SPECIAL GIANT ISSUE COVERING THE 1997 MICROPROCESSOR FORUM

VOLUME 11, NUMBER 14
OCTOBER 27, 1997

MICROPROCESSOR REPORT

THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE

Intel, HP Make EPIC Disclosure

IA-64 Instruction Set Goes Beyond Traditional RISC, VLIW

by Linley Gwennap

Breaking out of the 1980s RISC mind set, Intel and Hewlett-Packard have designed a new instruction set, IA-64, geared toward the highly parallel processors of the next decade. IA-64 goes beyond previous CISC, RISC, and VLIW instruction sets with a new set of features that its creators call EPIC (explicitly parallel instruction computing). This strategy should give Merced, the first IA-64 chip, a leg up on its old-fashioned competitors when it debuts in 1999.

EPIC is similar in concept to VLIW (see MPR 2/14/94, p. 18) in that both allow the compiler to explicitly group instructions for parallel execution. This technique eliminates much of the dependency-checking and grouping logic that consumes an increasingly large portion of advanced RISC and x86 processors. EPIC's flexible grouping mechanism solves VLIW's two fatal flaws: excessive code expansion and lack of scalability.

The new instruction set attacks other problems with current architectures. With four times as many addressable registers as a typical RISC processor, IA-64 eliminates the need for register renaming and reduces time-consuming cache accesses. When cache accesses are required, speculative loads can hide cache latency even when branches are in the way. Some of these branches can be eliminated entirely with predicated execution, reducing opportunities for onerous branch mispredictions.

Speaking at the recent Microprocessor Forum, architects Jerry Huck (HP) and John Crawford (Intel) disclosed these key features of IA-64 but did not provide a description of the new instruction set. Nonetheless, what was revealed clearly tags IA-64 as a new type of instruction set compared with today's RISC and x86 chips. RISC processor vendors can't simply retrofit these features into their existing instruction sets, forcing them to create new instruction sets or, as Intel has in the past, limp along with an inferior design.

Compiler Provides Explicit Directions

Modern compilers analyze a program to exploit opportunities for parallel instruction execution, carefully arranging the instructions for optimum performance on today's superscalar processors. Yet the serial programming model of RISC and CISC instruction sets forces the processor to dynamically re-evaluate the compiled code, instruction by instruction, and perform its own parallelization. Today's instruction sets provide no means for the compiler to communicate parallelism to the hardware, forcing the processor to duplicate this work using complicated out-of-order circuitry.

Combining several instructions into a single group is a technique that dates back to early VLIW designs, but IA-64 provides two key advantages. The early VLIW processors suffered from severe code expansion because, in many cases, instructions cannot be grouped due to dependencies. This problem prevents a VLIW machine from using all the instruction slots in a single long instruction word.

IA-64 instructions use a unique format that allows the compiler to direct hardware execution without severely bloating the software. As Figure 1 shows, a single 128-bit "bundle" contains three IA-64 instructions along with "template" information about the bundle. The template

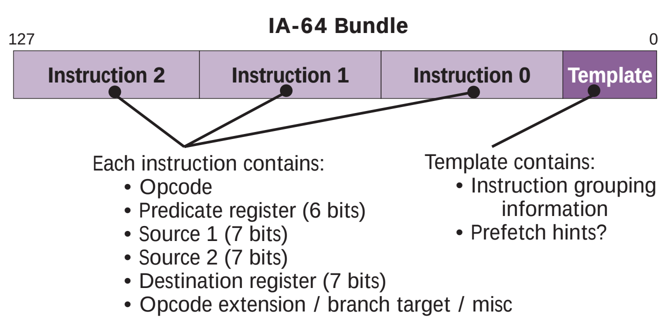


Figure 1. Three IA-64 instructions are encoded into a "bundle" along with a "template" that provides grouping information. The companies did not provide the width of the instructions or the template, or details about their contents.

indicates whether the instructions in the bundle can be executed in parallel or if one or more must be executed serially, due to register dependencies. The template also indicates whether the bundle can be executed in parallel with the following bundle. Bundles can be chained to create instruction groups of any length.

Although the speakers declined to describe the size or format of the template, this grouping information could be encoded in as few as three bits. The first bit would indicate whether the first instruction can be grouped with the previous bundle; the second bit would indicate whether the second instruction can be grouped with the first instruction; and the third bit would apply to the third instruction in the bundle. This hypothetical encoding method is similar to the one used by Texas Instruments' VLIW DSP, the 'C6201 (see MPR 2/17/97, p. 14).

This technique reduces code expansion by allowing a single ungrouped instruction to be combined with other instructions in a bundle, rather than forcing the rest of the bundle to be empty, as in a VLIW design. At the other extreme, the compiler can create arbitrarily long groups of parallel instructions without changing the bundle size.

This latter ability addresses another problem with the original VLIW processors: their instruction sets were mapped directly to the hardware, assuming a specific combination of function units. To run on a different VLIW processor, the code would have to be recompiled, possibly to a different instruction width, breaking binary compatibility.

IA-64 processors will include instruction-issue logic to take a potentially large group of parallel instructions and map them onto the available hardware resources. For a group of 12 parallel instructions, one processor might take two cycles to issue them all, whereas a more advanced implementation might issue them all in one cycle. The same binary code would thus run on both processors without any modification.

This issue logic makes an IA-64 processor more complex than a pure VLIW design, but the ability to build a family of binary-compatible processors is well worth the extra logic. This logic is much less complicated than the issue logic in an out-of-order superscalar processor.

A RISC processor will have a code-density advantage, however; since IA-64 fits three instructions rather than four into 128 bits, it requires 33% more bits to encode the same number of instructions. The actual impact could be a bit less, because IA-64 programs should require fewer loads and stores and fewer branches than a comparable RISC program. At the Forum, Huck characterized the code expansion as "modest." Code expansion will require IA-64 systems to have more main memory, larger caches, and higher bandwidth than a comparable RISC or x86 system.

Large Register Set Aids Compiler

The size of the register set is one of the key features that distinguishes an EPIC processor from a RISC or x86 design. Due to the physical constraints of 1970s manufacturing technology, x86 processors have only eight general-purpose registers. More advanced IC processes in the 1980s allowed RISC processors to include 32 general registers.

Today, more than a decade after the first commercial RISC chips appeared, IC technology has advanced enough to handle a larger register set. In addition, higher CPU clock speeds and deeper pipelines have increased the load-use penalty (the delay caused by accessing data in the primary data cache) from one cycle to two in many leading-edge designs. Thus, the advantage of keeping data in registers is greater than ever.

Recognizing these trends, the IA-64 architects included 128 general registers and 128 floating-point registers in their design. The compiler can take advantage of this increase by performing more aggressive optimizations. For example, unrolling short loops several times often increases performance, but each instantiation of the loop requires more registers to hold additional copies of the local variables. With 128 registers, the compiler can unroll loops more often while still leaving global variables in registers.

Most high-performance processors today use register renaming to increase the effective size of the register file. This technique dynamically maps the logical registers onto a larger physical register file. Some of these processors have as many as 64 physical registers, but because the compiler can't access all of them directly, they are often used inefficiently. Furthermore, these processors must include extensive circuitry to create and maintain the register map and to recover in case of an exception or mispredicted branch. With its large logical register file, an IA-64 processor needs no complicated renaming hardware.

Compared with x86, IA-64's large floating-point register file provides an extra benefit: it eliminates the x86's painful floating-point stack architecture. In x86 code, most FP instructions can operate only on the top of the eight-entry stack, forcing many pointless FXCH (exchange with top of stack) operations. RISC processors don't suffer from this disadvantage but typically provide only one-quarter of IA-64's 128 FP registers.

The larger register files will consume extra die space, but this area will not be significant in the advanced 0.18-micron process that will be used for the initial IA-64 processors. Context-switch times will also increase, due to the greater amount of context that must be saved.

The major disadvantage is an increase in instruction size. Seven bits are needed to address 128 registers, two more than for a RISC register file. Assuming IA-64 instructions use three operands, like RISC instructions, the larger register file will add six bits to each instruction. Fitting these extra bits into a 32-bit instruction simply isn't possible, one reason that IA-64 allows roughly 40 bits per instruction.

Predication Eliminates Some Branches

IA-64 processors will include 64 predicate registers (PR), each just one bit. Most IA-64 instructions include a predicate field; the instruction is executed only if the selected predicate register is "true." Predicates are generated by CMP instructions that compare the value of two registers (using a variety of conditions). A single CMP instruction stores the result of the comparison in one PR and automatically stores the inverse of the comparison in a second PR. The presenters did not specify whether CMP can perform logical operations (AND, OR) on PRs to create compound conditions.

This mechanism allows the processor to more efficiently handle the common IF-THEN-ELSE construction with small routines in each of the blocks. Figure 2(a) shows an example program with two instructions in each of the THEN and ELSE blocks. In an x86 or RISC processor, the code contains two branches, at least one of which is difficult to predict. Furthermore, there are few opportunities for parallel execution: at most, only the two instructions in each of the blocks can be executed together.

Figure 2(b) shows how an IA-64 processor will handle this routine. Once the CMP generates the needed predicate values, the instructions in both clauses are simply predicated by the appropriate value. This technique completely eliminates both branches, reducing code size and avoiding the opportunity for a multicycle misprediction penalty.

Another key benefit is the increased opportunity for parallel execution. Clearly, instructions 5 and 6 will not have any dependencies on instructions 3 and 4, so they can be executed in parallel. This doesn't really save any time, since a traditional machine would have executed either one set or the other. By merging all the instructions into a single basic block, however, instructions 7 and 8 (and subsequent instructions) can now be easily grouped with preceding instructions for parallel execution. Similarly, instructions 1 and 2 can be grouped with instructions 3–6, assuming the compare doesn't depend on their results.

Predicated, or conditional, execution is part of the ARM instruction set (see MPR 12/18/91, p. 11), generally considered to be a RISC architecture. ARM instructions can be predicated only on the current condition codes, however, preventing multiple conditions from being precomputed and used simultaneously. Both Philips' Trimedia (see MPR 12/5/94, p. 12) and the 'C6201 DSP store predicates in registers, but Trimedia uses the general registers, wasting an entire register to store a single bit. Like IA-64, the TI DSP has separate predicate registers, but only five of them.

Several desktop RISC architectures—including Alpha, SPARC v9, and MIPS IV—have conditional-move instructions. These instructions offer some of the benefits of predicated execution, but only for move operations. The IA-64 method is much more flexible, allowing any type of instruction to be conditionally executed.

Not all branches can be removed using predication. HP's Huck quoted a study from the University of Illinois (see *www.crhc.uiuc.edu/impact/papers/topic.html*) showing that, in an eight-issue machine, predication could eliminate more than half of all branches in a typical program and 43% of all mispredictions. Merced is likely to be roughly an eight-issue machine, and removing half of all mispredictions could boost its performance by 5–10%.

For More Information

Only limited information about IA-64 is available at this time. For more information, access Intel's Web site at *www.intel.com/pressroom/kits/events/mpf1097.htm.*

Speculation Hides Memory Latency

Despite the large register file in IA-64, many instructions will need to load data from memory. Many processors today require only one cycle to access data in the primary cache, but by the time Merced debuts, two-cycle data caches will be common; in fact, both Pentium II and HP's PA-8x00 use two-cycle caches today. If a load misses the primary cache, a hit in the second-level cache takes several cycles to complete. If the data is needed by a subsequent instruction, the entire processor might grind to a halt until the cache access is completed. Stalling an eight-issue machine for even two cycles wastes 16 issue slots, a problem that will grow worse as issue widths continue to increase in future IA-64 processors.

Out-of-order processors handle this situation by executing nondependent instructions while the load is being processed, dynamically reordering the instruction flow. This mechanism requires extensive and complex circuitry that consumes die space and is difficult to debug. It has the advantage, however, that if branches are predicted correctly, instructions that follow a conditional branch can be executed during the latency of the load.

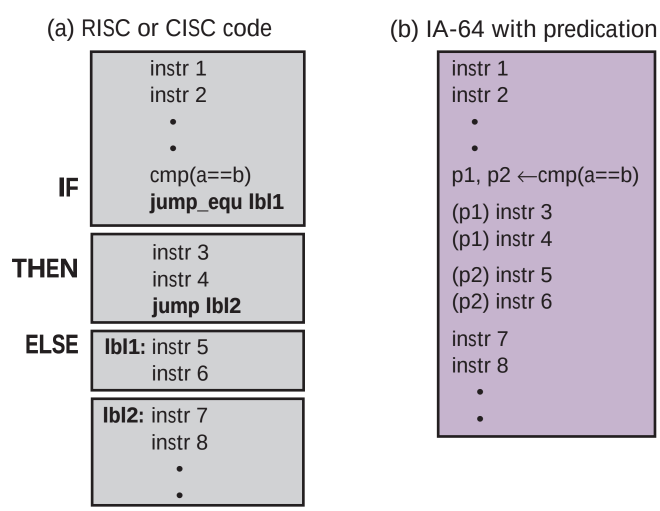


Figure 2. (a) In a RISC or CISC processor, branches control the program flow between THEN and ELSE clauses. (b) Predication eliminates the branches and allows the two clauses to be merged into a single basic block.

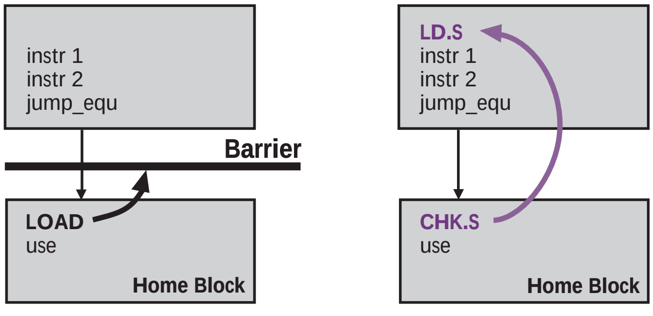


Figure 3. A traditional load must be placed within the same basic block as the "use" instruction, but a speculative load can be moved further up, providing more instructions to fill the hardware latency between the load and the use of its data.

The IA-64 designers wanted to eliminate this complex out-of-order logic, but stalling for load latencies would demolish any performance advantage. The obvious solution is for the compiler to insert nondependent instructions between a load and the first instruction that uses the data, enough to hide the latency. This code transformation is difficult, however, when a conditional branch is in the way, as Figure 3 shows. If the load is put before the branch, it would get executed even if the branch goes the other way. While this mistake could simply waste some of the processor's time, the situation becomes intolerable if the load generates an exception that never would have occurred if the load had been executed in the correct program sequence.

The solution, as implemented in IA-64, is a speculative, or nonfaulting, load. A speculative load, indicated by the .S suffix, will not trigger an exception; instead, if an exception occurs, the target register will be marked invalid. This is a fairly simple mechanism, requiring only that each register have a valid bit. As Figure 3 shows, a CHK.S instruction is later used to check whether the register contents are valid; if not, an exception occurs. The CHK.S instruction is typically placed before the first usage of the loaded data.

Using this mechanism, the load can be placed as early as possible in the code, as long as the address can be computed. If the data is never checked, no exception will be triggered; if an exception occurs when the data is needed, the exception will be recognized in the load's original "home block." Speculative loads thus provide the compiler with maximum flexibility to hide cache latency.

The need for the CHK.S instruction isn't clear. Presumably, the "use" instruction (the first instruction to use the data) could check the valid bit of the register and signal an exception. Intel's Crawford claimed there was a good reason for the separate CHK.S instruction and that it would be revealed "at a later date."

SPARC v9 (see MPR 2/15/93, p. 1) and PA-RISC 2.0 both allow data to be prefetched without signaling an exception if the address is invalid. Both RISC designs put the data into the cache but not into a register, however, so a subsequent load instruction is needed before the data can be used, costing at least one cycle. The IA-64 approach is superior in that it eliminates this delay.

Other New Features Likely

The initial disclosures of IA-64 closely match our previous expectations (see MPR 8/5/96, p. 14). Other features we expect to see in IA-64, but that were not disclosed at the Forum, include branch-prediction hints, multiway branches, and instruction-prefetch hints.

HP's PA-8x00 processors (see MPR 11/14/94, p. 1) rely on compiler-driven branch prediction. In this design, the compiler sets a bit in each branch indicating whether it should be predicted taken or not taken. This method can improve prediction accuracy for many branches, particularly on the first iteration. Other branches are better predicted using dynamic methods, such as branch-history tables. We expect IA-64 to include both compiler-driven and dynamic prediction methods.

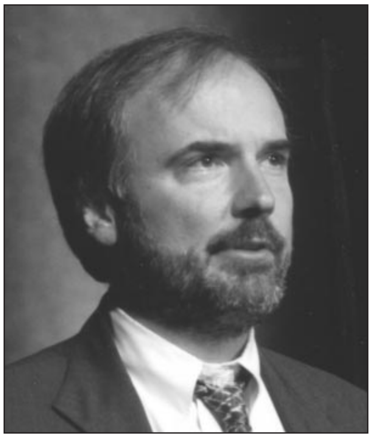
Multiway branches are useful in combination with predication. As described, predication can be used to combine two instruction streams after a conditional branch. If one or both instruction streams contain another conditional branch, a multiway branch would be an efficient way to transfer control to the correct path. Because a three- or four-way branch would require multiple target addresses, the extra bits in the IA-64 encoding might come in handy.

Speculative loads address the problem of data-cache latency, but instruction caches have a similar problem. Prefetching instructions into the cache before they are needed will hide nasty cache-miss penalties. IA-64 is likely to include some mechanism to do this, either through explicit instructions or a "hint" in the template field.

In other ways, IA-64 is likely to resemble today's RISC instruction sets. As the name implies, it will be a full 64-bit architecture, with 64-bit registers and a 64-bit addressing model, much like Alpha and other RISC processors. The FP registers are likely to be 80 bits wide, for compatibility with x86 programs. We expect IA-64 to use a load/store model, eliminating the x86's operations on data in memory.

The individual IA-64 instructions will include the standard array of arithmetic, logical, and floating-point operations. The instruction set will also include MMX-like features, packing several data words into a single register and operating on them in parallel. Although the vendors would not discuss the details of these instructions, they are likely to offer both parallel integer and parallel FP operations.

IA-64 is a bi-endian architecture, providing data-set compatibility with both the little-endian x86 and the big-endian PA-RISC. Intel did not otherwise discuss compatibility with x86, but it has said Merced will execute all x86 instructions in hardware. We expect the chip to include an x86-to-IA-64 translator and support intermixing of x86 and IA-64 routines (see MPR 3/31/97, p. 16).


MICHAEL MUSTACCHI

Intel architect John Crawford explains the benefits of moving to an EPIC architecture.


MICHAEL MUSTACCHI

HP architect Jerry Huck describes key features of the forthcoming IA-64 instruction set.


MICHAEL MUSTACCHI

Intel Fellow Fred Pollack says his company plans to follow Merced with an even faster IA-64 chip.

Intel Reveals Little of Merced

When Intel and HP first announced their partnership (see MPR 6/20/94, p. 1), they committed only to delivering a processor "before the end of the decade." At the Forum, Intel's Fred Pollack confirmed that Merced is on target for shipments in 1999, using a 0.18-micron process; we expect it to appear in systems around the middle of that year.

With two years to go before the first processor ships, Intel remains mum about any details of the Merced design, despite widespread speculation (see MPR 3/10/97, p. 9). We expect the first IA-64 chip to combine the radical new processor core with extensive x86 compatibility logic and a large on-chip cache. To support the high-performance core, the external interface is likely to provide much higher L2 cache and system bandwidth than Intel's current chips.

Even with just the features disclosed at the Forum, any IA-64 chip will have an inherent performance advantage over processors using current RISC or x86 instruction sets. Crawford demonstrated how predication and speculation alone could nearly double performance on a simple program, the Eight Queens loop. Real applications are unlikely to see such a performance increase, but a 30–50% architectural advantage is realistic.

Merced, however, probably won't demonstrate the full benefits of the instruction set, at least at first. For any new instruction set, compilers take some time to mature; all the simulation in the world can't match the development that can be done on dozens of real machines, and these won't be available until late next year. Thus, the initial benchmarks won't reflect the full performance of the processor.

Merced may also be hampered by the x86-compatibility logic. Even if this logic doesn't reduce clock speed or native-mode performance, its mere existence consumes die area that could have been used to enhance native performance.

Still, the advantages of IA-64 should provide a potent weapon. Pollack asserted that Merced will deliver "industry-leading performance" when it first begins shipping.

That lead is likely to grow over time as the IA-64 compilers mature and Intel squeezes more clock speed out of the Merced design. Pollack revealed that work has already started on a follow-on to Merced that will offer up to twice the performance of the initial chip in the same IC process. That chip, due in 2001, is likely to combine an even more powerful EPIC core with a new system interface that boosts performance to spectacular levels.

EPIC Offers Potential Performance Advantage

The EPIC design style offers clear advantages over today's RISC and x86 instruction sets, but it is not entirely new. Crawford pointed out that many of the ideas come from previous RISC and VLIW machines as well as from recent academic research. The concept of a large register set is certainly not new, but no currently popular instruction set can directly address as many as 128 registers.

The definition of EPIC is likely to be argued for as long as the definition of RISC was, but we'll take a first shot:

- Parallel instruction encodings
- Flexible instruction grouping
- Large directly addressable register file (128 or more)
- Fully predicated instruction set

Current processors such as Trimedia and the 'C6201 could be considered EPIC-like, and other vendors may create EPIC instruction sets in the future to compete with IA-64. Any new instruction sets, however, would cost billions of dollars to develop and would break compatibility with existing systems and applications, making it difficult to compete with Intel.

Ultimately, the best EPIC processors should establish a significant performance gap over RISC processors, much as the top RISC processors consistently offer better performance than CISC chips. Implementation issues will remain important: a strong RISC processor might match the performance of a weak EPIC design. But until other companies move to EPIC, IA-64 should help Intel change from a performance laggard to a performance leader. M