The Itanium processor, part 3b: How does spilling actually work?

devblogs.microsoft.com/oldnewthing/20150730-01

July 30, 2015

Raymond Chen

Answering some interesting questions that arose yesterday. I didn't know the answers either, so I went and read the manual, specifically Volume 2 (IA-64 System Architecture), chapter 6 (IA-64 Register Stack Engine).

Evan asks, Is the spilling to the stack done by the hardware, or does it trap into the OS?"

According to the manual:

> The RSE operates concurrently and asynchronously with respect to instruction execution by taking advantage of unused memory bandwidth to dynamically perform register spill and fill operations.

So yeah, it's done in hardware.

Let's look at our register allocation diagram again. But I'm going to use colors to describe how they are treated by the Register Stack Engine.

static		a		open	g	f	e	d	c	b
		LR	O		LR	LR	LR	LR	LR	LR
••••••		•••••	•••	••••••••••••••	•••••	•••••	••••••	••••••	••••	••••••

The static registers do not participate in register stacking, so I've grayed them out.

The registers belonging to the currently active function are never spilled to backing store. Those are marked red in the diagram above.

The registers with undefined contents are in white. The ones marked "open" are registers that were discarded when a function returned, or registers which have never been used at all. There might also be registers, like those belonging to function *g*, which got flushed out to backing store, then got reused to satisfy some deeply-nested function call, and haven't yet been reloaded from backing store.

The green registers are those that are clean. They contain valid values, and the values have been flushed out to backing store.

The yellow registers are dirty. They contain valid values, and the values have not yet been flushed out to backing store.

We saw last time that when a function call occurs, the registers in the local region are rotated to the end of the diagram, and the other registers slide left to make room. What also happens is that the formerly-red registers are colored yellow, to indicate that they are dirty and can be flushed. The function being called carves out some white and green registers and colors them red. These registers become the new local registers and output registers.

When a function returns, its red registers are colored white, and the previous function's local region is slid into position (assuming the registers are yellow or green) and colored red.

What the Register Stack Engine does is look for unused memory bandwidth and use it to extend the size of the green region. To turn white registers green, it loads them from backing store. To turn yellow registers green, it saves them to backing store.

Turning white registers green means that when you return from a function, the caller's registers will already be on-chip and don't need to be loaded from memory. Turning yellow registers green means that when you call a function, the new function will have enough white and green registers available to satisfy its needs.

If there aren't enough registers of the appropriate color to satisfy the needs of a function call or return, the processor will perform spills or loads immediately in order to make room.

Bob wonders if the large number of registers makes context switching expensive. On a context switch, only the red and yellow registers need to be flushed. Since the processor is working in the background to eliminate yellow registers, the hope is that by the time you perform the context switch, there aren't many red and yellow registers left.

But yeah, context switching is more expensive.

Raymond Chen

Follow