3DNow Boosts Non-Intel 3D Performance

Compact Instruction Set Extends MMX With Packed FP Operations

by Brian Case

Following through on promises made at last year's Microprocessor Forum (see MPR 10/27/97, p. 19), AMD has formally introduced x86 instruction-set extensions for speeding 3D-graphics applications. As we speculated, the parallel-floating-point instructions pack two 32-bit floating-point values into a single MMX register and compute two results in parallel. Also confirming speculation, the first implementations will be able to issue and execute two arithmetic instructions simultaneously, yielding a peak throughput of four floating-point operations per cycle.

Since last October, AMD has made a few name changes. Along with co-collaborators Cyrix and IDT, the company has officially dubbed the technology 3DNow! instead of AMD-3D (our style, however, is to leave off the gratuitous exclamation point). Wisely, the three companies agreed to support a single MMX-like, parallel floating-point instruction set instead of each trying to promote proprietary designs. In addition to being vendor-neutral, the name 3DNow emphasizes that users need not wait for Intel's forthcoming Katmai processor to achieve 3D acceleration.

To agree on a common set of extensions, the companies were forced to make some changes to their original designs. Because AMD already had first silicon of its 3D extensions, the others essentially dropped their own designs, despite some superior features. Cyrix, for example, planned for better exception handling (see MPR 10/27/97, p. 22), and IDT sacrificed its expanded floating-point register file (see MPR 11/17/97, p. 17). AMD's changes were simply to drop three instructions that the others didn't want to implement.

AMD's first chip with 3DNow is the K6-2 (see MPR 6/1/98, p. 16), formerly known as the K6 3D. IDT says its WinChip 2 3D (see MPR 6/1/98, p. 1) will ship in July. Cyrix will include 3DNow in products using its Cayenne core, the first of which, the integrated MXi processor, is expected to ship late this year.

New Instructions Focus on Basic Arithmetic

As Table 1 shows, 3DNow consists of a relatively lean set of 21 instructions that do for single-precision floating-point data what the original 57 MMX instructions do for integers. (Centaur originally planned a set of 53 new instructions.)

Ten 3DNow instructions provide basic add, subtract, multiply, divide, and square-root functions. Addition is provided in two forms. As illustrated in Figure 1, PFADD sums corresponding values from the two source operands and places the results in the corresponding positions; PFACC sums the two values in the first source operand to produce one result and sums the two values in the second source operand for the second result. PFADD is used to form intermediate results in, for example, a matrix multiply, while PFACC performs the final steps of two dot products. The choice of PFADD or PFACC depends on how data is packed into operands.

Divide and square root are supported by providing primitives that compute approximations of reciprocal and reciprocal square-root. The advantage of this approach is that these primitives are easy to implement and can be fully pipelined. Properly scheduled code can reap high performance from simple hardware.

The PFRCP or PFRCSQRT instructions provide an approximation with 14 or 15 bits of accuracy via table lookups; if this accuracy is sufficient, two divisions can be computed with only four instructions, including operand memory loads, with four cycles of latency. If full single-precision accuracy is required, the PFRCPIT1/PFRSQIT1 and PFRCPIT2 instructions are used to implement efficient Newton-Raphson iteration. Six instructions, including the operand loads, can compute two 24-bit-precision divides with eight cycles of latency. Some programs will be able to achieve even greater efficiency by scheduling integer operations along with the 3DNow instructions.

Subtraction is supplied in normal and in operand-reversed versions to make up for the limitations of the x86

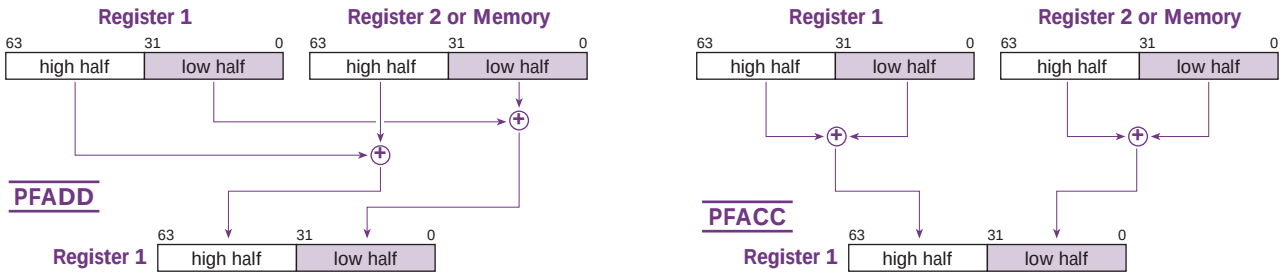

Figure 1. The PFADD and PFACC instructions both perform two packed floating-point adds, but the way they combine operands is different. The PFACC instruction is used at the end of a vector dot product to compute a single final sum. The PFADD instruction is usually used simply to compute two independent sums simultaneously.

Mnemonic	Description
FEMMS	Fast entry/exit of MMX or FP state
PREFETCH	Prefetch at least 32-byte line into L1 data cache
PAVGUSB	Packed 8-bit unsigned integer average
PMULHRW	Packed 16-bit int rounded multiply, keep high half
PI2FD	Packed convert 32-bit integer to FP
PF2ID	Packed convert FP to 32-bit integer
PFADD	Packed FP add
PFACC	Packed FP accumulate
PFSUB	Packed FP subtract
PFSUBR	Packed FP subtract reverse
PFMUL	Packed FP multiply
PFRCP	Packed FP reciprocal approximation (table lookup)
PFRSQRT	Packed FP reciprocal sqrt approx (table lookup)
PFRCPIT1	Packed FP reciprocal first step (Newton-Raphson)
PFRSQIT1	Packed FP reciprocal-sqrt first step (N-R)
PFRCPIT2	Packed FP reciprocal/recip-sqrt second step (N-R)
PFMIN	Packed FP minimum
PFMAX	Packed FP maximum
PFCMPGE	Packed FP compare, greater than or equal to
PFCMPGT	Packed FP compare, greater than
PFCMPEQ	Packed FP compare, equal to

Table 1. 3DNow adds 21 instructions to the x86 architecture. Most deal with two 32-bit floating-point numbers packed into a single 64-bit MMX register.

architecture. The x86's two-operand format combined with only eight registers makes it useful to have subtract-reverse.

Two new integer instructions plug holes in the original MMX instruction set. PAVGUSB computes the unsigned, rounded integer averages of the eight corresponding pairs of bytes in two operands. This instruction was included to speed motion compensation in MPEG-2 video decoding. Interestingly, Cyrix's parallel-sum-of-differences instruction for accelerating motion estimation (used in video encoding) didn't make the 3DNow final cut.

The PMULHRW instruction calculates the signed, rounded integer products of the four corresponding pairs of 16-bit words in two operands and stores the high 16 bits of each product. This operation is a numerically more accurate version of the existing MMX PMULHW instruction, which truncates instead of rounding.

Two Instructions Reduce Wasted Cycles

The FEMMS instruction is a faster version of the existing EMMS (Empty MMX State) instruction. EMMS leaves the shared MMX/floating-point registers in a valid state, but setting a valid value into each register and register tag takes time. FEMMS speeds the switch between MMX and floating-point code by leaving the shared registers in an undefined state. The code being entered can therefore make no assumptions about the values in the registers, but code normally operates under this assumption anyway.

Though widely implemented in many RISC architectures, 3DNow's PREFETCH instruction provides a capability that is fundamentally new to the x86 architecture. The action of PREFETCH is to bring a line of data into the L1 data cache; it has the same semantics as a byte load operation, but data is not actually loaded into a processor register. For the K6-2, the line size is 32 bytes; for all future implementations of PREFETCH, a cache line is specified to be at least 32 bytes. The byte address generated by PREFETCH need not be aligned in any special way, such as on a cache-line boundary.

PREFETCH is important for applications to exploit 3DNow instructions, because the raw processing power of the parallel arithmetic will be wasted if even a few cycles are spent idle waiting on cache misses. With careful scheduling of PREFETCH operations, it will be possible for some applications to significantly reduce time wasted due to cache misses. AMD's "Code Optimization Application Note" shows an example of a matrix multiply that uses PREFETCH along with other carefully scheduled 3DNow instructions to achieve maximum performance.

The PREFETCH instruction is available in two variants initially, and more can be defined if needed. The three register/opcode bits in the *modR/M* byte of the PREFETCH instruction determine the variant. Thus, six new variants can easily be defined.

PREFETCH loads a cache line and sets its MESI-protocol state to Exclusive, while the PREFETCHW variant loads the line and sets the MESI state to Modified. If it is known that the prefetched data will be modified, the PREFETCHW opcode will save one cycle that the PREFETCH opcode would spend setting the cache line MESI state to Exclusive.

While the PREFETCH instruction is clearly beneficial for speeding up code with 3DNow operations, it will also help MMX code and some applications that use only the standard x86 instructions. It seems likely that Intel will introduce one or more prefetch instructions in Katmai.

PREFETCH brings to the x86 some of the benefits of speculative loads, which have received significant attention lately, especially in the context of new architectures like IA-64 (see MPR 10/27/97, p. 1). Like a speculative load, if an exception is detected, no activity occurs and the PREFETCH is treated as a no-op instruction. A true speculative load, however, also hides the L1 cache latency by actually loading the data into a target register. With its dearth of registers, a true speculative load probably doesn't make sense for the x86 architecture.

Rounding, Overflow Handling Simpler Than IEEE

The 3DNow single-precision floating-point format is compatible with IEEE-754, but results computed by 3DNow instructions do not fully comply with the IEEE standard. Where IEEE dictates support for four rounding modes, 3DNow supports only one, which can be either round-to-nearest or round-to-zero (i.e., truncation). The choice is up to the designers of each implementation of 3DNow. AMD's first implementation, in the K6-2, will provide round-to-nearest mode.

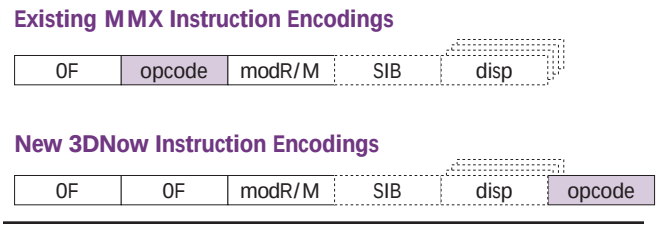

Figure 2. Most 3DNow instructions have their opcode at the end of the instruction, where an 8-bit displacement would normally appear in the x86 format. This scheme has a minimum impact on the existing x86 opcode space, but it makes 3DNow instructions slightly longer and demands sophisticated decoding logic to hide the decoding overhead.

Regardless of the mode implemented by the hardware for the computation instructions, the PF2ID and PI2FD integer/floating-point conversion instructions always implement round-to-zero mode.

Overflow is also handled in one of two ways, determined by the implementation. All overflowing results are saturated to either the appropriately signed value with maximum representable magnitude or to the appropriately signed infinity. The K6-2 implements the former approach, but perhaps saturation to infinity will be implemented in chips from Cyrix and/or Centaur. Values that underflow the minimum representable value, which is $\pm 2^{-126}$, are clamped to zero. Finally, 3DNow floating-point instructions do not generate exceptions and do not set any status flags.

All of these restrictions are acceptable in the application domain of 3D scene generation, either because exceptional conditions and out-of-range results and operands are detected by algorithms before the 3DNow instructions come into play or because the behavior as defined by 3DNow is sufficient. If full IEEE compatibility or double- or extended-precision calculations are required, programs must use the standard x86 floating-point instructions.

Instruction Format Uses Few x86 Opcodes

AMD chose an encoding scheme for the 3DNow instructions that minimizes the use of unassigned x86 opcodes. Intel's encoding of the MMX instructions uses the initial 0x0F byte ("two-byte escape") followed by a second opcode byte. In keeping with the traditional x86 format, the second opcode byte is followed by the *modR/M* byte, to specify operand location information, and then any SIB (scale/index/base) and/or displacement bytes.

As Figure 2 shows, the 3DNow encoding uses the previously unassigned 0x0F code in the second opcode byte, creating what could be called a "three-byte escape." The familiar x86 instruction components come next, including the optional SIB and displacement. The distinguishing 3DNow opcode byte comes at the end, where an eight-bit displacement would normally be placed according to the traditional x86 format rules.

This encoding scheme is certainly nontraditional and potentially creates a serial decoding headache: the instruction's function is specified by a byte whose location isn't known until the number of SIB and displacement bytes is known. But with the aggressive decode and predecode strategies used in modern x86 instruction-fetch units, this nonconformist format should cause no significant problem. Plus, with the *modR/M*, SIB, and displacement bytes in standard positions, changes to existing decode logic were minimized.

A significant advantage of this encoding scheme is that only a single previously unassigned two-byte opcode is needed to encode the 3DNow extensions, which minimizes 3DNow's "footprint" in the traditional x86 opcode space.

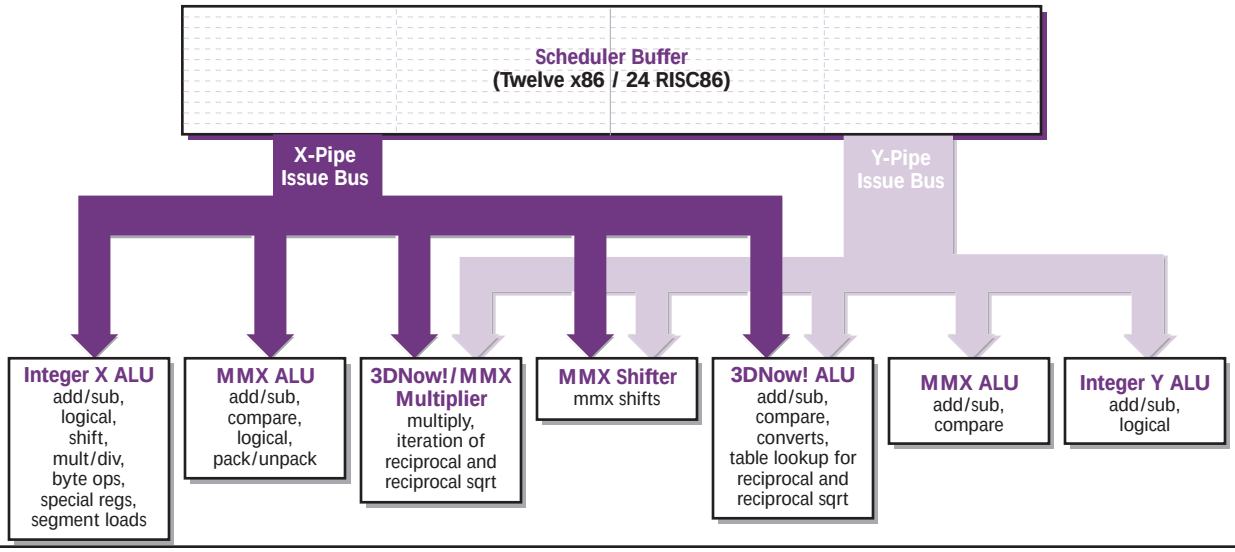

Figure 3. The new execution units that implement the 3DNow instructions are shared by the K6's two execution pipelines. Both pipelines have equal access to the shared units, and as long as there are no data dependencies and no contention for resources, instructions may issue with no restrictions. Note also the addition of a second MMX execution unit, and that the X-pipeline is, in general, more capable than the Y-pipeline.

AMD did take the liberty of assigning two-byte opcodes for two instructions: FEMMS and PREFETCH. These instructions are generally useful both for 3D and non-3D code, so it makes sense to keep their encodings short.

An important question is whether the 3DNow instruction encodings will conflict with the encodings for Intel's forthcoming Katmai new instructions (KNI). Sources indicate that Intel has used unassigned two-byte opcodes, which means that KNI and 3DNow should not overlap. Wisely, Intel's competitors seem to have said to Intel, "We recognize your right to the juiciest opcodes; let's just stay out of each other's way."

Execution Units Shared by Two Pipelines

The first chip with 3DNow is the K6-2, which is now in production. To support the new instructions, its execution resources have been significantly enhanced over those of the K6. In addition to the execution units that support 3DNow, the K6-2 adds a second integer MMX unit, as Figure 3 shows.

One major execution unit, the 3D ALU, has been added to support 3DNow, and the existing MMX multiplier has been enhanced to support the extensions. The 3D multiplier handles the multiply and reciprocal iteration instructions, while the 3D ALU handles everything else: addition, subtraction, converts, compares, and table lookups.

AMD's implementation of the 3DNow floating-point instructions is fully pipelined, yielding a latency of two cycles and a throughput of one cycle. PAVGUSB and PREFETCH both have single-cycle latency and throughput. This performance is better than what Cyrix had originally planned for its 3D instructions (see MPR 10/27/97, p. 23), so it seems possible that Cyrix and IDT may choose to implement these instructions differently.

Figure 3 makes it clear that two similar 3DNow instructions cannot begin execution in the K6-2 simultaneously; only one 3D ALU or one 3D multiply can begin per cycle. The units are fully pipelined, however, so at most a single cycle of delay is incurred if two operations contend for the same resource. If two operations do not contend and operand dependencies permit, both may start execution on the same cycle, because the shared resources are equally available to the operation-issue buses.

3DNow Overshadowed by AltiVec, KNI

The core arithmetic instructions in 3DNow are roughly comparable to the basic floating-point operations provided by the recently announced AltiVec extensions to the PowerPC architecture (see MPR 5/11/98, p. 1). For example, both 3DNow and AltiVec use reciprocal approximation and iteration to perform fast divide and square root. 3DNow's table lookups provides 14 or 15 bits of accuracy, more than the 12 bits provided by AltiVec.

AltiVec goes far beyond 3DNow in most aspects, however, with 32 dedicated registers that are twice as wide, and it has unique instructions such as base-2 logarithm estimation, two-to-the-power, versatile multiply-accumulate, and general data permutation.

In addition, AltiVec is much more faithful to the IEEE standard in handling events such as overflow. Better IEEE compliance was necessary to meet the requirements of applications beyond 3D acceleration. For example, in audio applications, it is necessary to know when a computed value overflows, so audible distortion is not generated. 3DNow will be adequate for speeding 3D graphics, but AltiVec is more general-purpose, and KNI may be as well.

Intel has said that KNI, which is also expected to emphasize parallel floating-point computation, will consist of about 70 new instructions. Sources (see *www.tbcnet.com/~clive/vcomwinp.html#KNI*) suggest that KNI supports a set of new registers (see MPR 5/11/98, p. 4) that will each be able to hold four FP values. Thus, KNI will likely be much more powerful than 3DNow for a wide range of applications.

For More Information

For more information on 3DNow, access the Web at *www.amd.com/products/cpg/k623d/inside3d.html*.

3DNow, KNI Later

AMD, Cyrix, and Centaur have chosen a wise path, implementing as small a set of extensions as possible while still offering significant acceleration for 3D game applications. Games are important in the consumer market, where these chip vendors are successfully competing with Intel.

With Katmai set to debut early next year, 3DNow must garner support and achieve a payoff very quickly. Significant support from software vendors and a marketing blitz will help secure some consumer mindshare for the coming holiday buying season. Having all three chip vendors united in support of 3DNow presents the strongest position possible, but while some software ISVs may support both 3DNow and Intel's KNI, no vendor can justify supporting 3DNow instead of KNI.

3DNow will be unique in the market for a few months, but the technology is destined to coexist with Intel's. Thus, when KNI becomes available in Intel's chips, the value of 3DNow will quickly fade. The likely scenario is that 3DNow has only this holiday season to make its impact. But to many ISVs, especially 3D-game companies, gaining an advantage for even a single product cycle is enough motivation to commit to using a technology like 3DNow. Many of the ISVs interested in 3DNow are accustomed to making major modifications to their products once a year.

Although 3DNow is well designed, AMD, Cyrix, and IDT must work hard over the next few months to guarantee a return on their investment. Eventually, they will all implement KNI in their processors, and 3DNow is likely to end up as a footnote in x86 history. M