第2章 8086微处理器

- ✓2.1 8086 CPU系统结构
- ✓2.2 8086 CPU中寄存器
- ✓2.3 存储器
- ✓2.4 堆 栈
- ✓2.5 汇编源程序举例

处理器简介

- ✓中央处理器(CPU)是微型计算机的核心部件，也是划分计算机档次的一个决定性指标。

- ✓8086和8088芯片是Intel公司于1978年先后推出的两种同档次的CPU产品，在此之后，该公司又陆续推出80286、80386、80486和Pentium系列CPU产品。

处理器简介

- ✓8086与8088微处理器内部结构几乎相同，均由执行部件(EU)和总线接口部件(BIU)组成，有相同的寄存器和内部总线。
- ✓但二者的数据线不同，8086具有16位数据线，8088仅有8位数据线，是准16位微处理器，8086运行速度较快。
- ✓同时二者的指令队列长度不同，8086指令队列寄存器为6个字节长，8088仅有4个字节长。
- ✓本书内容针对的处理器均以8086为主。

2.1.1 8086 CPU组成

- ✓8086 CPU具有16条数据线，20条地址线，寻址范围可达1MB，是Intel公司于1977年推出的16位微处理器，按功能可以分为两大部件：
- ✓执行部件(EU)
- ✓总线接口部件(BIU)
- ✓8086 CPU内部结构如下图所示。

1MB?

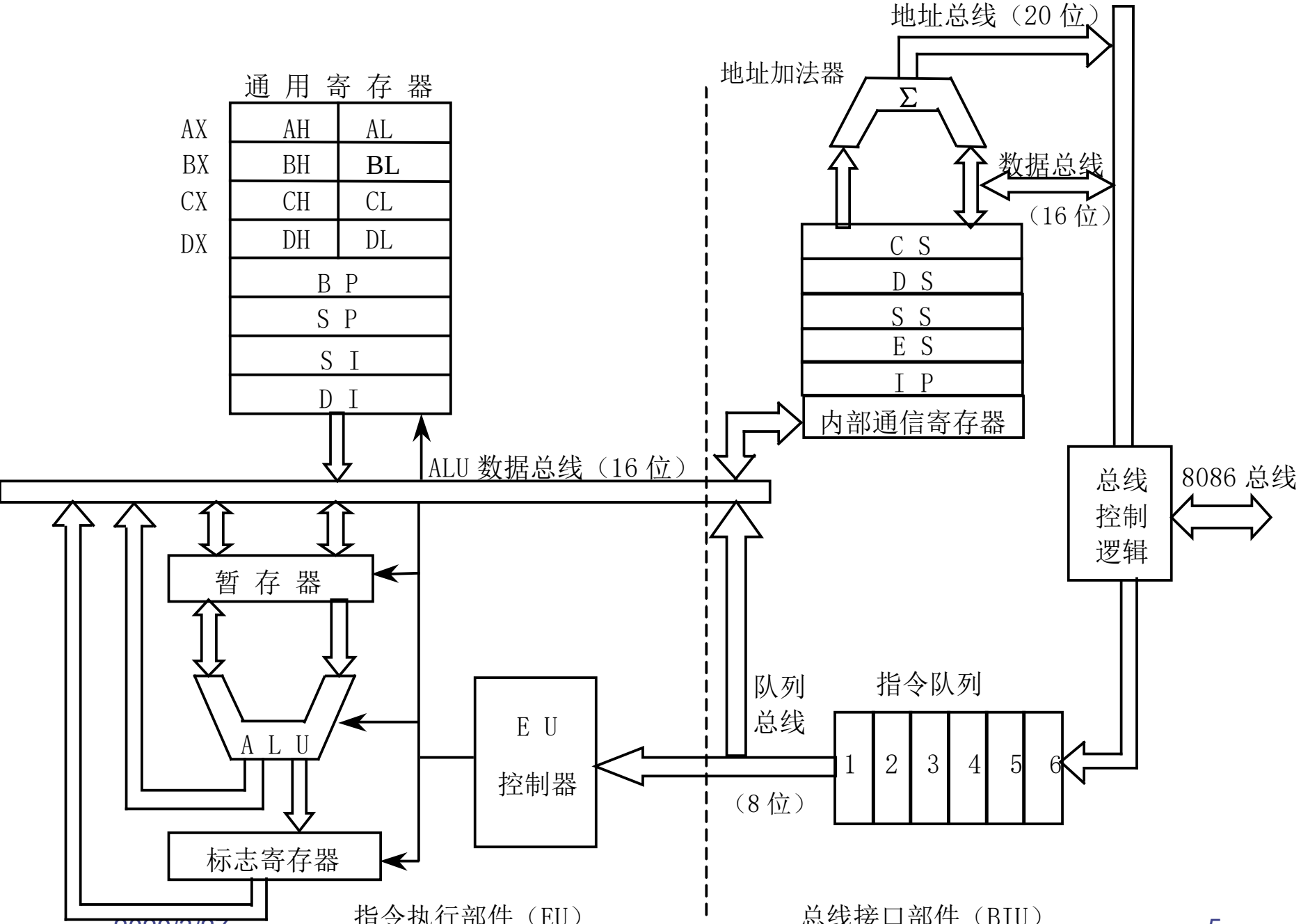
1Byte(字节)＝8bit(比特/二进制位), 1KB=1

存储器中是以字节为基本单元的！

2^{20}字节=1MB！

64-bit AMD/Intel	
Address Bus	40-bit
Bytes	1,099,511,627,776
KiB	1,073,741,824
MiB	1,048,576
GiB	1024
TiB	1

The data bus and address bus are independent, and chip designers can use whatever size they want for each. Usually, however, chips with larger data buses


地址总线（20 位）
地址加法器
Σ
数据总线
（16 位）
通 用 寄 存 器
AX
BX
CX
DX
AH
AL
BH
BL
CH
CL
DH
DL
B P
S P
S I
D I
C S
D S
S S
E S
I P
内部通信寄存器
ALU 数据总线（16 位）
总线
控制
逻辑
8086 总线
暂 存 器
A L U
E U
控制器
队列
总线
指令队列
1
2
3
4
5
6
（8 位）
标志寄存器
指令执行部件（EU）
总线接口部件（BIU）

1. 执行部件(EU)

程序指令的执行

- ✓寄存器组共有8个16位寄存器，这些寄存器均属于CPU专用存储器，存在于CPU内部，是CPU内部临时存放数据的部件，它的存取速度比内存更快。
- ✓算术逻辑部件(ALU)是用来进行算术和逻辑运算的。
- ✓标志寄存器又称程序状态寄存器，用于存放当前指令执行的状态和运算结果的特征。

2. 总线接口部件(BIU)

- ✓总线接口部件用来执行所有的总线操作。BIU由地址加法器、段寄存器、指令指针IP、指令队列和总线控制逻辑组成，负责CPU与存储器或外部设备之间交换数据。

- ✓地址加法器是将指令指针IP和段寄存器CS，或将EU送来的偏移量与段寄存器DS形成一个20位的物理地址，用以从存储器中取出指令或数据。

2. 总线接口部件(BIU)

✓指令队列是一个6字节的寄存器（8086），最多可存放6个字节的指令。指令队列是一个先进先出的栈，当空闲2个指令字节时，BIU自动从存储器中取出指令存入指令队列中，供EU部件使用。

✓总线控制逻辑是用来控制BIU中各部件的协同操作。

3. EU与BIU的关系

- ✓执行部件(EU)和总线接口部件(BIU)的操作是独立进行的，因此可以并行工作。
- ✓在EU执行指令过程中，BIU就可以取出指令存放在指令队列，而当EU执行完一条指令后就可以立刻到指令队列中去取下一条将要执行的指令。
- ✓从而节省了CPU因等待到内存取指令所需要的时间，提高了CPU的利用率，加快了系统运行速度。

2.1.2 程序执行过程

- ✓为了提高CPU的运行速度，8086系统设计为并行工作方式，即指令和数据的存取电路与指令的执行电路是并行工作的。

- ✓假设程序的指令代码预先已存放在存储器中，为了执行程序，CPU依据时钟的节拍，产生一系列控制信号，有规则地重复执行下面的过程：

- ✓(1) BIU从存储器中取出一条指令存入指令队列寄存器中。

2.1.2 程序执行过程

- ✓(2) EU从指令队列中取出指令并执行，利用总线空闲时间，BIU从内存取出第二条指令存入指令队列寄存器中，或取第三条指令到指令队列中。

- ✓(3) EU执行下一条指令，如果前一条指令有写存储器的要求，则通知BIU，由BIU把前条指令的结果写到存储器中，然后再取指令到指令队列中。

- ✓(4) 如指令执行，要求读取操作数，则由BIU取操作数完成指令读功能。

- ✓(5) EU执行再下一条指令，返回(1)处继续执行上述操作过程。

2.1.2 程序执行过程

- ✓总之，在指令执行过程中，利用EU分析操作码和执行指令过程中不占用总线时间这一特点，由BIU自动地通过总线取存储器指令到指令队列中，从而使指令的执行可以不间断的进行，提高了执行指令的速度。

- ✓程序的执行，就是上述指令执行的重复过程。

2.2 8086 CPU中寄存器

- ✓寄存器是CPU内部临时存放数据的部件，每一个寄存器相当于CPU中的一个存储单元，寄存器的名字就是该存储单元的符号地址。
- ✓寄存器的存取速度比内存更快，可以把数据通过内部总线送往运算器进行运算，或者接收来自运算器的结果。充分利用CPU的内部寄存器可以加快程序的执行速度。
- ✓程序可见寄存器可以分为通用寄存器、专用寄存器和段寄存器3类。

2.2.1 通用寄存器

- ✓8086有8个16位的字型通用寄存器，各个寄存器都有自己特定的功能，在程序中常用来临时存放中间结果。
- ✓通用寄存器包括数据寄存器(AX，BX，CX，DX)和地址寄存器(SI，DI，BP，SP)两种

- ✓(1) AX—累加器。这是算术运算的主要寄存器，仅用于存放操作数或运算结果，同时外设的操作指令(IN、OUT)都使用AX传送数据。
- ✓(2) BX—基址寄存器。用于存放操作数的偏移地址，也可存放数据。
- ✓(3) CX—计数寄存器。用于循环指令、移位指令以及串操作指令的隐含计数控制。
- ✓(4) DX—数据寄存器。在乘除法运算中，与AX组合在一起存放双字型数据，DX用来存放高位字。
- ✓(5) SI—源变址寄存器。作为串操作指令的源操作数的地址指针，也可用于存放操作数的偏移地址。
- ✓(6) DI—目的变址寄存器。作为串操作指令的目的操作数的地址指针，也可用于存放操作数的偏移地址。
- ✓(7) BP—堆栈基址寄存器。用于存放堆栈中操作数的EA。
- ✓(8) SP—栈顶指针。用于指示栈顶的偏移地址，与堆栈段寄存器SS联用来确定堆栈段中的某一存储单元的地址。

- ✓上述每个寄存器都是16位的，可以存放一个字类型数据，
- ✓通用寄存器中的AX、BX、CX和DX，每一个寄存器又可以拆成两个8位寄存器使用，分别为AH、AL、BH、BL、CH、CL、DH、DL。
- ✓对于SI、DI、BP、SP，它们只能以字(16位)为单位进行使用，不可拆分为相应的两个8位寄存器使用。

AX，AH，AL是否互相影响？

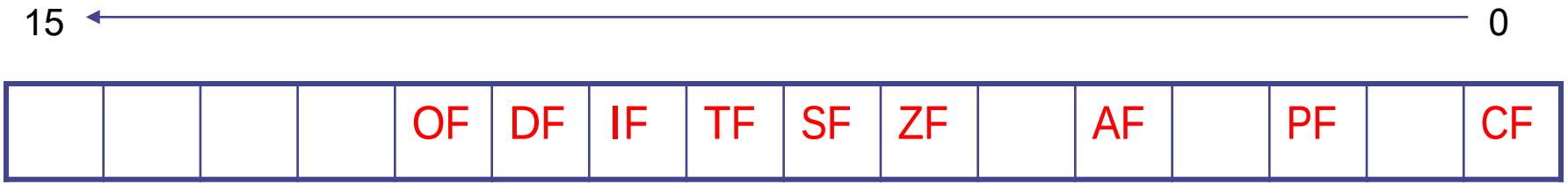
- ✓对于80386及其后继机型而言，它们是32位的通用寄存器，包括EAX，EBX，ECX，EDX，ESP，EBP，EDI和ESI。在这些机型中，它们可以用来保存不同宽度的数据，如可以用EAX保存32位数据，用AX保存16位数据，用AH或AL保存8位数据。

E是什么含义？

2.2.2 专用寄存器

- ✓IP(Instruction Pointer)为**指令指针寄存器**，又称指令指针，用于存放代码段中**汇编指令的入口地址**(即**偏移地址**)。
- ✓在程序运行过程中，CPU从代码段的偏移地址为IP的内存单元中取出指令代码的一个**字节**后，**IP自动加1**，指向指令的下一个字节。
- ✓**用户程序不能直接访问IP。**

- ✓FLAGS为标志寄存器，又称程序状态寄存器(Program Status Word, PSW)，共16位，用于存放当前程序指令执行的状态和运算结果的特征。
- ✓标志寄存器中的每一位又称标志位，8086使用其中的9个标志位，各标志位分布如下图所示。



标志寄存器中标志位分布图

✓FLAGS标志寄存器的9个标志位可分为两组：条件标志位:

CF—进位标志。
ZF—零标志。
SF—符号标志。
OF—溢出标志。
PF—奇偶标志。
AF—辅助进位标志。

Carry
Zero
Sign
Parity
Auxiliary

控制标志位:

TF—单步中断允许标志。 Trace
IF—外中断屏蔽标志。 Interrupt
DF—方向标志。 Direction

课本表2.1

与标志寄存器有关的指令

1. 标志寄存器传送指令

✓ (1) LAHF　　Load AH with FLAGS

功能：将标志寄存器的低8位送入AH中，即(FLAGS)7~0→AH。该指令的执行对标志位无影响。

例1：LAHF

执行前：(FLAGS) =0485H，(AX)=0FFFFH

执行后：(FLAGS)= 0485H，(AH)= 85H，(AL)=0FFH

✓ (2) SAHF　　Store AH into FLAGS

功能：将AH中的内容送入标志寄存器的低8位中，而高8位保持不变，即(AH)→(FLAGS)7~0。因此，该指令执行后，OF、DF、IF、TF的值均不会改变。

2. 标志寄存器进栈指令PUSHF

功能：将标志寄存器的内容压入堆栈，即(FLAGS)→↓(SP)。

例2：PUSHF

执行前：(FLAGS)=0485H

执行后：系统堆栈的内容为：([SP]) =85H04H

书上图??

3. 标志寄存器出栈指令POPF

功能： (SP)↑→FLAGS，即将栈顶内容弹出送入标志寄存器中。

例3：POPF

执行前：([SP])=0000H，(FLAGS)=0485H

执行后：(FLAGS)=0000H

2.2.3 段寄存器

- ✓段寄存器也是一种专用寄存器，它们专用于存储器寻址，用来直接或间接地存放逻辑地址中的段地址部分，段寄存器的长度为16位。
- ✓在8086中，存储器是分段使用的，且有四个专门存放段地址的寄存器，称为段寄存器。
- ✓(1) 代码段CS，用来存放当前正在运行的程序代码。
- ✓(2) 数据段DS，用来存放当前运行程序所用的数据，如果程序中使用了串处理指令，则其源操作数必存放在数据段中。

✓(3) 堆栈段SS，堆栈段定义了堆栈的所在区域，堆栈是一种数据结构，是一个特殊的数据存储区，并以先进后出的方式来访问数据，它常用来存放子程序的入口地址。

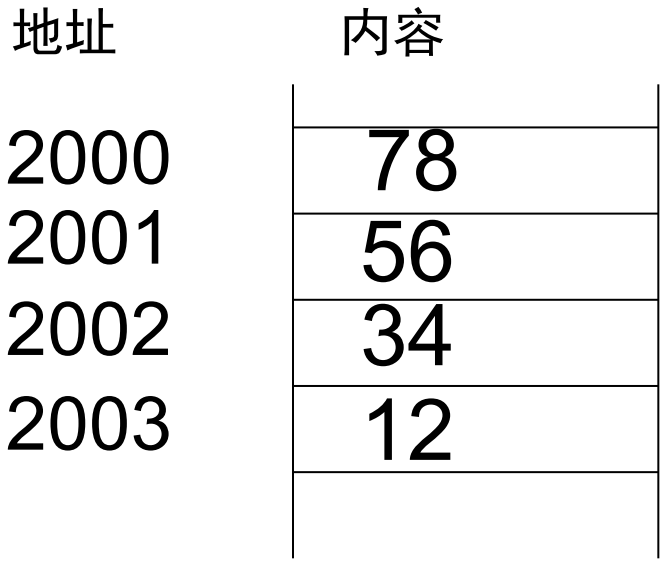
✓(4) 附加段ES，附加段是附加的数据段，它是一个辅助的数据存储区，也是串处理指令的目的操作数存放区。

2.3存储器

2.3.1 存储单元的地址和内容

- ✓内存储器是计算机中存储数据和信息的物理部件，所有程序必须被调入内存后，才能被计算机执行。
- ✓内存储器中存储信息的基本单位是一个二进制位(Bit)，一位可存储一个二进制数0或1，每8位组成一个字节(Byte)，每16位组成一个字(Word)，在存储器里以字节为基本单位来存储信息。

✓8086有20根地址线，直接可寻址的地址空间为2^{20}＝1MB，这1MB的内存单元其地址编号的范围用十六进制数表示为00000-FFFFFH。



字节单元：(2000H)=78H

字单元：(2000H)=5678H？

双字单元：

(2000H)=12345678H？

低字节低地址，高字节高地址！

2.3.2 存储器地址的分段

- ✓8086微处理器有20根地址线，直接寻址能力为1M＝2^{20}字节，也就是说，主存容量可达1M字节，物理地址编号从00000H～0FFFFFH。
- ✓CPU与存储器交换信息必须使用这种20位的物理地址。但是，8086内部却是16位结构，它里面与地址有关的寄存器全部都是16位的，如，SP、BP、SI、DI、IP等。因此，它只能进行16位地址运算，表示16位地址，寻找操作数的范围最多也只能是64K字节。
- ✓为了能表示20位物理地址，8086的设计人员提出了**存储器地址分段技术**的解决方案。

分段技术的解决方案

- ✓该技术方案是将1兆字节的存储器分成**若干个逻辑段**，每个逻辑段的容量最大可为64KB，允许逻辑段在整个存储器空间中浮动，各个段之间可以**相连、重叠、部分重叠或完全重叠**。
- ✓而且设置四个段寄存器CS、DS、SS、ES保存当前可使用段的**段首址**。

- ✓段不能起始于任意地址，而必须从任一小段的首地址开始。
- ✓机器规定：从0地址开始，每16个字节为一小段，段的大小为16字节的整数倍。这样就使各段的段首址都从能被16整除的地址开始，那么，这些段首址的最低4位总是0，暂时忽略这些0，则段首址的高16位正好装入一个段寄存器中。
- ✓访问存储单元时，CPU可以根据操作的性质和要求，选择某一适当的段寄存器，将它里面的内容左移4位二进制，即在最低位后面补入了4个0，恢复了段首址原来的值，再与本段中某一待访问存储单元的偏移地址相加，则得到该单元 20位物理地址。这样一来，寻找操作数的范围就可达到1M字节。

- ✓实际上，程序员在编制程序时可以根据需要来确定段的大小，它可以是1B、100B、1000B或在64KB范围内的任意个字节。
- ✓存储器中所划分的段共有四种类型，包括**代码段、数据段、堆栈段和附加数据段**，每个段的起始地址即段地址都由相应段寄存器保存，当然，每个汇编程序可以包含多个段，对于同种类型的段而言，只有**一个当前段**。

- ✓由此可见，存储器的分段使用技术可以很方便地将程序中的代码、数据、堆栈分开存放在不同的存储区中。
- ✓尽管CPU在某一时刻最多只能同时访问4个段，但它并不限制程序中也只能定义4个段，用户完全可以根据自己的要求定义多个代码段、多个数据段和多个堆栈段。
- ✓如果CPU需要访问4个段以外的存储区，只要改变相应段寄存器的内容即可。

2.3.3 存储器物理地址的生成

- ✓8086/8088系统中，每一个存储单元都有一个唯一的20位地址，称为该存储单元的物理地址。
- ✓CPU访问存储器时，必须首先确定所要访问的存储单元的物理地址才能取得该存储单元的内容。
- ✓20位的物理地址由16位段地址和16位段内偏移地址计算所得。因此任一存储单元物理地址的计算方法是：

物理地址＝段地址×10H＋偏移地址　　？

20位　　16位　　16位

✓代码段是程序代码的存储区。当前代码段在主存中的起始地址由代码段寄存器CS确定，指令指针IP总是保存着下一条将要执行指令相对于CS的偏移地址，如果要取出这条指令执行，它的物理地址PA应为：

$$PA=(CS)\times 10H+(IP)$$

✓数据段是程序中所使用的数据存储区，它可以分为数据段和附加数据段，其大小均可达64K字节，当前数据段的段首址由数据段寄存器DS确定，当前附加数据段是数据存储区的附加区域，其段首址由附加数据段寄存器ES确定，物理地址为：

PA＝(DS或ES)×10H十该存储单元的偏移地址

- ✓堆栈段可以用来作程序的临时数据存储区，存放暂时不用的数据。
- ✓特别是在作子程序调用、系统功能调用、中断处理等操作时，堆栈段更是必不可少的。
- ✓用户一旦定义好堆栈段，系统则自动以SP为指针并设置好指针的位置。往堆栈中压入数据和从栈中弹出数据时，只能使用PUSH和POP命令，这时栈顶的物理地址为：

$$PA=(SS)\times 10H+(SP)$$

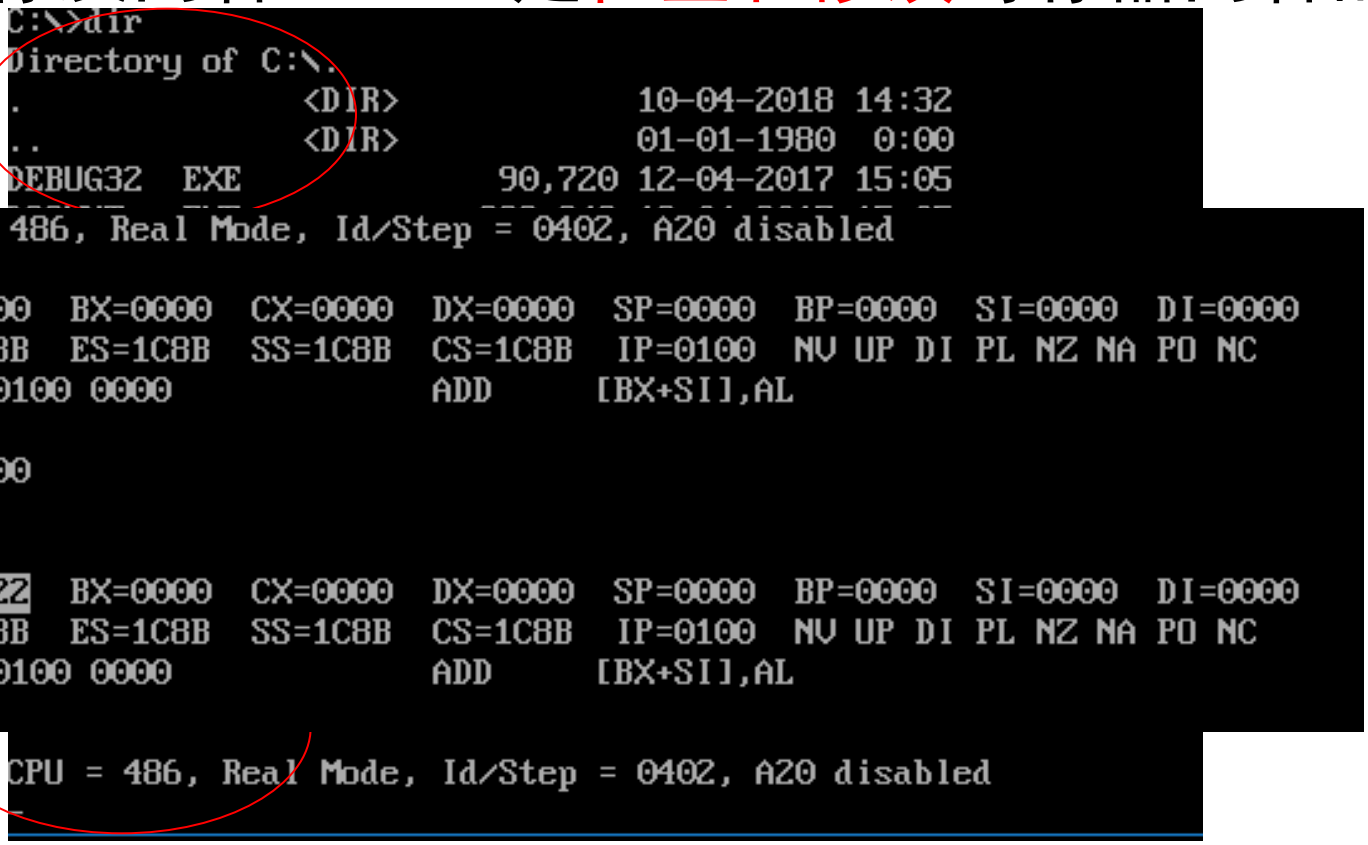
2.3.4 存储单元中数据的操作

- ✓存储单元是存储数据的地址空间，在计算机内部，存储单元所在的物理区域有寄存器、存储器、外设接口等。
- ✓计算机工作时，一般先由只读存储器中的引导程序启动系统，再从外存中读取系统程序和应用程序送到内存中运行。本节所讲的存储单元中数据的操作主要指寄存器与内存中数据的操作，数据包括数值数据、地址数据、寄存器数据和指令代码。使用的工具是Debug调试工具

存储单元中数据的操作方法

1.寄存器中数据的显示

✓DOS提示符下键如Debug命令，然后出现命令提示符“—”，输入“R”命令后，即可显示出各寄存器的名称及内容。“R”是检查和修改寄存器内容的命令。



存储单元中数据的操作方法

2.内存单元内容的查看

✓DOS提示符下键如Debug命令，然后出现命令提示符“—”，输入D和U命令后，即可查看内存单元的内容。

```
-d
1C8B:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
***Duplicate Line(s)***
1C8B:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
-u
1C8B:0102 0000            ADD     [BX+SI],AL
1C8B:0104 0000            ADD     [BX+SI],AL
1C8B:0106 0000            ADD     [BX+SI],AL
1C8B:0108 0000            ADD     [BX+SI],AL
1C8B:010A 0000            ADD     [BX+SI],AL
1C8B:010C 0000            ADD     [BX+SI],AL
1C8B:010E 0000            ADD     [BX+SI],AL
1C8B:0110 0000            ADD     [BX+SI],AL
1C8B:0112 0000            ADD     [BX+SI],AL
1C8B:0114 0000            ADD     [BX+SI],AL
1C8B:0116 0000            ADD     [BX+SI],AL
1C8B:0118 0000            ADD     [BX+SI],AL
-_
```

存储单元中数据的操作方法

3.内存单元内容的修改

✓DOS提示符下键如Debug命令，然后出现命令提示符“—”，输入E命令后，即可修改内存单元的内容。

```
-e
1C8B:0000  CD.88
-d
-e ds:1000 3f 'xyz' 8d
-d ds:1000 10004
1C8B:1000 3F 78 79 7A 8D 00 00 00-00 00 00 00 00 00 00 00  ?xyz............
1C8B:1010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
***Duplicate Line(s)***
2C8B:0000 00 00 00 00 00                                   .....

1C8B:0030 00 00 14 00 18 00 8B 1C-FF FF FF FF 00 00 00 00  ................
1C8B:0040 05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1C8B:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 00 00 00 00  M!K.............
1C8B:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1C8B:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

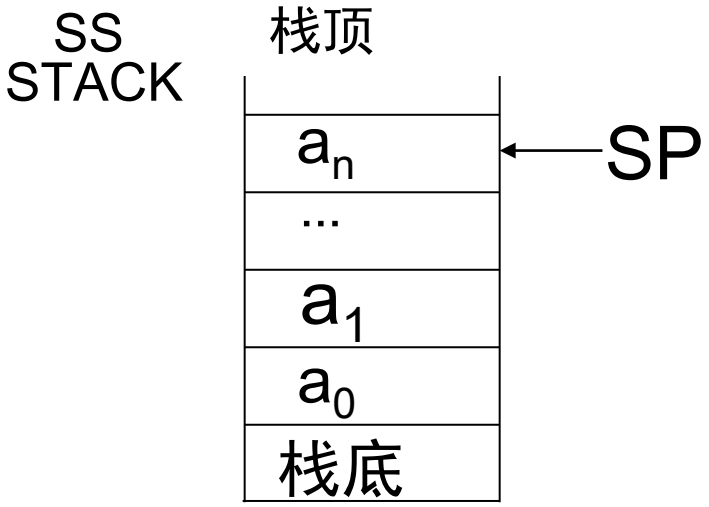
2.4 堆栈

- ✓堆栈是在主存中开辟的一片特殊的数据存储区，这片存储区采用的是一端固定，另一端活动，即只允许在一端插入或删除数据的“先进后出”存储方式。
- ✓堆栈中数据的存取也是遵循“先进后出”的原则。
- ✓堆栈中的数据也称元素或栈项，数据进栈称“压入”，出栈称“弹出”。

堆栈

- ✓从硬件的观点看，堆栈是由一片存储单元和一个指示器组成。固定端叫栈底,活动端叫栈顶。栈指针用来指示栈元素进栈和出栈时偏移地址的变化，指针所指示的最后存入数据的单元叫栈顶，所有信息的存取都在栈顶进行，因而栈指针总是指向栈顶的 。
- ✓8086允许用户建立自己的字堆栈，最大空间可达64K字节，其存储区位置由堆栈段寄存器SS给定，并固定采用SP作栈顶指针，即SP的内容为栈顶相对于SS的偏移地址。

✓空栈时，SP指向堆栈段的最高地址即栈底，存入数据时栈顶均由高地址向低地址变化



堆栈和栈顶指针示意图

在堆栈中存取数据必须采用专门的堆栈指令进行，而且只能是**字操作**。!

1、进栈指令PUSH

语句格式：PUSH OPS

功能：将寄存器、段寄存器或存储器中的一个字数据压入堆栈中。

例1 PUSH AX

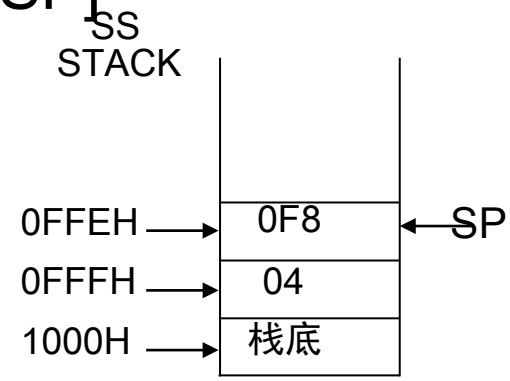
执行前：（AX）=04F8H，（SP）=1000H

执行：①（SP）-1→SP；（AH）→[SP]　　（AX）→↓[SP]

②（SP）-1→SP；（AL）→[SP]

执行后: (SP)=0FFEH

([SP])=04F8H



2、出栈指令POP

功能：将栈顶元素弹出送至某一寄存器，段寄存器（除CS外）或存储器中。

例2 POP BX

执行前：（BX）=1111H，堆栈内容如例1所示。

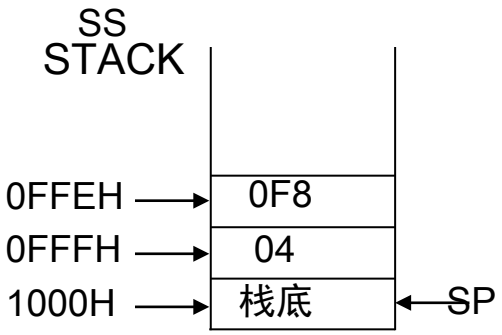
执行：①（[SP]）→BL；（SP）+1→SP

②（[SP]）→BH；（SP）+1→SP ↑（[SP]）→BX。

执行后：（BX）=04F8H；

（SP）=1000H，堆栈成为空栈。

除了PUSH和POP指令外，如果其它指令要访问堆栈时，只能通过基址寄存器BP进行。



2.5汇编源程序举例

- ✓ 汇编语言程序与其它高级语言程序有显著的区别，一个汇编源程序一般由几个段组成，其中必不可少的是代码段，如果程序中需要使用数据存储区，还要定义数据段，必要时还要定义附加数据段。在子程序调用时，还要定义堆栈段。
- ✓ 下面以一个完整的汇编源程序实例来说明汇编语言程序的有关规定和格式。

屏幕显示字符串“ Hello world ! ”，该汇编源程序可用C语言编程实现，C语言程序代码如下。

```
#include “stdio.h”
Main()
{
  char *str=” Hello world !”;  //数据部分
  /*……*/
  Printf(“%s\n”,str);          //代码部分
}
```

汇编语言实现字符串显示1

```
DATA    SEGMENT          ；数据段
BUF     DB  0AH, 0DH, " HELLO WORLD ! $"
DATA    ENDS
STACK   SEGMENT  STACK ；堆栈段
        DB  200  DUP(0)
STACK   ENDS
CODE    SEGMENT          ；代码段
ASSUME  CS:CODE, DS:DATA
  START: MOV   AX,  DATA ；将数据段首址
DATA→AX(MOV为传送指令)
        MOV   DS,   AX  ；将数据段首址DATA置入数据
段寄存器DS
```

汇编语言实现字符串显示2

```
        LEA   DX, BUF ；将变量BUF的偏移地址传给DX
          MOV  AH, 9   ；将立即数9传给AH
          INT  21H      ；DOS中断调用
          MOV  AH，4CH ；4CH→AH
          INT  21H      ；DOS中断调用，这两条指令执行
完后，计算机将结
                        ；束本程序的运行，返回DOS状态
  CODE  ENDS
        END  START
```