Cahners
MICROPROCESSOR
www.MPRonline.com
THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

HP, Intel Complete IA-64 Rollout

Virtual Memory, Interrupts More Conventional Than ISA

By Keith Diefendorff {4/10/00-01}

HP and Intel have finally finished the long-drawn-out task of revealing the IA-64 architecture, a process that began more than two years ago at the 1997 Microprocessor Forum. Speaking at Intel's spring developer forum (IDF), IA-64-architects Jerry Huck and Rumi Zahir laid out the details of the IA-64 system architecture, including descriptions of its virtual-memory mechanisms, memory protection, multiprocessor coherence, and interrupt structure.

Unlike the unconventional approach HP and Intel took with IA-64's VLIW instruction-set architecture (ISA) (see *MPR 5/31/99-01*, "IA-64: A Parallel Instruction Set"), the companies took a far less radical tack on system architecture. Borrowing liberally from existing RISC processors, especially from PA-RISC and PowerPC, IA-64 processors will require little redesign of operating-system internals.

Although the vendors released system-architecture details to selected IA-64 software developers under nondisclosure agreements some time ago, the recent IDF disclosure allows any company to create any level of software, right down to and including operating-system kernels and platform BIOSs. With this step, HP and Intel have now completed paving the way for the first Itanium processors to enter the market, an event we expect to occur during the second half of this year. Itanium (née Merced) silicon has been in the hands of anointed software developers since late last year, and at the International Solid-State Circuits Conference (ISSCC) Intel revealed that the silicon will be introduced at a speed of 800MHz (see MPR 2/28/00-msb, "Itanium Meets 800MHz Goal"), as we had previously anticipated.

VM Supports MAS and SAS

Operating systems are generally built on one of two virtual-memory models: the multiple-address space (MAS) model—implemented by Windows NT, Linux, BSD, VMS, and Mach-based versions of Unix—or the single-address-space (SAS) model used in proprietary versions of Unix, such as HP-UX and IBM's AIX. (The SAS model is also referred to as the global-address-space model.) Historically, processors have supported one model or the other, making them more or less adaptable to a given OS.

MAS operating systems typically use unique process identifiers (PIDs) and simple multilevel-indexed page-table

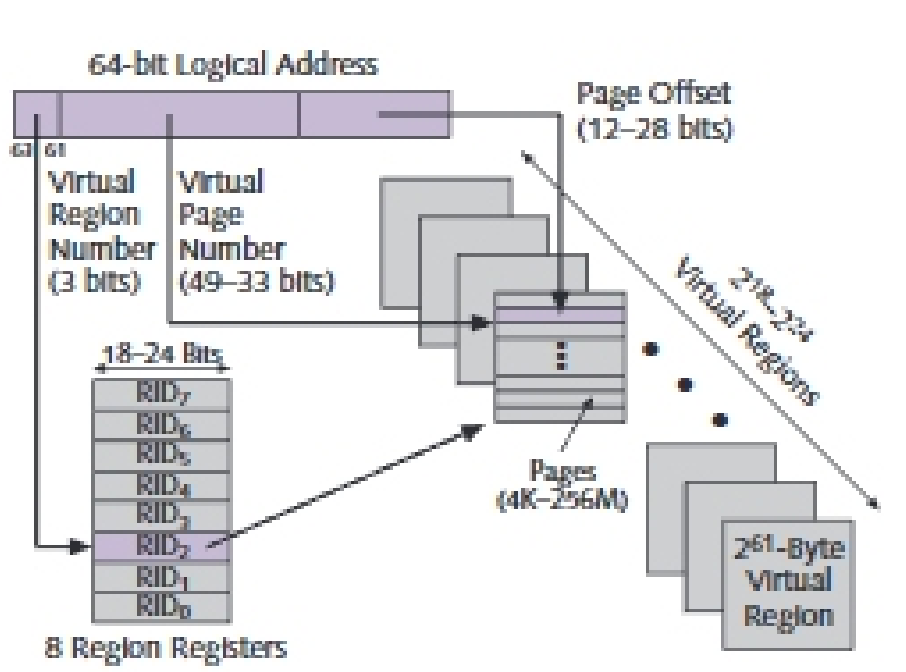

Figure 1. An IA-64 process is mapped onto a 2^{85}-byte virtual address space through eight region registers. This mapping supports either single-address-space or multiple-address-space views of virtual memory.

structures that isolate processes into separate address spaces. SAS-based OSs, in contrast, allocate all tasks into a single, very large virtual-address space, typically mapped by an inverted or hashed page table. The choice of model is largely one of OS-designer preference and of the target market for the OS. The MAS model tends to be simpler and somewhat more efficient for small systems, while the SAS model tends to be favored for large multiprocessor systems, because it makes sharing data among tasks more straightforward.

Because the processor hardware that is required to support each model is somewhat different, porting a MAS-based OS to a SAS-flavored processor, or vice versa, requires the OS to go through a number of contortions to force-fit its natural data structures to the dissimilar hardware. Reworking the OS to conform to the opposite flavor of processor is generally impractical, and the contortions can introduce a significant performance burden.

With Microsoft (and Windows NT) on one side and its partner HP (and HP-UX) on the other, Intel—in its inimitable fashion—took the obvious way out: it implemented both models. This schizophrenic approach, however, is probably justified in this case. Taking this approach makes IA-64 processors OS agnostic—which fits nicely into Intel's plan of conquering the entire world with IA-64. And the hardware cost for supporting both address-space models is not large; while the MAS and SAS models are different in critical respects, the actual hardware difference is small. As a result, support for both models is unlikely to have any substantive adverse effects on cycle time or die area. Besides, the burden IA-64 bears of supporting the IA-32 (x86) memory-mapping model probably swamps the incremental cost associated with dual MAS/SAS hardware.

Processes See Flat 2^{64}-Byte Space

As on other 64-bit RISC systems, processes that run on IA-64 systems see a simple, flat 2^{64}-byte (≈2 x 10^{19}-byte) logical-address space. Thus, IA-64 processes perform arithmetic on full 64-bit pointers and can reach any addresses within the enormous 2^{64}-byte space with a single load or store—without the overhead of manipulating segment registers. This logical-address space is over 4 billion times as large as that accessible by today's IA-32-based (x86) processes.

Although such an enormous address space is massive overkill for today's PC applications, addressability beyond the 2^{32}-byte (≈4 x 10^{9}-byte) limit of x86 processors is already important for large database applications. In fact, 64-bit addressing may soon become the ante to play in the large-server market that HP and Intel covet for initial IA-64 processors. It may eventually become important in other application areas as well; it does enable certain classes of algorithms that require a large address space but populate it only sparsely with data.

To support both the SAS and MAS models, the IA-64 system architecture defines virtual regions. Each processor implements a minimum of 2^{18} virtual regions and a maximum of 2^{24}, each 2^{61} bytes in size. Each process can have up to eight regions, which, as Figure 1 shows, are selected by the three most-significant bits (called the virtual-region number, or VRN) of a logical address.

The OS maps a process's eight regions onto the system's 2^{18} (or 2^{24}) possible regions through region identifiers (RIDs), which are loaded into eight hardware region registers (RRs) before the process executes. These registers are part of the process context and are saved and restored across process (context) switches. For SAS systems, RIDs can be interpreted as the most-significant address bits of a 2^{79}- (minimum) or 2^{85}- (maximum) byte global-virtual-address space. For MAS systems, RIDs are treated as address-space identifiers. Sharing between processes is facilitated by mapping shared regions into the RRs of multiple processes, as Figure 2 shows.

IA-64 supports 32-bit virtual addressing by three means: zero extension to 64 bits (used for all IA-32 accesses), sign extension to 64 bits (which requires software to ensure that the upper 32 bits of an address are always the same as bit 31), and pointer swizzling.

In the pointer-swizzling approach, the upper 2 bits of a 32-bit address select one of four virtual regions, and the entire 32-bit address (zero extended to 61 bits) is used as an offset into each of the four accessible regions. Swizzling the address bits in this manner supports either a flat 2^{32}-byte space (with a single RID) or a multiregion space.

The multiregion capability is important to ease the porting of old 32-bit programs onto new 64-bit IA-64 platforms. Using pointer swizzling, the compiler and OS can provide sharing and protection without requiring major surgery on the 32-bit source code to make it 64-bit safe. Sharing such things as dynamically linked libraries (DLLs) is a useful method of conserving precious resources, such as TLB entries.

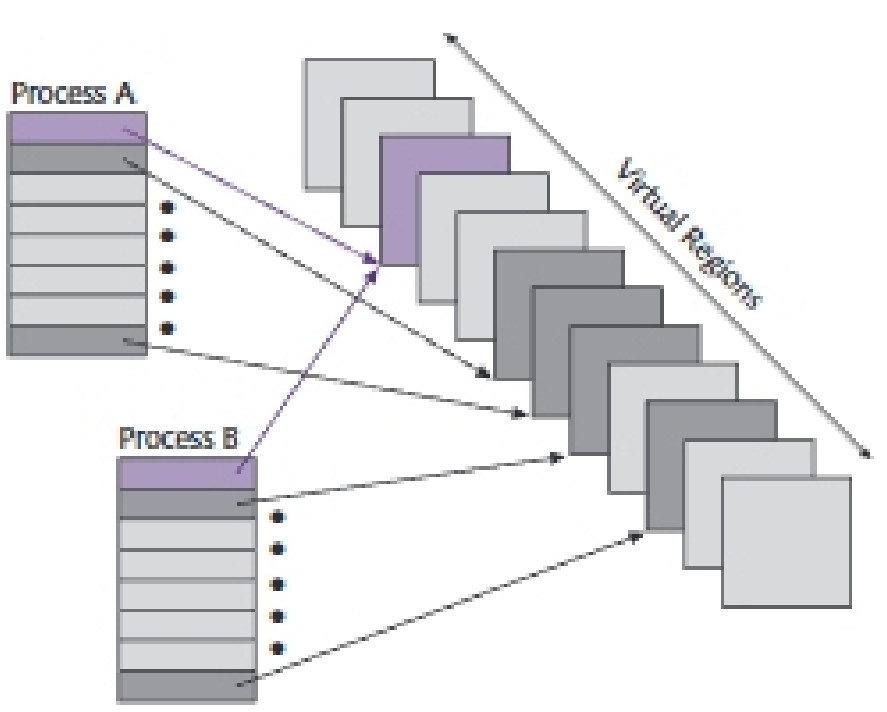

Figure 2. Two IA-64 processes can share data or instructions by having the OS allocate a common region (purple) to both processes.

Mapping to Physical Memory

At any point in time, a system can have many processes ready to run (with virtual memory allocated), although only a few are likely to be active (vying for CPU time), and only one can actually be running (occupying the CPU). To avoid having expensive physical memory (DRAMs) behind every byte of allocated virtual memory, IA-64 systems exploit temporal locality with demand paging, as do most modern systems. IA-64 pages are variable in size from 4K to 256M in 10 power-of-two increments (4K, 8K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, and 256M). The architecture supports up to a 2^{63}-byte physical-address space, but current page-table definitions limit this space to 2^{50} bytes. (The 64th physical address bit is usurped as a cachable/uncachable memory attribute [described later] when virtual addressing is disabled.)

For mapping virtual pages onto physical pages, the IA-64 architecture defines a memory-based data structure called the virtual hashed page table, or VHPT. The VHPT has two possible organizations. The first, a simple per-region linear page table, is directly indexed by the virtual-page number (VPN); it uses a short-form (8-byte) entry. In this format, the per-region page table is part of the same region as the virtual address being translated. This organization will commonly be used by MAS systems. In MAS systems using the short format, the VHPT typically doubles as the OS page-table structure.

For SAS systems, IA-64 offers an alternative page-table organization. This scheme, which is similar to that used by PA-RISC and PowerPC, uses a hash function to index the VHPT (hence the name, virtual hashed page table). Conventional indexed page tables aren't suitable for the SAS model, because their size would be proportional to the size of virtual memory, which is enormous. Hashing, however, compacts the page table by taking advantage of the fact that the virtual-address space is sparsely occupied. This trick makes the page-table size proportional to the amount of implemented physical memory rather than to the much larger virtual-address space. In this form, IA-64 VHPT entries are 32 bytes in size.

VHPT entries are set up and maintained entirely by software. Although the VHPT could be searched by software for each translation, most IA-64 processors (including Itanium) will implement a hardware tablewalker to increase the search speed. This feature avoids draining the pipeline to run an out-of-line software tablewalk routine, and it allows asynchronous concurrent speculative translations.

While hardware tablewalkers are generally faster than software for servicing individual TLB misses, the IA-64 hardware tablewalker is limited to one of the two predefined page-table formats (per-region indexed or hashed). For some operating systems, however, neither of these organizations is optimal, and better overall performance can sometimes be obtained with a different structure, despite the lack of hardware tablewalk support. Therefore, motivated by a strong desire to achieve ubiquity with IA-64, HP and Intel also provided mechanisms to support software tablewalking, allowing IA-64 to use any page-table format. To facilitate software tablewalking, the IA-64 architecture defines fast, vectored TLB-miss interruptions, as well as special instructions for manually loading and purging the TLBs.

TLB Includes Block-Address Translation

Even with a hardware tablewalker, however, performance would be unacceptable if every address translation required a search through a memory-based page table. So, like most modern processors, IA-64 processors implement translation-lookaside buffers to cache recent page translations on chip. The IA-64 architecture provides for separate instruction and data TLBs. Each TLB comprises two translation mechanisms: translation registers (TRs) and a translation cache (TC).

TC entries are similar to the traditional TLB entries found in most microprocessors; TRs are analogous to the block-address translation registers (BATs) of PowerPC. They are designed to efficiently map large, relatively static, areas of memory, thereby improving the utilization of precious TC entries, which usually map smaller, more-dynamic areas of memory. TRs, while they have the same format as TC entries, are managed strictly by software; that is, they are not automatically replaced and filled by the hardware tablewalker. TRs are filled by register number, using the insert-translation-register (itr) instruction, and they are purged according to virtual-address match using the purge-translation-register (ptr) instruction. Similar instructions (itc and ptc) are provided for software managing the TC.

Each IA-64 implementation is expected to have a minimum of eight instruction TRs and eight data TRs that are fully associative. The size and organization of the TCs are left to the discretion of the implementation, but every processor must have at least one data-TC entry and one instruction-TC entry. The architecture supports multilevel TCs, but only the first level is required to support all page sizes.

As Figure 3 indicates, memory accesses initiate an associative search for a matching virtual address across all TLB entries. The search attempts to locate an entry that matches both the virtual-page number (VPN) and the RID; including the region identifier eliminates the need to flush the TLB on a process switch.

On a TLB miss, the hardware tablewalker—if enabled—searches the memory-based VHPT and attempts to load an entry into the TC. In the event the tablewalker finds an empty VHPT entry (a miss) or detects a hash collision (VPN and RID do not match), it defers to software for a more exhaustive search of the operating system's primary page tables or of alternate VHPT collision chains (associativities). Also, if the tablewalker itself misses the TLB it will defer to the software VHPT-miss handler.

Address translations stall the pipeline for as long as it takes to resolve the physical address. In the case of a TLB hit, no stall is introduced; a TLB miss adds tens to hundreds of stall cycles, depending on which level of the memory