

第14章 算术协处理器、MMX和SIMD技术

罗文坚
中国科大 计算机学院

<http://staff.ustc.edu.cn/~wjluo/mcps/>

1

概述

- Intel系列的算术协处理器包括8087、80187、80287、80387SX、80387DX、80487SX。
- 80486DX~Core2微处理器均有内置的算术协处理器。
- 但是，某些兼容的80486 CPU（由IBM和Cyrix生产）内部并不包含算术协处理器。
- 对于各种协处理器，指令系统和编程几乎完全相同，主要区别是每种协处理器被设计成与Intel不同型号的微处理器共同工作。

2

概述

- 80X87协处理器可以实现乘法、除法、加法、减法、求平方根、部分正切、部分反正切和对数运算。
- 数据类型包括：
 - 16位、32位和64为带符号整数；
 - 18位BCD数据；
 - 32位、64位和80位浮点数。
- 应用80X87执行的操作，通常比使用微处理器常用指令系统写出的最有效程序来执行同等的操作快许多倍。

3

概述

- 使用改进的Pentium协处理器，其运算速度比同等时钟频率下80486微处理器执行速度快5倍。
 - 注意：Pentium微处理器常常可以同时执行一条协处理器指令和两条整数指令。
- Pentium Pro ~ Pentium 4协处理器与Pentium协处理器的操作类似，但增加了两条新的指令，即FMOV和FCOMI。

4

概述

- 多媒体扩展（MMX）与算术协处理器共享寄存器。
 - MMX扩展是一种特殊的内部处理器，设计用于为外部多媒体设备高速执行指令。
- SSE（Streaming SIMD Extensions）：“与MMX指令类似，但作用于浮点数（而不是整数）；SSE并不使用协处理器的寄存器空间”。
 - SSE系列是MMX的超集，直到SSE2才跟MMX有本质的区别，添加了对64位双精度浮点数的支持，以及对整型数据的支持，也就是说这个指令集中所有的MMX指令都是多余的了，同时也避免了占用浮点数寄存器。

5

本章内容

- 算术协处理器的数据格式
- 80X87的结构
- 指令系统
- 算术协处理器编程
- MMX技术简介
- SSE技术概述

6

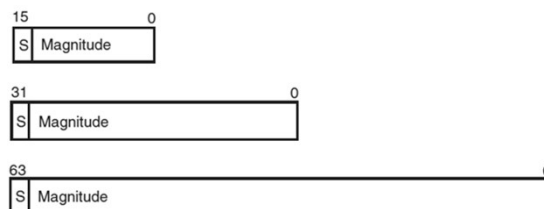
算术协处理器的数据格式

- 算术协处理器支持的数据类型包括：
 - 带符号整数
 - BCD数
 - 浮点数
- 注意：针对协处理器进行汇编语言编程常常局限于修改诸如C/C++高级语言生成的代码。

7

带符号整数

- 带符号整数：以补码形式存储。
 - 字（16位）、双字（32位）、四字（64位）。



8

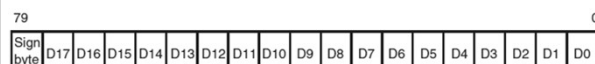
带符号整数

- 数据定义：采用汇编伪指令DD，DW和DQ。
- 例：
 - DATA1 DW 2
 - DATA2 DW -34
 - DATA3 DD 1234
 - DATA4 DD -100
 - DATA5 DQ 23456
 - DATA6 DQ -122

9

BCD数

- BCD数：以原码形成存储，而不是以10的补码形式存储的。
- 一个BCD数需要80位的内存，占用10个字节。
 - 每个BCD数有18个数位，以压缩整数形式存储，每个字节有2个数位，共占用9个字节。
 - 第10个字节只包含带符号的18位BCD数的符号位。



10

BCD数

- 数据定义：使用汇编伪指令DT。
- 例：
 - 0000 0000000000000000200 DATA1 DT 200
 - 000A 80000000000000000010 DATA2 DT -10
 - 0014 000000000000000010020 DATA3 DT 10020
- 这种格式很少用，因为它唯一用于Intel协处理器。

11

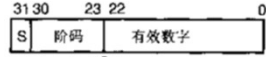
浮点数

- 浮点数通常称为实数，因为它们支持带符号整数、分数和混合数。
- 一个浮点数包括3个部分：符号位、阶码、有效数字。
- 浮点数通过科学二进制计数法来表示的。
- Intel系列算术协处理器支持三种类型的浮点数：
 - 短浮点数（32位），即单精度浮点数
 - 长浮点数（64位），即双精度浮点数
 - 临时浮点数（80位），即扩展精度浮点数

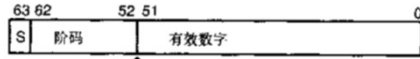
12

浮点数

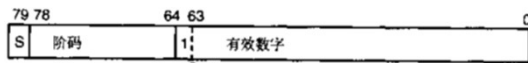
- 浮点数格式以及算术协处理器对它们的操作都遵循IEEE-754标准。



(a) 短 (单精度) 浮点数, 带有偏移量7FH



(b) 长 (双精度) 浮点数, 带有偏移量3FFH



(c) 临时 (扩展精度) 浮点数, 带有偏移量3FFFH

13

浮点数

- 有效数字是带有隐含位1 (整数部分) 的数字。以扩展精度存储时, 整数部分的1是可见的。

- 表示规则:

- 零: 指数部分为0, 小数部分为0。
- 无穷大/无穷小: 指数部分为 2^e-1 , 小数部分为0。
- NaN: 指数部分为 2^e-1 , 小数部分非零。
- 规约形式: 指数部分 $1 \sim 2^e-2$, 小数部分为任意值。
- 非规约形式: 指数部分为0, 小数部分非0。
 - 在非规约形式下整数部分默认为0, 其他情况下一律默认为1。

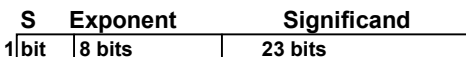
14

IEEE 754标准

规格化数: $\pm 1.x \times 2^{\text{Exponent}}$

规定: 小数点前总是“1”, 故可隐含表示。

Single Precision:



- Sign bit: 1 表示 negative; 0 表示 positive
- Exponent (阶码 / 指数编码): 全0和全1用来表示特殊值!
 - SP规格化数阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
 - bias为127 (single), 1023 (double) 为什么用127? 若用128, 则阶码范围是多少?
- Significand (尾数):
 - 规格化尾数最高位总是1, 所以隐含表示, 省1位
 - 1 + 23 bits (single), 1 + 52 bits (double)

SP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$ 0000 0001 (-127) ~
 DP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-1023)}$ 1111 1110 (126)

15

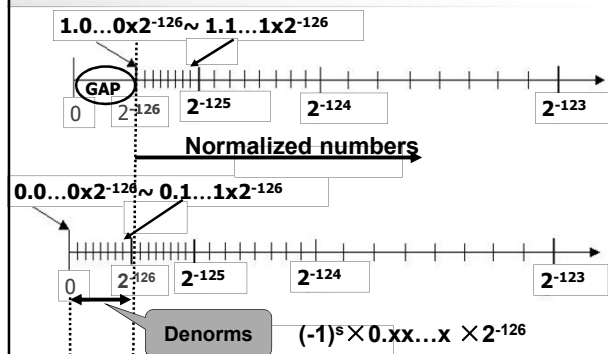
IEEE 754标准

以单精度为例:

Exponent	Significand	Object
0	0	+/- 0
0	nonzero	Denorms
1-254	Anything (implicit leading 1)	Norms
255	0	+/- infinity
255	nonzero	NaN

16

Representation for Denorms



17

浮点数

- 在Visual C++2008或Express版中, 采用了float, double, Decimal三种数据类型。

- float: 32位;
- double: 64位;
- Decimal

- Visual Studio的特殊类型。用于要求浮点数非常精确的领域, 如银行。decimal变量是Visual Studio 2005之后才有的。
- Base为10, 而不是2。
- 背景: double $x=0.1$, 但实际上, $x \neq 0.1$ 。

18

浮点数

- 将一个十进制数转换为浮点数形式的步骤：
 - 将十进制数转换为二进制数。
 - 规格化二进制数。
 - 计算出阶码。
 - 以浮点数格式存储该数。
- 例，将十进制数100.25转换为单精度浮点数。

步骤1: 100.25 => 1100100.01

步骤2: 1100100.01 => 1.10010001×2^6

步骤3: 000000110 + 01111111 => 10000101

步骤4: 符号位= 0

指数=10000101

有效数字=10010001000000000000000

19

浮点数

- 将一个浮点数转换为十进制数形式的步骤：
 - 分离符号位、阶码和有效数字。
 - 通过减去偏移量，将阶码转换为真正的指数。
 - 将此数写为规格化的二进制数。
 - 将规格化的二进制数转换为非规格化二进制数。
 - 将非规格化二进制数转换为十进制数。

20

浮点数

- 例，将浮点数转换为十进制数。

步骤1: Sign => 1

Exponent => 10000011

Significand => 100100100000000000000000

步骤2: 100 = 10000011 - 01111111

步骤3: 1.1001001×2^4

步骤4: 11001.001

步骤5: -25.125

21

浮点数

- 使用汇编语言存储浮点数时：
 - 用DD伪指令存储单精度浮点数；
 - 用DQ伪指令存储双精度浮点数；
 - 用DT伪指令存储扩展精度浮点数。
- MASM6.0版本有个错误：不允许正浮点数使用加号。
 - 例，+92.45必须定义为92.45，否则汇编程序不能正常运行。
- MASM6.11版本中，用REAL4取代DD；REAL8取代DQ；REAL10取代DT。

22

浮点数

- 例，浮点数的定义。

内存中数据	变量名	伪指令	浮点数
1. C377999A	DATA7	DD	-247.6
2. 40000000	DATA8	DD	2.0
3. 486F4200	DATA9	REAL4	2.45E+5
4. 4059100000000000	DATAA	DQ	100.25
5. 3F543BF727136A40	DATAB	REAL8	0.001235
6. 400487F34D6A161E4F76	DATA C	REAL10	33.9876

23

浮点数

- 没有带协处理器的CPU，则由汇编语言提供了一个可以访问的8087仿真器。
 - 此仿真器存在于微软的所有高级语言中，或者作为共享程序（如EM87）。
 - 此仿真器是通过在程序中的.MODEL语句后面加上OPTION EMULATOR语句来进行访问的。
 - 注意：此仿真器不仿真某些协处理器指令。
- 带协处理器的CPU，则不要使用此仿真器。
- 任何情况下，需使用.8087，.80187，.80287，.80387，.80487，.80587，.80687开关来使能协处理器指令。

24

关于浮点数精度的一个例子

```
#include <iostream>
using namespace std;
int main()
{
    float heads;
    cout.setf(ios::fixed,ios::floatfield);
    while(1)
    {
        cout << "Please enter a number: ";
        cin>> heads;
        cout<<heads<<endl;
    }

    return 0;
}
```

25

关于浮点数精度的一个例子

<pre>#include <iostream> using namespace std; int main() { float heads; cout.setf(ios::fixed,ios::f); while(1) { cout << "Please enter a cin>> heads;</pre>	<p>运行结果:</p> <pre>Please enter a number: 61.419997 61.419998 Please enter a number: 61.419998 61.419998 Please enter a number: 61.419999 61.419998 Please enter a number: 61.42 61.419998 Please enter a number: 61.420001 61.420002 Please enter a number:</pre>
---	---

61.419998和61.420002是两个可表示数，两者之间相差0.000004。当输入数据是一个不可表示数时，机器将其转换为最邻近的可表示数。

26

本章内容

- 算术协处理器的数据格式
- **80X87**的结构
- 指令系统
- 算术协处理器编程
- **MMX**技术简介
- **SSE**技术概述

27

80X87的结构

- **80X87与微处理器协同工作。**
 - **80486DX~Core2**包含内置的、与**80387**完全兼容的协处理器。
 - 对于其它的**Intel**系列微处理器，协处理器是并联在微处理器上的外部集成电路。
- **80X87可以执行超过68条不同的指令。**
- 算术协处理器是一种特殊用途的微处理器，专门为有效地执行算术或超越函数的运算而设计的。

28

80X87的结构

- 微处理器执行所有的常规指令，80X87只执行算术协处理器指令。
 - 微处理器截取和执行常规指令系统中的指令，而协处理器只截取和执行协处理器指令。
- 协处理器指令实际上是换码（ESC）指令。微处理器使用这些指令为协处理器产生一个内存地址，使得协处理器可以执行协处理器指令。
- 微处理器和协处理器可以同时或并发地执行各自的指令。

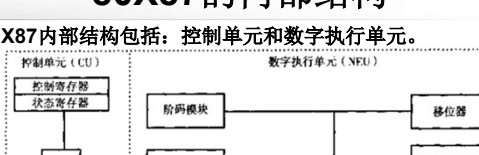
```
graph TD; ID[Instruction Decoder and Sequencer] -.-> IU[Integer Unit]; ID -.-> FPU[FPU]; IU <-->|Data Bus| FPU;
```

The diagram illustrates the 80X87 architecture. At the top, a box labeled 'Instruction Decoder and Sequencer' is connected by dashed lines to two boxes below it: 'Integer Unit' and 'FPU'. A solid double-headed arrow labeled 'Data Bus' connects the 'Integer Unit' and 'FPU' boxes, indicating bidirectional data flow between them.

29

80X87的内部结构

80X87内部结构包括：控制单元和数字执行单元。



该图展示了80X87的内部结构，分为控制单元（CU）和数字执行单元（NEU）两部分。

控制单元（CU）包含：

- 控制寄存器
- 状态寄存器
- 数据缓冲区
- 总线跟踪
- 异常

数字执行单元（NEU）包含：

- 阶码模块
- 指令译码器
- 操作数队列
- 移位器
- 算术运算模块
- 临时寄存器
- 80位宽堆栈（包含8个寄存器，编号0-7）
- 标记寄存器

数据总线连接了控制单元的数据缓冲区和数字执行单元的堆栈。状态地址总线连接了控制单元的状态跟踪和异常模块。

30

控制单元

- 控制单元
 - 控制单元将协处理器连接到微处理器系统数据总线上。
 - 微处理器和协处理器均监视指令流。
 - 如果是ESC（协处理器）指令，则由协处理器执行，否则由微处理器执行。

31

数字执行单元

- 数字执行单元（Numeric Execution Unit, NEU）
 - 负责执行所有协处理器指令。
 - NEU有一个包含8个寄存器的堆栈，用于存储算术指令的操作数和结果。指令或者寻址堆栈中指定寄存器的数据，或者使用PUSH/POP机制在栈顶存储和取回数据。
 - NEU的其他寄存器分别是状态、控制、标记和异常指针寄存器。
 - 很少有指令在协处理器和微处理器的AX寄存器之间传输数据。FSTSW AX指令是协处理器允许通过AX寄存器和微处理器直接通信的唯一指令。
 - 注意：8087不包含FSTSW AX指令。

32

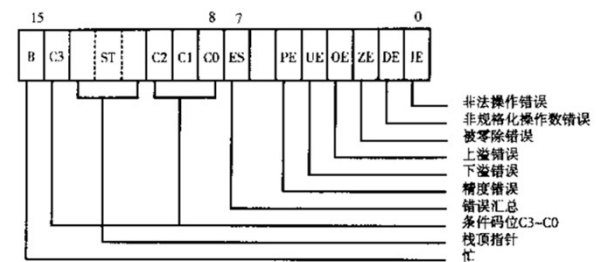
寄存器堆栈

- 寄存器堆栈
 - 协处理器的堆栈包括8个寄存器，每个80位宽。
 - 这些寄存器中总是包含一个80位的扩展精度浮点数。
 - 当数据从内存中移到协处理器寄存器堆栈中时，协处理器将这些带符号的整数、BCD数、单精度或双精度转换为扩展精度浮点数。
 - 数据只有驻留在内存时，才可能是任何其他格式。

33

状态寄存器

- 状态寄存器：反映协处理器所有指令的运行情况。



更新的协处理器（80187或更高）使用状态位6（SF）来识别堆栈上溢或下溢错误。

34

状态寄存器

- 状态寄存器：
 - 可用FSTSW指令把状态寄存器的内容送到内存单元中。
 - 对于80187及其以后的协处理器，可用指令“FSTSW AX”把状态寄存器的值传送给AX。
 - 一旦状态寄存器的值复制到内存或AX中，就可用常规软件检测状态寄存器中的各位。
- 协处理器与微处理器之间的通信：
 - 对于80187和80287，可通过I/O地址00FAH~00FFH来实现协处理器与CPU之间的数据交换。
 - 对于80387~Core2 CPU，则是通过I/O地址80000FAH~80000FFH来实现这两者之间的数据交换。
 - 注意：不使用这些端口来连接I/O设备到微处理器。

35

状态寄存器

- B: Busy bit, 忙标志位，用来表明协处理器是否正在执行协处理器指令。
 - 它可用FWAIT指令来测试。在80287及其以后的协处理器中，协处理器和CPU能自动实现同步，所以，在运行任务时，无须测试忙标志。
- C0~C3: Condition code bits, 条件码位，表明协处理器的条件。
 - 对于不同的指令，这些位有不同的含义。

36

状态寄存器

• 状态寄存器的条件码

指令	C3	C2	C1	C0	意义
FIST, FCOM	0	0	X	0	ST > 操作数
	0	0	X	1	ST < 操作数
	1	0	X	1	ST = 操作数
	1	1	X	1	ST 不可比较
FPMEM	Q1	0	Q0	Q2	商的最右3位
	?	1	?	?	未完成
FXAM	0	0	0	0	+ unnormal
	0	0	0	1	+ NAN
	0	0	1	0	- unnormal
	0	0	1	1	- NAN
(未完)	0	1	0	0	+ normal

37

状态寄存器

• 状态寄存器的条件码

指令	C3	C2	C1	C0	意义
FXAM (续)	0	1	0	1	正无穷大
	0	1	1	0	- normal
	0	1	1	1	负无穷大
	1	0	0	0	+0
	1	0	0	1	空
	1	0	1	0	-0
	1	0	1	1	空
	1	1	0	0	+ denormal
	1	1	0	1	空
	1	1	1	0	- denormal
	1	1	1	1	空

denormal: 阶码是最大的负值

38

状态寄存器

• TOP: Top-of-stack bit (ST), 栈顶位

- 三位二进制000~111用来表明当前作为栈顶的寄存器, 通常其值为000。

• ES: Error summary bit, 错误汇总位

- $ES = PE + UE + OE + ZE + DE + IE$ (逻辑或运算)。
- 在8087协处理器中, 当ES为1时, 将发出一个协处理器中断请求; 但在其后的协处理器中, 不再产生这样的协处理器中断申请。

39

状态寄存器

• SF: 堆栈溢出错误

- 该状态位用来表明协处理器内部的堆栈是否有上溢或下溢错误。

• PE: Precision error, 精度错误

- 该状态位用来表明运算结果或操作数是否超过先前设定的精度。

• UE: underflow error, 下溢错误

- 该状态位用来表明一个非0的结果太小, 不能用控制字节所选定的当前精度来表示。

40

状态寄存器

• OE: overflow error, 上溢错误

- 该状态位用来表明一个非0的结果太大, 不能用控制字节所选定的当前精度来表示, 即超过了当前精度所能表示的数据范围。
- 如果在控制寄存器中屏蔽该错误标志, 即设控制寄存器中的OM为1, 那么, 协处理器把上溢结果定义为无穷大。

• ZE: zero error, 除法错误

- 该状态位用来表明当前执行了“0作除数”的除法运算。

41

状态寄存器

• DE: denormalized error, 非规格化错误

- 该状态位用来表明当前参与运算的操作数中至少有一个操作数是没有规格化的。

• IE: invalid error, 非法错误

- 该状态位用来表明执行了一个错误的操作。
- 例如, 求负数的平方根, 也可用来表明堆栈的溢出错误、不确定的格式(0÷0, +∞, -∞等)错误, 或用NAN作为操作数。

42

状态寄存器

- 用指令“**FSTSW AX**”把状态寄存器的内容传给**AX**之后，一般可用下面二种方法来检测协处理器的状态。
 - 用**TEST**指令来检测其相应的状态位。
 - 用**SAHF**指令把**AX**的低字节传送给**CPU**的标志寄存器，然后再用条件转移指令来完成相应的检测。

43

状态寄存器

- 例，检测是否有“0作除数”的错误。

```
FDIV DATA1
FSTSW AX ;复制状态寄存器的内容到AX
TEST AX, 4 ;测试ZE位
JNZ DIVIDE_ERROR
```

44

状态寄存器

- 例，检测是否有“非法操作数”的错误。

```
FSQRT
FSTSW AX
TEST AX, 1 ;测试IE
JNZ FSQRT_ERROR
```

45

状态寄存器

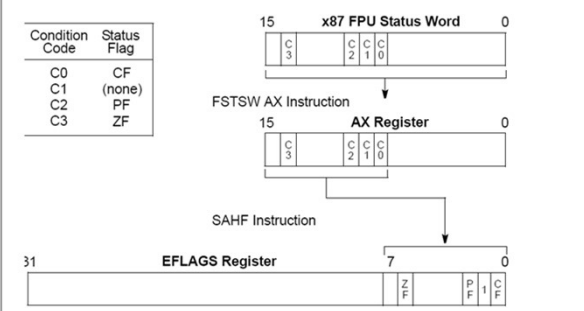
- 例，使用**SAHF**和条件转移指令。
 - 注意：**SAHF**在64位模式下不起作用。

```
FCOM DATA1 ;栈顶数据域DATA1比较
FSTSW AX
SAHF ;拷贝AH到标志寄存器
JE ST_EQUAL
JB ST_BELOW
JA ST_ABOVE
```

46

状态寄存器

- 从状态寄存器到**EFLAGS**



47

状态寄存器

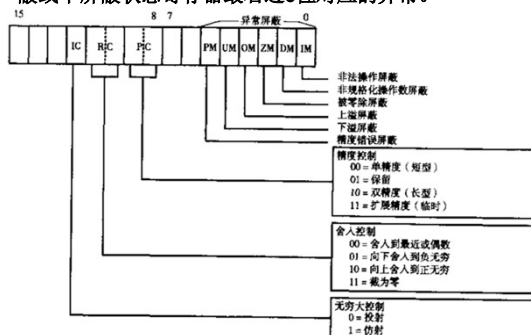
- 例，在Pentium 4或Core2的64位模式下，用**TEST**指令来测试先决条件。

```
FCOM DATA1
FSTSW AX
TEST AX, 100H
JNZ ST_BELOW
TEST AX, 4000H
JNZ ST_EQUAL
JMP ST_ABOVE
```

48

控制寄存器

- 控制寄存器包含精度控制、舍入控制和无穷大控制，以及屏蔽或不屏蔽状态寄存器最右边6位对应的异常。



49

控制寄存器

- IC: 无穷大控制, infinity control**, 选择是仿射无穷大还是投射无穷大。仿射允许正无穷大和负无穷大, 而投射则假定无穷大为无符号数。
- 80287使用; 80387以后没有意义。
- RC: 舍入控制, rounding control**, 确定舍入的类型。
- PC: 精度控制, precision control**, 设置结果的精度。
- Exception Masks: 异常屏蔽**, 决定异常错误是否影响状态寄存器的错误位。
- FLDCW指令**用于给控制寄存器赋值。

50

控制寄存器

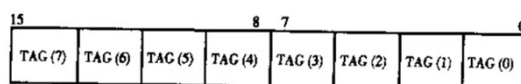
- RC: 舍入控制**

Rounding Mode	RC Field Setting	Description
Round to nearest (even)	00B	Rounded result is the closest to the infinitely precise result. If two values are equally close, the result is the even value (that is, the one with the least-significant bit of zero). Default
Round down (toward $-\infty$)	01B	Rounded result is closest to but no greater than the infinitely precise result.
Round up (toward $+\infty$)	10B	Rounded result is closest to but no less than the infinitely precise result.
Round toward zero (Truncate)	11B	Rounded result is closest to but no greater in absolute value than the infinitely precise result.

51

标记寄存器

- 标记寄存器: 表明协处理器堆栈中每个单元的内容。**



TAG值:
00 = 合法
01 = 零
10 = 非法或无穷大
11 = 空

- 通过程序查看标记寄存器的唯一方法是使用 **FSTENV**、**FSAVE**或**FRSTOR**指令存储协处理器操作环境。

52

本章内容

- 算术协处理器的数据格式
- 80X87的结构
- 指令系统
- 算术协处理器编程
- MMX技术简介
- SSE技术概述

53

指令系统

- 80X87**可以执行超过68条不同的指令。
- 一旦协处理器指令要访问内存, 则微处理器自动给指令产生内存地址。
- 协处理器在协处理器指令期间使用数据总线传送数据。
- 微处理器在常规指令期间使用数据总线传送数据。
- 协处理器使用微处理器的存储器寻址方式。

54

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

55

数据传送指令

- 浮点数传送
 - FLD, FST, FSTP, FXCH
- 带符号整数传送
 - FILD, FIST, FISTP
- BCD数据传送
 - FBLD, FBSTP
- 数据只有在内存中才以带符号整数形式或BCD数形式出现。
- 在协处理器内部，数据总是以80位扩展精度浮点数形式出现。

56

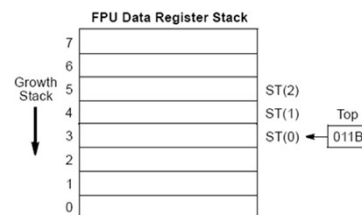
浮点数传送

- **FLD**: 将内存浮点数据装入由ST指向的内部栈顶。
 - 该指令先将堆栈指针减1，然后将数据存储在栈顶。
 - 装入栈顶的数据来自于任何存储单元，或来自于协处理器的另一个寄存器。
 - 传送数据的长度自动由汇编程序通过伪指令决定，如DD或REAL4表示单精度数据，DQ或REAL8表示双精度数据，而DT或REAL10表示扩展精度数据。
- 当协处理器复位或初始化时，栈顶为寄存器0。

57

浮点数传送

- 例，FLD ST(2)
 - 将寄存器ST(2)中的内容复制到栈顶。
- 例，FLD DATA7
 - 将DATA7的内容复制到栈顶。



58

浮点数传送

- **FST**: 将栈顶的内容复制到存储单元或由操作数指示的协处理器寄存器中。
 - 在存储过程中，内部的扩展精度浮点数舍入成为由控制器指定的浮点数长度。
- **FSTP**: 将栈顶内容复制到内存或协处理器寄存器中，然后从栈顶弹出该数据。
 - FST是复制指令，FSTP是移动指令。
- **FXCH**: 交换由操作数指定的寄存器或栈顶中的内容。
 - 例，FXCH ST(2): 交换栈顶数和ST(2)中的数据。

59

整数传送指令

- **FILD**: 装入整数。
- **FIST**: 存储整数。
- **FISTP**: 存储并弹出整数。
- 这三条指令的功能与FLD、FST和FSTP一样，只不过传送的数据类型为整数，而不是浮点数。
- 协处理器自动将内部的扩展精度浮点数转换为整数。
- 数据长度由汇编语言中用DW、DD和DQ定义标识的方法来决定。

60

BCD数据传送指令

- **FBLD**: 将内存BCD数装入栈顶。
- **FBSTP**: 存储并弹出栈顶数据。

61

FCMOV指令

- **FCMOV**是Pentium Pro~Core2新增的一条指令。
- 此指令包含一个条件，如果条件为真，则将源内容复制到目标单元中。

指令	条件	说明
FCMOVB	低于则传送	CF=1
FCMOVE	等于则传送	ZF=1
FCMOVBE	低于或等于则传送	CF=1或ZF=1
FCMOVU	无序则传送	PF=1
FCMOVNB	不低于则传送	CF=0
FCMOVNE	不等于则传送	ZF=0
FCMOVNBE	不低于或等于则传送	CF=0且ZF=0
FCMOVNU	有序则传送	PF=0

62

FCMOV指令

- **FCMOV**的注意事项:
 - **FCMOV**的测试条件或者是顺序检验，或者是非顺序检验。
 - **FCMOV**不检验NAN和非规格化操作数。
- 例，当ST(2)低于ST时，将ST(2)复制到ST。

FCOM ST(2)
FSTSW AX
SAHF
FCMOVB ST(2)

或:
FCOMI ST(2)
FCMOVB

FCOMI是Pentium Pro~Core2新增的一条指令。

63

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

64

算术运算指令

- 协处理器的算术运算操作指令:
 - 加法、减法、乘法、除法
- 与算术运算操作相关的指令:
 - 求平方根、比例运算、舍入运算、求绝对值运算、改变符号等。

65

算术运算操作指令

- 算术运算操作指令的寻址方式:
 - 堆栈寻址
 - 寄存器寻址
 - 存储器

66

算术运算的寻址方式

- 堆栈寻址：以栈顶作为源操作数，次栈顶为目的操作数。用弹出操作从堆栈移走栈顶的源操作数，使得目的寄存器的结果保留在栈顶。
 - 使用这种寻址方式，程序中的指令无需任何操作数。
 - 例，FADD：将ST与ST(1)相加，结果存储在栈顶中，而且弹出ST。
 - 例，FSUB：执行 $ST(1)=ST(1)-ST$ 操作，而且弹出ST。
- 注意：反向指令FSUBR和FDIVR有所不同。
 - FSUBR： $ST(1)=ST(0)-ST(1)$ ，弹出ST。
 - FDIVR： $ST(1)=ST(0)\div ST(1)$ ，弹出ST。

操作数是隐含的！

67

算术运算操作指令的寻址方式

- 寄存器寻址方式：一个操作数是ST，另一操作数是ST(n)，其中n为寄存器号0~7。
 - 对于大部分指令，ST或ST(n)都可以作为目的操作数。
- 例，FADD ST(1), ST
 - 操作： $ST(1)=ST(1)+ST$ 。
- 例，FADD ST, ST(0)。
 - 操作：使栈顶内容加倍。

68

算术运算操作指令的寻址方式

- 存储器寻址：由于协处理器是面向堆栈的机器，所以栈顶总被存储器寻址方式用作目的操作数。
- 例，FADD DATA
 - 操作：将存储单元DATA中的实数加到栈顶。

69

算术运算操作指令的寻址方式

- 以FADD为例。

方式	形式	例子
堆栈	ST(1), ST	FADD
寄存器	ST, ST(n) ST(n), ST	FADD ST, ST(1) FADD ST(4), ST
寄存器弹出	ST(n), ST	FADDP ST(4), ST
存储器	操作数	FADD DATA3

70

算术运算操作指令

- 算术运算操作：
 - 加法：FADD / FADDP / FIADD
 - 减法：FSUB / FSUBP / FISUB
 - 乘法：FMUL / FMULP / FIMUL
 - 除法：FDIV / FDIVP / FIDIV
 - 反向减法：FSUBR / FSUBRP / FISUBR
 - 反向除法：FDIVR / FDIVRP / FIDIVR
- 操作码中的字母P：表示操作结束后有寄存器弹出。
- 操作码中的字母R（只在减法和除法中出现）：表示反向模式。
- 操作码中第2个字母I：表明内存操作数是整数。

71

算术运算操作指令

- 例，FADDP和FADD的不同。
 - FADD ST(4), ST
 - FADDP ST(4), ST
- 例，如果栈顶的值为10，而存储单元DATA1为1：
 - FSUB DATA：栈顶的值为+9。
 - FSUBR DATA：栈顶的值为-9。

72

算术运算操作指令

- 例，FSUBR ST, ST(1)和FSUBR ST(1), ST的区别：
 - FSUBR ST, ST(1): $ST = ST(1) - ST$
 - FSUBR ST(1), ST: $ST(1) = ST - ST(1)$
- 例，FADD DATA是浮点加法，而FIADD DATA指令是整数加法，将存储单元DATA中的整数加到栈顶的浮点数中。

73

与算术运算操作相关的指令

- FSQRT: 求平方根。
 - 格式: FSQRT
 - 功能: 求栈顶数据的平方根，将平方根存于栈顶。
 - 若对负数求平方根，则会出现非法错误，此时状态寄存器的IE位置位。
- FSCALE: 对一个数进行比例运算。
 - 格式: FSCALE
 - 功能: 将ST(1)中的内容（被认为是整数）加到栈顶的指数中。ST(1)的值必须在 2^{+15} 与 2^{-15} 之间。
 - FSCALE能快速地乘以或除以2的幂。

74

与算术运算操作相关的指令

- FPREM/FPREM1: 取模。
 - 格式: FPREM 或 PPREM1
 - 功能: 完成ST对ST(1)的取模操作，结果（余数）放在栈顶。
 - 注意1: 取模结果只有余数，没有商。
 - 注意2: FPREM是为8087和80287兼容而设置的，不遵循IEEE 754的余数规范；在新的协处理器中，应使用遵循IEEE 754余数规范的指令PPREM1。

75

与算术运算操作相关的指令

- FRNDINT: 舍入为整数。
 - 格式: FRNDINT
 - 功能: 对栈顶的数进行舍入运算，使之成为整数。
- FXTRACT: 提取阶码和有效数字。
 - 格式: FXTRACT
 - 功能: 将栈顶的数分成2个独立部分，分别代表无偏移的阶码和有效数字。所提取的有效数字放在栈顶，而无偏移的阶码放在ST(1)。
 - 注意: 此指令常用以生成混合数的打印格式。

76

与算术运算操作相关的指令

- FABS: 求绝对值。
 - 格式: FABS
 - 功能: 将栈顶中数的符号变为正号。
- FCHS: 改变符号。
 - 格式: FCHS
 - 功能: 将正数变为负数，或者将负数变为正数。

77

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

78

比较指令

- 比较指令用于比较栈顶的数据和另一单元的数据，并将比较结果返回到状态寄存器中的条件码。
- 协处理器支持的比较指令：
 - **FCOM**：浮点数比较；可加P或PP：**FCOMP**，**FCOMPP**
 - **FICOM**：整数比较；可加P：**FICOMP**
 - **FTST**：测试
 - **FXAM**：检查
 - **FUCOM**：无序浮点数比较；可加P或PP：**FUCOMP**，**FUCOMPP**
 - **FCOMI/ FUCOMI**：Pentium Pro新增，比较并设置标志寄存器，可加P：**FCOMIP**，**FUCOMIP**

79

比较指令

- **FCOM**：浮点数比较
 - 格式：**FCOM** 或 **FCOM ST(n)** 或 **FCOM mem**
 - 功能：比较栈顶浮点数与寄存器操作数或内存操作数。
 - 注意：如果不含操作数，则**ST(0) - ST(1)**。
- **FCOMP**：浮点数比较并弹出
 - 格式：**FCOMP**或**FCOMP ST(n)**或**FCOMP mem**
- **FCOMPP**：浮点数比较并2次弹出
 - 格式：**FCOMPP**

80

比较指令

- **FICOM**：整数比较
 - 格式：**FICOM mem**
 - 功能：比较栈顶浮点数与内存中的整数比较。
- **FICOMP**：整数比较并弹出
 - 格式：**FCOMP mem**
- **FTST**：测试
 - 格式：**FTST**
 - 功能：执行**ST(0)-0.0**，比较结果被编码与状态寄存器中的条件码。

81

比较指令

- **FXAM**：检查
 - 格式：**FXAM**
 - 功能：检测栈顶，并修改条件码来指示栈顶内容是否为正数、负数或规格化数等。
- **FUCOM / FUCOMP / FUCOMPP**：浮点数无序比较
 - 格式：**FUCOM** 或 **FUCOM ST(n)**
FUCOMP 或 **FUCOMP ST(n)**
FUCOMPP 或 **FUCOMPP ST(n)**
 - 功能：比较栈顶浮点数与寄存器操作数。如果不含操作数，则**ST(0) - ST(1)**。
 - 注意：无序比较检查要比较的数值的类别。

82

比较指令

- **FUCOM** 指令执行的操作与 **FCOM** 指令的相同。唯一的差异在于：
 - **FUCOM** 指令仅在一个或两个操作数是 **SNaN**或是不支持的格式时，才触发算术操作数无效异常 (#IA)；**QNaN** 导致将条件代码标志设置为无序，但不导致生成异常。
 - 两个操作数中至少有一个是任何种类的 **NaN** 值，或是不支持的格式时，**FCOM** 指令触发操作无效异常。
- **QNaN** (Quiet NaN) 和 **SNaN** (Singaling NaN)：
 - **QNaN**的尾数部分最高位定义为1 (1.1xxx)，**SNaN**则定义为0 (1.0xxx)。
 - **QNaN**一般表示未定义的算术运算结果，最常见的莫过于除0运算；**SNaN**一般被用于标记未初始化的值，以此来捕获异常。

83

比较指令

- **FCOMI / FCOMIP / FUCOMI / FUCOMIP**：(Pentium Pro新增) 浮点数比较并设置标志寄存器
 - 格式：**FCOMI ST, ST(n)** **FCOMIP ST, ST(n)**
FUCOMI ST, ST(n) **FUCOMIP ST, ST(n)**
 - 功能：执行**ST-ST(n)**操作，并根据比较结果设置标志寄存器中**ZF**，**PF**，**CF**。
 - 注意1：操作码中的**P**表示弹出操作，**U**表示支持无序比较。
 - 注意2：当栈下溢出时，FPU状态寄存器中的条件码**C1**为1，否则为0；**C0**、**C2**和**C3**不受影响。

84

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

85

超越计算

- 超越运算指令包括：
 - **FPTAN**: 求正切
 - **FPATAN**: 求反正切
 - **FSIN**: 求正弦值
 - **FCOS**: 求余弦值
 - **FSINCOS**: 求正弦和余弦值
 - **F2XM1**: 计算 $2^x - 1$
 - **FYL2X**: 计算 $Y \log_2 X$
 - **FYL2XP1**: 计算 $Y \log_2 (x+1)$

86

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

87

常数操作

- 协处理器包括返回常数到栈顶的指令。

<i>Instruction</i>	<i>Constant Pushed in ST</i>
FLDZ	+ 0.0
FLD1	+ 1.0
FLDPI	π
FLDL2T	$\log_2 10$
FLDL2E	$\log_2 e$
FLDLG2	$\log_{10} 2$
FLDLN2	$\log_e 2$

88

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

89

协处理器控制指令

- 协处理器的控制指令用于初始化、异常处理和任务切换。
- 所有的控制指令都有2种形式。
 - 例如，**FINIT**和**FNINIT**都实现了对协处理器的初始化，但**FNINIT**不产生任何等待状态，而**FINIT**产生等待状态。微处理器通过测试协处理器上的**BUSY**引脚来等待**FINIT**指令。

90

协处理器控制指令

- **FINIT / FNINIT**: 复位操作。
- **FSETPM**: 改变协处理器的寻址模式为保护寻址模式。
- **FLDCW**: 将由操作数寻址的字装入控制寄存器。
- **FSTCW**: 将控制寄存器中的内容存入长度为一个字长的内存操作数中。

91

协处理器控制指令

- **FSTSW AX**: 将控制寄存器中的内容复制到AX。8087不支持。
- **FCLEX**: 清除状态寄存器中的“错误”标志和“忙”标志。
- **FSAVE**: 将机器的全部状态写入内存。
- **FRSTOR**: 从内存复原机器状态。
- **FSTENV**: 存储协处理器的环境。

92

协处理器控制指令

- **FINCSTP**: 堆栈指针加1。
- **FDECSTP**: 堆栈指针减1。
- **FFREE**: 通过把目的寄存器的标记改变为空来释放寄存器，但不影响寄存器的内容。
- **FNOP**: 浮点处理器的NOP。
- **FWAIT**: 使微处理器等待协处理器完成一个操作。FWAIT指令应该用在微处理器访问被协处理器影响的内存数据之前。

93

指令系统

- 数据传送指令
- 算术运算指令
- 比较指令
- 超越计算
- 常数操作
- 协处理器控制指令
- 协处理器指令

94

协处理器指令

- 80387、80487SX、80486、Pentium~Core2均包含如下新增指令：
 - FCOS、FPREM1、FSIN、FSINCOS
 - FUCOM / FUCOMP / FUCOMPP
- Pentium Pro~Core2均包含如下新增指令：
 - FCMOV、FCOMI
- 算术协处理器的指令系统见表14-9。

95

本章内容

- 算术协处理器的数据格式
- 80X87的结构
- 指令系统
- 算术协处理器编程
- MMX技术简介
- SSE技术概述

96

算术协处理器编程

- 计算圆的面积
- 求谐振频率
- 使用二次方程求根
- 使用存储器数组存储结果
- 将单精度浮点数转换成字符串

97

计算圆的面积

- 例，计算5个圆的面积 ($A=\pi R^2$)。

```

RAD      DD      2.34
          DD      5.66
          DD      9.33
          DD      234.5
          DD      23.4
AREA     DD      5 DUP(?)

FINDA    PROC    NEAR

          FLDPI
          MOV     ECX,0
          .REPEAT
              FLD     RAD[ECX*4]
              FMUL    ST,ST(0)
              FMUL    ST,ST(1)
              FSTP    AREA[ECX*4]
              INC     ECX
          .UNTIL    ECX = 5
          FCOMP
          RET

FINDA    ENDP
```

98

本章小结

- 算术协处理器的数据格式
- 80X87的结构
- 指令系统
- 算术协处理器编程
- MMX技术简介
- SSE技术概述
- 要求
 - 掌握协处理器的数据格式、结构。
 - 了解协处理器的指令系统。
 - 掌握基本的协处理器编程。

99

作业

- 习题17，习题23，习题43，习题45。

100