VOLUME 10, NUMBER 3
MARCH 5, 1996

MICROPROCESSOR REPORT
THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

Intel's MMX Speeds Multimedia

Instruction-Set Extensions to Aid Audio, Video, and Speech

By Linley Gwennap

The first major extension to the x86 instruction set since 1985 will greatly improve the venerable architecture's handling of emerging multimedia applications. Collectively known as MMX, these 57 new instructions accelerate calculations common in audio, 2D and 3D graphics, video, speech synthesis and recognition, and data communications algorithms by as much as 8×. Overall, users will see a 50–100% performance improvement or more on these types of programs when using MMX instructions, as Figure 1 shows.

Intel plans to implement MMX throughout its product line in 1997. The first instantiation of MMX will be the P55C, a Pentium derivative due in 4Q96. MMX will also be included in Klamath *(see* **1003ED.PDF***)*, a cost-reduced Pentium Pro that we expect to debut in 1H97. By the end of 1997, these two devices (and their successors) will displace most or all of Intel's non-MMX processors. AMD plans to incorporate MMX in its future processors *(see* **1001MSB.PDF***)*, and we expect Cyrix will follow suit.

MMX is designed to have no impact on the operating system, making it compatible with existing x86-based OSs. Applications can take advantage of MMX in two ways: either by calling MMX-enabled drivers, such as a graphics driver, or by adding MMX instructions to critical routines. Most applications will take the driver route.

Intel has been working with dozens of key software and hardware vendors for months to help them add MMX to their applications and drivers. Today's public disclosure of the instruction set will enable any programmer to begin recoding their software for the new instructions. The number of MMX-enabled drivers and applications will build quickly once P55C systems are released.

These new instructions will provide PC users with a highly visible performance boost on many of today's most performance-critical applications. This boost should foster increased growth in the PC market and give Intel a leg up on competitors, such as PowerPC, that are lagging in adopting similar technology.

Simple Software Model

The mark of a good organization is learning from past mistakes. In designing the MMX software model, Intel took pains to avoid creating any new modes or new user state that would complicate an already complex architecture. MMX instructions can be used in any processor mode and at any privilege level. They generate no new interrupts or exceptions. These features eliminate the need for changes in the operating system to allow use of the new instructions.

From the programmer's view, there are eight new MMX registers (MM0–MM7) along with new instructions that operate on these registers. But to avoid adding new state, these registers are mapped onto the existing floating-point registers (FP0–FP7). When a multitasking operating system (or application) executes an FSAVE instruction, as it does today to save state, the contents of MM0–MM7 are saved in place of FP0–FP7 if MMX instructions are in use.

The obvious drawback is that programs cannot use both FP and MMX instructions within the same routines, as both share the same register set. This is rarely an issue, since most programs don't use FP at all, and those that do typically use these calculations to generate data, while MMX is typi-

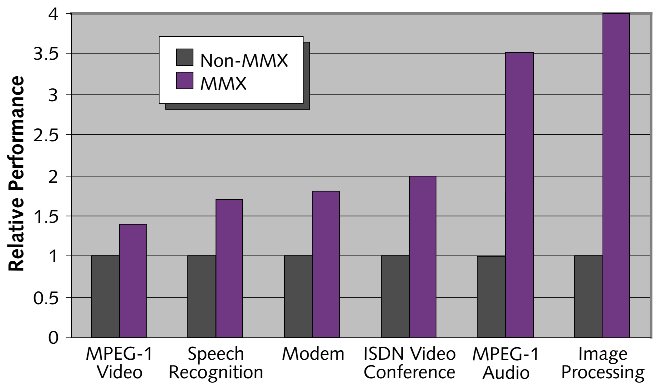

Figure 1. MMX improves performance on most multimedia applications by 50–100%, according to simulations of the forthcoming P55C processor. MPEG-1 performance refers to decoding; image processing is pixel manipulation, as in Photoshop. (Source: Intel)

cally used in separate routines that display data. For 3D graphics, Intel recommends that geometry calculations remain in floating point while MMX is used to accelerate 3D rendering routines.

Without a mode bit, there is no foolproof way to prevent FP instructions from corrupting MMX data, and vice versa. Intel has taken some precautions, however, to trap the most common foolish situations. When data is loaded from memory into any MM register, it marks all the FP registers as busy, causing any subsequent FP instruction to trap. At the end of an MMX routine, the programmer must insert an EMMS instruction to restore the registers for FP use.

The converse situation is mostly covered. Taking advantage of the fact that the MMX registers are 64 bits wide while the FP registers are 80, MMX instructions always set the 16-bit exponent to NaN (not a number) while storing the result in the 64 fraction bits. Thus, although there is no way to trap an MMX instruction that is executed during a sequence of FP instructions, any subsequent FP calculation on the modified data will produce a floating-point exception. Alas, an FST instruction could store the corrupted data in memory; Intel could not find an easy way to plug this hole.

Single Instruction, Multiple Data

The new instructions, listed in Table 1, use a SIMD (single instruction, multiple data) model, operating on several values at a time. In this respect, they are similar to multimedia instructions in the Motorola 88110, HP's PA-7100LC, and Sun's UltraSparc. Using the 64-bit MMX registers, these instructions can operate on eight bytes, four words, or two double words at once, greatly increasing throughput.

Figure 2 shows the three new data types: packed byte, packed word, and packed double word. (RISC devotees should note that Intel words are 16 bits, and double words are 32 bits.) These data types are particularly suited to multimedia because many algorithms work on small data sizes.

For example, audio data is usually stored in 8-, 12-, or 16-bit samples; the average person cannot appreciate further precision. Video is represented in pixels, commonly encoded as RGB (red, green, blue) triplets. Each of the three color values can be stored in 4, 6, or 8 bits; the last provides 16 million possible colors, more than most people can discern.

Most of the new mnemonics begin with "P" for packed; for example, PADD means packed add. The opcodes all begin with the byte 0F, as do existing long jump, set byte, and Pentium-specific instructions. MMX uses previously reserved values for the second byte (none of which is used by other x86 vendors). The next two (or more) bytes provide the two operands, using the same encodings as other x86 instructions, except the target registers are the MMX registers, not the integer registers (EAX, etc.).

For example, the MOVD and MOVQ instructions can move data to and from memory using the same multitude of addressing modes as the standard MOV instruction; they also move data from one MM register to another. The MOVD instruction can even exchange data with the integer registers. Likewise, PADD performs register-to-register or memory-to-register operations, just like the integer ADD instruction. One exception is that register-to-memory mode is not supported in MMX.

Group	Mnemonic	Opcode*	Description
Data Transfer, Pack, Unpack	MOV[D,Q]	6E/7E,6F/7F	Move [double,quad] to/from MM register
	PACKUSWB	67	Pack words into bytes with unsigned saturation
	PACKSS[WB,DW]	63,6B	Pack [words into bytes, doubles into words] with signed saturation
	PUNPCKH [BW,WD,DQ]	68,69,6A	Unpack (interleave) high-order [bytes, words, doubles] from MM register
	PUNPCKL [BW,WD,DQ]	60,61,62	Unpack (interleave) low-order [bytes, words, doubles] from MM register
Arithmetic	PADD[B,W,D]	FC,FD,FE	Packed add on [byte, word, double]
	PADDS[B,W]	EC,ED	Saturating add on [byte, word]
	PADDUS[B,W]	DC,DD	Unsigned saturating add on [byte, word]
	PSUB[B,W,D]	F8,F9,FA	Packed subtraction on [byte, word, double]
	PSUBS[B,W]	E8,E9	Saturating subtraction on [byte, word]
	PSUBUS[B,W]	D8,D9	Unsigned saturating subtraction on [byte, word]
	PMULHW	E5	Multiply packed words to get high bits of product
	PMULLW	D5	Multiply packed words to get low bits of product
	PMADDWD	F5	Multply packed words, add pairs of products
Shift	PSLL[W,D,Q]	F1/71,F2/72, F3/73†	Packed shift left logical [word, double, quad]
	PSRL[W,D,Q]	D1/71,D2/72, D3/73†	Packed shift right logical [word, double, quad]
	PSRA[W,D]	E1/71,E2/72†	Packed shift right arithmetic [word, double]
Logical	PAND	DB	Bitwise logical AND
	PANDN	DF	Bitwise logical AND NOT
	POR	EB	Bitwise logical OR
	PXOR	EF	Bitwise logical XOR
Compare	PCMPEQ[B,W,D]	74,75,76	Packed compare if equal [byte, word, double]
	PCMPGT[B,W,D]	64,65,66	Packed compare if greater than [byte, word, dbl]
Misc	EMMS	77	Empty MMX state

Table 1. Intel's MMX multimedia extensions include 57 new opcodes. Brackets indicate a set of options where only one may be chosen for a given instruction. B=byte, W=word, D=double word, Q=quad word. *All opcodes start with 0F followed by the extension byte shown here. †Opcodes with 71, 72, and 73 as the second byte end with a third byte: Dr (PSRL), Er (PSRA), or Fr (PSLL), where "r" is the first operand.

Pack and Unpack Instructions Shuffle Bytes

In many cases, byte or word data is already stored in consecutive locations in memory and thus can be operated on by the new instructions. If the data is stored as aligned 32-bit values, however, it may be necessary to rearrange it to the packed format. The PACKxxDW instruction reads two double words from memory and combines them with two double words in a register, resulting in four packed 16-bit words.

If the original item exceeds the maximum value expressible in 16 bits, it is saturated: items that are too small are set to the smallest possible value, and items that are too large are set to the largest possible value. There are two options for calculating the saturation values, signed and unsigned, depending on how the source data is expressed. Similarly, the PACKxxWB instruction converts packed word data to packed byte data.

These operations can be reversed to unpack data. The mellifluous mnemonic PUNPCKxBW converts packed bytes into packed words, zero-extending the bytes, as Figure 3(a) shows. It can also interleave two sets of byte data into word data, as Figure 3(b) demonstrates. Similarly, PUNPCKxWD and PUNPCKxDQ convert packed words to double words and packed double words to quad words, respectively.

New Instructions Calculate in Parallel

Once the data is packed, calculations proceed in parallel. Each MMX calculation combines two 64-bit operands and produces a 64-bit result, so packed-byte instructions calculate eight results in parallel. Similarly, packed-word instructions generate four results, and instructions that operate on packed double words produce two results. Because most x86 instructions generate only one result at a time, this parallel calculation ability is the key to MMX's performance gains.

The performance boost is even greater on the P55C, due to the way MMX instructions are issued. The current Pentium design, while nominally two-way superscalar, executes only one FP calculation at a time and cannot pair these instructions with integer operations. The P55C allows pairing of MMX and integer instructions and can even pair MMX instructions with each other as long as they use different function units. Thus, the P55C can calculate up to 16 results (of one byte each) per cycle, helping to generate the performance gains seen previously in Figure 1.

The calculation instructions are all similar and, for the most part, straightforward. As Figure 3(c) shows, the PADDW (packed add word) instruction performs a parallel add of each of the four words in the source operand with the corresponding word in the destination operand, storing the result in the specified destination. Any carry out of a 16-bit addition is ignored. The results of the integer flags (carry, overflow, zero, etc.) are unchanged by any MMX calculation.

Both the add and subtract instructions can operate on packed bytes, words, or (in some cases) double words. Mnemonically, the suffixes B, W, or D are appended to indicate the data type. A set of logical operations (AND, AND

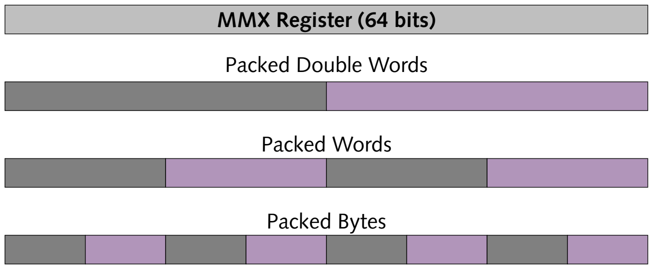

Figure 2. MMX adds three new data types: packed byte, packed word, and packed double word.

NOT, and OR) operate on a bit-by-bit basis and thus do not require a data-type suffix. They are identical to the existing logical instructions, except that they operate on MMX registers instead of integer registers.

New shift instructions differ from integer shift instructions in that each packed data element is treated individually. For example, PSLL (packed shift logical left) shifts items left while filling the lower bits of each with zeroes. The logical right shift fills the upper bits with zeroes, while the arithmetic right shift inserts sign bits. The shift count can be specified by an immediate value or an MMX register. These instructions can be used to quickly multiply or divide signed and unsigned data by powers of two.

Saturating and Unsaturating Arithmetic

The add and subtract instructions have three variations. The default (no suffix) option is simple, nonsaturating arithmetic. The other two options apply saturating arithmetic; as with the saturating PACK instructions, any overflow causes the result to be "clamped" to its maximum value, and underflows set the result to the minimum value. The suffix S indi-

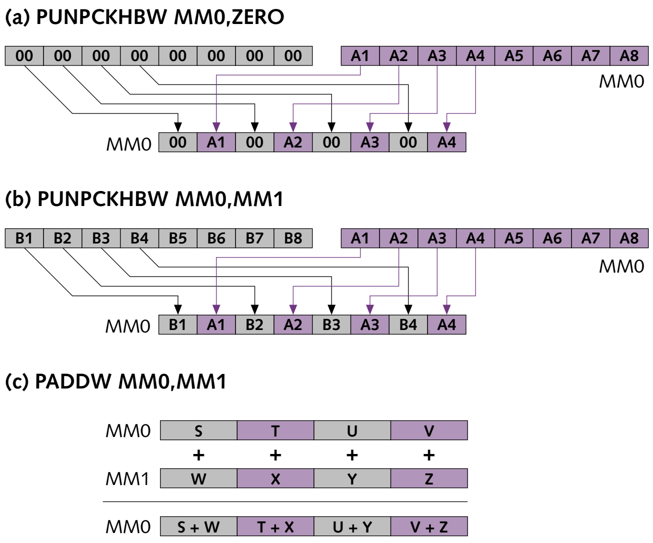

Figure 3. (a) Unpack operation converts packed bytes to words with zero extension. (b) PUNPCK can also interleave bytes from two MM registers. (c) Parallel add instruction calculates four 16-bit sums simultaneously.

cates signed saturating arithmetic; the most significant bit in each field is treated as a sign bit.

The third case is unsigned saturating (US) arithmetic, typically used for pixel operations. When two intensities, for example, are added, the result can never be whiter than white or blacker than black; saturating arithmetic handles this automatically, avoiding the long series of overflow and underflow checks needed with traditional instruction sets. In fact, a single PADDUSB instruction could replace 40 non-MMX x86 instructions.

These options are not completely orthogonal, a fact that should not surprise any x86 programmer. Although the nonsaturating form supports bytes, words, and doubles, the saturating forms cannot handle 32-bit data. The combination of the extra logic required to perform saturation with the longer carry chain of the 32-bit adder failed to meet the cycle-time requirement of the P55C.

For most situations, the saturating and nonsaturating forms are equivalent. In fact, it is dangerous to use the non-saturating form for values that might cause an overflow, as it has no overflow trap. Of the nonsaturating adds and subtracts, only the 32-bit versions are typically used, to compensate for the lack of 32-bit saturating instructions.

Two Instructions Perform 16-bit Multiplication

Simple multiplication is handled by PMULHW and PMULLW. These instruction operate only on 16-bit values. Because the result of a multiplication can be twice the width of its operands, PMULHW stores only the high-order word of the result in the destination register. (Of course, it generates and stores four results in parallel.) In some situations, this 16-bit result will provide adequate precision.

For full 32-bit precision, the second half of the result is generated by PMULLW. The results of the two instructions must then be combined using PUNPCKWD, which interleaves the two destination registers. The following code multiplies the four words in MM1 by the four words in MM2, storing the four products as two double words in MM1 and two double words in MM2:

```
MOVQ  MM0, MM1             ;Make copy of MM1
PMULHW  MM0, MM2           ;Calculate high bits in MM0
PMULLW  MM1, MM2           ;Calculate low bits in MM1
MOVQ  MM2, MM1             ;Make copy of low bits
PUNPCKHWD  MM1, MM0        ;Merge first two dwords
PUNPCKLWD  MM2, MM0        ;Merge second dwords
```

This code calculates four products in 6 cycles on a P55C, whereas a non-MMX Pentium requires 10 cycles to complete a single $16 \times 16 \rightarrow 32$-bit integer multiplication.

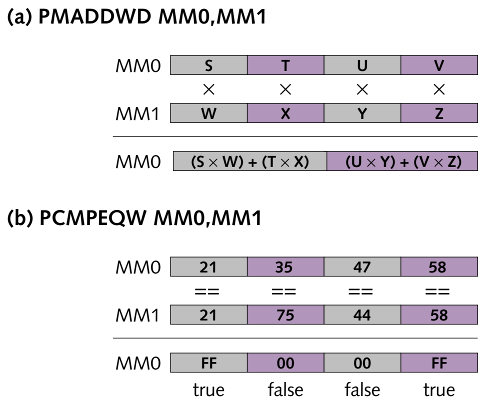

Figure 4. (a) Packed multiply-add sums two pairs of products with a single instruction. (b) Packed compare-if-equal compares packed words and generates a packed Boolean output, where all zeroes indicates false and all ones indicates true.

Multiply-Add Speeds Signal Processing

The packed multiply-add instruction differs from the other calculation instructions in that the data type of the result is different from that of the source. As Figure 4(a) shows, PMADDWD multiplies two pairs of 16-bit words, then sums each pair, producing two 32-bit results. It executes in just three cycles on a P55C and is fully pipelined.

Multiply-add is at the heart of many audio and video algorithms, such as the fast Fouriér transform (FFT). This procedure multiplies two vectors and accumulates the sum of the products. Using PMADDWD, a P55C can multiply and accumulate four vector entries per cycle (assuming the loop has been unrolled three times) while freeing the second pipe to perform loads, stores, index calculations, and branches. This is eight times the peak FP performance of a Pentium (which has no FP multiply-add instruction), not counting the advantage of executing instructions in the second pipe.

One drawback to MMX is the lack of a multiply or multiply-add for 32-bit operands. A fast 32-bit multiplier consumes four times more die area than a 16-bit multiplier, and Intel felt this feature was not worth the extra area. Besides, multiplication of 32-bit data can be performed using the standard integer multiply instruction. Although this instruction takes 10 cycles in the Pentium core and is not pipelined, it requires 4 cycles on Pentium Pro (and presumably Klamath) and, more important, is fully pipelined.

The integer multiplier, however, operates on the integer registers, not the MMX registers, and it cannot perform parallel calculations like the MMX units. Furthermore, there is no integer multiply-add instruction in x86. Because 16-bit precision is inadequate for advanced audio algorithms, such as wavetable sound, and for most 3D geometry calculations, the lack of a 32-bit multiply-add prevents these types of routines from taking advantage of MMX.

Parallel Comparisons Eliminate Branches

The MMX extensions include parallel compare operations that seem awkward at first but will produce big performance savings, particularly for Klamath and its successors. The PCMPEQW instruction, for example, compares two packed words; the fields in the result are set to zero if the comparison is false (not equal, in this case) or all ones if the comparison is true (equal), as Figure 4(b) shows.

This function is useful when combining or overlaying two images. For example, a common video technique known as chroma keying allows an object (such as the weatherman) in front of a blue screen to be superimposed on another image (such as the weather map). In a digital implementation, this technique requires combining two images such that any blue pixels in the first image are replaced by the corresponding pixels in the second image.

Assume that X[i] is the first image, Y[i] is the background image, and the result is put back into X[i]. Using traditional x86 code, a single iteration might look like this:

```
CMP  X[i], BLUE          ;Check if blue
JNE  next_pixel          ;If not, skip ahead
MOV  X[i], Y[i]          ;If blue, use second image
```

In this case, three instructions are needed per pixel.

Using MMX instructions, this sequence can be recoded as follows, assuming 16-bit pixels:

```
MOV  MM1, X[i]           ;Make a copy of X[i]
PCMPEQW  MM1, BLUE       ;Check four pixels in X[i]
PAND  Y[i], MM1          ;Zero out non-blue pixels in Y
PANDN  MM1, X[i]         ;Zero out blue pixels in X
POR  MM1, Y[i]           ;Combine two images
```

Note that this sequence assumes all pixels are in MMX registers. The compare instruction generates four results in register MM1, setting each to zero if the corresponding pixel is not blue. The PAND combines this result with Y[i], zeroing any pixels corresponding to non-blue values in X[i]. Conversely, the PANDN zeroes the blue pixels in X[i].

At first glance, this routine appears to be about 2.5× faster than the non-MMX routine, processing four pixels in five instructions. The actual performance will be even better, however, because the second routine eliminates a branch. Although modern processors predict branches, they mispredict perhaps 10–20% of the time. Pentium's misprediction penalty is 4–5 cycles, while Pentium Pro (and presumably Klamath) takes an average of 15 cycles to recover from a misprediction. Thus, eliminating branches in this way significantly improves performance.

Outshined by VIS But Ahead of Others

In many ways, MMX is quite similar to the VIS instruction set *(see* **081604.PDF***)* developed by Sun for UltraSparc. Both MMX and VIS pack 8-, 16-, and 32-bit data into 64-bit registers for parallel operations, including addition, multiplication, comparisons, and logical operations. Both use the floating-point registers to store these values. Both perform saturating and unsaturating arithmetic.

To this baseline feature set, VIS adds some highly specialized instructions. For example, PDIST calculates the sum of the absolute values of the differences of two sets of eight pixels. This instruction vastly accelerates the motion estimation process in MPEG and other video-compression algorithms, allowing UltraSparc to perform real-time MPEG-1 encoding. MMX will accelerate motion estimation compared with non-MMX processors, but not as much as VIS.

For More Information

For more information on MMX, obtain the document *Intel MMX Technology Programmer's Reference Manual* by calling Intel at 800.628.8686, or access the Web at *www.intel.com/pc-supp/multimed/mmx/index.htm*, or contact your local Intel sales office.

UltraSparc also includes instructions to accelerate the discrete cosine transform (used in video decompression), pixel masking, and 3D rendering. As with other SPARC instructions, the VIS instructions use three-operand encoding rather than the two-operand MMX style, which would alleviate some of the awkwardness seen in the above MMX code examples.

The VIS instructions also operate on 32 registers instead of the limited set of 8 MMX registers. A 4 × 4 matrix of constants, commonly used in digital filters, fits within the VIS register set but requires extra memory accesses in MMX. MMX has one advantage in its multiply-add instruction; it takes two instructions to perform this task under VIS.

While MMX may not go quite as far as VIS, it provides a much wider range of multimedia-oriented instructions than any other popular instruction set. HP's recent processors include some parallel arithmetic *(see* **080103.PDF***)* but operate on only two 16-bit quantities at once.

To date, the other leading desktop RISCs—PowerPC, MIPS, and Alpha—have somehow failed to implement multimedia instructions, despite the significant performance benefits and minimal cost. One advantage that PowerPC has over current Intel chips is in floating-point performance, which can be used to speed audio processing and speech recognition, for example. For these multimedia applications, MMX processors should improve Intel's position.

Both NexGen and Cyrix have been developing their own multimedia extensions to the x86 instruction set. AMD's purchase of NexGen and its subsequent licensing agreement with Intel *(see* **1001MSB.PDF***)* ensure that company's processors will move to MMX, starting with the K6. Cyrix has said its M2 processor, due in early 1997, will include its own multimedia extensions. We expect Cyrix will eventually switch to MMX, although perhaps not in the first version of the M2.

MMX to Appear Mainly in Drivers

Programming in MMX is challenging. Taking full advantage of the SIMD architecture often requires unrolling loops and carefully arranging instructions, yet there is no compiler support planned other than allowing in-line MMX assembly code. Intel plans to provide libraries of routines for common multimedia functions as well as an assembler and debugger that support MMX. Over time, third parties will also supply MMX tools and library code.

Most multimedia applications will take advantage of the new instructions simply by calling MMX-enabled drivers or including the new library routines. One advantage of relying on drivers is that an application can automatically take advantage of a hardware accelerator for 3D graphics, sound, or MPEG decoding if one is installed. This model, of course, pushes the coding effort onto the driver writers.

A few applications will have to incorporate MMX instructions directly. These include programs that do image processing (e.g., Photoshop) or speech recognition, since APIs for these tasks are not yet defined. Fortunately, a significant speedup can often be obtained by simply modifying a few critical inner loops.

One problem is managing separate versions of each application for MMX and for non-MMX systems, which will be the majority of the installed base for several years. Software can check bit 23 of the CPUID to determine if a processor implements MMX. Again, simply relying on drivers eliminates this problem for the application.

Improved Multimedia Fuels PC Sales Growth

Although the installed base of MMX processors is nonexistent today, it will grow rapidly. We project that more than half of Intel's 1997 processor shipments, and virtually all thereafter, will contain MMX, totaling more than 30 million processors by the end of 1997. The 50–100% performance gain will motivate multimedia software vendors to use MMX; those that don't will be uncompetitive. Intel expects dozens of drivers and applications to be shipping with MMX code when P55C systems first appear; this number will quickly increase during the course of 1997.

While the two are not directly connected, MMX is clearly designed to accelerate native signal processing (NSP). NSP will be used mainly in low-end systems to perform multimedia tasks; more expensive PCs are likely to include hardware accelerators. The P55C will significantly increase the baseline multimedia capabilities of low-end systems without accelerator chips. As Klamath reaches the mainstream in 1998, it will offer another big performance boost, possibly eliminating accelerator chips even in midrange systems.

It's not every day that you get a sizable step up in performance with minimal die cost, but that's what MMX promises. Once the combination of MMX-based processors and applications reaches the market, it should increase the growth of PC sales, particularly in the already hot consumer market, where multimedia is used most today. Even businesses will see the benefit as more use their PCs for videoconferencing and similar tasks.

One problem will be measuring this new performance level. Current benchmarks (SPEC95, Winstone, etc.) do not measure improvements of this type. The computer industry, with prodding from Intel, will most likely come up with new benchmarks to solve this problem, but perhaps not in time for the P55C's debut. Fortunately, the increase in graphics and video performance is something the buyer can see. M