

CHAPTER 14

The Arithmetic Coprocessor

and MMX Technology

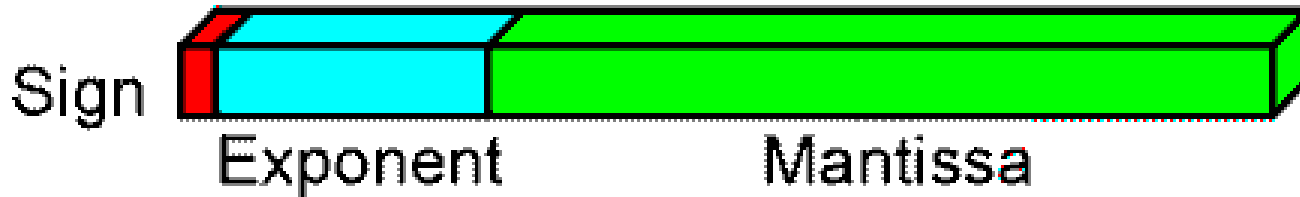
同济大学计算机系

熊齐邦

2007年春

§ 14-1 DATA FORMATS FOR THE **ARITHMETIC COPROCESSOR**

Floating-point Numbers



- Components of a floating point number

Sign: 0=pos(+); 1=neg(-)

Exponent: (with Constant bias added)

Mantissa: $1 \leq M < 2$

- Number = (Mantissa) * $2^{(\text{exponent}-\text{bias})}$

- Example: 10.25 (decimal)

= 1010.01 (binary)

= + 1.01001 * 2^3 (normalized)

Floating-point Format

- **Single-precision:** 32 bits

- sign, 8 exponent (bias 7Fh), 23 mantissa, mantissa first digit is implied

- **Double-precision:** 64 bits

- sign, 11 exponent (bias 3FFh), 52 mantissa, mantissa first digit is implied

- **Extended-precision:** 80 bits

- sign, 15 bits exponent (bias 3FFFh), 64 bits mantissa, mantissa first digit is explicit

Converting to Floating-point Form

1. Convert the decimal number into binary
 2. Normalize the binary number
 3. Calculate the biased exponent
 4. Store the number in floating point format
- EXAMPLE 14-3 (*to 32-bit Floating-point Form*):
 1. $100.25 = 01100100.01$
 2. $1100100.01 = 1.10010001 \times 2^6$
 3. $110 + 01111111(7Fh) = 10000101(85h)$
 4. Sign = 0
Exponent = 10000101
Mantissa = 100 1000 1000 0000 0000 0000

Converting from Floating-point Form

(EXAMPLE 14-4)

1. Sign = 1
Exponent = 1000,0011
Mantissa = 100,1001,0000,0000,0000
2. 100 = 1000,0011 - 0111,1111 (convert biased exponent)
3. 1.1001001×2^4 (a normalized binary number)
4. 11001.001 (a de-normalized binary number)
5. -25.125 (a decimal number)

§ 14-2 THE 80X87

ARCHITECTURE

The 80x87 Floating Point Unit

- On-chip hardware for fast computation on floating point numbers
- FPU operations run in parallel with integer operations
- Historically implemented as separate chip
 - ❑ 8087-80387 Coprocessor
- 80486DX-Pentium III contain their own internal and fully compatible versions of the 80387
- CPU/FPU Exchange data through memory
 - ❑ X87 always converts Single and Double to Extended-precision

Arithmetic Coprocessor Architecture

- Control Unit

- ❑ interfaces the coprocessor to the microprocessor-system data bus

- Numeric Execution Unit (NEU)

- ❑ executes all coprocessor instructions

- NEU has eight-register stack

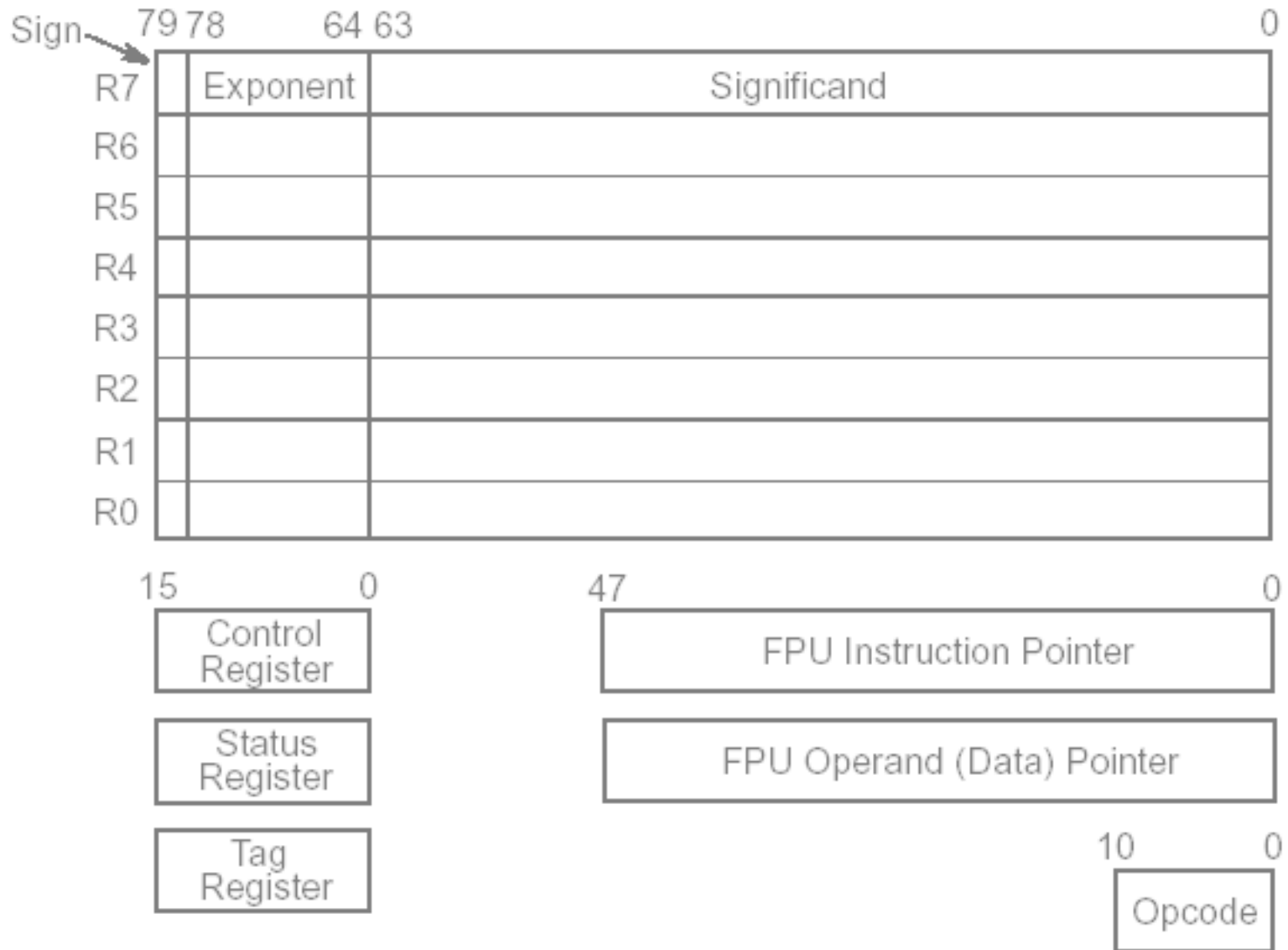
- ❑ each register is 80-bit wide (extended-precision numbers)

- data appear as any other form when they reside in the memory system

- ❑ holds **operands** for arithmetic instructions and the **results** of arithmetic instructions

- **Other registers** - status, control, tag, exception

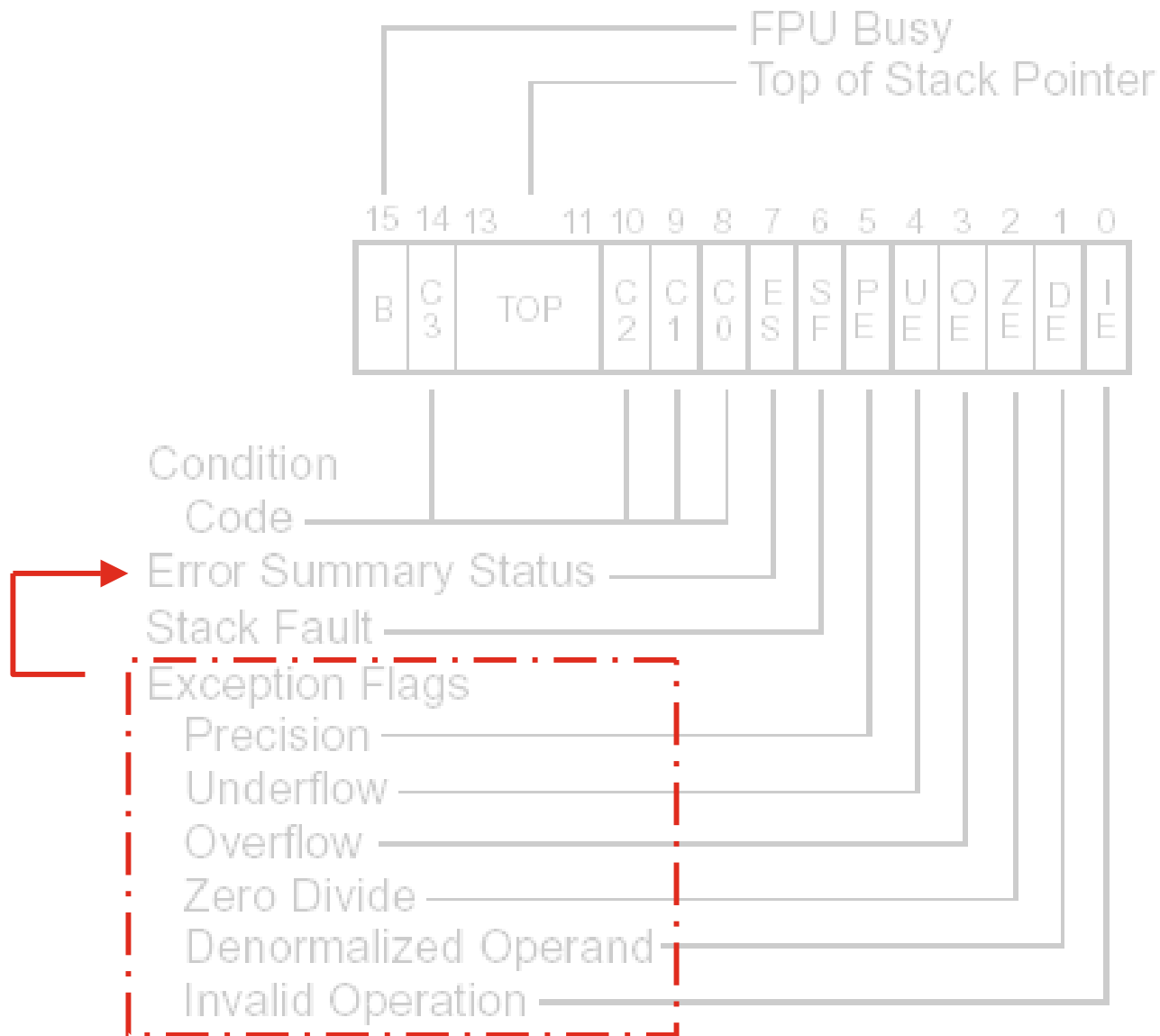
FPU Execution Environment



FPU Data Register Stack

- The eight 80-bit data registers are organized as a **stack**
- All coprocessor data are stored in registers in the **10-byte real format**
 - Internally, all calculations are done on the greatest precision numbers
- Instructions that transfer numbers between memory and coprocessor *automatically* convert numbers to and from the 10-byte real format

FPU Status Register



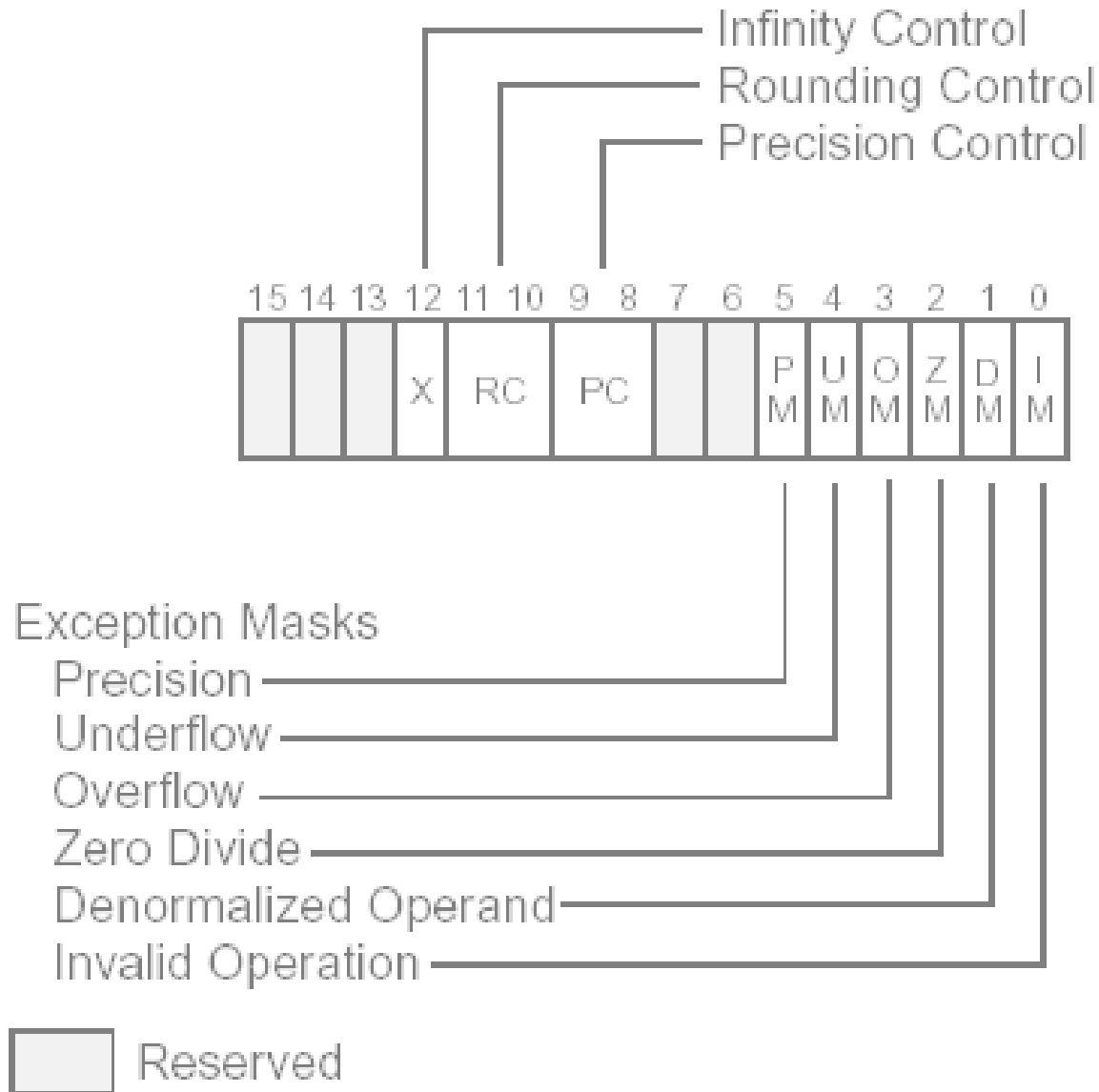
Exception FLAGS

- The six exception flags IE, DE, ZE, OE, UE, and PE (bits 0 through 5) of the status word ***indicate*** that one or more floating point exceptions has been detected since the bits were last cleared.
 - Each of the exception flags can be masked by an exception mask bit in the FPU control word.
- The exception summary status (ES) flag (bit 7) is set when any of the unmasked exception flags are set.

Condition Code Flags

- The four FPU condition code flags (C0 through C3) indicate the results of floating-point comparison and arithmetic operations.
- These condition code bits are used principally for conditional branching and for storage of information used in exception handling

FPU Control Register



Precision Control Field

- The precision-control (PC) field (bits 8 and 9 of the FPU control word) determines the precision (64, 53, or 24 bits) of floating-point calculations made by the FPU

Precision	PC Field
Single Precision (24-Bits*)	00B
Reserved	01B
Double Precision (53-Bits*)	10B
Extended Precision (64-Bits)	11B

* Includes the implied integer bit.

Rounding Control Field

- The rounding control (RC) field of the FPU control register (bits 10 and 11) controls how the results of floating-point instructions are rounded.

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).
Round down (toward $-\infty$)	01B	Rounded result is close to but no greater than the infinitely precise result.
Round up (toward $+\infty$)	10B	Rounded result is close to but no less than the infinitely precise result.
Round toward zero (Truncate)	11B	Rounded result is close to but no greater in absolute value than the infinitely precise result.

FPU Tag Register

- The 16-bit tag word (refer to Figure 7-11) indicates the contents of each the 8 registers in the FPU data-register stack (one 2-bit tag per register).
 - ❑ The tag codes indicate whether a register contains a valid number, zero, or a special floating-point number (NaN, infinity, denormal, or unsupported format), or whether it is empty.



TAG Values

00 — Valid

01 — Zero

10 — Special: invalid (NaN, unsupported), infinity, or denormal

11 — Empty

Not a Number

||

§ 14-3 INSTRUCTION SET

Sample 80x87 FPU OPCODES

- FLD: Load
 - ❑ Load fp memory data into an fp reg
- FST: Store
 - ❑ Store a copy of fp reg to memory address
- FADD: Addition
- FSUB: Subtraction
- FMUL: Multiplication
- FDIV: Division
 - ❑ Divides the destination by the source
- FDIVR: **Reverse** Division
 - ❑ Divides the source by the destination

Sample 80x87 FPU OPCODES (Cont'd)

- FSQRT: Square Root
- FABS: Absolute Value
- FSIN: Sine (uses radians)
- FCOS: Cosine (uses radians)
- FYL2X: Calculates $Y * \log_2(X)$
- FYL2XP1: Calculates $Y * \log_2(X+1)$

❖ Format: **OPCODE [destination,] source**

❖ Restrictions: **One register must be ST0**

❖ Use of ST0 as destination is implicit when not specified otherwise

FPU Data Register Stack



All addressing of the data registers is relative to the register on the top of the stack

The register number of the current top-of-stack register is stored in the TOP field in the x87 FPU status word.

- Load operations decrement TOP by one and load a value into the new top of stack register
- Store operations store the value from the current TOP register in memory

Example x87 FPU Computation

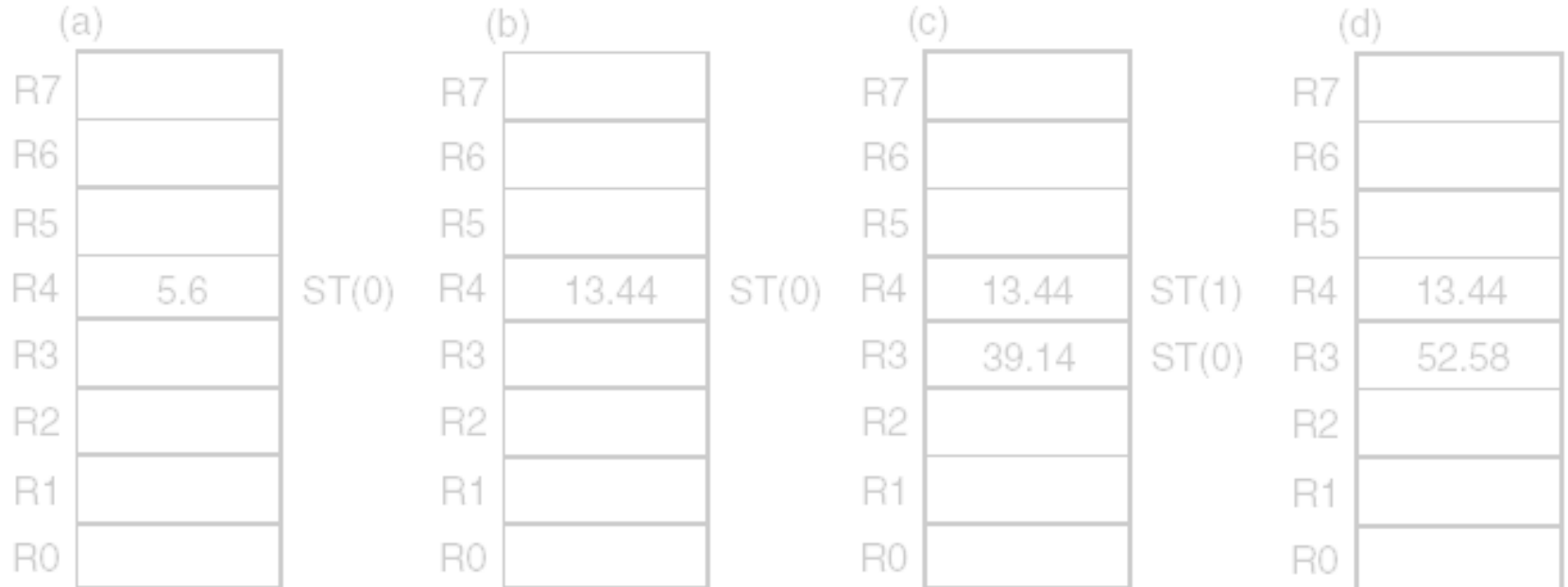
Computation

Dot Product = $(5.6 \times 2.4) + (3.8 \times 10.3)$

Code:

```
FLD  value1 ; (a) value1=5.6  
FMUL value2 ; (b) value2=2.4  
FLD  value3 ; value3=3.8  
FMUL value4 ; (c) value4=10.3  
FADD ST(1) ; (d)
```

[插讲论坛](#)



§ 14-5 INTRODUCTION TO **MMX TECHNOLOGY**

MMX (MultiMedia eXtensions)

- Designed to accelerate multimedia and communication applications
 - motion video, image processing, audio synthesis, speech synthesis and compression, video conferencing, 2D and 3D graphics
- Includes **new instructions and data types** to significantly improve application performance
- Exploits the **parallelism** inherent in many multimedia and communications algorithms
- Maintains **full compatibility** with existing operating systems and applications

MMX (Cont'd)

- Provides a set of basic, general purpose **integer instructions**
- **Single Instruction, Multiple Data (SIMD)** technique
 - ❑ Allows many pieces of information to be processed with a single instruction, providing parallelism that greatly increases performance
- 57 new instructions
- Four new data types
- First available in 1997
- Supported on:
 - ❑ Intel Pentium-MMX, Pentium II, Pentium III (and later)
 - ❑ AMD K6, K6-2, K6-3, K7 (and later)
 - ❑ Cyrix M2, MMX-enhanced MediaGX, Jalapeno (and later)

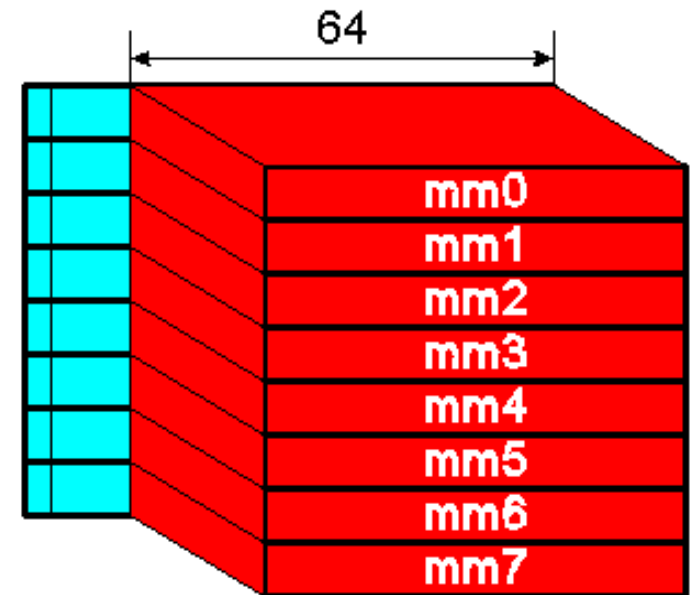
Internal Register Set of MMX Technology

Uses 64-bit *mantissa portion* of 80-bit FPU registers

- This technique is called **aliasing** because the floating-point registers are **shared** as the MMX registers

- ☐ Aliasing the MMX state upon the floating point state **does not preclude** applications from executing *both* MMX technology instructions *and* floating point instructions

- The same techniques used by FPU to interface with the operating system are used by MMX technology



MMX Registers

Data Types

- MMX architecture introduces **new packed**打包的, 压紧的 **data types**

- ☐ Multiple integer words are grouped into a single 64-bit quantity

Eight 8-bit packed bytes (B)



Four 16-bit packed words (W)



Two 32-bit packed doublewords (D)



One 64-bit quadword (Q)



- *Example:* consider graphics pixel data represented as bytes.

- ☐ with MMX, eight of these pixels can be packed together in a 64-bit quantity and moved into an MMX register
- ☐ MMX instruction performs the arithmetic or logical operation on all eight elements in **parallel**

Arithmetic Instructions (1/2)

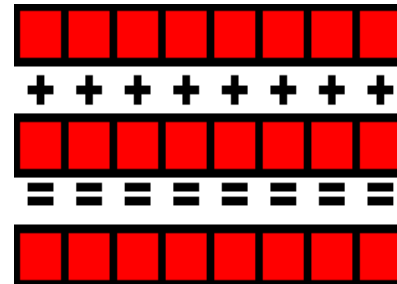
● **PADD(B/W/D):** Addition

PADDB MM1, MM2

adds 64-bit contents of MM2 to MM1,

byte-by-byte, any **carries** generated

are **dropped**, e.g., byte A0h + 70h = 10h



MMX Addition:
Each 8-byte
entity added
in parallel

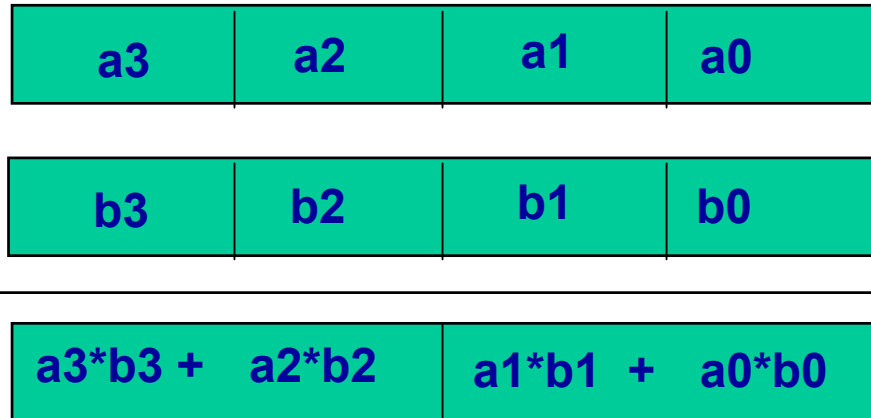
● **PSUB(B/W/D):** Subtraction

Arithmetic Instructions (2/2)

- **PMUL**: Multiplication

- ☐ Multiplies four pairs of 16-bit operands, producing 32 bit result

- **PMADDWD**: Multiply and Add



- ☐ Key instruction to many **signal processing algorithms** like matrix multiplies or FFTs

Logical, Shifting, and Compare Instructions

Logical

PAND: Logical **AND** (64-bit)

POR: Logical **OR** (64-bit)

PXOR: Logical Exclusive OR (64-bit)

Shifting

PSLL(W/D/Q): Packed Shift Left (Logical)

PSRL(W/D/Q): Packed Shift Right (Logical)

PSRA(W/D/Q): Packed Shift Right (Arithmetic)

Compare

PCMPEQ: Compare for Equality

PCMPGT: Compare for Greater Than

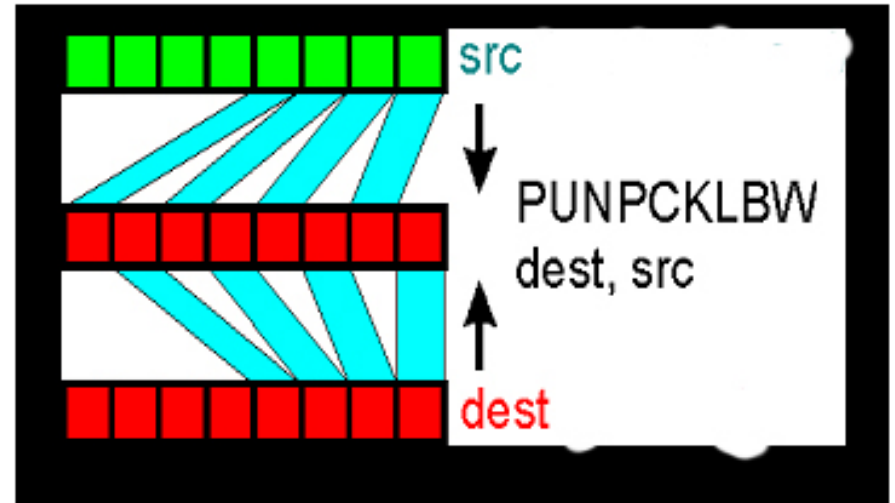
- Sets **Result Register** to ALL 0's or ALL 1's

Conversion Instructions (1/2)

● Unpacking

PUNPCKLBW Reg1, Reg2:

- ❑ Unpacks and interleaves **lower** four Bytes to create four Words.



PUNPCKLWD Reg1, Reg2:

- ❑ Interleaves **lower** two Words to create two Doubles

PUNPCKLDQ Reg1, Reg2:

- ❑ Interleaves **lower** Double to create Quadword

Conversion Instructions (2/2)

[插讲论坛](#)

● Packing

PACKSSDW Reg1, Reg2: Pack Double to Word

- ❑ Four doubles in Reg2:Reg1 compressed to Four words in Reg1

PACKSSWB Reg1, Reg2: Pack Word to Byte

- ❑ Eight words in Reg2:Reg1 compressed to Eight bytes in Reg1

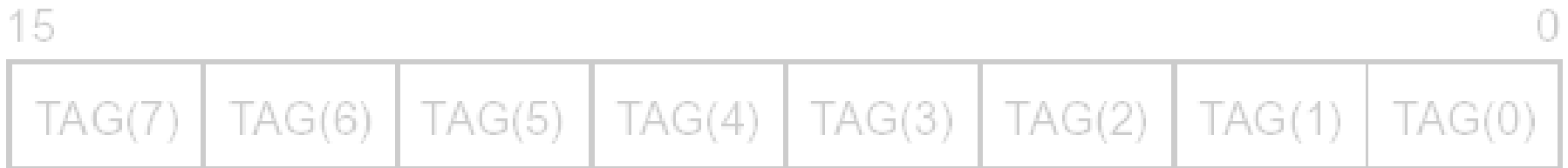
- The pack and unpack instructions are especially important when an algorithm needs **higher precision** in its **intermediate calculations**, e.g., an image filtering

Data Transfer Instructions

- MOVQ Dest, Source : 64-bit move
 - ❑ One or both arguments must be a MMX register
- MOVD Dest, Source : 32-bit move
 - ❑ Zeros loaded to upper MMX bits for 32-bit move

Use of FPU Registers for Storing MMX Data

- The **EMMS** (empty MMX-state) **instruction sets** all the tags in the FPU, so the floating-point registers are listed as ***Empty***



TAG Values

00 — Valid

01 — Zero

10 — Special: invalid (NaN, unsupported), infinity, or denormal

11 — Empty

Use of FPU Registers for Storing MMX Data (Cont'd)

- EMMS must be executed before the return instruction **at the end of any MMX procedure**
 - ❑ Otherwise any *subsequent floating point operation* will cause a floating point interrupt error, potentially crashing your application
- If you **use floating point within MMX procedure**, you must use EMMS instruction before executing the floating point instruction
- Any **MMX instruction** **resets** all FPU tag bits, so the floating-point registers are listed as ***Valid***

Homework

1. Page 604 Problem 13
2. Page 605 Problem 55