Technical Note PT38

PowerPC Compatibility and Performance Issues

CONTENTS

This Technical Note discusses differences between the PowerPC 601 chip and future 603 and 604 chips, and how these differences affect application compatibility and performance.

[Sep 01 1994]

Introduction

The PowerPC 601 chip is a transitional CPU, bridging the new PowerPC architecture with the old POWER architecture from which it is descended. As such, it implements most of the old POWER instruction set, as well as the PowerPC instruction set. Subsequent PowerPC CPUs, such as the 603 and 604 only implement the PowerPC architecture. Additionally, implementation differences between the 601 and 603/604 chips can also affect performance. This note discusses the implications for compatibility and performance arising from these differences.

> **Note:**
> These issues typically affect developers who are actually generating code for the PowerPC. Most application developers have little control over the code generated as that is the responsibility of the compiler. However, at the time this is being written, few compilers fully address these issues, so all developers should check with their tools developers to determine which versions of tools do (or will) address these issues.

Back to top

POWER Instructions

A variety of instructions which were part of the POWER architecture have been eliminated from the PowerPC architecture, 34 instructions, in all. However, most of these instructions are included in the PowerPC 601 implementation as part of the transition from POWER. This made it easier to bring up POWER code on PowerPC, but they have been eliminated in subsequent implementations, such as the 603 and 604. A list of this instructions can be found in Table B-3 of the *PowerPC 601 RISC Microprocessor User's Manual* ., and for convenience is reproduced below.

Table 1. POWER Instructions Deleted from PowerPC Architecture

abs	absolute	rrib	rotate right and insert bit
clcs	cache line compute size	sle	shift left extended
clf	cache line flush+	sleq	shift left extended with MQ
cli	cache line invalidate+	sliq	shift left immediate with MQ
dclst	data cache line store+	slliq	shift left long immediate with MQ
div	divide	sllq	shift left long with MQ
divs	divide short	slq	shift left with MQ
doz	difference or zero	sraiq	shift right algebraic immediate with MQ
dozi	difference or zero immediate	sraq	shift right algebraic with MQ
lscbx	load string and compare byte indexed	sre	shift right extended
maskg	mask generate	srea	shift right extended algebraic
maskir	mask insert from register	sreq	shift right extended with MQ
mfsrin	move from segment register indirect	sriq	shift right immediate with MQ
mul	multiply	srliq	shift right long immediate with MQ
nabs	negative absolute	srlq	shift right long with MQ
rac	real address compute+	srq	shift right with MQ
rlmi	rotate left then mask insert	scvx	supervisor call, with SA = 0+

+ Instructions *not* implemented in PowerPC 601

Most compilers designed for PowerPC do not emit these instructions, however, some compilers originally designed for POWER code generation may. You should contact your compiler vendor for the latest information.

The IBM xlc C compiler and xlC C++ compilers have an option to suppress POWER code generation, -qarch=ppc. Anyone using these compilers ***must*** use this option to generate code that runs safely on 603/604 CPUs. For developers seeded by Apple with these compilers, this option was part of the recommended cmac stanza in the xlc.cfg file. Developers should verify that this option is in effect for all parts of their code.

The GNU gcc compiler is also a POWER compiler, but currently has no option for PowerPC only code generation. Developers should beware of code generated by this compiler.

Anyone writing PowerPC assembly should also take care not to use these instructions.

The latest Prerelease MPW DumpPEF tool (version 2.0b1 from E.T.O. #15) has an option, -w601, to scan for PowerPC 601 specific instructions. It is one way to test if your code is affected. Be aware, however, that DumpPEF cannot always distinguish between code and data and may flag POWER opcodes that are really data. You should check all warnings from DumpPEF to be sure they are not spurious. Even if the tool finds valid POWER opcodes, there is no guarantee the instructions are part of an executable code path. You should, of course, test your application on 603/604 hardware as soon as it is available.

Back to top

POWER register usage

In addition to the POWER instructions that are only implemented on 601, the 601 has internal registers that are unavailable on subsequent PowerPC processors. These are the multiply-quotient (MQ) and the real-time clock (RTC) registers.

The MQ register is generally accessed using POWER MQ instructions (see table above), and is covered in the previous section.

The RTC register can be useful for timing purposes, but is not accessible from high level languages. It would only be a problem if assembly language code was written to directly access the register.

Back to top

Load/store string and load/store multiple word instructions

A variety of instructions can interfere with instruction pipelining. Most problematic for application code are the multicycle load/store string and load/store multiple word instructions because many compilers make use of them. These instructions are referred to as completion serialized instructions because they cause all prior instructions to complete before they execute. This interferes with performance on more heavily pipelined implementations, such as the 604 where this can cause a 6 cycle delay before instruction execution.

The possible implementation limitations of these instructions were noted in PowerPC 601 documentation but compilers use them anyway for convenience and to reduce code expansion. For example, string instructions are often used to copy contiguous data in memory, such as when assigning a struct to a struct. Load/store multiple word instructions are often used for saving and restoring registers as part of function prolog/epilog code.

Apple is working with compiler developers to establish guidelines for using these instructions appropriately. Developers should check with compiler vendors for the latest information on their tools and their use of these instructions.

Back to top

Cache coherency

PowerPC 601 features a unified instruction/data cache, while 603 and 604 feature separate instruction and data caches. This leads to potential cache coherency problems analogous to those encountered when MC68040 machines were released. Fortunately, the PowerPC runtime architecture reduces this risk as much as possible.

Almost all code for PowerPC is loaded and prepared by the Code Fragment Manager. The CFM ensures that all such code is suitable for execution. If all your code is loaded by the Code Fragment Manager, you don't have to worry about cache coherency.

However, if you generate code in memory for execution, you should be concerned about this issue. This includes compilers that generate code for immediate execution and interpreters that compile an interpreted language into memory for execution.

You can eliminate the cache coherency problem by notifying the system that data is subject to execution. Use the call MakeDataExecutable, defined in OSUtils.h:

```
extern pascal void MakeDataExecutable(void *baseAddress, unsigned long length);
```

This call is currently only implemented for PowerPC, so you must conditionally compile it. It takes an address, which is the start of the data to be flushed and a length, for the amount of data. Be very careful about flushing the cache unnecessarily as you will adversely affect performance.

Back to top

Data alignment

Modern RISC designs generally prefer *natural* alignment for data (for example, shorts only need be 2-byte aligned, but doubles need to be 8-byte aligned.) But 680x0 code typically aligns data on 16-bit boundaries, and PowerPC was explicitly designed to support this kind of data access. The 601 does not suffer much of a performance penalty with most misaligned data accesses. This will no longer be true on later PowerPC CPUs, however. Furthermore, as the PowerPC processor family grows, it is likely that the performance hit for misaligned accesses will grow as well.

While the 603 and 604 designs support misaligned data access, an alignment exception is thrown under some conditions. These exceptions are handled silently by the nanokernel, and you will see no evidence of the exception other than a decrease in performance on 603 and 604 processors. Because these loads and stores are handled by the nanokernel instead of the hardware, a significant performance hit is taken for every misaligned access. As an example, take the following data structure:

```
struct ArrayWithHeader {
   short    BlockHeader;
   double    Elements[kSomeLargeNumber];
};
```

When compiled with 68K alignment, an iteration over the array elements (meaning a data access on halfword boundaries for each `double`) may take up to forty times longer on a 603 or 604 than on a 601. When compiled with PowerPC alignment (which inserts a halfword pad in the data structure, to guarantee that accesses to the array will be on word boundaries), an iteration completes in less time than on a 601.

Note that while this significant performance hit is only present when accessing `floats` or `doubles` on non-word boundaries, optimal performance for iterating over this array takes place when accessing its elements on 8 byte boundaries. On the PowerPC misaligned access of integer types do not cause an alignment exception, but there is still a performance loss when compared to aligned integer accesses. If speed is important, make sure that access to any of your data structure fields fall on natural boundaries, or are (minimally) compiled with PowerPC alignment.

While it is essential that you use 680x0 data alignment for data shared with 680x0 code (such as the Toolbox), you should always use PowerPC alignment for data used internally to your application. In particular **do not turn on global 680x0 data alignment for your PowerPC code**. Use alignment pragmas to turn on 680x0 data alignment only when absolutely necessary.

Back to top

References

PowerPC(TM) 601 RISC Microprocessor User's Manual

PowerPC(TM) 603 RISC Microprocessor User's Manual

The PowerPC(TM) Architecture

Back to top

Downloadables

Acrobat version of this Note (K). Download