To understand the impact of computer architecture on the ability to efficiently virtualize a given architecture,

we discuss some of the findings of the vBlades project at HP-Laboratories [228]. The goal of

the vBlades project was to create a VMM for the Itanium family of IA64 Intel processors,14 capable of

supporting the execution of multiple operating systems in isolated protection domains with security and

privacy enforced by the hardware. The VMM was also expected to support optimal server utilization

and allow comprehensive measurement and monitoring for detailed performance analysis.

The discussion in Section 5.4 shows that to be fully virtualizable, the ISA of a processormust conform

to a set of requirements, but unfortunately the IA64 architecture does not meet these requirements,

and that fact made the vBlades project more challenging. We first review the features of the Itanium

processor that are important for virtualization, starting with the observation that the hardware supports

four privilege rings, PL0, PL1, PL2, and PL3. Privileged instructions can only be executed by the

kernel running at level PL0, whereas applications run at level PL3 and can only execute nonprivileged

instructions; PL2 and PL4 rings are generally not used. The VMMuses ring compression and runs itself

at PL0 and PL1 while forcing a guest OS to run at PL2. A first problem, called privilege leaking, is that

several nonprivileged instructions allow an application to determine the current privilege level (CPL);

thus, a guest OS may not accept to boot or run or may itself attempt to make use of all four privilege

rings.

Itaniumwas selected because of its multiple functional units and multithreading support. The Itanium

processor has 30 functional units: six general-purpose ALUs, two integer units, one shift unit, four data

cache units, six multimedia units, two parallel shift units, one parallel multiply, one population count,

three branch units, two 82-bit floating-point multiply-accumulate units, and two SIMD floating-point

multiply-accumulate units. A 128-bit instruction word contains three instructions; the fetch mechanism

can read up to two instruction words per clock from the L1 cache into the pipeline. Each unit can

execute a particular subset of the instruction set.

The hardware supports 64-bit addressing; it has 32 64-bit general-purpose registers numbered from

R0 to R31 and 96 automatically renumbered registers, R32 through R127, used by procedure calls.When

a procedure is entered, the alloc instruction specifies the registers the procedure can access by setting

the bits of a 7-bit field that controls the register usage. An illegal read operation from such a register out

of range returns a zero value, whereas an illegal write operation to it is trapped as an illegal instruction.

The Itanium processor supports isolation of the address spaces of different processes with eight

privileged region registers. The Processor Abstraction Layer (PAL) firmware allows the caller to set the values in the region register. The VMM intercepts the privileged instruction issued by the guest OS to

its PAL and partitions the set of address spaces among the guest OSs to ensure isolation. Each guest is

limited to 218 address spaces.

The hardware has an IVA register to maintain the address of the interruption vector table. The

entries in this table control both the interrupt delivery and the interrupt state collection. Different types

of interrupts activate different interrupt handlers pointed from this table, provided that the particular

interrupt is not disabled. Each guest OS maintains its own version of this vector table and has its own

IVA register. The hypervisor uses the guest OS IVA register to give control to the guest interrupt handler

when an interrupt occurs.

First, let’s discuss CPU virtualization.When a guest OS attempts to execute a privileged instruction,

the VMM traps and emulates the instruction. For example, when the guest OS uses the rsm psr.i

instruction to turn off delivery of a certain type of interrupt, the VMM does not disable the interrupt

but records the fact that interrupts of that type should not be delivered to the guest OS, and in this case

the interrupt should be masked. There is a slight complication related to the fact that the Itanium does

not have an instruction register (IR) and the VMM has to use state information to determine whether

an instruction is privileged. Another complication is caused by the register stack engine (RSE), which

operates concurrently with the processor and may attempt to accessmemory (load or store) and generate

a page fault. Normally, the problem is solved by setting up a bit indicating that the fault is due to RSE

and, at the same time, the RSE operations are disabled. The handling of this problem by the VMM is

more intricate.

A number of privileged-sensitive instructions behave differently as a function of the privilege level.

The VMM replaces each one of them with a privileged instruction during the dynamic transformation

of the instruction stream. Among the instructions in this category are:

• cover, which saves stack information into a privileged register. The VMM replaces it with a

break.b instruction.

• thash and ttag, which access data from privileged virtual memory control structures and have

two registers as arguments. The VMMtakes advantage of the fact that an illegal read returns a zero

and an illegal write to a register in the range 32 to 127 is trapped and translates these instructions

as:

thash Rx=Ry –> tpa Rx=R(y+64) and ttag Rx=Ry –> tak Rx=R(y+64), where

0   y   64.

• Access to performance data from performance data registers is controlled by a bit in the processor

status register with the PSR.sp instruction.

Memory virtualization is guided by the realization that a VMM should not be involved in most

memory read and write operations to prevent a significant degradation of performance, but at the

same time the VMM should exercise tight control and prevent a guest OS from acting maliciously. The

vBlades VMM does not allow a guest OS to access the memory directly. It inserts an additional layer of

indirection called metaphysical addressing between virtual and real addressing. A guest OS is placed

in metaphysical addressing mode. If the address is virtual, the VMM first checks to see whether the

guest OS is allowed to access that address and, if it is, it provides the regular address translation. If

the address is physical the VMM is not involved. The hardware distinguishes between virtual and real

addresses using bits in the processor status register.