First Merced Patent Surfaces

Intel Document Reveals Possibilities for Processors With Dual Instruction Sets

by Linley Gwennap

Patents are two-edged swords: they offer a company long-term legal protection for specific inventions, but the patent procedure reveals the content of these inventions to the world. In the United States, patents do not become public until they are issued, which often takes years. In other parts of the world, however, patent applications become public 18 months after they are filed. Because of this process, an Intel patent application titled "Method and Apparatus for Transitioning Between Instruction Sets in a Processor," has recently become public.

The Intel application describes a processor, which we assume to be Merced, that executes both x86 instructions and a second "64-bit instruction set," which we assume to be IA-64. Intel patents contain many ideas that are never implemented, however, so there is no certainty that the application describes the current Merced or IA-64 implementations. Intel would not comment on the relationship between this application and the products being developed.

The most interesting feature of the patent application is a detailed description of several instructions used to switch modes and share data between the two instruction sets. These appear to be the first IA-64 instructions to be made public, and their descriptions shed some light on Merced.

The patent application is number 96/24895, administered by the World Intellectual Property Organization (WIPO). If granted, the patent would cover 81 countries from Armenia to Uzbekistan but not the United States; the application is based on an earlier U.S. patent application, 08/386,931, dated February 1995, roughly a year after the Intel/HP partnership was consummated.

IA-64 Instructions		
Mnemonic	**Operands**	**Name of Instruction**
x86MT	*isrc2, dest*	Move to x86 register
x86MF	*src2, idest*	Move from x86 register
x86SMT	*isrc2, dest*	Move to x86 segment register
x86SMF	*src2, idest*	Move from x86 segment register
x86FMT	*fsrc2, dest*	Floating-point move to x86 register
x86FMF	*src2, fdest*	Floating-point move from x86 register
x86JMP	*rel17/isrc1*	Jump and change to x86 ISA
EVRET	none	Event return
New or Modified x86 Instructions		
Mnemonic	**Operands**	**Name of Instruction**
JMPX	*rel32/16* *r/m32/16*	Jump and change to 64-bit ISA
IRET	none	Interrupt return

Table 1. IA-64 and x86 instructions for switching processor modes and accessing data, as listed in an Intel patent application.

Flexible Coding Philosophy

The application notes that Digital's early VAX systems also executed PDP-11 code. (Other processors to implement two instruction sets in hardware include Data General's Nova and ARM's Thumb.) The Digital processor required the operating system to execute in VAX code, and applications could not mix VAX and PDP-11 code. The Intel document describes a processor that can support operating systems and applications that use either or both instruction sets.

Executing x86 operating systems, such as Windows 95, is essential for Merced to maintain full x86 compatibility. To take advantage of the performance of IA-64, however, operating systems and applications must be recompiled and/or recoded. The mechanism described gives programmers "the option of implementing a new instruction set where justified by performance advantages and utilizing the existing software where justified by cost considerations." In other words, when porting operating systems and applications, only those portions that will benefit from IA-64 need be converted.

This philosophy would make the IA-64 transition resemble Apple's conversion from 68K to PowerPC. In the initial Power Macintosh, only a small percentage of Mac OS routines ran in native mode, although these routines were the most frequently used. Three years later, much of the OS has been converted to PowerPC, but some portions remain in 68K. By carefully choosing which parts to convert, Apple has kept overall performance at a reasonable level while reducing the development effort.

We believe IA-64 code will consume much more memory than x86 or even most RISC code *(see* **110302.PDF***)*. If only the performance-critical areas are converted to IA-64, the overall increase in code size will be greatly reduced. If instruction sets are mixed at a low level, the switching time from one mode to the other must be kept short, which should be possible in the structure described.

In some places, the document gives the impression that Intel will treat IA-64 as simply a 64-bit extension to x86, much as when the 386 pioneered new 32-bit modes. A key difference between this and previous x86 transitions, however, is that the new IA-64 instructions will presumably overlap many of the encodings of current x86 instructions. Thus, a mode bit is needed to properly interpret incoming instructions. The patent application describes methods for handling this mode bit.

New Instructions for IA-64, x86

The application outlines several specific instructions, listed in Table 1. Most of them are intended for IA-64, but two will be added to the x86 instruction set. Six instructions allow

A Glossary for Merced

The 96/24895 patent application uses a number of terms in a consistent fashion to describe the dual-mode processor. Several of these terms are listed below with our best guess as to their official meaning.

64-bit extensions, 64-bit ISA—The IA-64 instruction set.
Extension processor—The IA-64 processor.
Extension register—An IA-64 register.
Event—An interrupt, machine check, or other exception.
XIP—Extended (64-bit) instruction pointer (i.e., PC). This register corresponds to the x86 EIP.
XIP1—A copy of XIP saved during event handling.
XPCR—Extension processor control register. This 64-bit register corresponds to the x86 EFLAGS register.
XPCR.ISA—This bit in the XPCR indicates the current instruction-set mode (x86 or IA-64).
XPCR.d86i—If this bit in the XPCR is set, the processor is not able to decode x86 instructions.
XPCR.d86r—If this bit in the XPCR is set, the processor does not implement separate x86 registers.
XPCR.dxi—If this bit in the XPCR is set, the processor is not able to decode IA-64 instructions.
XPCR1—A copy of XPCR saved during event handling.

x86JMP: Jump to x86 ISA

WO 96/24895 describes this IA-64 instruction:

"The processor switches execution to the x86 instruction set and executes the next instruction at the target address. The relative form computes the target address in the 64-bit ISA relative to the current XIP and code-segment base, i.e., XIP=XIP + *rel17**4-CS_base. Note that the target-instruction pointer is converted into the effective address space. *rel17* is sign-extended and multiplied by four. The target XIP is truncated to 32 bits.

"The register form performs a far-control transfer by loading the code segment specified by the 16-bit selector in *isrc1[47:32]* and the 32-bit offset in *isrc1[31:0]*. If EFLAGS.VM86 is set, the processor shifts the 16-bit selector left by four bits to load the CS base. If EFLAGS.VM86 is zero, the processor loads the CS descriptor for the LDT/GDT and performs segmentation-protection checks. The target XIP is truncated to 32 bits.

"x86JMP can be performed at any privilege level and does not change the privilege level of the processor.

"If the target XIP exceeds the CS limit, an x86 GP fault is reported on the target instruction. CS segment-protection faults are reported on the target instruction. Gate descriptors are not allowed and result in a GP fault on the target instruction. If XPCR.d86i or XPCR.d86r is set, the instruction is nullified and a disabled x86 ISA fault or disabled x86 register fault is generated on the x86JMP instruction. If jump breakpoints are enabled, a jump-debug trap is taken after the instruction completes.

"On 64-bit-only subset implementations, x86JMP causes a reserved opcode fault.

"**Exceptions:** disabled-x86 ISA fault, disabled-x86 register fault, general protection fault, reserved-opcode fault, debug-jump breakpoint."

IA-64 routines to transfer data from 32-bit x86 registers to the 64-bit "extension" registers, and vice versa. They allow IA-64 routines to read x86 data, process it, and return it to the x86 registers for further processing by x86 routines.

Values are truncated or sign-extended to fit the target register size. Presumably, x86 and IA-64 routines could also exchange data via memory accesses, but this is not discussed in the application. Note that x86 routines cannot access the IA-64 registers.

Two instructions allow IA-64 routines to jump to x86 code, and vice versa. JMPX is a new x86 instruction similar in form and function to other x86 jumps, supporting both PC-relative and register/memory-indirect addressing. In addition, it toggles the mode bit that switches the processor to IA-64 mode, so it can jump to an IA-64 routine.

The corresponding IA-64 instruction is x86JMP, which switches the processor to x86 mode *(see "x86JMP" sidebar)*. If we assume this instruction is similar in form and function to other IA-64 jump instructions, we infer that IA-64 uses 17-bit relative branch addressing and also supports branching through a register. Like most RISC architectures, IA-64 shifts the branch target address left by two bits, implying that IA-64 instructions are 32 bits wide.

The patent application implies that an interrupt places the processor into x86 mode, regardless of whether the interrupted code was running in x86 or IA-64 mode. The interrupt handler may include a JMPX instruction, however, allowing it to complete in IA-64 mode. Thus, two instructions are supplied to return from an interrupt, also known as an "event" in IA-64 lingo *(see "Glossary" sidebar)*.

The standard x86 instruction IRET (interrupt return) has been modified to support the new instruction mode. After popping the program counter and flag register from the stack, IRET examines a new flag, XPCR.ISA, to determine if the interrupt occurred in IA-64 mode. If so, it saves the flags in the new XPCR register and loads the program counter in the XIP register. If XPCR.ISA is not set, IRET returns to x86 code using the same procedure as in current x86 processors.

The corresponding IA-64 instruction, EVRET (event return), assumes the XIP and XPCR have been saved in shadow registers instead of on the stack. This method of saving state is similar to the way PA-RISC processors handle interrupts. Like IRET, EVRET can return to either x86 or IA-64 code, depending on the state of XPCR.ISA.

Implications for IA-64

Much of the terminology in these IA-64 instructions is reminiscent of PA-RISC instead of x86. For example, the destination register follows the source registers, and where there is only one source register, it is *src2* rather than *src1*. The x86 term "illegal opcode" has been replaced with HP's term "reserved opcode." These items support rumors that most of the IA-64 instruction-set definition was done by HP, not Intel, and was largely complete before the Intel/HP partnership was officially launched. These similarities may make it easier to convert PA-RISC code to IA-64.

The instruction descriptions imply that IA-64 is a full 64-bit architecture: both the general registers and the program counter are 64 bits wide, allowing both instructions and data to be accessed linearly throughout a 64-bit memory space. Intel's influence is apparent, however, on the floating-point side: the IA-64 floating-point registers are 80 bits wide.

The description of EVRET contains the following note: "EVRET does not perform a memory fence operation nor is full serialization performed like the SRLZ instruction." We certainly assume that Merced will reorder memory operations for maximum bandwidth, but this note may imply that IA-64 instructions can execute out of order, contrary to our working assumptions.

Similarly, the document notes "IRET serializes instruction execution." This would seem to imply that x86 instructions can execute out of order in Merced. This would be less surprising: to achieve high performance on x86 code without translation, instruction reordering is a must.

There is nothing in the application that supports the idea of a traditional VLIW architecture, that is, a unified long instruction word. In fact, the RISC-like concepts of 32-bit instructions encoding single operations (e.g., move to x86 register) are antithetical to VLIW. The application does not rule out the option that these 32-bit instructions contain "grouping" bits to simplify parallel dispatch, nor does it specify, for example, the size of the register file. Based on this description, however, it appears IA-64 is not as radical a departure from the concepts of RISC as the companies had originally indicated.

Possible Merced Implementations

The application gives less insight into the high-level design of Merced, as it provides several different alternatives for implementing the two instruction modes. All but one use a single execution unit to process instructions, rather than having an x86 processor and an IA-64 processor, for example. This execution unit would certainly have multiple function units, and these could possibly be partitioned into x86 and IA-64 units, but the application strongly suggests that Merced will have a single integrated processor core.

Most of the implementations include an x86-to-IA-64 translator. For thoroughness, the application covers translators implemented as state machines, as logic devices, or in microcode. One design translates x86 instructions before they are cached; others translate them after they pass through the cache. Figure 1 shows one of the latter versions.

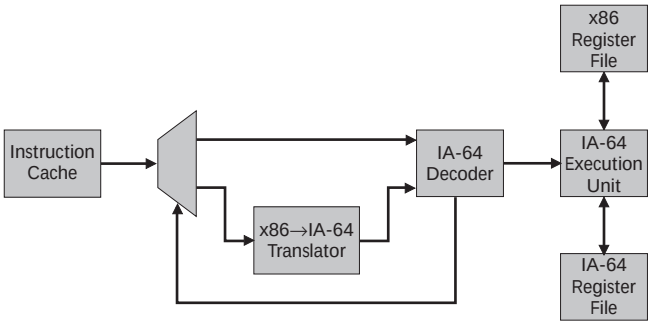

Figure 1. One of several possible implementations given in the Intel patent application shows instructions from the cache being fed to either an x86 translator or an IA-64 decoder before execution. All instructions are executed in a single execution unit. (Source: WO 96/24895; x86 and IA-64 labels added by MDR)

All implementations in the application include separate register files for x86 and IA-64 data, as Figure 1 shows. The text notes, however, that some designs may alternatively alias the x86 registers onto the IA-64 registers, eliminating the need for two register files. The instruction descriptions note a bit in the control register that allows software to check whether the x86 registers are implemented *(see "Glossary" sidebar)*, implying that some IA-64 processors may use one technique while some use the other.

The application makes it clear Merced will execute x86 instructions in hardware. Its performance in x86 mode is highly dependent on the complexity of the "translator." A simple one-to-one translator would probably produce 486-like performance. To achieve competitive x86 performance, the chip must include a complex translator with multiple parallel decoders and a large reorder buffer. Alternatively, x86 translation could be done in software, supercharging the on-chip translator for maximum performance.

A Patent Is Not a Processor

One must be cautious in drawing specific conclusions from a patent or patent application. This application, like most, describes several methods of achieving the same result; if granted, this breadth will increase Intel's legal protection. The actual Merced implementation, however, could use any of the methods described here—or none of them.

In its broadest interpretation, this patent application could prevent other vendors from building dual-mode processors, even if neither of the instruction sets is x86. The core of the application, however, appears to be the claims that reference the implementation of specific x86 and IA-64 instructions. If granted, these claims could prevent competitors from building processors compatible with Merced, an outcome Intel certainly desires. M

For more information, additional excerpts from the Intel patent application WO 96/24895 are posted on our Web site at www.MDRonline.com/mpr/merced.