Intel Multimedia Instructions (MMX, SSE, SSE2, SSE3, SSSE3 and SSE4)

Priya Periaswamy

Computer Architecture (CSE5302)

Overview

- MMX (**M**ulti**M**edia e**X**tention) Architecture
- MMX Instructions
- SSE (**S**treaming **S**IMD **E**xtensions)
- SSE2
- SSE3
- SSSE3(**S**upplemental **S**treaming **S**IMD **E**xtensions **3**)
- SSE4

MMX Architecture

Why did Intel go for MMX?

To make the common case fast

A wide range of multimedia applications shows many common, fundamental characteristics :

- small integer data types (for example: 8-bit pixels, 16-bit audio samples)
- small, highly repetitive loops
- frequent multiplies and accumulates
- compute-intensive algorithms
- highly parallel operations

so..

The MMX technology focuses

- ➢ to accelerate multimedia, communications and numeric intensive applications
- ➢ To exploit the parallelism inherent in many multimedia and communications algorithms, yet maintains full compatibility with existing operating systems and applications.

MMX technology

MMX technology allowed later Pentium processors to handle multimedia tasks without expensive DSPs
-> lowered the cost of multimedia systems

The highlights of the technology are:

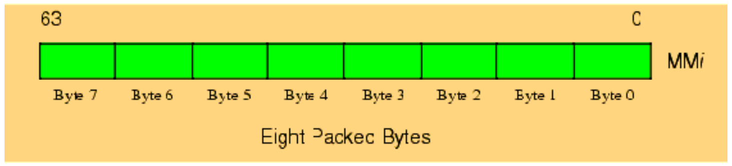
- Packed Data types - small data elements packed together into one register
- Enhanced instruction set - 57 new instructions that operate on all data elements in a register in parallel, in a SIMD fashion
- 8 64-bit wide MMX registers, named MM0 to MM7, that are mapped on the IA floating point registers
- Full IA compatibility

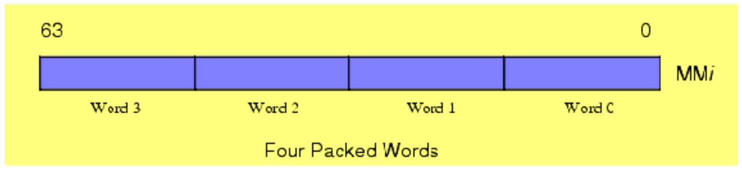
MMX data types

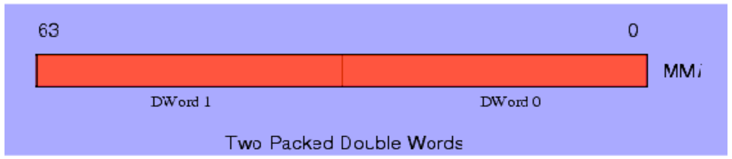
Supports 4 data-types

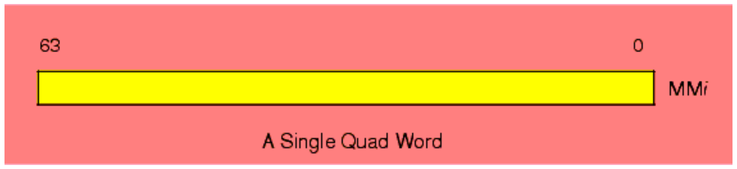
- Packed byte -> 8 bytes packed into one 64-bit quantity
- Packed word -> 4 16-bit words packed into one 64-bit quantity
- Packed Double word -> 2 32-bit double words packed into one 64-bit quantity
- Packed Quad word -> one 64-bit quantity

Each MMX register processes one of these four data types









- Why such data types?

Typical elements are small, 8 bits for pixels, 16 bits for audio, 32 bits for graphics and general computing

Compatibility

- No new exceptions or states are added.
- Aliases to existing FP registers:

The exponent field of the corresponding floating-point register (bits 64-78) and the sign bit (bit 79) are set to ones (1's), making the value in the register a NaN (Not a Number) or infinity when viewed as a floating-point value.

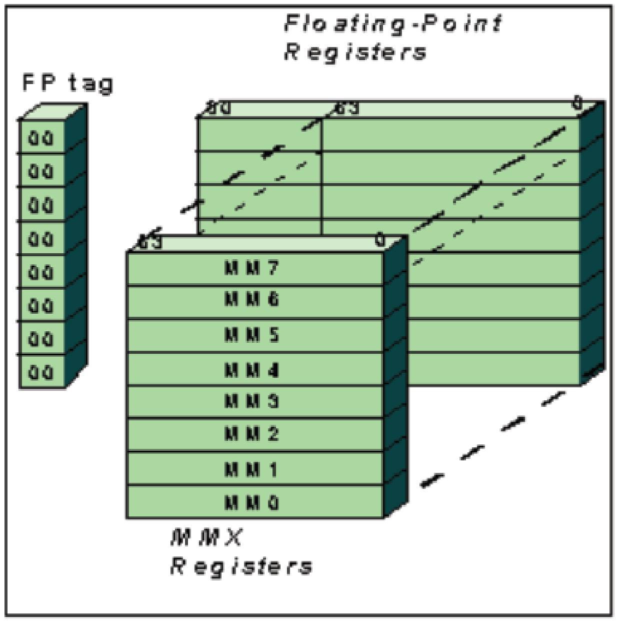
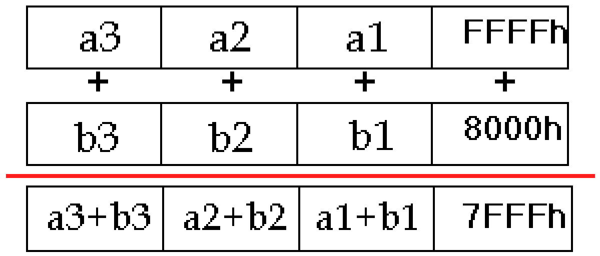


Figure 4. Mapping of MMX Registers to Floating-Point Registers

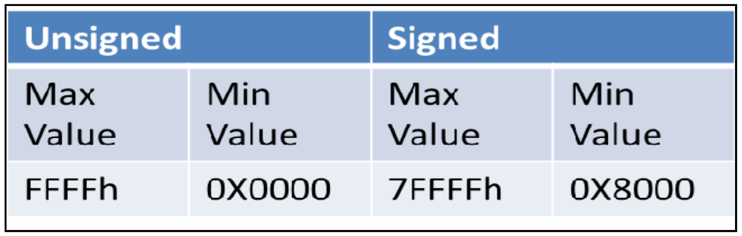
Saturation and wrap-around modes

Wrap-around mode: Result is truncated and only the lower (least significant) bits of the result are returned

PADD[W]: Wrap-around Add



Saturation mode: Results that overflow (from addition) or underflow (from subtraction) are clamped to the largest or the smallest value representable.



Unsigned		Signed	
Max Value	Min Value	Max Value	Min Value
FFFFh	0X0000	7FFFFh	0X8000

This is important for pixel calculations where this would prevent a wrap-around add from causing a black pixel to suddenly turn white while, for example, doing a 3D graphics Gouraud shading loop.

MMX instruction syntax

- All instructions operate on 2 operands : source and destination (except EMMS instruction)
- First operand is destination and second is source
- Instruction overwrites the destination operand

For example, a two-operand instruction OPERATION DEST, SRC
would be decoded as:
DEST = DEST OPERATION SRC
A typical MMX instruction has this syntax:

Prefix: P for Packed

Instruction operation: for example - ADD, CMP, or XOR

Suffix:

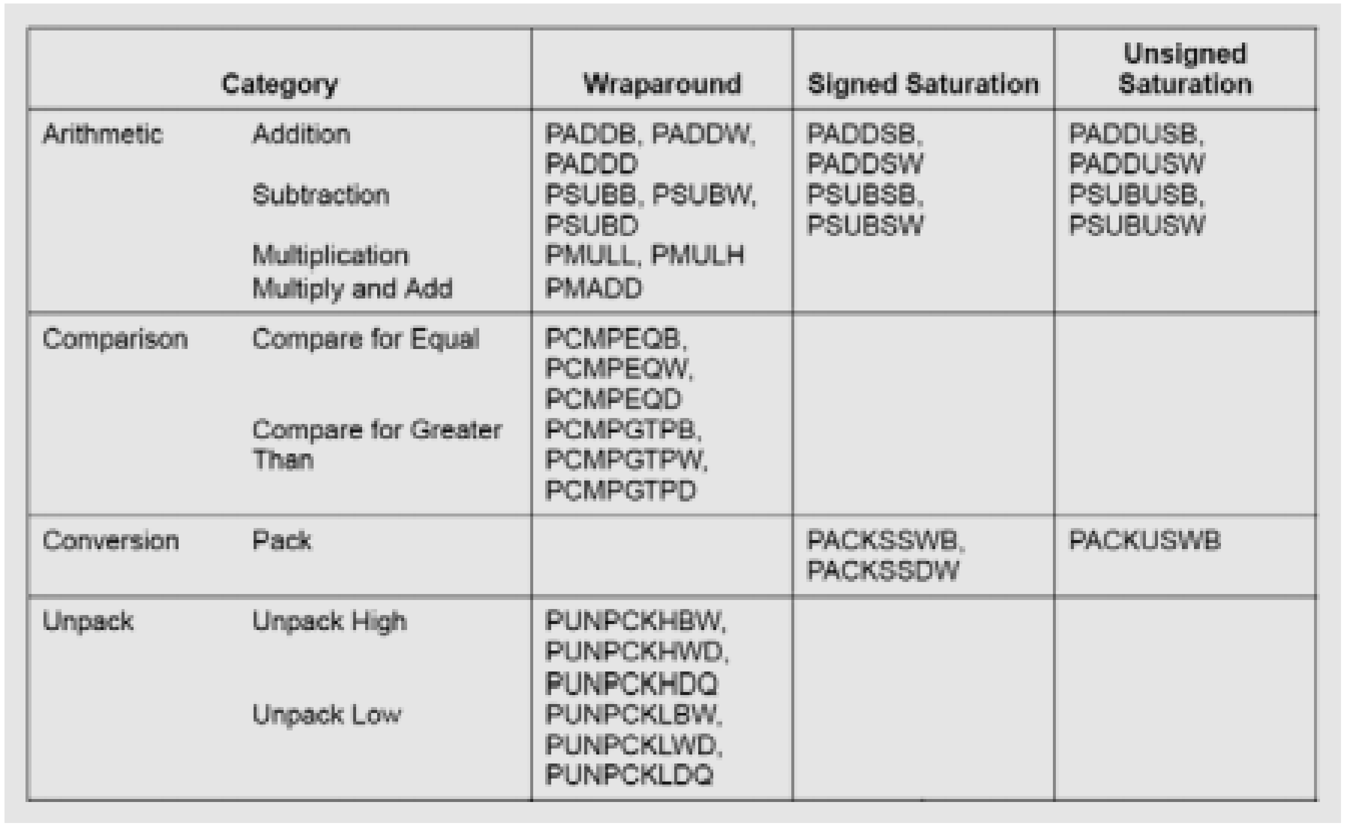
US for Unsigned Saturation

S for Signed saturation

B, W, D, Q for the data type: packed byte, packed word, packed doubleword, or quadword.

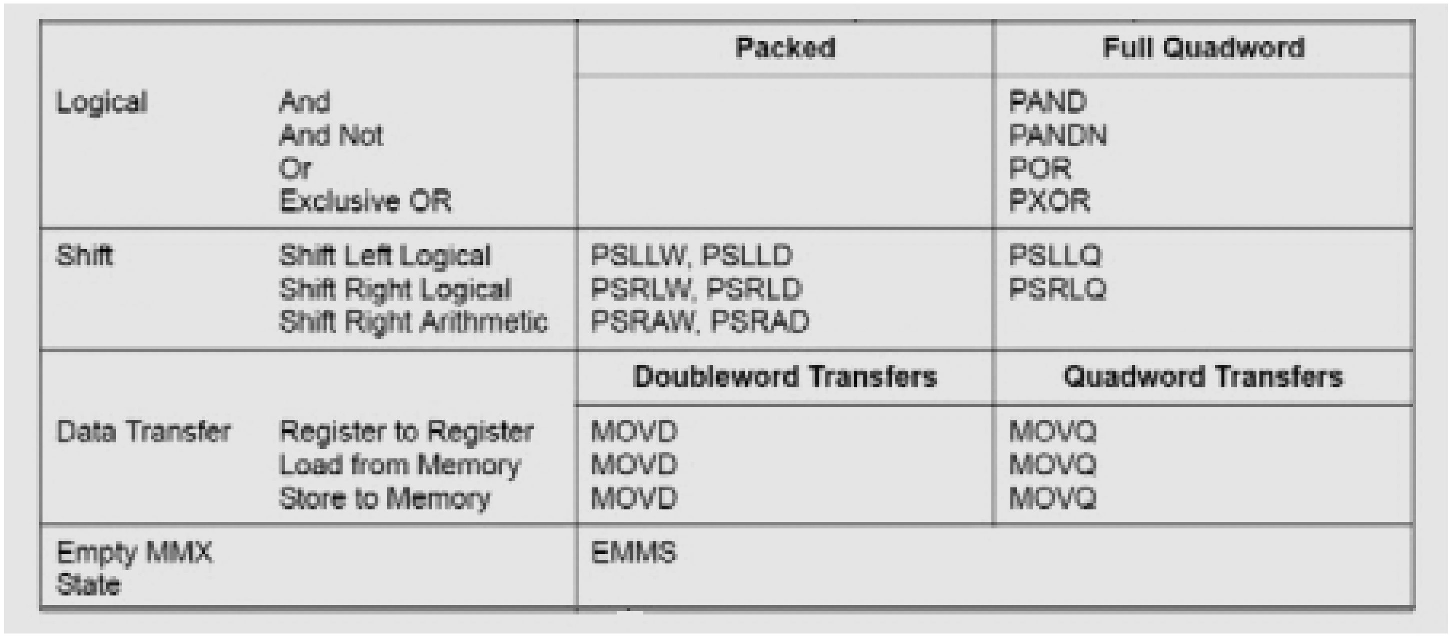
As an example, PADDSB is a MMX instruction (P) that sums (ADD) the 8 bytes (B) of the source and destination operands and saturates the result (S).

MMX instruction set



Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		

MMX instruction set (contd..)



		Packed	Full Quadword
Logical	And And Not Or Exclusive OR		PAND PANDN POR PXOR
Shift	Shift Left Logical Shift Right Logical Shift Right Arithmetic	PSLLW, PSLLD PSRLW, PSRLD PSRAW, PSRAD	PSLLQ PSRLQ
		Doubleword Transfers	**Quadword Transfers**
Data Transfer	Register to Register Load from Memory Store to Memory	MOVD MOVD MOVD	MOVQ MOVQ MOVQ
Empty MMX State		EMMS	

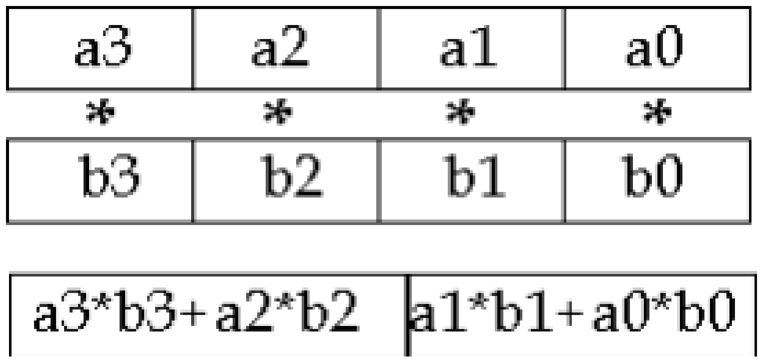
EMMS : To switch back to FP mode safely, the EMMS instruction must be issued.

- mandatory 53 cycle stall
- Empties the MMX state before calling FP routines

MMX instruction examples

PMADD (Packed multiply add)

The PMADD instruction starts from a 16-bit, packed data type and generates a 32-bit packed, data type result

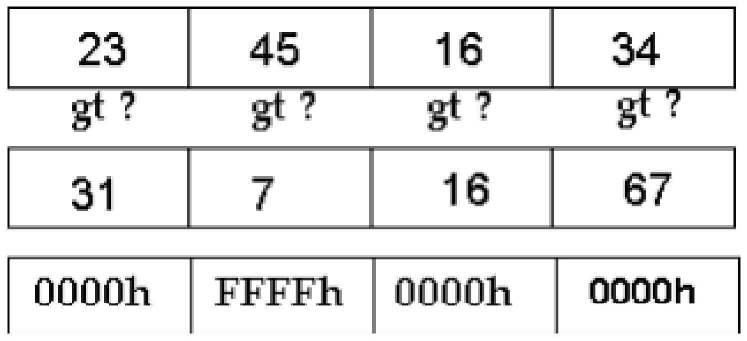


PMADDWD: 16b x 16b -> 32b Multiply Add

Multiply-accumulate operations used in many signal processing algorithms like vector-dot-products, matrix multiplies, FIR and IIR Filters, FFTs, DCTs etc.

MMX instruction examples

PCMPGT[W]: Parallel Compares



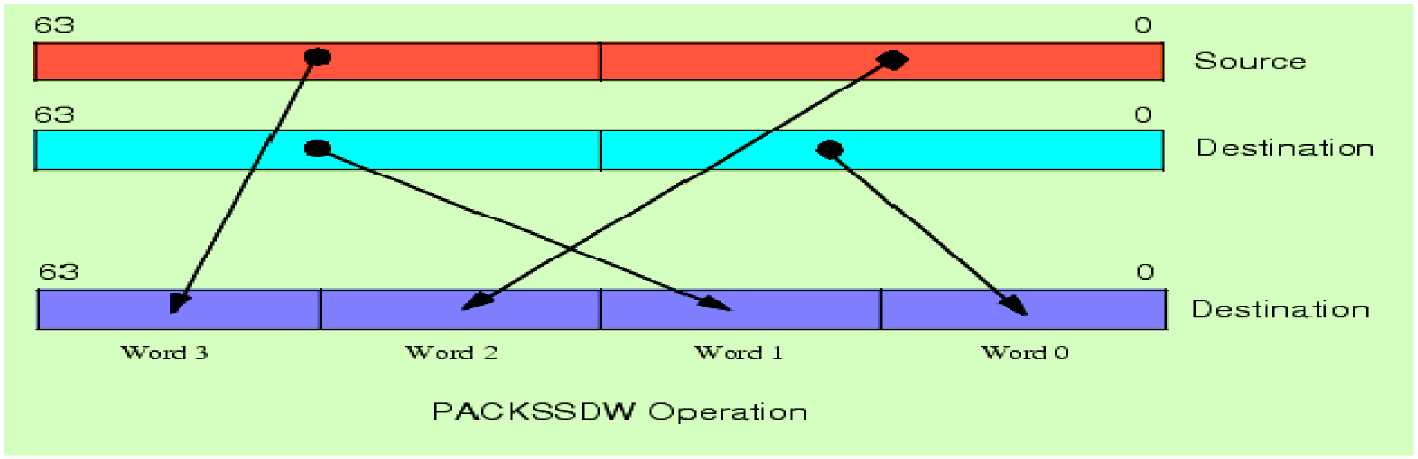
PCMPGT[W]: Parallel Compares

- no new condition code flags
- No existing IA condition code flags affected
- EQ/GT, no LT
- Result can be used as a mask to select elements from different inputs using a logical operation, eliminating branch instructions

MMX instruction examples

Pack

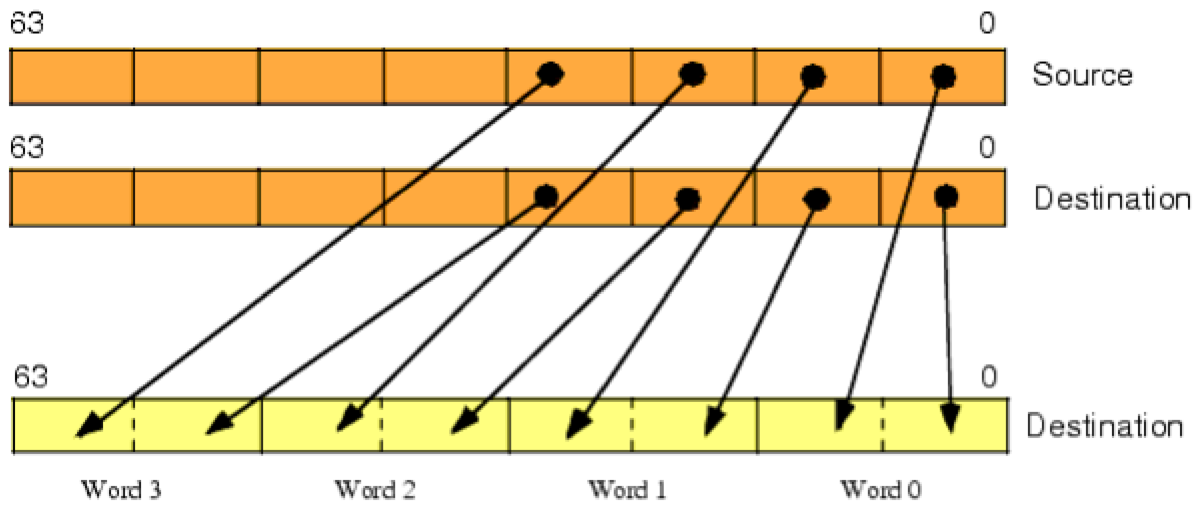
- Important when an algorithm needs higher precision in its intermediate calculations, as in image filtering.
- convert UNICODE to ASCII (ANSI), to translate a 16-bit audio stream to an eight-bit stream



MMX instruction examples

Unpack

- sequence of smaller, packed, values and translate them into larger values.
- produces a 64-bit result from a single 32-bit result
- two sets of unpack instructions: one set unpacks the data from the L.O. double word of a 64-bit object, the other set of instructions unpacks the H.O. double word of a 64-bit object.

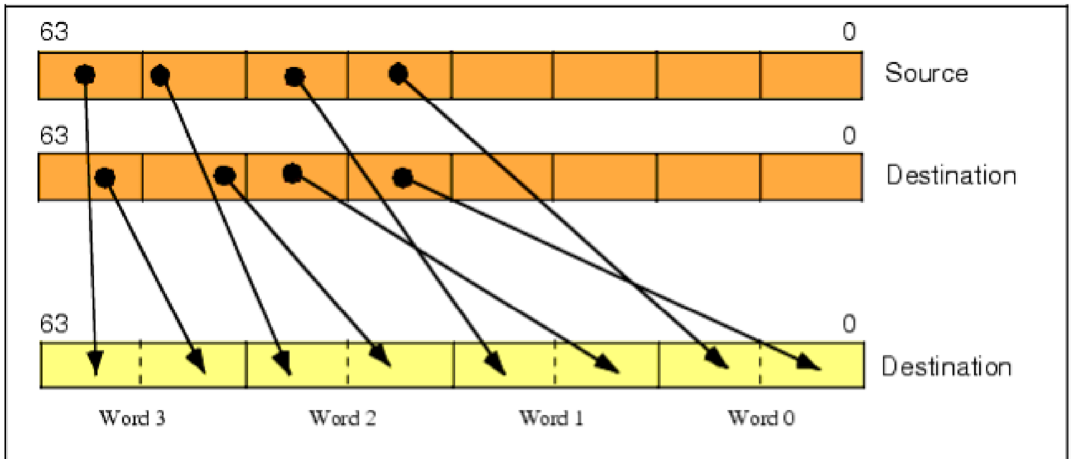


PUNPCKLBW Operation

Unpack from lower order bytes

MMX instruction examples

Unpack (from higher order bytes)

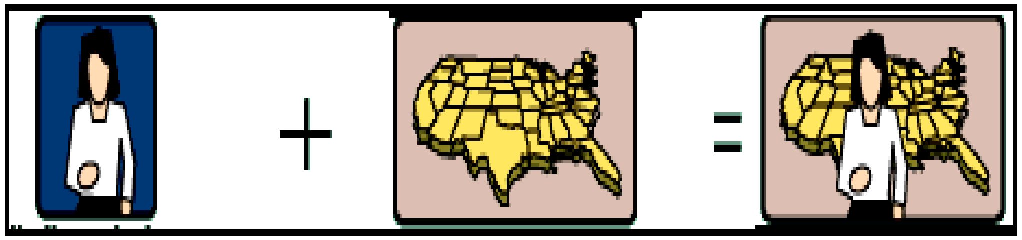


PUNPCKHBW Operation

MMX Application examples

Chroma Keying :

-> conditional selection using the MMX instruction set removes branch mis-predictions, in addition to performing multiple selection operations in parallel

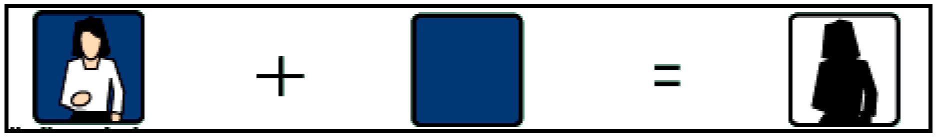


MMX code sequence for performing a conditional select

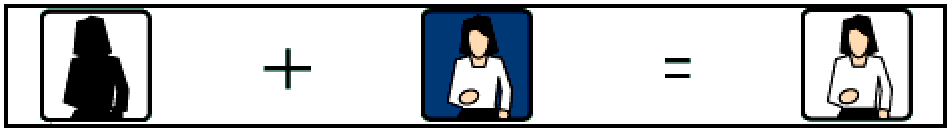
```
Movq        mm3,mem1     //Load eight pixels from woman's image
Movq        mm4,mem2     //Load eight pixels from the map image
Pcmpeqb     mm1,mm3      //generating the selection bit mask
Pand        mm4,mm1      //
Pandn       mm1,mm3
Por         mm4,mm1
```

Chroma Keying (cont..)

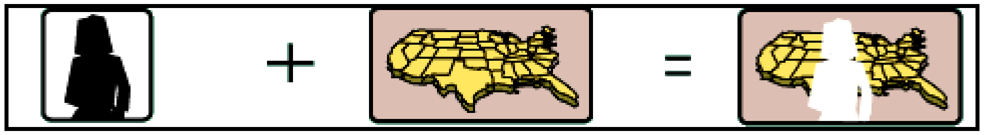
PCMPEQ (packed compare for equality) is performed on the weathercaster and blue-screen images, yielding a bitmask that traces the outline of the weathercaster.



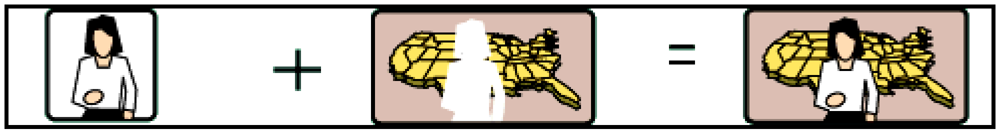
This bitmask image is PANDNed (packed and not) with the weathercaster image, yielding the first intermediate image: now the weathercaster has no background behind her.



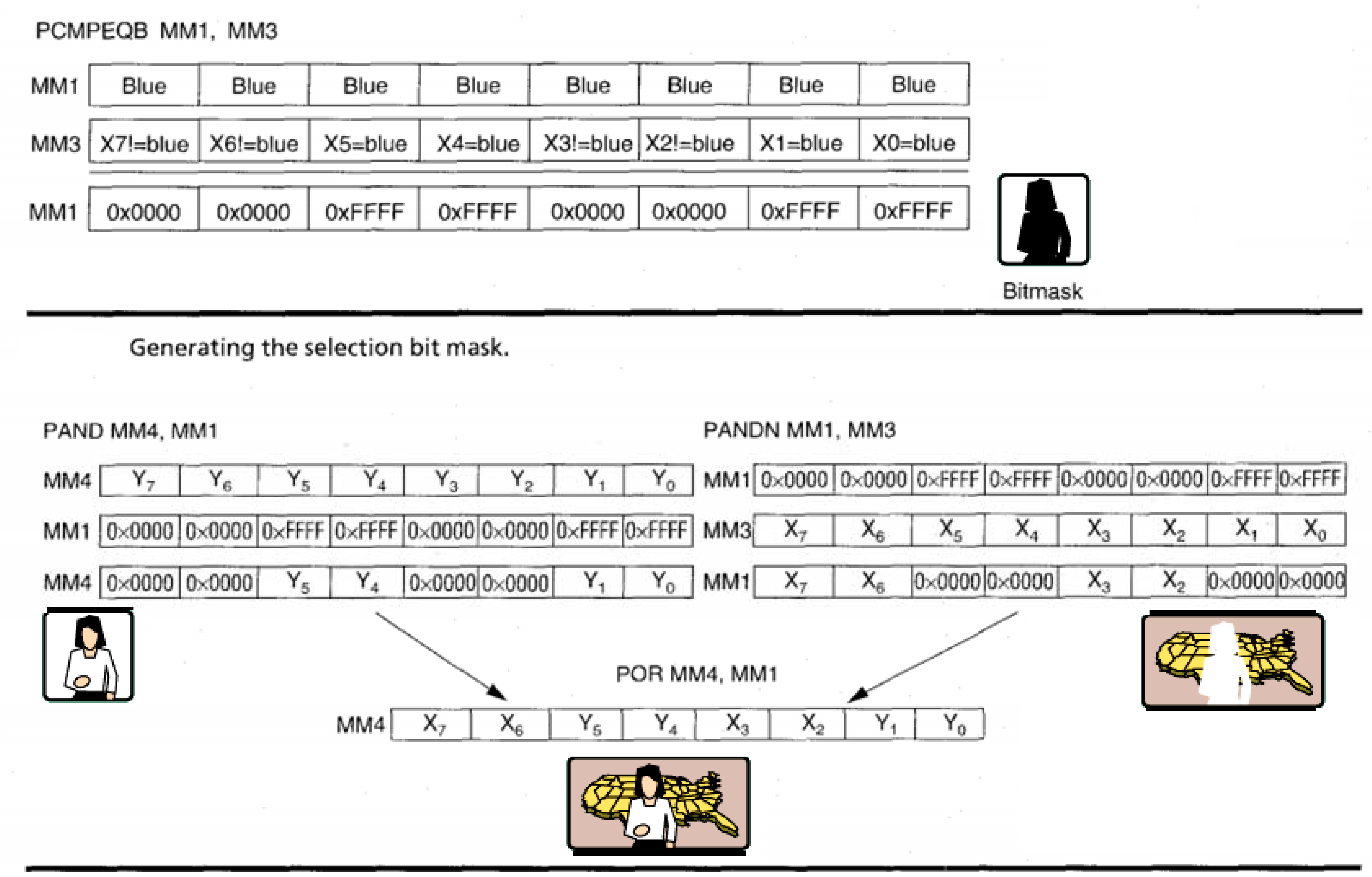
The same bitmask image is PANDed (packed and) with the weather map image, yielding the second intermediate image.



The two intermediate images are PORed (packed or) together, resulting in final composite of the weathercaster over weather map



Chroma Keying (cont..)



PCMPEQB MM1, MM3

MM1	Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue
MM3	X7!=blue	X6!=blue	X5=blue	X4=blue	X3!=blue	X2!=blue	X1=blue	X0=blue
MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF

Generating the selection bit mask.

PAND MM4, MM1

MM4	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
MM4	0x0000	0x0000	Y_5	Y_4	0x0000	0x0000	Y_1	Y_0

PANDN MM1, MM3

MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
MM3	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
MM1	X_7	X_6	0x0000	0x0000	X_3	X_2	0x0000	0x0000

POR MM4, MM1

MM4	X_7	X_6	Y_5	Y_4	X_3	X_2	Y_1	Y_0

MMX technology

Merits:

- According to Intel, an MMX microprocessor runs a multimedia application up to 60% faster. In addition, it runs other applications about 10% faster
- In a Pentium processor architecture, the MMX code processes eight pixels in 3 cycles, ie., 3/8 cycles per pixel. Regular IA integer instruction requires 3 cycles per pixel.
- Advantage in instruction count resulting from the multiple parallel operations performed in each SIMD MMX instruction
- Exploiting parallelism between instructions via the advanced micro architectural implementations of Intel processors

Demerits:

In MMX

• An application cannot perform MMX and floating-point operations simultaneously.

• AN expensive EMMS instruction need to be executed to change the state from MMX to FP operations

SSE (Streaming SIMD Extensions) technology

- Introduced in Pentium III processor
- 8 new 128-bit SIMD floating-point registers (XMM0 - XMM7)
- 50 new instructions that work on packed floating-point data
- 12 new instructions that extend the MMX instruction set. Eg., **PAVG**
- Most SSE instructions require 16-aligned addresses

Since media apps are
-> inherently parallel
-> wide dynamic range, hence floating-point based
-> regular memory access patterns
-> data independent control flow

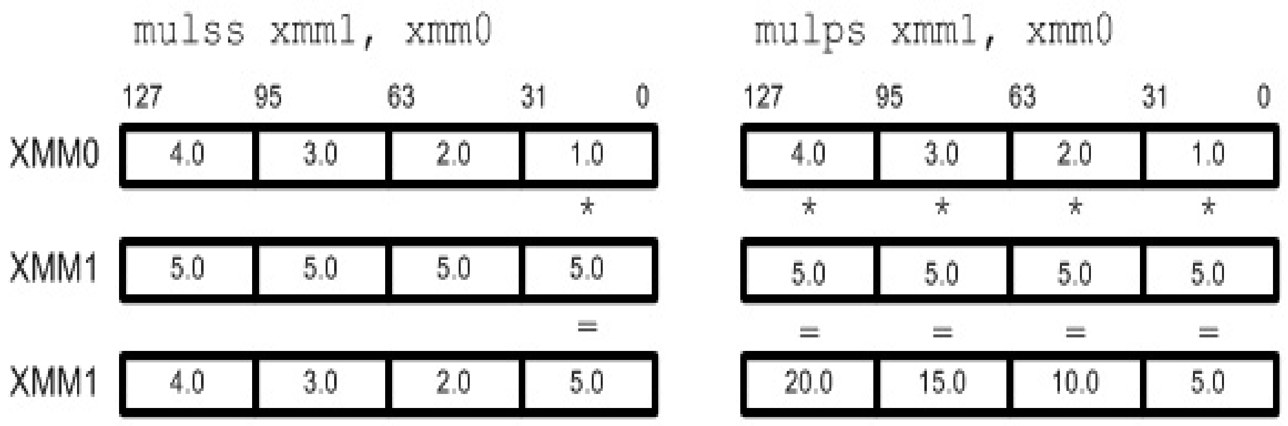
Programmers can mix and match data types

SSE (Streaming SIMD Extensions) instructions

Defines 2 types of instructions

- **Scalar ->** operates on the least-significant data element (bit 0~31)
- **Packed ->** operates on all four elements in parallel

SSE instructions have a suffix -ss for scalar operations (Single Scalar) and -ps for packed operations (Parallel Scalar).



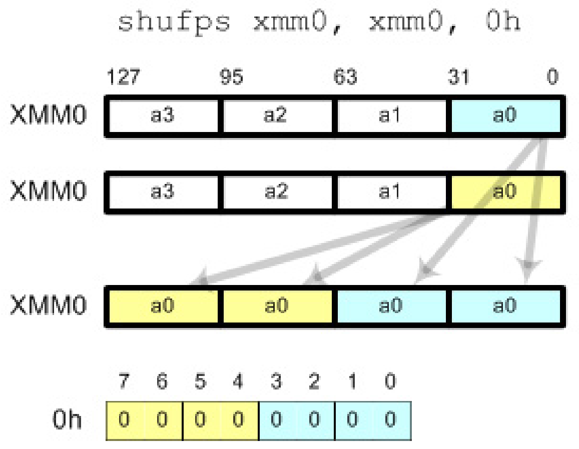
Note that upper 3 elements in xmm0 for scalar operation remain unchanged.

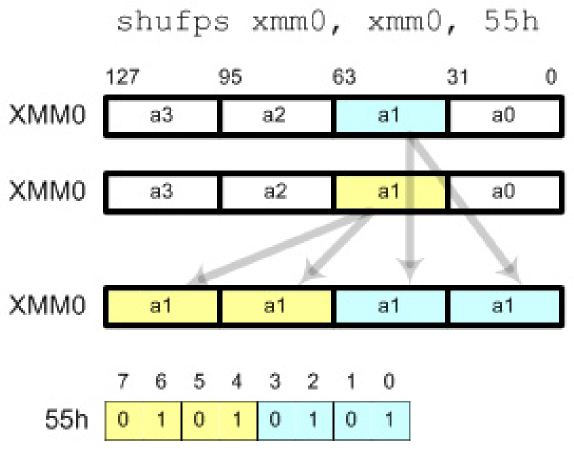
SSE Instruction - shuffle

- Requires 2 operands and 1 mask
- Selects 2 elements from each operand (register) based on the mask.
- Frequent usages of **shufps** are broadcast, swap and rotate.

Application in Broadcasting:

It copies all 4 fields with a single data element. The possible masks are 00h (copies LS element), 55h (copies 2nd element) , AAh (copies 3rd element), FFh (copies 4th element)





SSE2

- First introduced on the Intel Pentium 4 and Intel Xeon processors
- Work with double precision floating-point values (64 bit) as well as single precision (32 bits)
- Means to accelerate operations typical of 3D graphics, real-time physics, spatial (3D) audio, video encoding/decoding, encryption, and scientific application.

-> SSE instruction set worked on 32-bit floating-point data elements, processing 4 of them in parallel (4x32 = 128 bit)

SSE2 Instruction set:

- Can only be executed on Intel 64 and IA-32 processors
- 144 new instructions
- MMX instructions can work on 128-bit data blocks -> doubling parallelism
- Support for these instructions can be detected with the CPUID instruction

The instructions are divided into four subgroups (note that the first subgroup is further divided into subordinate subgroups):

- Packed and scalar double-precision floating-point instructions
- Packed single-precision floating-point conversion instructions
- 128-bit SIMD integer instructions
- Cacheability-control and instruction ordering instructions

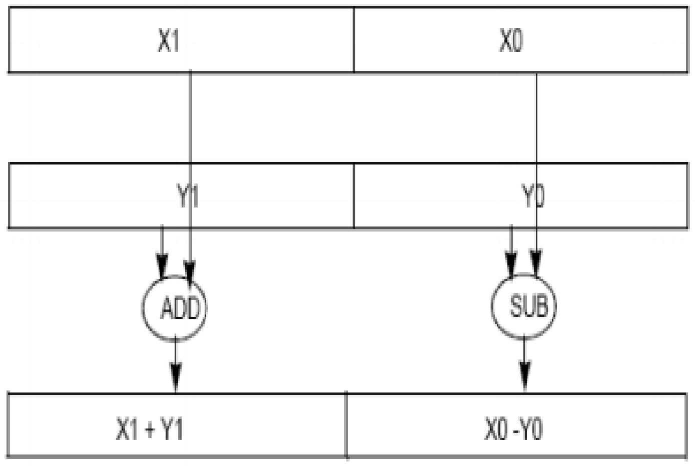
SSE3

SSE3: Streaming SIMD Extensions 3

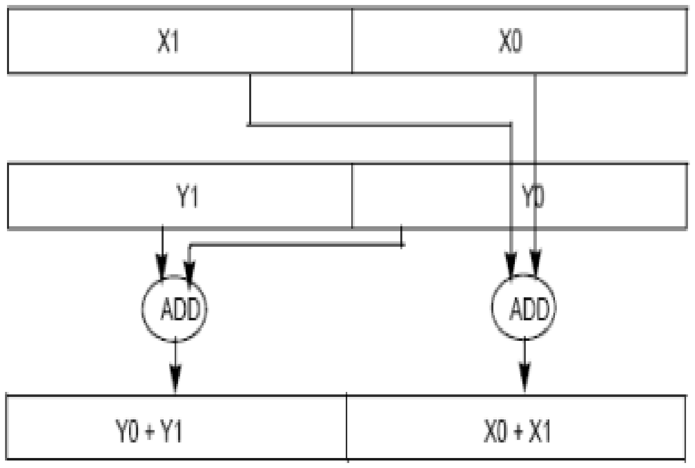
- 13 new instructions
- Some instructions does horizontal operations (operating across a single register instead of down through multiple registers) and asymmetric processing
- Unaligned access instructions are new type of instructions.
- Process control instructions to boost performance with Intel's hyper-threading
- feature.

SSE3

- Asymmetric processing
- Horizontal data movement



ADDSUBPD



HADDPD

SSSE3 and SSE4

SSE3: Supplemental Streaming SIMD Extensions 3

- SIMD instructions added with the Pentium Xeon and Core 2 processors
- 32 new instructions designed to accelerate a variety of multimedia and signal processing applications

SSE4:

- SSE4 comprises of two sets of extensions
  - SSE4.1: targeted to improve the performance of media,imaging and 3D graphics. It also adds instructions for improving compiler vectorization and significantly
  increase support for packed dword computation. It has 47 new instructions.
  - SSE4.2: improves performance in string and text processing. It has 7 new instructions.

• SSE4 instructions do not use MMX registers. Two of the SSE4.2 instructions operate on general-purpose registers; the rest of SSE4.2 instruction and SSE4.1 instructions operate on XMM registers

MMX technology

- http://www.engr.uconn.edu/~zshi/course/cse5302/ref/peleg96mmx.pdf
- Intel Developer Service's - MMX Technology Technical Overview
- Chapter Eleven The MMX Instruction Set, The Art of Assembly
- About MMX/SSE/SSE2 by S Tommesani
- Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1