The PowerPC Architecture™: 64-Bit Power with 32-Bit Compatibility

C. Ray Peng†, Thomas A. Petersen‡, and Ron Clark*

†Motorola Inc., ‡International Business Machines Corporation
Somerset Design Center, 9737 Great Hills Trail, Austin, Texas 78759

*International Business Machines Corporation
11400 Burnet Road, Austin, Texas 78758.

Abstract

This paper details the 64-bit PowerPC Architecture specification. It compares and contrasts the 32-bit subset specification against the full 64-bit specification. Architecture, application, OS, and hardware implications of the 64-bit specifications are all explored in detail. In addition, 32- and 64-bit compatibility and OS migration strategies are described.

The PowerPC 620™ microprocessor implementation is used as a vehicle when examining the 64-bit features. The 620's MMU is described, and potential performance implications are discussed.

1. Introduction

Today's high-end workstations and server environments stretch system resources like never before. High data bandwidth is required for data intensive applications like multi-media and commercial database processing. Commercial applications such as transaction processing and technical applications are finding ways to exploit large effective and virtual address spaces. Physical memory size must scale up with aggregate system processing power as well as for large individual applications, otherwise all applications suffer in performance. An implementation of the PowerPC™ 64-bit architecture such as the PowerPC 620 microprocessor delivers dramatic improvements on all of these fronts making it an ideal solution for the high-end workstation and server markets.

The PowerPC 64-bit architecture is a proper superset of the PowerPC 32-bit architecture. 32-bit PowerPC application binaries run unmodified on a 64-bit implementation. Operating systems written for a PowerPC 32-bit implementation can run on a 64-bit implementation. The modifications required are similar to those typically associated with an OS port within the same family. In addition, a 32-bit PowerPC OS has a clear and concise migration path that allows adoption of portions of the 64-bit feature set as the OS grows to meet expanding needs. From the OS perspective, the PowerPC implementation is one family and one architecture, allowing scalable implementation from the PowerPC 601™ microprocessor[1] up to the latest member of the PowerPC family, the PowerPC 620 microprocessor.

This paper presents an overview of the PowerPC 64-bit architecture. The PowerPC architecture translation scheme is discussed in detail, and operating system usage of the mechanism is explored. The mechanics of translation are presented for both 32 and 64-bit translation schemes. Differences between the two mechanisms are discussed. Compatibility between 32 and 64-bit mechanisms is illustrated through a discussion of OS evolution from 32-bit to 64-bit. Finally, details of the 620 translation implementation are discussed.

2. PowerPC 64-bit architecture features

The PowerPC 64-bit architecture defines a true 64-bit architecture to the operating systems and application programs. The definition expands the 32-bit fixed-point and load/store data path in the PowerPC 32-bit architecture to 64 bits to allow faster manipulation of an extended-range of data values (the floating-point data path was 64-bit all along). It also expands the effective address width from 32 bits to 64 bits and thus increases the address space from 4 giga-bytes to 16 hepto-bytes.

2.1. 64-bit data path

The PowerPC 64-bit architecture extends the width of the General Purpose Registers (GPRs), the Link Register (LR), and the Count Register (CTR) to 64 bits. This allows `long long` data type in C to be directly manipulated by programs without resorting to time consuming multiple-instructions arithmetics. New double-word fixed-point instructions are introduced to operate on 64-bit quantities. These include double-word multiplication (mulld/mulhd/mulhdu), division (divd/divdu), comparison (cmpdi/cmpd/cmpldi/cmpld), rotation (rldicl/rldic/rldcl/rldcr/rldimi), shifting (sld/srd/sradi/srad), trapping (tdi/td), and leading zero count-

ing (cntlzd). In addition, existing 32-bit arithmetic and logic instructions are augmented to handle 64-bit quantities. Instructions that modify the Condition Register (CR) are also augmented to take into account the 32-/64-mode bit in the Machine Status Register (MSR) and test out the appropriate fields accordingly before setting the CR.

To accommodate 64-bit registers, internal data paths between the GPRs, the fixed-point functional units, and the branch unit, as well as the load/store data bus between the GPRs and the cache/memory are all expanded from 32 bits to 64 bits. This expansion speeds up data movement tremendously in 64-bit PowerPC implementations, and brings relief to applications that need to move large volumes of data around such as multi-media video applications. New double-word load (ld/ldu/ldx/ldux) and store (std/stdu/stdx/stdux) instructions are added to move double-word quantities between the GPRs and the memory.

2.2. 64-bit address path

The PowerPC 64-bit architecture also extends effective addresses from 32 bits to 64 bits. For instruction fetches, an effective address can be taken directly from the Count/Link Registers, or it can be calculated by adding the immediate field in the branch instruction to the current instruction address. For data accesses, an effective address can be calculated by adding the immediate field in the instruction to a GPR, or by adding two GPR values. All calculations are based on 64-bit arithmetic.

Once an effective address is formed it is sent to the instruction or data memory management unit (MMU) for translation into a real address. The PowerPC 64-bit architecture defines a real address as a 64-bit quantity. However, implementations may choose to implement less than 64 bits by assuming the unimplemented upper bits are all 0's. The PowerPC 620 microprocessor, for example, implements a 40-bit real address space.

2.3. Segment-table based translation

A major extension to the PowerPC 64-bit architecture is the extension of effective address space from 32 bits to 64 bits and the addition of a segment-table based translation scheme. Figure 1 shows the formats of 32- and 64-bit effective addresses. For 32-bit architecture the most significant 4 bits of an effective address designate a segment register (SR) from which the effective address is translated into a virtual address. The 64-bit architecture uses the 36-bit Effective Segment ID (ESID) as a key to search into a 4 Kbyte segment table. A unique segment table entry (STE) is extracted from the segment table and is used to translate the effective address into a virtual address.

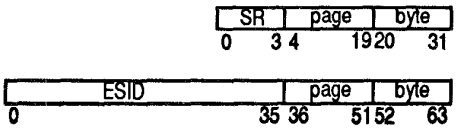

Figure 1. Effective Address Format

While these two schemes may require very different hardware implementations, they are actually very similar from application software's point of view. The only difference is that there are a lot more segments in segment-table based translation (2^{36}) than in the segment-register based translation (2^4). A detailed discussion of this segment-table based translation scheme and its implications to the system software appears in subsequent sections.

3. PowerPC address translation overview

The PowerPC architecture employs a 2-stage address translation scheme to translate an effective address into a real address. The first stage maps an effective address (EA) into a virtual address (VA). In the 32-bit architecture this mapping is accomplished by a segment register mapping as shown in Figure 2. The upper 4 EA bits select 1 of 16 segment registers which provides the VA mapping.

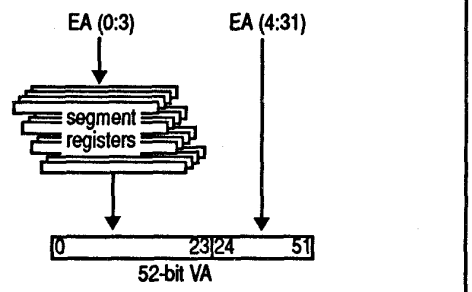

Figure 2. 32-bit EA to 52-bit VA Translation

This same EA to VA translation is accomplished in the 64-bit architecture using a segment table as shown in Figure 3. The 36 upper order EA bits are used to search a segment table stored in the memory. The segment table is a fixed 4-Kbyte structure. Its location is kept in the architectural Address Space Register (ASR) and maintained by the operating system. Each entry in this table stores the mapping for one segment and occupies 16 bytes; so there can be up to 256 segments simultaneously accessible to the processor. When additional mapping is needed, entries can be swapped in and out of this table through operating system services (typically following a *segment fault*).

To speed up the search in the segment table, the PowerPC 64-bit architecture defines an optional Segment

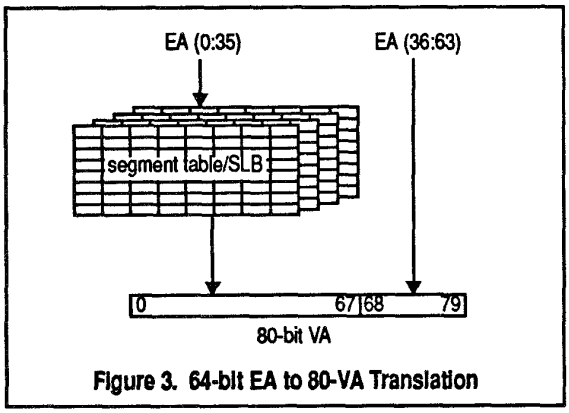

Figure 3. 64-bit EA to 80-VA Translation

Lookaside Buffer (SLB) for caching recently accessed STE's. In addition, the segment table is organized as a hash table to increase the density of the table. A carefully thought-out collision resolution scheme is integrated into the table lookup procedure to reduce the amount of search per lookup[2].

The second stage of the memory translation maps a VA to a Real Address (RA). Most operating systems will use this level of translation to handle page level memory swapping and to manage the main memory as a cache of a larger secondary store.

The VA-to-RA translation is accomplished using a page table stored in memory for both 32- and 64-bit architectures. Figure 4 shows the translation for 32-bit architecture The page table is searched using the upper 40 bits of a VA. If the search is successful, a 20-bit real page number (RPN) is retrieved from this table. The RA is formed by concatenating this 20-bit RPN with the bottom 12-bit byte offset from the EA, which remains unchanged throughout the translation.

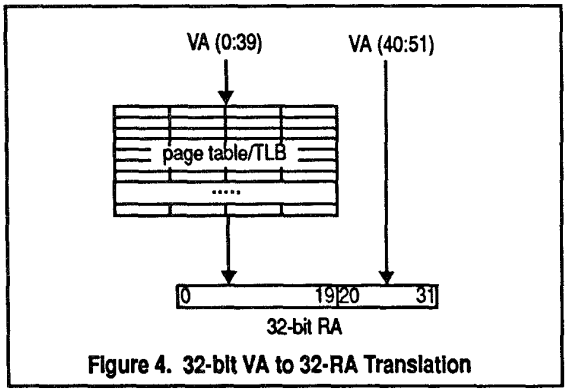

Figure 4. 32-bit VA to 32-RA Translation

The VA-to-RA translation in 64-bit architecture is almost identical to the 32-bit architecture except that all the fields are longer (a page table entry is 8-bytes long for the 32-bit architecture vs. 16-bytes long for the 64-bit architecture). As shown in Figure 4, the upper 68 bits of a VA are used to search for an entry in the page table. A 52-bit RPN is then extracted from the entry and concatenated with the 12-bit byte offset to form a 64-bit RA. As mentioned previously, the 620 only implements 40-bit real address space. Consequently, the page table translation on the 620 will only produce a 28-bit RPN.

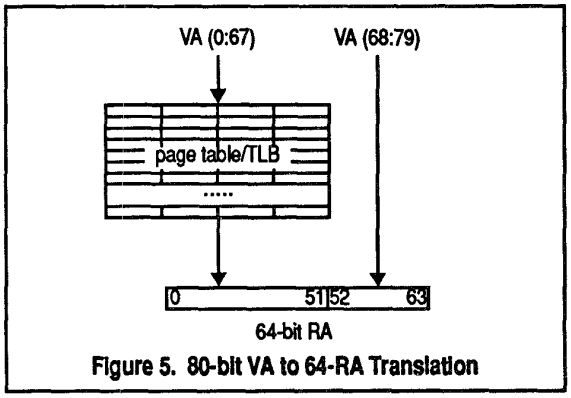

Figure 5. 80-bit VA to 64-RA Translation

Similar to the first stage translation, the architecture defines a cache structure called Translation Lookaside Buffer (TLB) for storing recently accessed page table entries (PTE's). This TLB is defined in both 32- and 64-bit architectures and is maintained by the hardware (to be discussed in section 6.4).

Like the segment table, the page table is organized as a hash table with a similar collision resolution scheme to increase the density of the table and reduce the amount of collisions. It is also maintained exclusively by the operating system (typically through *page fault* service routine). Unlike the segment table the page table is a variable-size structure. Its size can be 2^n bytes long where $16 \le n \le 25$ for 32-bit architecture and $18 \le n \le 46$ for 64-bit architecture. The size of the page table is stored in the architectural Storage Description Register 1 (SDR1) which also stores the location of the page table in the main memory. The hashing function takes into account the page table size when hashing VA bits into the page table entry address during the search.

The 32- and 64-bit translation mechanisms differ only in the size of their address spaces, as summarized in Table 1.

Table 1: Address Space Sizes in PowerPC Environments

	Effective	Virtual	Real
32-Bit	2^{32}	2^{52}	2^{32}
64-Bit	2^{64}	2^{80}	2^{64}

The symmetry of the two addressing models is no accident. It provides an opportunity for clean migration from the 32- to 64-bit environment. Additionally, the two stage translation mechanism is key to high performance operating system design.

4. Operating system perspective of PowerPC address space structure

Modern operating systems use a multi-address-space memory structure to support multi-user and/or multi-tasking programming environments. A typical address space structure for such an OS is shown in Figure 6. An effective address space exists for each running program. Multiple effective address spaces co-exist in the system simultaneously. The software entity for which an effective address space exists is usually called a *process*.

Figure 6 shows two processes, Proc A and B, running concurrently in two different effective address spaces. The figure illustrates some interesting scenarios that take advantage of the PowerPC two stage translation mechanism. For example, ESeg B and ESeg C are mapped to the same virtual segment VSeg B through *segment sharing*. VPage 2 and VPage 4 are mapped to the same real page RPage B through *page sharing*. These two kinds of sharing are frequently used to share instructions and data. ESeg D is currently unmapped. The operating system needs to set up the link (shown as a dotted arrow) before this segment can be accessed. VPage 7 also does not map into any real page. Reference to this page will result in a page fault and cause the operating system to map the page into the real address space. Finally, RPage F and G are I/O pages which are not associated with real memory. Instead

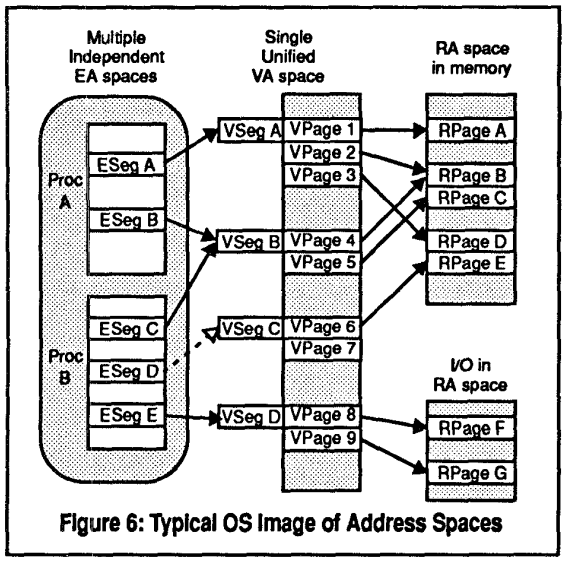

Figure 6: Typical OS Image of Address Spaces

these pages are typically used to directly access a hardware device.

A hypothetical layout of a process effective address space is shown in Figure 7. The effective address space is a combination of application program areas and global OS areas. It is typical to see the OS mapped to the same relative segment location in each process address space. This allows the OS to field *system calls* (i.e., application program requests for OS services) without having to change addressing context in order to access its global data. The OS areas are not modifiable by application programs. In secure operating systems, the OS data is hidden and not accessible at all to application programs. This is accomplished using the PowerPC storage protection mechanism.

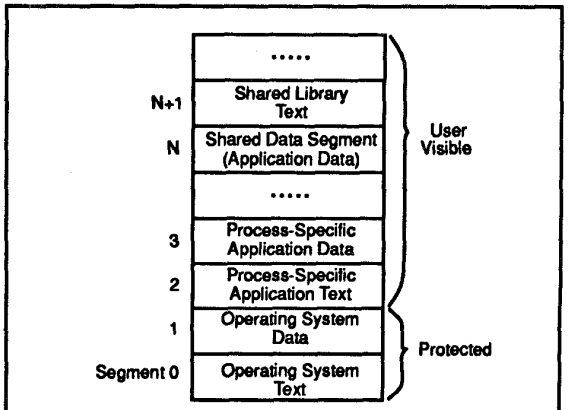

Figure 7. Hypothetical Process Address Space Layout

Each process effective address space contains instructions and data segments for the application program. These areas are accessible in non-privileged execution mode and typically consist of a combination of areas. Some areas are private to each process and some are shared across multiple processes. How much is shared and whether the sharing is accomplished via segment sharing or page sharing is dependent on operating system implementation. In the AIX™ operating system, for example, the instruction text areas of the application's address space (being read only) are shared across all processes needing the executable text. This includes both shared library text and program text when multiple processes happen to be executing the same program concurrently. Data is normally private, per process, unless a shared area of data is explicitly set up by two or more cooperating processes for purposes such as multiple-process dynamic data or shared access to a mapped file.

Overall, the PowerPC memory model is designed to allow efficient operation of modern operating systems. It provides hardware facilities that simplify the software

overhead associated with maintaining multiple independent address spaces. It also provides a firm foundation for secure software systems.

5. 32/64-bit architectural compatibility and operating system migration

Fundamentally, a goal of the PowerPC architecture is to provide binary compatibility for 32-bit applications across all PowerPC environments. Additionally, the PowerPC architecture allows flexibility in how a 32-bit OS approaches 64-bit evolution. Market forces will determine when a particular OS evolves to 64-bits, and when that time comes, the PowerPC architecture allows OS's to pick the 64-bit features it needs to exploit. Table 2 summarizes the compatibility properties of the PowerPC architecture.

Table 2: PowerPC Compatibility

	32-bit hardware implementation	64-bit hardware implementation
32-bit OS	• Runs 32-bit apps.	• Runs 32-bit apps. • Minimal changes in OS port
32-bit OS evolving towards 64-bit	• Runs 32-bit apps.	• Runs 32-bit apps. • Allows stepwise evolution • OS's pick from 64-bit features
64-bit OS	• N/A	• Runs 32-bit apps. • Runs 64-bit apps. • New OS version

This table illustrates one of the most important elements of the PowerPC architecture - compatibility between 32-bit and 64-bit operating environments. The architecture is defined as a full 64-bit architecture with a 32-bit subset that includes a 32-bit execution mode and 32-bit (effective) address spaces. Furthermore, 32-bit implementations of the PowerPC architecture only need to implement the 32-bit subset. The 32-bit execution environment is a subset of the 64-bit PowerPC environment. This allows 32-bit application binaries written to a standard PowerPC 32-bit application binary interface (ABI) to run unmodified, with appropriate OS support, on either a 32-bit or a 64-bit PowerPC system. Simply put, 32-bit applications run on all PowerPC systems.

Table 2 also highlights the issues associated with compatibility and migration at the OS level. The PowerPC architecture is designed to support a full 64-bit OS (i.e. one that fully supports and exploits the 64-bit capabilities of the hardware). It also enables existing 32-bit OS's to migrate onto 64-bit hardware. This is accomplished by building an OS migration path into the PowerPC architecture. The migration path for a 32-bit OS is evolutionary, not revolutionary.

The first row of Table 2 shows the first phase in OS evolution; simply migrate the 32-bit OS to run on PowerPC 64-bit systems. Ideally, this occurs with minimal change to the 32-bit OS. PowerPC limits this re-work as follows:

- 64-bit and 32-bit implementations have the same functional definition for key 32-bit privileged instructions that may be used by 32-bit OS's.
- 64-bit implementations support the same fundamental translation facilities as 32-bit hardware.

The first point allows a 32-bit OS to run transparently on 64-bit hardware in 32-bit execution mode, without even being aware that there is a 64-bit environment capability present on the hardware. This is accomplished in a manner closely analogous to the 32-bit compatibility of application instructions (see section 2.1). Key 32-bit privileged OS instructions retain the same semantics between 32- and 64-bit systems. Comparable operations for 64-bit data targets are defined as new, extended 64-bit instructions. Execution mode can be initialized to 32-bit by hardware prior to the OS getting control, and the use of 32-bit privileged instructions does not ever change the execution mode.

While the 64-bit architecture continues to provide a segmented structure for the effective address space, the segment table translation mechanism is introduced to meet the requirements of a 64-bit effective address space. A wholesale replacement of the segment register concept would require a significant rewrite in a 32-bit OS, so the PowerPC architecture retains segment registers even in the 64-bit environment. Thus for a 32-bit OS that just wants to run on a 64-bit implementation the segmentation mechanism is unchanged.

An OS port to the 64-bit architecture will include modifications to support the enhanced 64-bit page table format. For 32-bit OS's, this modification does not imply support of the large real address space.

Row 2 of the compatibility table highlights the evolution path of a 32-bit OS. There are four steps that a 32-bit OS can consider when migrating toward full 64-bit support. They relate to the sizes of architected address spaces in a 64-bit PowerPC system and are relatively independent of one another. The four steps for a 32-bit OS to consider are, in likely order of implementation:

- supporting large real address space;
- supporting 64-bit application address space;

• supporting large virtual address space;

• and internal OS exploitation of 64-bit addresses.

The key attribute of the PowerPC architecture which allows this flexibility and progressive exploitation of different elements of 64-bit facilities is the way that 32-bit compatibility co-exists within the full 64-bit PowerPC architecture. 32-bit compatibility is not a separate mode or state of the processor, but a subset of the integrated whole. The 32-bit OS looking at 64-bit PowerPC sees the subset relationship differently; it sees the 64-bit PowerPC architecture as a proper superset of the environment it currently runs in. Existing applications and 3rd-party software continue to run as the OS begins to support 64-bit applications and/or large real memory configurations.

6. PowerPC 620 implementation

The 620 fully implements the 64-bit translation architecture including 32-bit compatibility hooks. It uses a novel 2-level translation scheme to achieve its performance and speed goals. Figure 8 shows the block diagram of the 620 MMU. The first level MMU consists of 2 independent Effective to Real Address Translation buffers (ERATs): one for instruction address translation (IERAT) and one for data address translation (DERAT). Both ERATs are organized as 64-entry, fully-associative buffers and each is capable of translating one 64-bit effective address into a 40-bit real address every cycle. Upon a miss, the EA is sent to the shared, second-level MMU for translation. The second level MMU consists of a 20-entry, fully-associative SLB and a 128-entry, 2-way set-associative TLB. The first 16-entries of the SLB also serve as the 16 Segment Registers for 32-bit translation. The SLB and TLB translate EA to RA according to the description in Section 3. The translated RA is refilled into the requesting ERAT, which then re-translates the missed EA.

If the SLB/TLB can not translate the EA, a hardware table walk mechanism (TWalk) is initiated. The TWalk searches the segment table, the page table or both, refills the missing entry, then retries the translation. If the TWalk fails, a segment or page fault occurs, and control is passed to the operating system. Figure 9 shows the translation

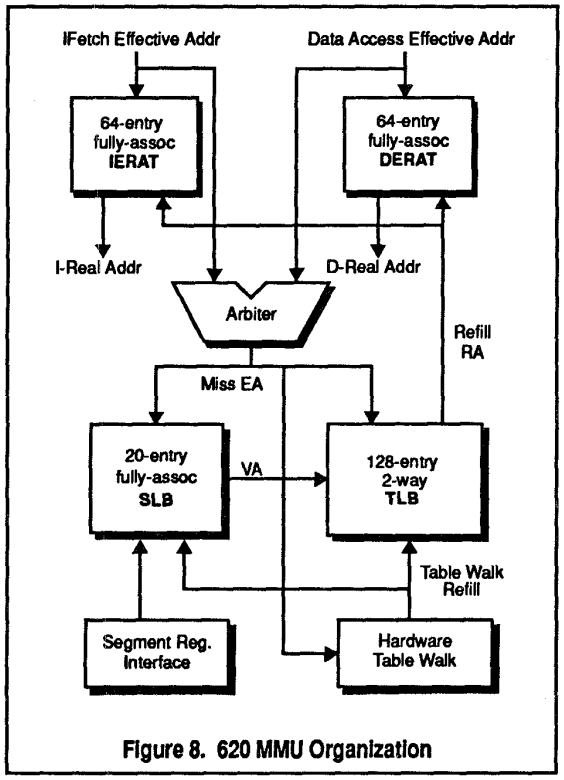

Figure 8. 620 MMU Organization

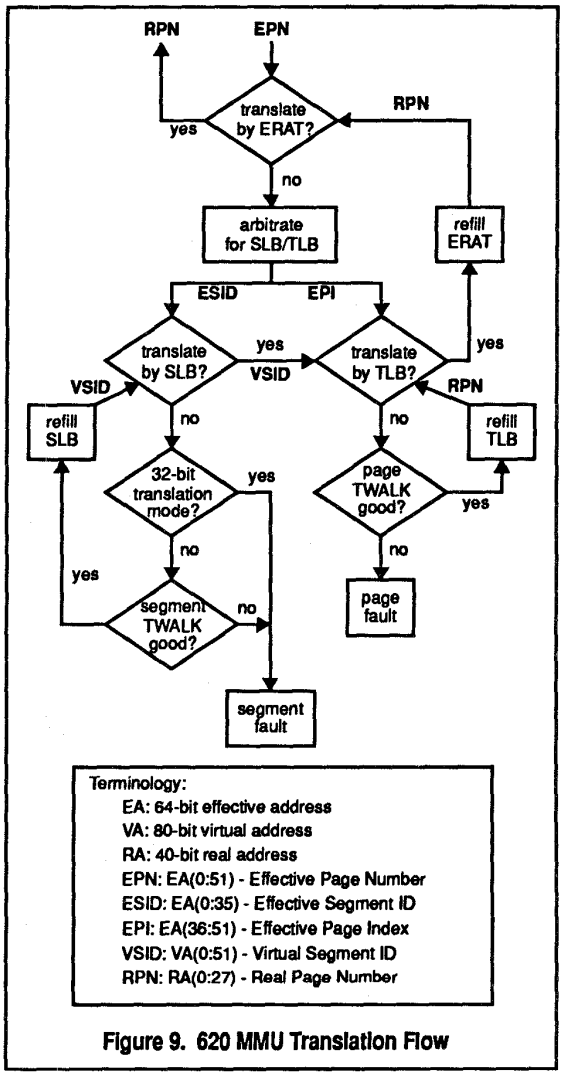

Figure 9. 620 MMU Translation Flow

flow through the 620 MMU. Not shown in this figure are some miscellaneous checks for other storage faults such as protection violation, instruction fetch into a no-execute segment or guarded page, etc.

6.1. First-level MMU - ERAT

A major design goal of the 620 MMU is to support one cycle access to the on-chip L1 caches. Since both the 620 instruction and data L1 caches are physically tagged, the MMU translation is on the critical path for the cache hit signal. To make one-cycle cache access possible, the MMU must complete the translation and deliver the real page number to the cache for tag comparison in less than one cycle. In addition, the MMU needs to maintain a high hit ratio to avoid frequent stalls in cache accesses, which can not be completed without a real page number. To accomplish this, the MMU needs to store enough segment table entries and page table entries for the 32 kilobyte instruction cache and the 32 kilobyte data cache.

To solve the speed problem, the straightforward implementation of the SLB and the TLB as suggested in the architecture books are abandoned because they require several 52-bit VSID comparators* to compare the VSID outputs from the SLB and the TLB in order to determine MMU hit. This creates a speed path through these comparators and requires extra routing space for the VSIDs, which may further slow down the speed path.

Instead the 620 combines the function of the SLB and the TLB into a single fully-associative cache called an ERAT, which maps the EA directly into the RA without going through the intermediate virtual addresses. Because it is fully-associative, there is no need for back-end comparators to verify a hit in the ERAT. An ERAT hit signal can be generated directly from the comparison in its content addressable memory (CAM). Also, because the translation bypasses the intermediate virtual address, there is no need to store the 52-bit VSIDs, which saves chip area and reduces routing.

Because the SLB and TLB are architecturally defined structures, an instruction that manipulates these two buffers may also affect the contents of the ERAT. Specifically, the 620 supports slbie/tlbie/slbia instructions†, which invalidate all or some entries in the ERAT based on different criteria. To implement the slbie and tlbie instructions, the CAM in the ERAT is broken up into two independent CAMs: an ESID CAM, which compares to the upper 36 EA bits, and an EPI CAM, which compares to the next 16 EA bits. For normal translation operations, the match lines of these two CAMs are ANDed together row-by-row to produce an unique match. For slbie, the match lines of the EPI CAM are ignored and the ESID CAM alone selects the entries that need to be invalidated. Similarly for tlbie, the ESID CAM match lines are ignored and the EPI CAM selects entries to be invalidated. The slbia instruction will simply invalidates all the entries in the ERAT.

Figure 10 shows a simple block diagram of the 620 ERAT. The breakup of the CAM into 2 independent CAMs actually helps the speed of ERAT during normal translation. This is because both CAMs are implemented with dynamic logic and by breaking the long 52-bit match line into 2 shorter match lines, the loads on the match lines are smaller.

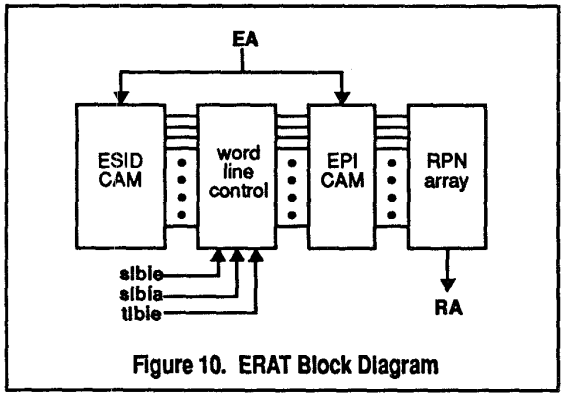

Figure 10. ERAT Block Diagram

To ensure that a sustainable high hit ratio is maintained in the first-level MMU, 64 entries are squeezed into both ERATs. This equips the ERAT 8 times the minimal number of entries required to translate a 32 kilobyte cache. This also results in a relatively low ratio between the number of cache lines and the number of ERAT entries.

In case the ERAT does miss, it can be quickly refilled from the second-level MMU to avoid the long stalls that would have been needed for searching memory resident segment and page tables. In addition, both ERATs are capable of delivering hits under one miss, reducing the penalty of an ERAT miss.

6.2. Second-level MMU - SLB and TLB

The single-table ERAT approach for the first-level MMU is able to solve the speed problem adequately. However, even with 64 entries, the hit rate may remain low if the ERAT has to "cold-start" frequently due to context switches (such as in transaction processing environment). The PowerPC architecture suggests that an slbia instruction be executed upon context switch if the segment table is not shared between processes. This has the undesirable

*. Assuming the SLB is implemented as a fully-associative structure and the TLB as a set-associative structure, the number of VSID comparators needed is equal to the size of the TLB set.

†. The 620 also supports mtsr/mtsrin, which will be covered in section 6.3.

effect of invalidating all ERAT entries, even though many of the entries may still have good RPNs due to segment and/or page sharing between processes.

To speed up the recovery of the ERAT, a second-level MMU implements the SLB and TLB as suggested in the architecture. The slbia instruction will only invalidate the content of the SLB. Therefore, if there are segments and/or pages shared between the outgoing process and the incoming process, or if there are still good TLB entries left from the previous process tenure, they can be quickly brought into the ERAT after the required SLB entry is restored. This avoids unnecessary table searches.

6.3. SLB as segment registers

The SLB also serves as the 16-entry segment register file. A binary decoder is built into the SLB word line circuit to provide support for the mtsr/mtsrin instructions. Executing either of these 2 instructions will activate the binary decoder to select one of the first 16 rows in the SLB for writing. The 4-bit index which selects the row, is also stored into bits 32:35 of the 36-bit ESID-CAM in the SLB. The upper 32 bits of the ESID are filled with 0's. This allows an SLB entry to mimic the operation of a segment register and to match on a 32-bit address designated to use that segment register. A side-effect of the mtsr/mtsrin instructions is that related entries in the ERATs are invalidated. This partial invalidation is accomplished by treating the these instructions like an slbie operation to the ERAT. This is required to maintain the coherency between the ERAT and the SLB.

6.4. Hardware table walk

In case an EA misses in both the first-level and second-level MMU, a hardware table walk mechanism (TWalk) is initiated to search for the missing segment table entry (STE) and/or page table entry (PTE). The 620 will hash the EA into a STE/PTE group address according to the algorithm specified in the PowerPC Architecture Book III[3]. Entries from the group are then brought in for comparison with the target ESID/VSID. If a matching entry is found, it is refilled into the SLB/TLB, which will then restart the lookup cycle to produce hit and refill back to the requesting ERAT. If no matching entry can be found, a segment fault or page fault occurs and control is passed to the operating system. TWalk can also be initiated to update the C-bit (Change-bit) in a page table entry according to the PowerPC architecture specification. This happens when a memory store is about to take place in a clean (C-bit = 0) page.

An important aspect of TWalk is that it can not be initiated by speculative memory accesses. This is mainly to save bus bandwidth and prevent unnecessary pollution of the L2 cache, since every TWalk could fetch up to 256 bytes of data from memory.

7. Summary

The PowerPC 64-bit architecture meets the needs of server and high-performance workstation applications. The architecture delivers large address spaces and high data bandwidth that enables new levels of performance.

Application level binary compatibility is a cornerstone of the PowerPC architecture. 32-bit applications run on any PowerPC compliant system protecting software investments as users upgrade to the 64-bit environment.

Facilities that support the migration of 32-bit OS's are built into the 64-bit architecture. A migration path exists for these 32-bit OS's that allows quick ports to 64-bit implementations, and subsequent stepwise evolution to a full 64-bit OS.

Finally, the PowerPC 620 microprocessor, as the first member of the 64-bit family provides 64-bit power and 32-bit compatibility. Its aggressive design goals are met in the MMU through a novel 2-level hierarchy which provides both high translation hit rates and short cycle times.

References

[1] C. Moore, "The PowerPC 601 Microprocessor," IBM RISC System/6000 Technology: Volume II, IBM Corporation, 1993.

[2] "PowerPC™ Microprocessor Family: The Programming Environments," Motorola Data Book: MPCFPE/AD, IBM Data Book: MPRPPCFPE-01, 9/94.

[3] E. Silha, "The PowerPC Architecture," IBM RISC System/6000 Technology: Volume I,II,III, IBM Corporation, 1993.

In this paper, the term "620" is used as an abbreviation for the phrase "PowerPC 620 microprocessor."