MIPS

R8000 MICROPROCESSOR CHIP SET

USERS MANUAL

Revision 3.0
July, 1994

Steve Rhodes

Silicon Graphics Computer Systems
MIPS Technologies, Incorporated
2011 N. Shoreline Blvd.
Mountain View, CA. 94039

The information in this document is subject to change without notice.

Silicon Graphics, Inc. (SGI) reserves the right to change any portion of the product described herein to improve function or design. SGI does not assume liability arising out of the application or use of any product or circuit described herein. No part of this document may be copied by any means without written permission of Silicon Graphics, Inc.

INTRODUCTION TO THE R8000 MICROPROCESSOR CHIP SET

1

The MIPS R8000 Microprocessor Chip Set from MIPS Technologies implements a superscalar architecture, providing low-end vector supercomputer performance at a fraction of the cost. The 64 bit architecture of the MIPS R8000 Microprocessor Chip Set is implemented using separate integer and floating point devices. The impressive floating point performance of the R8000 Microprocessor Chip Set makes it ideal for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multi-user systems. The high throughput is achieved through complete separation of the integer and floating point functions, the use of wide, dedicated data paths, and large on- and off- chip caches.

The R8000 Microprocessor Chip Set implements the MIPS IV instruction set. MIPS IV is a superset of the MIPS III instruction set and is backward compatible. Implementing a 3.3 volt technology with a target frequency of 75 MHz, the R8000 Microprocessor Chip Set delivers peak performance of 300 MIPS and 300 MFLOPS. The R8000 CPU contains 2.6 million transistors. The R8010 Floating Point Unit contains 830 thousand transistors. Each device is housed in a 591 pin PGA package and is fabricated using the Toshiba VHMOSIII 0.7-micron silicon technology. Two Tag RAM's and 4 MBytes of Static RAM comprise the second level streaming cache.

1.1 R8000 MICROPROCESSOR CHIP SET FEATURES

- o Advanced Superscalar Architecture
  - Supports Four Instructions per Cycle
  - Two Load/Store Instructions per Cycle
  - Two Integer and Two Floating Point Execute Instructions per Cycle
- o High Performance Design
  - 75 MHz Clock Rate
  - 3.3 Volt Technology
  - Separate Integer and Floating Point Chips
  - 300 Double Precision MFLOPS Peak
  - On-Chip Floating Point Instruction Queue
  - Separate 64 bit Load and Store Data Busses
  - Implements MIPS IV Instruction Set
- o High Integration Chip-Set
  - R8000 CPU Contains:
  - 16 KByte Dual Ported Data Cache
  - 16 KByte Single Ported Instruction Cache
  - 384 Entry Dual Ported Translation Lookaside Buffer
  - 1K Entry Branch Prediction Cache
  - Second Level Cache Support
- o Optimized for Floating Point Performance
  - Separate-Chip Floating Point Unit
  - Two Floating point Execution Units
  - Two Floating Point Arithmetic and Two Floating Point Memory Operations per Clock
  - Large Load/Store Data Queues
- o Second Level Cache Support
  - Two 4-Way Set Associative Tag RAM Chips
  - Supports 4 MBytes of Second Level Cache
  - Delivers Two 64-bit Operands to the Floating Point Unit Every Clock.
  - Each Tag RAM has a Dedicated Bus Interface to the R8000 CPU.
- o Compatible with Industry Standards
  - ANSI/IEEE Standard 754-1985 for Binary Floating Point Arithmetic
  - MIPS III Instruction Set Compatible
  - Conforms to MESI Cache Consistency Protocol
  - IEEE Standard 1149.1/D6 Boundary Scan Architecture

1.2 ARCHITECTURAL INNOVATIONS

The design of the R8000 Microprocessor Chip Set incorporates many architectural innovations which enhance performance. Many of these innovations are unique in the industry. Each of the items listed below is explained further in the following sections.

1) A five-stage pipeline which swaps the execution and address generation stages.
2) Super-scalar dispatch unit which allows execution of 4 instructions per clock and is NOT boundary dependent.
3) Large Set associative TLB.
4) Data Cache invalidation down to the word (32 bit) level.
5) Split Level Cache resulting in the separation of Integer and Floating Point Data.
6) Address Bellow Register which resolves bank conflicts and helps maintain a uniform flow of even and odd references to the interleaved streaming cache.
7) Very fast 4 cycle integer multiply mechanism.
8) Use of instruction and data queues to streamline the movement of instructions between the Integer and Floating Point Units.
9) Prefetch instruction allows for the early fetching of data which can be placed as close as possible to the processor until it is required.
10) Addition of conditional move instructions helps avoid unnecessary branches.

1.2.1 Five Stage Pipeline

The R8000 Microprocessor contains a five stage pipeline which differs from the typical five stage RISC pipeline in that the execution stage and the address stage have been switched. The typical RISC pipeline contains the five stages configured in the following sequence: (F) Fetch, (D) Decode, (E) Execute, (A) Address cache, (W) Write result to register.

In this typical pipeline configuration, any instruction which follows a load and is dependent on the load incurs a cycle delay in execution. This delay can impact performance in a superscalar implementation because when it occurs the compiler must locate four instructions to put in the delay slot in order to maintain full utilization of the instruction bandwidth.

The R8000 Microprocessor Chip Set incorporates the following pipeline sequence:

(F) - Fetch and partial decode of the instruction. Branch prediction.
(D) - Decode instruction, read register file, perform scoreboarding and dependency checks.
(A) - Generate the required address
(E) - ALU execution, Data Cache access, TLB lookup, exception detection.
(W) - Write the result to the register file.

In the FDAEW pipeline the delay slot is placed before the load instruction and allows the processor to dispatch multiple instructions immediately following the load, including the instruction usually scheduled in the load delay slot.

Branch resolution occurs one cycle later, thereby increasing the branch penalty by one cycle. To overcome this problem the R8000 Microprocessor actually predicts the branch in the Fetch stage. Branch and delay instructions are fetched in F-stage and if the predict bit is on the program counter is modified and on the next clock the instruction cache starts fetching from the new target address.

Refer to Figure 1-1. Quad 1 is fetched (PC= x) and then enters D-stage in (PC = x+1). At the same time quad 2 is fetched which contains a branch and corresponding delay. Since the predict bit in the branch cache is on, the new branch target address 't' is loaded into the program counter. Quad 3 is fetched with the new target address. In the next clock (PC = t+1) the branch and delay in quad 2 enter the A-stage. In t+2 Quad 2 enters the E-stage. It is here that the instruction is executed and a determination is made as to whether the branch prediction was correct. The first instruction from the target address, Quad 3, is now in A-stage, one stage behind the branch and delay instructions. Once the branch has been executed and the target address is determined, the value is then compared with the target for the instruction in A-stage. If where the instruction wanted to branch is the same as where the pipeline had branched to 3 cycles earlier (E-stage and A-stage target compare is valid), then the pipeline continues without interruption. A branch mis-prediction causes a three cycle delay as the instructions in stages D, A, and E must be flushed.

Figure 1-1 below shows the pipeline flow of the R8000 Microprocessor. A Quad is defined as four 32 bit instructions. PC = program counter.

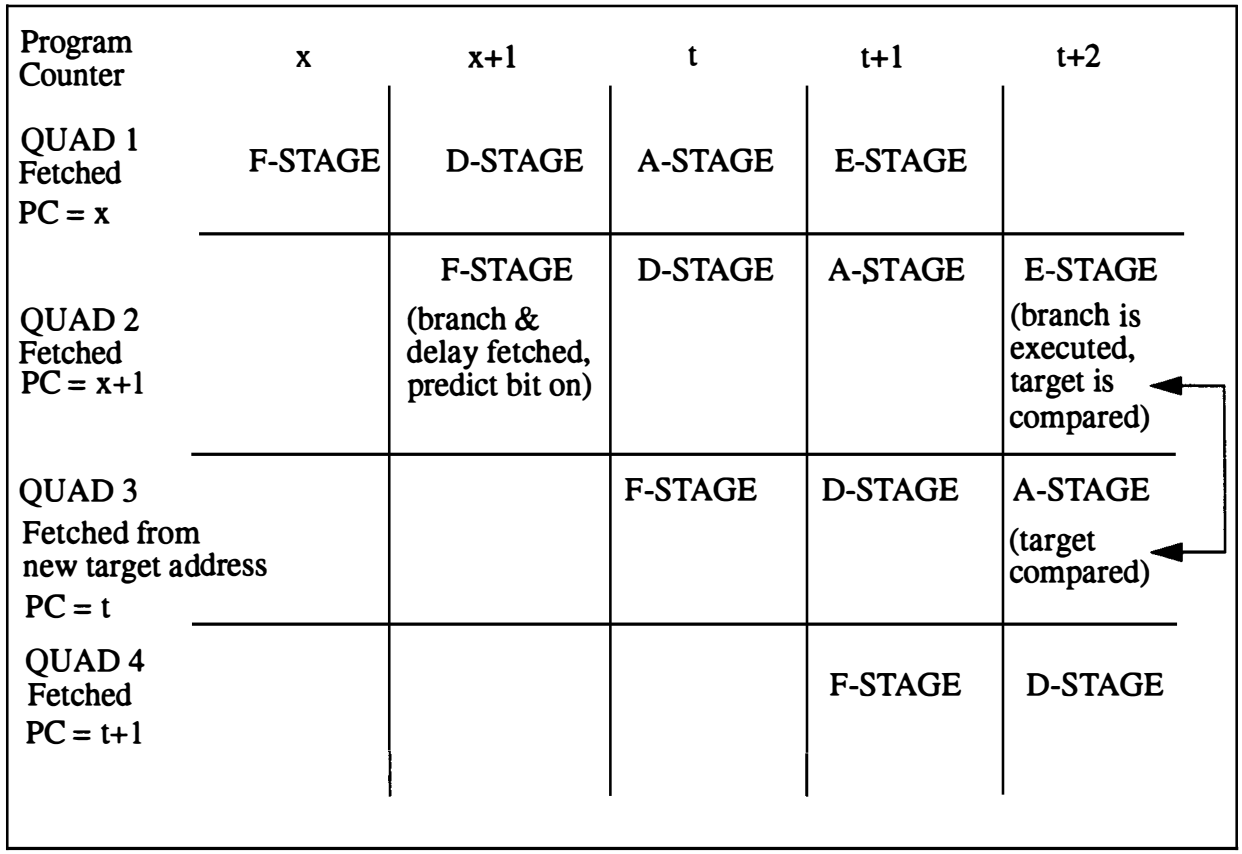


Figure 1-1 Integer Pipeline

As shown in Figure 1-1, on a branch mis-prediction, the execution of each branch instruction and the corresponding change of control flow takes 3 clocks.

1.2.2 Superscalar Dispatch Unit

The R8000 Microprocessor can dispatch four instructions each cycle regardless of how many instructions were issued in the previous cycle. There are no boundary alignment restrictions. Instructions are fetched and placed in a six-quad deep instruction queue which acts as temporary storage for instructions waiting to be executed. When instructions are fetched from the I-cache they undergo predecoding before being placed in the queue. The purpose of predecoding is to reduce instruction processing time in the decode stage of the pipeline. Eighteen additional characterization bits are added to each original 32 bit instruction. Addition of the predecode bits expands each instruction to 50 bits, hence the width of the instruction queue is 200 bits. These bits are used for two cycles, after which for integer operations they are no longer needed. Instructions which reach the floating point queue are 37 bits wide as five bits of the original 18 additional bits are used by the R8010 FPU. Predecoding accomplishes three things:

1) Consistent alignment of the 5-bit destination field.
2) Instruction Category encoding.
3) Addition of timing critical bits.

A crossbar mechanism determines which of the four instructions to send depending on the resources available from cycle to cycle. This process is called Resource Modeling. The idea behind resource modeling is that instructions are not dispatched until there is sufficient resources available for them to complete. The crossbar monitors the status of each execution unit as well as determines interdependencies between any of the four instructions in the dispatch unit at any given line.

Figure 1-2 shows a diagram comprised of four cycles and the flow of instructions through the supersclar dispatch mechanism.

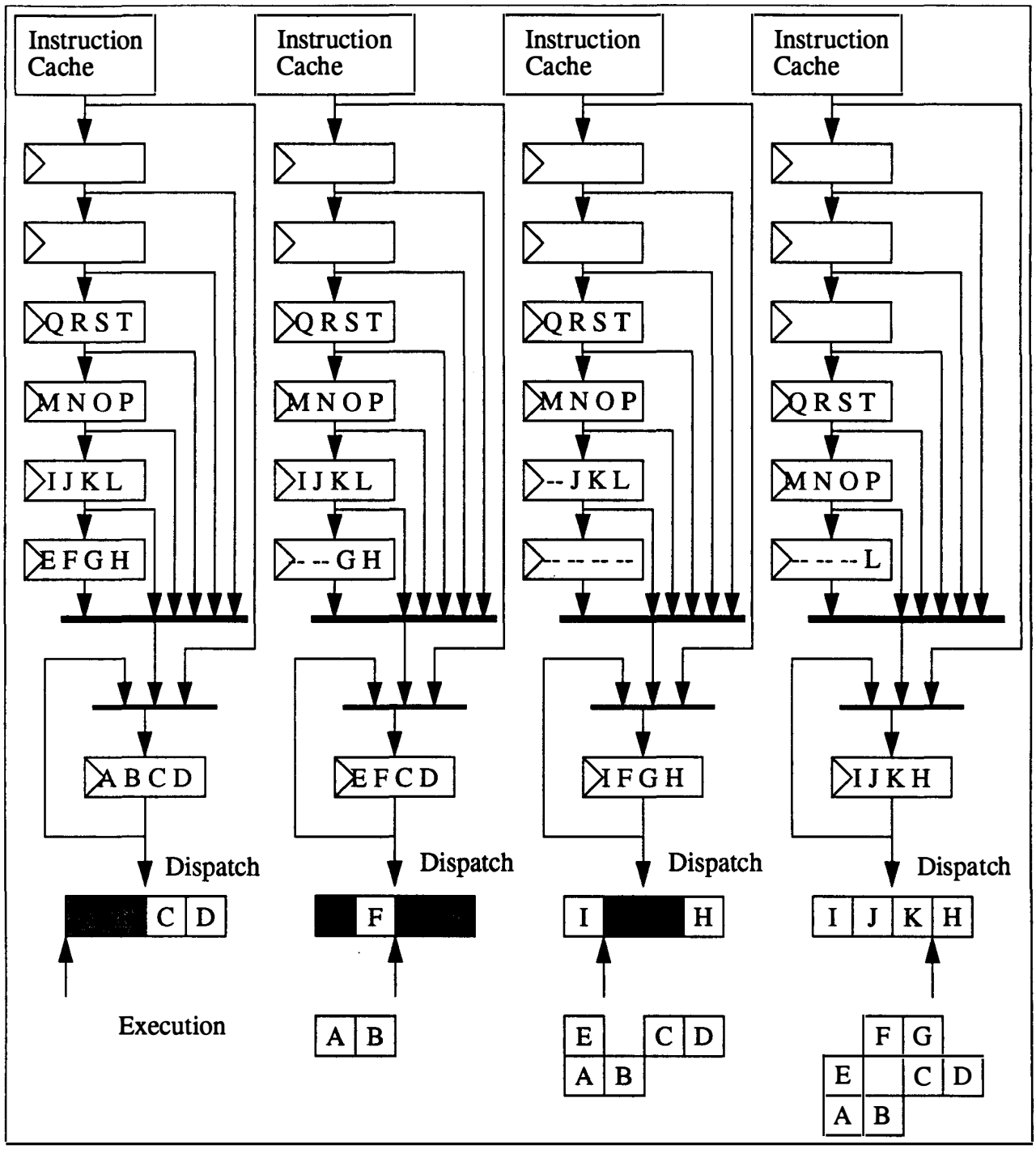


Figure 1-2 R8000 Instruction Dispatch Mechanism

The shaded areas in Figure 1-2 indicate those instructions which were dispatched. In the first clock instructions A, B, C, and D are presented to the dispatch logic. In the above example only instructions A and B are dispatched and sent to the execution stage of the pipeline. Instructions C and D remain. Instructions E and F are then read from the first

stage of the queue and passed through the multiplexor logic and placed in the dispatch unit in the next clock. Instructions G and H remain in the first stage of the queue. The dispatch unit is again loaded with four instructions (E, F, C, D).

In the second dispatch in Figure 1-2 instructions E, C, and D are dispatched to the execution stage but instruction F remains. The execution of C and D allows instructions G and H to pass through to the dispatch unit. The first stage of the queue is now empty and new information can be clocked into it. Instruction I is also read from the second stage of the queue and placed in the dispatch unit. The dispatch unit now contains the four instructions I,F,G,H.

The third cycle dispatches instructions F and G. I and H remain.The execution of F and G allows instruction J and K from the second stage of the queue to be moved to the dispatch unit. The instruction queue is then clocked, causing the four instructions in each stage to shift one stage down the queue as shown. The first stage of the queue now contains only instruction L because instructions I, J and K have already been shifted out.

The first stage of the queue must be completely empty before any other instructions can be shifted into it. All stages of the queue are clocked simultaneously. Should a situation arise where the queue is full, meaning that all stages contain one or more instructions, a stall is issued and the instruction cache will cease fetching instructions until the stall condition is removed.

1.2.3 Large Set Associative TLB

The Translation Lookaside Buffer (TLB) is dual ported and is physically split into two halves. Each half contains 128 entries and is 3-way set associative, yielding a total of 384 entries each. One half contains the virtual tags (VTAGS), the other the actual physical address (PA) corresponding to **each** virtual tag.

TLB, Data Cache, and Data Cache Tag RAM lookups are performed in the execution stage (E-stage) of the pipeline. The VTAG portion of the TLB is used to determine whether a certain range of addresses resides in the PA portion. If it is determined that the translation for the virtual address resides in the TLB, the contents of the PA portion is compared to that in the Data cache tag RAM, resulting in either a hit or a miss to the Data cache. Either a TLB or a Data cache miss initiates an external memory cycle.

Figure 1-3 shows a block diagram of the TLB.

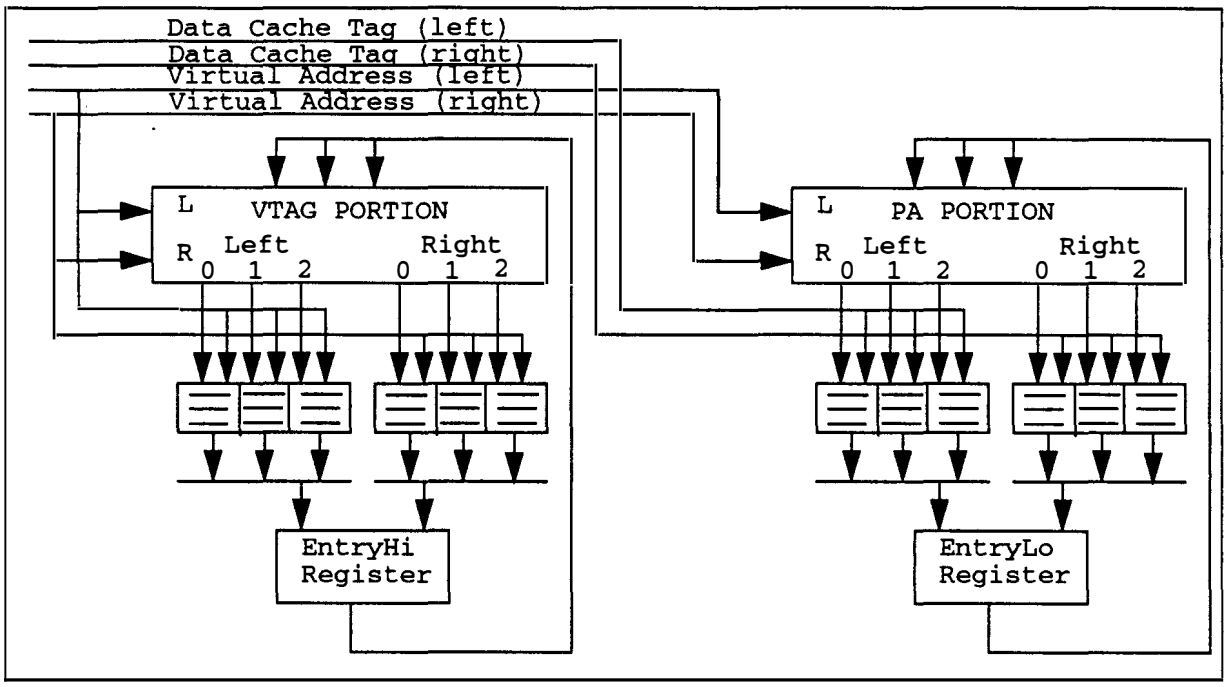


Figure 1-3 Translation Lookaside Buffer

Table 1-1 shows the page sizes supported and the corresponding virtual address bits used to index the TLB.

Page Size	Virtual Address Bits
4K	VA<18:12>
8K	VA<19:13>
16K	VA<20:14>
64K	VA<22:16>
1M	VA<26:20>
4M	VA<28:22>
16M	VA<30:24>

Table 1-1 TLB Page Sizes

The number of entries in the TLB is large enough to minimize the miss rate but at the same time is not so large as to create speed problems.

1.2.4 Data Cache Invalidation

Data cache invalidation in the R8000 CPU is performed by accessing the Data Cache Valid RAM which is 4 bits wide and contains 1024 entries. Each entry represents a 32 bit value, hence there are two valid bits per 64 bit doubleword. Each index to the valid RAM corresponds to the status of two 64 bit Data cache doublewords. There are two reasons for a separate valid RAM with individual invalidation bits down to the word level.

The first reason is to alleviate invalidating entire lines of the cache when floating point data is found. Sometimes integer and floating point data reside in the same data cache (D-cache) line. Floating point loads and stores interface directly to the streaming cache and do not usually affect the contents of the R8000 data cache. However, the data cache of the R8000 must be kept coherent with the streaming cache. Therefore, if a FP store is done to a given location in the streaming cache which also resides in the D-cache, the D-cache entry must be invalidated. By having individual valid bits for each 32 bit word in the data cache, the mixing of floating point and integer data in a given D-cache line is better accommodated. This way if an integer load is done to that same location a D-cache miss occurs, forcing the R8000 to fetch the data from the streaming cache.

The second reason for having a separate valid RAM is to be able to easily invalidate the data for integer stores which miss in the D-cache. In the R8000 data is stored to the D-cache in the same cycle that the TLB and D-cache hit/miss status is determined. This is done so that the store data does not have to wait for the result of the lookup before it is written to the D-cache. If the TLB detects a store hit the cycle is already completed as the data has already been written. If a store miss occurs the data is invalidated in the following cycle by turning off the valid bit for that D-cache entry.

Allowing invalidation down to the word level also helps to reduce 'false sharing', which occurs when data is unintentionally forced to bounce back and forth between caches.

1.2.5 Split Level Cache

The caching scheme of the R8000 microprocessor consists of a 16 KByte integer only first level data cache housed on the R8000, and a 4 MByte second level streaming cache. The 4 MByte streaming cache acts as the second level cache for the R8000 and the first level cache for the R8010 FPU. Since integer data is stored on-chip in the R8000 access latencies are very short. Due to the large data sets that are normally required for floating point operations, the R8010 FPU interfaces only to the streaming cache. Separation of integer and floating point data helps to alleviate 'thrashing', which can occur when large vectors are moved in and out of the smaller on-chip data cache. For example, if large floating point vectors were handled in the data cache, all of the contents of the data cache would need to be moved out to make room for the vector, the vector then moved in and executed, then moved out, and the integer data moved back in. Also since the data cache

is normally not big enough to hold an entire floating point vector, a portion of the vector would be moved in and executed, then another, then another. Each of these results in a cache miss.

In addition, the R8000 microprocessor allows either two loads or one load and one store in the same cycle. If a load follows a store, the store can write to the same address as the load is reading from. Bypass circuitry exists inside the R8000 which allows the store and the load to occur simultaneously. This is helpful when the compiler cannot differentiate between whether the store address for pointer A and the load address for pointer B are the same. This is why the store is done before the load. When the address is the same a clock is saved because the store to the cache does not have to complete before the load can be executed.

1.2.6 Address Bellow Register

The R8000 Microprocessor contains two Tag RAM's which support the two way interleaved streaming cache and can perform two Tag RAM accesses per cycle. One bank contains even addresses and the other odd. In order to facilitate two accesses per cycle one address must be even and the other odd. However, the compiler cannot always guarantee that one access will be even and the other odd and a situation can arise where there are either two odd or two even accesses in the same clock. When this occurs only one of the two accesses can execute as they are both to the same bank. Multiple mis-alignments by the compiler can degrade system performance. The address bellow register assures uniform distribution of even and odd references. Hardware manages and resolves the alignment problems.

Figure 1-4 shows how the address bellow resolves bank conflicts when both accesses alternate between odd and even. Each access has been numbered for clarity. Note that either of the even or odd accesses could be delayed. Those accesses shown in figure 1-4 as delayed are arbitrary. The numerical values have been added for clarity to show the movement through the bellow register.

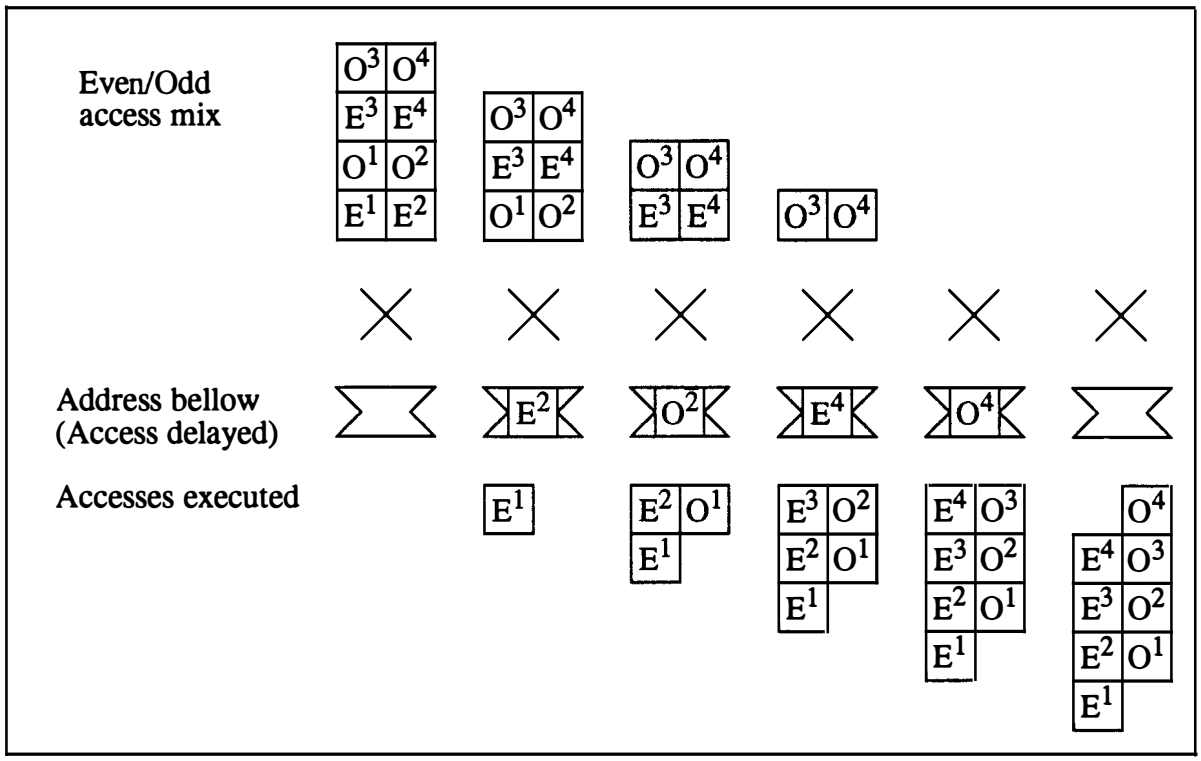


Figure 1-4 Effect of address bellow on bank conflicts

Figure 1-4 shows four separate accesses in an alternating even/odd sequence. Because only one of the two even accesses can be dispatched to the Tag RAM at a time, the second access is delayed in the bellow register, hence a single access is performed in the first clock of the sequence. Since accesses to the Tag RAM's are single cycle, the E^2 access previously delayed in the bellow register is released on the next clock along with one of the odd accesses. The bank conflict has been resolved in that both an even and an odd access are now allowed to occur simultaneously, even though they were not dispatched at the same time by the compiler.

In the next clock O^2 is released from the bellow and dispatched along with E^3 while E^4 is held in the bellow. Again two Tag RAM accesses are allowed to execute. In the next clock E^4 is released from the bellow along with O^3 while access O^4 is held in the bellow. On the final clock of the sequence access O^4 is released from the bellow. Since there are no subsequent accesses shown, the last clock in the sequence is also a single access.

In Figure 1-4 the first and last accesses in the sequence shown are single accesses. However, the effect of the bellow register is such that all of the cycles in between allow for two Tag RAM accesses at the same time. Note that the compiler plays a major role in the efficient scheduling of accesses. Poor scheduling techniques by the compiler can effectively cut the cache access bandwidth in half and effectively render the streaming cache as one-way interleaved.

1.2.7 Integer Multiply

The integer multiply function is performed in the R8000 CPU and is several times faster than most other implementations currently on the market. Only four clocks are required for a 32 bit multiply and six clocks for a 64 bit multiply. Integer multiplies are frequently used in array index calculations. The superscalar nature of the R8000 microprocessor allows the computing portion of the operation to execute in less time. Addressing time is also reduced by lowering the integer multiply time.

1.2.8 Floating Point Multiply-Add

The addition of the four floating point multiply-add/subtract instructions allows two floating point computations to be performed with one instruction. The four instructions are multiply-add, multiply-subtract, negative multiply-add, and negative multiply-subtract.

The product of two operands is either added to or subtracted from a third operand to produce one result. The intermediate result is calculated to infinite precision and is not rounded prior to the addition. The result is then added to or subtracted from the contents of a floating point register specified in the instruction. The result is then rounded according to the rounding mode specified by the instruction. The final result is then placed in another floating point register whose location is also defined in the instruction.

1.2.9 Floating Point Queues

The Floating Point Queue mechanism consists of a floating point instruction queue and a load data queue which together allow the R8000 to run ahead of the R8010 FPU. Because FP operations are decoupled from the R8000, long FP operations can be executed in parallel with other integer operations. The R8000 is not held up, allowing vector start-up time to be reduced. For example, in transitioning from one loop to another, while the R8010 FPU is completing the first loop, the R8000 can begin processing the overhead code and get started on the second loop, even though the R8010 FPU is not yet finished with the first loop.

When instructions are fetched from the instruction cache of the R8000 CPU, predecoding is performed to determine the nature of the instruction and where it is to be executed. Floating point instructions are placed in the FP queue and an access to the streaming cache is initiated to retrieve the corresponding FP vector. The FP queue is located in the R8000. Once the data is available it is placed in the load data queue. The R8000 then releases the instructions from the FP queue and sends them to the R8010 FPU where they can begin execution.

1.2.10 Prefetch Support

The R8000 Microprocessor supports a prefetch instruction which allows the compiler to issue instructions early so the corresponding data can be fetched and placed as close as possible to the CPU. For example, if data for a given operation resides in main memory, the prefetch instruction can be used to retrieve the data before it is required by the CPU. Once the CPU requests the data, the main memory access time has already elapsed and the data resides close to the CPU in a cache or data buffer where it can be accessed by the CPU quickly.

Normally the prefetch instruction is used in loops and in most cases the prefetched data will be used by the CPU. The prefetch instruction is most helpful in large multi-processor systems where a cache miss can take many cycles.

1.2.11 Conditional Moves

The R8000 Microprocessor has defined a set of four conditional move operators which allow IF statements to be represented without branches. The bodies of the THEN and ELSE statements are computed unconditionally and their results placed in temporary registers. Conditional move operators then transfer the temporary results to a permanent register file. Both legs of the IF statement are computed and one of them discarded.

Conditional moves must be able to test both integer and floating point conditions in order to support the full range of IF statements. Integer tests are done by comparing a general register against a zero value. This is similar to the way integer branches are performed.

Floating point tests are done by examining the floating point condition code. This is similar to the way Coprocessor 1 branches are handled. The conditional move operators in the R8000 microprocessor support both integer and floating point data for the THEN and ELSE clauses, hence there are four conditional move operators.

Since floating point conditional moves test the floating point condition code, multiple condition codes have been added to give the compiler some flexibility in scheduling the comparison and the conditional moves. The R8000 microprocessor contains eight condition code bits.

1.3 ARCHITECTURAL OVERVIEW

This section discusses briefly the architecture of each component in the R8000 microprocessor chip set. Each of the components, the R8000 CPU, R8010 FPU, Tag RAM's, and Streaming Cache Data RAM's have dedicated chapters which cover the respective components in more detail. Refer to the table of contents for more information

on a specific component. There are 7 basic parts to the system.

1) R8000 Microprocessor
2) R8010 Floating Point Unit
3) Tag RAM (even addresses)
4) Tag RAM (odd addresses)
5) Streaming Cache SRAM (Even data)
6) Streaming Cache SRAM (Odd data)
7) Cache Controller

Both the R8000 CPU and R8010 FPU have multiple execution units, allowing execution of 4 instructions per clock; two load/store instructions and two register to register or floating point execute instructions. Separate integer and floating point units maximize floating point throughput and allow for simultaneous execution of integer and floating point instructions. The R8010 FPU contains two pipelines, each of which can perform a double precision multiply-add every cycle. Two identical tag RAM's store address information for the even and odd data banks of the interleaved streaming cache. The R8000 CPU and R8010 FPU interface only to the streaming cache. Updates to the tag RAM's as well as all transactions requiring interface to main system memory are handled by the cache controller. Separate load and store data busses on the R8010 FPU eliminate bus turnaround time and allow both loads and stores to second level cache to execute simultaneously. Multiple tag RAM's provide an interleaved caching scheme, allowing access times to the cache to be hidden and providing two 64 bit operands to the R8010 FPU every clock. A separate dirty bit RAM within each Tag RAM allows for updating of the dirty bit status for one cycle at the same time as a Tag RAM access for another cycle. Figure 1-5 shows a block diagram of the R8000 microprocessor chip set.

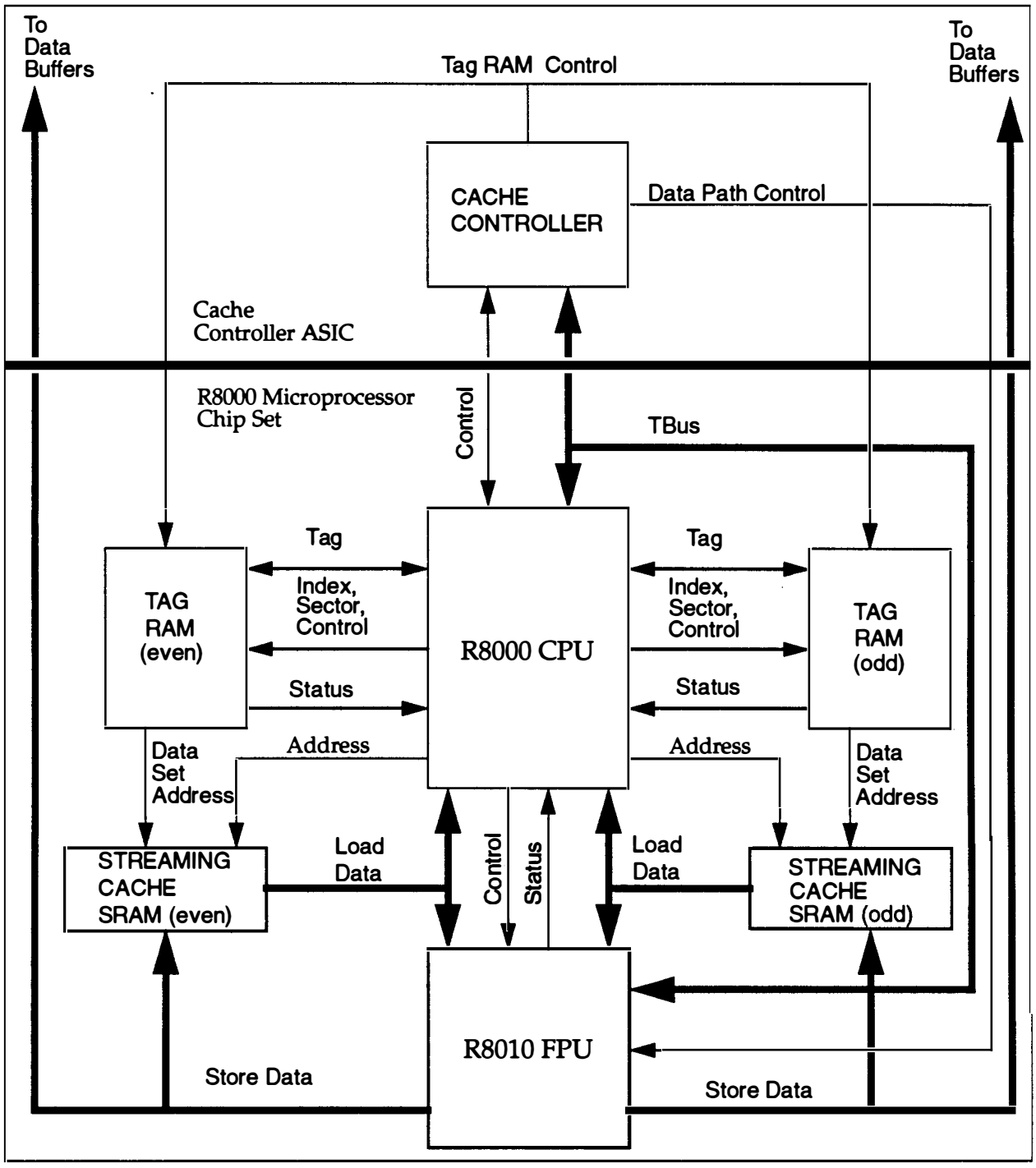


Figure 1-5 R8000 Microprocessor Chip Set Block Diagram

1.3.1 R8000 Microprocessor.

The R8000 Microprocessor is a 591 pin device which handles all integer operations and is the main computing component of the system. The high pin count is a result of the numerous dedicated busses provided by the R8000. This dedicated bussing scheme helps to take full advantage of the multiple execution units within the R8000, allowing each unit to run independently of the other, alleviating not only the need for multiplexing data and address, but also allowing loads and stores to the streaming cache to occur simultaneously.

The R8000 contains four caches and has dedicated interfaces to all components in the system. The R8000 performs address generation and provides address information for interfacing to the streaming cache via separate and dedicated address busses for the even and odd banks. Instruction and data interface to the R8010 Floating Point Unit is via a dedicated 80 bit TBus, and addresses to the Tag RAM's are provided via separate and dedicated tag, index, and sector busses for both the even and odd tag RAM's.

The R8000 contains two arithmetic logic units (ALU) as well as two address generation units, yielding a maximum of 4 instructions per cycle. The on-chip 16 KByte Data cache is dual ported and contains separate address and 64 bit data busses for each port. This allows multiple accesses to the cache to occur simultaneously. The 16 KByte instruction cache is 128 bits wide and single ported. Both caches are virtually indexed. The instruction cache is virtually tagged, alleviating the need for address translation on I-cache accesses. The data cache is physically tagged to maintain coherency with second level cache. Each 32 byte line in the I-cache contains a specific address space identifier (ASID). This value is assigned by the operating system and is process specific. There are at least two specific ASID values per process, one for the instruction cache, one for the TLB. The ASID helps to differentiate between multiple processes within the same cache and helps to reduce I-cache flushing by allowing the operating system to invalidate only those lines whose process is no longer valid. The operating system can also flush the I-cache when all 256 ASID values have been used.

In addition to the data and instruction caches, the R8000 also contains Branch and Translation Lookaside Buffer (TLB) caches. The Branch cache is accessed along with the instruction cache and is used to predict and modify the program counter on branch or jump instructions. The Branch Cache is a 15 bit field concatenated to each aligned 128 bit quadword of the I-cache. The Branch Cache implements a simple branch prediction mechanism which branches depending on the state of the predict bit associated with each Branch Cache entry.

The TLB cache is used to convert virtual addresses to physical addresses. A single TLB services both the data and instruction caches. The instruction cache only requires address translation on a miss. Similar to the instruction cache, each entry of the TLB also contains an ASID. However, this value is different from that contained in the I-cache. Having separate ASID values for each cache allows separate flushing of the Instruction and TLB caches. Below is a list of features of the four caches.

- Instruction cache;
- 16 KBytes
- Virtually indexed
- Virtually tagged
- Direct mapped, no hashing
- Single ported, 128 bit path
- Fetches 4 instructions (128 bits) per cycle
-32 Byte line size
- No parity
- 11 cycle miss penalty to streaming cache
- No coherency maintained with streaming cache
- Alignment on 128-bit boundaries
- Separate ASID values for I-cache tags

- Data Cache;
- 16 KBytes
- Virtually indexed
- Physically tagged
- Direct mapped, no hashing
- Dual ported, 64 bit data paths
- Two loads or one load and one store per cycle
- 32 Byte line size
- No parity
- 8 cycle miss penalty to streaming cache
- Coherency maintained with the streaming cache
- Write through with allocate protocol
- Separate ASID for D-cache tags

- Branch Cache;
- 1K Entries, one entry per 4 instr.
- Virtually indexed in parallel with Instruction cache
- Direct mapped, no hashing
- 3 cycle miss penalty

- TLB Cache;
- Dual ported, 2 translations/clock
- 3-way set associative,
- 384 entries total (128 X 3 way)
- Implements random replacement algorithm
- Supports 4K,8K,16K,64K,1M,4M,16M page sizes
- Maps one virtual to one physical page
- Indexed by low-order 7 bits of virtual address
- Index is hashed by Exclusive-OR of low order 7 bits of TLB cache ASID.
- Software Refilled

Figure 1-6 shows a block diagram of the R8000 Microprocessor.

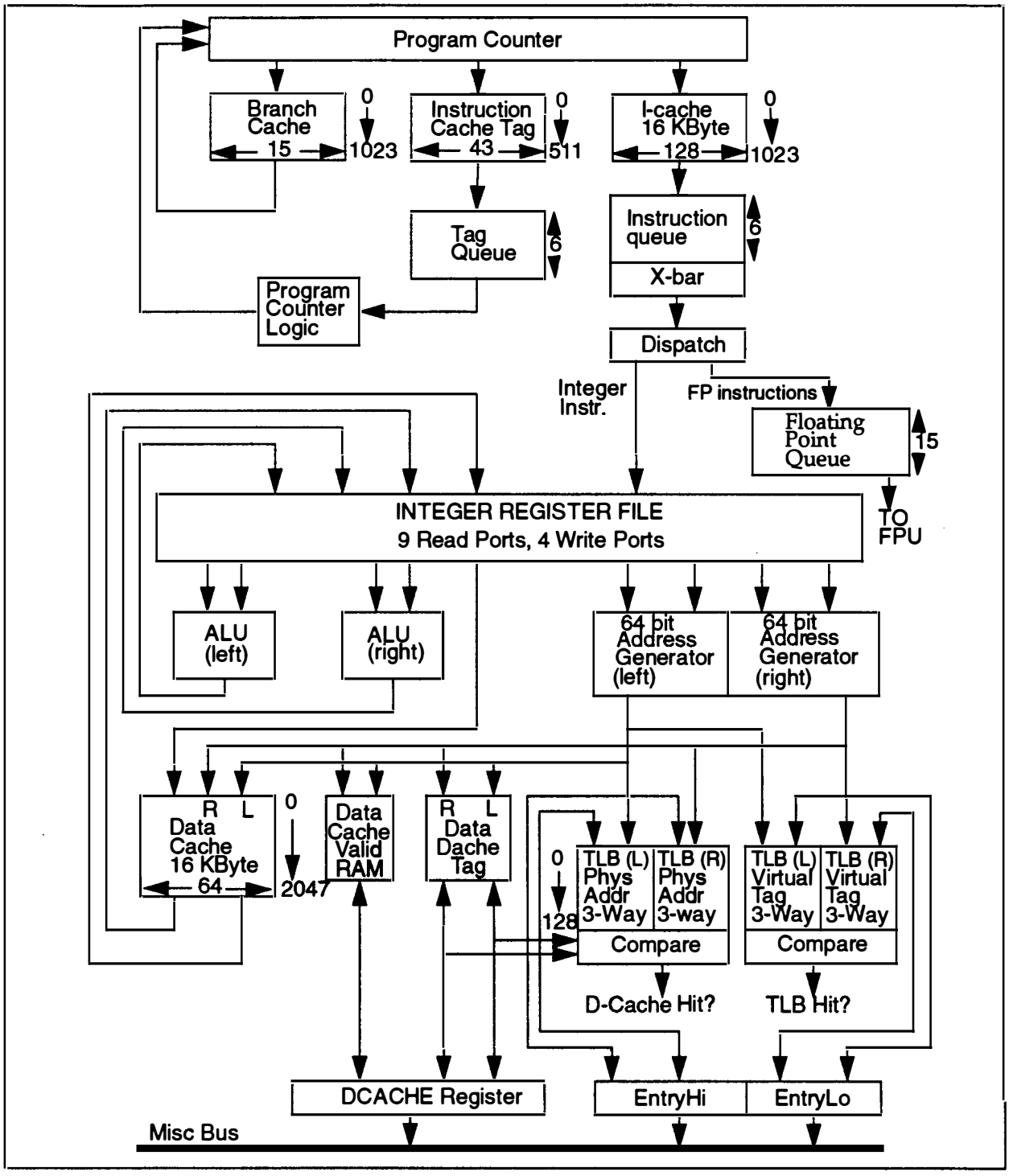


Figure 1-6 R8000 Microprocessor Block Diagram

1.3.2 R8010 Floating Point Unit

The R8010 Floating Point Unit (FPU) is a 591 pin device which performs all floating point functions for the R8000 Microprocessor Chip Set. The R8010 FPU has two execution units, allowing two arithmetic and two Floating Point memory operations to be executed every clock. The Floating Point Register File contains 8 read ports and 4 write ports. Large load and store data queues, each 32 entries deep, allow for a pipelined interface between the R8000 CPU and the R8010 FPU, streamlining the flow of data and minimizing wait time. With a target frequency of 75 MHz, the R8010 FPU offers a peak performance of 300 MFLOPS.

The R8010 FPU has no on-chip cache and uses the streaming cache, which is the second level cache of the R8000, as its memory. Dedicated load and store data busses to both the even and odd banks of streaming cache allow either a read or write operation to each bank to be performed every clock. An 80 bit TBus interface forms the control bus for the R8010 FPU and allows the R8010 FPU to interface to both the R8000 CPU and the Cache Controller (CC). Normally the R8010 FPU is controlled by the R8000. Dispatching of instructions, floating point loads and stores to the streaming cache, integer stores to the streaming cache, etc. are all under control of the R8000 CPU. Cycles which miss in the streaming cache and require interface to the main memory are handled by the Cache Controller. For these cycles the R8010 FPU is used only to transfer data from the load data bus to the store data bus.

Floating point instructions are received from the R8000 microprocessor through the TBus. The instructions are executed and the result written back to the FP register file. Floating Point data is retrieved from the streaming cache on the load data pins and then placed in the Load Data Queue. For store operations data from the result is placed in the store data queue. As soon as the corresponding address information from the Tag RAM is made available, the data is written out to the streaming cache. In addition to floating point operations, the R8010 FPU is also used during integer stores to the streaming cache, handled by the R8000, as well as stores to main memory, handled by the CC.

A set of fused multiply-add instructions have been added, taking advantage of the fact that the majority of floating point computations use the chained multiply-add paradigm. The operator for the multiply-add instructions is not defined by the IEEE and does not perform intermediate rounding. Eliminating the intermediate rounding step allows for a lower inherent latency and has higher precision and higher performance than an operator which performs intermediate rounding.

Figure 1-7 shows a block diagram of the R8010 FPU.

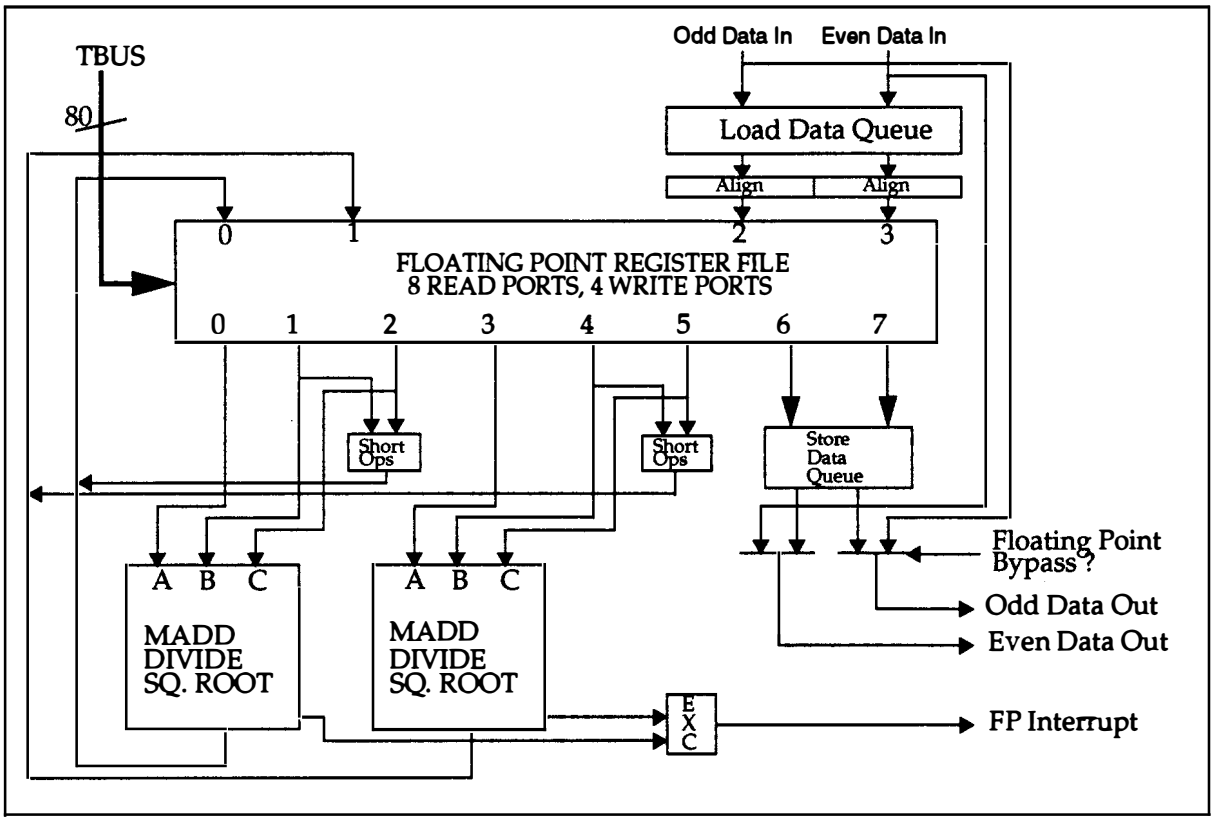


Figure 1-7 R8010 Floating Point Unit Block Diagram

1.3.3 Tag RAM

Two identical tag RAM's are required in the R8000 Microprocessor Chip Set in order to support the interleaved architecture of the second level streaming cache. Both RAM's contain the same information. One is used for the even bank, the other for the odd bank. The two banks are differentiated by the state of address bit A3. If this bit is low the access is to the even bank. A3 high enables the odd bank.

Both Tag RAM's are always written simultaneously and contain the exact same address, state, and virtual synonym information. The dirty bit information can be different between the even and odd bank devices. If either the even or odd double-words of a cache line are dirty the CC will write back the entire line. The Tag RAM is 4-way set associative. Each indexed entry of the Tag RAM contains 128 bits divided as four 32 bit values. Each 32 bit value contains a 20 bit tag address, a four bit virtual synonym field, and 8 state bits which define the coherency attributes. Either the tag address or the state and virtual synonym information can be written at any given time. A single 20 bit external tag bus handles the flow of both through the device. In addition the tag bus is bi-directional and used for both reading and writing of the device. The Cache controller

is responsible for both reading and writing the Tag RAM and must control whether the address or state information is allowed to be written.

A separate 16 bit dirty bit RAM is the only portion of the Tag RAM where the information is different between the two Tag RAM's. Figure 1-8 shows a block diagram of the tag RAM.

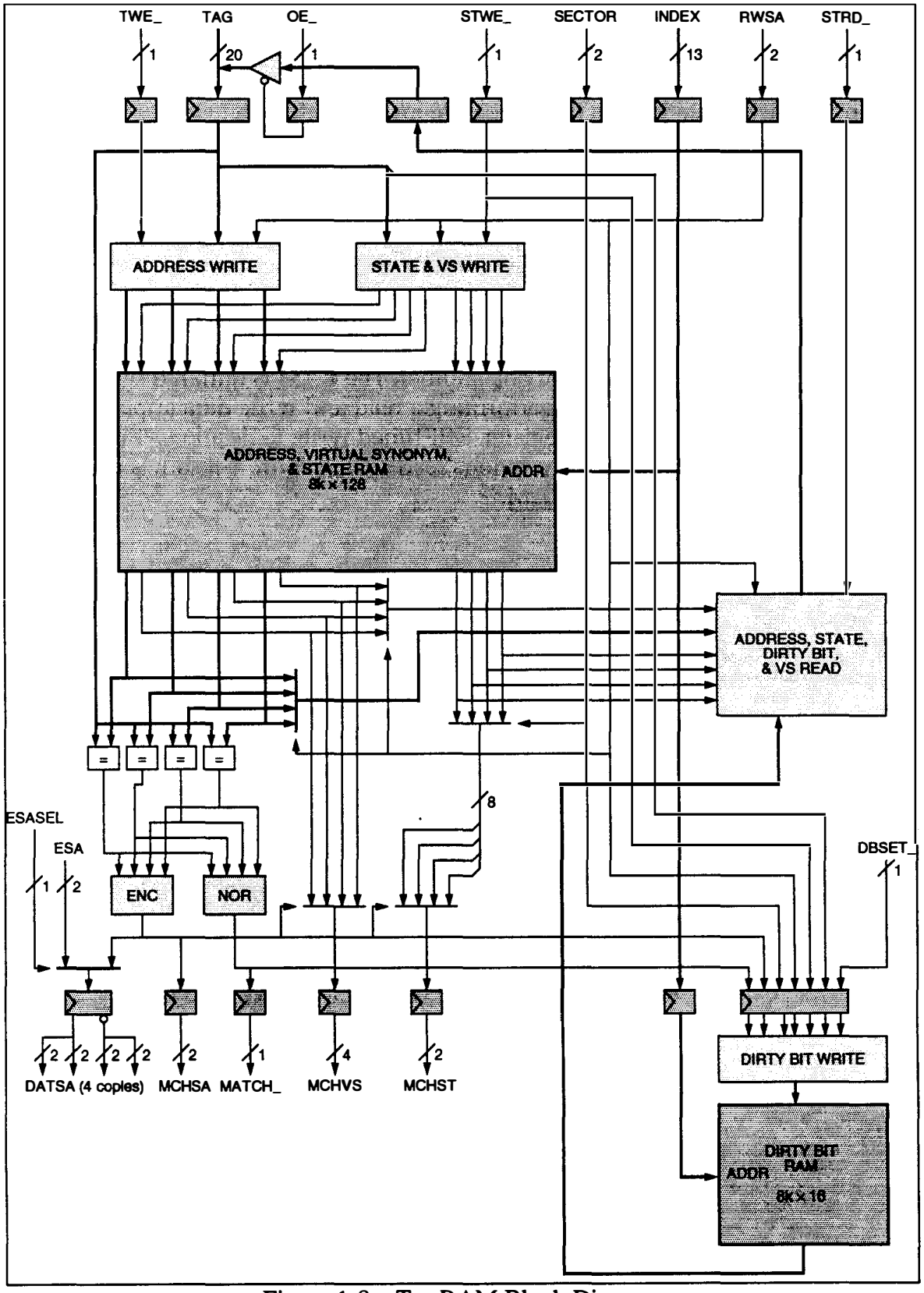


Figure 1-8 Tag RAM Block Diagram

1.3.4 Streaming Cache Data RAM's

The streaming cache data RAM's have separate load and store data busses. Although only one cycle can be performed by the data RAM's at a time, both read and write data can be on their respective busses at the same time. Having separate busses eliminates any bus turnaround time, which occurs on back to back read followed by write cycles, and allows read and write data to be pipelined to the RAM, effectively allowing the RAM to perform a read or write operation every clock.

The total memory size is split between the even and odd banks. Each bank contains 2 MBytes and has a dedicated Tag RAM, allowing accesses to the banks to operate simultaneously and independently of one another. The RAM is buffered by input and output data registers. If the RAM is performing a read and write data appears on the input bus, the data is placed in the register. Self-timed write logic allows the RAM to write the data as soon it finishes the previous read cycle. Figure 1-9 shows a block diagram of a streaming cache data RAM.

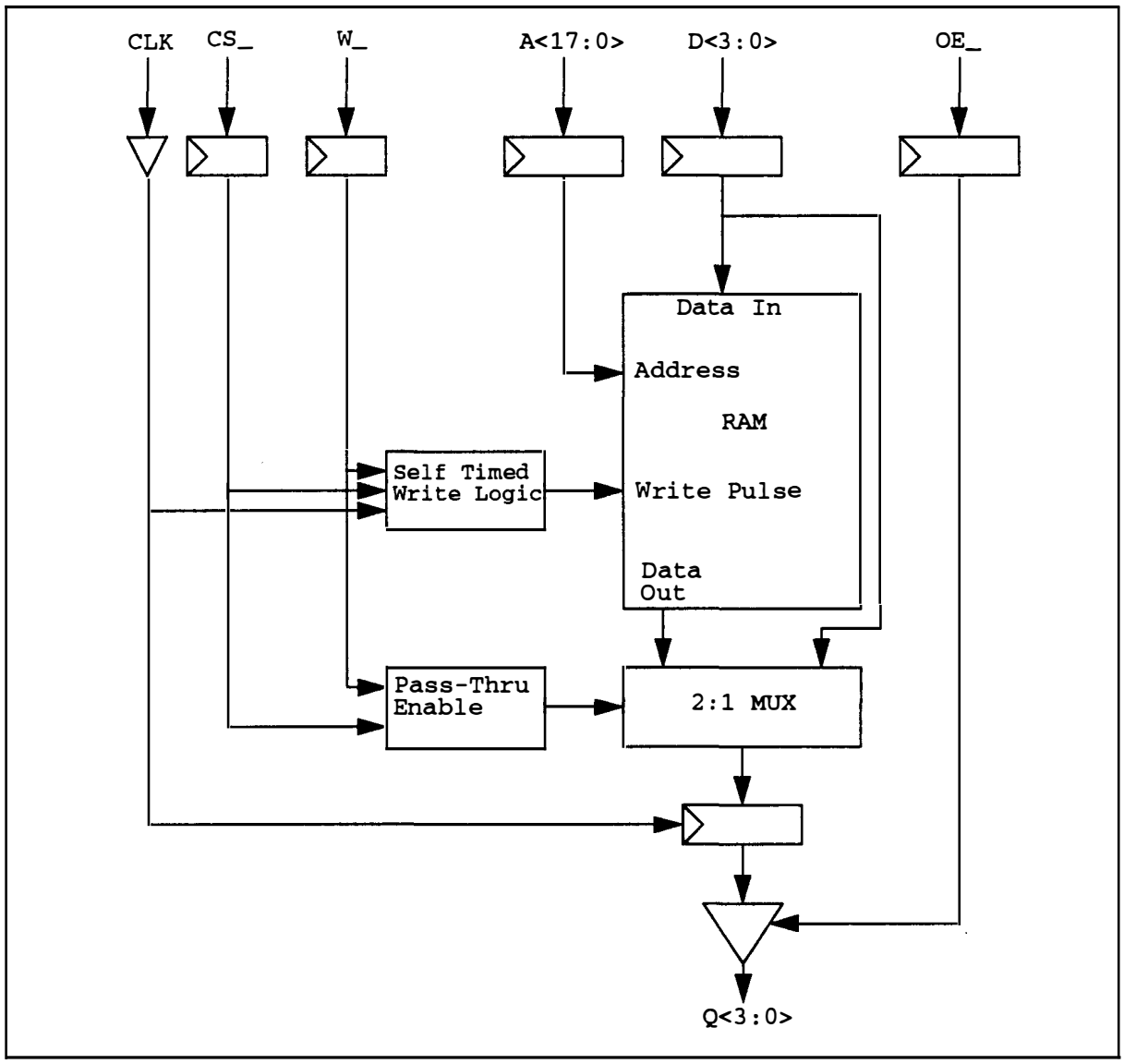


Figure 1-9 Streaming Cache Data RAM Block Diagram

1.3.5 Streaming Cache Memory Architecture

The cache memory system architecture consists of sets, lines, and sectors. In order to help the reader understand the cache memory architecture, the following example discusses a 4 MByte SIM module implementation in a 4-way set associative configuration with 128 bytes per sector and four sectors per line.

There are nine devices per SIM module, 8 data RAM's and 1 parity RAM, which yield 1

MByte. A minimum of two SIM modules per bank are required to interface to each of the 64 bit external busses of the R8000 Microprocessor. Connection to the mother board is via two parallel 75 pin SIP connectors. The modules are soldered directly to the board.

The 4-way set associative cache has 4 sets. Each set consists of 2048 lines. Each line consists of four sectors. Each sector contains sixteen 64-bit words divided as 8 words per bank. Figure 1-10 shows a block diagram of how the cache memory is organized.

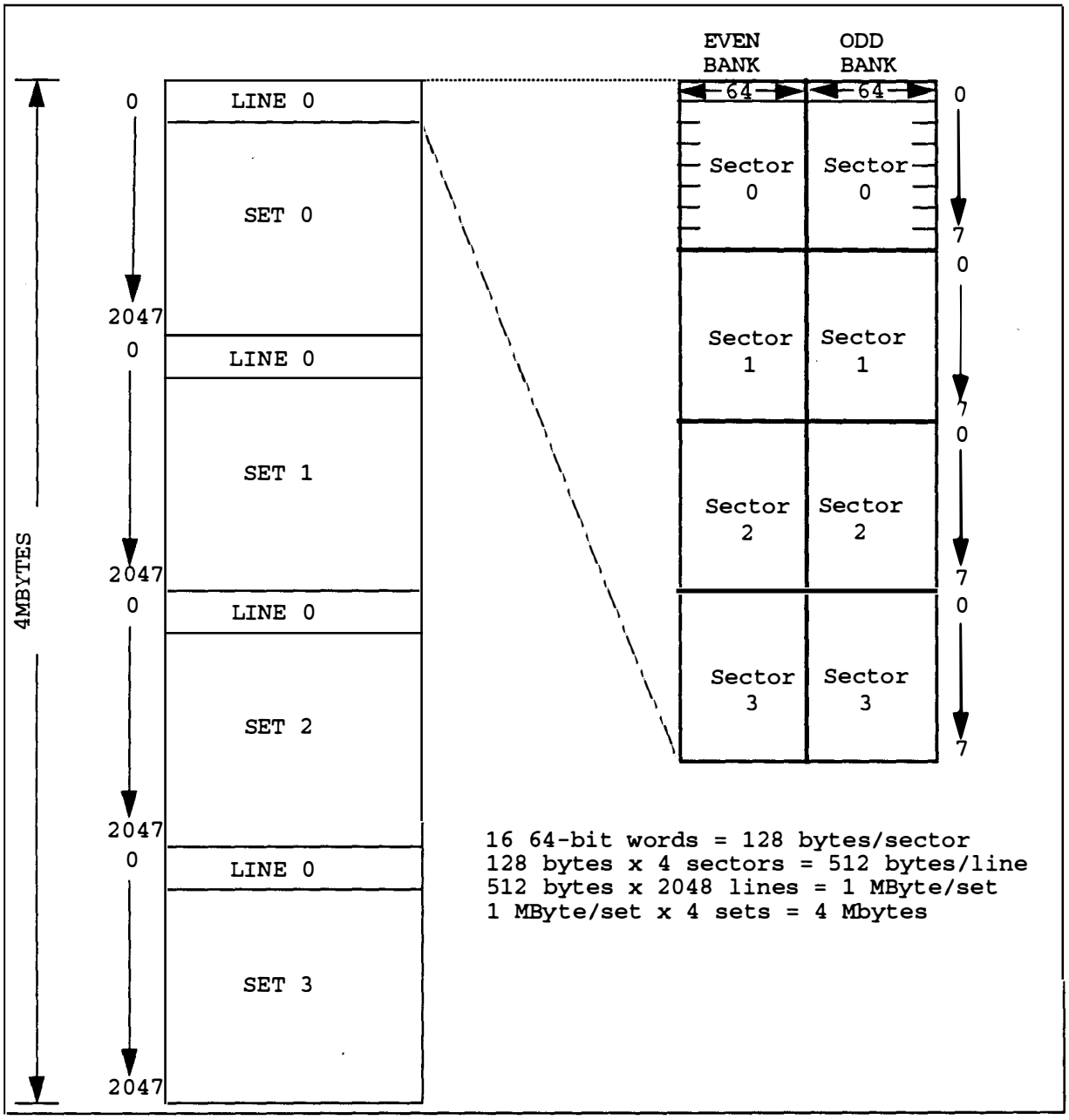


Figure 1-10 Streaming Cache Memory Architecture

REGISTERS

2

The R8000 Microprocessor Chip Set provides four sets of architecturally visible registers:

1) Thirty-two general purpose registers (GPR) located on the R8000, thirty-one of which are usable. Register (r0) always contains a value of zero. Each register is 64 bits wide. These registers are numbered r31..r0 and are used for virtual address generation as well as general movement and temporary storage of load and store data throughout the device.

2) Thirty-two Floating-Point Registers (FPR) located on the R8010 FPU. Each register is 64 bits wide.These registers are numbered f31..f0 and are used for general movement and temporary storage of load and store data throughout the device.

3) Thirty-two system control registers located on the R8000. Each register is 64 bits wide. These registers are accessible through the double Move To-From Coprocessor-0 instructions such as DMTCO and DMFCO. The 32 bit versions of these instructions, MTCO and MFCO, are not defined. The R8000 system control registers are defined as Coprocessor-0 (Cop0) registers.

4) Two floating point control registers located on the R8000 CPU. Each register is 32 bits wide. Although the architecture provides for thrity-two control registers, only two, registers 31 and 0 are visible. These registers are accessible through the Move To-From Coprocessor-0 instructions such as MTCO and MFCO. The 64 bit versions of these instructions, DMTCO and DMFCO, are not defined. The R8010 FPU system control registers are defined as Coprocessor-1 (Cop1) registers.

Table 2-1 below shows a listing of Coprocessor-0 and Coprocessor-1 registers.

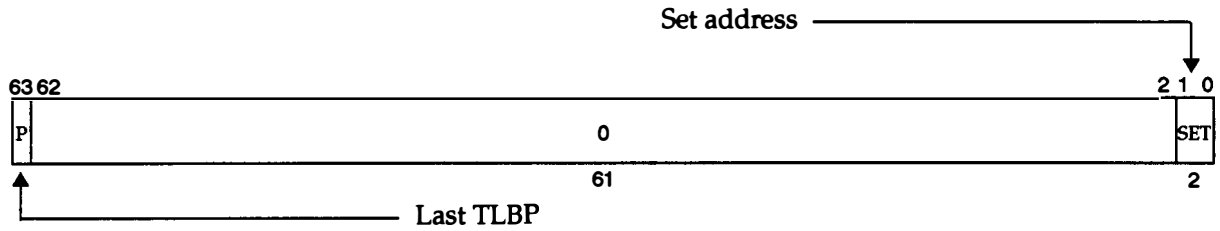
Cop#	Reg#	Mnemonic	Description
0	0	TLBSet	Select set in set-associative TLB
0	1	-----	Register not used
0	2	EntryLo	Low half of TLB entry
0	3	-----	Register not used
0	4	UBase	Pointer to User PTE table
0	5	ShiftAmt	Shift amount to align Virtual page number
0	6	TrapBase	Base address of trap vectors
0	7	BadPAddr	Bad Physical Address
0	8	VAddr	Virtual Address Register
0	9	Counts	Cycle and operation counters
0	10	EntryHi	High half of TLB entry
0	11	-----	Register not used
0	12	SR	Status Register
0	13	Cause	Reason for last exception
0	14	EPC	Exception Program Counter
0	15	PRId	Processor Revision Identifier
0	16	Config	Configuration register
0	17	——	Register not used
0	18	Work0	Uninterpreted temporary register
0	19	Work1	Uninterpreted temporary register
0	20	PBase	Pointer to Kernel Private PTE table
0	21	GBase	Pointer to Kernel Global PTE table
0	22-23	——	Register not used
0	24	Wired	Indicies of wired entries in the TLB
0	25-27	——	Reserved for additional Wired registers
0	28	DCache	Data Cache control register
0	29	ICache	Instruction Cache control register
0	30-31	——	Register not used
1	0	FConfig	Floating-point Configuration register
1	1-30	-----	Register not used
1	31	FSR	Floating-point Status Register

Table 2-1 Control Registers

2.1 COPROCESSOR 0 REGISTER SET

This section lists the 32 system control registers referred to as Coprocessor 0. The registers are listed in numerical order with their corresponding register number in parentheses. The following registers are reserved by MIPS technologies and should not be used: r1, r3, r11,r17, r22, r23, r25, r26, r27, r30, and r31. These registers do not appear in the following section.

2.1.1 TLBSet (r0)



P if set, the last TLBP operation was unsuccessful.
SET specifies the set select address within a TLB entry.

The TLBSet Register is a read-write register used to index a TLB entry's set and to provide access status as the result of a TLBP operation.

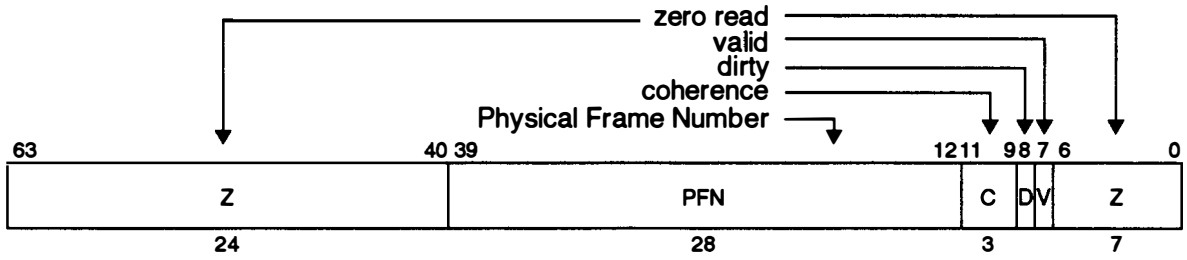
The SET field is used to select a TLB entry's set for a TLBW or a TLBR instruction. When a TLB Refill (User, Kernel Private, and Kernel Global) exception occurs, TLBSet is loaded with a random set to be replaced. When a TLB Invalid or TLB Modified exception occurs, TLBSet is loaded with the set which contains the virtual tag match. This value may be overwritten under program control to write to a specific set number.

The TLBSet register also contains status regarding the TLB Probe (TLBP) instruction execution. The P bit is set if the last TLBP instruction did not find a TLB entry which matched VADDR and the ASID value in the EntryHi register. If the last TLBP was successful, P=0 and SET holds the set number which matched. The format of the SET field is shown below.

00	Set 0
01	Set 1
10	Set 2
11	Reserved

The TLBSet register is undefined on reset.

2.1.2 EntryLo (r2)



Z are fields that may be written with anything but always read as 0

PFN Physical Frame Number
C specifies the page cache coherence algorithm
D if set, page is dirty and writable
V if set, entry is valid

The EntryLo register is a read-write register used to access the lower half of the TLB. EntryLo contains the Physical Page Number (PFN) and its associated Cache Algorithm (C), Write Permission (D), and Valid (V) state bits.

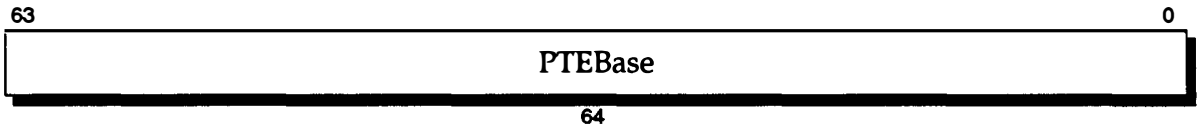
The C field encoding is as follows:

C Field	Cycle Type
000	Processor-ordered Uncachable
001	Reserved
010	Sequential-ordered Uncachable
011	Cachable Non-Coherent
100	Cachable Coherent Exclusive
101	Cachable Coherent Exclusive on Write
110	Reserved
111	Reserved (Cachable Write-through)

Table 2-2 Cache Coherency Field Encoding

The EntryLo register is undefined on reset.

2.1.3 UBase (r4)

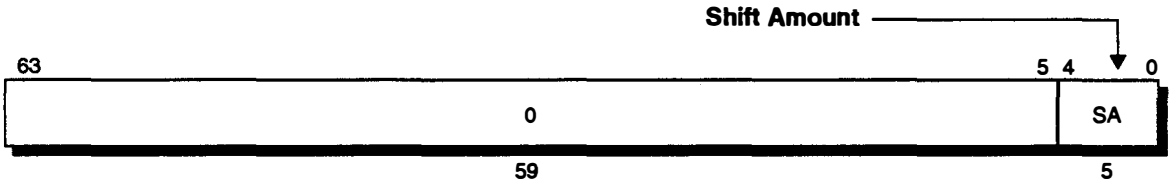


PTEBase Base address of Page Table Entries

The UBase register is a read-write register which holds the base address of the PTE table for the associated User region. The UBase, PBase, and GBase registers have identical formats.

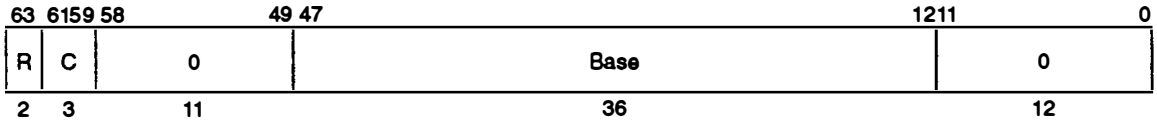
The UBase register is undefined on reset.

2.1.4 ShiftAmt (r5)



The ShiftAmt register is a read-only register that assists software in aligning pointers into page tables. In the User Region, right-shifting the VA register by the amount in the SA field correctly aligns the Virtual Page Number (VPN) field based on page size for the most recently failed translation.

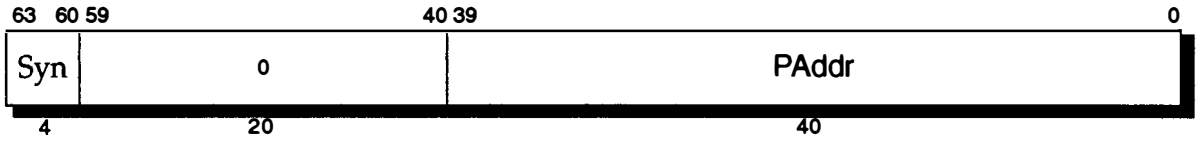
2.1.5 TrapBase (r6)



Base Base address of trap vectors
R Region bits of trap vectors
C Cache algorithm bits of trap vectors

The TrapBase is a read-write register which contains the base address of all exception vectors except Reset, Soft Reset, and NMI. When an exception occurs, the 12-bit exception vector offset is concatenated with the 36-bit Base and the R and C bits to form the new program counter.
This register is undefined on reset.

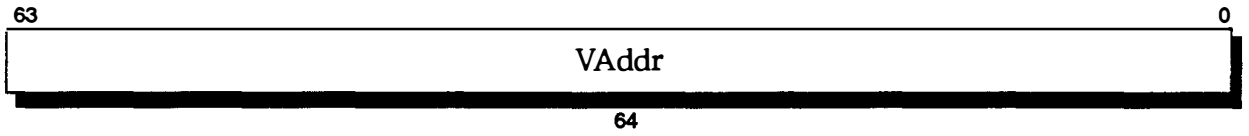
2.1.6 BadPAddr (r7)



BadPaddr Bad Physical Address
Syn Bits [15:12] of the virtual address

The BadPAddr register is a read-only register that contains the physical address which caused the virtual coherence error (floating) exception.

2.1.7 VAddr (r8)

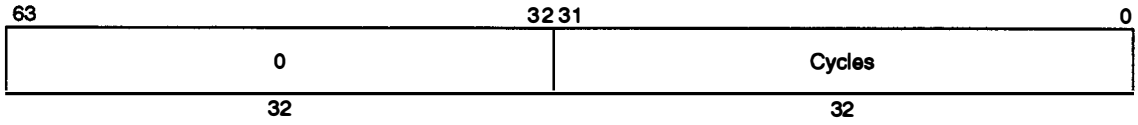


VAddr Virtual Address

The VAddr register is a read-write register that holds a 64-bit virtual address. VAddr is loaded both under software and hardware control. VAddr is loaded by hardware with the virtual address which causes a TLB fault, TLB Refill, TLB Invalid, TLB Modified, or Address Error Exception. VAddr is also writable by software, and is used to address the Cop0 instructions TLBW, TLBR, TLBP, DCTR, DCTW.

For TLB faults resulting from trying to fetch instructions for an instruction cache miss, VAddr is loaded with the virtual address of the begining of the instruction cache block, not with the address of the instruction which caused the instruction cache miss.

2.1.8 Counts (r9)

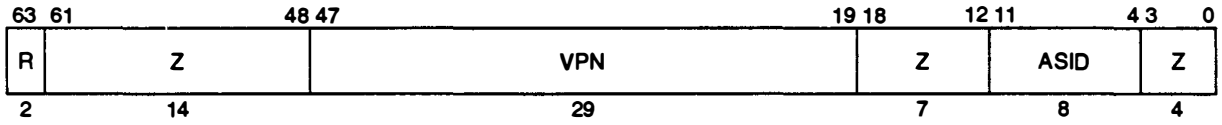


Cycles Count the number of processor clock cycles

The Counts register is a read-write register consisting of a 32-bit counter. The Cycles counter is incremented once per clock cycle. IP^{10} is wired to bit [31] of the Counts Register.

The Counts register is undefined on reset.

2.1.9 EntryHi (r10)



P	Two-bit Region (00 ≡ user, 01 ≡ KV0, 11 ≡ KV1).
Z	Fields that may be written with anything but always read as 0.
VPN	Virtual Page Number field.
ASID	Address Space Identifier.

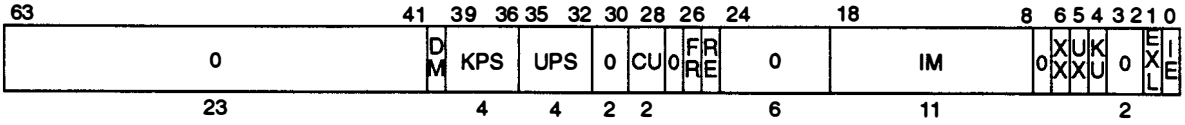
The EntryHi register is a read-write register used to access the upper half of the TLB. In addition, EntryHi contains the Address Space Identifier (ASID) used to match the virtual address with a TLB entry when virtual addresses are presented for translation.

When a TLB-related exception occurs, EntryHi is loaded with the Virtual Page Number (VPN) and the Region (R) of the virtual address that failed translation. The VPN field contains bits [47:19] of the faulting virtual address. It is *not* right justified according to page size. VPN[23:19] is conditionally set to zero by hardware on a per-bit basis based on page size. The ASID field already contains the Address Space Identifier for the virtual address which caused the exception, and so is not loaded when a exception occurs.

The VPN field does not contain bits [18:12] of the virtual address, for these are not stored in the TLB.

The EntryHi register is undefined on reset.

2.1.10 Status (r12)



IE	is the Interrupt Enable (0 ≡ disabled, 1 ≡ enabled).
EXL	is the Execution Level (0 ≡ normal, 1 ≡ exception).
KU	is the Execution Mode (0 ≡ kernel, 1 ≡ user).
UX	If set enables MIPS-III opcodes in user mode.
XX	If set enables SGI-extended opcodes in user mode.
IM	Interrupt Mask (0 ≡ disabled, 1 ≡ enabled).
RE	Reverse endian in user mode.
FR	enables additional floating-point registers (0 ≡ 16 registers, 1 ≡ 32 registers).
CU	controls the usability of coprocessors zero and one (0 ≡ unusable, 1 ≡ usable). Coprocessor zero is always usuable when in kernel mode, regardless of the setting of the CU_0 bit.
UPS	User Page Size.
KPS	Kernel Page Size.
DM	Floating-point precise exception Mode.
0is	Reserved for future use: 0 on read, must be 0 on write.

The SR register is a read-write register that contains the kernel/user mode, interrupt enable, and various other information.

Interrupts are enabled when IE=1 and EXL=0.

The base execution mode is set by the kernel/user bit (KU). The actual execution mode is modified by the execution level (EXL). The processor is in user mode when KU=1 and EXL=0, otherwise it is in kernel mode. The user instruction-set architecture is specified by the UX and XX field. Clearing XX inhibits SGI-extension opcodes. Clearing UX inhibits MIPS-III opcodes. All opcodes are permanently available in kernel mode.

The interrupt mask field (IM) is a 11-bit field that controls the enabling of the 11 maskable interrupt conditions. A maskable interrupt is taken if interrupts are enabled, and the corresponding bits are set in both the interrupt mask field of the SR and the interrupt pending field of the Cause register. Note that there are unmaskable interrupts as defined in the Cause Register.

The endian of the processor in kernel mode is set by BE bit in the Config

register. The Reverse Endian (RE) bit in the SR is used to modify the endian of the processor in user mode.

The FR bit enables additional registers in the floating-point coprocessor.

The Coprocessor Usuable (CU) field is a 2-bit field that controls whether coprocessor instructions will cause an exception. Regardless of the setting of the CU field, coprocessor zero is always usable when in kernel mode. The User Page Size (UPS) field specifies the page size of the User Virtual (UV) region of the address space. The Kernel Page Size (KPS) field specifies the page size of the Kernel Virtual (KV0 and KV1) regions of the address space. The encodings are as follows.

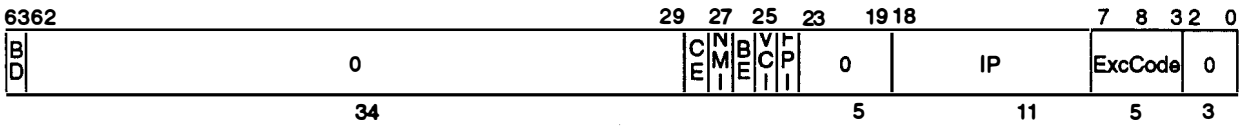
Page Size	Encoding
4K	0000
8K	0001
16K	0010
64K	0011
1M	0100
4M	0101
16M	0110
reserved	all others combinations

Table 2-3 UPS/KPS Field Encoding

The DM bit controls whether the floating-point unit is in performance or precise exception mode. DM=0 is performance mode, DM=1 is precise exception mode.

The Status register is initialized during reset.

2.1.11 Cause (r13)



BD	Last exception was taken while executing in a branch delay slot (0 normal, 1 delay slot).
CE	Coprocessor unit number is referenced when a Coprocessor Unusable exception is taken.
NMI	Non-maskable interrupt has occured.
BE	Bus Error pending
VCI	Coprocessor virtual coherence interrupt or TLBX pending.
FPI	Floating point exception has occurred.
IP	Interrupt pending.
ExcCode	Exception Code field (described below).

The Cause register is a read-write register that describes the nature of the last exception. A 5-bit exception code indicates the cause of the exception and the remaining fields contain detailed information relevant to the handling of certain types of exceptions.

The Interrupt Pending (IP) field indicates which maskable external, internal, coprocessor, and software interrupts are pending. These interrupts are maskable by the IM field in the Status reg.

$IP_{0..1}$ are software interrupts, and may be written into to set or reset software interrupts.

$IP_{2..7}$ are external interrupts which are set and cleared by transactions through the T-bus. Software cannot set or clear these bits.

IP_8 is the even bank G-cache parity error flag. This bit is set by hardware when a parity error is detected in the even bank and must be cleared by software.

IP_9 is the odd bank G-cache parity error flag. This bit is set by hardware when a parity error is detected in the odd bank and must be cleared by software.

IP_{10} is the cycle counter overflow flag. This flag is wired to the most significant bit of the Cycle counter.

The FPI is the floating-point exception flag. This flag is the logical and of the Enable and Flag fields of the FSR in the floating-point unit.

The VCI flag indicates a coprocessor virtual-coherence interrupt. Coprocessor loads and stores cause imprecise virtual-coherence exceptions which are reported as interrupts by setting this flag. The VCI flag is cleared by software.

The BE flag indicates a Bus Error is pending from the system interface. This flag is set or reset by transactions through the TBus. Software cannot set or clear this bit.

The NMI flag indicates a non-maskable interrupt. The NMI interrupt is not enabled or disabled by the EXL or IE fields of the Status Register. The NMI flag is set by transactions through the TBus. Software setting of this bit does not cause an interrupt.

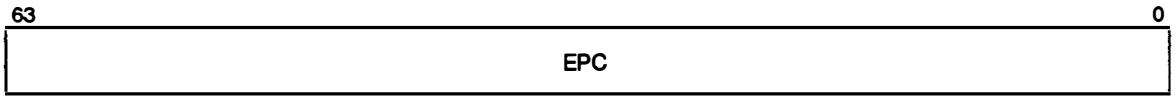
The Cause register is undefined on reset.

Table 2-4 shows a listing of the exception code fields

Number	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB Modification exception
2	TLBL	TLB exception (Load or instruction fetch)
3	TLBS	TLB exception (Store)
4	AdEL	Address Error exception (Load or instruction fetch)
5	AdES	Address Error exception (Store)
6-7	-----	Not used
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved Instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic Overflow exception
13	Tr	Trap exception (i.e. the instruction "trap")
14-31	-----	Not used

Table 2-4 Exception Codes

2.1.12 Exception Program Counter (r14)



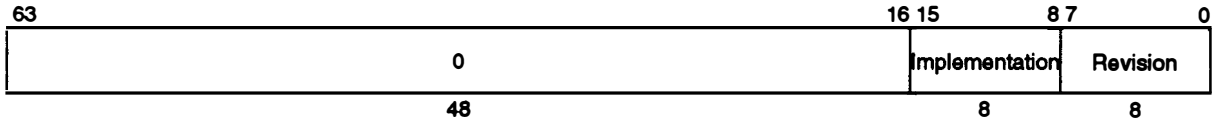
EPC Exception Program Counter

The EPC is a read-write register which contains the address at which instruction processing may resume after servicing an exception. For synchronous exceptions, the EPC register contains either the virtual address of the instruction which was the direct dause of the exception, or when that instruction is in a branch delay slot, the EPC contains the virtual address of the immediately preceeding branch or jump instruction and the Branch Delay (BD) bit in the Cause register is set.

For asynchronous exceptions the EPC points to where execution should resume.

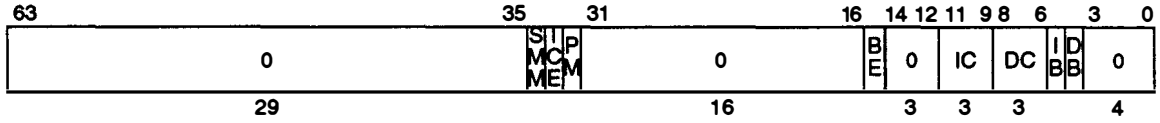
The EPC register is undefined on reset.

2.1.13 Process Revision Identifier (r15)



The PRId register is a read-only register containing the Process Revision Identifier for the R8000 Microprocessor.

2.1.14 Config (r16)



DB	Data cache block size = 2^{DB+4} bytes (ro) [1]
IB	Instruction cache block size = 2^{IB+4} bytes (ro) [1]
DC	Data cache size = 2^{DC+12} bytes (ro) [2]
IC	Instruction cache size = 2^{IC+11} bytes (ro) [3]
BE	Big endian memory (rw) [1]
PM	Parity mode: 0 ≡ even parity, 1 ≡ odd parity (rw) [0]
ICE	Inhibit Count during Exception (rw) [1]
SMM	Sequential Memory Model (rw) [1]

The Config register is a read-write register that specifies the various configuration options. Power-up values are shown in brackets [ ].

The instruction and data cache parameters are fixed by hardware and are displayed in the IC, IB, DC, and DB fields.

The endian of the memory system is set by the BE field.

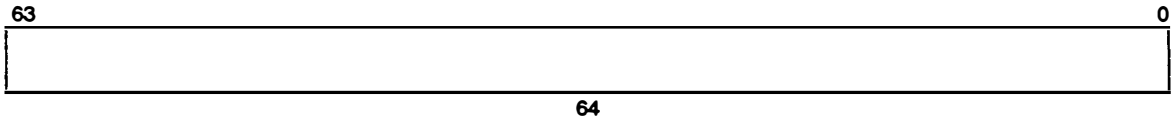
The Parity Mode (PM) bit specifies the mode of the parity error detection and generation. Even parity (PM=0) means all zeros including the parity bit is good parity; odd parity (PM=1) means all zeros including the parity bit is bad parity. Parity errors are reported via interrupt IP_8. Whether or not the error causes an exception is controlled by the interrupt masking mechanism of the Status Register.

The ICE flag disables the Count register when the processor is executing in exception level.

The SMM selects between the Sequential Memory Model and a Coprocessor Ordered Stores Model. Setting SMM causes integer and floating-point loads and stores to execute in order. Clearing SMM causes the processor to execute in "Coprocessor Order".

The configuration register is initialized during reset.

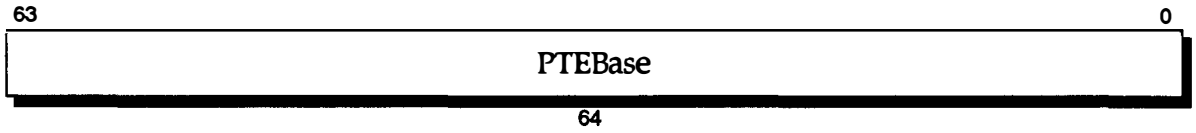
2.1.15 Work0 (r18), Work1 (r19)



The Work0 and Work1 registers are read-write registers for software use. The hardware does not interpret the contents of these registers.

Both registers are undefined on reset.

2.1.16 PBase (r20)

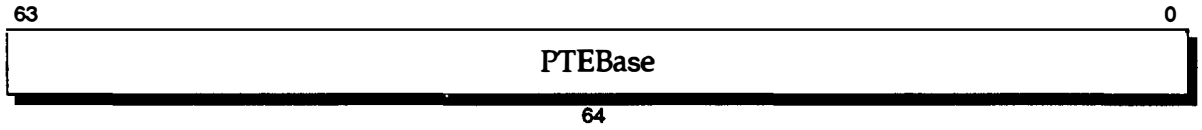


PTEBase Base address of Page Table Entries

The PBase register is a read-write register which holds the base address of the PTE table for the associated Kernel Private region. The UBase, PBase, and GBase registers have identical formats.

The PBase register is undefined on reset.

2.1.17 GBase (r21)

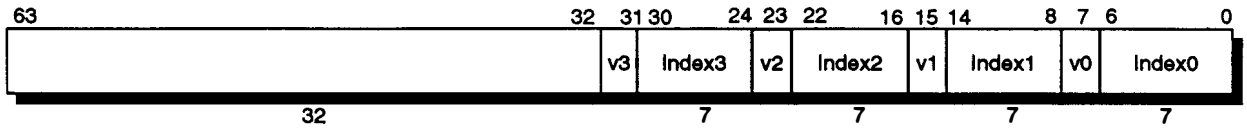


PTEBase Base address of Page Table Entries

The GBase register is a read-write register which holds the base address of the PTE table for the associated Kernel Global region. The UBase, PBase, and GBase registers have identical formats.

The UBase register is undefined on reset.

2.1.18 Wired (r24)



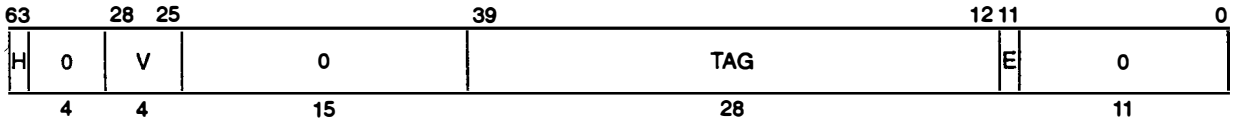
Index The seven bit TLB entry to be 'wired'
v Valid Bit set if corresponding Index is valid

The Wired register is a read-write register used to control the TLB replacement algorithm. Up to four entries may be wired down under program control. The four entries must be in different congruence classes.

The TLB is three-way set associative. Only set 0 may be 'wired'. When a TLB Refill exception occurs, the congruence class of the missing virtual address is compared to each of the four indices in the wired register. If a match is found for a valid entry in the wired register, a random value in the range 0..2 is loaded into the TLBSet register. If a valid match is not found, a random value in the range 1..2 is loaded into the TLBSet register.

The Wired register is undefined on reset.

2.1.19 DCache (r28)

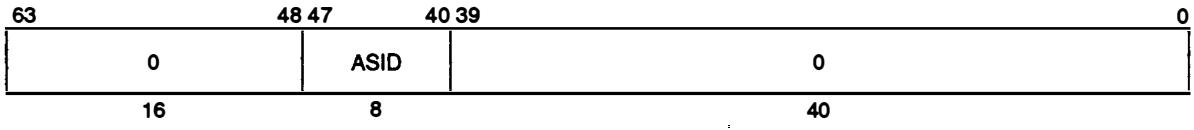


H	Interrogate Cache operation resulted in a hit
TAG	Physical tag field
V	Valid bits, one per 32-bit word
E	set if cache line is exclusively owned

The Data Cache register is a read-write register used to read and write the data cache tag in conjuction with the DCTR and DCTW instructions.

The DCache register is undefined on reset.

2.1.20 ICache (r29)



IASID Instruction cache Address Space Identifier

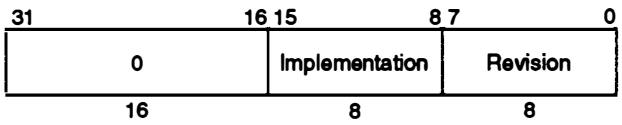
The Instruction Cache register is a read-write register used to store the Instruction Address Space Identifier.

The ICache register is undefined on reset.

2.2 COPROCESSOR 1 REGISTER SET

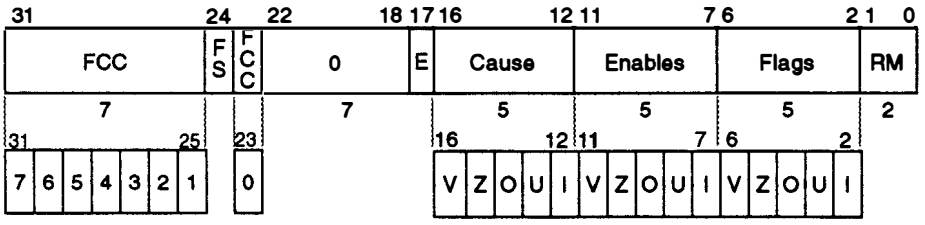
The CoProcessor 1 register and dedicated floating point registers located on the R8010 FPU. Although 32 register locations have been allocated for floating point operations, currently only two registers, f0 and f31, are used. Registers f1..f30 are reserved.

2.2.1 FConfig (f0)



The FConfig register is a 32 bit read-only register accessible by instructions running in kernel or user mode.The Implementation field is an 8-bit number that defines this particular implementation of the floating-point coprocessor. The Revision field is an 8-bit number that defines this particular revision of this implementation of the floating-point coprocessor.

2.2.2 Floating Point Status (f31)



FCC	Floating-point Condition Code
FS	Flush denormalized results to zero
RM	Rounding Mode
V	Invalid operation
Z	Division by zero
I	Inexact exception
O	Overflow exception
U	Underflow exception
E	Emulation exception

The Floating Point Status Register (FSR) register contains status and control information and is accessible by instructions running in either kernel or user mode. The FSR controls the arithmetic rounding mode and the enabling of user-mode traps, as well as indicating when exceptions have occurred.

A read and subsequent use of the FSR causes all previous instructions that have not been completed in the floating-point coprocessor's pipeline to be completed. Refer to chapter 5, section 5.3.2 for more information on synchronizing the floating-point coprocessor.

The contents of the FSR are unpredictable and undefined after a processor reset or a power-up event. Software should initialize this register.

The bit descriptions are as follows:

The RM field controls the rounding mode of all floating-point operations.

The Flags bits are cumulative and indicate that an exception was raised on some operation since the time which they were explicitly reset. Flag bits are set to 1 if an IEEE 754 exception is raised, and unchanged otherwise. The flag bits are never cleared as a side effect of floating-point operations, but

may be set or cleared by writing a new value into the status register using a "move to coprocessor control" instruction.

The Cause bits specify the exceptions raised by the last floating-point operation and cause an interrupt if the corresponding Enable bit is set. The Cause bits are written by each floating-point operation *in performance mode only.* Load, store, and move operations do not change the state of the **Cause** bits. The Cause bits are set to 0 or 1 to indicate the occurrence or non-occurrence of an IEEE 754 exception. Setting a Cause bit via a "move to coprocessor control" instruction causes an interrupt if the corresponding Enable bit is set, and always sets the corresponding flag bit. The precision of the interrupt is the same as that of a floating-point operation exception.

Unimplemented floating-point coprocessor opcodes cause a reserved instruction exception in the R8000 Microprocessor. All other FPU exceptions are reported via IP_{10} in the Cause register.

The Enable bits control whether a floating-point exception should cause an interrupt. The precision of the interrupt is dependent on which mode the floating-point unit is running in.

Setting the FS bit to 0 in precise exception mode causes operations involving denormalized numbers to be handled by the kernel. Setting the FS bit to 1 in precise exception mode causes denormalized operands and results that would be denormalized to be flushed to zero instead of causing an unimplemented operation exception. The state of the FS bit in performance mode is irrelevant. The machine acts as if the bit is set to 1.

The FCC bits are the floating-point condition codes. FCC[0] is the same as the "c" bit in the MIPS-III architecture. FCC[1]..FCC[7] are new condition code bits. Any one of bits FCC[0]..FCC[7] can be written by the "floating-point compare" instruction.

The E bit of the FSR indicates unimplemented operand exceptions. Like the Flag bits, the E bit is cumulative and indicates that an exception was raised on some operation since the time it was explicitly reset.

MIPS IV INSTRUCTION SET SUMMARY

3

The R8000 Microprocessor Chip Set runs the MIPS IV instruction set, which is a superset of the MIPS III instruction set and backward compatible. The additions of these new instructions enables the MIPS architecture to compete in the high-end numeric processing market which has traditionally been dominated by vector architectures.

A set of compound multiply-add instructions has been added, taking advantage of the fact that the majority of floating point computations use the chained multiply-add paradigm. The operator for the multiply-add instructions is not defined by the IEEE and does not perform intermediate rounding. Eliminating the intermediate rounding step allows for a lower inherent latency and has higher precision and higher performance than an operator which performs intermediate rounding.

A register + register addressing mode for floating point loads and stores has been added which eliminates the extra integer add required in many array accesses. Register + register addressing for integer memory operations is not supported.

A set of four conditional move operators allows floating point arithmetic 'IF' statements to be represented without branches. 'THEN' and 'ELSE' clauses are computed unconditionally and the results placed in a temporary register. Conditional move operators then transfer the temporary results to their true register. Table 3-1 lists in alphabetical order the new instructions which comprise the MIPS IV instruction set.

Instruction	Definition
BC1F	Branch on FP Condition Code False
BC1T	Branch on FP Condition Code True
BC1FL	Branch on FP Condition Code False Likely
BC1TL	Branch on FP Condition Code True Likely
C.cond.fmt (cc)	Floating Point Compare
LDXC1	Load Double Word indexed to COP1
LWXC1	Load Word indexed to COP1
MADD.sd	Floating PointMultiply-Add
MOVF	Move conditional on FP Condition Code False
MOVN	Move on Register Not Equal to Zero
MOVT	Move conditional on FP Condition Code True
MOVZ	Move on Register Equal to Zero
MOVF.fmt	FP Move conditional on Condition Code False
MOVN.fmt	FP Move on Register Not Equal to Zero
MOVT.fmt	FP Move conditional on Condition Code True
MOVZ.fmt	FP Move conditional on Register Equal to Zero
MSUB.sd	Floating Point Multiply-Subtract
NMADD.sd	Floating Point Negative Multipy-Add
NMSUB.sd	Floating Point Negative Multiply-Subtract
PFETCH	Prefetch Indexed --- Register + Register
PREF	Prefetch --- Register + Offset
RECIP.fmt	Reciprocal Approximation
RSQRT.fmt	Reciprocal Square Root Approximation
SDXC1	Store Double Word indexed to COP1
SWXC1	Store Word indexed to COP1

Table 3-1 MIPS IV Instruction Set Additions and Extensions

Table 3-2 lists the COP0 instructions for the R8000 microprocessor. COP0 instructions are those which are not architecturally visible and are used by the kernel.

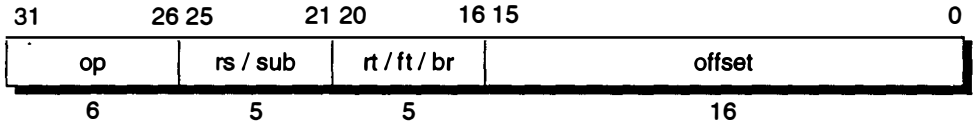
COP0 Instruction	Definition
ERET	Return from Exception
TLBP	Probe for TLB Entry
TLBR	Read TLB Entry
TLBW	Write TLB Entry
DCTR	Data Cache Tag Read
DCTW	Data Cache Tag Write

Table 3-2 R8000 COP0 Instructions

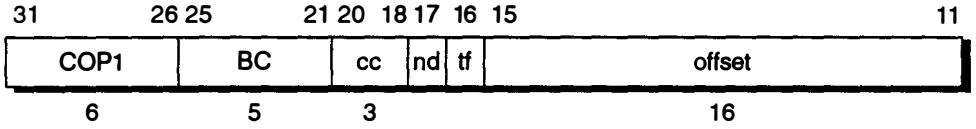
3.1 INSTRUCTION FORMATS

Each CPU instruction consists of a single 32-bit word, aligned on a word boundary. There are three instruction formats—immediate (I-type), jump (J-type), and register (R-type)—as shown in Figure 3-1. The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated (and less frequently used) operations and addressing modes from these three formats as needed.

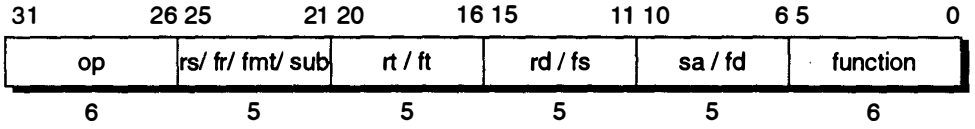
I-type (Immediate)



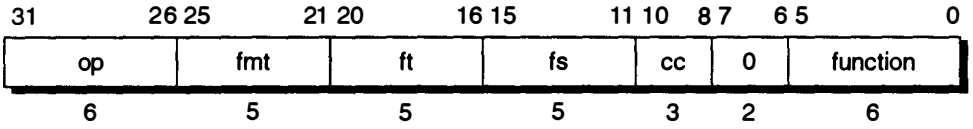
CI-type (Floating-point condition-code I-type)



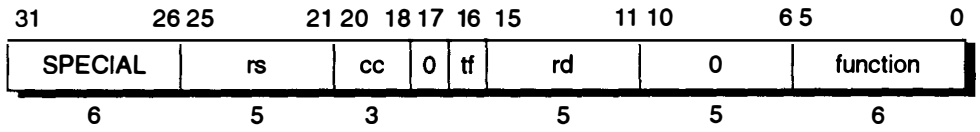
R-type (Register)_



RC-type (Register to floating-point condition code)



CR-type (Floating-point condition-code R-type)



J-type (Jump)

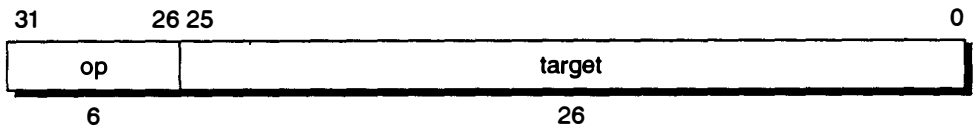


Figure 3-1 Instruction Formats

3.2 LOAD AND STORE INSTRUCTIONS

Load and store are immediate (I-type) instructions that move data between memory and the general registers. The only addressing mode that load and store instructions directly support is *base register plus 16-bit signed immediate offset.*

3.2.1 Scheduling a Load Delay Slot

A load instruction that does not allow its result to be used by the instruction immediately following is called a d*elayed load instruction.* The instruction slot immediately following this delayed load instruction is referred to as the l*oad delay slot.* In the R8000 Microprocessor the instruction immediately following a load instruction can use the contents of the loaded register, however in such cases hardware interlocks insert additional real cycles. Consequently, scheduling load delay slots can be desirable, both for performance and maintaining R4x00-Series microprocessor compatibility. However, the scheduling of load delay slots is not absolutely required.

The data from a load instruction is available for use by an instruction issued in the cycle after the load instruction. Because the R8000 is a super-scalar microprocessor instruction scheduling to improve performance may cause instructions to be inserted between the load and the use.

3.2.2 Defining Access Types

Access type indicates the size of a R8000 Microprocessor data item to be loaded or stored, set by the load or store instruction opcode. Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a big-endian configuration, the low-order byte is the most-significant byte; for a little-endian configuration, the low-order byte is the least-significant byte.

The access type, together with the three low-order bits of the address, define the bytes accessed within the addressed doubleword. Only the combinations shown in Figure 3-2 are permissible; other combinations cause address error exceptions.

Access Type Mnemonic (*Value*)	Low Order Address Bits			Bytes Accessed															
				Big endian (63——31——0) Byte								Little endian (63——31——0) Byte							
	2	1	0																
Doubleword (7)	0	0	0	0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
Septibyte (6)	0	0	0	0	1	2	3	4	5	6			6	5	4	3	2	1	0
	0	0	1		1	2	3	4	5	6	7	7	6	5	4	3	2	1	
Sextibyte (5)	0	0	0	0	1	2	3	4	5					5	4	3	2	1	0
	0	1	0			2	3	4	5	6	7	7	6	5	4	3	2		
Quintibyte (4)	0	0	0	0	1	2	3	4							4	3	2	1	0
	0	1	1				3	4	5	6	7	7	6	5	4	3			
Word (3)	0	0	0	0	1	2	3									3	2	1	0
	1	0	0					4	5	6	7	7	6	5	4				
Triplebyte (2)	0	0	0	0	1	2											2	1	0
	0	0	1		1	2	3									3	2	1	
	1	0	0					4	5	6			6	5	4				
	1	0	1						5	6	7	7	6	5					
Halfword (1)	0	0	0	0	1													1	0
	0	1	0			2	3									3	2		
	1	0	0					4	5					5	4				
	1	1	0							6	7	7	6						
Byte (0)	0	0	0	0															0
	0	0	1		1													1	
	0	1	0			2											2		
	0	1	1				3									3			
	1	0	0					4							4				
	1	0	1						5					5					
	1	1	0							6			6						
	1	1	1								7	7							

Figure 3-2 Byte Access within a DoubleWord

3.3 COMPUTATIONAL INSTRUCTIONS

Computational instructions can be either in register (R-type) format, in which both operands are registers, or in immediate (I-type) format, in which one operand is a 16-bit immediate.
Computational instructions perform the following operations on register values:

- arithmetic
- logical
- shift
- multiply
- divide

These operations fit in the following four categories of computational instructions:

- ALU Immediate instructions
- three-Operand Register-Type instructions
- shift instructions
- multiply and divide instructions

For word-oriented ALU operations all operands must be 32-bit sign extended. The result of operations that use incorrect sign-extended-bit values is unpredictable.

3.4 JUMP AND BRANCH INSTRUCTIONS

Jump and branch instructions change the control flow of a program. All jump and branch instructions occur with a delay of one instruction: that is, the instruction immediately following the jump or branch (this is known as the instruction in the *delay slot*) always executes while the target instruction is being fetched from storage. Branches are predicted when the instruction quad is fetched and have no penalty. A branch mis-prediction incurs a 3 cycle penalty.

3.4.1 Overview of Jump Instructions

Subroutine calls in high-level languages are usually implemented with Jump or Jump and Link instructions, both of which are J-type instructions. In J-type format, the 26-bit target address shifts left 2 bits and combines with the high-order 4 bits of the current program counter to form an absolute address.

Returns, dispatches, and large cross-page jumps are usually implemented with the Jump Register or Jump and Link Register instructions. Both are R-type instructions that take the 32-bit or 64-bit bit address contained in one of the general purpose registers.

3.4.2 Overview of Branch Instructions

All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit *offset* (shifts left 2 bits and is sign-extended to 32 bits). All branches occur with a delay of one instruction. If a conditional "branch likely" is not taken, the instruction in the delay slot is nullified.

3.5 COPROCESSOR INSTRUCTIONS

Coprocessor instructions perform operations in their respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor-dependent formats. CP0 instructions perform operations specifically on the System Control Coprocessor registers to manipulate the memory management and exception handling facilities of the processor.

3.6 SUMMARY OF INSTRUCTION SET ADDITIONS

The following is a brief description of the additions to the MIPS III instruction set. These additions comprise the MIPS IV instruction set.

3.6.1 Indexed Floating Point Load

LWXC1 - Load word indexed to Coprocessor 1.
LDXC1 - Load doubleword indexed to Coprocessor 1.

The two Index Floating Point Load instructions are exclusive to the MIPS IV instruction set and transfer floating-point data types from memory to the floating point registers using register + addressing mode. There are no indexed loads to general registers. The contents of the general register specified by the base is added to the contents of the general register specified by the index to form a virtual address. The contents of the word or doubleword specified by the effective address are loaded into the floating point register specified in the instruction.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. Also, if the address is not aligned, an address exception occurs.

3.6.2 Indexed Floating Point Store

SWXC1 - Store word indexed to Coprocessor 1.
SDXC1 - Store doubleword indexed to Coprocessor 1.

The two Index Floating Point Store instructions are exclusive to the MIPS IV instruction set and transfer floating-point data types from the floating point registers to memory using register + addressing mode. There are no indexed loads to general registers. The contents of the general register specified by the base is added to the contents of the general register specified by the index to form a virtual address. The contents of the floating point register specified in the instruction is stored to the memory location specified by the effective address.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. Also, if the address is not aligned, an address exception occurs.

3.6.3 Prefetch

PREF - Register + offset format
PFETCH Indexed - Register + register format

The two prefetch instructions are exclusive to the MIPS IV instruction set and allow the compiler to issue instructions early so the corresponding data can be fetched and placed as close as possible to the CPU. Each instruction contains a 5-bit 'hint' field which gives the coherency status of the line being prefetched. The line can be either shared, exclusive clean, or exclusive dirty. The contents of the general register specified by the base is added either to the 16 bit sign-extended offset or to the contents of the general register specified by the index to form a virtual address. This address together with the 'hint' field is sent to the cache controller and a memory access is initiated.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. The prefetch instruction never generates TLB-related exceptions. The PREF instruction is considered a standard processor instruction while the PFETCH instruction is considered a standard Coprocessor 1 instruction. Refer to section 1.2.10 for more information on the prefetch instruction.

3.6.4 Branch on Floating Point Coprocessor

BC1T - Branch on FP condition True
BC1F - Branch on FP condition False
BC1TL - Branch on FP condition True Likely
BC1FL - Branch on FP condition False Likely

The four branch instructions are upward compatible extensions of the Branch on Floating point Coprocessor instructions of the MIPS instruction set. The BC1T and BC1F instructions are extensions of MIPS I. BC1TL and BC1FL are extensions of MIPS III. These instructions test the floating point condition codes. If no condition code is specified then condition code bit zero is selected. This encoding is downward compatible with previous MIPS architectures.

The branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit offset, shifted left two bits and sign-extended to 64 bits. If the contents of the floating point condition code specified in the instruction are equal to the test value, the target address is branched to with a delay of one instruction. If the conditional branch is not taken and the nullify delay bit in the instruction is set, the instruction in the branch delay slot is nullified.

3.6.5 Integer Conditional Moves

MOVT - Move conditional on condition code true
MOVF - Move conditional on condition code false
MOVN - Move conditional on register not equal to zero
MOVZ - Move conditional on register equal to zero

The four integer move instructions are exclusive to the MIPS IV instruction set and are used to test a condition code or a general register and then conditionally perform an integer move. The three bit floating point condition code specified in the instruction, or the 5 bit general register specifier, is compared to zero. If the result indicates that the move should be performed, the contents of the specified source register is copied into the specified destination register.

3.6.6 Floating Point Multiply-Add

MADD - Floating Point Multiply-Add
MSUB- Floating Point Multiply-Subtract
NMADD - Floating Point Negative Multiply-Add
NMSUB - Floating Point Negative Multiply-Subtract

These four instructions are exclusive to the MIPS IV instruction set and accomplish two floating point operations with one instruction. Refer to section 1.2.8 for more information.

3.6.7 Floating Point Compare

C.cond - Compare
C.cond - Implies cc=0

The two compare instructions are upward compatible extensions of the floating point compare instructions of the MIPS I instruction set and produce a boolean result which is stored in one of the condition codes.

The contents of the two FP source registers specified in the instruction are interpreted and arithmetically compared. A result is determined based on the comparison and the conditions specified in the instruction. If one of the values is not a number and the high order bit of the condition field is set, an invalid operations trap occurs. Comparisons are exact and neither overflow or underflow.

Timing restrictions exist for these instructions. The contents of the destinations condition code specified in the instruction is immediately available only within the R8010 FPU. A one-instruction delay is provided to propogate the condition code to the R8000 Microprocessor. The value of the condition code is undefined during this one instruction and no hardware interlock detection mechanism is provided.

The implications for compiler code scheduling is that a compare instruction may be immediately followed by a dependent floating point conditional move instruction, but may not be immediately followed by a dependent branch on floating point coprocessor condition instruction or a dependent integer conditional move instruction. Note that this restriction applies only to the condition code specified in the 3-bit condition code specifier of the instruction. All other condition codes are unaffected

3.6.8 Floating Point Conditional Moves

MOVT.fmt - Floating Point Conditional Move on condition code true
MOVF.fmt - Floating Point Conditional Move on condition code false
MOVN.fmt - Floating Point Conditional Move on register not equal to zero
MOVZ.fmt - Floating Point Conditional Move on register equal to zero

The four floating point conditional move instructions are exclusive to the MIPS IV instruction set and are used to test a condition code or a general register and then conditionally perform a floating point move. The three bit floating point condition code

specified in the instruction, or the 5 bit general register specifier, is compared to zero. If the result indicates that the move should be performed, the contents of the specified source register is copied into the specified destination register. All of these conditional floating point move operations are non-arithmetic. Consequently, no IEEE 754 exceptions occur as a result of these instructions.

3.6.9 Reciprocal's

RECIP.fmt - Reciprocal Approximation
RSQRT.fmt - Reciprocal Square Root Approximation

The reciprocal instruction performs a reciprocal approximation on a floating point value. The reciprocal of the value in the floating point source register is approximated and placed in a destination register. The numerical accuracy of this operation is implementation dependent based on the rounding mode used.

The reciprocal square root instruction performs a reciprocal square root approximation on a floating point value. The reciprocal of the positive square root of a value in the floating point source register is approximated and placed in a destination register. The numerical accuracy of this operation is implementation dependent based on the rounding mode used.

The approximation is due to the fact that neither of these instruction meets IEEE accuracy requirements. In both cases a small amount of precision has been sacrificed, thereby significantly reducing execution time. For example, in the case of a RECIP instruction, X/Y is computed by taking the reciprocal of Y and multiplying that result by X. The reduced execution time of the reciprocal operation allows a RECIP followed by a MUL (multiply) instruction to be executed faster than a single DIV (divide) instruction. The performance difference between a RSQRT instruction and a SQRT followed by a DIV instruction is implementation dependent.

Refer to appendix A for more information on the RECIP and RSQRT instructions.

Table 3-3 shows the integer instruction latencies in the R8000 Microprocessor.

Instruction Group	Latency	Dispatch
Arithmetic and Logical	1	2/cycle
Shift	1	1/cycle
Load	1	2/cycle
Store	N/A	1/cycle
Multiply (32-bit)	4	1/cycle
Multiply (64-bit)	6	1/cycle

Table 3-3 R8000 Integer Instruction Latencies

Table 3-4 shows the floating point instruction latencies of the R8000 CPU.

Instruction Group	Latency	Dispatch
Load	0	2/cycle
Store	N/A	2/cycle
Compare	1	2/cycle
Absolute	1	2/cycle
Negative	1	2/cycle
Move	1	2/cycle
Conditional Moves	1	2/cycle
Add	4	2/cycle
Subtract	4	2/cycle
MADD	4	2/cycle
DIV.s	14	1*
DIV.d	20	1*
SQRT.s	14	1*
SQRT.d	23	1*

Table 3-4 R8000 Floating Point Instruction Latencies

Instruction Group	Latency	Dispatch
RECIP.s	8	1*
RECIP.d	14	1*
RSQRT.s	8	1*
RSQRT.d	17	1*
mtc1, dmtc1	***	1/cycle
mfc1, dmfc1	N/A	2**

* Functional unit is busy for Latency-3 cycles.
** Holds up FP Dispatch unit for the next 3 cycles.
*** May incur a floating point resynchronization delay

Table 3-4 R8000 Floating Point Instruction Latencies

MEMORY MANAGEMENT

4

The R8000 Microprocessor provides a full-featured memory management unit (MMU) which uses an on-chip translation lookaside buffer (TLB) to translate virtual addresses to physical addresses. This chapter describes the processor virtual and physical address spaces, virtual to physical address translation, operation of the TLB, and system control registers which provide the software interface to the TLB.

The R8000 Microprocesor supports a 48 bit paged virtual address space. The physical address is 40 bits. The R8000 Microprocessor Chip Set implements the 64 bit MIPS IV instruction set which supports 32-bit user-mode applications as a proper subset. *The R8000 Microprocessor does not support 32-bit kernels.*

Both forward-mapped and reverse-mapped virtual memory management schemes are supported. Multiple page sizes are supported on a per process basis.

4.1 ADDRESS SPACE

The R8000 Microprocessor provides a 48 bit virtual address and a 40 bit physical address. The maximum user process size is 128 terabytes (2^{48}). The virtual address is encoded into 64 bits as shown in Table 4-1.

63 62	61 48	47 0
Region	Filler	Virtual Page Number (VPN) and Offset

Table 4-1 Virtual Address

The virtual address shown in Table 4-1 consists of a 2 bit region field, a 14 bit filler, and a 48 bit virtual page number concatenated with a page offset. The virtual address space is divided into four regions as defined by the region bits VA[63:62]. Table 4-8 shows how the virtual address space is divided.

Virtual Address Space	Region [63:62]	Beginning Address	Ending Address
Kernel Virtual (KV1) KV1 synonyms	11 11	0x_ffff_0000_0000_0000 0x_c000_0000_0000_0000	0x_ffff_ffff_ffff_ffff 0x_c000_ffff_ffff_ffff
Kernel Physical (KP)	10	0x_8000_0000_0000_0000	0x_b000_ffff_ffff_ffff
Kernel Virtual (KV0)	01	0x_4000_0000_0000_0000	0x_4000_ffff_ffff_ffff
User Virtual (UV)	00	0x_0000_0000_0000_0000	0x_0000_ffff_ffff_ffff

Table 4-2 Virtual Address Divisions

The R8000 Microprocessor supports only kernel and user mode address spaces. *No supervisor modes exists.* The KV1, KV0, and KP address spaces shown in Table 4-8 are accessible only in kernel mode. User space is defined with the region bits = 00 as shown.

The address map is defined such that, with the exception of the KV1 and KP address spaces, all of the 14 filler bits (VA[61:48]) must be zero. KP is the only address space where the filler bits can be a combination of ones and zeros. Non-zero filler bits in either the UV or KV0 address spaces shown in Table 4-2 constitutes an illegal access and generates an address error exception. Figure 4-1 shows where the illegal access areas reside in the address map.

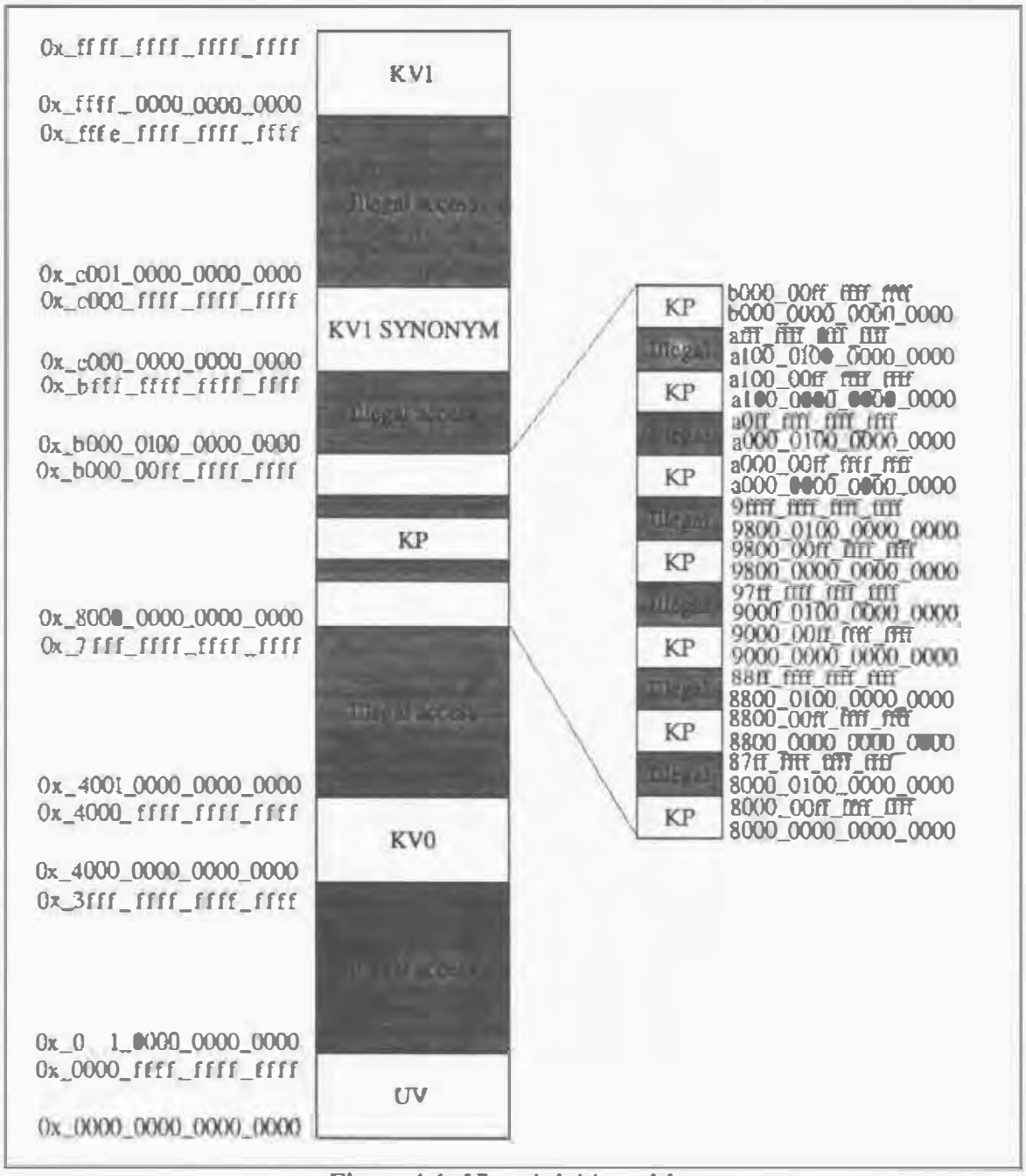


Figure 4-1 Virtual Address Map

4.1.1 User Virtual (UV) Space

The region bits VA[63:62] determine which of the four regions of virtual address space is being accessed. The User Virtual space is accessed when both regions bits are zero. When the operating system starts a user process in UV space the process can only generate addresses where the region bits are both zero. Any other combination of region bits causes an address exception error indicating that the program is attempting to access memory outside the designated range. In this way the region bits act as a protection scheme. User Virtual addresses are extended with an 8-bit Address Space Identifier (ASID) and are mapped by the TLB. Refer to section 4.2 for more information on the ASID.

4.1.2 Kernel Virtual 0 (KV0) Space

The KV0 space is defined as 'kernel private' virtual address space. Virtual addresses in KV0 space are extended by the 8-bit ASID value contained in the **EntryHi** register. KV0 space is used by the kernel for accesing virtual address space on behalf of a specific process. Both the TLB and the eight bit ASID are used in KV0 space.

4.1.3 Kernel Virtual 1 (KV1) Space

The KV1 space is defined as 'kernel global' virtual address space. In contrast to KV0 space, virtual addresses in KV1 space are not extended with an 8-bit ASID. In KV1 space the kernel can access global virtual address space independent of any specific process. KV1 space is accessible with the filler bits set to all zero's (see section 4.1.4 below) or all ones to facilitate the use of negative-offset addressing from register r0. Both forms of the address are treated as the same virtual address. The region bits, the virtual page number (VPN), and the offset are exactly the same.

4.1.4 Kernel Virtual 1 (KV1) Synonyms

As stated in section 4.1.3, KV1 and KV1 synonyms are treated as the same virtual address. KV1 synonyms are provided to allow negative addressing using register r0. In KV1 space the TLB is used but not the ASID.

4.1.5 Kernel Physical Space

In the kernel physical (KP) address space the virtual and physical addresses are

considered the same. No address translation is performed in KP space. The processor simply takes the lower 40 bits of virtual address and places them on physical address bits 39:0. The physical address comes directly from the virtual address generation units in the R8000 Microprocessor. KP space allows physical addresses to be generated without TLB intervention.

Note in Figure 4-1 that KP is the only space where the filler bits may be a combination of ones and zeros. The uppermost three filler bits (VA[61:59]) are used to divide the KP address space into subspaces, each having a different cachability attribute. Attempting to access memory with virtual address outside of these sub-ranges results in an address error exception . Table 4-3 shows how the KP address space is divided.

Coherence	Filler [61:59]	Beginning Address	Ending Address
Cachable Coherent Exclusive on Write	101	0x_a800_0000_0000_0000	0x_a800_00ff_ffff_ffff
Cachable Coherent Exclusive	100	0x_a000_0000_0000_0000	0x_a000_00ff_ffff_ffff
Cachable Non-Coherent	011	0x_9800_0000_0000_0000	0x_9800_00ff_ffff_ffff
Uncached Sequential Ordered	010	0x_9000_0000_0000_0000	0x_9000_00ff_ffff_ffff
Uncached Co-Processor Ordered	000	0x_8000_0000_0000_0000	0x_8000_00ff_ffff_ffff

Table 4-3 Division of the KP address space

The fields in Table 4-3 define the coherency attributes and are necessary because in KP space no address translation is performed. In each of the other three address spaces (KV1, KV0, and UV) the coherency field is stored in the TLB. Since address translation is not performed when generating KP addresses, virtual address bits 61:59 are used for this purpose. This is why KP address space is the only portion of the address map in which the filler bits (61:59) can be a combination of ones and zeros. Table 4-4 gives a summary of the various address spaces.

Virtual Address Space	Region [63:62]	Uses TLB	Uses ASID
Kernel Virtual (KV1)	11	Yes	No
Kernel Physical (KP)	10	No	No
Kernel Virtual (KV0)	01	Yes	Yes
User Virtual (UV)	00	Yes	Yes

Table 4-4 Address Spaces and TLB Usage

4.2 ADDRESS SPACE IDENTIFIERS

Kernel Private (KV0) and User Virtual (UV) virtual address spaces are extended with an 8-bit ASID value to reduce the frequency of TLB and instruction cache flushing on a context switch. The existence of the ASID allows multiple processes to exist simultaneously in both the TLB and the instruction caches. There are actually two distinct ASID values for a given process. The instruction cache ASID (IASID) is stored in the **ICache** register and is compared to the ASID value in the instruction cache tag during an instruction cache access. The TLB ASID (TASID) is stored in the **EntryHi** register and is compared to the ASID value in the Virtual tag (VTAG) portion of the TLB on a TLB access. The IASID and TASID values are independent of one another. Having different ASID values for a given process allows the kernel to independently flush and control the instruction and TLB caches.

4.3 REGISTER ADDRESSING MODES

The R8000 Microprocessor provides two addressing modes: **Register + Register** and **Register + Immediate**. Addresses are derived by adding an offset to a base register. The offset value can either come from a register or the immediate field of the instruction. This section discusses how these modes are used and how the actual address is generated.

4.3.1 Register + Register Addressing

When register + register addressing mode is used both portions of the address come from the General Purpose Register (GPR). The values are read from the two registers in the D-stage of the pipeline and are added together in A-stage to form a 64 bit virtual address. Since the R8000 Microprocessor can perform two address generations per cycle, a total of four registers are read from the GPR at the same time and used to construct the two 64-bit virtual addresses.

4.3.2 Register + Immediate Addressing

In register + immediate mode a portion of the address comes from the GPR with the remaining portion coming from the immediate field of the instruction. The base register is read in the D-stage of the pipeline and is added to the immediate value in the A-stage to form a 64 bit virtual address. The values are not concatenated. Since the R8000 Microprocessor can perform two address generations per cycle, two registers are read from the GPR at the same time. Each register value is added to the corresponding immediate field from the instruction to construct the two 64-bit virtual addresses.

4.3.3 Region Bits, the Base Register, and Legal Addresses

There is one restriction on the generation of legal virtual addresses. For both register + register and register + immediate addressing, it is required that the region of the generated virtual address be the same as the region of the base register. The base region cannot change as a result of the addition of the offset. If the region of the generated virtual address does not equal the region of the base address, an address error exception occurs. However, there is one exception to this rule. If the processor is operating in kernel mode it is legal to use a negative offset with register r0 as the base register to generate an address into KV1 space. If register r0 is the base portion of the address and the offset is negative then the region bits are both forced to ones which places the machine in KV1 space. If for some reason a user process (UV space) tries to generate an address using this technique, the region bits will mis-compare and an address exception error will occur. This address generation technique is used by the kernel to allow address generation without having to use any of the registers allocated to that process.

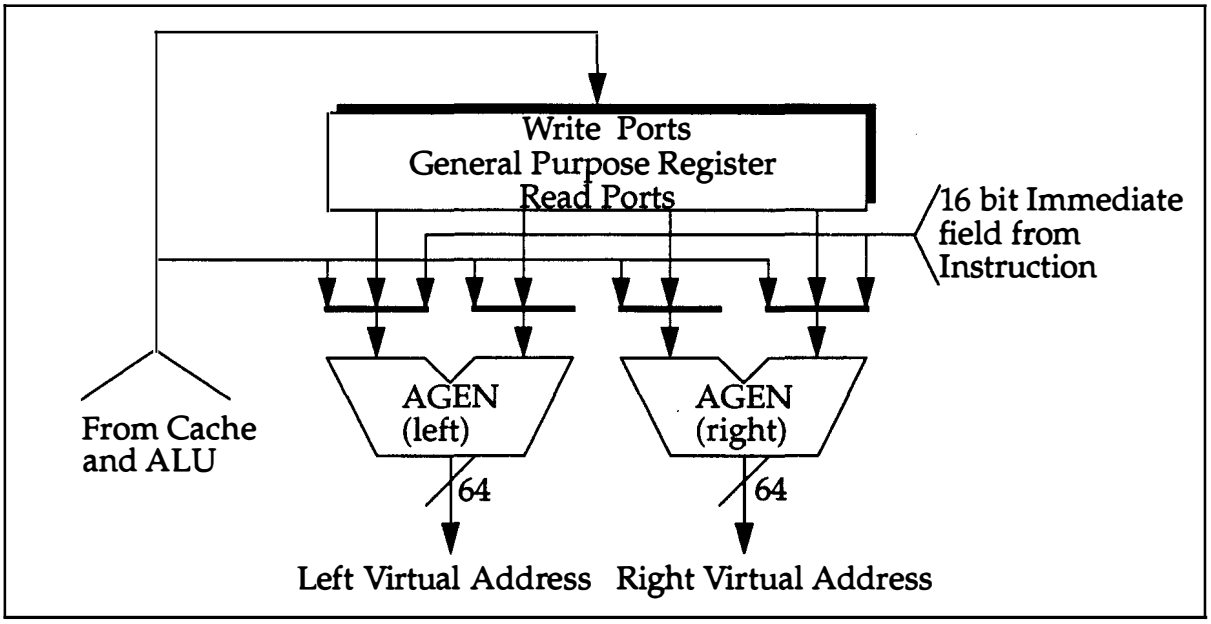


Figure 4-2 Address Generation Block Diagram

4.4 DATA FORMATS

The R8000 microprocessor supports four data formats: a 64-bit doubleword, a 32-bit word, a 16-bit half-word, and an 8-bit byte. Byte ordering within the larger data formats -- half-word, word, and doubleword-- can be configured in either big endian or little endian format.

The default mode of operation for the R8000 microprocessor is big endian. All code accesses are done in big endian and multi-byte data values are stored in memory in big endian format. In big endian format byte 0 is the most-significant byte and byte 7 is the least-significant byte. When operating in kernel mode the endian of the processor is set by the BE bit in the **Config** register. The Reverse Endian (RE) bit in the **Status** register can be used to dynamically switch the endian in user mode. Figure 4-3 and Figure 4-4 show little endian and big endian byte ordering within a 64-bit doubleword.

Figure 4-3 Little Endian Byte Ordering

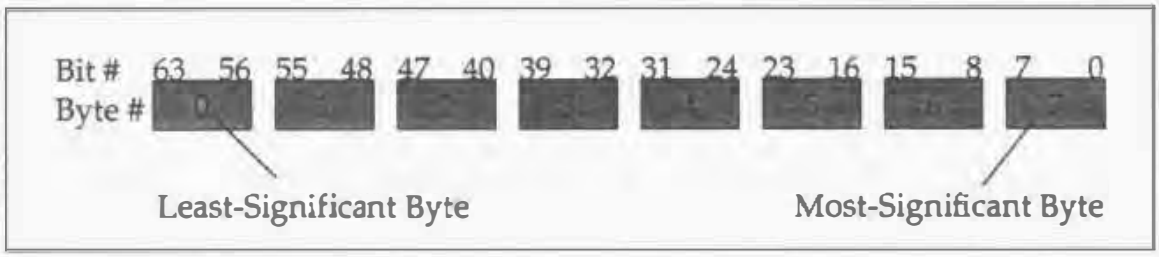


Figure 4-4 Big Endian Byte Ordering

Data alignment requirements are as follows. Any violation results in an address error exception.

* 64 bit values are aligned on 8-byte boundaries when referenced in memory. The three least significant address bits must be zero.
* 32 bit values are aligned on 4-byte boundaries when referenced in memory. The two least significant address bits must be zero.
* 16 bit values are aligned on 2-byte boundaries when referenced in memory. The least significant address bit must be zero.

The following eight special instructions can be used to load and store words or doublewords that are not aligned on 4- or 8-byte boundaries. The instructions are used in four pairs as shown to provide addressing of misaligned words. Addressing misaligned data incurs one additional instruction cycle over that required for addressing aligned data due to the required execution of two instructions.

LWL - Load Word Left	SWL - Store Word Left
LWR - Load Word Right	SWR - Store Word Right
LDL - Load Doubleword Left	SDL - Store Doubleword Left
LDR - Load Doubleword Right	SDR - Store Doubleword Right

The Load Word Left (LWL) Instruction is used in combination with the LWR instruction to load the lower 32 bits of a register with four consecutive bytes from memory when the bytes cross a word boundary. The LWL instruction loads the left portion of the register

with the appropriate part of the high-order word. The LWR instruction loads the right portion of the register with the appropriate part of the low-order word. Similarly, the Load Doubleword Left (LDL) and Load Doubleword Right (LDR) instructions are used to load mis-aligned double-words.

The Store Word Left (SWL) instruction is used in combination with the SWR instruction to store the lower 32 bits of a register to memory when the bytes cross a word boundary. The SWL instruction stores the left portion of the register with the appropriate part of the high-order word. The SWR instruction stores the right portion to the register to the appropriate part of the low-order word. The same rules apply for the Store Doubleword Left (SDL) and Store Doubleword Right (SDR) instructions.

4.5 ADDRESS TRANSLATION

The R8000 Microprocessor uses a Translation Lookaside Buffer (TLB) to perform virtual to physical address translation. The TLB is internal to the R8000 CPU and is used to determine if a given address exists in the physically indexed data cache. Since the instruction cache is virtually indexed address translation is necessary only on a miss. Hence the R8000 Microprocessor has a single TLB to service both the instruction and data caches.

The TLB is dual-ported and can perform two virtual address to physical address translations per cycle. The TLB is 128 entries deep and 3-way set associative for a total of 384 entries. The large number of entries helps to minimize the TLB miss rate but still maintain a single cycle access rate. The TLB supports multiple page sizes on a per process basis. *The TLB may contain multiple page sizes at any given time, but each process, defined by a specific ASID value, can have only one page size associated with it.*

TLB, Data Cache, and Data Cache Tag RAM lookups are performed in the execution stage (E-stage) of the pipeline. The VTAG portion of the TLB is used to determine whether a certain range of addresses resides in the physical address (PA) portion. If it is determined that the translation for the virtual address resides in the TLB, the contents of the PA portion is compared to that in the Data cache tag RAM, resulting in either a hit or a miss to the Data cache. Either a TLB or a Data cache miss initiates an external memory cycle. Figure 4-5 shows the organization of the Translation Lookaside Buffer.

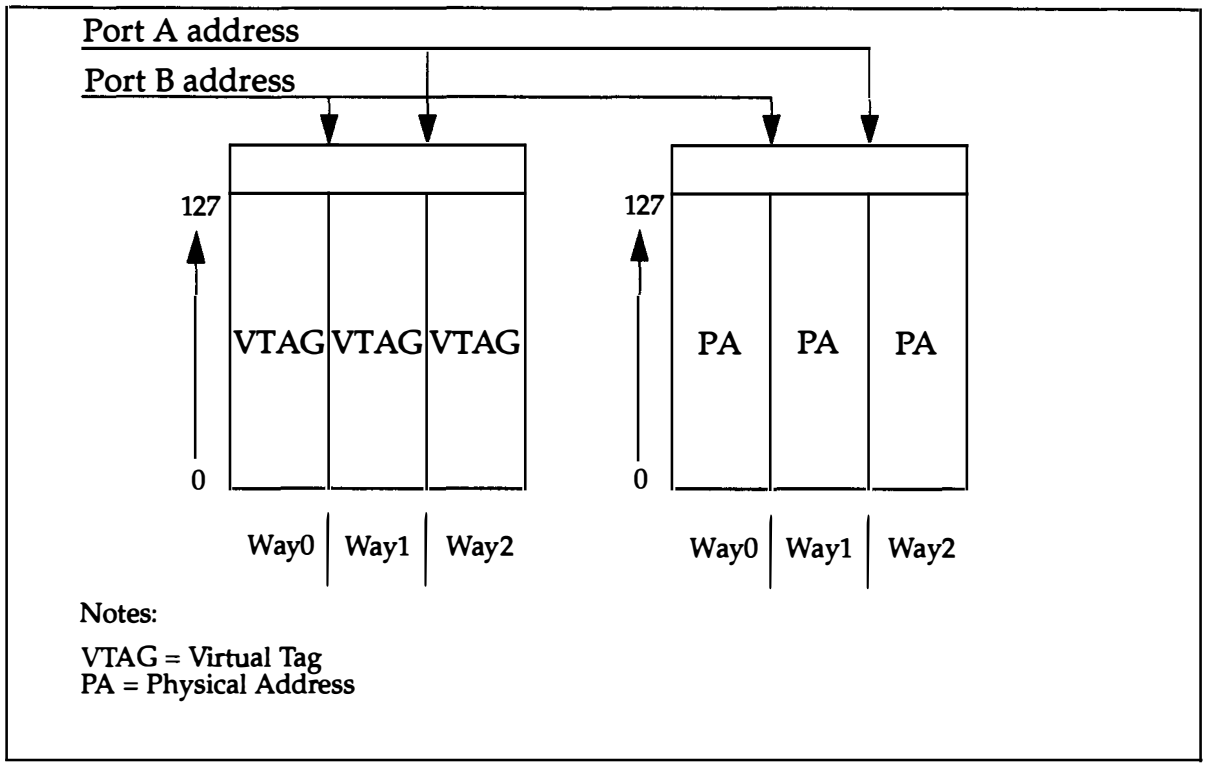


Figure 4-5 TLB Organization

The physical address portion of Figure 4-5 is comprised of 28 bits. The organization of the physical address portion of the TLB is identical to that of the **EntryLo** register. The EntryLo register serves as the data register for the physical addres portion of the TLB RAM.

4.5.1 Indexing the TLB

The TLB is indexed by the lower seven bits of the virtual page number. Which of the seven bits are used varies depending on the page size. The index is hashed by logically XOR'ing the lower 7-bits of the virtual page number with the lower 7-bits of the TLB ASID.

Figure 4-6 shows how the TLB is indexed. The region bits control a multiplexor which passes either the 4-bit Kernel Page Size (KPS) field or the 4-bit User Page Size (UPS) field to select the seven virtual address bits which form the index to the TLB. If the region bits are zeros, the UPS field is passed through the multiplexor. A non-zero value causes the KPS field to be passed. However, there is one exception to this rule. When base register r0 is used with a negative offset to generate an address, the Kernel Page Size (KPS) field is selected. The KPS field is defined by bits 39:36 of the **Status** register. Bits 35:32 define the UPS field.

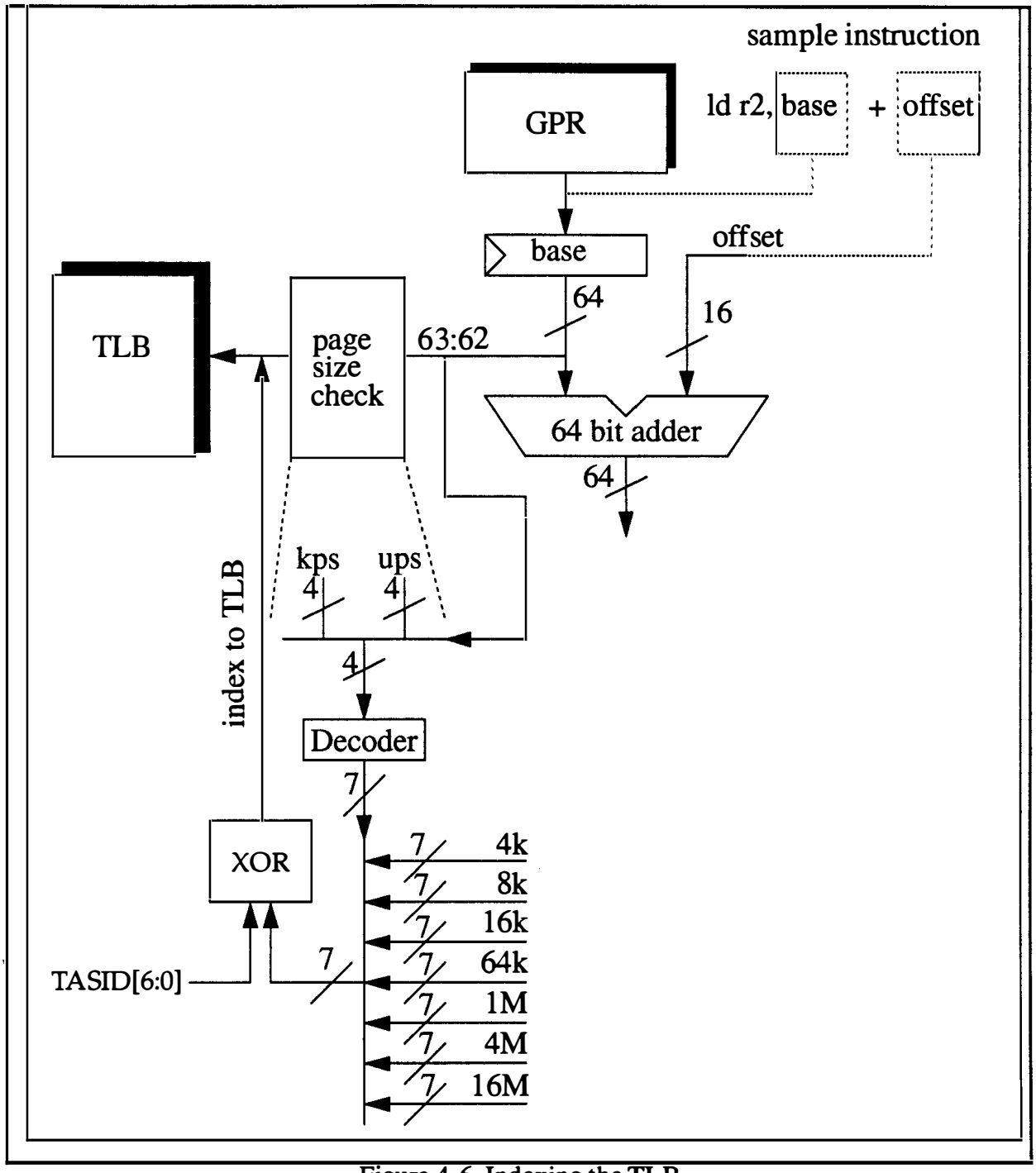


Figure 4-6 Indexing the TLB

As shown in Figure 4-6, the TLB supports seven different page sizes. The 128 entries of the TLB mean that seven virtual address bits are necessary to index the TLB. Which virtual address bits are used depends on the page size. Figure 4-7 through Figure 4-13 show the organization of the virtual address and which bits are used to index the TLB for the various page sizes.

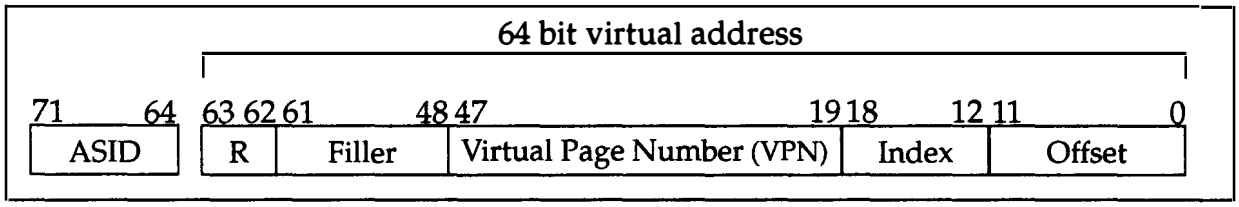


Figure 4-7 4 KByte Page Size

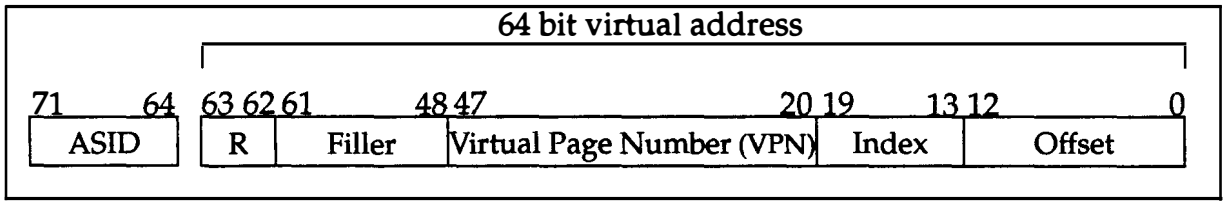


Figure 4-8 8 KByte Page Size

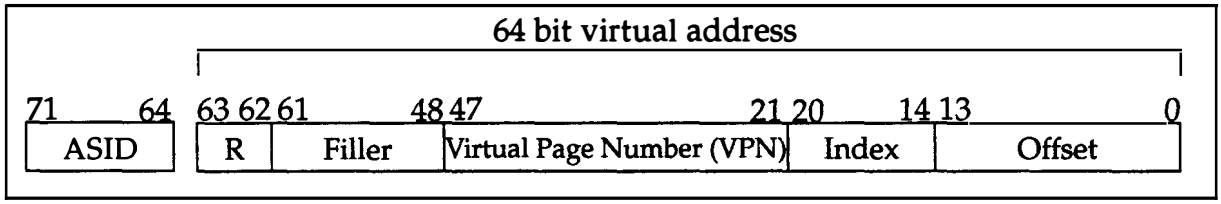


Figure 4-9 16 KByte Page Size

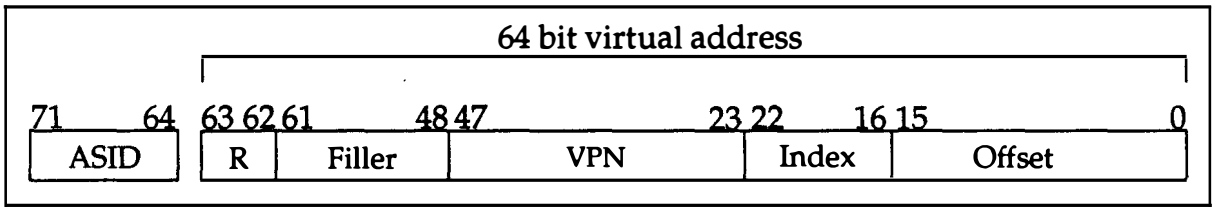


Figure 4-10 64 KByte Page Size

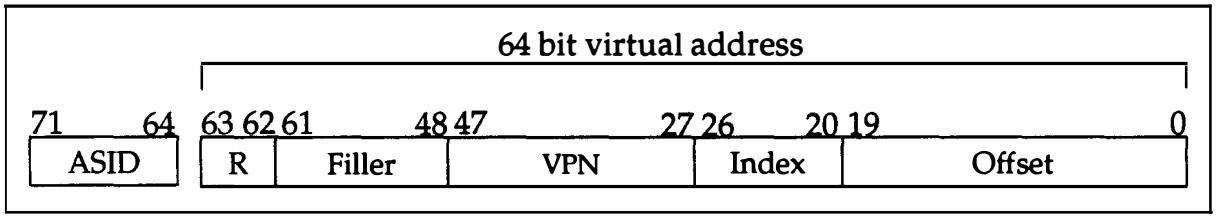


Figure 4-11 1 MByte Page Size

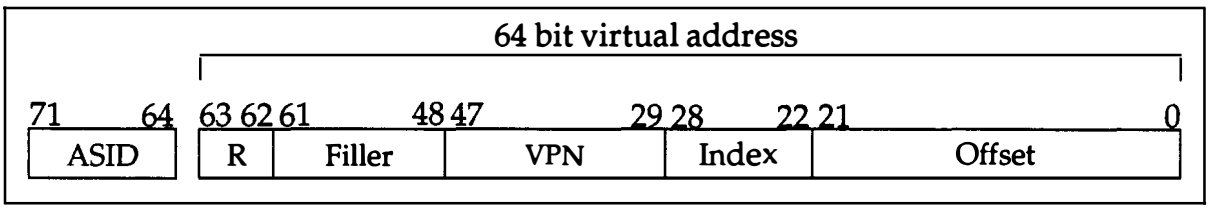


Figure 4-12 4 MByte Page Size

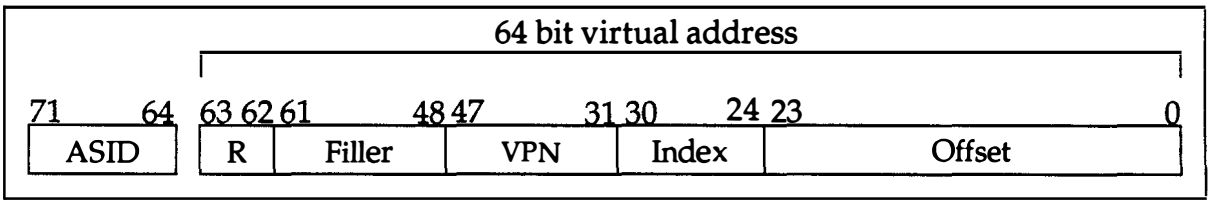


Figure 4-13 16 MByte Page Size

4.5.2 TLB Writes

The TLB Read and TLB Write operations make use of several COP0 registers: **EntryHi, EntryLo, VAddr, TLBSet. EntryHi** and **EntryLo** serve as the data registers for the TLB. Information to be written to the VTAG portion of the TLB is placed in the **EntryHi** register. Information to be written to the PA portion of the TLB is placed in the **EntryLo** register. The **VAddr** register serves as the address register for the TLB providing a 64-bit virtual address. The **TLBSet** register selects which of the 3 sets is to be written or read.

The TLB implements a random replacement algorithm, hence under most cases the 2 bit value in the **TLBSet** register has been randomly generated. However, this value can be overwritten under program control in order to write a specific number. The contents of both the **EntryHi** and the **TLBSet** registers are undefined at reset. The format of the set bits of the **TLBSet** register is shown in Table 4-5.

TLBSET [1:0]	Set
00	Set 0
01	Set 1
10	Set 2
11	Reserved

Table 4-5 TLB Set Replacement Field

4.5.2.1 Wiring Down TLB Entries

The **Wired** register is provided to assist the operating system in preventing certain TLB entries from being replaced during a TLB refill exception. Such entries are said to be "wired down". The operating system (OS) has the ability to "wire" certain TLB locations. Normally TLB locations are wired down either to enhance performance or maintain correctness. To enhance performance the OS can "wire down" pages containing frequently accessed data structures. To maintain correctness the OS can wire down pages on which it cannot take a TLB refill exception.

When a TLB refill exception is detected, the TLBSet register is loaded with a random number which indicates the set to replace. The random number chosen is normally a value between 0 and 2. However, if the index of the congruence class for a given TLB exception is marked as "wired" in the **Wired** register, then a random value of 1 or 2 is loaded into the **TLBSet** register, indicating that set 0 is not available. The **Wired** register can hold four TLB indeces. *Only set 0 of each index can be wired.*

Each index in the **Wired** register has a valid bit associated with it. The valid bit must be set in order for the TLB location to be wired by the OS. If a TLB miss occurrs and the location is wired, but the valid bit in the register is not set, that location is overwritten. The valid bit provides a mechanism to determine whether the contents of the **Wired** register mave meaning. Figure 4-14 shows an example of the **Wired** register containing entries for indexed locations 1, 2, 3, and 127.

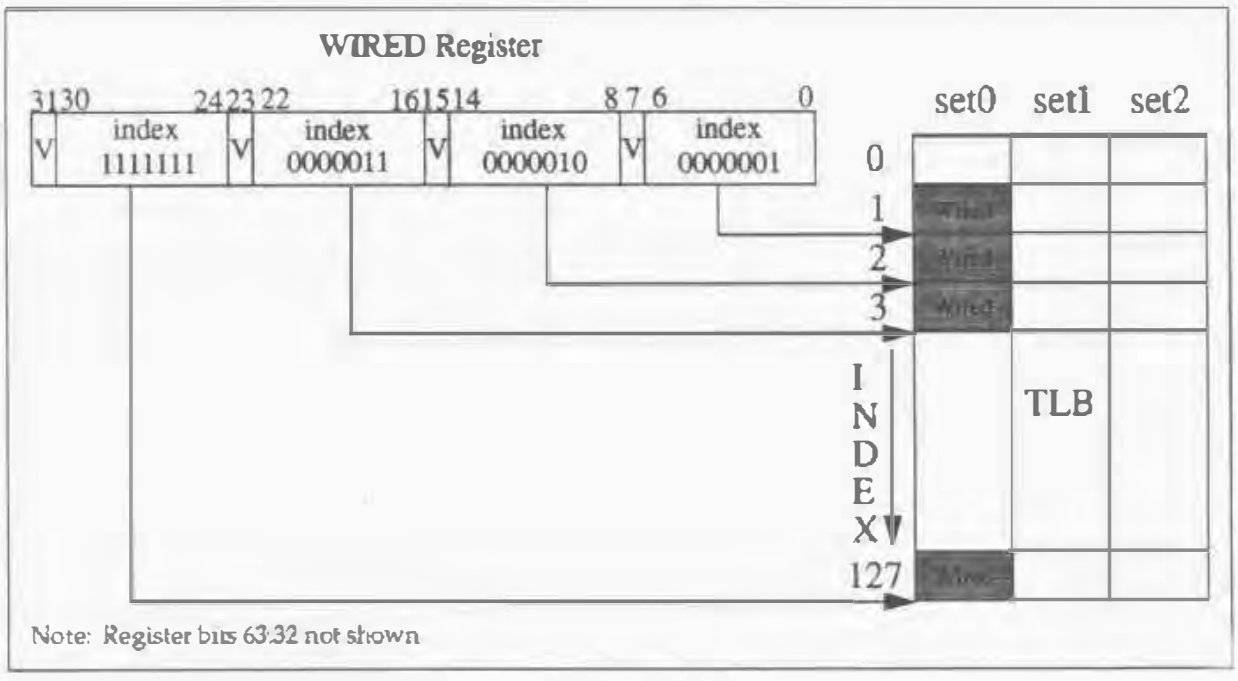


Figure 4-14 Wired TLB Locations

4.6 FORWARD AND REVERSE MAPPING

The R8000 Microprocessor provides support for both forward and reverse mapped memory management schemes. A forward mapping table contains the virtual-to-physical address translation for a given virtual address. Figure 4-15 shows a forward-mapped table entry which is identical to the format of the EntryLo register. The address contains the Physical Frame Number (PFN) and the associated Cache algorithm (C), the write permission (D), and the Valid state bits (V).

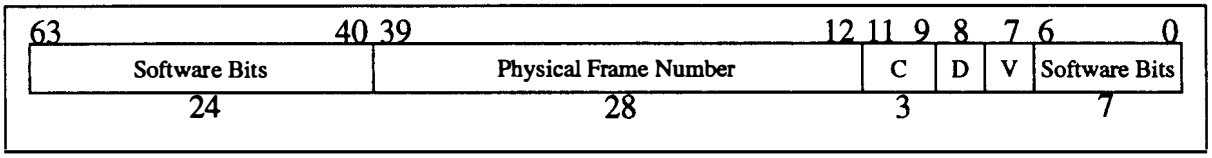


Figure 4-15 Forward Mapped Page Table Entry

For each virtual page in a forward mapped scheme there is an entry in the page table. The more virtual pages the larger the page table. In a reverse mapped memory there is an entry in the page table for each physical page. Multiple virtual pages are aliased to the same entry. The entry contains a virtual tag which is read and compared to the virtual page being accessed. A valid compare indicates that the entry contains the translation for the given virtual page. The aliased locations form a linked list which contain pointers to one another. Hence if the virtual tag mis-compared with the virtual page, the entry would contain a pointer to the next entry in the linked list. The new entry is then accessed and again compared. This process is continued until the correct page table entry is located. A reverse mapped memory scheme can be useful when the page table size is a concern. Since physical address space is much smaller than virtual address space, a reverse-mapping scheme can provide for a much smaller page table.

A reverse mapped entry contains three doublewords. Doubleword 0 is the same format as the forward mapped address shown in Figure 4-15. Doubleword 1 contains the Virtual Page Number (VPN) and ASID of the entry. Doubleword 1 is identical in format to the EntryHi register. Doubleword 2 contains a virtual pointer to the next entry in the linked list. Figure 4-16 shows a page table entry for a reverse-mapped memory scheme.

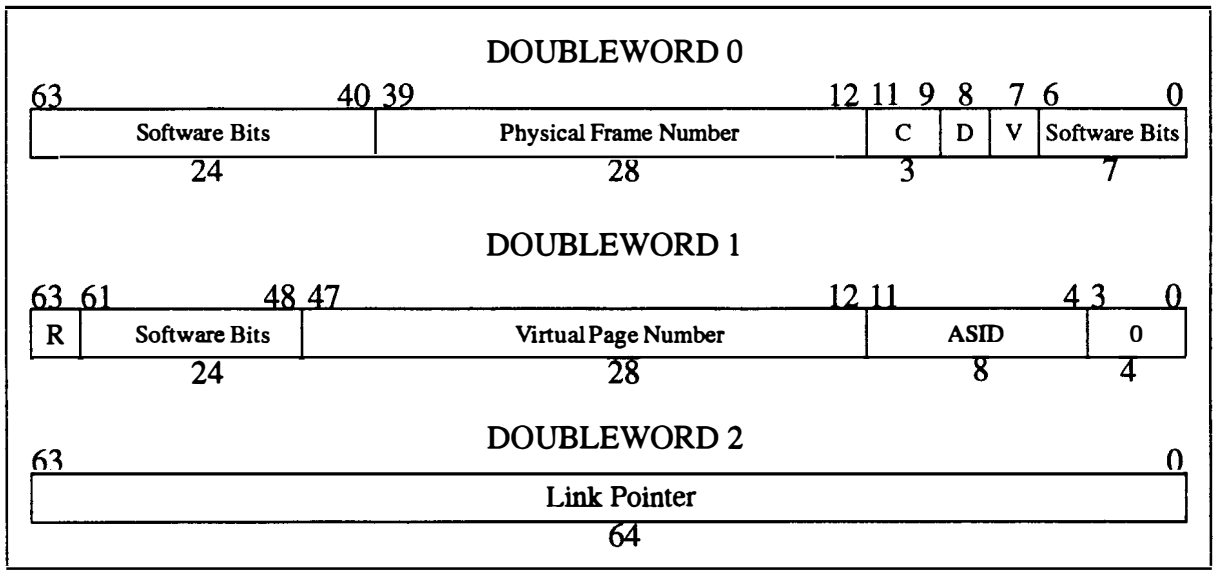


Figure 4-16 Reverse Mapped Page Table Entry

4.7 TLB EXCEPTIONS

The following section discusses the various types of TLB exceptions. Specific exception vectors are provided for each of the four main virtual address spaces. The TLB specific exceptions and their corresponding vector locations are listed inTable 4-6.

Name	Vector	Cause Code	Description
TLB Refill, User, (EXL=0)	TrapBase + 0x_000	TLBL TLBS	The referenced address was to the User Region (UV) and did not match any TLB entry.
TLB Refill, Kernel Private (EXL=0)	TrapBase + 0x_400	TLBL TLBS	The referenced address was to the Kernel Private Region (KV1) and did not match any TLB entry.
TLB Refill, Kernel Global (EXL=0)	TrapBase + 0x_800	TLBL TLBS	The referenced address was to the Kernel Global Region (KV0) and did not match any TLB entry.
TLB Refill (EXL=1)	TrapBase + 0x_c00	TLBL TLBS	The referenced address (to any Region) did not match any TLB entry.
TLB Invalid	TrapBase + 0x_c00	TLBL TLBS	Virtual-address that matches an invalid TLB entry.
TLB Modified	TrapBase + 0x_c00	Mod	An attempt to write to a virtual address that did not have D bit in the corresponding TLB entry set.

Table 4-6 TLB Exception Vectors

Each of the vectors shown in figure 4-6 can occur on either a load or a store with the exception of the TLB Modified exception, which occurs only on a store. The TLBL and TLBS exception codes, indicated by bits [7:3] of the Cause register, indicate whether the instruction, defined by the contents of the Exception Program Counter (EPC) register, and the BD bit [63] of the Cause register, was a load or a store. Table 4-7 shows the cause register codes for TLB specific exceptions. A complete list of exception cause codes can be found in chapter 5, table 5-2.

ExcCode Number	Mnemonic	Description
1	Mod	TLB Exception Modification
2	TLBL	TLB Exception (Load or Instruction Fetch)
3	TLBS	TLB Exception (Store)
30	TLBX	TLB Exception (Multiple Hits due to a Duplicate Entry

Table 4-7 TLB Exception Codes

4.7.1 TLB Refill

A TLB Refill exception occurs when no TLB entry matches a reference to a mapped address space. *TLB refill exceptions are precise and are not maskable.* Four exception vector locations are provided for TLB refill exceptions. Which vector is used depends on the state of the region bits (VA[63:62]) of the faulting address and the execution level (EXL) of the faulting process, indicated by bit [1] in the Cause register. A complete list of exception vectors can be found in chapter 5, table 5-1.

When a TLB refill exception occurs, the **VAddr** and **EntryHi** registers contain the virtual address that failed address translation. The **EntryHi** register also contains the Address Space Identifier (ASID) from which the translation failed. A random set number is generated and then qualified by the contents of the **Wired** register to assure that the set chosen does not correspond to any of those in the **Wired** register. The random value is then placed in the **TLBSet** register. The contents of **EntryLo** is undefined.

The 64-bit **Exception Program Counter** (EPC) register points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction resides in a branch delay slot, the EPC register points at the branch instruction which preceedes it, and the BD bit of the **Cause** register is set.

Three Base registers are provided by hardware, one per mapped region: **UBase, PBase,** and **GBase.**

The **UBase** register specifies the base address of the page table for a per-process user

virtual address space miss. The User space (UV) is defined by the region bits VA[63:62] of the virtual address (R = 00) .

The **PBase** register specifies the table address for a per-process kernel private address space miss. Kernel Private space (KV0) is defined by the region bits VA[63:62] of the virtual address (R = 01).

The **GBase** register specifies the table address for a per-process kernel global address space miss. Kernel global space (KV1) is defined by the Region bits VA[63:62] of the virtual address (R = 11). No register is defined for the Kernel Physical address space as address translation is not performed when operating in this space.

In a forward mapping scheme, multiple Base registers allows different regions to have different page sizes. In a reverse mapping scheme, multiple base registers allow different regions to have different page and reverse table sizes or share tables with a common page size. Base addresses are specified by software.

On a TLB-miss, hardware loads the **VAddr** register with the virtual address of the missing reference. *The actions taken by hardware during a TLB-miss is mapping scheme independent.* The **ShiftAmt** register is loaded with a shift amount based on page size. For instance, a 4k-byte page size has a shift-amount of 12, a 16k-byte page size a shift-amount of 14, etc. Hardware provides a unique exception entry point for User space (UV), Kernel Private space (KV0) and Kernel Global space (KV1) TLB Refill exceptions. TLB Refill exception action depends on the Memory Management Unit (MMU) scheme. Which scheme is chosen is under software control and is transparent to hardware.

4.7.1.1 TLB Refill: Forward Mapping Table

The forward mapping table TLB-miss handler is required to justify **VAddr** register relative to page and page table entry size, reload the **EntryLo** register and load a new TLB entry via the TLBW instruction . Miss handler code is independent of page size.

Example code for a forward-mapped translation tables is discussed below. The example assumes that the highest page of virtual memory, which is reachable by a negative offset from register r0, is wired down and some small number of words are available to the handlers for holding various constants.

Multiple page sizes are supported by shifting the **VAddr** register to the right. Since software knows the page sizes of each region, each PTEBase can be pre-shifted to the left by the appropriate number of bits to compensate. It is most convenient to always set the high-order bit of PTEBase and use an arithmetic right shift to add a series of logical one's. The unwanted values can be logically AND'ed away.

The following code contains explicit NOP's to show where interlocks would occur.

```
#
# Fetch VAddr and adjust according to page size and PTE entry size.
#
dmfc0 k1, VAddr# get VA
dmfc0 k0, ShiftAmt# get page size shift amount value
srav k1, k1, k0# page size adjustment
nop
sll k1, k1, $entrysize# PTE entry size adjustment

#
# Fetch appropriate PTEBase for address space where fault occurred
# This can issue with sll above.

#ifdef UTLB_user
dmfc0 k0, UBase# region 00 handler
#else ifdef UTLB_kernal_private
dmfc0 k0, PBase# region 01 handler
#else ifdef UTLB_kernel_global
dmfc0 k0, GBase# region 11 handler
#endif
or      k1,k1, k0# combine PTEBase with vpn offset
nop

#
# Finish refilling the TLB
#
ld      k0, 0(k1)# fetch translation
nop
dmtc0 k0, EntryLo
tlbw# write new entry in random set
eret
```

A cycle could be saved in the handler by keeping the contents of the **ShiftAmt** register as a constant in the wired-down page of virtual memory. In this way, **ShiftAmt** would be obtained with a **ld** instead of a **dmfc0**, and the **ld** is capable of issuing with the **dmfc0 VAddr**.

4.7.1.2 TLB Refill: Reverse Mapping Table

The reverse mapping table TLB-miss handler is responsible for generating the reverse table hash index and traversing the list of entries per table index in search of a translation. Once a translation is found, the **EntryLo** register is loaded and the TLB refilled via the TLBW instruction. *Miss handler code is independent of page size and reverse-mapping page table size.*

The idea behind a fast reverse-mapped TLB Refill handler is that the hash table should typically have very few "collisions". Therefore it is possible to search the first few nodes quickly using unrolled and hand-optimized code, reverting to a more general handler for the uncommon case (or if no translation is found).

Four constants are required in the following example code. The constant HASHVAL typically contains some function of the current ASID to "randomize" probes into the hash table. Only bits [47:12] should be non-zero. The constant TBLMASK is used to define the hash table size. Distinct TBLMASK's are needed to support differing User and Kernel page sizes.

A temporary variable, LINKPTR, located in the same "negatively-addressed" area is used to save away a node pointer. This optimization is optional and it is possible to retrace the linked list from the **VAddr** register.

Another detail is that the virtual address field in the page table entry must be identical to the format of the **EntryHi** register *including the zero fields.*

Example code for a reverse-mapped translation table is discussed below. The example assumes that the highest page of virtual memory, which is reachable by a negative offset from register r0, is wired down and some small number of words are available to the handlers for holding various constants.

Multiple page sizes are supported by shifting the **VAddr** register to the right. Since software knows the page sizes of each region, each PTEBase can be pre-shifted to the left by the appropriate number of bits to compensate. It is most convenient to always set the high-order bit of PTEBase and use an arithmetic right shift to add a series of logical one's. The unwanted values can be logically AND'ed away.

The following code contains explicit NOP's to show where interlocks would occur. The code should run very close to one instruction per cycle. The following code does not check for end-of-list. There is an implicit assumption that lists are terminated into a "terminal" node which does not ever match and whose link points to itself.

```
Look_in_first_node:
dmfc0k1, VAddr# fetch vpn
ldk0, HASHVAL(r0) # fetch hash value;
# ld can issue with previous dmfc0.
xork1, k1, k0# hash VPN[47:12]
#
# Adjust vpn relative to page size and PTE entry size

dmfc0k0, ShiftAmt# fetch page size shift amount value;
# dmfc0 can issue with xor.
sravk1, k1, k0# page size adjustment
nop
sllk1, k1, $entrysize# PTE entry size adjustmen

# Fetch appropriate PTEBase for address space where fault occurred.
# This can issue with sll above.

#ifdef UTLB_user
dmfc0 k0, UBase# region 00 handler
#else ifdef UTLB_kernal_private
dmfc0 k0, PBase# region 01 handler
#else ifdef UTLB_kernel_global
dmfc0 k0, GBase# region 11 handler
#endif
ork1, k1, k0# combine PTEBase with vpn offset

#
# Mask according to table size (and to zero high-order bits).
#
ldk0, TBLMASK(r0)# fetch reverse table size mask
# can issue with previous OR.
andk0, k1, k0# table size hash index
# adjustment
nop
sdk0, LINKPTR(r0)# save away entry index
nop
#
# Compare EntryHi against virtual address in table entry
#
ldk0, 8(k0)# fetch entry containing VPN/ASID
dmfc0 k1, EntryHi# if no match look at next
# node can issue with
# previous ld.
#
#ifdef UTLB_kernel_global
```

```
srlk1, 12# global reference must clear out
nop
sllk1, 12# ASID field in EntryHi
nop
#endif

bneqk0, k1, Look_In_Second_Node
ld k0, LINKPTR(r0)
#
# Translation found. Write new TLB entry.
#
ldk1, 0(k0)# fetch translation; we won't fault
nop
dmtc0k1, EntryLo
tlbw# tlb allocation
eret


Look_In_Second_Node:
ldk0,16(k0)# fetch next node pointer
nop
sdk0, LINKPTR(r0)# save away entry index
#
# Compare EntryHi against virtual address in table entry
#
ldk0, 8(k0)# fetch entry containing VPN/ASID
# k1 still has EntryHi
bneqk0, k1, Look_In_Third_Node
ld k0,LINKPTR(r0)

# Translation found. Write new TLB entry.
#
ldk1, 0(k0)# fetch translation; we won:t fault
dmtc0k1, EntryLo
nop
tlbw# tlb allocation
eret
```

4.7.2 TLB Invalid

The TLB Invalid entry in Table 4-6 occurs when a virtual address reference matches a TLB entry marked invalid. *This exception is precise and not maskable.*

The TLB Invalid exception uses the common exception vector located at offset 0x_c00 in Table 4-6. The TLBL and TLBS exception codes, indicated by bits [7:3] of the **Cause** register, indicate whether the instruction, defined by the contents of the **EPC register,** and the BD bit [63] of the **Cause** register, was a load or a store.

When a TLB Invalid exception occurs, the **VAddr** and **EntryHi** registers contain the virtual address that failed address translation. The **EntryHi** register also contains the Address Space Identifier (ASID) from which the translation failed. The set which matches the virtual address reference is placed in the **TLBSet** register.

The 64-bit **EPC** register points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction resides in a branch delay slot, the **EPC** register points at the branch instruction which preceedes it, and the BD bit of the **Cause** register is set.

The valid bit of a TLB entry is typically cleared when a virtual address does not exist, when it exists but is not in memory (a page fault), or when a trap is desired on any reference to the page (for example, to maintain a reference bit). After servicing the particular cause of this exception, the TLB entry can, if appropriate (i.e. no subsequent exception was possible), be validated by reading the invalid TLB entry into the **EntryLo** register with a TLBR operation, moving **EntryLo** to a general register, setting the V bit, moving it back to **EntryLo,** and doing a TLBW operation.

4.7.3 TLB Modified

A TLB Modified exception occurs when the virtual address of a store instruction matches a TLB entry marked valid but not dirty/writable. *The TLB Modified exception is precise and not maskable.*

The TLB Modified exception uses the common exception vector located at offset 0x_c00 in Table 4-6. The Mod exception code, indicated by bits [7:3] of the **Cause** register, is set.

When a TLB Modified exception occors, the **VAddr** and **EntryHi** registers contain the virtual address that failed address translation. The **EntryHi** register also contains the Address Space Identifier (ASID) from which the translation failed. The set which matches the virtual address reference is placed in the **TLBSet** register.

The 64-bit **EPC** register points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction resides in a branch delay slot, the **EPC** register points at the branch instruction which preceedes it, and the BD bit [63] of the **Cause** register is set.

The kernel uses the failing virtual address to identify the corresponding access control information. The page identified may or may not permit write accesses, and if not permitted, a "Write Protection Violation" occurs. Otherwise, if write accesses are permitted, the page frame is marked dirty/writable by the kernel in its own data structures and the TLB entry is updated

4.8 DATA AND CONTROL REGISTERS

The R8000 microprocessor supports two groups of registers defined as CoProcessor 0 (COP0) and CoProcessor 1 (COP1). CoProcessor 0 contains control, status, data, and configuration registers for the Integer Unit. CoProcessor 1 contains status and congifuration registers for the R8010 floating point unit.

There are thirty-two 64 bit system control registers which are accessible via the double Move To/From CoProcessor 0 instructions (DMTC0, DMFC0). *Thirty-two bit versions of these instructions are not supported.*

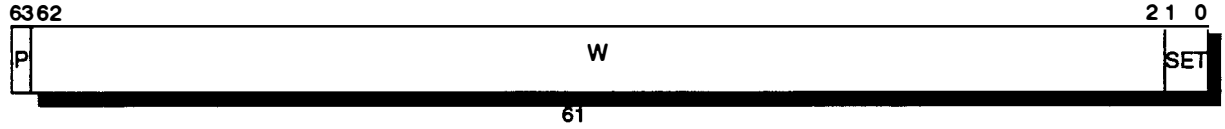
There are two 32 bit floating point control registers which are accessible via the double Move To/From CoProcessor 1 instructions (DMTC1, DMFC1). *Thirty-two bit versions of these instructions are not supported.*

This section focuses on those registers used to support and manage TLB functions. Not all of the 32 system control registers are defined here. Refer to the Registers chapter for a complete listing of all registers.

The following pages list the TLB specific registers and their functions.

4.8.1 TLBSet

COP0 Register #	Register Mnemonic	Description
0	TLBSet	Select TLB set



P	if set, the last TLBP operation was unsuccessful.
SET	specifies the set select address within a TLB entry.
W	Fields that may be written with anything but are always read as 0

Description

The TLBSet Register is a read-write register used to index a TLB entry's set and to provide access status as the result of a TLBP operation.

The SET field is used to select a TLB entry's set for a TLBW or a TLBR instruction. When a TLB Refill (User, Kernel Private, and Kernel Global) exception occurs, TLBSet is loaded with a random set to be replaced. When a TLB Invalid or TLB Modified exception occurs, TLBSet is loaded with the set which contains the virtual tag match. The Set field may be overwritten under program control to write to a specific set number.

The TLBSet register also contains status regarding the TLB Probe (TLBP) instruction execution. The P bit is set if the last TLBP instruction did not find a TLB entry which matched EntryHi. If the last TLBP was successful, P=0 and SET holds the set number which matched.

Format for SET field:

00	Set 0
01	Set 1
10	Set 2
11	Reserved

The TLBSet register is undefined on reset.

4.8.2 EntryLo

COP0 Register #	Register Mnemonic	Description
2	EntryLo	Physical Address portion of TLB entry

PFN	Physical Frame Number
C	Specifies the page cache coherence algorithm
D	When set the page is dirty and writable
V	When set the entry is valid
W	Fields that may be written with anything but are always read as 0

Description:

The EntryLo register is a read-write register used to access the physical portion of the TLB. EntryLo contains the Physical Page Number (PFN) and its associated Cache Algorithm (C), Write Permission (D), and Valid (V) state bits.

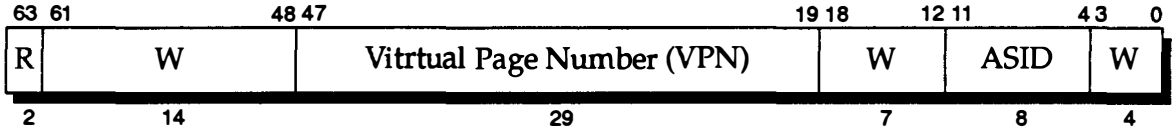
The C field encoding is as follows:

000	uncacheable processor-ordered
001	reserved
010	uncacheable sequential-ordered
011	cacheable non-coherent
100	cacheable coherent, exclusive
101	cacheable coherent, exclusive on write
110	reserved
111	reserved (cacheable, write-through)

The EntryLo register is undefined on reset.

4.8.3 EntryHi

COP0 Register #	Register Mnemonic	Description
10	EntryHi	Virtual Tag portion of TLB entry



R	Two bit Region field (00=user, 01 =KV0, 11=KV1)
VPN	Virtual Page Number field
ASID	Address Space Identifier.
W	Fields that may be written with anything but always read as 0

Description:

The EntryHi register is a read-write register used to access the virtual tag portion of the TLB. In addition, EntryHi contains the Address Space Identifier (ASID) used to match the virtual address with a TLB entry when virtual addresses are presented for translation.

When a TLB-related exception occurs, EntryHi is loaded with the Virtual Page Number (VPN) and the Region (R) of the virtual address that failed translation. The VPN field contains bits [47:19] of the faulting virtual address. It is *not* right justified according to page size. VPN[23:19] is conditionally set to zero by hardware on a per-bit basis based on page size.

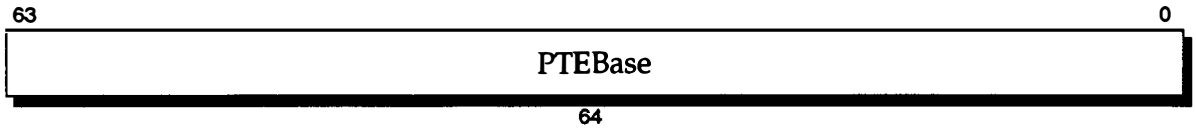
The ASID field already contains the Address Space Identifier for the virtual address which caused the exception, and so is not loaded when an exception occurs.

The VPN field does not contain bits [18:12] of the virtual address. These are not stored in the TLB.

The EntryHi register is undefined on reset.

4.8.4 UBase

COP0 Register #	Register Mnemonic	Description
4	UBase	User Page Table Entry Base Address



PTEBase Base address of Page Table Entries

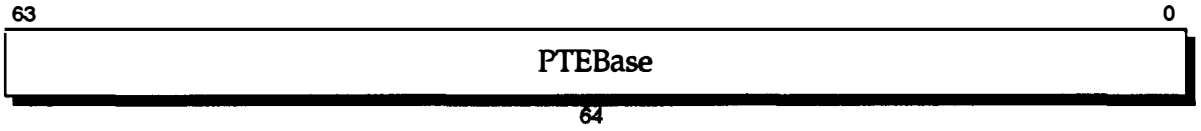
Description

The UBase register is a read-write register which holds the base address of the PTE table for the associated User region. The UBase, PBase, and GBase registers have identical formats.

The UBase register is undefined on reset.

4.8.5 PBase

COP0 Register #	Register Mnemonic	Description
20	PBase	KV0 space Page Table Entry Base Address



PTEBase Base address of Page Table Entries

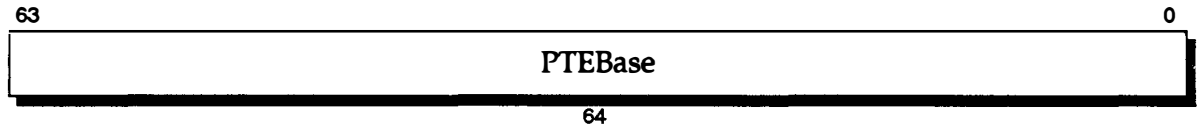
Description

The PBase register is a read-write register which holds the base address of the PTE table for the associated Kernel Virtual 0 region, also referred to as Kernel Private. The UBase, PBase, and GBase registers have identical formats.

The PBase register is undefined on reset.

4.8.6 GBase

COP0 Register #	Register Mnemonic	Description
21	GBase	KV1 space Page Table Entry Base Address



PTEBase Base address of Page Table Entries

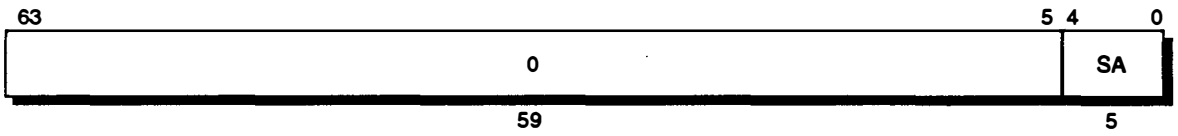
Description

The GBase register is a read-write register which holds the base address of the PTE table for the associated Kernel Virtual 1 region, also referred to as Kernel Global. The UBase, PBase, and GBase registers have identical formats.

The GBase register is undefined on reset.

4.8.7 ShiftAmt

COP0 Register #	Register Mnemonic	Description
5	ShiftAmt	Shift amount to align virtual page number



SA Shift Amount

Description:

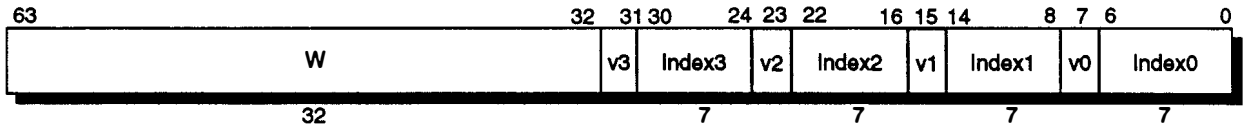
The ShiftAmt register is a read-only register that assists software in aligning pointers into page tables. In the User Region, right-shifting the VA register by the amount in the SA field correctly aligns the Virtual Page Number (VPN) field based on page size for the most recently failed translation. The SA value for each page size is as follows:

Page Size	ShiftAmt Value
4K	01100
8K	01101
16K	01110
64K	10000
1M	10100
4M	10110
16M	11000

Table 4-8 Page Size ShiftAmt Values

4.8.8 Wired

COP0 Register #	Register Mnemonic	Description
24	Wired	Indices of wired locations in the TLB



$Index_x$	TLB entry to be 'wired' down
v_x	Valid Bit set if corresponding Index is valid
w	Fields that may be written with anything but always read as 0

Description:

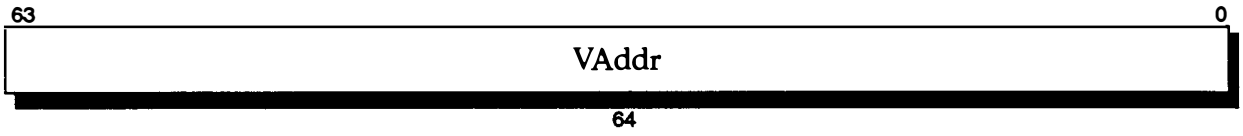
The Wired register is a read-write register used to control TLB replacement algorithm. Up to four entries may be wired down under program control. The four entries must be in different congruence classes.

The TLB is three-way set associative. Only set 0 may be 'wired'. When a TLB Refill exception occurs, the congruence class of the missing virtual address is compared to each of the four indices in the Wired register. If a match is found for a valid entry in the Wired register, a random value in the range 1..2 is loaded into the TLBSet register. If a valid match is not found, a random value in the range 0..2 is loaded into the TLBSet register.

The Wired register is undefined on reset.

4.8.9 VAddr

COP0 Register #	Register Mnemonic	Description
8	VAddr	Virtual Address Register



VAddr Virtual Address

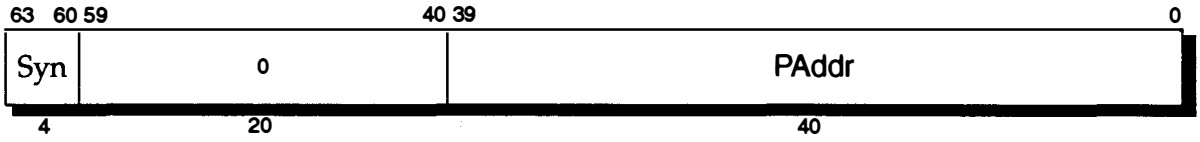
Description:

The VAddr register is a read-write register that holds a 64-bit virtual address. VAddr is loaded both under software and hardware control.

VAddr is loaded by hardware with the virtual address which causes a TLB Refill, TLB Invalid, TLB Modified, or Address Error Exception. VAddr is also writable by software, and is used to address the TLB for TLBW, TLBR, TLBP, DCTR and DCTW Cop0 instructions.

4.8.10 BadPAddr

COP0 Register #	Register Mnemonic	Description
7	BadPAddr	Bad Physical Address



BadPaddr	Bad Physical Address
Syn	Bits [15:12] of the virtual address

Description:

The BadPAddr register is a read-only register that contains the physical address which caused the virtual coherence error (floating) exception.

INTERRUPTS AND EXCEPTIONS

5

The exception processing system of the R8000 Microprocessor is responsible for efficiently handling relatively infrequent events such as address translation misses, arithmetic overflows, I/O interrupts, and system calls. These events cause the interruption of normal flow of control. Dedicated locations contain vectors which service the various exceptions. Once the exception has been serviced the program contents, which were saved in temporary strorage prior to servicing of the exception, are re-loaded and normal execution resumes.

The R8000 Microprocessor treats all events which interrupt the normal flow of execution as exceptions. Interrupts are a type of exception, and exceptions can be both precise and imprecise.

5.1 EXCEPTIONS

When an exception is taken and the contents of the **Exception Program Counter** (EPC) register contains the exact address of the instruction which caused the exception, the exception is precise. When the Exception Program Counter register contains a value which is near the offending address but not exact, the exception is imprecise. External interrupts, which have no relationship to any instructions, are also classified as imprecise exceptions. The type of exception taken indicates whether it is precise or imprecise.

The R8000 Microprocessor can generate interrupts internally as well as accept interrupts from external sources. There are no dedicated interrupt pins on the R8000 Microprocessor. External interrupts are handled by the Cache Controller and the status of the Cache Controller's interrupt register is transferred to the R8000 via the TBus. The register contents are decoded within the R8000 and the appropriate service routine is executed. Figure 5-1 shows the different types of exceptions.

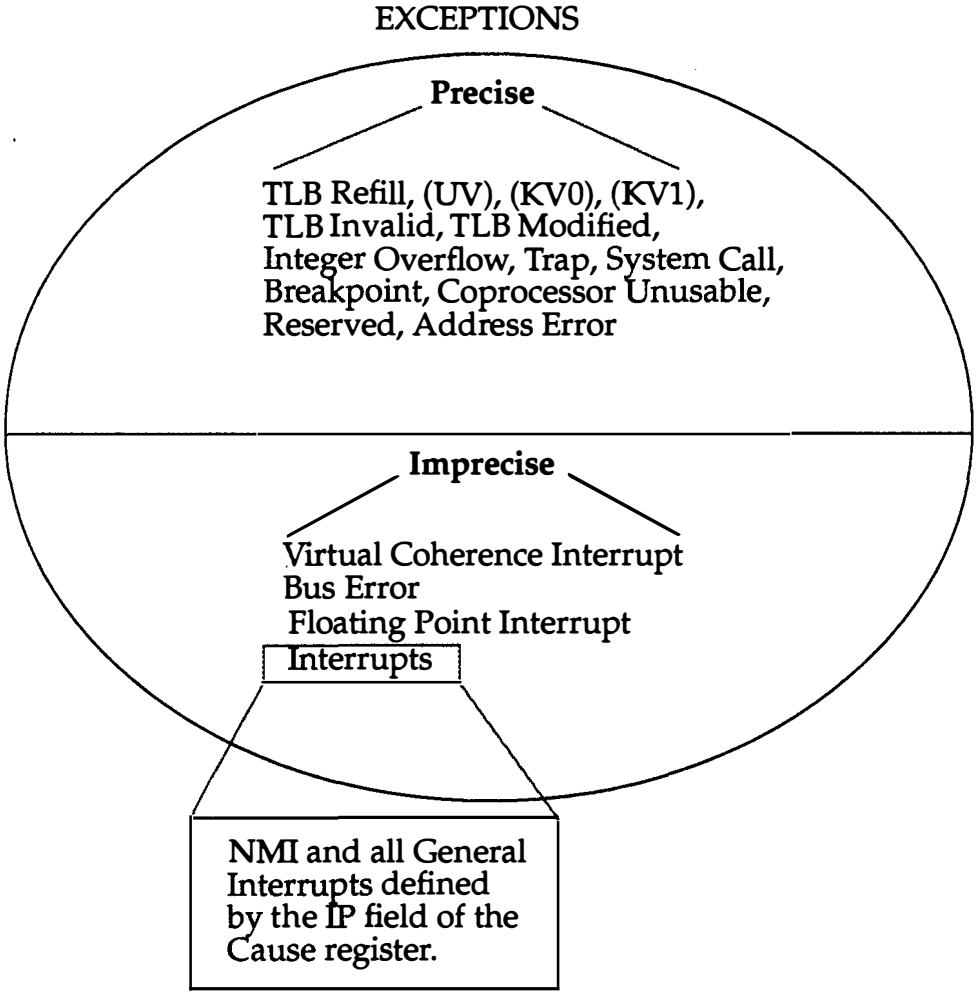


Figure 5-1 Exception Types

To handle an exception, the processor vectors to a fixed address in kernel mode with interrupts disabled. Once the exception has been serviced, the program counter, operating mode, and interrupt enable must be restored. Hence it is these values which must be saved when an exception occurs.

When an exception occurs, the **Exception Program Counter** (EPC) register is loaded with the appropriate restart location at which execution may resume after servicing the exception. The counter can also be thought of as containing the address of the instruction that caused the exception. If the instruction was executing in a delay slot the counter contains the address of the previous instruction and the DB bit is set.

The base operating mode is defined by the **KU** (Kernel/User) and the **IE** (Interrupt Enable) bits of the **Status** register. The execution level is set by the **EXL** bit, also from the **Status** register. Interrupts are enabled when IE=1 and EXL=0. The operating mode is specified by the base mode when the execution level is normal, and is in kernel mode when the execution level is exception. Returning from an exception consists of resetting the execution level (bit [1] of the Status register) to normal. From a register standpoint there are three basic types of exceptions;

1) Hard reset
2) Non-Maskable Interrupt (NMI)
3) All others.

5.1.1 Hard Reset

When returning from hard reset exception the state of the **Config** (configuration) and **Status** registers are as shown in Figure 5-2.

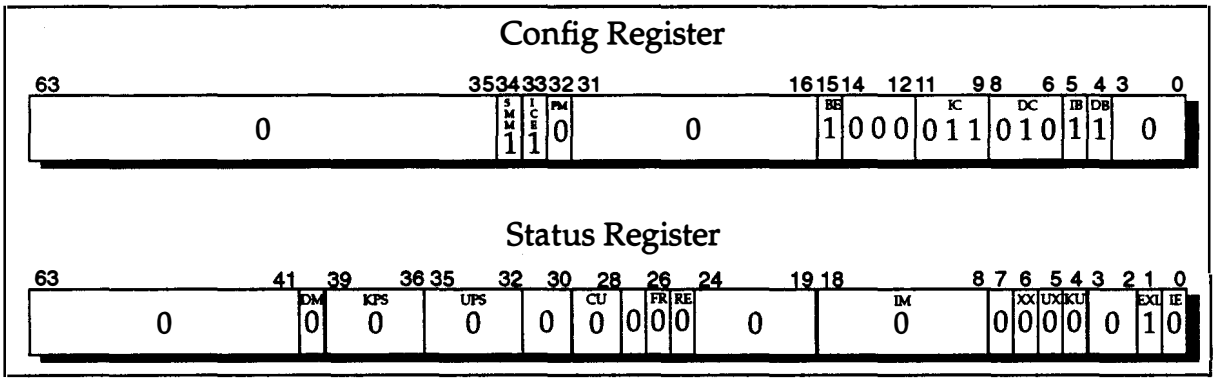


Figure 5-2 Register Contents Following a Hard Reset

The program counter is loaded with the following 64-bit hexidecimal value.

0x_9000_0000_1fc0_0000

5.1.2 Non-Maskable Interrupt

When a non-maskable interrupt exception is taken the contents of the main program counter are loaded into the **Exception Program Counter** (EPC) register and indicate the address at which the exception occurred. Note that the existing EPC counter value is lost when the EPC is loaded. The main program counter can then be loaded with the vector address so that servicing can begin. The main program counter is loaded with the following 64-bit hexidecimal address.

0x_9000_0000_1fc0_0000

The contents of the **Config** register does not change. Bit [1] of the **Status** Register is set to a high value (1), indicating that the execution level is for an exception. In addition, **Cause** register bit [27] is also set to a high value, indicating that the exception was a NMI. The hardware modification to the **Status** register is shown in Figure 5-3.

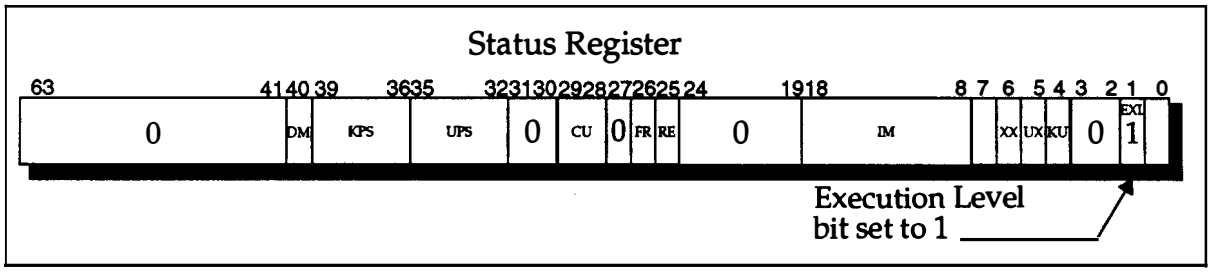


Figure 5-3 Status Register Contents Following an MNI

5.1.3 General Exceptions

Hard Reset and NMI exceptions each vector to a fixed entry point as discussed in sections 5.1.1 and 5.1.2. All other exceptions vector to an offset relative to the contents of the **TrapBase** register . When a general exception is taken the **Status** register is read and the **main program counter** (PC) is loaded with the contents of the **TrapBase** register. Loading of the main program counter occurs regardless of the exception level. Note that the current value is lost when the PC is loaded.

The EXL bit [1] of the **Status** register determines the execution level. If EXL=0 when the status register is read, the contents of the PC are transferred to the **exception program counter** (EPC) so that the address which caused the exception can be saved.

If EXL=1 when the status register is read, indicating that another exception is currently begin serviced, the **EPC** already contains the address of the exception currently being serviced. The contents of the **PC** are not transferred to the **EPC** so as not to overwrite the current EPC value. Figure 5-4 shows how general exceptions are handled.

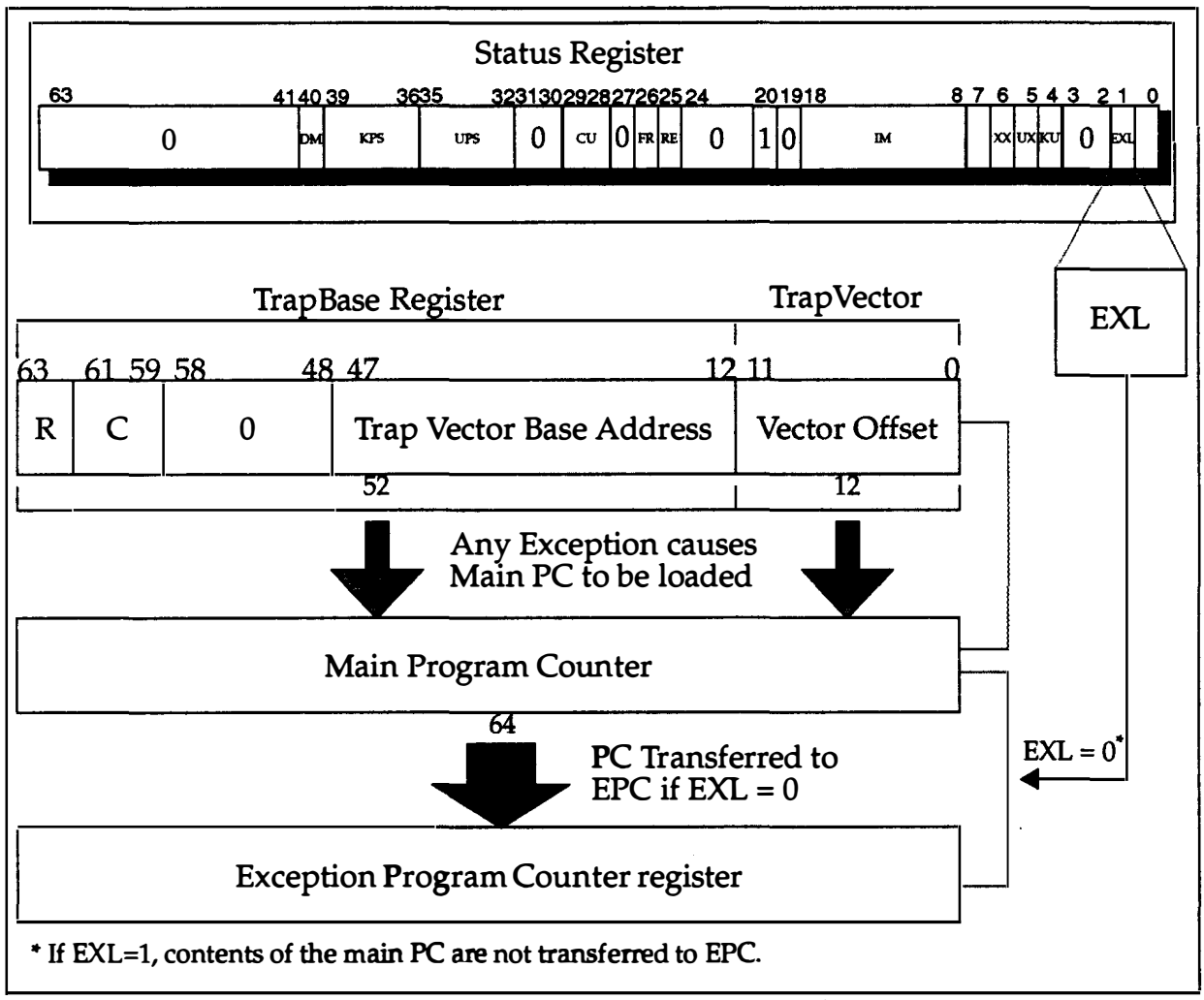


Figure 5-4 Handling a General Exception

The **Config** register contents do not change when a general exception is taken. The **Cause** register is modified depending on which general exception was taken. For example, a Coprocessor Unusable exception sets the **CE** bit, a Virtual Coherence exception sets the **VCI** bit, a floating point exception sets the **FPI** bit, etc. Figure 5-5 shows the Cause register.

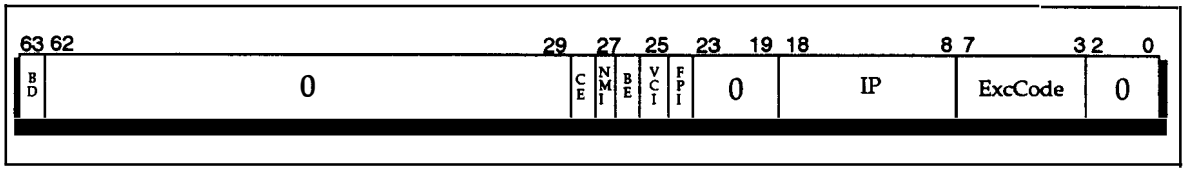


Figure 5-5 Cause Register

Multiple trap vector entry points are provided by hardware based on the exception taken. Table 5-1 below lists the various types of exception vectors and their TrapVector offsets.

Name	Vector	Cause Code	Description
Hard Reset	ResetVector 0x_9000_0000_1fc0_0000	---	Resets everything.
NMI	NMIVector 0x_9000_0000_1fc0_0000	---	Requested by external logic
TLB Refill, User, (EXL=0)	TrapBase + 0x_000	TLBL TLBS	The referenced address was to the User Region (UV) and did not match any TLB entry.
TLB Refill, Kernel Private (EXL=0)	TrapBase + 0x_400	TLBL TLBS	The referenced address was to the Kernel Private Region (KV1) and did not match any TLB entry.
TLB Refill, Kernel Global (EXL=0)	TrapBase + 0x_800	TLBL TLBS	The referenced address was to the Kernel Global Region (KV0) and did not match any TLB entry.
TLB Refill (EXL=1)	TrapBase + 0x_c00	TLBL TLBS	The referenced address (to any Region) did not match any TLB entry.
TLB Invalid	TrapBase + 0x_c00	TLBL TLBS	Virtual-address that matches an invalid TLB entry.
TLB Modified	TrapBase + 0x_c00	Mod	An attempt to write to a virtual address that did not have D bit in the corresponding TLB entry set.
Common Exceptions	TrapBase + 0x_c00	See Cause	General exception vector for all other exceptions.

Table 5-1 Exception Vectors

Exceptions are always generated by a particular instruction, and are reported precisely with respect to that instruction. All other events are termed interrupts. This section lists the types of General exceptions and how they are handled and serviced. These exceptions are indicated by loading a specific value into the 'Exception Code' field of the Cause register, located at bits [7:3]. Below is a listing of each general exception and the corresponding 5 bit hexidecimal value. Those Exception codes specific to the TLB are discussed in chapter 4, sections 4.7.1 through 4.7.3.

ExcCode Number	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB Exception Modification
2	TLBL	TLB Exception (Load or Instruction Fetch)
3	TLBS	TLB Exception (Store)
4	AdEL	Address Error Exception (Load or Instruction Fetch)
5	AdES	Address Error Exception (Store)
6-7	-----	Reserved by MIPS Technologies
8	Sys	SysCall Exception
9	Bp	Breakpoint Exception
10	RI	Reserved Instruction Exception
11	CpU	Coprocessor Unusable Exception
12	Ov	Arithmetic Overflow Exception
13	Tr	Trap Exception
14-31	-----	Reserved by MIPS Technologies

Table 5-2 Exception Codes

5.1.3.1 Address Error Exception

An address error exception occurs when an attempt is made to load, fetch, or store a quantity which is not properly aligned, to reference kernel address space from user mode, or when the region bits of the effective address do not match the region bits in the base register. The address error exception is precise and not maskable.

The AdEL and AdES exception codes are defined by ExcCodes 4 and 5 in Table 5-2 and indicate whether the instruction, defined by the contents of the **EPC register**, and the **BD** bit [63] of the **Cause** register, was a load or a store.

5.1.3.2 System Call Exception

The System Call exception occurs when an attempt is made to execute the corresponding instruction. The SysCall exception is precise and is not maskable. The common exception vector in Table 5-1 is used for this exception. The SysCall exception is defined by ExcCode 8 in Table 5-2. The **Sys** code in the **Cause** register is set.

The **EPC** points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which preceedes it, and the **BD** bit of the **Cause** register is set.

Control is transfered to the applicable system routine. To resume execution, the **EPC** must be altered so that the offending instruction is not re-executed.

5.1.3.3 Breakpoint Exception

The Breakpoint exception occurs when an attempt is made to execute the corresponding instruction. The Breakpoint exception is precise and is not maskable. The common exception vector in Table 5-1 is used for this exception. The Breakpoint exception is defined by ExcCode 9 in Table 5-2. The **Bp** code in the **Cause** register is set.

The **EPC** points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which preceedes it, and the **BD** bit of the **Cause** register is set.

Control is transfered to the applicable system routine. To resume execution, the **EPC** must be altered so that the offending instruction is not re-executed.

5.1.3.4 Reserved Instruction Exception

The Reserved Instruction exception occurs when an attempt is made to execute the corresponding instruction. The Reserved Instruction exception is precise and is not maskable. The common exception vector in Table 5-1 is used for this exception. The reserved Instruction exception is defined by ExcCode 10 in Table 5-2. The **RI** code in the **Cause** register is set.

The **EPC** points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which preceedes it, and the **BD** bit of the **Cause** register is set.

Control is transfered to the applicable system routine. To resume execution, the **EPC** must be altered so that the offending instruction is not re-executed.

5.1.3.5 Coprocessor Unusable Exception

The Coprocessor Unusable exception occurs when an attempt is made to execute the corresponding instruction. The Coprocessor Unusable exception is precise and is not maskable. The common exception vector in Table 5-1 is used for this exception. The CoProcessor Unusable exception is defined by ExcCode 11 in Table 5-2. The **CpU** code in the **Cause** register is set.

The EPC points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which preceedes it, and the **BD** bit of the **Cause** register is set.

Control is transfered to the applicable system routine. To resume execution, the **EPC** must be altered so that the offending instruction is not re-executed

5.1.3.6 Integer Overflow Exception

An Integer Overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction results in a two's complement overflow. This exception is precise and not maskable. The common exception vector in Table 5-1 is used for the Integer Overflow exception. The Integer Overflow exception is defined by ExcCode 12 in Table 5-2. The **Ov** code in the **Cause** register is set.

The **EPC** points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which precedes it, and the **BD** bit of the **Cause** register is set. The process executing at the time is handed a UNIX SIGFPE/FPE_INTOVF_TRAP signal.

5.1.3.7 Trap Exception

A Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TGEI, TGEUI, TLTI, TLTUI, TLEQI, or TNEI instruction results in a true condition. The trap exception is precise and not maskable. The common exception vector in Table 5-1 is used for this exception. The Trap exception is defined by ExcCode 13 in Table 5-2. The **Tr** code in the **Cause** register is set. The **EPC** points at the instruction which caused the exception, unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which preceedes it, and the **BD** bit of the Cause register is set. The process executing at the time is handed a UNIX SIGFPE/ FPE_INTOVF_TRAP signal.

5.1.3.8 Reserved Instruction Exception

There are two classes of reserved instruction exceptions relating to opcodes and values. An illegal coprocessor-1 opcode is reported via this exception. Value related exceptions cause a floating-point interrupt which is explained in section 5.3. The reserved instruction exception is precise and is not maskable.

The common exception vector is used for this exception. The **FPE** code in the **Cause** register is set. The **EPC** points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the **EPC** points at the branch instruction which preceedes it, and the **BD** bit of the **Cause** register is set.

Control is transfered to the applicable system routine. To resume execution, the **EPC** must be altered so that the offending instruction is not re-executed.

5.2 INTERRUPTS

The Interrupt exception occurs when one of the interrupt conditions are asserted. Interrupts are a type of exception and are imprecise and maskable. The general exception vector is used for servicing Interrupt exceptions. The **Int** code in the **Cause** register is set.

The Cause register in Figure 5-5 contains an 11 bit Interrupt Pending (IP) field which indicates the current interrupt requests. It is possible that more than one of the bits will be set at once, or even that no bits are set (if an interrupt is asserted and then deasserted before the Cause register is read). If the interrupt is caused by software, the condition is cleared by setting the corresponding **Cause** register bit to zero. Table 5-3 describes the IP field.

IP #	Cause Register Bits	Interrupt Type	Description
0..1	8..9	Software	Used to set or clear software interrupts
2..7	10..15	External	Set and cleared through the TBus. External interrupts cannot be cleared by software
8	16	Parity	Streaming cache even bank parity flag
9	17	Parity	Streaming cache odd bank parity flag
10	18	Overflow	Cycle counter overflow flag. Wired to the most significant bit of the cycle counter

Table 5-3 Interrupt Pending Fields

5.3 INTERRUPT TYPES

The R8000 Microprocessor supports 11 asynchronous interrupts. Two are managed by software, three are generated internally by hardware, and six are generated externally via the Tbus. Interrupts are posted in the **IP** field of the **Cause** register.

5.3.1 Virtual Coherence (Coprocessor) Interrupt

A Virtual Coherence (Coprocessor) interrupt (VCI) occurs when each of the following three conditions are true.

1) A coprocessor load or store hits in the streaming cache.
2) Virtual address bits [15:12] of the reference are different than the Virtual synonym (VS) bits stored in the streaming cache.
3) The SMM bit of the Status register is reset.

The VS bits in the streaming cache are set to bits [15:12] of the virtual address when a reference causes a streaming cache miss. The Virtual coherence interrupt is not maskable. When a VCI interrupt occurs the VCI bit in the **Cause** register is set, and the **BadPAddr** register contains the offending physical address along with bits [15:12] of the virtual

address. The offending instruction as well as an unbounded number of subsequent instructions will have already completed by the time the interrupt is posted. If additional virtual coherence (coprocessor) violations occur before the interrupt is serviced, no additional addresses are captured into **BadPAddr**.

Coprocessor memory references directly address the physically-addressed streaming cache and are not susceptible to virtual coherence violations. Therefore, there is no harm in allowing coprocessor memory references to complete even when they cause virtual coherence exceptions. However, these coprocessor memory references may fail to invalidate the on-chip integer data cache and thus not maintain coherency between the caches. If the processor is operating in Sequential Memory Model mode, the hardware interlocks subsequent integer memory references to avoid the possibility of reading stale data, at the expense of some performance loss. The virtual coherence interrupt is non-recoverable.

5.3.2 Floating Point Interrupt

The R8000 Microprocessor implements imprecise IEEE-compliant floating-point exceptions which are reported via the Floating-Point Interrupt.

In performance mode a floating-point operation that raises an exception will write the appropriate substitution value (e.g. NaN) into the register file, regardless of whether the exception is enabled or not, and continue execution. If the exception is enabled, bit IP10 in the **Cause** register is set and an imprecise floating-point interrupt occurs some time later.

In precise exception mode a floating-point operation that raises an enabled exception does not write the result into the register file, and bit IP10 in the **Cause** register is set to cause a precise floating-point interrupt with the **EPC** register pointing to the offending instruction. A disabled exception will write the appropriate substitution value (e.g. NaN) into the register file and continue execution.

Imprecise floating-point exceptions in normal mode can only be observed after the fact. Therefore trap handlers that count exceptions or abort the process are possible, but trap handlers that alter the result value based on an exception are not possible. Software should service this interrupt by clearing bit IP_{10} in the **Cause** register. Precise exception mode is specified by setting the DM bit in the **Status** Register. After a mode change an implementation-dependent number of instructions may not be floating-point instructions. Note that the **Cause** field in the R8010 **Floating Point Status** (FSR) register cannot be used to determine the cause of the interrupt in normal mode, and the **Flags** field must be used instead. However, the **Cause** field can be used in any mode to determine the status of the last floating-point operation.

5.3.3 Counter Overflow Interrupt

The **Count** register increments once per clock cycle. A counter-overflow interrupt is posted when the high-order bit becomes set. Bit IP10 in the **Cause** register is set and an imprecise counter-overflow interrupt occurs some time later.

Software must reset the **Count** register and clear bit IP10 in the **Cause** register.

5.3.4 Parity Error Interrupt

The R8000 Microprocessor has a very large off-chip streaming cache which is protected by parity bits. Parity errors are imprecise and nonrecoverable. Separate parity flags exist for both the even and odd banks of the streaming cache. Cause register bit IP8 pertains to the even bank of streaming cache. Cause register bit IP9 pertains to the odd bank of streaming cache.

If a parity error is detected on a floating-point load operation or an instruction or data cache refill, bit IP8 or IP9 in the **Cause** register is set and an imprecise parity-error interrupt occurs some time later. Parity errors should be logged and the process aborted. Software should clear bits IP8 and IP9 in the **Cause** register.

5.3.5 Bus Error Interrupt

A Bus Error interrupt exception is generated by board-level circuitry and indicates events such as bus time-out, backplane bus parity error, and invalid physical memory addresses or access types. The Bus Error interrupt is imprecise and not maskable.

The common exception vector is used for this interrupt and the process executing at the time is handed a UNIX SIGBUS signal. Since the TFP Microprocessor implementation of this exception is imprecise, the kernel should "clean up" any outstanding references (e.g. writes buffered in the I/O systems) by reading from uncacheable device registers to avoid accidentally killing the wrong process.

INITIALIZATION INTERFACE

6

This chapter describes the initialization and testing requirements for the R8000 Microprocessor Chip Set. The initialization sequence is discussed and code examples are provided for both the R8000 microprocessor and the Tag RAM. General testing requirements are also discussed including some specific testing characteristics which must be addressed.

In general the boot-up procedure consists of four steps:

1) Invalidate Instruction and Data Caches.
2) Flush Store Address Queue.
3) Set all Tag RAM states to invalid and assure that each of the four sets of any given Tag RAM index have different tags.
4) Initialize the TLB.

6.1 Instruction and Data Cache Invalidation

Invalidation of the instruction cache is accomplished by the invalidation of each entry in the cache. Uncached instructions are fetched from memory and placed in the cache and the entry is marked as invalid. Each entry in the instruction cache contains four 32-bit instructions (quadword). An instruction cache line (32 bytes) contains two quadwords. There is one valid bit per instruction cache line.

When the first entry is accessed with the "Jump Immediate" instruction cache hardware control logic marks the line as invalid due to the fact that the instructions were fetched from the following accress range;

9000_0004_1FCx_xxxx

The 9h value on the upper address bits indicates that the access is non-cachable and umapped in the TLB. The "Jump Immediate " instruction with the correct offset is used to step through the cache and invalidate the entries.

The Data Cache is invalidated using the DCTW instruction. The 4-bit valid field in the DCACHE register must be loaded with all zero's. The contents of the DCACHE register are undefined on reset.

Unlike the instruction cache, where the valid bit is included as part of the entry, the data cache has a separate valid bit RAM. One valid bit exists per word (32-bits) and there are 4 bits per valid RAM entry. Hence two writes to the valid RAM are necessary to invalidate one line (32 bytes) of the data cache.

Successive invalidations of the data cache valid RAM are accomplished by incrementing the VADDR register by 16 each time.

Note that any data loads, including non-cachable loads from a ROM device, are not guaranteed to work correctly until this step is completed due to the fact that the R8000

cached data as opposed to taking a data cache miss. Refer to section 6.5 for more information.

6.2 Flushing the Store Address Queue

The Store Address Queues are flushed using 32 writes to an even location (Address<3> clear) and 32 writes to an odd location (Address<3> set). The addresses used are two unused local address space registers and the writes are non-cacheable. The data is not important.

6.3 Tag RAM State Invalidation

In order to avoid memory reads going onto the backplane, the data RAM tests are divided into two groups. Each group does the same test but skips the part of Set 3 which the test itself is read through so that no cacheable reads or writes go to the area which has been invalidated by instruction reads and cause a read on the backplane.

During this step no single non-cacheable read should be done from any address except the ROM space which will read through the area of RAM which is being skipped (eg. the code itself).

Before any of the Tag RAMs are written, two non-cacheable reads from the ROM space at the current PC must be done to initialize the Annex pipe so that it doesn't corrupt the values after they are written.

The value of the tag RAM address used for set three is chosen so that any instruction fetches which occur during this step will put exactly the same value into the the location. The state, virtual synonym, and dirty bits should also be set so that the instructions will load the identical value.

The tag RAM addresses should be as follows:

Set	Tag
0	0x041CC
1	0x041DC
2	0x041EC
3	0x041FC

Address<21:20> will be written from the address used to access the tags and need not be varied explicitly for caches larger than 4MB.

The dirty bits do not matter; the state bits should all be exclusive; and the virtual synonym bits should be set to address<15:12> (part of the index) so that no virtual synonym misses can occur when used with virtual equals physical mapping.

For all processors except the master the procedure for the first two initializations of the streaming cache should be repeated except that the states should all be written to invalid instead of exclusive. These processors will not run C code and will not need a stack.

The master processor should be initialized in the same way as the slaves except that a stack region should be created at whatever index and address tag is convenient and of whatever size (up to the size of one set) may be convenient. The stack should be created now because we do not want to write the streaming cache tags again later.

The stack provision is made by picking a section of Set 1 (it must not be set 0 or 3). The Tag Address and state writes may be done after the complete slave style initialization has been accomplished. The state should be exclusive and the dirty bits do not matter. Again the virtual synonym bits must be set to Address<15:12> to avoid reads on the backplane. The Set Allow register must be set not to allow access to Set 1. (Note that Force Set Three is still on until the next step.) Before running User Mode, Set Allow should be set back to allow all sets in replacement.

The stack made in this way will work whether there is memory board in the system or not. The states are already exclusive and do not have to be filled from memory and the set allow does not include Set 1 so that the stack cannot be kicked out. Set 0 will be kicked out by non-cacheable accesses which miss in the cache, so it is not used; Set 3 will be kicked out in Force Set Three Mode, so it is not used.

All the cache lines in the system are now in legal states; Some of them are exclusive in the master processor and invalid elsewhere while the rest are invalid everywhere. In addition, at every index in every cache, the address tags in the four sets are all different.

6.4 Initializing the TLB

Initialization of the TLB consists of the following two steps;

1) Mark all entries invalid in the physical address (PTAG) portion of the TLB.
2) Assure that each set of each index in the virtual tags (VTAG) portion of the TLB contains different tag information.

The entries of the PTAG are invalidated by executing the TLBW instruction for each entry. The **EntryLo** register must be set to all zero's as its contents are written to the PTAG. Writing all zero's to the register assures that the valid bit for the entry will not be set when the TLB entry is written.

The EntryHi register is used to write the VTAG portion of the TLB. The initialization of the VTAG must assure that none of the three sets of a given entry contains the same tag information. The following sequence shows an example of how this can be accomplished.

1) Load EntryHi with all zero's.
2) Write EntryHi to set 0 of a given index.
3) Increment the Virtual Page Number (VPN) field of EntryHi by 1.
4) Write EntryHi to set 1 of the same index.
5) Again increment the Virtual Page Number (VPN) field of EntryHi by 1.
6) Write EntryHi to set 2 of the same index.

This sequence will assure that an interrupt does not occur when accessing the TLB for the first time due to multiple sets having the same value.

6.5 R8000 Microprocessor Functional Characteristics

The following list of characteristics must be understood when attempting to perform a boot-up procedure on the R8000 Microprocesor. Each characteristic is explained and a solution is offered.

- Non-Cachable instruction fetches change the state of the instruction cache.

When loading instructions from unmapped address space virtual address bits 61:59 are encoded to contain the coherence protocol. For uncached instructions the value of virtual address bits 63:59 can be one of two values:

63	62	61	60	59	
1	0	0	0	0	Write Gatherer for Graphics
1	0	0	1	0	Normal Mode - Uncached Sequential

As shown above, 10010 forces the uppermost hex value of the address to 9h, while 10000 forces the value to 8h. If the uppermost value is 8h, the instruction cache is checked and if the value is there an instruction cache hit occurs. A miss causes the instruction cache hardware control logic to force the state of bit 60 to a logical one, causing the access to become uncachable sequential. Once the instruction is fetched and brought into the instruction cache the entry is executed again. Executing each instruction twice eliminates the state change from occurring. An access to 8000_xxxx_xxxx_xxxx causes the access to be non-cachable. The first execution causes an instruction cache miss but the instruction is held in the cache. The second execution of the instruction results in a hit and does not

corrupt the state of the streaming cache.

- Non-Cachable data fetches change the state of the data cache.

Uncached data references use the data cache as a buffer. Uncached data is placed in the cache and marked invalid. Uncached data references can happen anytime and do not have any relationship to when the data cache is invalidated.

The data cache must be invalidated each time a page is converted from cachable to non-cachable. Data loads from the ROM must not be performed until after the data cache has been invalidated. If during power-up the data cache happens to contain the address which is being accessed the invalid data associated with that data cache location will be read as opposed to the location in the ROM.

- Non-cachable data loads which check the tag address or tag state information corrupt the streaming cache location to which the data was mapped. This location may or may not be the same as the location which the tag controls.

The data and corresponding set address must be written into each of the tags and corresponding state in both tag RAM's and then read back to assure each is functioning correctly. As with the non-cachable fetch routine explained above, the entire test should be run twice and the results ignored on the first pass. This also assures that the code will fit into the 16 KByte direct mapped instruction cache.

The instruction reads will modify the values placed in set 3 in some of the tags. To avoid these modifications the entire test should be run on sets 0, 1, and 2 by writing to all locations in these sets and then reading all locations back. The test of set 3 can then be run by writing to each location in set 3 and then immediately reading it back before the location can be potentially modified by another read.

The other solution for avoiding these modifications is to write the values to all four sets and then read them back in a particular order which begins with the location in set 3 which reads through itself and the other 63 locations which read through the same line and expand from there. These locations are different between the tag RAM's.

- Dirty Bit Testing

While writes to the Tag RAM addresses are straight forward, writes to the Tag RAM states are more complex. The method for writing the dirty bits to non-identical values requires two writes; One for all of the sectors with the dirty bit clear, and another for the remaining sectors with their dirty bits set. The write enable bits within the data are used to accomplish this.

- Non-cachable Accesses put Data into the Streaming Cache

Between the time when a hard reset is performed until the intitalization procedure is complete all data or instruction should be placed only in set 3 of the streaming cache. At the conclusion of the boot procedure the condition should be removed so that all four sets of the cache can be used.

6.6 Initialization Code Examples

The following routines contain examples for initializing the data and instruction caches, tag RAM's, streaming cache, and registers.

```
/
*************************************************************\
* This file contains the initialization and startup code for  *
* the R8000 Microprocessor.*                                  *
\*************************************************************/

#include "ml.h"
#include <asm.h>
#include <sys/regdef.h>
#include <sys/sbd.h>
#include <sys/cpu.h>
#include <sys/fpu.h>
#include <sys/loaddrs.h>
#include <sys/EVEREST/gda.h>
#include <sys/EVEREST/everror.h>
#include <sys/EVEREST/evconfig.h>
#include "ip21prom.h"
#include "prom_leds.h"
#include "prom_config.h"
#include "pod.h"
#include "pod_failure.h"
#include "prom_intr.h"


#define  CAUSE_NMI    0x08000000   /* Non-maskable interrupt

#define PROM_SR       SR_CU1|SR_FR|SR_PAGESIZE
#define SR_CU1        0x20000000        /* coprocessor 1 usable */
#define SR_FR         0x04000000    /* enable additional fp regs */

#define C0_TLBSET     $0    /* Select set in set-associative tlb */
#define C0_TLBLO      $2    /* Low half of tlb entry */
#define C0_UBASE      $4    /* Base of user page tables */
#define C0_TRAPBASE   $6    /* Base addr of exc. vectors */
#define C0_BADVADDR   $8    /* Virtual address register */
#define C0_COUNT      $9    /* Free-running counter */
#define C0_TLBHI      $10   /* High half of tlb entry */
#define C0_SR         $12   /* Status register */
#define C0_CAUSE      $13   /* Cause register */
#define C0_EPC        $14   /* Exception program counter */
#define C0_WORK0      $18    /* Uninterpreted temp. register */
#define C0_WORK1      $19    /* Uninterpreted temp. register */
#define C0_PBASE      $20  /* Base of kernel private page tables*/
#define C0_GBASE      $21   /* Base of kernel global page tables */
```

```
#define C0_WIRED      $24    /* Indices of tlb wired entries */
#define C0_DCACHE     $28    /* Dcache control register */
#define C0_ICACHE     $29    /* Icache control register */

#define ICACHE_LINE_CODE                                    \
        addu    zero, zero, zero;       /* 0 */             \
        addu    zero, zero, zero;       /* 4 */             \
        addu    zero, zero, zero;       /* 8 */             \
        addu    zero, zero, zero;       /* 12 */            \
        addu    zero, zero, zero;       /* 16 */            \
        addu    zero, zero, zero;       /* 20 */            \
        addu    zero, zero, zero;       /* 24 */            \
        addu    zero, zero, zero        /* 28 */

/* Four icache lines per streaming cache line (128 bytes) */
#define FOUR_LINES_CODE         \
        ICACHE_LINE_CODE;       \
        ICACHE_LINE_CODE;       \
        ICACHE_LINE_CODE;       \
        ICACHE_LINE_CODE        \

/* 32 icache lines per 1k */
#define THIRTYTWO_LINES_CODE                        \
        FOUR_LINES_CODE; FOUR_LINES_CODE;           \
        FOUR_LINES_CODE; FOUR_LINES_CODE;           \
        FOUR_LINES_CODE; FOUR_LINES_CODE;           \
        FOUR_LINES_CODE; FOUR_LINES_CODE            \

#define POD_STACKADDR       0xa8000000000fc000
#define SAQ_INIT_ADDRESS    0x9000000018000380  /* 2 unused local
                                                #CC register address */
#define SAQ_DEPTH            32

#define BB_BUSTAG_ADDR      SBUS_TO_KVU(0x18080000) /* bus tag
                                                    # address */
#define BB_PTAG_E_ADDR       SBUS_TO_KVU(0x18100000) /* proc tag
                                                   # even address */
#define BB_BUSTAG_ST SBUS_TO_KVU(0x180c0000) /* bus tag state */
#define BB_PTAG_E_ST SBUS_TO_KVU(0x18140000) /* proc tag even
                                                    # state */
#define BTAG_ST_INIT        0x000000000001f000  /* bus tag state
                                                  # init value */
#define PTAG_ST_INIT        0x0000007c00000000  /* proc tag state
                                                 # init value */
```

```
          .text
          .set      noreorder
          .set      at

#define GOTO_CHANDRA_MODE\

          LA        k0, 8f;                         \
          LI        kl, 0x8fffffffffffffff;         \
          and       k0, kl;                         \
          jr        k0;                             \
          nop;                                      \
/*
 * entry -- When the CPU begins executing it jumps
 *          through the power-on vector to this point.
 *          This routine initializes the processor and starts
 *          basic system configuration.
 */
LEAF(entry)
          /*
           * Check to see if we got an NMI.  If so, jump to the NMI
           * handler code.
           */
          DMFC0(k0, C0_CAUSE) # Load the Cause register
          and       k0, CAUSE_NMI       # Check for an NMI
          bnez      k0, bev_nmi         # IF NMI, jump to NMI
                                        #  handler
          nop

initialize:
          dla       k0,trap_table
          DMTC0(k0,C0_TRAPBASE)/* init TrapBase register */

          dli       v0, PROM_SR
          DMTC0(v0, C0_SR)    # Put SR into known state

          jal       pon_invalidate_IDcaches#Invalidate I&D caches
          nop

          jal       init_cpu  # Set up the main processor
          nop                                   # (BD)


          /*
           * Clear the cache tags
           */
          jal     pon_invalidate_dcache# Invalidate dcache tags
```

```
        nop

/*
 * Routine init_cpu
 *      Set up the R8000's basic control registers and TLB.
 *
 *      For R8000, may need to add code for initializing Icache,
 *      Dcache, COP0 registers, SAQs, and Gcache.
 */

LEAF(init_cpu)
        move    s6, ra                          # Save return
                                                # address

        /* Initialize COP0 register.  C0_TRAPBASE assumed setup
         * before calling this rountine. */

        dli     v0, PROM_SR
        DMTC0(v0, C0_SR)    # Put SR into known state
        DMTC0(zero, C0_TLBSET)# Clear TLBset
        DMTC0(zero, C0_TLBLO)# Clear EntryLo
        DMTC0(zero, C0_UBASE)
        DMTC0(zero, C0_BADVADDR)# Clear VAddr
        DMTC0(zero, C0_COUNT)# Clear Counts
        DMTC0(zero, C0_TLBHI)# Clear EntryHi
        DMTC0(zero, C0_CAUSE)# Clear interrupts
        DMTC0(zero, C0_EPC) # Clear Exception Program Counter

/* No need to init Config reg.  It is initialized by hardware at
reset. */

        DMTC0(zero, C0_WORK0)
        DMTC0(zero, C0_WORK1)
        DMTC0(zero, C0_PBASE)
        DMTC0(zero, C0_GBASE)
        DMTC0(zero, C0_WIRED)
        DMTC0(zero, C0_DCACHE)
        DMTC0(zero, C0_ICACHE)

        ctc1    zero, fpc_csr       # clear fp csr

        jal     flush_tlb           # Clear out the TLB
        nop

        jal     flush_SAQueue       # Initialize/flush SAQs
        nop
```

```
        jal     pon_initialize_scache# Initialize G-cache
        nop

        j       s6                  # Return to caller
        nop
        END(init_cpu)



LEAF(pon_invalidate_IDcaches)
        .set    noreorder
        .align 5                    # first invalidate the
icache

        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 2k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 4k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 6k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 8k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 10k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 12k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 14k */
        THIRTYTWO_LINES_CODE; THIRTYTWO_LINES_CODE/* 16k */

        j       pon_invalidate_dcache# now go do the dcache
        nop
END(pon_invalidate_IDcaches)



/* Initialize DCache by invalidating it.  DCache is 16KB, with
 * 32Byte line, 28 bits Tag + 1 bit Exclusive + 8 bits Valid + 32
 * Bytes Data. Need two consecutive writes to half-line boundaries
 * to clear Valid bits for each line.
 */



LEAF(pon_invalidate_dcache)
        .set    noreorder
        move    t3, ra
        li      v1, 16      # size of the half of Dcache line
        jal     get_dcachesize
        nop

      # v1: half line size, v0: cache size to be initialized
```

```
        dli    t1,POD_STACKADDR# Starting address for Dcache
                                # initialization.
        daddu   t2, t1, v0      # t2: loop terminator for each
                                # cache line: mark it invalid
1:
        DMTC0(t1,C0_BADVADDR)
        DMTC0(zero,C0_DCACHE)# clear all (4) Valid bits
        dctw                # write to the Dcache Tag
        ssnop; ssnop; ssnop # pipeline hazard prevention
      .set    reorder

      daddu   t1, v1          # increment to next half-line
      bltu    t1, t2, 1b# t2 is termination count
      j       t3
END(pon_invalidate_dcache)



/* Initialize Store Address Queue by issuing (SAQ_DEPTH) Even and
 * (SAQ_DEPTH) Odd uncached-writes to two even and odd aligned
 * local registers - 2 unused local CC register addresses used
 */

LEAF(flush_SAQueue)
      .set    noreorder
     dli     t0, SAQ_INIT_ADDRESS # address to start (writing) at
     li     t1, SAQ_DEPTH         # number of entries in the queue
1:
      sd      zero, 0(t0)         # write (even) 8 bytes
      sd      zero, 8(t0)         # write (odd) 8 bytes
      addi    t1, -1
      bnez    t1, 1b
      nop
      j       ra
      nop
      .set    reorder
END(flush_SAQueue)



LEAF(pon_initialize_scache)
      .set    noreorder
        move      s4, ra
```

```
        dli     t0, 0x900000001fc0f000
        ld      t1, 0(t0)
        ld      t1, 0(t0)

        dli     v0, BB_BUSTAG_ADDR# bus tag address base
        dli     v1, BB_PTAG_E_ADDR# proc tag address base
        dli     t8, BB_BUSTAG_ST# bus tag state base
        dli     t9, BB_PTAG_E_ST# proc tag state base
        li      a1, BTAG_ST_INIT# bus tag state init. value
        dli     a2, PTAG_ST_INIT# proc tag state init value
        li      a3, 0x01000000  # increment value for next set
                                # tag address.
        li      s3, 0x010000 # increment value for address of
                             # tag address - next set
1:
        sll     t2, t1, 3 # shift index count into appropriate
                          # position
        or      t3, v0, t2# address of bus tag addr.
        li      ta0, 0x01cc00000# data to be written into set
                                # 0 of bus & proc
                           # tag addr. (i.e.  start from set 0)
        or      ta1, v1, t2# address of proc tag addr.
        or      ta2, t8, t2# address of bus tag state
        or      ta3, t9, t2# address of proc tag state
        li      a0, 4               # set loop count

/* Write into 4 sets of the index, starting from set 0  */

2:
        sd      ta0, 0(t3)# write into bus tag addr.
        sd      ta0, 0(ta1)# write into proc tag addr.
        sd      a1, 0(ta2)# write into bus tag state
        sd      a2, 0(ta3)# write into proc tag state


        addu    ta0, a3   # increment tag value to be written
                          # to next set
        daddu   t3, s3    # incr. addr. of bus tag addr. to
                          # next set
        daddu   ta1, s3   # incr. addr. of proc tag addr. to
                          # next set
        daddu   ta2, s3   # incr. addr. of bus tag state to
                          # next set
        daddu   ta3, s3   # incr. addr. of proc tag state to
                          # next set
        addi    a0, -1
```

```
          bgt       a0, zero, 2b# are we done with all 4 sets?
          nop
        addiu   t1, 1
        ble     t1, t0, 1b
        nop
                                          # done with no error
        move    v0, zero                  # zero retn -> no error
        j       s4
          nop
          .set      reorder

99:
        #report error
        .set    noreorder

          j         s4
          nop
        .set    reorder
END(pon_initialize_scache)
```

CLOCK INTERFACES

7

The R8000 Microprocessor Chip Set contains dedicated clock interfaces for each component in the system. The clock interface for the integer and floating point units are identical. The clock interfaces for the integer, floating point, tag RAM, and streaming cache SRAM modules are covered in the following sections.

The clock interface signals for the R8000 Microprocessor and R8010 Floating Point Unit are connected as shown in Figure 7-1.

7.1 R8000/R8010 CLOCK INTERFACE

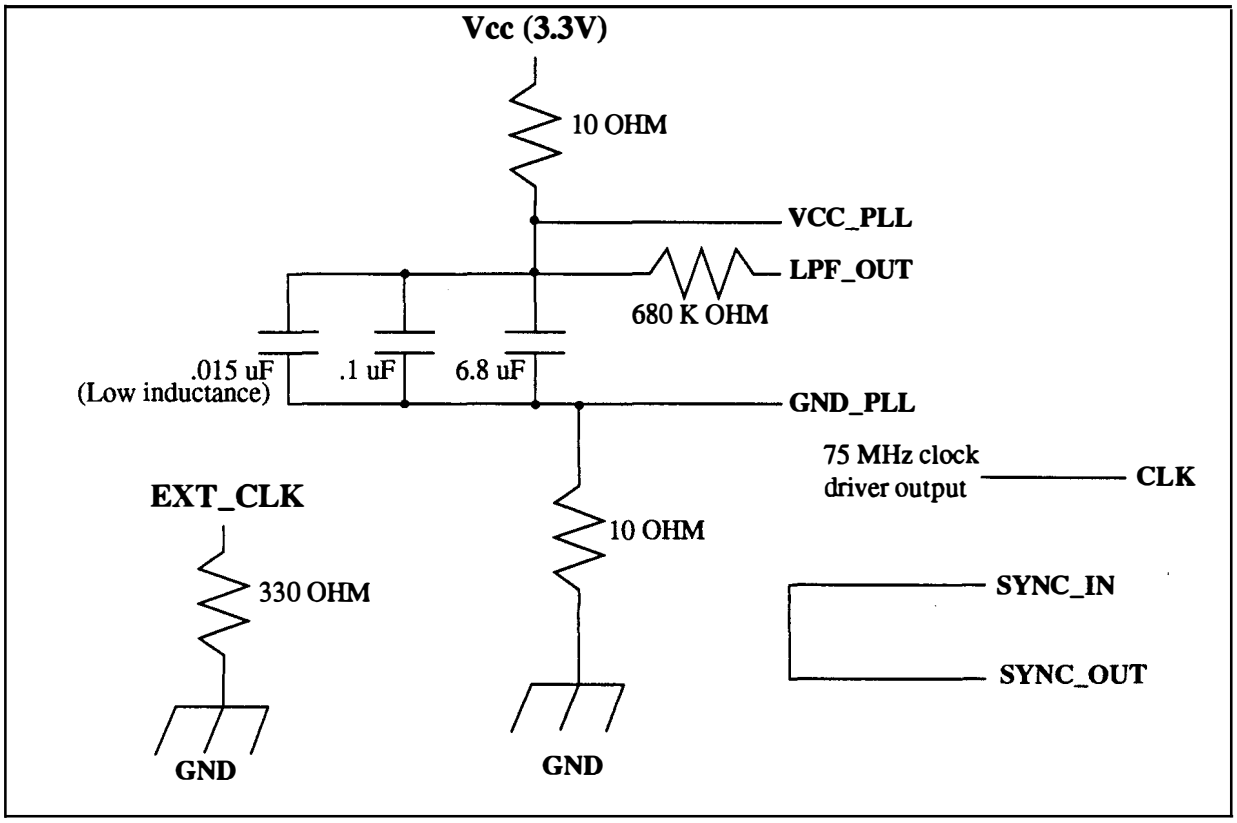


Figure 7-1 R8000/R8010 Clock Interface Connection Diagram

CLK (Reference Clock) Active High Input

Master input clock to the Phase Lock Loop (PLL) circuitry of the R8000. The output of the PLL is then used as the master clock for the chip. CLK is normally connected directly to the ouput of the external clock driver. In most cases it is desirable to use the PLL circuitry, but for those applications which to not wish to use the PLL, the clock drivers should be connected to EXT_CLK.

EXT_CLK (External Clock) Active High Input

The EXT_CLK input allows the system designer to bypass the internal PLL of the R8000 and drive the chip directly from the system clock. When not in use this pin should be tied to ground through a 330 ohm resistor.

GND_PLL (Ground Phase Lock Loop)

Ground source for the phase lock loop circuitry. GND_PLL can be connected to VCC_PLL through .1 microfarad and .015 microfarad capacitors in parallel. See figure 7.1.

LPF_OUT (Low Pass Filter Output) Active High Output

LFP_OUT is a special pin used to test the PLL circuitry during component test for monitoring the status of the low-pass filter. LPF_OUT must be connected to VCC_PLL through a 680K ohm resistor.

SYNC_IN (Synchronized PLL input) Active High Input

SYNC_IN is part of the PLL feedback path and must be connected to SYNC_OUT in order for the PLL circuitry to work correctly. The pins are made avaliable externally to allow the user to manually alter the phase of the PLL by lengthening the connection between SYNC_IN and SYNC_OUT.

SYNC_OUT (Synchronized PLL input) Active High Output

SYNC_OUT is part of the PLL feedback path and must be connected to SYNC_IN in order for the PLL circuitry to work correctly. The pins are made avaliable externally to allow the user to manually alter the phase of the PLL by lengthening the connection between SYNC_IN and SYNC_OUT.

VCC_PLL (Voltage Phase Lock Loop)

Voltage source for the phase lock loop circuitry. Connected to a regulated 3.3 volt source. VCC_PLL can be connected to GND_PLL through .1 and .015 microfarad capacitors in parallel.

7.2 TAG RAM CLOCK INTERFACE

The clock interface for the even and odd bank Tag RAM's in the R8000 Microprocessor Chip Set is identical. The following diagram shows how the vaious clock interface pins are connected together.

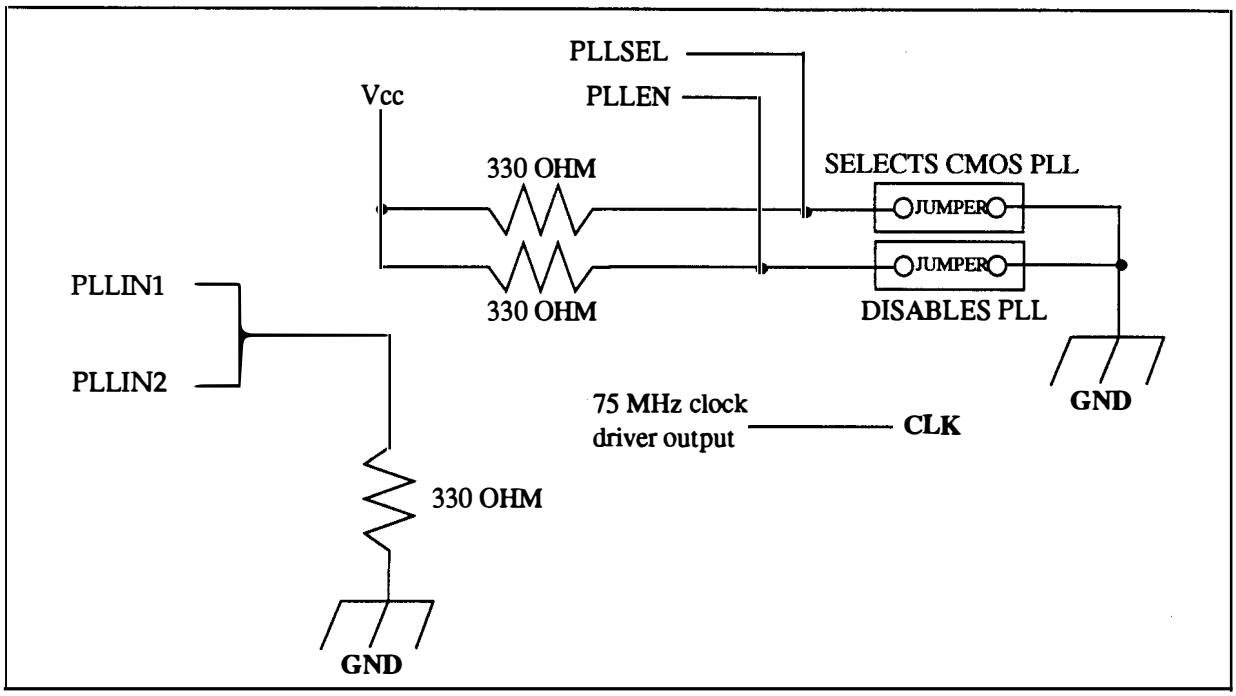


Figure 7-2 Tag RAM Clock Interface Connection Diagram

CLK (Clock Input) Active high Input

Clock input for the Tag RAM. The input frequency is 75 MHz. The clock can drive the device directly, or function as an input to a phase lock loop based on the state of the input pin PLLEN. See figure 7-2.

PLLEN (Phase Lock Loop Enable) Active High Input

Enable pin for the bi-polar phase lock loop. Assertion of PLLEN enables the phase lock loop. Deassertion of this pin disables the phase lock loop and allows the input clock to drive the device directly. See figure 7-2.

PLLIN1 (Phase Lock Loop 1) Active High Input

There are two phase lock loop circuits on-chip. PLLIN1 is the 75 MHz clock input for the CMOS phase lock loop. This pin should be tied to ground for normal PLL operation. See figure 7-2.

PLLIN2 (Phase Lock Loop 1) Active High Input

PLLIN2 is the 75 MHz clock input for the Bi-Polar phase lock loop.This pin should be tied high for normal PLL operation. See figure 7-2.

PLLSEL (Phase Lock Loop Select) Active High Input

There are two types of phase lock loop circuits inside the Tag RAM, one of which is used for testing purposes. PLLSEL must be tied high for proper operation of the device. See figure 7-2.

7.3 STREAMING CACHE CLOCK INTERFACE

The clock interface to the streaming cache Data RAM's consists of twelve separate clocks per data bank, 24 clocks in all. Six clocks are used for the upper 32-bit module and six clocks for the lower 32-bit module for the even data bank. Six clocks are used for the upper 32-bit module and six clocks for the lower 32-bit module for the odd data bank. Each module contains 12 devices; eight 256K X 4 Data RAM's, one 256K X 4 Parity RAM, and three address buffers. Each clock drives two devices.

EU_CLKA (Even Upper Clock A) Active High Input
EU_CLKB (Even Upper Clock B) Active High Input
EU_CLKC (Even Upper Clock C) Active High Input
EU_CLKD (Even Upper Clock D) Active High Input
EU_CLKE (Even Upper Clock E) Active High Input
EU_CLKF (Even Upper Clock F) Active High Input

EL_CLKA (Even Lower Clock A) Active High Input
EL_CLKB (Even Lower Clock B) Active High Input
EL_CLKC (Even Lower Clock C) Active High Input
EL_CLKD (Even Lower Clock D) Active High Input
EL_CLKE (Even Lower Clock E) Active High Input
EL_CLKF (Even Lower Clock F) Active High Input

Figure 7-3 below shows the clock connections to the upper and lower halves of the even bank. Twelve other separate clocks interface to the odd bank in the exact same manner as shown in figure 7-3.

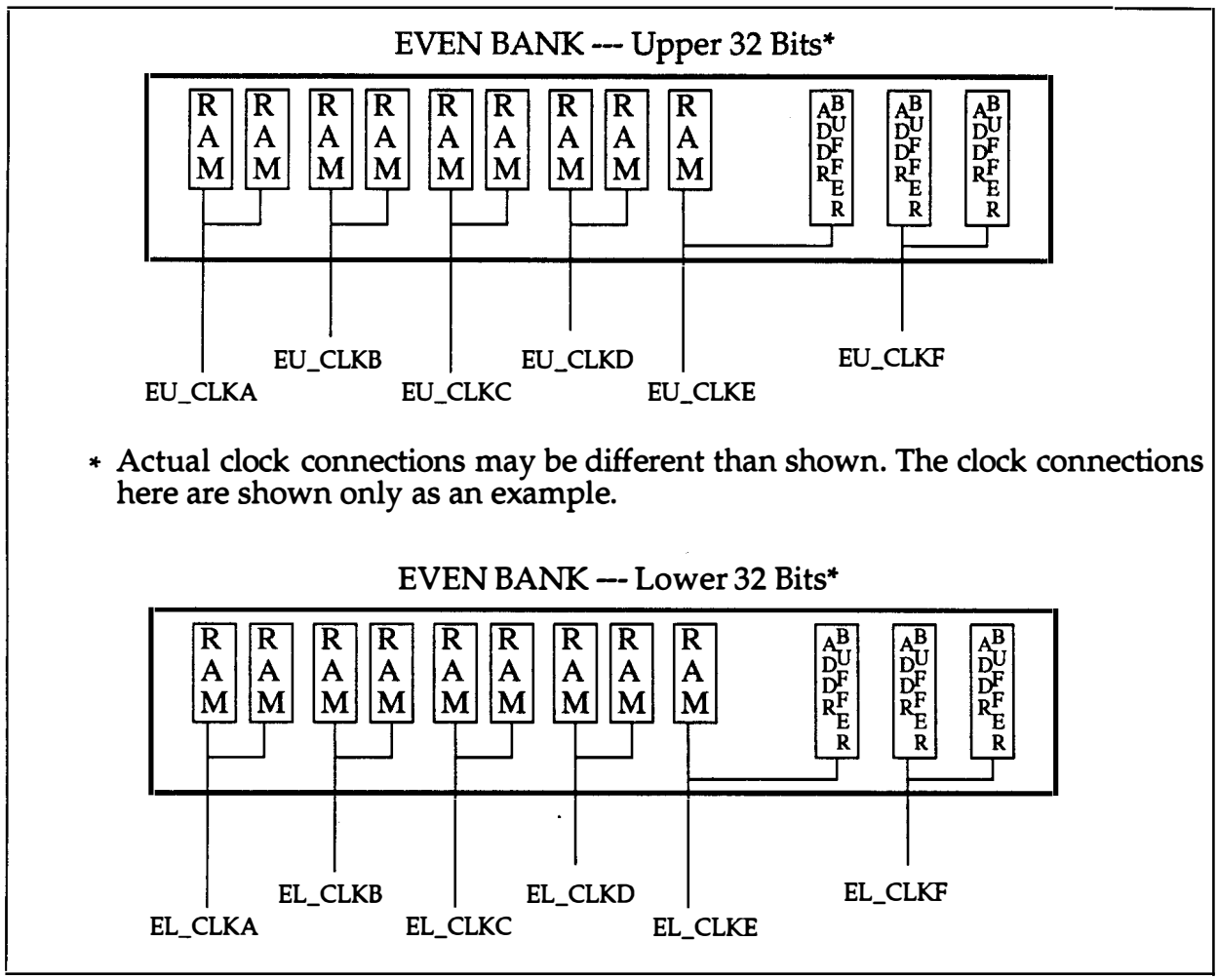


Figure 7-3 Streaming Cache SRAM Clock Connection Diagram

ELECTRICAL SPECIFICATIONS AND MECHANICAL DATA

8

The following chapter contains electrical specifications and packaging information for the R8000 Microprocessor, R8010 FPU, Tag RAM, and Streaming Cache Data RAM's.

8.1 ELECTRICAL SPECIFICATIONS

The following sections lists the electrical specifications for each of the components in the R8000 Microprocessor Chip Set. The electrical specifications for the R8000 Microprocessor and R8010 Floating Point Unit are exactly the same.

8.1.1 R8000 Microprocessor/R8010 FPU

SYMBOL	PARAMETER	RATING	UNIT
V_{CC}	Supply Voltage	-0.5 to +7.0	V
V_{IN}	Input Voltage	-0.5 to VCC+0.5	V
V_O	Output Voltage	-0.5 to VCC+0.5	V
P_D	Allowable Power Dissipation	TBD	W
T_{OPR}	Operating Temperature	0 to +70	^oC
T_{STG}	Storage Temperature	-55 to +150	^oC

Table 8-1 R8000/R8010 Absolute Maximum Ratings

SYMBOL	PARAMETER	MIN	MAX	UNIT
C_{IN}	Input Capacitance	--	5	pF
C_{CK}	Clock Input Capacitance	--	8	pF
C_{OUT}	Output Capacitance	--	8	pF
V_{CC}	Supply Voltage	3.135	3.465	V
I_{CC}	Supply Current[1]	--	4.3	A
V_{IH}	Input High Signal Voltage	2.2	--	V
V_{IL}	Input Low Signal Voltage	--	0.8	V
V_{OH}	Output High Signal Voltage	2.4	--	V@4ma
V_{OL}	Output Low Signal Voltage	--	0.4	V@8ma

Table 8-2 R8000/R8010 DC Electrical Characteristics

1. 15W @ 75 MHz, V_{CC}=Max, all outputs switching

SYMBOL	PARAMETER[1]	MIN	MAX	UNIT
t_{CP}	Clock Period	13.3	inf	nS
t_{CH}	Clock Pulse High	4	9	nS
t_{CL}	Clock Pulse Low	4	9	nS
t_S	Setup Time[2]	2	--	nS
t_H	Hold Time	1	--	nS
t_{CQ}	Clock to Output[3]	2	7	nS
t_{HZ}	Clock Output to High Impedance	2	7	nS
t_{LZ}	Clock to Output Low Impedance	2	7--	nS

Table 8-3 R8000/R8010 AC Tuning Characteristics

1. All parameters are specified over the range 0-70 °C.
2. Setup and hold times assume a 3 nS rise and fall time between V_{IL} and V_{IH} and with input pulse levels of 0 to 3.0V. Clock rise time is 1 nS between V_{IL} and V_{IH} and with pulse level of 0 to 3.0V. Input timing measuremtnt reference level is 1.5V.
3. Standard output loading of 50 pF with all outputs switching simultaneously. Output timing measurement reference levels are 0.8V and 2.0V.

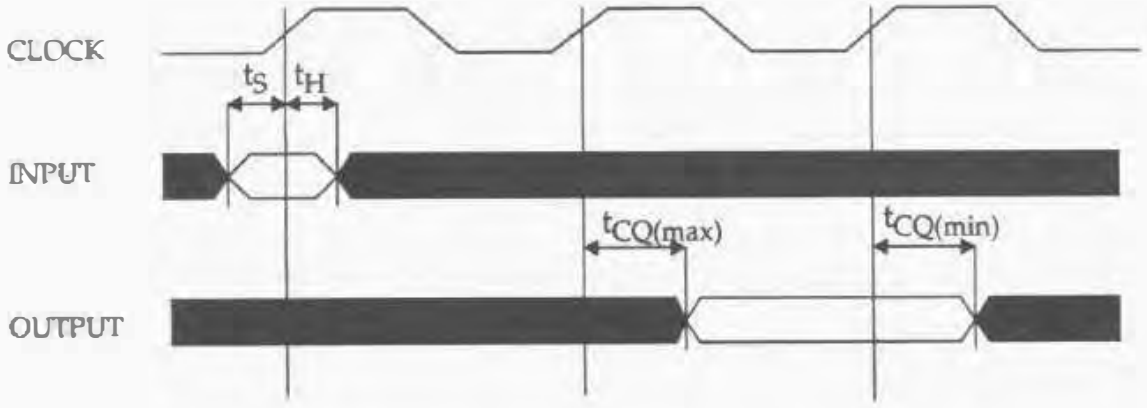


Figure 8-1 R8000/R8010 Setup, Hold, and Clock-to-Out Tuning Parameters

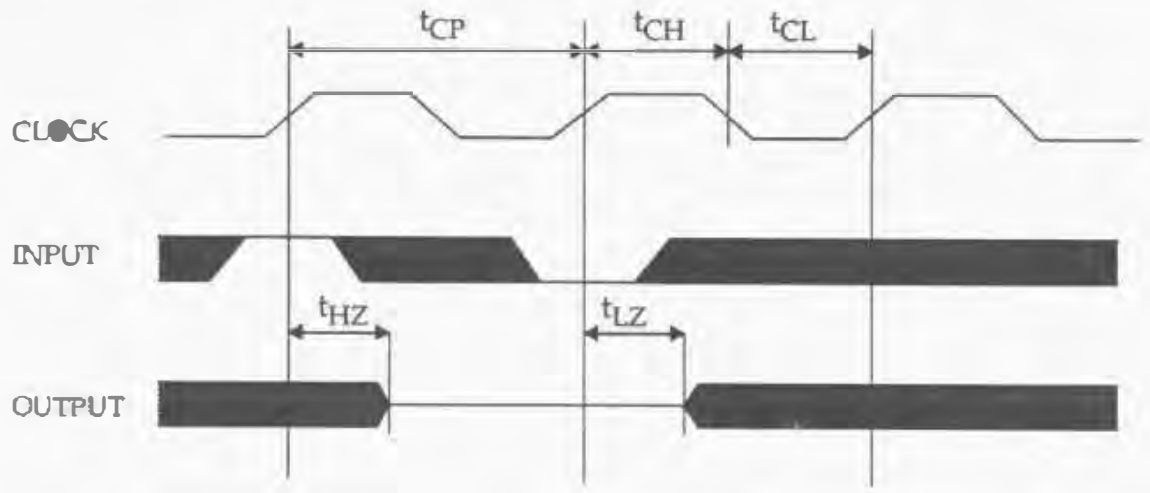


Figure 8-2 R8000/R8010 Clock and Tristate Timing Parameters

8.1.2 TAG RAM

SYMBOL	PARAMETER	RATING	UNIT
V_{CC}	Supply Voltage	-0.5 to+7.0	V
V_{IN}	Input Voltage	-0.5 to VCC+0.5	V
V_O	Output Voltage	-0.5 to VCC+0.5	V
P_D	Allowable Power Dissipation	2	W
T_{OPR}	Operating Temperature	0 to +70	°C
T_{STG}	Storage Temperature	-55 to +150	°C

Table 8-4 Tag RAM Absolute Maximum Ratings

SYMBOL	PARAMETER	MIN	MAX	UNIT
C_{IN}	Input Capacitance	–	5	pF
C_{CK}	Clock Input Capacitance	–	8	pF
C_{OUT}	Output Capacitance	–	8	pF

Table 8-5 Tag RAM DC Electrical Characteristics

SYMBOL	PARAMETER	MIN	MAX	UNIT
V_{CC}	Supply Voltage	4.75	5.25	V
I_{CC}	Supply Current[1]	--	570	mA
V_{IH}	Input High Signal Voltage	2.2	--	V
V_{IL}	Input Low Signal Voltage	--	0.8	V
V_{OH}	Output High Signal Voltage (except DATSA)	2.4	--	V@4ma
V_{OL}	Output Low Signal Voltage (DATSA)	--	0.4	V@8ma
V_{OH}	Output High Signal Voltage (DATSA)[2]	2.4	--	V@15ma
V_{OL}	Output Low Signal Voltage (DATSA)	--	0.4	V@48ma

Table 8-5 Tag RAM DC Electrical Characteristics

1. 3W @80 MHz, V_{CC}=Max, all outputs switching
2. These signals drive heavy capacitive loads. All are complementary outputs.

SYMBOL	PARAMETER[1]	MIN	MAX	UNIT
t_{CP}	Clock Period	13.3	inf	nS
t_{CH}	Clock Pulse High	4	--	nS
t_{CL}	Clock Pulse Low	4	--	nS
t_S	Setup Time[2] (except ESA)	2	--	nS
t_S	Setup Time[2] (ESA)	3	--	nS
t_H	Hold Time	1	--	nS
t_{CQ}	Clock to Output[3]	2	7	nS
t_{HZ}	Clock Output to High Impedance	2	10	nS
tLZ	Clock Output to Low Impedance	1	10	nS

Table 8-6 Tag RAM AC Timing Characteristics

1. All parameters are specified over the range 0-70 ^{o}C.
2. Setup and hold times assume a 3 nS rise and fall time between V_{IL} and V_{IH} and with input pulse levels of 0 to 3.0V. Clock rise time is 1 nS between V_{IL} and V_{IH} and with pulse level of 0 to 3.0V. Input timing measuremtnt reference level is 1.5V.
3. Standard output loading of 50 pF with all outputs switching simultaneously. Output timing measurement reference levels are 0.8V and 2.0V.

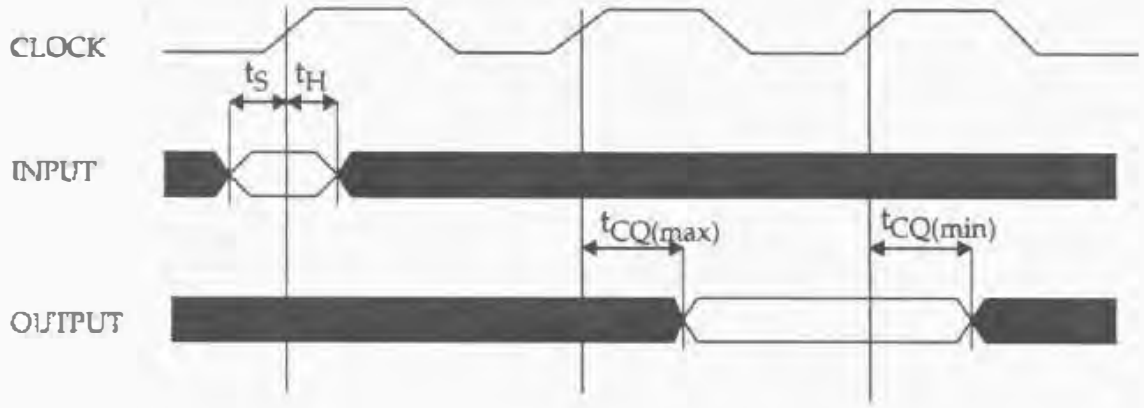


Figure 8-3 Tag RAM Setup, Hold, and Clock-to-Out Timing Parameters

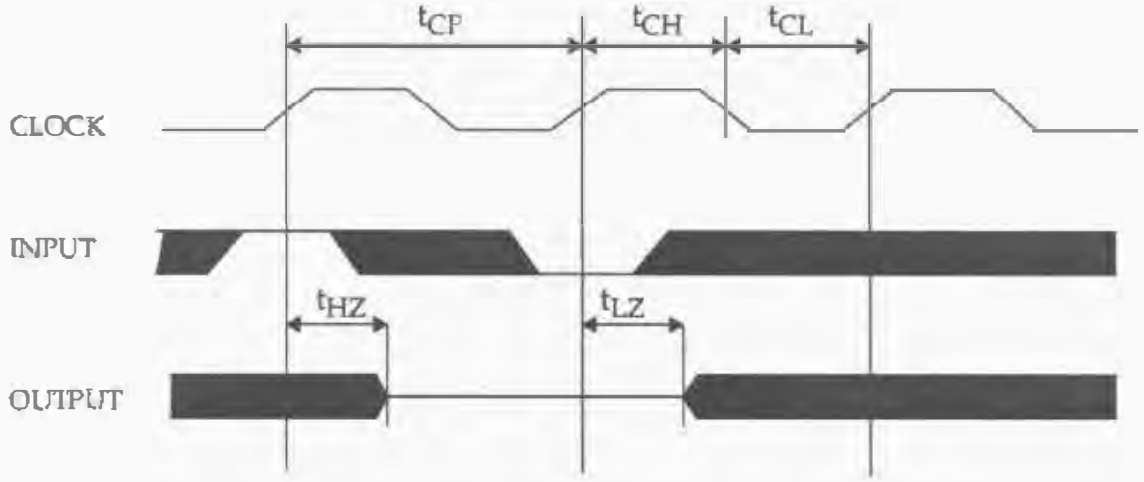


Figure 8-4 Tag RAM Clock and Tristate Timing Parameters

8.1.3 SYNCHRONOUS SRAM MODULE

SYMBOL	PARAMETER	RATING	UNIT
V_{CC}	Supply Voltage	-0.5 to +7.0	V
V_{IN}	Input Voltage	-0.5 to VCC+0.5	V
V_O	Output Voltage	-0.5 to VCC+0.5	V
P_D	Allowable Power Dissipation	10	W
T_{OPR}	Operating Temperature	0 to +70	°C
T_{STG}	Storage Temperature	-55 to +150	°C

Table 8-7 Cache RAM Absolute Maximum Ratings

SYMBOL	PARAMETER	MIN	MAX	UNIT
V_{CC}	Supply Voltage	4.75	5.25	V
V_{IH}	Input High Voltage	2.20	5.25	V
V_{IL}	Input Low Voltage	-0.5[1]	+.80	V
I_{LI}	Inpue Leakage Current	--	--	uA
I_{LO}	Output Leakage Current	--	--	uA
I_{CC}	Average Operating Current	--	--	mA
I_{SB}	Standby Current	--	--	mA
V_{OH}	Output High Signal Voltage[2]	2.4	--	V@4ma
V_{OL}	Output Low Signal Voltage[3]	--	0.4	V@8ma

Table 8-8 Cache RAM DC Electrical Characteristics

1. VIL = -1V Minimum for 3 nS per cycle
2. IOH = -4.0 mA
3. IOL = 8.0 mA

SYMBOL	PARAMETER	MIN	MAX	UNIT
tCHCH	Clock Cycle Time	12.5	--	nS
tCH	Clock high Pulse Width	4.0	--	nS
tCL	Clock Low Pulse Width	4.0	--	nS
tCHQV	Clock High to Data Valid	2.0	7.0	nS
tAVCH1	Address Setup to Clock High (AD15:AD0)	2.0	--	nS
tCHAX1	Address Hold From Clock High (AD15:AD0)	1.0	--	nS
tAVCH2	Address Setup to Clock High (DATSAX0, DATSAX1, DATSAY0, DATSAY1)	2.0	--	nS
tAVCH2	Address Hold from Clock High (DATSAX0, DATSAX1, DATSAY0, DATSAY1)	1.0	--	nS
tECHV	Module enable setup to clock high	2.0	--	nS
tCHEX	Module enable hold from clock high	1.0	--	nS
tWVCH	Write enable setup to clock high	2.0	--	nS
tCHWX	Write enable hold from clock high	1.0	--	nS
tGVCH	Output enable setup to clock high	2.0	--	nS
tCHGX	Output enable hold from clock high	1.0	--	nS
tDVCH	Input data setup to clock high	2.0	--	nS
tCHDX	Input data hold from clock high	1.0	--	nS
tCHQLZ	Clock high to output low-Z	2.0	--	nS
tCHQHZ	Clock high to output high-Z	2.0	--	nS

Table 8-9 Cache RAM AC Timing Characteristics

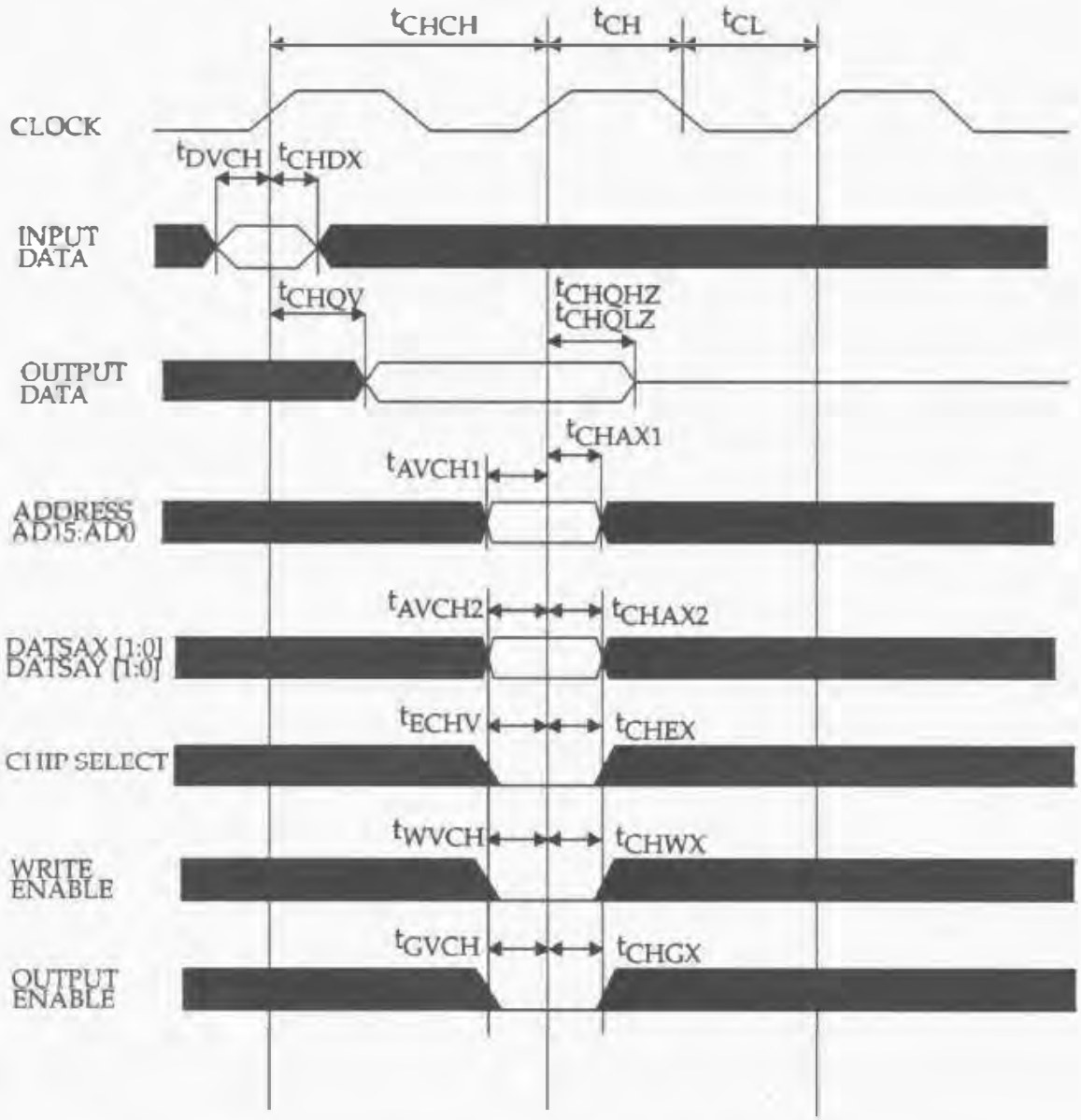


Figure 8-5 Cache RAM Setup, Hold, and Clock-to-Out Timing Parameters

8.2 MECHANICAL DATA

The following section contains the package pinout and device measurements for the R8000 Microprocessor, R8010 FPU, Tag RAM, and synchronous cache SRAM.

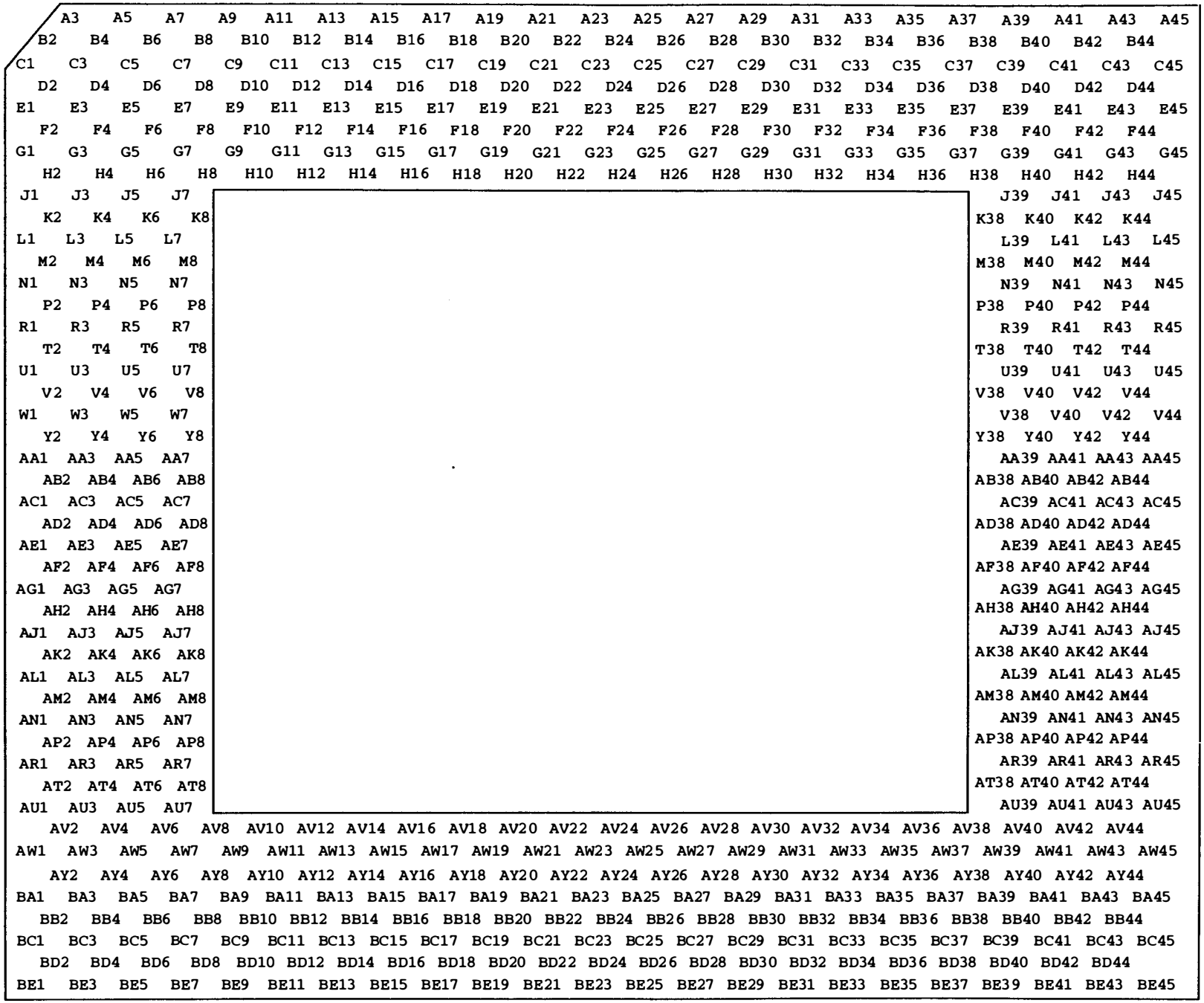


Figure 8-6 R8000 Microprocessor Package

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
A03.....Vcc	B26.....TBUS<46>	D4.....LDE<1>	E27.....Vss	G5.....LDO<3>	H28.....Vss
A05.....TBUS<0>	B28.....TBUS<50>	D6.....TBUS<3>	E29.....Vcc	G7.....LDO<2>	H30.....Vss
A07.....TBUS<6>	B30.....TBUS<54>	D8.....TBUS<7>	E31.....Vss	G9.....LDE<0>	H32.....Vss
A09.....TBUS<12>	B32.....TBUS<58>	D10.....TBUS<11>	E33.....TBUS<67>	G11.....Vcc	H34.....Vss
A11.....TBUS<16>	B34.....TBUS<62>	D12.....TBUS<15>	E35.....TBUS<71>	G13.....Vcc	H36.....NC
A13..... ..TBUS<20>	B36.....TBUS<66>	D14.....TBUS<21>	E37.....TBUS<75>	G15.....Vcc	H38.....Vss
A15.....TBUS<24>	B38.....TBUS<72>	D16.....TBUS<25>	E39.....TBUS<79>	G17.....Vcc	H40.....Vcc
A17... ....TBUS<28>	B40.....TBUS<78>	D18.....TBUS<29>	E41.....X_SDI	G19.....Vcc	H42.....JTAG_R
A19.....TBUS<32>	B42.....NC	D20.....TBUS<33>	E43.....SYNC_OUT	G21.....Vcc	H44.....JTAG_O
A21.....TBUS<36>	B44.....Vss	D22.....TBUS<37>	E45.....CLK	G23.....Vcc	J1.....LDE<5>
A23.....TBUS<40>	C1.....Vcc	D24.....TBUS<43>	F2.....LDE<34>	G25.....Vcc	J3.....LDE<36>
A25.....TBUS<44>	C3.....Vss	D26.....TBUS< 47>	F4.....LDO<34>	G27.....Vcc	J5.....LDO<5>
A27.....TBUS<48>	C5.....LDO<0>	D28.....TBUS<51>	F6.....LDO<33>	G29.....Vcc	J7.....LDO<4>
A29.....TBUS<52>	C7.....TBUS<4>	D30.....TBUS<55>	F8.....Vss	G31.....Vcc	J39.....VCC_PLL
A31.....TBUS<56>	C9.....TBUS<10>	D32.....TBUS<59>	F10.....Vcc	G33.....Vcc	J41.....JTAG_S
A33.....TBUS<60>	C11.....TBUS<17>	D34.....TBUS<65>	F12.....Vss	G35.....Vcc	J43.....FCCR
A35.....TBUS<64>	C13.....TBUS<19>	D36.....TBUS<69>	F14.....Vcc	G37.....FVALID#	J45.....ADDRE<17>
A37.....TBUS<68>	C15.....TBUS<23>	D38.....TBUS<73>	F16.....Vss	G39.....Vcc	K2.....LDE<37>
A39.....TBUS<74>	C17.....TBUS<27>	D40.....TBUS<77>	F18.....Vcc	G41.....SYNC_IN	K4.....LDO<38>
A41.....TBOE#	C19.....TBUS<31>	D42.....RESET#	F20.....Vss	G43.....JTAG_I	K6.....LDO<6>
A43.....Vcc	C21.....TBUS<35>	D44.....X_SDO	F22.....Vcc	G45.....JTAG_C	K8.....Vss
A45.....Vss	C23.....TBUS<39>	E1.....Vss	F24.....Vss	H2.....LDE<4>	K38.....GND_PLL
B2.....Vss	C25.....TBUS<45>	E3.....LDE<2>	F26.....Vcc	H4.....LDO<36>	K40.....Vss
B4.....LDO<32>	C27.....TBUS<49>	E5.....LDO<1>	F28.....Vss	H6.....LDO<35>	K42.....FCCL
B6.....TBUS<2>	C29.....TBUS<53>	E7.....TBUS<1>	F30.....Vcc	H8.....Vcc	K44.....ADDRE<16>
B8.....TBUS<8>	C31.....TBUS<57>	E9.....TBUS<5>	F32.....Vss	H10.....LDE<32>	L1.....LDE<38>
B10.....TBUS<14>	C33.....TBUS<61>	E11.....TBUS<9>	F34.....Vcc	H12.....Vss	L3.....LDE<6>
B12.....TBUS<18>	C35.....TBUS<63>	E13.....TBUS<13>	F36.....Vss	H14.....Vss	L5.....LDO<8>
B14.....TBUS<22>	C37.....TBUS<70>	E15.....Vss	F38.....Vcc	H16.....Vss	L7.....LDO<37>
B16.....TBUS<26>	C39.....TBUS<76>	E17.....Vss	F40.....Vss	H18.....Vss	L39.....Vcc
B18.....TBUS<30>	C41.....FPINTR#	E19.....Vcc	F42.....EXT_CLK	H20.....Vss	L41....ADDRO<17>.
B20.....TBUS<34>	C43.....Vss	E21.....Vss	F44.....LPF_OUT	H22.....Vss	L43.....ADDRO<16>
B22.....TBUS<38>	C45.....Vcc	E23.....TBUS<41>	G1.....LDE<35>	H24.....Vss	L45.....ADDRE<15>
B24.....TBUS<42>	D2.....LDE<33>	E25.....Vss	G3.....LDE<3>	H26.....Vss	M2.....LDE<7>

Table 8-10 R8000 Microprocessor Pinout

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
M4.....LDO<41>	T38.....Vss	Y44.....ADDRE<6>	AE5.....LDO<48>	AJ41.....Vcc	AP2.....LDE<55>
M6.....LDO<39>	T40.....Vcc	AA1.....LDE<14>	AE7.....Vss	AJ43.....ESAO<0>	AP4.....LDO<23>
M8.....Vcc	T42.....ADDRO<10>	AA3.....LDE<45>	AE39.....Vcc	AJ45.....ESASELE	AP6.....LDO<55>
M38.....Vss	T44.....ADDRE<10>	AA5.....LDO<14>	AE41.....Vss	AK2.....LDE<52>	AP8.....Vss
M40.....Vcc	U1.....LDE<11>	AA7.....Vss	AE43....ADDRO<0>	AK4.....LDO<50>	AP38.....Vss
M42....ADDRO<15>	U3.....LDE<42>	AA39.....Vcc	AE45....ADDRE<1>	AK6.....LDO<21>	AP40.....Vss
M44....ADDRE<14>	U5.....LDO<43>	AA41.....Vss	AF2.....LDE<49>	AK8.....Vss	AP42.....DEQSE#
N1.....LDE<8>	U7.....Vss	AA43...ADDRO<5>	AF4.....LDO<17>	AK38.....Vss	AP44.....ENQLE#
N3.....LDE<39>	U39.....Vcc	AA45....ADDRE<5>	AF6.....Vcc	AK40.....Vss	AR1.....LDE<24>
N5.....LDO<9>	U41.....Vss	AB2.....LDE<46>	AF8.....Vss	AK42.....ESAO<1>	AR3.....LDE<56>
N7.....LDO<7>	U43.....ADDRO<9>	AB4.....LDO<15>	AF38.....Vss	AK44.....ESAE<0>	AR5.....LDO<56>
N39.....Vcc	U45.....ADDRE<9>	AB6.....LDO<46>	AF40.....Vss	AL1.....LDE<21>	AR7.....LDO<57>
N41....ADDRO<14>	V2.....LDE<43>	AB8.....Vss	AF42.....WEO<1>	AL3.....LDE<53>	AR39.....Vcc
N43....ADDRO<13>	V4.....LDO<13>	AB38.....Vss	AF44....ADDRE<0>	AL5.....LDO<52>	AR41.....MCHSAO0
N45....ADDRE<13>	V6.....Vcc	AB40.....Vss	AG1.....LDE<18>	AL7.....LDO<54>	AR43.....CCREQ
P2.....LDE<40>	V8.....Vss	AB42....ADDRO<4>	AG3.....LDE<50>	AL39.....Vcc	AR45.....ENQLP#
P4.....LDO<11>	V38.....Vss	AB44....ADDRE<4>	AG5.....LDO<18>	AL41.....Vss	AT2.....LDE<25>
P6.....LDO<40>	V40.....Vss	AC1.....LDE<15>	AG7.....Vss	AL43.....IUREL#	AT4.....LDO<25>
P8.....Vss	V42.....ADDRO<8>	AC3.....LDE<47>	AG39.....Vcc	AL45.....ESAE<1>	AT6.....LDO<26>
P38.....Vss	V44.....ADDRE<8>	AC5.....LDO<47>	AG41.....Vss	AM2.....LDE<22>	AT8.....Vcc
P40.....Vss	W1.....LDE<44>	AC7.....Vss	AG43.....WEO<0>	AM4.....LDO<20>	AT38.....MCHSTO1
P42.....ADDRO<12>	W3.....LDE<12>	AC39.....Vcc	AG45.....WEE<1>	AM6.....LDO<22>	AT40.....Vcc
P44.....ADDRE<12>	W5.....LDO<44>	AC41....ADDRO<3>	AH2.....LDE<19>	AM8.....Vcc	AT42.....MCHSAO1
R1.....LDE<41>	W7.....Vss	AC43....ADDRO<2>	AH4.....LDO<49>	AM38.....Vss	AT44.....DEQSO#
R3.....LDE<9>	W39.....Vcc	AC45....ADDRE<3>	AH6.....LDO<51>	AM40.....Vcc	AU1.....LDE<57>
R5.....LDO<42>	W41.....Vcc	AD2.....LDE<16>	AH8.....Vcc	AM42.....UPDE#	AU3.....LDE<26>
R7.....Vss	W43.....ADDRO<7>	AD4.....LDO<16>	AH38.....Vss	AM44.VALIDOUT#	AU5.....LDO<58>
R39.....Vcc	W45.....ADDRE<7>	AD6.....Vcc	AH40.....Vcc	AN1.....LDE<54>	AU7.....LDO<59>
R41.....Vss	Y2.....LDE<13>	AD8.....Vcc	AH42.....ESASELO	AN3.....LDE<23>	AU39.....MCHVS0
R43.....ADDRO<11>	Y4.....LDO<45>	AD38.....Vss	AH44.....WEE<0>	AN5.....LDO<53>	AU41.....MATCHO#
R45.....ADDRE<11>	Y6.....Vcc	AD40.....Vcc	AJ1.....LDE<51>	AN7.....LDO<24>	AU43.....MCHSAE0
T2.....LDE<10>	Y8.....Vcc	AD42...ADDRO<1>	AJ3.....LDE<20>	AN39.....Vcc	AU45.....INTMODE
T4.....LDO<12>	Y38.....Vss	AD44...ADDRE<2>	AJ5.....LDO<19>	AN41.....ENQLO#	AV2.....LDE<58>
T6.....LDO<10>	Y40.....Vcc	AE1.....LDE<48>	AJ7.....Vss	AN43.....DEBUG#	AV4.....LDO<27>
T8.....Vcc	Y42.....ADDRO<6>	AE3.....LDE<17>	AJ39.....Vcc	AN45.....UPDO#	AV6.....LDO<28>

Table 8-10 R8000 Microprocessor Pinout

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
AV8.....Vss	AW33.....Vcc	BA13.....TAGE<17>	BB38...INDEXE<4>	BD18.....TAGO<9>	BE43.....Vcc
AV10.....LDE<31>	AW35.....Vcc	BA15.....Vss	BB40..INDEXE<2>	BD20.....TAGO<7>	BE45.....Vss
AV12.....,Vss	AW37..INDEXE<0>	BA17.....Vcc	BB42....MCHVSE3	BD22.....TAGO<5>	
AV14.....Vss	AW39.....Vcc	BA19.....Vss	BB44....MCHVSE2	BD24.....TAGO<3>	
AV16.....Vss	AW41....MCHVSO1	BA21.....Vss	BC1.....Vcc	BD26.....TAGO<1>	
AV18.....Vss	AW43.....MCHSTE0	BA23.....TAGE<3>	BC3.....Vss	BD28INDEXO<16>	
AV20.....Vss	AW45.....MATCH#	BA25.....Vss	BC5.....LDO<31>	BD30INDEXO<14>	
AV22.....Vss	AY2.....LDE<28>	BA27.....Vcc	BC7.....PMODE	BD32INDEXO<12>	
AV24.....Vss	AY4.....LDE<29>	BA29.....Vss	BC9.....TAGO<19>	BD34INDEXO<10>	
AV26.....Vss	AY6.....LDO<61>	BA31.....Vss	BC11.....TAGE<15>	BD36..INDEXO<8>	
AV28.....Vss	AY8.....Vcc	BA33..INDEXE<7>	BC13.....TAGE<14>	BD38..INDEXO<5>	
AV30.....Vss	AY10.....Vss	BA35..INDEXE<5>	BC15.....TAGE<12>	BD40..INDEXO<2>	
AV32.....Vss	AY12.....Vcc	BA37..INDEXE<3>	BC17.....TAGE<10>	BD42.....DBSETO#	
AV34.....Vss	AY14.....Vss	BA39..INDEXE<1>	BC19.....TAGE<8>	BD44.....Vss	
AV36.....BDSETE#	AY16.....Vcc	BA41.....MCHVSO3	BC21.....TAGE<6>	BE1.....Vss	
AV38.....Vss	AY18.....Vss	BA43.....MCHVSE1	BC23.....TAGE<4>	BE3.....Vcc	
AV40.....Vss	AY20.....Vcc	BA45.....MCHVSE0	BC25.....TAGE<1>	BE5.....LDO<63>	
AV42.....MCHSTO0	AY22.....Vss	BB2.....LDE<30>	BC27.INDEXE<16>	BE7.....TAGO<21>	
AV44.....MCHSAE1	AY24.....Vcc	BB4.....LDE<62>	BC29.INDEXE<14>	BE9.....TAGO<18>	
AW1.....LDE<27>	AY26.....Vss	BB6.....PERRE	BC31.INDEXE<12>	BE11.....TAGO<16>	
AW3.....LDE<59>	AY28.....Vcc	BB8.....TAGE<20>	BC33.INDEXE<10>	BE13.....TAGO<14>	
AW5.....LDO<60>	AY30.....Vss	BB10.....TAGE<18>	BC35...INDEXE<9>	BE15....TAGO<12>	
AW7.....LDO<29>	AY32.....Vcc	BB12.....TAGE<16>	BC37...INDEXO<6>	BE17.....TAGO<10>	
AW9.....LDE<63>	AY34.....Vss	BB14.....TAGE<13>	BC39...INDEXO<3>	BE19.....TAGO<8>	
AW11.....Vcc	AY36.....Vcc	BB16.....TAGE<11>	BC41...INDEXO<0>	BE21.....TAGO<6>	
AW13.....Vcc	AY38.....Vss	BB18.....TAGE<9>	BC43.....Vss	BE23.....TAGO<4>	
AW15.....Vcc	AY40.....Vcc	BB20.....TAGE<7>	BC45.....Vcc	BE25.....TAG0<2>	
AW17.....Vcc	AY42.....MCHVSO2	BB22.....TAGE<5>	BD2.....Vss	BE27.....TAGO<0>	
AW19.....Vss	AY44.....MCHSTE1	BB24.....TAGE<2>	BD4.....LDO<62>	BE29.INDEXO<15>	
AW21.....Vcc	BA1.....LDE<60>	BB26.....TAGE<0>	BD6.....PERRO	BE31.INDEXO<13>	
AW23.....Vcc	BA3.....LDE<61>	BB28.INDEXE<15>	BD8.....TAGO<20>	BE33.INDEXO<11>	
AW25.....Vcc	BA5.....LDO<30>	BB30.INDEXE<13>	BD10....TAGO<17>	BE35..INDEXO<9>	
AW27.....Vcc	BA7.....NC	BB32.INDEXE<11>	BD12....TAGO<15>	BE37..INDEXO<7>	
AW29.....Vcc	BA9.....TAGE<21>	BB34...INDEXE<8>	BD14.....TAGO<13>	BE39...INDEXO<4>	
AW31.....Vcc	BA11.....TAGE<19>	BB36...INDEXE<6>	BD16.....TAGO<11>	BE41....INDEX0<1>	

Table 8-10 R8000 Microprocessor Pinout

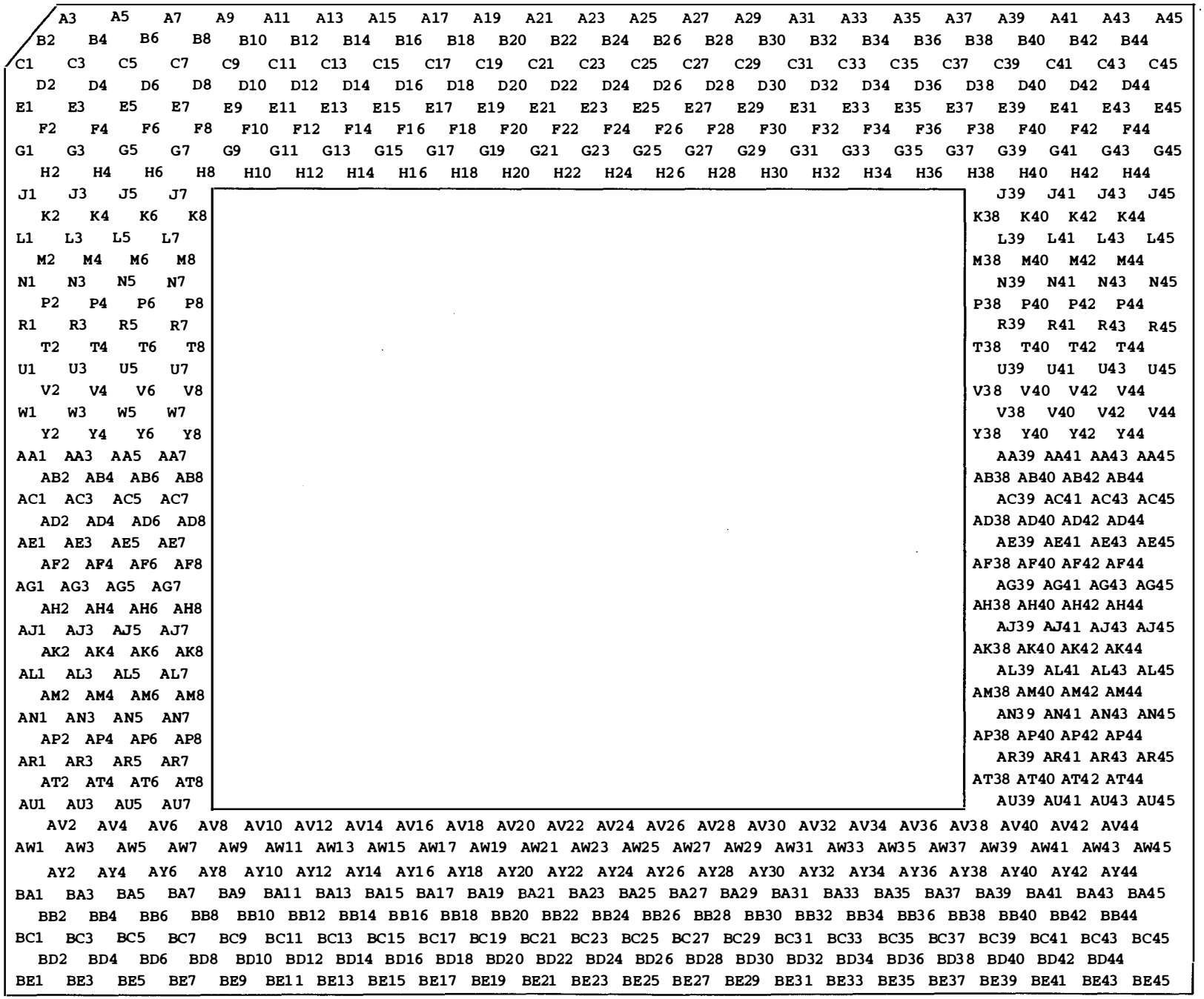


Figure 8-7 R8010 Floating Point Unit Package

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
A03.....Vcc	B26.....TBUS<46>	D4.....LDE<1>	E27.....Vss	G5.....LDO<3>	H28.....Vss
A05.....TBUS<0>	B28.....TBUS<50>	D6.....TBUS<3>	E29.....Vcc	G7.....LDO<2>	H30.....Vss
A07.....TBUS<6>	B30.....TBUS<54>	D8.....TBUS<7>	E31.....Vss	G9.....LDE<0>	H32.....Vss
A09.....TBUS<12>	B32.....TBUS<58>	D10.....TBUS<11>	E33.....TBUS<67>	G11.....Vcc	H34.....Vss
A11.....TBUS<16>	B34.....TBUS<62>	D12.....TBUS<15>	E35.....TBUS<71>	G13.....Vcc	H36.....NC
A13.....TBUS<20>	B36.....TBUS<66>	D14.....TBUS<21>	E37.....TBUS<75>	G15.....Vcc	H38.....Vss
A15.....TBUS<24>	B38.....TBUS<72>	D16.....TBUS<25>	E39.....TBUS<79>	G17.....Vcc	H40.....Vcc
A17.....TBUS<28>	B40.....TBUS<78>	D18.....TBUS<29>	E41.....X_SDI	G19.....Vcc	H42.....JTAG_R
A19.....TBUS<32>	B42.....NC	D20.....TBUS<33>	E43.....SYNC_OUT	G21.....Vcc	H44.....JTAG_O
A21.....TBUS<36>	B44.....Vss	D22.....TBUS<37>	E45.....CLK	G23.....Vcc	J1.....LDE<5>
A23.....TBUS<40>	C1.....Vcc	D24.....TBUS<43>	F2.....LDE<34>	G25.....Vcc	J3.....LDE<36>
A25.....TBUS<44>	C3.....Vss	D26.....TBUS< 47>	F4.....LDO<34>	G27.....Vcc	J5.....LDO<5>
A27.....TBUS<48>	C5.....LDO<0>	D28.....TBUS<51>	F6.....LDO<33>	G29.....Vcc	J7.....LDO<4>
A29.....TBUS<52>	C7.....TBUS<4>	D30.....TBUS<55>	F8.....Vss	G31.....Vcc	J39.....VCC_PLL
A31.....TBUS<56>	C9.....TBUS<10>	D32.....TBUS<59>	F10.....Vcc	G33.....Vcc	J41.....JTAG_S
A33.....TBUS<60>	C11.....TBUS<17>	D34.....TBUS<65>	F12.....Vss	G35.....Vcc	J43.....DEBUG_
A35.....TBUS<64>	C13.....TBUS<19>	D36.....TBUS<69>	F14.....Vcc	G37.....FVALID#	J45.....SDO<0>
A37.....TBUS<68>	C15.....TBUS<23>	D38.....TBUS<73>	F16.....Vss	G39.....Vcc	K2.....LDE<37>
A39.....TBUS<74>	C17.....TBUS<27>	D40.....TBUS<77>	F18.....Vcc	G41.....SYNC_IN	K4.....LDO<38>
A41.....TBOE#	C19.....TBUS<31>	D42.....RESET#	F20.....Vss	G43.....JTAG_I	K6.....LDO<6>
A43.....Vcc	C21.....TBUS<35>	D44.....X_SDO	F22.....Vcc	G45.....JTAG_C	K8.....Vss
A45.....Vss	C23.....TBUS<39>	E1.....Vss	F24.....Vss	H2.....LDE<4>	K38.....GND_PLL
B2.....Vss	C25.....TBUS<45>	E3.....LDE<2>	F26.....Vcc	H4.....LDO<36>	K40.....Vss
B4.....LDO<32>	C27.....TBUS<49>	E5.....LDO<1>	F28.....Vss	H6.....LDO<35>	K42.....NC
B6.....TBUS<2>	C29.....TBUS<53>	E7.....TBUS<1>	F30.....Vcc	H8.....Vcc	K44.....SDO<32>
B8.....TBUS<8>	C31.....TBUS<57>	E9.....TBUS<5>	F32.....Vss	H10.....LDE<32>	L1.....LDE<38>
B10.....TBUS<14>	C33.....TBUS<61>	E11.....TBUS<9>	F34.....Vcc	H12.....Vss	L3.....LDE<6>
B12.....TBUS<18>	C35.....TBUS<63>	E13.....TBUS<13>	F36.....Vss	H14.....Vss	L5.....LDO<8>
B14.....TBUS<22>	C37.....TBUS<70>	E15.....Vss	F38.....Vcc	H16.....Vss	L7.....LDO<37>
B16.....TBUS<26>	C39.....TBUS<76>	E17.....Vss	F40.....Vss	H18.....Vss	L39.....Vcc
B18.....TBUS<30>	C41.....NC	E19.....Vcc	F42.....EXT_CLK	H20.....Vss	L41.....SDE<0>
B20.....TBUS<34>	C43.....Vss	E21.....Vss	F44.....LPF_OUT	H22.....Vss	L43.....SDE<32>
B22.....TBUS<38>	C45.....Vcc	E23.....TBUS<41>	G1.....LDE<35>	H24.....Vss	L45.....SDO<1>
B24.....TBUS<42>	D2.....LDE<33>	E25.....Vss	G3.....LDE<3>	H26.....Vss	M2.....LDE<7>

Table 8-11 R8010 Floating Point Unit Pinout

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
M4.....LDO<41>	T38.....Vss	Y44.....SDO<5>	AE5.....LDO<48>	AJ41.....Vcc	AP2.....LDE<55>
M6.....LDO<39>	T40.....Vcc	AA1.....LDE<14>	AE7.....Vss	AJ43.....SDE<40>	AP4.....LDO<23>
M8.....Vcc	T42.....SDE<35>	AA3.....LDE<45>	AE39.....Vcc	AJ45.....SDO<8>	AP6.....LDO<55>
M38.....Vss	T44.....SDO<35>	AA5.....LDO<14>	AE41.....Vss	AK2.....LDE<52>	AP8.....Vss
M40.....Vcc	U1.....LDE<11>	AA7.....Vss	AE43.....SDE<7>	AK4.....LDO<50>	AP38.....Vss
M42.....SDE<1>	U3.....LDE<42>	AA39.....Vcc	AE45.....SPO<1>	AK6.....LDO<21>	AP40.....Vss
M44.....SDO<33>	U5.....LDO<43>	AA41.....Vss	AF2.....LDE<49>	AK8.....Vss	AP42.....SDE<43>
N1.....LDE<8>	U7.....Vss	AA43.....SDE<37>	AF4.....LDO<17>	AK38.....Vss	AP44.....SDO<42>
N3.....LDE<39>	U39.....Vcc	AA45.....SPE<2>	AF6.....Vcc	AK40.....Vss	AR1.....LDE<24>
N5.....LDO<9>	U41.....Vss	AB2.....LDE<46>	AF8.....Vss	AK42.....SDE<9>	AR3.....LDE<56>
N7.....LDO<7>	U43.....SDE<4>	AB4.....LDO<15>	AF38.....Vss	AK44.....SDO<40->	AR5.....LDO<56>
N39.....Vcc	U45.....SDO<4>	AB6.....LDO<46>	AF40.....Vss	AL1.....LDE<21>	AR7.....LDO<57>
N41.....SDE<33>	V2.....LDE<43>	AB8.....Vss	AF42.....SDE<39>	AL3.....LDE<53>	AR39.....Vcc
N43.....SDE<2>	V4.....LDO<13>	AB38.....Vss	AF44.....SDO<7>	AL5.....LDO<52>	AR41.....SDE<44>
N45.....SDO<2>	V6.....Vcc	AB40.....Vss	AG1.....LDE<18>	AL7.....LDO<54>	AR43.....SDE<12>
P2.....LDE<40>	V8.....Vss	AB42.....SPE<3>	AG3.....LDE<50>	AL39.....Vcc	AR45.....SDO<11>
P4.....LDO<11>	V38.....Vss	AB44.....SDO<37>	AG5.....LDO<18>	AL41.....Vss	AT2.....LDE<25>
P6.....LDO<40>	V40.....Vss	AC1.....LDE<15>	AG7.....Vss	AL43.....SDE<41>	AT4.....LDO<25>
P8.....Vss	V42.....SDE<36>	AC3.....LDE<47>	AG39.....Vcc	AL45.....SDO<9>	AT6.....LDO<26>
P38.....Vss	V44.....SDO<36>	AC5.....LDO<47>	AG41.....Vss	AM2.....LDE<22>	AT8.....Vcc
P40.....Vss	W1.....LDE<44>	AC7.....Vss	AG43.....SPO<3>	AM4.....LDO<20>	AT38.....BYPASS#
P42.....SDE<34>	W3.....LDE<12>	AC39.....Vcc	AG45.....SPO<2>	AM6.....LDO<22>	AT40.....Vcc
P44.....SDO<34>	W5.....LDO<44>	AC41.....SDE<6>	AH2.....LDE<19>	AM8.....Vcc	AT42.....SDE<13>
R1.....LDE<41>	W7.....Vss	AC43.....SDE<38>	AH4.....LDO<49>	AM38.....Vss	AT44.....SDO<43>
R3.....LDE<9>	W39.....Vcc	AC45.....SDO<6>	AH6.....LDO<51>	AM40.....Vcc	AU1.....LDE<57>
R5.....LDO<42>	W41.....Vcc	AD2.....LDE<16>	AH8.....Vcc	AM42.....SDE<10>	AU3.....LDE<26>
R7.....Vss	W43.....SPE<0>	AD4.....LDO<16>	AH38.....Vss	AM44.....SDO<41>	AU5.....LDO<58>
R39.....Vcc	W45.....SPE<1>	AD6.....Vcc	AH40.....Vcc	AN1.....LDE<54>	AU7.....LDO<59>
R41.....Vss	Y2.....LDE<13>	AD8.....Vcc	AH42.....SDE<8>	AN3.....LDE<23>	AU39.....LPO<3>
R43.....SDE<3>	Y4.....LDO<45>	AD38.....Vss	AH44.....SDO<39>	AN5.....LDO<53>	AU41.....PMODE
R45.....SDO<3>	Y6.....Vcc	AD40.....Vcc	AJ1.....LDE<51>	AN7.....LDO<24>	AU43.....SDO<44>
T2.....LDE<10>	Y8.....Vcc	AD42.....SPO<0>	AJ3.....LDE<20>	AN39.....Vcc	AU45.....SDO<12>
T4.....LDO<12>	Y38.....Vss	AD44.....SDO<38>	AJ5.....LDO<19>	AN41.....SDE<11>	AV2.....LDE<58>
T6.....LDO<10>	Y40.....Vcc	AE1.....LDE<48>	AJ7.....Vss	AN43.....SDE<42>	AV4.....LDO<27>
T8.....Vcc	Y42.....SDE<5>	AE3.....LDE<17>	AJ39.....Vcc	AN45.....SDO<10>	AV6.....LDO<28>

Table 8-11 R8010 Floating Point Unit Pinout

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
AV8.....Vss	AW33.....Vcc	BA13..... ..SDE<62>	BB38.....SDE<47>	BD18.....SDO<26>	BE43.....Vcc
AV10.....LDE<31>	AW35.....Vcc	BA15.....Vss	BB40.....SDE<46>	BD20.....SDO<25>	BE45.....Vss
AV12.....Vss	AW37.....SDE<5>	BA17.....Vcc	BB42.....LPE<0>	BD22.....SDO<24>	
AV14.....Vss	AW39.....Vcc	BA19.....Vss	BB44.....LPE<1>	BD24.....SDO<23>	
AV16.....Vss	AW41.....LPO<2>	BA21.....Vss	BC1.....Vcc	BD26.....SDO<22>	
AV18.....Vss	AW43.....DEQSO#	BA23.....SDE<55>	BC3.....Vss	BD28.....SDO<21>	
AV20.....Vss	AW45.....NC	BA25.....Vss	BC5.....LDO<31>	BD30.....SDO<20>	
AV22.....Vss	AY2.....LDE<28>	BA27.....Vcc	BC7.....ENQLP	BD32.....SDO<19>	
AV24.....Vss	AY4.....LDE<29>	BA29.....Vss	BC9.....SDO<31>	BD34.....SDO<18>	
AV26.....Vss	AY6.....LDO<61>	BA31.....Vss	BC11.....SDE<61>	BD36.....SDO<17>	
AV28.....Vss	AY8.....Vcc	BA33.....SDE<17>	BC13.....SDE<29>	BD38.....SDO<47>	
AV30.....Vss	AY10.....Vss	BA35.....SDE<16>	BC15.....SDE<28>	BD40.....SDO<14>	
AV32.....Vss	AY12.....Vcc	BA37.....SDE<15>	BC17.....SDE<27>	BD42.....FCCL	
AV34.....Vss	AY14.....Vss	BA39.....SDE<14>	BC19.....SDE<26>	BD44.....Vss	
AV36.....FCCR	AY16.....Vcc	BA41.....LPO<0>	BC21.....SDE<25>	BE1.....Vss	
AV38.....Vss	AY18.....Vss	BA43.....LPE<2>	BC23.....SDE<24>	BE3.....Vcc	
AV40.....Vss	AY20.....Vcc	BA45.....LPE<3>	BC25.....SDE<54>	BE5.....LDO<63>	
AV42.....DEQSE#	AY22.....Vss	BB2.....LDE<30>	BC27.....SDE<53>	BE7.....PERRE	
AV44.....SDO<13>	AY24.....Vcc	BB4.....LDE<62>	BC29.....SDE<52>	BE9.....SDO<62>	
AW1.....LDE<27>	AY26.....Vss	BB6.....ENQLE#	BC31.....SDE<51>	BE11.....SDO<61>	
AW3.....LDE<59>	AY28.....Vcc	BB8.....PERRO	BC33.....SDE<50>	BE13.....SDO<60>	
AW5.....LDO<60>	AY30.....Vss	BB10.....SDE<31>	BC35.....SDE<18>	BE15.....SDO<59>	
AW7.....LDO<29>	AY32.....Vcc	BB12.....SDE<30>	BC37.....SDO<16>	BE17.....SDO<58>	
AW9.....LDE<63>	AY34.....Vss	BB14.....SDE<60>	BC39.....SDO<46>	BE19.....SDO<57>	
AW11.....Vcc	AY36.....Vcc	BB16.....SDE<59>	BC41.....FPINTR#	BE21.....SDO<56>	
AW13.....Vcc	AY38.....Vss	BB18.....SDE<58>	BC43.....Vss	BE23.....SDO<55>	
AW15.....Vcc	AY40.....Vcc	BB20.....SDE<57>	BC45.....Vcc	BE25.....SDO<54>	
AW17.....Vcc	AY42.....LPO<1>	BB22.....SDE<56>	BD2.....Vss	BE27.....SDO<53>	
AW19.....Vcc	AY44.....FOE#	BB24.....SDE<23>	BD4.....LDO<62>	BE29.....SDO<52>	
AW21.....Vcc	BA1.....LDE<60>	BB26.....SDE<22>	BD6.....ENQLO#	BE31.....SDO<51>	
AW23.....Vcc	BA3.....LDE<61>	BB28.....SDE<21>	BD8.....SDO<63>	BE33.....SDO<50>	
AW25.....Vcc	BA5.....LDO<30>	BB30.....SDE<20>	BD10.....SDO<30>	BE35.....SDO<49>	
AW27.....Vcc	BA7.....NC	BB32.....SDE<19>	BD12.....SDO<29>	BE37.....SDO<48>	
AW29.....Vcc	BA9.....NC	BB34.....SDE<49>	BD14..... SDO<28>	BE39.....SDO<15>	
AW31.....Vcc	BA11.....SDE<63>	BB36.....SDE<48>	BD16.....SDO<27>	BE41.....SDO<45>	

Table 8-11 R8010 Floating Point Unit Pinout

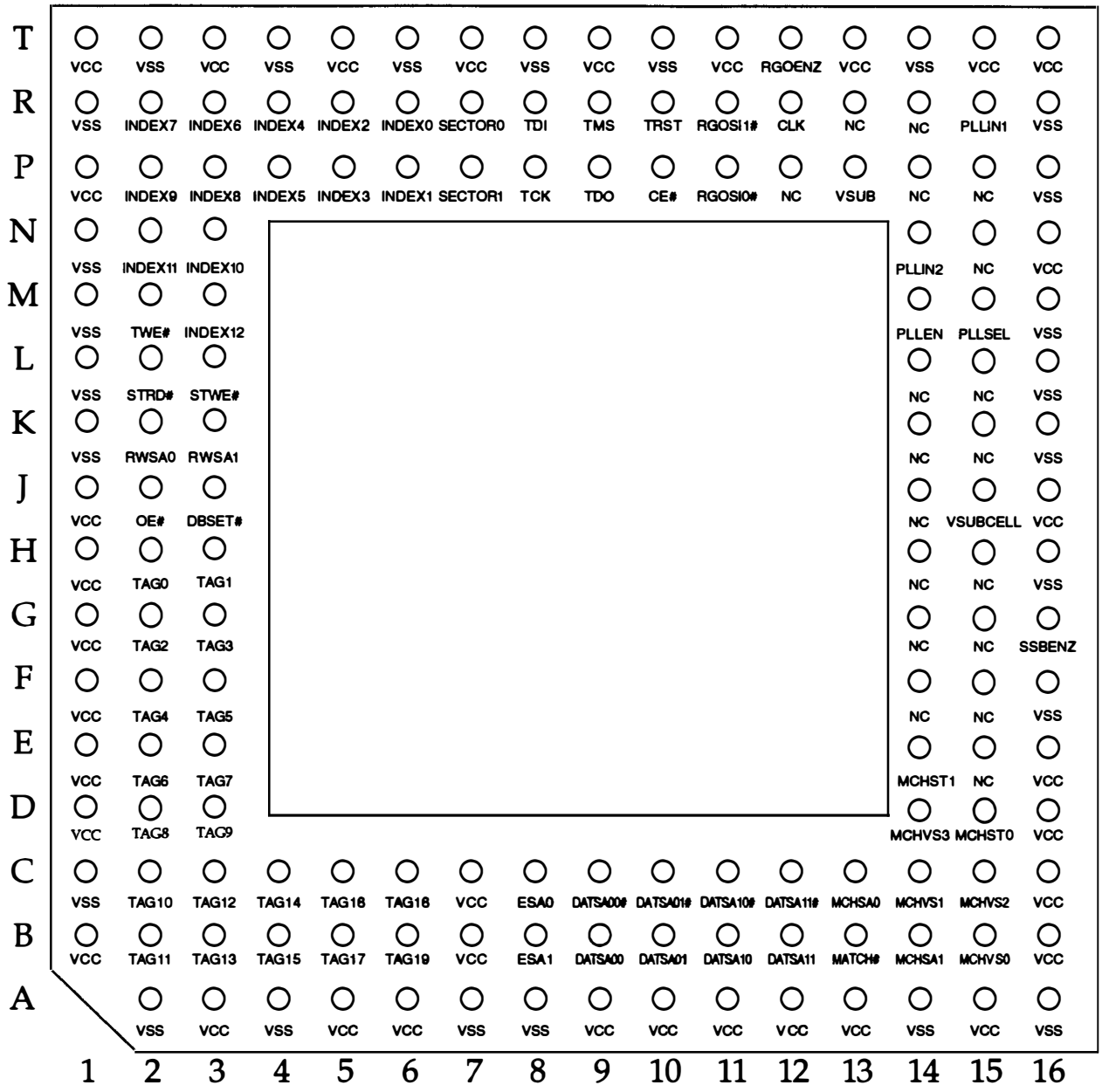


Figure 8-8 Tag RAM Package

Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal	Location .. .Signal
A2.....Vss	B12.....DATSAY<1>	D16.....Vcc	J2.....OE#	N14.....PLLIN2	R8.....TDI
A3.....Vcc	B13.....MATCH#	E1.....Vcc	J3.....DBSET#	N15.....NC	R9.....TMS
A4.....Vss	B14.....MCHSA<1>	E2.....TAG<6>	J14.....NC	N16.....Vcc	R10.....TRST
A5.....Vcc	B15.....MCHVS<0>	E3.....TAG<7>	J15.....VSUBCELL	P1.....Vcc	R11.....RGOSI1
A6.....Vcc	B16.....Vcc	E14.....MCHST<1>	J16.....Vcc	P2.....INDEX<9>	R12.....CLK
A7.....Vss	C1.....Vss	E15.....NC	K1.....Vss	P3.....INDEX<8>	R13.....NC
A8.....Vss	C2.....TAG<10>	E16.....Vcc	K2.....RWSA<1>	P4.....INDEX<5>	R14.....Vcc
A9.....Vcc	C3.....TAG<12>	F1.....Vcc	K3.....RWSA<0>	P5.....INDEX<3>	R15.....PLLIN1
A10.....Vcc	C4.....TAG<14>	F2.....TAG<4>	K14.....NC	P6.....INDEX<1>	R16.....Vss
A11.....Vcc	C5.....TAG<16>	F3.....TAG<5>	K15.....NC	P7.....SECTOR<1>	T1.....Vcc
A12.....Vcc	C6.....TAG<18>	F14.....NC	K16.....Vss	P8.....TCK	T2.....Vss
A13.....Vcc	C7.....Vss	F15.....NC	L1.....Vss	P9.....TDO	T3.....Vcc
A14.....Vss	C8.....ESA<0>	F16.....Vss	L2.....STRD#	P10.....CE#	T4.....Vss
A15.....Vcc	C9....DATSAW#<0>	G1.....Vcc	L3.....STWE#	P11.....RGOSI0	T5.....Vss
A16.....Vss	C10..DATSAW#<1>	G2.....TAG<2>	L14.....NC	P12.....NC	T6.....Vcc
B1.....Vcc	C11...DATSAZ#<0>	G3.....TAG<3>	L15.....NC	P13.....VSUB	T7.....Vss
B2.....TAG<11>	C12...DATSAZ#<1>	G14.....NC	L16.....Vss	P14.....NC	T8.....Vcc
B3.....TAG<13>	C13.....MCHSA<0>	G15.....NC	M1.....Vss	P15.....NC	T9.....Vss
B4.....TAG<15>	C14.....MCHVS<1>	G16.....SSBENZ	M2.....TWE#	P16.....Vss	T10.....Vss
B5.....TAG<17>	C15.....MCHVS<2>	H1.....Vss	M3.....INDEX<12>	R1.....Vss	T11.....Vcc
B6.....TAG<19>	C16.....Vcc	H2.....TAG<0>	M14.....PLLEN	R2.....INDEX<7>	T12.....RGOENZ
B7.....ESASEL	D1.....Vcc	H3.....TAG<1>	M15.....PLLSEL	R3.....INDEX<6>	T13.....Vcc
B8.....ESA<1>	D2.....TAG<8>	H14.....NC	M16.....Vss	R4.....INDEX<4>	T14.....Vss
B9.....DATSAX<0>	D3.....TAG<9>	H15.....NC	N1.....Vss	R5.....INDEX<2>	T15.....Vcc
B10.....DATSAX<1>	D14.....MCHVS<3>	H16.....Vss	N2.....INDEX<11>	R6.....INDEX<0>	T16.....Vcc
B11.....DATSAY<0>	D15.....MCHST<0>	J1.....Vcc	N3.....INDEX<10>	R7.....SECTOR<0>	

Table 8-12 Tag RAM Unit Pinout

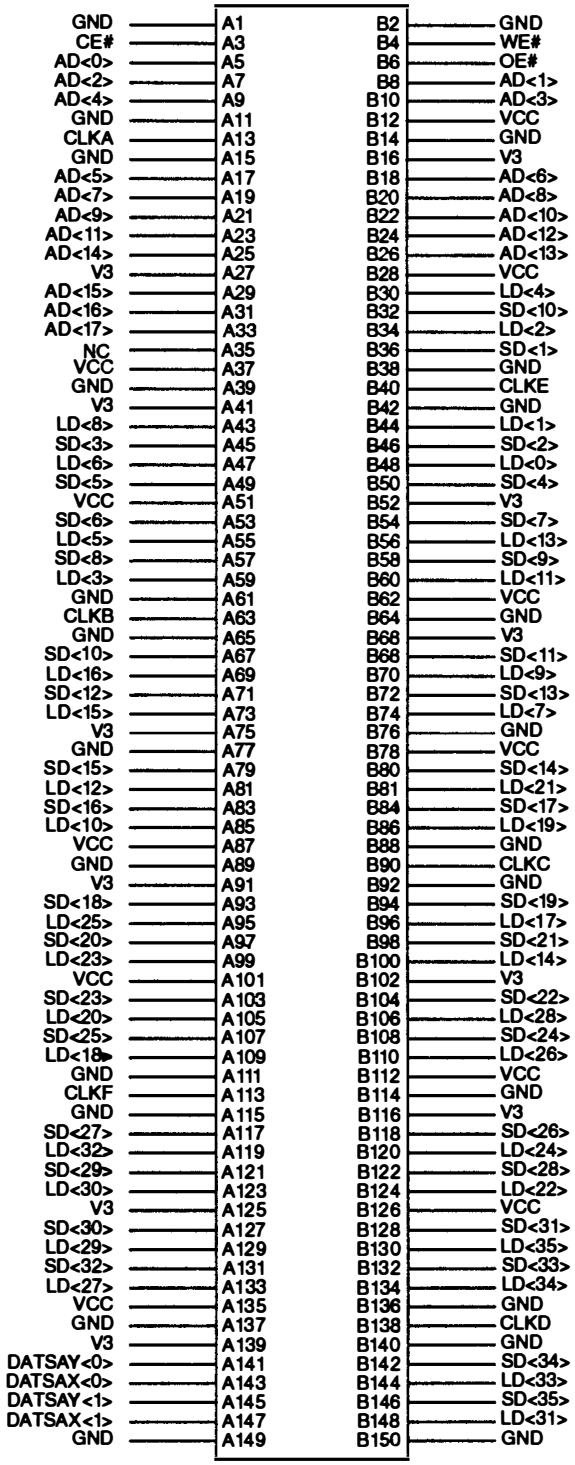


Figure 8-9 Streaming Cache SIM Module

* All measurements in millimeters.

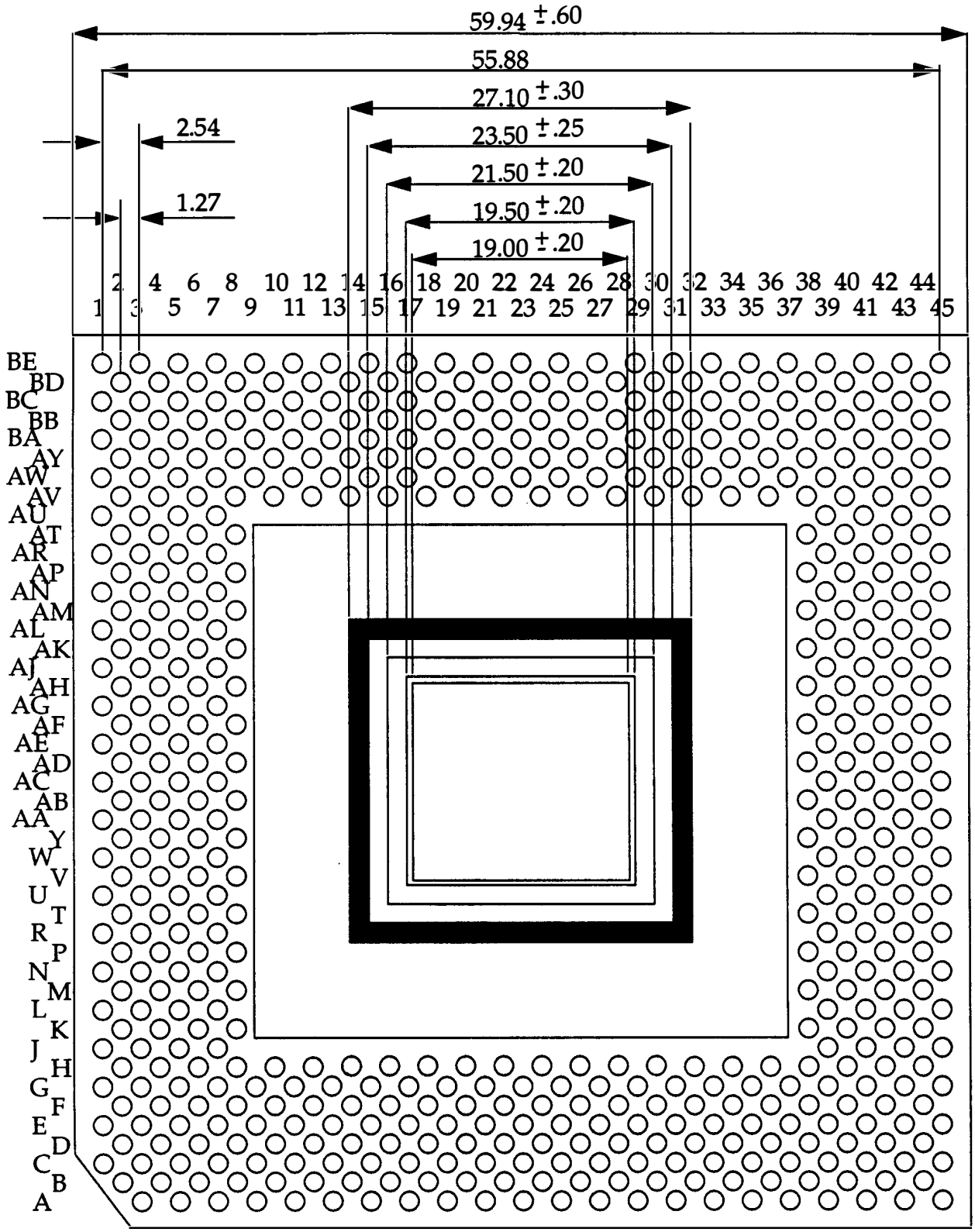


Figure 8-10 R8000/R8010 Package Dimensions -- Top View

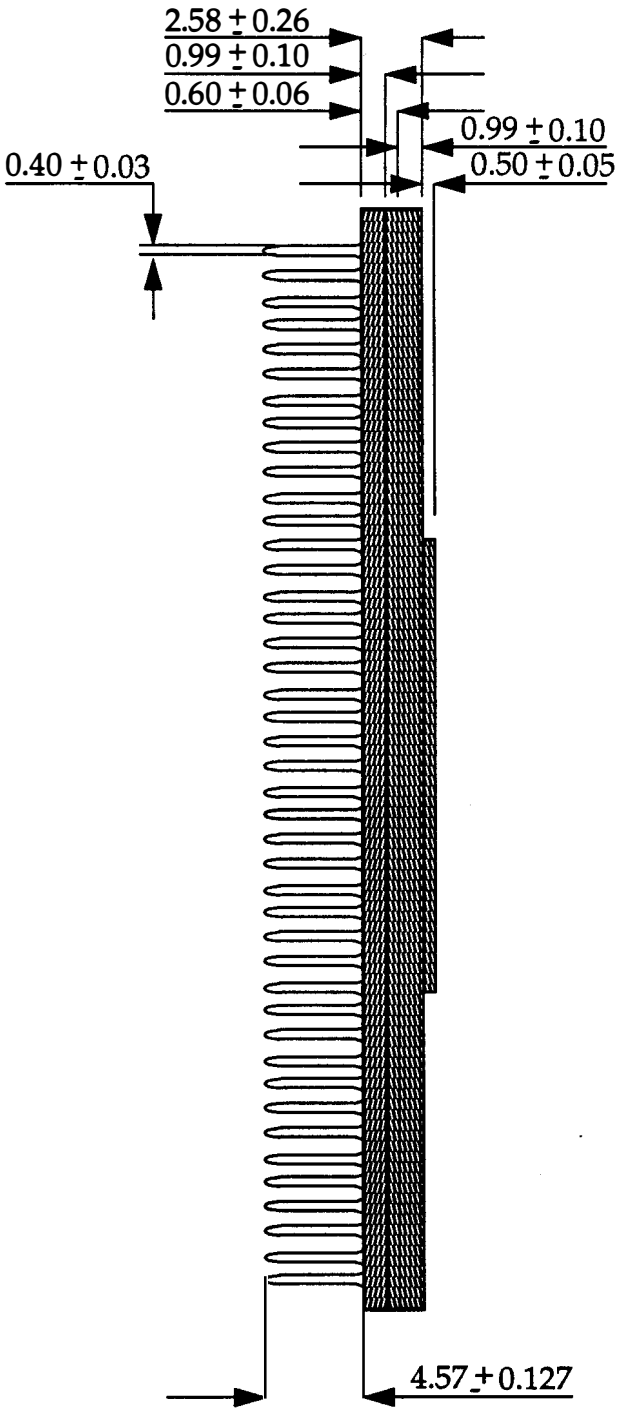


Figure 8-11 R8000/R8010 Package Dimensions -- Side View

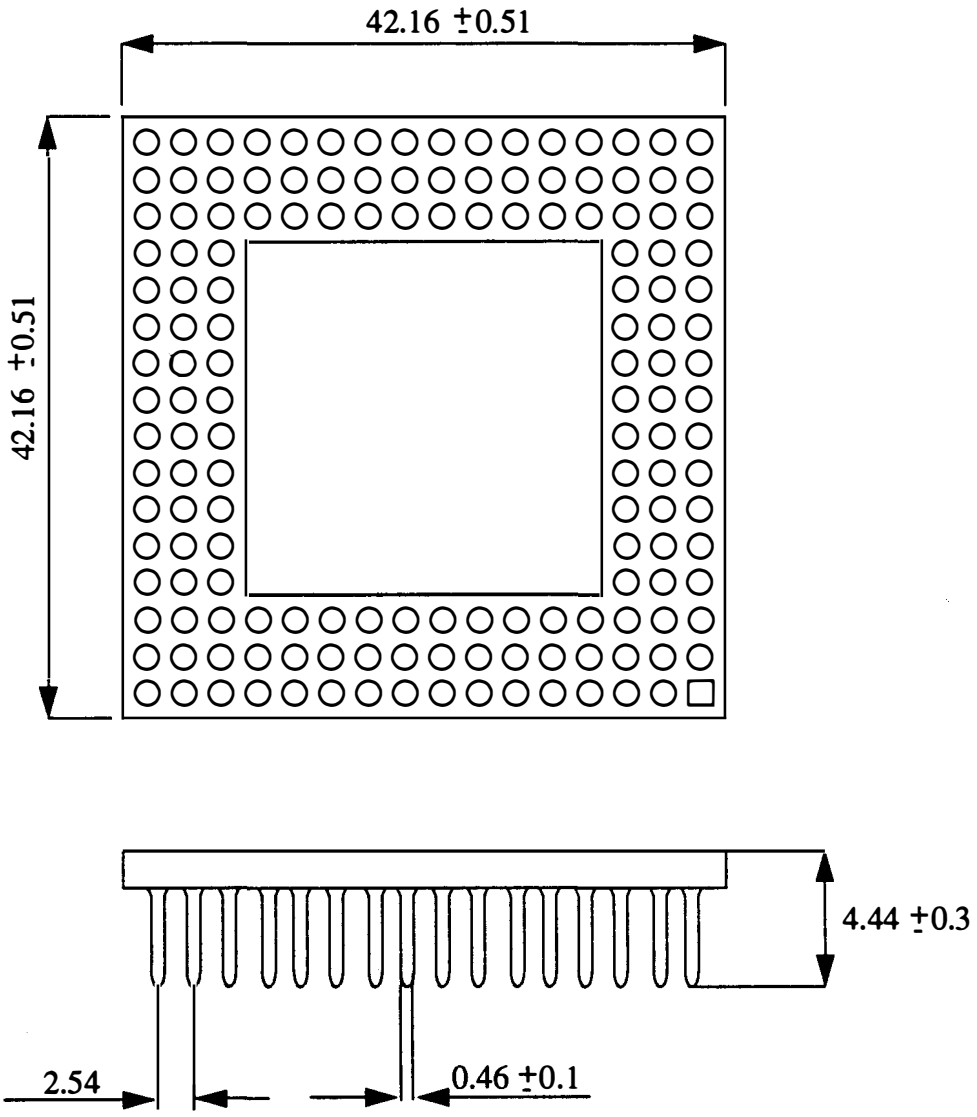


Figure 8-12 Tag RAM Package Dimensions

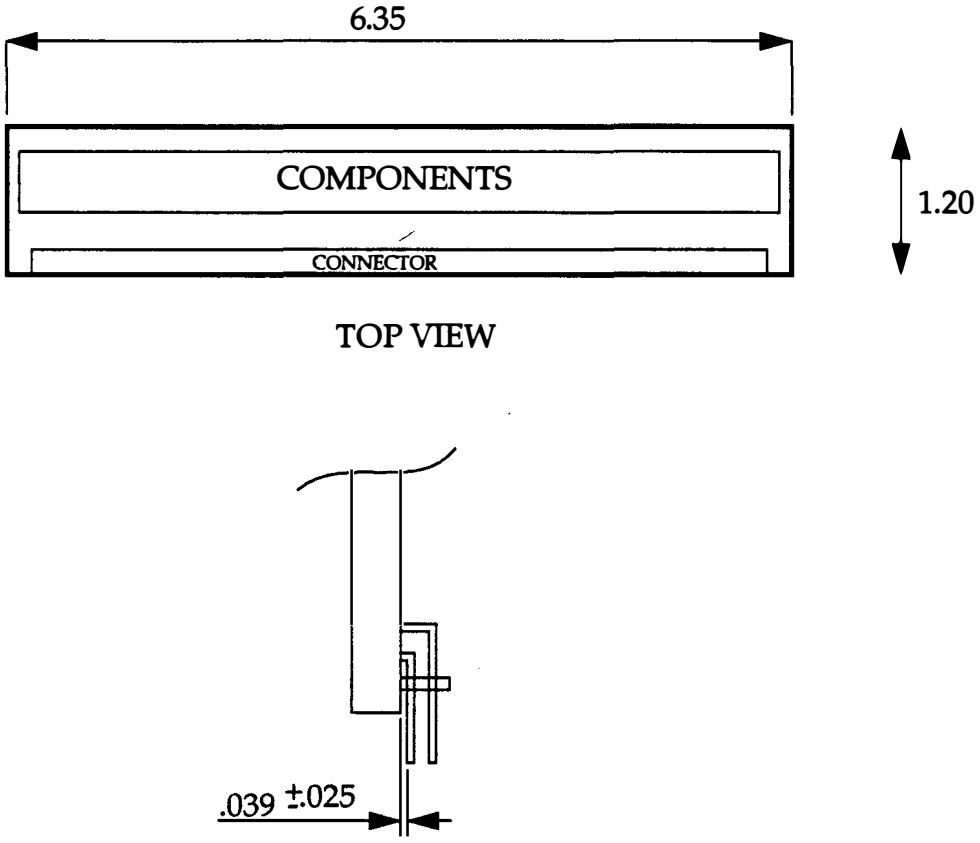


Figure 8-13 Cache RAM SIM Module Package Dimensions

HARDWARE INTERFACE

9

Chapter 9 defines the hardware interface for the R8000 Microprocessor, R8010 Floating Point Unit, Tag RAM, and streaming cache data RAM's. A (_) at the end of a signal name denotes that the signal is active low. In this chapter the terms "Streaming Cache RAM", "Data RAM's", and "Second Level Cache SRAM" all have the same meaning. Figure 9-1 shows how the different components of the R8000 Microprocessor Chip Set connect together.

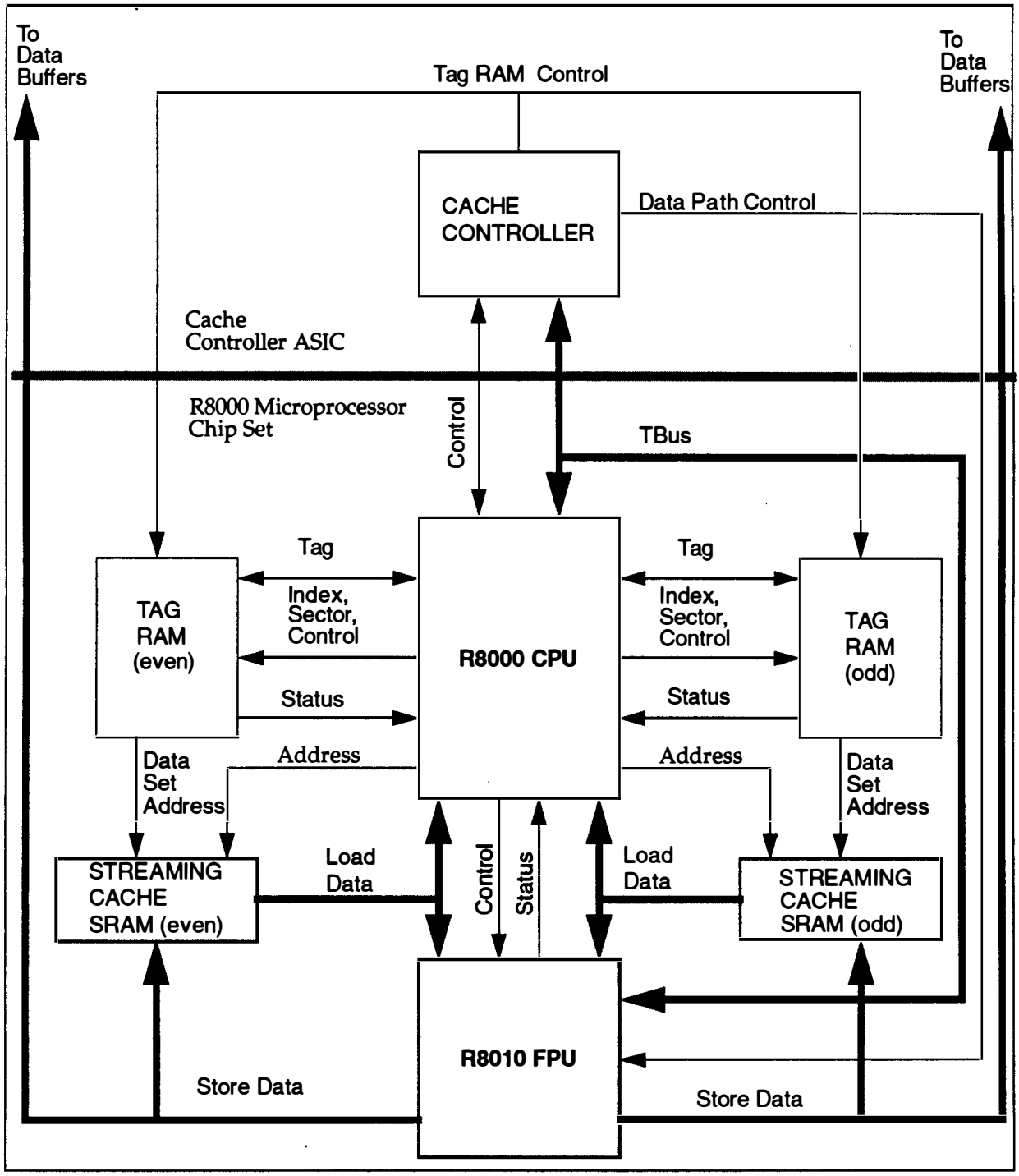


Figure 9-1 R8000 Microprocessor Component Connection Diagram

9.1 R8000 MICROPROCESSOR SIGNAL DESCRIPTIONS

The R8000 Microprocessor is a 591 pin device and interfaces to all other components in the chip set. The following sections define the external pinout of the R8000 Microprocessor and are divided into specific component interfaces. Figure 9-2 shows the functional pin groupings of the R8000.

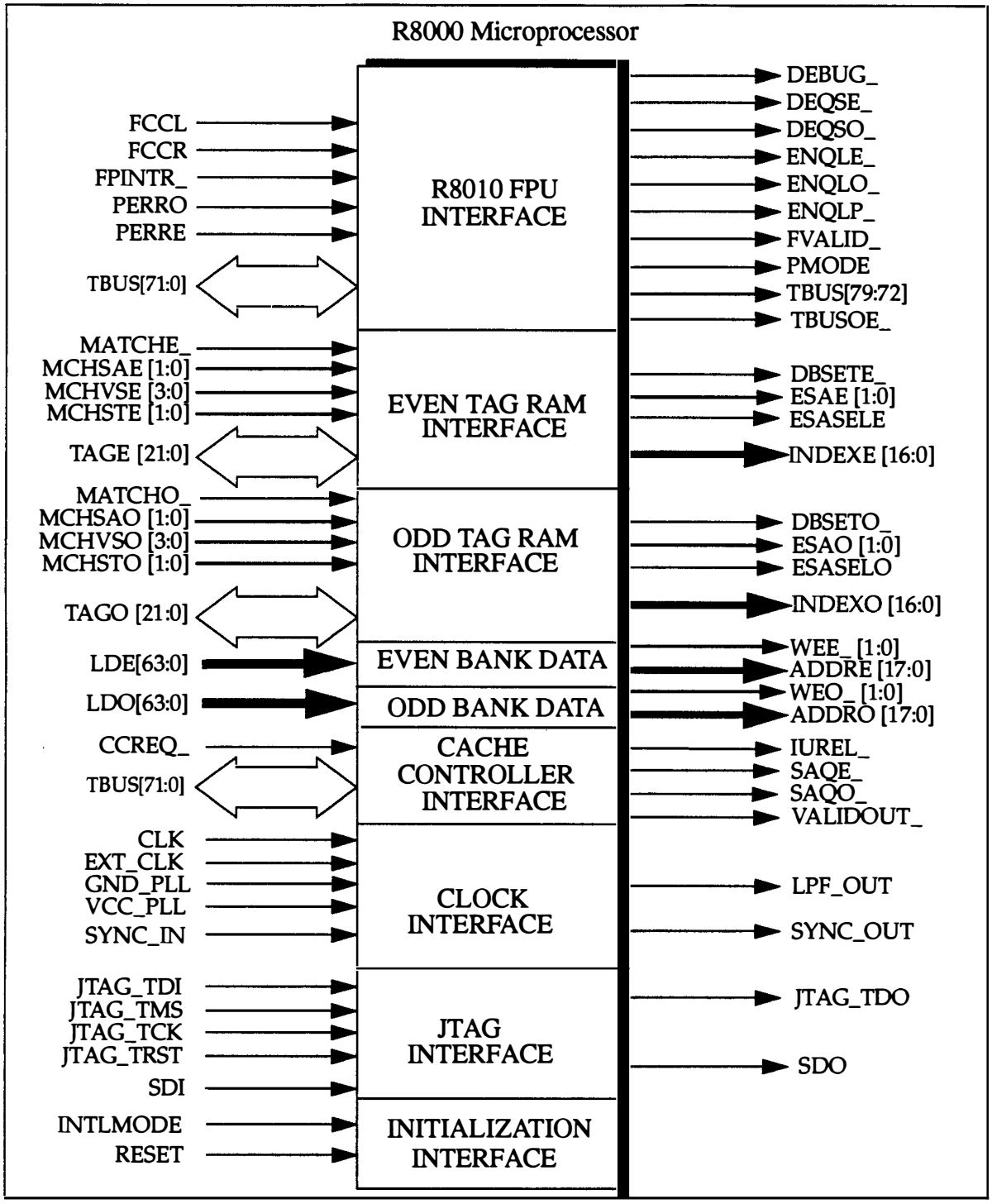


Figure 9-2 R8000 Microprocessor Signal Groupings

Table 9-1 shows a pin summary of the R8000 Microprocessor in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
ADDRE[17:0]	Address Even	High	Even Bank Data
ADDRO[17:0]	Address Odd	High	Odd Bank Data
DBSETE_	Dirty Bit Set Even	Low	Even Tag RAM
DBSETO_	Dirty Bit Set Odd	Low	Odd Tag RAM
DEBUG_	Debug	Low	R8010 FPU
DEQSE_	Dequeue Store Even	Low	R8010 FPU
DEQSO_	Dequeue Store Odd	Low	R8010 FPU
ENQLE_	Enqueue Load Even	Low	R8010 FPU
ENQLO_	Enqueue Load Odd	Low	R8010 FPU
ENQLP_	Enqueue Load Priority	Low	R8010 FPU
ESAE[1:0]	External Set Address Even	High	Even Tag RAM
ESAO[1:0]	External Set Address Odd	High	Odd Tag RAM
ESASELE	Ext. Set Addr. Select Even	High	Even Tag RAM
ESASELO	Ext. Set Addr. Select Odd	High	Odd Tag RAM
FVALID_	R8010 FPU Valid	Low	R8010 FPU
INDEXE[16:0]	Index Even	High	Even Tag RAM
INDEXO[16:0]	Index Odd	High	Odd Tag RAM
IUREL_	Integer Unit Release	Low	Cache Controller
JTAG_TDO	Jtag Test Data Output	High	External Source
LPF_OUT	PLL Low Pass Filter Test	High	External Source
PMODE	Parity Mode	High	R8010 FPU
SAQE_	Store Adress Queue Even	Low	Cache Controller
SAQO_	Store Address Queue Odd	Low	Cache Controller
SDO	Serial Data Out	High	External Source
SYNC_OUT	PLL Feedback Loop	High	Sync_In pin of R8000
TBUS[79:72]	TBus	High	R8010 FPU
TBUSOE_	TBus Output Enable	Low	R8010 FPU
VALIDOUT_	Valid Out	Low	Cache Controller
WEE[1:0]_	Write Enable Even	Low	Even BankData
WEO[1:0]_	Write Enable Odd	Low	Odd Bank Data
Input Pins			
CCREQ_	Cache Controller Request	Low	Cache Controller
CLK	Reference Clock (Uses PLL)	High	External Source
EXT_CLK	Ext. Clock (Bypasses PLL)	High	External Source
FCCL	FPU Condition Code Left	High	R8010 FPU

Table 9-1 R8000 Microprocessor Pin Summary

Pin ID	Pin Name	Active Level	Connects To
FCCR	FPU Condition Code Right	High	R8010 FPU
FPINTR_	R8010 FPU Interrupt	Low	R8010 FPU
GND_PLL	Ground Source for PLL	Low	External Source
INTLMODE	Internal Mode	High	External Source
JTAG_TCK	JTAG Clock	High	External Source
JTAG_TDI	JTAG Test Data In	High	External Source
JTAG_TRST	JTAG Test Reset	High	External Source
JTAG_TMS	JTAG Test Mode Select	High	External Source
LDE[63:0]	Load Data Even	High	Even Bank Load Data
LDO[63:0]	Load Data Odd	High	Odd Bank Load Data
MATCHE_	Match Even	Low	Even Tag RAM
MATCHO_	Match Odd	Low	Odd Tag RAM
MCHSAE[1:0]	Match Set Address Even	High	Even Tag RAM
MCHSAO[1:0]	Match Set Address Odd	High	Odd Tag RAM
MCHSTE[1:0]	Match State Even	High	Even Tag RAM
MCHSTO[1:0]	Match State Odd	High	Odd Tag RAM
MCHVSE[3:0]	Match Virtual Synonym Even	High	Even Tag RAM
MCHVSO[3:0]	Match Virtual Synonym Odd	High	Odd Tag RAM
PERRE	Parity Error Even	High	R8010 FPU
PERRO	Parity Error Odd	High	R8010 FPU
RESET_	R8000 Reset Pin	Low	External Source
SDI	Serial Data In	High	External Source
SYNC_IN	PLL Feedback Loop	High	Sync_Out pin of R8000
VCC_PLL	Voltage Source for PLL	High	External Source
Input/Output Pins			
TAGE[21:0]	Tag Address Even	High	Even Tag RAM
TAGO[21:0]	Tag Address Odd	High	Odd Tag RAM
TBUS[71:0]	TBus Interface	High	R8010 FPU/CC

Table 9-1 R8000 Microprocessor Pin Summary

9.1.1 R8000 Microprocessor to R8010 FPU Interface

This section defines the pins between the R8000 and the R8010 Floating Point Unit. The pins are listed in alphabetical order.

DEBUG_ (Debug) Active Low Output

This signal indicates whether the results of a FP operation are written into the Floating Point Register File when an exception occurrs. When DEBUG is active (0), a floating point operation that raises the enabled exception does not write its result to the FPR. For a disabled exception the FPR is updated with the appropriate substitution value. When DEBUG is inactive (1), a floating point operation that raises an exception writes its appropriate substitution value to the FPR regardless of whether the exception is enabled or disabled. DEBUG_ is connected directly to the DEBUG_ pin of the R8010 FPU.

DEQSE_ (Dequeue Store Even) Active Low Output

This signal indicates when an even 64 bit doubleword of data should be read from the store data queue and the Dequeue Even Pointer updated. DEQSE_ is connected directly to the DEQSE_ pin of the R8010 FPU.

DEQSO_ (Dequeue Store Odd) Active Low Output

This signal indicates when an odd 64 bit doubleword of data should be read from the store data queue and the Dequeue Even Pointer updated. DEQSO_ is connected directly to the DEQSO_ pin of the R8010 FPU.

ENQLE_ (Enqueue Load Even) Active Low Output

This signal indicates when an even 64 bit doubleword of data from the streaming cache should be written into the load data queue and the enqueue load pointer incremented. ENQLE_ is connected directly to the ENQLE_ pin of the R8010 FPU.

ENQLO_ (Enqueue Load Odd) Active Low Output

This signal indicates when an odd 64 bit doubleword of data from the streaming cache should be written into the load data queue and the enqueue load pointer incremented. ENQLO_ is connected directly to the ENQLO_ pin of the R8010 FPU.

ENQLP_ (Enqueue Load Priority) Active Low Output

This signal indicates the order of execution between two loads to be enqueued. ENQLP asserted indicates that the odd 64 bit doubleword should be enqueued before the even 64 bit doubleword. ENQLP_ is connected directly to the ENQLP_ pin of the R8010 FPU.

FCCL (Floating Point Condition Code Left) Active High Input

This signal is the condition code generated by the left R8010 FPU execution unit. The R8000 keeps track of the floating point operation, the CC destination, and the pipeline stage. FCCL is connected directly to the FCCL pin of the R8010 FPU.

FCCR (Floating Point Condition Code Right) Active High Input

This signal is the condition code generated by the right R8010 FPU execution unit. The R8000 keeps track of the floating point operation, the CC destination, and the pipeline stage. FCCR is connected directly to the FCCR pin of the R8010 FPU.

FPINTR_ (Floating Point Interrupt) Active Low Input

This signal indicates that a floating point exception has occurred on either execution unit and the enable bit of the FSR for that type of exception was active. FPINTR_ is connected directly to the FPINTR_ pin of the R8010 FPU.

FVALID_ (FPU Valid) Active Low Output

This signal active indicates that the TBus has a valid FPU operation during the current cycle. FVALID is connected directly to the FVALID pin of the R8010 FPU.

PERRE (Parity Error Even) Active High Input

This signal is generated by the R8010 FPU and is asserted when a parity error is detected on the load data and load parity busses of the even bank. PERRE is connected directly to the PERRE pin of the R8010 FPU.

PERRO (Parity Error Odd) Active High Input

This signal is generated by the R8010 FPU and is asserted when a parity error is detected on the load data and load parity busses of the odd bank. PERRO is connected directly to the PERRO pin of the R8010 FPU.

PMODE (Parity Mode) Active High Output

The PMODE pin determines whether odd or even parity checking and generation is to

be performed by the R8010 FPU and is connected directly to the PMODE pin of the R8010 FPU. When enabled, even parity is checked on the even and odd load data busses of the R8010 FPU and is generated for the even and odd store data busses of the R8010 FPU for data going to the streaming cache.

TBUS [79:72] Active High Output

Part of the 80 bit TBus. These bits are output only and connect directly to the TBUS [79:72] pins of the R8010 FPU and are used for transferreing data and operations to the R8010 FPU. When TBUS [79:78] equals '10', the TBus is able to transfer two memory specifiers and two floating point operations to the R8010 FPU. When TBUS [79:78] does not equal '10', the TBus is used for special operations such as integer stores and moves to and from the R8010 FPU. TBUS [79:72] is valid whenever the FVALID_ signal is active.

TBUS [71:0] Active High Bi-Directional

This bi-directional bus connects between the CC, the R8000, and the R8010 FPU. The function of each bit changes depending on which device is driving. Normally the CC drives the TBus when reading or writing the tag RAM's or reading the Data RAM's and for general communication with the R8000. The R8010 FPU uses the TBus to transfer Move data from the floating point register file (FPR) to the Integer Register File of the R8000 as requested by the R8000. The R8000 uses the TBus for integer stores to the data RAM's, general communication with the CC and the R8010 FPU, and R8010 FPU to R8000 move instructions. TBUS [71:0] connects directly to TBUS [71:0] of the FPU as well as TBUS [71:0] of the Cache Controller. TBus connection between the CC and the R8010 FPU is by virtue of the fact the the R8000 communicates with both. There is no TBus communication protocol between the R8010 FPU and the CC.

TBUSOE_ (Tbus Output Enable) Active Low Output

This signal is used as a tri-state enable for the TBus [71:0] pin connections to the R8010 FPU. TBUSOE_ is connected directly to the TBUSOE_ pin of the R8010 FPU. Two cycles after TBUSOE_ is asserted the R8000 receives valid input data on the TBus.

9.1.2 R8000 Microprocessor to Even Tag RAM

This section defines the pins between the R8000 Microprocessor and the even tag RAM.

The signals are listed in alphabetical order.

DBSETE_ (Dirty Bit Set Even) Active Low Output

This pin is the dirty bit RAM write enable and is connected to the DBSETE_ pin of the even tag RAM. The tag RAM contains a separate on-chip 16-bit wide dirty bit RAM which contains the status of each of the sectors for each of the 4 ways of the tag RAM. The dirty bit RAM is accessed on lookup cycles regardless of the state of the line. If the access is a hit and DBSETE_ is active, the dirty bit is written one clock after the lookup is done.

ESAE [1:0] (External Set Address Even) Active High Output

When a tag RAM lookup is performed, logic inside the tag RAM determines which of the four ways compared correctly with the tag address on the bus and encodes this information as a two bit value called the 'match set address'. Oftentimes the R8000 will perform the tag lookup many cycles before the corresponding write to the data RAM's is done. When this occurrs there must be a mechanism to store the 2 bit set address value for use during the actual write. The lookup is performed ahead of time and the result is stored to the R8000 via the match set address (MCHSA) pins of the tag RAM. When the write is done this information is driven back through the tag RAM via the ESAE pins and becomes the upper two bits of the even data RAM address of the second level cache. ESAE [1:0] is connected directly to the ESAE [1:0] pins of the even tag RAM.

ESASELE (External Set Address Select Even) Active High Output

This pin controls a multiplexor inside the tag RAM which allows the data set address for the data RAM's to be driven either directly from the tag RAM compare result logic or from the ESAE pins. If ESASELE is asserted the ESAE [1:0] values are driven out to the data RAM's. This pin is normally only asserted on write cycles when the tag RAM lookup is performed before the corresponding write. ESASELE is connected directly to the ESASELE pin of the even tag RAM.

INDEXE [16:0] (Index Even) Active High Output

These bits form the index to the even tag RAM. This bus is output only. In a four Megabyte cache implementation the connections to the Tag RAM are as follows:

INDEXE [1:0] Not Used. To be left unconnected.
INDEXE [3:2] connect to SECTOR [1:0] pins of the even Tag RAM.
INDEXE [13:4] connect to INDEX [9:0] of the even Tag RAM.
INDEXE [16:14] connect to INDEX [12:10] of the even Tag RAM through hardware

jumpers.

MATCHE_ (Match Even) Active Low Input

This signal is an output of the even Tag RAM. When the R8000 Microprocessor performs a lookup cycle to the tag RAM a portion of the physical address is used as an index which selects one of 2048 Tag RAM entries. Each entry contains all four sets of the four way set associative even tag RAM. Assertion of the MATCHE_ signal by the tag RAM indicates that the address on the bus compared to one of the four sets. MCHSAE [1:0] below encodes which of the four sets compared. MATCHE_ is connected directly to the MATCHE_ pin of the even tag RAM.

MCHSAE [1:0] (Match Set Address Even) Active High Input

These two bits are used on R8000 lookup cycles to the Tag RAM and encode which of the four ways in the 4-way set associative Tag RAM the compare occurred. Oftentimes the lookup for a store, which determines whether or not the requested address is in the cache, is performed many cycles before the corresponding store data becomes avaliable. MCHSAE [1:0] allows the R8000 to store the result of the compare until the store data is available. The information is used by the R8000 Microprocessor to construct the upper two bits of the Data RAM cache address. MCHSAE [1:0] connect directly to the MCHSAE [1:0] pins of the even tag RAM.

MCHSTE [1:0] (Match State Even) Active High Input

This two bit value is driven by the even Tag RAM on lookup cycles and indicates the state information for one of four 128 byte sectors in the streaming cache corresponding to the address which compared. This two bit value encodes which state a given line in the streaming cache is in; Shared, Invalid, or Exclusive. MCHSTE [1:0] connect directly to the MCHSTE [1:0] pins of the even tag RAM.

MCHVSE [3:0] (Match Virtual Synonym Even) Active High Input

These signals represent virtual address bits [15:12]. This four bit virtual synonym value is stored in the even Tag RAM along with each 20 bit tag address. Because the data cache is virtually indexed and is larger than some of the programmable page sizes, multiple virtually indexed locations in the Data cache can have the same physical address. To insure that not more than one physical location is active at a time, a check of virtual address bits [15:12] is done at the same time as the address compare to assure that the same virtual address is being accessed. These four virtual synonym bits tell of the last location in the Data cache to be used. MCHVSE [3:0] connect directly to the MCHVSE [3:0] pins of the even tag RAM.

TAGE [21:0] (Tag Address Even) Active High I/O

The Tag RAM address bus is bi-directional. The address is driven by the R8000 on lookup cycles, and driven by the CC on writes through the TBus. On read cycles the tag RAM drives the address information to the CC through the TBus as requested. In a 4 Megabyte cache implementation connections to the Tag RAM are as follows:

TAGE [0] Not Used. To be left unconnected.
TAGE [1] connects to TAG [17] of the even Tag RAM through a hardware jumper.
TAGE [3:2] connect to TAG [19:18] pins of the even Tag RAM.
TAGE [20:4] connect to TAG [16:0] of the even Tag RAM.
TAGE [21] connects to TAG [17] of the even Tag RAM through a hardware jumper.

9.1.3 R8000 Microprocessor to Odd Tag RAM

This section defines the pins between the R8000 Microprocessor and the odd tag RAM. The signals are listed in alphabetical order.

DBSETO_ (Dirty Bit Set Odd) Active Low Output

This pin is the dirty bit RAM write enable and is connected to the DBSETO_ pin of the odd tag RAM. The tag RAM chip contains a separate 16-bit wide dirty bit RAM which contains the status of each of the sectors for each of the 4 ways of the tag RAM. The dirty bit RAM is accessed on lookup cycles regardless of the state of the line. If the access is a hit and DBSETO_ is active, the dirty bit is written one clock after the lookup is done. DBSETO_ is connected directly to the DBSETO_ pin of the odd tag RAM.

ESAO [1:0] (External Set Address Odd) Active High Output

When a tag RAM lookup is performed, logic inside the tag RAM determines which of the four ways compared correctly with the tag address on the bus and encodes this information as a two bit value called the 'match set address'. Oftentimes the R8000 will perform the tag lookup many cycles before the corresponding write to the data RAM's is done. When this occurrs there must be a mechanism to store the 2 bit set address value for use during the actual write. The lookup is performed ahead of time and the result is stored to the R8000 CPU via the match set address (MCHSA) pins of the tag RAM. When

the write is done this information is driven back through the tag RAM via the ESAO pins and becomes the upper two bits of the odd bank streaming cache RAM address. ESAO [1:0] are connected directly to the ESA [1:0] pins of the odd tag RAM.

ESASELO (External Set Address Select Odd) Active High Output

This pin controls a multiplexor inside the tag RAM which allows the data set address for the data RAM's to be driven either directly from the tag RAM compare result logic or from the ESAO pins. If ESASELO is asserted the ESAO values are driven out to the data RAM's. This pin is normally only asserted on write cycles when the tag RAM lookup is performed before the corresponding write. ESASELO is connected directly to the ESASELO pin of the odd tag RAM.

INDEXO [16:0] (Index Odd) Active High Output

These bits form the index to the odd tag RAM. This bus is output only. In a 4 Megabytre implementation connections to the Tag RAM are as follows:

INDEXE [1:0] Not Used. To be left unconnected.
INDEXE [3:2] connect to SECTOR [1:0] pins of the odd Tag RAM.
INDEXE [13:4] connect to INDEX [9:0] of the odd Tag RAM.
INDEXE [16:14] connect to INDEX [12:10] of the odd Tag RAM through hardware jumpers.

MATCHO_ (Match Odd) Active Low Input

This signal is an output of the odd Tag RAM. When the R8000 performs a lookup cycle to the tag RAM a portion of the physical address is used as an index which selects one of 2048 Tag RAM entries. Each entry contains all four sets of the four way set associative odd tag RAM. Assertion of the MATCHO_ signal by the tag RAM indicates that the address on the bus compared to one of the four sets. MCHSAO [1:0] below encodes which of the four sets compared. MATCHO_ is connected directly to the MATCHO_ pin of the odd tag RAM.

MCHSAO [1:0] (Tag RAM Match Set Address Odd) Active High Input

These two bits are used on R8000 lookup cycles to the Tag RAM and encode which of the four ways in the 4-way set associative Tag RAM the compare occurred. Oftentimes the lookup for a store, which determines whether or not the requested address is in the cache, is performed many cycles before the corresponding store data becomes avaliable. MCHSAO [1:0] allows the R8000 to store the result of the compare until the store data is available. The information is used by the R8000 to construct the upper two bits of the

Data RAM cache address. MCHSAO [1:0] connect directly to the MCHSAO [1:0] pins of the odd tag RAM.

MCHSTO [1:0] (Tag RAM Match State Odd) Active High Input

This two bit value is driven by the odd Tag RAM on lookup cycles and indicates the state information for one of four 128 byte sectors in the streaming cache corresponding to the address which compared. This two bit value encodes which state a given line in the streaming cache is in; Shared, Invalid, or Exclusive. MCHSTO [1:0] are connected directly to the MCHSTO [1:0] pins of the odd tag RAM.

MCHVSO [3:0] (Match Virtual Synonym Odd) Active High Input

These signals represent virtual address bits [15:12]. This four bit virtual synonym value is stored in the odd Tag RAM along with each 20 bit tag address. Because the data cache is virtually indexed and is larger than some of the progaammable page sizes, multiple virtually indexed locations in the Data cache can have the same physical address. To insure that not more than one physical location is active at a time, a check of virtual address bits [15:12] is done at the same time as the address compare to assure that the same vitrual address is being accessed. These four virtual synonym bits tell of the last location in the Data cache to be used. MCHVSO [3:0] are connected directly to the MCHVSO [3:0] pins of the odd tag RAM.

TAGO [21:0] (Tag Address Odd) Active High I/O

The Tag RAM address bus is bidirectional. The address is driven by the R8000 on lookup cycles, and driven by the CC on writes through the TBus. On read cycles the tag RAM drives the address information through the TBus to the CC as requested. In a 4 Megabyte cache implementation connections to the Tag RAM are as follows:

TAGE [0] Not Used. To be left unconnected.
TAGE [1] connects to TAG [17] of the even Tag RAM through a hardware jumper.
TAGE [3:2] connect to TAG [19:18] pins of the even Tag RAM.
TAGE [20:4] connect to TAG [16:0] of the even Tag RAM.
TAGE [21] connects to TAG [17] of the even Tag RAM through a hardware jumper.

9.1.4 R8000 Microprocessor to Even Bank Streaming Cache

This section defines the pins between the R8000 CPU and the even bank of streaming cache data RAM's. The signals are listed in alphabetical order.

ADDRE [17:0] (Address Even) Active High Output

These pins supply the address to the second level cache. ADDRE [17:0] of the R8000 connect directly to ADDRE [17:0] of both SIM modules of the even data bank .

LDE [63:0] (Load Data Even) Active High Input

Load data bus between the R8000 CPU and the even bank of the streaming cache. This bus is unidirectional and accepts load input data only. Store data is transferred through the FPU via the TBus. Data transfers between the streaming cache and the R8000 are synchronous and conform to a 5 cycle pipeline. When a streaming cache load is initiated by the R8000, data is returned in 5 clocks. Streaming cache misses take approximately 50 clocks. There are two 32 bit wide SIM modules on the even data bank. LDE [63:32] are connected directly to the LD [32:0] pins of the upper 32 bit even data RAM module. LDE [31:0] are connected directly to the LD [32:0] pins of the lower 32 bit even data RAM module.

WEE_ [1:0] (Write Enable Even) Active Low Output

The streaming cache data RAM's can be written either by the R8000 or by the CC. Writes by the CC are handled through the FPU via the TBus. WEE_ [1] is connnected to the WE_ pin of the RAM's containing the upper 32 bits of data for the even bank. WEE_ [0] is connnected to the WE_ pin of the RAM's containing the lower 32 bits of data for the even bank. Normally the R8000 CPU drives these pins during write hits to the streaming cache. But the CC can also write the streaming cache. This is normally done on streaming cache misses where the requested data has to be fetched from main memory. In this case the CC is responsible for fetching the data from main memory and writing the data to the streaming cache through the TBus. Bit 63 of the TBus is passed through the R8000 CPU onto WEE_ [1] to write the upper 32 bits of the even bank. Bit 62 of the TBus is passed through the R8000 CPU onto WEE_ [0] to write the lower 32 bits of the even bank.

9.1.5 R8000 Microprocessor to Odd Bank Streaming Cache

This section defines the pins between the R8000 Microprocessor and the odd bank of streaming cache data RAM's. The signals are listed in alphabetical order.

ADDRO [17:0] (Address Odd) Active High Output

These pins supply the address to the second level cache. ADDRO [17:0] of the R8000 CPU connect directly to ADDRO [17:0] of both SIM modules of the odd data bank .

LDO [63:0] (Load Data Odd) Active High Input

Load data bus between the R8000 and the odd bank of cache SRAM. This bus is unidirectional and accepts load input data only. Store data is transferred through the FPU via the TBus. Data transfers between the streaming cache and the R8000 are synchronous and conform to a 5 cycle pipeline. When a streaming cache load is initiated by the R8000 CPU, data is returned in 5 clocks. Streaming cache misses take approximately 50 clocks. There are two 32 bit wide SIM modules on the odd bank. LDO [63:32] are connected directly to the LD [32:0] pins of the upper 32 bit odd data RAM module. LSO [31:0] are connected directly to the LD [32:0] pins of the lower 32 bit odd data RAM module.

WEO_ [1:0] (Write Enable Odd) Active Low Output

The streaming cache data RAM's can be written either by the R8000 or by the CC. Writes by the CC are handled through the TBus. WEO_ [1] is connnected to the WE_ pin of the SIMM containing the upper 32 bits of data for the odd bank. WEO_ [0] is connnected to the WE_ pin of the SIMM containing the lower 32 bits of data for the odd bank. Normally the R8000 drives these pins during write hits to the streaming cache. But the CC also can write the streaming cache. This is normally done on streaming cache misses where the requested data has to be fetchèd from main memory. In this case the CC is responsible for fetching the data from main memory and writing it to the streaming cache through the TBus. Bit [61] of the TBus is passed through the R8000 onto WEO_ [1] to write the upper 32 bits of the odd bank. Bit [60] of the TBus is passed through the R8000 onto WEO_ [0] to write the lower 32 bits of the odd bank.

9.1.6 R8000 Microprocessor to Cache Controller

This section defines the pins between the R8000 Microprocessor and the cache controller. The signals are listed in alphabetical order.

CCREQ_ (Cache Controller Request) Active Low Input

CCREQ_ is driven by the CC and when asserted indicates that the CC is either

requesting control or does not wish to give up control of the TBus. CCREQ_ is connected directly to the CCREQ_ pin of the Cache Controller.

IUREL_ (Integer Unit Release) Active Low Output

IUREL_ is driven by the R8000 and when asserted indicates that the R8000 CPU is giving control of the TBus to the Cache Controller. IUREL_ is connected directly to the IUREL_ pin of the Cache Controller. The R8000 can release control of the TBus in response to the assertion of CCREQ_ by the cache controller, or it can release the bus immediately after incurring a data cache instruction cache, or TLB miss.

NXTDATE_ (Store Address Queue Even) Active Low Output

This signal is driven by the R8000 CPU and indicates that store data associated with a non-cacheable write will be on the even store data bus on the next clock. When the CC is in control of the TBus, the R8000 asserts SAQE_ if the even store address queue contains an address for which bits [17:7] match the Tag RAM index bits [17:7] which were on the TBus four cycles earlier. The signal remains deasserted if no such match is detected. SAQE_ should always be considered together with SAQO_. If either is asserted, a compare hit has occurred. Address comparisons are done on a 128 byte minimum quantity and take one cycle. Address range comparisons larger than 128 bytes require multiple cycles. SAQE_ is connected directly to the SAQE_ pin of the Cache Controller.

NXTDATO_ (Store Address Queue Odd) Active Low Output

This signal is driven by the R8000 CPU and indicates that store data associated with a non-cacheable write will be on the odd store data bus on the next clock. When the CC is in control of the TBus, the R8000 assertes SAQO_ if the odd store address queue contains an address for which bits [17:7] match the Tag RAM index bits [17:7] which were on the TBus four cycles earlier. The signal remains deasserted if no such match is detected. SAQO_ should always be considered together with SAQE_. If either is asserted, a compare hit occurred. Address comparisons are done on a 128 byte minimum quantity and take one cycle. Address range comparisons larger than 128 bytes require multiple cycles. SAQO_ is connected directly to the SAQO_ pin of the Cache Controller.

TBUS [71:0] Active High I/O

These pins form the general communication interface between the R8000 CPU and the CC and connect directly to TBUS [71:0] pins of both the Cache Controller and the R8010 FPU. The function of each bit changes depending on which device is driving. Normally the CC drives the TBus when reading or writing the tag RAM's or reading the Data RAM's and for general communication with the R8000. The R8010 FPU uses the TBus to transfer Move data from the floating point register file (FPR) to the Integer Register File

of the R8000 as requested by the R8000. The R8000 CPU uses the TBus for integer stores to the data RAM's, general communication with the CC and the R8010 FPU, and R8010 FPU to R8000 CPU move instructions. TBUS [71:0] connects directly to TBUS [71:0] of the R8010 FPU as well as TBUS [71:0] of the Cache Controller. TBus connection between the CC and the R8010 FPU is by virtue of the fact the the R8000 CPU communicates with both. There is no TBus communication protocol between the R8010 FPU and the CC.

TBus [67:64] forms the Function field when the CC is in control. The function field is the only field of the TBus by which the CC changes the internal state of the R8000 chip itself rather than just the tag RAM's or the Data RAM's. The 4 bit field translates to 16 possible functions that the CC can perform on the R8000 Microprocessor. TBUS [71:0] connects directly to the TBUS [71:0] pins of the Cache Controller.

VALIDOUT_ (Active Low Output)

VALIDOUT_ is driven by the R8000 to indicate that the information on the TBus is valid for the Cache Controller to receive. VALIDOUT_ is connected directly to the VALIDOUT_ pin of the Cache Controller.

9.1.7 Clock Interface Signals

CLK (Reference Clock) Active High Input

Master input clock to the Phase Lock Loop (PLL) circuitry of the R8000 Microprocessor. The output of the PLL is then used as the master clock for the chip. CLK is normally connected directly to the ouput of the external clock driver. In most cases it is desirable to use the PLL circuitry, but for those applications which to not wish to use the PLL, the clock drivers should be connected to EXT_CLK.

EXT_CLK (External Clock) Active High Input

The EXT_CLK input allows the system designer to bypass the internal PLL of the R8000 CPU and drive the chip directly from the system clock. When not in use this pin should be tied to ground through a 330 ohm resistor. Refer to figure 7-1.

GND_PLL (Ground Phase Lock Loop)

Ground source for the phase lock loop circuitry. GND_PLL can be connected to VCC_PLL through .1 microfarad and .015 microfarad capacitors in parallel. Refer to figure 7-1.

LPF_OUT (Low Pass Filter Output) Active High Output

LFP_OUT is a special pin used to test the PLL circuitry during component test for monitoring the status of the low-pass filter. LPF_OUT must be connected to VCC_PLL through a 680K ohm resistor.

SYNC_IN (Synchronized PLL input) Active High Input

Sync_in is part of the PLL feedback path and must be connected to Sync_Out in order for the PLL circuitry to work correctly. The pins are made avaliable externally to allow the user to manually alter the phase of the PLL.

SYNC_OUT (Synchronized PLL input) Active High Output

Sync_Out is part of the PLL feedback path and must be connected to Sync_In in order for the PLL circuitry to work correctly. The pins are made avaliable externally to allow the user to manually alter the phase of the PLL.

VCC_PLL (Voltage Phase Lock Loop)

Voltage source for the phase lock loop circuitry. Connected to a regulated 3.3 volt source. VCC_PLL can be connected to GND_PLL through .1 and .015 microfarad capacitors in parallel.

9.1.8 JTAG Interface Signals

The following signals comprise the Test Access Port (TAP) of the FPU. The TAP provides access to many test support functions built into the chip. The TAP consists of three required synchronous inputs, one required synchronous output, and an optional input for asynchronous initialization of the TAP. When the TAP controller is not reset at power-up as a result of features built into the test logic, the asynchronous TRST_ input must be provided to reset the TAP.

In addition to the JTAG compliant logic, the R8000 Microprocessor also contains a seperate scan chain which connects every flip-flop in the random logic control blocks. This scan chain is non-IEEE compliant and is used during component verification testing to test the random logic. These two pins, SDI and SDO, are NOT part of the JTAG circuitry and should not be used.

JTAG_TCK (JTAG Test Clock) Active High Input

The dedicated test clock input is used to provide system-clock independent testing of the serial data paths between components. The dedicated clock input allows for the shifting of test data through the device concurrently with normal operation of the component. In addition, the independent test clock allows test data to be transferred on- and off- chip without changing the state of the on-chip system logic. *TCK is a required signal for proper boundary scan operation.*

JTAG_TMS (JTAG Test Mode Select) Active High Input

The voltage level at the Test Mode Select input is decoded by the TAP controller and used to control on-chip test operations. TMS is sampled on the rising edge of TCK and a change in state to the TMS input should occur on the falling edge of TCK. A pull-up resistor should be used so that an un-driven TMS input appears as high to the internal logic. *TMS is a required signal for proper boundary scan operation.*

JTAG_TDI (JTAG Test Data Input) Active High Input

Serial test instructions and data are transferred to the test logic by the TDI input. The TDI and TDO pins provide for serial movement of test data through the circuit. Information on the TDI input is sampled on the rising edge of TCK and a change in state to the TDI input should occur on the falling edge of TCK. A pull-up resistor should be used so that an un-driven TDI input appears as high to the internal logic. *TDI is a required signal for proper boundary scan operation.*

JTAG_TDO (JTAG Test Data Output) Active High Output

The Test Data Output (TDO) pin is used to transfer test instructions and data from the internal test logic. To avoid race conditions, the TDI and TMS inputs are sampled on the rising edge of TCK, while changes to the TDO output occur on the falling edge. Using opposite edges of the clock allows output data from one device to propogate to another device and be sampled on the following rising edge. *TDO is a required signal for proper boundary scan operation.*

JTAG_TRST_ (JTAG Test Reset) Active High Input

The optional TRST_ input provides for asynchronous initialization of the TAP controller. The TAP controller is asynchronously reset whenever TRST_ is asserted with a low voltage level. A pull-up resistor should be used so that an un-driven TRST_ input appears as high to the internal logic. *Use of TRST_ is NOT required for proper boundary scan operation.* In addition, the TRST_ input must not be used to initialize any system logic within the component. To ensure proper operation of the test logic, the TMS input should be held at a high voltage level while the TRST_ input changes state from a logic zero to a logic one. If rising edges occur simultaneously at the TRST_ and TCK inputs when a logic zero is applied to TMS, a race condition can occur and the operation of the TAP controller is unpredictable. The controller may either remain in the reset state or change to the run state.

SDI (Serial Data In) Active High Input

Non-IEEE compliant scan chain input which allows for testing of the random control logic blocks within the R8000 CPU during component test and verification. SDI should be tied to ground through a 330 ohm resistor.

SDO (Serial Data Out) Active High Output

Non-IEEE compliant scan chain output which allows for testing of the random control logic blocks within the R8000 CPU during component test and verification. SD should be left unconnected during normal operation.

9.1.9 Initialization Interface

INTLMODE (Internal Mode) Active High Input

Internal address register for interfacing to the second level streaming cache. Currently external address registers are used to buffer the address due to the large number of RAM's required to facilitate a 4 MegaByte implementation. However, in the future it is conceivable that fewer RAM's will be needed to suffice the same memory requirements. The R8000 Microprocessor provides an on-chip address register for this purpose. Currently this register is not used because it is unable to drive all of the devices required for the cache. The external register can hopefully be eliminated as RAM sizes increase. The INTLMODE pin should be tied to ground through a 330 ohm resistor.

RESET_ (Processor Reset) Active Low Input

Reset pin of the R8000. Reset should be held low for 4096 clocks before being released. A 12-bit counter can be used to accomplish this. Fifteen clocks after the rising edge of RESET is sampled the R8000 asserts VALIDOUT_ for one clock. Two clocks after the rising edge of VALIDOUT_ the Cache Controller MUST assert the signal CCREQ_ to allow the CC to begin fetching from the boot PROM. CCREQ_ should remain asserted until the R8000 issues IUREL_, indicating that the control of the TBus has been relinquished. Refer to the initialization interface in chapter 6 for more information on RESET.

9.2 R8010 FPU SIGNAL DESCRIPTIONS

The R8010 Floating Point Unit (FPU) is a 591 pin device and has dedicated interfaces to all other components in the system with the exception of the Tag RAM's. There are no signals which go between the FPU and the Tag RAM's. The following sections define the external pinout of the FPU and are divided into specific component interfaces. Figure 9-3 shows the functional pin groupings of the R8010 FPU.

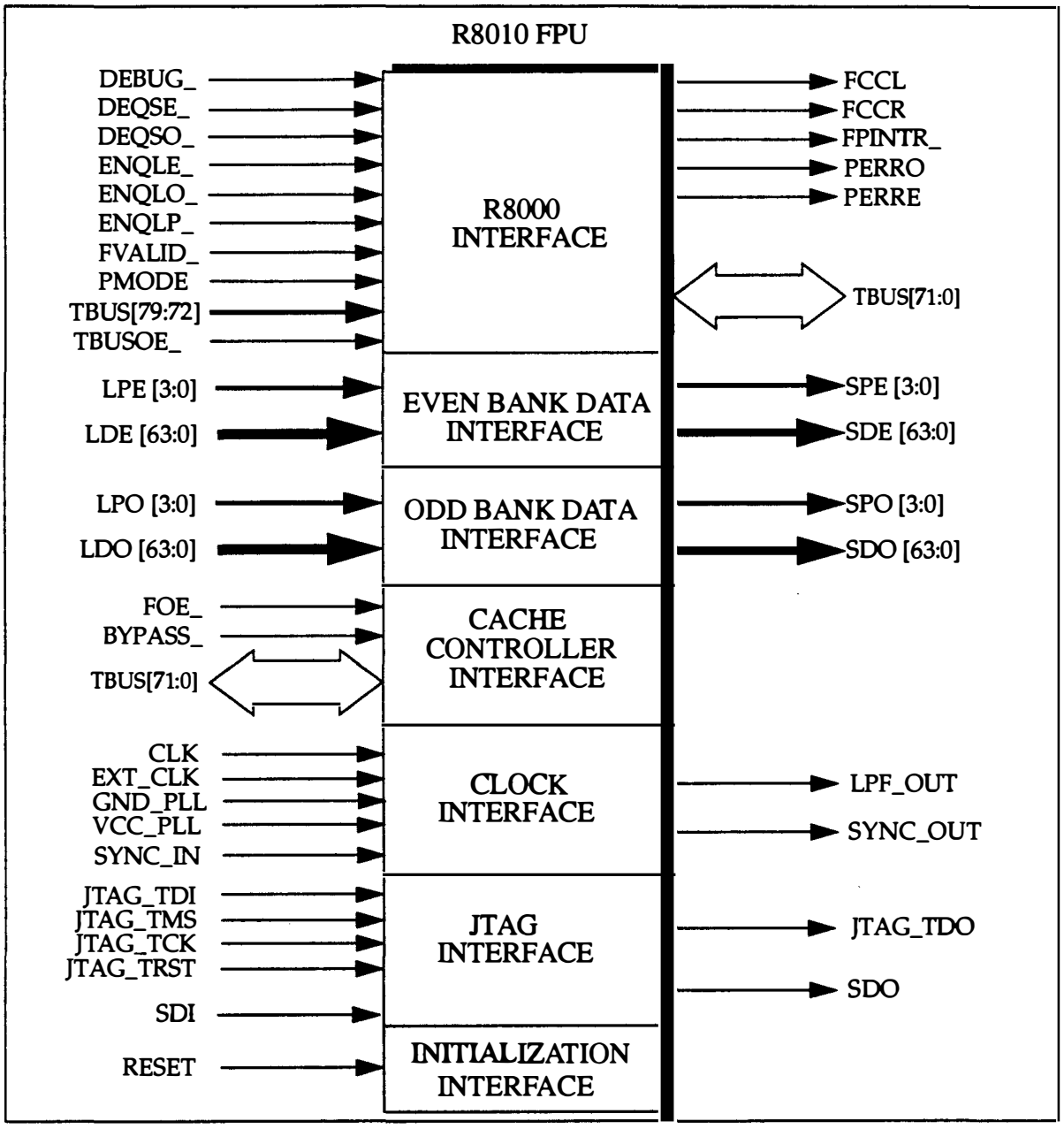


Figure 9-3 R8010 FPU Signal Groupings

Table 9-2 shows a pin summary of the R8010 FPU in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
FCCL	FPU Condition Code Left	High	R8000 CPU
FCCR	FPU Condition Code Right	High	R8000 CPU
FPINTR_	FPU Interrupt	Low	R8000 CPU
JTAG_TDO	JTAG Test Data Output	High	External Source
LPF_OUT	PLL Low Pass Filter Test	High	External Source
PERRE	Parity Error Even	High	R8000 CPU
PERRO	Parity Error Odd	High	R8000 CPU
SDE [63:0]	Store Data Even	High	Even Bank Store Data
SDO	Serial Data Out	High	External Source
SDO [63:0]	Store Data Odd	High	Odd Bank Store Data
SPE [3:0]	Store Parity Even	High	Even Bank Store Parity
SPO [3:0]	Store Parity Odd	High	Odd Bank Store Parity
SYNC_OUT	PLL Feedback Loop	High	Sync_In pin of FPU
Input Pins			
BYPASS_	Floating Point Bypass	Low	Cache Controller
CLK	Reference Clock (Uses PLL)	High	External Source
DEBUG_	Debug	Low	R8000 CPU
DEQSE_	Dequeue Store Even	Low	R8000 CPU
DEQSO_	Dequeue Store Odd	Low	R8000 CPU
ENQLE_	Enqueue Load Even	Low	R8000 CPU
ENQLO_	Enqueue Load Odd	Low	R8000 CPU
ENQLP_	Enqueue Load Priority	Low	R8000 CPU
EXT_CLK	External Clock	High	External Source
FOE_	Floating Point Output Enable	Low	Cache Controller
FVALID_	FPU Valid	Low	R8000 CPU
GND_PLL	Ground Source for PLL	Low	External Source
JTAG_TCK	JTAG Test Clock	High	External Source
JTAG_TDI	JTAG Test Data In	High	External Source
JTAG_TMS	JTAG Test Mode Select	High	External Source
JTAG_TRST	JTAG Test Reset	High	External Source
LDE [63:0]	Load Data Even	High	Even Bank Load Data
LDO [63:0]	Load Data Odd	High	Odd Bank Load Data
LPE [3:0]	Load Parity Even	High	Even Bank Load Parity
LPO [3:0]	Load Parity Odd	High	Odd Bank Load Parity
PMODE	Parity Mode	High	R8000 CPU
RESET_	FPU Reset Pin	Low	External Source

Table 9-2 R8010 FPU Pin Summary

Pin ID	Pin Name	Active Level	Connects To
SDI	Serial Data In	High	External Source
SYNC_IN	PLL Feedback Loop	High	Sync_In pin of FPU
TBUS [79:72]	TBus	High	R8000 CPU
TBUSOE_	TBus Output Enable	Low	R8000 CPU
VCC_PLL	Voltage Source for PLL	High	External Source
Input/Output Pins			
TBUS [71:0]	TBus Interface	High	R8000/CC

Table 9-2 R8010 FPU Pin Summary

9.2.1 R8010 FPU to R8000 Microprocessor

The signals in this section comprise the interface between the R8000 Microprocessor and the R8010 FPU. These signals are described further in section 9.1.

DEBUG_ (Debug Mode) Active Low Input

This signal indicates whether the results are written into the Floating Point Register File when an exception occurrs. When DEBUG is active (0), a floating point operation that raises the enabled exception does not write its result to the FPR. For a disabled exception the FPR is updated with the appropriate substitution value.When DEBUG is inactive (1), a floating point operation that raises an exception writes its appropriate substitution value to the FPR regardless of whether the exception is enabled or disabled. DEBUG_ is connected directly to the DEBUG_ pin of the R8000. Refer to section 9.1.1 for more information on DEBUG_.

DEQSE_ (Dequeue Store Even) Active Low Input

This signal indicates when an even 64 bit doubleword of data should be read from the store data queue and the Dequeue Even Pointer updated. DEQSE_ is connected directly to the DEQSE_ pin of the R8000. Refer to section 9.1.1 for more information on DEQSE_.

DEQSO_ (Dequeue Store Odd) Active Low Input

This signal indicates when an odd 64 bit doubleword of data should be read from the store data queue and the Dequeue Even Pointer updated. DEQSO_ is connected directly to the DEQSO_ pin of the R8000.

ENQLE_ (Enqueue Load Even) Active Low Input

This signal indicates when an even 64 bit doubleword of data from the streaming cache should be written into the load data queue and the enqueue load pointer incremented. ENQLE_ is connected directly to the ENQLE_ pin of the R8000 CPU.

ENQLO_ (Enqueue Load Odd) Active Low Input

This signal indicates when an odd 64 bit doubleword of data from the streaming cache should be written into the load data queue and the enqueue load pointer incremented. ENQLO_ is connected directly to the ENQLO_ pin of the R8000. Refer to section 9.1.1 for more information on ENQLO_.

ENQLP_ (Enqueue Load Priority) Active Low Input

This signal indicates the order of execution between two loads to be enqueued. ENQLP_ asserted indicates that the odd 64 bit doubleword should be enqueued before the even 64 bit doubleword. ENQLP_ is connected directly to the ENQLP_ pin of the R8000. Refer to section 9.1.1 for more information on ENQLP_.

FCCL (Floating Point Condition Code Left) Active High Input

This signal is the condition code generated by the left R8010 FPU execution unit. The R8000 keeps track of the floating point operation, the CC destination, and the pipeline stage. FCCL is connected directly to the FCCL pin of the R8000 CPU. Refer to section 9.1.1 for more information on FCCL.

FCCR (Floating Point Condition Code Right) Active High Input

This signal is the condition code generated by the right R8010 FPU execution unit. The R8000 keeps track of the floating point operation, the CC destination, and the pipeline stage. FCCR is connected directly to the FCCR pin of the R8000 CPU. Refer to section 9.1.1 for more information on FCCR.

FPINTR_ (Floating Point Interrupt) Active Low Output

This signal is driven by the R8010 FPU and indicates that a floating point exception has occurred on either execution unit and that the enable bit of the FSR for that type of

occurred on either execution unit and that the enable bit of the FSR for that type of exception was active. FPINTR_ is connected directly to the FPINTR_ pin of the R8010 FPU. Refer to section 9.1.1 for more information on FPINTR_.

FVALID_ (FPU Valid) Active Low Input

This signal active indicates that the TBus has a valid R8010 FPU operation during the current cycle. FVALID_ is connected directly to the FVALID_ pin of the R8000 CPU. Refer to section 9.1.1 for more information on FVALID_.

PERRE (Parity Error Even) Active High Output

The PERRE signal is generated by the R8010 FPU when PMODE is active and is asserted when a parity error is detected on the load data and load parity busses of the even bank. PERRE is connected directly to the PERRE pin of the R8000 CPU. Refer to section 9.1.1 for more information on PERRE.

PERRO (Parity Error Odd) Active High Output

The PERRO signal is generated by the R8010 FPU and is asserted when a parity error is detected on the load data and load parity busses of the odd bank. PERRO is connected directly to the PERRO pin of the R8000 CPU. Refer to section 9.1.1 for more information on PERRO.

PMODE (Parity Mode) Active High Input

The PMODE pin determines whether even or odd parity checking and generation is to be performed by the R8010 FPU. PMODE is connected directly to the PMODE pin of the R8000. Refer to section 9.1.1 for more information on PMODE.

TBUS [79:72] (Tbus) Active High Input

Part of the 80 bit TBus. These bits are input only and connect directly to the TBUS [79:72] pins of the R8000 Microprocessor and are used for transferring data and operations to the R8010 FPU. TBUS [79:72] is valid whenever the FVALID_ signal is active. Refer to section 9.1.1 for more information on TBUS [79:72].

TBUSOE_ (Tbus Output Enable) Active Low Input

This signal is used as a tri-state enable for the TBus [71:0] pin connections to the R8010

FPU. TBUSOE_ is connected directly to the TBUSOE_ pin of the R8000 CPU. Refer to section 9.1.1 for more information on TBUSOE_.

9.2.2 R8010 FPU to Even Bank Streaming Cache

LDE [63:0] (Load Data Even) Active High Input

Load data bus between the R8010 FPU and the even bank of the streaming cache. This bus is unidirectional and accepts load input data only. Even bank store data is transferred via the dedicated SDE bus. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the even bank. LDE [63:32] are connected directly to the LD [32:0] pins of the upper 32 bit even data RAM module. LDE [31:0] are connected directly to the LD [32:0] pins of the lower 32 bit even data RAM module.

LPE [3:0] (Load Parity Even) Active High Input

Load parity bus between the R8010 FPU and the even bank of the streaming cache. This bus is unidirectional and accepts load parity data only. Even bank store data is transferred via the dedicated store parity bus. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the even bank. LPE [3:2] are connected directly to the LP [1:0] pins of the upper 32 bit even data RAM module. LPE [1:0] are connected directly to the LP [1:0] pins of the lower 32 bit even data RAM module. Parity is checked and generated in 16-bit quantities. Hence only the lower two parity bits (LP [1:0]) are currently used. On both modules LP [3:2] of the SIM module connector are not used and should be left unconnected.

SDE [63:0] (Store Data Even) Active High I/O

Store data bus between the R8010 FPU, the even bank of the streaming cache, and the external data buffers. The bus itself is bidirectional, although it is unidirectional from the R8010 FPU's point of view. When the R8010 FPU is in control the bus becomes unidirectional and drives store output data only. Even bank load data is transferred via the dedicated LDE bus. There are three basic uses of the store data bus; Floating Point stores, Integer stores, and streaming cache data transfers to/from main memory. On Floating Point stores the R8000 instructs the R8010 FPU to drive data from a register file out onto the store data bus. Address information for the store is provided by the R8000 via a dedicated address bus between the R8000 CPU and the even bank of the streaming

cache. Integer Store data and alignment information is transferred via the TBus to the R8010 FPU. Control signals from the R8000 allow the R8010 FPU to pass the data from the TBus to the SDE pins. When data from the streaming cache is to be transferred to main memory, the Cache Controller initiates a streaming cache load. The load data is retrieved from the cache using the LDE pins and transferred to the R8010 FPU. The CC controls the flow of data through the R8010 FPU asserting the FOE_ and BYPASS_ signals to the R8010 FPU, allowing data to pass internally from the LDE pins to the SDE pins and out to the main memory data buffers. When the CC is in control of the bus during a cache line fill, the SDE [63:0] bus is driven by the external data buffers. Since data from main memory is written to the streaming cache using the SDE pins, the SDE pins on the R8010 FPU side are tri-stated and the data is witten to the streaming cache. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the even bank. SDE [63:32] are connected directly to the SD [32:0] pins of the upper 32 bit even data RAM module. SDE [31:0] are connected directly to the SD [32:0] pins of the lower 32 bit even data RAM module.

SPE [3:0] (Store Parity Even) Active High Input

Store parity bus between the R8010 FPU and the even bank of the streaming cache. This bus is unidirectional and transfers store parity data only. Even bank load data is transferred via the dedicated load parity bus. There are two 32 bit wide SIM modules on the even bank. SPE [3:2] are connected directly to the SP [1:0] pins of the upper 32 bit even data RAM module. SPE [1:0] are connected directly to the SP [1:0] pins of the lower 32 bit even data RAM module. Parity is checked and generated in 16-bit quantities. Hence only the lower two parity bits (SP [1:0]) of each module are currently used. On both modules SP [3:2] should be left unconnected.

9.2.3 R8010 FPU to Odd Bank Streaming Cache

LDO [63:0] (Load Data Odd) Active High Input

Load data bus between the R8010 FPU and the odd bank of the streaming cache. This bus is unidirectional and accepts load input data only. Odd bank store data is transferred via the dedicated SDO bus. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the even bank. LDO [63:32] are connected directly to the LD [32:0] pins of the upper 32 bit odd data RAM module. LDO [31:0] are connected directly to the LD [32:0] pins of the lower 32 bit odd data RAM module.

LPO [3:0] (Load Parity Odd) Active High Input

Load parity bus between the R8010 FPU and the odd bank of the streaming cache. This bus is unidirectional and accepts load parity data only. Odd bank store data is transferred via the dedicated store parity bus. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the odd bank. LPO [3:2] are connected directly to the LP [1:0] pins of the upper 32 bit odd data RAM module. LPO [1:0] are connected directly to the LP [1:0] pins of the lower 32 bit odd data RAM module. Parity is checked and generated in 16-bit quantities. Hence only the lower two parity bits (LP [1:0]) are currently used. On both modules LP [3:2] should be left unconnected.

SDO [63:0] (Store Data Odd) Active High I/O

Store data bus between the R8010 FPU, the odd bank of the streaming cache, and the external data buffers. The bus itself is bidirectional, although it is unidirectional from the R8010 FPU's point of view. When the R8010 FPU is in control the bus becomes unidirectional and drives store output data only. Odd bank load data is transferred via the dedicated LDO bus. There are three basic uses of the store data bus; Floating Point stores, Integer stores, and streaming cache data transfers to/from main memory. On Floating Point stores the R8000 Microprocessor instructs the R8010 FPU to drive data from a register file out onto the store data bus. Address information for the store is provided by the R8000 via a dedicated address bus between the R8000 Microprocessor and the odd bank of the streaming cache. Integer Store data and alignment information is transferred via the TBus to the R8010 FPU. Control signals from the R8000 CPU allow the R8010 FPU to pass the data from the TBus to the SDO pins. When data from the streaming cache is to be transferred to main memory, the Cache Controller initiates a streaming cache load. The load data is retrieved from the cache using the LDO pins and transferred to the R8010 FPU. The CC controls the flow of data through the R8010 FPU asserting the FOE_ and BYPASS_ signals to the R8010 FPU, allowing data to pass internally from the LDO pins to the SDO pins and out to the main memory data buffers. When the CC is in control of the bus during a cache line fill, the SDO [63:0] bus is driven by the external data buffers. Since data from main memory is written to the streaming cache using the SDO pins, the SDO pins on the R8010 FPU side are tri-stated and the data is witten to the streaming cache. In a 4 Megabyte implementation there are two 32 bit wide SIM modules on the odd bank. SDO [63:32] are connected directly to the SD [32:0] pins of the upper 32 bit odd data RAM module. SDO [31:0] are connected directly to the SD [32:0] pins of the lower 32 bit odd data RAM module.

SPO [3:0] (Store Parity Odd) Active High Input

Store parity bus between the R8010 FPU and the odd bank of the streaming cache. This bus is unidirectional and transfers store parity data only. Odd bank load data is transferred via the dedicated load parity bus. There are two 32 bit wide SIM modules on the odd bank. SPO [3:2] are connected directly to the SP [1:0] pins of the upper 32 bit odd data RAM module. SPO [1:0] are connected directly to the SP [1:0] pins of the lower 32 bit odd data RAM module. Parity is checked and generated in 16-bit quantities. Hence

only the lower two parity bits (SP [1:0]) of each module are currently used. On both modules SP [3:2] should be left unconnected.

9.2.4 R8010 FPU to Cache Controller

BYPASS (Floating Point Bypass) Active Low Input

Assertion of BYPASS_ allows data from the load data pins to bypass the internal circuits of the R8010 FPU and be transferred directly to the store data output buffers. The assertion of FOE_ enables the buffers and allows the streaming cache load data to be driven out onto the store data pins. Bypass deasserted allows data from within the registers of the R8010 FPU to be driven out onto the SDE or SDO pins (assuming FOE_ is active).

FOE_ (Floating Point Output Enable) Active Low Input

Assertion of FOE_ and FBYPASS_ on streaming cache transfers to main memory enables the output drivers of the R8010 FPU which allow data to be driven directly from the LDE [63:0]/LDO [63:0] pins to the SDE [63:0]/SDO [63:0] pins of the R8010 FPU respectively. When data from the streaming cache is to be transferred to main memory, the Cache Controller initiates a streaming cache load. The load data is then transferred on the LDE or LDO pins to the R8010 FPU. The CC controls the flow of data through the R8010 FPU by asserting the FOE_ and BYPASS_ signals to the R8010 FPU, allowing data to be driven onto the SDE or SDO busses. When data is to be transferred from main memory to the streaming cache, the CC deasserts FOE_, tri-stating the store data busses (SDE/SDO) of the R8010 FPU.

TBUS [71:0] (TBus interface) Active High I/O

TBus [71:0] are connected between the CC, R8000, and R8010 FPU. There is no communication protocol between the R8010 FPU and the CC. The TBus of the R8010 FPU and the CC are connected only because the R8000 must communicate with both. Refer to section 9.1.1 for more information on the TBus.

9.2.5 R8010 FPU Clock Interface

CLK (Reference Clock) Active High Input

Master input clock to the Phase Lock Loop (PLL) circuitry of the R8010 FPU. The output of the PLL is then used as the master clock for the chip. CLK is normally connected directly to the ouput of the clock driver. In most cases it is desirable to use the PLL circuitry, but for those applications which to not wish to use the PLL, the clock drivers should be connected to EXT_CLK.

EXT_CLK (External Clock) Active High Input

The EXT_CLK input allows the system designer to bypass the internal PLL of the R8010 FPU and drive the chip directly from the system clock. EXT_CLK should be tied to ground through a 330 ohm resistor.

GND_PLL (Ground Phase Lock Loop)

Ground source for the phase lock loop circuitry. GND_PLL can be connected to VCC_PLL through .1 microfarad and .015 microfarad capacitors in parallel.

LPF_OUT (Low Pass Filter Output) Active High Output

LFP_OUT is a special pin used to test the PLL circuitry during component test for monitoring the status of the low-pass filter. LPF_OUT must be connected to VCC_PLL through a 680K ohm resistor.

SYNC_IN (Synchronized PLL input) Active High Input

SYNC_IN is part of the PLL feedback path and must be connected to SYNC_OUT in order for the PLL circuitry to work correctly. The pins are made available externally to allow the user to manually alter the phase of the PLL.

SYNC_OUT (Synchronized PLL input) Active High Output

SYNC_OUT is part of the PLL feedback path and must be connected to SYNC_IN in order for the PLL circuitry to work correctly. The pins are made available externally to allow the user to manually alter the phase of the PLL.

VCC_PLL (Voltage Phase Lock Loop)

Voltage source for the phase lock loop circuitry. Connected to a regulated 3.3 volt source. VCC_PLL can be connected to GND_PLL through .1 and .015 microfarad capacitors in parallel.

9.2.6 JTAG Interface

The JTAG interface of the R8010 FPU is identical to that of the R8000 Microprocessor. All of the pins have the same function and characteristics. Below is a listing of the JTAG pins which interface to the R8010 FPU. For a description of these pins, refer to the JTAG pin description in section 9.1.8.

JTAG_TCK (JTAG Test Clock) Active High Input
JTAG_TMS (JTAG Test Mode Select) Active High Input
JTAG_TDI (JTAG Test Data Input) Active High Input
JTAG_TDO (JTAG Test Data Output) Active High Output
JTAG_TRST_ (JTAG Test Reset) Active High Input
SDI (Serial Data In) Active High Input
SDO (Serial Data Out) Active High Output

9.2.7 Initialization Interface Signals

RESET

Reset Pin of the R8010 FPU. The signal should be asserted for 4096 clocks. This count is easily accomplished with a 12-bit counter. Refer to the reset description of the R8000 CPU in section 9.1.9.

9.3 EVEN TAG RAM UNIT SIGNAL DESCRIPTIONS

Two identical Tag RAM's are required in the R8000 Microprocessor environment for support of the 2-way interleaved second level streaming cache. The even Tag RAM maintains address, state and virtual synonym information for the even bank of streaming cache data and the odd Tag RAM maintains address, state, and virtual

synonym information for the odd bank of streaming cache data.

The Even Tag RAM Unit is a 155 pin device which interfaces to the R8000 Microprocessor, the Cache Controller, and the even bank of streaming cache. The following sections define the external pinout of the Tag RAM and are divided into specific component interfaces. Figure 9.4 shows the functional pin groupings of the Even Tag RAM.

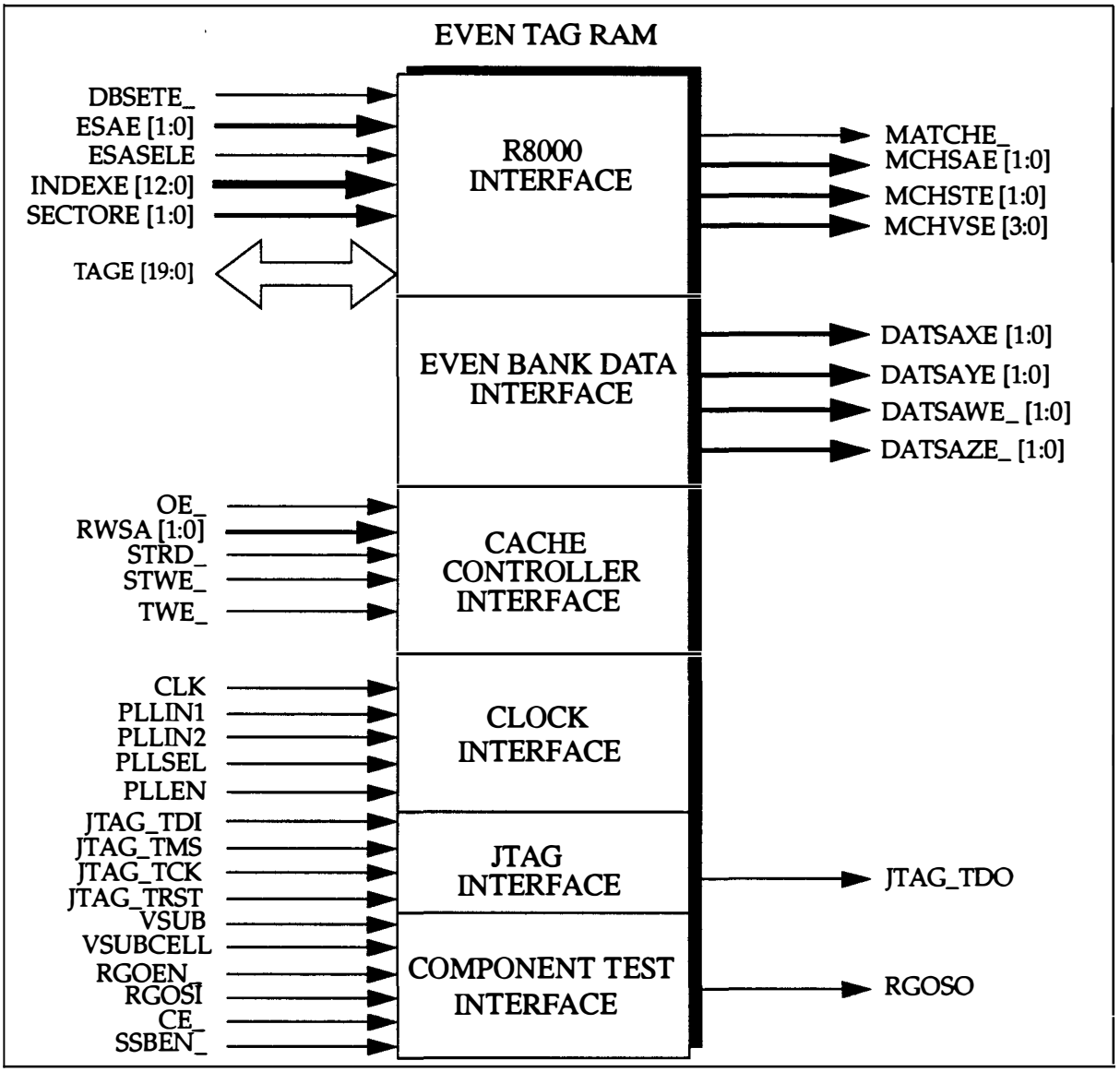


Figure 9-4 Even Tag RAM Signal Groupings

Table 9-3 shows a pin summary of the even Tag RAM in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
DATSAWE_ [1:0]	Data Set Address Even	Low	Even Bank Address
DATSAXE [1:0]	Data Set Address Even	High	Even Bank Address
DATSAYE [1:0]	Data Set Address Even	High	Even Bank Address
DATSAZ_ [1:0]	Data Set Address Even	Low	Even Bank Address
JTAG_TDO	JTAG Test Data Out	High	External Source
MATCHE_	Address Match	Low	R8000 CPU
MCHSAE [1:0]	Match Set Address	High	R8000 CPU
MCHSTE [1:0]	Match State	High	R8000 CPU
MCHVSE [3:0]	Match Virtual Synonym	High	R8000 CPU
RGOSO	Ring Oscillator Output	High	External Source
Input Pins			
CE_	Chip Enable	Low	External source
CLK	Reference Clock	High	External Source
DBSETE_	Even Dirty Bit Set	Low	R8000 CPU
ESAE [1:0]	Even External Set Address	High	R8000 CPU
ESASELE	External Set Address Select	High	R8000 CPU
INDEXE [12:0]	Even Tag RAM Index	High	R8000 CPU
JTAG_TCK	JTAG Clock	High	External Source
JTAG_TDI	JTAG Test Data In	High	External Source
JTAG_TMS	JTAG Test Mode Select	High	External Source
JTAG_TRST	JTAG Test Reset	Low	External Source
OE_	Output Enable	Low	Cache Controller
PLLEN	Phase Lock Loop Enable	High	External Source
PLLIN1	Phase Lock Loop Input	High	External Source
PLLIN2	Phase Lock Loop Input	High	External Source
PLLSEL	Phase Lock Loop Select	High	External Source
RGOEN_	Ring Oscillator Enable	Low	External Source
RGOSI	Ring Oscliiator Input	High	External Source
RWSA [1:0]	Read Write Set Address	High	Cache Controller
SECTORE [1:0]	Even Sector Address	High	R8000 CPU
SSBEN_	Self Sub-Bias Enable	Low	External Source

Table 9-3 Even Tag RAM Pin Summary

Pin ID	Pin Name	Active Level	Connects To
STRD_	State Read	Low	Cache Controller
STWE_	State Write Enable	Low	Cache Controller
TWE_	Tag Write Enable	Low	Cache Controller
VSUB	Self Sub-Bias Voltage Input	High	External Source
VSUBCELL	Self Sub-Bias Voltage Input	High	External Source
Input/Output Pins			
TAGE [19:0]	Even Tag Address	High	R8000 CPU

Table 9-3 Even Tag RAM Pin Summary

9.3.1 Even Tag RAM to R8000 Microprocessor

The following signals connect to the R8000 and are explained further in section 9.1.2.

DBSETE_ (Even Dirty Bit Set) Active Low Input

The dirty bit RAM is conditionally written based on the state of DBSETE_. Refer to the DBSETE_ pin description in section 9.1.2 for more information.

ESAE [1:0] (External Set Address Even) Active High Input

These pins are connected directly to the ESAE [1:0] pins of the R8000 CPU. Refer to the ESAE pin description in section 9.1.2 for more information.

ESASELE (External Set Address Select Even) Active High Input

The ESASEL pin acts as a mux select to drive the ESA [1:0] bits through to the DATSA [1:0] bits. Refer to the ESASELE pin description in section 9.1.2 for more information.

INDEXE [16:0] (Tag RAM Index Even) Active High Input

These pins form the index into the Tag RAM and are connected to the INDEXE pins of the R8000 Microprocessor. Refer to the INDEXE pin descriptions in section 9.1.2 for connectivity information.

MATCHE_ Active Low Output

This signal is an output of the even Tag RAM. Assertion of this signal by the tag RAM

indicates that the address on the bus compared to one of the four sets. MATCH_ is connected directly to the MATCHE_ pin of the R8000 CPU. Refer to the MATCHE_ pin description in section 9.1.2 for more information.MCHSAE [1:0] (Match Set Address Even) Active High Output

On a lookup cycle, these 2 bits are used form an encoded value of which of the four ways in the Tag RAM was a hit. Refer to the MCHSAE pin descriptions in section 9.1.2 for more information.

MCHSTE [1:0] (Match State Even) Active High Output

These 2 bits encode the state information of a given 128 byte sector in the data RAM's. The state can be shared, exclusive, or invalid. Refer to the MCHSTE [1:0] pin descriptions in section 9.1.2 for more information.

MCHVSE [3:0] (Match Virtual Synonym Even) Active High Output

The bits correspond to virtual address bits 15:12 and form the virtual synonym entry in the Tag RAM. Refer to the MCHVSE [3:0] pin descriptions in section 9.1.2 for more information.

SECTOR [1:0] (Tag RAM Sector Address) Active High Input

These two pins act as a 4:1 mux selector inside the Tag RAM. Each line of the Data RAM contains 4 sectors, with each sector containing sixteen 64 bit words, 8 in the odd bank and 8 in the even bank. The SECTOR [1:0] pins are used to select the correct state information for the sector to be accessed. Refer to the INDEXE pin descriptions in section 9.1.2 for more infomation.

TAGE [19:0] (Tag RAM Tag Address Even) Active High I/O

These pins form the tag address of the tag RAM. The TAG pins are written and read by the CC. The R8000 CPU uses TAG [19:0] for lookup cycles. Reads and writes from the CC are passed onto the TAG pins through the TBus. Refer to the TAGE pin descriptions in section 9.1.2 for connectivity information.

9.3.2 Even Tag RAM to Even Bank Streaming Cache

DATSAXE [1:0] (Data Set Address X Even) Active High Output
DATSAYE [1:0] (Data Set Address Y Even) Active High Output
DATSAWE_ [1:0] (Data Set Address W Even) Active Low Output
DATSAZE_ [1:0] (Data Set Address Z Even) Active Low Output

These two bits form the upper two address bits for the even bank of the streaming cache. Each of the four sets of DATSA pins contains the exact same information and are buffered versions of one another. Two sets are inverted.

9.3.3 Tag RAM to Cache Controller

The Cache Controller does not distinguish between even and odd Tag RAM's. Each interface pin of the CC goes to both the even and odd Tag RAM's.

TOE_ (Tag Output Enable) Active Low Input

The TOE_ pin is driven by the CC when the CC is reading the Tag RAM. Assertion of TOE_ allows the information currently on the 20 bit Tag Address bus to be driven back to the R8000 Microprocessor where it is returned to the CC via the TBus. Either tag address or state and virtual synonym information is driven out onto the tag bus based on the state of STRD_ (see below).

RWSA [1:0] (Read Write Set Address) Active High Input

This two bit field is driven by the CC when the Cache Controller is reading or writing to the Tag RAM and indicates which ways of the four-way set associative Tag RAM is to be read or written. Normally the CC modifies the Tag RAM based on the result of a previous R8000 lookup cycle. The result of which of the four ways compared during the lookup is driven onto the MCHSA [1:0] pins of the tag RAM and returned to the R8000. The R8000 CPU in turn passes this encoded information to the CC via a field of the TBus. When the CC actually modifies the Tag RAM, the way to be modified is selected using the RWSA [1:0] pins. The Dirty Bit RAM also uses these pins to update the correct dirty bit entry.

STRD_ (State Read) Active Low Input

STRD_ is a control pin used in the Cache Controller read logic block inside the Tag RAM and is used to drive either State and V.S. information, or tag address information onto

the bi-directional tag pins. At the same time the TOE_ pin must be asserted to enable this information onto the Tag address bus. Which information is driven out onto the tag pins is determined by STRD_. The information on the tag bus enters the R8000 and is returned to the CC via a field of the TBus.

STWE_ (State Write Enable) Active Low Input

STWE_ is asserted whenever the CC is writing state and Virtual Synonym information to the Tag RAM. Both the 20 bit tag address and the corresponding 12 bits of state and V.S. information are carried to the Tag RAM via the tag address bus. The CC tracks the information on the pins and assertes STWE_ if state and V.S. information is to be written. TWE_ is asserted if tag address information is to be written.

TWE_ (Tag Write Enable) Active Low Input

TWE_ is driven by the CC whenever tag address information is to be written to the Tag RAM. Refer to STWE_ above.

9.3.4 Tag RAM Clock Interface

The following signals comprise the clock interface for the even Tag RAM.

CLK (Clock Input) Active high Input

Clock input for the Tag RAM. The input frequency is 75 MHz. The clock can drive the device directly, or function as an input to a phase lock loop based on the state of the input pin PLLEN.

PLLEN (Phase Lock Loop Enable) Active High Input

Enable pin for the bi-polar phase lock loop. Assertion of PLLEN enables the phase lock loop. Deassertion of this pin disables the phase lock loop and allows the input clock to drive the device directly.

PLLIN1 (Phase Lock Loop 1) Active High Input

There are two phase lock loop circuits on-chip. PLLIN1 is the 75 MHz clock input for the CMOS phase lock loop. This pin should be tied to ground for normal PLL operation.

PLLIN2 (Phase Lock Loop 1) Active High Input

PLLIN2 is the 75 MHz clock input for the Bi-Polar phase lock loop.This pin should be tied high for normal PLL operation.

PLLSEL (Phase Lock Loop Select) Active High Input

there are two types of phase lock loop circuits inside the Tag RAM, one of which is used for testing purposes. PLLSEL must be tied high for proper operation of the device.

9.3.5 JTAG Interface

The JTAG interface of the Tag RAM is identical to that of the R8000 and R8010 FPU. All of the pins have the same function and characteristics. Below is a listing of the JTAG pins which interface to the Even Tag RAM. For a description of these pins, refer to the JTAG pin description in section 9.1.8.

JTAG_TCK (JTAG Test Clock) Active High Input
JTAG_TMS (JTAG Test Mode Select) Active High Input
JTAG_TDI (JTAG Test Data Input) Active High Input
JTAG_TDO (JTAG Test Data Output) Active High Output
JTAG_TRST_ (JTAG Test Reset) Active High Input

9.3.6 Component Test Interface

The following signals comprise the component test interface to the Tag RAM. These signals are listed only for completeness and have no functional purpose other than form device testing. It is important that the user connect these pins in the manner specified in the pin descriptions below.

CE_ (Chip Enable) Active Low Input

Chip Enable for the test logic. This pin is NOT a master chip enable for the device. It simply enables certain testing functions which are performed during component verification and testing. CE_ must be tied to ground through a 330 ohm resistor.

RGOEN_ (Ring Oscillator Enable) Active Low Input

The ring oscillator is part of the clock circuitry of the Tag RAM. RGOEN_ is used to observe the behavior during certain stages of the oscillator during the component test and verification porocess. For normal operation RGOEN_ must be tied to ground through a 330 ohm resistor.

RGOSI (Ring Oscillator Input) Active High Input
RGOSO (Ring Oscillator Output) Active High Output

These pins are used in conjunction with RGOEN_ during the component test and verification process. For normal operation RGOSI must be tied to ground through a 330 ohm resistor. RGOSO must be left unconnected.

SSBEN_ (Self-Sub-Bias Enable) Active Low Output

Enables the bias voltage inputs which are used during the component test and verification process to monitor current flow through the device. SSBEN_ must be tied to ground through a 330 ohm resistor.

VSUB (Bias Voltage Input) Active High Input
VSUBCELL (Bias Voltage Input) Active High Input

These two pins work in conjunction with SSBEN_ to monitor current flow through the device during the component test and verification process. Both of these pins must be left unconnected for normal operation.

9.4 ODD TAG RAM UNIT SIGNAL DESCRIPTIONS

Two identical Tag RAM's are required in the R8000 Microprocessor environment for support of the 2-way interleaved second level streaming cache. The odd Tag RAM maintains address, state and virtual synonym information for the odd bank of streaming cache data and the even Tag RAM maintains address, state, and virtual synonym information for the even bank of streaming cache data.

The Odd Tag RAM Unit is a 155 pin device which interfaces to the R8000, the Cache Controller, and the odd bank of streaming cache. The following sections define the external pinout of the Tag RAM and are divided into specific component interfaces. Figure 9-5 shows the functional pin groupings of the odd Tag RAM.

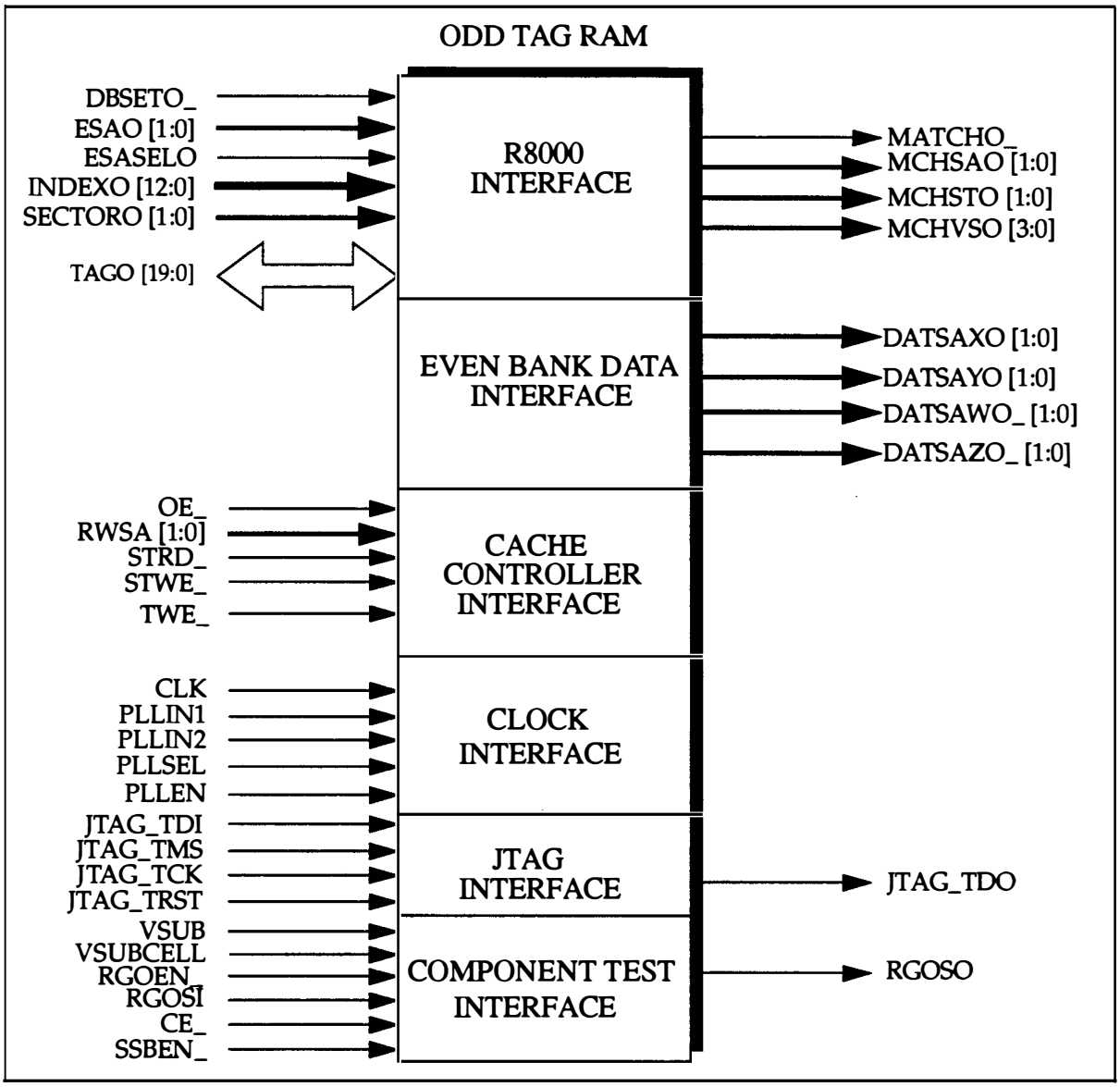


Figure 9-5 Odd Tag RAM Unit Signal Groupings

Table 9-4 shows a pin summary of the odd Tag RAM Unit in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
DATSAWO_ [1:0]	Data Set Address Odd	Low	Odd Bank Address
DATSAXO [1:0]	Data Set Address Odd	High	Odd Bank Address
DATSAYO [1:0]	Data Set Address Odd	High	Odd Bank Address
DATSAZO_ [1:0]	Data Set Address Odd	Low	Odd Bank Address
JTAG_TDO	JTAG Test Data Out	High	External Source
MATCHO_	Address Match	Low	R8000 CPU
MCHSAO [1:0]	Match Set Address	High	R8000 CPU
MCHSTO [1:0]	Match State	High	R8000 CPU
MCHVSO [3:0]	Match Virtual Synonym	High	R8000 CPU
RGOSO	Ring Oscillator Output	High	External Source
Input Pins			
CE_	Chip Enable	Low	External source
CLK	Reference Clock (Uses PLL)	High	External Source
DBSETO_	Odd Dirty Bit Set	Low	R8000 CPU
ESAO [1:0]	Odd External Set Address	High	R8000 CPU
ESASELO	External Set Address Select	High	R8000 CPU
INDEXO [12:0]	Odd Tag RAM Index	High	R8000 CPU
JTAG_TCK	JTAG Clock	High	External Source
JTAG_TDI	JTAG Test Data In	High	External Source
JTAG_TMS	JTAG Test Mode Select	High	External Source
JTAG_TRST	JTAG Test Reset	Low	External Source
OE_	Output Enable	Low	Cache Controller
PLLEN	Phase Lock Loop Enable	High	External Source
PLLIN1	Phase Lock Loop Input	High	External Source
PLLIN2	Phase Lock Loop Input	High	External Source
PLLSEL	Phase Lock Loop Select	High	External Source
RGOEN_	Ring Oscillator Enable	Low	External Source
RGOSO	Ring Oscillator Output	High	External Source
RWSA [1:0]	Read Write Set Address	High	Cache Controller
SECTORO [1:0]	Odd Sector Address	High	R8000 CPU
SSBEN_	Self Sub Bias Voltage Enable	Low	External Source
STRD_	State Read	Low	Cache Controller
STWE_	State Write Enable	Low	Cache Controller
TWE_	Tag Write Enable	Low	Cache Controller
VSUB	Bias Voltage Input	High	External Source

Table 9-4 Odd Tag RAM Pin Summary

Pin ID	Pin Name	Active Level	Connects To
VSUBCELL	Bias Voltage Input	High	External Source
Input/Output Pins			
TAGO [19:0]	Odd Tag Address	High	R8000 CPU

Table 9-4 Odd Tag RAM Pin Summary

9.4.1 Odd Tag RAM to R8000 Microprocessor

The following signals connect to the R8000 and are explained further in section 9.1.3.

DBSETO_ (Odd Dirty Bit Set) Active Low Input

The dirty bit RAM is written conditionally based on the state of DBSETO_. Refer to the DBSETO_ pin description in section 9.1.3 for more information.

ESAO [1:0] (External Set Address Odd) Active High Input

These pins are connected directly to the ESAO [1:0] pins of the R8000. Refer to the ESAO pin description in section 9.1.3 for more information.

ESASELO (External Set Address Select Odd) Active High Input

The ESASEL pin acts as a mux select to drive the ESA [1:0] bits through to the DATSA [1:0] bits. Refer to the ESASELE pin description in section 9.1.3 for more information.

INDEXO [12:0] (Tag RAM Index Odd) Active High Input

These pins form the index into the Tag RAM and are connected to the INDEXO pins of the R8000. Refer to the INDEXO pin descriptions in section 9.1.3 for connectivity information.

MATCHO_ Active Low Output

This signal is an output from the odd Tag RAM. When the R8000 performs a lookup cycle to the tag RAM an index is supplied which corresponds to all four sets of the four way set associative odd tag RAM. Assertion of this signal by the tag RAM indicates that the address on the bus compared to one of the four sets. MATCHO_ is connected directly

to the MATCHO_ pin of the R8000 CPU.

MCHSAO [1:0] (Match Set Address Odd) Active High Output

On a lookup cycle, these 2 bits are used to form an encoded value indicating which of the four ways in the Tag RAM was a hit. Refer to the MCHSAE pin descriptions in section 9.1.3 for more information.

MCHSTO [1:0] (Match State Odd) Active High Output

These 2 bits encode the state information of a given 128 byte sector in the data RAM's. The state can be shared, exclusive, or invalid. Refer to the MCHSTE [1:0] pin descriptions in section 9.1.3 for more information.

MCHVSO [3:0] (Match Virtual Synonym Odd) Active High Output

The bits correspond to virtual address bits [15:12] and form the virtual synonym entry in the Tag RAM. Refer to the MCHVSE [3:0] pin descriptions in section 9.1.3 for more information.

SECTORO [1:0] (Tag RAM Sector Address Odd) Active High Input

These two pins act as a 4:1 mux selector inside the Tag RAM. Each line of the Data RAM contains 4 sectors, with each sector containing sixteen 64 bit words, 8 in the odd bank and 8 in the even bank. The SECTORO [1:0] pins are used to select the correct state information for the sector to be accessed. Refer to the INDEXO pin descriptions in section 9.1.3 for more infomation.

TAGO [19:0] (Tag RAM Tag Address Odd) Active High I/O

These pins form the tag address of the tag RAM. The TAGO pins are written and read by the CC. The R8000 uses TAGO [19:0] for lookup cycles. Reads and writes from the CC are passed onto the TAGO pins through the TBus. Refer to the TAGO pin descriptions in section 9.1.3 for connectivity information.

9.4.2 Odd Tag RAM to Odd Bank Streaming Cache

DATSAXO [1:0] (Data Set Address X Odd) Active High Output
DATSAYO [1:0] (Data Set Address Y Odd) Active High Output

DATSAWO_ [1:0] (Data Set Address W Odd) Active Low Output
DATSAZO_ [1:0] (Data Set Address Z Odd) Active Low Output

These two bits form the upper two address bits for the odd bank of the streaming cache. Each of the four sets of DATSA pins contains the exact same information and are buffered versions of one another. Two sets are inverted.

9.4.3 Odd Tag RAM to Cache Controller

The Cache Controller does not distinguish between even and odd Tag RAM's. Each interface pin of the CC goes to both the even and odd Tag RAM's. Below is a list of signals which constitute the Tag RAM to Cache Controller interface. Refer to section 9.3.3, Tag RAM to Cache Controller, for pin defintions of the cache controller interface.

TOE_ (Tag Output Enable) Active Low Input
RWSA [1:0] (Read Write Set Address) Active High Input
STRD_ (State Read) Active Low Input
STWE_ (State Write Enable) Active Low Input
TWE_ (Tag Write Enable) Active Low Input

9.4.4 Tag RAM Clock Interface

The following signals comprise the clock interface for the odd Tag RAM. These signals are identical in functionality and characteristics to those of the even Tag RAM. Refer to clock interface in section 9.3.4 for the definitions of these pins.

CLK (Clock Input) Active high Input
PLLEN (Phase Lock Loop Enable) Active High Input
PLLIN1 (Phase Lock Loop 1) Active High Input
PLLIN2 (Phase Lock Loop 1) Active High Input
PLLSEL (Phase Lock Loop Select) Active High Input

9.4.5 JTAG Interface

The JTAG interface of the Tag RAM is identical to that of the R8010 FPU and R8000 CPU. All of the pins have the same function and characteristics. Below is a listing of the JTAG pins which interface to the tag RAM's.

JTAG_TCK (JTAG Test Clock) Active High Input
JTAG_TMS (JTAG Test Mode Select) Active High Input
JTAG_TDI (JTAG Test Data Input) Active High Input
JTAG_TDO (JTAG Test Data Output) Active High Output
JTAG_TRST_ (JTAG Test Reset) Active High Input

9.4.6 Component Test Interface

The following signals comprise the initialization interface to the odd Tag RAM. The functionality of these pins is identical to the even Tag RAM. Refer to section 9.3.6 for a description of these pins.

CE_ (Test Logic Chip Enable) Active Low Input
RGOEN_ (Ring Oscillator Enable) Active High Input
RGOSI (Ring Oscillator Input) Active High Input
RGOSO (Ring Oscillator Output) Active High Output
SSBEN_ (Self Sub-Bias Voltage Enable) Active High Output
VSUB (Sub-Bias Voltage Input) Active High Output
VSUBCELL (Sub-Bias Voltage Input) Active High Output

9.5 EVEN BANK STREAMING CACHE PIN DESCRIPTIONS

The even bank of Data RAM's interfaces to the even Tag RAM. The odd bank of Data RAM's interfaces to the odd Tag RAM. Both banks interface to the R8000 CPU and the R8010 FPU. The following sections define the external pinout of the even and odd data RAM's and are divided into specific component interfaces. Note that the even bank contains twelve clock inputs. Each line is an individual output of the clock driver circuitry. The even bank contains 24 devices between the two modules, hence no clock line is required to drive more than two devices. Figure 9-6 shows the functional pin groupings of the Even Data RAM bank.

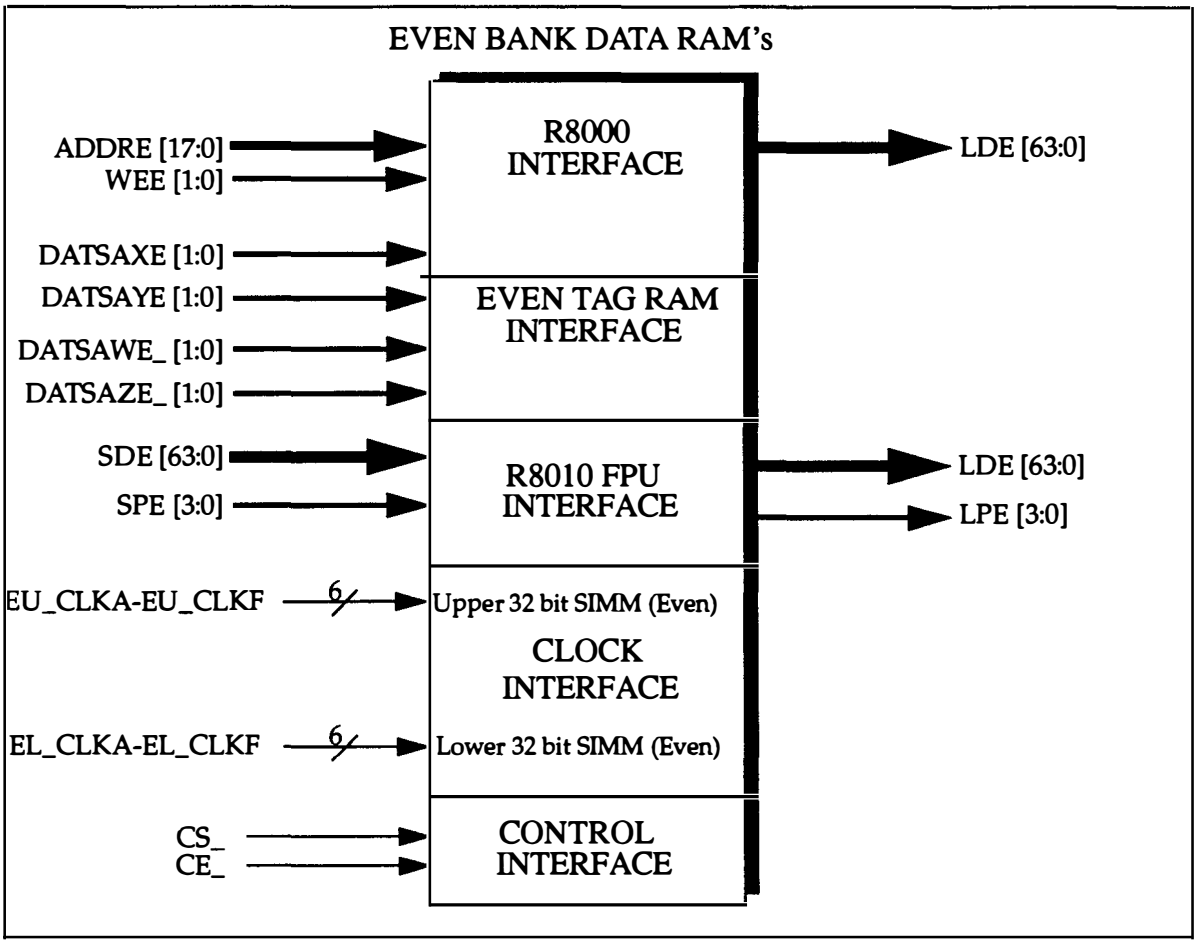


Figure 9-6 Even Bank Data RAM Signal Groupings

Table 9-5 shows a pin summary of the even data RAM bank in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
LDE [63:0]	Load Data Even	High	R8000/R8010
LPE [3:0]	Load Parity Even	High	R8000 CPU
Input Pins			
ADDRE [17:0]	Even Bank Address	High	R8000 CPU
CE_	Chip Enable	Low	External source
CS_	Chip Select	Low	External Source
DATSAWE_ [1:0]	Data Set Address Odd	Low	Even Tag RAM

Table 9-5 Even Bank Data RAM Pin Summary

Pin ID	Pin Name	Active Level	Connects To
DATSAXE [1:0]	Data Set Address Odd	High	Even Tag RAM
DATSAYE [1:0]	Data Set Address Odd	High	Even Tag RAM
DATSAZE_ [1:0]	Data Set Address Odd	Low	Even Tag RAM
EU_CLKA	Even Upper Clock A	High	External Source
EU_CLKB	Even Upper Clock B	High	External Source
EU_CLKC	Even Upper Clock C	High	External Source
EU_CLKD	Even Upper Clock D	High	External Source
EU_CLKE	Even Upper Clock E	High	External Source
EU_CLKF	Even Upper Clock F	High	External Source
EL_CLKA	Even Lower Clock A	High	External Source
EL_CLKB	Even Lower Clock B	High	External Source
EL_CLKC	Even Lower Clock C	High	External Source
EL_CLKD	Even Lower Clock D	High	External Source
EL_CLKE	Even Lower Clock E	High	External Source
EL_CLKF	Even Lower Clock F	High	External Source
SDE [63:0]	Store Data Even	High	R8010 FPU
SPE [3:0]	Store Parity Even	High	R8010 FPU
WEE_ [1:0]	Even Bank Write Enables	Low	R8000 CPU

Table 9-5 Even Bank Data RAM Pin Summary

9.5.1 Even Bank Streaming Cache to R8000 Microprocessor

ADDRE [17:0] (Address Even) Active High Input

These pins form the address to the even bank of the streaming cache. Bits 17:0 of the R8000 connect directly to address pins 17:0 of both SIM modules of the even bank . Refer to section 9.1.4 for more information on ADDRE [17:0].

LDE [63:0] (Load Data Even) Active High Output

Load data bus between the R8000 and the even bank of the streaming cache. This bus is unidirectional and drives only load input data to the R8000. Refer to section 9.1.4 for more information on LDE [63:0].

WEE_ [1:0] (Even Bank Write Enable) Active Low Input

The data RAM's can be written either by the R8000 or by the CC. The CC writes to the

data RAM's through the TBus. The R8000 Microprocessor provides a total of four write enables, two for the odd bank and two for the even bank. Streaming cache designs must allow for 32 bit writeability. WEE_ [1] is connnected to the WE_ pin of the upper 32 bits of data for the even bank. WEE_ [0] is connnected to the WE_ pin of the lower 32 bits of data for the even bank. Refer to section 9.1.4 for more information on WEE_ [1:0].

9.5.2 Even Bank Streaming Cache to Even Tag RAM

DATSAXE [1:0] (Data Set Address X Even) Active High Input
DATSAYE [1:0] (Data Set Address Y Even) Active High Input
DATSAWE_ [1:0] (Data Set Address W Even) Active Low Input
DATSAZE_ [1:0] (Data Set Address Z Even) Active Low Input

These two bits form the upper two address bits for the Even bank of Data RAM's. Each of the four sets of DATSA pins contains the exact same information and are buffered versions of one another. Two of the sets are inverted.

9.5.3 Even Bank Streaming Cache to R8010 FPU

LDE [63:0] (Even Bank Load Data) Active High Output

The LD [63:0] pins of the even bank of the streaming cache connect directly to the LDE [63:0] busses of both the R8000 CPU and the R8010 FPU. Refer to section 9.1.4 for more information on LDE [63:0].

LPE [3:0] (Even Bank Load Parity) Active High Input

Load parity bus between the R8010 FPU and the even bank of the streaming cache. This bus is unidirectional and drives load parity data only. Refer to section 9.2.2 for more information on LPE [3:0].

SDE [63:0] (Store Data) Active High Output

The SD [63:0] pins of the even bank of the streaming cache connect directly to the SDE [63:0] pins of the R8010 FPU. There is no direct store data interface between the R8000 CPU and the streaming cache data RAM's. Integer stores are handled through the R8010 FPU via the TBus. Refer to section 9.2.2 for more information on SDE [63:0].

SPE [3:0] (Store Parity Even) Active High Input

Store parity bus between the R8010 FPU and the even bank of the streaming cache. This bus is unidirectional and accepts store parity data only. Even bank load data is transferred via the dedicated load parity bus. Refer to section 9.2.2 for more information on SPE [3:0].

9.5.4 Even Bank Clock Interface

The clock interface to the even bank of Data RAM's consists of twelve separate clocks. Six clocks are used for the upper 32-bit module and six clocks for the lower 32-bit module. Each module contains 12 devices; eight 256K X 4 Data RAM's, one 256K X 4 Parity RAM, and three address buffers. Each clock drives two devices.

EU_CLKA (Even Upper Clock A) Active High Input
EU_CLKB (Even Upper Clock B) Active High Input
EU_CLKC (Even Upper Clock C) Active High Input
EU_CLKD (Even Upper Clock D) Active High Input
EU_CLKE (Even Upper Clock E) Active High Input
EU_CLKF (Even Upper Clock F) Active High Input

EL_CLKA (Even Lower Clock A) Active High Input
EL_CLKB (Even Lower Clock B) Active High Input
EL_CLKC (Even Lower Clock C) Active High Input
EL_CLKD (Even Lower Clock D) Active High Input
EL_CLKE (Even Lower Clock E) Active High Input
EL_CLKF (Even Lower Clock F) Active High Input

9.5.5 Even Bank Control Interface

The control interface for the even bank of streaming cache Data RAM's consists of a chip select and an output enable. Because the R8000 CPU and R8010 FPU provide dedicated load and store data busses for both the even and odd banks, no multiplexing of data is necessary. In addition, these busses interface only to certain devices within the system. This allows the OE_ pin to be permenantly asserted by tying it to ground. In addition, the CS_ pin can also remain permenantly asserted by tying it to ground.

CS_ (Chip Select) Active Low Input

Chip Select for the module. The individual chip selects for each RAM are tied together and routed to the module connector. CS_ should be tied to ground through a 330 ohm resistor.

OE_ (Output Enable) Active Low Input

Output enable for the module. Assertion of CE_ allows the RAM to drive data onto the load data bus. The individual output enables for each RAM are tied together and routed to the module connector. OE_ should be tied to ground through a 330 ohm resistor.

9.6 ODD BANK STREAMING CACHE PIN DESCRIPTIONS

The following sections define the external pinout of the odd bank of streaming cache data RAM's and are divided into specific component interfaces. The following sections define the external pinout of the even and odd data RAM's and are divided into specific component interfaces. Note that the even bank contains twelve clock inputs. Each line is an induvidual output of the clock driver circuitry. The even bank contains 24 devices between the two modules, hence no clock line is required to drive more than two devices

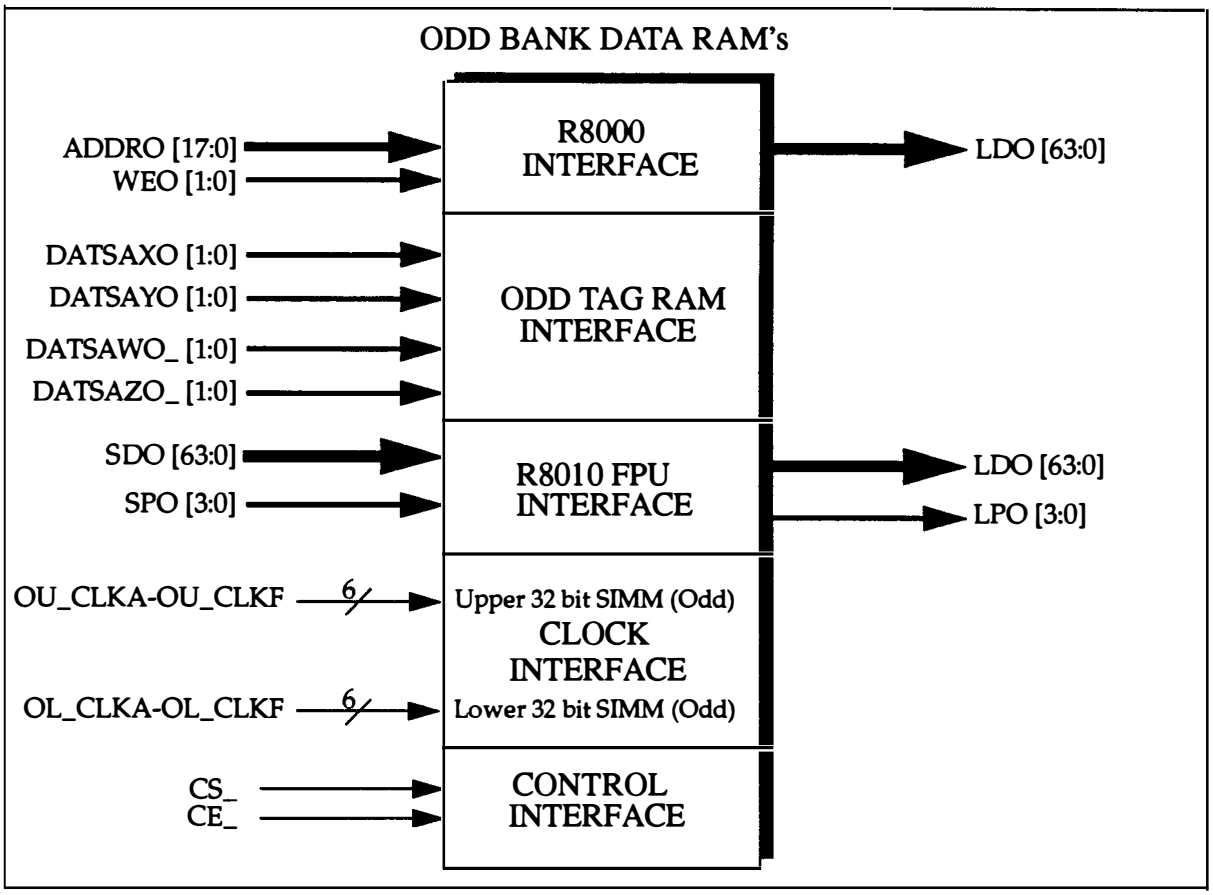


Figure 9-7 Odd Bank Data RAM Signal Groupings

Table 9-6 shows a pin summary of the odd data RAM bank in alphabetical order.

Pin ID	Pin Name	Active Level	Connects To
Output Pins			
LDO [63:0]	Odd Bank Load Data	High	R8000/R8010 FPU
LPO [3:0]	Odd Bank Load Parity	High	R8000 CPU
Input Pins			
ADDRO [17:0]	Odd Bank Address	High	R8000 CPU
CE_	Chip Enable	Low	External source
CS_	Chip Select	Low	External Source
DATSAWO_ [1:0]	Data Set Address Odd	Low	Odd Tag RAM
DATSAXO [1:0]	Data Set Address Odd	High	Odd Tag RAM
DATSAYO [1:0]	Data Set Address Odd	High	Odd Tag RAM

Table 9-6 Odd Bank Data RAM Pin Summary

Pin ID	Pin Name	Active Level	Connects To
DATSAZO_ [1:0]	Data Set Address Odd	Low	Odd Tag RAM
OU_CLKA	Odd Upper Clock A	High	External Source
OU_CLKB	Odd Upper Clock B	High	External Source
OU_CLKC	Odd Upper Clock C	High	External Source
OU_CLKD	Odd Upper Clock D	High	External Source
OU_CLKE	Odd Upper Clock E	High	External Source
OU_CLKF	Odd Upper Clock F	High	External Source
OL_CLKA	Odd Lower Clock A	High	External Source
OL_CLKB	Odd Lower Clock B	High	External Source
OL_CLKC	Odd Lower Clock C	High	External Source
OL_CLKD	Odd Lower Clock D	High	External Source
OL_CLKE	Odd Lower Clock E	High	External Source
OL_CLKF	Odd Lower Clock F	High	External Source
SDO [63:0]	Store Data Odd	High	R8010 FPU
SPO [3:0]	Store Parity Odd	High	R8010 FPU
WEO_ [1:0]	Odd Bank Write Enable	Low	R8000 CPU

Table 9-6 Odd Bank Data RAM Pin Summary

9.6.1 Odd Bank Streaming Cache to R8000 Microprocessor

ADDRO [17:0] (Address Odd) Active High Input

These pins form the address to the odd bank of the streaming cache. Bits 17:0 of the R8000 Microprocessor connect directly to address pins 17:0 of both SIM modules of the odd bank . Refer to section 9.1.5 for more information on ADDRO [17:0].

LDO [63:0] (Load Data Odd) Active High Output

Load data bus between the R8000 CPU and the odd bank of the streaming cache. This bus is unidirectional and drives only load input data to the R8000. Refer to section 9.1.5 for more information on LDO [63:0].

WEO_ [1:0] (Odd Bank Write Enable) Active Low Input

The data RAM's can be written either by the R8000 CPU or by the CC. The CC writes to

the data RAM's through the TBus. The R8000 Microprocessor provides a total of four write enables, two for the odd bank and two for the even bank. Streaming cache designs must allow for 32 bit writeability. WEO_ [1] is connnected to the WEO_ pin of the upper 32 bits of data for the odd bank. WEO_ [0] is connnected to the WEO_ pin of the lower 32 bits of data for the odd bank.

9.6.2 Odd Bank Streaming Cache to Odd Tag RAM

DATSAXO [1:0] (Data Set Address X Odd) Active High Input
DATSAYO [1:0] (Data Set Address Y Odd) Active High Input
DATSAWO_ [1:0] (Data Set Address W Odd) Active Low Input
DATSAZO_ [1:0] (Data Set Address Z Odd) Active Low Input

These two bits form the upper two address bits for the odd bank of Data RAM's. Each of the four sets of DATSA pins contains the exact same information and are buffered versions of one another. Two of the sets are inverted.

9.6.3 Odd Bank Streaming Cache to R8010 FPU

LDO [63:0] (Odd Bank Load Data) Active High Output

The LDO [63:0] pins of the odd bank of the streaming cache connect directly to the LDO [63:0] busses of both the R8000 CPU and the R8010 FPU. Refer to section 9.2.3 for more information on LDO [63:0].

LPO [3:0] (Odd Bank Load Parity) Active High Input

Load parity bus between the R8010 FPU and the odd bank of the streaming cache. This bus is unidirectional and drives load parity data only. Odd bank store parity data is transferred via the dedicated store parity bus. Refer to section 9.2.3 for more information on LPO [3:0].

SDO [63:0] (Store Data Odd) Active High Output

The SDO [63:0] pins of the odd bank of the streaming cache connect directly to the SDO [63:0] pins of the R8010 FPU. There is no direct store data interface between the R8000 CPU and the streaming cache data RAM's. Integer stores are handled through the R8010

FPU via the TBus. Refer to section 9.2.3 for more information on SDO [63:0].

SPO [3:0] (Store Parity Odd) Active High Input

Store parity bus between the R8010 FPU and the odd bank of the streaming cache. This bus is unidirectional and accepts store parity data only. Odd bank load parity data is transferred via the dedicated load parity bus. Refer to section 9.2.3 for more information on SPO [3:0].

9.6.4 Odd Bank Streaming Cache Clock Interface

The clock interface to the odd bank of Data RAM's consists of twelve separate clocks. Six clocks are used for the upper 32-bit module and six clocks for the lower 32-bit module. Each module contains 12 devices; eight 256K X 4 Data RAM's, one 256K X 4 Parity RAM, and three address buffers. Each clock drives two devices.

OU_CLKA (Even Upper Clock A) Active High Input
OU_CLKB (Even Upper Clock B) Active High Input
OU_CLKC (Even Upper Clock C) Active High Input
OU_CLKD (Even Upper Clock D) Active High Input
OU_CLKE (Even Upper Clock E) Active High Input
OU_CLKF (Even Upper Clock F) Active High Input

OL_CLKA (Even Lower Clock A) Active High Input
OL_CLKB (Even Lower Clock B) Active High Input
OL_CLKC (Even Lower Clock C) Active High Input
OL_CLKD (Even Lower Clock D) Active High Input
OL_CLKE (Even Lower Clock E) Active High Input
OL_CLKF (Even Lower Clock F) Active High Input

9.6.5 Odd Bank Streaming Cache Control Interface

The control interface for the odd bank of streaming cache Data RAM's consists of a chip select and an output enable. Because the R8000 CPU and R8010 FPU provide dedicated

load and store data busses for both the even and odd banks, no multiplexing of data is necessary. In addition, these busses interface only to certain devices within the system. This allows the OE_ pin to be permenantly asserted by tying it to ground. In addition, the CS_ pin can also remain permenantly asserted by tying it to ground.

CS_ (Chip Select) Active Low Input

Chip Select for the module. The individual chip selects for each RAM are tied together and routed to the module connector. CS_ should be tied to ground through a 330 ohm resistor.

OE_ (Output Enable) Active Low Input

Output enable for the module. Assertion of CE_ allows the RAM to drive data onto the load data bus. The individual output enables for each RAM are tied together and routed to the module connector. OE_ should be tied to ground through a 330 ohm resistor.

TBUS INTERFACE

10

The TBus is an 80 bit high speed 75 MHz bus which connects between the R8000 Microprocessor, the R8010 Floating Point Unit (FPU), and the Cache Controller (CC). The TBus is used for general communication between the devices. Ownership of the TBus changes depending on the operation to be performed. The format of the TBus also changes depending on which device is driving the bus.

Under normal operating conditions the R8000 CPU drives the TBus. The R8000 CPU performs operations and transfers the result or status of the operations across the TBus to the CC. However, there is frequent exchange of TBus ownership between the R8000 CPU and the CC. When conditions occur which warrant CC interaction to the system, such as a streaming cache miss, TBus ownership is transferred to the CC and the necessary operation completed. The TBus is the Cache Controller's main interface to the rest of the system. Among other things, the CC uses the TBus to transfer tag RAM address, state, and virtual synonym information, Data RAM address and write enable information, and interrupt status to the R8000 CPU. Because the CC does not have dedicated bus interfaces to the tag RAM's or the Streaming Cache, it is the responsibility of the R8000 CPU to take the information on the TBus and route it to the proper address or control busses in the system. Refer to figure 1-1 in chapter 1 for a block diagram of the R8000 Microprocessor Chip Set.

There are four types of transfers which can occur between the R8000 CPU and the R8010 FPU. However, during only one of these transfers, moving data from a floating point

register to a general purpose register inside the R8000 CPU, does the R8010 FPU actually drive the TBus. Under all other conditions the TBus pins of the R8010 FPU are in input mode.

Although there are physical pin connections between the CC and the R8010 FPU, they exist only because the R8000 CPU must communicate with both devices. There is no protocol or information exchange of any kind between the CC and the R8010 FPU.

The lower 72 bits of the TBus (TB71:0) connect between the three devices. The uppermost 8 bits (TB<79:72>) connect only between the R8000 CPU and the R8010 FPU.

Figure 10-1 shows the TBus connections between the three devices.

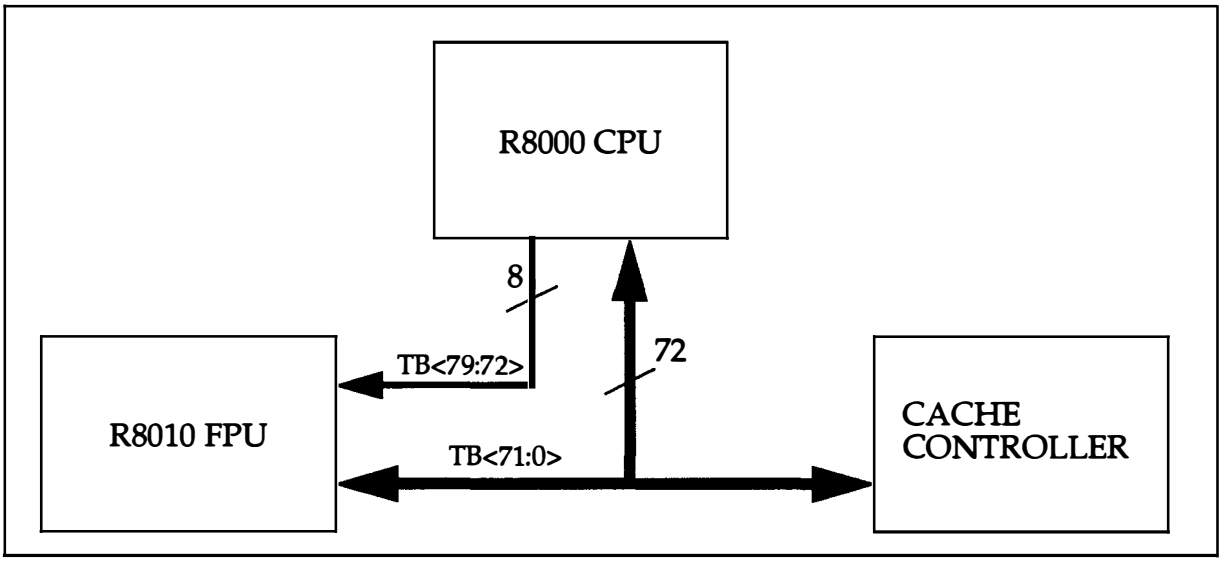


Figure 10-1 TBus Connections

10.1 PROCESSOR CONTROLLED TBUS FIELDS

The fields comprising the TBus include all of the basic information available for a given R8000 CPU access. If the R8000 CPU requires CC intervention to complete a cycle, this information is dispatched across the TBus and control of the bus transferred to the CC. The CC then uses the TBus information to take the appropriate action.

The Command and Coherence fields indicate what type of operation the R8000 CPU was trying to do. The Match, Set Address, State, and Virtual Synonym fields encode the information received during the Tag RAM lookup. The Size field indicates the size of non-cachable information and is only valid when non-cachable operations are being performed.

The format of the TBus changes depending which device is driving. When the R8000

CPU is in control of the TBus, the format of TB<71:0> is as shown in Table 10-1.

R8000 CPU Controlled Tbus Fields	Width	Tbus Bits
Reserved	12	71:60
Command	4	59:56
Size	3	55:53
Coherence Protocol	3	52:50
No Match	2	49:48
Set Address	2	47:46
State Information	2	45:44
Virtual Synonym	4	43:40
Physical Address	40	39:0

Table 10-1 R8000 CPU Controlled TBus Fields

Table 10-1 shows the bit orientations of the TBus under R8000 CPU control. Not all of the fields are used for each type of cycle. There are four divisions of the TBus and which portions are driven or tri-stated depends on the operation being executed.

1) Bits **79:72** connect between the R8000 CPU and the R8010 FPU and are always driven regardless of the operation.
2) Bits **71:64** are driven only by the R8000 CPU or the CC. Which device is driving depends on the state of the TBus state machine.
3) Bits **63:40** can be driven by the R8000 CPU, CC, or R8010 FPU. The CC drives the pins as dictated by the TBus state machine. The R8010 FPU drives the pins as commanded by the R8000 CPU, such as when the R8010 FPU is commanded to move data from a R8010 FPU register and place it in an R8000 CPU register. In all other situations the pins are driven by the R8000 CPU.
4) Bits **39:0** are also driven by the R8000 CPU, CC, and R8010 FPU. The same rules apply as those for bits 63:40. In addition, during the CC state (CC is in control of the TBus), seven cycles after one of the tag RAM read functions is driven, bits 39:0 of the TBus are tri-stated by the CC to allow the R8000 CPU to return data. Refer to section 11.2.2 for more information on the tag read function.

The following sections explain the functions in table 10-1 in more detail.

10.1.1 Reserved Field

The reserved field comprises bits 71:60. These bits are reserved by MIPS Technologies

Incorporated.

10.1.2 Command Field

The Command field is a four bit field which encodes the type of memory access the R8000 CPU was doing which caused the transfer. The Command field is always valid whenever VALIDOUT_ is asserted. Also included in the command field are the Cachable and Non-cachable coherence attributes of each access type.

Command	Encoded Value	Cachable	Non-Cachable
Read	0	yes	yes
Instruction Fetch	1	yes	yes
Write	2	yes	no
Write	3	yes	no
Reserved	4-5	---	---
Non-Cachable Write	6	no	yes
Sequential Non-Cachable Write	7	no	yes
Prefetch	8	yes	no
Reserved	9-15	---	---

Table 10-2 Command Field Encoding

The command field allows the R8000 CPU to inform the CC what type of cycle was being performed which caused the R8000 CPU to request action. The CC can then decode the four bit field and initiate the appropriate cycle. The command field is sampled by the CC during the clock when VALIDOUT_ is asserted by the R8000 CPU. Normally after VALIDOUT_ is asserted by the R8000 CPU the TBus state machine will transition from the RUN state to the CC state to allow the CC control of the TBus.

The **read** entry of the command field indicates any type of read cycle, such as the PROM, the Non-volatile RAM, registers in the CC chip, registers in the system address chips, main memory, I/O busses, etc.

Even though a read and an instruction fetch are both read cycles, having a separate **instruction fetch** entry in the command field allows the system designer to differentiate between the fetching of instructions and the fetching of data. Different coherence protocols can then be assigned to each. For example, if it is determined during a data fetch that no other processor in the system has that data the data can be marked as

fetch that no other processor in the system has that data the data can be marked as 'Exclusive' and written immediately without requiring further state modification transactions. This boosts system performance as data can be written and used immediately as it becomes available.

For instructions there is normally no necessity for writing, hence the instructions can be fetched and marked as 'Shared'. Therefore, if another processor requests the same instruction there is no need to do an 'Intervention' cycle (see section 11.4.9 for a definition of intervention). The other processor can simply fetch the instruction without requiring additional overhead.

The **write** entry of the command field comprises encoded values 2 and 3. These cycles are identical. Duplicate writes appear simply for the convenience of the gating arrangements inside the R8000 CPU. The CC must react to either of these bits active when VALIDOUT_ is asserted.

The **reserved** entries of the command field pertain to encoded entries 4, 5, and 9-15 and are reserved by MIPS Technologies.

The **R8000 CPU non-cachable write** entry of the command field is a partial write, meaning that less than 128 bytes can be written. During this cycle the R8000 CPU does not give up the TBus. In non-cachable write cycles there is no place to put the data and have it transferred at a later time with a write-back cycle. Non-cachable data is transferred to the Data cache but is marked as invalid. The R8000 CPU can use the data only once. If the R8000 CPU tries to read the non-cachable data a second time it will incur a miss because the line is marked as invalid. The R8000 CPU non cachable write cycle writes the data without concern for the sequential order of instructions.

In a **Sequential non-cachable write** the processor writes out the data and then halts execution and will not restart until commanded to do so. This assures that nothing happens out of order.

The **Prefetch** entry allows the CC to fetch data which may be required by the R8000 CPU. The prefetch is non-binding in the sense that there is no penalty involved and no requirement that the data be used by the R8000 CPU after being fetched. The prefetch is a special instruction which creates an address and fetches the corresponding data or instructions. Normal instructions cause movement of data between registers or devices. The prefetch does not demand any such action but rather generates an address which causes the CC to fetch data or instructions which may or may not be used. The advantage of prefetching is that at the time the prefetch is initiated the data is not needed by the R8000 CPU. However, if the prefetched data or instructions are used later on they are available immediately, the access time for the data having already been completed. Prefetched instructions or data are fetched by the CC and moved into the Data Cache and marked as invalid in the Data Cache Valid RAM. Note that prefetch instructions are inserted into the instruction stream by the compiler. There is no system requirement that they be supported.

10.1.3 Size Field

The Size field is an encoded three bit value which indicates the size of the operation, one through eight bytes, for non-cachable operations. The size field is used only on non-cachable operations and is valid for any command indicating a partial read. Cachable operations always fetch 128 bytes. The page table entries in memory determine whether a given location is cachable or non-cachable. The size field relates to the coherence protocol field described in section 11.1.4 below. The coherence field encodes the type of non-cachable operation and the size field indicates the corresponding amount of data to be transferred.

There is one exception to the size field. Non-cachable instruction fetches always fetch 32 bytes. During this type of non-cachable cycle the CC does not decode the size field because the fetch size has been pre-determined by the instruction itself. Non-cachable instruction fetches exist in order to facilitate the boot-up procedure. There are no non-cachable instructions fetches after completion of the boot-up procedure. Although 32 bytes is the specified minimum, more bytes can be fetched is desired. The amount fetched is system dependent. The encoded values of the size field are shown in Table 10-3.

Encoded Value	Size
0	1 byte
1	2 bytes
2	3 bytes
3	4 bytes
4	5 bytes
5	6 bytes
6	7 bytes
7	8 bytes

Table 10-3 Size Field Encoding

10.1.4 Coherence Protocol Field

The Coherence Protocol field is a three bit field used to indicate the coherence attributes in the page table entry for the access and is always valid whenever VALIDOUT_ is asserted. It is used to determine whether the access is cachable, and what type of protocol to follow in filling the streaming cache. The Coherence field allows data to maintain coherent in a multi-processor environment.

Non-cachable cycles transfer data from an external non-cachable source into the data cache of the R8000 CPU. When the R8000 CPU requests non-cached data the CC retrieves the data and places it in the Data Cache of the R8000 CPU and marks the status of the entry as invalid. The R8000 CPU has internal tracking mechanisms which allow the location to be read as soon as data becomes available without taking a miss due to the invalid status of the entry. However, the R8000 CPU can read the location only once. Any subsequent reads to that location result in a Data Cache miss. Table 10-4 shows the different cycles supported by the coherence field.

Encoded Value	Coherence Attribute
0	Processor Ordered Uncachable
1	Reserved
2	Uncachable Sequential
3	Cachable Non-Coherent
4	Cachable Coherent Exclusive
5	Cachable Coherent Exclusive on Write
6-7	Reserved

Table 10-4 Coherence Field

The **Processor Ordered Uncachable** entry is used for writing hardware registers in the CC which support the write-gatherer operation. The purpose of a write-gatherer is to gather 32 bit write operations from another source (such as a graphics engine) into a 128 byte block before sending them across the bus. Otherwise each single 32 bit write would have to be sent across the bus which would decrease system data bandwidth and degrade overall system performance.

The **Reserved** entry is reserved by MIPS Technologies and should not be used.

The **Uncachable Sequential** entry is the standard non-cachable protocol which can be

used by internal CC registers, I/O registers, memory, etc. Only those exact bytes specified are actually written or read. There is no minimum block size requirement.

The **Cachable Non-Coherent** allows data to be cached which is not coherent between processors. This function is supported for those systems which do not require that data coherency between processors be maintained.

The **Cachable Coherent Exclusive** entry allows data to be read from another device and marked as exclusive. It is the responsibility of the CC to retrieve the data from another processor and assure that the entry in the other processor is marked as invalid.

The **Cachable Coherent Exclusive on Write** entry is the standard protocol that all cachable cycles should follow. Data can be shared between processors and read access allowed to each at the same time. Any processor which desires to write the data must assure that all of the other processors mark their entries as invalid. Any other processor which then desired the newly written data must perform an intervention cycle in order to retrieve the data.

10.1.5 Match Field

The Match field in table 10-1 is a two bit field which reflects the status of the MATCH pin which was returned to the R8000 CPU by the Tag RAM. The R8000 CPU samples the various control signals from the Tag RAM and encodes the status of the Tag RAM lookup in the Match field of the TBus. The Match field also contains the status of whether the virtual synonym returned from the Tag RAM matched the virtual address bits from the original address. The number of virtual synonym bits actually checked depends on the size of the Data Cache. The result of the Tag RAM lookup is sent across the TBus to the CC and the appropriate action is taken by the CC.

The MATCH field is valid whenever VALIDOUT_ is asserted and is driven by the R8000 CPU whenever the R8000 CPU is doing a write cycle and the line is marked either shared or invalid. If the line is marked shared it must be upgraded. If the line is marked invalid the data must be retrieved. The state status is determined by the CC from the state field explained in section 10.1.7.

If the R8000 CPU is performing either a read or a write cycle and the line is marked exclusive no CC action is required. Also, if the R8000 CPU is performing a read cycle and the line is marked shared no CC action is required.

Separate Match and State fields are required due to the sectored nature of the streaming cache. There are four sectors per streaming cache line but only one address per line. Each of the four sectors of the line must be looked at individually to ascertain the status of each. It is possible to have an address match but no sector match (all sectors are in the State field are marked invalid). This is important to determine because if the address matches, the data must be placed at that address even though all of the corresponding

sectors may be invalid. If this were seen simply as a miss the CC would pick a random set to put the data in. You could then end up with multiple sets in the cache containing the same address. Table 10-5 lists the different codes of the Match field and their descriptions

Encoded Value	Coherence Attribute
0	Tag Match and Virtual Synonym Match (Cachable)
1	Tag Match and Virtual Synonym Miss-Match OR Tag Match and Non-Cachable Operation
2	Reserved
3	No Tag Match

Table 10-5 Match Field Encoding

If the Match field indicates both a Tag address match and a virtual synonym match (encoded value 0), the CC simply fetches and adds a sector to the existing line in both the Data cache and the Streaming cache.

If the field indicates a Tag address match and a virtual synonym miss-match (encoded value 1), the CC removes the entire line, invalidates the corresponding entry in the Data Cache, fetches the needed sector and places it in the streaming cache (in the same set).

If there is no Tag address match (encoded value 3), indicating a miss, the CC removes the entire line at a set which the CC chooses, invalidates the Data Cache entry for that line, fetches the needed sector and places it in the streaming cache at that set.

10.1.6 Set Address Field

The Set Address field in table 10-1 is a two bit field which reflects the status of the Tag RAM lookup and distinguishes which of the four sets in the Tag RAM the address compared. The Set Address field is valid whenever there is an address match as determined by the Match field. When a lookup cycle is performed on the 4-way set associative Tag RAM, and there is an address match, the Tag RAM encodes onto the MCHSA<1:0> pins which of the four sets of a given Tag RAM entry actually compared. This information is then routed by the R8000 CPU from its MCHSA inputs onto the Set Address field of the TBus (bits 47:46). Table 10-6 shows the encoding of the set address field.

Encoded Value	Set Address Match
0	Set 0 Match
1	Set 1 Match
2	Set 2 Match
3	Set 3 Match

Table 10-6 Set Address Field

10.1.7 State Field

The two bit State field indicates the coherency status for the given access. The state field is always driven whenever VALIDOUT_ is asserted. Every entry in the Tag RAM has address, state, and virtual synonym information associated with it. If there is a match to one of the 4 sets, the set address information is returned by the Tag RAM on the MCHSA<1:0> pins. The corresponding state information is returned by the Tag RAM on the MCHST<1:0> (Match State) pins. This information is then routed by the R8000 CPU from its MCHST inputs onto the State field of the TBus (bits 45:44).The state field is used to determine the reason for the miss.

Table 10-7 below lists the different codes of the State field and their descriptions.

Encoded Value	Definition
0	Invalid
1	Shared State
2	Exclusive State
3	Reserved

Table 10-7 State Field Encoding

10.1.8 Virtual Synonym Field

The four bit Virtual Synonym field contains virtual address bits 15:12 for the access which caused the operation. Every entry in the Tag RAM has address, state, and virtual synonym information associated with it. If there is a match to one of the 4 sets, the set address and state information are returned by the Tag RAM on the MCHSA and MCHST pins respectively. The corresponding virtual synonym information is returned by the Tag RAM on its MCHVS <3:0> (Match Virtual Synonym) pins. This information is then routed by the R8000 CPU from its MCHVS inputs onto the Virtual Synonym field of the TBus (bits 43:40). The virtual synonym field is required so that the Data Cache inside the R8000 CPU can be correctly invalidated.

The data cache is virtually indexed and physically tagged. The data cache is also larger than the minimum page size configuration of 4 KBytes. Consequently there could be multiple places in the data cache which a given doubleword could reside. The virtual synonym (V.S.) is used to assure that a given doubleword of data cannot reside in multiple places in the data cache.

If the correct 4 KByte page is chosen, indicated by a data cache tag match, the cache is accessed and the cycle completes. If the data cache tag misses the R8000 CPU initiates a streaming cache cycle to determine whether the data is then in the second level cache. The virtual synonym bits are stored in the Tag RAM and indicate that, if the data is in the data cache, there is the only location which it can reside, otherwise the V.S. bits would not match and a miss would occur. The R8000 CPU compares the V.S. bits of the address with the value returned from the Tag RAM lookup. Because there is only one V.S. in the data cache at any given time the only way to change from 4K Byte page to another is to have a data cache miss. On a data cache miss the R8000 CPU checks in the tag ram to see which virtual synonym is allowed for the corresponding cycle. If the V.S. in the data cache is the current one then the data has to be in the data cache. If there is a V.S. mismatch the data could be somewhere else in the data cache so action must be taken. In this case the whole 512 bytes is transferred to main memory and the 512 bytes in the data cache is then invalidated. The V.S. in the streaming cache is then changed and execution resumes. No action between the streaming cache and main memory is required.

10.1.9 Physical Address Field

The 40 bit Physical Address is the result of the TLB lookup corresponding to the access which caused the operation. Since there is no address bus interface between the R8000 CPU and the CC, the address information is transferred on TBus bits 39:0.

10.2 CC CONTROLLED TBUS FIELDS

The following section discusses the protocol involved when the cache controller is in control of the TBus. The CC can either arbitrate for control of the TBus, or it can be granted immediate control by the R8000 CPU. The CC is responsible for monitoring the activity on the system bus which is used by other processors in the system to communicate with each other. If another processor in the system requires access to local streaming cache memory, the CC will arbitrate for control of the streaming cache in order to service the incoming request.

A miss to the streaming cache causes the R8000 CPU to bypass arbitration and grant the bus to the CC immediately. The CC then fetches the data being requested from main memory, places it in the streaming cache, and releases control of the bus back to the R8000 CPU. Transition of the TBus state machine to 'CC' state allows the CC control of the TBus. Refer to section 10.3 for more information on the TBus state machine.

Table 10-8 shows the TBus protocol when the CC is in control.

CC Controlled Tbus Fields	Width	Tbus Bits
Reserved	4	71:68
Function	4	67:64
Streaming Cache Write Enables	4	63:60
External Set Address	2	59:58
Streaming Cache Address<21:4>	18	57:40
Tag RAM Address<39:5>	35	39:5
Reserved	1	4
Virtual Synonym	4	3:0

Table 10-8 CC Controlled TBus Fields

10.2.1 Reserved

TBus bits <71:68> and bit 4 are reserved by MIPS Technologies Incorporated and should not be used.

10.2.2 Function Field

The **Function** Field is a four bit field and is the only field by which the CC changes the internal state of the R8000 CPU instead of just the Tag RAM's or the streaming cache RAM's. The function field is valid on every clock which the TBus state machine is in the CC state. There are a number of different operations which can be performed based on the state of the function field. The function field controls such operations as invalidation of the first level Data Cache, instructing the R8000 CPU to empty the floating point store address and store data queues by briefly returning the TBus to the R8000 CPU for this operation, interrupt status, and signal generation for reading the Tag RAM.

The function field is represented by TBus bits 67:64. The 4 bit value is encoded as follows.

Encoded Value	Description
0	No R8000 CPU operation
1	Reserved*
2	Interrupt
3	Empty Queue
4	Reserved*
5	Read of Even Tag RAM
6	Read of Odd Tag RAM
7	Combined Read of both Tag RAM's
8-9	Reserved*
10	Invalidate Data Cache Line (32 bytes)
11-15	Reserved*

Table 10-9 Function Field Encoding

* Encoded values reserved by MIPS Technologies Incorporated.

Below is an explanation of each of the various Function fields.

10.2.2.1 No R8000 CPU Operation

No R8000 CPU Operation is provided so that the CC is not forced to do any of the other operations and is analogous to the VALIDOUT_ signal. When the R8000 CPU is in

control of the TBus the information on the TBus is valid only when VALIDOUT_ is active. However, when the CC is in control the TBus is valid every cycle. But the CC does not always drive valid information every cycle. The No R8000 CPU Operation mechanism is used to indicate to the rest of the system that, although the TBus is being driven during a given cycle, it contains no valid information.

10.2.2.2 Interrupt

The Interrupt function changes the internal state of the R8000 CPU by setting the interrupt pending bit. The R8000 CPU then reads some memory mapped registers inside the CC to find out more information about the interrupt so that the proper interrupt service routine can be executed. The R8000 CPU is responsible for clearing the interrupt bit once servicing has been completed. Refer to section 10.4.13 for more information.

10.2.2.3 Empty Queue

The Empty Queue function is initiated when the CC finds an address match in the Store Address Queue (SAQ). There are two reasons why the CC does a SAQ compare, Intervention and write back or line replacement from the streaming cache to main memory. The address match is detected by the CC while the TBus state machine is in the 'CC' state. A valid SAQ compare causes the Cache Controller to transition from CC state to EQ state. Control of the TBus is returned to the R8000 CPU allowing the instructions corresponding to the addresses in the SAQ to be executed. After the queue is emptied the TBus is returned to the CC. Refer to section 10.3 for more information in the CC state machine. Refer to the Store Address Queue Compare cycle in section 10.4 for more information.

10.2.2.4 Tag Read Even/Odd

The Tag Read Even and Tag Read Odd functions affect the tri-state enables of the R8000 CPU on TBus bits <39:0>, which represent the Tag address and virtual synonym fields, as well as the Tag address field of the Tag RAM connections. Refer to Figure 10-2 below:

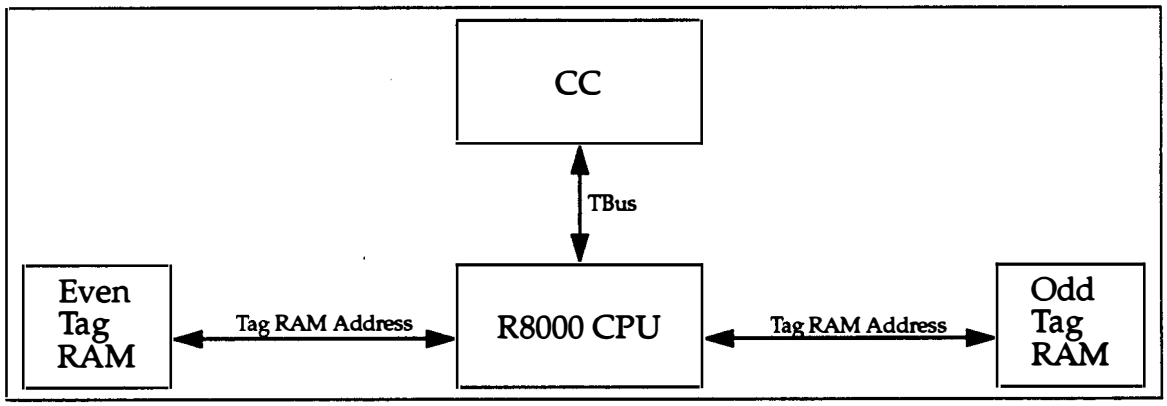


Figure 10-2 Interface Busses of the Tag RAM Read Function

Whether the even or odd tag RAM is read depends on the state of address bit A3. If this bit is deasserted (0), the even tag RAM is read. Assertion of A3 (1) causes the odd tag RAM to be read. The Tag RAM even/odd read operation does not change the internal state of the R8000 CPU but rather it controls a multiplexor inside the R8000 CPU to assure that the correct Tag RAM information is transferred across the TBus.

Once the tag read function is placed on the TBus, the different busses in the above diagram tri-state and drive at various times to complete the operation. The table below shows a behavior of the various busses during a Tag RAM read. Refer to the Tag RAM read cycle and associated timing diagram in section 10.4 for more information.

Cycle	1	2	3	4	5	6	7	8	9	10
Tag RAM Address	CPU	CPU	CPU	Z	TAG	Z	CPU	CPU	CPU	CPU
TBus	CC	CC	CC	CC	CC	CC	Z	CPU	Z	CC

Table 10-10 Bus Behavior During a Tag RAM Read

In Table 10-10 above the CC has encoded the function field of the TBus to indicate a Tag RAM read and placed the request on the TBus pins <67:64> in Clock 0 (not shown). For the first three clocks (1-3) after the tag read request is placed on the bus by the CC the R8000 CPU continues to drive the tag RAM address pins (refer to Figure 10-2 above). During this time the CC is driving the TBus.

In clock 4 the R8000 CPU stops driving the tag address bus and tri-states. The CC also asserts the signal OE_ to the Tag RAM in clock 4. Assertion of OE_ allows the information on the 20 bit Tag RAM tag address bus to be driven by the Tag RAM back to the R8000 CPU in clock 5. In clock 6 the Tag RAM releases control of the tag address bus back to the R8000 CPU and the R8000 CPU begins driving again in clock 7.

At this point the Tag RAM has been accessed and the appropriate information driven back to the R8000 CPU by the Tag RAM. The information must now be transmitted by the R8000 CPU across the Tbus and back to the CC. There is a three clock delay internal to the R8000 CPU between when the information is driven by the Tag RAM in clock 5 and when it is driven by the R8000 CPU onto the TBus in clock 8.

When the Tag Read function appears on the Tbus (clock 0 in table 10-10, not shown) the CC will continue to drive the Tbus for 6 clocks (clocks 1-6 in Table 10-10). In clock 7 the CC releases control of the TBus, allowing the Tag address information to be driven by the R8000 CPU in clock 8. In clock 9 the R8000 CPU tri-states the TBus and relinquishes control back to the CC in clock 10, thereby completing the cycle.

10.2.2.5 Tag RAM Combined

The Tag Read Combined function allows the CC to read the state and dirty bit information from each Tag RAM at the same time. The even and odd dirty bits must then be logically OR'ed together external to the R8000 CPU. Because both the even and odd Tag RAM's are controlled by the same write enable from the CC, they will always have the same address, state, and virtual synonym information. However, each Tag RAM has a separate Dirty Bit Set (DBSET) write enable pin. The Tag Read combined function is used by the CC to determine if any of the sectors in a given line of the cache has entered the dirty exclusive state. When both RAM's are read at the same time, the state and virtual synonym information from the Odd tag RAM is returned by default. Only the dirty bit information from the even Tag RAM is returned. Refer to the Tag Read combined timing diagram and accompanying table in section 10.1.3 for more information.

10.2.2.6 Invalidate Data Cache

The Invalidate Data Cache function causes 32 bytes of the Data Cache to be invalidated at the address specified by the Virtual Synonym and Tag RAM address fields of the Tbus. When the Invalidate Data Cache function is initiated by the CC, bits <11:5> of the Tag RAM address field (TBus bits 11:5) are concatenated with the four bit virtual synonym field (TBus bits 3:0) to form an index to the Data Cache Valid RAM. The remaining TBus Tag address bits (TBus bit 39:12) are compared to the physical tag to determine whether the addresses to be invalidated resides in the Data Cache. When a data cache invalidate is initiated, the CC does not know whether the data is in the data cache. If the tag matches, the eight valid bits representing the 32 byte line are cleared. Refer to Chapter 1 for more information on the Data Cache Valid RAM architecture. Figure 10-3 how the TBus is used during a Data Cache invalidation cycle.

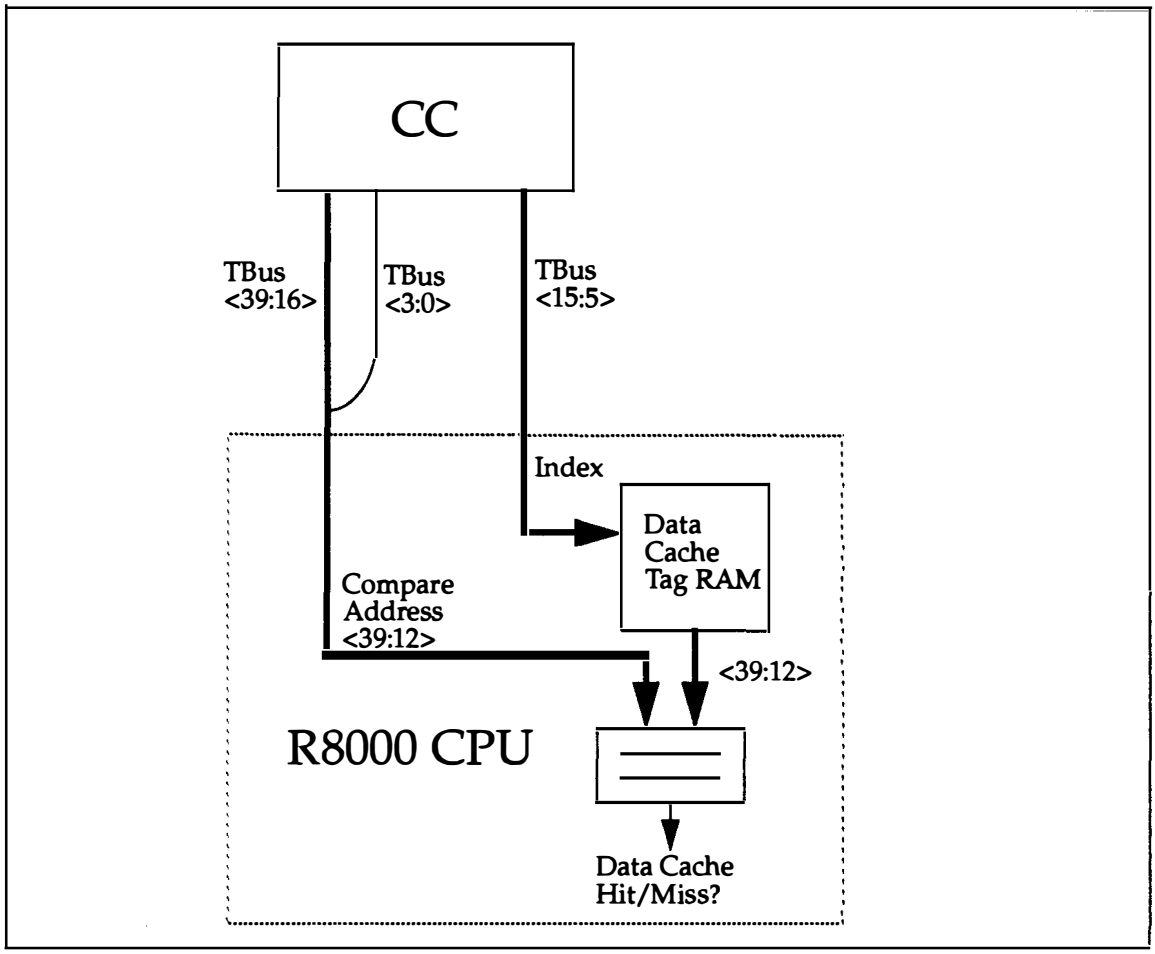


Figure 10-3 TBus format During a Data Cache Invalidation Cycle

10.2.3 Streaming Cache Write Enables Field

The four bit Streaming Cache Write Enables field is represented by TBus bits 63:60 and is used when writing data to the streaming cache. The smallest writable quantity supported by the streaming cache is 32 bits. The 4 MByte streaming cache consists of four SIM modules, each 1 MByte in size and organized as 256K X 36 bits. During streaming cache write cycles initiated by the CC the Cache Controller places the write enable information onto the TBus. The R8000 CPU then routes TBus bits 63:60 onto the correct write enable pins which connect between the R8000 CPU and the Streaming Cache. The Streaming Cache Write Enable field can contain valid cycle information and be driven at the same time as the Tag RAM address and other fields which do not pertain to a streaming cache cycle. This allows information for multiple cycles to be on the TBus at the same time. Figure 10-4 shows how the write enable field of the TBus corresponds to the Streaming Cache.

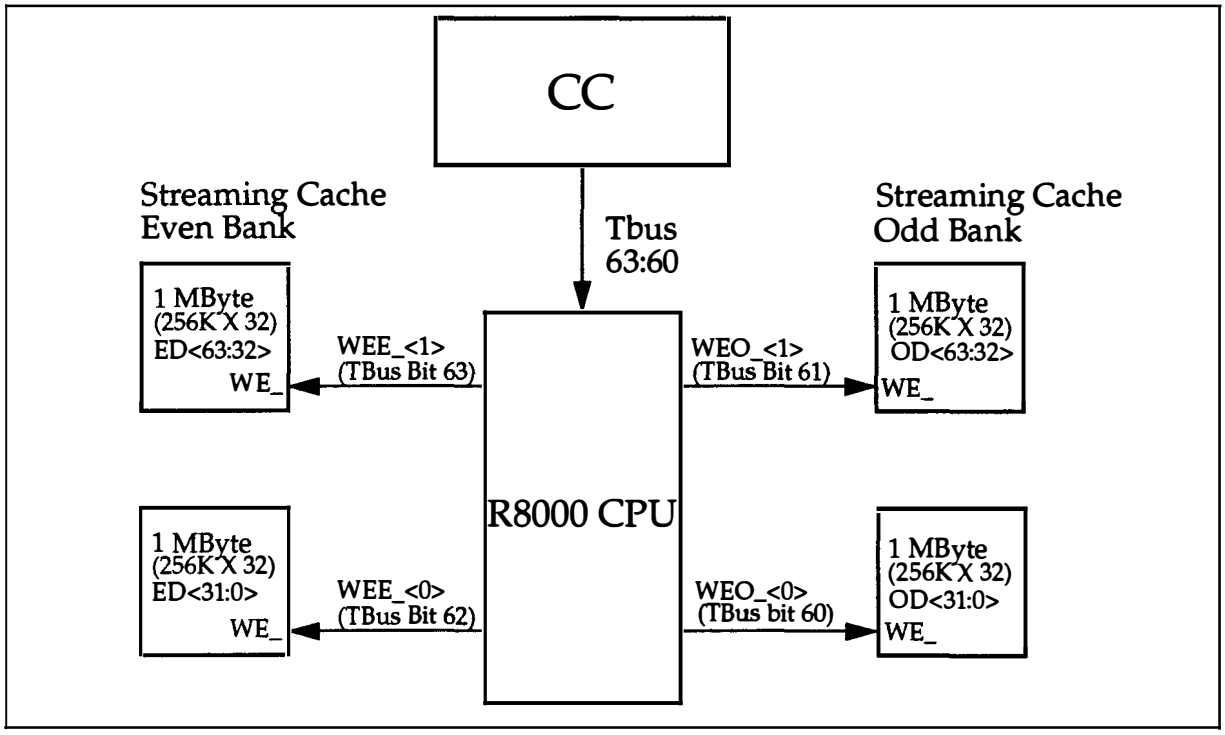


Figure 10-4 Streaming Cache Write Enables

10.2.4 External Set Address Field

The External Set Address field forms the two uppermost bits of address for the streaming cache and is used to differentiate in which of the four sets of the streaming cache the data resides. Both the Tag RAM and the streaming cache are 4-way set associative. When the Tag RAM is accessed all four ways in the entry are compared to the address on the bus. If there is a Tag Match the Tag RAM must then encode which of the 4 ways actually compared. The Tag RAM uses the MCHSA<1:0> pins to encode this information. The information is sent back to the R8000 CPU and stored. When it comes time to use this information the R8000 CPU places the 2 bit value onto the External Set Address (ESA<1:0>) pins. The External Set Address field can contain valid cycle information and be driven at the same time as the Tag RAM address and other fields which do not pertain to a streaming cache cycle. This allows information for multiple cycles to be on the TBus at the same time. Figure 10-5 shows how the CC uses the R8000 CPU and the TBus to address the streaming cache.

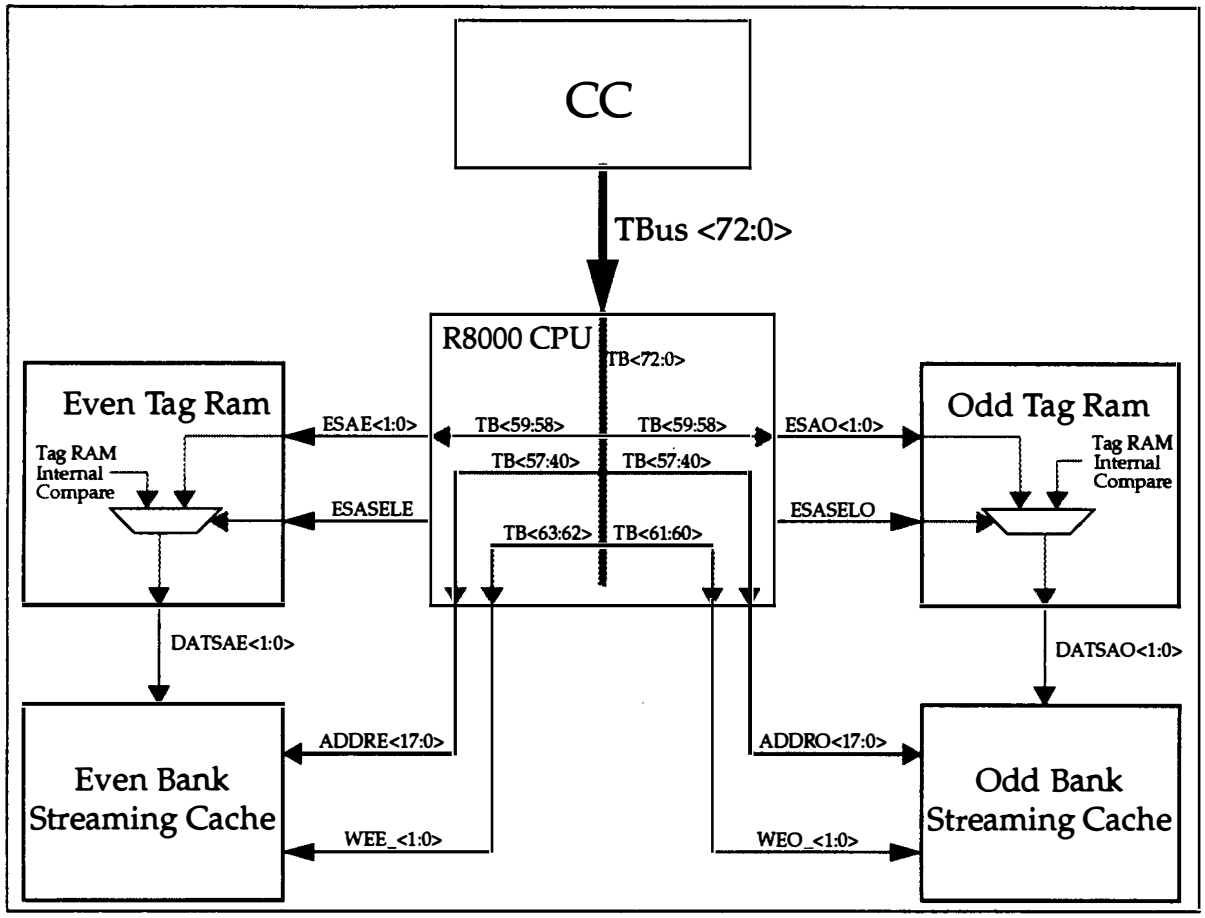


Figure 10-5 CC Addressing of the Streaming Cache

Sixteen bits of address and two bits of Data Set Address (DATSA) are used to address the streaming cache. On a read cycle the DATSA bits are derived from an R8000 CPU lookup of the Tag RAM. The correct set which matched is encoded by the Tag RAM and passed onto the DATSA outputs. This is denoted by "Tag RAM Internal Compare" in figure 10-5. On a write cycle the lookup is still performed, but often times the lookup is performed many cycles before the write data actually becomes available. The result of the lookup is passed to the CC and stored. When the data becomes available the CC transfers the corresponding set information it originally received from the R8000 CPU on TBus bits <59:58>. The two bit value passes through the R8000 CPU onto the ESA<1:0> outputs, through the multiplexor logic in the Tag RAM, and finally out onto the DATSA<1:0> output pins of the Tag RAM to the streaming cache. The R8000 CPU can supply up to 18 bits of address for use with 16 MBit DRAM's. Currently only 16 of the 18 address bits are used.

10.2.5 Streaming Cache Address Field

The Streaming Cache Address field is transmitted at the same time as the Write Enable and External Set Address fields. These fields combined together provide the necessary information to perform a Streaming Cache cycle. The field is 18 bits in size and is transferred on TBus bits <57:40>. The Streaming Cache Address field can contain valid cycle information and be driven at the same time as the Tag RAM address and other fields which do not pertain to a streaming cache cycle. This allows information for multiple cycles to be on the TBus at the same time.

10.2.6 Tag RAM Address Field

The Tag RAM Address is used to write and read the Tag Address of the Tag RAM, check the Store Address Queue, and invalidate the first level data cache of the R8000 CPU. The tag address of the Tag RAM is 20 bits wide. Since tag address, state, and virtual synonym information are all transferred to and from the Tag RAM using the same 20 bit tag bus, either tag information of state and virtual synonym information can be written at any given time.

When the tag address is read, the index must be placed on bits <21:5>. The result is returned by the Tag RAM to the R8000 CPU, and in turn from the R8000 CPU to the CC on TBus bits <39:0>. Although all 40 bits are driven, only those bits corresponding to the actual tag will be valid. The remaining bits are undefined. When the Store Address Queue is checked, Tag RAM address bits <19:7> are used for comparison.

10.2.7 Virtual Synonym Field

The virtual synonym field is used to assure that the correct location is invalidated during a data cache invalidation cycle. The index to the Data Cache Tag RAM inside the R8000 CPU is formed by TBus bits <15:5>. The four bit virtual synonym field (TBus <3:0>), is concatenated with tag address bits (TBus 39:16) to form data cache address. The address is used for the data cache tag comparison to determine whether the location to be invalidated resides on the data cache.

10.3 TBUS STATE MACHINE

The TBus state machine is used for arbitration of the TBus between the R8000 CPU and the CC. It is important to note that the state diagram in Figure 10-6 below shows only a functional representation of how the external signal pins function during TBus

arbitration. It is not an actual representation of the TBus state machine. In actuality there are two TBus state machines, one each inside the R8000 CPU and the CC, which are designed to work together in a synchronous pipelined environment. Although each is similar to the machine below, the two machines differ in both the number of states and the signals generated. The intent of the diagram below is to provide the reader with a functional overview of how TBus arbitration is performed and to show how the signal pins act during TBus arbitration and ownership transfer.

In Figure 10-6 a bar ($\overline{xxxx}$)over the signal name indicates the signal is deasserted. An underscore (_) indicates the signal is active low.

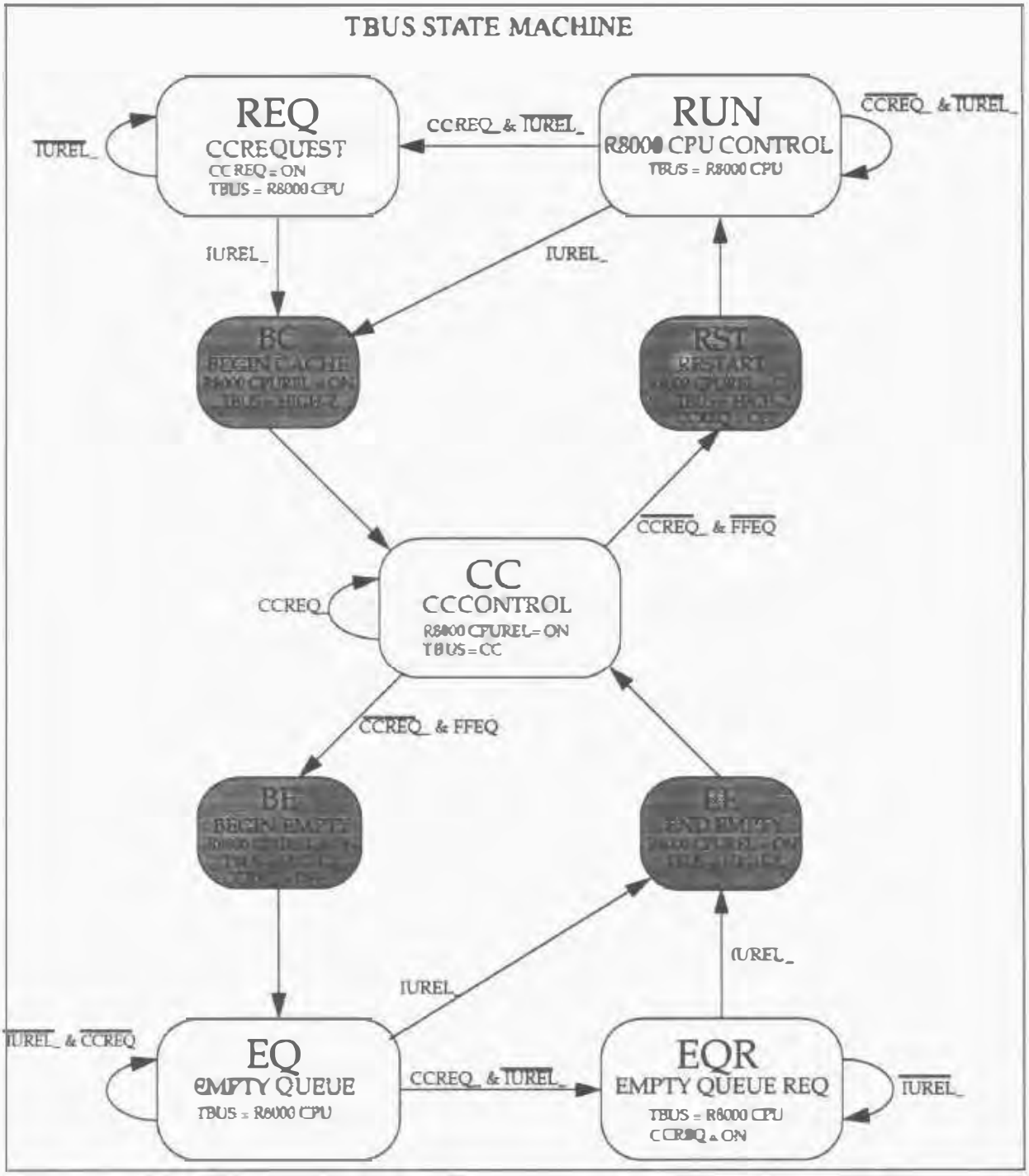


Figure 10-6 TBus State Machine

10.3.1 RUN State

The RUN state is the normal state of operation when the R8000 CPU is in control of the TBus. There are two conditions under which the machine will transition from the RUN

state. Assertion of CCREQ_ by the Cache Controller (CC) causes the machine to transition to the REQ state. The R8000 CPU can also give up the TBus without it being requested by the CC. On a streaming cache miss, for example, the R8000 CPU will assert IUREL_ and give up the TBus immediately. The CC then fetches the requested data from main memory.

10.3.2 REQ State

The REQ state is entered when the CC asserts CCREQ_ to inform the R8000 CPU that the TBus is being requested. While in the REQ state the CC is guaranteed to have to wait no longer than 1500 clocks before being granted the bus. Until IUREL_ is asserted by the R8000 CPU the machine remains in the REQ state. Assertion of IUREL_ causes the machine to transition from REQ state to BC state.

When BC state is entered the TBus buffers on the R8000 CPU are in the output mode and the buffers on the CC are in the input mode. The BC state is one clock in duration and allows each device to switch from input to output mode, or vise-versa. On the following clock the machine transitions to CC state.

10.3.3 CC State

Transition to the CC state indicates that the CC is now in control of the TBus. From the CC state the machine can either transition to BE state or to RST state. Deassertion of CCREQ_ by the CC causes the transition to BE state. This transition occurs when the CC wishes to give control of the TBus back to the R8000 CPU temporarily in order to allow the R8000 CPU to empty the store address queue (i.e. execute all remaining instructions). For example, when the CC wishes to write back a line from the streaming cache to main memory, it first checks the store address queue (SAQ) to determine whether the address corresponding to the line to be written out is in the SAQ. Checking the queue assures that the most current data is written out. Refer to figure 10-16 for more information on the SAQ compare operation.

10.3.4 FFEQ Signal Generation

If there is no valid address compare, the line is written out. However, if the address compare is valid, the CC must allow the R8000 CPU to write out the data which corresponds to the address in the queue. Assertion of the 'ffeq' signal allows the machine to transition from CC state to BE state. The 'ffeq' signal causing the transition is generated by the state machine and is the decoded equivalent of the function field, represented by bits 67:64 of the TBus. If the bit orientation on these pins is 0111, indicating that the CC wishes the queue to be emptied, the 'ffeq' signal is generated and the machine transitions to BE state. The BE state allows one clock for bus turn-around.

10.3.5 EQ State

The machine transitions to EQ state on the next clock, IUREL_ and CCREQ_ are deasserted and TBus control is returned to the R8000 CPU. Once the SAQ is emptied the R8000 CPU reasserts IUREL_ and the machine transitions to the EE state to allow for bus turn-around and then back to CC state.

10.3.6 EQR State

There are some circumstances under which the CC cannot wait for the SAQ empty operation to be completed. When this occurs the CC asserts CCREQ_ and the machine transitions to EQR state. After the current operation is completed the R8000 CPU asserts IUREL_ and the machine transitions through EE state and back to CC state. Transition from EQR to EE state means that the SAQ may or may not be empty.

When the CC is finished completing the necessary operations, the machine transitions from CC state to RST state. This transition is caused by deassertion of the 'ffeq' signal, which is in turn caused by the function field changing orientation to reflect something other than empty queue status.

Any information which remains in the SAQ at the time the state machine was instructed to transition remains there and is executed as soon as the R8000 CPU resumes control of the TBus.

10.4 CYCLE TYPES

The CC is required to read and write the Tag RAM's and Streaming Cache, perform Data Cache invalidate cycles, and transmit interrupt status among other things. Because the CC does not have any dedicated busses between itself and other devices in the system, all communication with these other devices takes place via the TBus. This section discusses the timing issues involved with transmitting cycle information across the TBus.

10.4.1 Tag Address Write

The CC is responsible for management of the Tag RAM. This includes writing to the Tag RAM whenever its contents require modification, and reading the Tag RAM to determine the state of a given entry.

A tag address write requires a 20 bit tag address, an 11 bit index, and a 2 bit sector field. Both Tag RAM's are written at the same time? Refer to Figure 10-7 below. TBus <39:5> are placed on the bus in clock 1. Two clocks are required for the information to propagate through the R8000 CPU and out onto the Tag RAM tag and index pins. The two bit set address field, RWSA<1:0> (Read Write Set Address), is hard-wired from the CC to each Tag RAM. Hence in order to assure that all of the signals arrive to the Tag RAM at the same time, the RSWA field is issued two clocks later in clock 3. Also in clock 3 the signal TWE_ (Tag Write Enable) is asserted by the CC. This signal is also hard-wired to the Tag RAM's. The signal STWE_ (State Write Enable) must be deasserted in clock 3 to assure that the Tag RAM writes tag address information and not state or virtual synonym information.

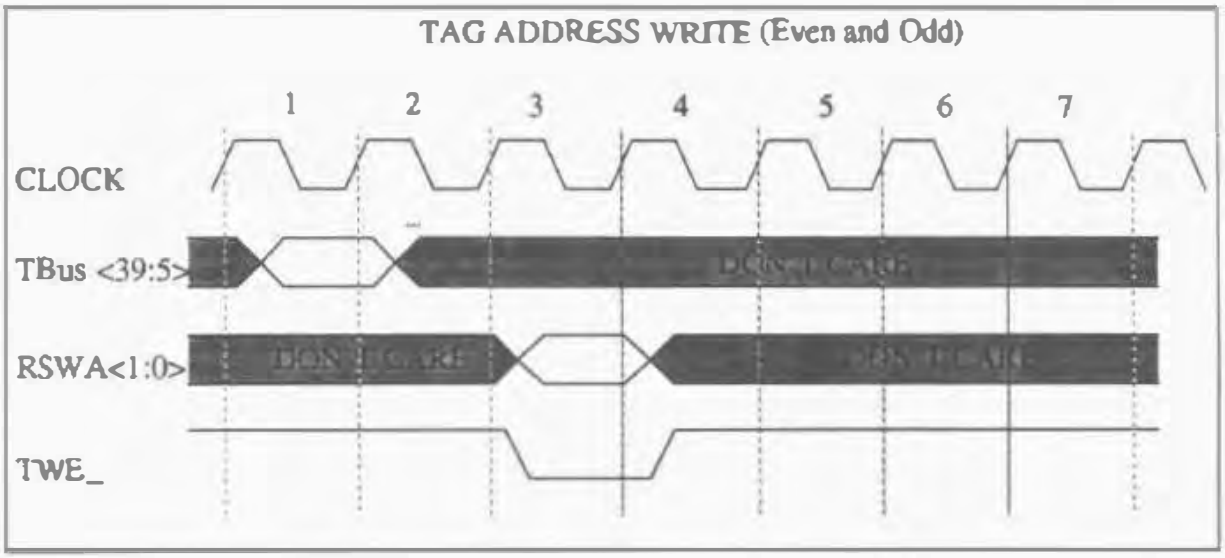


Figure 10-7 : Tag Address Write (Even and Odd)

Table 10-11 below shows how TBus<39:20> connect to the R8000 CPU, and how the R8000 CPU tag pins connects to the Tag RAM tag pins. The pin connections are the same for both the even and odd Tag RAM's.

TBus Bits <39:20>	R8000 CPU Tag Bits <21:2>	Tag RAM Tag Bits <19:0>
TBus<39>	Tag <21>	Tag<17>
TBus<38>	Tag <20>	Tag<16>
TBus<37>	Tag <19>	Tag<15>

Table 10-11 Tag RAM Tag Pin Connection Chart

TBus Bits <39:20>	R8000 CPU Tag Bits <21:2>	Tag RAM Tag Bits <19:0>
TBus<36>	Tag <18>	Tag<14>
TBus<35>	Tag <17>	Tag<13>
TBus<34>	Tag <16>	Tag<12>
TBus<33>	Tag <15>	Tag<11>
TBus<32>	Tag <14>	Tag<10>
TBus<31>	Tag <13>	Tag<9>
TBus<30>	Tag <12>	Tag<8>
TBus<29>	Tag <11>	Tag<7>
TBus<28>	Tag <10>	Tag<6>
TBus<27>	Tag <9>	Tag<5>
TBus<26>	Tag <8>	Tag<4>
TBus<25>	Tag <7>	Tag<3>
TBus<24>	Tag <6>	Tag<2>
TBus<23>	Tag <5>	Tag<1>
TBus<22>	Tag <4>	Tag<0>
TBus<21>	Tag <3>	Tag<19>
TBus<20>	Tag <2>	Tag<18>

Table 10-11 Tag RAM Tag Pin Connection Chart

Table 10-12 below shows how TBus<19:5> connect to the R8000 CPU, and how the R8000 CPU index pins connects to the Tag RAM index and sector pins. The pin connections are the same for both the even and odd Tag RAM's.

TBus Bits <19:7>	R8000 CPU Index Bits <14:2>	Tag RAM Index Bits <10:0>
TBus<19>	Index <14>	Index <10>

Table 10-12 Tag RAM Index and Sector Pin Connection Chart

TBus Bits <19:7>	R8000 CPU Index Bits <14:2>	Tag RAM Index Bits <10:0>
TBus<18>	Index <13>	Index <9>
TBus<17>	Index <12>	Index <8>
TBus<16>	Index <11>	Index <7>
TBus<15>	Index <10>	Index <6>
TBus<14>	Index <9>	Index <5>
TBus<13>	Index <8>	Index <4>
TBus<12>	Index <7>	Index <3>
TBus<11>	Index <6>	Index <2>
TBus<10>	Index <5>	Index <1>
TBus<9>	Index <4>	Index <0>
TBus<8>	Index <3>	Sector <1>
TBus<7>	Index <2>	Sector <0>

Table 10-12 Tag RAM Index and Sector Pin Connection Chart

10.4.2 Tag Address Read of Even Tag RAM

A Tag Address Read is performed whenever the CC wants to know the status of a given entry. An encoded value of 5h in the function field informs the R8000 CPU that the cycle in progress is an Even Tag RAM read and indicates to the R8000 CPU such parameters as when to expect the data back from the Tag RAM, when to place it on the TBus, and what TBus bits to place the data on. It is not necessary that the R8000 CPU knows whether the cycle is a tag address read or a state and virtual synonym read. This decision is made by the CC. Hard-wired signals between the CC and the Tag RAM assure that the appropriate information is routed to the outputs of the Tag RAM.

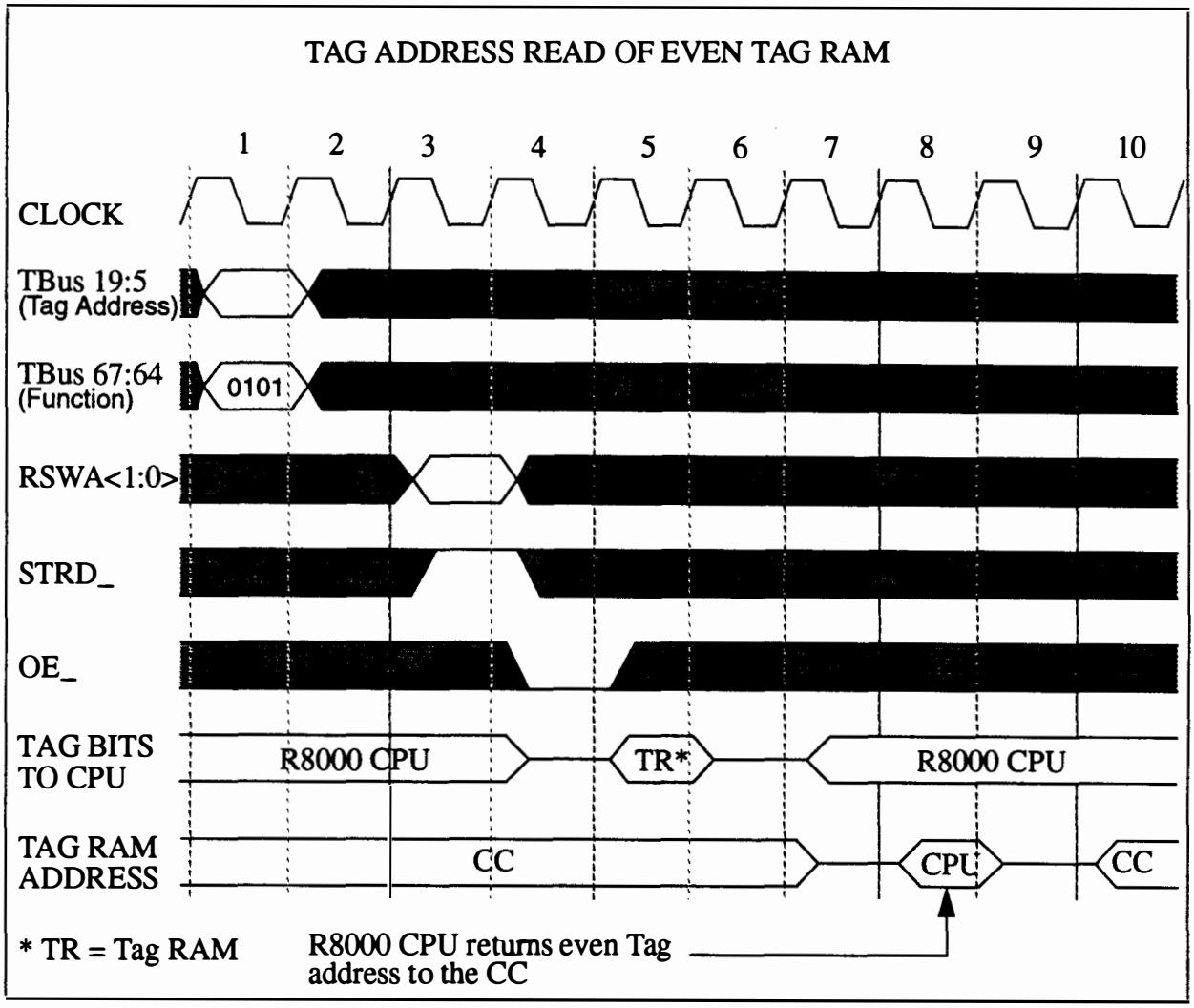


Figure 10-8 Tag Address Read of Even Tag RAM

The index information is placed on the TBus by the CC in clock 1. Also in clock 1 the function field is encoded and placed on the TBus. As with the Tag RAM address write cycle above, the RSWA<1:0> field is driven by the CC two clocks later in clock 3 to allow the R8000 CPU adequate time to route the index information to the Tag RAM. The STRD_ (State Read) signal must be deasserted in clock 3. Deassertion of STRD_ allows the Tag RAM to fetch tag address information as opposed to state and virtual synonym information.

The Tag RAM requires 1 clock after the RWSA field becomes valid to access the requested information. The Tag address bus is bidirectional. The signal OE_ is asserted by the CC in clock 4 which routes the read information out onto the tag address bus.

One clock after OE_ is asserted, in clock 5, the Tag RAM drives the requested

information onto the tag address bus and back to the R8000 CPU. From the CC's point of view, the passage of information from the Tag RAM back to the R8000 CPU is transparent to the operation and hence is shown only for clarity. The R8000 CPU then requires clocks 6 and 7 for propagation time and in clock 8 the requested information is driven by the R8000 CPU onto the TBus and back to the CC.

Although the TBus state machine is in the CC state and the CC is driving the TBus, the R8000 CPU must drive the requested information back to the CC. Hence the CC must give up the bus for one clock to allow the transfer to take place. In clocks 1-6 the CC is driving the TBus. In clock 7 the CC deasserts CCREQ_, causing the state machine to transition to state RST and the TBus to tri-state. In clock 8 the machine transitions from state RST to state RUN and control of the TBus is granted to the R8000 CPU. The R8000 CPU then drives the TBus during clock 8. In clock 9 the R8000 CPU asserts IUREL_ and the machine transition to state BC, where the TBus is again tri-stated. The machine then transitions from state BC to state CC where the CC again takes control of the TBus.

10.4.3 Tag Address Read of Odd Tag RAM

A Tag Address Read is performed whenever the CC wants to know the status of a given entry. An encoded value of 6h in the function field informs the R8000 CPU that the cycle in progress is an Odd Tag RAM read and indicates to the R8000 CPU such parameters as when to expect the data back from the Tag RAM, when to place it on the TBus, and what TBus bits to place the data on. It is not necessary that the R8000 CPU knows whether the cycle is a tag address read or a state and virtual synonym read. This decision is made by the CC. Hard-wired signals between the CC and the Tag RAM assure that the appropriate information is routed to the outputs of the Tag RAM.

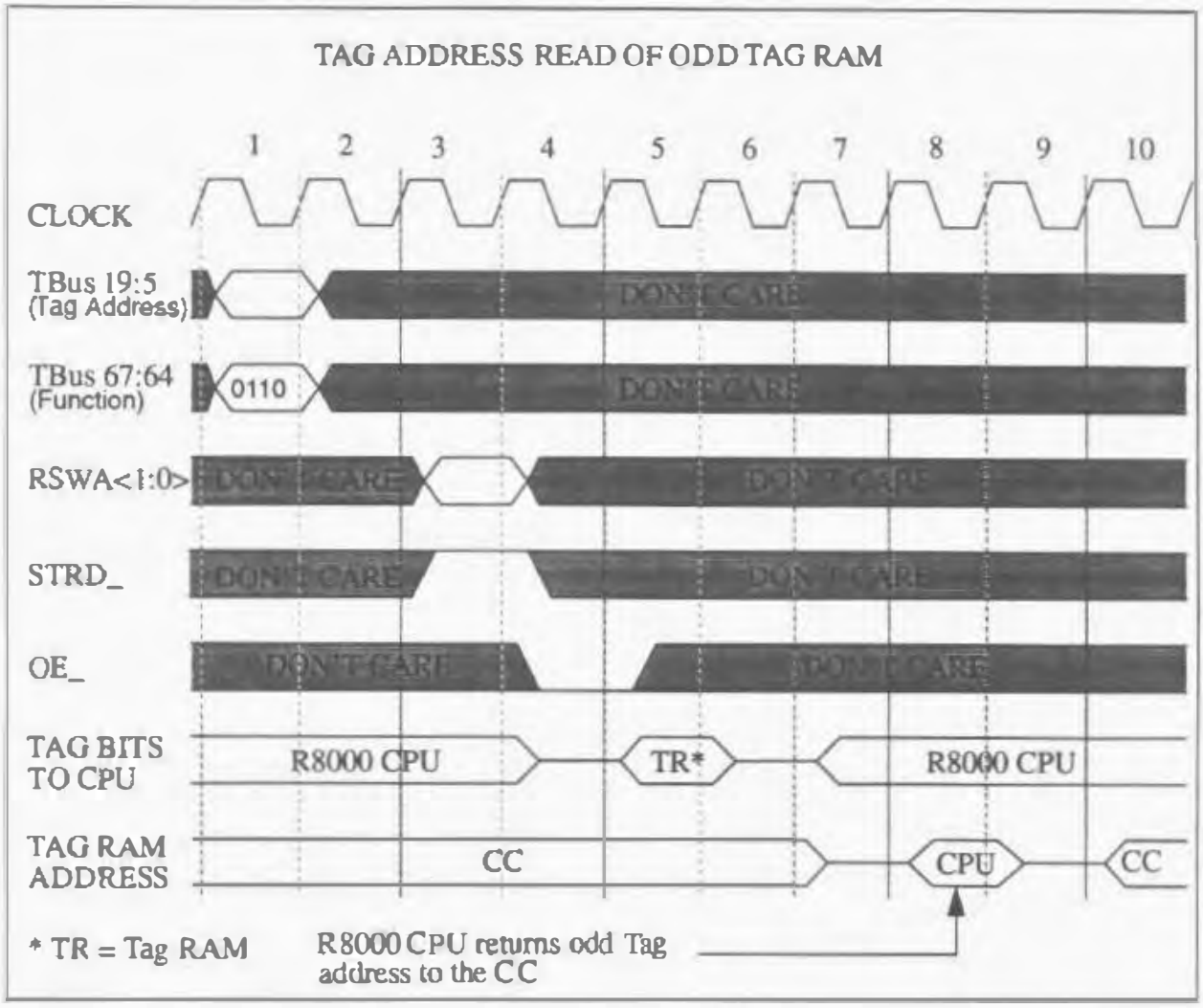


Figure 10-9 Tag Address Read of Odd Tag RAM

The index information is placed on the TBus by the CC in clock 1. Also in clock 1 the function field is encoded and placed on the TBus. As with the Tag RAM address write cycle above, the RSWA<1:0> field is driven by the CC two clocks later in clock 3 to allow the R8000 CPU adequate time to route the index information to the Tag RAM. The STRD_ (State Read) signal must be deasserted in clock 3. Deassertion of STRD_ allows the Tag RAM to fetch tag address information as opposed to state and virtual synonym information.

The Tag RAM requires 1 clock after RSWA becomes valid to access the requested information. The Tag address bus is bidirectional. The signal OE_ is asserted by the CC in clock 4 which routes the read information out onto the tag address bus.

One clock after OE_ is asserted, in clock 5, the Tag RAM drives the requested information onto the tag address bus and back to the R8000 CPU. From the CC's point of

view, the passage of information from the Tag RAM back to the R8000 CPU is transparent to the operation and hence is shown only for clarity. The R8000 CPU then requires clocks 6 and 7 for propagation time and in clock 8 the requested information is driven by the R8000 CPU onto the TBus and back to the CC.

Although the TBus state machine is in the CC state and the CC is driving the TBus, the R8000 CPU must drive the requested information back to the CC. Hence the CC must give up the bus for one clock to allow the transfer to take place. In clocks 1-6 the CC is driving the TBus. In clock 7 the CC deasserts CCREQ_, causing the state machine to transition to state RST and the TBus to tri-state. In clock 8 the machine transitions from state RST to state RUN and control of the TBus is granted to the R8000 CPU. The R8000 CPU then drives the TBus during clock 8. In clock 9 the R8000 CPU asserts IUREL_ and the machine transition to state BC, where the TBus is again tri-stated. The machine then transitions from state BC to state CC where the CC again takes control of the TBus.

10.4.4 Back to Back Tag Address Read

Due to the pipelined nature of the TBus, the CC can generate a new operation every clock. Figure 10-10 shows a timing diagram of a back to back tag address read operation. In clock 1 the index and function fields are transmitted across the TBus. In clock 2 information for the second operation (cycle B) is dispatched. Also during clock 2 the information for cycle A has begun propagating through the R8000 CPU. In clock 4 the Tag RAM drives cycle A tag information back to the R8000 CPU. At the same time the index information for cycle B is being used to access the next Tag RAM location, and in clock 5 this information is driven out.

In clock 7 the tag information for cycle A has propagated through the R8000 CPU and is placed on the TBus and returned to the CC. Also in clock 7 cycle B is in the last stage of propagation through the R8000 CPU, and in clock 9 the information is driven onto the TBus. Figure 10-10 shows a timing diagram of a back to back tag address read.

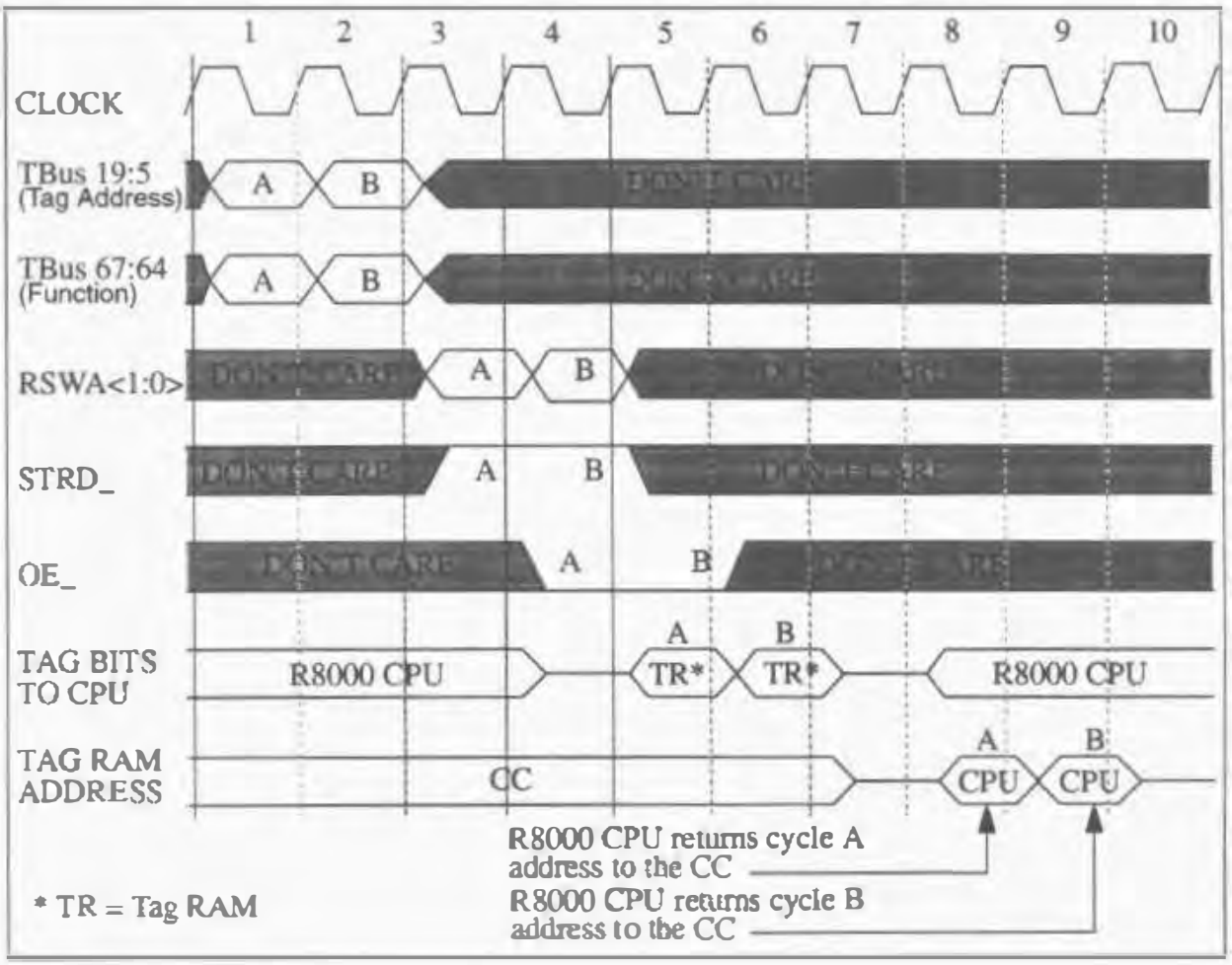


Figure 10-10 Back to Back Tag Address Read

10.4.5 Tag RAM State and Virtual Synonym Write

The 20 bit tag address bus of the Tag RAM is used for reading and writing of both the tag address and state and virtual synonym information. When a cycle is initiated by the CC the Tag RAM does not know whether tag or state information is on the bus. The CC controls which is written by asserting either TWE_ (Tag Write Enable) or STWE_ (State Write Enable). TBus timing is the same as for a tag address write. However, the format of the TBus changes. Note that TWE_ must be deasserted in clock 3 to assure that state information is written.

Figure 10-11 shows a timing diagram of a state write, which shows TBus bits 39:5 being driven by the CC.

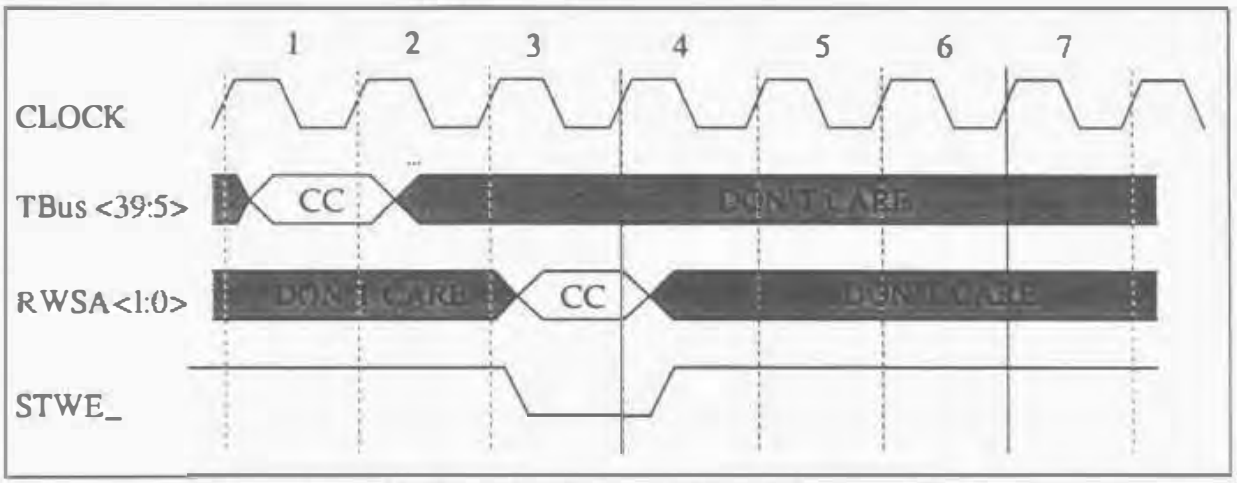


Figure 10-11 Tag RAM State and Virtual Synonym Write

Bit Description	Tbus bits
Write Data for Dirty Bit	39
Enable Virtual Synonym Write	38
Enable Sector 0 State and Dirty Bit Write	37
Enable Sector 1 State and Dirty Bit Write	36
Enable Sector 2 State and Dirty Bit Write	35
Enable Sector 3 State and Dirty Bit Write	34
Virtual Synonym	33:30
Sector 0 State Information	29:28 Odd
Sector 1 State Information	27:26 Odd
Sector 2 State Information	25:24 Odd
Sector 3 State Information	23:22 Odd
Unused	21:20
Tag RAM Index	19:9

Table 10-13 Tag RAM State and V.S. Write TBus Bit Definitions

Table 10-13 shows the orientation of TBus <39:5> during a state and V.S. write. The enable bits for each sector and dirty bit of each set allows for modification of data for any or all sets of a given tag RAM entry.

10.4.6 Even Tag RAM State and Virtual Synonym Read

When a state and virtual synonym read cycle is initiated by the CC, an index is sent across the TBus which selects one line of the Tag RAM. Note that the Tag RAM is 4-way set associative, hence each line contains four different sets of data. State information for all four sectors of a given set are read out, as well as the four bit virtual synonym value for that set. Each set contains 32 bits of information Which of the four sets is read is determined by the value of RWSA<1:0>. Figure 10-12 below shows a diagram of how set 0 is organized. The remaining three sets are organized in the exact same manner.

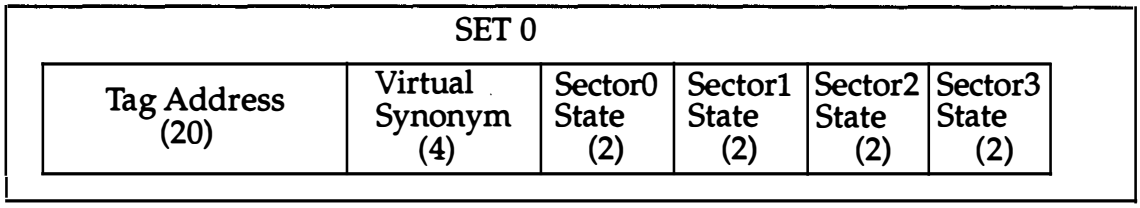


Figure 10-12 Tag RAM Set Organization

The format of the function field of the TBus is the same as for an even tag address read. The type of read cycle (tag address or state) is not differentiated in the TBus transfer. The CC controls this using hard-wired control pins. Figure 10-12 shows a timing diagram for the even Tag RAM state and V.S. read. Note that the corresponding data is returned in clock 9. For clocks 1-9 the TBus state machine must remain in the CC state. No pipelined cycles which may cause the TBus state machine to transition from the CC are allowed in clocks 2-9. The TBus "Don't Care" portion in clocks 2-9 must be CC cycles.

Also note that TBus <19:5> is driven for two consecutive clocks. The Dirty Bit RAM inside the Tag RAM is physically separate from the actual Tag RAM portion of the device and a there is a register between the two devices. Hence it takes one clock longer for an access to the Dirty Bit RAM to occur. For cycles where the reading of the Dirty Bit RAM is required, the TBus information must be driven for two clocks. If the dirty bit RAM information is not required it is only necessary to drive the TBus for one clock. Clock 1 in the diagram can be eliminated.

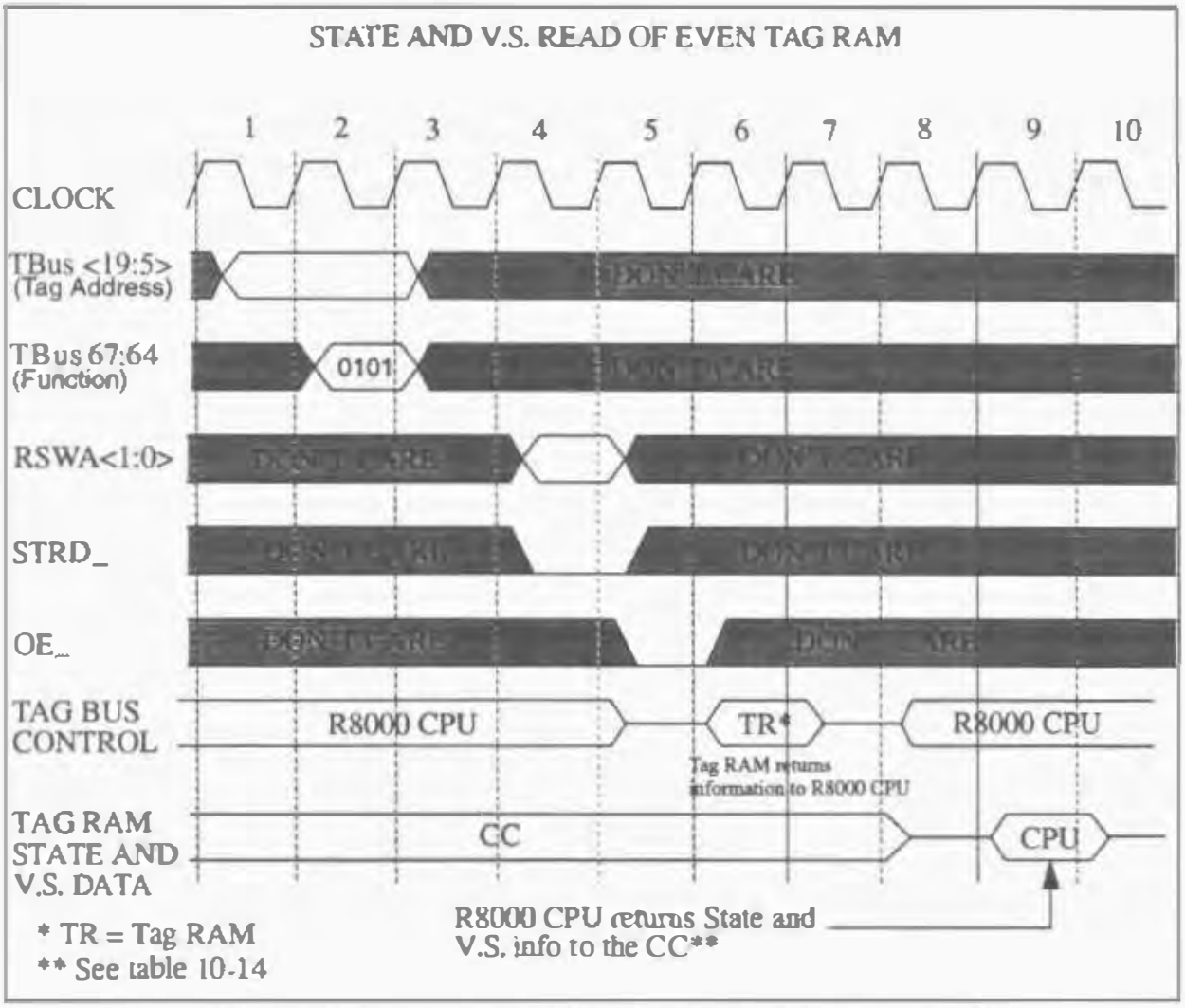


Figure 10-13 Even Tag RAM State and Virtual Synonym Read

The RWSA<1:0> field is driven by the CC two clocks later in clock 4 to allow the R8000 CPU adequate time to route the index information to the Tag RAM. STRD_ (State Read) is also asserted in clock 3 to enable the read information out onto the Tag RAM's internal tag bus.

The encoded value in the function field indicates to the R8000 CPU the type of cycle being performed and indicates the appropriate action to be taken. In clock 5, three clocks after receiving the function field, the R8000 CPU tri-states the tag address bus. Also in clock 5 the CC asserts OE_ which drives the requested information onto the tag pins. The OE_ signal is pipelined and is driven one clock before the information is actually driven out by the Tag RAM. Even though the OE_ pin is deasserted in clock 6 it is effectively still active in clock 6 as the Tag RAM is driving information to the R8000 CPU.

In clock 6 the Tag RAM drives the requested information back to the R8000 CPU. This

operation is transparent to the cycle and is shown only for clarity of flow through the devices. In clock 7 the Tag RAM tag bus is tri-stated and is controlled by the R8000 CPU in clock 8.

By the end of clock 8 the state information has propagated through the R8000 CPU and is returned to the CC in clock 9. In clock 10 the R8000 CPU relinquishes control of the TBus back to the CC. The behavior of the state machine during a state read is the same as during a tag address read. Refer to section11.4.2 for an explanation of state machine behavior.

When the Tag RAM is accessed 20 bits are returned to the R8000 CPU. When the R8000 CPU returns the information to the CC all 40 bits are driven. Table 10-14 below shows the bit orientation of the TBus during a state read. The orientation is the same for both even and odd state read cycles.

Although single reads of either the even or odd tag RAM's are provided, these functions are normally used only for diagnostic purposes. Under normal conditions a Combined read of both Tag RAM's is the most desirable so that comparison of the state and dirty bit information from both devices can be done inside the R8000 CPU.

Bit Description	Tbus Bits
Undefined	39:38
Tag RAM Sector 0 Dirty Bit	37
Tag RAM Sector 0 Dirty Bit	36
Tag RAM Sector 0 Dirty Bit	35
Tag RAM Sector 0 Dirty Bit	34
Virtual Synonym Data	33:30
Sector 0 State Information	29:28
Sector 1 State Information	27:26
Sector 2 State Information	25:24
Sector 3 State Information	23:22
Unused	21:20
Undefined	19:5

Table 10-14 Tag RAM State and V.S. Read TBus Bit Definitions

10.4.7 Odd Tag RAM State and Virtual Synonym Read

The timing for an odd Tag RAM state read is the same as for the even Tag RAM state read explained in section 10.4.6 above. The only difference is the bit orientation of the function field, which is encoded to indicate a read of the Tag RAM.

Refer to Figure 10-14 below. The RWSA<1:0> field is driven by the CC two clocks later in clock 3 to allow the R8000 CPU ad equate time to route the index information to the Tag RAM. STRD_ (State Read) is also asserted in clock 3 to enable the read information out onto the Tag RAM's internal tag bus.

The encoded value in the function field indicates to the R8000 CPU the type of cycle being performed and indicates the appropriate action to be taken. In clock 5, three clocks after receiving the function field, the R8000 CPU tri-states the tag address bus. Also in clock 5 the CC asserts OE_ which drives the requested information onto the tag pins. The OE_ signal is pipelined and is driven one clock before the information is actually driven out by the Tag RAM. Even though the OE_ pin is deasserted in clock 6 it is effectively still active in clock 6 as the Tag RAM is driving information to the R8000 CPU.

In clock 6 the Tag RAM drives the requested information back to the R8000 CPU. This operation is transparent to the cycle and is shown only for clarity of flow through the devices. In clock 7 the Tag RAM tag bus is tri-stated and is controlled by the R8000 CPU in clock 8.

By the end of clock 8 the state information has propagated through the R8000 CPU and is returned to the CC in clock 9. In clock 10 the R8000 CPU relinquishes control of the TBus back to the CC. The behavior of the state machine during a state read is the same as during a tag address read. Refer to section 11.4.2 for an explanation of state machine behavior.

When the Tag RAM is accessed 20 bits are returned to the R8000 CPU. When the R8000 CPU returns the information to the CC all 40 bits are driven. Table 10-15 below shows the bit orientation of the TBus during a state read. The orientation is the same for both even and odd state read cycles.

Although single reads of either the even or odd tag RAM are provided, these functions are normally used only for diagnostic purposes. Under normal conditions a Combined read of both Tag RAM's is the most desirable so that comparison of the state and dirty bit information from both devices can be done inside the R8000 CPU.

Refer to Table 10-14 above shows the bit orientation of the TBus during an odd Tag RAM state read. The orientation is the same for both even and odd state read cycles.

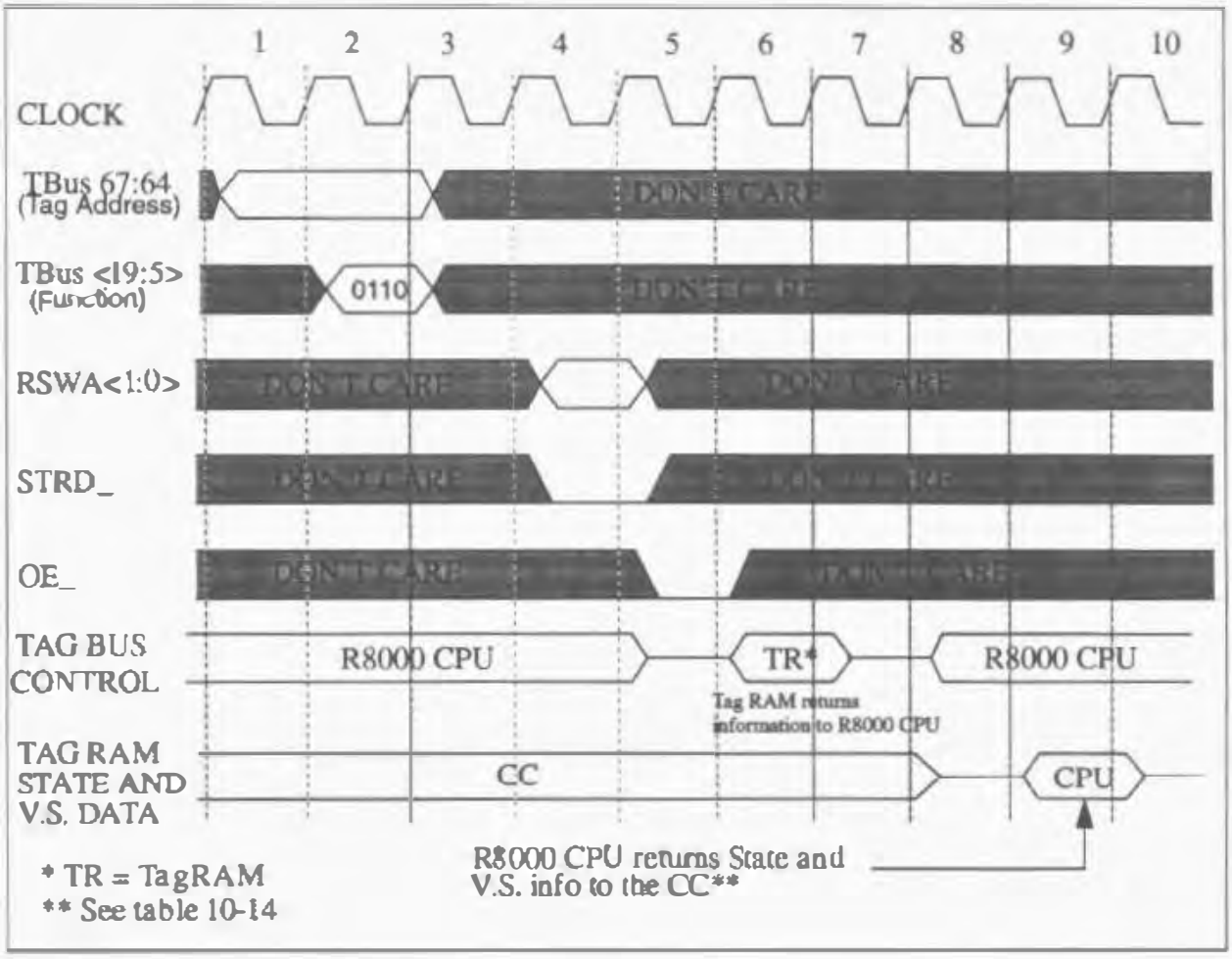


Figure 10-14 Odd Tag RAM State and Virtual Synonym Read

10.4.8 Combined Tag RAM State and Virtual Synonym Read

The combined operation is provided so that the dirty bit information from both the even and odd Tag RAM's can be read in the same cycle.

For a combined read cycle the TBus <19:5> index must be driven by the CC for two consecutive clocks, shown as clocks 1 and 2 in Figure 10-15 below. The extra clock is required because inside the Tag RAM, the dirty bit RAM is physically separate from the address and state RAM and there is a register in the path which separates the two RAM's.

The function field is encoded to indicate to the R8000 CPU a combined read function. The state of the function field informs the R8000 CPU where to place the information on the TBus. When the Tag RAM's are accessed, the information from the 20 bit tag bus is placed onto TBus <37:18>.

Figure 10-15 below shows a timing diagram for a combined state read.

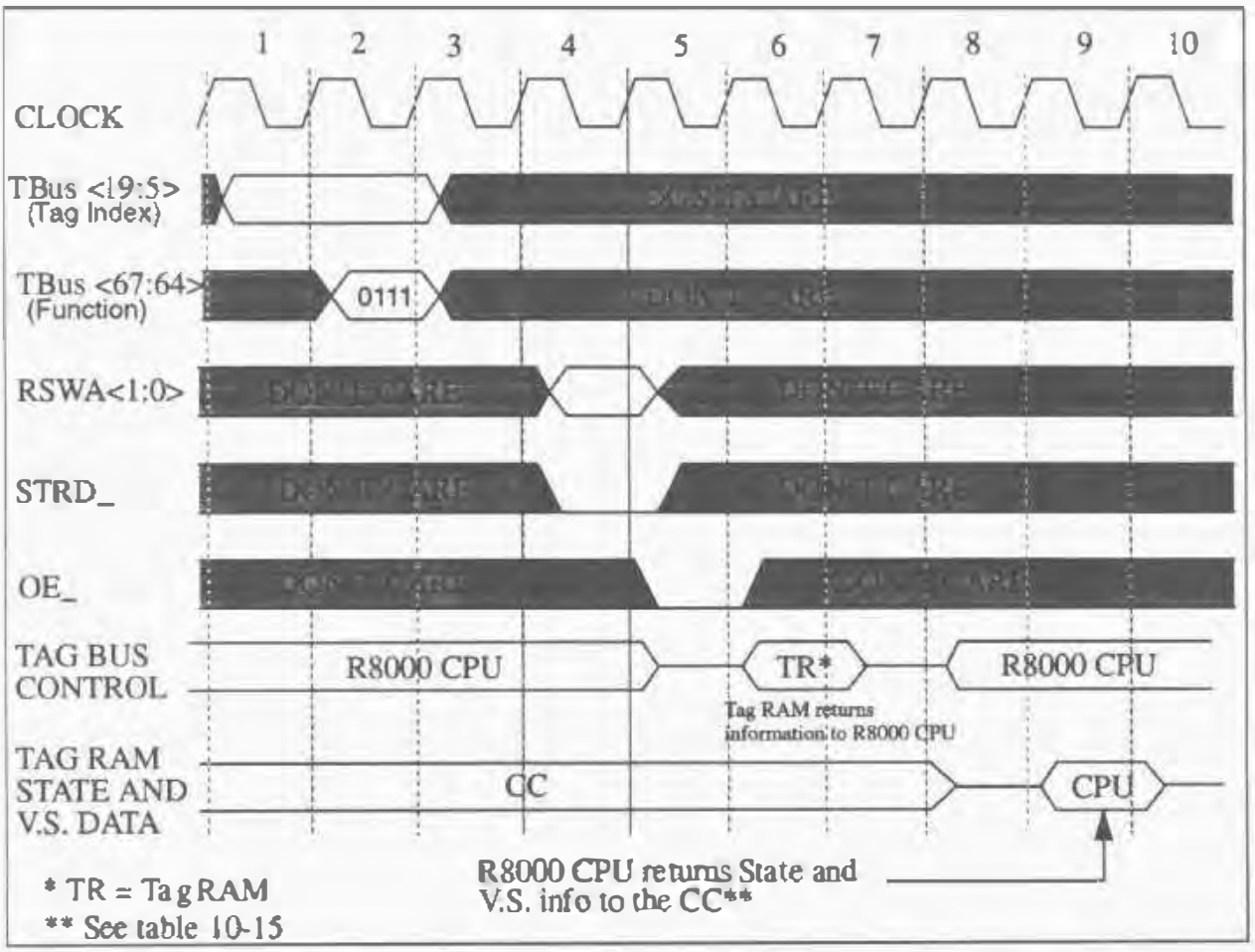


Figure 10-15 Combined Tag RAM State and Virtual Synonym Read

The sequence of signal generation and the returning of data back to the CC is the same as for the even and odd state read cycles in sections 10.4.6 and 10.4.7. When data is returned to the CC all 40 bits of the physical address field (TBus <39:0>) are driven by the R8000 CPU. State and virtual synonym information are read from the odd Tag RAM by default.

Table 10-15 below shows the bit orientation of TBus<39:0>.

Bit Description	Tbus bits
Undefined	39:38
Odd Tag RAM Sector 0 Dirty Bit	37
Odd Tag RAM Sector 1 Dirty Bit	36
Odd Tag RAM Sector 2 Dirty Bit	35
Odd Tag RAM Sector 3 Dirty Bit	34
Virtual Synonym Data	33:30 Odd
Sector 0 State Information	29:28 Odd
Sector 1 State Information	27:26 Odd
Sector 2 State Information	25:24 Odd
Sector 3 State Information	23:22 Odd
Even Tag RAM Sector 0 Dirty Bit	21
Even Tag RAM Sector 1 Dirty Bit	20
Even Tag RAM Sector 2 Dirty Bit	19
Even Tag RAM Sector 3 Dirty Bit	18
Undefined	17:0

Table 10-15 Combined Tag RAM State and V.S. Read TBus Bits

10.4.9 Store Address Queue Compare

The purpose of a store address queue compare operation is to determine whether the address placed on the bus by the CC compares to any of those which are in the Store Address Queue (SAQ). The SAQ is located in the R8000 CPU and contains addresses which have already been verified in the Tag RAM and are known to be in the Streaming Cache. Most of the time these addresses are in the queue because the data corresponding to them has not yet become available.

The SAQ stores the entire streaming cache address including the set information, which it uses to write the streaming cache. For coherence reasons the contents of the SAQ are considered to be in the streaming cache. Therefore, whenever there is a CC initiated read of the streaming cache, the SAQ must also be checked. There are two reasons why the CC reads the streaming cache.

1) Intervention
2) Write back or line replacement from the streaming cache to main memory.

Intervention occurs when another processor wants data in the streaming cache which is exclusive. The line may either be clean or dirty. If the line is clean the CC will allow the external agent to retrieve the data from the streaming cache. If the line is dirty the CC must place the correct data in the cache before allowing the external agent to perform the read.

When a write-back cycle from the streaming cache to main memory is to be performed, a SAQ operation is performed by the CC first to assure that the address corresponding to the requested data is not in the queue. This way the CC assures that the most up to date information is written out to main memory.

If the SAQ check is valid the CC causes the TBus state machine will transition from CC state to EQ state based on the state of the function field of the TBus. The transition allows control of the streaming cache to be given back to the R8000 CPU so that the queue may be emptied. Transition to the empty queue (EQ) state does not mean that the queue is flushed, but rather that all of the cycles in the queue are allowed to finish. Once the queue is emptied control of the bus is given back to the CC and the write-back to main memory can be completed.

In the SAQ operation the CC places address bits [17:7] onto the TBus Tag RAM address field. This value is then compared with bits [17:7] of each entry in the SAQ. Bits [6:0] are not checked due to the 128 byte sector size of the streaming cache. Four clocks are required to check both the right and left SAQ's to determine whether there is a valid compare. The result of the compare is returned to the CC using the **SAQE_** and **SAQO_** pins. These pins are hard-wired between the CC and the R8000 CPU, hence there is no information which need be returned by the R8000 CPU through the TBus. If the requested address compares to any one of the values in either SAQ the CC cannot perform a streaming cache cycle because the address corresponding to the data for that location is in the SAQ and has not yet been written out. So the CC must wait.

During a SAQ compare operation a condition can occur in which the address thought there was a valid compare but there really was not. This condition is called a 'false positive'. The R8000 Microprocessor Chip Set defines a sector as 128 bytes. If the address compare is for a given 32 byte value, and a store did not exist in this 32 bytes, but one of the other 96 bytes out of the 128 bytes does contain a store, a false positive occurs even though the store does not exist in the given 32 byte value being compared. When a false positive does occur the SAQ must be emptied.

If either **SAQE_** or **SAQO_** is returned active the CC releases **CCREQ_** and places the Empty Queue operation on the function pins of the TBus. This activity causes the TBus state machine to transition from CC state to EQ state, granting control of the bus back to the R8000 CPU. The CC then waits for the queue to empty, at which time the R8000 CPU asserts the signal IUREL_. Assertion of this signal causes the state machine to transition

back to the CC state and the CC resumes execution of the cycle.

Figure 10-16 below shows a block diagram of the Store Address Queue operation.

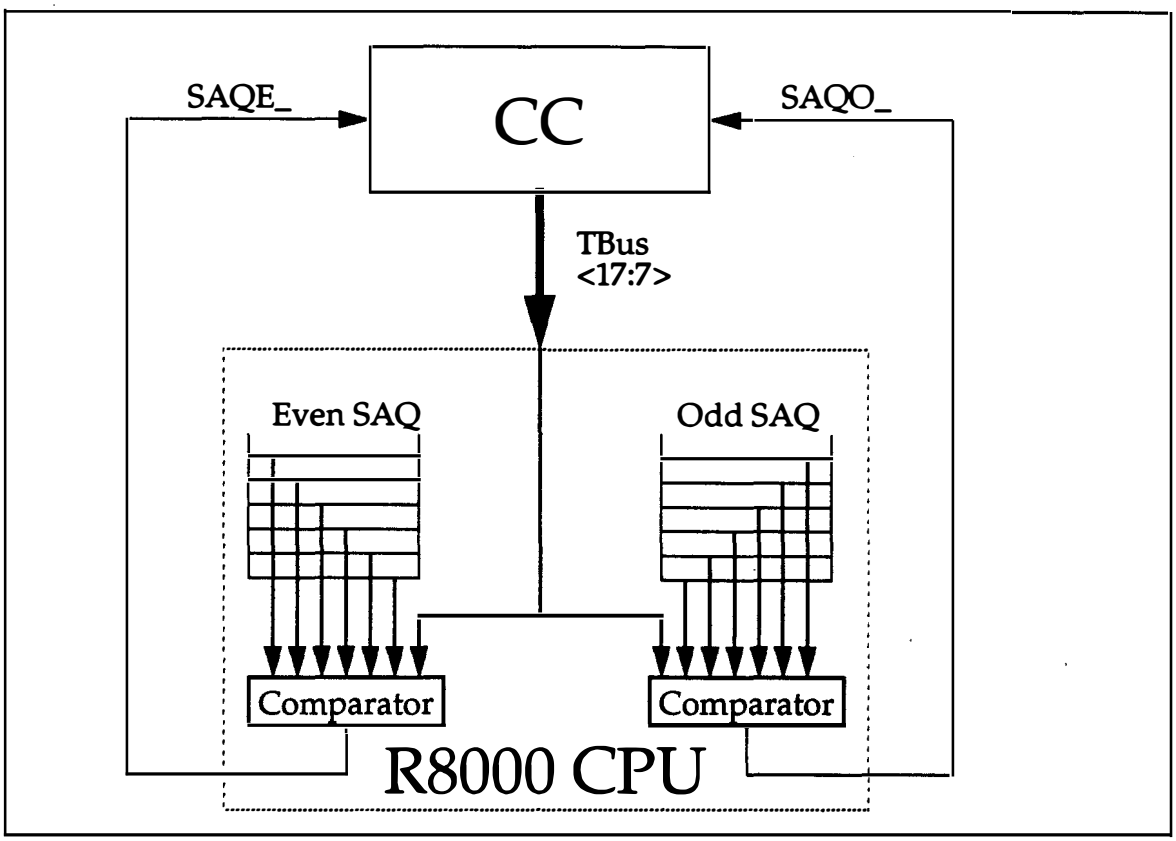


Figure 10-16 Store Address Queue Compare Operation

From the time the address information is placed on the TBus by the CC, four clocks are required to perform the SAQ compare operation. The results of the compare are returned to the CC by **SAQE_** and **SAQO_**. Figure 10-17 below shows a timing diagram of the SAQ compare operation. For clarity the signals **SAQE_** and **SAQO_** are both shown as active in clock 5 of the diagram. In actuality the signals act independently of one another and can be in either the asserted or deasserted state. In addition, due to the pipelined nature of the R8000 Microprocessor, a different SAQ operation could be performed in each clock of the timing diagram below. Therefore, the signals **SAQE_** and **SAQO_** can be active in clocks 1-4 as well as clocks 6-7 returning the status of SAQ compare operations pertaining to other cycles.

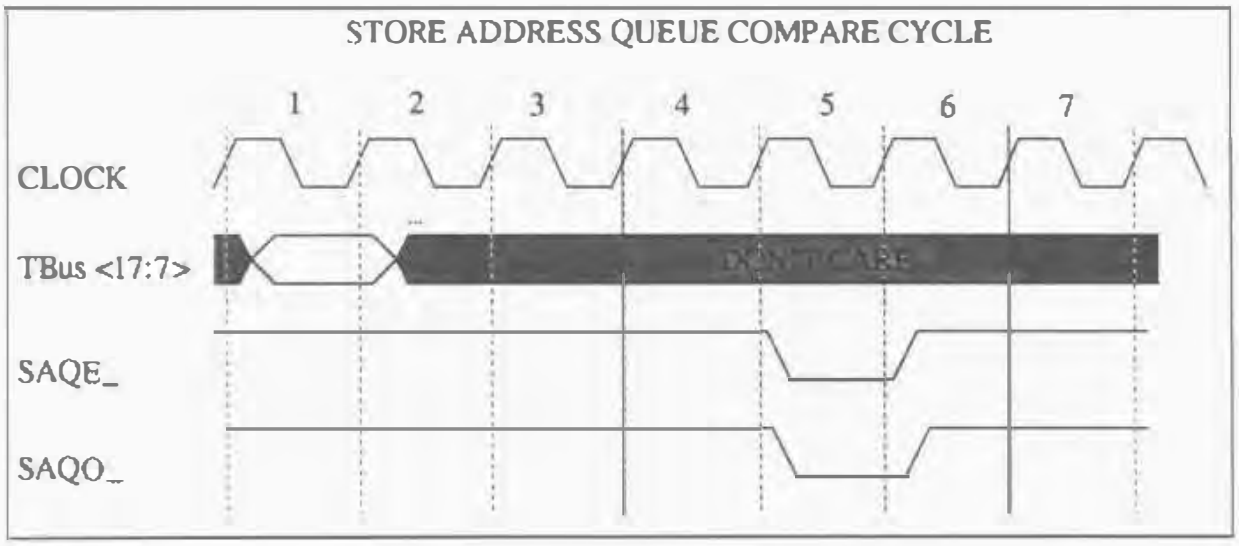


Figure 10-17 Store Address Queue Compare

10.4.10 Data Cache Invalidate

Data Cache invalidate operations are performed by the R8000 CPU as instructed by the CC and are done for the same reasons as the SAQ operation explained in section 11.4.9, intervention and write back or line replacement. The invalidate operation is unidirectional in that no information is returned by the R8000 CPU. The CC passes the necessary information required to perform the invalidation across the TBus. No handshake mechanism exists to inform the CC that the operation was performed.

Each invalidate cycle performed by the CC invalidates 32 bytes, or one line, of the Data cache. This equates to 16 bytes from the left port and 16 bytes from the right port of the dual-ported data cache. Because the data cache Valid RAM contains a valid bit for every 32 bits of data, and there are 4 valid bits per Data cache Valid RAM entry. Thirty two bytes corresponds to 2 entries of the Data cache Valid RAM. Refer to chapter 1 for more information on Data Cache Valid RAM organization.

Figure 10-18 below shows a timing diagram for data cache invalidation.

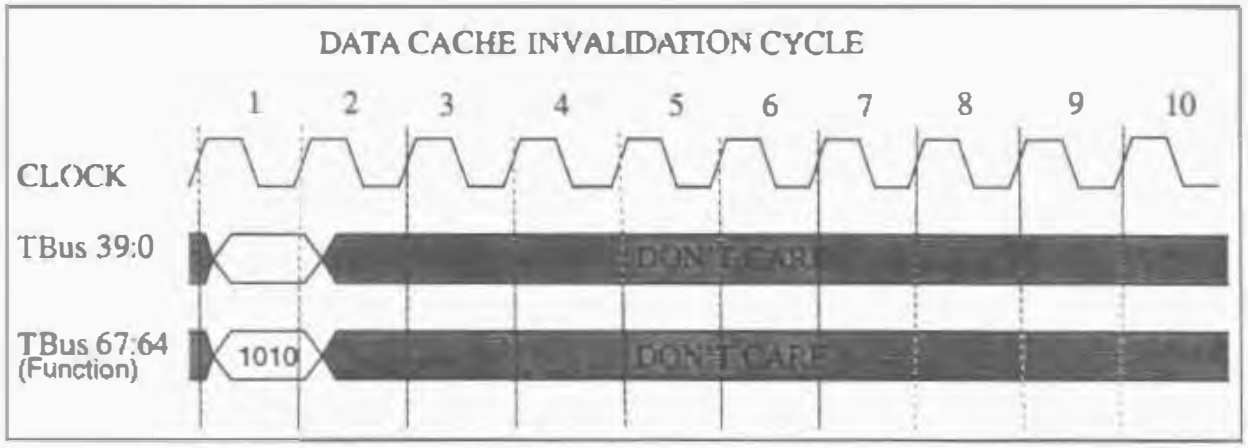


Figure 10-18 Data Cache Invalidation

The CC drives TBus <39:0> during a Data Cache Invalidate operation. These bits encompass the Tag RAM address field (TBus <39:5>) and the virtual synonym field (TBus <3:0>). Bit 4 is unused in the operation. Figure 10-19 below shows the bit orientation of the TBus during a data cache invalidation.

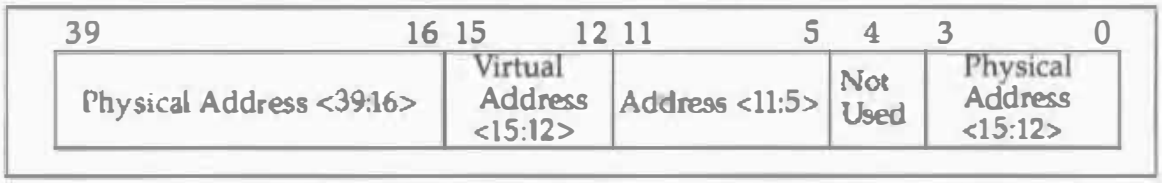


Figure 10-19 TBus Bit Orientation during a Data Cache Invalidation

10.4.11 Streaming Cache Data Write

The CC performs a streaming cache write in situations where it is necessary to transfer data from the system bus data buffers to the streaming cache. This normally occurs whenever the R8000 CPU incurs a miss to the streaming cache. The R8000 CPU transfers ownership of the TBus to the CC which allows the CC to retrieve the desired locations from main memory.

In order to write the streaming cache the CC must supply all of the necessary signals. Address, External Set Address and write enable information are all sent across the TBus simultaneously. In addition the CC must tri-state the store data bus buffers of the R8010 FPU to allow the data buffers to drive data onto the store data bus. This is accomplished by deassertion of the FOE_ pin, which is a hard-wired signal between the CC and the

R8010 FPU. Normally the signal FOE_ is asserted, allowing the R8010 FPU control of the store data bus. Refer to figure 10-22 for more information on how FOE_ is used. Figure 10-20 below shows a block diagram of the data bus interfaces in the R8000 Microprocesor.

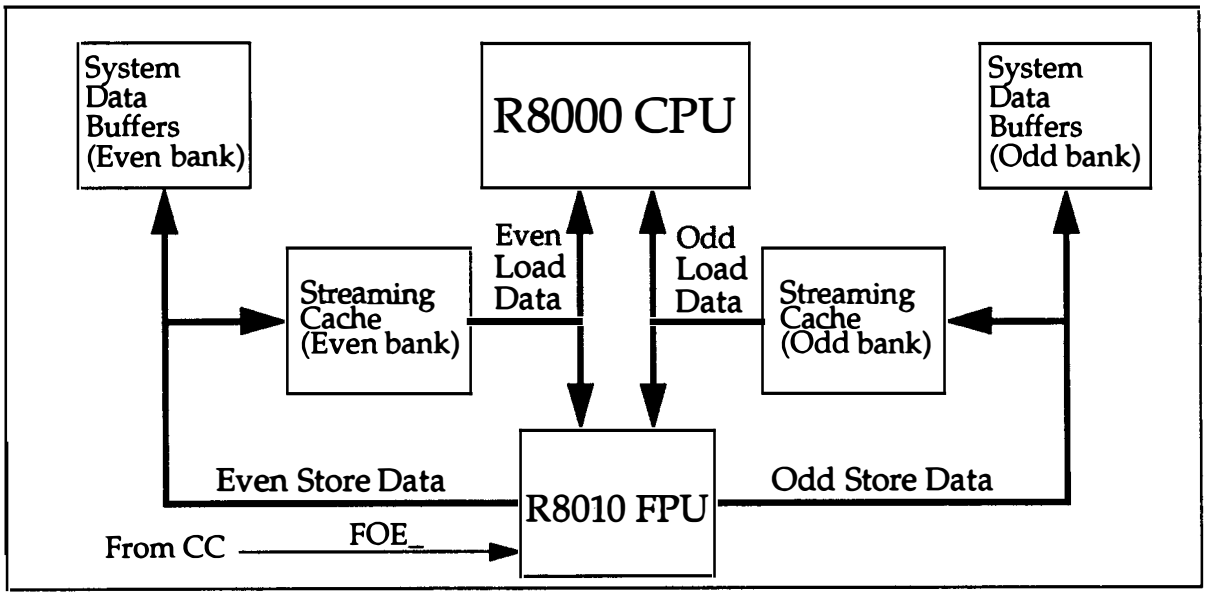


Figure 10-20 Data Bus Organization

Once ownership of the store data bus is granted to the system data buffers, 16 bytes can be written on each cycle and a write enable exists for every 4 bytes. Figure 10-21 shows a timing diagram of a Streaming Cache Data Write

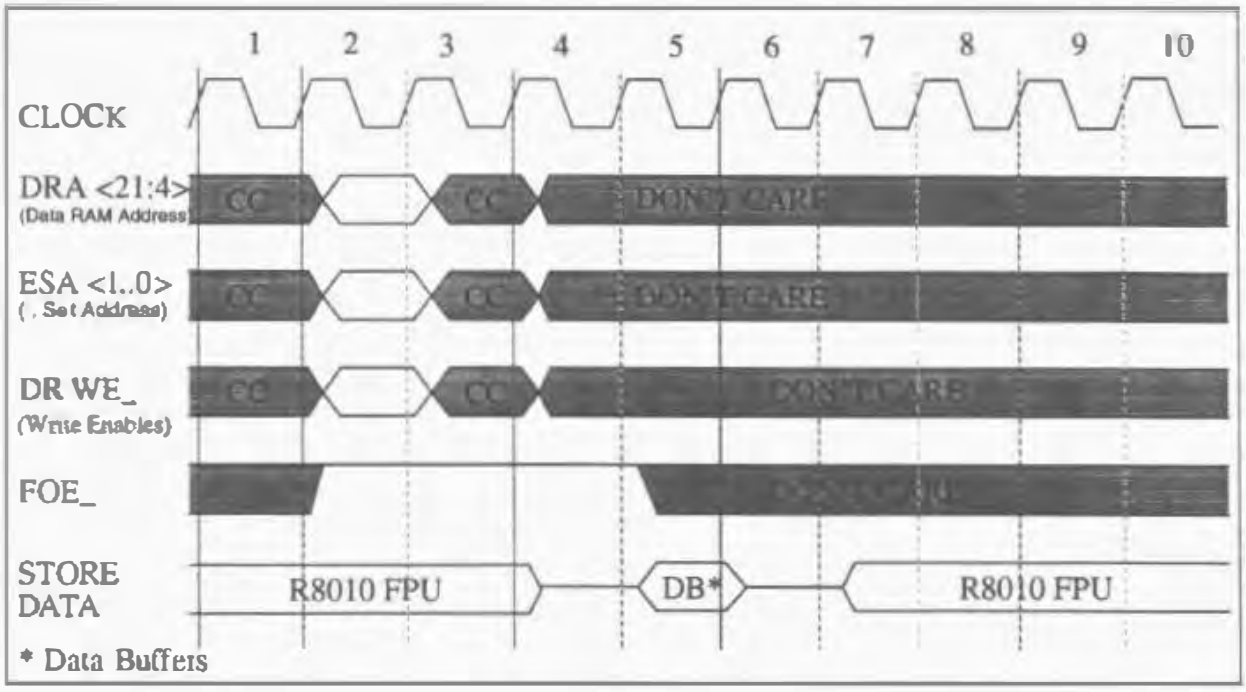


Figure 10-21 Streaming Cache Data Write

Address, ESA, and write enable information are sent across the TBus in clock 1. Note that there is a two clock delay (internal to the R8010 FPU) between the time when FOE_ is deasserted by the CC and when the R8010 FPU actually tri-states the store data bus. In cycles 1 and 3 the TBus is shaded and labeled 'CC', indicating that these cycles must be CC cycles.

The actual data is driven by the data buffers in clock 5. FOE_ is also asserted in clock 5. The two clock delay allows the R8010 FPU to again begin driving the store data bus in clock 7.

10.4.12 Streaming Cache Data Read

Streaming cache data reads by the CC occur when another processor wishes to obtain data in the streaming cache. Note in figure 10-20 that there is no dedicated read data bus between the data buffers and the streaming cache. Instead, a streaming cache data read cycle requires that data pass from the load data bus of the streaming cache to the R8010 FPU. The CC asserts the signal BYPASS_ to allow this load data to be routed from the load data bus of the R8010 FPU onto the store data bus of the R8010 FPU. This operation occurs internal to the R8010 FPU and requires two clocks. The FOE_ and BYPASS_ mechanism is shown in Figure 10-22.

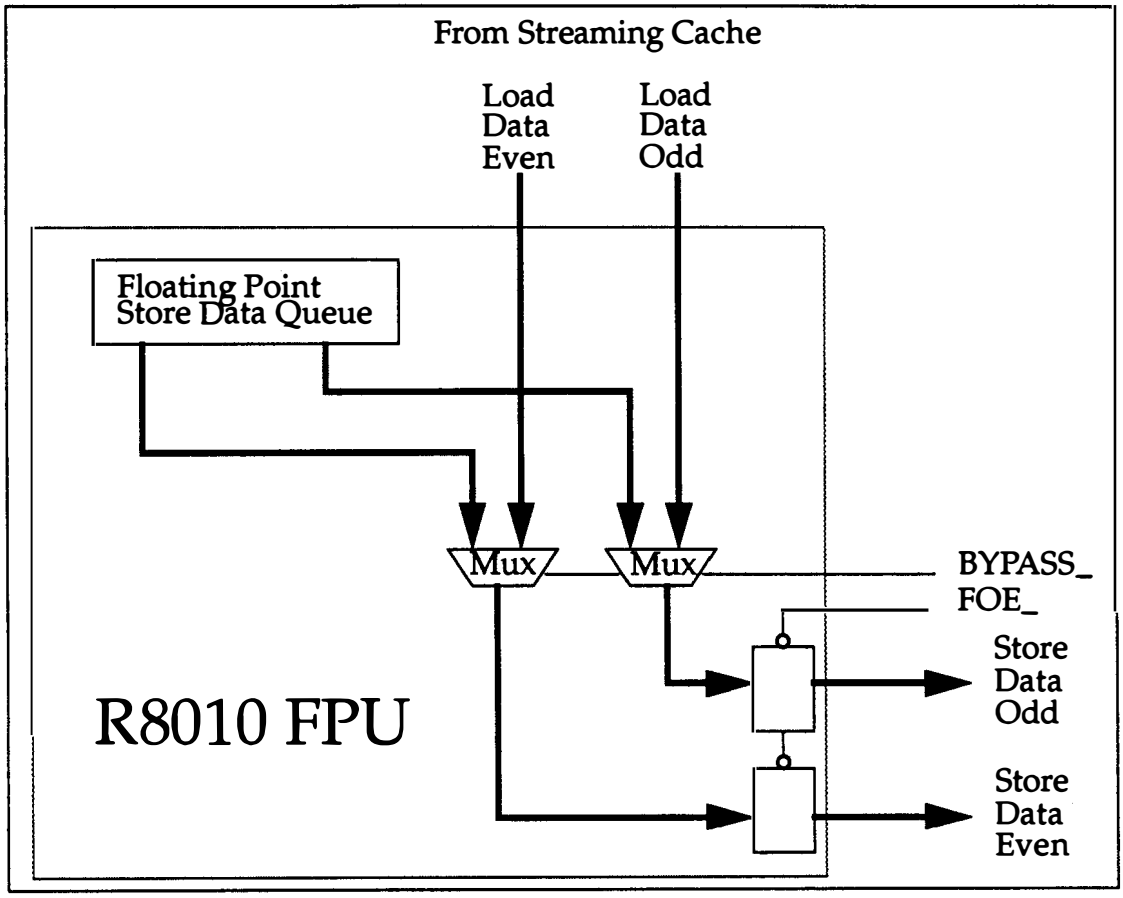


Figure 10-22 Functions of the BYPASS_ and FOE_ pins

Figure 10-23 shows a timing diagram of a streaming cache data read. Cycles 1-4 must be CC cycles. Sixteen bytes may be read on each cycle. All write enable pins must be deasserted at this time.

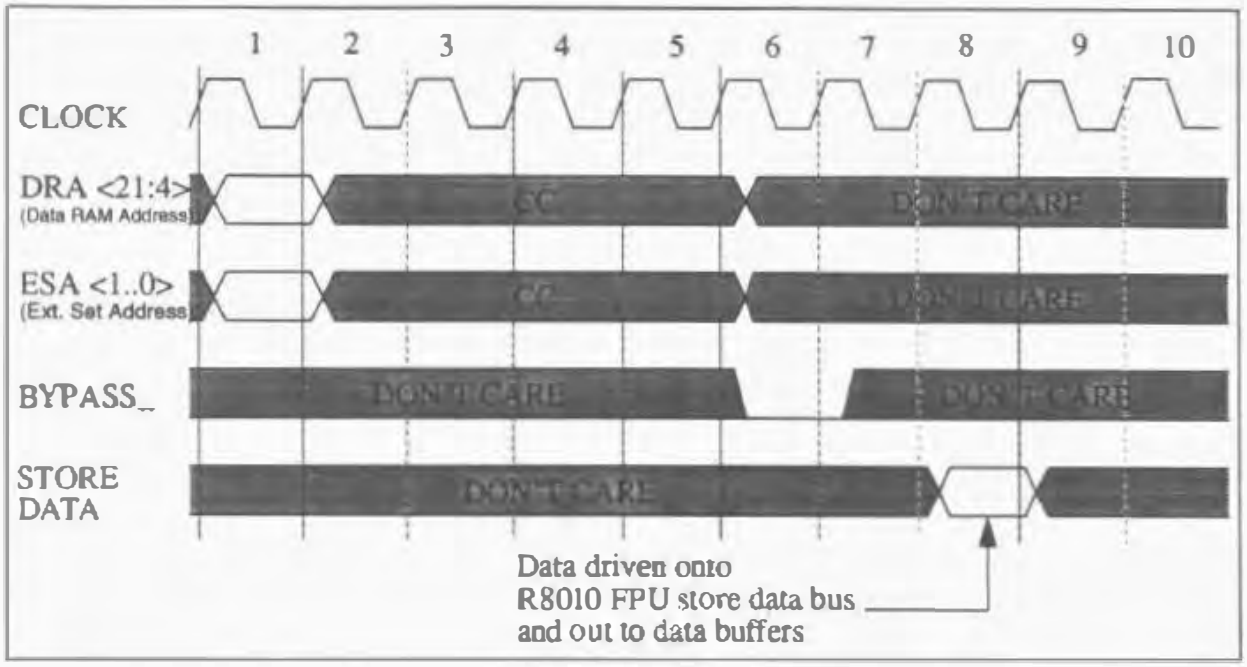


Figure 10-23 Streaming Cache Data Read

10.4.13 Interrupt Status

The CC can pass interrupt status information to the R8000 CPU during any cycle in which the TBus state machine is in the CC state (CC is in control of the TBus). The interrupt status cycle is necessary because the R8000 CPU does not have any dedicated interrupt pins. Interrupts generated in the system are sent to the interrupt status registers in the CC. Whenever the register is updated, the CC encodes the function field indicating to the R8000 CPU that the interrupt register has been updated. Hence the TBus actually functions as the interrupt pin mechanism of the R8000 CPU. The R8000 CPU then accesses the interrupt registers inside the CC to determine which interrupt was set and vectors to the appropriate interrupt service routine.

Interrupt status is transferred across TBus bits <39:29> whenever the function field indicates the interrupt function. Table 10-16 shows the TBus bit orientation during interrupt status transfer.

IP Enable	NMI Enable	Bus Error Enable	IP Field	NMI Field	Bus Error Field
TB<39>	TB<38>	TB<37>	TB<36:31>	TB<30>	TB<29>

Table 10-16 TBus Interrupt Status Transfer

10.5 R8010 FPU TBUS PROTOCOL

Refer to figure 10-1. Note that TBus bits TB<79:72> connect only between the R8000 CPU and the R8010 FPU. No CC connection to these bits exists. The following section requires no intervention by the CC and is explained only for clarity. The R8010 FPU TBus protocol is specific between the R8000 CPU and the R8010 FPU. The protocol is system independent and has no parameters which can be modified in any way.

Instructions are dispatched by the R8000 CPU to the R8010 FPU through the TBus. There are four basic types is transmissions which are differentiated by encoding the uppermost two bits of the TBus (TB<79:78>). Table 10-17 shows the TBus format for the four types of transmissions. Each of these is explained in more detail below.

TBus Bits	7978	77	76	75	74	73 65	64	63 56	55 28	27 0
Normal	1 0	Vma	Vmb	Vfa	Vfb	MemSpecA	MemSpecB		FpOP-A	FpOP-B
MoveFrom	0 1	Vmf	Vmb	Vfa	Vfb	MfSpec	MemSpecB		FpOP-A	FpOP-B
IntStore	0 0	1	----	---	---	IStSpec	-	Data		
MoveTo	1 1	1	----	---	---	MtSpec	-	Data		

Table 10-17 R8010 FPU TBus Protocol

Legend:

FpOP-A = Floating Point Operand A
FpOP-B = Floating Point Operand B
MemSpec-A = Memory Specifier A
MemSpec-B = Memory Specifier B
MfSpec = MoveFrom Specifier
IStSpec = Integer Store Specifier
MtSpec = MoveTo Specifier
Vma = Memory Specifier A (bits 73:65) Valid
Vmb = Memory Specifier B (bits 64:56) Valid
Vfa = Floating Point Operation A (bits 55:28) Valid
Vfb = Floating Point Operation B (bits 27:0) Valid
Vmf = MoveFrom Specifier Valid

10.5.1 Normal Transfer

A normal dispatch contains two FP arithmetic operations, each 28 bits wide, and two FP memory operations, each 9 bits wide. There are roughly 30 FP arithmetic operations which can be dispatched by the R8000 CPU. The 28 bit TBus format for arithmetic operations is different for each operation. For FP memory operations, the 9 bit value contains Floating Point Register destination as well as data alignment information. Each of the four potential instructions contains a valid bit associated with it, denoted by bits <77:74> in the table. Setting this bit indicates to the R8010 FPU that a given instruction is valid and should be executed.

10.5.2 MoveFrom Transfer

MoveFrom is similar in format to Normal mode except that the FP memory operation normally on TBus bits 73:64 is substituted with a move specifier. This operation moves data from a FP register to a general purpose register (GPR) in the R8000 CPU and is the only time which the R8010 FPU drives the TBus. Bit 77 indicates whether the MoveFrom specifier is valid.

10.5.3 IntStore Transfer

IntStore - The IntStore operation supports integer stores to the streaming cache. As shown in figure 1-1 of chapter 1, there is no direct path for integer stores from the R8000 CPU to the streaming cache. Instead they are transmitted across the TBus and out onto the store data pins of the R8010 FPU. In IntStore mode the TBus contains the 64 bit integer data along with some store alignment information.

10.5.4 MoveTo Transfer

The MoveTo operation moves data from a General Purpose Register (GPR) in the R8000 CPU to a Floating Point Register. The MoveTo format is similar to the IntStore format except that instead of store alignment information, TBus bits [73:65] contain the FPR destination. The 64 data is transmitted on TBus pins [63:0].

RESPONSIBILITIES OF THE CACHE CONTROLLER

11

The R8000 Microprocessor Chip Set has large first and second level caches which minimize the need for interfacing to external main memory. Main memory is typically slow and when accessed frequently can degrade the overall performance of the processor. To maximize performance, the Integer Unit interfaces only to separate 16 KByte on-chip instruction and data caches and the 4 MByte streaming cache. The Floating Point Unit interfaces only to streaming cache. Neither device initiates cycles or interfaces directly to the main memory. However, there are times when interfacing to the external main memory is necessary.

The Cache Controller is a stand-alone device which manages the interface between the R8000 Microprocessor Chip Set, main memory, and the system back-plane. The cache controller was not included as part of the R8000 Microprocessor Chip Set in order to allow the designer maximum flexibility in memory and overall system design.

Although some hardwired pins must be provided by the Cache Controller (CC), the majority of communication between the R8000 CPU and the CC is done via the 72 bit TBus. The TBus protocol is complex and changes depending on whether the R8000 CPU or the Cache Controller is driving. Refer to chapter 10 for more information on TBus protocol.

Cache Controller designs will differ greatly from system to system. This chapter offers

some examples of how a cache controller might handle some commonly used bus transfers.

In general the Cache Controller is required to manage:

1) The fetching of data from main memory after a streaming cache miss or as instructed by the Integer Unit.

2) The write-back of dirty data from the streaming cache to main memory.

3) Modification of the streaming cache Tag RAM tag address, state, and virtual synonym information.

4) Invalidation of the first level data cache (inside the R8000 CPU) when the streaming cache is modified in order to assure coherency between caches.

5) All coherence issues between the various streaming caches in a multiprocessor system.

6) The filtering of coherence activity such that the processor is protected from unnecessary interruptions.

7) Checking of the Store Address Queue (inside the R8000 CPU) to assure that the most up to date data is transferred.

8) On board local registers for interrupt prioritizing and management.

The system bus interface contains data and address buffer devices as well as a third Tag RAM responsible for bus snooping to maintain coherency between processors. Although use of a third Tag RAM is not required it is highly recommended in order to allow the two Tag RAM's which support the streaming cache to maintain a single cycle access rate. Forcing either of these two Tag RAM's to support the streaming cache as well as bus snooping and back-plane monitoring could severely hamper overall system performance. Control of these devices must be provided by the CC. In addition the CC interfaces to all system I/O devices. If a boot PROM is used in the system the CC is responsible for moving PROM data into the data cache to facilitate the boot-up procedure.

11.1 STREAMING CACHE DATA MANAGEMENT OVERVIEW

On R8000 CPU misses to the streaming cache the R8000 CPU informs the CC that the requested data was not available. Control of the bus is then transferred to the CC. Since neither the R8000 CPU or the R8010 FPU communicate directly with external main memory, it is the responsibility of the CC to fetch the requested data from main memory and

place it in the streaming cache. Once the cycle is completed the CC relinquishes control of the bus back to the R8000 CPU. The R8000 CPU then fetches the requested data from the streaming cache and execution resumes.

In addition to streaming cache misses, the CC is responsible for monitoring the cache coherency attributes of each line in the cache. This is done via the 16 bit Dirty Bit RAM portion of the Tag RAM as well as the Tag RAM itself. If the state of the line in the cache changes, the state information in the Tag RAM must also be updated to reflect the change. If the state of the line remains the same and only the modified status changes, only the dirty bit is accessed. State information remains the same. Changing the state information is the responsibility of the CC. Updating of the dirty bit RAM is done by the R8000.

Since the first level cache is write-through, all writes by the R8000 to the first level Data cache are also written out to the streaming cache. If a write is executed to a line which has already been modified, the CC forces the R8000 to halt the cycle, whereby the CC takes control of the TBus. The modified data is then written out to main memory and control of the TBus returned to the R8000.

11.2 TAG RAM MANAGEMENT OVERVIEW

The Cache Controller is responsible for monitoring and updating of the Tag RAM. The R8000 performs only Look-up cycles to the Tag RAM. Lookups are done when the R8000 CPU wishes to read or write the streaming cache and desires to know state, virtual synonym, or set information corresponding to that line. With the exception of the dirty bits the R8000 CPU cannot update the contents of the tag RAM. Tag RAM management is the responsibility of the CC.

The CC can write either Tag information, or state and virtual synonym information to the Tag RAM in a given cycle. The Tag address is multiplexed with either tag or state and virtual synonym information. When the tag address is updated, the 20 bit tag bus contains all address bits. When the state and virtual synonym information is updated, the tag bus conforms to a specific bit orientation. The bit orientation changes for state and virtual synonym read cycles. Whenever there is a streaming cache miss, or when a line of the streaming cache is to be written out to main memory, the Tag RAM information must be updated to reflect this change. The CC must supply all necessary control signals to the Tag RAM.

Figure 11-1 and Table 11-1 below show the TBus state machine and the encoding of the function field respectively. Both of these diagrams should be used for reference as they are referred to frequently throughout the remainder of this chapter in the explanation of each timing diagram.

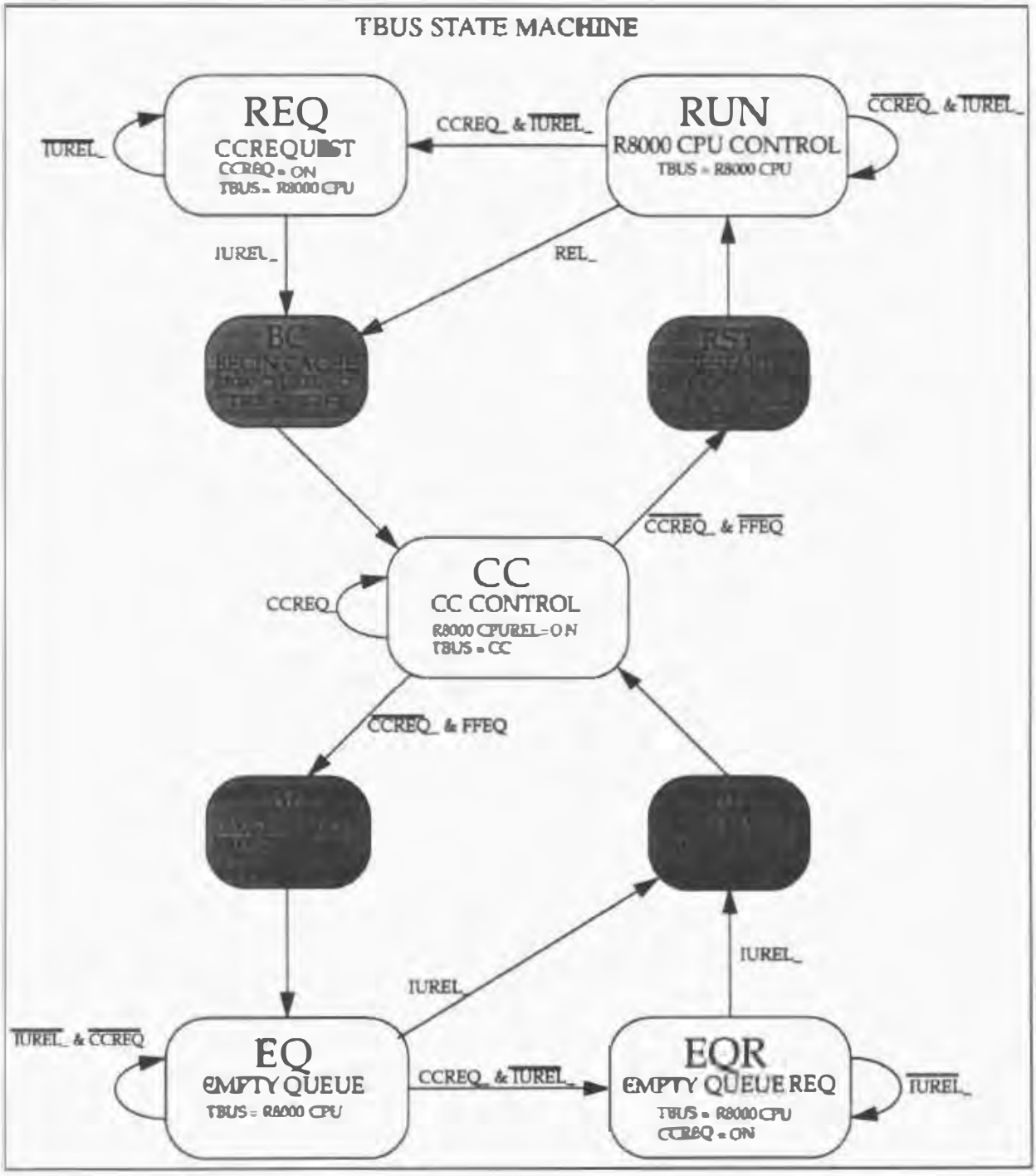


Figure 11-1 TBus State Machine

Encoded Value	Description
0	No R8000 operation
1	Reserved*
2	Interrupt
3	Empty Queue
4	Reserved*
5	Read of Even Tag RAM
6	Read of Odd Tag RAM
7	Combined Read of both Tag RAM's
8-9	Reserved*
10	Invalidate Data Cache Line (32 bytes)
11-15	Reserved*

Table 11-1 TBus Function Field Encoding

11.3 SYSTEM BUS OPERATIONS

This section outlines three common operations which can be initiated by a system bus.

1) Inbound Invalidate
2) Shared Intervention
3) Exclusive Intervention

11.3.1 Inbound Invalidate

An inbound invalidate is required when another processor in a system wants an exclusive copy of some data in the cache. The CC is informed by the other processor of its desire to obtain an exclusive copy of the data. The cache controller then requests the TBus and waits to receive it. Once the TBus is granted the following operations must be performed by the CC.

1) The state of the addressed sector in the streaming cache must be set to invalid by performing a Tag RAM state write.
2) One sector of the Data Cache must be invalidated by performing a data cache invalidate cycle.

If multiple operations are required the CC is not required to relinquish control of the TBus back to the R8000 Microprocessor.

Figure 11-2 shows a timing example of an invalidate. The CC asserts CCREQ_ to begin bus arbitration. The first break line shown in Figure 11-2 indicates that, once CCREQ_ is asserted, as many as 1024 clocks can elapse before the R8000 CPU gives up the bus. The R8000 CPU relinquishes control of the TBus by asserting IUREL_.

The tag RAM state write is performed in the first CC state. The new state information is placed on TRA[39:5]. This new sector information is for the tag RAM and does not affect the R8000 CPU, hence the function field in the first CC state is 0h, indicating no R8000 CPU operation. STWE_ is asserted by the CC two clocks later when the state change to the tag RAM is actually made. The two clock delay is the time required for the information to propagate through the R8000 CPU. The set address information on RWSA[1:0] is also active at this time and is required in order for the tag RAM to determine which of the four sets to update.

In the next 4 CC states the function field changes to Ah (1010b), indicating a data cache invalidation cycle. At the same time the addresses to be invalidated (Adr0-Adr3) are placed on the TRA bus. Each address corresponds to 32 bytes of data. Since invalidation is by sector and a sector is 128 bytes, four addresses must be output by the CC.

The write data bus is not used during an inbound invalidate. Transitions labeled "processor" indicate that the data bus belongs to the R8000 CPU but that the data is not relevant to the invalidate cycle. The 'x' indicates that the data bus does not belong to the R8000 CPU.

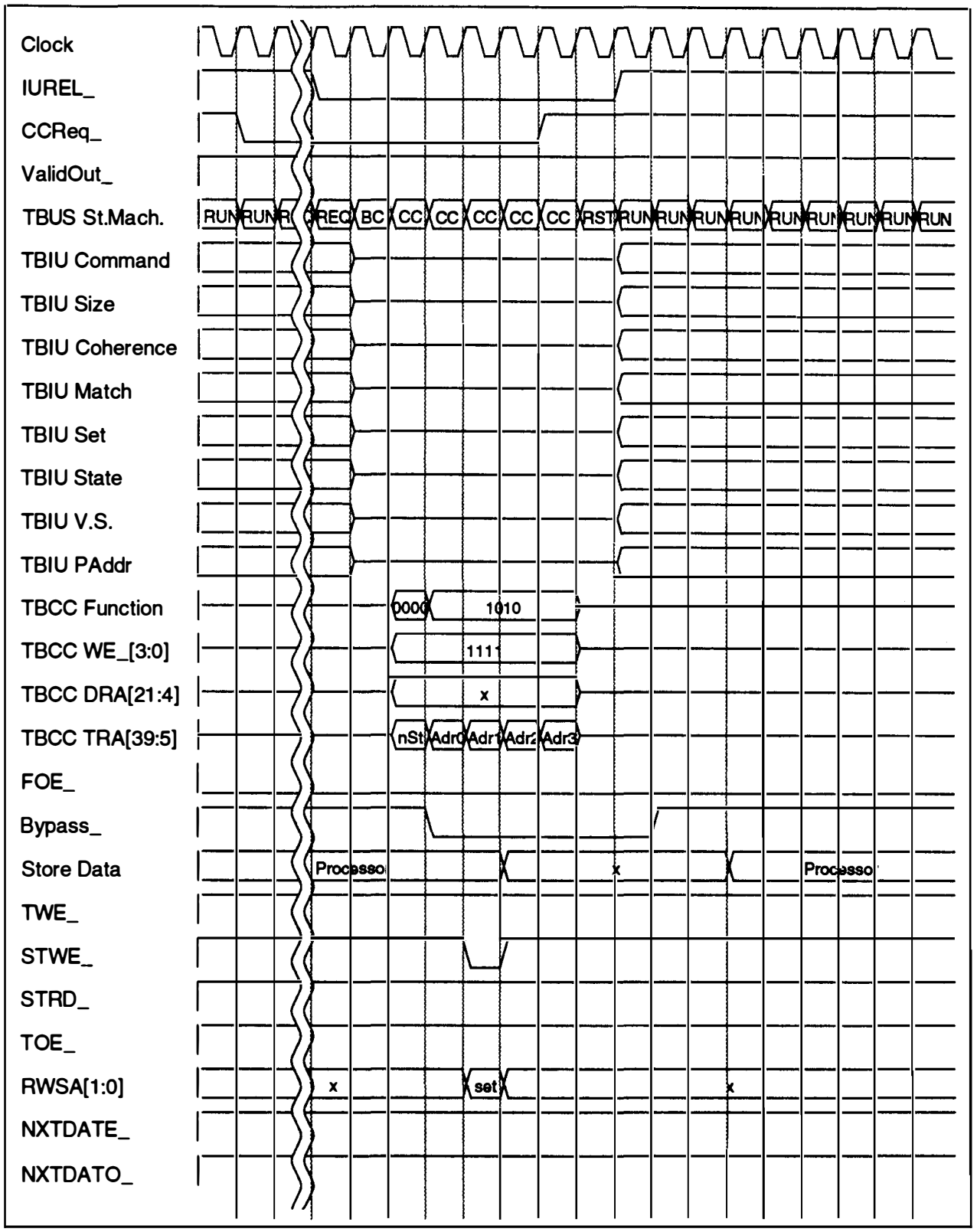


Figure 11-2 Inbound Invalidate

11.3.2 Shared Intervention

A shared intervention occurs when the local processor has an exclusive copy of some data and another processor wants to read it. Once the data is read the status of the line must be changed to shared as the data now exists in more than one location. The CC is informed by the other processor of its desire to obtain a copy of the exclusive data. The CC requests the TBus and once it is granted the following operations must be performed. The sequence of these operations and the duty each one performs constitutes a shared intervention cycle. The cycles below should be pipelined together to maximize system performance. For example, all of these cycles can be performed while the streaming cache is being accessed by the CC.

1) One sector of data must be read from the streaming cache by performing a streaming cache read cycle.

2) The store address queue must be checked to determine whether the requested address resides in the queue by performing a store address queue compare. The operation compare address bits <17:7> on the bus with each entry in the even and odd store address queues. If any of the compares are valid ownership of the TBus must be given back to the R8000 CPU temporarily to allow the cycle to complete.

3) The dirty bit for the requested sector must be checked by performing a combined Tag RAM read. This operation allows the dirty bits from both Tag RAM's to be returned to the CC in the same cycle. A compare is then performed internal to the CC to determine if any of the sectors for the requested line is dirty.

4) The state of the streaming cache line must be set to shared by performing a tag RAM state write.

5) The data cache must be invalidated to remove a possible exclusive tag. This is accomplished by performing a data cache invalidate cycle.

If the dirty bit was found to be clear, or the line is in a state other than exclusive, then the data read may be ignored. The streaming cache must be set to the shared state if the state was clean exclusive and unchanged otherwise.

If the requested address matches one of those in the Store Address Queue, the data read should be ignored and the streaming cache state should remain unchanged. The CC must then return ownership of the TBus to the R8000 CPU so that the store address queue may be emptied. These two cases cannot both happen as the dirty bit must be set in order for the corresponding address to get into the SAQ.

Figure 11-3 shows a timing example of a shared intervention. The CC requests and is eventually granted the TBus when the R8000 CPU asserts IUREL_. In the first "CC" state the address for the read is generated on TRA[39:5]. The data is available 7 clocks later as

shown on the write data bus. Each of the 8 addresses transferred on the TRA bus equates to 128 bits of data, hence an intervention cycle transfers 16 bytes of data.

During the 'CC' states in Figure 11-3 the function field indicates the operation currently being performed. First the data in the data cache must be invalidated. The first address to be invalidated is referred to as A0 on the DRA[21:4]. In the second 'CC' state the invalidation occurs as shown by the value 1010 ('a' hex) in the function field. The definitions of the function field can be found in Table 11-1. In the third and fourth 'CC' states addresses A2 and A3 are transferred and a combined read of the tag RAM's is performed. The result of the combined read is pipelined and the result will not be available for a few clocks.

In the fifth, sixth, and seventh 'CC' states the remaining three addresses to be invalidated; A4, A5, and A6, are transferred. In these three clocks the function field is 1010, indicating an invalidation cycle. In the eighth 'CC' state address data RAM address A7 is transferred. Also in this clock the new state information, indicated on the TRA bus as 'nSt', is written. Note that this value is supplied before the old state information, indicated by 'oSt' on the TRA bus. The old state information was read from the Tag RAM in CC clock 6 by the assertion of STRD_ and placed on the tag bus one clock later with the assertion of TOE_. Remember that the tag bus connects between the tag RAM and the R8000 CPU. The information must then propagate through the R8000 CPU and out onto the TBus. Even though the new state appears on the TBus before the old state, the new state information does not actually arrive at the Tag RAM until the old state is read out, thereby not allowing one to over write the other. The new state is written when STWE_ is asserted.

It is important to note in all of these timing diagrams that the signal activity shown on the TBus is when the information actually appears on the TBus, not necessarily when this information arrives at its final destination (Cache RAM, Tag RAM, etc.). This is why in Figure 11-3 that the new state information can be sent across the TBus three clocks ahead of the old information being sent and still not actually overwrite the old state.

During the time when the new state information is being transferred across the TBus the function field is 0000, indicating no operation to the R8000 CPU. This is because tag RAM updates are handled by the CC. Asserting 0000 onto the function field tells the R8000 CPU to ignore the information on the TBus.

In the next clock function 0011 is placed on the function field, indicating a store address queue hit. However, the R8000 CPU will only respond to this function if it occurs on the last 'CC' state, after which the bus transitions to the BE state. Refer to Figure 11-1 for a flow chart of the TBus state machine.

However, note that function 0011 does not occur on the last CC state. This indicates to the R8000 CPU that a SAQ hit did not occur and to ignore the function. Figure 12-4 shows an intervention with a SAQ hit.

Note in chapter 1, figure 1-5 that the external data buffers interface only to the store data bus of the R8000 microprocessor chip set. When a read is to be performed the information must be read from the streaming cache by the CC and placed on the load data bus. The data is then sent to the R8010 FPU and routed internally onto the store data bus where it is sent back to the data buffers. Hence both loads and stores to main memory use the store data bus. In the second 'CC' state BYPASS_ is asserted which causes the R8010 FPU to route the incoming load data onto the store bus.

The seven clock delay between when the first streaming cache address appears on the TBus and when the corresponding data appears on the store data bus is derived as follows;

a) Two cycles for the address on the TBus to propagate through the R8000 CPU onto the streaming cache address bus.
b) One cycle to pass through an address fan-out register in order to drive all of the necessary RAM's.
c) One cycle to address the RAM.
d) One cycle to retrieve the data.
e) Two cycles to pass the data through the R8000 CPU.

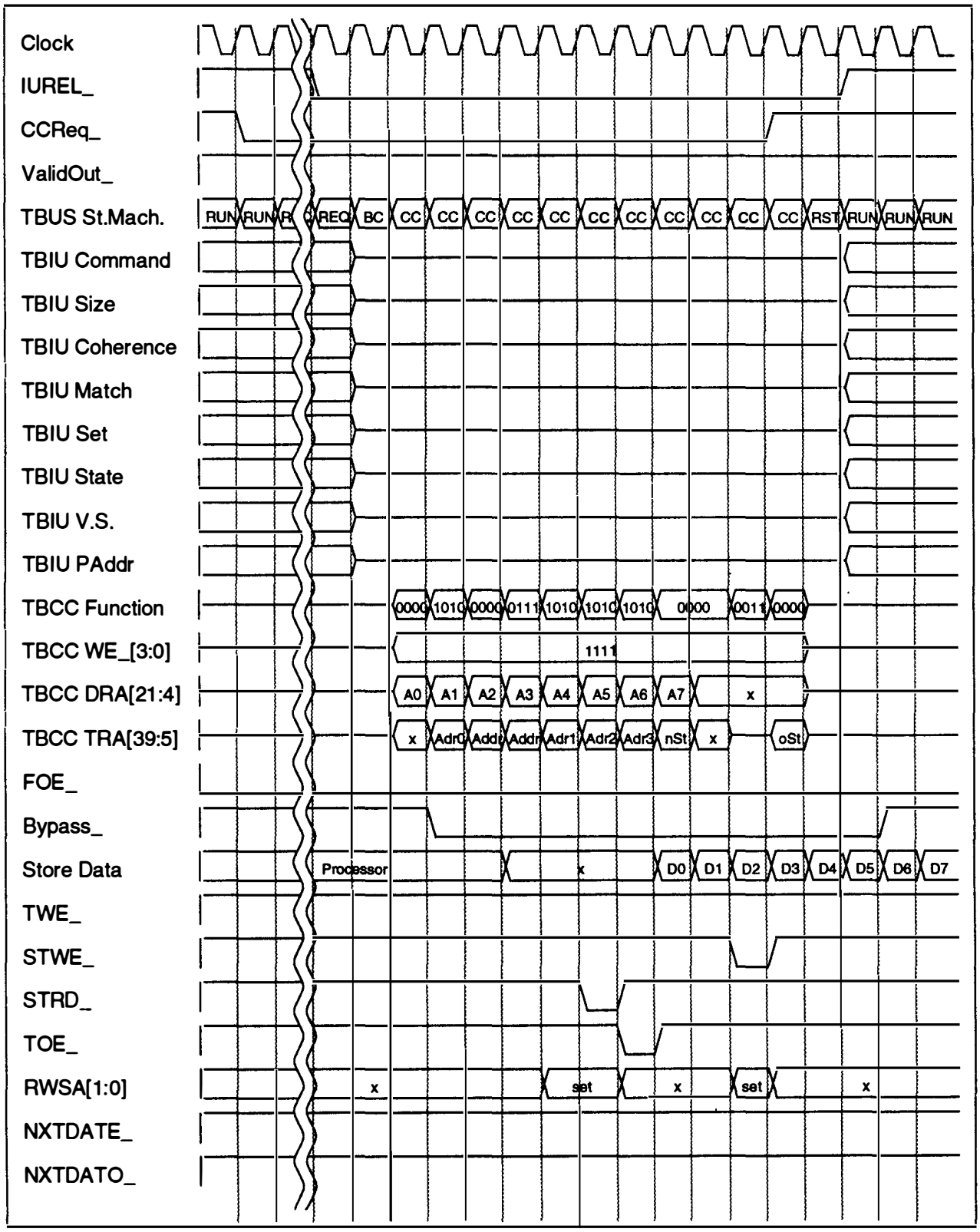


Figure 11-3 : Shared Intervention (read)

11.3.3 Shared Intervention with Store Address Queue Match

The shared intervention with SAQ match operation is identical to the shared intervention operation discussed in section 11.3.2 except that the empty queue function (0011 on the function field of the TBus) occurs on the last 'CC' state, causing the TBus state machine to perform the empty queue operation.

Figure 11-4 shows a timing example of a shared intervention with a SAQ match. A store address queue check is performed and the result appears on NXTDATE_, indicating that the even SAQ got a compare hit. Either NXTDATE_ or NXTDATO_ or both can be asserted in a given clock. The assertion of NXTDATE_ causes the CC to de-assert CCREQ_ which in turn causes the TBus state machine to transition from CC to BE and then to EQ state where control of the TBus is returned to the R8000 CPU and the empty queue operation begins. The break at the end of Figure 11-4 indicates that an undetermined number of cycles can elapse during the EQ operation depending on the types of instructions in the queue to be executed. Note that the data on the store data bus is aborted once the SAQ hit occurs.

Once the EQ operation is completed in Figure 11-5 control of the TBus is returned to the CC and the shared intervention operation is restarted. The operation shown in Figure 11-5 is now exactly the same as that shown in Figure 11-3. Refer to the shared intervention discussion in section 11.3.2.

THIS PAGE INTENTIONALLY LEFT BLANK

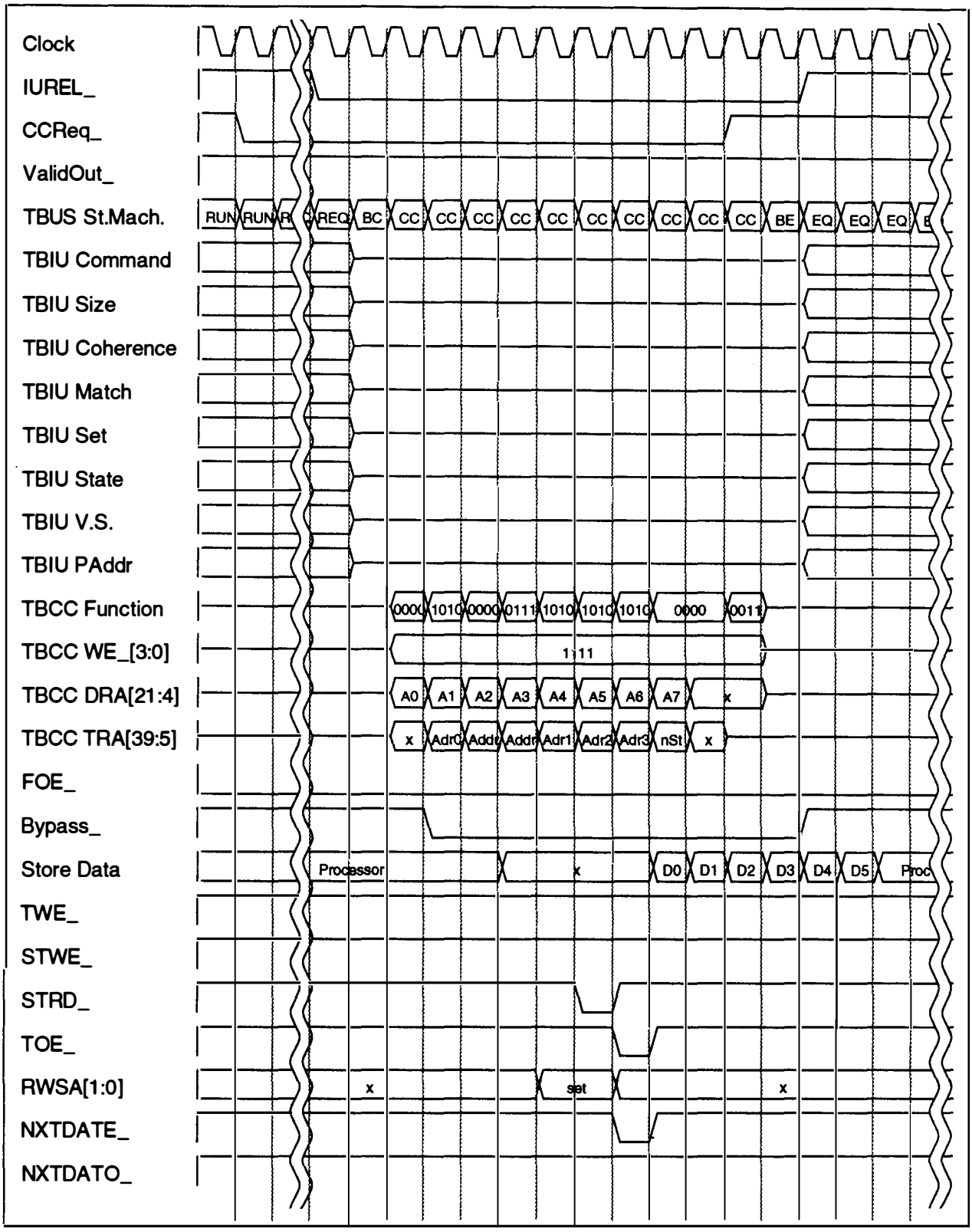


Figure 11-4 Shared Intervention with Store Address Queue Match

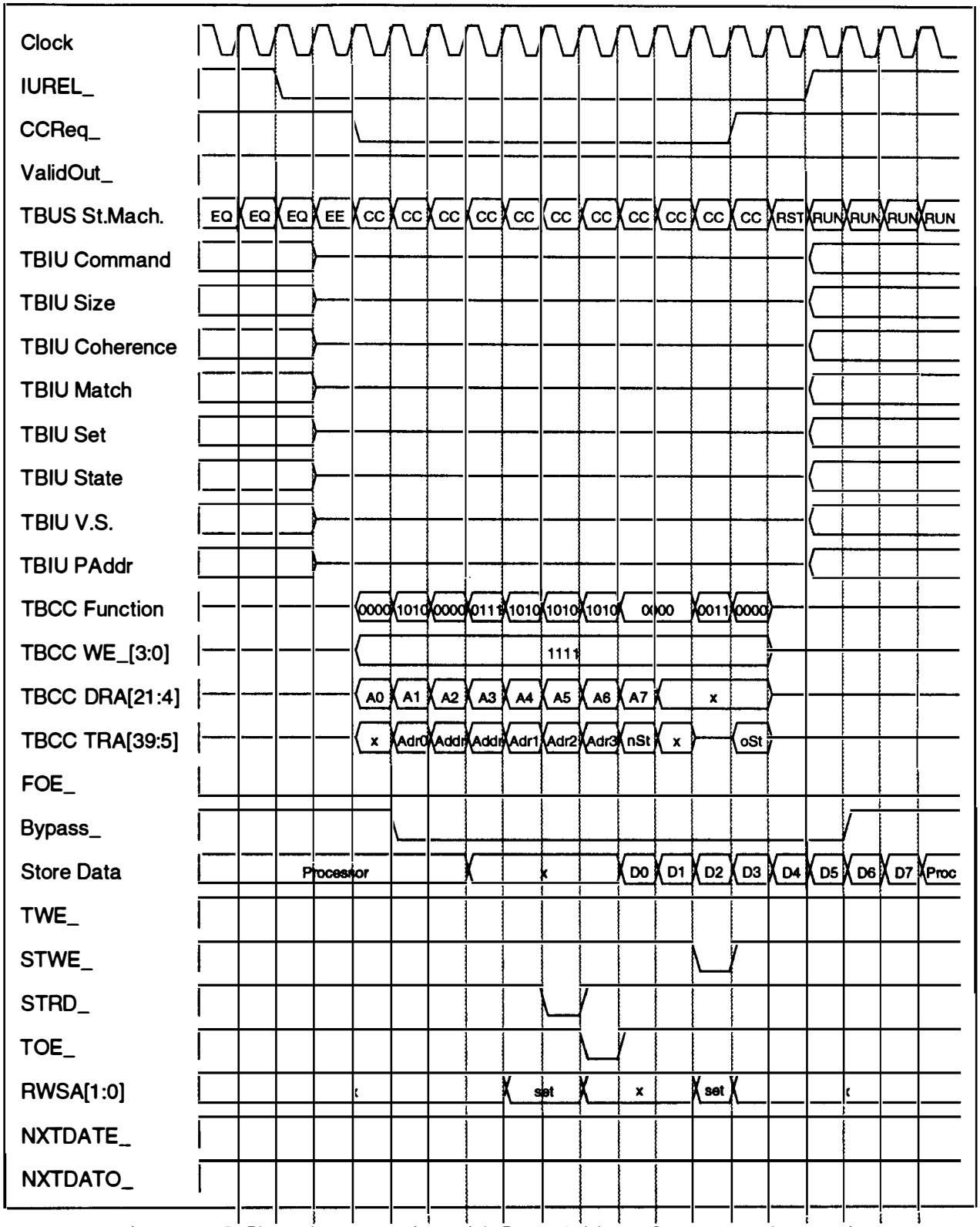


Figure 11-5 Shared Intervention with Store Address Queue Match -- con't.

11.3.4 Exclusive Intervention

An exclusive intervention is similar to a shared intervention except that the other processor has requested an exclusive copy of the data as opposed to simply reading the data.- Once the data is read by the other processor the CC must mark the line as invalid. The CC is informed by the other processor of its desire to obtain an exclusive copy of the data. The CC then requests the TBus and once it is granted the following operations must be performed. The sequence of these operations and the duty each one performs constitutes an exclusive intervention cycle.

1) One sector of data must be read from the data cache by performing a streaming cache data read.

2) The store address queue must be checked to determine whether the requested address resides in the queue. The operation compare address bits <17:7> on the bus with each entry in the even and odd store address queues. If any of the compares are valid ownership of the TBus must be given back to the R8000 CPU temporarily to allow the cycle to complete.

3) The dirty bit for the requested sector must be checked by performing a combined Tag RAM read. This operation allows the dirty bits from both Tag RAM's to be returned to the CC in the same cycle. A compare is then performed internal to the CC to determine if any of the sectors for the requested line is dirty.

4) The state of the streaming cache line must be set to invalid by performing a tag RAM state write.

5) The data cache must be invalidated to remove a possible exclusive tag. This is accomplished by performing a data cache invalidate cycle.

If the dirty bit was found to be clear, or the line is in a state other than exclusive, then the data read may be ignored. The streaming cache must be set to the invalid state if the state was clean exclusive and unchanged otherwise.

If the requested address matches one of those in the Store Address Queue, the data read should be ignored and the streaming cache state should remain unchanged. The Cc must then return ownership of the TBus to the R8000 CPU so that the store address queue may be emptied. These two cases cannot both happen as the dirty bit must be set in order for the corresponding address to get into the SAQ.

The diagram for an exclusive intervention is identical to the shared intervention in Figure 11-3. The only exception is that in a shared intervention the new state information in 'nSt' causes the line to be changed to the shared state. In an exclusive intervention cycle the 'nSt' causes the line to be changed to the invalid state, thereby allowing the requesting agent to obtain the data exclusively.

The exclusive intervention with store address queue match is identical to the shared intervention with store address queue match in Figure 11-4 and Figure 11-5. The only exception is that in a shared intervention with SAQ hit the new state information in 'nSt' causes the line to be changed to the shared state. In an exclusive intervention with SAQ hit the 'nSt' causes the line to be changed to the invalid state, thereby allowing the requesting agent to obtain the data exclusively.

11.4 PROCESSOR INITIATED OPERATIONS

Under certain conditions the R8000 CPU can initiate operations requiring interface to the CC via the TBus. The processor asserts VALIDOUT_ and waits for servicing by the CC. Multiple cycles can occur back to back, hence there is no maximum limit of cycles placed on the CC by the R8000 CPU. The following are some conditions under which the R8000 CPU asserts VALIDOUT_ and IUREL_ simultaneously.

1) A write back cycle needs to be executed.
2) The result of a Tag RAM lookup is a virtual synonym mis-match or no address match.
3) The store address queue is empty.

Under conditions where the R8000 CPU has requested data from memory there is latency involved in the access. During this time the R8000 CPU does not need to relinquish control of the bus immediately. As long as the SAQ is not empty the R8000 CPU can continue execution. VALIDOUT_ is asserted by the R8000 CPU and the requested information is passed to the CC. IUREL_ is asserted by the R8000 CPU when:

1) There are no more addresses on the SAQ and the processor is now idle.
2) The CC requests the TBus. The R8000 CPU then asserts IUREL_ as soon as the current TBus transfer(s) are completed.

Table 11-2 shows a listing of cycles which the R8000 CPU may need to perform. In the State column, "ALL" refers to all combinations of the No Match and State fields of the TBus. No Match means that the MATCH_ pin was not asserted when the Tag RAM was accessed for the corresponding cycle. MATCH_ asserted indicates the line was in either the Invalid, Shared, or Exclusive state. The cycles in the "Operation Required" column are discussed in the following sections.

Command	Coherence Protocol	State	Operation Required
Read/Instruction Fetch	Non-Cachable	All	Non-Cachable Read
Read/Instruction Fetch	Cachable	No-Match	Miss and Replace
Read/Instruction Fetch	Cachable	Invalid	Simple Miss
Write	Cachable	No-Match	Miss and Replace
Write	Cachable	Invalid	Simple Miss
Write	Invalidate	Shared	Upgrade
Write	Non-Cachable	All	Non-Cachable Write
Write	Non-Cachable	All	Sequential Non-Cachable Write

Table 11-2 Processor Initiated TBus Operations

From Table 11-2 above, the following operations can be required of the R8000 CPU:

1) Miss and Replace
2) Simple Miss
3) Upgrade
4) Non-Cachable Read
5) Non-Cachable Write
6) Sequential Non-Cachable Write

11.4.1 Miss and Replace

Miss and Replace means that a sector was required for which there was not already a place in the streaming cache. Hence one of the four lines at the required cache index must be replaced. When the R8000 CPU requests this operation, IUREL_ is asserted by the R8000 CPU, causing the TBus state machine to transition from RUN state to CC state, thereby transferring ownership of the TBus to the CC. Once the CC is granted ownership, the following operations are required to perform a streaming cache miss and replace. The sections referred to in each step show the corresponding timing diagrams.

1) One line in the cache index must be chosen for replacement. Which set of the cache is chosen for replacement depends on the state of the Match Field. If the Match field indicates an either an address match and virtual synonym match, or an address match and a virtual synonym mis-match, the line is chosen based on the state of the Set Address field.

If the Match field indicates no address match the set is normally chosen using the random replacement algorithm. However, there are two exceptions where the set is not chosen randomly:

1a) A cachable read/I-fetch (second entry in Table 11-2) which has no address match. The corresponding Miss and Replace operation chooses a set based on the state of the set address field.
1b) A cachable write (fourth entry in Table 11-2) which has no address match. The corresponding Miss and Replace operation chooses a set based on the state of the set address field.

In the above two cases the intent of the Miss and Replace operation is to obtain a new virtual synonym and a perform a data cache invalidate.

2) The Store Address Queue (SAQ) must be checked for conflict with any of the four sectors of that line by performing a SAQ compare operation. The operation compares address bits <17:7> on the bus with bits <17:7> for each entry in the even and odd store address queues. If any of the compares are valid ownership of the TBus must be given back to the R8000 CPU temporarily to allow the cycle to complete.

3) The upper physical address bits must be retrieved from the Tag RAM for use in writing back the dirty sectors and in invalidating the data cache. This can be accomplished by performing a Tag RAM address read.

4) The dirty bits must be checked for each of the sectors of that line. In addition, the virtual synonym bits for the line which is to be replaced must be retrieved from the Tag RAM. Both operations are accomplished by performing a Tag Read Combined operation.This operation allows the dirty bits from both Tag RAM's to be returned to the CC in the same cycle. A compare is then performed internal to the CC to determine if any of the sectors for the requested line is dirty. The state read operation allows the CC to read the state and virtual synonym information for a given entry in the cache.

5) If it is determined in step 3 that any of the sectors is dirty, the dirty data from the replaced line must be read from the streaming cache by performing a streaming cache read.

6) The data cache must be invalidated to remove the potential copy of any of the four sectors of the replaced line. This is accomplished by performing a data cache invalidate cycle. The number of Data Cache invalidates which must be executed depends on the system parameters. For example, data cache invalidates can invalidate 32 bytes. If the streaming cache line size is 512 bytes then 16 back to back data cache invalidates must be executed in order to invalidate the entire line.

7) A new sector of data must be read from the system bus and placed in one sector of the chosen line by performing a streaming cache write cycle.The criteria for determining which set should be replaced is explained in step 1.

8) The tag address of the new data must be placed in the Tag RAM.

9) The state of the streaming cache must be set either to Shared or Exclusive, whichever state is appropriate, for the new sector, and to Invalid for the other three sectors of the line. The dirty bit must be cleared for all four sectors of that line. This is accomplished by performing a Tag RAM state write.

If the SAQ compare needs to be emptied control of the TBus is returned to the R8000 CPU and the queue allowed to empty. The entire process is then repeated from the beginning. The states and dirty bits control which sectors are written back to the system. If the states are not exclusive and the dirty bits are clear for blocks which have not been read in parallel with checking the states and dirty bits, then those sectors do not need to be read.

There are several minor variations for the reading of the new sector from the system. The type of bus transaction used and the state to which the streaming cache sector is set when another cache makes the shared response is shown in Table 11-3.

Cause of Miss	Coherence Protocol	Bus Transaction	Not Shared	Shared
Any	Non-Coherent	Non-Coherent	Exclusive	Exclusive
Any	Exclusive	Exclusive Read	Exclusive	------
Load	Shared	Read	Exclusive	Shared
I-Fetch	Shared	Read	Shared	Shared
Store	Shared	Exclusive Read	Exclusive	-------

Table 11-3 Bus Transactions and their Resulting States

While waiting for the new data to be retrieved, those cycles not required for these operations to complete may be transferred to the floating point and store machines by requesting that the queue be emptied and then requesting the TBus in anticipation of the earliest time data might return. The following timing diagrams show some examples of how various miss and replace cycles could be performed.

Figure 11-6 shows a line replacement with address miss. The line replacement operation replaces data for 32 consecutive addresses. With 16 bytes of data corresponding to each address, a total of 512 Bytes of data are moved. The set chosen for the transfer is arbitrary. The first four valid addresses on TRA[39:5] are for store address queue checking and reading out the state information from the tag RAM. These operations correspond to functions 0110 and 0111 of the function field.

After the tag RAM information is read out the data cache is invalidated as shown by Inv0-Inv3 on the TRA pins. The actual invalidation addresses continue up to Inv15. As with previous timing diagrams, the empty queue function 0111 on the TBus does not occur on the last 'CC' state and hence is ignored by the R8000 CPU.

The addresses and corresponding write-back data complete about half way through Figure 11-7. The new address and data then appear on A0-A7 of the DRA field and R0-R7 of the store data bus. The new tag address information appears on the TRA field followed by the state field. TWE_ is asserted two clocks after the tag address appears on the TBus, again to allow for propagation time through the R8000 CPU, and the tag address information is written. In the next clock STWE_ is asserted to allow the state information to be written. Which of the four sets is to be written is determined by the information on the RWSA[1:0] pins when TWE_ and STWE_ are asserted. The set information shown valid for two clocks during the read in Figure 11-6 is the same set used to write the new information in Figure 11-7. This assures that the same set is read and written.

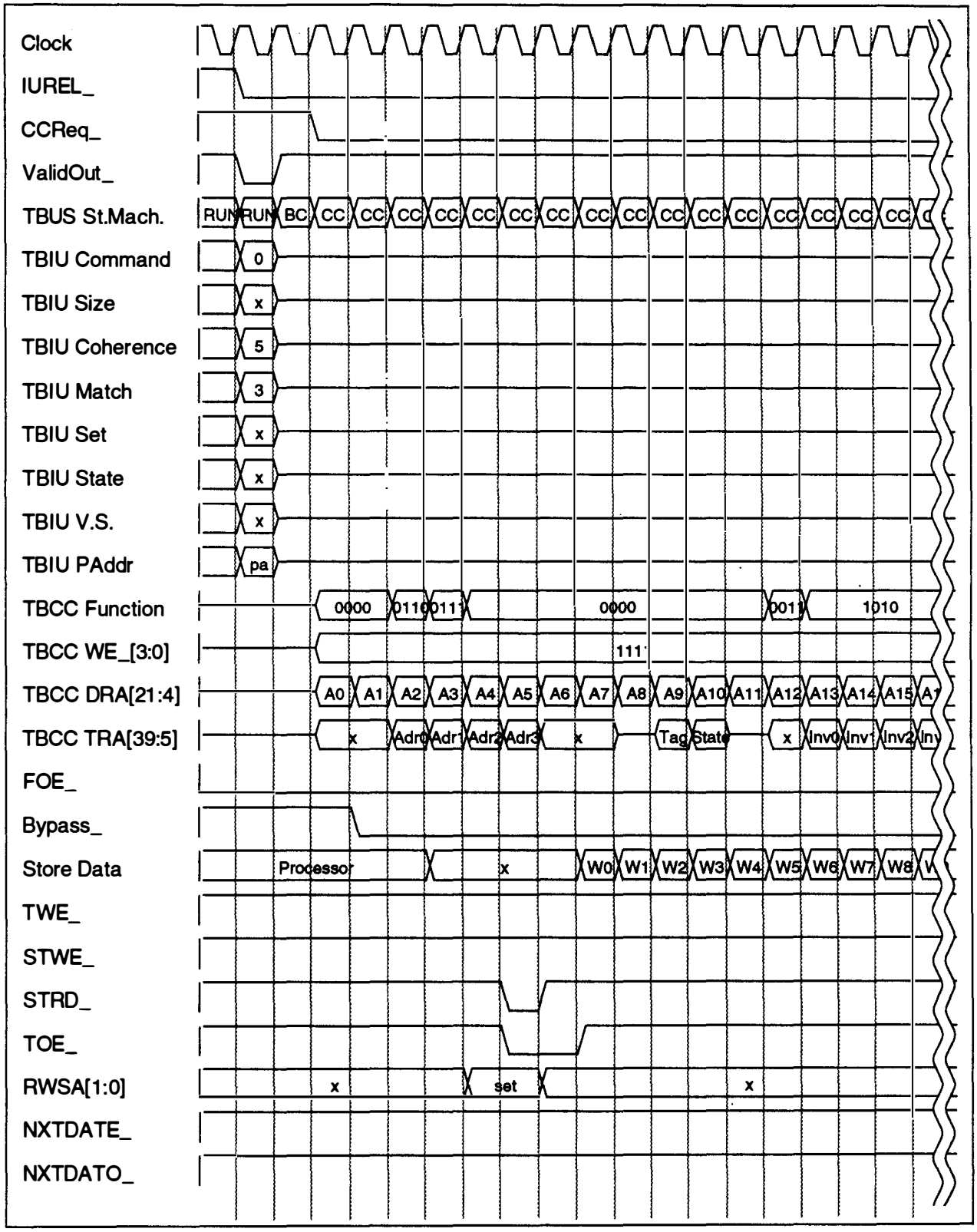


Figure 11-6 Read with Line Replacement -- Address Miss

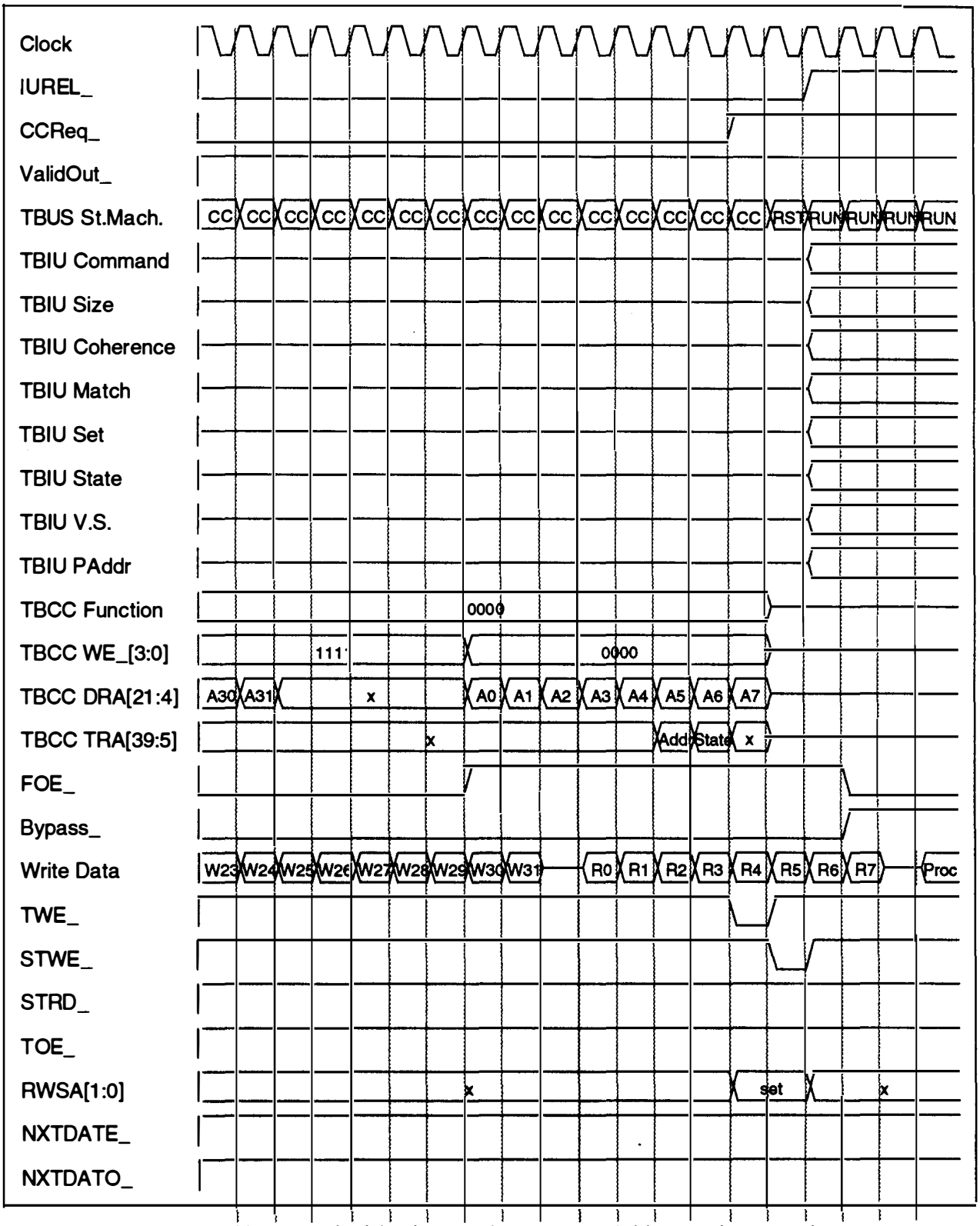


Figure 11-7 Read with Line Replacement -- Address Miss -- con't

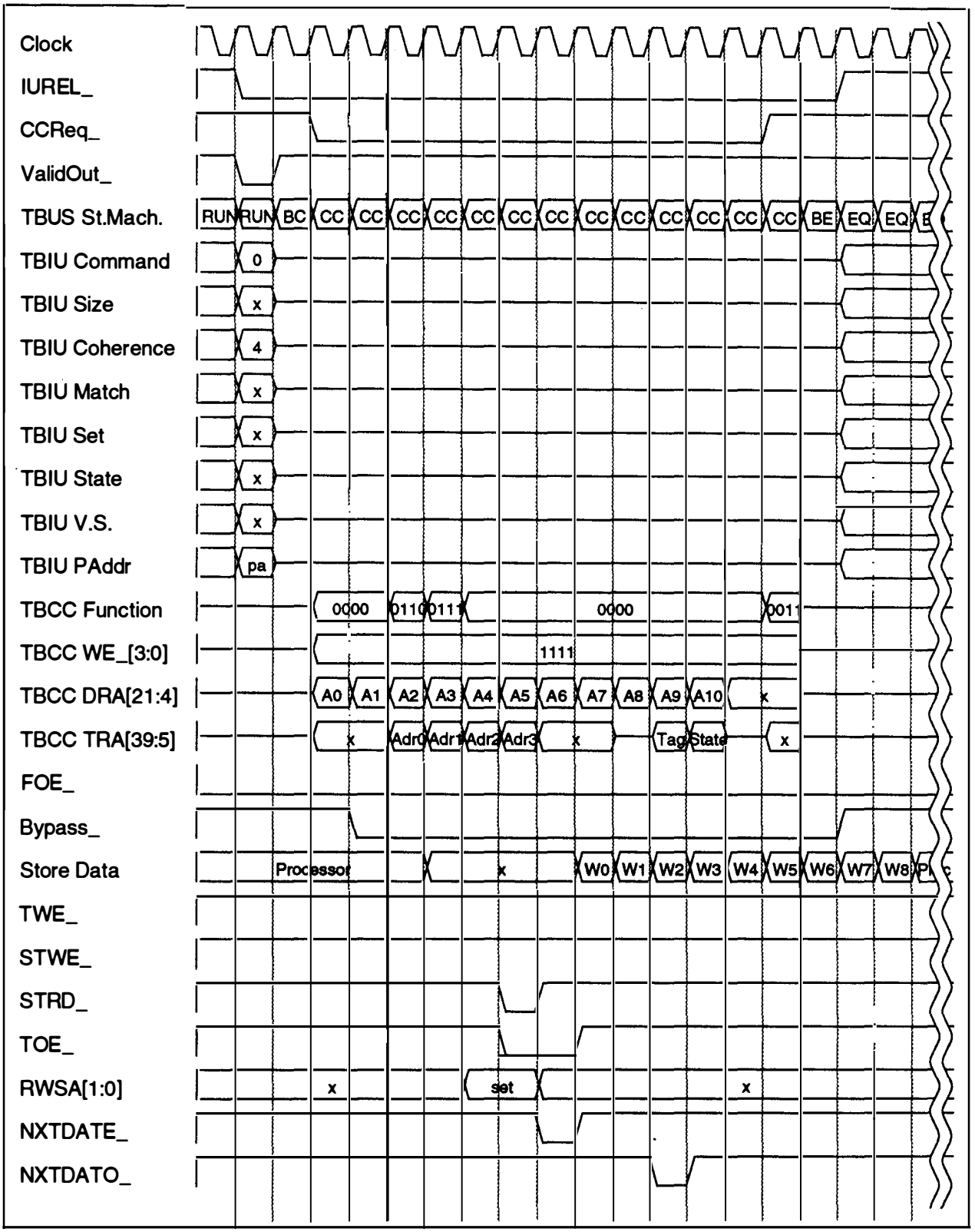


Figure 11-8 Miss and Replace with Store Address Queue Match

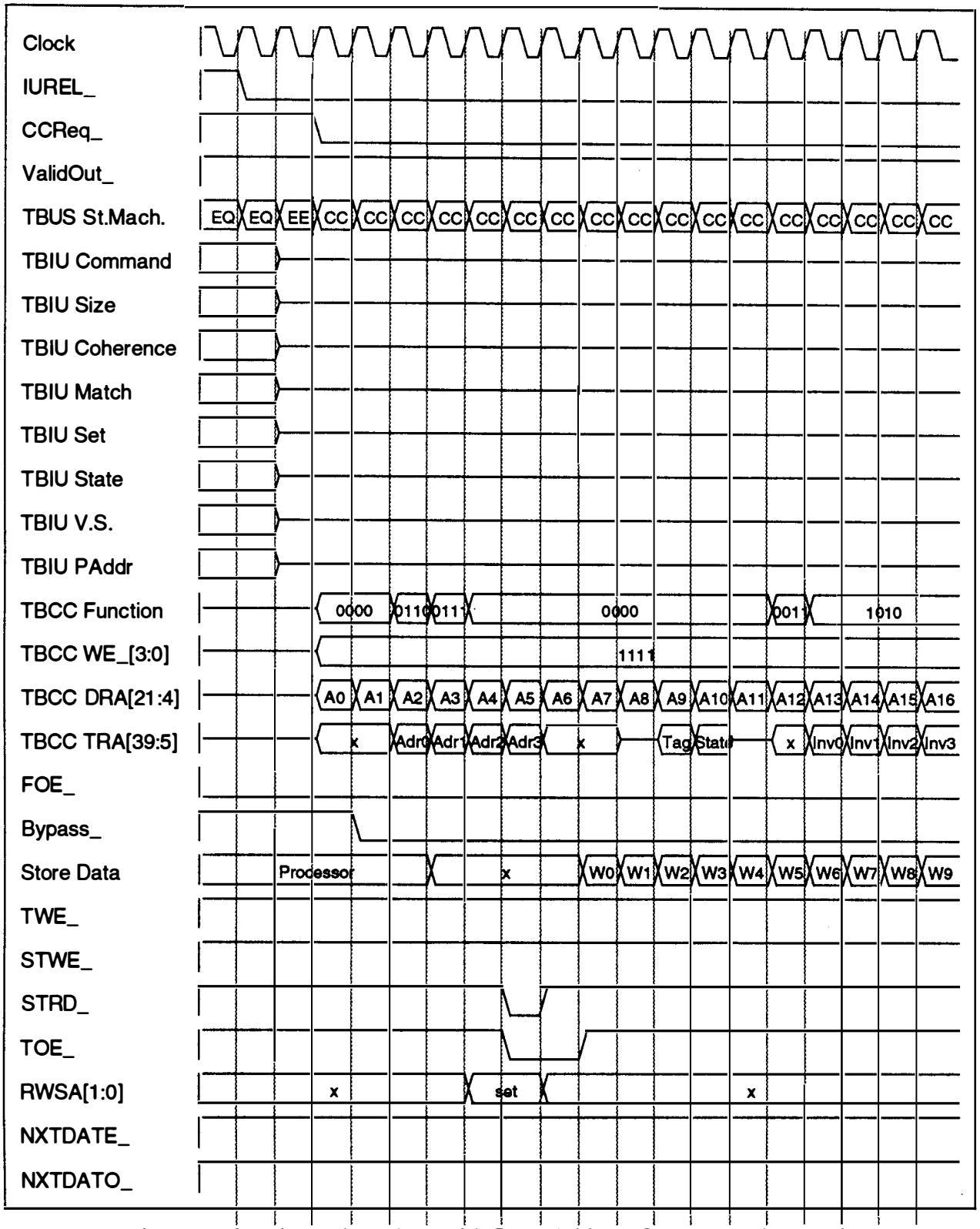


Figure 11-9 Miss and Replace with Store Address Queue Match -- con't

11.4.2 Simple Miss

The simple miss is much more straight forward that the Miss and Replace operation because the sector to be filled is currently invalid and there is no data to write back. Contrary to the Miss and Replace operation, in the Simple Miss operation the R8000 CPU does not assert IUREL_ at the same time as VALIDOUT_, hence control of the TBus is not passed immediately to the CC. The following operations are required in order to complete the Simple Miss operation.

1) A new sector of data must be read from the system and placed in one sector of the chosen line at a set determined by the value on the set address field. This is accomplished by performing a streaming cache write.

2) The state of the streaming cache must be set to shared or exclusive, as appropriate, and the dirty bit cleared for the new sector.This is accomplished by performing a Tag RAM state write.

These two operations are accomplished in the same manner as the corresponding operations for the Miss and Replace except that the streaming cache is only modified for one of the sectors.

Figure 11-10 and Figure 11-11 shows a timing diargam of a simple miss.

THIS PAGE INTENTIONALLY LEFT BLANK

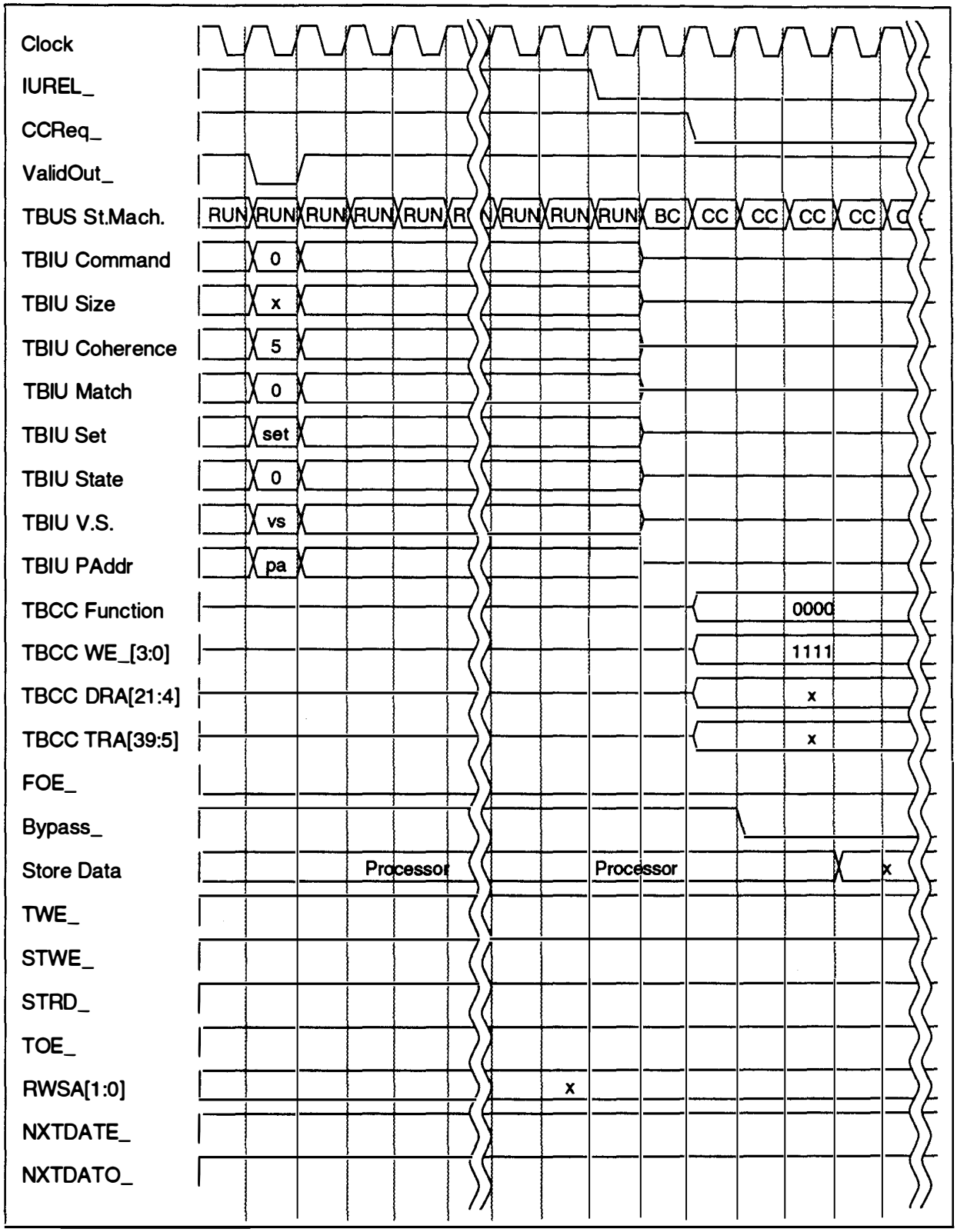


Figure 11-10 Simple Miss with Tag RAM Address Match

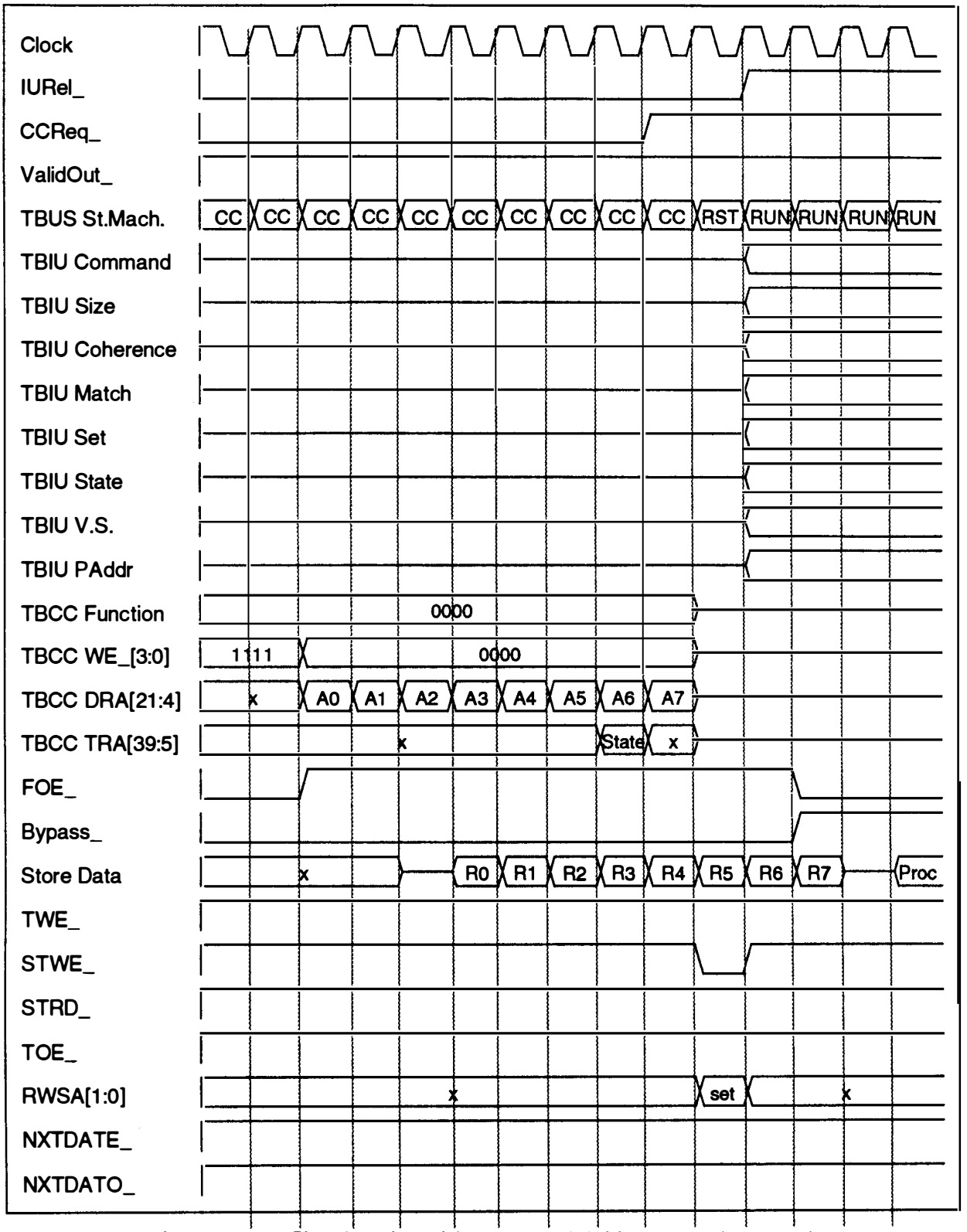


Figure 11-11 Simple Miss with Tag RAM Address Match --- con't

11.4.3 Upgrade

An upgrade occurs when the R8000 CPU wants to write a line which is shared. The sector is in the shared state but must be changed to the exclusive state so that it can be written. For the upgrade operation the R8000 CPU does not assert IUREL_ in the same clock as VALIDOUT_ and control of the TBus is not passed immediately to the CC. If the upgrade fails, meaning that another processor has already performed the upgrade, nothing is done and control is requested and then returned to the R8000 CPU for a retry. If the upgrade succeeds the state of the sector in the streaming cache is updated from shared to exclusive state and the dirty bit is cleared. Control of the TBus must be passed to the CC so that a Tag RAM state write can be executed and the status of the line changed and the corresponding dirty bit cleared. The dirty bit is then set by the R8000 CPU when the write occurs. Refer to section 11.4.5 for more information on how to execute a Tag RAM state write.

The upgrade timing diagram is quite simple since the intention of the cycle is only to change the state of the Tag RAM. Once the TBus state machine is in the 'CC' state the cache controller places the new tag RAM state information on the TRA bus. The information propagates through the R8000 CPU and two clocks later STWE_ is asserted and the write is performed. At the same time as STWE_ is asserted, RWSA[1:0] is valid which indicates to the tag RAM which set is to be written.

Figure 11-12 shows a timing diagram of a line upgrade.

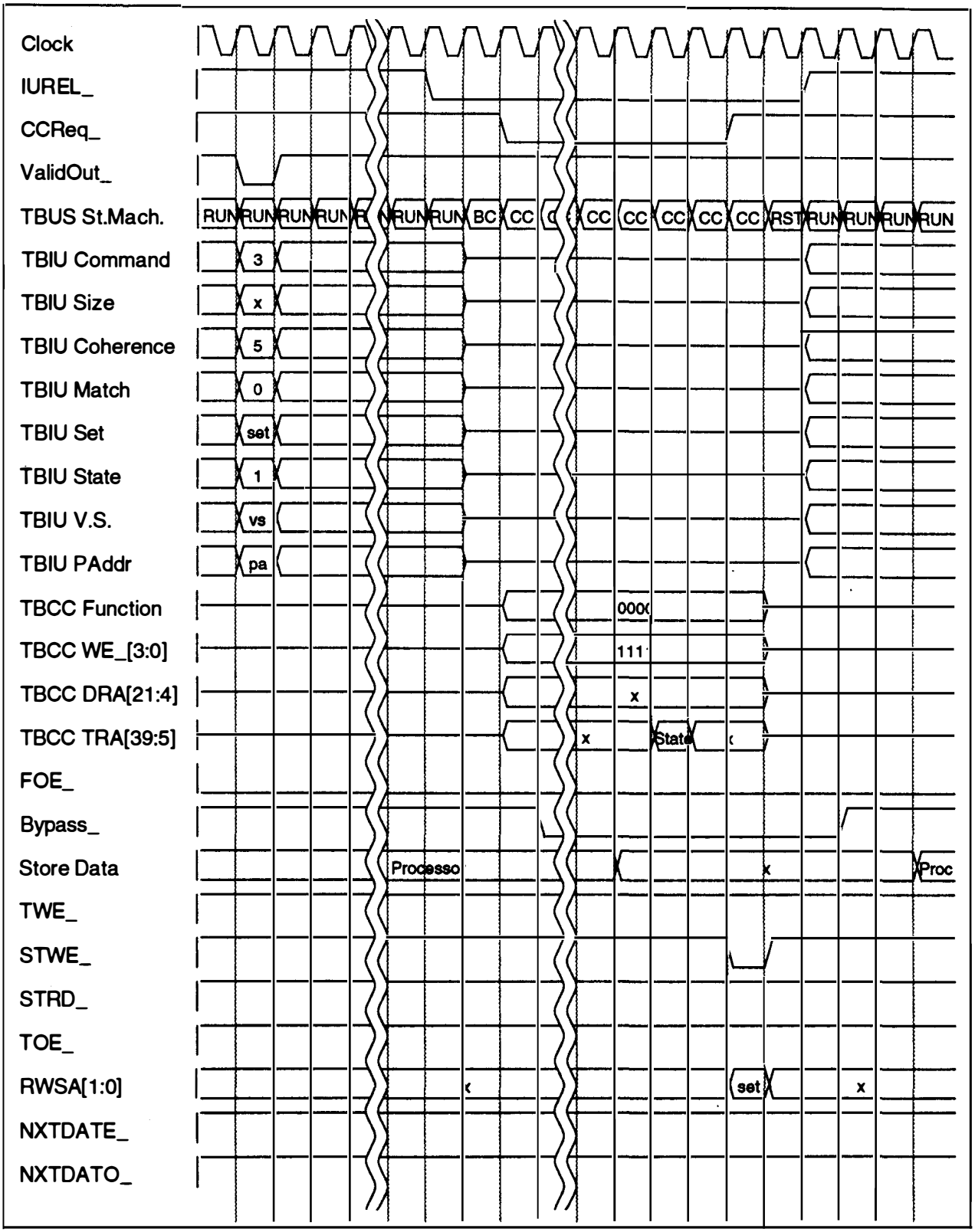


Figure 11-12 Upgrade

11.4.4 Non-Cachable Read

A non-cachable read occurs when any read is executed from a non-cachable protocol page. Unlike a cachable read a non-cachable read can occur only with a line replacement. The steps are as follows:

1) One line in the cache index must be chosen for replacement. Which set of the cache is chosen for replacement depends on the state of the Match Field. If the Match field indicates an either an address match and virtual synonym match, or an address match and a virtual synonym mis-match, the line is chosen based on the state of the Set Address field.

If the Match field indicates no address match the set is normally chosen using the random replacement algorithm. However, there are two exceptions where the set is not chosen randomly:

1a) A cachable read/I-fetch which has no address match. The corresponding Miss and Replace operation chooses a set based on the state of the set address field.
1b) A cachable write which has no address match. The corresponding Miss and Replace operation chooses a set based on the state of the set address field.

In the above two cases the intent of the Miss and Replace operation is to obtain a new virtual synonym and a perform a data cache invalidate.

2) The Store Address Queue (SAQ) must be checked for conflict with any of the four sectors of that line by performing a SAQ compare operation. The operation compares address bits <17:7> on the bus with bits <17:7> for each entry in the even and odd store address queues. If any of the compares are valid ownership of the TBus must be given back to the R8000 CPU temporarily to allow the cycle to complete.

3) The upper physical address bits must be retrieved from the Tag RAM for use in writing back the dirty sectors and in invalidating the data cache. This can be accomplished by performing a Tag RAM address read.

4) The dirty bits must be checked for each of the sectors of that line. In addition, the virtual synonym bits for the line which is to be replaced must be retrieved from the Tag RAM. Both operations are accomplished by performing a Tag Read Combined operation.This operation allows the dirty bits from both Tag RAM's to be returned to the CC in the same cycle. A compare is then performed internal to the CC to determine if any of the sectors for the requested line is dirty. The state read operation allows the CC to read the state and virtual synonym information for a given entry in the cache.

5) If it is determined in step 3 that any of the sectors is dirty, the dirty data from the replaced line must be read from the streaming cache by performing a streaming cache read.

6) The data cache must be invalidated to remove the potential copy of any of the four sectors of the replaced line. This is accomplished by performing a data cache invalidate cycle. The number of Data Cache invalidates which must be executed depends on the system parameters. For example, data cache invalidates can invalidate 32 bytes. If the streaming cache line size is 512 bytes then 16 back to back data cache invalidates must be executed in order to invalidate the entire line.

7) For a data non-cachable read the entire 128 byte sector is fetched. Instruction fetch non-cachable reads fetch only from the on-board PROM or from main memory. In this case 32 bytes are read at a time. A new sector of data must be read from the system bus and placed in one sector of the chosen line by performing a streaming cache write cycle. The criteria for determining which set should be replaced is explained in step 1.

8) The tag address of the new data must be placed in the Tag RAM.

9) The state of the streaming cache must be set either to Invalid. This is accomplished by performing a Tag RAM state write.

The non-cachable read cycle is almost identical to the read with line replacement cycle shown in Figure 11-6 and Figure 11-7. The differences are as follows:

1) When VALIDOUT_ is asserted in for the line replacement in Figure 11-6 the three bit coherence protocol field is a 5 (101), indicating a cachable coherent exclusive on write. In Figure 11-13 the protocol is a 2 (010), indicating an uncachable sequential operation.

2) In the next to last 'CC' cycle in Figure 11-7 the state information remains the same. In the next to last 'CC' cycle in Figure 11-14 the state is changed to invalid.

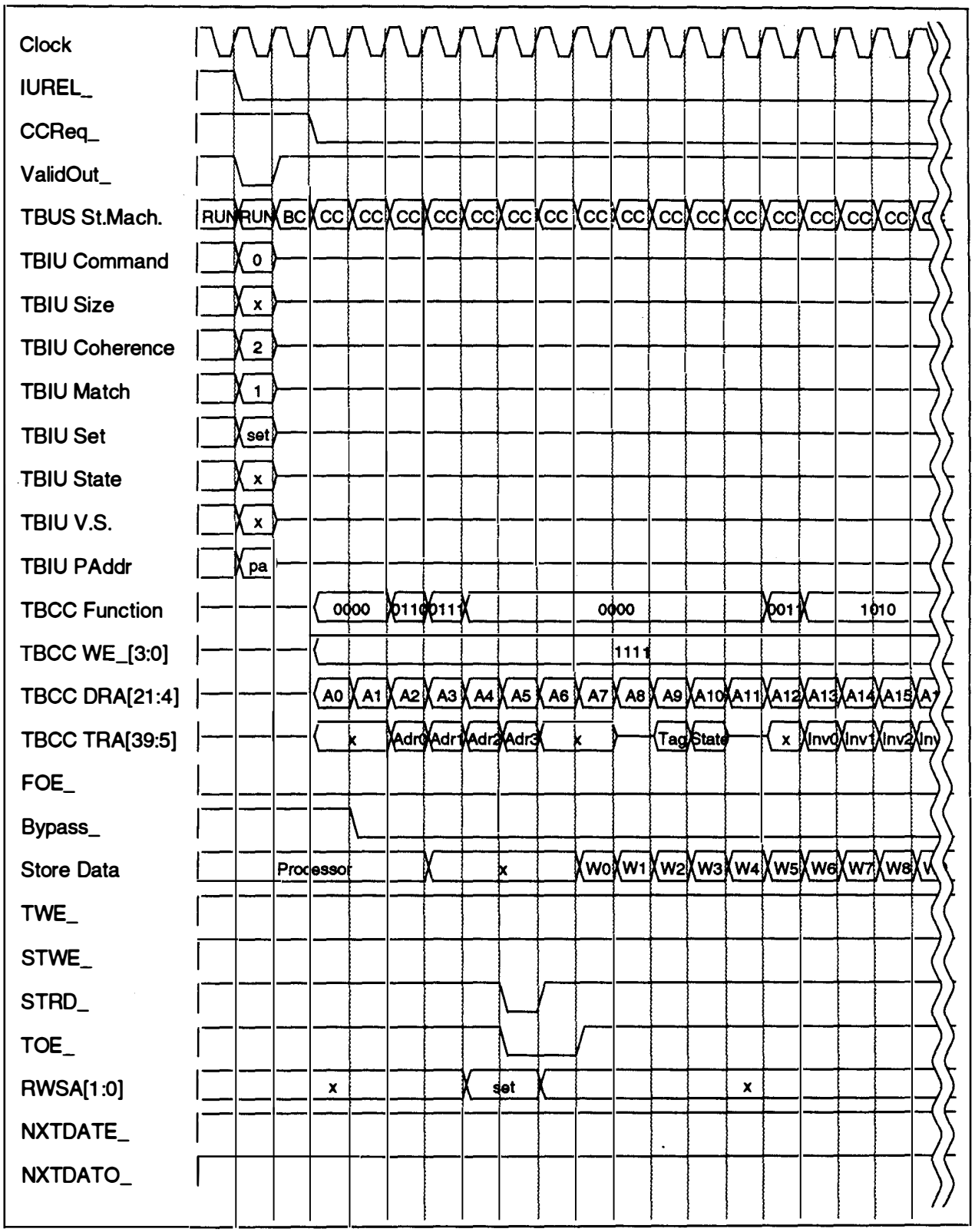


Figure 11-13 Non-Cachable Read

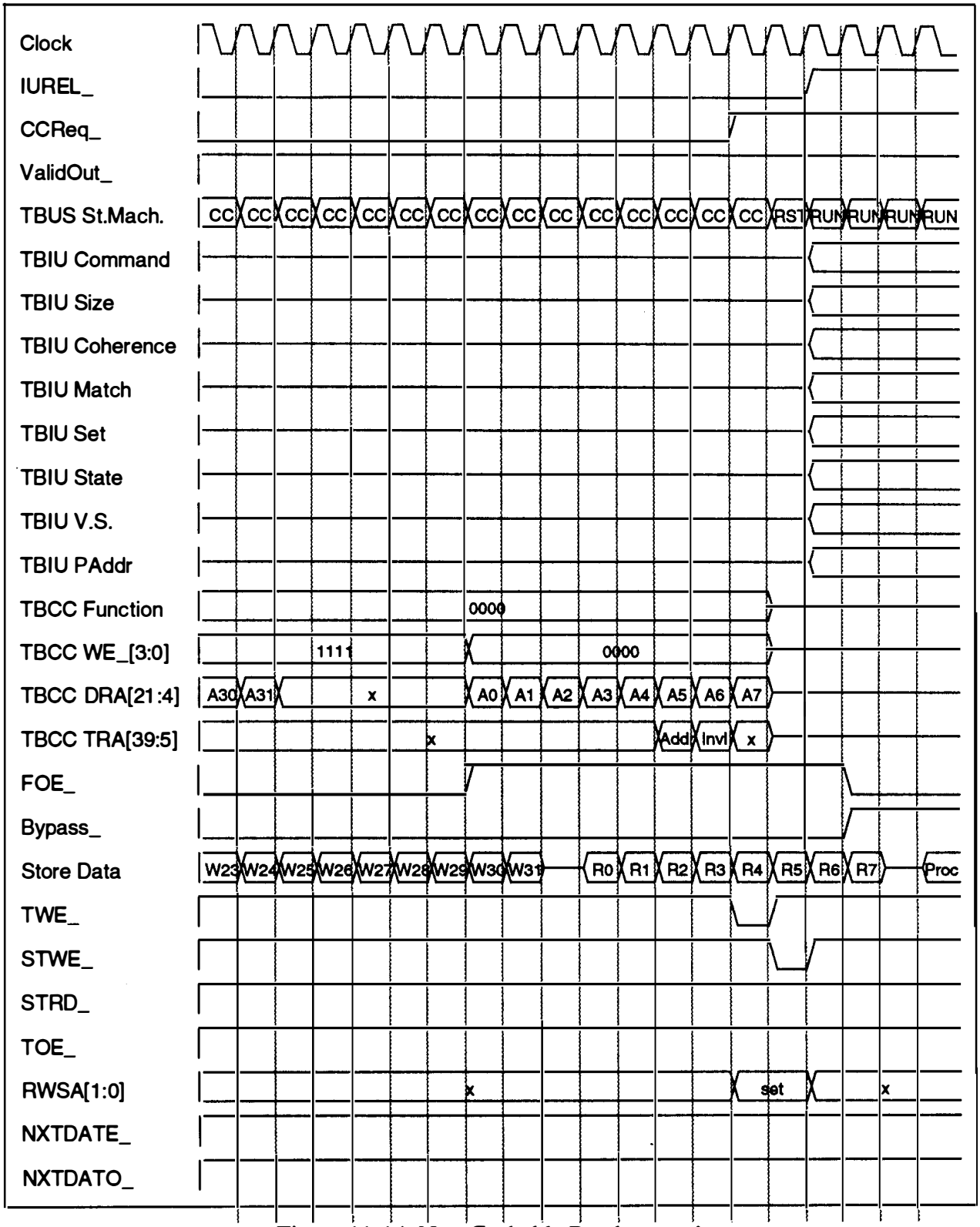


Figure 11-14 Non-Cachable Read --- con't

11.4.5 Processor Ordered Non-Cachable Write

A non-cachable write occurs when the Sequential Mode bit in the R8000 CPU is clear and any write is done to a non-cachable protocol page. For a non-cachable write operation the R8000 CPU does not assert IUREL_ in the same clock as VALIDOUT_ and control of the TBus is not passed to the CC. A non-cachable write must wait for the corresponding data to show in the correct update data queue. The two data are then sent together to the system or to the addressed local register. The Processor Ordered Non Cachable function is used for writing hardware registers in the CC which support the write-gatherer operation. The purpose of a write-gatherer is to gather 32 bit write operations from another source (such as a graphics engine) into a 128 byte block before sending them across the bus. Otherwise each single 32 bit write would have to be sent across the bus which would decrease system data bandwidth and degrade overall system performance.

The processor ordered non-cachable write timing diagram in Figure 11-15 never leaves the RUN state. The NXTDATE_ and NXTDATO_ pins perform different functions depending on whether the R8000 CPU of the CC is in control of the TBus. When the R8000 CPU is in control of the TBus each is driven by the R8000 CPU and indicates that store data associated with a non-cachable write will be on the even store data bus on the next clock. When the CC is in control of the TBus, the R8000 CPU asserts NXTDATE_ or NXTDATO_ if either store address queue contains an address for which bits [17:7] match the Tag RAM index bits [17:7] which were on the TBus four cycles earlier. The signals remain de-asserted if no such match is detected.

In Figure 11-15 when NXTDATE_ is asserted data on the even store data bus becomes valid one clock later. The even data is always in order with the even addresses and the odd data is always in order with the odd addresses. However, the even and odd references are not always in order relative to each other.

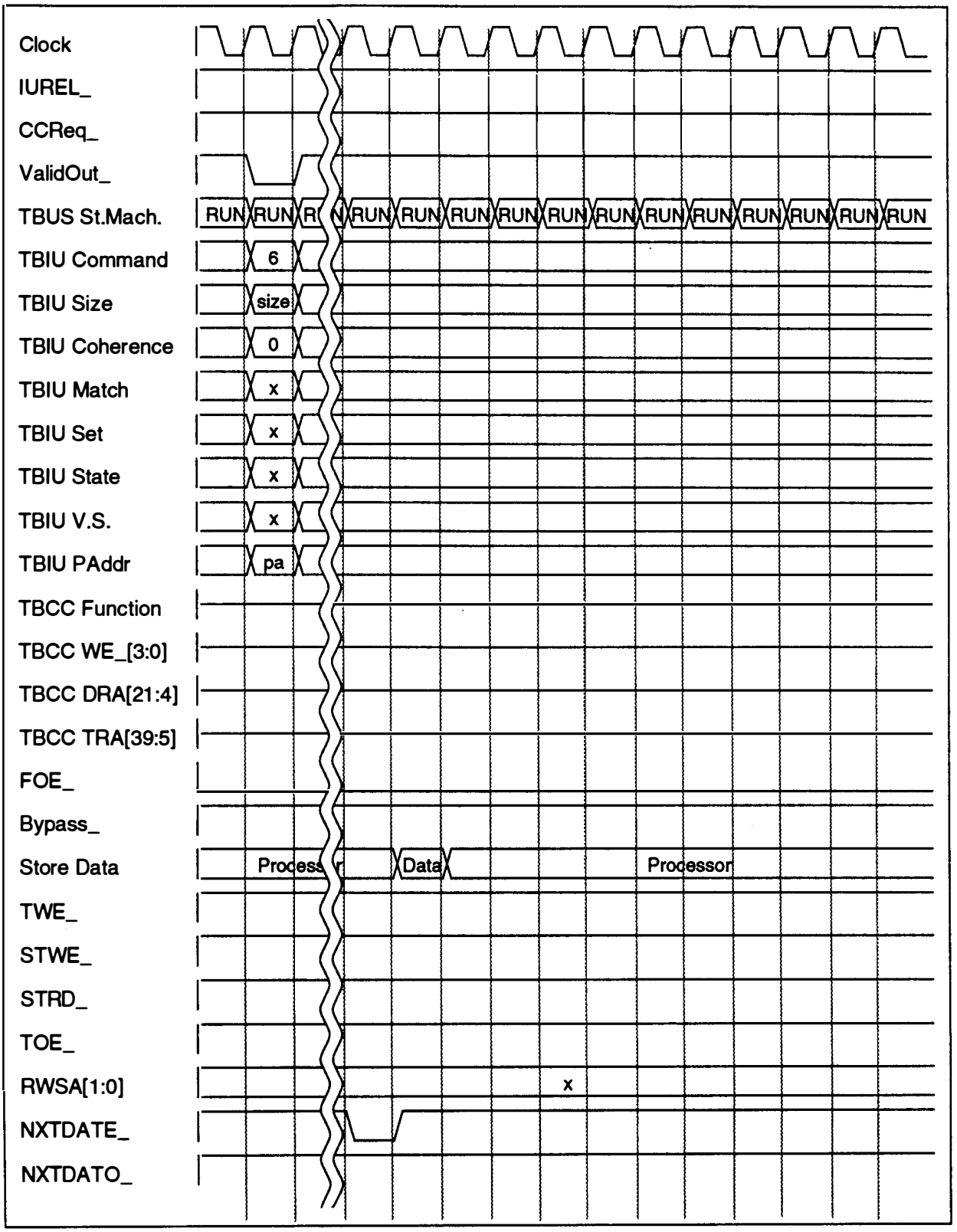


Figure 11-15 Processor Ordered Non-Cachable Write

11.4.6 Sequential Non-Cachable Write

For a sequential non-cachable write operation the R8000 CPU does not assert IUREL_ in the same clock as VALIDOUT_ and control of the TBus is not immediately passed to the CC. Figure 11-16 shows a timing example of a sequential ordered non-cachable write.

The only real difference between the sequential and processor-ordered non-cachable writes is that in a sequential non-cachable write the R8000 CPU asserts IUREL_ which allows the it to determine when the actual non-cachable write has been completed. At the end of the cycle CCREQ_ is asserted by the CC indicating that the cycle has completed.

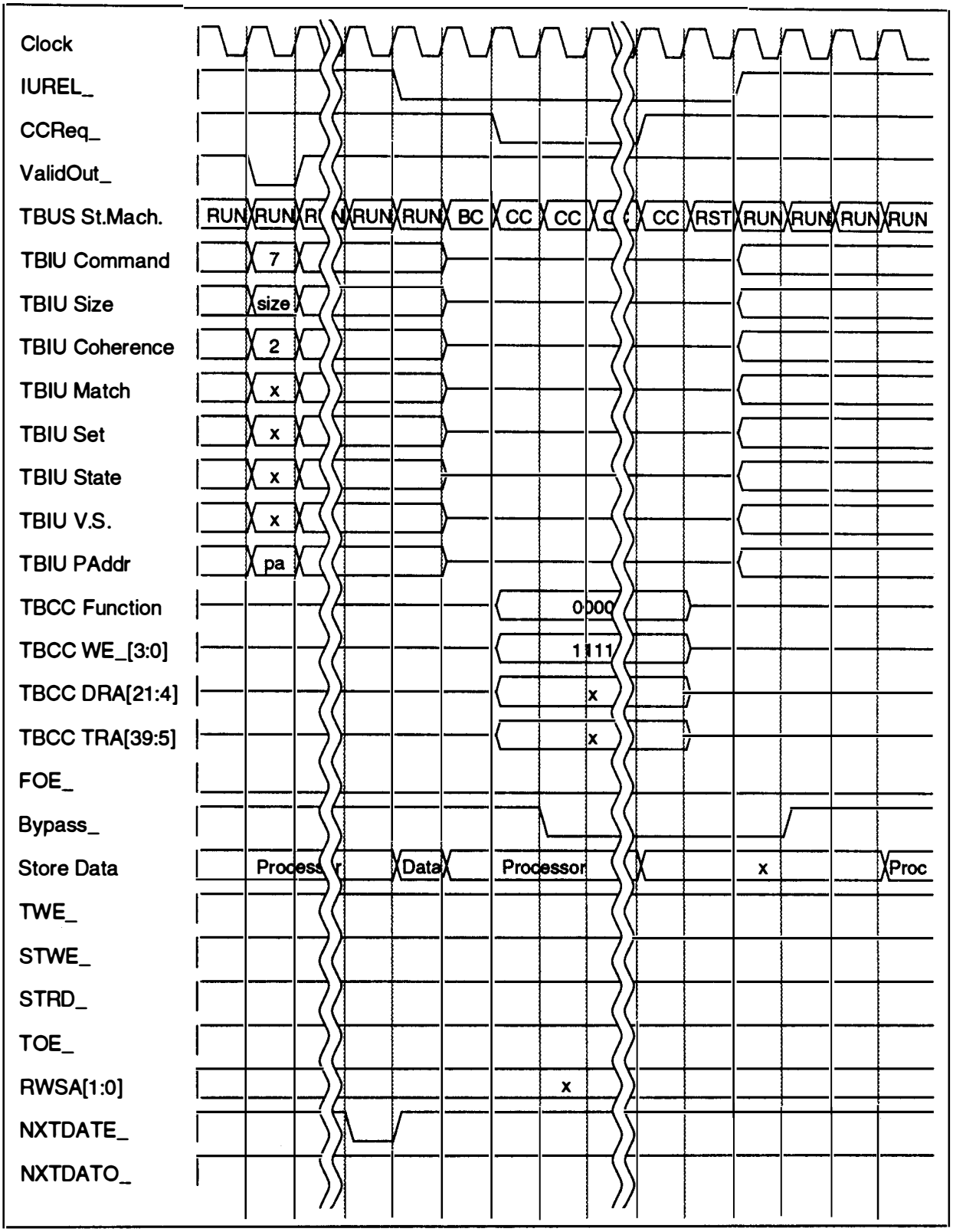


Figure 11-16 Sequential Ordered Non-Cachable Write

11.4.7 Interrupt

Interrupt detection and status is handled by the cache controller. The R8000 CPU does not have a specific interrupt pin but rather accepts interrupt information from the CC over the TBus. When the interrupt is detected by the CC, the contents of the interrupt register must be passed to the R8000 CPU so that proper servicing can begin.

After obtaining control of the TBus the CC places the value 0010 on the function field as shown in Figure 11-17, indicating an interrupt cycle. The TRA field contains the actual interrupt information on TRA [39:29]. The encoding of these bits is shown in Table 11-4.

IP Enable	NMI Enable	Bus Error Enable	IP Field	NMI Field	Bus Error Field
TB<39>	TB<38>	TB<37>	TB<36:31>	TB<30>	TB<29>

Table 11-4 TBus Interrupt Status Transfer

At the same time the information is placed on the TBus, the CC de-asserts the signal CCREQ_ to indicate that the cycle is over. Control is then returned to the R8000 CPU.

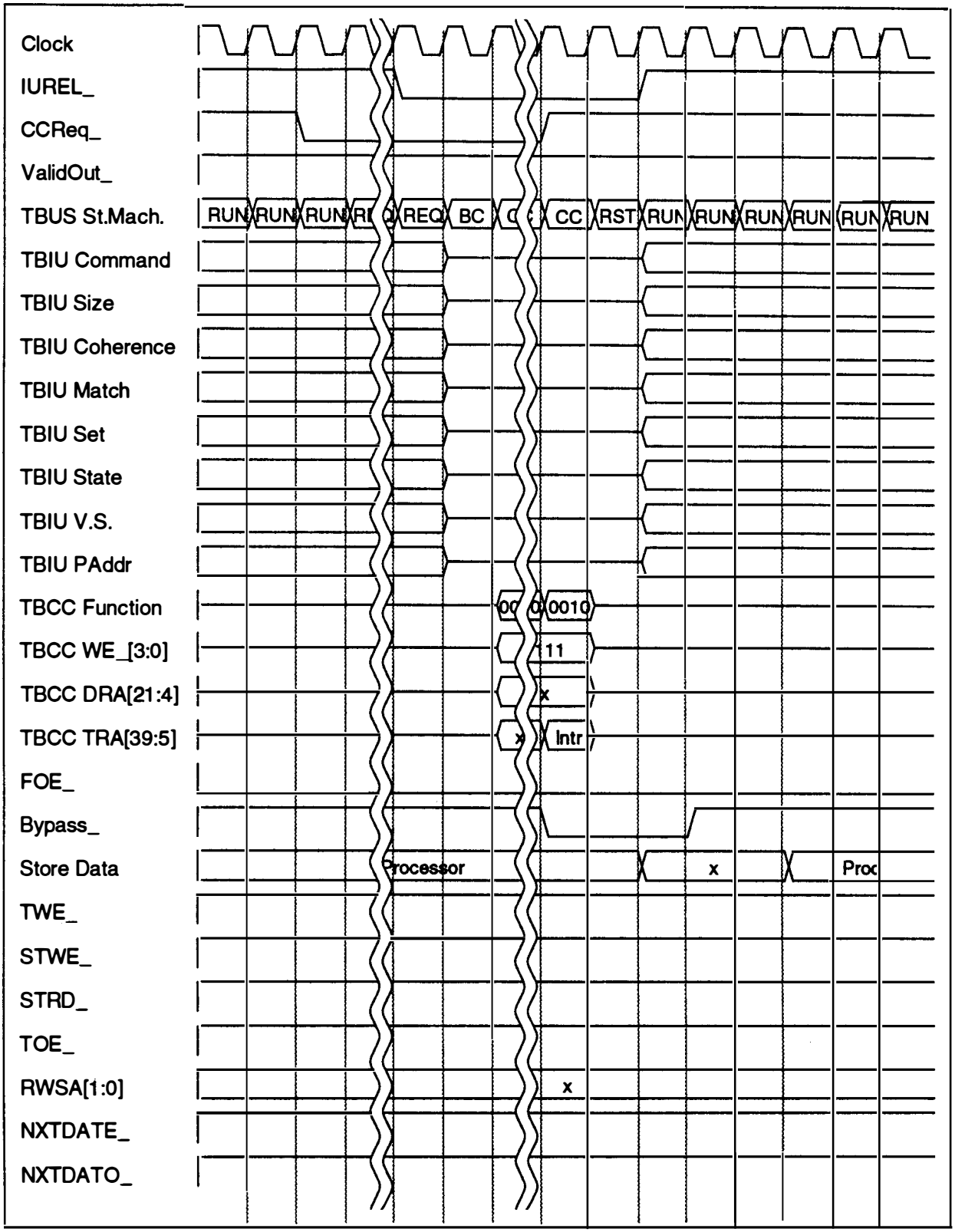


Figure 11-17 Interrupt Cycle

11.5 HARDWIRED CONTROL FUNCTIONS

Although the majority of communication between the CC and the R8000 Microprocessor Chip Set is done via the TBus, there are some hardwired control functions which the CC must support. The CC interfaces to the R8000 CPU, the R8010 FPU, and the Tag RAM's. The following table lists the hardwired control signals between the CC and each of these devices.

Signal	Type	Interface	Definition
VALIDOUT_	I	R8000 CPU	Bus information valid
IUREL_	I	R8000 CPU	R8000 CPU releases control of TBus
NXTDATE_	I	R8000 CPU	Even store address queue compare valid
NXTDATO_	I	R8000 CPU	Odd store address queue compare valid
CCREQ_	O	R8000 CPU	Cache Controller TBus request
FOE_	O	R8010 FPU	Floating point output enable
BYPASS_	O	R8010 FPU	Floating point data bypass control
RWSA<1:0>	O	TAG RAM	Read write set address
TWE_	O	TAG RAM	Tag address write enable
STWE_	O	TAG RAM	State and virtual synonym write enable
STRD_	O	TAG RAM	Read state information control
TOE_	O	TAG RAM	Output read information onto tag pins
TBus<72:0>	I/O	R8000 CPU/ R8010 FPU	Bus communication

Table 11-5 Cache Controller Hardwired Control Signals

11.5.1 Integer Unit Interface

VALIDOUT_ is driven by the R8000 CPU to indicate that the information on the TBus is valid for the Cache Controller to receive. The CC must monitor the state of VALIDOUT_ and be prepared to latch all 72 bits of the TBus whenever it is asserted.

IUREL_ is driven by the R8000 CPU and indicates to the CC that the R8000 CPU has relinquished control of the TBus. Assertion of IUREL_ causes transition of the TBus state machine to the 'CC' state. IUREL_ remains active as long as the R8000 CPU does not

have control of the bus. IUREL_ is de-asserted whenever the TBus state machine is in the 'RUN' state.

NXTDATE_ is driven by the R8000 CPU and indicates the result of the even store address queue compare. When the CC is in control of the TBus, the R8000 CPU asserts this signal low if the even store address queue contains an address for which bits <17:7> match the Tag RAM index bits <17:7> which were on the TBus four cycles earlier. The signal remains de-asserted if no such match is detected. NXTDATE_ should always be considered together with NXTDATO_. If either is asserted, a compare hit has occurred. Address comparisons are done on a 128 byte minimum quantity and take one cycle. Address range comparisons larger than 128 bytes require multiple cycles. NXTDATE_ is connected directly to the NXTDATE_ pin of the Cache Controller.

NXTDATO_ is driven by the R8000 CPU and indicates the result of the odd store address queue compare. When the CC is in control of the TBus, the R8000 CPU asserts this signal low if the odd store address queue contains an address for which bits <17:7> match the Tag RAM index bits <17:7> which were on the TBus four cycles earlier. The signal remains de-asserted if no such match is detected. NXTDATO_ should always be considered together with UPDE_. If either is asserted, a compare hit occurred. Address comparisons are done on a 128 byte minimum quantity and take one cycle. Address range comparisons larger than 128 bytes require multiple cycles. NXTDATO_ is connected directly to the NXTDATO_ pin of the Cache Controller.

CCREQ_ is driven by the CC and indicates to the R8000 CPU that the CC either has requested control of the TBus or does not wish to give up control of the TBus. Assertion of CCREQ_ causes the TBus state machine to transition to the 'REQ' state. A maximum of 1500 clocks can elapse between the time CCREQ_ is asserted and IUREL_ is finally asserted and control of the TBus granted.

11.5.2 Floating Point Unit Interface

FOE_ is driven by the CC and allows the R8010 FPU to drive data onto the store data bus. De-assertion of FOE_ tri-states the store data pins. FOE_ works in conjunction with the BYPASS_ on streaming cache transfers to main memory and allows data to be driven directly from the load data pins (LDE<63:0> and LDO<63:0>) to the store data pins (SDE<63:0> and SDO<63:0>) of the R8010 FPU respectively. When data from the streaming cache is to be transferred to main memory, the Cache Controller initiates a streaming cache load. The load data is then transferred on the LDE or LDO pins to the R8010 FPU. The CC controls the flow of data through the R8010 FPU by asserting the FOE_ and BYPASS_ signals to the R8010 FPU, allowing data to be driven onto the SDE or SDO busses. Refer to figure 10-22 for a graphical representation of how these pins manage the flow of data through the R8010 FPU.

BYPASS_ (Floating Point Bypass) Active Low Input

Assertion of BYPASS_ allows data from the even and odd load data pins to bypass the internal circuits of the R8010 FPU and be transferred directly to the store data output buffers. The assertion of FOE_ enables the buffers and allows the streaming cache load data to be driven out onto the store data pins. BYPASS_ de-asserted allows data from within the registers of the R8010 FPU to be driven out onto the SDE or SDO pins (assuming FOE_ is active). Refer to figure 11-22 for a graphical representation of how these pins manage the flow of data through the R8010 FPU.

11.5.3 Tag RAM Interface

The Cache Controller does not distinguish between even and odd Tag RAM's. Therefore each interface pin described below connects to both the even and odd Tag RAM's.

The two bit **RWSA<1:0>** field is driven by the CC when the Cache Controller is reading or writing to the Tag RAM. RWSA<1:0> are used to choose between one of the 4 sets of the 4-way set associative tag RAM. The Dirty Bit RAM also uses these pins to update the correct dirty bit entry. The values placed on the RWSA pins by the CC are obtained from the R8000 CPU during a Tag RAM lookup cycle and are passed to the CC via the Set Address field of the TBus.

STRD_ is driven by the CC and is used to control multiplexor logic inside the Tag RAM which enables either State and V.S. information, or tag address information, onto the internal 20 bit tag bus. When the Tag RAM is read by the CC, the TOE_ pin must also be asserted to enable this information onto the external Tag address bus. The information enters the R8000 CPU and is returned to the CC via the TBus.

TOE_ is driven by the CC when the CC is reading the Tag RAM. Assertion of OE_ allows the information in the Tag RAM to be driven out onto the external 20 bit Tag Address bus and back to the R8000 CPU where it is returned to the CC via the TBus.

STWE_ is asserted whenever the CC is writing state and virtual synonym information to the Tag RAM. Both the tag address and the state and V.S. information are carried to the Tag RAM via a single 20 bit tag address bus. The CC tracks the information on the pins and asserts STWE_ if state and V.S. information is to be written. TWE_ is asserted if tag address information is to be written. TWE_ and STWE_ must never be asserted at the same time.

TWE_ is driven by the CC whenever tag address information is to be written to the Tag RAM. Refer to STWE_ above.

11.5.4 TBus Interface

This 72 bit bidirectional bus goes between the CC, the R8000 CPU, and the R8010 FPU. The function of each bit changes depending on which device is driving. Normally the CC drives the TBus when reading or writing the tag RAM's or reading the Data RAM's and for general communication with the R8000 CPU. The R8010 FPU uses the TBus to transfer Move data from the floating point register file (FPR) to the Integer Register File of the R8000 CPU as requested by the R8000 CPU. The R8000 CPU uses the TBus for integer stores to the data RAM's, general communication with the CC and the R8010 FPU, and R8010 FPU to R8000 CPU move instructions. TBUS<63:0> connects directly to TBUS<63:0> of the R8010 FPU as well as TBUS<63:0> of the Cache Controller.TBus connection between the CC and the R8010 FPU is by virtue of the fact the R8000 CPU communicates with both. There is no TBus communication protocol between the R8010 FPU and the CC. Chapter 10 discusses the TBus interface in detail.

11.5.5 Data Bus Interface

There is no direct data bus interface between the CC and any other device in the system. However, the CC is responsible for the flow of data between the store data busses of the even and odd banks of the streaming cache, and the even and odd data buffers respectively. The data buffers separate the R8000 Microprocessor from the main memory back-plane bus. SDE[63:0] and SDO[63:] comprise the even and odd store data busses respectively.

SYSTEM CONTROL COPROCESSOR - INSTRUCTION SET DETAILS

A

This appendix provides a detailed description of the operation of the System Control Coprocessor (Coprocessor 0) instructions implemented by the R8000 Microprocessor. The instructions are listed in alphabetical order.

Exceptions that may occur due to the execution of each instruction are listed after the description of each instruction. Descriptions of the immediate cause and manner of handling exceptions are omitted from the instruction descriptions in this appendix.

Tables at the end of this appendix list the bit encoding for the constant fields of each instruction, and the bit encoding for each individual instruction is included with that instruction.

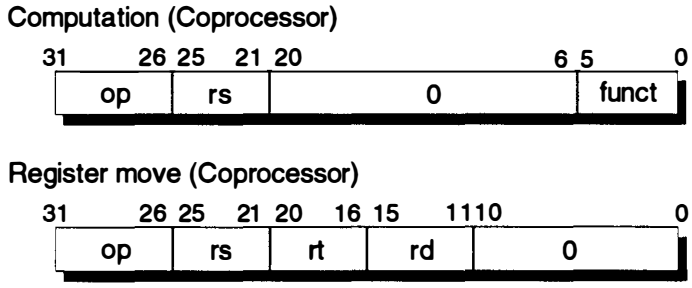
A.1 System Control Coprocessor Instructions

The MIPS architecture provides a uniform abstraction for a few coprocessor units, alternate execution units with register files separate from the R8000 CPU. The System Control function of MIPS processors is implemented using this mechanism as coprocessor 0. The System Control Coprocessor manipulates the processor control, memory management, and exception handling facilities of the processor. Though many processors have similar system control facilities, the System Control Coprocessor instructions are R8000 CPU-specific.

A.2 Instruction Formats

Every R8000 CPU instruction consists of a single word (32 bits) aligned on a word boundary. The instructions to transfer data between general registers and coprocessor registers have one format, common to most coprocessors. Coprocessor computational instructions have coprocessor-dependent formats. The CP0 instructions have the major opcode of COP0, and specify further operations with subfields

.



op	6-bit operation code (COP0 for all these instructions)
rs	5-bit specifier: select move to/from or operation instruction
rt	5-bit R8000 CPU source/destination general register specifier
rd	5-bit Coprocessor source/destination general register specifier

Table A-1 CP0 Instruction Formats

A.3 Instruction Notation Conventions

In this appendix, all variable subfields in an instruction format (such as *rs, rt, immediate,* etc.) are shown in lowercase names.

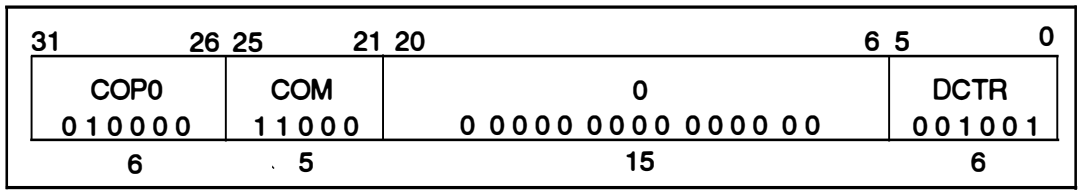
The bit encoding for the opcode constants accompanies each instruction and is also summarized in figures located at the end of this Appendix. Fields which are not shown as opcode constants, but are shown to contain zero are reserved fields and must be coded as zeros for correct operation.

In the instruction descriptions that follow, the *Operation* section describes the operation performed by each instruction using a high-level language notation.
The registers in Coprocessor 0 are referred to by name in the high-level language.
Special symbols and functions used in the notation are described in the table below.

Symbol	Meaning
←	Assignment.
||	Bit string concatenation.
xy	Replication of bit value *x* into a *y*-bit string. Note: *x* is always a single-bit value.
xy..z	Selection of bits *y* through *z* of bit string *x*. Little-endian bit notation is always used. If *y* is less than *z*, this expression is an empty (zero length) bit string.
+ - *	2's complement or floating-point arithmetic: addition, subtraction, multiplication
div	2's complement integer division.
mod	2's complement modulo.
/	Floating-point division.
<	2's complement less than comparison.
nor	Bit-wise logical NOR.
xor	Bit-wise logical XOR.
and	Bit-wise logical AND.
or	Bit-wise logical OR.
GPR[*x*]	General-Register x. The content of GPR[0] is always zero.
CPR[*z*,*x*]	Coprocessor unit *z*, general register *x*.
$REGISTER_{FIELD}$	The value of the field "FIELD" in the specified register or structure. This is similar to the bit selection notation, but uses a field name.
LLbit	Bit of state to specify synchronization instructions. Set by *LL*, cleared by *ERET* and *Invalidate* and read by *SC*.
TLB[index, set]	The TLB cache entry selected by a specified index and set value. This has fields HI, LO, VPN (in HI) and ASID (in HI).
DCache_Tag_Ram[VAddr]	The Data Cache Tag Ram entry indexed by the address in the VAddr register.
TLBIndex(VAddr, KPS, UPS)	The TLB cache index for the address in the VAddr register given the current processor operating mode and the current page sizes.

Table A-2 CP0 Instruction Operation Notations

DCTR Data Cache Tag Read DCTR



Format: **TFP specific**

DCTR

Description:

The *VAddr* register specifies an entry in the Data Cache Tag Ram. DCTR reads the tag from the specified Data Cache Tag entry and writes it to the *DCache* register.

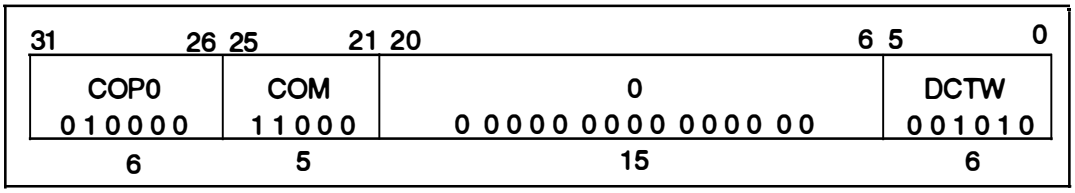
Operation:

DCache ← DCache_Tag_Ram[VAddr]

Exceptions:

Coprocessor unusable exception

DCTW Data Cache Tag Write DCTW



Format: **TFP specific**

DCTW

Description:

The *VAddr* register specifies an entry in the Data Cache Tag Ram. DCTR reads the tag from the *DCache* register and writes it into the specified Data Cache Tag entry.

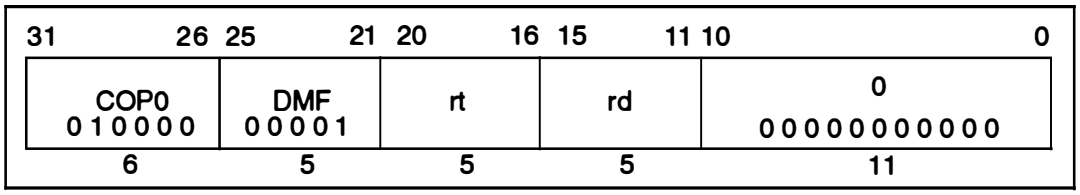
Operation:

DCache_Tag_Ram[VAddr] ← DCache

Exceptions:

Coprocessor unusable exception

DMFC0 Doubleword Move From System Control Coprocessor DMFC0



Format: **TFP specific**

DMFC0 rt, rd

Description:

The contents of the system control coprocessor (CP0) general register *rt* are loaded into R8000 CPU general register *rd*. This instruction is used by software to read from the system control registers.

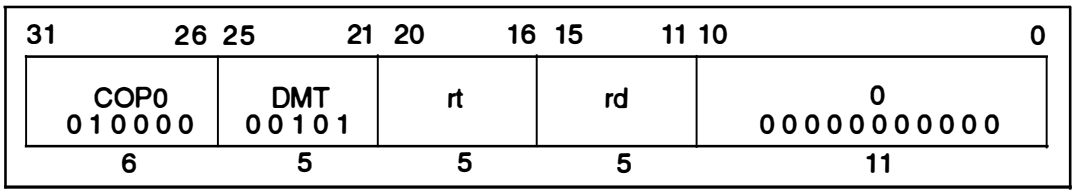
Operation:

GPR[rd] ← CPR[0,rt]

Exceptions:

Coprocessor unusable exception

DMTC0 Doubleword Move To System Control Coprocessor DMTC0



Format: **TFP specific**

DMTC0 rt, rd

Description:

The contents of R8000 CPU general register *rt* are loaded into system control coprocessor (CP0) general register *rd*. This instruction is used by software to write to the system control registers.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load and store instructions and TLB operations within an implementation-dependent window, prior to and after this instruction, are undefined.

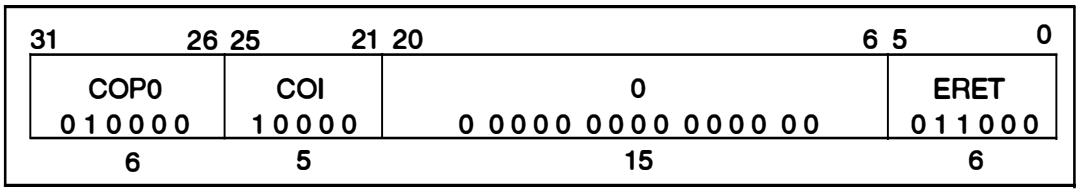
Operation:

CPR[0,rd] ← GPR[rt]

Exceptions:

Coprocessor unusable exception

ERET Exception Return ERET



Format: **TFP specific**

ERET

Description:

ERET is the instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction. ERET must not itself be placed in a branch delay slot.

An ERET executed between a LL and SC causes the SC to fail.

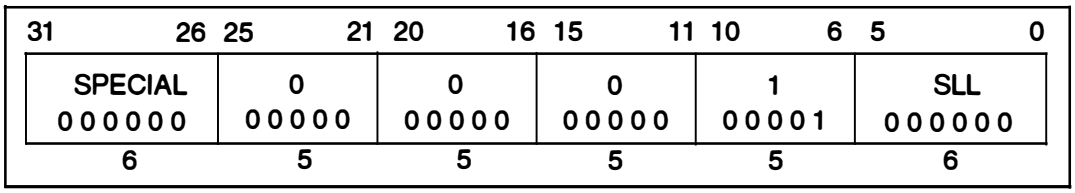
Operation:

PC ← EPC
SR_1 ← 0
LLbit ← 0

Exceptions:

Coprocessor unusable exception

SSNOP Superscalar Inhibit NOP SSNOP



Format: **TFP Specific**

SSNOP

Description:

This is a No-OPeration instruction that alters the superscalar instruction dispatch behavior of the TFP processor. No instruction that follows a SSNOP in program order can be dispatched until after the SSNOP has been dispatched. The SSNOP can be dispatched in the same cycle as instructions that precede it.

For example, consider the sequence:

```
dmtc0  r4,VAddr
ssnop
ssnop
ssnop
tlbw
```

The DMTC0 that writes the *VAddr* register and the first SSNOP can issue in the same cycle, say T, but the second SSNOP can't issue because of the first SSNOP. The second SSNOP will issue in cycle T+1 by itself. The third SSNOP will issue in cycle T+2. Finally the TLBP (and following instructions) can issue in cycle T+3.

This is not a coprocessor 0 instruction. It is documented with the TFP CP0 instructions because it is TFP-specific and the primary use is to serialize code sequences that perform system control functions.

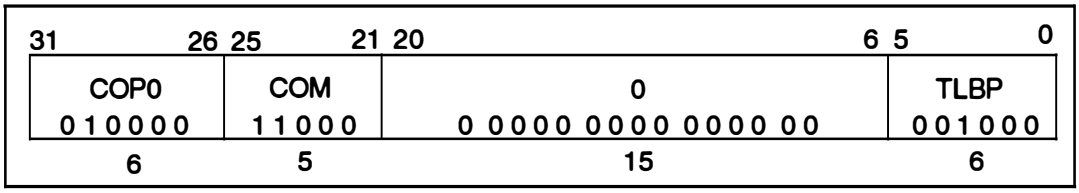
The instruction is the encoding of "SLL $0,$0," but the mnemonic is recognized by the TFP assembler and the instruction is recognized as the special superscalar inhibit operation by the processor and is not treated as a shift.

Operation:

Exceptions:

None

TLBP Probe TLB For Matching Entry TLBP



Format: **TFP specific**

TLBP

Description:

The TLB cache is probed for a translation that matches the address in the *VAddr* register and the ASID in the *EntryHi* register. The TLB cache index is determined from the value in the *VAddr*

The result of the probe is recorded in the *TLBSet* register. If there is a match, then the *P* bit is cleared to zero, and the set number is recorded in the *SET* field. If there is no match, then the *P* bit is set to one and the *SET* field is undefined.

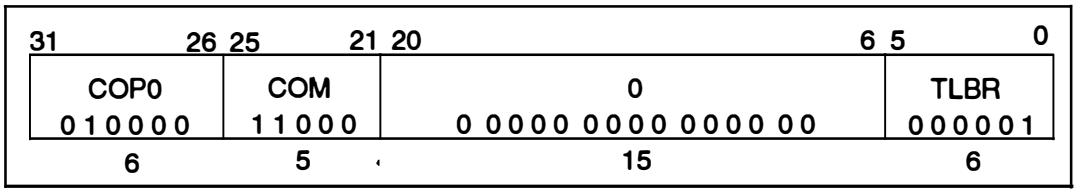
Operation:

```
index ← TLBIndex(VAddr, KPS, UPS)
TLBSetP ← 1
/* this is an associative match in actual hardware */
for i in 0..2
    match ← (VAddrREGION = TLB[index, i]REGION)
            and (VAddrVPN = TLB[index, i]VPN)
            and (EntrHIASID = TLB[index, i]ASID)
    if match = true then
        TLBSetP ← 0
        TLBSetSET ← matching set number
    endif
endfor
```

Exceptions:

Coprocessor unusable exception

TLBR Read Indexed TLB Entry TLBR



Format: **TFP specific**

TLBR

Description:

The TLB entry is specified by the *VAddr* register and the *TLBSet* register. The contents of the specified TLB entry is moved to the *EntryHi* and *EntryLo* registers.

This operation is undefined for an incorrectly specified *SET* address.

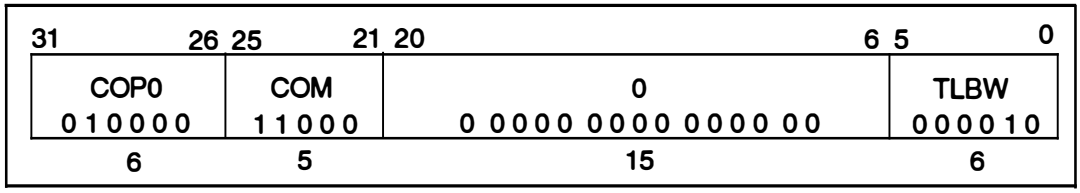
Operation:

```
index ← TLBIndex(VAddr, KPS, UPS)
set ← TLBSet1..0
EntryHi ← TLB[index, set]Hi
EntryLo ← TLB[index, set]Lo
```

Exceptions:

Coprocessor unusable exception

TLBW Write Indexed TLB Entry TLBW



Format: **TFP specific**

TLBW

Description:

The TLB entry is specified by the *VAddr* register and the *TLBSet* register. The contents of the *EntryHi* and *EntryLo* registers is moved into the specified TLB entry.

These operations are undefined for an incorrectly specified *SET* address.

Operation:

```
index ← TLBIndex(VAddr, KPS, UPS)
set ← TLBSet1..0
TLB[index, set]Hi ← EntryHi
TLB[index, set]Lo ← EntryHi
```

Exceptions:

Coprocessor unusable exception

TFP Coprocessor 0 – Opcode Bit Encoding

All TFP System Control Coprocessor instructions have major opcode COP0. The encoding of the COP0 major opcode is shown here with all other major opcodes omitted.

bits [31...29]	[28..26] Major Opcode							
	0	1	2	3	4	5	6	7
2	COP0							

bits [25..24]	[23..21] COP0 rs							
	0	1	2	3	4	5	6	7
0	*	DMF	*	*	*	DMT	*	*
1	*	*	*	*	*	*	*	*
2	COI							
3	COM							

bits [5..3]	[2..0] COP0 function when rs equals COI							
	0	1	2	3	4	5	6	7
0	*	*	*	*	*	*	*	*
1	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	ERET	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

bits [5..3]	[2..0] COP0 function when rs equals COM							
	0	1	2	3	4	5	6	7
0	*	TLBR	TLBW	*	*	*	*	*
1	TLBP	DCTR	DCTW	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

HAZARDS AND INTERLOCKS

B

The R8000 Microprocessor contains certain restrictions which must be adhered to in order to attain maximum instruction execution performance. Certain combinations of instructions are not permitted and the results of executing these illegal combinations can be unpredictable. Most hazards result from instructions modifying and reading a state in different pipeline stages. Such hazards are defined between pairs of instructions, not on a single instruction in isolation. These constraints are discussed throughout this appendix. Refer to chapter 6, section 6.5 for information regarding hazards during the boot-up procedure.

B.1 The MiscBus

The MiscBus is a 64-bit utility bus internal to the R8000 CPU and is used for transferring data in situations where dedicated buses are not available. Instructions which use the MiscBus include JAL, MFC0, MTC0. It is important to note that the MiscBus is a single resource and is not controlled by score-boarding. Certain restrictions apply when using the MiscBus.

- Only one MFC0 instruction at a time can occur in the E-stage of the pipeline.
- Only one MTC0 instruction at a time can occur in the W-stage of the pipeline.
- Only one JAL instruction at a time can occur in the E-stage of the pipeline.

The MoveFrom and MoveTo Coprocessor 0 instructions both use the MiscBus to transfer data between the general purpose register (GPR) and the coprocessor 0 registers. However, these instructions use the MiscBus in different cycles of the pipeline. MFC0 uses the MiscBus in the E-stage, while MTC0 uses the MiscBus in the W-stage. The following restriction applies.

A MFC0 instruction must not occur in the cycle following a MTC0 instruction. Hence a MFC0 instruction in the E-stage of the pipeline cannot be followed by a MTC0 instruction in W-stage.

A JAL uses the MiscBus to pass the program counter value from the instruction unit to the execution unit and stores the result into register r31 of the GPR. JAL uses the MiscBus in the E-stage, as does the MFC0 instruction. Therefore, a hazard exists if JAL and MFC0 attempt to execute in the same cycle. The following restriction applies.

The JAL and MFC0 instruction must not occur at the same time in the E-stage of the pipeline.

In addition, the MTC0 instruction uses the MiscBus in the W-stage of the pipeline. Due to the interdependency of these instructions, a hazard occurs if a MTC0 instruction in the W-stage is followed by a JAL instruction in E-stage, since the JAL is actually issued before the MTC0 instruction. The following restriction applies.

A JAL instruction must not occur in the cycle following a MTC0.

B.2 The Address Generation Pipeline

The R8000 CPU has two address generation pipelines that are used for all loads and stores. Instructions flow down the pipeline through stages A-E-W. Coprocessor 0 memory (Cop0mem) instructions are those Coprocessor 0 instructions that use the memory pipeline, as opposed to using the integer pipeline.

However, Coprocessor 0 memory instructions do not flow through the pipeline in the same manner as other loads and stores. Coprocessor 0 memory instructions use the address pipeline two cycles later than a normal load/store memory operation. Hence, if a memory operation follows a Coprocessor 0 memory operation by two cycles in the pipeline, the normal memory operation (loads/stores) is aborted in it's E-stage, the pipeline is flushed, and the memory operation is restarted.

B.3 Coprocessor 0 Register Latencies

This section lists each of the Coprocessor 0 registers and the latencies incurred when executing certain instructions.

B.3.1 Virtual Address (VAddr) Register

The VAddr register is used in addressing Coprocessor 0 memory instructions. When a Coprocessor 0 memory instruction is in the E-stage, the VAddr is forced back into the memory pipeline in the D-stage. When executing a MTC0 instruction using the VAddr register and expecting the new value of VAddr to index either the TLB of the Data Cache, the MTC0 instruction has a latency of 2. The 2-cycle latency is shown below.

Cycle	
0	ssnop instruction issued in cycle 0
1	MTC0 VAddr instruction issued in cycle 1
1	ssnop also issued in cycle 1
2	ssnop issued in cycle 2
3	TLBR issued in cycle 3 two cycles after MTC0

B.3.2 Status Register

The KPS/UPS fields of the Status Register may affect addressing of the TLB for the

TLBR, TLBW, TLBP Coprocessor instructions. If the region of the address in the VAddr register points to kernel physical (KP) space, the Status Register page size fields have no effect on the indexing of the TLB. The Status register has no effect on the DCTR and DCTW Coprocessor instructions.

If the region of the address in VAddr register points to a virtual address space (as opposed to kernel physical space), then the KPS/UPS field is used to choose the lower 7-bits of the virtual page. The MTC0 Status Register has a latency of 1 to the TLBR, TLBW, and TLBP Coprocessor instructions. The one cycle latency is shown below.

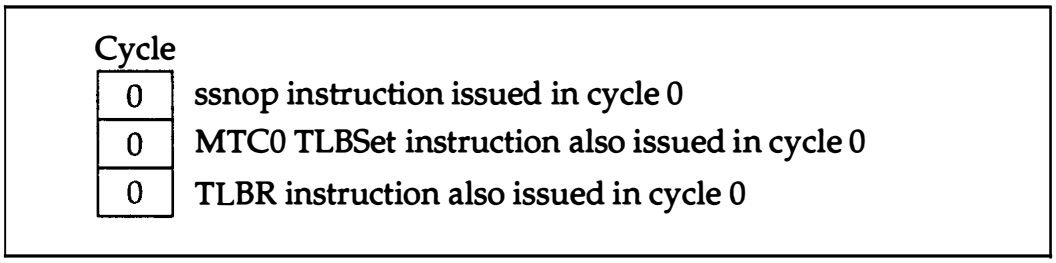
Cycle	
0	ssnop instruction issued in cycle 0
1	MTC0 Status Register (SR) instruction issued in cycle 1
1	ssnop also issued in cycle 1
2	TLBR issued in cycle 2 one cycle after MTC0

B.3.3 TLBSet Register

The TLBSet register is used to determine the status of the TLB write-enables when the TLBW instruction is in the W-stage. The MTC0 TLBSet operation has a latency of 1 to a TLBW instruction. TLBSet has a latency of 0 to a TLBR instruction. The one cycle latency of a TLBW operation is shown below.

Cycle	
0	ssnop instruction issued in cycle 0
1	MTC0 TLBSet instruction issued in cycle 1
1	ssnop also issued in cycle 1
2	TLBW issued in cycle 2 one cycle after MTC0

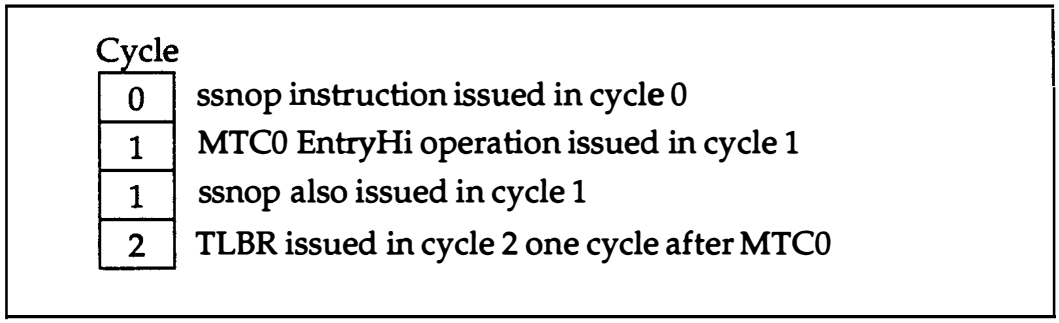
The zero cycle latency of a TLBR operation is shown below.



B.3.4 EntryHi Register

The EntryHi register serves as a data register for the TLBW instruction to write the TLB. TLBW picks up the data from EntryHi when the TLBR instruction is in the H-stage. A MTC0 instruction writes to EntryHi at the end of the W-stage. Since the TLBW picks up the value from EntryHi so late in the pipeline, the MTC0 is 0 latency with respect to the TLBW.

The MTC0-EntryHi operation can come down the pipeline in the same cycle with the TLBW, and the TLBW will get the new data. EntryHi also may affect the index into the TLB for TLBR, TLBW, and TLBP operations. If the address in the VAddr register points to Kernel Global (KV1) space, then the 8-bit ASID field in EntryHi does not affect the indexing. Otherwise, the ASID is XOR'd with the least significant 7-bits of the virtual page number to form an index into the TLB. For this reason the MTC0 operation must occur one cycle before the TLBW, TLBR, or TLBP operations in the pipeline. The MTC0 has a latency of one cycle with respect to these instructions. The TLBR operation is shown below.



The TLBR operation loads the EntryHi and EntryLo registers late in the pipeline. TLBR has a latency of 3 to a MFC0 operation. The MFC0-EntryHi/EntryLo must come 3 cycles after the TLBR operation. The three cycle latency is shown below. The following sequence places new data in EntryHi/EntryLo from a TLBR.

Cycle	
0	TLBR instruction issued in cycle 0
0	ssnop also issued in cycle 0
1	ssnop issued in cycle 1
2	ssnop issued in cycle 2
3	MFC0-EntryHi issued in cycle 3 three cycles after TLBR

There is no score-boarding between two writes to the same register, such as between TLBR and MTC0-EntryHi operations, both of which target the EntryHi register. For example, if a MTC0-EntryHi or MTC0-EntryLo operation occurs in the next cycle after a TLBR, the result of the MTC0 will be put into EntryHi or EntryLo respectively. The following restrictions apply.

- A DCTR operation in H-stage cannot occur at the same time as a MTC0-DCACHE operation in W-stage.

- A TLBR operation in H-stage cannot occur at the same time as a MTC0-EntryHi/ EntryLo operation in W-stage.

- A TLBP operation in H-stage cannot occur at the same time as a MTC0-TLBSet operation in W-stage.

B.4 VAddr Multiplexing

The execution of a MTC0-VAddr operation in W-stage and any Coprocessor 0 memory instruction in E-stage require a multiplexor in the pipeline which is a single resource. If a Coprocessor 0 memory executes in the cycle after a MTC0-VAddr operation, a collision for this resource occurs. Hence the following restriction applies.

- A MCT0-VAddr operation must not occur in the cycle following a Coprocessor 0 memory instruction.

B.5 Integer Store Cancellation

In order to alleviate a potential timing problem that can occur when cancelling integer stores a mechanism exists in the R8000 CPU which recirculates the store address back through the address pipeline to invalidate the store data in the data cache while the pipeline is being flushed. This mechanism requires the use of the same multiplexor mentioned in the previous example and if used can collide with a MTC0-VAddr executing in the same cycle (W-stage). To avoid this problem, the following restriction applies.

- Do not schedule Integer stores in the same cycle with a MTC0-VAddr.

B.6 Instruction Latency and Control Registers

This section describes the latency of instructions which affect address translation.

B.6.1 TLBW Instruction

The TLBW instruction uses the memory pipeline in a different fashion than loads and stores. While these standard memory operations access the TLB in the E-stage, the TLBW instruction does not access the TLB until two cycles later in the H-stage. At the end of the H-stage, the TLB is written with a new value which can be used for translation. Hence a memory operation can occur 3 cycles behind a TLBW instruction and receive the new translation. The actual latency of a TLBW instruction is 3 cycles.

It is possible for the effective latency of the TLBW instruction can be 0 at the end of an exception handler. For example, if the TLBW is at the end of an exception handler, indicating a Return From Exception (ERET) is coming, the ERET causes a pipeline flush and creates a bubble in the pipeline of 3 cycles.

If a TLBW is placed at the end of a fast TLB exception handler in order to place a translation into the TLB, and an ERET is then executed which causes the code to return to the instruction that caused the TLB miss, no ssnop operations are needed to retrieve the required 3 cycles following the TLBW and before the use of the translation.

B.6.2 MTC0-Status Register (UPS/KPS Fields)

The MTC0-SR operation has a latency of 3. Three cycles after a MTC0-SR, new values in the KPS and UPS fields will affect address translation. MTC0 takes effect at the end of the W-stage. The page size is used in A-stage to set-up TLB controls.

B.6.3 MTC0-EntryHi (ASID)

The MTC0-EntryHi operation has a Latency of 3. Three cycles after a MTC0-EntryHi operation, the new value in the ASID field affects the hashing into the TLB. MTC0 takes effect at end of the W-stage. The ASID size is used in A-stage to hash TLB-index into the pre-decoder.

B.7 Use of the SSNOP Instruction

In addition to the standard NOP instruction, the MIPS IV instruction set includes a super-scalar no-op instruction (SSNOP). Insertion of a SSNOP instruction breaks superscalar dispatch in the TFP microprocessor and assures that no instruction that follows the SSNOP in the assembly language is dispatched in the same cycle as the SSNOP. The SSNOP instruction is provided to help the software developer adhere to certain restrictions inherent to the MIPS architecture, namely to keep certain instruction pairs separated by one or more cycles.

Table B-1 shows the insertion of SSNOP instructions as a way of separating certain instructions. However, in some cases single cycle integer operations can be substituted.

Instruction	Followed By	Number of SSNOP's	Comments
dmfc0	dmfc0	1	Move From any COP0 register
dmfc0	dmtc0	1	Move To/From any COP0 register
dmfc0	jal	1	Move From any COP0 register
dmfc0	jalr	1	Move From any COP0 register
dmtc0	dmfc0	2	Move To/From any COP0 register
dmtc0	dmtc0	1	Move To any COP0 register
dmtc0	jal	2	Move to any COP0 register
dmtc0	jalr	2	Move To any COP0 register
dmtc0 (Vaddr)	any TLB op	2	Move to Vaddr Register Only
dmtco (Status)	any TLB op	2	Move to Status Register Only
dmtc0 (TLBSet)	any TLB op	2	Move to TLBSet Register Only
dmtc0 (EntryHi)	any TLB op	2	Move to EntryHi Register Only
dmtc0 (EntryLo)	any TLB op	2	Move to EntryLo Register Only
dmtc0 (Status)	any Integer operation	4	When accessing SR to enable/disable interrupts.
dmtc0 (Status)	any FP operation	4	Four cycles before new interrupt mask takes effect.
any TLB op	any Memory op	3	Any TLB operation followed by any Memory operation.
tlbp	dmfc0 (TLBSet)	4	Move From TLBSet register only
tlbr	dmfc0 (EntryHi)	4	Move From EntryHi register only
tlbr	dmfc0 (EntryLo)	4	Move From EntryLo register only
dctr	dmfc0 (DCACHE)	3	Move From DCACHE register only
jal	dmfc0	1	Move From any COP0 Register
jal	dmtc0	1	Move From any COP0 Register
jalr	dmfc0	1	Move From any COP0 Register
jalr	dmtc0	1	Move To any COP0 Register

Table B-1: SSNOP Requirements