

Method for implementing precise exceptions

M. K. Gschwind

Precise exceptions are an important part of the specification in many architectures. (This is true for both real architectures and virtual machines such as the Java Virtual Machine) The present invention describes how to achieve correct implementation of such architectures during binary translation when speculation and instruction scheduling are to be used to achieve high performance.

The present invention describes a code generation technique and runtime environment to be used for implementing precise exceptions while preserving scheduling freedom. The present invention has multiple applications, such as in the binary translation of CISC architectures or in scheduling of unsafe operations.

Binary translation of CISC architectures

When using system-level binary translation in CISC architectures, correct exception and trap points must be recognized at the CISC instruction boundary even when a CISC instruction has been cracked into multiple execution primitives. When using an incremental commit strategy, atomic instruction execution can be achieved by establishing whether an instruction will succeed before actually modifying any architected processor state.

An example is the translation of the load multiple instruction LM. Consider the translation LM RG=(0,7), 0x10(8) into RISC-like execution primitives:

```
LWZ  R0, 0x10(R8)
LWZ  R1, 0x14(R8)
LWZ  R2, 0x18(R8)
LWZ  R3, 0x1C(R8)
LWZ  R4, 0x20(R8) (*)
LWZ  R5, 0x24(R8)
LWZ  R6, 0x28(R8)
LWZ  R7, 0x2C(R8)
```

When the memory area referenced by the load sequence spans a page boundary (e.g., at address 0x20(R8) as indicated by the asterisk), and the second page is not program accessible, taking an exception at that point would violate the atomicity requirement, since the instruction cannot be completed successfully, but part of the architected processor state has already been obliterated by the partial execution of the LM primitives. This is usually resolved by probing the high end of the addressable data range using a victim "probe" instruction, e.g.:

```
LWZ RTMP, 0x1C(R8) (+)
LWZ R0, 0x10(R8)
LWZ R1, 0x14(R8)
LWZ R2, 0x18(R8)
LWZ R3, 0x1C(R8)
LWZ R4, 0x20(R8) (*)
LWZ R5, 0x24(R8)
LWZ R6, 0x28(R8)
LWZ R7, 0x2C(R8)
```

In the case where an LM sequence spans a page boundary at the location indicated by the asterisk, an exception would be raised by the first victim load instruction into an unarchitected resource marked by +. (For the store multiple SM instruction, a special probe-store instruction needs to be provided which tests for write permissions but does not actually change the target location.)

While pre-probing can be used to establish atomic behavior for exceptions due to easy to predict and test conditions such as page boundary crossings, it is not sufficient to establish atomic exceptions in the presence of hardware support for a PER-mechanism (which allows to establish data breakpoints on the change to a memory location), or similar breakpointing mechanisms, implemented by the hardware execution engine. (A software PER implementation where breakpoint checks are compiled into the binary-translation generated code exhibits none of these problems.)

Consider a PER breakpoint set to the address specified by the * reference. The victim probe instruction would succeed without any problem, but an exception would be raised for the load into R4. The resulting exception is not raised at an atomic CISC instruction boundary, if it is raised immediately.

According to the present invention, this is solved by implementing a roll-forward strategy in the exception handler. Thus, when control transfers to the native VLIW exception handler, the native VLIW handler detects that a PER exception was raised at a non-atomic instruction location, and starts to interpret primitive instructions from the binary-translated code until the boundary of the instruction which experienced the PER exception is recognized. At this point, control is reported for the emulated architecture at the point after the LM instruction as specified by the architecture.

In a long instruction word architecture (VLIW), the last primitives from such complex operations may well be interspersed with primitives from other CISC operations in a VLIW. Then, a mechanism can be provided to detect only the primitives which were derived from the LM instruction, so only they are executed. This mechanism can consist of a bitmap identifying the primitives belonging to the instruction spanning a VLIW boundary, or by imposing some ordering (e.g., all VLIW primitives belonging to the CISC instruction which spans a VLIW boundary must precede all other execution primitives in that VLIW).

Scheduling and speculation of unsafe operations

The present invention can also be used to achieve exact interruption points in the presence of unsafe scheduling and/or speculation in an architecture without hardware support for deferring exceptions due to such speculation. Consider the following (original) PowerPC code sequence:

```
1      add r4,r3,r4
2      lwz r0,r0(r5)
3      add r3,r0,r4
4      blr
```

The code could be improved by scheduling the longer latency lwz instruction (2) before add instruction (1). However, depending on the exception model, this schedule may be unsafe because the add instruction (1) may modify state which must be in-order with respect to any exception which may be generated by the load operation.

Unsafe scheduling may be used in conjunction with the present invention to generate the following schedule by moving the long-latency lwz instruction (2) across the add instruction (1). In addition, the original in-order position of instruction (2) is marked (e.g., in a static table):

```
2'     lwz r0,r0(r5) (+)
1      add r4,r3,r4
2      * (in-order position of lwz instruction)
3      add r3,r0,r4
4      blr
```

When instruction 2' (marked with '+') raises an exception, the native exception handler is invoked in accordance with the present invention. The exception handler then tests whether the instruction having caused the exception was an in-order or out-of-order instruction. If the instruction was an in-order, the program's exception handler is invoked immediately. Otherwise, interpretation starts at the instruction until the in-order point of the instruction is encountered (marked with '*' in the code sample). When the in-order point has been encountered, control is transferred at the correct in-order point, thereby generating an exact exception.

The present invention can be implemented either entirely in software, or in hardware, or in any combination of hardware and software. In one hardware embodiment, instructions may contain a flag indicating regions of VLIW instructions which are to be executed atomically. Exception information is then retained and an exception will be invoked when control exits the exception-disabled region.