

RETROSPECTIVE:

Implementing Precise Interrupts in Pipelined Processors

James E. Smith

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
jes@ece.wisc.edu

I became interested, reluctantly at first, in precise interrupts while part of the Cyber 180/990 project at Control Data in 1979. CDC had implemented imprecise interrupts in the 6600 and 7600. Precise interrupts could complicate an otherwise clean design, and weren't the kind of thing designers liked to think about. But the Cyber 180 was a virtual memory architecture, and the specification called for precise interrupts.

The earlier CDC STAR-100 had also been a virtual memory machine, and its schedule had suffered a substantial delay when recoverable interrupts had to be added to the design late in the day. The solution used in the STAR-100 and later Cyber 200 follow-ons was to implement an "invisible exchange package" — basically a dump of the pipeline contents that could be later restored.

With the STAR-100 lesson clearly in mind, my manager Jim Stockard asked me to assume responsibility for pulling together an overall precise interrupt strategy. Some pieces had been implemented, but there was no cohesive strategy. I believe Jim thought there was some connection between my fault-tolerant background and precise interrupts; I couldn't convince him otherwise. The onerousness of the task must have impressed me, because to this day I can clearly recall that specific meeting with Jim. Working on things like branch prediction seemed like a lot more fun.

Before I joined the project, some of the designers — including Denny Longnecker, John Pearson, and Terry Lyon, probably others — had worked out a method of restoring register state with a history buffer mechanism.

Mechanisms for handling memory stores and "multi-micrand" instructions remained to be worked out. "Micrands" were similar to RISC operations, and some of the complex Cyber 180 instructions required many micrands for execution. I started with the history buffer and added a method for handling multi-micrands by adding markers that delineated groups of micrands

belonging to the same instruction. Stores were handled by signaling the store unit whenever all instructions preceding the store were known to be error-free; this was done via a simple reorder buffer-like mechanism.

Also at that time, I puzzled a little about other ways precise interrupts could be implemented, and decided that an alternative would be to replace the history file with a reorder buffer plus bypasses. But the history buffer method was in place, and the large number of bypasses seemed to be a problem.

I carried the precise interrupt problem back to Wisconsin and sat on it awhile — one of those projects worth doing at some point in the future. After Andy Pleszkun joined the faculty at Wisconsin, we started talking about the problem and started a joint research effort. At CDC, no serious study of the performance impact of precise interrupts was done — we felt it was small and were happy just to have something that worked. So, as part of our project, Andy and I measured performance impact of implementing precise interrupts (which turned out to be small).

Somewhere along the line, Andy and I came up with the future file method — it seemed to be the dual of the history buffer. Exactly how we came up with that one I don't recall. It seemed to round out the paper, though. Andy did all the performance work, and we spent many hours talking about the problem and the alternative solutions.

In retrospect, although neither of us thought it very significant at the time, the reorder buffer has probably turned out to be the main contribution of the paper. My colleague Guri Sohi made it really fly with the observation that the reorder buffer could be used for a lot more than supporting precise interrupts; it could also be used to support renaming and speculative execution. Sohi's method is described in another paper in this proceedings.