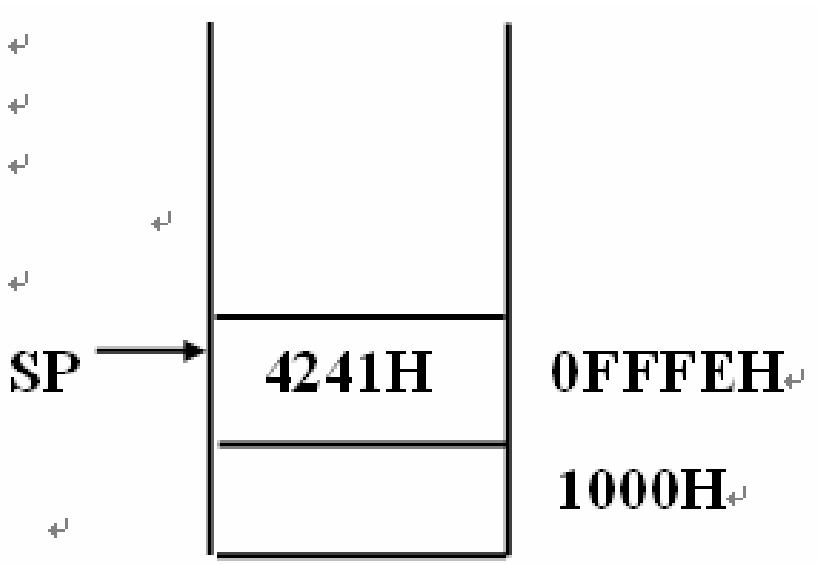
设：(AX)=4241H　(SP)=1000H

执行：① (SP)-2→SP

② (AX)→(SP)

1.3.2 堆栈





1.3.2 堆栈

出栈指令POP

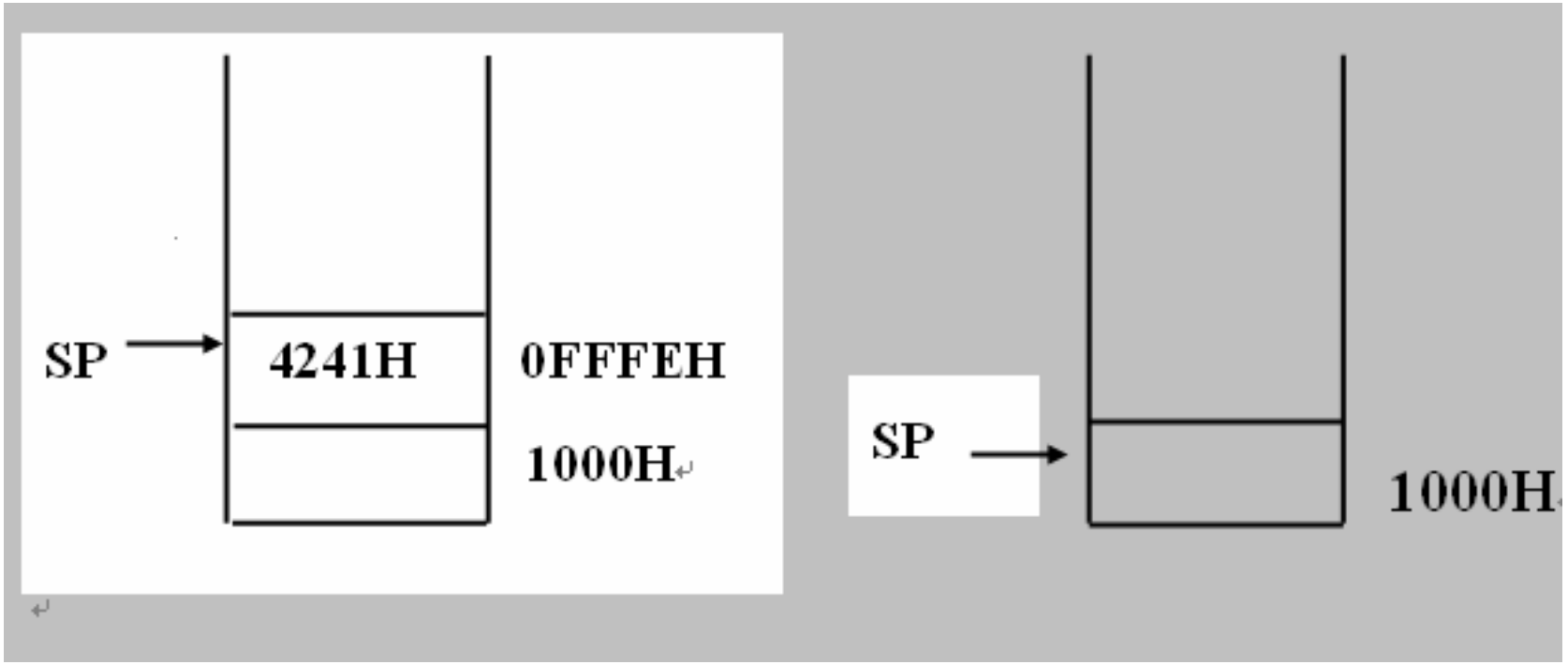
格式：POP　OPD

功能：将栈顶元素弹出送至某一寄存器、段寄存器(除CS外)或字/双字存贮单元中。

例：POP　BX

设指令执行前：(BX)=1111H，堆栈内容如上题所示。

1.3.2 堆栈



1.3.3 存储器物理地址的形成

问题的由来

8086的限制，20位地址总线，16位寄存器(SP，BP，SI，DI)。

问题：如何通过16位寄存器访问1MB的内存

解决：将1M字节主存分段，每段最多64K字节

1.3.3 存储器物理地址的形成

分段管理的方法：

段寄存器保存起始首地址 + 段内偏移地址的总体策略。

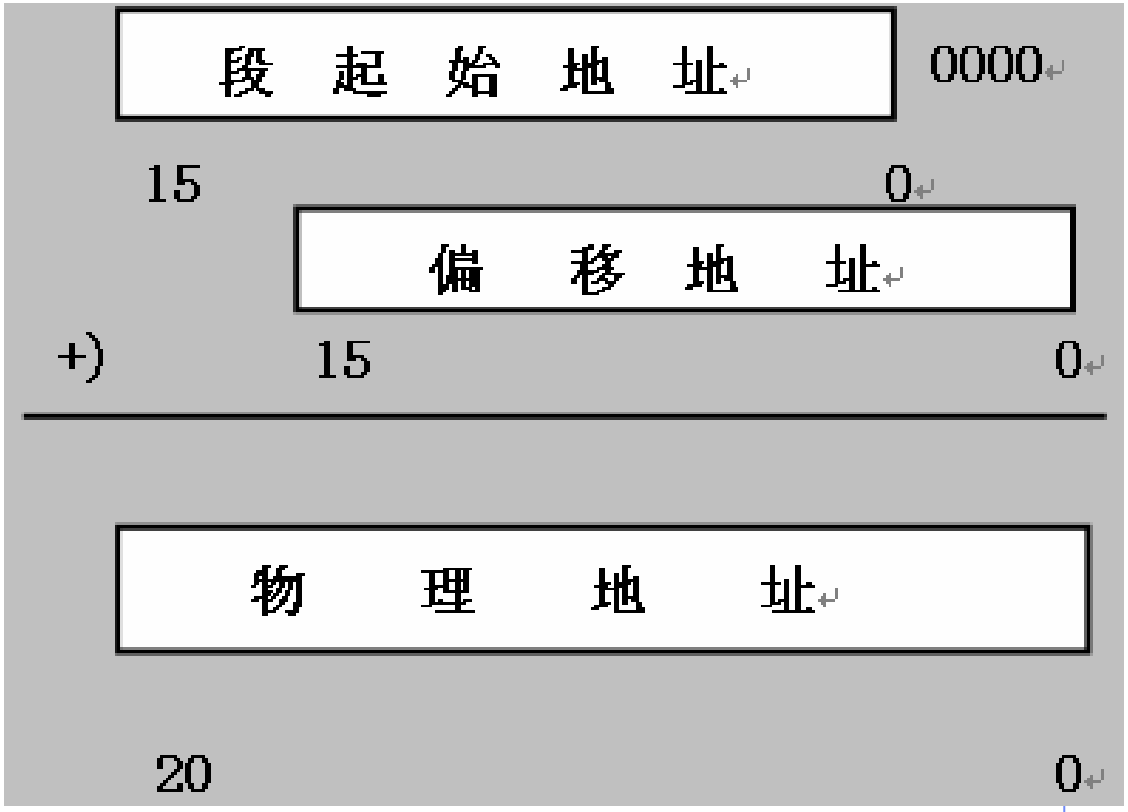
段首地址分别存放在：段寄存器CS、DS、ES、SS、FS、GS中

段内偏移地址：段内相对于段起始地址的偏移值，往往由SP、BP、SI、DI、IP给出。

1.3.3 存储器物理地址的形成

1. 实模式

PA = 段寄存器保存的16位二进制数后补4个0 + EA



1.3.3 存储器物理地址的形成

分段的实际作用：

(1) 实现了16位表示20位的地址；

(2) 当程序和数据的大小<64KB时，编制的程序可只关心EA，而不用管它的起始地址在哪（便于程序在主存中任何位置运行）

(3) 便于不同目的的程序或数据分开存放，使程序各部分的含义更加明确。

1.3.3 存储器物理地址的形成

2. 保护模式

保护什么：

分清不同程序使用的存储区域，不允许随便使用别人的数据和代码。

必要条件：

（1）要标记每段存储区的所有者或被使用的权限级别。

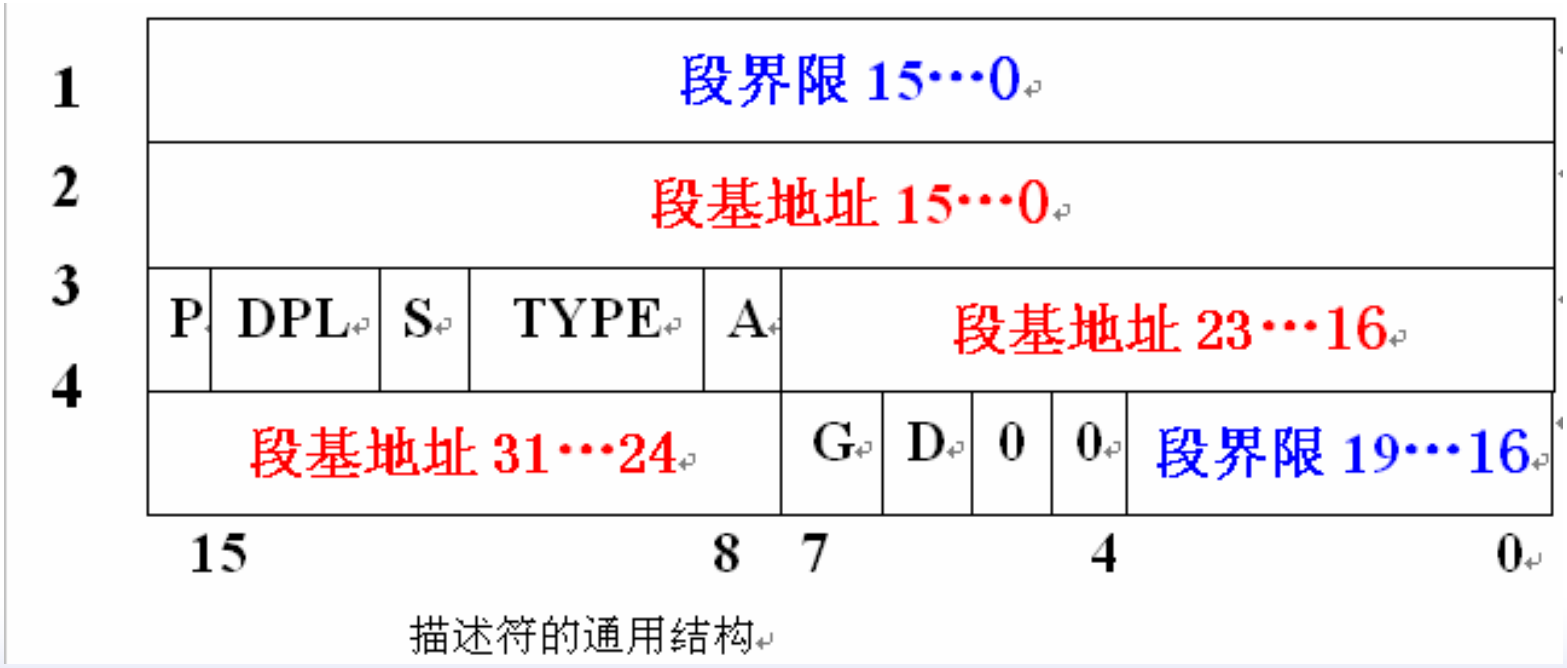
（2）要标记使用者是谁（权限级别）。

（3）中间环节：CPU要去判断此次访问是否合法。

1.3.3 存储器物理地址的形成

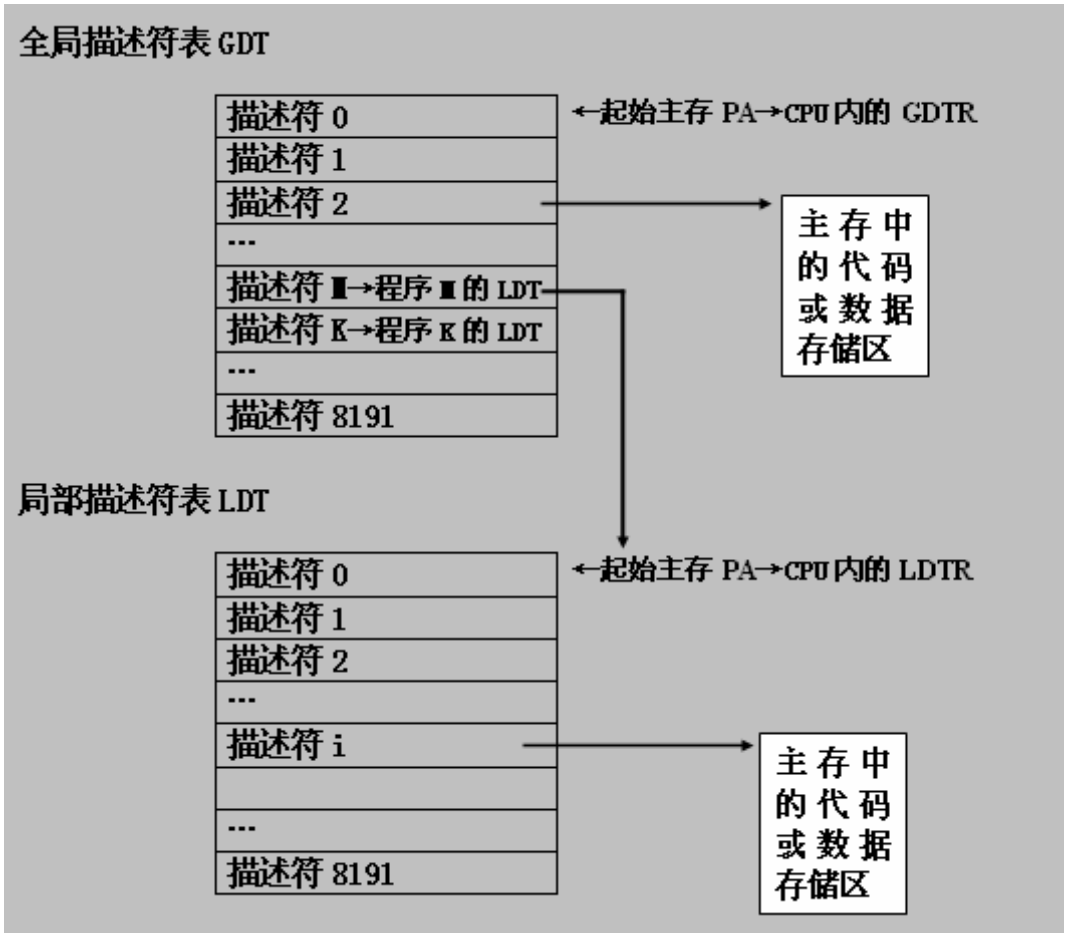
(1) 如何标记存储区

操作系统每次将某块存储区分给某程序使用时，用8个字节的描述符描述这段存储区的特征。



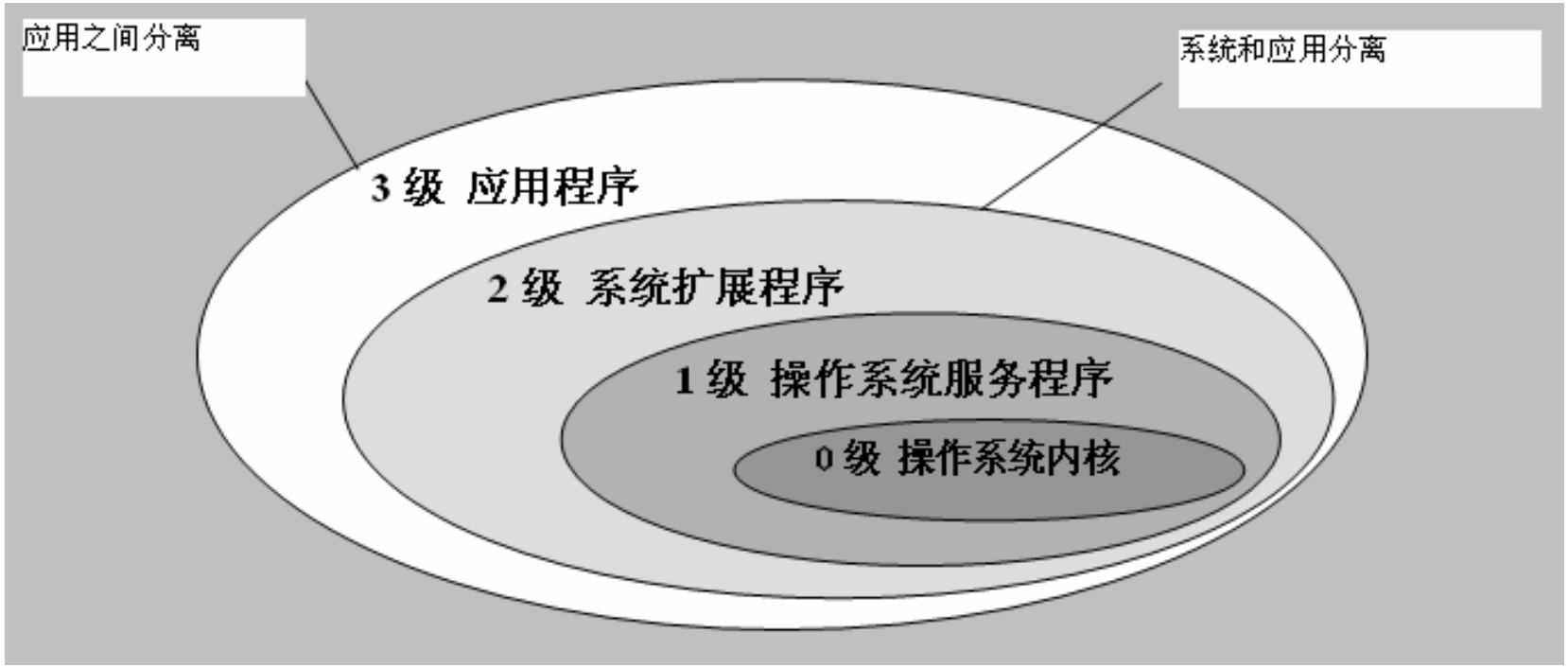
描述符的通用结构

1.3.3 存储器物理地址的形成



1.3.3 存储器物理地址的形成

(2) 如何标记使用者



1.3.3 存储器物理地址的形成

程序的权限由程序所在的段决定，由当前程序正在使用的段寄存器内的段选择符反映。

段选择符→段寄存器中

描述符索引	TI	特权级
15 4	3	1 0

TI =0 从全局描述符表中找。
=1 从局部描述符表中找。

描述符索引 13 位。2^{13} = 8192

1.3.3 存储器物理地址的形成

（3）中间执行环节：CPU去判断

例如，执行：MOV　AL，DS：[100H]

设：　（CS）= 0000000000001 1 11 B，

　　（DS）= 0000000000002 0 01 B，

（a）程序权限与被访问数据段的权限的关系；

（b）（DS）指向的描述表中寻找描述符；

（c）对应描述符描述的段是否可以被访问；（类别、权限）

（d）从描述符2中提取段的首地址（32位）；

（e）段的首地址（32位）+32位EA（100H）= 线性地址，不分页时即为PA

（f）送出PA，选中存储单元，将内容读入AL中。

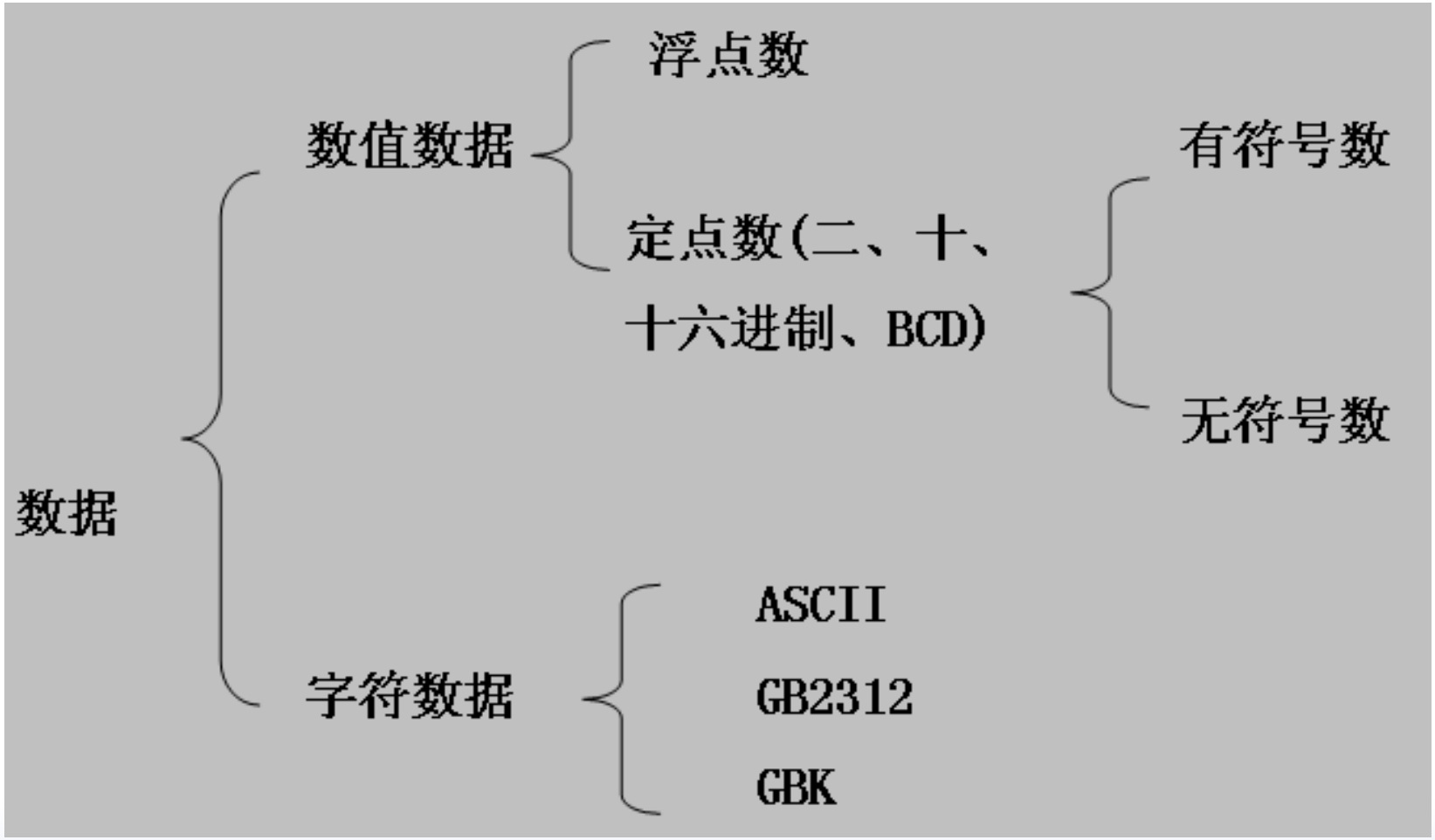
1.3.3 存储器物理地址的形成

保护方式的实际含义:

程序知道的“段首址”实际上是一个代号，是一个虚拟的起始地址，CPU通过查表转换才能获得真正的段首地址。

在这个由CPU完成的查表转换过程中，实现了访问权限的判断，达到了保护的目的。

1.4 数据在计算机内的表示形式



1.4.1 数值数据

对于有符号数，一律采用二进制补码

$$[X]_{补}=\begin{cases} X & 0 \leqslant X < 2^{n-1} \\ 2^n + X \ (2^n - |X|) & -2^{n-1} \leqslant X < 0 \end{cases}$$

1.4.1 数值数据

补码表示的几个特点:

1. 所有正数的补码表示最左(高)位为0，其二进制补码表示为本身;

2. 所有负数的补码表示最左(高)位为1，其补码表示为: (原码)除符号位保持不变外，其它位取反加1。

例如: 设n=8,

[50]补 = [00110010B]补 = 00110010B

[-50]补 = [-00110010B]补 =11001110B

1.4.1 数值数据

3. 一个二进制补码的最高位向左延伸S位，所得到的仍是此数的补码表示。

当 X=+1

n=8　[X]补=0000 0001B

n=16　[X]补=0000 0000←[0] 0000 0001B

=0000 0000 0000 0001B

当 X=−1

n=8　[X]补=1111 1111B

n=16　[X]补=1111 1111←[1] 1111 1111B

= 1111 1111 1111 1111B

1.4.1 数值数据

例：设n=8　X1=-0101 0111B(-87)，
X2=-0011 0101B(-53)，求[X1+X2]补

[X1+X2]补 = [X1]补+[X2]补
= 1010 1001B+1100 1011B
= 1 0111 0100B(116) > 0

相加后，原本8位的二进制数变成了9位，这多出的一位称为进位位。由于一个字节只能是8位，多出的一位(即进位位)就被丢掉了，丢失了符号位使得结果成为一正数，这种情况称为溢出。

1.4.1 数值数据

在标志寄存器中有一溢出位OF，当出现上述情况时，则OF位置1，检查OF就可判断运算是否溢出。

对加运算来说，使OF位置1的条件是：同符号数相加，结果的符号与之相反，则OF为1，否则为0。

即：正数 + 正数 = 负数　　负数 + 负数 = 正数

OF = 1

正数 + 正数 = 正数　　负数 + 负数 = 负数

OF = 0

1.4.1 数值数据

例：设n=8，X1=-0100 0010B(-66)，

X2=0110 1101B(109)，求[X1]补-[X2]补。

[X1]补-[X2]补 = [X1]补+[X2]补

= 1011 1110B + 1001 0011B

= 1 0101 0001B (81)>0

在用补码作减运算时，负数减正数，一定为负数（结果），而结果为正说明产生了溢出。

1.4.1 数值数据

对减运算来说，如果被减数与减数异号，且结果的符号与减数相同(与被减数相反)，说明运算产生了溢出，结果是错误，否则不是溢出。

即: 正数 - 负数 = 负数　　负数 - 正数 = 正数

OF = 1

正数 - 负数 = 正数　　负数 - 正数 = 负数

OF = 0

1.4.1 数值数据

n	十进制数表示范围		十六进制数表示范围	
	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

无符号数表示范围

1.4.1 数值数据

n	十进制数表示范围		二进制数表示范围		补码表示范围	
	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

有符号数表示范围

1.4.2 BCD码

BCD(Binary Coded Decimal)码

用4位二进制数（例如0000-1001）表示1位十进制数（0-9）。

如: 1983 = 0001 1001 1000 0011 BCD

非压缩的BCD码: 一个字节存放一个十进制数字位

0000 0000 = 0 0000 0001 = 1 0000 0010 = 2

压缩的BCD码: 一个字节存放2个十进制数字位

0001 0000 =10 1000 1001 =89

1.4.3 字符数据

键盘输入的字母和数字、显示器显示的文字等都是字符信息。

西文：常用的将字符与2进制数对应起来的编码方法是美国信息标准交换代码ASCII码。

汉字： GB2312编码

区位码表：分94区*94位，包括一级汉字（最常用汉字，按拼音排序）和二级汉字（稍常用汉字，按部首排序），均为简体汉字，共约6700个。

1.5 标志寄存器

标志寄存器用来保存在一条指令执行之后，CPU所处状态的信息及运算结果的特征

16位标志寄存器：FLAGS

32位标志寄存器：EFLAGS

1.5.1 标志位

一、条件标志位

1. 符号标志SF

若运算结果为负则SF=1，否则SF=0;

这里负是指运算结果的最高位为1则SF=1

2. 零标志ZF

若运算结果为0则ZF=1否则ZF=0

例:

```
MOV  AX,  0
ADD  AX,  1        SF=0  ZF=0
SUB  AX,  1        SF=0  ZF=1
SUB  AX,  2        SF=1  ZF=0
```

1.5.1 标志位

3. 溢出标志OF

当将操作数作为有符号数看时，使用该标志位判断运算结果是否溢出。

加法：若同符号数相加，结果的符号与之相反则OF=1，否则OF置0。

减法：被减数与减数异号，而结果的符号与减数相同则OF=1，否则置0。

1.5.1 标志位

4. 进位标志CF　将操作数看作无符号数时，使用该标志位判断运算结果是否发生了进位或者借位。

加法：若运算结果从字或字节的最高位向前产生了进位则CF置1，否则置0。

减法：两数相减，若将它们看作无符号数，则够减无借位置0，有借位置1。

1.5.1 标志位

实际运算结果	应得结果	
	看作无符号数	看作有符号数
0000 0100	4	+4
+ 0000 1011	+ 11	+ (+11)
0000 1111	15	+15
无符号数 15	CF=0	OF=0
有符号数 15		

1.5.1 标志位

实际运算结果	应得结果	
	看作无符号数	看作有符号数
0000 0111	7	+7
+ 1111 1011	+ 251	+ (−5)
1 0000 0010	258	+2
无符号数2	CF=1	OF=0
有符号数2		

1.5.1 标志位

实际运算结果	应得结果	
	看作无符号数	看作有符号数
0000 1001	9	+9
+ 0111 1100	+ 124	+ (+124)
1000 0101	133	+133
无符号数133	CF=0	OF=1
有符号数-123		

1.5.1 标志位

实际运算结果	应得结果	
	看作无符号数	看作有符号数
1000 0111	135	−121
+ 1111 0101	+ 245	+ (−11)
1 0111 1100	380	−132
无符号数124	CF=0	OF=0
有符号数124		

1.5.1 标志位

5. 奇偶标志位PF　当运算结果（指低8位）中1的个数为偶数时，PF置1，否则置0。该标志位主要用于检测数据在传输过程中的错误。

6. 辅助进位标志位AF　标识作字节运算的时候低半字节向高半字节的进位和借位。

1.5.1 标志位

二、控制标志位

1. 方向标志DF

控制串操作指令的处理方向

DF=0，正向（从低地址向高地址）

DF=1，反向（从低地址向高地址）

2. 中断允许标志IF

控制CPU是否允许响应外设的中断请求。

IF=0，关中断（CPU屏蔽外设的中断请求）

IF=1，开中断（CPU响应外设的中断请求）

3. 跟踪标志TF

控制单步执行。

IF=0，CPU连续工作

IF=1，CPU单步执行

1.5.2 标志寄存器操作指令

1. 标志寄存器传送指令

① LAHF

功能：将标志寄存器的低8位→AH

说明：本指令不带操作数，也不影响标志位。

② SAHF

功能：将(AH)→标志寄存器的低8位。

说明：标志寄存器原来低8位的内容全部被冲掉，以AH的内容来取代。但它算不影响高字节内容。(因高字节中以控制标志为主，不能随便改变)

1.5.2 标志寄存器操作指令

③ 标志寄存器进栈指令PUSHF

功能：将标志寄存器内容压入堆栈 即：

（SP）–2→SP　（FLAGS）→↓（SP）

④ 标志寄存器出栈指令POPF

功能：将栈顶的内容送入标志寄存器

1.5.2 标志寄存器操作指令

2. 标志位操作指令

进位标志CF
- CLC 使CF=0 (Clear carry)
- CMC 使CF取反 (Complement carry)
- STC 使CF=1 (Set carry)

方向标志DF
- CLD 使DF=0 (Clear direction)
- STD 使DF=1 (Set direction)

中断标志IF
- CLI 使IF=0 (Clear interrupt)
- STI 使IF=1 (Set interrupt)

1.6 汇编源程序举例

```
.386
;数据段
DATA    SEGMENT USE16   ;段为16位段
  SUM     DW   ?    ;SUM为字变量，初值不定
  BB      DD   1    ;BB为双字变量，初值为1
DATA    ENDS
```

1.6 汇编源程序举例

```
;堆栈段
STACK SEGMENT USE16 STACK;段名和组合类型
      DB 200 DUP(0) ;堆栈的大小为200个字节
STACK SEGMENT
```

1.6 汇编源程序举例

```
;代码段
CODE    SEGMENT    USE16
        ASSUME   CS:CODE,  SS:STACK,DS:DATA,  ES:DATA
START:  MOV    AX,  DATA
        MOV    DS,  AX    ;数据段首址送DS
```

1.6 汇编源程序举例

```
        MOV   CX, 50     ;循环计数器置初值
        MOV   AX, 0      ;累加器置初值
        MOV   BX, 1      ;1→BX
NEXT:   ADD   AX, BX     ;(AX)+(BX)→AX
        INC   BX
        INC   BX         ;(BX)+2→BX
        DEC   CX         ;(CX)-1→CX
        JNE    NEXT      ;(CX)≠0转NEXT
        MOV   SUM, AX    ;(CX)=0累加结果→SUM
```

1.6 汇编源程序举例

```
        MOV     AH, 4CH
        INT     21H         ;返回DOS
CODE    ENDS
        END     START
        ;源程序结束语句。程序运行时，启动地址为START
```

1.6 汇编源程序举例

与C语言程序的对比:

```
main()
{    int  sum=0;  //初始化变量
     int  i, j;
     j = 1;
    for(i=0; i<50; i++)
    {   sum = sum + j; //求和
        j +=2;  }
 }
```

比较：C语言程序编译成可执行文件之后，大小为24K字节。汇编语言为几百字节。

1.6 汇编源程序举例

总结：汇编语言语句的格式

[名字]　操作符　[操作数或地址]　　[;注释]

名字：段名、变量名、标号名

字母、数字、特殊符号（_、@等）组成，

不能以数字开始。

不可重名、不可与关键字相同

1.6 汇编源程序举例

操作符
- 机器指令助记符 => 机器指令语句
- 伪指令助记符 => 伪指令语句
- 宏定义名 => 宏指令语句

1.6 汇编源程序举例

操作数:

2个:	ADD AX, BX	双操作数指令
1个:	INC AX	单操作数指令
0个:	CLC	零操作数指令

第一章总结

本章应掌握的主要问题:

1. 汇编语言、汇编源程序、汇编程序的概念。
2. 通用寄存器组(EAX EBX ECX EDX)、作指示和变址寄存器的寄存器组(ESI EDI EBP ESP)、段寄存器组(CS DS ES SS)、指令指示器(EIP)的作用。

第一章总结

3. 主存编址方式、物理地址的形成方式。堆栈的概念和堆栈操作指令 PUSH、POP的功能及使用格式。
4. 在80x86机中，数的表示范围（有符号、无符号），压缩和非压缩BCD码的表示形式，字符在机内的表示形式。
5. 标志寄存器 SF、ZF、OF、CF标志位改变方法
6. 汇编源程序的基本结构

本章作业

- 1.2
- 1.5
- 1.11
- 1.14