第3章 程序设计的基本技术

在汇编语言程序中，最常见的形式有顺序程序、分支程序、循环程序和子程序。这几种程序的设计方法是汇编程序设计的基础。本章将结合实例详细地介绍这些程序的设计技术以及第2章尚未讲述的指令。如前所述，本书有关汇编语言程序设计的讨论只限于DOS环境下(MASM 5.0)的实地址方式，而在实地址方式下，一个逻辑段的空间最大为64KB，因此在实地址方式下最好采用16位寻址的控制转移。本章介绍的控制转移类指令，包括转移指令、子程序的调用指令和返回指令在实现转移或调用时都要修改IP或EIP，在实地址方式下的EIP就是16位的IP，所以关于这两类指令就只限于修改IP的介绍。

3.1 顺序程序设计

顺序程序是最简单的程序，它的执行顺序和程序中指令的排列顺序完全一致。下面先介绍乘除法指令及BCD运算的调整指令。

3.1.1 乘除法指令

乘除法指令应该有无符号数乘除法指令和符号数乘除法指令之分。这是因为乘除法不同于加减法，无符号数的乘法和除法指令对符号数进行乘除运算不能得到正确的结果。如用无符号数的乘法运算做FFH乘以FFH结果为FE01H。把它们看作无符号数为255×255＝65 025(FE01H＝65 025)，其结果是正确的；若把它们看作符号数(一般情况下，都将符号数看作补码数)为(－1)×(－1)＝－511(FE01H＝－511)，显然是错误的。因此符号数必须用专用的乘除法指令。

1. 乘法指令MUL和符号整数乘法指令IMUL(signed integer multiply)

指令格式　MUL source
　　IMUL source

其中源操作数source可以是字节、字或者双字，与其对应的目的操作数是AL、AX或EAX。源操作数只能是寄存器和存储器，不能为立即数。在乘法指令之前必须将另一个乘数送AL(字节乘)、AX(字乘)或者EAX(双字乘)。乘法指令所执行的操作是AL、AX或者EAX乘source，乘积放回到AX、DX和AX或者EDX和EAX，如图3-1所示。

如用乘法指令实现例2.4(将AX中小于255大于0的3位BCD数转换为二进制数，存入字节变量SB中)的程序段如下：

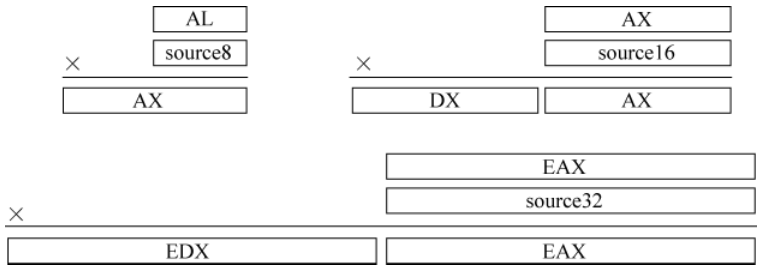

图 3-1　乘法指令的操作

```
MOV CH,10
MOV CL,4
MOV SB,AL          ; 暂存十位和个位
MOV AL,AH
MUL CH             ; 百位×10
MOV AH,SB
SHR AH,CL          ; 取十位
ADD AL,AH          ; 加十位
MUL CH             ; (百位×10+十位)×10
AND SB,0FH         ; 取个位
ADD SB,AL          ; (百位×10+十位)×10+个位
```

乘法指令对除 CF 和 OF 以外的状态标志位无定义(注意：无定义和不影响不同，无定义是指指令执行后这些状态标志位的状态不确定，而不影响则是指该指令的执行不影响状态标志位，因而状态标志应保持原状态不变)。对于 MUL 指令，如果乘积的高一半为 0(即字节操作时 AH＝0、字操作时 DX＝0 或双字操作时 EDX＝0)，则 CF 和 OF 均为 0；否则 CF 和 OF 均为 1。对于 IMUL 指令，如果乘积的高一半是低一半的符号扩展，则 CF 和 OF 均为 0，否则 CF 和 OF 均为 1。

除了 8086 微处理器外，符号整数乘法指令 IMUL 还有双操作数指令和三操作数指令，其格式及其功能如下。

```
IMUL REG, source              ; REG ← REG × source
IMUL REG, source, imm         ; REG ← source × imm
```

双操作数乘法指令的意义是用源操作数乘目的操作数，乘积存入目的操作数。目的操作数只能是 16 位和 32 位的寄存器，源操作数可以是寄存器和存储器，但其类型要与目的操作数一致。若目的操作数是 16 位的寄存器，则源操作数还可以是立即数。

三操作数乘法指令的意义是用源操作数乘立即数，乘积存入目的操作数。目的操作数只能是 16 位和 32 位的寄存器，源操作数可以是寄存器和存储器，但其类型要与目的操作数一致。

2. 除法指令 DIV 和符号整数除法指令 IDIV(singed integer divide)

指令格式　DIV source
　　IDIV source

其中源操作数 source 可以是字节、字或者双字，可为寄存器或存储器操作数，不能为立即数。目的操作数是 AX、DX 和 AX 或者 EDX 和 EAX。

除法指令所执行的操作是用指令中指定的源操作数 source 除 AX 中的 16 位二进制数或 DX 和 AX 中的 32 位二进制数或者 EDX 和 EAX 中的 64 位二进制数,被除数是 AX 还是 DX 和 AX 或者 EDX 和 EAX,由源操作数是字节还是字或者双字确定。商放入 AL、AX 或者 EAX,余数放入 AH、DX 或者 EDX,如图 3-2 所示。

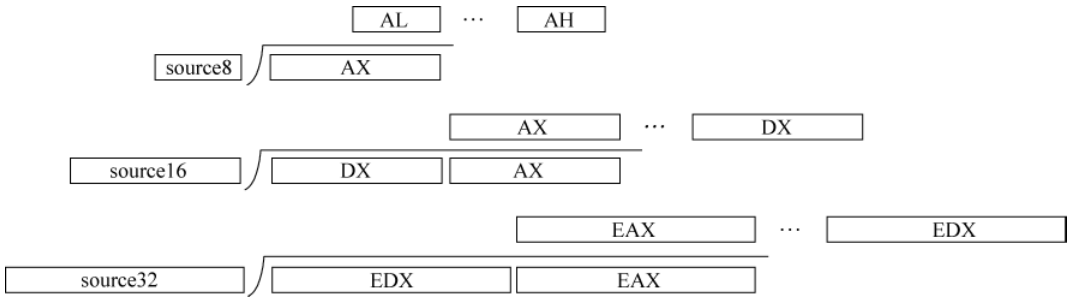

图 3-2　除法指令的操作

可用除法运算将二进制数转换为 BCD 数。如把 AL 中的 8 位无符号二进制数转换为 BCD 数放入 AX 中的程序段如下:

```
MOV CL,10
MOV AH,0              ; 8 位二进制数扩展为 16 位二进制数
DIV CL
MOV CH,AH             ; 暂存 BCD 数个位
MOV AH,0
DIV CL
MOV CL,4
SHL AH,CL             ; BCD 数十位移至高 4 位
OR CH,AH              ; BCD 数十位与个位拼合
MOV AH,0
MOV CL,10
DIV CL                ; AH 中的余数为 BCD 数百位
MOV AL,CH             ; BCD 数十位与个位送 AL
```

用除 10 取余法将 8 位二进制数 FFH 转换为 BCD 数 255H 的二进制运算如图 3-3 所示。

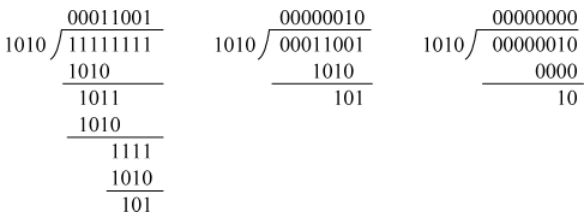

图 3-3　8 位二进制数 FFH 转换为 BCD 数 255H 的二进制运算

除法指令对所有的状态标志位均无定义。

3. 扩展指令

从除法指令的操作可知,要把一个 8 位二进制数除以一个 8 位二进制数,要有一个

16 位二进制数在 AX 中，只是把一个 8 位的被除数放入 AL 中是不行的，因为除法指令将把任何在 AH 中的数当作被除数的高 8 位。所以在做 8 位除以 8 位的除法之前先要把 8 位被除数扩展为 16 位，在做 16 位除以 16 位的除法之前要把 16 位被除数扩展为 32 位，在做 32 位除以 32 位的除法之前要把 32 位被除数扩展为 64 位，才能保证除法指令的正确操作。这种扩展对于无符号数除法是很容易办到的，只需将被除数的高半部清 0 即可。对符号整数除法就不能用将被除数的高半部清 0 来实现，而要通过扩展符号位来把被除数扩展。例如把－2 的 8 位形式 1111 1110 转换为 16 位形式 1111 1111 1111 1110，即要把高半部全部置 1(－2 的符号位)；而把＋3 的 8 位形式 0000 0011 转换成 16 位形式 0000 0000 0000 0011，却要把高半部全部置 0(＋3 的符号位)。

指令格式　CBW(convert byte to word)
CWD/CWDE(convert word to double word)
CDQ(convert double to quad)

将字节扩展为字指令 CBW 所执行的操作是把 AL 的最高位扩展到 AH 的所有位。将字扩展为双字指令 CWD 把 AX 的最高位扩展到 DX 的所有位，形成 DX 和 AX 中的双字；而将字扩展为双字指令 CWDE 把 AX 的最高位扩展到 EAX 的高 16 位，形成 EAX 中的双字。将字扩展为双字指令 CWDE 与符号位扩展传送指令功能相当，指令 CWDE 就等于指令 MOVS EAX,AX。将双字扩展为 4 字指令 CDQ 把 EAX 的最高位扩展到 EDX 的所有位，形成 EDX 和 EAX 中的 4 字。在做 8 位除以 8 位、16 位除以 16 位、32 位除以 32 位的除法之前，应先扩展 AL、AX 或 EAX 中的被除数。

例如，在数据段中，有一符号字数组变量 ARRAY，第 1 个字是被除数，第 2 个字是除数，接着存放商和余数，其程序段是：

```
MOV SI,OFFSET ARRAY
MOV AX,[SI]
CWD
IDIV WORD PTR 2[SI]
MOV 4[SI],AX
MOV 6[SI],DX
```

一般情况下，都将符号数看作补码数，扩展指令和符号整数除法指令仅对补码数适用。若特别指出该符号数为原码数，则其扩展和除法运算都要另编程序段实现。

3.1.2　BCD 数调整指令

第 2.3 节介绍的加减指令和本节介绍的乘除指令都是对二进制数进行操作。二进制数算术运算指令对 BCD 数进行运算，会得到一个非 BCD 数或不正确的 BCD 数。如

```
0000 0011B + 0000 1001B = 0000 1100B
0000 1001B + 0000 0111B = 0001 0000B
```

第一个结果是非 BCD 数；第二个结果是不正确的 BCD 数。其原因是 BCD 数向高位的进位是逢 10 进 1，而 4 位二进制数向高位进位是逢 16 进 1，中间相差 6。若再加上 6，就可以得到正确的 BCD 数：

```
0000 1100B + 0000 0110B = 0001 0010B
0001 0000B + 0000 0110B = 0001 0110B
```

8086/8088 对 BCD 数使用二进制数算术运算指令进行运算,然后执行一条能把结果转换成正确的 BCD 数的专用调整指令来处理 BCD 数的结果。

1. BCD 数加法调整指令 DAA(decimal adjust for add)和 AAA(ASCII adjust for add)

指令格式　DAA
　　AAA

DAA 指令的意义是将 AL 中的数当作两个压缩 BCD 数相加之和来进行调整,得到两位压缩 BCD 数。具体操作是,若(AL & 0FH) >9 或 AF=1,则 AL 加上 6;若(AL & 0F0H) >90H 或 CF=1,则 AL 加 60H。如

```
MOV AX,3456H
ADD AL,AH          ; AL = 8AH
DAA                ; AL = 90H
```

AAA 指令的意义是将 AL 中的数当作两个非压缩 BCD 数相加之和进行调整,得到正确的非压缩 BCD 数送 AX。具体操作是,若(AL & 0FH) >9 或 AF=1,则(AL+6) & 0FH 送 AL,AH 加 1 且 CF 置 1;否则 AL & 0FH 送 AL,AH 不变且 CF 保持 0 不变。应特别注意,AAA 指令执行前 AH 的值。如

```
MOV AX,0806H
ADD AL,AH          ; AX = 080EH
MOV AH,0
AAA                ; AX = 0104H
```

又如,若要将两个 BCD 数的 ASCII 码相加,得到和的 ASCII 码,可以直接用 ASCII 码相加,加后再调整。

```
MOV AL,35H         ; '5'
ADD AL,39H         ; '9',AL = 6EH
MOV AH,0
AAA                ; AX = 0104H
OR AX,3030H        ; AX = 3134H 即'14'
```

由调整指令所执行的具体操作可以看到,对结果进行调整时要用到进位标志和辅助进位标志,所以调整指令应紧跟在 BCD 数作为加数的加法指令之后。所谓"紧跟"是指在调整指令与加法指令之间不得有改变标志位的指令。

2. BCD 数减法调整指令 DAS(decimal adjust for subtract)和 AAS(ASCII adjust for subtract)

指令格式　DAS
　　AAS

DAS 指令的功能是将 AL 中的数当作两个压缩 BCD 数相减之差来进行调整,得到正确的压缩 BCD 数。具体操作是:若(AL & 0FH) >9 或 AF=1,则 AL 减 6,(AL & 0F0H) >90H 或 CF=1,则 AL 减 60H。如

```
MOV AX, 5634H
SUB AL,AH          ; AL = DEH,有借位
DAS                ; AL = 78H,保持借位即 134 - 56
```

AAS 指令的功能是将 AL 中的数当作两个非压缩 BCD 数相减之差进行调整得到正确的非压缩 BCD 数。具体操作是:若(AL & 0FH) >9 或 AF=1,则(AL-6) & 0FH 送

AL,AH 减 1；否则 AL & 0FH 送 AL,AH 不变。应特别注意,AAS 指令执行前 AH 的值。如

```
MOV AX,0806H
SUB AL,07H          ; AX = 08FFH
AAS                 ; AX = 0709H
```

3. 非压缩 BCD 数乘除法调整指令 AAM(ASCII adjust for multiply)和 AAD(ASCII adjust for divide)

压缩 BCD 数对乘除法的结果不能进行调整,故只有非压缩 BCD 数乘除法调整指令。

指令格式　AAM
　　AAD

AAM 指令的功能是将 AL 中的小于 64H 的二进制数进行调整,在 AX 中得到正确的非压缩 BCD 数。具体操作是 AL/0AH 送 AH,AL MOD 0AH 送 AL。如

```
MOV AL,63H
AAM                 ; AX = 0909H
```

AAD 指令的功能是将 AX 中的两位非压缩 BCD 数变换为二进制数。在做两位非压缩 BCD 数除以一位非压缩 BCD 数时,先将 AX 中的被除数调整为二进制数,然后用二进制除法指令 DIV 相除,保存 AH 中的余数后,再用 AAM 指令把商变回为非压缩的 BCD 数。如

```
MOV AX,0906H
MOV DL,06H
AAD                 ; AX = 0060H
DIV DL              ; AL = 10H、AH = 0
MOV DL,AH           ; 存余数
AAM                 ; AX = 0106H
```

应注意的是,除法的调整不同于加法、减法和乘法,它们的调整是在相应运算操作之后进行,而除法的调整在除法操作之前进行。

调整指令都隐含着 AX 或 AL,都在 AX 或 AL 中进行。下面举几个使用调整指令的例子。

例 3.1　已知字变量 W1 和 W2 分别存放着两个压缩 BCD 数,编写求两数之和,并将其和送到 SUM 字节变量中的程序。

此例应注意以下两个问题。

(1) 定义字变量 W1 和 W2 的 4 位数应为 BCD 数,其后要加 H,只有这样定义装入内存中的数据才是 4 位 BCD 数。

(2) BCD 数的加减运算只能做字节运算,不能做字运算。这是因为加减指令把操作数都当作二进制数进行运算,运算之后再用调整指令进行调整,而调整指令只对 AL 作为目的操作数的加减运算进行调整。

程序如下：

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
```

```
W1          DW  8931H
W2          DW  5678H
SUM         DB 3 DUP(0)
data        ends
code        segment
begin       proc far
            assume  ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AL,BYTE PTR W1          ; AL = 31H
            ADD AL,BYTE PTR W2          ; AL = A9H,CF = 0,AF = 0
            DAA                         ; AL = 09H,CF = 1
            MOV SUM,AL
            MOV AL,BYTE PTR W1 + 1      ; AL = 89H
            ADC AL,BYTE PTR W2 + 1      ; AL = E0H,CF = 0,AF = 1
            DAA                         ; AL = 46H,CF = 1
            MOV SUM + 1,AL
            MOV SUM + 2,0               ; 处理向万位的进位
            RCL SUM + 2,1               ; 也可用指令 ADC SUM + 2,0
            ret
begin       endp
code        ends
            end  begin
```

例 3.2 已知字变量 W1 和 W2 分别存放着两个非压缩 BCD 数,编写求两数之和,并将其和送到 SUM 字节变量中的程序。

定义字变量 W1 和 W2 的数应为两位非压缩 BCD 数,其后要加 H。程序如下:

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
W1          DW  0809H
W2          DW  0607H
SUM         DB 3 DUP(0)
data        ends
code        segment
begin       proc  far
            assume  ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AX,W1               ; AX = 0809H
            ADD AL,BYTE PTR W2      ; AL = 10H,AF = 1
            AAA                     ; AX = 0906H
            MOV SUM,AL
```

```
            MOV AL,AH
            ADD AL,BYTE PTR W2 + 1    ; AL = 0FH,AF = 0
            MOV AH,0
            AAA                       ; AL = 05H,AH = 01H
            MOV WORD PTR SUM + 1,AX
            ret
begin       endp
code        ends
            end  begin
```

例 3.3 字变量 W 和字节变量 B 分别存放着两个非压缩 BCD 数，编写求两数之积，并将它存储到 JJ 字节变量中的程序。

定义字变量 W 的数应为两位非压缩 BCD 数，其后要加 H。

由于是 BCD 数的乘法，所以只能用 AL 做被乘数，因此要做两次乘法。先将第一次乘法的部分积 0603H 存入 JJ＋1 和 JJ 两个单元(JJ＋1 存高 8 位 06H，JJ 存低 8 位 03H)，然后将两次乘法的部分积相加。第二次乘法的部分积 0207H(在 AX 中)与第一次乘法部分积相加，是第二次乘法部分积的低 8 位与第一次乘法的部分积的高 8 位相加，相加的进位加入第二次部分积的高 8 位中。由于这个加法也是非压缩 BCD 数的加法，故加后也要调整，调整后若产生进位，该进位直接加入 AH，由于此时 AH 的内容正是第二次乘法部分积的高 8 位，所以加法调整指令正好调整到位。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
W           DW 0307H
B           DB 9
JJ          DB 3 DUP(0)
data        ends
code        segment
begin       proc far
            assume ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AL,BYTE PTR W         ; AL = 07H
            MUL B                     ; AX = 003FH
            AAM                       ; AX = 0603H
            MOV WORD PTR JJ,AX
            MOV AL,BYTE PTR W + 1     ; AL = 03H
            MUL B                     ; AX = 001BH
            AAM                       ; AX = 0207H
            ADD AL,JJ + 1             ; 07H + 06H = 0DH,即 AL = 0DH
            AAA                       ; 调整后的进位,直接加入 AH,AX = 0303H
            MOV WORD PTR JJ + 1,AX
            ret
begin       endp
```

```
code        ends
            end begin
```


例 3.4 字变量 W 和字节变量 B 中分别存放着两个非压缩 BCD 数,编制程序求二者的商和余数,并分别存放到字变量 QUOT 和字节变量 REMA 中。

定义字变量 W 的数应为两位非压缩 BCD 数,其后要加 H。

由于是 BCD 数的除法,所以要先调整,因此先将 W 中的非压缩 BCD 数取到 AX 中,然后将 AX 中的非压缩 BCD 数调整为二进制数。二进制数的除法之后,又应用 AAM 指令将结果调整为非压缩 BCD 数。AAM 指令是将 AL 中的小于 100 的二进制数调整为非压缩 BCD 数,存入 AX 中,因此,调整前应将除法产生的余数存入 REMA 中。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
W           DW 0909H
B           DB 5
REMA        DB 0
QUOT        DW 0
data        ends
code        segment
begin       proc far
            assume ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AX,W
            AAD                    ; 0909H→63H
            DIV B                  ; 63H÷5 = 13H…4,AL = 13H,AH = 04H
            MOV REMA,AH
            AAM                    ; 13H→0109H
            MOV QUOT,AX
            ret
begin       endp
code        ends
            end begin
```

3.1.3 顺序程序设计举例

例 3.5 从键盘上输入 0～9 中任一自然数 N,将其立方值送显示器显示。

求一个数的立方值可以用乘法运算实现,也可以用查表法实现。查表法运算速度比较快,是常用的计算方法。因只要送显示,故将 0～9 的立方值的 ASCII 码按顺序造一立方表。立方值最大值为 729,需三个单元存放它的 ASCII 码,表的每项的单元数相同,再在每项之后加一个'$',所以立方表的每项均占 4 个字节。根据这种存放规律可推知,表的偏移首地址与自然数 N 的 4 倍之和,正是 N 的立方值和 $ 的 ASCII 码的存放单元的偏移首地址。

用查表法编制的程序如下：

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
INPUT       DB 'PLEASE INPUT N(0-9): $'
LFB         DB '0$   1$   8$  27$  64$125$216$343$512$729$'
N           DB 0
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV DX,OFFSET INPUT          ; 显示提示信息
            MOV AH,9
            INT 21H
            MOV AH,1                     ; 输入并回显 N(1 号功能调用)
            INT 21H
            MOV N,AL
            MOV AH,2                     ; 换行(2 号功能调用)
            MOV DL,0AH
            INT 21H
            MOV DL,N
            MOV DL,0FH                   ; 将'N'转换为 N
            MOV CL,2                     ; 将 N 乘以 4
            SHL DL,CL
            MOV DH,0                     ; 8 位 4N 扩展为 16 位
            ADD DX,OFFSET LFB            ; 4N + 表的偏移地址
            MOV AH,9
            INT 21H
            ret
start       endp
code        ends
            end start
```

例 3.6　编写两个 32 位无符号数的乘法程序。

使用 32 位指令编写的程序如下：

```
.386
stack       segment stack USE16'stack'
            dw 32 dup (0)
stack       ends
data        segment USE16
AB          DD 12345678H
CD          DD 12233445H
ABCD        DD 2 DUP(0)
```

```
data        ends
code        segment USE16
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV EAX,AB
            MUL CD
            MOV ABCD,EAX
            MOV ABCD + 4,EDX
            ret
start       endp
code        ends
            end start
```

若用 16 位指令编写该程序就要用 16 位乘法指令做四次乘法,然后把部分积相加,如图 3-4 所示,相应的程序如下。

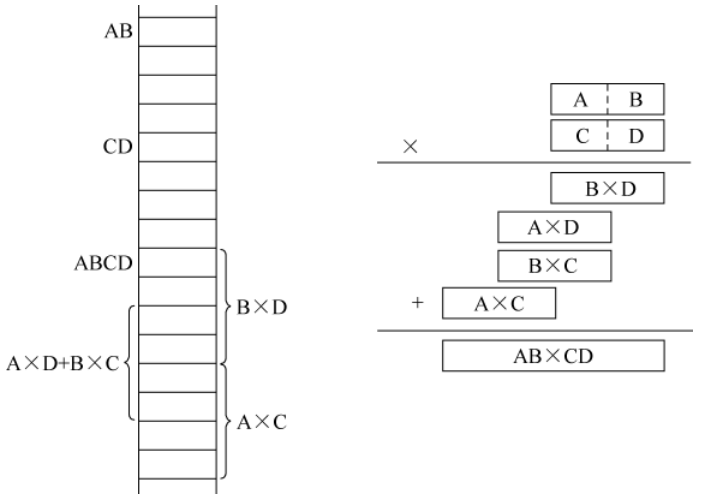

图 3-4　32 位无符号数的乘法

```
stack       segment stack 'stack'
            dw 32 dup (0)
stack       ends
data        segment
AB          DD 12345678H
CD          DD 12233445H
ABCD        DD 2 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
```