

SPARC™ Version 9: A Robust 64–Bit RISC

Dave Ditzel, Director of Advanced Systems, Sun Microsystems Laboratories, Inc.

The Version 9 SPARC™ Architecture is something you will hear a lot about, it has just been announced by SPARC International. Version 9 (V9) constitutes the first major architectural change made to SPARC since it was first announced five years ago. The following review should provide context to help you understand what V9 is all about.

First, V9 is an architectural specification from SPARC International that extends SPARC. Sometime the specification is confused with a new SPARC chip. V9 is an update to the V8 Architecture. V9 is not a SPARC chip, and it is not from Sun, it is an architectural specification.

When it is published, it will be the open specification of what defines a SPARC, in a book form like the V8 manual, and available in any bookstore worldwide. V9 is not in published form yet, but it is done, and the specification will not change at this point. One of the interesting things is that V9 is really the first major architectural change to SPARC since it was announced nearly five years ago. As a result, many different companies will begin implementing microprocessors and systems around this standard in the future.

Why SPARC™?

One of the reasons SPARC has been successful is that we've had real stability with the architecture. SPARC International member companies have implemented over twelve different SPARC compatible microprocessors since SPARC was first announced, and that's more than any other microprocessor family in history has done with this level of binary compatibility. As a result, SPARC has ended up with more than 5000 compatible applications, and is why SPARC has accumulated more volume. V9 maintains this same stability with upwards binary compatibility, a very important feature. However, we've learned a lot more about RISC in the past few years, and found there are some additional changes we would like, and believe that now is the time for the next

major change for SPARC.

Processor Needs for the 90s

In the early 1980s the idea of RISC began. Developers used what we had learned about the mix between compiler technology and hardware with 1980s level of knowledge, and provided a very minimal system interface for UNIX®. It was just enough to get UNIX® up and running.

The computer industry will need more in the 1990's. The industry has matured and people want growth with compatibility. There is a need we can clearly see in the future—over the next several years—for a larger address space. People want better interfaces to the operating system, as well as support for multiprocessors, lightweight threads, and object oriented programming. More reliability is needed for systems today, than in the past. We've learned more about computer design since the 1980s, and have been working hard to incorporate these new ideas in the design of V9. There are new pipelines that do more than one instruction per cycle and there needs to be support for that. And overall, users want their systems to have good performance growth.

Now if you look at what some of the competition has done—MIPS and Alpha for example, what they've really done is developed a 1980s style of RISC. They've left out a lot of things we think customers are going to need, and that is what SPARC V9 is addressing.

SPARC V9: Making RISC Robust for the Next Century

SPARC V9 is a robust RISC that will last into the next century. One of the main features of V9 is extending SPARC to 64-bit addressing. But V9 also does a lot of things which other architectures simply have not addressed, and are critical. The work on V9 includes performance enhancements, as well as developments to support advanced types of compiler optimization. As a result, we will be able to support today's compilers and the ones we see coming in the next several years.

Changes in V9 will support advanced superscalar pipeline implementations. We've completely redesigned the interface of the system architecture for the style of operating systems we expect in the mid to late 1990s. Fault tolerant features have been added to the basic system architecture, a rather unique feature not found in other microprocessor chips. Support has been added to make extremely fast traps and context switching in the architecture, because there is a belief that it will go with the style of programming we'll be using in the next several years.

Extending Addressing to 64 Bits

V9 supports 64-bit virtual addressing. We've used 64-bit data paths for a long time in SPARC implementations, now the architecture can scale up its addressing range as well. The benefit of this increased addressability is ensuring the long lifetime of the SPARC architecture. We don't want to end up with a 32-bit architecture that can't migrate forward, and that's a lot of what people want to know today. If SPARC can move forward with a bigger address space, then we know our software investment will last for many more years. If not, we might have to think about changing. The good news is that SPARC will indeed extend gracefully.

The way we implement extended addressing in SPARC is by extending all the integer registers to 64 bits. We've added a new condition-code register that is set by 64-bit computations. We've also added several new instructions that explicitly manipulate 64-bit values in the architecture, such as shifts, 64-bit load/store, things of this type.

Now we can still execute today's software programs on a new V9 microprocessor. The way this works is we continue to use the existing instructions which operate on registers. They might, for example, add register one to register two and put the results on register three. The basic instruction set doesn't really know whether the register is 32 or 64-bits long. So if you were to run one of today's programs on a V9 machine, it would get the same results as it does today. However, if you want to take advantage of the extended addressing and future capabilities of

SPARC V9, you can recompile the program and the compiler will know how to take advantage of these additional 64-bit features which are in the architecture and extend the address space and will give you all the additional functionality.

Performance Enhancements

One of the biggest concerns is performance, and we've changed some basic things in the hardware architecture of SPARC that will help performance. There are sixteen more double-precision floating point registers in SPARC, which brings us up to a total of 32. The benefit of registers is to reduce memory traffic; so you don't have to do as many loads and stores, and your program will run faster. There are also eight more quad-precision registers.

We supported a 128-bit floating point format which is also unique for microprocessors. We've added four floating-point condition registers; we've only had one in the past. This would then let you have more parallelism in the basic architecture for a superscalar machine, launching multiple instructions at a time. If you had only one condition code, you could have a serial dependence waiting for that one condition code to finish. This will allow us to launch multiple floating-point instructions simultaneously without slowing down the chip.

We've changed a number of other things in the instruction set area of SPARC purely for performance reasons. We've done things like add 64-bit integer multiply and divide instructions. We've added load and store quadword instructions to load and store 128 bits at a time. We've added branch prediction, because branches are something that are very difficult to implement in computers. By having branch prediction, we can eliminate a lot of the delays associated with branches. We have a new instruction called branch on register value. This reduces the total number of instructions you have to execute, thereby speeding up the program. This also gives the effect of having many integer condition codes, so we get the same advantages of parallelism on the integer side that we got from multiple condition codes on the floating point side.

Support for Advanced Compiled Optimizations

We have added support for advanced compiler optimizations. We are seeing a lot of interesting new optimizing compiler techniques that we'll be able to take advantage of in the 1990s to give us greater levels of performance. Just a couple of examples of things we have changed include instructions that do pre-fetching of data and instructions. The benefit here is to reduce the memory latency so the program isn't waiting for the memory to respond. If you get that response out long enough in advance then the answer is back from memory when you actually need to make use of it. There is support for misaligned data. One example of this benefit is in FORTRAN compilers. Because of the way the language is specified, in many cases compilers cannot analyze whether or not it is legal to use a double precision load, and many architectures do two load singles. Support for misaligned data will allow the compiler to always use the most optimal instruction. In those rare occasions where things aren't aligned properly, the underlying V9 architecture can fix things up. This will translate directly into increased performance. There is support for something called speculative loads which enables the compiler to do better code scheduling of instructions. This is something no other architecture has. We've added support for conditional move instructions. Conditional moves allow you to eliminate branch instructions, and any time you can make branch instructions go away or predict them better, the performance of the machine will go up. And we've added something I thought you might like to hear about called the TICK register. It's just a little timer that counts away once for every lock tick on the machine. It will let programmers make very accurate time and performance measurements.

Support for Advanced Superscalar Processors

V9 includes support for advanced superscalar processor designs. What we're finding is a trend where we are learning to execute more instructions per cycle every year with new pipelines. Two instructions at a time to three instructions at a time. We want to be able to get up to doing eight, nine, ten instructions at a time

with the SPARC architecture. In order to do this, we wanted to make some changes that would make superscalar execution work better. We already have superscalar SPARC chips with SuperSPARC and HyperSPARC. SPARC already has features which are good for superscalar by having simple to decode, fixed-length instructions, and having separate integer and floating point units. V9 supports superscalar processors in several ways. Having more floating point registers available allows computing more things in parallel. Support for speculative loads is very important. Having multiple condition codes allows more things to go on in parallel. Using branch prediction cuts down on branch penalties.

Support for 90's Operating Systems

The operating system interface in V9 has been completely redesigned. There are new privileged registers in the machine; there is a new structure to those registers which makes it much faster to get at important control bits in the machine. One of the interesting things to remember here is that the change in the operating system interface has no effect on user level applications. The user-level programs do not see these particular changes that have been made.

By making these changes we get support for the new microkernel style approach to operating system design, support for lightweight threads and are able to run them much more efficiently, much faster context switching by re-architecting this level, as well as support for object-oriented software. One of the things we have done is to take register windows and make them more flexible than they already were. This lets you do context switching, for example, between different processes and use the register windows as banks of registers to achieve no-overhead context switching.

We've added support for very large-scale multiprocessors. SPARC already supports multiprocessors, but for much larger systems in the future we found new ways to improve performance. One way is by relaxing the constraints on the memory system, using the new Relaxed Memory Ordering model. Individual processors can then synchronize efficiently using

a new Memory Barrier instruction.

Support for Reliability and Fault Tolerance

Something else that microprocessor design has pretty much ignored in the past is explicit support for reliability and fault-tolerance. Now you can build a reliable and fault-tolerant machine without some of the support, but it's a lot more work. So we've done a number of things.

First of all, we've added one very specific instruction called compare and swap. The benefit here is that this particular instruction has some very well-known fault-tolerant features and is also a very efficient way to do multiprocessor synchronization.

We've added multiple levels of stacked traps in V9. Stacked traps allow you to gracefully recover from various kinds of faults as well as allowing you to design your operating system to be much more efficient. If you look at SPARC V8 and other RISCs, they typically have only one trap level. This is okay, you can make things run, but in fact, to do a very nice fault-tolerant machine, having these multiple trap levels will be helpful.

Let me show you an example of what these levels do. There are five different levels. A program would normally execute at Level 0. You can trap into the operating system at level 1, and the operating system can go ahead and not worry too much about causing other traps to happen. If you take another kind of fault, you can trap again to level 2; for example, you can take yet another trap to get to the page fault handler. And, finally, we've added a special new mode in SPARC called RED mode, for Reset, Error and Debug mode. It fully defines the things you need to build a very fault-tolerant system.

Fast Traps and Context Switching

A last technical point is fast traps and context switching. Fast traps turn out to be something that system designers use a lot to make a number of things on the machine run faster. The way we've done this is to re-architect the trap entry code to get you into those instructions in the trap handler very quickly. We've added eight new

registers called "alternate globals" so that when you enter a trap routine, you have a fresh register set to work with, you don't have to worry about storing something away in order to do some computation. The benefit here is that it will give you very fast instruction emulation, and very quick system response time. When you do something like type a character on a keyboard or move the mouse around quite often, this will cause an interrupt in the machine. And so getting into the trap handler and getting back fast will directly effect the user's physical response to how his machine is operating.

We've added multiple levels of stacked traps. It's important that a trap routine itself will allow another trap to happen while you're in it. Otherwise, you'll have to do a tremendous number of checks to make sure that you don't get another problem. For example, in the register window trap handler, you currently need to check while you're in the trap handler that nothing else is going to go wrong, such as you don't take a page fault, or that something else doesn't happen. By adding these multiple levels of traps, we can reduce the number of instructions from one hundred down to about nineteen. That's a big increase in performance. The real benefit here is that it allows you to design your operating system with much better performance and a cleaner programming model.

We've also helped context switching. What we've done here is allow the machine to save or restore fewer registers during a context switch. The way we have done this is in both integer and floating point registers, we can tell more efficiently what registers are actually being used. The floating point registers now have a floating point enable bit that is split into two pieces with a "dirty bit" for the upper and lower part of the registers. So if you haven't written into the registers recently, when you do to do a context switch, there's no need to save them away. So the fastest way to save registers is to not save them at all.

We can also use integer registers in the same way by using register windows as banks of registers. So you could find that in a future

SPARC processor, you're doing a context switch without the need to save any registers away, either on the integer or floating point sides. That makes for very fast context switching.

Summary

So let me summarize some of the features and what we've done overall in V9. We have provided 64-bit addressing; support for fault tolerance; fast context switching; support for advanced compiler optimizations; very efficient design for superscalar processors; and a very clean structure for modern operating systems. And we've done it all with 100% upwards binary compatibility. That is a significant achievement. What I see in the future is good implementations of SPARC V9 chips that will give us superior performance, very robust systems, and very good cost efficiency. That is what our customers are asking for. SPARC has been the RISC leader for the last five years, and with these changes that we are making in SPARC V9, we expect SPARC to remain the RISC leader into the next century.