Technical Note PT39
The DR Emulator

CONTENTS

This Technical Note discusses the Dynamic Recompiling Emulator that will be available in the next generation of Power Macintosh CPUs.

[Feb 01 1995]

Introduction - What is the "DR Emulator?"

"DR Emulator" is short for *dynamic recompiling emulator* . It is an addition to the 680x0 emulator shipped with the original PowerMacs. It makes use of an emulation technology known as *code translation* or *dynamic recompilation* . Emulation performance can be greatly increased by using this technique, and as long as a few old rules are followed by developers, application compatibility will remain high.

Back to top

The Theory Behind Code Translation

An interpretive emulator (like the emulator currently shipped in the first line of PowerPC based Macintoshes) performs three main tasks in the emulation of a single 680x0 instruction. First, it must fetch the instruction from the 680x0 instruction stream. Second, it must decode the instruction and dispatch to a small semantic routine. Third, the semantic routine is responsible for carrying out the action of the original emulated instruction. An emulator which uses code translation attempts to short-cut two of these three steps. When a piece of code is first encountered, the code translator analyzes the 680x0 code it is about to emulate. This translation mechanism then generates the corresponding PowerPC code into a *translation cache.* After translation, the 680x0 code no longer needs to be fetched or decoded. The emulator can simply re-execute the cached PowerPC code and avoid the additional overhead.

Performance Benefits of Code Translation

Code translation will benefit emulator performance for "typical" 680x0 code. Speed increases will naturally vary for different pieces of code. Programs which contain many small loops will benefit most. For example, Speedometer benchmarks are typically two to four times faster when code translation is enabled. Performance increases for typical application or OS code will probably not be this substantial.

Back to top

Translation Cache & Cache Coherency

The translation cache is a block of RAM which holds translated code as well as the data structures which allow the DR Emulator to track previously translated code blocks. In some ways, the translation cache acts like an instruction cache on the 68040 processor. Like a hardware cache, performance will generally increase as the size of the translation cache is increased. The translation cache is also susceptible to the same compatibility pitfalls as the 68040. When 680x0 instructions are written to memory, either by system software such as the Segment Loader or BlockMove or through self-modifying code, the DR Emulator must be notified of the changes. For this reason, any code responsible for writing or copying 680x0 code must follow the same rules as required on the '040. In particular, code must call the FlushInstructionCache, FlushDataCache, or FlushCodeCacheRange routines whenever modifying code in memory. The DR Emulator also honors cache flushing instructions such as CPUSH, CINV, and MOVE to CACR. Failure to notify the DR Emulator of code changes will likely result in an incoherent cache and eventually to crashes.

Uses of the above calls should be kept to an absolute minimum. Wherever necessary, attempts should be made to use FlushCodeCacheRange. All other notification methods do not specify an address range and therefore require the DR Emulator to flush the entire translation cache. This operation is both time consuming and wasteful. Code which calls any of the above cache flushing mechanisms in a tight loop may perform very poorly under the DR Emulator.

As documented, calls to BlockMove will continue to provide cache coherency for blocks above a certain minimum size (12 bytes). Any developer wishing to use BlockMove for anything other than moving 680x0 instructions should use BlockMoveData instead to avoid the cache flushing overhead. (Note that BlockMove does not provide cache coherency for PowerPC instructions. Software which copies PowerPC code needs to explicity call MakeDataExecutable to guarantee cache coherency on split-cache processors such as the 603 and 604.)

Due to the design of the Mixed Mode Manager, many developers have released software which statically allocates routine descriptors. These data structures, central to the Mixed Mode mechanism, begin with a 680x0 A-Trap ($AAFE). Because developers have created these routine descriptors on the fly without flushing the 680x0 cache, the DR Emulator has a special check for this instruction. This guarantees that existing software works, and also allows developers to continue to allocate routine descriptors without worrying about cache flushing.

Interrupts

The interpretive emulator checks for pending external interrupts on each 680x0 instruction boundary. Because the DR Emulator translates entire blocks of code at a time, translated code only checks for interrupts at the end of each block. This means that the average interrupt latency may be slightly longer under the DR Emulator. However, the difference should be negligible.

Debugging & Trace Mode

In addition to interrupts, the interpretive emulator also detects trace exceptions at 680x0 instruction boundaries. Providing trace mode support within translated code is difficult, so the interpretive emulator is used to handle all execution while running under trace mode. (Note that trace mode is used by MacsBug and other machine-level debuggers to set break points in ROM.)

When a bus error occurs on a real 680x0 machine, the bus error exception frame contains the PC of the instruction which caused the fault. On the original interpretive emulator, the reported PC was not guaranteed to be on the exact instruction, but somewhere within the instruction that caused the fault. On the DR Emulator, the PC may point to a location before the faulting instruction. This may make debugging slightly more challenging when dynamic recompilation is active.

Back to top

Code With Timing Dependencies

Developers have been warned not to rely on timing of processor instructions, as exact instruction timings may not be consistent across processors or emulators. However, various pieces of software still use "DBRA loops" as a basic timing mechanism. Code translation introduces timing anomalies. The first time a piece of code is executed, the code translation process requires extra time. Subsequent executions are relatively fast.

The DR Emulator attempts to detect and deal with timing-dependent code. Whenever the 680x0 interrupt mask is raised to a non-zero level, code translation is temporarily suspended until the mask drops to zero. This means that any timing-dependent code running at interrupt time or running at a time when the interrupt mask has been raised will run with the interpretive emulator with relatively consistent timing.

Back to top

References

MC68040 32--Bit Microprocessor User's Manual

Technical Note HW 06 - "Cache As Cache Can"

New Inside Macintosh: PowerPC System Software

Back to top

Downloadables

Acrobat version of this Note (K). Download
