



## WHITE PAPER

# Migrating CPU Specific Code from the PowerPC to the Broadcom SB-1 Processor

05/22/02

# REVISION HISTORY

<i>Revision #</i>	<i>Date</i>	<i>Change Description</i>
SB-1_WP100-R	05/22/02	Initial Release

Broadcom Corporation  
16215 Alton Parkway  
P.O. Box 57013  
Irvine, California 92619-7013  
© Copyright 2002 by Broadcom Corporation  
All rights reserved  
Printed in the U.S.A.

Broadcom® and the pulse logo® are trademarks of Broadcom Corporation and/or its subsidiaries in the United States and certain other countries. All other trademarks are the property of their respective owners.

---

# TABLE OF CONTENTS

<b>Introduction</b> .....	1
<b>Programming Model</b> .....	1
<b>Processor Control</b> .....	2
<b>Memory Map Organization</b> .....	2
The MIPS 32-bit Memory Map .....	3
The Mips 64-bit Memory Map .....	5
<b>MMU/TLB Operation</b> .....	8
<b>Exception Processing</b> .....	9
<b>Conclusion</b> .....	11

---

## LIST OF FIGURES

Figure 1:	MIPS 32-Bit Memory Space .....	3
Figure 2:	Mapping 32-Bit to 64-Bit Addressing .....	5
Figure 3:	MIPS 64-Bit Address Map .....	6
Figure 4:	Xkphys Address Ranges .....	7
Figure 5:	Virtual to Physical Address Translation .....	8
Figure 6:	ROM-Based Exception Vection Assignments .....	10
Figure 7:	RAM-Based Exception Vection Assignments .....	11

## INTRODUCTION

The Broadcom SB-1 (SiByte-1) processor is an implementation of the MIPS64 architecture. MIPS processors, like the PowerPC, are RISC processors. This white paper is intended to give an overview of the SB-1 processor for those familiar with the PowerPC 60x, 7xx, and 74xx-based machines. Two devices using the SB-1 core are currently available: the single-core BCM112x and the dual-core BCM1250. The BCM1250 will be used when specific implementation details are outlined.

While a number of RISC processor details vary among architectures, RISC processors inherently provide minimal hardware facilities in return for performing simple tasks at high speed. Since they don't provide complex proprietary features, it's less likely that software will be written around an aspect of the processor that will disappear in the migration process.

This is not to say that there are no migration issues when moving from one processor to another. Any system designer that survived the CISC to RISC transition of the 1990's will certainly be skeptical of any claim that such migrations are simple. They are, however, far more reasonable than the previous experience most engineers have when porting operating systems and device drivers. The five main aspects of a processor that impact software are:

- Programming model
- Processor control
- Memory map organization
- MMU/TLB operation
- Exception processing

The above topics are listed in ascending order of complexity. This is not to imply that MMU operation is simpler than exception processing, but that the individuals writing exception handlers must be aware of the potential impact of their code on the MMU. Therefore, each of the topics requires an understanding of the topics that precede it in the list.

When covering the different aspects of the SB-1 processor in relation to the PowerPC processor, a basic difference of approach to the architecture emerges. The PowerPC is a descendant of the IBM mainframe mentality; whereas the MIPS architecture was directed at the high performance embedded environment. The key aspect of embedded system design is the need to respond to real-time events. Because of this difference, migrating from the PowerPC to the SB-1 processor is an order of magnitude easier than a migration in the opposite direction.

## PROGRAMMING MODEL

The programming model of both processors can be broadly divided into the user and privileged sections. The user mode of both processors contains 32 general-purpose registers. From a performance point of view, the SB-1 registers are 64 bits wide, as opposed to 32 bits wide. There are also a number of register usage variations.

On the PowerPC, the subroutine return address is stored and invoked through a dedicated link register, which is separate from the GPRs. In the MIPS architecture, the link register function is assigned to general-purpose register thirty-one. This obligates one of the GPRs, however it eliminates the need to move the link register to a GPR to perform subroutine nesting. On the PowerPC however, the movement of the link register through a GPR results in the practical obligation of a GPR, plus the overhead of the instructions to move the address to and from the dedicated link register.

General-purpose register zero always has a special meaning on RISC processors. It's typically used to insert a literal zero as an instruction operand. The PowerPC interprets GPR zero as a literal zero if it is the second register in selected ALU operations. MIPS processors always treat GPR zero as a literal zero. There is no register number zero in a MIPS CPU, and writes to it are ignored. The ability of the PowerPC to selectively use GPR zero as a literal and as a register in different contexts allows the register file to contain an additional register, at least in theory. On a practical level, most compilers avoid using register zero as anything but a literal. And most assembly language programmers have been exasperated when a lapse of concentration causes this feature to appear as a bug in their code. As a result, while GPR zero can function as a register in certain cases on the PowerPC, it is rarely used as such.

The MIPS architecture does not have condition registers for the integer ALU. All conditional branch instructions specify the operands for the compare in the branch. This prevents the need to maintain multiple condition registers, and to have ALU instructions that optionally alter condition flags. A branch on equal doesn't use the zero bit of a condition register that was set or cleared by an earlier ALU operation. Instead, it includes the two registers that are to be compared for the branch decision.

## PROCESSOR CONTROL

The privileged registers that control the processor are different on the two processors mainly in the area of terminology. The PowerPC groups its control registers in a class called the special purpose registers, referred to as SPRs. The MIPS processor also groups these registers together, but refers to them as coprocessor zero. The architectural significance of placing the control registers in a coprocessor is purely academic. They are typically accessed in privileged mode, (however, MIPS allows the kernel to make them visible to the user) and their values are manipulated by moving them to and from the GPRs. The fact that the PowerPC accesses them through an instruction that moves to or from an SPR, and that the MIPS processor uses an instruction that moves to or from a coprocessor register is simply a matter of typing a different mnemonic into the assembly code. All of the registers that control the intrinsic details of the processor: TLB control, cache control, system elapsed time, etc.; are present in both machines. After becoming familiar with the bit level details of these registers, the processors appear more similar than different.

Common to all advanced processors is the dynamic reordering of loads and stores to well-behaved memory. Along with this is the occasional need to force strict programmatic memory access. The PowerPC incorporates the *eieio* instruction. This places a barrier that flows through the load/store logic that prevents reordering across this barrier. This instruction counterpart on the MIPS processors is the *sync* instruction. Aside from the unfortunate coincidence that *sync* has another meaning on the PowerPC, the *sync* instruction on the MIPS is simpler than the *eieio* instruction. The MIPS *sync* instruction does not treat cacheable and non-cacheable accesses as separate classes. A *sync* instruction acts as an ordering barrier to all memory accesses.

For instruction synchronization, the PowerPC issues an *isync* instruction. The equivalent instruction on the SB-1 processor is the *ssnop* instruction. (Which stands for *superscalar nop*.) The *ssnop* instruction occupies one instruction issue slot across all of the parallel pipelines. For a six deep pipelined architecture, six *ssnop* instructions in a row perform the equivalent of an *isync*.

On the PowerPC, the *sync* instruction is a combination of a global *eieio* instruction and an *isync* instruction. On the MIPS architecture, these two operations are issued in series to have the same effect. Since this type of operation sacrifices all performance optimizations for deterministic operation, issuing additional instructions does not affect system throughput.

## MEMORY MAP ORGANIZATION

The memory map is one area where there are significant differences between the PowerPC and the MIPS architecture. Whereas the PowerPC memory map is completely flat and uniform, the MIPS memory map is highly patterned with dedicated address ranges that perform specific access types. To understand the MIPS memory layout, it helps to understand the problems that the architects were trying to solve. A completely flat memory model allows a uniform view of the memory from the point of view of all software. Any program can assume that it has the entire memory space for any purpose. To maintain this illusion, the operating system must manipulate the memory management hardware. This is all well and good for applications in a mainframe or workstation environment. The common attribute of programs in this type of system is that they are written by third parties and all expect the machine to appear completely dedicated to the program in question.

In the embedded environment, there is little need to maintain this illusion. Instead, real-time response and ease of debug is far more important. This is not to say that the MIPS architecture can't provide a true virtual operating environment. It can, it's just not necessary to do so just to implement a basic functional system.

A processor system has different types of addressable memory devices. SDRAM, flash, and I/O all cover the range of well behaved to ill behaved memory. Many devices may appear to exhibit one behavior at certain times or in certain contexts and another type of behavior at another time. For instance, flash memory is well behaved when it is accessed as program or data memory, and as ill behaved memory when it is being programmed. In a flat memory structure, this patterning of memory must be done through the MMU.

This means that even a small system must program the MMUs on the PowerPC if it wishes to control the caching of memory vs. I/O. Whats more, since the MMUs can map any address to any address without limit, they may have mapped the physical exception vectors to a different address. To prevent this, and to force the exception vectors to be fetched from physical address, the PowerPC turns off the MMUs when an exception occurs. But this also shuts down the behavior control of the memory. To survive this, the interrupt service routine must reactivate the MMUs, but it must be mindful of any potential translation conflicts that may exist with the interrupted routine. There is no clean way to have the exception occur outside the translation mechanism. Most embedded systems don't need to support virtual memory, so exception handlers typically just turn the MMUs back on as soon as they start. However, since the hardware doesn't know that this is the case, there is no way to have the MMUs left active. The bottom line is that the PowerPC assumes that a protected virtual memory model is in operation, with the ability to add software support for embedded applications. This negatively impacts the real-time exception response in embedded systems.

### THE MIPS 32-BIT MEMORY MAP

The MIPS architecture reverses this philosophy. It directly targets the embedded environment while providing the mechanism for a full, protected virtual memory system. The MIPS64 memory map, as implemented by the SB-1 processors, is derived from the 32-bit memory map of earlier MIPS machines. To understand the MIPS64 map, it helps to understand the 32-bit map and then expand it.

Figure 1 shows the layout of the MIPS 32-bit memory space. The address space is divided into five separate regions, as determined by the upper bits of the virtual address.

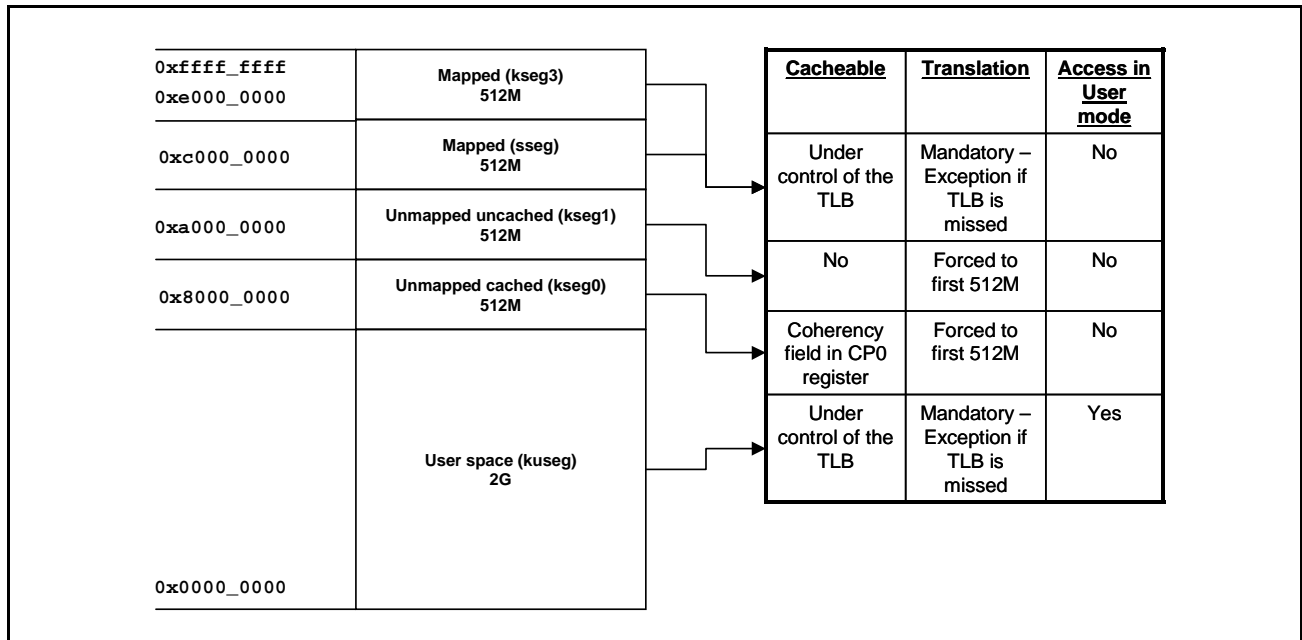


Figure 1: MIPS 32-Bit Memory Space

If the upper bits cause the address to fall into one of the three regions: kuseg, sseg, or kseg3, then the address is presented to the TLBs. The TLBs optionally perform translations, as well as provide cache control for that address. These three regions are referred to as “mapped” regions. Mapped regions are always presented to the TLBs. Unlike the PowerPC, the TLBs are never turned off. If an address accesses a mapped region, the TLBs must have been programmed to handle that address or a TLB miss exception will occur.

The remaining two 512MB regions, kseg0 and kseg1, are known as “unmapped”. Unmapped regions are never presented to the TLBs. The upper address bits of an unmapped region that are used to select the region are set to zero when that address is presented to physical memory. Therefore, an unmapped region, while it is never presented to the TLBs, is always translated down to the bottom of physical memory.

This may appear strange at first, until one realizes that all exception vectors are directed to unmapped regions. This means that the exception vectors will always be directed to addresses specified by the architecture regardless of the programming of the TLBs. Since the TLBs can't affect the exception vectors, the TLBs are never shut down.

Now, there is still the issue of accessing different types of memory from within the exception handler. The exception handler code itself should be cached for speed, as should most of its data structure accesses. The I/O, on the other hand, must be uncached. That's the reasoning behind the existence of two unmapped regions. The kseg1 region is always uncached. The kseg0 region is cached; with its cache control attributes controlled by one of the processor control registers in coprocessor zero. (An SPR by any other name.)

At reset all exception vectors are defined by the hardware to point to kseg1. They are therefore unmapped and uncached. The software can then direct the exception vectors to point to an area in kseg0, so that all future exceptions will run out of cached memory.

Since kseg0 and kseg1 are unmapped, they both translate to the bottom 512 MB of physical memory. This means that they're identical reflections of the same physical addresses, differing only in access through the cache. It turns out that the reset exception vectors are at the top of kseg1, and the relocated vectors are at the bottom of kseg0. As a result, MIPS systems have RAM at the very bottom of physical memory, flash at the top of the first 512 MB, and I/O between the two. If larger amounts of RAM are needed, they're placed above the bottom 512 MB of physical address and accessed through the mapped regions of the memory space. This is explained in more detail in the section covering exception processing.

### THE MIPS 64-BIT MEMORY MAP

The 64-bit size of the registers in the MIPS64 architecture allows the use of 64-bit pointers, and therefore a rather large memory map. However, compatibility with 32-bit code is necessary. To solve this problem without using a mode switch, 32-bit values are sign extended for loads and arithmetic operations. This causes the 32-bit pointers to consistently sign extend into 64-bit pointers. The result is that the 32-bit map appears to be split in the middle when bit thirty-one is set or cleared as shown in Figure 2. The lower two gigabytes of the 32-bit memory map occupy the lower two gigabytes of the 64-bit map. Likewise the upper two gigabytes are sign extended and appear at the top of the 64-bit map. The remaining “gap” in the middle of the 64-bit map is accessible using full 64-bit arithmetic when manipulating pointers.

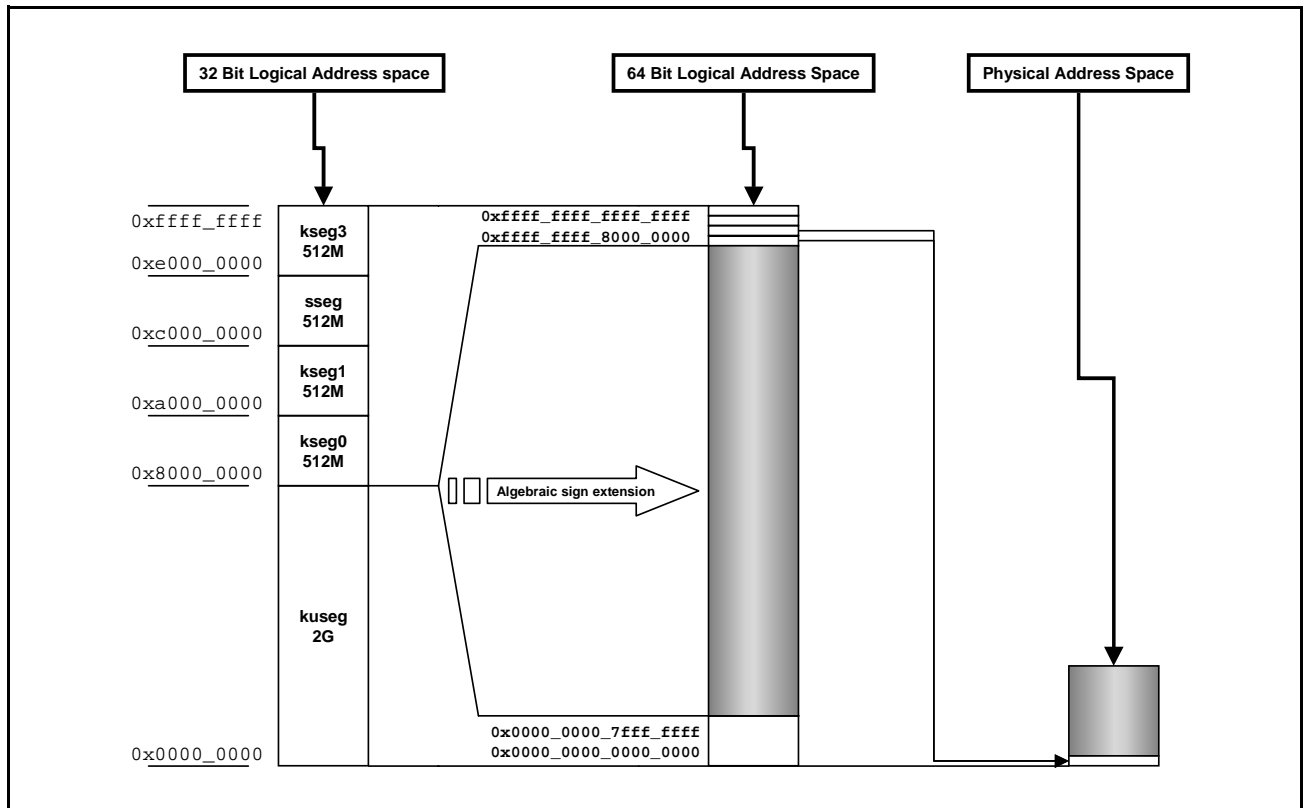


Figure 2: Mapping 32-Bit to 64-Bit Addressing

The full 64-bit map is also divided into regions. The top and bottom two gigabytes maintain the 32-bit compatibility format. The remaining space is divided into four regions. Three of which: xuseg, xsseg, and xkseg are mapped regions and are presented to the TLBs, as shown in Figure 3.

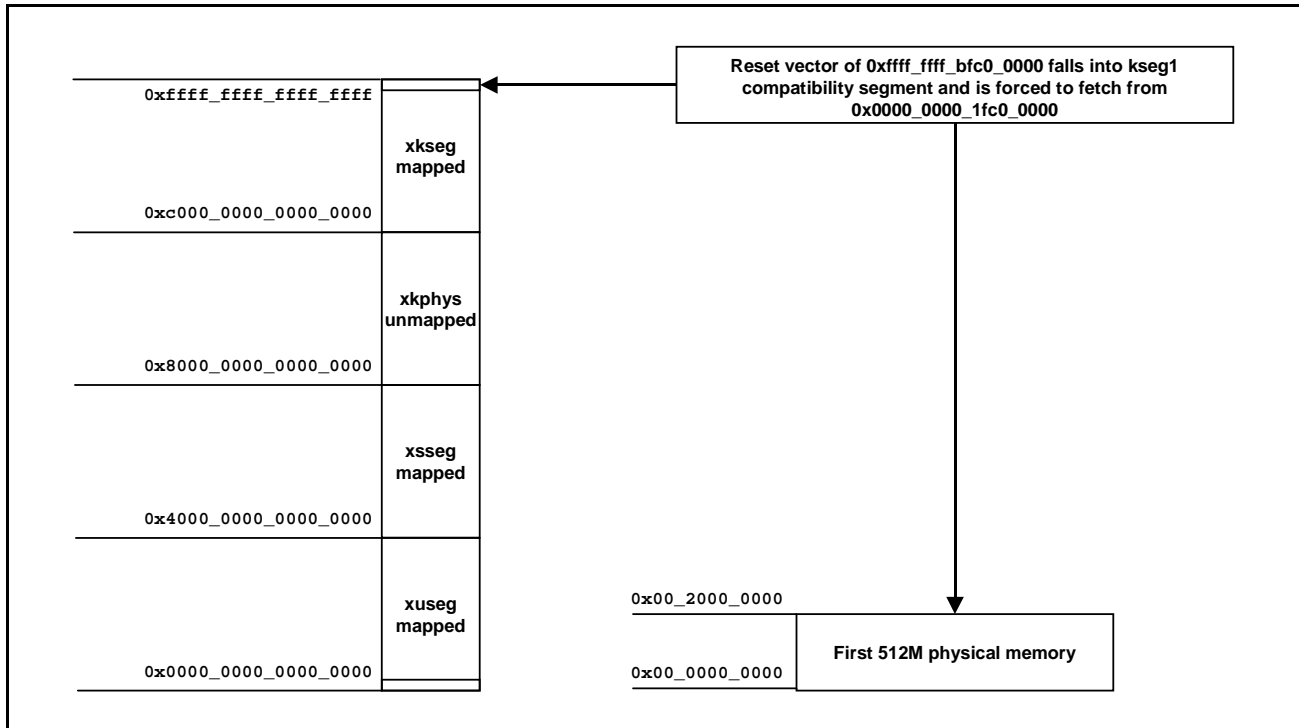


Figure 3: MIPS 64-Bit Address Map

The remaining segment shown in Figure 4, xkphys, is unmapped. The xkphys segment contains eight reflections of the 40-bit physical map, each with a different cache attribute. When accessing a mapped region or kseg0, a three-bit field, yielding eight possible values controls the cache attributes. In mapped regions, this field is in the TLB entry. For kseg0 it's in processor status register. For xkphys, each of the eight reflections allows any location in the physical address map to be accessed with any of the eight possible cache attributes. This allows cached or non-cached access to any physical address.

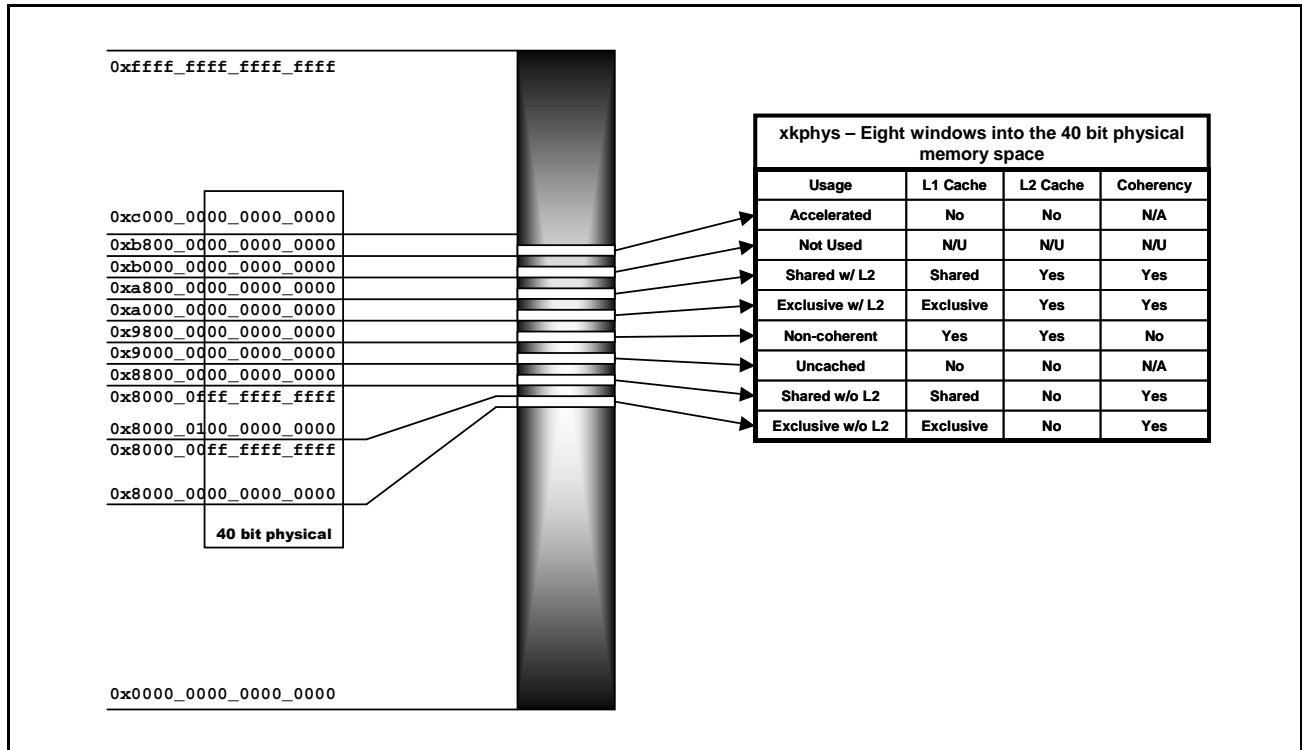


Figure 4: Xkphys Address Ranges

This patterning of the memory map, while complex at first glance, is organized in a way that anticipates the needs of an embedded system. Resets and exceptions aren't forced to deal with cache or TLB issues. The kernel can access any memory location under any context with or without the TLBs. Small 32-bit systems that remain in kernel mode and don't need to address more than 512 MB of total address space don't need to program the TLBs. 64-bit systems of any size that remain in kernel mode don't need to use the TLBs. And yet the mechanisms to support a demand paged virtual operating system are available.

## MMU/TLB OPERATION

It's important to understand the difference between the terms MMU and TLB. The MMU is the memory management unit that maintains the TLB. The TLB is the translation lookaside buffer that is essentially a cached subset of the translation tables in memory. The PowerPC has an MMU that contains a TLB. On some PowerPC processors, such as the 6xx, the MMU is rather simple, relying on a software tablewalk for a reload. More sophisticated processors, such as the 74xx family, have an MMU that will perform hardware tablewalks. The MIPS memory management system uses software to perform tablewalks. Since there is only the TLBs and some access control registers, the MIPS architecture refers to the memory management system simply as the TLB.

The TLBs accept the 64-bit address from the processor and translate it to the 40-bit physical address to be presented to the rest of the system as shown in Figure 5. Out of the potential of the full 64-bit address space, the mapped regions support 44 bits of addressing on the SB-1 processor. This is an implementation specific aspect of the processor, and may be expanded on future version of the CPU. These lower 44 bits, in addition to the upper 2 bits that determine the region, are the valid bits used for addressing. The intervening 18 bits must be all zeros for the 64-bit memory regions, and all ones for the upper 32-bit compatibility region.

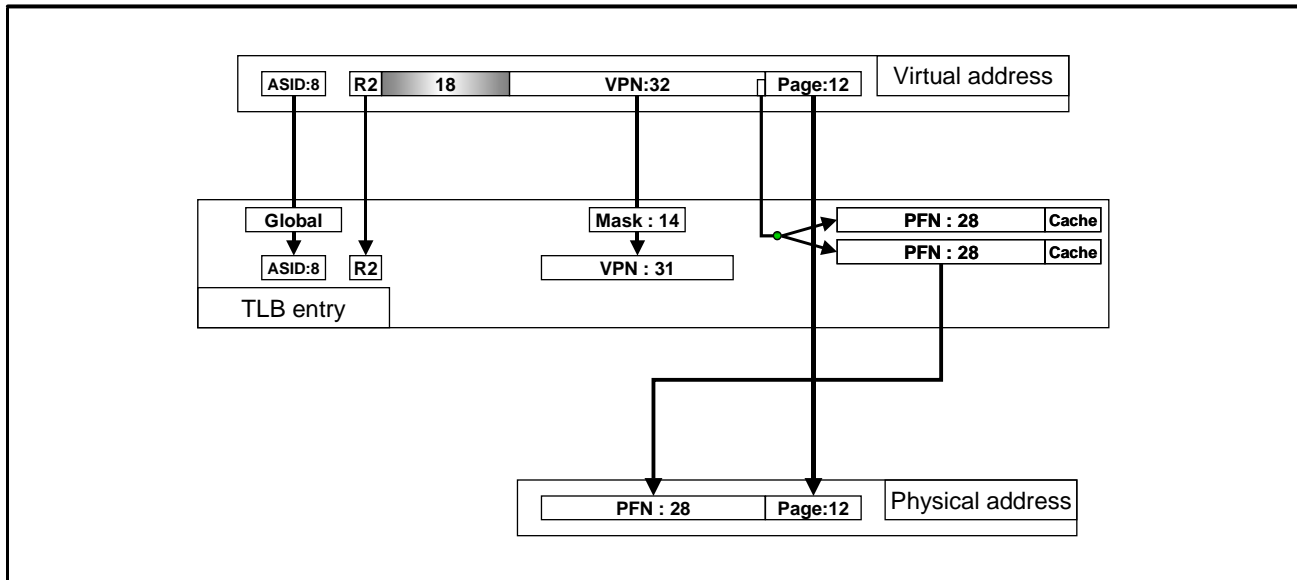


Figure 5: Virtual to Physical Address Translation

In addition to the address bits, the contents of the ASID field are also used in the compare to match the TLB entry. The ASID (Address Space ID) field is set by the operating system in the EntryHi register in coprocessor 0. (The equivalent of an SPR.) If the global bit in the TLB entry isn't set, then the ASID field in the TLB entry must match the value in the EntryHi register. Otherwise the ASID is ignored and just the address bits are compared. The ASID can then be used to prevent flushing the TLB and the instruction cache when a context switch occurs.

The TLB in the SB-1 processor is a unified TLB, used for both instruction and data accesses. There are 64 entries, with page sizes ranging from 4K to 64M in size. When a TLB miss occurs, an exception is taken and the operating system loads the proper entry from memory. Each of the 64 entries contains two physical pages, called physical frames in the MIPS vocabulary. Since an entry can map 64M of memory and there are 64 entries, it's possible to have 4GB of address space defined inside the TLB without using external page tables. If this isn't enough for a large system, then external page tables are easy to implement.

05/22/02

The biggest advantage of a software tablewalk system is its simplicity. A hardware driven system requires the memory tables to fit the structure defined by the processor vendor. Since the silicon vendor must design for the most sophisticated system, even simple systems can have extremely large tables using complicated hashing algorithms. A MIPS system can use linear or small address driven tables that are designed to contain only enough entries for the expected system.

It turns out that any performance increase implied by a hardware tablewalk is highly debatable. Since memory access time to the table is the gating factor, the processor typically stalls instruction execution during a hardware tablewalk. Executing instructions in this void to perform the tablewalk does not impact the speed of the tablewalk. Also, because of the implied sized of hardware tables, they're more likely to cause a page fault requiring the operating system to load them from secondary storage, resulting in a significant increase in latency.

Again, as demonstrated in the memory map section of this article, one of the key features of the TLBs on MIPS processors is the fact that their use is optional. Since cache control can be accomplished through address mapping, the TLBs are only necessary when it's necessary for an OS to dynamically map memory.

## EXCEPTION PROCESSING

The exception processing structure of the MIPS architecture is probably it's single greatest asset when designing an embedded system. There are three aspects of the MIPS exception architecture that are far superior to the PowerPC architecture:

- **True NMI support:** There are separate save registers for normal exceptions and NMIs. This means that NMIs can inherently nest within normal IRQs without any special software support. There is no recoverable exception flag (RI bit) in the MIPS architecture. All of the implications of a non-recoverable interrupt occurring during normal operation that must be handled on the PowerPC are non-existent on the MIPS processors.
- **True debug exceptions support:** On the both the PowerPC and the MIPS processors, debug mode is initiated by a special exception. However, the PowerPC has the same issue with debug exceptions that it has with NMIs. On the MIPS processors, the debug exception return state is stored in yet a third return register. This not only allows debug breakpoints anywhere within IRQs, but it also allows them within NMIs that may be nested within IRQs, all the while maintaining all return information.
- **Exception processing and TLB operation are mutually exclusive:** As outlined in the section on the memory map, it's not necessary to turn off the TLB to process exceptions. In addition, exception handler can freely used cached and non-cached accesses without support from the TLB.

As is typical with RISC processors, an exception causes the SB-1 processor to start executing at a particular exception vector. There are six vectors for accepting exceptions:

- Reset / NMI (0xffff\_fff\_bfc0\_0000 unmapped translates to 0000\_0000\_1fc0\_0000). This vector accepts reset events as well as NMIs.
- TLB (0xffff\_fff\_bfc0\_0200 unmapped translates to 0000\_0000\_1fc0\_0200). TLB events that occur in a processor mode (kernel/supervisor/user) that is set to **disallow** 64-bit addressing cause processing to start at this address.
- XTLB (0xffff\_fff\_bfc0\_0280 unmapped translates to 0000\_0000\_1fc0\_0280). TLB events that occur in a processor mode (kernel/supervisor/user) that is set to **allow** 64-bit addressing cause processing to start at this address.
- Cache (0xffff\_fff\_bfc0\_0300 unmapped translates to 0000\_0000\_1fc0\_0300). Cache errors vector to this location
- Others (0xffff\_fff\_bfc0\_0380 unmapped translates to 0000\_0000\_1fc0\_0380). All other exceptions, illegal instructions, address errors, etc., vector to this location. Interrupt requests are also sent to this vector unless the IV bit in the cause register is set to direct them to the following vector.
- Interrupts (0xffff\_fff\_bfc0\_0400 unmapped translates to 0000\_0000\_1fc0\_0400). If the IV bit in the interrupt cause register is set, then all external interrupt requests go to this vector instead of the "others" vector.



All of these vectors are in kseg1, which means that they are unmapped (as are all exception vectors) and uncached. This is the expected state of the processor at reset. Since these vectors are not only in kseg1, but are also at the top of this segment, they are directed to the top of the first 512 MB of physical memory. The BCM1250, like all SB-1 single chip processor systems, automatically direct these accessed to chip select zero on the generic bus. This means that at reset all exception vectors point to the default boot flash device. Figure 6 shows how these vectors are translated to physical memory.

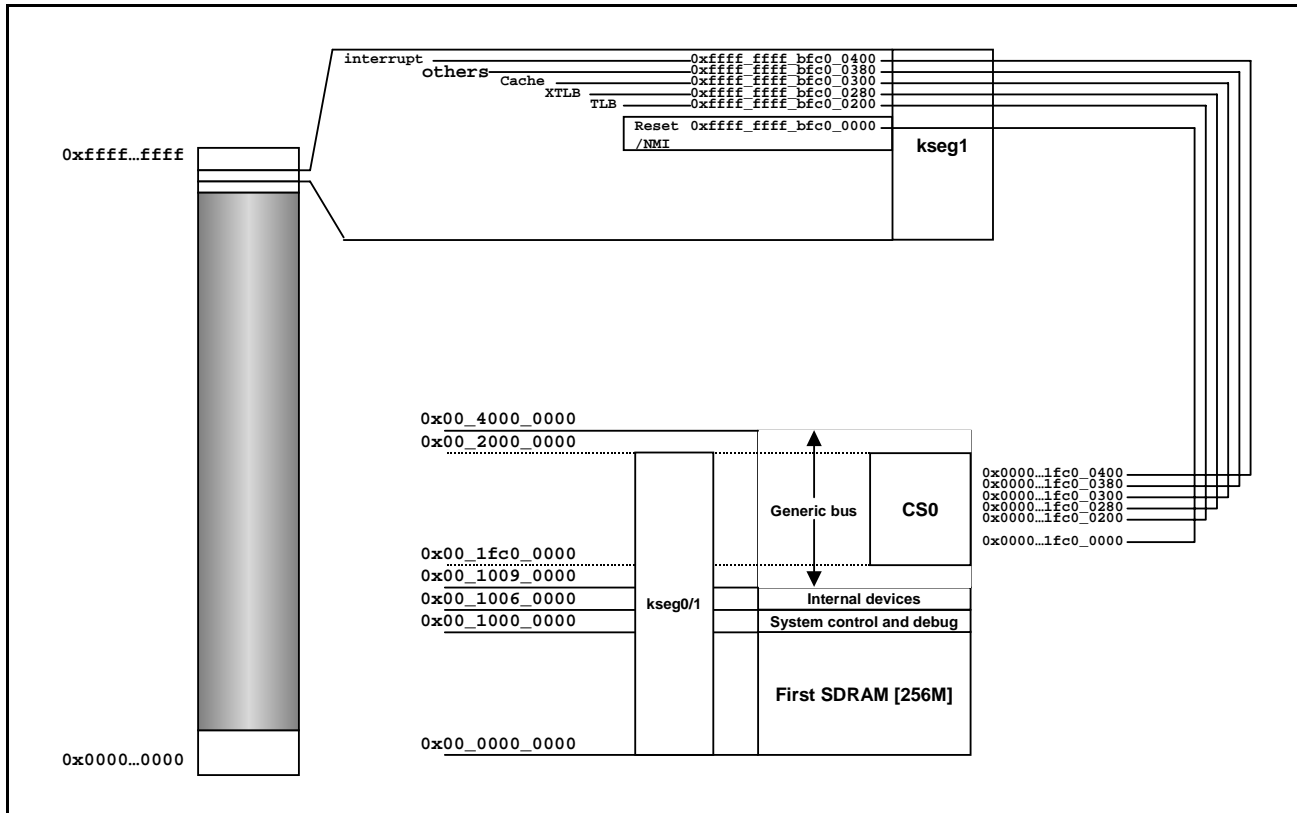


Figure 6: ROM-Based Exception Vection Assignments

After system initialization, the software can clear the boot exception vector (BEV) bit in the control and status register. This will cause the vectors to shift from kseg1 to kseg0. Basically, the "0xbfc0" field of the vector address is translated to "0x8000". By shifting from kseg1 to kseg0, the vectors move from an uncached region to a cached region. In addition, they also move from the top of the assigned segment to the bottom of the segment. This means that they move in the final translation to the bottom of the physical memory space. This takes the vectors out of the flash area on the generic bus access by chip select zero, to the area of physical memory that is directed to the SDRAM memory interface. The vectors therefore move from non-cacheable flash based memory to cacheable SDRAM.

The two exceptions to this rule are the reset/NMI vector and the cache error vector. The reset vector remains in its original position. The cache exception vector undergoes a slightly different modification. Since a cache error handler residing in cache would be an oxymoron, this vector remains non-cacheable. It's address field "0xbfc0" is replaced by "0xa000" instead of "0x8000". This keeps this vector in kseg1 to remain non-cacheable, however it does move it to the bottom of this segment so that moves to the SDRAM area with the other vectors. These vectors are shown in Figure 7.

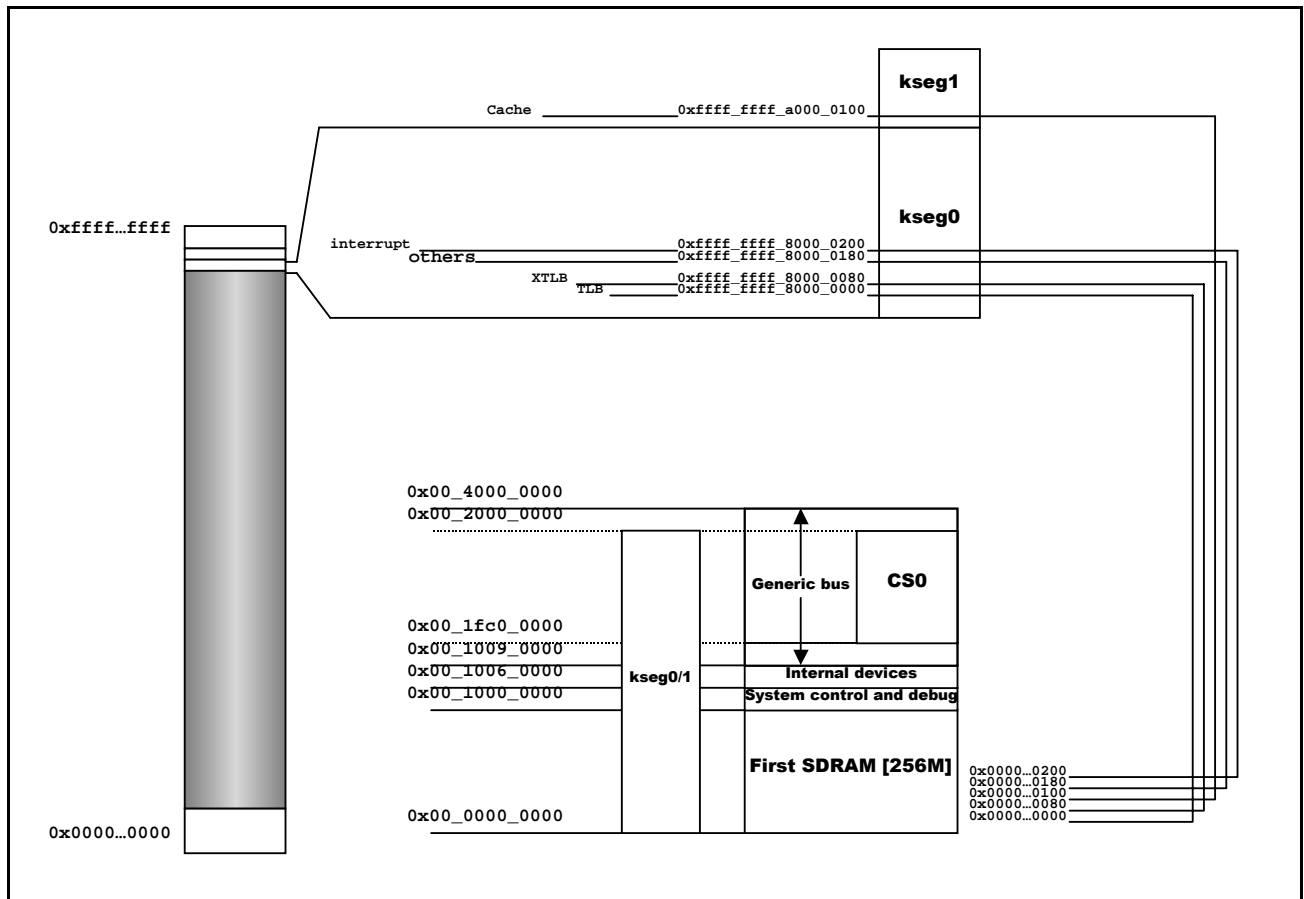


Figure 7: RAM-Based Exception Vection Assignments

An additional method of controlling the location of the vector table is in the multiprocessor vector (MPV) field of the configuration register. This 4-bit field allows the virtual address of the entire vector table to be shifted in increments of 64K. On multiprocessor systems, such as the BCM1250, both cores come out of reset using the same memory address for their vector tables. The reset routines can check the processor version register to discover which CPU is executing the code and then branch accordingly. During normal execution it's more desirable for each CPU to vector to its own code. This becomes more critical as the number of processors in the system increase.

## CONCLUSION

While both the PowerPC and the SB-1 processors are able to handle all of the situations encountered in a normal system design, the MIPS architecture directly address the issues faced when designing a dedicated system. The ability to use advanced features such as memory management without requiring them for the most basic systems allows for simpler system design and faster development cycles.

---

***Broadcom Corporation***

16215 Alton Parkway  
P.O. Box 57013  
Irvine, California 92619-7013  
Phone: 949-450-8700  
Fax: 949-450-8710

Broadcom® Corporation reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom Corporation is believed to be accurate and reliable. However, Broadcom Corporation does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.