**The Importance of PowerPC**

**Apple's strategy for the future**

PowerPC fits well into Apple's philosophy of building ever faster machines at affordable prices, maintaining upward compatibility, and designing different systems for different users. The PowerPC processor supports these goals in two ways:

* improved performance relative to comparable CISC processors
* a carefully defined Virtual Architecture

**Improving system performance**

The PowerPC processor's improved performance comes from:

* the RISC philosophy of keeping the basic design simple and optimizing wherever possible
* the application of advanced semiconductor manufacturing techniques that make for a smaller, faster chip

How do we quantify improved performance? Consider the PowerPC 601 processor. As of mid-1993 the 66 MHz 601 is a standard and readily available implementation of the PowerPC. The least sophisticated of the PowerPC chips, this version of the 601 has an integer SPECmark (SPECint92) rating of around 62 and a floating-point SPECfp92 rating around 72. (Ratings are given for complete computer systems, and so vary with the speed of the disk subsystem, the amount of secondary cache memory, and so on. These figures are for an IBM POWERstation 250 with a 66 MHz clock and no secondary cache.)

A comparable processor is Intel's Pentium, which is a pipelined, superscalar CISC processor. The 66 MHz Pentium boasts approximately the same integer SPECmark rating as a 601, and a somewhat lower floating-point rating.. However, the initial Pentium parts cost nearly three times as much as the 601 and dissipate nearly 50% more power, which impacts overall system cost.

**Maintaining compatibility**

RISC design helps Apple build faster machines while keeping costs down. They also help us maintain compatibility with 68K processor--based machines. The PowerPC processor--based Macintosh computers use a software emulator that allows them to run existing 68K code quickly.

The PowerPC is well suited to emulating other processors' code because its native instruction set is very similar to the microcode used in most CISC processors. A 66MHz 601 emulates the 68LC040 at an apparent clock speed greater than or equal to 25 MHz. (The 68LC040 is a lower cost version of the 68040 without an integral Floating Point Unit or Memory Management Unit.)

Your code will probably run faster than on a 33MHz 68030 or 68040 machine since significant portions of the Macintosh toolbox have been re-written to use "native" PowerPC code. Of course, if you re-compile your code to be native on the PowerPC, performance will improve over running the same code emulated. We'll look at those issues in the chapter entitled "Programming the PowerPC."

**Matching the processor to the problem**

The improved performance, low power consumption, and emulation capabilities of the PowerPC make it an attractive choice for building future Macintosh systems. However, considering the variety of Macintosh systems (and the desire to make PowerPC an industry-wide standard), the chip's basic design has to allow for customization. PowerPC's designers did this by defining a Virtual Architecture for PowerPC.

The PowerPC's Virtual Architecture defines the chip in terms of the apparent programming model and feature set instead of enforcing a specific implementation. Certain instructions may be implemented in the chip's hardware, while other instructions are implemented as small programs in ROM. For example, some versions of the PowerPC chip might not have specialized hardware to support misaligned storage accesses. Performing such an access would cause an exception, and the exception handler could run the appropriate ROM-based routine to complete the operation.

The important thing to remember here is that the programmer need not be aware whether a particular instruction is implemented in the chip's hardware or as some code in ROM. The code will look the same in either case --- only the time taken for a particular instruction will change.

PowerPC implementations might also vary in terms of power consumption. Lower-power versions are well suited for portable, laptop, and embedded systems. Faster, higher power versions are preferred for desktop computers.

Therefore, PowerPC implementations may be tailored for specific uses, from embedded controllers in automobiles to high-end desktop workstations.

**From CISC to RISC**

**The Heart of the Next Generation**

The forthcoming generation of Macintosh systems will be powered by the PowerPC family of RISC microprocessors. Apple’s decision to make this change wasn’t undertaken lightly. This month’s Powering Up will examine the differences between CISC and RISC, take a look at the PowerPC family of microprocessors, and close with a overview of the architecture of the first PowerPC implementation - the PowerPC “601”. This should help explain why Apple is making such a dramatic change to the Macintosh product line.

**A brief history of the (CISC) universe**

The earliest microcomputers were designed to be easy to program in assembly language and were designed to conserve memory, which was expensive and slow to access. (They were also designed according to the limited manufacturing techniques available.) This led to chips that had:

• Very few registers - often only an “accumulator” and one or two general-purpose registers

• “Complex” instructions that allowed assembly language programmers to write programs using a small number of these instructions instead of a large number of simpler instructions (this also conserved memory)

• “Variable length” instructions where the instruction (often 1 byte long) would be followed by the information needed by that instruction

• Multiple styles of accessing memory, known as “addressing modes,” which allowed programmers to access individual locations directly, via a pointer, by an offset to a pointer, by combining pointers, and so on

These processors also executed instructions serially - each instruction had to complete before the next instruction could begin.

As microprocessors evolved, from 8 bits to 16 bits to 32 bits, each new generation added more registers, more addressing modes, and new instructions; some chips even added a limited form of Pipelining - the ability to execute multiple instructions at once. But, the basic design was still oriented towards conserving memory and serving the needs of the assembly-language programmer, often at the expense of speed.

**Enter RISC**

In the early 1980s, several designers noticed that microprocessor design hadn’t kept up with the rest of the system. Memory was faster and much less expensive, assembly-languages had been replaced largely by such “high-level languages” as C and Pascal, and existing designs were pushing the limits of what could be manufactured. So they went back to the drawing boards, and came out with simpler designs that were optimized for speed and for use with high-level languages. These new designs used instruction sets made up of many simple instructions, and thus were dubbed “reduced” instruction set computers.

While the exact meaning of “RISC” is still a subject for debate, most RISC designs include:

• A large number of general purpose registers, and few special-purpose registers

• Instruction sets which are well matched to the needs of compilers, and which contain many “simple” instructions

• Instructions which fit completely in a single “word” (including the data used by the instruction), and which are encoded in an easy to process format.

• A “load/store” architecture, where information has to be loaded into registers before it can be used

• A small number of memory addressing modes, often only one or two, which use a pointer in one of the registers

These features allow most RISC implementations to apply a few simple techniques to get maximum performance:

• Pipelining, so the processor can process multiple instructions simultaneously

• Memory caches, which provide faster access to instructions and data than system RAM or ROM

• Restrictions on data alignment, where the processor requires that all two-byte values be aligned on an even address, all four-byte values be aligned on an even multiple of four, and so on.

The PowerPC is a RISC design which has all of these “common” RISC features, except that it relaxes the rules for memory alignment.

**The PowerPC - An Overview**

The PowerPC architecture is a collaborative effort of Apple, IBM, and Motorola to create a new generation of high performance microprocessors which can used in everything from personal computers, workstations, servers, and multiprocessor systems to embedded microcontrollers.

The PowerPC is based on IBM’s highly successful POWER architecture. The POWER architecture was designed for scientific workstations, and has been optimized for both integer and floating-point math operations. The POWER architecture also incorporates a “branch processor” which attempts to minimize the impact of branch instructions on the processor’s performance.

When the Apple-IBM-Motorola consortium set out to design PowerPC, the members modified the POWER architecture to reduce manufacturing costs and make the design more suitable for desktop computers. They eliminated parts of the POWER instruction set that made the POWER architecture more difficult to implement but had a minimal impact on performance. While the architects were modifying the instructions set for the architecture, they also removed dependencies between instructions, and added features which simplified building multi-processor systems.

The result of the these changes is a low-cost, high-performance RISC architecture with:

• Fixed length, consistently encoded instructions

• A register-to-register (load/store) architecture, with support for aligned data accesses, misaligned data accesses, and both big-endian and little-endian data

• A “simple” instruction set, with instructions which may be tailored to the task at hand (for example, setting the condition codes at the end of an arithmetic operation is an option, not a requirement)

• Simple, yet powerful, addressing modes applied consistently across the instruction set

• A large register set which includes both general-purpose and floating-point registers

• Floating-point as a first-class data type. This means that floating-point is a standard part of the architecture and therefore is better integrated than it is in many other RISC architecturers

Some of these features - notably the mis-aligned data support and the dual big-endian / little-endian support - are unusual in a RISC design, but were required to support past and future Macintosh designs.

**The PowerPC Family of Microprocessors**

The PowerPC family currently has the following four members:

601 - The 601 is a fusion of the POWER architecture and the PowerPC architecture. It is designed to drive mainstream desktop systems. A Macintosh with a 601 will deliver integer performance three to five times that of today’s high-end 68040-based Macintosh systems and floating point performance around ten times that of today’s high-end 68040-based Macintosh systems.

603 - The 603 is the first PowerPC only implementation of the PowerPC architecture. It is designed for low-cost and low-power consumption. The 603 will be used in portable and low-cost desktop Macintosh with PowerPC systems. In many ways, over time the 603 could become Apple’s replacement for the 68030.

604 - The 604 is designed for mainstream desktop personal computers. It will cost about as much as the 601, but will deliver higher performance.

620 - The 620, which is currently still in the design phase, is a high-performance microprocessor that Motorola and IBM believes will be well-suited for very high-end personal computers, workstations, servers, and multiprocessor systems.

**The PowerPC 601 in Context and why Apple likes RISC**

Many developers and customers have been asking how the 601 stacks up against Intel’s state-of-the-art CISC design, the “Pentium.” On a basis of price, performance, and power consumption, the PowerPC 601 compares quite favorably. As you can see from Table 1, the 601 delivers integer performance that matches and floating-point performance that exceeds Pentium’s for about half the cost. In addition it consumes about half the power of Pentium.

Pentium PowerPC 601

Frequency 66 MHz 66 MHz

Die Size 264 mm2 120 mm2

Cache 16K 32K

Power 14 Watts 9 Watts

SPECInt92 64 60

SPECfp92 57 80

Price $950.00 $450.00

This comparison should give you some idea why Apple is staking such a large part of its future on RISC. The PowerPC 601 is the first of its generation (though it does descend from previous RISC architectures), yet matches the performance of the latest CISC chips - and the next PowerPC implementation (603) is well under way. While CISC designers have to work increasingly hard to squeeze more performance out of their designs, at an ever increasing manufacturing cost, RISC designs have considerable room for growth. The evolution of RISC designs has the potential to outstrip the evolution of CISC.

**A Quick Tour of the 601**

Every PowerPC design begins with the fundamental architecture shown in Figure 1, with some chip-specific details. For example, the 601 incorporates single 32K cache which holds both instructions and data, while other PowerPC models are likely to separate the two caches as shown. Also, future implementations may include multiple arithmetic logic units in both the fixed-point and floating-point units, allowing multiple arithmetic operations to proceed simultaneously.

*Figure 1 - A General Diagram of the PowerPC Architecture*

Each of these units has a specific purpose:

• The Branch Unit collects instructions from the Instruction Queue, then locates and removes any branches from the instruction stream before sending instructions to the Fixed-Point and Floating-Point units. Unconditional branches can be removed from the instruction stream, while conditional branches (i.e. part of an “if” statement or a loop) might require the branch unit to “predict” the outcome of the branch. In any case, the branch unit tries to provide an uninterrupted stream of instructions to the units downstream.

• The Fixed-Point unit holds the 32 General-Purpose registers (including one which is used as the Stack Pointer, and another which resembles register A5 in a 68K-based Macintosh.) Each register is one “word” wide, where a word is 32 bits on a 32-bit PowerPC (601/603/604) and 64 bits on the 620 implementation.

The fixed-point unit also holds the Fixed-Point arithmetic unit. This unit implements the standard addition, subtraction, multiplication, and division operations, as well as some comparison, logical, and shift/rotate instructions.

On the 601, the Fixed-Point unit also manages the transfers of data between memory (the Data cache) and the internal registers. This function may be implemented in a separate functional unit on future PowerPC implementations. (Note that even though the Fixed-Point unit manages load and store operations, data cannot be transferred directly between the Fixed-Point and Floating-Point units - the transfer must go through memory.)

The Fixed-Point unit also serves to calculate addresses for use by the Branch Unit.

• The Floating-Point unit holds the 32 floating-point registers and the Floating-Point arithmetic unit. Each register is 64 bits wide (a “double precision” floating-point value), but can hold single-precision (32-bit) values as well.

The Floating-Point unit implements addition, multiplication, and division, combining addition/subtraction and multiplication into a single “multiply and accumulate” unit. This design fits well with most scientific computing needs, where a common operation involves multiplying two values and then adding the result to a running total.

Since the processor contains multiple functional units, each one of which can execute an instruction independently of the others, this is a variety of “multiple-issue” design (where multiple instructions may be executed in a single clock.) Under ideal conditions, the 601 can execute 3 instructions in a single clock - a branch instruction, a floating-point instruction, and a fixed-point instruction.

**Optimizing Code for the PowerPC**

One of the ways that a programmer can take advantage of the design of the PowerPC is by instruction scheduling - arranging instructions so that each functional unit can run without stopping to wait for information or another unit. The PowerPC compilers are designed to use instruction scheduling to create the smallest, quickest applications possible.

For example, a compiler has to implement an “if” statement using at least two operations - performing a test (which sets the appropriate condition codes) followed by a “conditional branch” instruction. Whenever possible, the compiler will schedule some operations to occur between the test and the branch instruction, which gives the branch unit time to forsee the branch, access the condition codes, and predict the outcome of the branch perfectly.

Another example involves loading registers well before they are actually needed, which gives the load operation time to complete (which may require several clock cycles if it has to go to RAM.)

A final example involves allocating “scratch” registers within a function. The Runtime Architecture designates several registers as “volatile”, i.e. not saved across function calls. The compiler can look at a group of functions which are compiled together, and locate which volatile registers are not changed across calls to a particular function, and thereby use that as a scratch register in the calling function.

All of these optimizations require an in-depth knowledge of the processor, and the ability to see the entire structure of a single compiled file. The compiler writers are able to build the myriad rules for instruction scheduling right into the compiler, and the compiler can keep track of the code it generates. Because of this, the compiler often generates better code than an assembly-language programmer will. In fact, Apple suggests that programmers move their entire program into a high-level language (probably portable ANSI C or C++) and only move to assembler those parts which absolutely cannot be expressed in a high-level language.

**Next Month in Powering Up**

The second most frequently asked questions about Macintosh with PowerPC - after, “When can I buy one?” - are “How can I program one” and “What is the average user going to do with that much power?” In next month’s column, we’ll take a look at the development tools for PowerPC and some applications which show off the PowerPC performance to good advantage.

**Further reading**

Space limitations have forced us to give you just the briefest of sketches on the evolution of RISC and the design features of the PowerPC 601. For more information on either of these topics, please consult the PowerPC RISC Microprocessor User’s Manual, published by Motorola (part number MPC601UM/AD) which is included in the Macintosh with PowerPC Starter Kit available from APDA, and the Programmer’s Introduction to RISC and PowerPC CD-ROM available from APDA.