第 2 章　微处理器的功能结构

2.1　8086 的内部结构

8086 CPU 是 Intel 系列的 16 位微处理器。它的内部寄存器、内部运算部件以及内部操作都是按 16 位设计的，它的外部总线包括 16 条数据线和 20 条地址线，可寻址的存储空间是 2^{20}=1MB。

如图 2.1 所示，8086 CPU 从功能上可分为两部分，总线接口单元（Bus Interface Unit，BIU）和执行单元（Execution Unit，EU）。

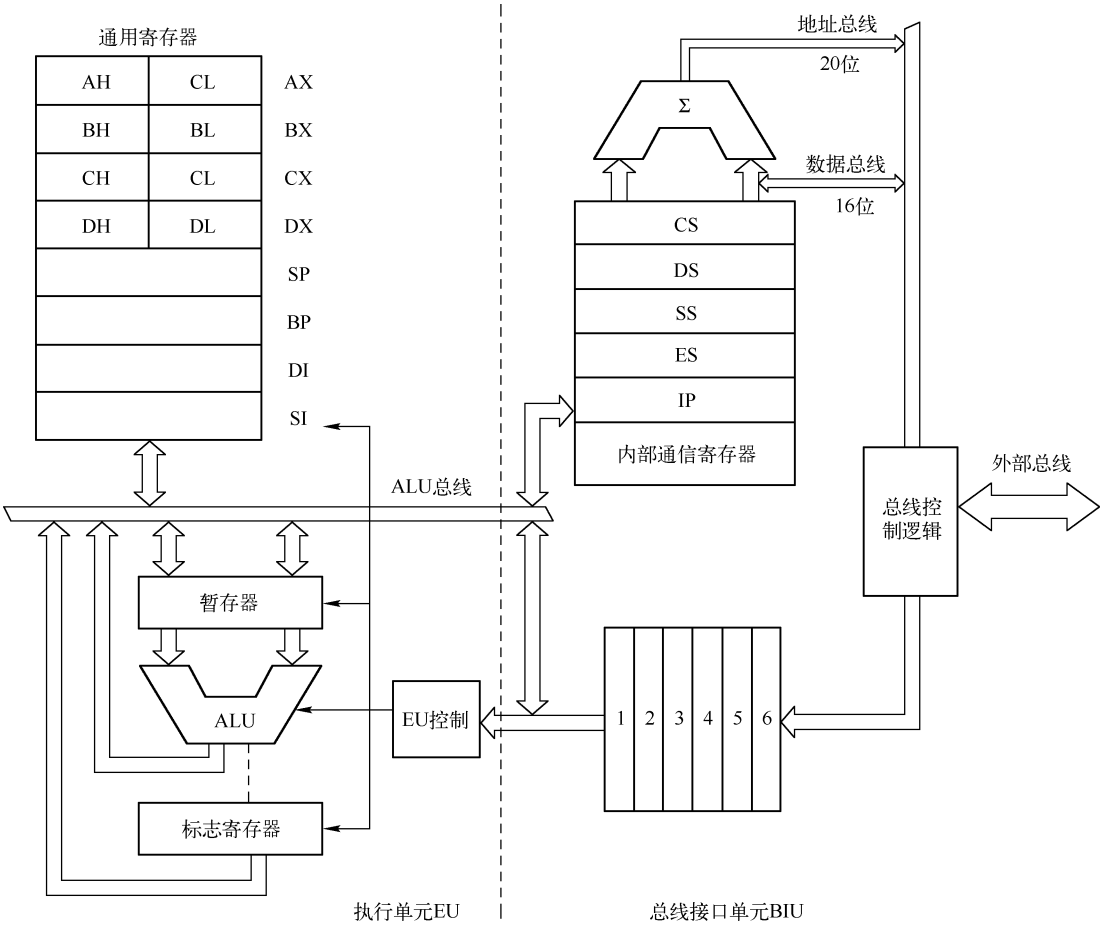

图 2.1　8086 CPU 的内部结构

1．总线接口单元

总线接口单元由 4 个段寄存器、16 位指令指针寄存器 IP、专用的 20 位的地址加法器和

6 字节的指令队列缓冲器组成。总线接口单元的功能是：①从内存读取指令送到指令队列；②CPU 执行指令时，总线接口单元要配合执行单元从指定的内存单元或者外设端口读取数据，或将数据送到指定的内存单元或者外设的端口。

2．执行单元

执行单元由 EU 控制电路、通用寄存器、算术逻辑运算单元 ALU 以及标志寄存器构成。执行单元的作用是：①从指令队列中取出指令；②对指令进行译码，发出相应的传送数据或算术运算的控制信号；③接收由总线接口单元传送来的数据，或把数据传送到总线接口单元；④进行算术或逻辑运算。

一条指令的执行过程分为取指令阶段和执行执行阶段，如图 2.2 所示。取指令的过程由 BIU 负责，先形成地址，然后从存储器读取指令编码送入指令队列。执行指令的过程由 EU 负责，包括从指令队列中取出指令、对指令进行译码并产生相应的控制信号。

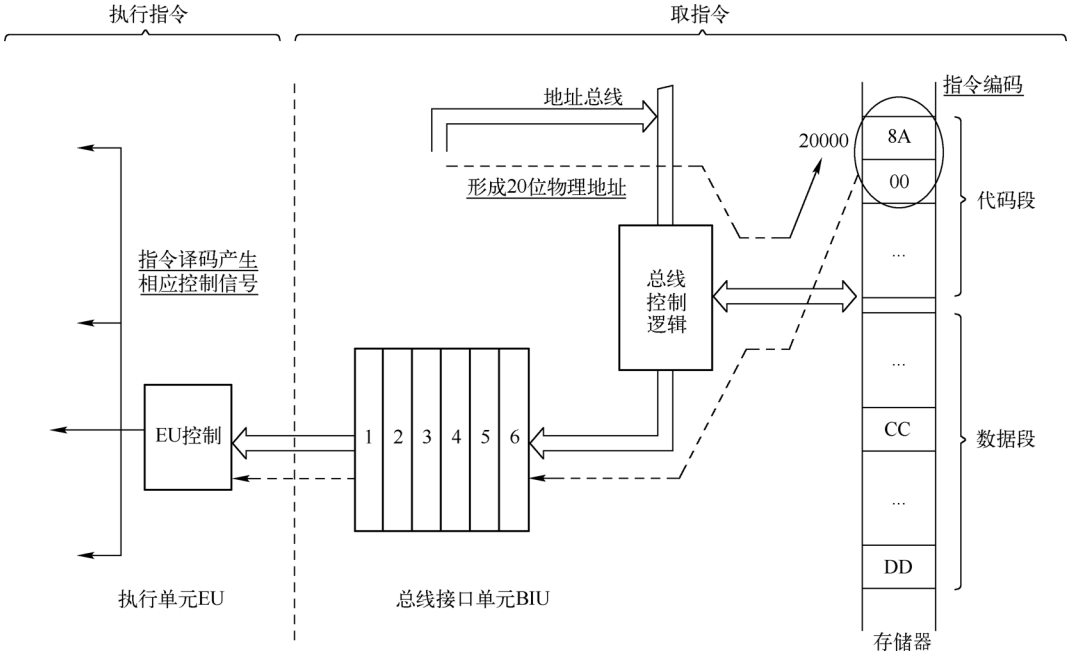

图 2.2　指令的执行过程

在指令的执行过程中，EU 和 BIU 之间通过指令队列联系起来。多数情况下，BIU 在不停地向队列写入指令，而 EU 每执行完一条指令后，就向队列读取下一条指令。EU 和 BIU 的相互独立性使得二者能够并行工作。当 EU 在执行指令时，空闲的 BIU 可以从内存读取后续指令到指令队列。这样就可以将取指令工作和执行指令工作重叠进行，从而提高 CPU 的工作效率，加快指令的执行速度。

2.2　8086 的寄存器结构

寄存器是 CPU 内部存储容量有限的高速存储部件，它们可用来暂存指令、数据和地址。寄存器的功能十分重要，CPU 对存储器中的数据进行处理时，往往先把数据取到内部

寄存器中，而后再作处理。

由于寄存器数量很少，所以 80x86 对寄存器的使用采用命名方式，即每个寄存器都有一个名字，而不采用类似存储器的地址编号的使用方法。80x86 的寄存器一般都使用有意义的命名，即寄存器的名字往往反映寄存器的常用功能。

8086 的寄存器结构如图 2.3 所示，主要包含 14 个 16 位寄存器。

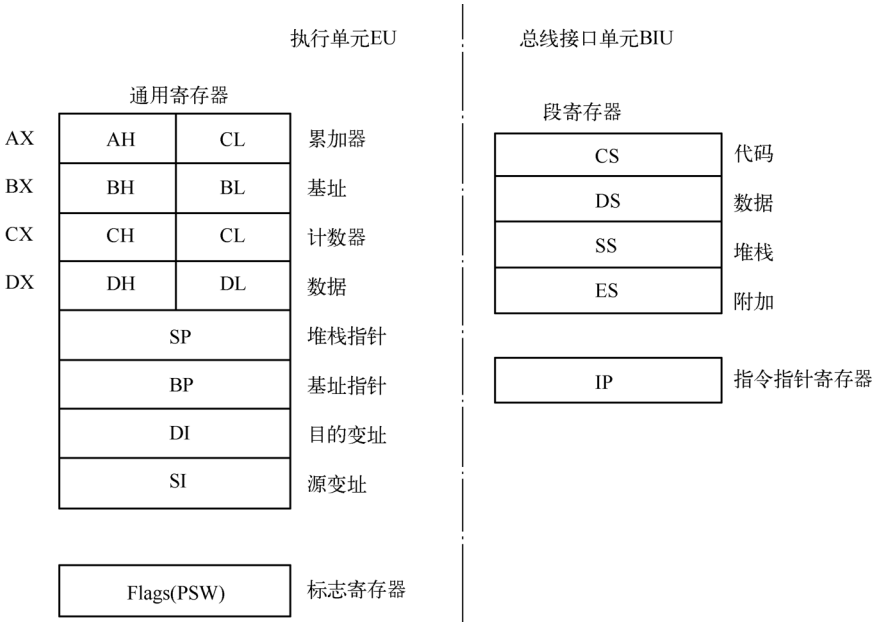

图 2.3　8086 的寄存器结构

2.2.1　通用寄存器

1．数据寄存器

（1）AX：累加器，它是算术运算的主要寄存器，I/O 指令使用该寄存器与外设传送信息。

（2）BX：基址寄存器，它常在基址寻址方式中用来保存基地址。

（3）CX：计数器，它在循环指令中作为隐含的计数器。

（4）DX：数据寄存器，DX 可以和 AX 一起存储双字，DX 存放高 16 位。

AX、BX、CX、CX 均为 16 位寄存器，但它们每个都可以分开作为两个单独的 8 位寄存器使用。AX 的高 8 位部分命名为 AH，低 8 位部分命名为 AL。同理，BX 分为 BH 和 BL，CX 分为 CH 和 CL，DX 分为 DH 和 DL。这样，8086 又有 8 个 8 位寄存器可以使用。例如：(AX)=1010101011111111 表示 AX 寄存器中保存的 16 位二进制数为 1010101011111111，这样 AH 寄存器存储的内容是(AH)=10101010，而 AL 寄存器存储的内容是(AL)=11111111。

2．指针寄存器

（1）SP：堆栈指针寄存器，用来指示栈顶的偏移地址。

（2）BP：基址指针寄存器，常用作堆栈区中的基地址指针。

指针寄存器不可分割成 8 位寄存器。作为通用寄存器，也可存储算术逻辑运算的操作数和运算结果。

3．变址寄存器

（1）SI：源变址寄存器。

（2）DI：目的变址寄存器。

SI 和 DI 常在变址寻址方式中用来保存变址，在串指令中 SI 和 DI 具有自动增或自动减功能。变址寄存器不可分割成 8 位寄存器。作为通用寄存器，也可存储算术逻辑运算的操作数和运算结果。

2.2.2 段寄存器

（1）CS：代码段段寄存器。

（2）DS：数据段段寄存器。

（3）ES：附加段段寄存器。

（4）SS：堆栈段段寄存器。

段寄存器是根据内存分段的管理模式而设置的，用来保存段地址。

2.2.3 指令指针寄存器

IP：指令指针寄存器，用来存放下一条要执行的指令在代码段中的偏移地址。

在程序运行过程中，IP 始终指向下一条指令的首地址，与 CS 寄存器联合确定下一条指令的物理地址。如图 2.4 中例子所示，BIU 根据 CS 和 IP 形成下一条指令的物理地址，当这一地址送到存储器后，就可以取得下一条要执行的指令并送入指令队列；而控制器一旦取得这条指令，马上修改 IP 的内容，将 IP 加上指令的长度即指令的字节数，保证 IP 始终指向下一条指令的首地址。8086 就是这样通过 IP 寄存器来控制指令序列的执行流程，因此，IP 寄存器是一个很重要的寄存器。

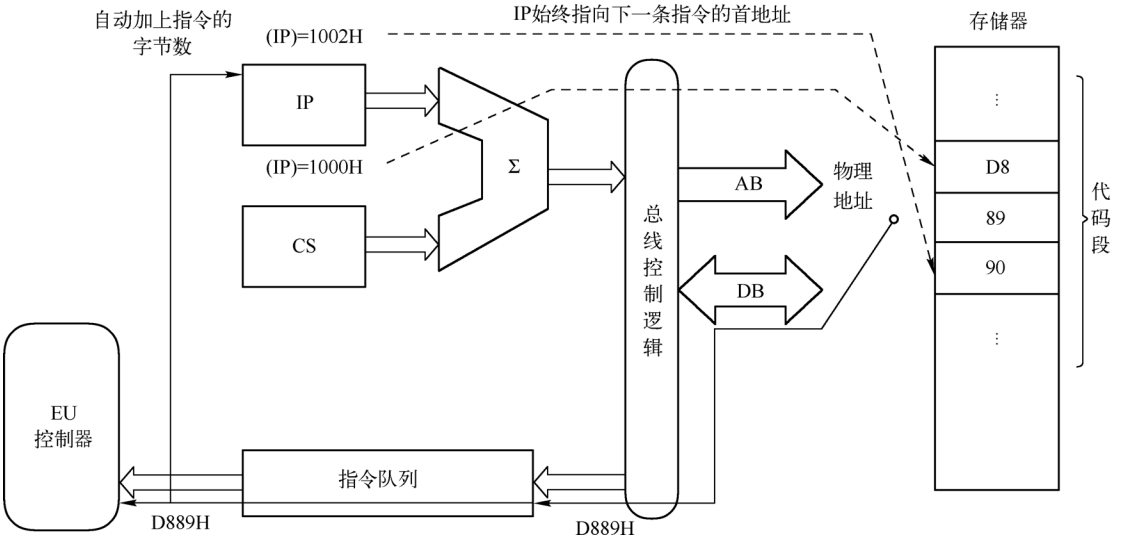

图 2.4　IP 的作用

2.2.4 标志寄存器

Flags：标志寄存器，用于保存状态标志和控制标志，其结构如图 2.5 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

图 2.5　Flags 寄存器结构

1．状态标志

状态标志用来记录程序运算结果的状态信息。由图 2.1 可以看出，状态标志是由 ALU 自动产生的，如果一条指令未经过 ALU 的处理就可认为该指令对状态标志没有影响。ALU 产生的这些状态信息往往用作后续条件转移指令的转移控制条件，所以状态标志又称为条件码，它包括以下 6 位。

（1）CF：进位标志。它记录运算时从最高有效位产生的进位值或借位值。最高有效位有进位或借位时 CF=1，否则 CF=0。

（2）PF：奇偶标志。它记录运算结果的奇偶检验条件。当结果操作数中“1”的个数为偶数时 PF=1，否则 PF=0。

（3）AF：辅助进位标志。在字节运算时，由低半字节（字节的低 4 位）向高半字节有进位值或借位值时，AF=1，否则 AF=0。

（4）ZF：零标志。运算结果为零时 ZF=1，否则 ZF=0。

（5）SF：符号标志。它记录运算结果的最高位，即有符号数的符号。

（6）OF：溢出标志。在运算过程中，如运算结果已超出了机器能表示的数值范围（指有符号数）称为溢出，此时 OF=1，否则 OF=0。

关于状态标志可以结合图 2.6 所示的串行加法器来理解。加法器的方式控制 M 用于控制加减法，当 M=0 时进行 S=A+B 操作，当 M=1 时进行 S=A–B 操作。当 M=1 时各个异或门对 B 的各位进行求反并将 1 作为初始进位加入结果，也就是执行对 B 的求反加 1，即求补操作。加法器计算结果同时产生状态标志：最高位的进位 C_n 就是 CF；结果的最高位 S_{n-1} 就是 SF；OF 是最高位的进位 C_n 和次高位的进位 C_{n-1} 的异或；ZF 表示结果是否是全 0；PF 表示结果是否有偶数个 1。

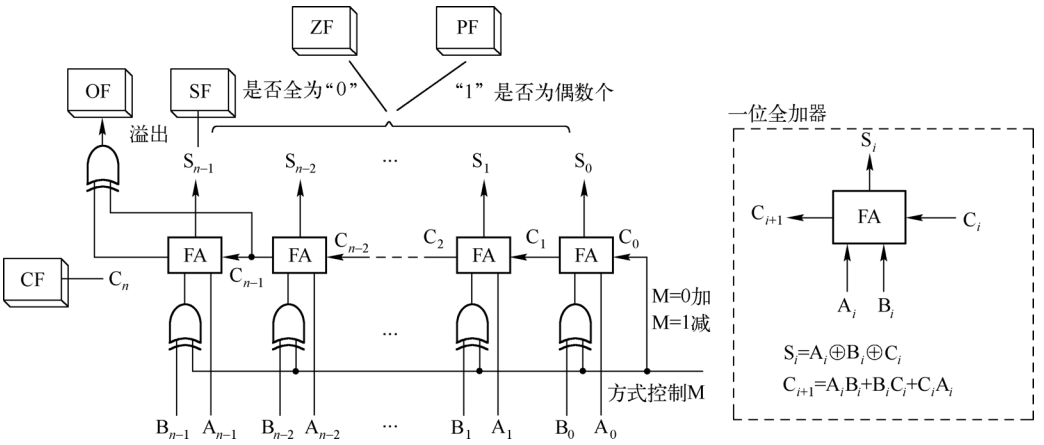

图 2.6　串行加法器和状态标志

【例 2.1】 计算 2345H–3219H 的结果，并观察主要状态标志的变化。

十六进制运算：2345H–3219H=0F12CH 或 2345H+（–3219）=2345H+0CDE7H=0F12CH；

十进制运算：9029–12825=–3796；

二进制运算：$-\dfrac{0010001101000101}{0011001000011001}$ 或 $+\dfrac{0010001101000101}{1100110111100111}$；

$$\dfrac{}{1111000100101100}\qquad\dfrac{}{1111000100101100}$$

SF=1，因为最高位为 1；

ZF=0，因为运算结果不为 0；

PF=?，虽然运算结果是偶数个 1，但 8086 只有在 8 位运算时才产生 PF 标志；

CF=0，因为最高位没有进位；

OF=0，由于 d_{14} 位没向 d_{15} 位产生进位，d_{15} 位也没有进位，即 0 和 0 异或为 0。

注：十六进制用后缀“H”或“h”表示，如果十六进制数以“A～F”开始，前面还要加“0”。

进位标志 CF 和溢出标志 OF 是两个不同性质的标志，不能混淆。

【例 2.2】 如表 2.1 所示，分别判断无符号数和有符号数的加法进位及溢出标志，并判断计算结果是否正确。

表 2.1 无符号数和有符号数加法示例

无符号数加法		有符号数加法	
$\begin{array}{r}00001001\\+\ 01111100\\\hline 10000101\end{array}$	$\begin{array}{r}00000010\\+\ 11111111\\\hline 00000001\end{array}$	$\begin{array}{r}00001001\\+\ 01111100\\\hline 10000101\end{array}$	$\begin{array}{r}00000010\\+\ 11111111\\\hline 00000001\end{array}$
CF=0 OF=1	CF=1 OF=0	CF=0 OF=1	CF=1 OF=0
正确：9+124=133	错误：2+255=1	错误：9+124=–123	正确：2+(–1)=1

从例 2.2 中的溢出情况可以得出结论：对于无符号数的加法，应该使用 CF 标志来判断是否溢出；而对于有符号数的加法，应该使用 OF 标志来判断是否溢出。

2．控制标志

在标志寄存器中还有 3 个控制标志。

（1）DF：方向标志。在串处理指令中控制方向。当 DF=1 时，每次操作后使 SI 和 DI 减量，这样就使串处理从高地址向低地址方向进行；当 DF=0 时，则使 SI 和 DI 增量，使串处理从低地址向高地址方向进行。

（2）IF：中断标志。当 IF=1 时，允许 CPU 响应可屏蔽中断请求，否则不允许响应。

（3）TF：跟踪标志。用于程序的单步调试。当 TF=1 时，每条指令执行完后产生一个内部中断，暂停程序的执行，进行调试。TF 又称陷阱标志。

2.3 8086 的存储组织

2.3.1 物理空间与字节序

8086 的存储器是按字节编址的。它具有 20 条地址总线，所以可寻址的存储器地址空间

容量为 2^{20}=1MB。每字节对应一个唯一的地址，地址范围为 0～(2^{20}–1)（用十六进制表示为 00000～FFFFFH），这个存储空间就称为 8086 的物理空间，是最终要使用物理器件实现的。访问物理空间所使用的 20 条地址线对应的 20 位地址称为物理地址。

8086 是 16 位 CPU，即字长为 16 位。也就是说，字由两字节组成，即高字节（高 8 位）和低字节（低 8 位）。字在存储器中使用两个连续的字节存储，因此，这其中存在一个字的高字节和低字节与存储器中的两字节（高地址字节、低地址字节）的对应关系问题。我们将在字节编址空间中存储超过 1 字节数据的存储方法（实际上就是刚提到的对应关系）称为字节序。8086 规定低字节存储在低地址空间、高字节存储在高地址空间，如图 2.7 所示。这种低字节存储在低地址空间的字节序通俗地叫作“低字节在前”。实际上，80x86 体系的所有 CPU 都采用这种字节序，所以它又被称为 Intel 字节序。

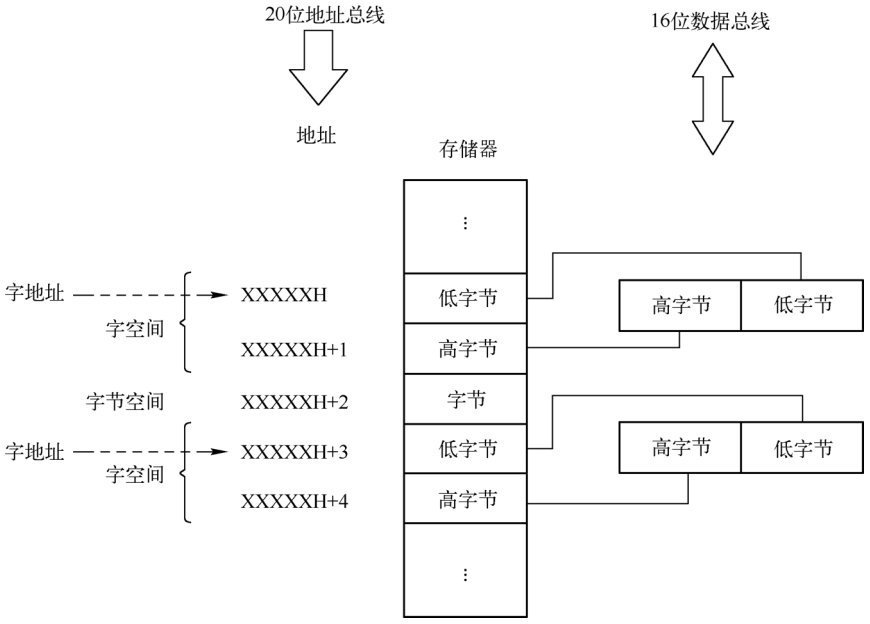

图 2.7　8086 的字节序

【例 2.3】 已知字 557AH 在内存中存储的地址为 10000H，问 10000H 和 10001H 的内容是什么？

如图 2.8（a）所示，按照 Intel 字节序，低地址 10000H 中存储低字节 7AH，高地址存储高字节 55H。

【例 2.4】 已知双字 1020557AH 存储在 20000H 的存储空间内，问 20001H 单元的存储内容是什么？

如图 2.8（b）所示，按照 Intel 字节序，双字 1020557AH 存储在地址 20000H 开始的连续四个字节内：7AH、55H、20H、10H，所以 20001H 单元的内容应是 55H。

字节序是一个容易被忽视的细节问题，但它在程序的可移植性方面具有很重要的地位，即使是在高级语言环境中，也是很重要的。

在图 2.7 的示例中给出了两个字的存储地址，这两个字的存储地址的区别是：一个字是偶地址开始存储，另一个是奇地址开始存储。虽然 8086 的数据总线是 16 位的，即一次就可以从内存读/写 16 位的数据，但对于这两个字的访问是有区别的，其区别在于：对于偶地址

开始的字只需要一个总线周期访问速度快；而对于奇地址开始的字则需要两个总线周期，访问速度慢。

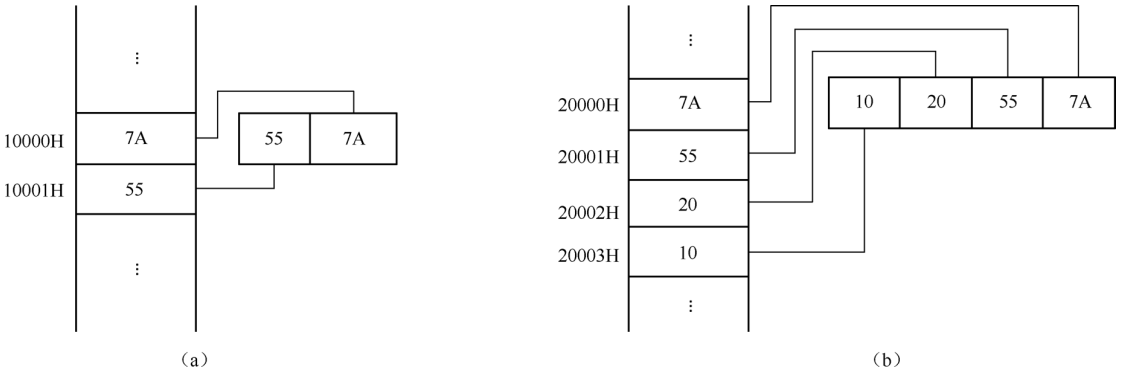

图 2.8　字和双字的存储示例

对于软件层次而言，访问偶地址开始的字要比访问奇地址开始的字快，这一点对程序性能的优化非常重要，所以要求编程者尽可能地将字数据保存在偶地址开始的空间内。这个问题称为“对齐”，16 位机要求 2 字节对齐，即字的地址应该是 2 的整数倍（依次类推，32 位机要求 4 字节对齐）。

2.3.2　逻辑空间

在计算机发展史上，从直接在物理空间编程发展到在逻辑空间编程是非常重要的进步。

直接在物理空间编程会有什么问题呢？首先，要求程序员必须了解物理空间及空间的使用情况，至少不能和系统程序冲突，如果是与其他人合作，还需要进行空间的协调；其次，一旦空间发生变化，就需要对程序进行修改，不仅繁琐而且有时是致命的；另外，就 8086 来说，内部 16 位的寄存器和 20 位地址之间在运算上也十分不便。

8086 采用分段存储管理方式，构建了由多个独立的逻辑段构成的逻辑空间，如图 2.9 所示。建立了逻辑空间的概念后，程序设计将针对逻辑空间进行。这样就解除了程序对实际物理空间的依赖，主要表现在以下三方面：第一，程序设计时不需要了解存储器的使用情况，操作系统或监控程序会将程序分配在空闲空间，不会和系统程序或其他程序冲突；第二，程序设计时采用的是逻辑空间的地址，存储管理部件将自动、透明地进行逻辑空间地址到物理空间地址的转换，即逻辑空间到物理空间的映射，所以，不同的空间分配对程序没有影响；第三，逻辑空间是相对“观察者”而存在的，即不同人面对的逻辑空间是独立无关的，所以每个编程者都是在自己的逻辑空间上编程。

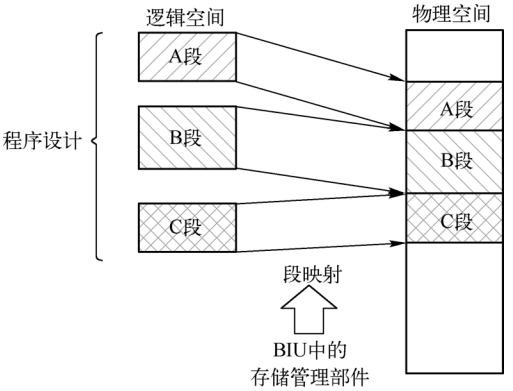

图 2.9　逻辑空间与物理空间

段式逻辑空间由多个独立的逻辑段构成，使得编程者可以将不同程序元素按不同的段

进行组织。一般将代码组织在代码段、数据组织在数据段、堆栈组织在堆栈段，其组织示例如例 2.5 所示。

【例 2.5】 分析下面程序设计中程序元素的逻辑存储结构。

```
DATA SEGMENT
      A     DW  ?     ; 变量 A
      B     DW  ?     ; 变量 B      } 数据段：将数据放在数据段内
      C     DW  ?     ; 变量 C
DATA ENDS
STACK SEGMENT STACK
            DW  128 DUP(?)      ;堆栈空间      } 堆栈段：提供堆栈空间
STACK ENDS
CODE SEGMENT
            ASSUME CS:CODE,DS:DATA,SS:STACK
START:      MOV AX,DATA
            MOV DS,AX
            MOV AX,A
            ADD AX,B                          } 代码段：将所有的程序代码组织在代码段内
            MOV C,AX
            MOV AX,4C00H
            INT 21H
CODE ENDS
            END START
```

2.3.3 存储管理

1．逻辑地址

对于一个段，8086 使用 16 位地址在段内寻址，称为段的偏移地址。16 位偏移地址决定了段的最大长度是 2^{16}=64KB。8086 规定，当段被加载到内存时，其在存储器中的起始地址必须是 16 的整数倍，即段的起始地址的最低 4 位为 0。段在存储器中的起始地址称为段址。为了方便处理，段址舍去了最低 4 位的 0 而形成了 16 位的地址值。在 8086 中，将 16 位的段址和 16 位的偏移地址合称为逻辑地址。逻辑地址可以使用“段址：偏移地址”的形式表示。

8086 是使用逻辑地址编程的，即所有在指令中出现的地址都是逻辑地址。8086 的这一机制，就是 2.3.2 节指出的针对逻辑空间编程。如图 2.10 所示，在程序设计时，段址指明要

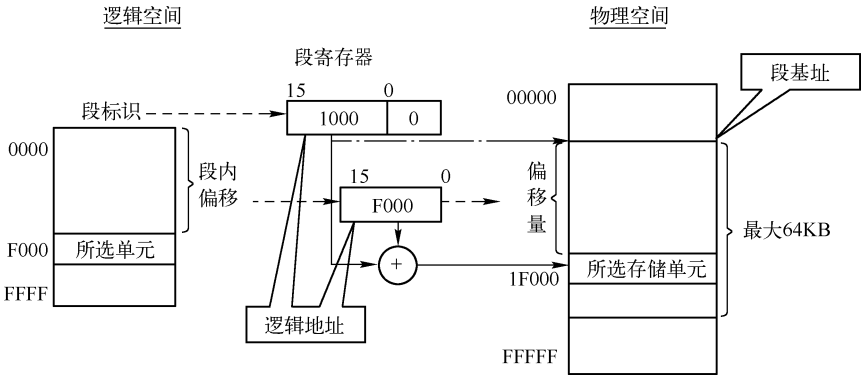

图 2.10　逻辑地址的意义

访问的存储单元所处的段，偏移地址是存储单元的段内地址；当程序被加载执行时，段被映射到物理空间上，段址表示段在物理空间上的起始位置，偏移地址表示要访问的存储单元相对于起始位置的偏移量。从图中可以看出，程序设计时的段内地址和程序被加载执行时的偏移量是一回事。所以，当程序被加载执行时，段被分配在实际的物理空间上，产生了本次执行的段址的实际值，此时只需要将程序中所有逻辑地址的段址修改为实际值即可，而偏移地址不需要修改。

2．逻辑地址到物理地址的转换

在 8086 中，由逻辑地址到物理地址的转换是由 BIU 中的专用加法器完成的，如图 2.11 所示，其实现方法是：20 位物理地址=16 位段地址×10H+16 位偏移地址。

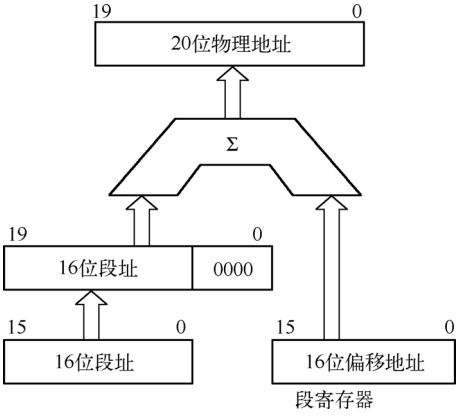

图 2.11　逻辑地址到物理地址的转换

【例 2.6】 某单元的逻辑地址为 4B09H：5678H，求存储单元的物理地址。

物理地址=段地址×10H+偏移地址=4B09H×10H+5678H=4B090H+5678H=50708H

地址转换是实现逻辑空间到物理空间映射的实际机制。不难想象，计算机对地址转换机制的效率要求是非常高的，因为每次访存都不可避免地要执行一次地址转换。

3．段寄存器的作用

为了保证地址转换的效率，访问存储器时，段址总是由段寄存器提供的。

8086 在 BIU 中设有 4 个段寄存器（CS、DS、SS、ES），所以，CPU 在某一时刻可以通过这 4 个段寄存器来访问 4 个不同的段。如果段较多，在程序中可以通过对段寄存器的内容进行修改，实现对所有段的访问。

8086 对段寄存器的使用有约定的规则，如表 2.2 所示，在取指令时，下一条指令的段址由 CS 提供，偏移地址由 IP 提供；如果不特殊指明，访问数据时，段址由 DS 提供，堆栈操作的段址由 SS 提供。

另外，段寄存器在程序设计层面上也有很大的作用。前面说过，对存储单元的访问需要提供逻辑地址，即需要指定段和偏移地址。在编程时，如果需要通过段标识（段名）来指定段就太繁琐了；而程序运行时，段址总是由段寄存器提供的，所以，可以通过在指定段和段寄存器的关系后由段寄存器来标识段。如例 2.5 中，一个程序由数据段（段名，DATA）、

堆栈段（段名，STACK）、代码段（段名，CODE）组成，语句“assume cs:CODE, ds:DATA,ss:STACK”就指明，运行时 CS 中是 CODE 段的段址、DS 中是 DATA 段的段址、SS 中是 STACK 段的段址，这样就可用 CS 标识 CODE 段、DS 标识 DATA 段、SS 标识 STACK 段。再进一步，如果程序中的访存操作符合表 2.1 中的规则，就不需要指定段了，而只需要偏移地址。实际编程时，在绝大多数情况下逻辑地址只需要指定偏移地址。

表 2.2　段寄存器的默认使用规则

段寄存器	访存类型	使用规则
CS	指令	取指令
SS	堆栈	堆栈操作，用 SP 和 BP 作为基址访存
DS	局部数据	除堆栈和串处理中的目的串之外的数据访问
ES	串处理	串处理中的目的串

2.4　8086 的 I/O 组织

在微机系统中，外部设备和主机通过 I/O 接口交换信息，如图 2.12 所示。

在主机与外部设备的通信中，I/O 接口起到了转换和缓冲数据的作用，如图 2.12 所示，CPU 查询外部设备的状态就是 CPU 从 I/O 接口的状态寄存器读取状态数据，状态数据是由 I/O 接口负责从外部设备收集的；CPU 向外部设备发送命令就是向 I/O 接口的命令寄存器写命令数据，I/O 接口负责将命令数据转换成对外部设备的控制信号或外部设备的命令编码；同样，CPU 与外部设备的数据交换是通过 I/O 接口的数据寄存器进行的。这样，就好像是为 CPU 建立了与外部设备通信的一个“进出口”，至于“进出口”外面是什么与 CPU 无关。主机与外部设备通信的“进出口”称为 I/O 端口。

8086 将 I/O 端口进行独立编址，每个 I/O 端口拥有唯一的地址，又称为 I/O 端口号，构成了独立于存储空间的 I/O 端口空间，如图 2.13 所示。

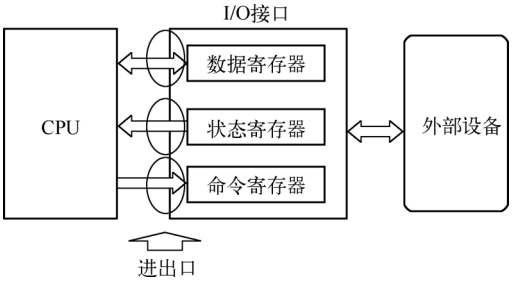

图 2.12　I/O 接口

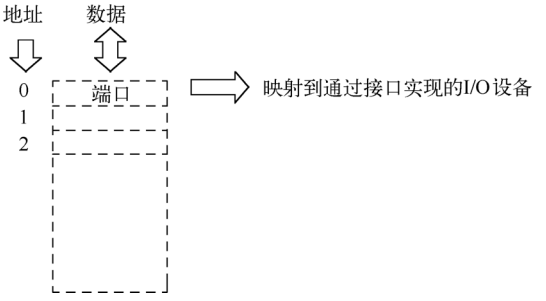

图 2.13　I/O 端口空间

8086 使用地址总线的低 16 位访问 I/O 端口，所以 I/O 端口号范围是从 0000H～FFFFH，即 I/O 端口空间大小是 64KB。

2.5　32 位微处理器

2.5.1　内部结构

我们以 80486 为例介绍 Intel 80x86 体系 32 位微处理器的逻辑结构，如图 2.14 所示。

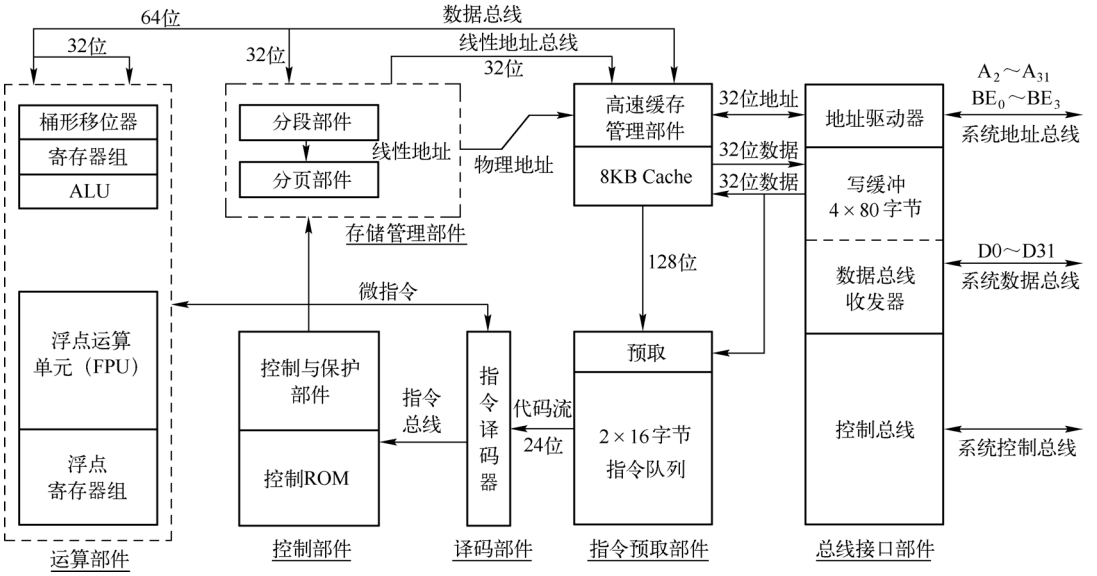

图 2.14　80486 的内部结构

80486 由以下 7 大部件组成。

（1）总线接口部件

总线接口部件完成指令预取、数据读/写等总线操作，对外提供包括地址、数据、控制的总线信号，对内借助 32 位地址总线和 32 位数据总线与 Cache 和指令预取部件进行通信。

（2）指令预取部件

指令预取部件包含一个 32 字节的指令队列。当指令队列不满且总线空闲时，指令预取部件可通过总线接口部件从存储器中读取指令存放在指令队列中。

（3）译码部件

译码部件从指令队列中读取指令并译码。若发现是转移指令或调用指令，可通知总线接口部件去新的目标地址取指令，以刷新指令队列。

（4）控制部件

控制部件负责解释来自译码部件中的指令字和控制 ROM 中的微代码，输出指令执行的控制信号。

（5）运算部件

运算部件包括算术逻辑运算部件 ALU 和浮点运算部件 FPU，完成整数运算和浮点运算。FPU 和 ALU 可并行执行。

（6）存储管理部件

存储管理部件包括分段管理部件和分页管理部件，分段管理部件对存储器实现分段管

理完成逻辑地址到线性地址的转换，分页管理部件对存储器实现分页管理完成线性地址到物理地址的转换。分页管理是可选的，如果不采用分页管理则线性地址就是物理地址。

（7）高速缓冲存（Cache）管理部件。

在 80486 CPU 中增加了片内 Cache，提高了指令和数据的访问速度。

在上述逻辑部件的支持下，80486 可按 6 级流水线方式工作。如表 2.3 所示，是 80486 的流水线工作的示例，虽然每条指令的执行仍然需要 6 个时钟周期，但从执行多条指令的整体时间上看指令的执行速度得到了提高。

表 2.3　80486 的流水线工作示例

指令步骤	时钟周期										
	1	2	3	4	5	6	7	8	9	10	11
取指令	I_1	I_2	I_3	I_4	I_5	I_6					
指令译码		I_1	I_2	I_3	I_4	I_5	I_6				
地址生成			I_1	I_2	I_3	I_4	I_5	I_6			
取操作数				I_1	I_2	I_3	I_4	I_5	I_6		
执行指令					I_1	I_2	I_3	I_4	I_5	I_6	
存储结果						I_1	I_2	I_3	I_4	I_5	I_6

2.5.2　寄存器结构

1．基本寄存器

Intel 80x86 体系 32 位微处理器的寄存器结构如图 2.15 所示。

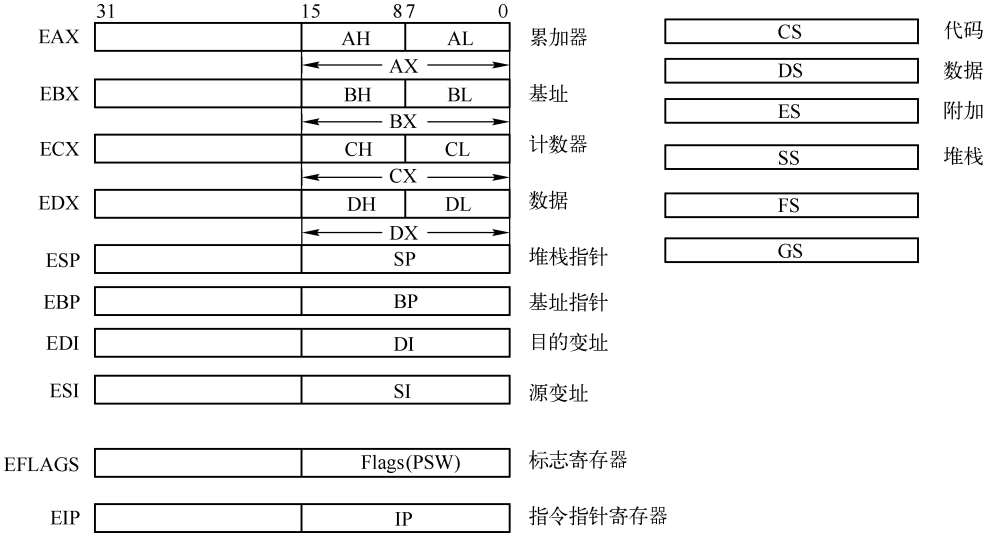

图 2.15　80x86 体系 32 位微处理器的寄存器结构

80x86 体系 32 位微处理器的寄存器结构是在 16 位微处理器基础上扩展的，主要在以下几方面进行扩展。

① 将 16 位通用寄存器扩展 16 位形成 32 位通用寄存器。将 AX 寄存器扩展 16 位形成

32 位的 EAX 寄存器，同理得到 EBX、ECX、EDX、ESP、EBP、EDI、ESI。在通用寄存器组内就有了 8 个 32 位寄存器，同时还保留了原来的 16 位寄存器和 8 位寄存器。

② 将指令指针寄存器 IP 扩展 16 位形成 32 位的 EIP 寄存器。

③ 段寄存器仍然是 16 位的，但增加了 FS 和 GS 两个段寄存器。

④ 将标志寄存器 Flags 扩展 16 位形成 32 位的 EFLAGS 标志寄存器，增加了新的标志，新增加的标志如表 2.4 所示。

表 2.4　新增加的标志

CPU	新 增 标 志	BIT 位置	说　　明
80286	IOPL	12、13	I/O 特权标志：I/O 特权标志用两位二进制位来表示，指定了要求执行 I/O 指令的特权级。只有当前的特权级别在数值上小于等于 IOPL 的值才能执行 I/O 指令
	NT	14	嵌套任务标志：NT=1 表示当前任务嵌套在另一任务内，引起任务转换
80386	RF	16	重启动标志：用来控制是否接受调试故障
	VM	17	虚拟 8086 方式标志：VM=1 表示工作在虚拟 8086 方式
80486	AC	18	对齐检查标志：若 AC=1，且 CR0 寄存器 AM=1，则进行字、双字或四字的对齐检查
Pentium	VIF	19	虚拟中断标志
	VIP	20	虚拟中断挂起标志
	ID	21	鉴别标志：Pentium CPU 是否支持 CPU ID 的指令

由图 2.15 可以看出，32 位微处理器完整保留了 16 位微处理器的寄存器结构，可保证 16 位架构的程序可以不经修改地运行在 32 位架构中。这就是 Intel 的 80x86 系列的前向兼容，是 Intel 商业成功的重要保证。另外，AL、BL、CL、DL 实际上是 8080（Intel 的 8 位 CPU）的寄存器结构。

2．系统寄存器

除了上述的基本寄存器外，32 位微处理器还扩充了以下寄存器：含有 4 个控制寄存器和 4 个地址寄存器的系统寄存器、8 个 32 位的调试寄存器和 5 个 32 位测试寄存器及浮点运算部件中的 13 个浮点寄存器。

系统控制寄存器 CR0、CR1、CR2、CR3 保存着全局性的、与任务无关的机器状态，以满足处理器的控制管理需求，其结构如图 2.16 所示。

图 2.16　系统控制寄存器的结构

CR0 包含整个系统的控制状态，共有 11 位，各个控制状态的作用如表 2.5 所示。

表 2.5　CR0 各个控制状态的作用

标　识	名　称	位　置	说　明
PG	允许分页	31	PG=1 时允许分页管理
CD	禁止 Cache	30	控制 CPU 内部的 Cache，当 NW=0 且 CD=0 时 Cache 使能
NW	非通写控制	29	

续表

标　识	名　称	位　置	说　明
AM	对齐屏蔽	18	AM=1 时允许对齐检查
WP	写保护	16	当 WP=1 时，对只读页面进行写操作会产生页故障
NE	数值异常中断控制	5	NE=1 时允许浮点指令错误触发异常中断
ET	处理机扩展类型	4	ET=1 表示协处理器为 80387
TS	任务转换	3	当进行任务转换时 TS=1，任务转换完毕 TS=0
EM	仿真协处理器	2	MP=0、EM=1 时表示正在软件仿真协处理器工作
MP	监视协处理器	1	MP=1 时表示协处理器在工作
PE	保护模式允许	0	PE=1 时 CPU 工作在保护模式，否则为实模式

CR1 保留未用。CR2 存放引起页故障的线性地址，只有在 PG=1 时，CR2 才有效。CR3 的高 20 位存放页目录的基地址，当 CR0 中的 PG=1 时有效；PCD 位控制是否对页目录进行高速缓存，PCD=0 时表示高速缓存；PWT 位为页级通写位，PWT=1 时表示片外 Cache 对页目录进行通写，否则回写。

系统地址寄存器包括全局描述符表寄存器（GDTR）、中断描述符表寄存器（IDTR）、局部描述符表寄存器（LDTR）和任务状态寄存器（TR）。

2.5.3　工作模式

80386 以后的 Intel 32 位微处理器有 3 种工作模式。

（1）实模式。实模式下的工作原理与 8086 相同，就相当于一个高速的 8086 处理器。

（2）保护模式。保护模式具有以下特点：

① 支持多任务；

② 支持存储器的分段管理和分页管理，易于实现虚拟存储系统；

③ 具有保护功能，包括存储器保护、任务特权级保护和任务之间的保护。

（3）虚拟 8086 模式。虚拟 8086 模式是为了在保护模式下执行 8086 程序而设置的。在虚拟 8086 模式下可以执行 16 位的软件，同时虚拟 8086 模式又可以以任务的形式与真正的 32 位任务进行切换，共享资源。

2.5.4　保护模式下的存储组织

在保护模式下，微处理器支持分段存储管理和可选的分页存储管理。如图 2.17 所示，分段存储管理完成由多个逻辑段构建的逻辑空间映射到线性空间的工作；程序设计面向逻辑空间进行；如果不使用分页存储管理，则线性空间就是物理空间；分页存储管理将线性空间按照一定大小（4KB）划分页（逻辑页），然后以页为单位在物理空间上进行分配（物理页）和回收，并实现逻辑页到物理页的映射；分页存储管理对使用者是透明的。

1．分段管理

保护模式下的存储器分段与 8086 的存储器分段原理是一样的，区别主要在于：第一，如图 2.18 所示，保护模式下的逻辑地址由 16 位的选择符和 32 位的偏移地址组成，16 位的选择符由段寄存器提供，32 位的偏移地址决定了段的最大空间是 2^{32}=4GB；第二，保护模

式下的分段是多任务环境下的分段，需要有越界检查、访问权限控制等保护措施，所以在地址转换时，通过段寄存器中选择符从描述符表中查询到的描述符中包含段基址、访问权限和界限值，如图 2.19 所示。

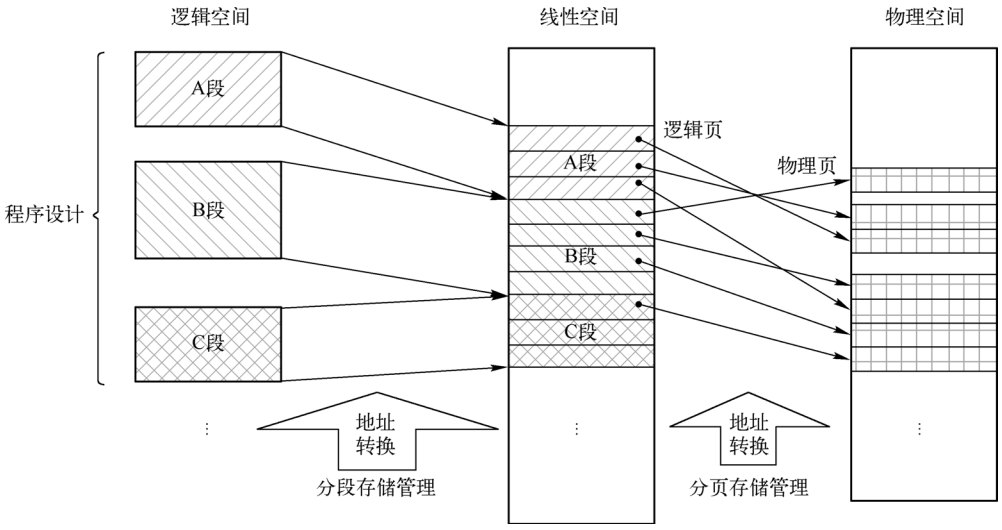

图 2.17　段页式存储管理

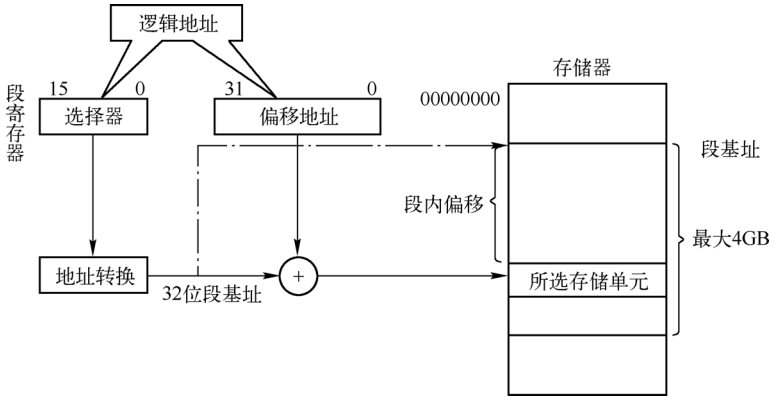

图 2.18　保护模式下的分段机制

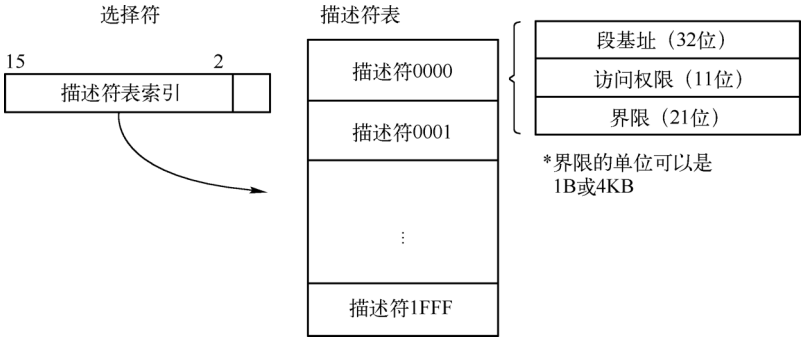

图 2.19　选择符、描述符和描述符表

图 2.19 中所示的描述符表就是分段存储管理的段表，它是分段存储管理的核心数据结构。在多任务环境下，将由多个任务共享的段组成的空间称为全局空间，全局空间由全局描述表 GDT 来管理。系统中断所使用的空间也属于全局空间，32 位微处理器架构为每个中断或异常都定义一个中断描述符，包含中断或异常服务程序的首地址等属性。中断描述符表 IDT 管理全部中断描述符。相对于全局空间，每个任务独自占用的空间称为局部空间。局部空间中任务私有的段由局部描述表 LDT 管理。另外局部空间还包括一个记录任务状态的特殊段 TSS。

为了保证存储管理的可用性和效率，80x86 体系的 32 位微处理器在内部设置了 4 个系统地址寄存器，包括：全局描述符表寄存器（GDTR）、中断描述符表寄存器（IDTR）、局部描述符表寄存器（LDTR）和任务状态寄存器（TR），如图 2.20 所示。由于系统只有一个 GDT 和一个 IDT，GDT 的基地址和大小直接由 GDTR 保存，IDT 的基地址和大小直接由 IDTR 保存。而每个任务都有自己的 LDT 和 TSS，所以系统中就存在多个 LDT 和 TSS。每个 LDT 和 TSS 都被看成一个段，也需要有描述符和选择符。LDTR 和 TR 保存 LDT、TSS 的选择符、基地址、大小和属性，当 LDT 或 TSS 的选择符被加载到 LDTR 或 TR 时，CPU 自动加载其基地址、大小和属性。

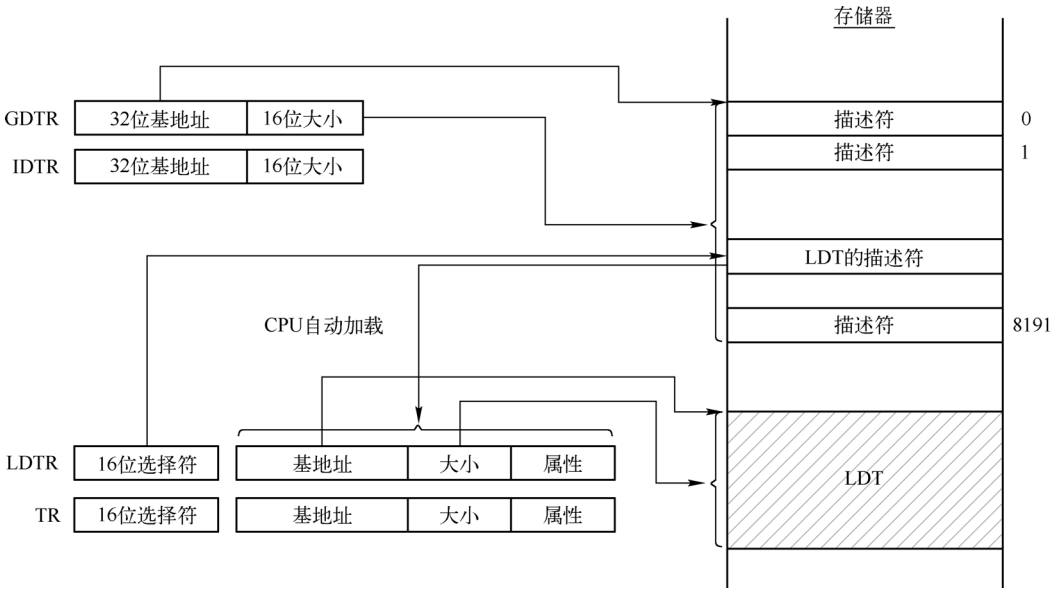

图 2.20　32 位微处理器的系统地址寄存器

2．分页管理

分页存储管理采用两级页表：页目录表和页表。一个表项 32 位，包括：页表或页的基地址、有效位、访问位、修改位和保护位等。32 位线性地址的高 10 位用于选择页目录表，所以页目录表包含 1024 个表项，4KB 大小。下一个 10 位用于选择页表，页表也是 4KB 大小，1024 个表项。最低 12 位是页内地址，页大小是 2^{12}=4KB。每个任务只有一个页目录表，首地址由 CR3 寄存器提供，最多 1024 个页表，最多 1024×1024=1M 个页。分页存储管理的地址转换过程如图 2.21 所示。

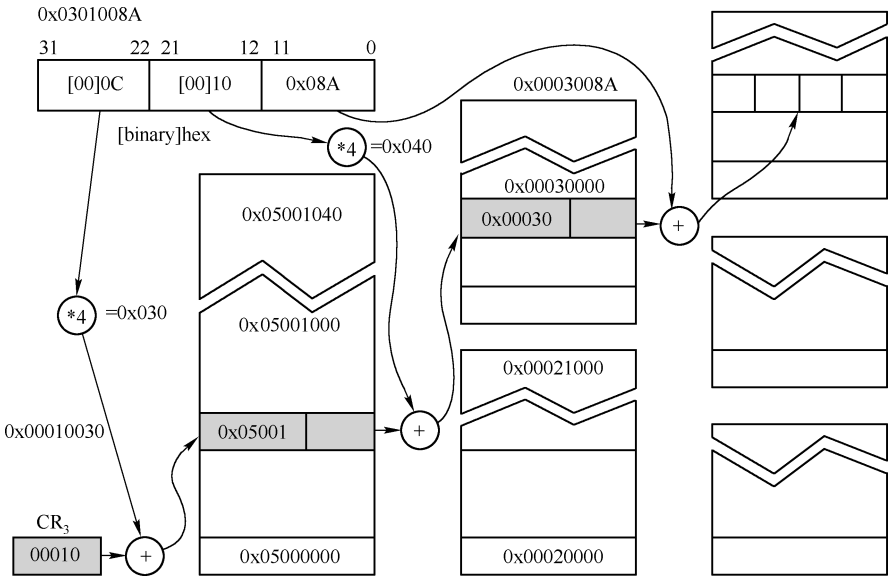

图 2.21　分页存储管理的地址转换

2.5.5　任务管理

在多任务环境下，处理器可以同时执行多个应用程序。一个应用程序的执行就是一个任务。一个任务包括两个部分：任务执行空间和任务状态段 TSS。TSS 为任务状态信息提供存储空间，如图 2.22 所示，任务状态信息主要包括以下内容：

任务状态段基本部分的格式

BIT31—BIT16	BIT15—BIT1	BIT0	Offset
0000000000000000	链接字段		0
ESP0			4
0000000000000000	SS0		8
ESP1			0CH
0000000000000000	SS1		10H
ESP2			14H
0000000000000000	SS2		18H
CR3			1CH
EIP			20H
EFLAGS			24H
EAX			28H
ECX			2CH
EDX			30H
EBX			34H
ESP			38H
EBP			3CH
ESI			40H
EDI			44H
0000000000000000	ES		48H
0000000000000000	CS		4CH
0000000000000000	SS		50H
0000000000000000	DS		54H
0000000000000000	FS		58H
0000000000000000	GS		5CH
0000000000000000	LDTR		60H
I/O许可位图偏移	0000000000000000	T	64H

图 2.22　TSS 的基本格式及主要信息

① 所有通用寄存器和段寄存器信息；
② 标志寄存器、程序指针 EIP、控制寄存器 CR3、任务寄存器和 LDTR 寄存器；
③ 段寄存器指定的任务当前执行空间；
④ I/O 映射位图基地址和 I/O 位图信息；
⑤ 特权级 0、1 和 2 的堆栈指针；
⑥ 链接至前一个任务的链指针。

在进行任务调度时，当前正在运行任务和调度任务之间会自动发生任务切换操作。在任务切换期间，当前运行任务的执行环境会被保存到它的 TSS 中并且暂停该任务的执行。新调度任务的上下文会被加载到处理器中，并且从加载的 EIP 指向的指令处开始执行新的任务。

习题 2

1．为何说 8086 CPU 是 16 位 CPU？
2．8086 CPU 由哪两个单元组成？其中，指令队列在哪个单元中，有何作用？
3．8086 CPU 中 8 位寄存器和 16 位寄存器是什么关系？
4．8086 CPU 中 IP 寄存器有何用途？
5．在标志寄存器中，用于反映运算结果属性的标志位有哪些？它们每一位所表示的含义是什么？
6．分别完成下面的 8 位运算，并说明各主要标志位的状态，以及结果是否产生溢出（提示：需要分为有符号数和无符号数两种情况）。
（1）90H + 3CH　　（2）3CH – 90H　　（3）7DH – 9CH
7．假设下列各组数值均分配在 10000H 开始的连续存储单元中，分别画出各组数值在内存中的存储形式。
（1）字节：2、‘2’、‘B’、‘b’和 0；
（2）字：12H、1004H、–1 和 0；
（3）双字：420H、12345678H 和 0。
8．什么是物理空间？8086 CPU 的物理地址多少位？其决定的物理空间有多大？
9．什么是逻辑空间？8086 CPU 的逻辑空间有何特点？其逻辑地址如何构成？
10．简述面向逻辑空间编程比面向物理空间编程的优势。
11．8086 CPU 是如何实现逻辑空间到物理空间的映射的？
12．简述段寄存器的作用。
13．已知当前(CS)=7F00H，(IP)=017CH，问下条指令的物理地址是什么？
14．已知当前(DS)=5C00H，(ES)=5D10H，而数据的物理地址是 63654H，若分别使用 DS 和 ES 段寄存器访问该数据，问偏移地址分别是多少？
15．什么是 I/O 端口？8086 CPU 的端口空间是多大？需要使用地址总线的多少位访问端口空间？
16．简述 Intel 的 32 位处理器如何对 16 位处理器的寄存器进行扩展，有何好处。
17．简述 32 位处理器的工作模式。
18．简述保护模式下存储管理的特点。