MIPS TECHNOLOGIES

MIPS32 4Kc™ Processor Core Datasheet

March 6, 2002

The MIPS32™ 4Kc™ core from MIPS® Technologies is a member of the MIPS32 4K™ processor core family. It is a high-performance, low-power, 32-bit MIPS RISC core designed for custom system-on-silicon applications. The core is designed for semiconductor manufacturing companies, ASIC developers, and system OEMs who want to rapidly integrate their own custom logic and peripherals with a high-performance RISC processor. It is highly portable across processes, and can be easily integrated into full system-on-silicon designs, allowing developers to focus their attention on end-user products. The 4Kc core is ideally positioned to support new products for emerging segments of the digital consumer, network, systems, and information management markets, enabling new tailored solutions for embedded applications.

The 4Kc core implements the MIPS32 Architecture and contains all MIPS II™ instructions; special multiply-accumulate (MAC), conditional move, prefetch, wait, and leading zero/one detect instructions; and the 32-bit privileged resource architecture. The R4000®-style Memory Management Unit contains 3-entry instruction and data TLBs (ITLB/DTLB) and a 16 dual-entry joint TLB (JTLB) with variable page sizes.

The synthesizable 4Kc core implements single cycle MAC instructions, which enable DSP algorithms to be performed efficiently. The Multiply/Divide Unit (MDU) allows 32-bit x 16-bit MAC instructions to be issued every cycle. A 32-bit x 32-bit MAC instruction can be issued every 2 cycles.

Instruction and data caches are fully configurable from 0 - 16 Kbytes in size. In addition, each cache can be organized as direct-mapped or 2-way, 3-way, or 4-way set associative. Load and fetch cache misses only block until the critical word becomes available. The pipeline resumes execution while the remaining words are being written to the cache. Both caches are virtually indexed and physically tagged to allow them to be accessed in the same clock that the address is translated.

An optional Enhanced JTAG (EJTAG) block allows for single-stepping of the processor as well as instruction and data virtual address breakpoints.

Figure 1 shows a block diagram of the 4Kc core. The core is divided into *required* and *optional* blocks as shown.

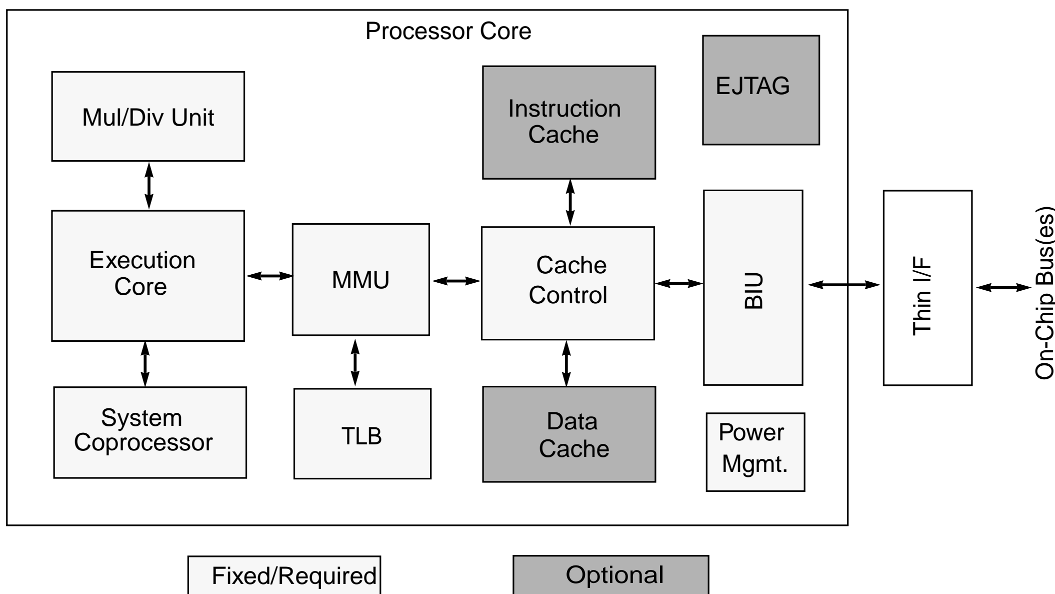


Figure 1 4Kc Core Block Diagram

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Datasheet4U.com

www.DataSheet4U.com

Features

- 32-bit Address and Data Paths
- MIPS32-Compatible Instruction Set
  - All MIPS II Instructions
  - Multiply-Accumulate and Multiply-Subtract Instructions (MADD, MADDU, MSUB, MSUBU)
  - Targeted Multiply Instruction (MUL)
  - Zero/One Detect Instructions (CLZ, CLO)
  - Wait Instruction (WAIT)
  - Conditional Move Instructions (MOVZ, MOVN)
  - Prefetch Instruction (PREF)
- Programmable Cache Sizes
  - Individually configurable instruction and data caches
  - Sizes from 0 - 16KB
  - Direct Mapped, 2-, 3-, or 4-Way Set Associative
  - Loads block only until critical word is available
  - Write-through, no write-allocate
  - 16-byte cache line size, word sectored
  - Virtually indexed, physically tagged
  - Cache line locking support
  - Non-blocking prefetches
- Scratchpad RAM Support
  - Can optionally replace 1 way of the I- and/or D-cache with a fast scratchpad RAM
  - 20 index address bits allow access of arrays up to 1MB
  - Memory-mapped registers attached to the scratchpad port can be used as a coprocessor interface
- R4000-style Privileged Resource Architecture
  - Count/Compare registers for real-time timer interrupts
  - I and D watch registers for SW breakpoints
  - Separate interrupt exception vector
- Programmable Memory Management Unit
  - 16 dual-entry R4000-style JTLB with variable page size
  - 3-entry ITLB
  - 3-entry DTLB
- Simple Bus Interface Unit (BIU)
  - All I/Os fully registered
  - Separate unidirectional 32-bit address and data buses
  - Two 16-byte collapsing write buffers
- Multiply/Divide Unit
  - Maximum issue rate of one 32x16 multiply per clock
  - Maximum issue rate of one 32x32 multiply every other clock
  - Early-in iterative divide. Minimum 11 and maximum 34 clock latency (dividend (*rs*) sign extension-dependent)
- Power Control
  - Minimum frequency: 0 MHz
  - Power-down mode (triggered by WAIT instruction)
  - Support for software-controlled clock divider
- EJTAG Debug Support with single stepping, virtual instruction and data address breakpoints

Architecture Overview

The 4Kc core contains both required and optional blocks. Required blocks are the lightly shaded areas of the block diagram in Figure 1 and must be implemented to remain MIPS-compliant. Optional blocks can be added to the 4Kc core based on the needs of the implementation.

The required blocks are as follows:

- Execution Unit
- Multiply/Divide Unit (MDU)
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- Transition Lookaside Buffer (TLB)
- Cache Controllers
- Bus Interface Unit (BIU)
- Power Management

Optional blocks include:

- Instruction Cache
- Data Cache
- Scratchpad RAM
- Enhanced JTAG (EJTAG) Controller

The section entitled "4Kc Core Required Logic Blocks" on page 3 discusses the required blocks. The section entitled "4Kc Core Optional Logic Blocks" on page 12 discusses the optional blocks.

Pipeline Flow

The 4Kc core implements a 5-stage pipeline with performance similar to the R3000® pipeline. The pipeline allows the processor to achieve high frequency while minimizing device complexity, reducing both cost and power consumption.

The 4Kc core pipeline consists of five stages:

- Instruction (I Stage)
- Execution (E Stage)
- Memory (M Stage)

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

- Align (A Stage)
- Writeback (W stage)

The 4Kc core implements a bypass mechanism that allows the result of an operation to be forwarded directly to the instruction that needs it without having to write the result to the register and then read it back.

Figure 2 shows a timing diagram of the 4Kc core pipeline.

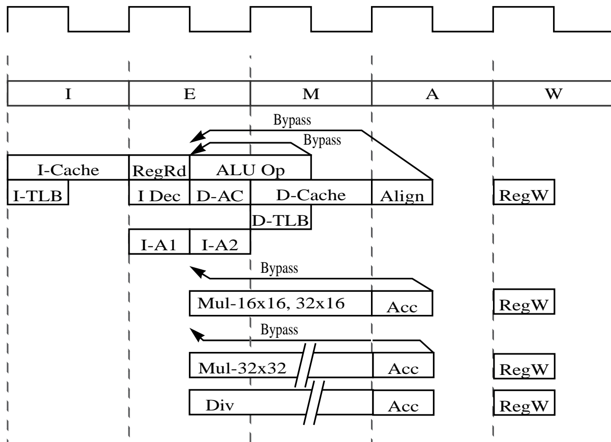


Figure 2 4Kc Core Pipeline

I Stage: Instruction Fetch

During the Instruction fetch stage:

- An instruction is fetched from instruction cache.

E Stage: Execution

During the Execution stage:

- Operands are fetched from register file.
- The arithmetic logic unit (ALU) begins the arithmetic or logical operation for register-to-register instructions.
- The ALU calculates the data virtual address for load and store instructions.
- The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.
- Instruction logic selects an instruction address.
- All multiply and divide operations begin in this stage.

M Stage: Memory Fetch

During the memory fetch stage:

- The arithmetic ALU operation completes.
- The data cache fetch and the data virtual-to-physical address translation are performed for load and store instructions.
- Data cache look-up is performed and a hit/miss determination is made.
- A 16x16 or 32x16 multiply calculation completes.
- A 32x32 multiply operation stalls for one clock in the M stage.
- A divide operation stalls for a maximum of 34 clocks in the M stage. Early-in sign extension detection on the dividend will skip 7, 15, or 23 stall clocks.

A Stage: Align

During the Align stage:

- A separate aligner aligns load data to its word boundary.
- A 16x16 or 32x16 multiply operation performs the carry-propagate-add. The actual register writeback is performed in the W stage.
- A MUL operation makes the result available for writeback. The actual register writeback is performed in the W stage.

W Stage: Writeback

- For register-to-register or load instructions, the instruction result is written back to the register file during the W stage.

4Kc Core Required Logic Blocks

The 4Kc core consists of the following required logic blocks as shown in Figure 1. These logic blocks are defined in the following subsections:

- Execution Unit
- Multiply/Divide Unit (MDU)
- System Control Coprocessor (CP0)
- Memory Management Unit (MMU)
- Transition Lookaside Buffer (TLB)
- Cache Controller
- Bus Interface Control (BIU)
- Power Management

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Execution Unit

The 4Kc core execution unit implements a load/store architecture with single-cycle ALU operations (logical, shift, add, subtract) and an autonomous multiply/divide unit. The 4Kc core contains thirty-two 32-bit general-purpose registers used for integer operations and address calculation. The register file consists of two read ports and one write port and is fully bypassed to minimize operation latency in the pipeline.

The execution unit includes:

- 32-bit adder used for calculating the data address
- Address unit for calculating the next instruction address
- Logic for branch determination and branch target address calculation
- Load aligner
- Bypass multiplexers used to avoid stalls when executing instructions streams where data producing instructions are followed closely by consumers of their results
- Leading Zero/One detect unit for implementing the CLZ and CLO instructions
- Arithmetic Logic Unit (ALU) for performing bitwise logical operations
- Shifter & Store Aligner

Multiply/Divide Unit (MDU)

The 4Kc core contains a multiply/divide unit (MDU) that contains a separate pipeline for multiply and divide operations. This pipeline operates in parallel with the integer unit (IU) pipeline and does not stall when the IU pipeline stalls. This setup allows long-running MDU operations, such as a divide, to be partially masked by system stalls and/or other integer unit instructions.

The MDU consists of a 32x16 booth recoded multiplier, result/accumulation registers (HI and LO), a divide state machine, and the necessary multiplexers and control logic. The first number shown (‘32’ of 32x16) represents the *rs* operand. The second number (‘16’ of 32x16) represents the *rt* operand. The 4Kc core only checks the value of the latter (*rt)* operand to determine how many times the operation must pass through the multiplier. The 16x16 and 32x16 operations pass through the multiplier once. A 32x32 operation passes through the multiplier twice.

The MDU supports execution of one 16x16 or 32x16 multiply operation every clock cycle; 32x32 multiply operations can be issued every other clock cycle. Appropriate interlocks are implemented to stall the issuance of back-to-back 32x32 multiply operations. The multiply operand size is automatically determined by logic built into the MDU.

Divide operations are implemented with a simple 1 bit per clock iterative algorithm. An early-in detection checks the sign extension of the dividend (*rs*) operand. If rs is 8 bits wide, 23 iterations are skipped. For a 16-bit-wide rs, 15 iterations are skipped, and for a 24-bit-wide rs, 7 iterations are skipped. Any attempt to issue a subsequent MDU instruction while a divide is still active causes an IU pipeline stall until the divide operation is completed.

Table 1 lists the repeat rate (peak issue rate of cycles until the operation can be reissued) and latency (number of cycles until a result is available) for the 4Kc core multiply and divide instructions. The approximate latency and repeat rates are listed in terms of pipeline clocks. For a more detailed discussion of latencies and repeat rates, refer to Chapter 2 of the *MIPS32 4K™ Processor Core Family Software User’s Manual*.

Table 1 4Kc Core Integer Multiply/Divide Unit Latencies and Repeat Rates

Opcode	Operand Size (mul *rt*) (div *rs*)	Latency	Repeat Rate
MULT/MULTU, MADD/MADDU, MSUB/MSUBU	16 bits	1	1
	32 bits	2	2
MUL	16 bits	2	1
	32 bits	3	2
DIV/DIVU	8 bits	12	11
	16 bits	19	18
	24 bits	26	25
	32 bits	33	32

The MIPS architecture defines that the result of a multiply or divide operation be placed in the HI and LO registers. Using the Move-From-HI (MFHI) and Move-From-LO (MFLO) instructions, these values can be transferred to the general-purpose register file.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

As an enhancement to the MIPS II ISA, the 4Kc core implements an additional multiply instruction, MUL, which specifies that multiply results be placed in the primary register file instead of the HI/LO register pair. By avoiding the explicit MFLO instruction, required when using the LO register, and by supporting multiple destination registers, the throughput of multiply-intensive operations is increased.

Two other instructions, multiply-add (MADD) and multiply-subtract (MSUB), are used to perform the multiply-accumulate and multiply-subtract operations. The MADD instruction multiplies two numbers and then adds the product to the current contents of the HI and LO registers. Similarly, the MSUB instruction multiplies two operands and then subtracts the product from the HI and LO registers. The MADD and MSUB operations are commonly used in DSP algorithms.

System Control Coprocessor (CP0)

In the MIPS architecture, CP0 is responsible for the virtual-to-physical address translation and cache protocols, the exception control system, the processor's diagnostics capability, the operating modes (kernel, user, and debug), and interrupts enabled or disabled. Configuration information such as cache size and set associativity is available by accessing the CP0 registers, listed in Table 2.

Table 2 Coprocessor 0 Registers in Numerical Order

Register Number	Register Name	Function
0	Index[1]	Index into the TLB array.
1	Random[1]	Randomly generated index into the TLB array.
2	EntryLo0[1]	Randomly generated index into the TLB array.
3	EntryLo1[1]	Low-order portion of the TLB entry for odd-numbered virtual pages.
4	Context[2]	Pointer to page table entry in memory.
5	PageMask[1]	Control for variable page sizes in TLB entries.
6	Wired[1]	Controls the number of fixed ("wired") TLB entries.
7	Reserved	Reserved.
8	BadVAddr[2]	Reports the address for the most recent address-related exception.
9	Count[2]	Processor cycle count.
10	EntryHi[1]	High-order portion of the TLB entry.
11	Compare[2]	Timer interrupt control.
12	Status[2]	Processor status and control.
13	Cause[2]	Cause of last general exception.
14	EPC[2]	Program counter at last exception.
15	PRId	Processor identification and revision.
16	Config	Configuration register.
16	Config1	Configuration register 1.
17	LLAddr	Load linked address.
18	WatchLo[2]	Low-order watchpoint address.
19	WatchHi[2]	High-order watchpoint address.
20 - 22	Reserved	Reserved.
23	Debug[3]	Debug control and exception status.
24	DEPC[3]	Program counter at last debug exception.
25 - 27	Reserved	Reserved.
28	TagLo/ DataLo	Low-order portion of cache tag interface.
29	Reserved	Reserved.
30	ErrorEPC[2]	Program counter at last error.
31	DeSave[3]	Debug handler scratchpad register.

1. Registers used in memory management.
2. Registers used in exception processing.
3. Registers used during debug.

Coprocessor 0 also contains the logic for identifying and managing exceptions. Exceptions can be caused by a variety of sources, including boundary cases in data, external events, or program errors. Table 3 shows the exception types in order of priority.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 3 4Kc Core Exception Types

Exception	Description
Reset	Assertion of *SI_ColdReset* signal.
Soft Reset	Assertion of *SI_Reset* signal.
DSS	EJTAG Debug Single Step.
DINT	EJTAG Debug Interrupt. Caused by the assertion of the external *EJ_DINT* input, or by setting the EjtagBrk bit in the ECR register.
NMI	Assertion of *EB_NMI* signal.
Machine Check	TLB write that conflicts with an existing entry.
Interrupt	Assertion of unmasked hardware or software interrupt signal.
Deferred Watch	Deferred Watch (unmasked by K|DM->!(K|DM) transition).
DIB	EJTAG debug hardware instruction break matched.
WATCH	A reference to an address in one of the watch registers (fetch).
AdEL	Fetch address alignment error. Fetch reference to protected address.
TLBL	Fetch TLB miss.
TLBL	Fetch TLB hit to page with V=0.
IBE	Instruction fetch bus error.
DBp	EJTAG Breakpoint (execution of SDBBP instruction).
Sys	Execution of SYSCALL instruction.
Bp	Execution of BREAK instruction.
RI	Execution of a Reserved Instruction.
CpU	Execution of a coprocessor instruction for a coprocessor that is not enabled.
Ov	Execution of an arithmetic instruction that overflowed.
Tr	Execution of a trap (when trap condition is true).
DDBL / DDBS	EJTAG Data Address Break (address only) or EJTAG Data Value Break on Store (address+value).

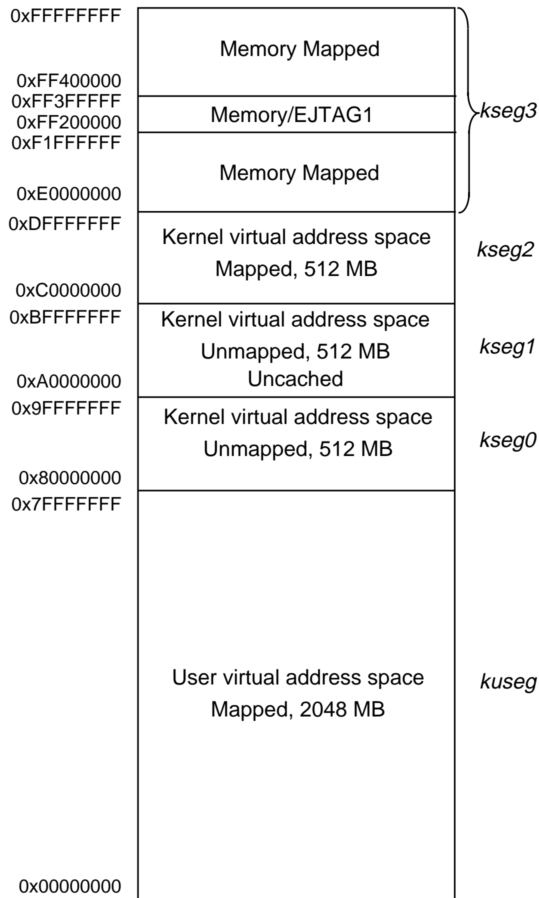
Table 3 4Kc Core Exception Types (Continued)

Exception	Description
WATCH	A reference to an address in one of the watch registers (data).
AdEL	Load address alignment error. Load reference to protected address.
AdES	Store address alignment error. Store to protected address.
TLBL	Load TLB miss.
TLBL	Load TLB hit to page with V=0.
TLBS	Store TLB miss.
TLBS	Store TLB hit to page with V=0.
TLB Mod	Store to TLB page with D=0.
DBE	Load or store bus error.
DDBL	EJTAG data hardware breakpoint matched in load data compare.

Modes of Operation

The 4Kc core supports three modes of operation: user mode, kernel mode, and debug mode. User mode is most often used for applications programs. Kernel mode is typically used for handling exceptions and operating system kernel functions, including CP0 management and I/O device accesses. An additional Debug mode is used during system bring-up and software development. Refer to the EJTAG section for more information on debug mode.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.



1. This space is mapped to memory in user of kernel mode, and by the EJTAG module in debug mode.

Figure 3 4Kc Core Virtual Address Map

Memory Management Unit (MMU)

The 4Kc core contains a fully functional MMU that interfaces between the execution unit and the cache controller. Although the 4Kc core implements a 32-bit architecture, the MMU is modeled after that found in the 64-bit R4000 family.

The TLB consists of two address translation buffers: a 16-dual-entry, fully associative Joint TLB (JTLB) and two 3-entry, fully associative Instruction/Data TLBs (ITLB/DTLB).

When an instruction address is calculated, the virtual address is compared to the contents of the 3-entry ITLB. If the address is not found in the ITLB, the JTLB is accessed. If the entry is found in the JTLB, that entry is then written into the ITLB. If the address is not found in the JTLB, a TLB refill exception is taken.

When a data address is calculated, the virtual address is compared to both the 3-entry DTLB and the JTLB. If the address is not found in the DTLB, but is found in the JTLB, that address is immediately written to the DTLB. If the address is not found in the JTLB, a TLB refill exception is taken.

Figure 4 shows how the ITLB, DTLB, and JTLB are implemented in the 4Kc core.

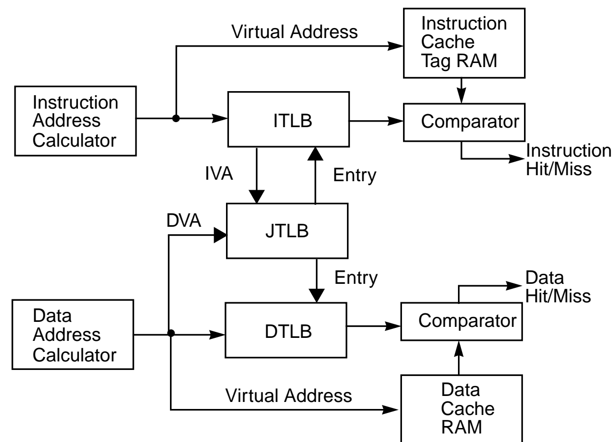


Figure 4 Address Translation During a Cache Access

Translation Lookaside Buffer (TLB)

The TLB consists of three address translation buffers:

- 16 dual-entry fully associative Joint TLB (JTLB)
- 3-entry fully associative Instruction TLB (ITLB)
- 3-entry fully associative Data TLB (DTLB)

Joint TLB (JTLB)

The 4Kc core implements a 16-dual-entry, fully associative JTLB that maps 32 virtual pages to their corresponding physical addresses. The JTLB is organized as 16 pairs of even and odd entries containing pages that range in size from 4 Kbytes to 16 Mbytes into the 4-Gbyte physical address space. The purpose of the TLB is to translate virtual addresses and their corresponding ASIDs into a physical memory address. The translation is performed by comparing the upper bits of the virtual address (along with the ASID) against each of the entries in the *tag* portion of the joint TLB structure.

The JTLB is organized in page pairs to minimize the overall size. Each *tag* entry corresponds to 2 data entries: an even page entry and an odd page entry. The highest order virtual address bit not participating in the tag comparison is

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

used to determine which of the data entries is used. Since page size can vary on a page-pair basis, the determination of which address bits participate in the comparison and which bit is used to make the even-odd determination is decided dynamically during the TLB look-up.

Instruction TLB (ITLB)

The ITLB is a small 3-entry, fully associative TLB dedicated to performing translations for the instruction stream. The ITLB only maps 4-Kbyte pages/subpages.

The ITLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store for the ITLB. If a fetch address cannot be translated by the ITLB, the JTLB is used to attempt to translate it in the following clock cycle. If successful, the translation information is copied into the ITLB for future use. There is a two cycle ITLB miss penalty.

Data TLB (DTLB)

The DTLB is a small 3-entry, fully associative TLB dedicated to performing translations for loads and stores. The DTLB also only maps 4-Kbyte pages/subpages.

The DTLB is managed by hardware and is transparent to software. The larger JTLB is used as a backing store for the DTLB. The JTLB is looked up in parallel with the DTLB to minimize the DTLB miss penalty. If the JTLB translation is successful, the translation information is copied into the DTLB for future use. There is a one cycle DTLB miss penalty.

Virtual-to-Physical Address Translation

Converting a virtual address to a physical address begins by comparing the virtual address from the processor with the virtual addresses in the TLB; there is a match when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either:

- The Global (*G*) bit of the TLB entry is set, or
- The ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a TLB *hit*. If there is no match, a TLB *miss* exception is taken by the processor and software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

Figure 5 shows a flow diagram of the address translation process.

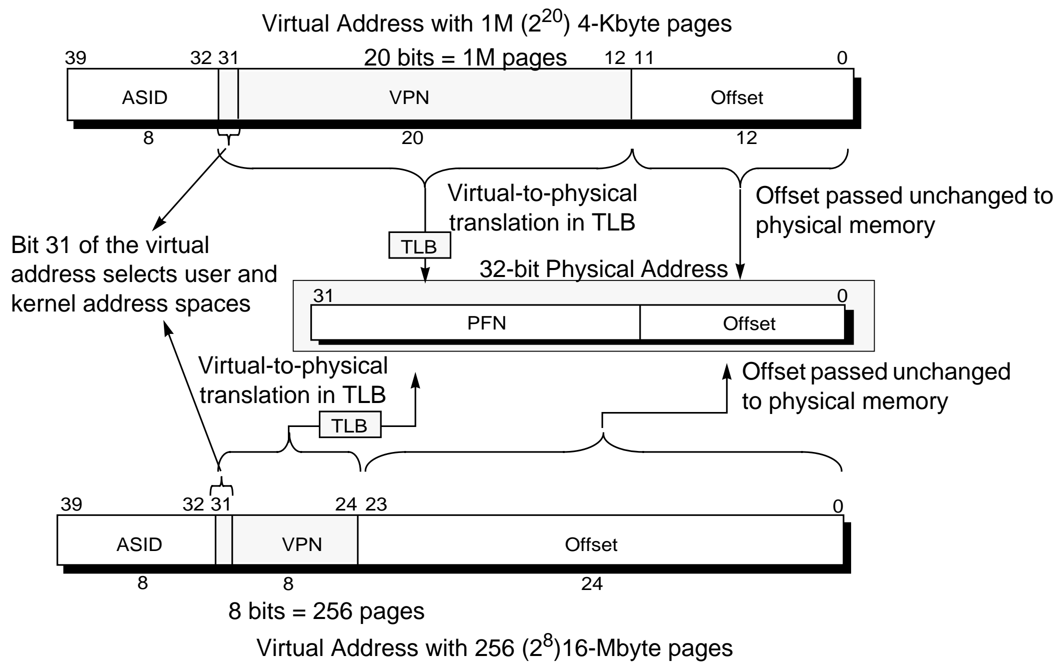


Figure 5 32-bit Virtual Address Translation

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

The top portion of Figure 5 shows a virtual address for a 4-Kbyte page size. The width of the *Offset* in Figure 5 is defined by the page size. The remaining 20 bits of the address represent the virtual page number (VPN), and index the 1M-entry page table.

The bottom portion of Figure 5 shows the virtual address for a 16-Mbyte page size. The remaining 8 bits of the address represent the VPN, and index the 256-entry page table.

In this figure, the virtual address is extended with an 8-bit address space identifier (ASID), which reduces the frequency of TLB flushing during a context switch. This 8-bit ASID contains the number assigned to that process and is stored in the CP0 *EntryHi* register.

Hits, Misses, and Multiple Matches

Each JTLB entry contains a tag portion and a data portion. If a match is found, the upper bits of the virtual address are replaced with the page frame number (PFN) stored in the corresponding entry in the data array of the joint TLB (JTLB). The granularity of JTLB mappings is defined in terms of TLB *pages*. The 4Kc core's JTLB supports pages of different sizes ranging from 4 KB to 16 MB in powers of 4.

If no match occurs (TLB miss), an exception is taken and software refills the TLB from the page table resident in memory. Software can write over a selected TLB entry or use a hardware mechanism to write into a random entry.

The 4Kc core implements a TLB write compare mechanism to ensure that multiple TLB matches do not occur. On the TLB write operation, the write value is compared with all other entries in the TLB. If a match occurs, the 4Kc core takes a machine check exception, sets the TS bit in the CP0 *Status* register, and aborts the write operation.

Table 4 shows the address bits used for even/odd bank selection depending on page size and the relationship between the legal values in the mask register and the selected page size.

Table 4 Mask and Page Size Values

Pagemask[24:13]	Page Size	Even/Odd Bank Select Bit
000000000000	4KB	VAddr[12]
000000000011	16KB	VAddr[14]
000000001111	64KB	VAddr[16]
000000111111	256KB	VAddr[18]
000011111111	1MB	VAddr[20]
001111111111	4MB	VAddr[22]
111111111111	16MB	VAddr[24]

TLB Tag and Data Formats

Figure 6 shows the format of a TLB *tag* entry. The entry is divided into the follow fields:

- Global process indicator
- Address space identifier
- Virtual page number
- Compressed page mask

Setting the global process indicator (G bit) indicates that the entry is global to all processes and/or threads in the system. In this case, the 8-bit ASID value is ignored since the entry is not relative to a specific thread or process.

The address space identifier (ASID) helps to reduce the frequency of TLB flushing on a context switch. The existence of the ASID allows multiple processes to exist in both the TLB and instruction caches. The current ASID value is stored in the *EntryHi* register and is compared to the ASID value of each entry. Figure 6 and Table 5 show the TLB tag entry format.

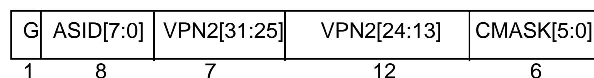


Figure 6 TLB Tag Entry Format

Table 5 TLB Tag Entry Fields

Field Name	Description
G	Global Bit. When set, indicates that this entry is global to all processes and/or threads and thus disables inclusion of the ASID in the comparison.
ASID[7:0]	Address Space Identifier. Identifies with which process or thread this TLB entry is associated.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 5 TLB Tag Entry Fields (Continued)

Field Name	Description
VPN2[31:25], VPN2[24:13]	Virtual Page Number divided by 2. This field contains the upper bits of the virtual page number. Because it represents a pair of TLB pages, it is divided by 2. Bits 31:25 are always included in the TLB lookup comparison. Bits 24:13 are included depending on the page size.
CMASK[5:0]	Compressed page mask value. This field is a compressed version of the page mask. It defines the page size by masking the appropriate VPN2 bits from being involved comparison. It is also used to determine which address bit is used to make the even-odd page determination.

Figure 7 and Table 6 show the TLB data array entry format.

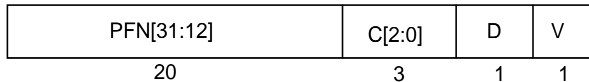


Figure 7 TLB Data Array Entry Format

Table 6 TLB Data Array Entry Fields

<table>
<tr><th>Field Name</th><th>Description</th></tr>
<tr><td>PFN[31:12]</td><td>Physical Frame Number. Defines the upper bits of the physical address. For page sizes larger than 4KB, only a subset of these bits is actually used.</td></tr>
<tr><td>C[2:0]</td><td>Cacheability. Contains an encoded value of the cacheability attributes and determines whether the page should be placed in the cache or not. The field is encoded as follows:
<table>
<tr><th>CS[2:0]</th><th>Coherency Attribute</th></tr>
<tr><td>000*</td><td>Maps to entry 011b.</td></tr>
<tr><td>001*</td><td>Maps to entry 011b.</td></tr>
<tr><td>010</td><td>Uncached</td></tr>
<tr><td>011</td><td>Cacheable, noncoherent, write through, no write allocate</td></tr>
<tr><td>100*</td><td>Maps to entry 011b.</td></tr>
<tr><td>101*</td><td>Maps to entry 011.b</td></tr>
<tr><td>110*</td><td>Maps to entry 011b.</td></tr>
<tr><td>111*</td><td>Maps to entry 010b</td></tr>
</table>
*Values 2 and 3 are the required MIPS32 mappings for uncached and cacheable references; other values may have different meanings in other MIPS32 processors.</td></tr>
</table>

Table 6 TLB Data Array Entry Fields (Continued)

Field Name	Description
D	"Dirty" or write-enable bit. Indicates that the page has been written and/or is writable. If this bit is set, stores to the page are permitted. If the bit is cleared, stores to the page cause a TLB Modified exception.
V	Valid bit. Indicates that the TLB entry, and thus the virtual page mapping, are valid. If this bit is set, accesses to the page are permitted. If the bit is cleared, accesses to the page cause a TLB Invalid exception.

Page Sizes and Replacement Algorithm

To assist in controlling both the amount of mapped space and the replacement characteristics of various memory regions, the 4Kc core provides two mechanisms. First, the page size can be configured, on a per-entry basis, to map a page size of 4 Kbytes to 16 Mbytes (in multiples of 4). The CP0 *PageMask* register is loaded with the mapping page size, which is then entered into the TLB when a new entry is written. Thus, operating systems can provide special purpose maps. For example, a typical frame buffer can be memory mapped with only one TLB entry.

The second mechanism controls the replacement algorithm when a TLB miss occurs. To select a TLB entry to be written with a new mapping, the 4Kc core provides a random replacement algorithm. However, the processor also provides a mechanism where a programmable number of mappings can be locked into the TLB via the *Wired* register, thus avoiding random replacement.

Cache Controllers

The 4Kc core instruction and data cache controllers support caches of various sizes, organizations, and set-associativity. For example, the data cache can be 2 Kbytes in size and 2-way set associative, while the instruction cache can be 8 Kbytes in size and 4-way set associative. Each cache can each be accessed in a single processor cycle. In addition, each cache has its own 32-bit data path and both caches can be accessed in the same pipeline clock cycle. Refer to the section entitled "4Kc Core Optional Logic Blocks" on page 12 for more information on instruction and data cache organization.

The cache controllers also have built-in support for replacing one way of the cache with a scratchpad RAM.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

See the section entitled "4Kc Core Optional Logic Blocks" on page 12 for more information on scratchpad RAMs.

Bus Interface (BIU)

The Bus Interface Unit (BIU) controls the external interface signals. Additionally, it contains the implementation of the 32-byte collapsing write buffer. The purpose of this buffer is to store and combine write transactions before issuing them at the external interface. Since the 4Kc core caches follow a write-through cache policy, the write buffer significantly reduces the number of writes transactions on the external interface and reduces the amount of stalling in the core due to issuance of multiple writes in a short period of time.

The write buffer is organized as two 16-byte buffers. Each buffer contains data from a single 16-byte aligned block of memory. One buffer contains the data currently being transferred on the external interface, while the other buffer contains accumulating data from the core. Data from the accumulation buffer is transferred to the external interface buffer under one of these conditions:

- When a store is attempted from the core to a different 16-byte block than is currently being accumulated
- SYNC Instruction
- Store to an invalid merge pattern
- Any load or store to uncached memory
- A load to the line being merged

Note that if the data in the external interface buffer has not been written out to memory, the core is stalled until the memory write completes. After completion of the memory write, accumulated buffer data can be written to the external interface buffer.

Merge Pattern Control

The 4Kc core implements two 16-byte collapsing write buffers that allow byte, halfword, tri-byte, or word writes from the core to be accumulated in the buffer into a 16-byte value before bursting the data out onto the bus in word format. Note that writes to uncached areas are never merged.

The 4Kc core provides two options for merge pattern control:

- No merge
- Full merge

In *No Merge* mode, writes to a different word within the same line are accumulated in the buffer. Writes to the same word cause the previous word to be driven onto the bus.

In *Full Merge* mode, all combinations of writes to the same line are collected in the buffer. Any pattern of byte enables is possible.

SimpleBE Mode

To aid in attaching the 4Kc core to existing busses, there is a mode that only generates “simple” byte enables. Only byte enables representing naturally aligned byte, half, and word transactions will be generated. Legal byte enable patterns are shown in Table 7. Writes with illegal byte enable patterns will be broken into two separate write transactions. This splitting is independent of the merge pattern control in the write buffer. The only case where a read can generate illegal byte enables is on an uncached tri-byte load (LWL/LWR). These reads will be converted into a word read on the bus.

Table 7 Valid SimpleBE Byte Enable Patterns

EB_BE[3:0]
0001
0010
0100
1000
0011
1100
1111

4Kc Core Reset

The 4Kc core has two types of reset input signals: *Reset* and *ColdReset*.

The *ColdReset* signal must be asserted on either a power-on reset or a cold reset. In a typical application, a power-on reset occurs when the machine is first turned on. A cold reset (also called a hard reset) typically occurs when the machine is already on and the system is rebooted. A cold reset completely initializes the internal state machines of the 4Kc core without saving any state information. The *Reset* and *ColdReset* signals work in conjunction with one

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

another to determine the type of reset operation (see Table 8).

Table 8 4Kc Reset Types

Reset	ColdReset	Action
0	0	Normal Operation, no reset.
1	0	Warm or Soft reset.
X	1	Cold or Hard reset.

The *Reset* signal is asserted for a warm reset. A warm reset restarts the 4Kc core and preserves more of the processors internal state than a cold reset. The *Reset* signal can be asserted synchronously or asynchronously during a cold reset, or synchronously to initiate a warm reset. The assertion of *Reset* causes a soft reset exception within the 4Kc core. In debug mode, EJTAG can request that the soft reset function be masked. It is system dependent whether this functionality is supported. In normal mode, the soft reset cannot be masked.

Power Management

The 4Kc core offers a number of power management features, including low-power design, active power management, and power-down modes of operation. The 4Kc core is a static design that supports slowing or halting the clocks, which reduces system power consumption during idle periods.

The 4Kc core provides two mechanisms for system-level low power support:

- Register-controlled power management
- Instruction-controlled power management

Register-Controlled Power Management

The RP bit in the CP0 Status register provides a software mechanism for placing the system into a low power state. The state of the RP bit is available externally via the *SI_RP* signal. The external agent then decides whether to place the device in low power mode, such as by reducing the system clock frequency.

Three additional bits, $Status_{EXL}$, $Status_{ERL}$, and $Debug_{DM}$ support the power management function by allowing the user to change the power state if an exception or error occurs while the 4Kc core is in a low power state. Depending on what type of exception is taken, one of these three bits will be asserted and reflected on the *SI_EXL*, *SI_ERL*, or *EJ_DebugM* outputs. The external agent can look at these signals and determine whether to leave the low power state to service the exception.

The following 4 power-down signals are part of the system interface and change state as the corresponding bits in the CP0 registers are set or cleared:

- The *SI_RP* signal represents the state of the RP bit (27) in the CP0 Status register.
- The *SI_EXL* signal represents the state of the EXL bit (1) in the CP0 Status register.
- The *SI_ERL* signal represents the state of the ERL bit (2) in the CP0 Status register.
- The *EJ_DebugM* signal represents the state of the DM bit (30) in the CP0 Debug register.

Instruction-Controlled Power Management

The second mechanism for invoking power-down mode is through execution of the WAIT instruction. When the WAIT instruction is executed, the internal clock is suspended. However, the internal timer and some of the input pins (*SI_Int[5:0]*, *SI_NMI*, *SI_Reset*, and *SI_ColdReset*) continue to run. Once the CPU is in instruction-controlled power management mode, any interrupt, NMI, or reset condition causes the CPU to exit this mode and resume normal operation.

The 4Kc core asserts the *SI_SLEEP* signal, which is part of the system interface bus, whenever the WAIT instruction is executed. The assertion of *SI_SLEEP* indicates that the clock has stopped and the 4Kc core is waiting for an interrupt.

4Kc Core Optional Logic Blocks

The 4Kc core consists of the following optional logic blocks as shown in the block diagram in Figure 1.

Instruction Cache

The instruction cache is an optional on-chip memory block of up to 16 Kbytes. Because the instruction cache is virtually indexed, the virtual-to-physical address translation occurs in parallel with the cache access rather than having to wait for the physical address translation. The tag holds 22 bits of physical address, 4 valid bits, a lock bit, and the fill replacement bit.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

The instruction cache block also contains and manages the instruction line fill buffer. Besides accumulating data to be written to the cache, instruction fetches that reference data in the line fill buffer are serviced either by a bypass of that data, or data coming from the external interface. The instruction cache control logic controls the bypass function.

The core supports instruction-cache locking. Cache locking allows critical code or data segments to be locked into the cache on a "per-line" basis, enabling the system programmer to maximize the efficiency of the system cache.

The cache-locking function is always available on all instruction-cache entries. Entries can then be marked as locked or unlocked on a per entry basis using the CACHE instruction.

Data Cache

The data cache is an optional on-chip memory block of up to 16 Kbytes. This virtually indexed, physically tagged cache is protected. Because the data cache is virtually indexed, the virtual-to-physical address translation occurs in parallel with the cache access. The tag holds 22 bits of physical address, 4 valid bits, a lock bit, and the fill replacement bit.

In addition to instruction-cache locking, the 4Kc core also supports a data-cache locking mechanism identical to the instruction cache. Critical data segments are locked into the cache on a "per-line" basis. The locked contents can be updated on a store hit, but cannot be selected for replacement on a cache miss.

The cache-locking function is always available on all data cache entries. Entries can then be marked as locked or unlocked on a per-entry basis using the CACHE instruction.

Cache Memory Configuration

The 4Kc core incorporates on-chip instruction and data caches that can each be accessed in a single processor cycle. Each cache has its own 32-bit data path and can be accessed in the same pipeline clock cycle. Table 9 lists the 4Kc core instruction and data cache attributes.

Table 9 4Kc Core Instruction and Data Cache Attributes

Parameter	Instruction	Data
Size	0 - 16 Kbytes	0 - 16 Kbytes
Organization	1 - 4 way set associative	1 - 4 way set associative
Line Size	16 bytes	16 bytes
Read Unit	32 bits	32 bits
Write Policy	na	write-through *without write-allocate*
Miss restart after transfer of	miss word	miss word
Cache Locking	per line	per line

Cache Protocols

The 4Kc core supports the following cache protocols:

- **Uncached:** Addresses in a memory area indicated as uncached are not read from the cache. Stores to such addresses are written directly to main memory, without changing cache contents.
- **Write-through:** Loads and instruction fetches first search the cache, reading main memory only if the desired data does not reside in the cache. On data store operations, the cache is first searched to see if the target address is cache resident. If it is resident, the cache contents are updated, and main memory is also written. If the cache look-up misses, only main memory is written.

Scratchpad RAM

The 4Kc core also supports replacing up to one way of each cache with a scratchpad RAM. The scratchpad RAM is user-defined and can consist of a variety of devices. The main requirement is that it must be accessible with timing similar to a regular cache RAM. This means that an index will be driven one cycle, a tag will be driven the following clock, and the scratchpad must return a hit signal and the data in the second clock. The scratchpad can thus easily contain a large RAM/ROM or memory-mapped registers.

The core's interface to a scratchpad RAM is slightly different than to a regular cache RAM. Additional index bits allow access to a larger array, 1MB of scratchpad RAM

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

versus 4KB for a cache way. These bits come from the virtual address, so on a 4Kc core care must be taken to avoid virtual aliasing.The core does not automatically refill the scratchpad way and will not select it for replacement on cache misses. Additionally, stores that hit in the scratchpad will not generate write-throughs to main memory.

EJTAG Debug Support

The 4Kc core provides for an optional Enhanced JTAG (EJTAG) interface for use in the software debug of application and kernel code. In addition to standard user mode and kernel modes of operation, the 4Kc core provides a Debug mode that is entered after a debug exception (derived from a hardware breakpoint, single-step exception, etc.) is taken and continues until a debug exception return (DERET) instruction is executed. During this time, the processor executes the debug exception handler routine.

Refer to the section called "4Kc Core Signal Descriptions" on page 19 for a list of signals EJTAG interface signals.

The EJTAG interface operates through the Test Access Port (TAP), a serial communication port used for transferring test data in and out of the 4Kc core. In addition to the standard JTAG instructions, special instructions defined in the EJTAG specification define what registers are selected and how they are used.

Debug Registers

Three debug registers (DEBUG, DEPC, and DESAVE) have been added to the MIPS Coprocessor 0 (CP0) register set. The DEBUG register shows the cause of the debug exception and is used for the setting up of single-step operations. The DEPC, or Debug Exception Program Counter, register holds the address on which the debug exception was taken. This is used to resume program execution after the debug operation finishes. Finally, the DESAVE, or Debug Exception Save, register enables the saving of general-purpose registers used during execution of the debug exception handler.

To exit debug mode, a Debug Exception Return (DERET) instruction is executed. When this instruction is executed, the system exits debug mode, allowing normal execution of application and system code to resume.

EJTAG Hardware Breakpoints

There are several types of simple hardware breakpoints defined in the EJTAG specification. These stop the normal operation of the CPU and force the system into debug mode. There are two types of simple hardware breakpoints implemented in the 4Kc core: Instruction breakpoints and Data breakpoints.

The 4Kc core can be configured with the following breakpoint options:

- No data or instruction breakpoints
- One data and two instruction breakpoints
- Two data and four instruction breakpoints

Instruction breaks occur on instruction fetch operations, and the break is set on the virtual address on the bus between the CPU and the instruction cache. Instruction breaks can also be made on the ASID value used by the MMU. Finally, a mask can be applied to the virtual address to set breakpoints on a range of instructions.

Data breakpoints occur on load/store transactions. Breakpoints are set on virtual address and ASID values, similar to the Instruction breakpoint. Data breakpoints can be set on a load, a store, or both. Data breakpoints can also be set based on the value of the load/store operation. Finally, masks can be applied to both the virtual address and the load/store value.

4Kc Core Instructions

The 4Kc core instruction set complies with the MIPS32 instruction set architecture. Table 10 provides a summary of instructions implemented by the 4Kc core.

Table 10 4Kc Core Instruction Set

Instruction	Description	Function
ADD	Integer Add	`Rd = Rs + Rt`
ADDI	Integer Add Immediate	`Rt = Rs + Immed`
ADDIU	Unsigned Integer Add Immediate	`Rt = Rs` $+_U$ `Immed`
ADDU	Unsigned Integer Add	`Rd = Rs` $+_U$ `Rt`

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 10 4Kc Core Instruction Set (Continued)

Instruction	Description	Function
AND	Logical AND	Rd = Rs & Rt
ANDI	Logical AND Immediate	Rt = Rs & (0_{16} || Immed)
BEQ	Branch On Equal	if Rs == Rt PC += (int)offset
BEQL	Branch On Equal Likely	if Rs == Rt PC += (int)offset else Ignore Next Instruction
BGEZ	Branch on Greater Than or Equal To Zero	if !Rs[31] PC += (int)offset
BGEZAL	Branch on Greater Than or Equal To Zero And Link	GPR[31] = PC + 8 if !Rs[31] PC += (int)offset
BGEZALL	Branch on Greater Than or Equal To Zero And Link Likely	GPR[31] = PC + 8 if !Rs[31] PC += (int)offset else Ignore Next Instruction
BGEZL	Branch on Greater Than or Equal To Zero Likely	if !Rs[31] PC += (int)offset else Ignore Next Instruction
BGTZ	Branch on Greater Than Zero	if !Rs[31] && Rs != 0 PC += (int)offset
BGTZL	Branch on Greater Than Zero Likely	if !Rs[31] && Rs != 0 PC += (int)offset else Ignore Next Instruction
BLEZ	Branch on Less Than or Equal to Zero	if Rs[31] || Rs == 0 PC += (int)offset
BLEZL	Branch on Less Than or Equal to Zero Likely	if Rs[31] || Rs == 0 PC += (int)offset else Ignore Next Instruction
BLTZ	Branch on Less Than Zero	if Rs[31] PC += (int)offset
BLTZAL	Branch on Less Than Zero And Link	GPR[31] = PC + 8 if Rs[31] PC += (int)offset
BLTZALL	Branch on Less Than Zero And Link Likely	GPR[31] = PC + 8 if Rs[31] PC += (int)offset else Ignore Next Instruction
BLTZL	Branch on Less Than Zero Likely	if Rs[31] PC += (int)offset else Ignore Next Instruction

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 10 4Kc Core Instruction Set (Continued)

<table>
<tr><th>Instruction</th><th>Description</th><th>Function</th></tr>
<tr><td>BNE</td><td>Branch on Not Equal</td><td>if Rs != Rt
  PC += (int)offset</td></tr>
<tr><td>BNEL</td><td>Branch on Not Equal Likely</td><td>if Rs != Rt
  PC += (int)offset
else
  Ignore Next Instruction</td></tr>
<tr><td>BREAK</td><td>Breakpoint</td><td>Break Exception</td></tr>
<tr><td>CACHE</td><td>Cache Operation</td><td>See Software User’s Manual</td></tr>
<tr><td>COP0</td><td>Coprocessor 0 Operation</td><td>See Software User’s Manual</td></tr>
<tr><td>CLO</td><td>Count Leading Ones</td><td>Rd = NumLeadingOnes(Rs)</td></tr>
<tr><td>CLZ</td><td>Count Leading Zeroes</td><td>Rd = NumLeadingZeroes(Rs)</td></tr>
<tr><td>DERET</td><td>Return from Debug Exception</td><td>PC = DEPC
Exit Debug Mode</td></tr>
<tr><td>DIV</td><td>Divide</td><td>LO = (int)Rs / (int)Rt
HI = (int)Rs % (int)Rt</td></tr>
<tr><td>DIVU</td><td>Unsigned Divide</td><td>LO = (uns)Rs / (uns)Rt
HI = (uns)Rs % (uns)Rt</td></tr>
<tr><td>ERET</td><td>Return from Exception</td><td>if SR[2]
  PC = ErrorEPC
else
  PC = EPC
  SR[1] = 0
SR[2] = 0
LL = 0</td></tr>
<tr><td>J</td><td>Unconditional Jump</td><td>PC = PC[31:28] || offset<<2</td></tr>
<tr><td>JAL</td><td>Jump and Link</td><td>GPR[31] = PC + 8
PC = PC[31:28] || offset<<2</td></tr>
<tr><td>JALR</td><td>Jump and Link Register</td><td>Rd = PC + 8
PC = Rs</td></tr>
<tr><td>JR</td><td>Jump Register</td><td>PC = Rs</td></tr>
<tr><td>LB</td><td>Load Byte</td><td>Rt = (byte)Mem[Rs+offset]</td></tr>
<tr><td>LBU</td><td>Unsigned Load Byte</td><td>Rt = (ubyte))Mem[Rs+offset]</td></tr>
<tr><td>LH</td><td>Load Halfword</td><td>Rt = (half)Mem[Rs+offset]</td></tr>
<tr><td>LHU</td><td>Unsigned Load Halfword</td><td>Rt = (uhalf)Mem[Rs+offset]</td></tr>
<tr><td>LL</td><td>Load Linked Word</td><td>Rt = Mem[Rs+offset]
LL = 1
LLAdr = Rs + offset</td></tr>
<tr><td>LUI</td><td>Load Upper Immediate</td><td>Rt = immediate << 16</td></tr>
<tr><td>LW</td><td>Load Word</td><td>Rt = Mem[Rs+offset]</td></tr>
<tr><td>LWL</td><td>Load Word Left</td><td>See Software User’s Manual</td></tr>
</table>

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 10 4Kc Core Instruction Set (Continued)

Instruction	Description	Function
LWR	Load Word Right	See Software User's Manual
MADD	Multiply-Add	HI | LO += (int)Rs * (int)Rt
MADDU	Multiply-Add Unsigned	HI | LO += (uns)Rs * (uns)Rt
MFC0	Move From Coprocessor 0	Rt = CPR[0, n, sel] = Rt
MFHI	Move From HI	Rd = HI
MFLO	Move From LO	Rd = LO
MOVN	Move Conditional on Not Zero	if Rt ≠ 0 then Rd = Rs
MOVZ	Move Conditional on Zero	if Rt = 0 then Rd = Rs
MSUB	Multiply-Subtract	HI | LO -= (int)Rs * (int)Rt
MSUBU	Multiply-Subtract Unsigned	HI | LO -= (uns)Rs * (uns)Rt
MTC0	Move To Coprocessor 0	CPR[0, n, SEL] = Rt
MTHI	Move To HI	HI = Rs
MTLO	Move To LO	LO = Rs
MUL	Multiply with register write	HI | LO =Unpredictable Rd = ((int)Rs * (int)Rt)$_{31..0}$
MULT	Integer Multiply	HI | LO = (int)Rs * (int)Rd
MULTU	Unsigned Multiply	HI | LO = (uns)Rs * (uns)Rd
NOR	Logical NOR	Rd = ~(Rs | Rt)
OR	Logical OR	Rd = Rs | Rt
ORI	Logical OR Immediate	Rt = Rs | Immed
PREF	Prefetch	Load Specified Line into Cache
SB	Store Byte	(byte)Mem[Rs+offset] = Rt
SC	Store Conditional Word	if LL = 1 mem[Rs+offset] = Rt Rt = LL
SDBBP	Software Debug Break Point	Trap to SW Debug Handler
SH	Store Half	(half)Mem[Rs+offset] = Rt
SLL	Shift Left Logical	Rd = Rt << sa
SLLV	Shift Left Logical Variable	Rd = Rt << Rs[4:0]

SLT	Set on Less Than	if (int)Rs < (int)Rt
		Rd = 1
		else
		Rd = 0

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 10 4Kc Core Instruction Set (Continued)

Instruction	Description	Function

SLTI	Set on Less Than Immediate	`if (int)Rs < (int)Immed`
		`  Rt = 1`
		`else`
		`  Rt = 0`

SLTIU	Set on Less Than Immediate Unsigned	`if (uns)Rs < (uns)Immed`
		`  Rt = 1`
		`else`
		`  Rt = 0`

SLTU	Set on Less Than Unsigned	`if (uns)Rs < (uns)Immed`
		`  Rd = 1`
		`else`
		`  Rd = 0`

SRA	Shift Right Arithmetic	`Rd = (int)Rt >> sa`
SRAV	Shift Right Arithmetic Variable	`Rd = (int)Rt >> Rs[4:0]`
SRL	Shift Right Logical	`Rd = (uns)Rt >> sa`
SRLV	Shift Right Logical Variable	`Rd = (uns)Rt >> Rs[4:0]`
SSNOP	Superscalar Inhibit No Operation	`NOP`
SUB	Integer Subtract	`Rt = (int)Rs - (int)Rd`
SUBU	Unsigned Subtract	`Rt = (uns)Rs - (uns)Rd`
SW	Store Word	`Mem[Rs+offset] = Rt`
SWL	Store Word Left	See Software User's Manual
SWR	Store Word Right	See Software User's Manual
SYNC	Synchronize	See Software User's Manual
SYSCALL	System Call	`SystemCallException`
TEQ	Trap if Equal	`if Rs == Rt` `  TrapException`
TEQI	Trap if Equal Immediate	`if Rs == (int)Immed` `  TrapException`
TGE	Trap if Greater Than or Equal	`if (int)Rs >= (int)Rt` `  TrapException`
TGEI	Trap if Greater Than or Equal Immediate	`if (int)Rs >= (int)Immed` `  TrapException`
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	`if (uns)Rs >= (uns)Immed` `  TrapException`
TGEU	Trap if Greater Than or Equal Unsigned	`if (uns)Rs >= (uns)Rt` `  TrapException`
TLBWI	Write Indexed TLB Entry	`See Software Users Manual`
TLBWR	Write Random TLB Entry	`See Software Users Manual`
TLBP	Probe TLB for Matching Entry	`See Software Users Manual`

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 10 4Kc Core Instruction Set (Continued)

Instruction	Description	Function
TLBR	Read Index for TLB Entry	`See Software Users Manual`
TLT	Trap if Less Than	`if (int)Rs < (int)Rt` `TrapException`
TLTI	Trap if Less Than Immediate	`if (int)Rs < (int)Immed` `TrapException`
TLTIU	Trap if Less Than Immediate Unsigned	`if (uns)Rs < (uns)Immed` `TrapException`
TLTU	Trap if Less Than Unsigned	`if (uns)Rs < (uns)Rt` `TrapException`
TNE	Trap if Not Equal	`if Rs != Rt` `TrapException`
TNEI	Trap if Not Equal Immediate	`if Rs != (int)Immed` `TrapException`
WAIT	Wait for Interrupts	Stall until interrupt occurs
XOR	Exclusive OR	`Rd = Rs ^ Rt`
XORI	Exclusive OR Immediate	`Rt = Rs ^ (uns)Immed`

4Kc Core Signal Descriptions

This section describes the signal interface of the 4Kc microprocessor core.

The pin direction key for the signal descriptions is shown in Table 11 below.

Table 11 4Kc Core Signal Direction Key

Dir	Description
I	Input to the 4Kc core sampled on the rising edge of the appropriate CLK signal.
O	Output of the 4Kc core, unless otherwise noted, driven at the rising edge of the appropriate CLK signal.
A	Asynchronous inputs that are synchronized by the core.
S	Static input to the 4Kc core. These signals are normally tied to either power or ground and should not change state while *SI_ColdReset* is deasserted.

The 4Kc core signals are listed in Table 12 below. Note that the signals are grouped by logical function, not by expected physical location. All signals, with the exception of *EJ_TRST_N*, are active-high signals. *EJ_DINT* and *SI_NMI* go through edge-detection logic so that only one exception is taken each time they are asserted.

Table 12 4Kc Signal Descriptions

Signal Name	Type	Description
System Interface		
Clock Signals:		
SI_ClkIn	I	Clock Input. All inputs and outputs, except a few of the EJTAG signals, are sampled and/or asserted relative to the rising edge of this signal.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 12 4Kc Signal Descriptions

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SI_ClkOut</td><td>O</td><td>Reference Clock for the External Bus Interface. This clock signal provides a reference for deskewing any clock insertion delay created by the internal clock buffering in the core.</td></tr>
<tr><td colspan="3">Reset Signals:</td></tr>
<tr><td>SI_ColdReset</td><td>A</td><td>Hard/Cold Reset Signal. Causes a Reset Exception in the core.</td></tr>
<tr><td>SI_NMI</td><td>A</td><td>Non-Maskable Interrupt. An edge detect is used on this signal. When this signal is sampled asserted (high) one clock after being sampled deasserted, an NMI is posted to the core.</td></tr>
<tr><td>SI_Reset</td><td>A</td><td>Soft/Warm Reset Signal. Causes a SoftReset Exception in the core.</td></tr>
<tr><td colspan="3">Power Management Signals:</td></tr>
<tr><td>SI_ERL</td><td>O</td><td>This signal represents the state of the ERL bit (2) in the CP0 Status register and indicates the error level. The core asserts SI_ERL whenever a Reset, Soft Reset, or NMI exception is taken.</td></tr>
<tr><td>SI_EXL</td><td>O</td><td>This signal represents the state of the EXL bit (1) in the CP0 Status register and indicates the exception level. The core asserts SI_EXL whenever any exception other than a Reset, Soft Reset, NMI, or Debug exception is taken.</td></tr>
<tr><td>SI_RP</td><td>O</td><td>This signal represents the state of the RP bit (27) in the CP0 Status register. Software can write this bit to indicate that the device can enter a reduced power mode.</td></tr>
<tr><td>SI_SLEEP</td><td>O</td><td>This signal is asserted by the core whenever the WAIT instruction is executed. The assertion of this signal indicates that the clock has stopped and that the core is waiting for an interrupt.</td></tr>
<tr><td colspan="3">Interrupt Signals:</td></tr>
<tr><td>SI_Int[5:0]</td><td>A</td><td>Active-high Interrupt Pins. These signals are driven by external logic and, when asserted, indicate the corresponding interrupt exception to the core. These signals go through synchronization logic and can be asserted asynchronously to SI_ClkIn.</td></tr>
<tr><td>SI_TimerInt</td><td>O</td><td>This signal is asserted whenever the Count and Compare registers match and is deasserted when the Compare register is written. In order to have timer interrupts, this signal needs to be brought back into the 4K core on one of the 6 SI_Int interrupt pins. Traditionally, this has been accomplished via muxing SI_TimerInt with SI_Int[5]. Exposing SI_TimerInt as an output allows more flexibility for the system designer. Timer interrupts can be muxed or ORed into one of the interrupts, as desired in a particular system. In a complex system, it could even be fed into a priority encoder to allow SI_Int[5:0] to map up to 63 interrupt sources.</td></tr>
<tr><td colspan="3">Configuration Inputs:</td></tr>
<tr><td>SI_Endian</td><td>S</td><td>Indicates the base endianess of the core.
<table><tr><th>EB_Endian</th><th>Base Endian Mode</th></tr><tr><td>0</td><td>Little Endian</td></tr><tr><td>1</td><td>Big Endian</td></tr></table></td></tr>
</table>

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 12 4Kc Signal Descriptions

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>SI_MergeMode[1:0]</td><td>S</td><td>The state of these signals determines the merge mode for the 16-byte collapsing write buffer.
<table>
<tr><th>Encoding</th><th>Merge Mode</th></tr>
<tr><td>00_2</td><td>No Merge</td></tr>
<tr><td>01_2</td><td>Reserved</td></tr>
<tr><td>10_2</td><td>Full Merge</td></tr>
<tr><td>11_2</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>SI_SimpleBE[1:0]</td><td>S</td><td>The state of these signals can constrain the core to only generate certain byte enables on EC™ interface transactions. This eases connection to some existing bus standards.
<table>
<tr><th>SI_SimpleBE[1:0]</th><th>Byte Enable Mode</th></tr>
<tr><td>00_2</td><td>All BEs allowed</td></tr>
<tr><td>01_2</td><td>Naturally aligned bytes, half-words, and words only</td></tr>
<tr><td>10_2</td><td>Reserved</td></tr>
<tr><td>11_2</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td colspan="3">External Bus Interface</td></tr>
<tr><td>EB_ARdy</td><td>I</td><td>Indicates whether the target is ready for a new address. The core will not complete the address phase of a new bus transaction until the clock cycle after EB_ARdy is sampled asserted.</td></tr>
<tr><td>EB_AValid</td><td>O</td><td>When asserted, indicates that the values on the address bus and access types lines are valid, signifying the beginning of a new bus transaction. EB_AValid must always be valid.</td></tr>
<tr><td>EB_Instr</td><td>O</td><td>When asserted, indicates that the transaction is an instruction fetch versus a data reference. EB_Instr is only valid when EB_AValid is asserted.</td></tr>
<tr><td>EB_Write</td><td>O</td><td>When asserted, indicates that the current transaction is a write. This signal is only valid when EB_AValid is asserted.</td></tr>
<tr><td>EB_Burst</td><td>O</td><td>When asserted, indicates that the current transaction is part of a cache fill or a write burst. Note that there is redundant information contained in EB_Burst, EB_BFirst, EB_BLast, and EB_BLen. This is done to simplify the system design—the information can be used in whatever form is easiest.</td></tr>
<tr><td>EB_BFirst</td><td>O</td><td>When asserted, indicates the beginning of the burst. EB_BFirst is always valid.</td></tr>
<tr><td>EB_BLast</td><td>O</td><td>When asserted, indicates the end of the burst. EB_BLast is always valid.</td></tr>
<tr><td>EB_BLen<1:0></td><td>O</td><td>Indicates the length of the burst. This signal is only valid when EB_AValid is asserted.
<table>
<tr><th>EB_BLength<1:0></th><th>Burst Length</th></tr>
<tr><td>0</td><td>reserved</td></tr>
<tr><td>1</td><td>4</td></tr>
<tr><td>2</td><td>reserved</td></tr>
<tr><td>3</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>EB_SBlock</td><td>SI</td><td>When sampled asserted, sub-block ordering is used. When sampled deasserted, sequential addressing is used.</td></tr>
</table>

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 12 4Kc Signal Descriptions

<table>
<tr><th>Signal Name</th><th>Type</th><th>Description</th></tr>
<tr><td>EB_BE<3:0></td><td>O</td><td>Indicates which bytes of the EB_RData or EB_WData buses are involved in the current transaction. If an EB_BE signal is asserted, the associated byte is being read or written. EB_BE lines are only valid while EB_AValid is asserted.
<table>
<tr><th>EB_BE Signal</th><th>Read Data Bits Sampled</th><th>Write Data Bits Driven Valid</th></tr>
<tr><td>EB_BE<0></td><td>EB_RData<7:0></td><td>EB_WData<7:0></td></tr>
<tr><td>EB_BE<1></td><td>EB_RData<15:8></td><td>EB_WData<15:8></td></tr>
<tr><td>EB_BE<2></td><td>EB_RData<23:16></td><td>EB_WData<23:16></td></tr>
<tr><td>EB_BE<3></td><td>EB_RData<31:24></td><td>EB_WData<31:24></td></tr>
</table></td></tr>
<tr><td>EB_A<35:2></td><td>O</td><td>Address lines for external bus. Only valid when EB_AValid is asserted. EB_A[35:32] are tied to 0 in this core.</td></tr>
<tr><td>EB_WData<31:0></td><td>O</td><td>Output data for writes.</td></tr>
<tr><td>EB_RData<31:0></td><td>I</td><td>Input Data for reads.</td></tr>
<tr><td>EB_RdVal</td><td>I</td><td>Indicates that the target is driving read data on EB_RData lines. EB_RdVal must always be valid. EB_RdVal may never be sampled asserted until the rising edge after the corresponding EB_ARdy was sampled asserted.</td></tr>
<tr><td>EB_WDRdy</td><td>I</td><td>Indicates that the target of a write is ready. The EB_WData lines can change in the next clock cycle. EB_WDRdy will not be sampled until the rising edge where the corresponding EB_ARdy is sampled asserted.</td></tr>
<tr><td>EB_RBErr</td><td>I</td><td>Bus error indicator for read transactions. EB_RBErr is sampled on every rising clock edge until an active sampling of EB_RdVal. EB_RBErr sampled with asserted EB_RdVal indicates a bus error during read. EB_RBErr must be deasserted in idle phases.</td></tr>
<tr><td>EB_WBErr</td><td>I</td><td>Bus error indicator for write transactions. EB_WBErr is sampled on the rising clock edge following an active sample of EB_WDRdy. EB_WBErr must be deasserted in idle phases.</td></tr>
<tr><td>EB_EWBE</td><td>I</td><td>Indicates that any external write buffers are empty. The external write buffers must deassert EB_EWBE in the cycle after the corresponding EB_WDRdy is asserted and keep EB_EWBE deasserted until the external write buffers are empty.</td></tr>
<tr><td>EB_WWBE</td><td>O</td><td>When asserted, indicates that the core is waiting for external write buffers to empty.</td></tr>
<tr><td colspan="3">EJTAG Interface</td></tr>
<tr><td colspan="3">TAP interface. These signals comprise the EJTAG Test Access Port. These signals will not be connected if the core does not implement the TAP controller.</td></tr>
<tr><td>EJ_TRST_N</td><td>I</td><td>Active-low Test Reset Input (TRST*) for the EJTAG TAP. At power-up, the assertion of EJ_TRST_N causes the TAP controller to be reset.</td></tr>
<tr><td>EJ_TCK</td><td>I</td><td>Test Clock Input (TCK) for the EJTAG TAP.</td></tr>
<tr><td>EJ_TMS</td><td>I</td><td>Test Mode Select Input (TMS) for the EJTAG TAP.</td></tr>
<tr><td>EJ_TDI</td><td>I</td><td>Test Data Input (TDI) for the EJTAG TAP.</td></tr>
<tr><td>EJ_TDO</td><td>O</td><td>Test Data Output (TDO) for the EJTAG TAP.</td></tr>
</table>

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 12 4Kc Signal Descriptions

Signal Name	Type	Description
EJ_TDOzstate	O	Drive indication for the output of TDO for the EJTAG TAP at chip level: 1: The TDO output at chip level must be in Z-state 0: The TDO output at chip level must be driven to the value of *EJ_TDO* IEEE Standard 1149.1-1990 defines TDO as a 3-stated signal. To avoid having a 3-state core output, the 4K core outputs this signal to drive an external 3-state buffer.
Debug Interrupt:		
EJ_DINTsup	S	Value of DINTsup for the Implementation register. A 1 on this signal indicates that the EJTAG probe can use the DINT signal to interrupt the processor.
EJ_DINT	I	Debug exception request when this signal is asserted in a CPU clock period after being deasserted in the previous CPU clock period. The request is cleared when debug mode is entered. Requests when in debug mode are ignored.
Debug Mode Indication:		
EJ_DebugM	O	Asserted when the core is in Debug Mode. This can be used to bring the core out of a low power mode. In systems with multiple processor cores, this signal can be used to synchronize the cores when debugging.
Device ID bits:		
These inputs provide an identifying number visible to the EJTAG probe. If the EJTAG TAP controller is not implemented, these inputs are not connected. These inputs are always available for soft core customers. On hard cores, the core "hardener" can set these inputs to their own values.		
EJ_ManufID[10:0]	S	Value of the ManufID[10:0] field in the Device ID register. As per IEEE 1149.1-1990 section 11.2, the manufacturer identity code shall be a compressed form of JEDEC standard manufacturer's identification code in the JEDEC Publications 106, which can be found at: `http://www.jedec.org/` ManufID[6:0] bits are derived from the last byte of the JEDEC code by discarding the parity bit. ManufID[10:7] bits provide a binary count of the number of bytes in the JEDEC code that contain the continuation character (0x7F). Where the number of continuations characters exceeds 15, these 4 bits contain the modulo-16 count of the number of continuation characters.
EJ_PartNumber[15:0]	S	Value of the PartNumber[15:0] field in the Device ID register.
EJ_Version[3:0]	S	Value of the Version[3:0] field in the Device ID register.
System Implementation Dependent Outputs:		
These signals come from EJTAG control registers. They have no effect on the core, but can be used to give EJTAG debugging software additional control over the system.		
EJ_SRstE	O	Soft Reset Enable. EJTAG can deassert this signal if it wants to mask soft resets. If this signal is deasserted, none, some, or all soft reset sources are masked.
EJ_PerRst	O	Peripheral Reset. EJTAG can assert this signal to request the reset of some or all of the peripheral devices in the system.
EJ_PrRst	O	Processor Reset. EJTAG can assert this signal to request that the core be reset. This can be fed into the *SI_Reset* signal.
Performance Monitoring Interface		
These signals can be used to implement performance counters, which can be used to monitor hardware/software performance.		
PM_DCacheHit	O	This signal is asserted whenever there is a data cache hit.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Table 12 4Kc Signal Descriptions

Signal Name	Type	Description
PM_DCacheMiss	O	This signal is asserted whenever there is a data-cache miss.
PM_DTLBHit	O	This signal is asserted whenever there is a hit in the data TLB.
PM_DTLBMiss	O	This signal is asserted whenever there is a miss in the data TLB.
PM_ICacheHit	O	This signal is asserted whenever there is an instruction-cache hit.
PM_ICacheMiss	O	This signal is asserted whenever there is an instruction-cache miss.
PM_InstComplete	O	This signal is asserted each time an instruction completes in the pipeline.
PM_ITLBHit	O	This signal is asserted whenever there is an instruction TLB hit.
PM_ITLBMiss	O	This signal is asserted whenever there is an instruction TLB miss.
PM_JTLBHit	O	This signal is asserted whenever there is a joint TLB hit.
PM_JTLBMiss	O	This signal is asserted whenever there is a joint TLB miss.
PM_WTBMerge	O	This signal is asserted whenever there is a successful merge in the write-through buffer.
PM_WTBNoMerge	O	This signal is asserted whenever a non-merging store is written to the write-through buffer.
Scan Test Interface		
These signals provide the interface for testing the core. The use and configuration of these pins are implementation-dependent.		
ScanEnable	I	This signal should be asserted while scanning vectors into or out of the core. The *ScanEnable* signal must be deasserted during normal operation and during capture clocks in test mode.
ScanMode	I	This signal should be asserted during all scan testing both while scanning and during capture clocks. The *ScanMode* signal must be deasserted during normal operation.
ScanIn<n:0>	I	This signal is input to the scan chain.
ScanOut<n:0>	O	This signal is output from the scan chain.
BistIn<n:0>	I	Input to the BIST controller.
BistOut<n:0>	O	Output from the BIST controller.

4Kc Core Bus Transactions

The 4Kc core implements the EC™ interface for its bus transactions. This interface uses a pipelined, in-order protocol with independent address, read data, and write data buses. The following subsections describe the four basic bus transactions: single read, single write, burst read, and burst write.

Single Read

Figure 8 shows the basic timing relationships of signals during a simple read transaction. During a single read cycle, the 4Kc core drives the address onto *EB_A[35:2]* and byte enable information onto *EB_BE[3:0]*. To maximize performance, the EC interface does not define a maximum number of outstanding bus cycles. Instead it provides the *EB_ARdy* input signal. This signal is driven by external logic and controls the generation of addresses on the bus.

In the core, the address is driven whenever it becomes available, regardless of the state of *EB_ARdy*. However, the core always continues to drive the address until the clock after *EB_ARdy* is sampled asserted. For example, at the rising edge of the clock 2 in Figure 8, the *EB_ARdy* signal is sampled low, indicating that external logic is not ready to accept the new address. However, the 4Kc core still drives *EB_A[35:2]* in this clock as shown. On the rising edge of

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

clock 3, the 4Kc core samples *EB_ARdy* asserted and continues to drive the address until the rising edge of clock 4.

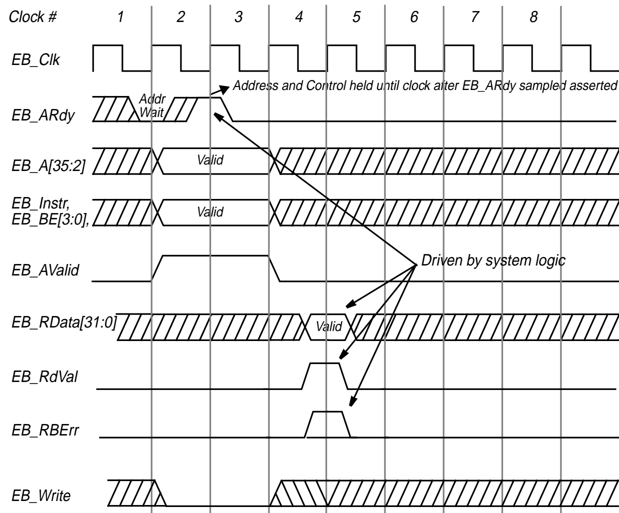


Figure 8 Single Read Transaction Timing Diagram

The *EB_Instr* signal is only asserted during a single read cycle if there is an instruction fetch from non-cacheable memory space. The *EB_AValid* signal is driven in each clock that *EB_A[35:2]* is valid on the bus. The 4Kc core drives *EB_Write* low to indicate a read transaction.

The *EB_RData[31:0]* and *EB_RdVal* signals are first sampled on the rising edge of clock 4, one clock after *EB_ARdy* is sampled asserted. Data is sampled on every clock thereafter until *EB_RdVal* is sampled asserted.

If a bus error occurs during the data transaction, external logic asserts *EB_RBErr* in the same clock as *EB_RdVal*.

Single Write

Figure 9 shows a typical write transaction. The 4Kc core drives address and control information onto the *EB_A[35:2]* and *EB_BE[3:0]* signals on the rising edge of clock 2. As in the single read cycle, these signals remain active until the clock edge after the *EB_ARdy* signal is sampled asserted. The 4Kc core asserts the *EB_Write* signal to indicate that a valid write cycle is on the bus and *EB_AValid* to indicate that valid address is on the bus.

The 4Kc core drives write data onto *EB_WData[31:0]* in the same clock as the address and continues to drive data until the clock edge after the *EB_WDRdy* signal is sampled asserted. If a bus error occurs during a write operation, external logic asserts the *EB_WBErr* signal one clock after asserting *EB_WDRdy.*

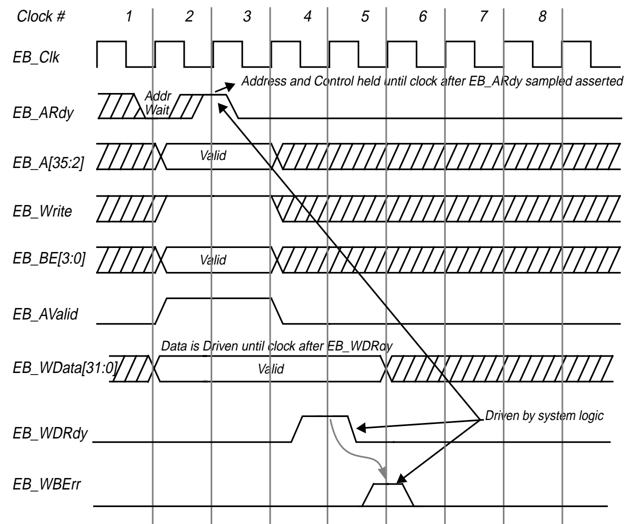


Figure 9 Single Write Transaction Timing Diagram

Burst Read

The 4Kc core is capable of generating burst transactions on the bus. A burst transaction is used to transfer multiple data items in one transaction.

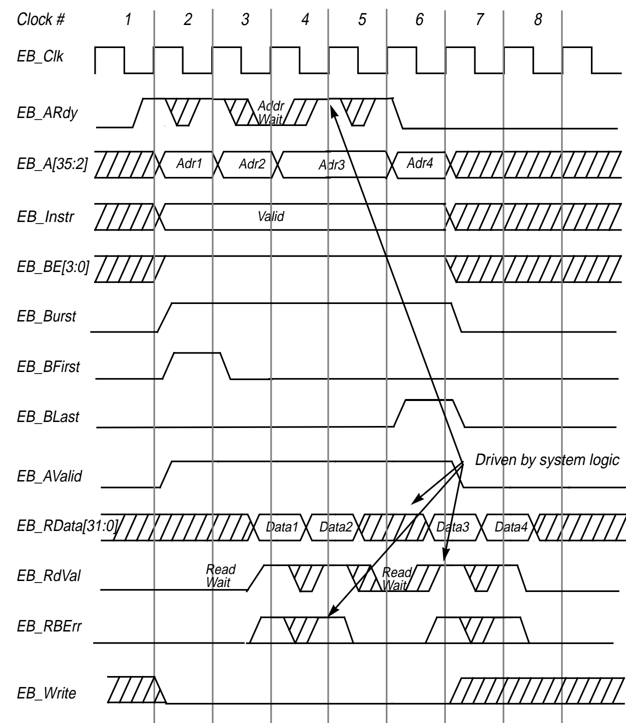


Figure 10 Burst Read Transaction Timing Diagram

Figure 10 shows an example of a burst read transaction. Burst read transactions initiated by the 4Kc core always

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

contain four data transfers in a sequence determined by the critical word (the address that caused the miss) and *EB_SBlock*. In addition, the data requested is always a 16-byte aligned block.

The order of words within this 16-byte block varies depending on which of the words in the block is being requested by the execution unit and the ordering protocol selected. The burst always starts with the word requested by the execution unit and proceeds in either an ascending or descending address order, wrapping when the block boundary is reached. Table 13 and Table 14 show the sequence of address bits 2 and 3.

Table 13 Sequential Ordering Protocols

Starting Address EB_A[3:2]	Address Progression of EB_A[3:2]
00	00, 01, 10, 11
01	01, 10, 11, 00
10	10, 11, 00, 01
11	11, 00, 01, 10

Table 14 Sub-Block Ordering Protocols

Starting Address EB_A[3:2]	Address Progression of EB_A[3:2]
00	00, 01, 10, 11
01	01, 00, 11, 10
10	10, 11, 00, 01
11	11, 10, 01, 00

The core drives address and control information onto the *EB_A[35:2]* and *EB_BE[3:0]* signals on the rising edge of clock 2. As in the single read cycle, these signals remain active until the clock edge after the *EB_ARdy* signal is sampled asserted. The 4Kc core continues to drive *EB_AValid* as long as a valid address is on the bus.

The *EB_Instr* signal is asserted if the burst read is for an instruction fetch. The *EB_Burst* signal is asserted while the address is on the bus to indicate that the current address is part of a burst transaction. The 4Kc core asserts the *EB_BFirst* signal in the same clock as the first address is driven and the *EB_BLast* signal in the same clock as the last address to indicate the start and end of a burst cycle.

The 4Kc core first samples the *EB_RData[31:0]* signals two clocks after *EB_ARDy* is sampled asserted. External logic asserts *EB_RdVal* to indicate that valid data is on the bus. The 4Kc core latches data internally whenever *EB_RVal* is sampled asserted.

Note that on the rising edge of clocks 3 and 6 in Figure 10, the *EB_RdVal* signal is sampled deasserted, causing wait states in the data return. There is also an address wait state caused by *EB_ARdy* being sampled deasserted on the rising edge of clock 4. Note that the core holds address 3 on the *EB_A* bus for an extra clock because of this wait state. External logic asserts the *EB_RBErr* signal in the same clock as data if a bus error occurs during that data transfer.

Burst Write

Burst write transactions are used to empty one of the write buffers. A burst write transaction is only performed if the write buffer contains 16 bytes of data associated with the same aligned memory block, otherwise individual write transactions are performed. Figure 11 shows a timing diagram of a burst write transaction. Unlike the read burst, a write burst always begins with *EB_A[3:2]* equal to 00b.

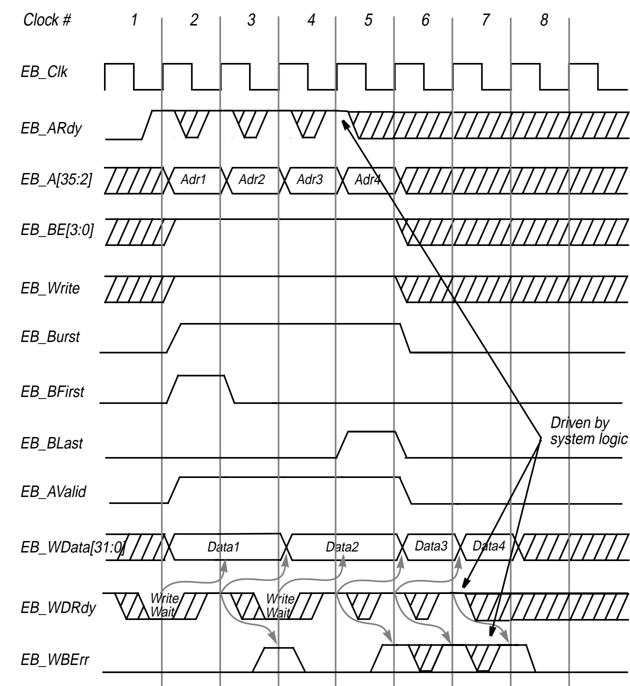


Figure 11 Burst Write Transaction Timing Diagram

The 4Kc core drives address and control information onto the *EB_A[35:2]* and *EB_BE[3:0]* signals on the rising edge of clock 2. As in the single read cycle, these signals remain active until the clock edge after the *EB_ARdy* signal is sampled asserted. The 4Kc core continues to drive *EB_AValid* as long as a valid address is on the bus.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

The 4Kc core asserts the *EB_Write*, *EB_Burst*, and *EB_AValid* signals during the time the address is driven. *EB_Write* indicates that a write operation is in progress. The assertion of *EB_Burst* indicates that the current operation is a burst. *EB_AValid* indicates that valid address is on the bus.

The 4Kc core asserts the *EB_BFirst* signal in the same clock as address 1 is driven to indicate the start of a burst cycle. In the clock that the last address is driven, the 4Kc core asserts *EB_BLast* to indicate the end of the burst transaction.

In Figure 11, the first data word (Data1) is driven in clocks 2 and 3 due to the *EB_WDRdy* signal being sampled deasserted at the rising edge of clock 2, causing a wait state. When *EB_WDRdy* is sampled asserted on the rising edge of clock 3, the 4Kc core responds by driving the second word (Data2).

External logic drives the *EB_WBErr* signal one clock after the corresponding assertion of *EB_WDRdy* if a bus error has occurred as shown by the arrows in Figure 11.

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.

Copyright © 1999-2002 MIPS Technologies, Inc. All rights reserved.

Unpublished rights reserved under the Copyright Laws of the United States of America.

This document contains information that is proprietary to MIPS Technologies, Inc. ("MIPS Technologies"). Any copying, reproducing, modifying, or use of this information (in whole or in part) which is not expressly permitted in writing by MIPS Technologies or a contractually-authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

MIPS Technologies or any contractually-authorized third party reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error of omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Any license under patent rights or any other intellectual property rights owned by MIPS Technologies or third parties shall be conveyed by MIPS Technologies or any contractually-authorized third party in a separate license agreement between the parties.

The information contained in this document shall not be exported or transferred for the purpose of reexporting in violation of any U.S. or non-U.S. regulation, treaty, Executive Order, law, statute, amendment or supplement thereto.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or any contractually-authorized third party.

MIPS®, R3000®, R4000®, R5000® and R10000® are among the registered trademarks of MIPS Technologies, Inc. in the United States and certain other countries, and MIPS16™, MIPS16e™,MIPS32™, MIPS64™, MIPS-3D™, MIPS-based™, MIPS I™, MIPS II™, MIPS III™, MIPS IV™, MIPS V™, MDMX™, SmartMIPS™, 4K™, 4Kc™, 4Km™, 4Kp™, 4KE™, 4KEc™, 4KEm™, 4KEp™, 4KS™, 4KSc™, 5K™, 5Kc™, 5Kf™, 20K™, 20Kc™, R20K™, R4300™, ATLAS™, CoreLV™, EC™, JALGO™, MALTA™, MGB™, SEAD™, SEAD-2™, SOC-it™ and YAMON™ are among the trademarks of MIPS Technologies, Inc.

All other trademarks referred to herein are the property of their respective owners.

Document Number: MD00039
01.03-2B

Copyright © 1999-2002 MIPS Technologies Inc. All right reserved.