SIMD 指令（单指令多数据流）

Single Instruction Multiple Data

- MMX（MultiMedia eXtension）------ 15 章
- SSE（Streaming SIMD Extension）-- 16 章
- AVX（Advanced Vector eXtension）- 17 章

第15章 MMX程序设计

多媒体数据（音频、图像、视频等）处理中包含大量的具有共同特征的操作（计算）。为此，Intel公司于1997年推出了MMX（Multimedia eXtension）指令集。有57条指令，每条指令可以一次处理多个数据。

第15章 MMX程序设计

学习重点

单指令多数据流的基本概念；

环绕运算、饱和运算的概念

采用MMX指令，提高程序运行效率

15.1 MMX技术简介

Pentium Ⅱ

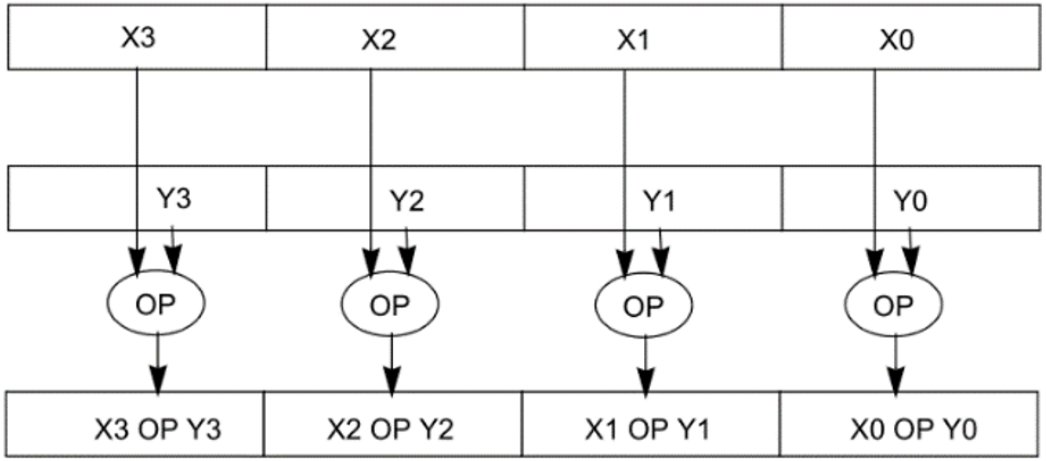
- 单指令多数据流　SIMD
  Single Instruction Multiple Data

- 多媒体扩展指令集　MMX
  Multi-Media eXtension

15.1 MMX技术简介

华中科技大学

- 单指令多数据流　SIMD



	10 (000AH)	20 (0014H)	30 (001EH)	40 (0028H)
+	25 (0019H)	-1 (0FFFFH)	-35(0FFDDH)	35 (0023H)
	35 (0023H)	19 (0013H)	-5 (0FFFBH)	75 (004BH)

15.1 MMX技术简介

MMX寄存器

- 8个64位的寄存器：MM0 ～ MM7。它们是FPU的浮点寄存器 ST0-ST7（80位）的低64位。
- MM0~MM7只支持整数运算（8个字节、4个字、2个双字）。如果需要进行浮点运算，需要使用EMMS指令将FPU状态复位。
- 直接使用这些寄存器：寄存器寻址方式。
- 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。

15.2 MMX指令简介

数据传送
算术运算
比较运算
逻辑运算
移位、转换、解组
状态控制

常用字母后缀来标识需要处理的元素大小，
b、w、d、q分别对应字节、字、双字、四字。

15.1 MMX技术简介

环绕运算 VS 饱和运算

环绕加减运算

等价于普通的ADD、SUB指令（直接加减，舍弃进/借位）

例如字节运算：

38H + F9H = 31 H

38H - F9H = 3F H

15.1 MMX技术简介

饱和加减运算

运算结果超出范围（溢出）时截断。例如，对于无符号字节运算，若结果超出255(FFH),则直接取FFH；对于有符号字节运算，若结果超出[-128,127](溢出),则直接取 80H/7FH(-128的补码为80H)。

例如字节运算：

6FH + 23H = 7F H （有符号,正溢出）
6FH + 23H = 92 H （无符号）
6FH + F3H = FF H （无符号）
6FH + F3H = 62 H （有符号）
8FH + F3H = 80 H （有符号,负溢出）

对于加法,只有2个加数的符号位相同时才可能产生溢出.

15.2 MMX指令简介


```
 x      db   70H, 0A0H, 50H,  50H, 0F0H, 0F0H, 0F0H, 0F0H
 y      db  0A0H,  70H, 30H, 0F0H,  01H,  20H,  81H, 0F0H
movq     mm0,qword ptr x ; mm0 = F0 F0 F0 F0 50 50 A0 70
movq     mm1,qword ptr y ; mm1 = F0 81 20 01 F0 30 70 A0
paddb    mm0,mm1         ; mm0 = E0 71 10 F1 40 80 10 10
paddsb   mm0,mm1         ; mm0 = E0 80 10 F1 40 7F 10 10
paddusb  mm0,mm1         ; mm0 = FF FF FF F1 FF 80 FF FF
```

paddb： 打包字节整数环绕加法指令

ADD Packed Byte integers

paddsb：有符号饱和字节加法指令

ADD Packed Signed Byte integers with signed saturation

paddusb ：无符号饱和字节加法指令

ADD Packed UnSigned Byte integers with unsigned saturation

15.3 MMX编程示例

实现两个向量的内积
设有向量 a=(a1,a2,a3,a4)， 向量b=(b1,b2,b3,b4)。
向量a、b的内积为 <a,b>=a1*b1+a2*b2+a3*b3+a4*b4。

```
.686P
.MMX
.model flat, c
  ExitProcess proto stdcall :DWORD
  printf      proto  :vararg
  includelib  libcmt.lib
  includelib  legacy_stdio_definitions.lib
.data
  buf1        sword    1,-2, 3, 400H
  buf2        sword    2, 3, 4, 500H
  buf3        sdword   0, 0
  lpFmt       db  "%d  %x(H)", 0dh,0ah,0
.stack 200
```

15.3 MMX编程示例


```
.code
main  proc
    movq      mm0,   qword ptr  buf1     ; mm0=04000003FFFE0001H
    movq      mm1,   qword ptr  buf2     ; mm1=0500000400030002H
    pmaddwd   mm0,   mm1                 ; mm0=0014000CFFFFFFFCH
    movq      qword ptr buf3,  mm0
    mov       eax,   buf3
    add       eax,   buf3 + 4
    emms   ;清除MMX状态
    invoke printf, offset lpFmt, eax, eax
    invoke Exitprocess, 0
main  endp
End
```

pmaddwd：向量点积，[a1,a2,a3,a4]*[b1,b2,b3,b4] = [a1*b1+a2*b2, a3*b3+a4*b4]

15.4 用C语言编写MMX应用程序


```
#include <stdio.h>      #include <time.h>
#include <stdlib.h>     #include <conio.h>
#define LEN 100000 // 数组大小
int main() {
        clock_t stTime, edTime;
        int   i,j;
        unsigned short  a[LEN], b[LEN], c[LEN];
        srand(time(NULL));
        for (i = 0; i < LEN; i++) // 生成随机数组
        {  a[i] = rand() ;  b[i] = rand(); }
        stTime = clock();
        for (j = 0; j < 1000; j++) { // 重复做1000遍
                for (i = 0; i < LEN; i++)
                        c[i] = a[i] + b[i];
        }
        edTime = clock();  // edTime - stTime;
        ……
}
```

15.4 用C语言编写MMX应用程序


```
#include  <mmintrin.h>
__m64  *pa, *pb, *pc;  // 指向数组 a
int  LEN4;  // 一次运算4个数，总循环次数减少
for (j = 0; j < 1000; j++) {
        pa = (__m64 *)a;
        pb = (__m64 *)b;
        pc = (__m64 *)c;
        LEN4 = LEN / 4;
        for (i = 0; i < LEN4; i++) {
                *pc = _m_paddw(*pa, *pb);
                pa += 1;   // 反汇编后，地址是加 8
                pb += 1;
                pc += 1;
        }
}
_m_empty();  // 实际是 EMMS指令
```

第15章 MMX程序设计

華中科技大學

单指令多数据流的基本概念

环绕运算

有符号/无符号的饱和运算

采用 MMX 指令提高程序运行速度

第16章　SSE程序设计

華中科技大學

16.1　SSE技术简介

16.2　SSE指令简介

16.3　SSE2及后续版本的指令简介

16.4　SSE编程示例

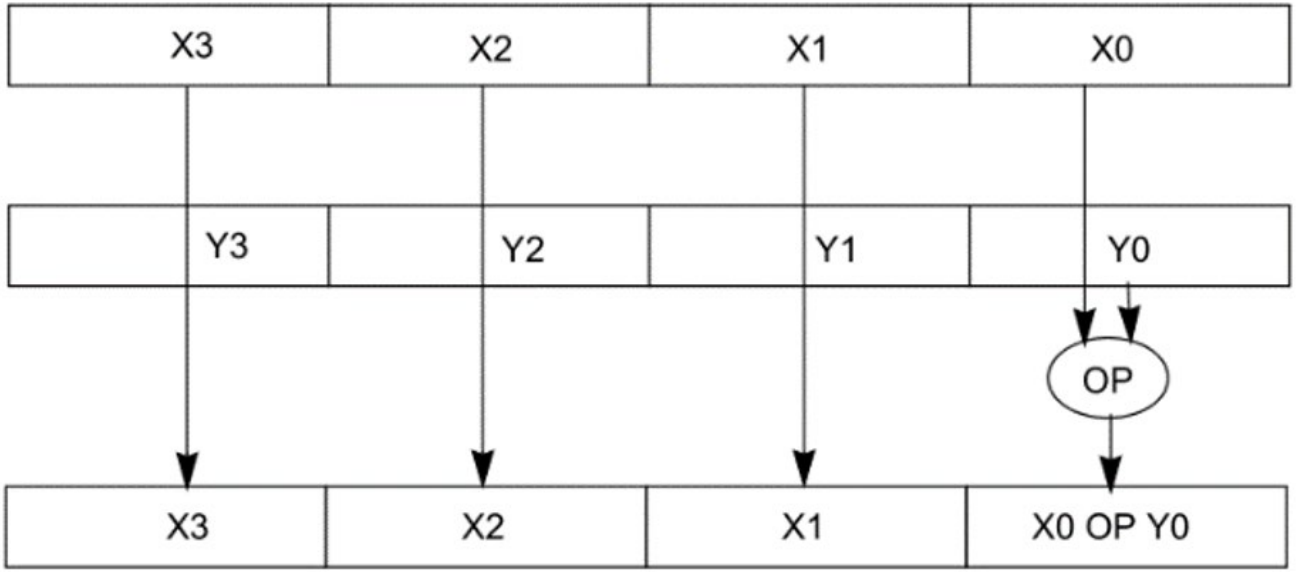
16.5　用C语言编写SSE应用程序

16.1 SSE技术简介

- Pentium Ⅲ 中，在MMX基础上引入了更多的流式SIMD扩展（Streaming SIMD Extension），称为 SSE。SSE兼容MMX指令，并且可以同时处理4个单精度数据。
- 8个128位的寄存器：xmm0～xmm7
- 保留了MMX的64位寄存器对于组合整数进行运算
- 增加了单精度浮点数(float)打包运算的指令
- 增加了标量单精度浮点数运算指令

16.1 SSE技术简介

- 标量单精度浮点数运算



16.1 SSE技术简介

SSE中的数据寄存器

- 8个128位的寄存器：xmm0～xmm7
- 直接使用这些寄存器：寄存器寻址方式。
- 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
- 可以用于整数运算，又可以用于浮点运算

16.1 SSE技术简介

SSE中的控制和状态寄存器

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

16.2 SSE指令简介

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 SSE指令简介

标量单精度浮点 、组合单精度浮点指令

数据传送

算术运算

比较运算

逻辑运算

重排和解组

转换

16.2 SSE指令简介

SSE 64位SIMD整数指令

- 采用64位MMX寄存器和64位的内存操作数
- 保留了 MMX 指令
- 同时增加了一些新的指令

16.2 SSE指令简介

MXCSR状态管理指令

缓存控制指令

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

组合和标量双精度浮点指令

64位和128位整数指令

16.4 SSE编程示例


```
#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;
}
```

16.4 SSE编程示例


```
        x = 3.14;
movss       xmm0, dword ptr [__real@4048f5c3 (0E67B34h)]
movss       dword ptr [x], xmm0   标量单精度浮点数传送
.....                  Scalar Single-precision floating-point
        z = x + y;
movss       xmm0, dword ptr [x]
addss       xmm0, dword ptr [y]   标量单精度浮点数加法
movss       dword ptr [z], xmm0
        printf("%f\n", z);
cvtss2sd    xmm0, dword ptr [z]   float => double
sub         esp, 8
movsd       mmword ptr [esp], xmm0 标量双精度浮点数传送
push        offset string "%f\n" (0E67B30h)
call        _printf (0E61046h)
add         esp, 0Ch
```

16.4 SSE编程示例


```
.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
```

16.4 SSE编程示例


```
.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   ;; printf("%f\n", z)
   add       esp,  16
   invoke    ExitProcess, 0
main  endp
end
```

16.4 SSE编程示例

若将 z的定义改为 z real8 0.0，
程序中的片段可简化：

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

printf 显示浮点数时，要求一个双精度浮点数

16.5 用C语言编写SSE应用程序


```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <conio.h>
#include <emmintrin.h>
#define LEN 100000    //数组大小
```

16.5 用C语言编写SSE应用程序


```
int main() {
    clock_t stTime, edTime;
    int   i, j;
    _declspec(align(16)) unsigned short  a[LEN], b[LEN];
    _declspec(align(16)) unsigned short  c[LEN];
    __m128i  *pa, *pb, *pc;
    int LEN8;
    srand(time(NULL));
    for (i = 0; i < LEN; i++) {
        a[i] = rand();   b[i] = rand();
    }
```

16.5 用C语言编写SSE应用程序


```
    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);
   return 0;
}
```

16.5 用C语言编写SSE应用程序

如只有 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。

在VS2019平台中，有多个 *intrin.h,

它们对应着不同版本的指令封装。

第16章 SSE程序设计

Streaming SIMD Extensions

数据位数更多的寄存器：128位的寄存器：xmm0～xmm7

更快的运算速度，更多地指令

第17章 AVX程序设计

華中科技大學

17.1 AVX技术简介

17.2 AVX指令简介

17.3 AVX编程示例

17.1 AVX技术简介

高级向量扩展 AVX

Advanced Vector eXtension

- 2011年,在MMX、SSE序列之后出现的SIMD增强版。

  Intel酷睿处理器i3、i5、i7系列中使用

- 2013年，Intel发布了 AVX2
- 2016年推出了AVX-512

  酷睿i7、i9等CPU中都支持AVX-512

17.1 AVX技术简介

- 8个256位的寄存器　ymm0 ～ ymm7
- YMM寄存器的低128位可看成是一个XMM寄存器
- 直接使用寄存器的名字，即采用寄存器寻址方式访问组合整数、组合浮点数和标量浮点数。
- 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。

17.1 AVX技术简介

半精度浮点数

- 用16个二进制位来存储
- 最高位为符号位，之后是指数部分（5位）、有效数字部分（10位）。
- 半精度浮点数无法进行加、减、乘、除等运算，主要是用来节约存储空间的。

17.1 AVX技术简介

乘法与加法混合运算

- 采用乘法与加法混合运算（Fused Multiply Add，FMA）指令，不会对乘法的结果做舍入处理，只有在得到最后加法的结果后，才会做舍入操作。
- 这样使用FMA指令可以提高乘法累加（如向量内积）运算的性能和精度。

17.2 AVX指令简介

- 在指令编码模式中采用了一种新的前缀(VEX)。
- 大多数AVX指令采用了三目运算符的格式：

  VOP　DesOp, SrcOp1, SrcOp2
- SrcOp1和SrcOp2为源操作数地址
- DesOp为目的操作数地址。

17.2 AVX指令简介

AVX指令集大致可分为三类

- 用新的表示方法但功能等效于SSE指令的升级版本指令；
- 新引入的指令；
- 功能扩展指令，包括半精度浮点数变换、乘法加法混合运算指令和新的通用寄存器指令。

17.3 AVX编程示例


```
.686P
.XMM
.model flat, c
  ExitProcess proto stdcall :dword
  printf      proto  :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
```

17.3 AVX编程示例


```
.code
main proc
    vmovss    xmm0,x
    vaddss    xmm0,xmm0,dword ptr y
    vmovss    z,xmm0
    cvtss2sd  xmm0,z
    sub       esp,8
    vmovsd    qword ptr [esp], xmm0
    invoke    printf,offset lpFmt
    add       esp,16
    invoke    Exitprocess,0
main endp
end
```

17.3 AVX编程示例

与C语言编程中使用MMX、SSE技术一样，

在C语言编程中可以使用AVX函数，以提高大规模数据运算的能力。

具体的函数可以参考头文件 immintrin.h。