IA-64 ITANIUM, A VERY COMPLEX CHIP, NOW AVAILABLE
7 YEARS IN THE MAKING.

Intel pulls the plug on McKinley preview

By Ken Popovich IN SAN FRANCISCO

INTEL CORP.'S FAILURE LAST WEEK to go through with a scheduled presentation of a new 64-bit processor fueled speculation that the chip may endure delays.

Aside from the withdrawal at the International Solid-State Circuits Conference here, there are other indications the processor, code-named McKinley, may be running behind schedule. For one, the processor's design has yet to be sent off for manufacturing, as was expected late last year.

McKinley processors are scheduled to be released in pilot systems this year. Meanwhile, Itanium, Intel's original 64-bit chip, which was due to be launched last year, has yet to be released.

Intel officials downplayed the cancelled presentation. "Basically, after reviewing what was submitted to the conference, we just thought it was too early in the ballgame to release that sort of architectural information," said an Intel spokeswoman.

Intel, of Santa Clara, Calif., expects to use its 64-bit, or IA-64, chips to break into the high-end server market dominated by RISC processors from Sun Microsystems Inc., Hewlett-Packard Co. and IBM.

While the 800MHz Itanium, now due this quarter, will be the first available IA-64 chip, many within the industry view it only as a software development platform. However, the 1.2GHz McKinley is expected to perform twice as fast as Itanium on some applications and is seen as competitive with RISC.

Intel's IA-64 program, begun in 1994, has been plagued by delays. Although Intel maintains Itanium's delay won't affect McKinley, contending the two products have different design teams, one analyst said Itanium's delays will impact McKinley.

"The McKinley team is in flux until the Itanium team actually locks down the chip and delivers it in final form," said Rob Enderle, an analyst at Giga Information Group Inc., in Santa Clara. "That really hasn't happened yet."

McKinley's no-show at the ISSCC didn't go unnoticed by Intel's rivals. "What IA-64? What Itanium?" IBM researcher Robert Montoye asked during a panel discussion at the conference. "Has anyone seen an IA-64? Does anyone know when it's coming out?"

An official with Houston-based Compaq Computer Corp., which plans to integrate Intel's 64-bit chips into its servers, said Intel's cancellation will further erode confidence in the IA-64 products.

"People will assume the worst," he said, asking that his name not be used. "Intel has its work cut out for it in winning over support for the IA-64." *e*

IA-64 timeline

June 1994 Intel and HP announce plans to jointly develop a 64-bit chip code-named Merced

May 1998 Intel gives Merced mid-2000 ship date

Oct. 1998 Merced officially named Itanium

Dec. 1999 Intel ships pilot systems containing chip

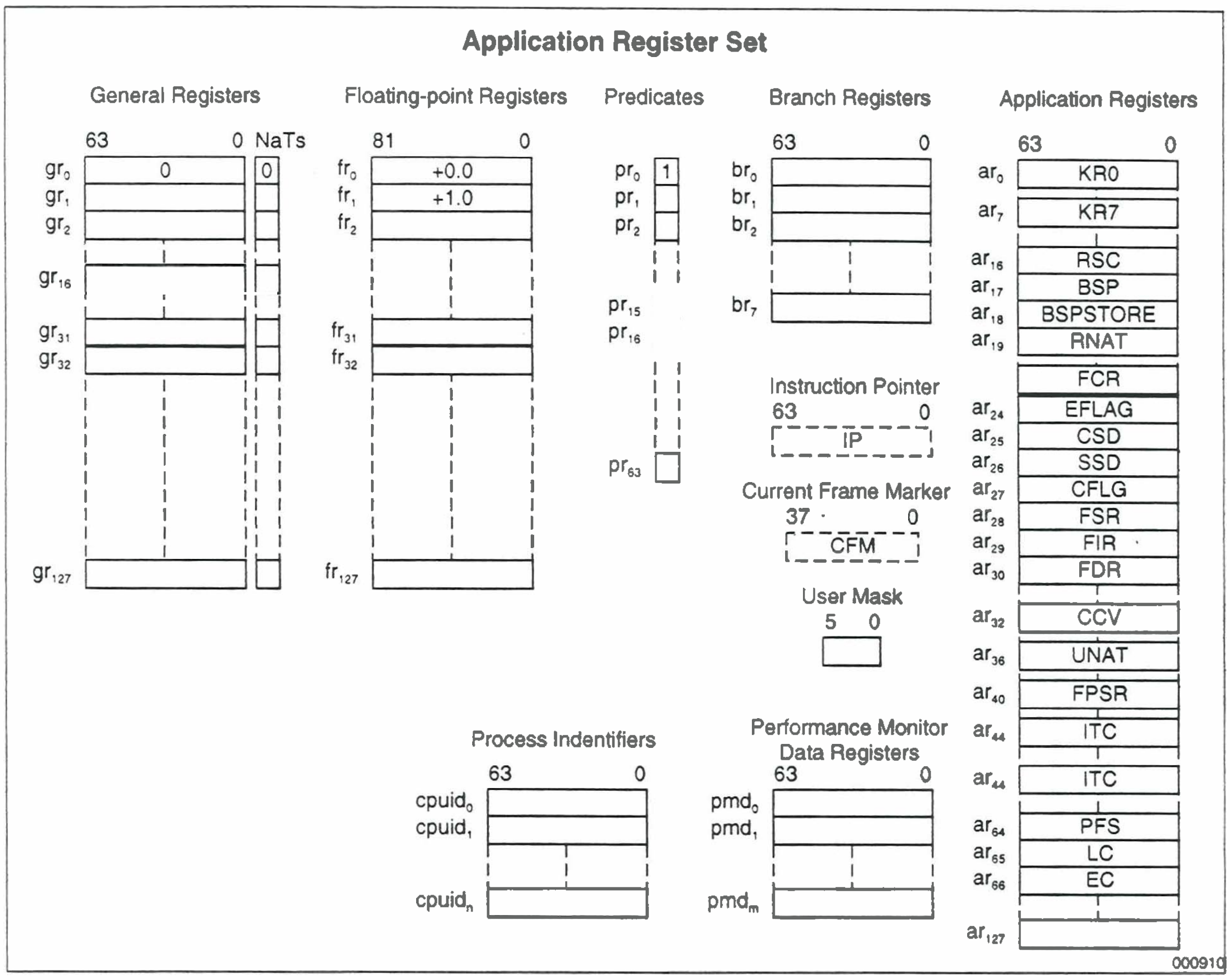
July 2000 Itanium's introduction pushed back to the fourth quarter of 2000. Introduction of second-generation 64-bit chip, code-named McKinley, pushed back to 2002

Aug. 2000 Release of Itanium delayed until early 2001

Feb. 2001 Canceled McKinley presentation fuels rumors of further delays

IA-64

Figure 3-1. Application Register Model



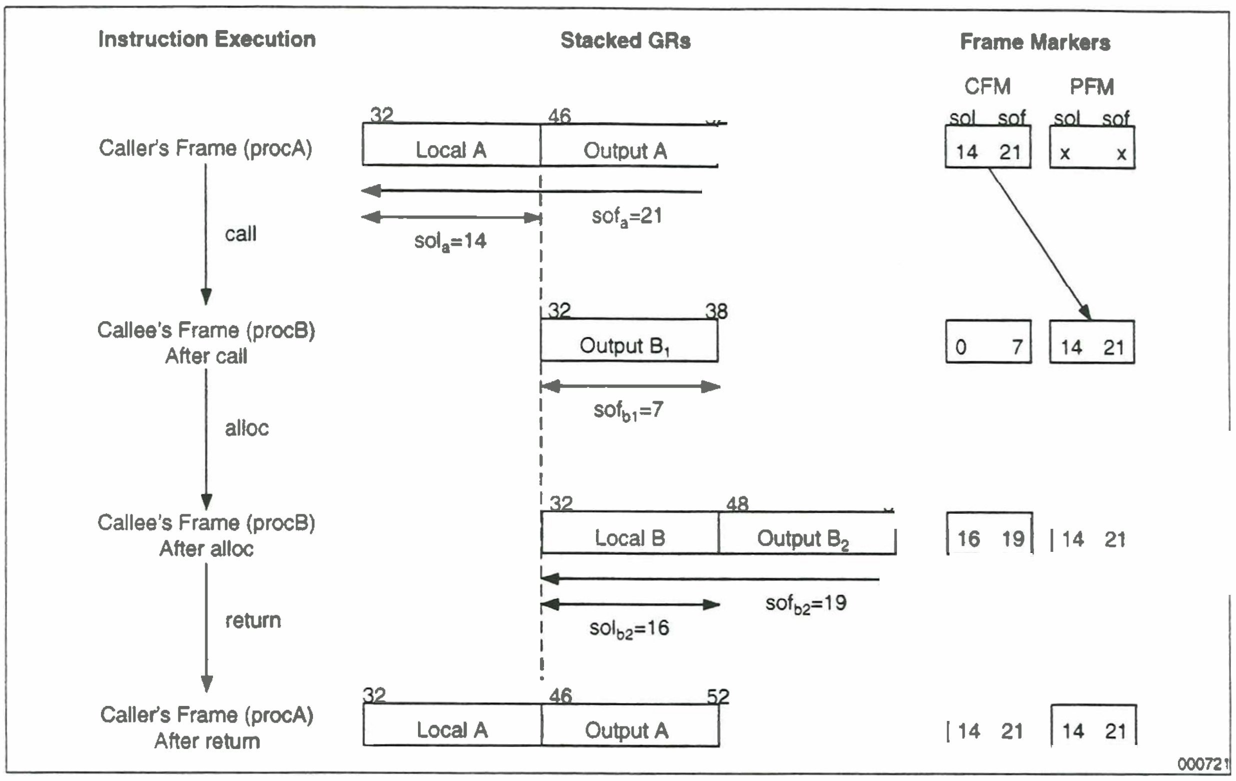
3.1.2 General Registers

A set of 128 (64-bit) **general registers** provide the central resource for all integer and integer multimedia computation. They are numbered GR0 through GR127, and are available to all programs at all privilege levels. Each general register has 64 bits of normal data storage plus an additional bit, the **NaT** bit (Not a Thing), which is used to track deferred speculative exceptions.

The general registers are partitioned into two subsets. General registers 0 through 31 are termed the **static general registers.** Of these, GR0 is special in that it always reads as zero when sourced as an operand, and attempting to write to GR 0 causes an Illegal Operation fault. General registers 32 through 127 are termed the **stacked general registers.** The stacked registers are made available to a program by allocating a register stack frame consisting of a programmable number of local and output registers. See "Register Stack" on page 4-1 for a description. A portion of the stacked registers can be programmatically renamed to accelerate loops. See "Modulo-scheduled Loop Support" on page 4-28.

L3

Figure 4-1. Register Stack Behavior on Procedure Call and Return



4.1.2 Register Stack Instructions

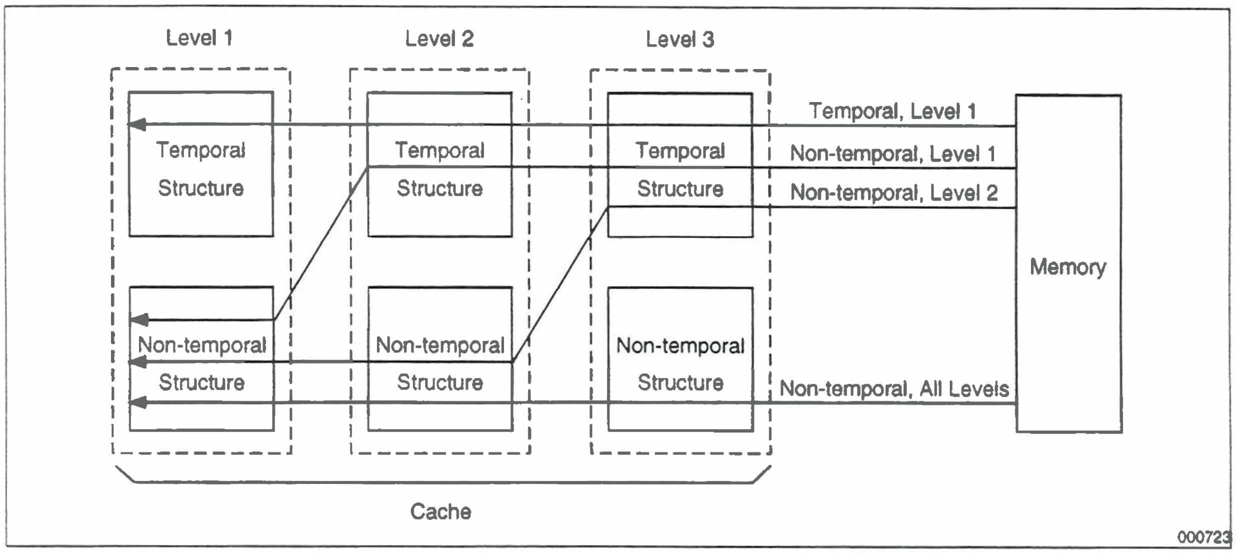
The `alloc` instruction is used to change the size of the current register stack frame. An `alloc` instruction must be the first instruction in an instruction group otherwise the results are undefined. An `alloc` instruction affects the register stack frame seen by all instructions in an instruction group, including the `alloc` itself. An `alloc` cannot be predicated. An `alloc` does not affect the values or NaT bits of the allocated registers. When a register stack frame is expanded, newly allocated registers may have their NaT bit set.

In addition, there are three instructions which provide explicit control over the state of the register stack. These instructions are used in thread and context switching which necessitate a corresponding switch of the backing store for the register stack. See Chapter 6 for details on explicit management of the RSE.

The `flushrs` instruction is used to force all previous stack frames out to backing store memory. It stalls instruction execution until all active frames in the physical register stack up to, but not including the current frame are spilled to the backing store by the RSE. A `flushrs` instruction must be the first instruction in an instruction group; otherwise, the results are undefined. A `flushrs` cannot be predicated.

The `cover` instruction creates a new frame of zero size (sof = sol = 0). The new frame is created above (not overlapping) the present frame. Both the local and output areas of the previous stack frame are automatically saved. A `cover` instruction must be the last instruction in an instruction group otherwise an Illegal Operation fault is taken. A `cover` cannot be predicated.

Figure 4-5. Allocation Paths Supported in the Memory Hierarchy



Another form of hint that can be provided on loads is the `ld.bias` load type. This is a hint to the implementation to acquire exclusive ownership of the line containing the addressed data. The bias hint does not affect program functionality and may be ignored by the implementation.

Two instructions are defined for flush control: flush cache (`fc`) and flush write buffers (`fwb`). The `fc` instruction invalidates the cache line in all levels of the memory hierarchy above memory. If the cache line is not consistent with memory, then it is copied into memory before invalidation. The `fwb` instruction provides a hint to flush all pending buffered writes to memory (no indication of completion occurs).

Table 4-19 summarizes the memory hierarchy control instructions and hint mechanisms.

Table 4-19. Memory Hierarchy Control Instructions and Hint Mechanisms

Mnemonic	Operation
`.nt1` and `.nta` completer on loads	Load usage hints
`.nta` completer on stores	Store usage hints
prefetch line at post-increment address on loads and stores	Prefetch hint
`lfetch`, `lfetch.fault` with `.nt1`, `.nt2`, and `.nta` hints	Prefetch line
`fc`	Flush cache
`fwb`	Flush write buffers

4.4.6.2 Memory Consistency

IA-64 instruction accesses made by a processor are not coherent with respect to instruction and/or data accesses made by any other processor, nor are instruction accesses made by a processor coherent with respect to data accesses made by that same processor. Therefore, hardware is not required to keep a processor's instruction caches consistent with respect to any processor's data caches, including that processor's own data caches; nor is hardware required to keep a processor's instruction caches consistent with respect to any other processor's instruction caches. Data accesses