Comparing the New 64-Bit RISCs

Transition to 64-Bit Addressing Allows Architectural Update

By Brian Case

In the mid 1980s, computer architecture began a transition from CISC to RISC. Since that time, RISC has supplanted CISC only in some markets, and one of the reasons is that RISC processors do not provide any unique capabilities aside from greater price/performance. (Of course, price/performance is very important.) The next computer architecture transition, which is just now upon us, is the expansion of address space from 32 bits to 64 bits. Unlike RISC, 64-bit architectures do offer something unique: a linear virtual address space larger than 4 gigabytes.

At this point, the need for more than 32 bits of address is driven by only a few programs, such as large, interactive data bases, integrated circuit layout, and numerically-intensive time-series analysis. For example, today's high-end microprocessor designs are already too big to simulate in a 32-bit address space. While some observers question the widespread push to 64-bit addressing, applications like those just mentioned are of high strategic importance to business and technical users, and an architecture that lacks a 64-bit address space may lose out in a corporate buying decision even though the vast majority of the corporation's users do not need 64-bit addressing. Also, just as with the transition from 16 bits to 32 bits, the vast majority of uses for the increased address space are not yet obvious.

With any transition in computer architecture comes an opportunity for new players in the market. DEC has used this transition period to make a move from its aging VAX architecture to its new 64-bit Alpha RISC. Vendors of existing architectures are using the transition period to update their instruction sets. Three to six years of experience with optimizing compilers and real applications have given architects some hard data with which to justify a few basic architectural changes.

The architectures that have been updated so far during this transition period are MIPS, SPARC, and POWER. The new 64-bit architectures are called MIPS-III, SPARC V9, and PowerPC. The R3000 implemented the MIPS-I architecture, the ECL R6000 is the sole implementation of MIPS-II, and the R4000 is the first MIPS-III processor. Current SPARC chips are either V7 or V8, and HaL's proprietary processor design is expected to be the first V9 implementation. The first PowerPC chip is the IBM/Motorola 601, but it does not incorporate the 64-bit extensions. The 620, still at least a year away, will be the first 64-bit PowerPC.

It is widely expected that Intel's P7 implementation of the 486 architecture will incorporate 64-bit linear addressing, as well as significant architectural changes. Intel is, of course, not saying anything official about the processor, and it will be a couple of years before it could be on the market in any case.

HP's PA-RISC is not included here because it is not a true 64-bit architecture. While it allows for virtual addresses greater than 32 bits, they are segmented. HP is planning a 64-bit upgrade to PA-RISC, but no details have been released.

In addition to the architectural changes associated with 64-bit linear addressing and improvements to the instruction sets, there are very important language, ABI (application binary interface), and API (application programming interface) issues as well. Once an architecture is capable of 64-bit operations and addressing, compiler writers must decide how to expose the new capabilities in existing languages. For example, in C, what should be the sizes of short, int, and long?

A great many programs make assumptions about the sizes of primitive language types that are rooted in the characteristics of 32-bit architectures. If the transition is not handled carefully, old programs could be difficult to maintain and recompile. For this reason, system vendors are planning to continue to offer existing 32-bit development environments. Other sticky issues, such as how to handle 64-bit vs. 32-bit libraries and system calls, arise as well.

General Architectural Philosophy

SPARC V9, MIPS-III, PowerPC, and Alpha all provide a true 64-bit architecture with 64-bit registers and linear addressing, but each architecture takes a slightly different approach.

MIPS-III, the earliest 64-bit architecture, supplements the 32-bit MIPS-II architecture with 64-bit instructions for memory references, integer arithmetic, and shifts. These and other new instructions are available only when the processor is in 64-bit mode; in 32-bit mode, they cause illegal instruction traps, just as for a MIPS-I or MIPS-II processor. Thus, MIPS-III truly has distinct instruction subsets and modes.

Unlike MIPS-III, Alpha is not an extension of an existing architecture. Since the Alpha architects were aware of the impending need for more than 32 address bits, they simply defined Alpha as a 64-bit-only architecture. There are no mode bits, but there are a few 32-bit arithmetic instructions to simplify code sequences when

compilers and VAX code translators must provide 32-bit operations for backward compatibility.

SPARC V9 and PowerPC are 64-bit extensions to existing 32-bit architectures, and like Alpha, the entire instruction sets are always available—there is no mode bit for two instruction subsets. SPARC V9 and PowerPC do, however, have a bit that forces virtual addresses to be truncated to 32 bits.

SPARC V9 and PowerPC are condition-code architectures, which causes them to take a different approach from MIPS-III. SPARC V9 simply defines an extra set of "64-bit" condition codes to record the outcome of operations across all 64 bits, which allows the same arithmetic instructions to be used for 32-bit and 64-bit operations. For MIPS-III, which has no explicit overflow bit, separate instructions for 32-bit and 64-bit add and subtract that check overflow were needed.

PowerPC does not define a new set of 64-bit condition codes because it actually has eight sets that fill an entire 32-bit special register. Instead, it has a mode bit that determines whether the condition codes reflect 32-bit or 64-bit results. Thus, PowerPC and MIPS-III are similar in that they both have mode bits, but the effect of the mode bit is different on each architecture.

Alpha, designed without major compatibility constraints, is naturally the cleanest 64-bit architecture. Though Alpha is cleaner, MIPS-III, PowerPC, and SPARC V9 are by no means messy or "CISCy." It can be argued that the mode bits make MIPS-III and PowerPC less clean than SPARC V9's modeless instruction set, but this is at most a minor point.

In terms of instruction-set philosophy, MIPS and Alpha are the most alike. Both use compares that generate values in registers instead of condition codes; both have the overflow trap for integer add and subtract specified in the opcode (as opposed to a condition-code bit or an overflow-detect mode bit in the processor status register); and both use load-locked/store-conditional as the primitive for mutual exclusion in operating systems and multiprocessor programs.

True 64-Bit Architectures

The qualifying characteristics of a 64-bit architecture are the ability to easily operate on 64-bit integers and to use all 64 bits of an integer as an address. Generally, all four of these architectures provide similar capabilities for manipulating 64-bit data and linear addresses because they have 64-bit integer registers and use all 64 bits in loads, stores, and indirect branches.

Some early implementations, though, will trap addresses above predetermined limits. For example, the R4000 implements "only" a 40-bit (1024-gigabyte) user virtual address space, and the first Alpha has "only" a 43-bit (8192-gigabyte) space. Addresses with any non-zero bits above these limits will cause run-time traps, which prevents the incompatible use of upper address bits for miscellaneous address tag information. This is a lesson learned from IBM's trouble moving from the 24-bit 360 to the 31-bit 370 and from Apple's transition from 24 to 32 bits in the Macintosh.

PowerPC is unique in that all 64 bits of virtual address will be significant in the first implementations: the MMU will translate the full address width. This may require a little more circuitry in the MMU, but it lets programmers experiment with ways of using the full address space. HaL's SPARC V9 processor is also expected to support the full address width.

Being a 64-bit architecture from its inception, Alpha naturally has a full set of basic arithmetic, logical, and shift instructions for 64-bit operands.

For MIPS, SPARC, and POWER, some instructions work correctly regardless of operand size, while in other cases, new instructions were needed in the 64-bit architectures because backward compatibility required that no changes be made to existing opcodes. Bit-wise logical operations work as expected regardless of register width and so can be used by either 32-bit or 64-bit programs. For right shifts, however, it is necessary to specify the width of the operand in the instruction so that zero extension or sign extension can start at either bit 63 or 31. New, 64-bit shifts are a part of MIPS-III, SPARC V9, and PowerPC.

For add and subtract, SPARC V9 and PowerPC take an approach different from MIPS-III because of their different conditional-branch architectures. Since SPARC has one set of integer condition codes, SPARC V9 simply defines a new set of condition codes for 64-bit operands. The old 32-bit condition codes get set correctly by add and subtract instructions regardless of the doubled register width. When a program needs it, overflow detection for integer arithmetic is done by explicitly testing the appropriate overflow condition code with a separate instruction. Thus, SPARC V9 does not define a new set of add and subtract instructions.

Similarly, PowerPC has condition codes, but the eight sets made it impractical to define a separate set of 64-bit condition codes. Instead, PowerPC has a mode bit to determine whether the condition codes reflect 32-bit or 64-bit results. As with SPARC V9, there is one set of integer arithmetic instructions, and overflow is checked with a separate instruction if required.

MIPS, on the other hand, optionally detects and traps integer overflow as a function of its add and subtract instructions. To differentiate between 32-bit and 64-bit overflow thus requires separate opcodes.

Load/Store Facilities

MIPS-III, SPARC V9, and PowerPC have new instructions to load and store full 64-bit values in a single instruction. SPARC has always had doubleword

Architecture Characteristic	Alpha	MIPS-III	PowerPC	SPARC V9
32/64 mode bit	no	yes	yes	no (but AM bit)
Address bits in current chips	43	40	64*	64*
Byte/halfword load/store	no	yes	yes	yes
Condition codes	no	no	yes	yes
Conditional moves	yes	only in TFP	no	yes
Delayed branches	no	yes	no	yes
Overflow checking	in opcode	in opcode	trap instruction	trap instruction
Multiply support	64-bit result	128-bit result	64-bit	64-bit result
Divide support	no	128-bit result	64-bit	64-bit result
FP double-precision registers	32	16 or 32	32	32
FP precisions	single, double	single, double, quad	single, double	single, double, quad
FP formats	IEEE, some DEC	IEEE	IEEE	IEEE
Prefetch instructions	data	no	data	data, relative branch
Prefetch hints	many	some branch predict	branch predict bit	branch predict bit
Memory models	weak	strong	weak	3 levels
Trap model	imprecise	precise	2 levels	3 levels

Table 1. Comparison of key features in 64-bit architectures (*expectations for first implementation).

loads and stores, but these instructions are defined to operate on a pair of 32-bit registers and thus do not load a full-width value into a 64-bit register. These three architectures add an instruction to load a 32-bit value as a sign-extended 64-bit value; to be compatible with 32-bit semantics, the existing 32-bit load zero-extends to 64 bits in each architecture.

With the new instructions, MIPS-III, SPARC V9, and PowerPC provide a complete set of load-signed, load-unsigned, and store instructions for all important operand sizes: byte, halfword (16 bits), word (32 bits), and doubleword (64 bits).

Alpha, on the other hand, provides instructions for only two operand sizes: 32 and 64 bits. Loads and stores for smaller operands are synthesized with appropriate sequences of load, store, extract, and insert instructions. Thus, a byte load requires a minimum of two instructions, and a byte store requires a minimum of three instructions.

The rationale for this approach is a desire for implementation simplicity and high performance. The multiplexing and shifting hardware required to perform byte and halfword loads and stores is in the logic path between the processor execution units and the first-level caches, which is nearly always the most critical timing path in a processor implementation.

Furthermore, first-level caches that maintain ECC (error-correction) bits and use a write-back policy are an important feature in the large, high-performance systems that DEC typically sells. These caches, however, implement ECC across at least a full 32-bit word; thus, any write smaller than a word to an ECC cache requires the cache to perform a read-modify-write cycle to correctly set the new ECC information. Such sequencing logic can complicate the cache design and slow cache access. Since cache access time has a first-order effect on overall performance, anything that increases access time must either be avoided or win back huge performance benefits in other ways.

What Alpha does, in effect, is force the byte and halfword multiplexing and read-modify-write sequencing to be performed by software in the integer execution units with sequences of Alpha instructions. This can, of course, lead to excessively long instruction sequences, but they are subject to compiler optimization.

Alpha does, however, have both 32-bit and 64-bit loads and stores, which requires an ECC cache to maintain ECC information across 32-bit words. Eliminating the 32-bit loads and stores from the Alpha instruction set would allow the cache to keep ECC information across 64-bit doublewords. This larger ECC word size would have the benefit of requiring fewer total ECC bits for a given cache size, but in practice, 32-bit loads and stores are so frequent in both new and old programs that it makes sense to pay a cost in the cache so that frequent instruction sequences can be shorter.

Integer Arithmetic Instructions

These architectures all have similar support for 32-bit and 64-bit add and subtract. Although there is some convergence in architectural support for multiply and divide, this is still an area of significant difference.

Originally, SPARC had no divide support and multiplication was supported with a single-bit multiply-step instruction. In V8, eight instructions were added—signed and unsigned multiplication and division with and without setting the condition codes. In SPARC V9, these eight instructions, still very new to SPARC, plus the multiply-step are “deprecated,” which means they should not be used in new software (because there are better 64-bit instructions).

To replace the V8 instructions, SPARC V9 defines three new ones: MULX, SDIVX, and UDIVX. None of these modifies any condition codes. MULX can be used for either

signed or unsigned multiplication.

The MIPS architecture has integer multiply and divide instructions predicated on an implementation having an autonomous multiply/divide unit. Unlike other architectures, the result of a multiply or divide is returned in two special registers—HI and LO—instead of general registers. This makes it much simpler for an implementation to return all computed product or quotient/remainder bits without requiring two write ports into the general register file or register scoreboarding.

MIPS-III and Power PC each define four new instructions for 64-bit multiply and divide in signed and unsigned versions. In MIPS-III, these instructions follow the strategy of returning results in HI and LO, which are expanded to 64 bits. Consequently, MIPS-III provides 128-bit results for multiply and divide.

Alpha has good multiply support but no direct divide support at all. Its 32-bit multiply instruction uses only the low 32 bits of its source registers, truncates the result to 32 bits, and sign extends it in the destination register. The 64-bit multiply uses all 64 bits and truncates the result to 64 bits. If needed, the high 64 bits of the full 128-bit product are generated separately with the unsigned-multiply-high (UMULH) instruction.

For division, UMULH can be used to divide a variable by a constant very easily (using multiplication by the reciprocal), but division of two variables must be performed with a subroutine. The strategy of using a subroutine, which uses table lookup and UMULH for small values, probably yields good performance in most cases, but it is not as amenable to hardware performance tuning as a single divide instruction.

In the original 32-bit POWER architecture, there is an MQ register to provide full 64-bit results for multiply and divide. In PowerPC, MQ is eliminated and there are separate instructions that return the low and high 64 bits of multiply and divide results. In this way, PowerPC is similar to Alpha except that PowerPC directly supports division.

MIPS-III and PowerPC provide the most flexible support for multiplication and division (although the MIPS-III HI and LO registers can be a nuisance for aggressive implementations). All except MIPS-III require extra instructions to generate a full 128-bit product or quotient/remainder. SPARC V9 has good divide support but cannot generate the upper 64 bits of a full product as easily as even Alpha. In defense of Alpha and SPARC V9, they have adequate support for the semantics of most languages.

Floating Point

In the early days of RISC architectures, 32 single-precision floating-point registers seemed like a large number. Unfortunately, such a register file can only accommodate 16 double-precision operands, and double-precision floating-point arithmetic has grown in importance (at least in benchmark programs). The difference between 16 and 32 registers is significant for compiler optimization and parameter passing between floating-point subroutines.

As a 64-bit-only architecture, Alpha defines 32 floating-point registers that can hold either single-precision or double-precision operands. As a descendant of POWER, PowerPC also has 32 double-precision registers.

The early versions of MIPS and SPARC both had only 32 single-precision registers that could be paired for double-precision operands. For MIPS-III and SPARC V9, however, the architects found ways to double the number of double-precision registers. MIPS-III has a mode bit in its processor status register that determines whether the additional 16 double-precision registers are enabled. In SPARC V9, a clever register addressing scheme is used to address the additional registers in a fully backward-compatible way *(see **070201PDF**)*.

SPARC and MIPS support floating-point operations for single-, double-, and quad-precision IEEE operands; Alpha and PowerPC support only single- and double-precision IEEE arithmetic, but Alpha can also handle most of the proprietary VAX floating-point formats.

Both Alpha and MIPS have conditional-branch architectures that store branch conditions as data in any general register. One of the benefits of this conditional-branch architecture is that branch conditions—either floating-point or integer—can be computed well before their use because the only constraint on the number of pending conditions is the number of available registers.

The need for more than one pending branch condition arises most often in floating-point code, where complex loops are typically unrolled to reveal maximum operation overlap and pipeline utilization. This was the motivation for the multiple condition-code fields in POWER and PowerPC. To address this issue, SPARC V9 adds three new sets of floating-point condition codes for a total of four sets. This is fewer than the eight sets offered by PowerPC, but those eight are shared by integer and floating-point conditions.

Performance Instructions

All of these architectures have been influenced by recent experience with optimizing compilers and how real programs actually behave. As a result, new instructions have been included that combine operations but still fit in the RISC mold.

The most obvious examples are the Alpha and SPARC V9 conditional move instructions. These instructions combine the effect of a conditional branch and a register-to-register copy by testing a condition and then moving a source register to a destination register if the relationship is satisfied. Since no branch is actually executed, pipeline utilization is potentially increased.

Conditional moves are not currently part of the MIPS architecture, but the TFP processor being developed by SGI will implement them.

Another area ripe for performance improvement is branches. Branches can have a significant negative effect on the performance of advanced implementations because so much prefetching is required to keep multiple execution units fully occupied. Anything that can be done to warn instruction-fetch logic of the target of an upcoming branch will improve performance.

SPARC V9 needed a new class of branches to test its new 64-bit condition codes anyway, so the architects decided to make major changes to the branch architecture. The new branches can compare a register with zero or test either the 32-bit or 64-bit condition codes. In addition, a static branch prediction bit has been added. PowerPC has a similar prediction bit in its branches. By contrast, Alpha branch prediction is based on the direction of the branch: forward is predicted unlikely while backward is predicted likely.

An obvious differentiating factor between these architectures is delayed branches: SPARC and MIPS have them, while Alpha and PowerPC do not. Delayed branches are an advantage for simple RISC pipelines because they allow greater pipeline utilization with very little hardware cost. For a superscalar implementation, the situation is different because a superscalar processor has a sophisticated instruction-fetch unit. The I-fetch unit must follow branch targets ahead of time and attempt to fetch multiple instructions per cycle. Delayed branches are a special case that makes these tasks more complex to implement.

Prefetches and hints are another class of performance instructions. SPARC V9, PowerPC, and Alpha have prefetch (PowerPC calls them "touch") instructions that are intended to be used by programs to warn the memory hierarchy of the addresses of upcoming data accesses. The hardware is free to ignore the advice of the prefetch instructions or to attempt to move the data at the prefetch address into faster parts of the memory hierarchy, such as a first-level cache.

SPARC V9 also has a prefetch instruction for the instruction stream. Since the "branch-never-predicted" instruction can never cause a branch, software may use it to inform instruction fetch hardware of an upcoming branch long before the fetch hardware can prefetch and analyze it.

Alpha has special hints in its indirect branches to facilitate prefetching. The hint can encode the low bits of the likely target address, if known, which allows an instruction-cache access to start before the instruction has been dispatched. For indirect call and return, the hint encodes information about the use of a hardware stack of likely return addresses. Return addresses are pushed on the stack when subroutines are called and popped when subroutines are exited. This return stack can provide a likely target address for the fetch logic very early, which increases the effectiveness of instruction prefetching.

Memory and Trap Models

As processor implementations increase in sophistication, the intuitive, serial execution model of program behavior becomes a hindrance to high-performance hardware. What is preferable for implementations is to allow memory references and exceptions to be reordered arbitrarily.

As a result, architectures are starting to incorporate memory and trap models that sacrifice strict serial behavior in favor of performance. For example, SPARC V9 processors can implement three different memory models. These models permit the hardware to reorder memory references with varying degrees of freedom. The Alpha and PowerPC architectures specify a weakly ordered memory model for maximum freedom.

Alpha also explicitly specifies an imprecise arithmetic trap model to reduce the complexity of implementations that issue multiple instructions per cycle and potentially complete them out of order. SPARC V9 can have an imprecise trap model as well.

Weakly ordered, imprecise memory and trap models have many implementation and performance implications, and they will be covered in more detail in an upcoming article.

Software and ABI/API Considerations

There are several software issues surrounding the transition from 32 to 64 bits. One of the first issues encountered is deciding on the sizes of primitive data types. The original C language manual, for example, recommended that "int" be the machine's natural register size so operations involving int will be as fast as possible. Unfortunately, this advice is not very useful now, since these 64-bit architectures have no speed difference between 32- and 64-bit operands.

At one point, meetings between several architecture groups, including representatives from DEC, HP, SGI, MIPS, IBM, and Sun, convened to discuss potential sizes for primitive data types. In the case of C, there are only three useful combinations: long 32 bits, int 32 bits; long 64 bits, int 32 bits; and both 64 bits. A new data type, "longlong," has been proposed as a new, 64-bit data type. In the end, the groups could not agree unanimously. An Internet forum, C64, fosters discussions of these issues.

A very important concern is how primitive data types affect the size of user-defined, aggregate data types (records and structures). If basic integers and pointers suddenly double in size, structures may double as well. In some cases, this bloating is intolerable: many network protocols are set in stone with packet headers that encode routing information as 32-bit integers.

Thus, there must be at least one primitive, 32-bit data type. The most natural choice in keeping with the ANSI standard for C is int, but there are also compelling reasons for int to be 64 bits. According to the ANSI standard, intermediate results in complex expressions are implicitly of int length. Also, int and address pointers are usually assumed to be the same size. Consequently, it makes no sense for int to be 32 bits on a 64-bit architecture with a 64-bit compiler.

Yet another thorny issue is what interface libraries should provide. Libraries provide access to commonly used functions and serve as an interface to operating system services. To take full advantage of 64-bit addressing, the operating system needs to be a full 64-bit implementation. Unfortunately, a mixed environment requires at least two sets of libraries: one to provide a 64-bit interface to functions and services and one to translate between the 64-bit operating system and 32-bit applications.

The goal for development environments is to insulate programmers from multiple sets of libraries with one set of universal interface definition files (the so-called header or include files) and one set of documentation.

According to some who have already been dealing with these problems, the POSIX UNIX standard is fairly clean, with few "#ifdef 64BITS..." to chose between 32-bit and 64-bit code during compiles. Network code is requiring much more scrutiny and work since the protocols must be bit-for-bit compatible. The implementors of recent versions of the X Window system have already had to deal with a lot of cross-platform issues, and it is reported to be clean and portable.

As with the underlying architecture, system software architects are taking advantage of this transition period to improve software protocols. For example, the MIPS subroutine-calling convention uses a maximum of four registers to pass parameters to subroutines. Since the libraries and compilers need to be rewritten to some extent anyway, the protocols are being revised. Years of experience indicate that passing up to eight parameters in registers can improve performance.

Conclusions

The transition from 32-bit to 64-bit architectures is being motivated by a few large, existing applications and the anticipation of new classes of "killer apps" that need huge address spaces. The transition simultaneously causes problems and provides opportunities. The problems are mainly issues of backward compatibility with existing software and language definitions. The opportunities are for architectural improvements and new markets.

The key point about the transition from 32-bit to 64-bit architectures is that architects seems to have learned some lessons from the past: all four of the current 64-bit architectures provide good support for 64-bit operations and linear addressing. The folly of segmentation seems to be understood, and all implementations are making sure that software will not use unimplemented high address bits in non-upward-compatible ways. These architectures are all capable of delivering the full potential of a 64-bit address space.

Still, there are significant architectural differences, summarized in Table 1. Alpha lacks loads and stores for byte and halfword data and has no direct integer divide support. MIPS-III has direct support for 128-bit multiply and divide results. SPARC V9 has the most complete set of conditional-branch instructions: with/without annulling and predict true/false.

The 64-bit architectures will not have a significant impact on markets until both 64-bit processors and 64-bit operating systems are in place. Since a great deal of work is required to update an entire environment, these operating systems will emerge slowly and have increasing levels of 64-bit capability over time.

SGI already offers a machine with 16G of directly addressable real memory, but it is interesting to note that at current disk transfer rates (optimistically, 5 Mbytes/s), it would take approximately an hour just to fill all 16G with information from disk (although disk arrays with multiple controllers can dramatically improve transfer rates).

It is still too early to tell what impact 64-bit architectures will have on computer applications. It is certain, however, that by the end of the decade, microprocessors and dense memory chips will result in desktop computers with huge memories and incredible performance. It is up to software vendors to invent the applications that will justify the hardware.♦

For More Information

The Alpha architecture is documented in the *Alpha Architecture Handbook* published by DEC. This handbook documents the user programming model and registers, but leaves out some system-mode details. For a copy of this document, call 800/DEC-2717 or 508/568-6868. *(Also see **060302.PDF**).*

All MIPS architectures, including MIPS-III, will be fully documented in a forthcoming book from Prentice Hall; it should be available a few weeks after the issue date of this article. Information on specific MIPS processors is available directly from processor vendors. (Also see µPR 10/16/91, p. 10.)

SPARC V9 is documented in *SPARC-V9 Architecture Specification Release R1.2*, which is available from SPARC International. Contact Bob Smith at 415/321-8699, extension 245. A book from Prentice Hall will also be available later in the year. *(Also see **070201.PDF**).*

Information on PowerPC is not yet available publicly.

