Intel
MMX™ Technology
Overview

March 1996

Order Number: 243081-002

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to their specifications known as errata.

*Other brands and names are the property of their respective owners.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing product orders.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:
Intel Corporation
P.O. Box 7641
Mt. Prospect IL 60056-764
or call 1-800-879-4683

CONTENTS

INTRODUCTION

The volume and complexity of data processed by today’s personal computer are increasing exponentially, placing incredible demands on the microprocessor. New communications, games and “edutainment” applications feature video, 3D graphics, animation, audio and virtual reality, all of which demand ever increasing levels of performance.

Intel’s MMX™ technology is designed to accelerate multimedia and communications applications. The technology includes new instructions and data types that allow applications to achieve a new level of performance. It exploits the parallelism inherent in many multimedia and communications algorithms, yet maintains full compatibility with existing operating systems and applications.

MMX technology is the most significant enhancement to the Intel Architecture since the Intel386™ processor, which extended the architecture to 32 bits. Processors enabled with MMX technology will deliver enough performance to execute compute-intensive communications and multimedia tasks with headroom left to run other tasks or applications. They allow software developers to design richer, more exciting applications for the PC. The volume of MMX technology-enabled systems will grow rapidly in 1997 as the technology is incorporated into multiple processor generations from Intel.

The definition of MMX technology resulted from a joint effort between Intel’s microprocessor architects and software developers. A wide range of software applications was analyzed, including graphics, MPEG video, music synthesis, speech compression, speech recognition, image processing, games, video conferencing and more. These applications were broken down to identify the most compute-intensive routines, which were then analyzed in details using advanced computer-aided engineering tools. The results of this extensive analysis showed many common, fundamental characteristics across these diverse software categories. The key attributes of these applications were:

- Small integer data types (for example: 8-bit graphics pixels, 16-bit audio samples)
- Small, highly repetitive loops
- Frequent multiplies and accumulates
- Compute-intensive algorithms
- Highly parallel operations

MMX technology is designed as a set of basic, general purpose integer instructions that can be easily applied to the needs of the wide diversity of multimedia and communications applications. The highlights of the technology are:

- Single Instruction, Multiple Data (SIMD) technique
- 57 new instructions
- Eight 64-bit wide MMX registers
- Four new data types

The basis for MMX technology is a technique called Single Instruction, Multiple Data (SIMD). This allows many pieces of information to be processed with a single instruction, providing parallelism that greatly increases performance. This technology combined with the IA superscalar architecture will provide substantial performance enhancement to the PC platform. MMX technology is integrated into Intel Architecture processors in a way that maintains full compatibility with existing operating systems, including MS DOS*, Windows* 3.1, Windows 95, OS/2* and Unix*. In addition, the full base of Intel architecture software will run on MMX technology-enabled systems.

MMX technology was defined to be simple. MMX technology is general enough to address the needs of a large domain of PC applications built from current and future algorithms. MMX instructions are not privileged; they can be used in applications, codecs, algorithms, and drivers.

DATA TYPES

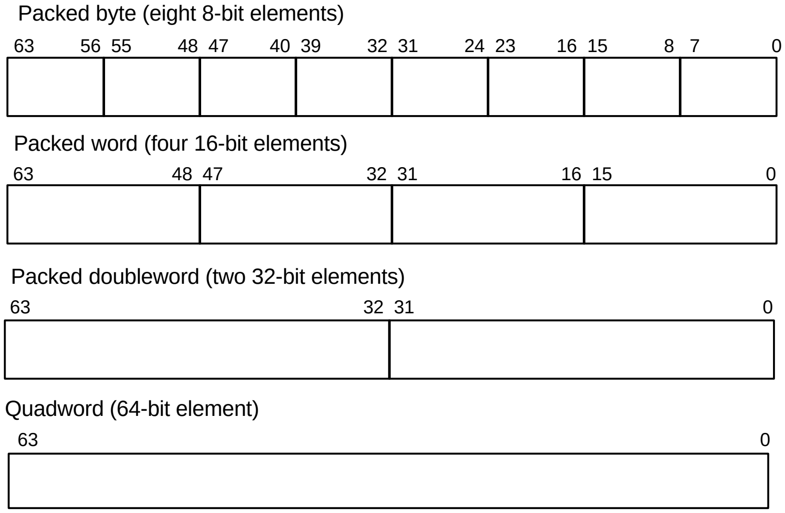
The principal data type of the IA MMX instruction set is the packed, fixed-point integer, where multiple integer words are grouped into a single 64-bit quantity. These 64-bit quantities are moved into the 64-bit MMX registers. The decimal point of the fixed-point values is implicit and is left for the programmer to control for maximum flexibility. The supported data types are signed and unsigned fixed-point integers, bytes, words, doublewords and quadwords.

The four MMX technology data types are:

• Packed byte	Eight bytes packed into one 64-bit quantity
• Packed word	Four 16-bit words packed into one 64-bit quantity
• Packed doubleword	Two 32-bit double words packed into one 64-bit quantity
• Quadword	One 64-bit quantity

As an example, graphics pixel data are generally represented in 8-bit integers, or bytes. With MMX technology, eight of these pixels are packed together in a 64-bit quantity and moved into an MMX register. When an MMX instruction executes, it takes all eight of the pixel values at once from the MMX register, performs the arithmetic or logical operation on all eight elements in parallel, and writes the result into an MMX register.

Data Types in 64-bit Registers


Packed byte (eight 8-bit elements)
63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
Packed word (four 16-bit elements)
63 48 47 32 31 16 15 0
Packed doubleword (two 32-bit elements)
63 32 31 0
Quadword (64-bit element)
63 0

COMPATIBILITY

MMX technology retains its full compatibility with existing operating systems and applications by aliasing its registers and state upon the IA floating-point registers and state. Therefore, no new registers or states are added to support MMX technology. This means that the operating system uses the standard mechanisms for interacting with the floating point state to save and restore MMX code. For example, during a task switch, the operating system would use an FSAV and FRSTR to preserve either floating point or MMX code. Aliasing the MMX state upon the floating-point state does not preclude applications from executing both MMX technology routines and floating point routines.

Floating-point instructions that save/restore the floating-point state also handle the MMX state (for example, during context switching). The same techniques used by the floating-point architecture to interface with the operating system are used by MMX technology. MMX technology does not introduce any new exception or state information, so today’s operating systems can enable applications using MMX instructions.

DETECTING THE PRESENCE OF MMX™ TECHNOLOGY

Detecting the existence of MMX technology on an Intel microprocessor is done by executing the CPUID instruction and checking a set bit. This gives software developers the flexibility to determine the specific code in their software to execute. During install or run time the software can query the microprocessor to determine if MMX technology is supported and install or execute the code that includes, or does not include, MMX instructions based on the result.

INSTRUCTIONS

The MMX instructions cover several functional areas including:

- Basic arithmetic operations such as add, subtract, multiply, arithmetic shift and multiply-add
- Comparison operations
- Conversion instructions to convert between the new data types - pack data together, and unpack from small to larger data types
- Logical operations such as AND, AND NOT,OR, and XOR
- Shift operations
- Data Transfer (MOV) instructions for MMX register-to-register transfers, or 64-bit and 32-bit load/store to memory

Arithmetic and logical instructions are designed to support the different packed integer data types. These instructions have a different op code for each data type supported. As a result, the new MMX technology instructions are implemented with 57 op codes.

MMX technology uses general-purpose, basic instructions that are fast and are easily assigned to the parallel pipelines in Intel processors. By using this general-purpose approach, MMX technology provides performance that will scale well across current and future generations of Intel processors.

MMX™ Instruction Set Summary

The instructions and corresponding mnemonics in the table below are grouped by function categories.

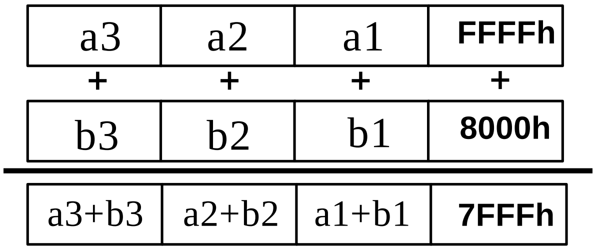
If an instruction supports multiple data types—byte (B), word (W), doubleword (DW), or quadword (QW), the datatypes are listed in brackets. Only one data type may be chosen for a given instruction. For example, the base mnemonic PADD (packed add) has the following variations: PADDB, PADDW, and PADDD. The number of opcodes associated with each base mnemonic is listed.

Category	Mnemonic	Number of Different Opcodes	Description
Arithmetic	PADD[B,W,D]	3	Add with wrap-around on [byte, word, doubleword]
	PADDS[B,W]	2	Add signed with saturation on [byte, word]
	PADDUS[B,W]	2	Add unsigned with saturation on [byte, word]
	PSUB[B,W,D]	3	Subtraction with wrap-around on [byte, word, doubleword]
	PSUBS[B,W]	2	Subtract signed with saturation on [byte, word]
	PSUBUS[B,W]	2	Subtract unsigned with saturation on [byte, word]
	PMULHW	1	Packed multiply high on words
	PMULLW	1	Packed multiply low on words
	PMADDWD	1	Packed multiply on words and add resulting pairs
Comparison	PCMPEQ[B,W,D]	3	Packed compare for equality [byte, word, doubleword]
	PCMPGT[B,W,D]	3	Packed compare greater than [byte, word, doubleword]
Conversion	PACKUSWB	1	Pack words into bytes (unsigned with saturation)
	PACKSS[WB,DW]	2	Pack [words into bytes, doublewords into words] (signed with saturation)
	PUNPCKH [BW,WD,DQ]	3	Unpack (interleave) high-order [bytes, words, doublewords] from MMX™ register
	PUNPCKL [BW,WD,DQ]	3	Unpack (interleave) low-order [bytes, words, doublewords] from MMX register
Logical	PAND	1	Bitwise AND
	PANDN	1	Bitwise AND NOT
	POR	1	Bitwise OR
	PXOR	1	Bitwise XOR
Shift	PSLL[W,D,Q]	6	Packed shift left logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRL[W,D,Q]	6	Packed shift right logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRA[W,D]	4	Packed shift right arithmetic [word, doubleword] by amount specified in MMX register or by immediate value
Data Transfer	MOV[D,Q]	4	Move [doubleword, quadword] to MMX register or from MMX register
FP & MMX State Mgmt	EMMS	1	Empty MMX state

Instruction Examples

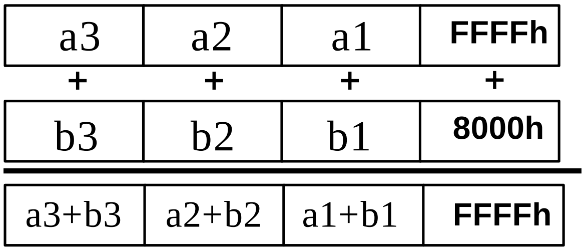
The following section will describe briefly five examples of MMX instructions. For illustration, the data type shown in this section will be the 16-bit word data type; most of these operations also exist for 8-bit or 32-bit packed data types.

The following example shows a packed add word with wrap around. It performs four additions of the eight, 16-bit elements, with each addition independent of the others and in parallel. In this case, the right-most result exceeds the maximum value representable in 16-bits—thus it wraps-around. This is the way regular IA arithmetic behaves. FFFFh + 8000h would be a 17 bit result. The 17th bit is lost because of wrap around, so the result is 7FFFh.



PADD[W]: Wrap-around Add

The following example is for a packed add word with unsigned saturation. This example uses the same data values from before. The right-most add generates a result that does not fit into 16 bits; consequently, in this case saturation occurs. Saturation means that if addition results in overflow or subtraction results in underflow, the result is clamped to the largest or the smallest value representable. For an unsigned, 16-bit word, the largest and the smallest representable values are FFFFh and 0x0000; for a signed word the largest and the smallest representable values are 7FFFh and 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.

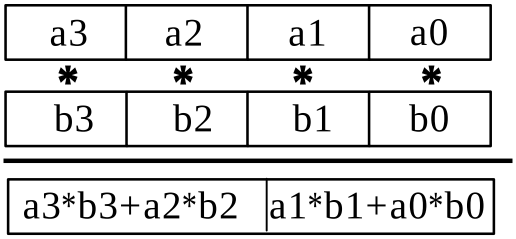


PADDUS[W]: Saturating Arithmetic

The specific instruction here is Packed Add Unsigned Saturation Word (PADDUSW). A complete set of ADD operations exists for signed and unsigned cases. The number FFFFh, treated as unsigned (65,535 decimal), is added to 0x8000 unsigned (32,768), and the result saturates to FFFFh - the largest representable unsigned 16-bit value.

There is no “saturation mode bit” as a new mode bit would require a change to the operating system. Separate instructions are used to generate wrap-around and saturating results.

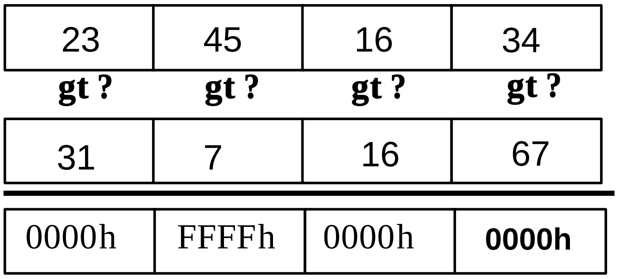
The next example shows the key instruction used for multiply-accumulate operations, which are fundamental to many signal processing algorithms like vector-dot-products, matrix multiplies, FIR and IIR Filters, FFTs, DCTs etc. This instruction is the packed multiply add (PMADD).



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

The PMADD instruction starts from a 16-bit, packed data type and generates a 32-bit packed, data type result. It multiplies all the corresponding elements generating four 32-bit results, and adds the two products on the left together for one result and the two products on the right together for the other result. To complete a multiply-accumulate operation, the results would then be added to another register which is used as the accumulator.

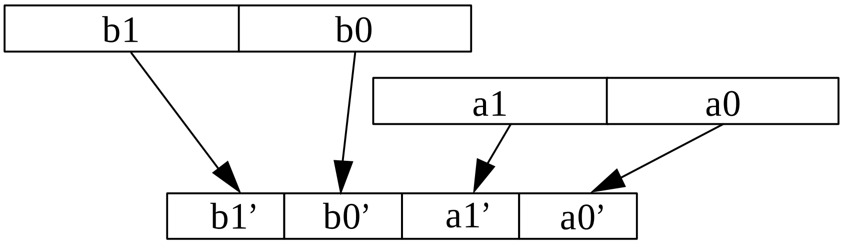
The following example is a packed parallel compare. This example compares four pairs of 16-bit words. It creates a result of true (FFFFh), or false (0000h). This result is a packed mask of ones for each true condition, or zeros for each false condition. The following example shows an example of a compare "greater than" on packed word data. There are no new condition code flags, nor are any existing IA condition code flags affected by this instruction.



PCMPGT[W]: Parallel Compares

The packed compare result can be used as a mask to select elements from different inputs using a logical operation, eliminating the need for a branch or a set of branch instructions. The ability to do a conditional move instead of using branch instructions is an important performance enhancement in advanced processors that have deep pipelines and employ branch prediction. A branch based on the result of a compare operation on the incoming data is usually difficult to predict, as incoming data in many cases can change randomly. Eliminating branches that are used to perform data selection by using the conditional select capability, together with the parallelism of the MMX instruction set, is an important performance enhancement feature of the MMX technology.

The following is an example of a pack instruction. It takes four 32-bit values and packs them into four 16-bit values, performing saturation if one of the 32-bit source values does not fit into a 16-bit result. There are also instructions that perform the opposite - unpack, for example, a packed byte data type into a packed word data type.



PACKSS[DW]: Pack Instruction

The pack and unpack instructions exist to facilitate conversion between the new packed data types. These are especially important when an algorithm needs higher precision in its intermediate calculations, as in image filtering. A filter on an image usually involves a set of multiply operations between filter coefficients and a set of adjacent image pixels, accumulating all the values together. These multiplies and accumulations need more precision than 8-bits, the original data type of the pixels. The solution is to unpack the image's 8-bit pixels into 16-bit words, perform the calculations in 16-bit words without concern for overflow, then pack back to 8-bit pixels before storing the filtered pixels to memory.

APPLICATION EXAMPLES

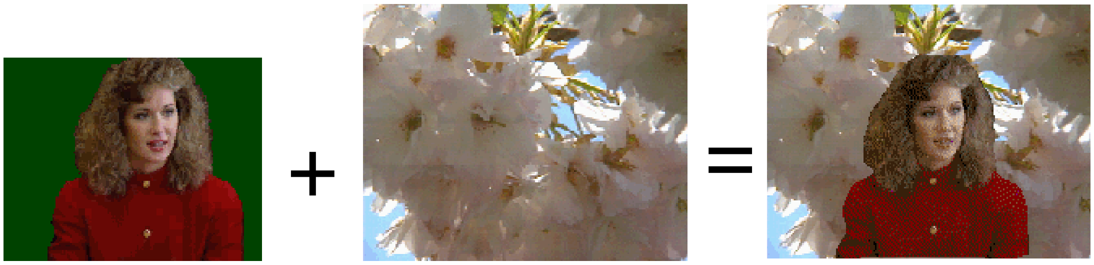
The following section describes example uses of the MMX instruction set to implement basic coding structures:

Conditional Select

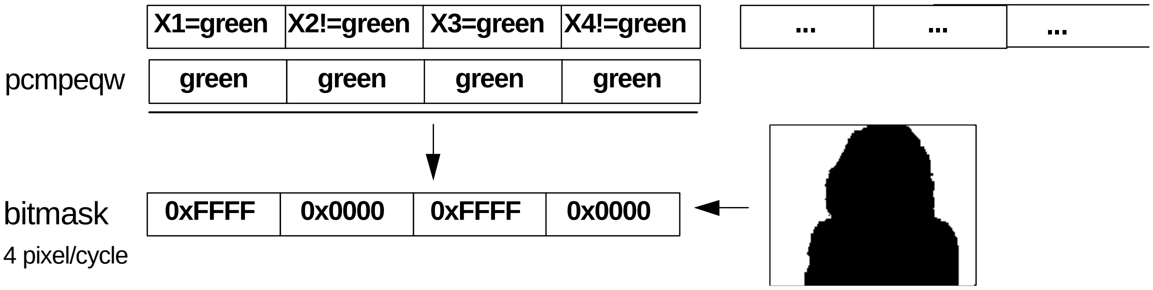
Multimedia applications must process large sets of data. In some cases there is a need to select the data based on a condition query performed on the incoming data. Intel has been able to improve performance in its family of processors by implementing micro-architectural features for increased performance and deeper pipelines. Branch prediction is an important part of making the pipelines run efficiently, as a misprediction can cause the pipelines to flush and degrade performance. The following example shows an efficient way to reduce the need to use branch instructions, especially those that are data dependent, and thus very difficult to predict. The Chroma Keying example demonstrates how conditional selection using the MMX instruction set removes branch mis-predictions, in addition to performing multiple selection operations in parallel. Text overlay on a graphics/video background, and sprite overlays in games are some of the other operations that would benefit from this technique.

Chroma Keying

Most have seen the television weather man overlaid on the image of a weather map. In this example we use a green screen to overlay an image of a woman on a picture of spring blossom. We'll illustrate this example by processing four 16-bit pixels in parallel. The instructions also allow the processing of eight 8-bit pixels in parallel for a substantial performance speed-up potential.

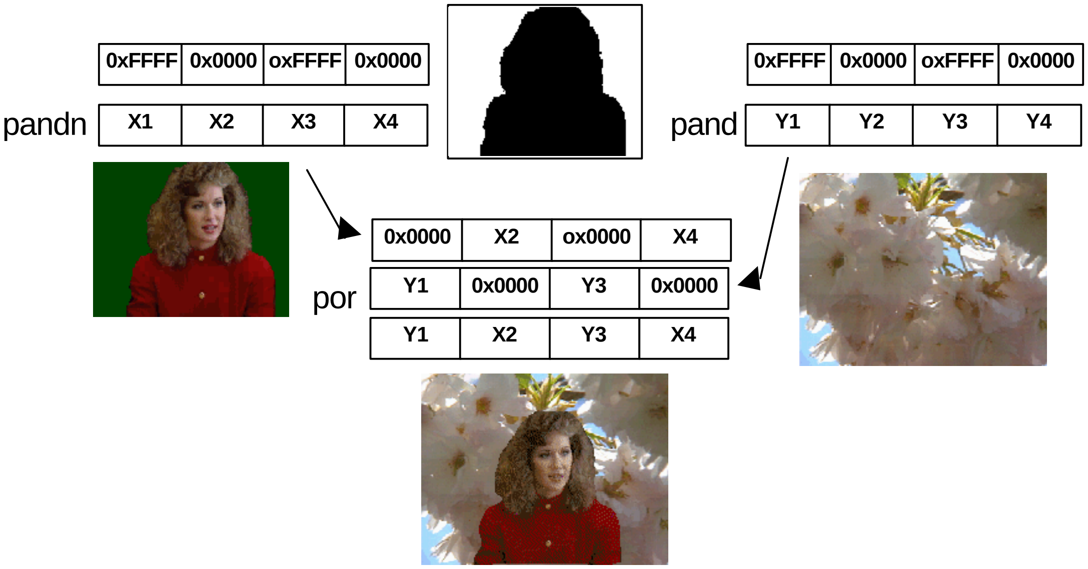


First we'll take four pixels from the picture with the woman on a green background. The top row of the data below represents pixels that alternate between green, not green, green, and not green. The compare instruction builds a mask for that data. That mask is a sequence of words that are all ones or all zeros representing the Boolean values of true and false. We now know what is the unwanted background and what we want to keep. This is shown below using a shadow picture.



This mask is now used on the same four pixels from the picture with the woman and the equivalent four pixels from the Spring blossom. The "AND NOT" and "AND" instructions use the mask to identify which pixels to keep from the Spring blossom and the woman. They also turn the unwanted pixels to zeros. The "OR" instruction builds the final picture. Four pixels were mapped using only four MMX instructions without any branches.

In working through this example, the PANDN instruction inverts all the bits in mask before applying the AND operation.



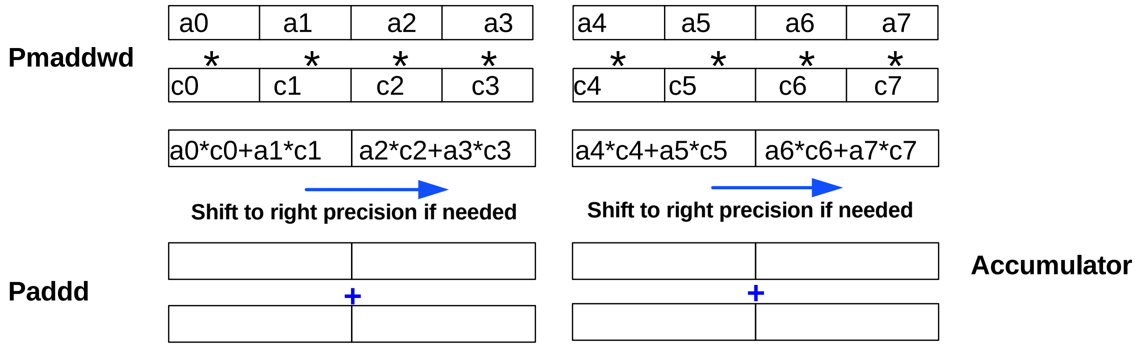
Without MMX technology, each pixel is processed separately and requires a conditional branch. Using MMX instructions, eight 8-bit pixels can be processed in parallel and no conditional branches are involved.

Vector Dot Product

The vector dot product is one of the most basic algorithms used in signal-processing of natural data such as images, audio, video and sound. The following example shows how the PMADD instruction helps speed up algorithms using vector dot products. The PMADD instruction will handle four multiplies and two additions at a time. Coupled with a PADD instruction, as described before, eight multiply-accumulate operations are performed. These eight element vectors fit nicely into two PMADD instructions and two PADD instructions.

Assuming that the precision supported by the PMADD instruction is sufficient, this dot-product example on eight-element vectors can be completed using eight MMX instructions: Two PMADDs, two more PADDs, two shifts (if needed to fix the precision after the multiply operation), and two memory moves to load one of the vectors (the other vector is loaded by the PMADD instruction which can have one of its operands come from memory).

$$x = \sum a(i) * c(i)$$



Note: Input data and coefficients are 16-bit precision.
If not, first unpack to 16 bit.

Comparing instruction counts with and without MMX technology for this operation yields the following:

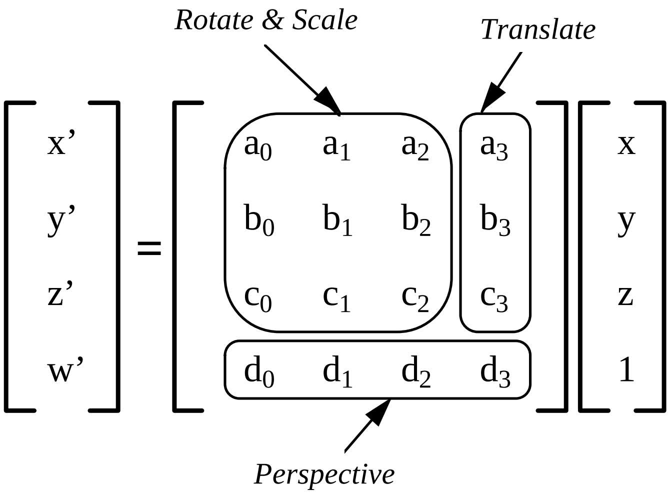
	Number of Instructions without MMX™ Technology	Number of MMX Instructions
Load	16	4
Multiply	8	2
Shift	8	2
Add	7	1
Miscellaneous	-	3
Store	1	1
Total	40	13

With MMX technology, one third of the number of instructions is needed.

Most MMX instructions can be executed in one clock cycle, so the performance improvement will be more dramatic than the simple ratio of instruction counts.

Matrix Multiply

Exciting new 3D games are coming to market every day. Typically, computations that manipulate 3D objects are based on 4-by-4 matrices that are multiplied with four element vectors many times. The vector has the X,Y, Z and perspective corrective information for each pixel. The 4-by-4 matrix is used to rotate, scale, translate and update the perspective corrective information for each pixel. This 4-by-4 matrix is applied to many vectors.



$$x' = a_0x + a_1y + a_2z + a_3$$

Applications which already use 16-bit integer or fixed-point data are able to make extensive use of the PMADD instruction. There would be one PMADD instruction per row in the matrix, for a total of four. Comparing instruction counts with and without MMX technology for this operation yields the following:

	Number of Instructions without MMX™ Technology	Number of MMX instructions
Load	32	6
Multiply	16	4
Add	12	2
Miscellaneous	8	12
Store	4	4
Total	72	28

With MMX technology, less than one half of the number of instructions without MMX technology is needed.

24-Bit Color

The MMX instruction set offers graphical applications the opportunity to move from 8-bit or 16-bit color lookup table to 24-bit, or “true” color, a feature which will greatly enhance the realism of a game’s graphics. In many cases, this can be done in the same amount of time that is currently required for 8-bit graphics. For 24-bit and 32-bit colors, red, green and blue are each represented by 8-bit values. There are eight additional bits in 32-bit color for alpha value.

Image compositing and alpha blending are operations that can be performed on 24-bit color images.

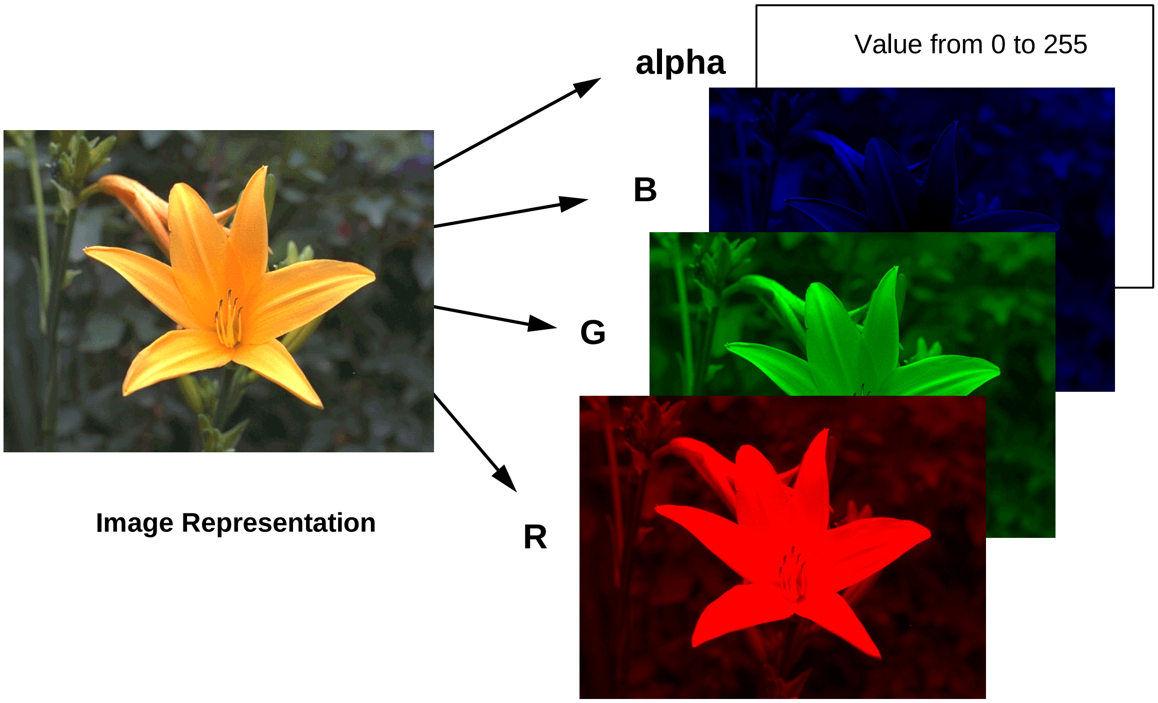


Image Representation

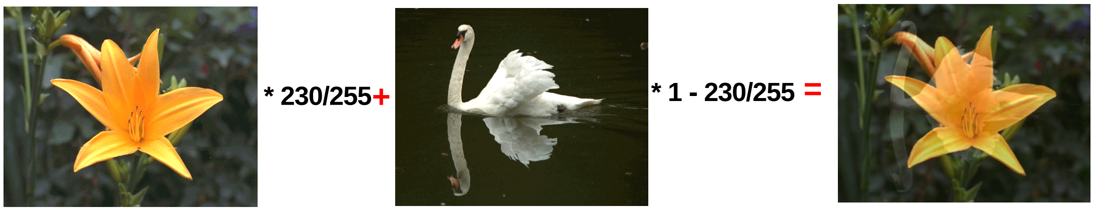
Image Dissolve Using Alpha Blending

This example shows how the MMX instruction set will speed up image compositing. In this example, a flower will dissolve into a swan. The screen starts with a picture of the flower. As the flower gradually fades away, the swan gradually appears.

The math for the dissolve is a straight-forward function. Alpha determines the intensity of the flower. At full intensity, the flower's 8-bit alpha value is FFH, or 255. By plugging 255 in the dissolve equation, each flower pixel is 100 percent and each swan pixel is 0 percent. The equation below calculates each pixel:

$$\text{Result\_pixel} = \text{Flower\_pixel} * (\text{alpha}/255) + \text{Swan\_pixel} * [1 - (\text{alpha}/255)]$$

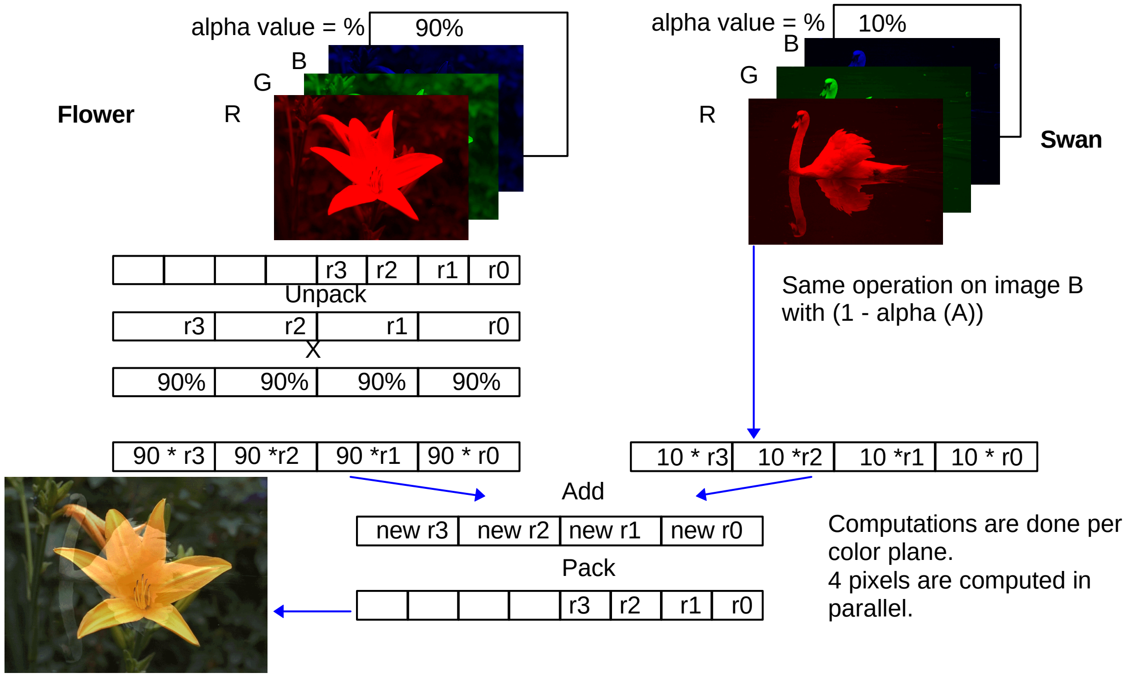
Illustrated below are the flower and swan when alpha = 230:



When the alpha value is 230, the resulting picture is 90 percent flower and 10 percent swan. On close examination, some of the swan image appears in the picture to the right of the equal sign.

This example assumes that the 24-bit color data is organized so that four pixels at a time are processed from one color plane, that is, the image is separated into individual color planes: one for red, one for green, one for blue. The first four red values from the flower and the swan will be processed first. After finishing the red plane, the processing moves to the green and blue planes.

The unpack instruction takes the first four bytes of the red data that are represented in 8-bit values and unpacks each pixel into 16-bit elements, putting them into a 64-bit MMX register. The alpha value, which is computed once per frame, is the other operand. The PMUL multiplies the two vectors in parallel. Similarly, an unpack and PMUL create the intermediate result for the swan. Now the two intermediate results are added together using a PADD and the final result is sent to memory using a PACK that converts the intermediate 16-bit values back to 8-bit pixel values that can be stored.



Same operation on image B with (1 - alpha (A))

Computations are done per color plane. 4 pixels are computed in parallel.

If these images use 640X480 resolution, and the dissolve technique uses all 255 steps of the alpha value, then 117 million PUNPCKs and PMULs, and 58 million PADDs and PACKs are used. Comparing instruction counts with and without MMX technology for this operation yields the following:

Operation	Calculation without MMX™ Technology	Number of Instructions without MMX Technology	Number of MMX Instructions
Load	(640*480)*255*3*2	470 million	117 million
Unpack	-	-	117 million
Multiply	(640*480)*255*3*2	470 million	117 million
Add	(640*480)*255*3	235 million	58 million
Pack	-	-	58 million
Store	(640*480)*255*3	235 million	58 million
Total		1.4 billion	525 million

Almost 1 billion fewer instructions are used in this example.

The dissolve technique, sometimes called combine, is one of several commonly used image compositing techniques used in multimedia applications and can be sped up substantially with MMX technology.

Combine	Dissolve: Fade in, fade out effect A * alpha(A) + B * (1 - alpha(A))
A over B	Transparent Image placed on background A + (B * (a - alpha(A)))
A in B	Image A only where B has Opacity A * alpha (B)
A out B	Image A only where B has transparency A * (1 - alpha(B))
A top B	(A in B) over B (A * alpha(B)) + (B * (1 - alpha(A)))
A XOR B	(B * (1 - alpha(A))) + (A * (1 - alpha(B)))

Alpha blending is a technique used by game developers that is similar to image compositing. Alpha blending allows race cars to drive realistically through fog or smoke, allows a more realistic view of fish in water, or a rabbit in a translucent tube. In these examples, the alpha values wouldn't necessarily be the same for the whole frame, but the basic concept remains the same.

SUMMARY

MMX technology brings more power to multimedia and communication applications. MMX technology adds new data types and instructions that can process data in parallel. MMX technology is fully compatible with existing operating systems and application software.

MMX technology brings a step improvement to the PC platform and enables new applications and usage of PCs. It helps establish a new paradigm in the industry with the PC as an improved communications and multimedia device. Systems enabled with MMX technology will ramp in high volume in 1997 as Intel incorporates the technology in multiple processor generations.

RELATED DOCUMENTATION

Refer to the following documentation for more information on MMX technology.

- *Intel Architecture MMX™ Technology Developers' Manual* (Order Number 243013)
- *Intel Architecture MMX™ Technology Programmer's Reference Manual* (Order Number 243007)

Refer to Intel's corporate website for the latest information on related documentation:

http://www.intel.com/

[1]