RISC Processors Can Emulate x86 Quickly

Fast Emulation Requires Both Hardware and Software Techniques

by Brian Case and Linley Gwennap

Despite offering significantly better price/performance, RISC processors have been unable to take market share from the dominant x86 family in the high-volume desktop market. The x86 has tenaciously held its lead due to its vast installed base of software, a base that is not compatible with RISC processors.

To overcome this incompatibility, RISC vendors have tried to convince software developers to port their applications to RISC instruction sets. Although this approach has had some success in niche markets, most general-purpose software is still unavailable on RISC. Now, many processor vendors are turning to a different strategy: acquiring the entire x86 software base in one fell swoop by emulating an x86 CPU.

During the past few years, products like Insignia Solutions' SoftPC and SoftWindows have become available, emulating the x86 processor and allowing users to run DOS and Windows on a variety of RISC platforms. These products provide a bridge to the x86 world, but many users are dissatisfied with the performance of emulated x86 applications, which is generally no better than that of a low-end 486SX.

More recently, processor vendors have begun to consider ways to improve the speed of RISC processors when emulating x86 code. IBM has confirmed that it is "investigating" ways to combine x86 execution with a RISC core in a new device reportedly called the PowerPC 615. MIPS Technologies' Glenn Henry has discussed a similar program under way at that company. International Meta Systems (IMS) is the furthest along: it is testing a prototype chip that executes x86 code on a proprietary RISC core *(see **0805MSB.PDF**)*.

There is a variety of ways to improve the performance of emulated x86 code, ranging from adding instructions that speed the software emulator to replacing it entirely with hardware emulation. A hardware-based strategy could be as "simple" as adding an x86 core to a RISC design or as complex as modifying a RISC pipeline to execute x86 instructions. These options offer different tradeoffs in emulated x86 performance, native RISC performance, manufacturing cost, and time to market.

Compatibility Is Two-Pronged Problem

Emulating the x86 instruction set has two aspects: decoding the x86 instructions and executing the operations (semantics) of the instructions. The CISC-style x86 instructions are difficult to decode because they vary in length, have inconsistent formats, and can have prefix bytes. A software emulator spends much of its time decoding the instructions and extracting their operands.

Executing the semantics of x86 instructions is also difficult for a standard RISC processor, which lacks x86-compatible condition codes as well as support for 8- and 16-bit ALU operations and the x86 multicomponent, segmented memory-addressing model. Without specific support, emulating each of these features requires many RISC instructions.

On the bright side, the x86 architecture specifies only eight registers, making it possible to directly map x86 registers onto the larger RISC register file while leaving enough room for the x86 segment registers and temporary values.

Floating-point instructions are also difficult for RISC-based emulators. Implementing the 80-bit operations of the x86 model with the 64-bit operations found on RISCs requires many instructions. Fortunately, most productivity applications do not use floating point, and many FP-intensive programs are either already ported or likely to be ported to popular RISC chips.

The following discussion focuses on the x86 integer instruction set. Even ignoring floating point, we can provide only an overview of some of the key decisions and issues involved in x86 emulation.

Software Solutions Cheap but Slow

A pure software solution is attractive because it adds no cost to the processor chip and can be used on any system. Emulation performance scales as native processor performance increases, and the emulation software can easily be upgraded and corrected.

There are some hidden costs, however. The first is the cost of the emulation software itself: Insignia's SoftWindows for Macintosh, for example, retails for $499 ($199 bundled with system). Another is the extra DRAM required to run the emulator. SoftWindows needs 13M of memory to run well on the Mac, more than is installed on most systems. Even in a 16M system, the emulator leaves room for the Mac OS but no native applications.

The biggest drawback of software emulation is its poor performance compared with native x86 chips. For example, ZD Labs' measurements of a 66-MHz Power Macintosh (*Macworld*, 5/94) show that CPU-bound benchmarks run about as fast as a 25-MHz 386. Application test performance is more varied but can be up to 25-MHz 486 speeds, depending on how much time is spent executing operating system code, some of which

runs natively. In any case, this speed—typical for software-only emulation—is inadequate for many x86 applications, especially in light of cheap Pentium boxes.

Accelerating Software Emulation

One way to improve x86 performance is to add hardware to accelerate the emulation software. Because the emulator spends much of its time decoding instructions, a valuable improvement would be to implement an x86 instruction decoder as a RISC execution unit, as Figure 1 shows. This unit could accept 32 bits of data—fetched from the x86 instruction stream—and extract the various register, opcode, and immediate fields. These fields could be placed into special registers or, assuming sufficient write ports into the register file, directly into predetermined general registers.

In addition to extracting register and immediate fields from the x86 instruction, the decode unit could use the opcode field(s) to generate a jump-table offset that the software emulator would use to index into a table of emulation routines. The offset would direct the emulator—via an indirect jump—to short routines for common instructions that can be decoded directly by the hardware. For long instructions that the decode unit cannot handle, the offset would direct the emulator to a routine that completes the decoding in software.

The decode unit would be accessed by a special instruction that specifies the source register containing the x86 instruction bytes and possibly one or more target registers. This structure could reduce the execution of a simple x86 instruction to four steps.

- Execute current x86 instruction (may take several RISC instructions).
- Load next x86 instruction into a register.
- Decode next x86 instruction (using decoder).
- Jump indirect to execution routine.

For efficiency, this generic routine overlaps the execution for one x86 instruction with the fetch-decode-dispatch overhead for the following x86 instruction. The "decode" instruction collapses potentially many shift and mask operations into a single RISC instruction.

This routine assumes the new RISC "decode" instruction automatically updates the emulated x86 PC. This action is not trivial, however, as any x86 instruction fetch must check for page faults and segment overruns. These checks cannot be done when instruction bytes are loaded, because the processor doesn't yet know how many bytes are in the current instruction. This problem must be addressed for proper x86 emulation.

Function Units Speed x86 Execution

Most x86 instructions cannot be executed by a single RISC instruction because of the x86's special condition codes and complex addressing modes. Many x86 instructions could be speeded by including an x86-style

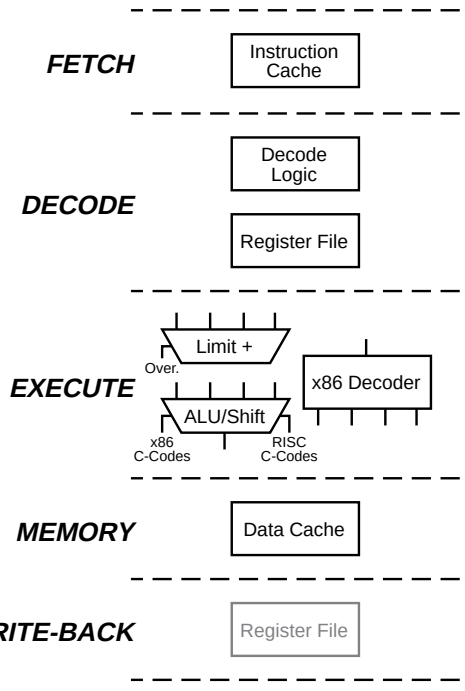

Figure 1. With the addition of four-input adders and a decode unit, a basic RISC pipeline can be modified for significantly faster x86 emulation without significantly increasing the die area or cost.

ALU to automatically generate the needed condition codes as well as a couple of three- or four-input adders to perform x86 address calculations and segment-limit checks. These could also be added as execute-stage function units, as Figure 1 shows, and accessed by new instructions that perform x86-like loads, stores, and arithmetic operations. Some of this hardware could be used to solve the PC-update problem noted above.

A potential problem with this approach is that extra inputs on the ALU may increase cycle time and thus decrease the maximum operating frequency, reducing native performance. Instead of the arrangement shown in Figure 1, it may be necessary for the EX stage to have three arithmetic units: a limit adder, an address adder, and a traditional ALU/shifter with two inputs.

This modified pipeline would execute RISC instructions as usual and still be able to accelerate x86 emulation. For example, consider an x86 memory-to-register ADD, which takes two cycles in the Pentium pipeline. With the pipeline in Figure 1, two instructions are required (neglecting the overhead of instruction decoding):

- LOADX86. This is a new "RISC" instruction that implements the x86 addressing model. It specifies a three- or four-part address, perhaps by naming some registers implicitly. It triggers an exception if the limit-check adder detects a segment overflow.
- ADD. This is the usual RISC ADD, but it also generates x86 condition codes, storing them in a special register.

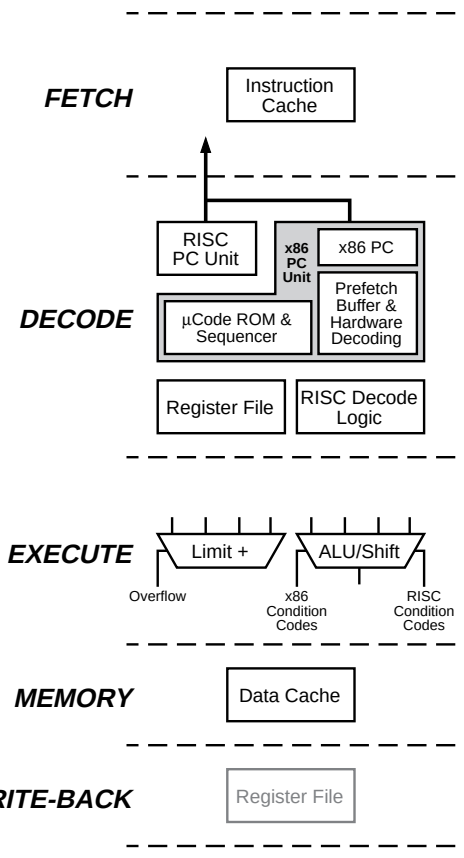

Figure 2. A combined RISC/x86 pipeline would look something like this. The execute stage would have an ALU and an adder to speed x86 address computations, and the decode stage would have fetch, decode, and dispatch logic for both RISC and x86.

This instruction takes two cycles, due to the load/use interlock, unless either the instruction flow or the pipeline is rearranged to avoid this penalty.

Note that this sequence assumes the x86 ADD instruction specifies 32-bit operands. If the instruction uses 8- or 16-bit operands, additional instructions or hardware are required for correct execution.

Emulation Improved with Modest Cost

After modifying the emulation software to use these new hardware features, most x86 instructions should take about half as many cycles as required by an emulator running on an unmodified RISC design. The new emulator would also have an improved cache hit rate due to the smaller instruction execution routines, which would further improve emulation performance and reduce system RAM requirements.

One drawback is that the emulator creates an instruction stream with some data dependencies and frequent unpredictable indirect branches, making it difficult to extract maximum performance from a superscalar RISC processor. The emulator could be coded to process two x86 instructions simultaneously to improve the chance of fully exploiting the superscalar RISC core, but doing so would make the emulator complex and more difficult to maintain; unpredictable x86 branches would still make it difficult for the emulator to reach peak performance.

In summary, a significantly modified RISC processor could reasonably double emulation performance compared with a standard RISC chip. The modifications consist of adding a couple of new function units to an existing design and, for a high-end chip, might increase the die area by only 5–10%.

This hardware-assisted approach retains some of the advantages (and disadvantages) of a pure software solution. With the emulation code in software, corrections and improvements can be made without changing the processor chip. Given that correct x86 emulation is difficult to achieve, this may be a compelling advantage. Also, a user can buy a base system and add x86 emulation later if needed.

A big disadvantage of this approach is that the emulation software is as much a part of the microprocessor product as is the chip itself. The software must be ported to all operating systems and platforms in which the microprocessor is used. Developing and supporting a software emulator may be too much to ask of most microprocessor vendors, but if a relationship could be developed with a company like Insignia, the software aspect could be manageable.

Hardware Approach Is Fastest

Even with special function units to accelerate decoding and executing x86 instructions, emulation performance significantly lags native performance. A fully hardware-based solution can use hardware pipelining to fetch, decode, and execute x86 instructions simultaneously. In this case, the control code would be contained in an on-chip ROM, but this code would be smaller than a software emulator because many tasks would be performed directly by the hardware. Some mechanism to switch between RISC and x86 execution (perhaps a mode bit in the processor status register) is also needed.

Figure 2 shows a RISC pipeline with an x86 PC unit that implements an x86 program counter, instruction fetcher, and instruction decoder. When the RISC chip operates in x86 mode, the x86 PC unit takes over the duties of keeping track of program flow, fetching instructions, and generating and dispatching RISC instructions into the pipeline(s). The PC unit would be essentially the same as is implemented by the 486 or Pentium, but it might directly decode and execute only the most frequently occurring x86 instructions.

This approach essentially creates an x86 processor on the RISC chip. The two personalities—RISC and x86—share most of the hardware of the chip, including

cache and execution resources, but have separate instruction fetch and dispatch logic. To keep the complexity of the x86 PC unit reasonable, it would be advantageous to "trap" to the RISC personality when a very complex or uncommon x86 instruction is encountered. The RISC routines could be either in an on-chip ROM (microcode) or in a software emulator (trap handler).

By modifying the RISC pipeline to efficiently execute x86 instructions, a processor designer can achieve the best possible x86 performance. Without doing a full design and extensive performance simulations, it is difficult to quantify the performance of such a chip. The modified pipeline will execute many x86 instructions at the same speed as a 486 or Pentium; some x86 instructions, such as memory operations and taken branches, will be slightly slower. Depending on the instruction mix, the modified RISC pipeline could achieve 75–100% of the performance of a 486 pipeline at the same clock rate.

This approach further improves cache utilization. With software emulation, the emulator itself is stored in the RISC instruction cache while x86 code and data are held in the data cache. Moving the emulator into microcode enables the RISC instruction cache to store x86 instructions, freeing the data cache for x86 data. This change, however, raises the bugaboo of self-modifying code, which x86 CPUs permit; the RISC instruction cache may have to be modified to handle this issue.

Clearly, the advantage of this approach is superior x86 emulation performance. Once a large amount of hardware assist is added and pipelines are restructured, an "x86-enhanced RISC" can become virtually indistinguishable from a "real" x86 chip.

As in a real x86 chip, however, the disadvantages are increases in complexity and design time and a likely sacrifice of native RISC performance, since the added complexity in the pipeline might make the critical paths longer or add a cycle to RISC instruction execution in certain cases. Some of the artifacts that can lengthen critical paths are the complexity of x86 instruction decode logic, microcode ROM access, microcode sequencing, and unaligned-access multiplexers on the data cache. If enough of these artifacts are grafted onto the processor, "the death of a thousand cuts" can significantly lengthen cycle time.

Die size may also increase significantly if large amounts of hardware assist are added. In the best case, the RISC/x86 chip could end up with a die size comparable to that of a high-end x86 processor, offering half the performance of the fastest x86 chip and a performance gain in its native RISC mode. In the worst case, the RISC/x86 chip could end up providing competitive performance for neither x86 nor RISC code. As one microprocessor designer who is currently wrestling with these tradeoffs points out, it is difficult to hit one price/performance target and even harder to hit two.

More Overhyped Vaporware

The prospect of x86-compatible RISC chips is currently generating a lot of speculation and hyperbole in the press. For example, in one issue of *PC Week* (5/2/94), an article and a separate editorial touted two different x86-on-RISC emulation schemes. The article claimed that the 100-MHz PowerPC 604 chip would be able, through software emulation, to execute x86 code at roughly the speed of a 60-MHz Pentium.

Is this realistic? As a sanity check, consider that 80-MHz 601-based Macintosh computers, using SoftWindows, are said to be achieving about 25-MHz 486 speeds running Windows. The 604, which is expected to have twice the native performance of the 80-MHz 601, might be able to double the emulation speed to 50- or 66-MHz 486 speeds, which, while excellent, is clearly not Pentium-class performance. As another sanity check, it is unreasonable to expect a 160-SPECint92 processor to deliver 60 SPECint92 in emulation: although an emulation overhead of 2.5 is theoretically possible—with aggressive dynamic compilation (see MPR 1/89, p. 4)—the complexity of this technique has prevented anyone from building a commercial version of such an emulator yet.

The editorial lavishes praise on the prospects for the IMS 3250 RISC chip with integrated 680x0 and x86 emulation and claims "its emulator technology is available now and will work with any RISC chip." While this statement may be true, it implies that adding the technology to "any RISC chip" is a simple task and can be completed quickly. Based on the discussion here, it should be clear that modifying pipelines, adding decoding hardware, and translating the x86 emulation microcode are nontrivial tasks.

Despite exciting, breathless accounts in the popular press, high-performance x86 compatibility is neither easy nor free. The decision to add x86 compatibility to a RISC chip must be made with the understanding that the instruction-set emulation is only part of the task of achieving PC compatibility, and that including the x86 instruction set as part of the chip architecture requires some sacrifices and tradeoffs.

—BC

Why Not Separate Processors?

Instead of expending lots of design effort to produce a dual-mode chip, it is possible to simply put two microprocessor chips in a system, or even on the same die. The advantages are excellent compatibility and performance, fast time to market, and no sacrifices in the RISC design; the crippling disadvantages are the added cost of the x86 processor, and the increased complexity of the system design. The disparate buses of the x86 and RISC chips must somehow use a common main memory and set of system peripherals. The operating system must be modified to coordinate two processors with different instruction sets and potentially different memory-management units.

Software Layer Also Needed

In favor of the hardware/software combination approach is the fact that in a real operating environment, where the main operating system is executed in the native RISC instruction set, some software management is required no matter how x86 compatibility is achieved. At a minimum, this software must create a virtual x86 environment, manage access to it, and enable simple functions such as cut-and-paste between environments. The software may also be required to emulate IBM PC hardware and assist DOS and Windows emulation.

Some or all of this layer is required regardless of whether the x86 instruction set is executed by the microprocessor alone or through a combination of hardware and software. Thus, requiring the use of a software package to implement part or all of the x86 compatibility may not be a significant burden.

In addition, the RISC microprocessor vendor may find it advantageous to avoid direct implementation of the x86 instruction set for several reasons:

- The documentation and maintenance of the x86 instruction set is a big burden. Documenting and maintaining a few instructions and registers that access the x86-acceleration hardware is significantly easier.
- It may be easier to avoid patent infringement with a small collection of acceleration hardware than with a complete processor implementation.
- The implementation is simpler. No microcode ROM is needed, which saves die area and design time.
- The substantial investment needed to develop x86 microcode is avoided.

Still, important PC system vendors will probably prefer complete on-chip x86 emulation precisely because of the software issues involved. From their point of view, the less special software required, the better. Thus, RISC vendors will probably be driven to build microprocessors that include full x86 compatibility on chip.

—BC

To reduce the cost of a separate x86 processor, one could add an x86 integer core to an existing RISC chip, sharing the caches and system interface. Assuming an x86 core is already available, this approach has many of the advantages described in the previous approach.

The 486 integer core operates at 100 MHz and requires 13 mm^2 in Intel's 0.5-micron, four-metal-layer process. If this core were combined with a PowerPC chip, for example, using IBM's advanced CMOS-5X process *(see **080504.PDF**)*, it would add only about 15% to the 601 or 10% to the 604—not insignificant but within reason for the important x86 compatibility it provides. In this technology, the 486 core should run at 120 MHz or faster, likely exceeding the integer performance of a 60-MHz Pentium.

Unfortunately, quirks in the x86 architecture require that modifications be made in the caches and system interface for such a shared approach to work properly. Either the x86 MMU (another 7 mm^2) must be added, or the RISC MMU must be modified for compatibility with the x86 segmented addressing model. The caches and bus interface must support (or emulate) unaligned accesses. The possibility of self-modifying code again rears its ugly head.

Still, for a vendor with access to a fast 486 core, this option enables a faster time to market than modifying the RISC pipeline. It provides competitive x86 performance without impacting native performance, unless some of the issues noted above degrade cache-access times. The increase in die area appears manageable and may be about the same as the merged-pipeline design. The danger, however, is that a dual-core chip may be so large that it loses its cost competitiveness in both the RISC and x86 markets.

Hardware Approach Is Difficult

Although the software-only approach has the benefit of being broadly applicable to a range of processor designs, including those with no special hardware, the performance of software-based solutions is inadequate to attract committed x86 users to RISC platforms. Emulation provides some value to RISC users that need to run one or two x86 applications that are not performance-critical. Even with some hardware assistance, software emulation speed will significantly lag the performance of high-end x86 processors.

On the other hand, hardware-based solutions can add significant cost and complexity to the RISC processor in order to boost x86 emulation speed. In the worst case, native performance may suffer as the complexity of the CPU increases. Even with full hardware emulation of x86 code, a software layer is still needed to manage access to the DOS/Windows environment (see "Software Layer" sidebar). But processors with x86 emulation in hardware will clearly offer x86 performance superior to software emulators and could come close enough to the speed of true x86 chips to attract PC users.

Hardware-based designs also require more design effort from the CPU vendor, which could instead reduce its effort by relying on a software house like Insignia to provide the needed emulation software. Legal issues must also be considered: while IBM, HP, and SPARC foundry Texas Instruments are protected by Intel patent licenses, other microprocessor vendors may be vulnerable to infringement charges if they include x86-specific hardware in their processors.

Any vendor that markets an "x86-compatible" RISC chip will have to prove this compatibility over time, just as AMD and Cyrix have in the past. IBM probably has the most credibility in this regard, but any new x86 vendor will be met with skepticism by some buyers for a year or more. Any minor incompatibilities found in the

early going will increase this skepticism and make it even more difficult to gain sales and design wins.

IBM, Intel Have Best Position

Because of these factors, the optimal emulation strategy may differ for each processor vendor. Digital, with limited resources and no Intel patent license, may find it best to rely on software emulation, perhaps including hardware assist in some of its processors. Sun is currently relying on its Wabi software emulation but will probably add hardware assist to some of its processors at some point, perhaps in a partnership with TI. HP has the patent license but not the resources to do more than minimal hardware tweaks. MIPS Technologies is aggressively pursuing the PC market and is developing some sort of hardware assistance for x86 emulation. Motorola, possibly lacking access to IBM's PowerPC 615, is said to be in talks with IMS.

IBM is probably in the strongest position among the RISC vendors. Along with its Intel patent license and tremendous development resources, it has significant expertise with the x86 architecture, having designed its own 486 core for the Blue Lightning family. Although the Blue Lightning core is restricted by IBM's contract with Intel, Big Blue recently obtained the rights to Cyrix's 486 and M1 cores *(see **080602.PDF**)* and has also purchased x86-related intellectual property from Chips and Technologies. Finally, IBM has already made a commitment to bring PowerPC to the desktop through its Power Personal Systems division.

We expect that the rumored PowerPC 615, using some of the hardware-based emulation techniques described here, will debut in 1995. This chip will likely combine a PPC 604 core with Pentium-class x86 performance, either by adding hardware to the RISC core or by adding a fast 486 integer unit to the chip.

Let us not forget Intel itself. Persistent rumors claim that the P7, already being designed and due to ship in 1997, will be based on some sort of RISC (or even VLIW) core processor. Intel swears that the P7 will be fully compatible with x86 software, but this could be achieved through the hardware emulation techniques described here. If any company can combine a RISC core with fast x86 emulation without ballooning the die size, Intel is the one. Of course, Intel also has the advantage of owning the necessary x86 patents.

Ultimately, it will be difficult for any RISC-based processor to match the absolute performance or cost/performance of a pure x86 implementation. With the appropriate combination of emulation software and hardware, adept RISC vendors may able to offer enough of a bridge to entice some x86 users to switch to RISC. Such a processor could be the ultimate weapon against the x86 hegemony—if the RISC companies can actually deliver it. Several appear to be willing to try. ♦

The Ultimate Weapon for RISC

Given the strategy of executing x86 applications in emulation, a RISC processor vendor must consider the required level of performance. Of course, more speed is always better, but more moderate levels typically make a better cost tradeoff. Assuming that a good RISC design can deliver twice the native performance of an x86 chip at a similar cost, it should be adequate for emulated x86 performance to be about half of the native RISC performance. This speed would be 2–4 times faster than that of the pure software emulators available today.

Ideally, the performance would not increase the chip's manufacturing cost, but an increase of 10–20% would be bearable. This premium could be absorbed by the profit margin of the processor or the system should the vendor wish to gain market share.

With such a weapon, a system maker could offer a RISC box that executed x86 code at the same speed as a comparably priced x86 system while running native applications at twice the speed of the x86 product. Such a product can be successful even if the number of native applications initially is small; if "x86" is replaced by "680x0," the above scenario aptly describes Apple's new Power Macintosh systems.

Over time, if a RISC/x86 product grows in popularity, more ISVs will port their applications to the native RISC instruction set, increasing the attractiveness of the product. In the best case, this feedback loop would result in a large base of native applications and vault the RISC machine past the x86 in sales. Of course, this outcome requires maintaining the 2:1 native performance advantage over the x86 and would take years.

This scenario is the dream of many RISC processor vendors and is the basis of the recent interest in x86 emulation. Because ISVs prefer to work with only a few processor architectures, however, it is likely that the first vendor to successfully start this feedback loop could block any others. Don't count Intel out; it could use the same strategy to introduce its own new architecture into the market *(see **0706ED.PDF**)*.

Note that this strategy works best with a base of existing RISC-based customers that will buy x86 emulation as a useful feature. These customers will provide an initial cash flow to the RISC vendor while it uses the new system to convert x86 users. Conversely, a vendor like IMS has no existing customer base; it must sell its chip essentially as a 486 competitor. In this case, emulation performance must be better than a comparably priced x86 chip, since the native RISC mode offers little advantage.

Apple may also be interested in such a strategy to convert x86/Windows users to the Power Macintosh platform. The software issues are more complicated in this case, but fast emulation of Windows applications could clearly help users make the switch to the Mac. IBM, however, may not share its PowerPC 615 chip.

—LG

