目录

第 16 章 SSE 程序设计

多媒体扩展（Multi-Media Extension，MMX）技术从 Pentium Ⅱ 开始引入到 IA-32 结构，首次在 x86 平台上实现了单指令多数据流（Single Instruction Multiple Data，SIMD）的增强功能。在 Pentium Ⅲ 中，引入了流式 SIMD 扩展（Streaming SIMD Extensions，SSE），进一步提高视频/图像处理、语音识别、音频合成、电话和视频会议、二维/三维图形处理等等方面的性能。在此之后出现了 SSE2、SSE3、SSSE3 等。本章介绍 x86-SSE 技术的基本概念、运行环境、指令系统，给出了 SSE 编程示例，以及用 C 语言开发 SSE 程序的示例。

16．1 SSE 技术简介

1、SSE 概览

流式 SIMD 扩展的首个版本称为 SSE，它增加了 8 个 128 位宽的寄存器 xmm0～xmm7。SSE 保留了使用 MMX 的 64 位寄存器对于组合整数进行运算，同时增加了单精度浮点数打包运算的指令，也增加了标量单精度浮点数运算指令。所谓标量单精度浮点运算就是用 XMM 寄存器的低位双字对一个单精度浮点数进行运算，图 16.1 给出了标量单精度浮点运算的示意图。从图中可以看到目的操作数的高位保持不变。

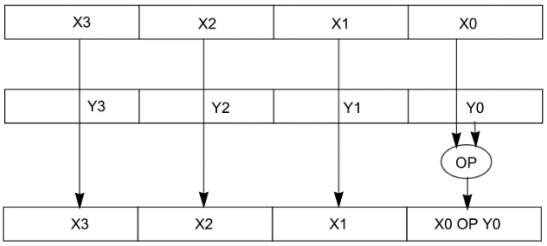

图 16.1 标量单精度浮点数运算示意图

在奔腾（Pentium）Ⅳ处理器中，对 SSE 进行了升级，称之为 SSE2。SSE2 支持 128 位的打包整型数据计算，能够一次打包 16 个字节、8 个字、4 个双字、2 个四字的整型数据。对于整数（字节、字、双字、四字）组合运算，运算模式与 MMX 中采用的模式是一样的，有环绕、有符号饱和和无符号饱和模式，它的整数运算指令保留了 MMX 中指令，例如 paddusb，同样表示无符号字节组合整型加法运算，同时也增加了一些新的指令，如 paddq，实现双字组合整型的环绕加法。对于相同的指令助记符，在机器编码上采用了用指令前缀（见第 4.9 节）来区分的方法。例如 “paddusb mm0，mm1” 与 “paddusb xmm0，xmm1”的机器码分别为“0F DC C1”和“66 0F DC C1”。

SSE2 除了增加整型数组合运算外，还将 SSE 中的单精度浮点数运算扩展为双精度浮点数运算，增加了更多的打包运算指令。在之后的英特尔酷睿双核处理器中，出现了 SSE3 和 SSSE3（Supplemental SSE3）、SSE4 等新的技术。本章中第 2 节介绍了 SSE 指令，在第 3 节介绍了一些 SSE2 和后续版本中指令。更详细的介绍可以查阅《Intel® 64 and IA-32

Architectures Software Developer’s Manual》。

2、SSE 中的数据寄存器

支持 SSEX 技术的处理器中有 8 个 128 位的寄存器，名为 xmm0～xmm7。这些寄存器与 CPU 中的 32 位寄存器 eax 等一样，在指令中可以直接使用这些寄存器的名字，即采用寄存器寻址方式访问。但是这些寄存器不能用于寄存器间接寻址、变址寻址和基址加变址寻址，即不能用于寻址内存中的操作数。

与 MMX 寄存器不同，XMM 寄存器完全独立于 x87 FPU 寄存器。在 SSE 指令与 FPU 指令之间切换时，不需要用 EMMS 指令改变状态。在用 VS2019 调试程序时，在寄存器窗口选择显示 SSE 寄存器后，可看到 xmm0～xmm7 中的值，同时，也可以看到 xmm00、xmm01、xmm02、xmm03 等，它们中的值为一个单精度浮点数。单个显示浮点数的目的是为了方便程序调试，在程序中不能使用 xmm00 之类的符号。(xmm03 … xmm00)组合为 xmm0，其他 XMM 寄存器是类似的。

3、SSE 中的控制和状态寄存器

SSE 执行环境中还有一个 32 位的控制和状态寄存器 MXCSR（Control and Status Register)。它中间的控制标志用来指定浮点运算和异常处理的方式，与 x87 FPU 中的控制寄存器(CTRL)的作用类似。MXCSR 中的状态标志用于存放浮点运算的结果状态，与 x87 FPU 中的状态寄存器（STAT）的作用类似。图 16.2 给出了 MXCSR 中各标志位的位置。

31 16	15	14 13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留（Reserved）	FTZ	RC	PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE

图 16.2 MXCSR 控制和状态寄存器

各标志位的含义如下。

FTZ （Flush to Zero)：结果强制冲刷为 0，当该位为 1，且发生了下溢错误且屏蔽了溢异常

RC ：舍入控制，00：四舍五入；01：地板，10：天花板；11：截断

PM - IM ： 1，屏蔽对应的异常

PE – IE ： 1 发生了异常， 参见 x87 FPU 的状态寄存器的介绍。

DAZ （Denormals-Are-Zeros)：1，将非规格化操作数强制转化为 0

与 x87 FPU 的状态寄存器一样，当发生异常后，处理器不会自动清除 MXCSR 的状态标志，必须写指令进行复位。当执行指令时，出现了异常状态，要看屏蔽控制标志是否为 0，在为 0，即非屏蔽控制下，进行异常处理。

16．2 SSE 指令简介

SSE Extensions（Streaming SIMD Extensions，单指令多数据流扩展)， 被 Pentium Ⅲ, Pentium Ⅲ Xeon, Pentium 4, Intel Xeon, Pentium M, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo, Intel Atom 等处理器所支持。

SSE 指令可以分为四类：组合和标量单精度浮点指令、64 位 SIMD 整数指令、状态管理指令、其他指令（Cache 控制、预取、内存排序）。

16.2.1 组合和标量单精度浮点指令

标量单精度浮点和组合单精度浮点指令可以分为：数据传送、算术运算、比较运算、逻辑运算、重排和解组、转换等类别。为了便于记忆指令的助记符，下面给出了助记符的英文全称，并将其中的 Single-precision floating-point 简记为 S.p.f.p。一般的指令中都有源操作数 ops 地址和目的操作数地址 opd，在指令中未列出，但从功能描述上不难想象对 ops 和 opd 的要求，如是寄存器还是内存地址等。

1、数据传送

① movss ： MOVe Scalar S.p.f.p 标量单精度浮点数传送

(ops）中的一个单精度浮点数 ->opd。opd、ops 中至少有一个 XMM 寄存器，ops 不能为立即数。若 opd 为一个 XMM 寄存器，则该寄存器的最低 4 个字节为传送的值，其他字节为 0。

② movaps ： MOVe Aligned Packed S.p.f.p 对齐的组合单精度浮点数传送

4 个单精度浮点组合数据传送（ops）->opd。对于内存地址必须对齐到 16 字节边界（即地址被 16 整除），否则出现异常。

③ movups ： MOVe Unaligned Packed S.p.f.p

功能与 MOVAPS 相同，只是无内存地址必须对齐到 16 字节边界的要求。

④ movlps ： MOVe Low Packed S.p.f.p

内存与 XMM 寄存器的低 4 字之间传送 2 个打包的单精度浮点，高 4 字不变。

⑤ movhps ： MOVe High Packed S.p.f.p

内存与 XMM 寄存器的高 4 字之间传送 2 个打包的单精度浮点，低 4 字不变。

⑥ movlhps ： MOVe Packed S.p.f.p Low to High

将源 XMM 寄存器中的低 4 字传送到目的 XMM 寄存器中的高 4 字。

⑦ movhlps ： MOVe Packed S.p.f.p High to Low

将源 XMM 寄存器中的高 4 字传送到目的 XMM 寄存器中的低 4 字。

⑧ movmskps ： MOVe Packed S.p.f.p MaSK

将 XMM 寄存器中的 4 个单精度浮点数的符号位传送到一个通用寄存器的低 4 位中，可用于后续指令的分支条件。

2、算术运算

算术运算包括对标量和组合单精度浮点数的加、减、乘、除。

① addss ： ADD Scalar S.p.f.p 标量单精度浮点数加法

② subss ： SUBtract Scalar S.p.f.p 标量单精度浮点数减法

③ mulss ： MULtiply Scalar S.p.f.p 标量单精度浮点数乘法

④ divss ： DIVide Scalar S.p.f.p 标量单精度浮点数除法

⑤ addps ： ADD Packed S.p.f.p 组合单精度浮点数加法

⑥ subps ： SUBtract Packed S.p.f.p 组合单精度浮点数减法

⑦ mulps ： MULtiply Packed S.p.f.p 组合单精度浮点数乘法

⑧ divps ： DIVide Packed S.p.f.p 组合单精度浮点数除法

除了一般的加、减、乘、除外，还有一些特殊的算术运算如下。

① sqrtss/sqrtps ： compute SQuare RooT of Scalar（Packed） S.p.f.p

计算标量（组合）单精度浮点数的平方根

② rcpss/rcpps ： compute ReCiProcal of Scalar（Packed） S.p.f.p

计算标量（组合）单精度浮点数的倒数

③ rsqrtss/rsqrtps ： Peciprocal of SQuare RooT of Scalar（Packed） S.p.f.p

计算标量（组合）单精度浮点数的平方根的倒数

④ maxss/maxps ：return MAXimum of Scalar（Packed） S.p.f.p

比较标量（组合）单精度浮点数的大小得到大的浮点数

⑤ minss/minps ：return MINimum of Scalar（Packed） S.p.f.p

比较标量（组合）单精度浮点数的大小得到小的浮点数

3、比较运算

① cmpss/ cmpps xmm1, xmm2/m128, imm8

按照立即是 imm8 指定的方式比较 xmm1 和 xmm2/m128 的关系是否成立，若成立，则目的操作数中的对应元素被置为全 1（即 FFFFFFFF），否则被置为全 0。

imm8=0 ,判断是否相等，指令等价于伪指令 cmpeqss/cmpeqps xmm1,xmm2/m128

imm8 =1，小于，等同 CMPLTSS/CMPLTPS

imm8 =2-7,依次表示小于等于（LE）、无序（UNORD）、不相等（NEQ）、不小于（NLT）、不小于等于（NLE）、有序（ORD），与 imm8 为 0/1 一样，有对应的伪指令。

② comiss / ucomiss ：（Unordered） Compare Scalar S.p.f.p and set EFLAGS

根据比较结果，设置 EFLAGS 中的 ZF、PF、CF，以确定两个浮点数之间的大于、小于、相等或者无序关系。两条指令的差别是对 QNaN 操作数是否产生一个非法操作的异常标志，即控制和状态寄存器中 IM 是否置 1。

4、逻辑运算

逻辑运算指令有 andps、andnps、orps、xorps 它们分别对源操作数和目的操作数进行按位的逻辑与、与非、逻辑或、逻辑异或操作。

5、转换 （Conversion Instructions）

转换指令实现从一种数据类型转换为另一种类型的功能。

① cvtpi2ps ：ConVerT Packed doubleword Integers to Packed S.p.f.p

将成组双字整型数转换为成组浮点数

② cvtsi2ss ： ConVerT a Signed doubleword Integer to Scalar S.p.f.p

将单个双字整型数转换为标量单精度浮点数

指令 cvtps2pi、cvtss2si 执行浮点数向双字整型数转换的功能。

例如：sf1 real4 15.67

执行 cvtss2si eax，sf1 ；当舍入控制 RC 为 00 时，（eax）=10H。

6、重排和解组（Shuffle and Unpack Instructions）

① shufps xmm1, xmm2/m128, imm8 ： 按指定方式重排数据

将 xmm1 中的 4 个浮点和 xmm2/m128 中的 4 个浮点数“洗牌”到一个 xmm1 中。imm8 是一个字节的立即数，控制洗牌的方法。假设 xmm1 为{x3,x2,x1,x0},xmm2 为{y3,y2,y1,y0}，将 imm8 看成一个 4 进制数 p3 p2 p1 p0，换句话说 pi 是 2 个二进制位对应 0-3，则重排的结果为 xmm1 = {y(p3) y(p2) x(p1) x(p0)}。

② unpckhps ： UNPpaCK and interleave High Packed S.p.f.p

设目的操作数为 {x3,x2,x1,x0}，源操作数为{y3,y2,y1,y0}，执行该指令后目的操作数为 {y3,x3,y2,x2}。

③ unpcklps ： UNPpaCK and interleave Low Packed S.p.f.p

设目的操作数为 {x3,x2,x1,x0}，源操作数为{y3,y2,y1,y0}，执行该指令后目的操作数为 {y1,x1,y0,x0}。

16.2.2 SSE 64 位 SIMD 整数指令

SSE 64 位整数运算采用的 64 位 MMX 寄存器和 64 位的内存操作数，它保留了第 15 章中介绍的指令，同时增加了一些新的指令。

① pavgb/pavgw ： compute AVeraGe of Packed unsigned Byte(Word) integers

计算对应整型数的平均值，((ops)+(opd)) /2->opd

② pextrw reg32, mm, imm8 ： EXTRact Word

从 MMX 寄存器中选择一个字数据传送给一个 32 位的通用寄存器，imm8 指明是送哪一个字，32 位通用寄存器的高 16 位置 0。

③ pinsrw mm, reg32/m32, imm8：INSeRt Word

将通用寄存器低 16 位或者内存中的一个字传送到 MMX 寄存器的指定位置，imm8 指明字的位置。

④ pmaxub/pminub ： MAXimum （MINimum） of Packed Unsigned Byte integers

将对应的两个无符号字节整型数中的大者（小者）送目的操作数的对应位置。

⑤ pmaxsw/ pminsw：MAXimum （MINimum） of Packed Signed Word integers

将对应的两个有符号字整型数中的大者（小者）送目的操作数的对应位置。

⑥ pmovmskb reg32, mm：MOVe Byte MaSK

将寄存器 MM 中各字节的最高位传送到 32 位通用寄存器的低 8 位。

⑦ pmulhuw ： MULtiply Packed Unsigned Word integers and store High result
无符号整型字数据相乘，保留结果的高位。

⑧ psadbw ： compute Sum of Absolute Differences
计算对应的无符号整型字节的差值的绝对值，然后将这些差值(绝对值)相加，将和存放在目的操作数的低字中。

⑨ pshufw mm1,mm2/m32, imm8：SHUFfle Packed Word integers
按指定的顺序对于源操作数中的字数据进行重排，结果放在目的操作数中。imm8 为一个字节立即数，每 2 位二进制对应一个位置编号。

16.2.3 MXCSR 状态管理指令

① ldmxcsr ： LoaD the state of the MXCSR
从内存中取一个双字送给 MXCSR.

② stmxcsr ： STore the state of the MXCSR
将 MXCSR 的内容存放到一个内存单元

16.2.4 缓存控制指令

SSE 中增加了缓存（Cache）控制指令，赋予了程序员对缓存的控制能力。缓存控制指令有两类。一类是数据据预存取(Prefetch)指令，能够增加从主存到缓存的数据流；另一类是内存流优化处理(Memory Streaming)指令，能够增加从处理器到主存的数据流。

数据预存取指令完成将要使用的数据预先从内存中取出存入缓存。这样处理器可以更快地获取信息，从而改进应用性能。缓存是层次结构，在预取数据时可指定缓存到哪一级。内存流优化处理指令允许应用越过缓存直接访问主存。通常情况下，处理器写出的数据都将暂时存储在缓存中以备处理器稍后使用。如果处理器不再使用它，数据最终将被移至主存。对于多媒体应用来说，很多数据在近期不会再使用，不必要放在 Cache 中，从而提高了缓存的利用率。

缓冲控制指令有以下几条：

① movntq ： store Quadword using Non-Temporal hint

② movntps ： store Packed S.p.f. using Non-Temporal hint

③ maskmovq ： store selected bytes of quadword

④ prefetcht0/prefetcht1/prefetcht2/prefetchnta ：Temporal data—fetch data into levels of cache hierarchy

16．3 SSE2 及后续版本的指令简介

在 SSE 之后，出现了 SSE2、SSE3、SSE4 等版本，不断的增强了多媒体数据的处理能力。下面列出了一些指令，不再区分不同的版本。

SSE 指令可以分为四类：组合和标量单精度浮点指令、64 位 SIMD 整数指令、状态管理指令、其他指令（Cache 控制、预取、内存排序）。

16.3.1 组合和标量双精度浮点指令

标量双精度浮点和组合双精度浮点指令可以分为：数据传送、算术运算、比较运算、逻辑运算、重排和解组、转换等类别。为了便于记忆指令的助记符，下面给出了助记符的英文全称，并将其中的 Double-precision floating-point 简记为 D.p.f.p。由于 SSE2 中的双精度浮点指令与 SSE 中的单精度浮点指令功能非常相似，本文不再一一解释。

1、数据传送

双精度浮点传送指令有：movsd、movapd、movupd、movlpd、movhpd、movmskpd，它们分别与单精度浮点传送指令 movss、movaps、movups、movlps、movhps、movmskps 对应。

2、算术运算

与单精度浮点运算对应有双精度浮点运算。标量双精度浮点数运算指令为：addsd、subsd、mulsd、divsd；它们分别与标量单精度浮点数运算指令 addss、subss、mulss、divss 对应。组合双精度浮点数运算指令为：addpd、subpd、mulpd、divpd，它们分别与组合单精度浮点数运算指令 addps、subps、mulps、divps 对应。

对于其他类指令，类似的有：sqrtsd/sqrtpd、maxsd/ maxpd、minsd/ minpd。

3、比较运算

比较指令也将单精度浮点数比较扩展到双精度浮点数比较，指令有：cmpsd/ cmppd、comisd / ucomisd。

4、逻辑运算

逻辑运算指令有 andpd、andnpd、orpd、xorpd 它们分别对源操作数和目的操作数进行按位的逻辑与、与非、逻辑或、逻辑异或操作。

5、转换 （Conversion Instructions）

转换指令实现从一种数据类型转换为另一种类型的功能。由于增加了双精度浮点数类型，因而有单、双精度浮点数之间的转换；双精度浮点数与双字整型之间的转换；单精度浮点数与双字整型之间的转换。不同类型数据之间的转换指令如图 16.3 所示。

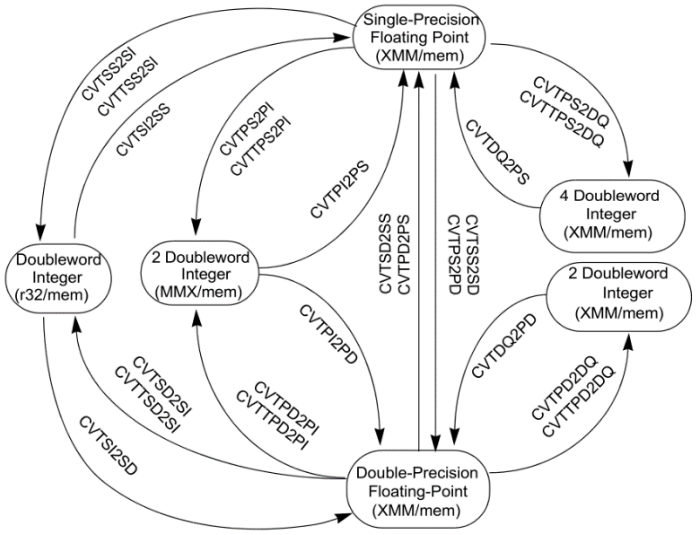

图 16.3 SSE 和 SSE2 的转换指令

6、重排和解组（Shuffle and Unpack Instructions）

重排与解组指令有：shufpd 、unpckhpd 、unpcklpd，它们分别与 shufps、unpckhps、unpcklps 的功能类似。

16.3.2 64 位和 128 位整数指令

组合整数传送指令有：movdqa、movdqu、movq2dq、movdq2q。

组合整数的算术运算有:paddq/psubq、pmulld、pmuludq/pmuldq、pmaddubsw、pmulhrsw、pminub/pminuw/pminud 、 pminsb/pminsw/pminsd 、 pmaxub/pmaxuw/pmaxud 、 pmaxsb/pmaxsw/pmaxsd 、 pabsb/pabsw/pabsd 、 psignb/psignw/psignd 、 phaddw/phaddd 、phsubw/phsubd、phaddsw/phsubsw。除了这些在 SSE2 及后续版本新增的指令外中，还可以使用 MMX 中算术运算指令,只是源操作数和目的操作数是 128 位的。例如,paddb/paddw/paddd 等等。

组合整数的比较指令有：pcmpeqb/pcmpeqw/ pcmpeqd/ pcmpeqq、pcmpgtb/ pcmpgtw / pcmpgtd/ pcmpgtq。

组合整数的比较指令有：packuswb/packusdw、pmovsxbw/ pmovsxbd/ pmovsxbq、pmovsxwd/ pmovsxwq、pmovsxdq、pmovzxbw/ pmovzxbd/ pmovzxbq、pmovzxwd/ pmovzxwq、pmovzxdq。

组合整数的重排和解组指令有：pshufd、pshuflw、pshufhw、punpcklqdq、punpckhqdq。

组合整数插入和提取指令有：pinsrb/pinsrw/pinsrd、pextrb/pextrw/pextrd。

组合整数移位指令有：pslldq、psrldq。

文本字符串处理指令有：pcmpestri、pcmpestrm、pcmpistri、pcmpistrm。

16．4 SSE 编程示例

1、C 语言程序示例

下面给出了一个简单的 C 语言程序，实现两个浮点数相加，然后显示结果。

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    float x, y, z;
    x = 3.14;
    y = 5.701;
    z = x + y;
    printf("%f\n", z);
    return 0;
}
```

该程序的反汇编程序如下。。

```
    x = 3.14;
00E61828  movss       xmm0,dword ptr [__real@4048f5c3 (0E67B34h)]
00E61830  movss       dword ptr [x],xmm0
    y = 5.701;
00E61835  movss       xmm0,dword ptr [__real@40b66e98 (0E67B38h)]
00E6183D  movss       dword ptr [y],xmm0
    z = x + y;
00E61842  movss       xmm0,dword ptr [x]
00E61847  addss       xmm0,dword ptr [y]
00E6184C  movss       dword ptr [z],xmm0
    printf("%f\n", z);
00E61851  cvtss2sd    xmm0,dword ptr [z]
00E61856  sub         esp,8
00E61859  movsd       mmword ptr [esp],xmm0
00E6185E  push        offset string "%f\n" (0E67B30h)
00E61863  call        _printf (0E61046h)
00E61868  add         esp,0Ch
```

注意，在编译时要设置使用的指令集。在“项目属性-> C/C++ ->代码生成->启用增强指令集”中选择“未设置”即可。

2、汇编语言程序示例

```
.XMM    ;处理器选择伪指令,支持 SSE、SSE2、SSE3 指令集
.model flat, stdcall
  ExitProcess proto stdcall :dword
  includelib  kernel32.lib
  printf      proto c :ptr sbyte, :vararg
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
.data
  lpFmt db  "%f",0ah, 0dh, 0
  x  real4  3.14
  y  real4  5.701
  z  real4  0.0
.stack 200
.code
 main  proc  c
   movss     xmm0, z
   addss     xmm0, y
   movss     z,  xmm0
   cvtss2sd  xmm0, z
   sub    esp,  8
   movsd  mmword ptr [esp], xmm0
   invoke printf, offset lpFmt
   invoke ExitProcess, 0
main  endp
end
```

若将 z 的定义改为 z real8 0.0，程序中的片段可进一步简化：

```
   movss     xmm0, x
   addss     xmm0, y
   cvtss2sd  xmm0,xmm0
   movsd     z, xmm0
   invoke printf,offset lpFmt, z
```

当然，对于第 15 章中用 MMX 实现的矩阵运算的例子也可以该为用 SSE 指令来实现，其运行效率得到进一步的提高。

16．5 用 C 语言编写 SSE 应用程序

在了解用 SSE 指令可以提高大数据（矩阵、向量）运算速度后，除了用汇编语言编写 SSE 应用程序外，同样可以用 C 语言编写类似的程序以提高计算速度。

完整的程序清单如下。

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <conio.h>
#include <emmintrin.h>
#define LEN 100000 // 数组大小
int main() {
    clock_t stTime, edTime;
    int   i, j;
    _declspec(align(16)) unsigned short  a[LEN];
    _declspec(align(16)) unsigned short  b[LEN];
    _declspec(align(16)) unsigned short  c[LEN];
    __m128i  *pa;
    __m128i  *pb;
    __m128i  *pc;
    int LEN8;
    srand(time(NULL));
    for (i = 0; i < LEN; i++) {
        a[i] = rand();
        b[i] = rand();
    }
    stTime = clock();
    for (j = 0; j < 1000; j++) {
        pa = (__m128i *)a;
        pb = (__m128i *)b;
        pc = (__m128i *)c;
        LEN8 = LEN / 8;
        for (i = 0; i < LEN8; i++) {
            *pc = _mm_adds_epu16 (*pa, *pb);
            pa += 1;
            pb += 1;
            pc += 1;
        }
    }
    edTime = clock();
    unsigned int spendtime = edTime - stTime;
    printf("time used: %d \n", spendtime);
    _getch();
```

```
    return 0;
}
```

在本人的机器上，上段程序的执行时间约为 120 毫秒。而使用基于 C 语言的 MMX 函数的运行时间约为 220 毫秒（见 15.4 节）。

有兴趣的读者，可以打开 emmintrin.h。在该头文件中定义了 UNION 联合体__m128i。该 128 位二进制信息可以看成是 16 个字节数据(int8)、8 个字数据(int16)、4 个双字数据(int32),或者 2 个 64 位数据(int64),它们都是有符号的数据；也可以成是无符号的字节、字、双字、四字数据。在头文件在还有很多函数，基本上就是在 SSE 指令上的封装，采用反汇编手段，可以看到其实现方法。

如果直接用在 C 语言中嵌入汇编代码，测试用时约为 60 毫秒。嵌入部分的代码如下：

```
    __asm{
      mov  ecx,1000
    l1:
      lea  edi, a
      lea  esi, b
      lea  eax, c
      mov  edx , LEN/8
   l2:
      movdqu  xmm0, xmmword ptr[esi]
      paddusw xmm0, xmmword ptr[edi]
      movdqu  xmmword ptr[eax], xmm0
      add  edi, 16
      add  esi, 16
      add  eax, 16
      dec  edx
      jnz  l2
      dec  ecx
      jnz  l1
}
```

如果单纯的只有 SSE 的指令，可以使用头文件 xmmintrin.h。在该文件中可以看到有 SSE 指令封装后的函数。除此之外，在 VS2019 平台中，有多个 *intrin.h,它们对应着不同版本的指令封装。

习题 16

16.1 SSE 中有哪 8 个 128 位的寄存器？控制和状态寄存器中有哪些控制和状态标志位？

16.2 设有变量

```
x       dw    70H,   0FFA0H,   50H,   50H,   0F0H, 0F0H, 0F000H, 0F0H
y       dw   0A0H,   0070H,   30H,   0F0H,   01H,   20H,   8001H, 0F0H
z       dw   8 DUP(0)
```

编写程序，完成 z=x+y，x、y 和 z 都看成为向量。要求分别用下面两种规则实现：

① 使用 SSE 指令，实现无符号字饱和字加法；

② 使用 SSE 指令，实现字环绕加法。

16.3　设有变量

```
buf1    sword   1, -2, 3, -4，5,-6,7,8
buf2    sword   2, 3, 4, 5,-6,-7,-8,-9
result  sqword  8 dup(0)
```

编写程序，用 SSE 指令实现向量 buf1 和 buf2 的点乘，即各个对应元素相乘，结果存放在 result 中。

上机实践 16

16.1　用 C 语言编写程序，实现两个矩阵的乘法。对矩阵相乘的部分进行计时。

假设矩阵中元素类型为 short，矩阵行列数都是 8 的倍数。

(1) 用传统方法（即逐个元素运算）实现

(2) 用 SSE 打包运算方法（参见 emmintrin.h 中的函数）

16.2 功能与上机实践 16.1 相同，即实现矩阵的乘法，但矩阵中的行、列数为任意正整数。