mips

Open RISC Technology

R4200 MICROPROCESSOR

Description:

The R4200 is a low-power, low-cost RISC processor targeted at embedded applications and portable computing. The R4200 is an ideal embedded processor, providing very high performance at a cost-effective price-point. Fully compatible with the R4000 family of high-performance microprocessors, the R4200 will power notebook, portable and low-cost desktop systems, running either Windows NT or UNIX. The R4200 is specifically designed to achieve low power consumption and low cost through small, high-yield die while retaining high levels of performance. It is this balance of power dissipation, cost and high performance that makes the R4200 uniquely suitable for the applications listed below.

Features:

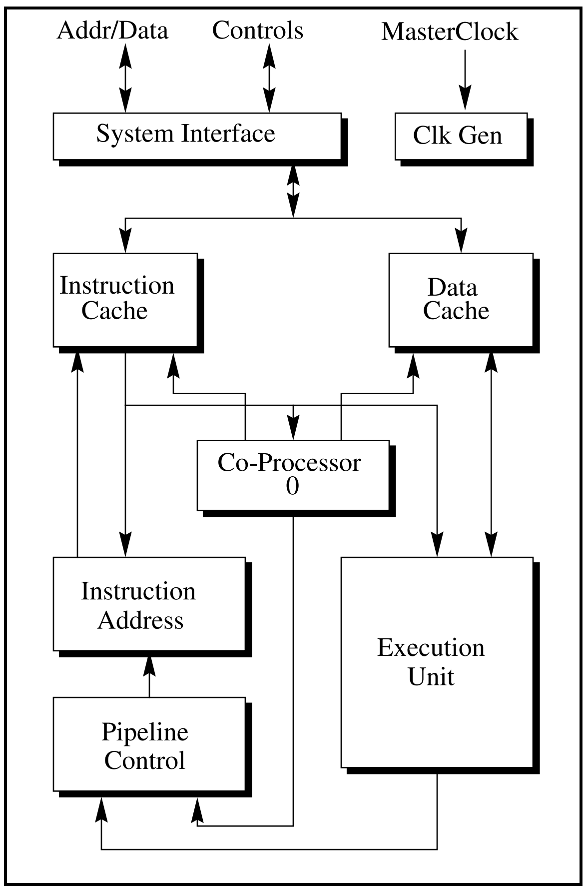
Low Power Dissipation:
- 1.5W (normal)
- 0.4W (reduced power)
- 0W (instant-off)

High Performance:
- 55 SPECint92
- 30 SPECfp92

High integration on-chip:
- 16K I-cache, 8K D-cache
- Unified datapath
- 32 double-entry J-TLB
- 2 entry micro I-TLB

Power Management Features:
- Reduced Power Mode
- Instant On/Off

R4000 compatibility:
- R4000PC software compatible
- R4000PC hardware compatible (R4200PC only) packaging



Applications:
- X-terminals
- Laser printers
- Notebook computers
- Low-cost desktop computers
- Low-cost workstations
- Factory automation
- Networking: routers, bridges
- Graphics acceleration
- Personal Digital Assistants

Packaging Options:
- R4200LP (208-pin PQFP)
- R4200PC (179-pin C-PGA)

Overview:

The R4200 is the first of a new family of processors based on the MIPS RISC architecture. This family of products meets the requirements of low-cost applications without compromising on 64-bit performance. It will put RISC levels of performance within reach of a much wider range of products and end-users.

The R4200 processor is remarkable for 3 reasons. It is the first microprocessor expected to offer a price/performance ratio of $1/SPECint. Secondly, it is the first high-performance RISC microprocessor with power consumption suitable for notebook computers. Thirdly, it boasts a number of unique design features which make both the above possible, in one chip. These include an all-dynamic design, unified integer and floating-point datapath, a reduced power mode and caches partitioned into separate banks for further power reduction.

There are a number of key applications which will be transformed by the availability of products based on the R4200. Notebook computers will at last run PC applications as well as most desktops. Portable computing devices of all types will receive >100% increases in processor performance. High-performance embedded applications will benefit from leading performance levels immediately, without having to wait for prices of these components to fall, as with past products.

R4200 ARCHITECTURE

DATA FORMAT

The R4200 defines a 64-bit double-word, a 32-bit word, a 16-bit halfword and an 8-bit byte. The byte ordering is configurable in either Big-endian or Little-endian format. Figures 1 & 2 show the ordering of bytes for Big-endian and Little-endian conventions.

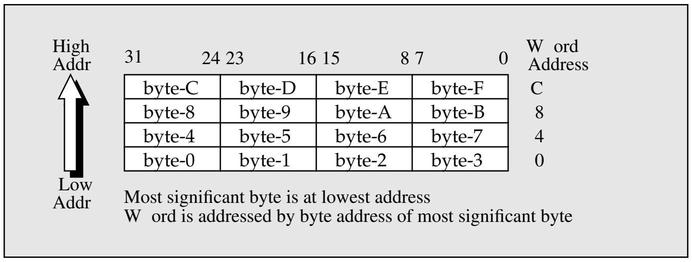


Figure 1: Big-endian Byte Alignment

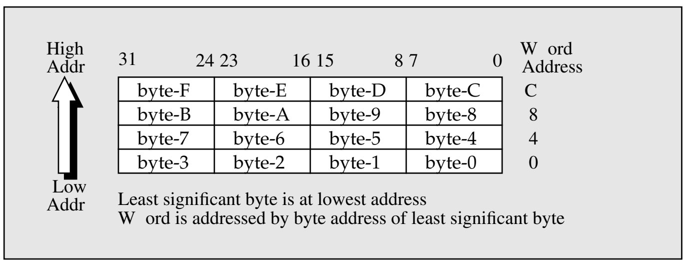


Figure 2: Little-endian Byte Alignment

PROCESSOR RESOURCES

The R4200 CPU provides sixty-four 64-bit wide registers. Thirty-two of these registers, referred to as General Purpose Register (GPRs), are reserved for integer operations while the other thirty-two register, referred to as Floating Point General Purpose Register (FGRs), are reserved for floating point operations.

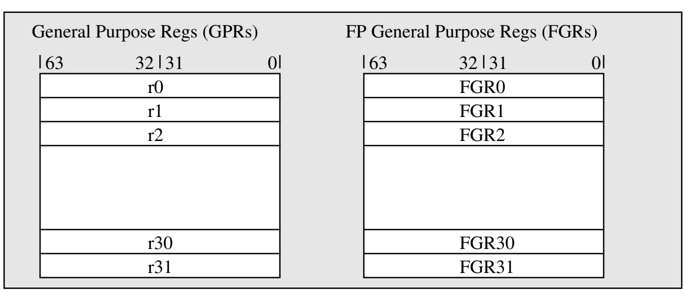


Figure 3: General Purpose Registers

The width of these registers depends on the mode of operation. In 32-bit mode, they are treated as 32 bits wide. In 64-bit mode, they are treated as 64 bits wide.

The R4200 also contains six special registers: the program counter (PC), multiply/divide result Hi and LO, load/link (LL) bit, and floating point implementation and control registers FCR0 and FCR31; shown in Figure 4. The program counter register contains the address of the current instruction. The multiply/divide registers store the result of integer multiply operations and the quotient and remainder of integer divide operations. The load/link bit is dedicated for load-link and store-conditional instructions which can be used to perform SYNC operations. The two floating point control register FCR0 and FCR31 provide the implementation/revision information and control/status of the floating point coprocessor (CP1)

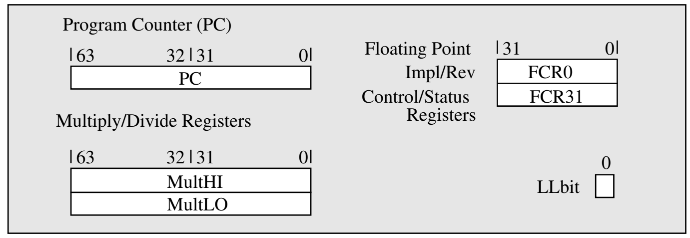


Figure 4: Special Registers

In addition, the R4200 operates with up to four tightly coupled coprocessors (designated CP0 through CP3). Coprocessor zero (CP0) is implemented as an integral part of the CPU and supports the virtual memory system together with exception handling. Coprocessor one (CP1) is reserved for the floating point unit, while coprocessor two and three are reserved for future use.

Coprocesssor zero contains registers shown in Figure 5 plus a 32-entry TLB with each entry consisting of an even/odd pair of physical addresses and tag field.

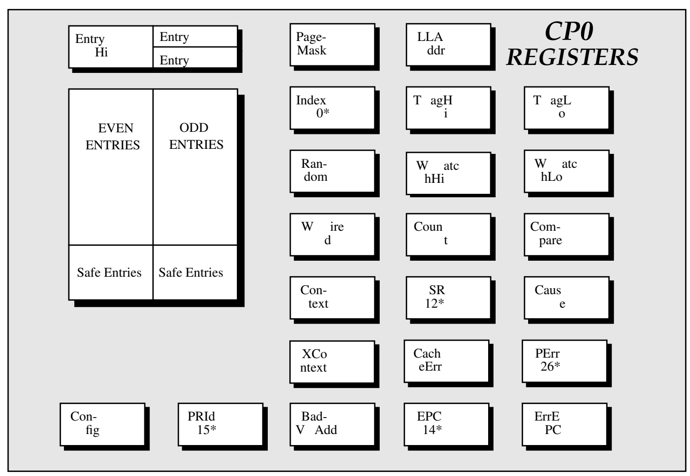
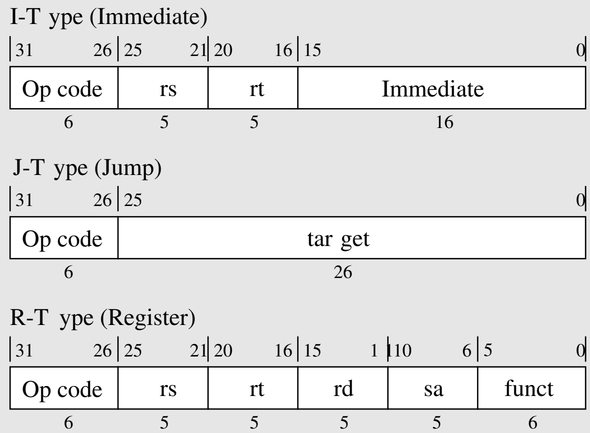


Figure 5: Coprocessor-0 Registers

The CP0 registers can be considered as two distinct functional sets. The first set comprises those which support the TLB operations. The second set is formed from those registers that reflect the status of the processor. All of the CP0 registers are accessible by software.

INSTRUCTION SET

All R4200 instructions are 32 bits (single word) long aligned on a word boundary. This approach simplifies instruction decoding, more complicated operations and addressing modes are synthesized by the compiler. There are three instruction formats: I-type (immediate), J-type (jump), and R-type (register), as shown in Figure 6.



Op code	is a 6-bit operation code
rs	is a 5-bit source register specifier
rt	is a 5-bit target register specifier
immediate	is a 16-bit immediate, branch displacement or address displacement
target	is a 26-bit jump target address
rd	is a 5-bit destination register specifier
sa	is a 5-bit shift amount
funct	is a 5-bit function field

Figure 6: Instruction Formats

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

Load and store instruction opcodes determines the access type which indicates the size of data item to be loaded or stored. Regardless of the access type or byte-numbering order (endianess), the address field specifies the lowest byte of the address location being accessed. Table 10 lists all load and store instructions defined by the MIPS Instruction Set Architecture (ISA) and ISA Extension.

Computational instructions perform arithmetic, logical, shift, multiply, and divide operations on values in registers. They occur in both register (R-Type) format, in which both operands are registers, and immediate (I-Type) format, in which one operand is a 16-bit immediate. When operated in 64-bit mode, 32-bit operands must be correctly sign extended. Table 11 lists all computational instructions defined by the MIPS ISA and ISA Extensions.

Jump and branch instructions change the control flow of a program. They occur in both register (R-Type) format, and immediate (I-Type) format. All jump and branch instructions occur with an architecture delay of one instruction; that is, the instruction immediately following the jump or branch is always executed while the target instruction is being fetched. Table 12 summarizes jump and branch instructions defined by the MIPS ISA and ISA Extensions.

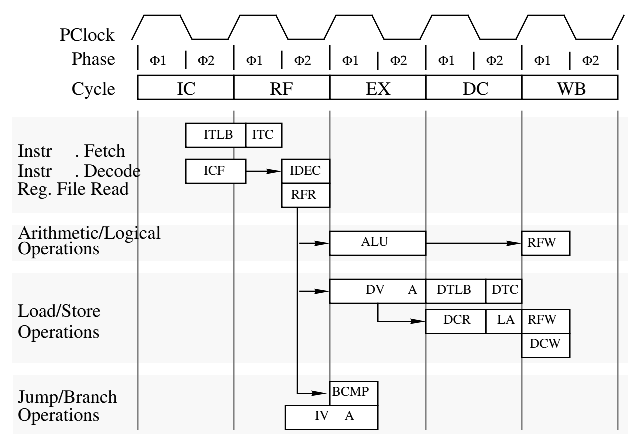
Special instructions allow the software to initiate traps and are always R-Type. Special instructions that are valid for all MIPS R-Series processors are shown in Table 12.

Exception instructions offer trapping mechanism to assist in software debug. They are shown in Table 14.

Coprocessor instructions perform operations in the respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor dependent formats. Table 12 summarizes the coprocessor instructions that are valid in all MIPS R-Series processor and the extensions to the MIPS ISA.

PIPELINE

The R4200 processor has a five-stage execution pipeline. Each pipeline stage takes one pcycle (pclock runs at twice the frequency of MasterClock). The execution of each instruction thus has a latency of at least five pcycle. Once the pipeline has been completely filled, five instructions are always being executed simultaneously. When the pipeline is not stalled, the processor has an throughput of one instruction per pcycle. The pipeline is in-order issue, in-order execution, and in-order completion, the same order as in the instruction stream.



Cycle	Phase	Mne-monic	Descriptions
IC	Φ1		No activity
	Φ2	ICF ITLB	Instruction Cache Fetch Instruction micro-TLB read
RF	Φ1	ITC	Instruction cache Tag Check
	Φ2	RFR IDEC IVA	Register File Read Instruction Decode Instruction Virtual Address Calculation
EX	Φ1	BCMP ALU DVA	Branch Compare Arithmetic Logic Operation Data Virtual Address Calculation
	Φ2		No new activity started
DC	Φ1	DCR DTLB	Data Cache Read Data TLB look-up
	Φ2	LA DTC	Load Data Alignment Data Cache Tag Check
WB	Φ1	DCW RFW	Data Cache Write Register File Write
	Φ2		No activity

Figure 7: Pipeline Activities

The five stages of the R4200 pipeline are: Instruction Cache (IC), Instruction Decode and Register File Read (RF), Execution (EX), Data Cache Read (DC), and Register File or Data Cache Write Back (WB). Figure 7 shows the activities occurring during each pipeline stage for ALU, load and store, and branch instructions. Below are detail descriptions of the activities during each stage.

- IC : An instruction address is presented by the address unit and the instruction cache fetch begins. The instruction micro-TLB starts the virtual-to-physical address translation.
- RF : The instruction becomes available and the instruction decoder decodes the instruction and check for interlock conditions. The instruction cache tag is checked against the page frame number obtained from the micro-TLB. Any required operands are read from the register file and the result from the EX or DC stages is bypassed to the following EX stage if required. The address unit generates next instruction address.
- EX : For ALU class instructions, the ALU performs an arithmetic or logical operation. For load and store class instructions, the ALU generates the data virtual address. For branch instructions, the ALU determines whether the branch condition is true .
- DC : For load and store instructions, the data cache is accessed and data virtual-to-physical address translation is performed. At the end of DC stage, the data becomes available and the load-align shifts the data to a word or double-word boundary. In addition, the data cache tag is checked against the page frame number obtained from the joint TLB.
- WB : For register-to-register or load instructions, the result is written back to the register file. For store instructions, the data cache is updated with the store data.

The R4200 has a branch delay of one cycle and a load delay of one cycle. The branch delay is observed by noting that the branch compare logic operates during the EX pipestage, producing the target address which is available for the IC pipestage of the second subsequent instruction. The first subsequent instruction is the instruction in the delay slot and will be allowed to complete whether the branch is taken of not, as illustrated in Figure 8. Similarly,

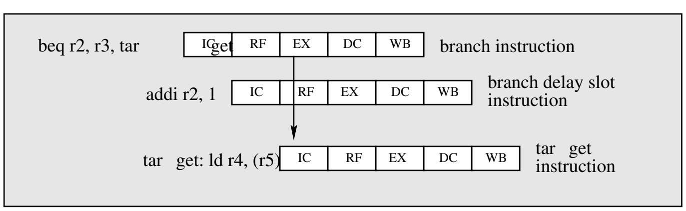


Figure 8: Branch Delay Slot

the load delay of one is evident when the completion of a load at the end of the DC pipestage, produces an operand which is available for the EX pipestage of the second subsequent instruction. The load delay is illustrated in Figure 9.. Hardware detects whether the instruction in the branch delay slot is dependent on register to be loaded,

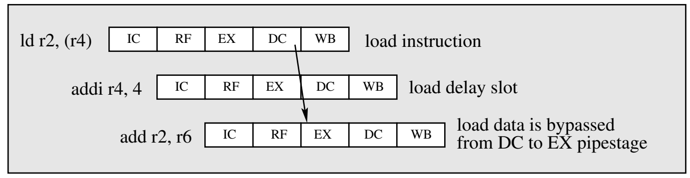
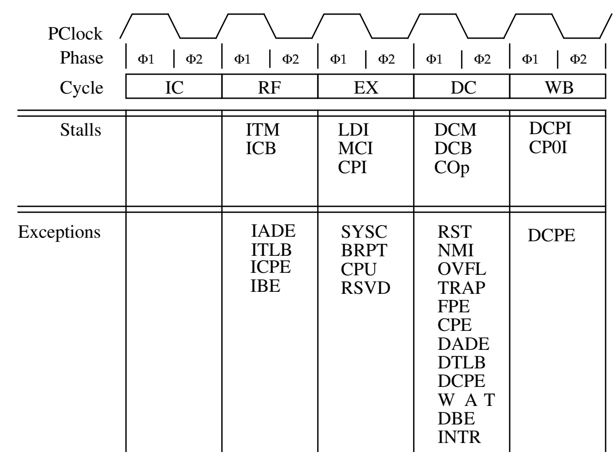


Figure 9: Load Delay Slot

and interlocks accordingly.

The pipeline flow is interrupted when an interlock condition is detected or when an exception occurs.An interlock condition is resolved by stalling the whole pipeline, on the other hand an exception aborts the relevant instruction and all those that follow. Figure 10 illustrates the various interlock conditions and the different types of exceptions, at their predefined pipestage.



Pipe Interlock Conditions			
Descriptions		Descriptions	
ITM	Instruction TLB Miss	ICB	Instruction Cache Busy
LDI	Load Interlock	MCI	Multi-Cycle Interlock
CPI	Coprocessor-2 Interlock	DCB	Data Cache Busy
COp	Cache Op	DCM	Data Cache Miss
DCPI	Data Cache Parity Interlock	CP0I	CP0 Bypass Interlock

Pipe Exception Conditions			
Descriptions		Descriptions	
IADE	Instruction Address Error	ITLB	Instruction TLB Miss
ICPE	Instruction Cache Parity Error	IBE	Instruction Bus Error
SYSC	System Call Instruction	BRPT	BreakPoint Instruction
RSVD	Reserved Instruction	CPU	Coprocessor Unusable
RST	External Reset Exception	NMI	Non-Maskable Interrupt
OVFL	Integer Overflow	TRAP	TRAP Exception
FPE	Floating Point Exception	CPE	Coprocessor-2 Exception
DAE	Data Address Error	DTLB	Data TLB Exception
DCPE	Data Cache Parity Error	WAT	Reference Watch Address
DBE	Data Bus Error	INTR	External Interrupt

Figure 10: Pipeline Interlocks and Exceptions

In the case of simultaneous stall requests from different pipestages, the pipeline control unit prioritizes which stall request is allowed to be resolved, in case of possible resource conflict. A stall request from the DC pipestage has higher priority than a stall request from the RF pipestage.

EXECUTION UNIT

The execution unit is designed to reduce power consumption and simplify the hardware requirements while providing a high level of performance by maximizing usage of each functional element. Integer performance is optimized for the R4200's target applications.

The execution unit is tightly coupled to the on-chip cache memory system, instruction and data caches, and the on-chip memory management unit, CP0. This unit has a multifunction pipe and is responsible for the execution of:

- Integer arithmetic and logic instructions
- Floating-point Coprocessor CP1 instructions
- Branch/Jump instructions
- Load/Store instructions
- Exception instructions
- Special instructions

All floating-point instructions, as defined in the MIPS ISA for the floating-point coprocessor CP1, are processed by the same hardware as used for the integer instructions. However, the execution of floating-point instructions can still be disabled via the coprocessor unusable CU bit defined in the CP0 Status Register.

The execution unit uses a modular design approach to further reduce dynamic power consumption. Control logic is partitioned into small independent blocks responsible for a set of instructions. When relevant instructions are not in the instruction stream, the corresponding control blocks become inactive. Also, when functional elements in the datapath are idle, they operate on a constant (0's or 1's) selected to minimize power dissipation.

The execution unit's datapath consists of: a 64-bit integer/mantissa datapath, an operand bypass network, 32 64-bit integer registers, 32 64-bit floating-point registers, a 12-bit exponent datapath, and a 64-bit instruction virtual address generator.

The integer/mantissa datapath is 64 bits wide and is compatible with both 32-bit and 64-bit operands for integer and floating-point numbers. It has a Boolean Logic functional unit, a carry-propagate adder and a bi-directional shifter, as illustrated in Figure 11.

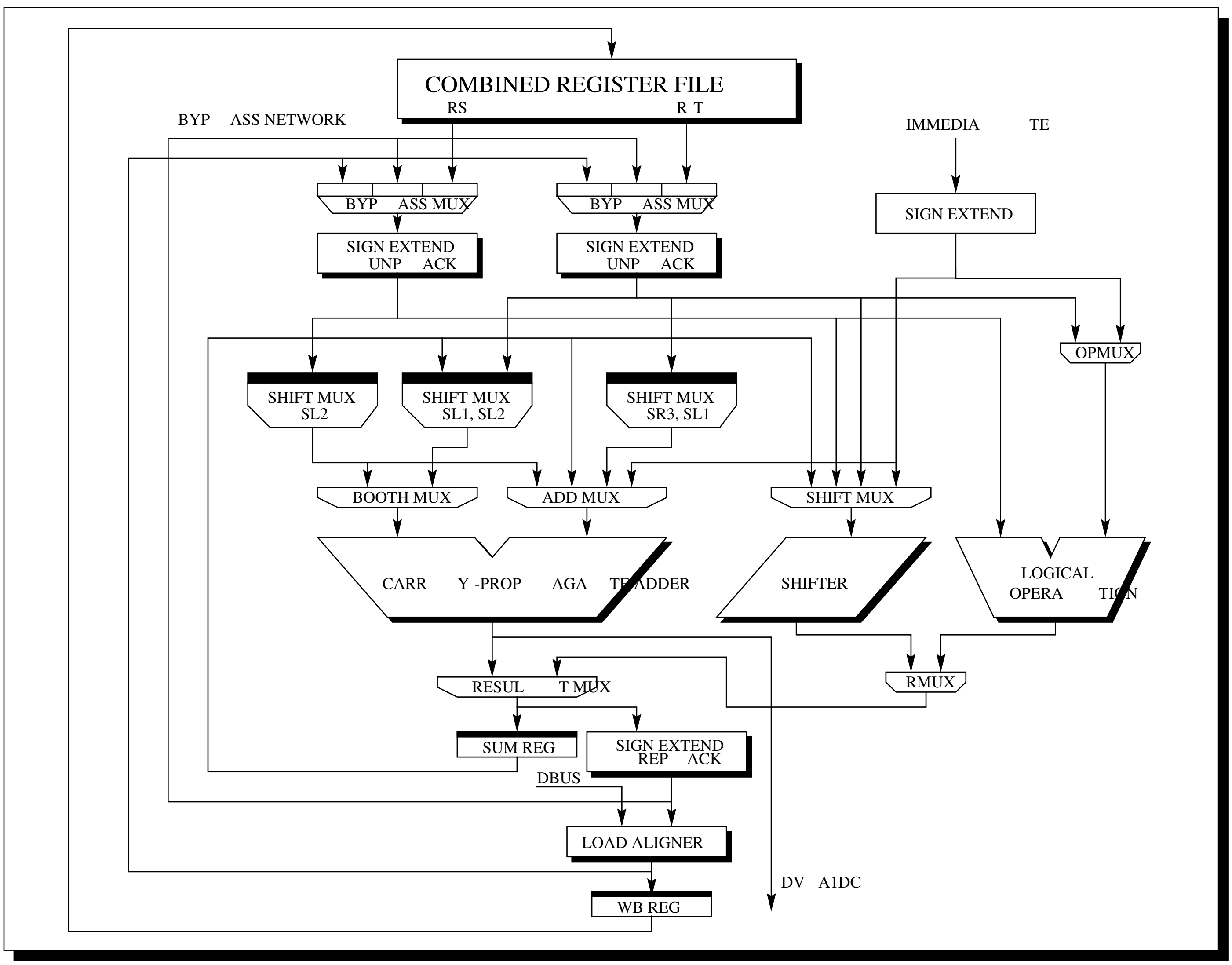


Figure 11: Integer/Mantissa Datapath

The Logical Operation unit performs all integer logical operations. The carry-propagated adder is used for all other integer and floating-point computational instructions. The adder is also used to compute data virtual address for load and store instructions, and to compare two operands in trap instructions. The shifter is responsible for integer variable shifts, store align shifts, and floating-point post-normalization. It also has build-in guard/round/sticky collection logic for floating-point pre-alignment shift. In addition, the datapath has a Leading Zero Counter for floating-point normalization shift calculation, and a floating-point unpacker and repacker.

For load and store class instructions, the datapath can handle partial-words in either big- or little-endian mode. For store instructions, the main bi-directional shifter performs an alignment shift on the register read data. No concatenation of register read data with the original memory data is necessary since the data cache has byte write enable controls. For load instructions, it is necessary to maintain a load delay of one pclock cycle. Due to the timing requirements imposed by this load delay, a dedicated byte-wide shifter (Load Aligner) is needed to shift the memory read data in bytes, halfwords, and words in the right or left direction.

The operand bypass network is built into the datapath to allow feedback of results from the EX and DC pipestage to the instructions in the following EX pipestage waiting to use the results as source operands rs and/or rt. This allows the following instruction to proceed without having to wait for the results to be written back to the register file. Similarly, to maintain the minimum branch delay slot of one pipeline clock cycle for all branch instructions on the floating-point condition, the results from the preceding floating-point compare instruction in the EX, DC, or WB pipestage will be fed back for branch condition testing in the RF pipestage.

The exponent datapath is 12 bits wide. The twelfth bit (MSB) is used as both sign bit and overflow bit. The exponent datapath consists of a feedback mux and 2 operand muxes to select the inputs from the adder, constant generating logic, a carry select adder, random logic to perform exception detection, and a register to hold the selected result from the adder, as shown in Figure 12.

The inputs to the exponent unit come from the unpack logic, where the exponents are extracted from single- or double-precision floating-point operands. The carry-selected adder performs exponent subtraction, pre-alignment shift calculation, and exponent addition for post-normalization final update. The result is sent to the repack logic to be merged with the mantissa.

The result of the exponent logic is compared with constants or ranges to check for various conditions by the result checker. These conditions include: underflow, overflow in single-precision number, overflow in double-precision number, one, zero, and convert limit check. The checks are performed as soon as data is available from the carry-select adder.

The instruction virtual address unit is responsible for the generation of 64-bit instruction virtual addresses to be used by the micro-TLB, I-Cache and CP0. It has its own incrementor to calculate the next sequential address. It also has an equality comparator and a separate ripple-carry adder to generate the branch target address.

In addition, the address unit has exception vector generator logic to decode the type of exception and then present the appropriate vector as the next PC address. It also has the exception PC register pipe chain to maintain a history of PC addresses for each pipestage so that the PC address associated with the exception causing instruction can be loaded into the Exception Program Counter (EPC) register.

CACHE ORGANIZATION

To achieve high performance, increase memory access bandwidth and reduce the latency of load and store instructions, the R4200 processor incorporates on-chip instruction and data caches. Each cache has its own 64-bit datapath and can be accessed in parallel with the other

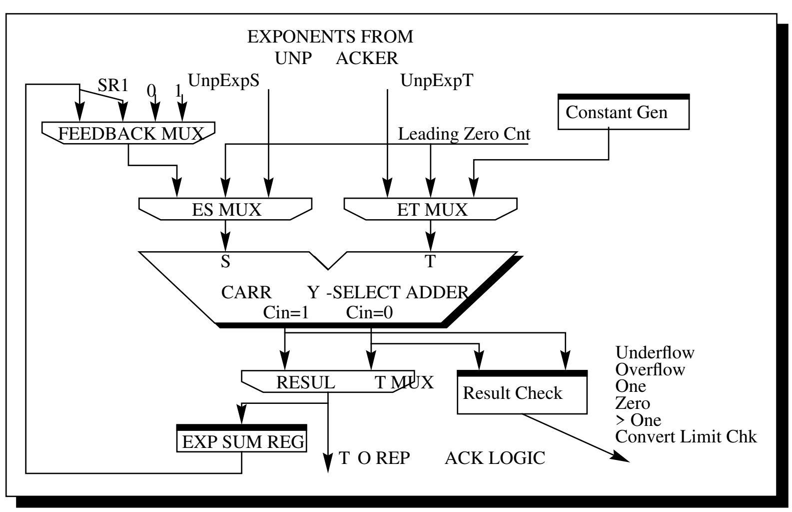


Figure 12: Exponent Datapath

cache. Both the instruction and data caches are direct mapped, virtually indexed, and use physical tags. There is no mechanism for external manipulation of the caches and no on-chip support for second level cache. The R4200 cache organization is shown in Figure 13.

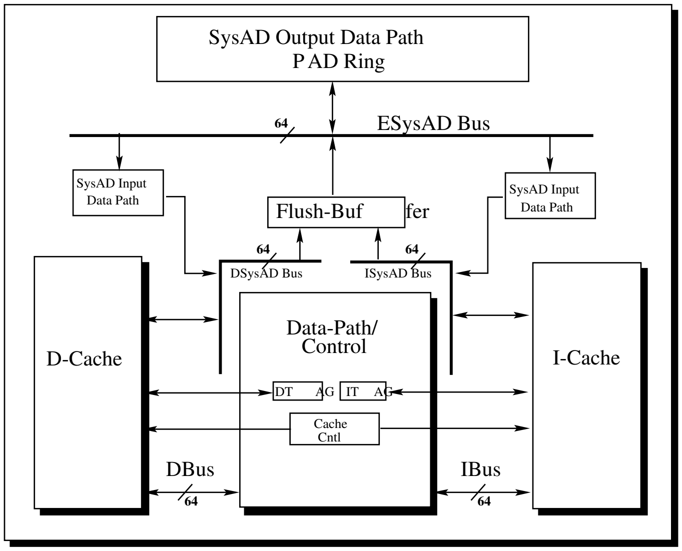


Figure 13: R4200 Cache Organization

The instruction cache is 16 kilobytes in size. It is organized as eight-word (32-byte) lines with a 23-bit tag entry associated with each line. The tag entry consists of one bit tag parity (P), a valid bit (V), and a 21-bit physical tag (bit 32:12 of the physical address). The format of an instruction cache line is shown in Figure 14.

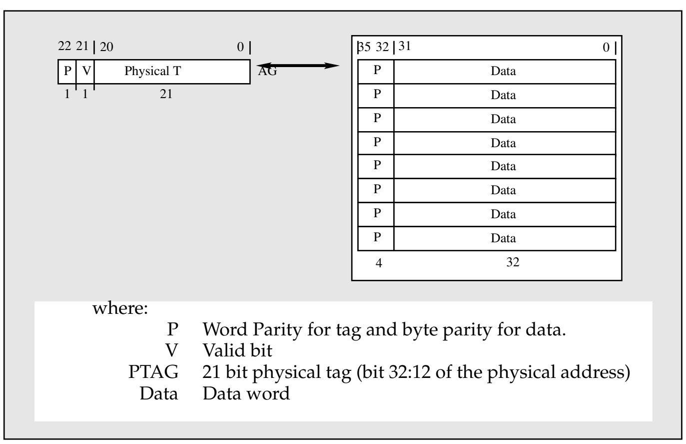


Figure 14: R4200 Instruction Cache Line Format

An instruction cache line has two possible cache states: Invalid and Valid. A cache line with invalid cache state does not contain valid information. A cache line in a valid state contains valid information.

The instruction cache is accessible in one p-cycle. Access begins on phase 2 of the IC pipestage and completes at the end of phase 1 of the RF pipestage. Each access will fetch two instructions. Therefore instruction fetching is required only on every other run cycle or when there is a jump/branch instruction in the EX pipestage. When there is a miss detected during an instruction cache access, a memory block read will be initiated from the system interface to replace the current cache line with the desired line.

The data cache is 8 kilobytes in size. It is organized as four-word (16-byte) lines with a 24-bit tag entry associated with each line.The tag entry consists of one bit tag parity (P), a valid bit (V), a dirty bit (D), and a 21 bit physical tag (bit 32:12 of the physical address). The format of a data cache line is shown in Figure 15.

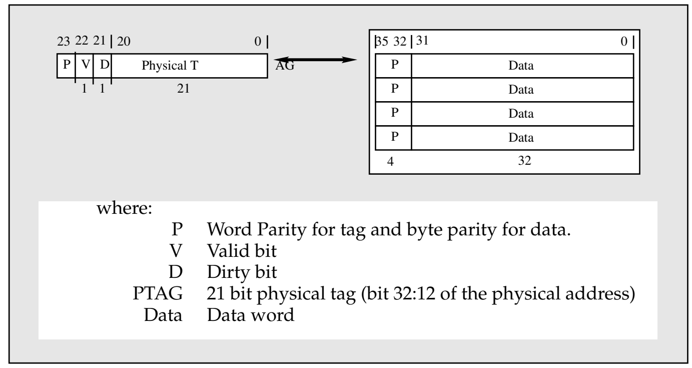


Figure 15: R4200 Data Cache Line Format

A data cache line has three possible cache states: invalid, valid clean, and valid dirty. A data cache line with an invalid state does not contain valid information. A cache line in valid clean state contains valid information and is consistent with main memory. A cache line in valid dirty state contains valid data but is not consistent with main memory. Figure 16 illustrates the data cache state transition sequence.

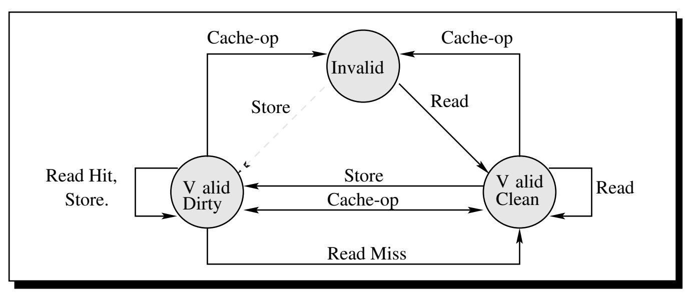


Figure 16: R4200 Data Cache State Transition

The data cache uses write-back write policy. This means that store data is written to the cache line rather than main memory. The modified cache line will be written back to main memory only when it needs to be replaced. For load or store misses, a cache block read will be issued to main memory to bring in a new line and the missed line will be handled in the following manner:

Data load miss:

- If the missed line is not dirty, it will be replaced with the new line.
- If the missed line is dirty, the missed line will be moved to the flush-buffer, the new line will replaced the missed line, and the data in the flush-buffer will be written back to main memory.

Data store miss:

- If the missed line is not dirty, it will be replaced with the new line.
- If the missed line is dirty, the missed line will be moved to the flush-buffer, the new line will be merged with the store data and written to the cache, and the data in the flush-buffer will be written back to main memory.

The data cache is accessible on reads in one p-cycle. The access begins on phase 1 of the DC pipestage and completes at the end of phase 2 of the DC pipestage. Each access will fetch a double word. The data cache writes, however, execute in two p-cycles. A cache read is initiated in the first p-cycle, and a cache write with dirty bit set is initiated in the second p-cycle.

The data cache can be accessed for byte, half-word, three-byte, word, five-byte, six-byte, seven-byte, and double-word. The data size of a partial load is derived from the access type from the integer control unit and the lower three address bits. The data alignment is performed by the datapath load aligner.

To reduce the cache miss penalty, the address of the block read request will point to location of the desired double-word. Since the data cache has a two double-word line size, the system interface will return the critical double-word first, followed by the remaining double-word. The return data will be written to the cache as it is put on the data bus to be used by the execution unit.

The instruction and data caches' data RAMs are byte parity protected and the tag RAMs are word parity protected. For the data cache tag RAM, the dirty bit is not included in the parity checking or generation. Parity is checked against its respective cache data for errors on every cache read access.

During a cache refill, a cache error will also be reported if the system interface provides data with bad parity and parity check enabled, or an erroneous data indicator is set in SysCmd bus bit 5.

A cache error exception is taken if a parity error is detected during an instruction cache access, a data cache access, or memory read access. When this occurs, the CacheErr register will be updated with information pointing to the source of the error.

The R4200 processor provides a variety of cache operations for use in maintaining the state and contents of the instruction and data caches. During the execution of the cache operation instructions, the processor may issue processor block read or write request to fill or write-back a cache line. Table 12 lists all cache operations supported by the R4200 processor.

FLUSH BUFFER

The R4200 processor contains an on-chip flush buffer. This buffer is used as a temporary data storage for outgoing data. It is capable of taking one double-word per pcycle. The flush buffer is four words wide, equal to data cache line size. It is capable of taking one memory reference at a time. The format of the flush buffer is shown in Figure 17.

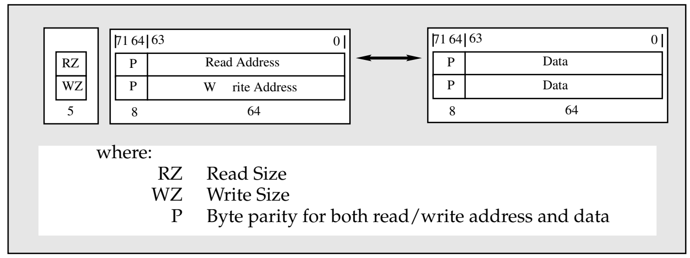


Figure 17: Flush Buffer Format

The read/write size indicator specifies the size of data to be transferred out. The read/write address field provide a 33 bits physical address, zero-extended to 64 bits. The data field holds the four words of data. Both the address and data fields are byte parity protected.

For uncached stores, the store data will be held in the buffer until it is taken by the external interface. For a load or store miss to a cache line in dirty state, the dirty line will be moved to the flush buffer while the processor is waiting for the data for line refill. After receiving the data, the flush buffer transfers write data to the external interface.

If the flush buffer is full and the processor attempts a load or a store which requires external resources, the processor will stall until this buffer is empty.

COPROCESSOR-0

The R4200 Coprocessor-0 functional unit performs two major functions. The first is memory management for the processor, and the second is exception processing. CP0 is logically divided into four subunits: CP0 Register Files (Cp0reg), Translation Look-aside Buffer (TLB), Instruction TLB (ITLB), and CP0 Control Unit (Cp0ctl). Figure 18 shows how these subunits interact with each other and other functional units.

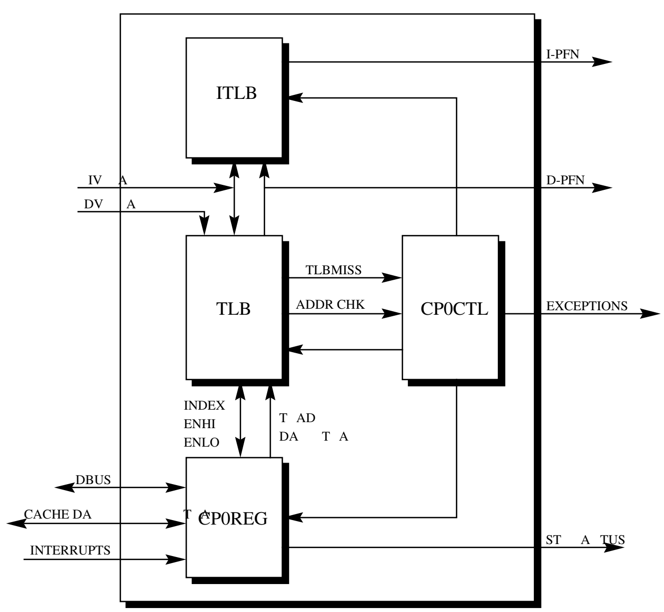


Figure 18: R4200 Coprocessor-0 Block Diagram

The CP0 registers consist of two functionally distinct sets. The first set comprises those that support the TLB operations, and the second set is formed from those registers that reflect the state of the processor. All of the CP0 registers are readable by software. All but the Random, BadVAddr, PrId, CacheErr and MskId registers are writable by software. The definitions of the CP0 registers are described in a later section.

The CP0ctl subunit controls data transfer to and from the TLB, ITLB and CP0 registers. It takes the instruction decode signals from the integer unit and interprets them to determine whether a CP0 register is to be read or written. It generates control signals for reading and writing TLB and ITLB entries.

In addition, the CP0ctl handles exception processing. Its function is to communicate with the pipeline control unit for interrupt signalling, and notification of TLB exceptions. It also performs address checking with regard to the type of instruction being executed, and various state bits in the Status register. For example, a User mode process may try to access a kernel data address. In this case, a Data Address Error exception would be detected, since the address Region bits would conflict with the KSU bits in the Status register.

The Translation Look-aside Buffers (micro-TLB and joint TLB) are responsible for the translation of both instruction and data virtual addresses into physical addresses. The physical address is needed by the cache for its tag checks, and by the system interface for off-chip memory access, and various CP0 registers. The jTLB translates the lower 40 bits of the 64-bit virtual address size defined in the MIPS-III architecture, and provides a 33-bit physical address.

The joint TLB contains 32 entries, each of which is simultaneously checked for a match with the extended virtual address. Each TLB entry maps an even-odd pair of pages. The page size is 4K, 16K, 64K, 256K, 1M, 4M or 16M bytes, which is specified on a per-entry basis by the MASK bit-mask field of the entry. The valid values of the MASK field and the effect on the translation is documented in the description of the PageMask Register.

A virtual address matches a TLB entry when the virtual page number (VPN) field of the virtual address equals the VPN field of the TLB entry, and either the Global (G) bit of the TLB entry is set or the address space identifier (ASID) field of the virtual address (as held in the EntryHi register) matches the ASID field of the TLB entry. Although the valid (V) bit of the entry must be set for a valid translation to take place, it is not involved in the determination of a matching TLB entry.

The operation of the TLB is not defined if more than one entry in the TLB matches. If one TLB entry matches, the physical address and access control bits (N, D, and V) are retrieved; otherwise a TLB refill exception occurs. If the access control bits (D and V) indicate that the access is not valid, a TLB modification or TLB invalid exception occurs.

The format of a TLB entry in 32-bit addressing mode is shown in Figure 19. The format of a TLB entry in 64-bit mode is shown in Figure 20.

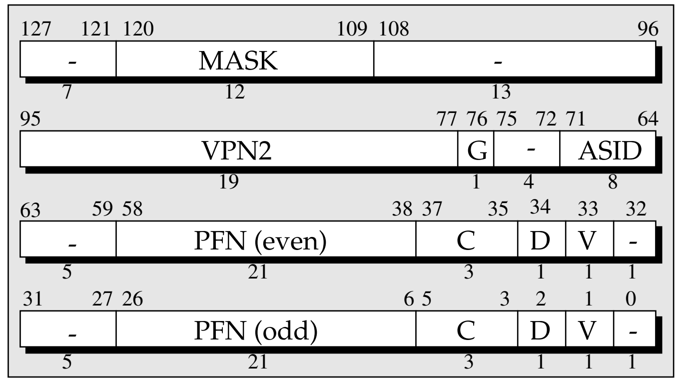


Figure 19: TLB Entry Format in 32-bit Addressing Mode

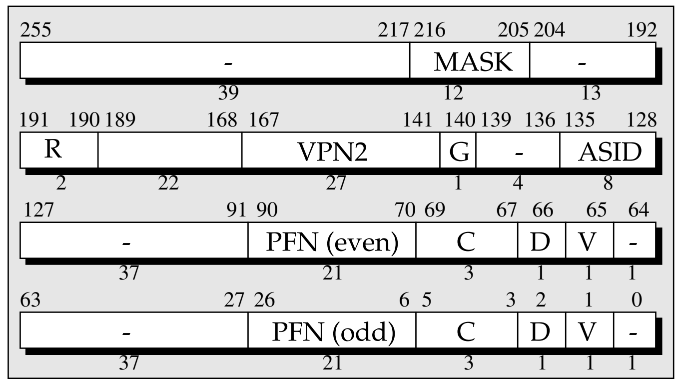


Figure 20: TLB Entry Format in 64-bit Addressing Mode

Where:

R	Is the Region used to match $VAddr_{63..62}$
MASK	is the comparison MASK
VPN2	is the Virtual Page Number / 2
G	Global bit
ASID	Address Space Identifier
PFN	Physical Page Number (upper 21 bits of the physical address)
C	Cache Algorithm (011=cached; 010=uncached)
D	If set, page is writable
V	If set, entry is valid

The R4200 also implements a two entry micro-TLB that is dedicated for instruction address translation. This allows instruction address translation to be done simultaneously with data address translation. If there is a miss in the micro-TLB, the pipeline will stall while the new TLB entry is transferred from the joint TLB to the micro-TLB. The two entry micro-TLB entry is fully associative with a Least Recently Used replacement algorithm. Each micro-TLB entry maps to a 4K byte page size only.

CP0 REGISTERS

The following table lists all CP0 registers used in the R4200. Attempting to write any unused register is undefined and may have an unpredictable effect on the R4200 processor. Attempting to read any unused register is undefined and may result in unpredictable data.

Table 1: System Control Coprocessor-0 Registers

Num	Mnemonic	Descriptions
0	Index	Programmable pointer into TLB entry
1	Random	Random pointer into TLB array
2	EntryLo0	Low half of the TLB entry for even VPN
3	EntryHi	Low half of the TLB entry for odd VPN

Table 1: System Control Coprocessor-0 Registers

4	Context	Pointer to kernel PTE table
5	PageMask	TLB page mask
6	Wired	Number of wire TLB entries
7	----	Unused
8	BadVAddr	Bad virtual address
9	Count	Timer count
10	EntryHi	High half of TLB entry
1 1	Compare	Timer compare
12	SR	Status register
13	Cause	Cause register
14	EPC	Exception program counter
15	PRid	Processor revision identifier
16	Config	Configuration register
17	LLAddr	Load linked address
18	WatchLo	Memory reference trap address lower bits
19	WatchHi	Memory reference trap address upper bits
20	XContext	Context register for MIPS-III addressing
21-25	----	Unused
26	PErr	Parity error in cache
27	CacheErr	Cache error and status register
28	TagLo	Cache tag register
29	TagHi	Cache tag register (Reserved)
30	ErrorEPC	Error exception program counter
31	----	Unused

The Index register is a read/write register in which 6 bits specify an entry into the on-chip TLB. The high order bit indicates the success or failure of a TLBP operation. The TLB Index register is used to specify the entry in the TLB affected by the TLBR and TLBWI instructions. Figure 21 shows the Index register format.

31	30 ... 6	5 ... 0
P	0	Index
1	25	6

where:

P	Result of last Probe operation. Set to 1 if last TLB Probe instruction was unsuccessful.
index	Index to entry in TLB
0	Must be all zeroes on reads and writes

Figure 21: CP0 Index Register

The Random register is a read-only register in which 6 bits specify an entry in the on-chip TLB. This register will decrement on every instruction executed. The values range between a low value determined by the TLB Wired register, and an upper bound of TLBENTRIES-1. The TLB Random register is used to specify the entry in the TLB affected by the TLBWR instruction. Upon system reset, or when the Wired register is written, this register will be set to the upper limit. Figure 22 shows the Random register format.

31 ... 6	5 ... 0
0	Random
26	6

where:

Random	Index to entry in TLB
0	Must be all zeroes on reads and writes

Figure 22: CP0 Random Register

The EntryLo0 and EntryLo1 registers are a read/write register that are used to access on-chip TLB. EntryLo0 is for the even virtual pages while EntryLo1 is for the odd virtual pages. They contain the Page Frame Number, along with several configuration bits for the TLB entry. They are used by the TLBR, TLBWI, and TLBWR instructions. Figure 23 shows the EntryLo0 and EntryLo1 registers format.

32 bit Mode:

31 30	29 ... 6	5 ... 3	2	1	0
0	PFN	C	D	V	G
2	24	3	1	1	1

64 bit Mode:

31 30	29 ... 6	5 ... 3	2	1	0
0	PFN	C	D	V	G
2	24	3	1	1	1

where:

PFN	Physical Frame Number
C	Cache Algorithm. If C=011, then page is cached. If C=010, then it is uncached. All other values of C are cached.
V	Page Valid if 1, invalid if 0
G	If set in both Lo0 and Lo1, then ignore ASID
0	Must be zero on all reads and writes
D	Page Dirty if 1, Clean if 0

Figure 23: CP0 EntryLo0 and EntryLo1 Registers

The Context register is a read/write register containing a pointer into a kernel Page Table Entry (PTE) array. It is designed for use in the TLB refill handler. The BadVPN2 field is not writable. It contains the VPN of the most recently translated virtual address that did not have a valid translation. It contains bits 31.13 of the virtual address that caused the TLB miss. Bit 12 is excluded because a single TLB entry maps an even-odd page pair. This format can be used directly as an address for pages of size 4K bytes. For pages of 16M bytes, this value must be shifted and masked. The PTEBase field is writable as well as readable, and indicates the base address of the PTE table of the current user address space. 24 shows the Context register format..

32 bit Mode:

31 ... 23	22 ... 4	3 ... 0
PTEBase	BadVPN2	0
9	19	4

64 bit Mode:

63 ... 23	22 ... 4	3 ... 0
PTEBase	BadVPN2	0
41	19	4

where:

PTEBase	Base address of the Page Entry Table
BadVPN2	Virtual Page Number of the failed virtual address divided by two
0	Must be zero on reads and writes

Figure 24: CP0 Context Register

The PageMask register is a read/write register that is used when reading or writing an on-chip TLB. The TLBR, TLBWI, and TLBWR instructions use this register as a source or destination.When virtual addresses are presented for translation, the corresponding bits in the TLB specify which of virtual address bits 24.13 participate in the comparison. This implements a variable page size on a per entry basis. R4200 implements 4K, 16K, 64K, 256K, 1M, 4M and 16M pages. Figure 25 shows the PageMask register format..

31 ... 25	24 ... 13	12 ... 0
0	MASK	0
7	12	13

where:

MASK	Mask for Virtual Page Number. For R4200, this is 000000000000 for 4K pages, 000000000011 for 16K pages and so on up to 111111111111 for 16M pages.
0	Must be zero on reads and writes

Figure 25: CP0 Mask Register

The TLB Wired register is a read/write register that specifies the boundary between the wired and random entries of the TLB. This register is set to 0 upon reset. Writing to this register also sets the Random register to 31. Writing a value greater than TLBENTRIES-1 to this register will result in undefined behavior. Figure 26 shows the Wired register format.

31 ... 6	5 ... 0
0	W ired
26	6

where:

Wired	TLB wired boundary
0	Must be zero on reads and writes

Figure 26: CP0 Wired Register

The Bad Virtual Address register is a read-only register that displays the most recently translated virtual address which failed to have a valid translation or which had an addressing error. Figure 27 shows the BadVAddr register format.

32 bit Mode:

31 ... 0
BadV Addr
32

64 bit Mode:

63 ... 0
BadV Addr
64

where:

BadVAddr	Most recently translated virtual address that failed to have a valid translation or that had an addressing error.

Figure 27: CP0 BadVAddr Register

The Count register is a read/write register used to implement timer services. It increments at a constant rate based on the clock cycle. On R4200, it will increment at one-half the PClock speed. When the Count register has reached all ones, it will roll over to all zeroes and continue counting. This register is both readable and writable. It is writable for diagnostic purposes. Figure 28 shows the Count register format..

The EntryHi Register is a read/write register that is used to access on-chip TLB. It is used by the TLBR, TLBWI, and TLBWR instructions. EntryHi contains the Address Space Identifier (ASID) and the Virtual Page Number. Figure 29 shows the EntryHi register format.

The Compare register is a read/write register. When

31 ... 0
Count
32

where:

Count	Constantly updating count value to be compared to the Compare register.

Figure 28: CP0 Count Register

32 bit Mode:

31 ... 13	12 ... 8	7 ... 0
VPN2	0	ASID
19	5	8

64 bit Mode:

63 62	61 ... 40	39 ... 13	12 ... 8	7 ... 0
R	FILL	VPN2	0	ASID
2	22	27	5	8

where:

R	Region (00=user, 01=supervisor,11=kernal) used to match $vAddr_{63..62}$
FILL	Reserved; undefined on read, should be 0 or -1 on write (should sign extend the virtual page number)
VPN2	Virtual Page Number divided by 2
0	Must be zero on reads and writes
ASID	Address Space Identifier

Figure 29: CP0 EntryHi Register

the value of the Count Register equals the value of the Compare register, IP_7 of the Cause register is set. This causes an interrupt on the next execution cycle in which interrupt is enabled. Writing to the Compare register will clear the timer interrupt. Figure 30 shows the Compare register format.

31 ... 0
Compare
32

where:

Compare	Value to be compared to Count register. An interrupt will be signaled when they are equal.

Figure 30: CP0 Compare Registers

The Status register is a read/write register that contains the various mode, enables, and status bits used on R4200. The contents of this register are undefined after a reset, except for TS, which is zero; ERL and BEV, which are one; and RP, which is zero. The SR bit is 0 after a Cold Reset, and 1 after NMI or (Soft) Reset. Figure 31 shows the Status register format.

The Cause register is a read/write register that describes the nature of the last exception. A five 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 Branch Delay bit indicates whether the EPC has been adjusted to point at the branch instruction which precedes the next restartable instruction. The Coprocessor Error field indicates the unit number referenced by an instruction causing a "Coprocessor Unusable" exception. The Interrupt Pending field indicates which interrupts are pending. This field indicates the current status and changes in response to external signals. IP7 is the timer interrupt

31 28	27	26	25	24	23	22	21	20	19	18	17	16
CU	RP	FR	RE	ITS	rsvd	BEV	TS	SR	0	CH	CE	DE
4	1	1	1	1	1	1	1	1	1	1	1	1

15 8	7	6	5	4 3	2	1	0
IM	KX	SX	UX	KSU	ERL	EXL	IE
8	1	1	1	2	1	1	1

where:

CU	Coprocessor Unit usable if 1. CP0 is always usable by the kernel
RP	Reduced Power. Changes clock to quarter speed.
FR	If set, enables MIPS III additional floating point registers.
RE	Reverse Endian. Changes endianess in user mode.
ITS	Instruction Trace Support. Enables trace support.
rsvd	Reserved for future use.
BEV	Controls location of TLB refill and general exception vectors. (0=normal, 1= bootstrap). Also affects Cache Error exceptions.
TS	TLB Shutdown has occurred. This happens to avoid destructive drive fights in the TLB if multiple hits occur. Once TLB shutdown has occurred, the processor must be reset to restore TLB functionality.
SR	Soft Reset or NMI has occurred.
0	Must be zeroes on read and write.
CH	This is the CP0 Condition bit. It is readable and writable by software only. It is not set or cleared by hardware.
CE	If set, contents of PErr register are used to set parity bits in cache. The tag parity bit will be written from the TagLo parity bit.
DE	if set, cache parity errors will not cause exceptions
IM	Interrupt Mask. Enables and disables interrupts.
KX	Kernel extended addressing enabled.
SX	Supervisor extended addressing enabled.
UX	User extended addressing enabled.
KSU	Mode (10=user, 01=supervisor, 00=kernel)
ERL	Error Level. Normal when zero, error if one
EXL	Exception Level. Normal when zero, exception if one
IE	Interrupt Enable.

Figure 31: CP0 Status Register

bit, set when the Count register equals the Compare register. IP6.2 are the external interrupts, set when the external interrupts are signalled. An external interrupt is set at one of the external interrupt pins or via a write request on the SysAD bus. IP1.0 are software interrupts, and may be written to set or clear software interrupts. Figure 32 shows the Cause register format.

The EPC register is a read/write register that 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 cause of the exception, or when that instruction is in a branch delay slot, the EPC contains the virtual address of the immediately preceding branch or jump instruction and the Branch Delay bit in the Cause register is set. If the exception is caused by recoverable, temporary conditions (such as a TLB miss), the EPC contains the virtual address of the instruction which caused the exception. Thus, after correcting the conditions, the EPC contains a point at which execution can be legitimately resumed. Figure 33 shows the format of EPC regsiter.

The PRId register is a read-only register that contains information that identifies the implementation and revision level of the processor and associated system control coprocessor. The revision number can distinguish some chip revisions. However, MIPS is free to change this register at any time and does not guarantee that changes to

31	30	29 28	27 16	15 8	7	6 2	1 0
BD	0	CE	0	IP	0	ExcCode	0
1	1	2	12	8	1	5	2

where:

BD	Branch Delay
CE	Coprocessor Error
IP	Interrupt pending
ExcCode	Exception Code:

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	IBE	Bus Error Exception (instruction fetch)
7	DBE	Bus Error Exception (data reference: load or store)
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	---	Reserved
15	FPE	Floating Point Exception
16-22		Reserved
23	Watch	Reference to WatchHi/WatchLo address
24-31		Reserved

0 Must be zeroes on read and write

Figure 32: CP0 Cause Register

32 bit Mode:

31 0
EPC
32

64 bit Mode:

63 0
EPC
64

where:

EPC Exception Program Counter.

Figure 33: CP0 EPC Register

its chips will necessarily change the revision number, or that changes to the revision number necessarily reflect real chip changes. For this reason, software should not rely on the revision number to characterize the chip. Figure 34 shows the Revision Identifier register format.

31 16	15 8	7 0
0	Imp	Rev
16	8	8

where:

Imp	Implementation identifier: 0x0A for R4200
Rev	Revision Number
0	Returns zeroes on reads.

Figure 34: CP0 Revision Identifier Register

The Config register specifies various configuration options that are available for R4200. It is compatible with the R4000 Config register, but only a small subset of the options available on the R4000 are possible on R4200. For that reason, there are many fields which are set to constant values. The EP and BE fields are written by hardware during Reset. Software only has read access to these fields. The CU and K0 fields are readable and writable by software. There is no other mechanism for writing to these fields, and their values are undefined after Reset. Figure 35 shows the Configuration register format.

31 28 27 24 23 2019 18 1716 15 14 13 12 1 1 98 6 5 4 3 2 0

0	EP	0	00	1	0	BE	1	1	0	010	001	1	0	CU	K0
4	4	4	2	1	1	1	1	1	1	3	3	1	1	1	3

where: 14

EP Pattern for writeback data on SYSAD port
0001 -> DDx
0110 -> DxxDxx
All other values of EP are undefined.
BE BigEndianMem
0 -> Memory and kernel are Little Endian
1 -> Memory and kernel are Big Endian
CU Reserved. (Read- and Writ-able by software)
K0 Kseg0 coherency algorithm. This has the same format as the C field in EntryLo0 and EntryLo1. The only defined values for K0 for R4200 are 010 (noncacheable) and 011 (cacheable).
0 Returns 0 on read.
1 Returns1 on read.

Figure 35: CP0 Configuration Register

The LLAddr register contains the physical address read by the most recent Load Linked instruction. This register exists for diagnostic purposes, and serves no function during normal operation. It is both readable and writable by software. Figure 36 shows the LLAddr register format.

31 0

P Addr

where:
Paddr Bits 35..4 of the physical address. Note that for R4200, the MSB of the physical address is only 32. A load linked instruction will write 0 into bits[31..29] of LLAddr, but a software write can set all 32 bits to any value. This maintains compatibility with R4000.

Figure 36: CP0 LLAddr Register

The R4200 processor provides a debugging feature to detect references to a physical address. Loads or stores to the location specified by the WatchHi/WatchLo register pair cause a Watch trap. Figure 37 shows WatchHi/ WatchLo register formats.

The XContext register is a read/write register containing a pointer into a kernel Page Table Entry (PTE) array. It is designed for use in the XTLB refill handler. The R and BadVPN2 fields are not writable. The register contains the VPN of the most recently translated virtual address that did not have a valid translation. It contains bits 39.13 of the virtual address that caused the TLB miss. Bit 12 is excluded because a single TLB entry maps an even-odd page pair. This format can be used directly as an address for pages of size 4K bytes. For pages of 16M bytes, this value must be shifted and masked. The PTEBase field is writable as well as readable, and indicates the base address of the PTE table of the current user address space. Figure 38 shows XContext register format.

W atchLo:

31 3 2 1 0

P Addr	0	R	W
29	1	1	1

W atchHi:

31 1 0

0	P Addr
31	1

where:
PAddr Physical address
WatchLo PAddr contains bits 31.3
WatchHi PAddr contains bit 32.
R Trap on Read Access if 1
W Trap on Write Access if 1
0 Must be zeroes on reads and writes

Figure 37: CP0 WatchLo and WatchHi Registers

63 3332 31 30 4 3 0

PTEBase	R	BadVPN2	0
31	2	27	4

where:
PTEBase Base address of the Page Entry Table
BadVPN2 Virtual Page Number of the failed virtual address divided by two
R Region (00=user, 01=supervisor, 11=kernel)
0 Must be zero on reads and writes

Figure 38: CP0 XContext Register

The PErr register is used to read or write the parity bits of the cache. It can be used for initialization, cache diagnostics, or cache error handling. The register is loaded from the cache array via the Cache operation Index Load Tag.The PErr register is substituted for the computed parity on stores if the CE bit of the Status register is set. PErr is also substituted for the computed parity on the Cache operation Fill if the CE bit is set. Figure 39 shows PErr register format.

31 8 7 0

0	Parity
24	8

where:
Parity Eight bit parity field
0 Must be zeroes on all reads and writes

Figure 39: CP0 PErr Register

CacheErr is a 32-bit read-only register for correcting and logging parity errors in the cache. The register provides the cache index and status bits indicating the source and nature of the error. It is loaded when a Cache Error exception is taken. Figure 40 shows CacheErr register format.

The TagLo register is a 32-bit read/write register used to hold the cache tag and parity during cache initialization, cache diagnostics, or cache error handling. On the cache operation Index Store Tag, the PTagLo and PState bits are written into the cache. If the CE bit is set in the status register, the parity bit will be written into the cache tag from the TagLo register. If the CE bit is cleared, parity will be recomputed for the tag. During the cache operation Fill,

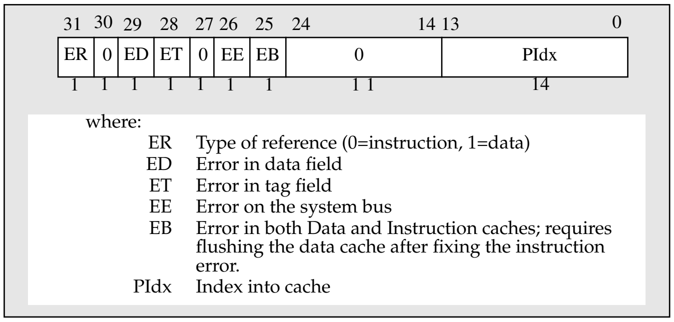


Figure 40: CP0 CacheErr Register

the parity bit from the TagLo will be written into the cache tag if the CE bit is set. The TagHi register is reserved for future use. The Tag registers are written by the CACHE and MTC0 instructions. Figure 41 shows TagLo and TagHi register formats.

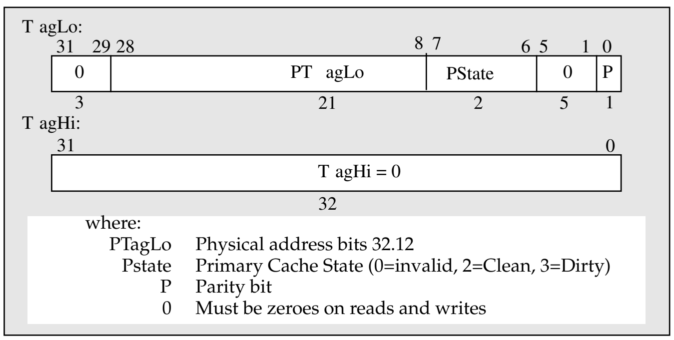


Figure 41: CP0 TagLo TagHi Registers

The ErrorEPC register is similar to the EPC, but is used on parity error exceptions. It is also used to store the PC on Reset and NMI exceptions. It is read/write and contains the virtual address at which instruction processing may resume after servicing an Error or Reset/NMI exceptions. The EPC register contains either the virtual address of the instruction which was the direct cause of the exception, or when that instruction is in a branch delay slot, the EPC contains the virtual address of the immediately preceding branch or jump instruction. There is no branch delay slot indication for ErrorEPC. Figure 42 shows ErrorEPC register format.

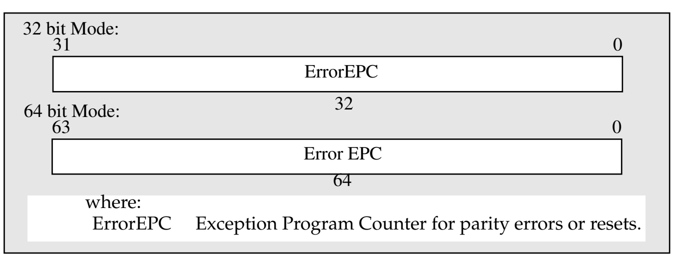


Figure 42: CP0 ErrorEPC Register

MEMORY MANAGEMENT

The R4200 processor provides a full-featured memory management unit (MMU) which uses an on-chip translation look-aside buffer (TLB) to translate virtual address into physical address. The TLB is a fully associative memory that holds 32 entries, which provide mapping to 32 odd/even pairs (64 pages). When address mapping is indicated, each TLB entry is checked simultaneously for a match with the virtual address that is extended with an ASID stored in the EntryHi register. The address is mapped to a page of size of between 4Kbytes and 16Mbytes.

The processor virtual address can be either 32 or 64 bits wide, depending on whether the processor is operating in 32-bit or 64-bit mode.

- In 32-bit mode, addresses are 32 bits wide. The maximum user process size is 2 gigabytes (2^{31}).
- In 64-bit mode, addresses are 64 bits wide. The maximum user process size is 1 terabyte (2^{40}).

Figure 43 shows the translation of a virtual address into a physical address.

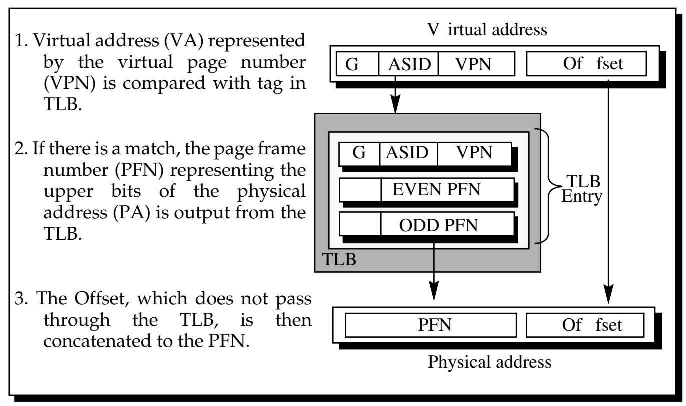


Figure 43: Virtual-to-physical address translation

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

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

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

If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the *Offset*, which represents an address within the page frame space.

Figure 44 shows the virtual-to-physical address translation of a 32-bit mode address.

- The top portion of Figure 44 shows a virtual address with a 12-bit, or 4-Kbyte, page size, labelled *Offset*. The remaining 20 bits of the address represent the VPN, and index the 1M-entry page table.
- The bottom portion of Figure 44 shows a virtual address with a 24-bit, or 16-Mbyte, page size,

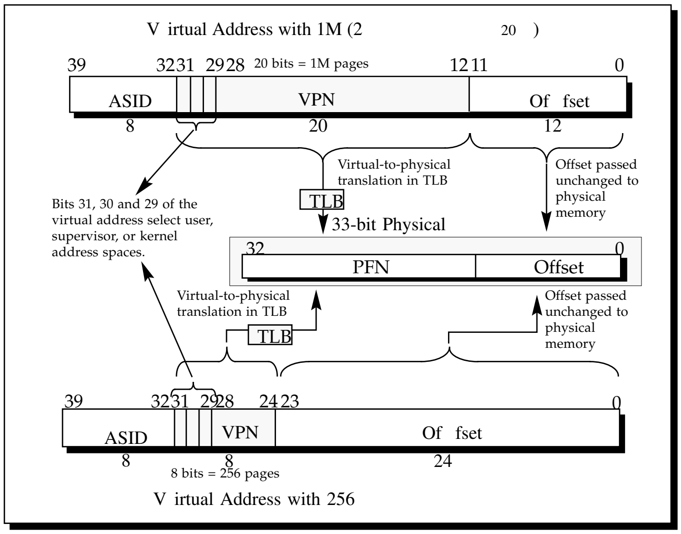


Figure 44: 32-bit-mode virtual address translation

labelled *Offset*. The remaining 8 bits of the address represent the VPN, and index the 256-entry page table.

Figure 45 shows the virtual-to-physical-address translation of a 64-bit mode address. This figure illustrates the two possible page sizes: a 4-Kbyte page (12 bits) and a 16-Mbyte page (24 bits).

- The top portion of Figure 45 shows a virtual address with a 12-bit, or 4-Kbyte, page size, labelled *Offset*. The remaining 28 bits of the address represent the VPN, and index the 256M-entry page table.
- The bottom portion of Figure 45 shows a virtual address with a 24-bit, or 16-Mbyte, page size, labelled *Offset*. The remaining 16 bits of the address represent the VPN, and index the 64K-entry page table.

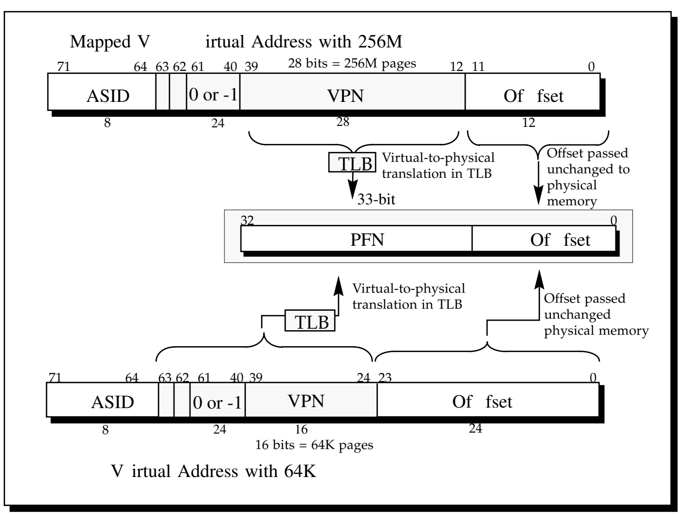


Figure 45: 64-bit-mode virtual address translation

The processor has three operating modes that function in both 32- and 64-bit operations: User mode, Supervisor mode, and Kernel mode. The address space for each mode of operation are described in the following sections.

In User mode, a single, uniform virtual address space—labelled User segment—is available; its size is:

- 2 Gbytes (2^{31} bytes) in 32-bit mode *(useg)*
- 1 Tbyte (2^{40} bytes) in 64-bit mode (x*useg*)

The User segment starts at address 0 and the current active user process resides in either *useg* (in 32-bit mode) or *xuseg* (in 64-bit mode). The TLB identically maps all references to *useg*/*xuseg* from all modes, and controls cache accessibility[1]. Figure 46 shows User mode virtual address space.

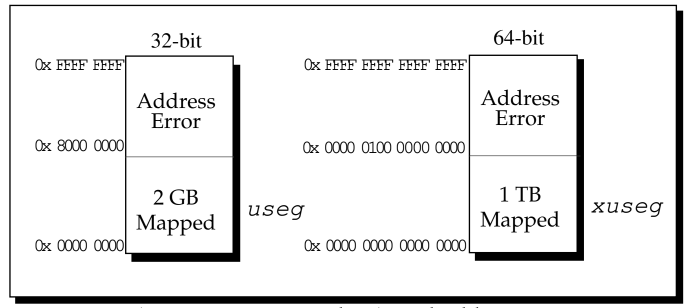


Figure 46: User mode virtual address space

The processor operates in User mode when the *Status* register contains the following bit-values:

- *KSU* bits = 10_2
- *EXL*= 0
- *ERL*= 0

In conjunction with these bits, the *UX* bit in the *Status* register selects between 32- or 64-bit User mode addressing as follows:

- when *UX* = 0, 32-bit *useg* space is selected
- when *UX* = 1, 64-bit *xuseg* space is selected

Supervisor mode is designed for layered operating systems in which a true kernel runs in R4200 Kernel mode, and the rest of the operating system runs in Supervisor mode.Figure 47 shows Supervisor mode address space.

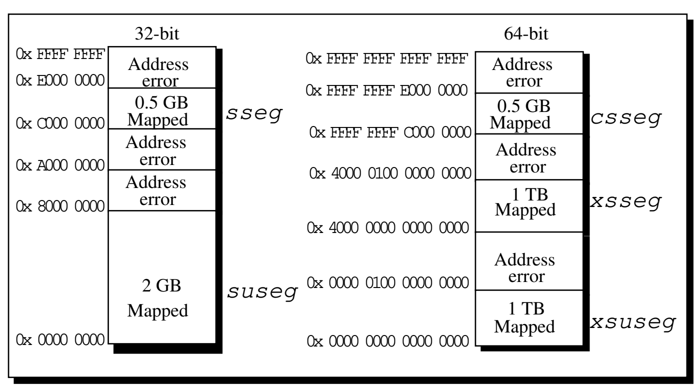


Figure 47: Supervisor mode virtual address space

1. The cached *(C)* field in a TLB entry determines whether the reference is cached; see Figure 20.

The processor operates in Supervisor mode when the *Status* register contains the following bit-values:

- *KSU*= 01_2
- *EXL*= 0
- *ERL*= 0

In conjunction with these bits, the SX bit in the *Status* register selects between 32- or 64-bit Supervisor mode addressing:

- when *SX* = 0, 32-bit supervisor space is selected
- when *SX* = 1, 64-bit supervisor space is selected

The processor operates in Kernel mode when the *Status* register contains one of the following values:

- *KSU*= 00_2
- *EXL*= 1
- *ERL*= 1

In conjunction with these bits, the *KX* bit in the *Status* register selects between 32- or 64-bit Kernel mode addressing:

- when *KX* = 0, 32-bit kernel space is selected
- when *KX* = 1, 64-bit kernel space is selected

The processor enters Kernel mode whenever an exception is detected and it remains in Kernel mode until an Exception Return (ERET) instruction is executed. The ERET instruction restores the processor to the mode existing prior to the exception.

Kernel mode virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 48.

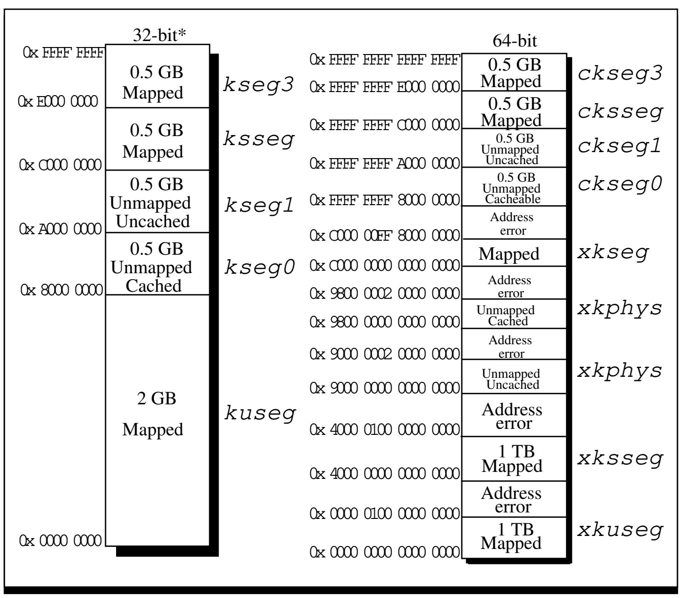


Figure 48: Kernel mode virtual address space

EXCEPTION MANAGEMENT

The R4200 processor receives exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, I/O interrupts, and system calls. When the CPU detects one of these exceptions, the normal sequence of instruction execution is suspended and the processor enters Kernel mode.

The processor then disables interrupts and forces execution of a software exception processor (called a *handler*) located at a fixed address. The handler saves the context of the processor, including the contents of the program counter, the current operating mode (User or Supervisor), and the status of the interrupts (enabled or disabled). This context is saved so it can be restored when the exception has been serviced.

When an exception occurs, the CPU loads the *Exception Program Counter* (*EPC*) register with a location where execution can restart after the exception has been serviced. The restart location in the *EPC* register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, the address of the branch instruction immediately preceding the delay slot.

The R4200 separates exceptions into five vector spaces: Reset and NMI vector, TLB Refill vector, XTLB Refill vector, Cache Error, and General exception vector. The values of each vector (except Reset and NMI) depend on the Boot Exception Vector (BEV) bit of the Status register, which allow two alternate sets of vectors to be used, with one set pointing to the PROM address space and the other set point to cacheable address space.

The Reset and NMI exceptions are always vectored to location 0xBFC0 0000 in 32-bit mode, and location 0xFFFF FFFF BFC0 0000 in 64-bit mode.

The addresses for all other exceptions are a combination of a *vector offset* and a *base address.* The base address is determined by the BEV bit of the Status register, as shows in Table 2. Table 3 shows the exception vectors for the different types of exceptions. The vector base address for Cache Error Exceptions is in uncached space for either value of BEV.

Table 2: Exception Vector Base Addresses

BEV	R4200 Processor Exception V ector Base	
	32-bit mode	64-bit mode
0	0x8000 0000	0xFFFF FFFF 8000 0000
1	0xBFC0 0200	0xFFFF FFFF BFC0 0200
0(Ca. Er .)	0xa000 0000	0xFFFF FFFF a000 0000
1(Ca. Er .)	0xBFC0 0200	0xFFFF FFFF BFC0 0200

Table 3: Exception Vectors

Excpt T ype	Of fs et	BE V	R4200 Processor Exception V ector	
			32-bit mode	64-bit mode
TLB Refill	0x00 0	0	0x8000 0000	0xFFFF FFFF 8000 0000
		1	0xBFC0 0200	0xFFFF FFFF BFC0 0200

Table 3: Exception Vectors

XTLB Refill	0x080	0	0x8000 0080	0xFFFF FFFF 8000 0080
		1	0xBFC0 0280	0xFFFF FFFF BFC0 0280
Cache Error	0x100	0	0xa000 0100	0xFFFF FFFF a000 0100
		1	0xBFC0 0300	0xFFFF FFFF BFC0 0300
All Other	0x180	0	0x8000 0180	0xFFFF FFFF 8000 0180
		1	0xBFC0 0380	0xFFFF FFFF BFC0 0380

While more than one exception can occur for a single instruction, only the exception with the highest priority is reported. Table 4 lists all exceptions in the order of their priority.

Table 4: Exception Priority Order

Reset (highest priority)
NMI
Address Error -- Instruction fetch
TLB Refill -- Instruction fetch
TLB Invalid -- Instruction fetch
Cache Error -- Instruction fetch
Bus Error -- Instruction fetch
System Call
Breakpoint
Coprocessor Unusable
Reserved Instruction
Trap Instruction
Integer Overflow
Floating-point Exception
Address Error -- Data access
TLB Refill -- Data access
TLB Invalid -- Data access
TLB Modified -- Data write
Cache Error -- Data access
Watch
Bus Error -- Data access
Interrupt

Exceptions are logically precise (the pipeline exception DCPE in the WB stage is the only imprecise exception); the instruction that causes an exception and all those that follow it are aborted, generally before committing any state, and can be re-executed after servicing the exception. When following instructions are killed, exceptions associated with those instructions are also killed, so that exceptions are not taken in the order detected, but in instruction fetch order.

The exception handling system is responsible for efficiently handling relatively infrequent events, such as translation misses, arithmetic overflow, I/O interrupts, and system calls. These events cause the interruption of the normal flow of execution; aborting instructions which cause exceptional conditions and all those which follow and have already begun executing, and a direct jump into a designated handler routine.

The architecture defines a minimal amount of additional state which is saved in coprocessor registers in order to facilitate the analysis of the cause of the exception, the servicing of the event which caused it, and the resumption of the original flow of execution, when applicable.

INTERF ACE SIGNALS

The processor interface signals allow the processor to access external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor internal resources. The signals include the System interface, the Clock/Control interface, the Interrupt interface, the Joint Test Action Group (JTAG) interface, and the Initialization interface. Figure 49 illustrates the functional grouping of the processor signals.

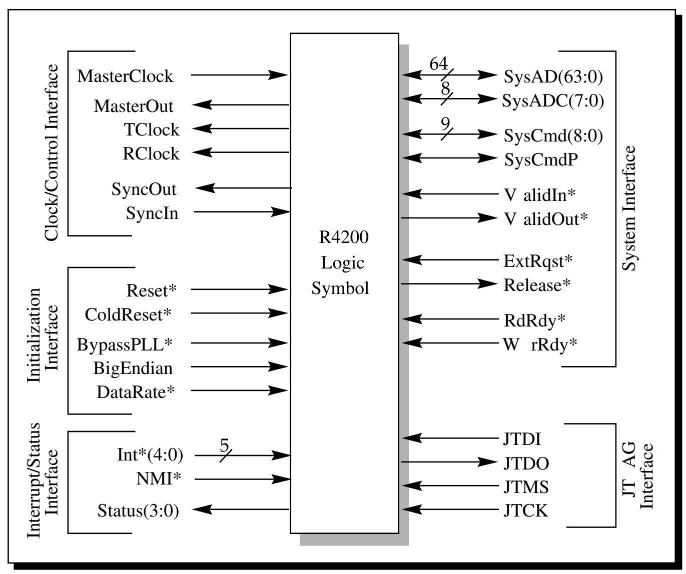


Figure 49: R4200 System Interface Signals

The system interface signals provide the connection between the R4200 processor and the other components in the system. It is compatible with the R4000 System interface. The system interface consists of: 64-bit address and data bus with 8-bit check bus, 9-bit command bus with a parity bit, and six handshaking signals. Table 5 lists the system interface signals.

Table 5: System Interface Signals

Name	Direc-tion	Description
SysAD(63:0)	Input/ Output	A 64-bit address and data bus for communication between the processor and an external agent.

Table 5: System Interface Signals

SysADC(7:0)	Input/Output	An 8-bit bus containing check bits for the SysAD bus.
SysCmd(8:0)	Input/Output	A 9-bit bus for command and data identifier transmission between the processor and an external agent.
SysCmdP	Input/Output	A single, even-parity bit for the SysCmd bus.
V alidIn*	Input	The external agent asserts V alidIn * when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus.
V alid-Out*	Output	The processor asserts ValidOut* when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus.
ExtRqst*	Input	An external agent asserts ExtRqst * to request use of the System interface. The processor grants the request by asserting Release *.
Release*	Output	In response to the assertion of ExtRqst *, the processor asserts Release *, signalling to the requesting device that the System interface is available.
RdRdy*	Input	The external agent asserts RdRdy * to indicate that it can accept a processor read or can accept a read followed by write request.
W rRdy*	Input	An external agent asserts W rRdy * when it can accept a processor write request.

The clock/control interface signals make up the interface for clocking and clock synchronization. Table 6 lists the Clock/Control interface signals.

Table 6: Clock/Control Interface Signals

Name	Direction	Description
Master-Clock	Input	Master clock input that establishes the processor operating frequency.
Master-Out	Output	Master clock output aligned with MasterClock .
TClock	Output	Transmit clocks that establish the System interface frequency. TClock is alignedwith MasterOut .
RClock	Output	Receive clocks that establish the System interface frequency. RClock leads MasterOut by 90 degrees.
SyncOut	Output	Synchronization clock output. Must be connected to SyncIn through an interconnect that models the interconnect between MasterOut , TClock , RClock , and the external agent.
SyncIn	Input	Synchronization clock input.

The initialization interface signals make up the interface by which an external agent initializes the processor operating parameters. Table 7 lists the initialization interface signals.

Table 7: Initialization Interface Signals

Name	Direction	Description
Reset*	Input	This signal must be asserted for any reset sequence.
Cold-Reset*	Input	When asserted, this signal indicates to the R4200 processor that the +3.3 volt power supply is stable and the R4200 chip should initiate a cold reset sequence. The assertion of ColdReset* will reset the PLL.
ByPass-PLL*	Input	This signal forces MasterClock to bypass the PLL and to feed directly to the clock buffers.
BigEn-dian*	Input	This signal defines the byte order of the system interface.
Dat-aRate*	Input	This signal defines the rate at which the R4200 processor will send data to the external agent on block write transactions. - 0 = DDx data pattern - 1 = Dxx data pattern

The interrupt/status interface signals make up the interface used by the external agent to interrupt the R4200 processor and to monitor instruction execution for the current processor cycle. The status pins are zero except when ITS bit in Status register is one. Table 8 lists the interrupt/status interface signals.

Table 8: Interrupt/Status Interface Signals

Name	Direction	Description
Int*(4:0)	Input	Five general processor interrupts. These are visible as bits 14 to 10 of the Cause register.
NMI*	Input	Non-maskable interrupt.
Status(3:0)	Input	This signals change with each rising edge of pclock and indicate the decode of the instruction occupying the WB pipestage for the current processor cycle. The decode for status(3:0) are: - 0 = Other integer instruction - 1 = Integer load - 2 = Integer untaken branch - 3 = Integer taken branch - 4 = Integer store - 5 - 7 = reserved - 8 = Other stall type - 9 = Stall on instruction cache access - a = Stall on data cache access - b = Reserved - c = Floating-point op - d = Killed by branch - e = Kill by exception - f = Reserved

The JTAG interface signals make up the interface that provides the JTAG boundary scan mechanism. Table 9 lists the JTAG interface signals.

Table 9: JTAG Interface Signals

Name	Direction	Description
JTDI	Input	Data is serially scanned in through this pin.
JTCK	Input	The processor receives a serial clock on JTCK . On the rising edge of JTCK , both JTDI and JTMS are sampled.
JTDO	Output	Data is serially scanned out through this pin.
JTMS	Input	JTAG command signal, indicating the incoming serial data is command data.

Figure 50 shows the primary communication paths for the System interface: a 64-bit address and data bus, SysAD(63:0), and a 9-bit command bus, SysCmd(8:0). These buses are bidirectional; that is, they are driven by the processor to issue a processor request, and by the external agent to issue an external request.

A request through the System interface consists of: an address, a system interface command that specifies the precise nature of the request, and a series of data elements if the request is for a write, or read response.

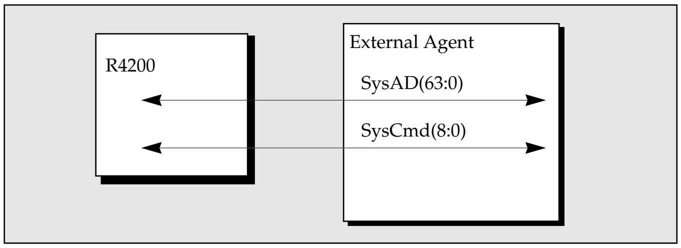


Figure 50: System Interface Bus

Cycles in which the SysAD bus contains a valid address are called *address cycles*. Cycles in which the SysAD bus contains valid data are called *data cycles*. The most significant bit of the SysCmd bus is used to indicate whether the current cycle is an address cycle or a data cycle. During address cycles, the remainder of SysCmd contains a system interface command. During data cycles, the remainder of SysCmd contains the data identifier.

The processor will repeat the address cycle until the external agent indicates that it is capable of accepting the request. The last address cycle is called the *issue cycle*. There are two types of issue cycle: processor read request issue cycles, and processor write request issue cycles. The processor samples the signal RdRdy* to determine the issue cycle for a processor read request; the processor samples the signal WrRdy* to determine the issue cycle of a processor write request. After the issue cycle, if the processor request requires data to be sent, the data transmission begins. There is only one issue cycle for any processor request.

As shown in Figure 51, RdRdy* must be asserted two cycles prior to the address cycle of the processor read request to define the address cycle as the issue cycle.

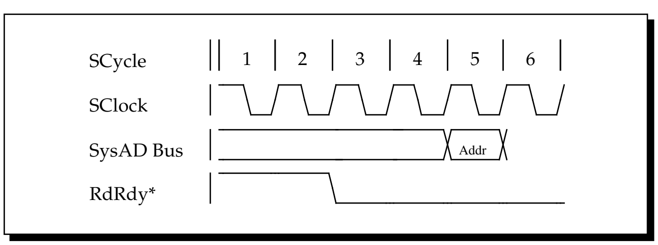


Figure 51: State of RdRdy Signal for Read Requests*

As shown in Figure 52, WrRdy* must be asserted two cycles prior to the address cycle of the processor write request to define the address cycle as the issue cycle.

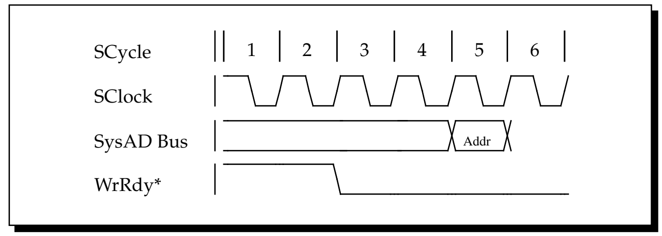


Figure 52: State of WrRdy Signal for Write Requests*

When the R4200 processor is driving the SysAD and SysCmd buses, the System interface is in *master state*. When the external agent is driving the SysAD and SysCmd buses, the System interface is in *slave state*. The processor is the default master of the system interface.

The external agent becomes master of the system interface only through arbitration protocol or *uncompelled change to slave state*. An uncompelled change to slave state is initiated by the processor, after it issued a read request, by asserting Release*. The arbitration cycle consists of the following steps:

1. The external agent asserts ExtRqst* when it wishes to submit an external request.
2. The processor waits until it is ready to handle an external request, whereupon it asserts Release* for one cycle.
3. The processor sets the SysAD and SysCmd buses to tri-state.
4. The external agent must begin driving the SysAD bus and the SysCmd bus two cycles after the assertion of Release*.
5. The external agent deasserts ExtRqst* two cycles after the assertion of Release*, unless the external agent wishes to perform an additional external request.
6. The external agent sets the SysAD and the SysCmd buses to tri-state at the completion of an external request.

The processor can start issuing a processor request one cycle after the external agent sets the bus to tri-state.

Figure 53 is a timing diagram of the arbitration protocol, in which slave and master states are shown. Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

There are two broad categories of requests: *processor requests* and *external requests*. Processor requests include: read requests, and write requests. External requests include: read responses, write requests, and null requests.

A ***processor request*** is a request through the System interface, to access some external resource. As shown in

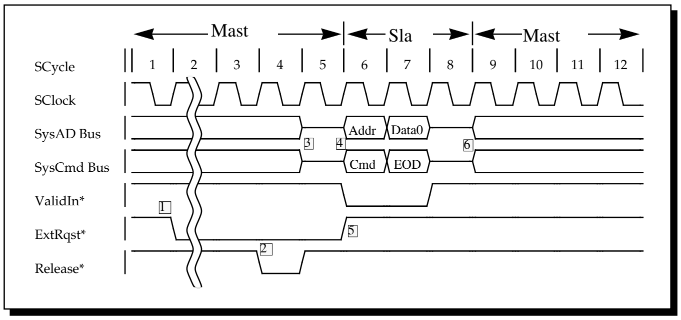


Figure 53: Bus Arbitration Protocol for External Request

Figure 54 processor requests include read, and write.

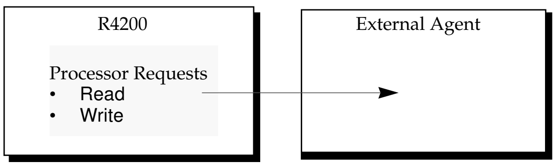


Figure 54: Processor Requests

The following rules apply to processor requests.

- After issuing a processor read request, the processor cannot issue a subsequent read request until it has received a read response.
- It is possible that back-to-back write requests can be issued by the R4200 with no idle cycles on the bus between requests.

Processor read request asks for a block, doubleword, partial doubleword, word, or partial word of data either from main memory or from another system resource. When a processor issues a read request, the external agent must access the specified resource and return the requested data.

A processor read request that has been issued, but for which data has not yet been returned, is said to be *pending*. A read remains pending until the requested read data is returned. Only one processor read request may be pending at a time.

The processor transitions to slave after the issue cycle of the read request by asserting the Release* signal for one cycle. An external agent may then return the requested data via a read response. A processor read request is completed after the last word of response data has been received from the external agent. Note that the data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to take a bus error.

The following sequence describes the protocol for a processor read request (the numbered steps below correspond to Figures 55 and 56).

1. RdRdy* is asserted low, indicating the external agent is ready to accept a read request.
2. With the System interface in master state, a processor read request is issued by driving a read command on the SysCmd bus and a read address on the SysAD bus.
3. At the same time, the processor asserts ValidOut* for one cycle, indicating valid data is present on the SysCmd and the SysAD buses.
4. The processor makes an uncompelled change to slave state either at the issue cycle of the read request, or sometime after the issue cycle of the read request by asserting the Release* signal for one cycle.

   NOTE: The external agent must not assert the signal ExtRqst* for the purposes of returning a read response, but rather must wait for the uncompelled change to slave state. The signal ExtRqst* can be asserted before or during a read response to perform an external request other than a read response.
5. The processor releases the SysCmd and the SysAD buses one SCycle after the assertion of Release*.
6. The external agent drives the SysCmd and the SysAD buses within two cycles after the assertion of Release*.

Once in slave state (starting at cycle 5 in Figure 55), the external agent can return the requested data through a read response. The read response can return the requested data or, if the requested data could not be successfully retrieved, an indication that the returned data is erroneous. If the returned data is erroneous, the processor takes a bus error exception.

Figure 55 illustrates a processor read request, coupled with an uncompelled change to slave state, that occurs as the read request is issued. Figure 56 illustrates a processor read request, and the subsequent uncompelled change to slave state, that occurs sometime after the read request is issued.

NOTE: Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

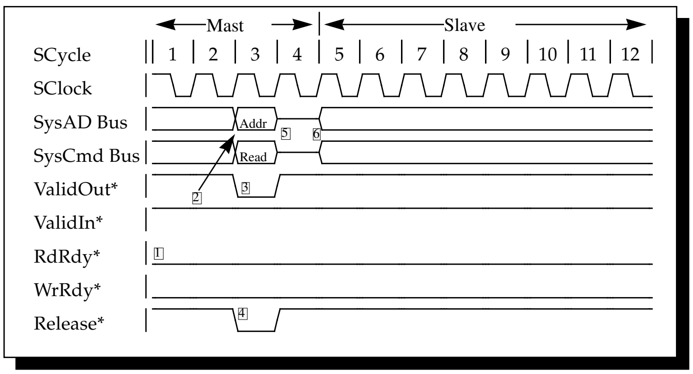


Figure 55: Processor Read Request Protocol

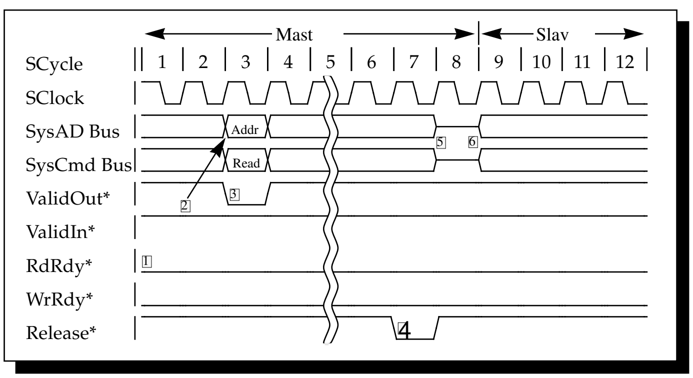


Figure 56: Processor Read Request Protocol, Change to Slave State Delayed

A processor read request can be split from the external agent's return of the requested data; in other words, the external agent can initiate an unrelated external request before it returns the response data for a processor read. This occurs when a read request is pending, ExtRqst* is asserted, and Release* is asserted for one cycle—it may be unclear if the assertion of Release* is in response to ExtRqst*, or represents an uncompelled change to slave state. In this cases, the processor accepts either a read response, or any other external request. If any external request other than a read response is issued, the processor performs another uncompelled change to slave state after processing the external request.

An external agent must return data to the processor in response to a processor read request by using a read response protocol. A read response protocol consists of the following steps:

1. The external agent waits for the processor to perform an uncompelled change to slave state.
2. The external agent returns the data through a single data cycle or a series of data cycles.
3. After the last data cycle is issued, the read response is complete and the external agent sets the SysCmd and SysAD buses to a tri-state.
4. The System interface returns to master state.

   NOTE: The processor always performs an uncompelled change to slave state after issuing a read request.
5. The data identifier for data cycles must indicate the fact that this data is *response data*.
6. The data identifier associated with the last data cycle must contain a *last data cycle* indication.

The data identifier associated with a data cycle can indicate that the data transmitted during that cycle is erroneous; however, an external agent must return a data block of the correct size regardless of the fact that the data may be in error. If a read response includes one or more erroneous data cycles, the processor then takes a bus error.

Read response data must only be delivered to the processor when a processor read request is pending. The behavior of the processor is undefined when a read response is presented to it and there is no processor read pending.

Figure 57 illustrates a processor word read request followed by a word read response. Figure 58 illustrates a read response for a processor block read with the System interface already in slave state.

NOTE: Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

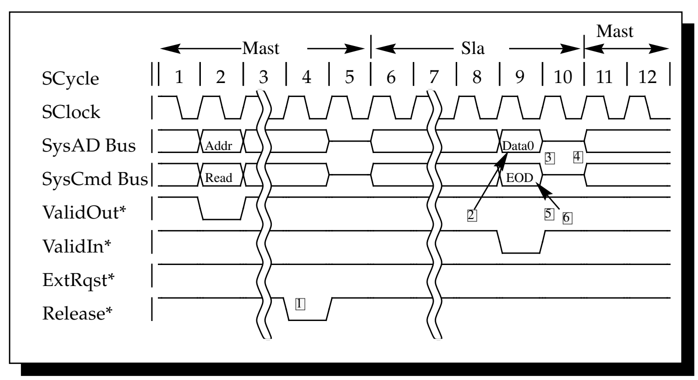


Figure 57: Processor Word Read Request, follow by a word Read Response Protocol

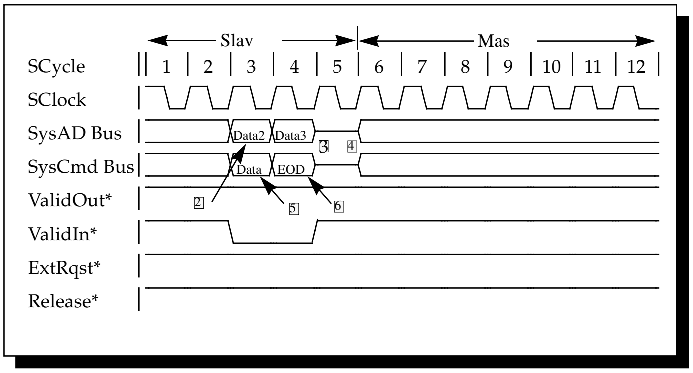


Figure 58: Block Read Response, System Interface already in Slave State

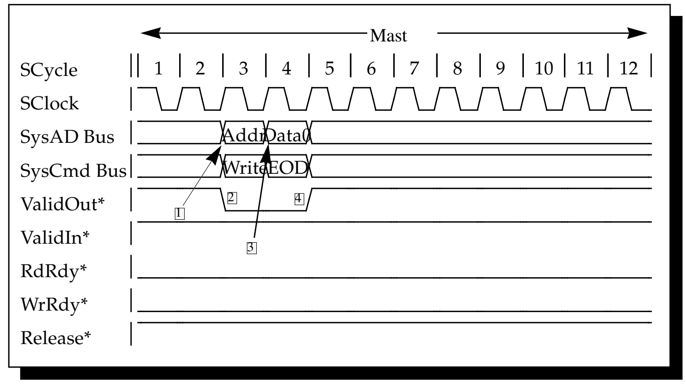


Figure 59: Processor Single Word Write Request Protocol

Processor write request provides a block, doubleword, partial doubleword, word, or partial word of data to be written either to main memory or to another system resource. When a processor issues a write request, the specified resource is accessed and the data is written to it.

A processor write request is complete after the last word of data has been transmitted to the external agent.

Processor write requests are issued using one of two protocols:

- Doubleword, partial doubleword, word, or partial word writes use a word[1] write request protocol.
- Block writes use a block write request protocol.

Processor doubleword write requests are issued with the System interface in master state, as described below in the steps below; Figure 59 shows a processor single word write request cycle.

1. A processor single word write request is issued by driving a write command on the SysCmd bus and a write address on the SysAD bus.
2. The processor asserts ValidOut*.
3. The processor drives a data identifier on the SysCmd bus and data on the SysAD bus.
4. The data identifier associated with the data cycle must contain a last data cycle indication. At the end of the cycle, ValidOut* is deasserted.

   NOTE: Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

Processor block write requests are issued with the System interface in master state, as described below; a processor block request for four words of data is illustrated in Figures 60 and 61.

1. The processor issues a write command on the SysCmd bus and a write address on the SysAD bus
2. The processor asserts ValidOut*.
3. The processor drives a data identifier on the SysCmd bus and data on the SysAD bus.
4. The processor asserts ValidOut* for a number of cycles sufficient to transmit the block of data.
5. The data identifier associated with the last data cycle must contain a last data cycle indication.

   NOTE: As shown in Figure 61, however, the first data cycle does not have to immediately follow the address cycle.

Figures 60 and 61 illustrate a processor block request for four words of data.

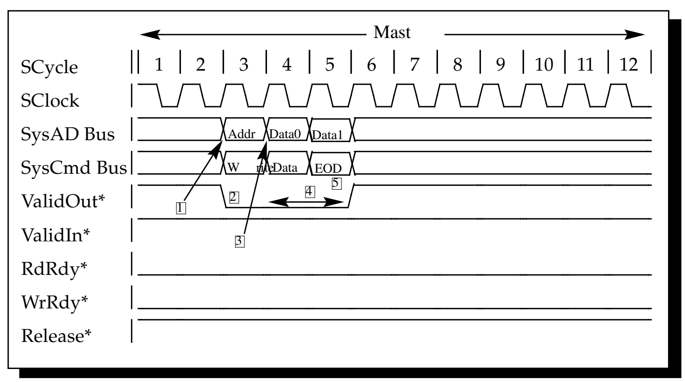


Figure 60: Processor Block Write Request Protocol

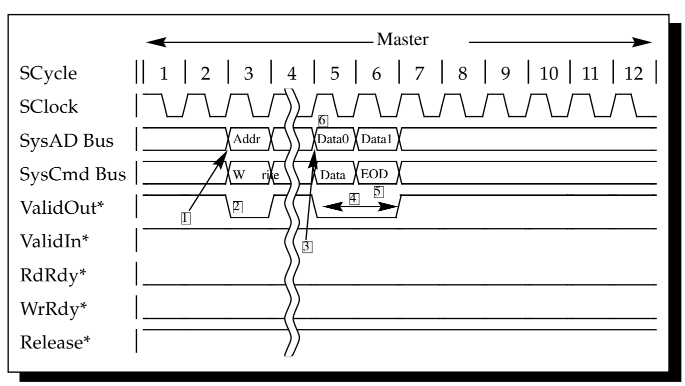


Figure 61: Processor Block Write Request Protocol (Delayed in Transmit Data)

1. Called *word* to distinguish it from *block* request protocol. Data transferred can actually be doubleword, partial doubleword, word, or partial word.

The external agent uses RdRdy* and WrRdy* signals to control the ***flow of the processor read and write requests***, respectively.

Figures 62 through 64 illustrate this flow control, as described in the steps below.

1. The processor samples the signal RdRdy* to determine if the external agent is capable of accepting a read request.
2. The signal WrRdy* controls the flow of a processor write request.
3. The processor does not complete the issue of a read request, until it issues an address cycle in response to the request for which the signal RdRdy* was asserted two cycles earlier.
4. The processor does not complete the issue of a write request until it issues an address cycle in response to the write request for which the signal WrRdy* was asserted two cycles earlier.

Figure 62 illustrates two processor write requests in which the issue of the second is delayed for the assertion of WrRdy*.

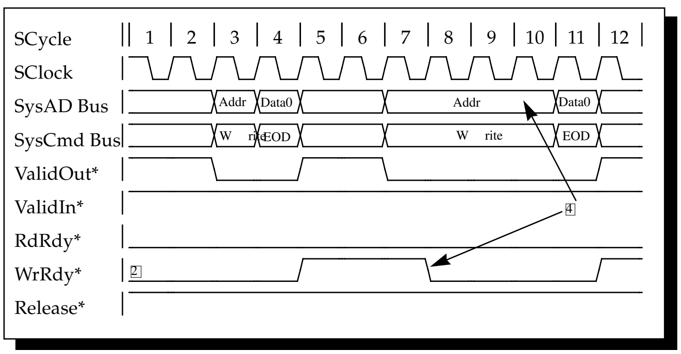


Figure 62: Processor Write Request Protocol (Delayed by WrRdy)*

Figure 63 illustrates a processor read request in which the issue of the read is delayed for the assertion of RdRdy*.

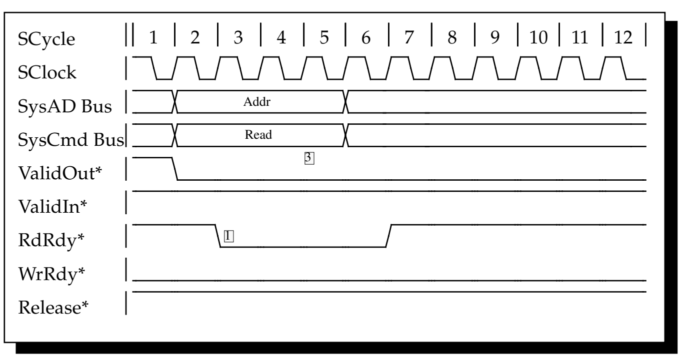


Figure 63: Processor Read Request Protocol (Delayed by RdRdy)*

Figure 64 illustrates the issue of a processor write request delayed for the assertion of WrRdy* and the completion of an external write request.

NOTE: Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

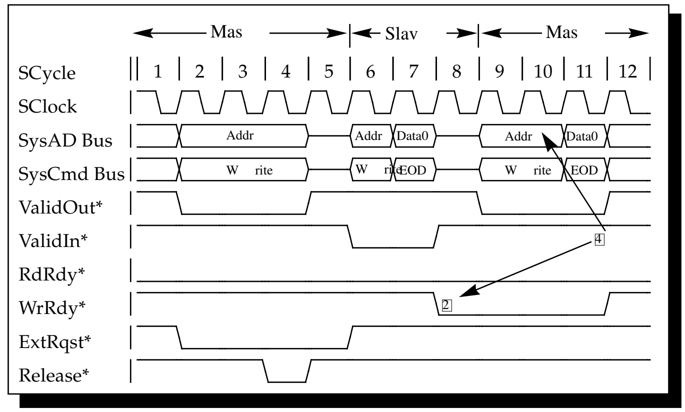


Figure 64: Processor Write Request Protocol Delayed for the Assertion of WrRdy and the Completion of an External Write Request*

An ***External request*** is a request through the System interface, to access the processor's internal resources. As shown in Figure 54 external requests include write, and null requests.

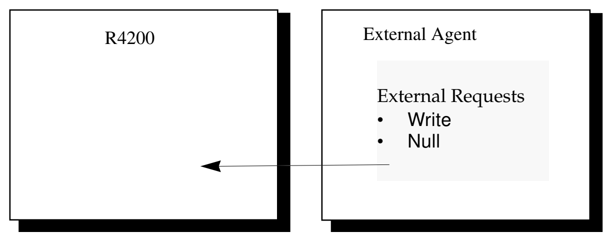


Figure 65: External Requests

Mastership of the System interface always returns to the processor after an external request is issued. The processor does not accept a subsequent external request until it has completed the current request. The processor can accept external requests between the issue of a processor read request.

If there are no processor requests pending, the processor decides, based on its internal state, whether to accept the external request, or to issue a new processor request. The processor can issue a new processor request even if the external agent is requesting access to the System interface.

The external agent asserts ExtRqst* indicating that it wishes to begin an external request. The external agent then waits for the processor to signal that it is ready to accept this request by asserting Release*. The processor signals that it is ready to accept an external request based on the criteria listed below.

- The processor completes any processor request that is in progress.
- While waiting for the assertion of RdRdy* to issue a processor read request, the processor can

accept an external request if the request is delivered to the processor one or more cycles before RdRdy* is asserted.

- While waiting for the assertion of WrRdy* to issue a processor write request, the processor can accept an external request provided the request is delivered to the processor one or more cycles before WrRdy* is asserted.
- If waiting for the response to a read request after the processor has made an uncompelled change to a slave state, the external agent can issue an external request before providing the read response data.

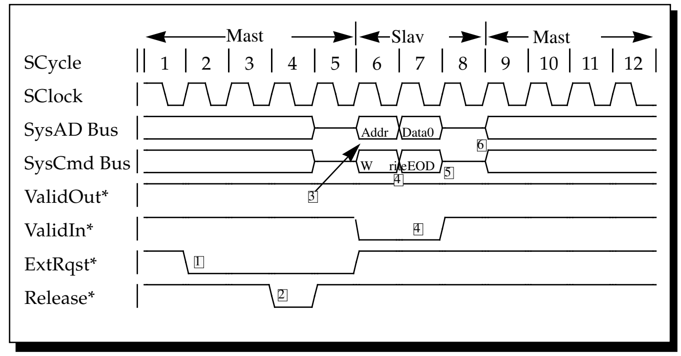


Figure 66: External Write Request Protocol

External Write request provides a word of data to be written to the processor's internal resource. When an external agent issues a write request, the specified resource is accessed and the data is written to it. An external write request is complete after the word of data has been transmitted to the processor.

The only processor resource available to an external write request is the *Interrupt* register.

External write requests use a protocol identical to the processor single word write protocol except the ValidIn* signal is asserted instead of ValidOut*. Figure 66 shows a timing diagram of an external write request, which consists of the following steps:

1. The external agent asserts ExtRqst* to arbitrate for the System interface.
2. The processor releases the System interface to slave state by asserting Release*.
3. The external agent drives a write command on the SysCmd bus, a write address on the SysAD bus, and asserts ValidIn*. This write address is ignored by the R4200 as the only writable resource is the Interrupt register.
4. The external agent drives a data identifier on the SysCmd bus, data on the SysAD bus, and asserts ValidIn*.
5. The data identifier associated with the data cycle must contain a last data cycle indication.
6. After the data cycle is issued, the write request is complete and the external agent sets the SysCmd and SysAD buses to a tri-state, allowing the System interface to return to master state. Timings for the SysADC and SysCmdP buses are the same as those of the SysAD and SysCmd buses, respectively.

   NOTE: External write requests are only allowed to write a word of data to the processor. Processor behavior in response to an external write request for any data element other than a word is undefined.

When the processor releases the System interface to slave state to accept an external request, it also allows the external agent to use the SysAD bus for a transfer unrelated to the processor. After the transfer is completed, the external agent may use ***External Null*** request to return control of the system interface to the master state without affecting the processor. This application is useful in a system that wishes to have additional drives and receivers on SysAD bus; as illustrated in Figure 67.

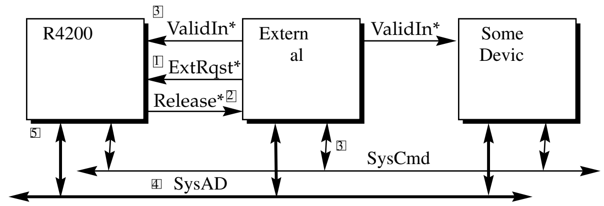


Figure 67: Multiple SysAD Drivers and Receivers System Application

Figure 68 shows timing diagrams of external null request cycle, which consist of the following steps:

1. The external agent asserts ExtRqst* to arbitrate for the System interface.
2. The processor releases the System interface to slave state by asserting Release*.
3. The external agent drives a external null request command on the SysCmd bus, and asserts ValidIn* for one cycle.
4. The SysAD bus is unused (does not contain valid data) during the address cycle associated with an external null request.
5. After the address cycle is issued, the null request is completed, and the System interface returns to master state.The processor controls the ***flow of external requests*** through the arbitration signals ExtRqst* and Release*. The external agent must acquire mastership of the System interface before it is allowed to issue an external request.

The R4200 system interface supports a maximum data rate of one double word per cycle. The rate at which data is delivered to the processor may be controlled by an external agent by driving data and asserting ValidIn* only when data is available. The processor will interpret cycles during which ValidIn* is asserted and the SysCmd bus contains a data identifier as valid data cycles. The processor will continue to accept data until the data word tagged as the last data word is received. An external agent may deliver data to the processor at the system interface maximum data rate.

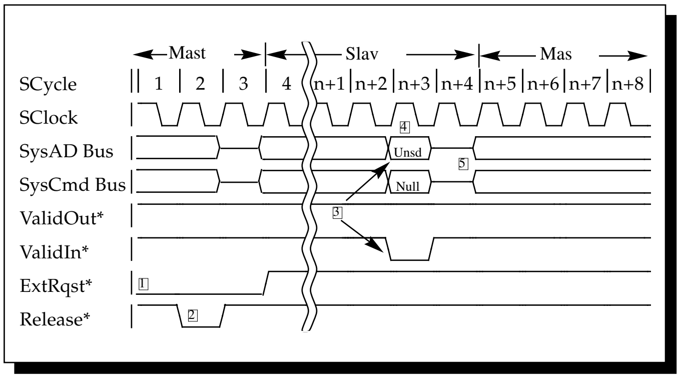


Figure 68: External Null Request Protocol

The rate at which the processor transmits data to an external agent is programmable via the Datarate* signal. Data patterns are specified using the letters "D" and "x", where "D" indicates a data cycle and "x" indicates an unused cycle. A data pattern is specified as a sequence of letters, indicating a sequence of data and unused cycles that will be repeated to provide the appropriate number of data cycles for a given transfer. The two data rates supported by the R4200 processor are: DDx and Dxx. Figure 69 shows a processor block write with a Dxx data rate.

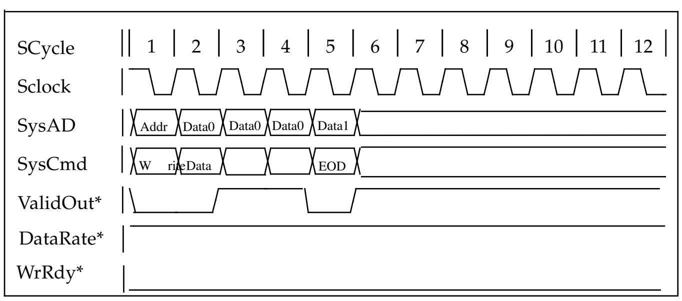


Figure 69:Processor Block Write with Dxx Data Pattern

CLOCKS

The clocks on the R4200 processor are controlled via an on-chip Phase Locked Loop circuit. This circuit keeps the internal clock edges aligned with the clock edges of the *MasterClock* signal, which acts as the system master clock.

Inside the R4200 processor, the *MasterClock* signal is multiplied by a factor of four, and then all internal clocks are derived by dividing that signal down by some factor of two. There are two primary internal clocks, the pipeline clock *PClock*, and the system interface clock *SClock*. While other internal edges may be generated, these are transparent to the user.

The R4200 processor also provides external clocks to the system. These clocks are the receive clock *RClock* and the transmit clock *TClock*. T*Clock* is aligned with *SClock*, while *RClock* leads by a quarter clock cycle. This is the clocking scheme identical to that used on the R4000 system interface.

The pipeline clock *PClock* is normally equal to twice the *MasterClock* frequency. In reduced power mode, when the RP bit in the Status register is set, *PClock* is driven at one half of the Master Clock frequency. This is one fourth the normal *PClock* speed.

The system interface clock, *SClock*, is normally equal to one half of the *PClock* frequency. This is equal to the *MasterClock* frequency. S*Clock* is always derived from *PClock*. Thus when *PClock* is in reduced power mode, *SClock* also reduces in frequency by a factor of four. The R4200 processor drives its outputs on this clock edge.

RClock is generated at the same frequency as SClock. It is an external clock used to clock input registers on external agents. It is skewed by the processor to lead TClock by 25% of the SClock frequency.

TClock is generated by the processor at the same frequency as SClock. It is aligned with SClock. It is used by external agents to drive data, and as the global clock for the external agent. TClock can be thought of as the synchronized external system interface clock.

The R4200 processor's clocks are controlled by a phase locked loop circuit. The phase locked loop circuitry can be disabled by the *BypassPLL** pin. In this mode, the internal clock is driven directly by the *MasterClock* pin of the R4200 processor. This mode can be used for test purposes, as well as in systems whose MasterClock is running too slowly for the phase locked loop circuit to lock to. The phase locked loop remains constant when the pipeline slows to one quarter of normal frequency. Power dissipated by the PLL is roughly estimated at 0.1W and therefore relatively insignificant.

MODE SELECTABLE FEATURES

The R4200 processor will support several user selectable modes, most of which can be set and reset by writing to the CP0 Status register. In addition, there are two dedicated pins, BigEndian and BypassPLL*, which are used to selected byte ordering on System interface and to disable the phase-lock-loop circuitry during the reset sequence. The following sections detail all modes supported by the R4200 processor.

The ***Reduced Power*** (RP) mode allows the user to change the processor operating frequency to quarter speed. This is established by setting bit 27 of the Status register.

This feature is included to allow the user to selectively reduce power when the system is not being heavily used. This feature will reduce the power consumed by the processor chip to 25% of its normal value. The default of this mode is normal clocking. The chip will return to this state after any reset.

The ***Floating-point Register*** (FR) mode (bit 26 of the Status register) enables the user to access the full set of 32 64-bit floating point registers as defined in MIPS-III. When reset, the processor will access the registers as defined in the MIPS II architecture. This functionality is the same as the R4000.

The ***Reverse Endianess*** (RE) mode (bit 25 of the Status register) allows the user to switch byte ordering, between

BigEndian and LittleEndian, as seen by user software.

The ***Instruction Trace Support*** (ITS) mode allows the user to track branches or jumps. This mode is set by writing bit 24 of the Status register. When the ITS bit is set, the physical address to which the CPU has branched will be reported on the SysAD bus by forcing an instruction cache miss whenever a branch, jump or exception is taken. The ITS bit will also enable the driving of the Status pins. When ITS=0, the Status pins will not be driven. By observing the status pins, it is possible to observe when the CPU has completed a Branch or Jump instruction, in addition to other information. Instruction Trace Support mode is new in the R4200 and is not implemented in the R4000 processor.

The ***Bootstrap Exception Vectors*** (BEV), bit 22 in the Status Register, when set, causes the TLB refill exception vector to be relocated to a virtual address of 0xbfc00200 and the general exception vector to 0xbfc00380. When cleared, these vectors are normally located at 0x80000000 (TLB refill), 0x00000080 (XTLB refill), and 0x80000180 (general), respectively. This bit is used when diagnostic tests cause exceptions to occur prior to verifying proper operation of the cache and main memory system.

The ***Cache Error*** check (CE), bit 17 of the Status register, will force the substitution of the 8-bit field in the PErr register for computed parity. On stores, this will enable the user to directly write the parity bits in the data cache, rather than having the computed parity bits used. The parity bits in the instruction cache can be directly written from the PErr register using the Cache operation Fill and having the CE bit set. If CE is set, the Tag parity bit will be written from the TagLo register rather than being computed form the tag. This feature is used for test and diagnostic purposes.

The ***Disable Errors*** (DE), bit 16 of the Status register, when set, will specify that the processor will not take an exception on a cache parity error.

The ***Kernel Extended Addressing*** (KX), bit 7 of the Status register, when set will enable the extended addressing TLB refill exception vector to be used for TLB misses on kernel addresses.

The ***Supervisor Extended Addressing*** (SX), bit 6 of the Status register, if set, enables MIPS III opcodes in supervisor-mode and causes TLB misses on supervisor addresses to use the Extended TLB refill exception vector.

The ***User Extended Addressing*** (UX), bit 5 of the Status register, if set, enables MIPS III opcodes in user-mode and causes TLB misses on user addresses to use the Extended TLB refill exception vector. If clear, implements MIPS II compatibility on virtual address translation.

The ***Interrupt Enable*** (IE), bit 0 of the Status register, when clear, will not allow interrupts with the exception of reset and non-maskable interrupt.

LOW POWER DISSIPATION

The R4200 is carefully designed to restrict power dissipation to make the processor suitable for operation in battery-powered devices. Power dissipation in the R4200 is controlled in two ways:

- Power management: The processor may be switched into reduced-power mode or power-down mode by external system logic.
- Reduced Power: A wide variety of logic and circuit design techniques is used to reduce power consumed in normal operating mode.

POWER MANAGEMENT

The R4200 may be dynamically switched to one quarter of its normal operating frequency by the setting Reduce Power (RP) mode bit in the Status register. This will slow the pipeline clock to 20 MHz and the processor clock to 10 MHz. This event is typically initiated by external logic which sets the RP bit in the status register via software. On setting this bit, the output of the clock dividers changes to provide the slower clock rate. Processing continues at one quarter the rate as before. There is a corresponding drop in power dissipated to one quarter of that previously.

The Power-down mode is typically initiated by external logic. All dirty cache lines are written back to main memory. All variable registers are readable and writable by software. This allows them to be saved and written to non-volatile memory while the processor is powered down. When power is restored, the processor state may be written back. State information is written using external software routines.

REDUCED POWER

A variety of circuit and logic design techniques is used to reduce power dissipation in the R4200. Many of these techniques save only a small amount of power but it is the cumulative effect of these that produces a significant result.

These techniques are:

- 3.3V Power supply
- dynamic (rather than static) logic design
- unified datapath for less capacitance
- turning off unused portions of the chip
- minimized feature set
- 5-stage pipeline for high efficiency
- write-back data cache
- double instruction fetch from I-cache
- use of 2-entry micro I-TLB
- unused logic operates on constants
- gating clocks to pipeline logic to preserve data in a pipeline stall
- Speed de-optimization (no duplicate critical paths)
- Slew rate control and optimal fanout ratios for minimized switching current.
- Minimized Vcc to Vss current path

These all combine to produce the low average power dissipation that allows the R4200 to be a practical battery-powered processor.

PACKAGE INFORMATION

R4200LP - 208-pin PQFP pin-out :

1	NC	53	NC	105	NC	157	NC
2	NC	54	NC	106	NC	158	NC
3	VSS	55	SysCmd2	107	VCCCore	159	VSS
4	VCC	56	SysAD36	108	VSSCore	160	RClock
5	SysAD45	57	SysAD4	109	VCC	161	SyncOut
6	SysAD13	58	SysCmd1	110	VSS	162	SysAD30
7	Status0	59	VSS	111	SysAD21	163	VCC
8	SysAD44	60	VCC	112	SysAD53	164	VSS
9	VSSCore	61	SysAD35	113	RdRdyB	165	SysAD62
10	VCCCore	62	SysAD3	114	BigEndian	166	MasterOut
11	SysAD12	63	SysCmd0	115	SysAD22	167	SysAD31
12	SysCmdP	64	SysAD34	116	SysAD54	168	SysAD63
13	SysAD43	65	VSS	117	VCC	169	VCCCore
14	SysAD11	66	VCC	118	VSS	170	VSSCore
15	VSS	67	VSSCore	119	ReleaseB	171	Div2
16	VCC	68	VCCCore	120	SysAD23	172	SysADC3
17	SysCmd8	69	SysAD2	121	SysAD55	173	SysADC7
18	SysAD42	70	NC	122	NMIB	174	VCC
19	SysAD10	71	SysAD33	123	VCCCore	175	VSS
20	SysCmd7	72	SysAD1	124	VSSCore	176	BypassPLLB
21	VSS	73	VSS	125	SysADC2	177	VCCP
22	VCC	74	VCC	126	SysADC6	178	VSSP
23	SysAD41	75	IntB4	127	DataRateB	179	PLLCap0
24	SysAD9	76	SysAD32	128	SysAD24	180	PLLCap1
25	SysCmd6	77	SysAD0	129	VCC	181	VCCP
26	SysAD40	78	IntB3	130	VSS	182	VSSP
27	VSSCore	79	VSSCore	131	SysAD56	183	VCCP
28	VCCCore	80	VCCCore	132	Status3	184	VSSP
29	VSS	81	IntB2	133	SysAD25	185	MasterClock
30	VCC	82	SysAD16	134	SysAD57	186	VCCCore
31	SysAD8	83	SysAD48	135	VCCCore	187	VSSCore
32	SysCmd5	84	IntB1	136	VSSCore	188	Syncin
33	SysADC4	85	VSS	137	Status2	189	VCC
34	SysADC0	86	VCC	138	SysAD26	190	VSS
35	VSS	87	SysAD17	139	SysAD58	191	JTCK
36	VCC	88	SysAD49	140	TestModeB	192	SysADC5
37	SysCmd4	89	IntB0	141	VCC	193	SysADC1
38	SysAD39	90	SysAD18	142	VSS	194	JTDI
39	SysAD7	91	VSS	143	SysAD27	195	VCCCore
40	SysCmd3	92	VCC	144	SysAD59	196	VSSCore
41	VSS	93	SysAD50	145	ColdResetB	197	SysAD47
42	VCC	94	ValidInB	146	SysAD28	198	SysAD15
43	SysAD38	95	SysAD19	147	VCC	199	JTDO
44	SysAD6	96	SysAD51	148	VSS	200	SysAD46
45	Status1	97	VSS	149	SysAD60	201	VCC
46	WrRdyB	98	VCC	150	ResetB	202	VSS
47	SysAD37	99	ValidOutB	151	SysAD29	203	SysAD14
48	SysAD5	100	SysAD20	152	SysAD61	204	JTMS
49	VSS	101	SysAD52	153	VCC	205	TClock
50	VCC	102	ExtRqstB	154	VSS	206	NC
51	NC	103	NC	155	NC	207	NC
52	NC	104	NC	156	NC	208	NC

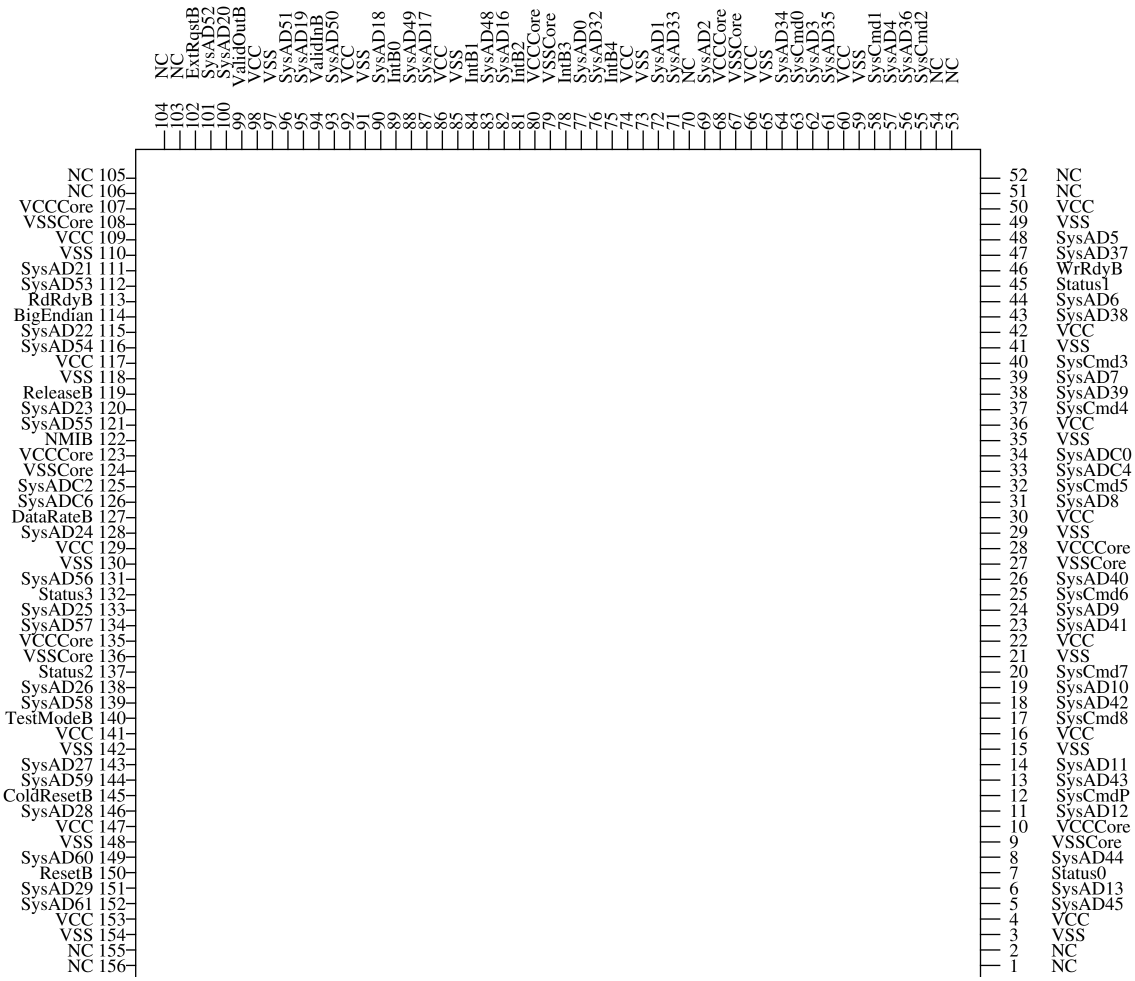

NC 104
NC 103
ExtRqstB 102
SysAD52 101
SysAD20 100
ValidOutB 99
VCC 98
VSS 97
SysAD51 96
SysAD19 95
ValidInB 94
SysAD50 93
VCC 92
VSS 91
SysAD18 90
IntB0 89
SysAD49 88
SysAD17 87
VCC 86
VSS 85
IntB1 84
SysAD48 83
SysAD16 82
IntB2 81
VCCCore 80
VSSCore 79
IntB3 78
SysAD0 77
SysAD32 76
IntB4 75
VCC 74
VSS 73
SysAD1 72
SysAD33 71
NC 70
SysAD2 69
VCCCore 68
VSSCore 67
VCC 66
VSS 65
SysAD34 64
SysCmd0 63
SysAD3 62
SysAD35 61
VCC 60
VSS 59
SysCmd1 58
SysAD4 57
SysAD36 56
SysCmd2 55
NC 54
NC 53
NC 105
NC 106
VCCCore 107
VSSCore 108
VCC 109
VSS 110
SysAD21 111
SysAD53 112
RdRdyB 113
BigEndian 114
SysAD22 115
SysAD54 116
VCC 117
VSS 118
ReleaseB 119
SysAD23 120
SysAD55 121
NMIB 122
VCCCore 123
VSSCore 124
SysADC2 125
SysADC6 126
DataRateB 127
SysAD24 128
VCC 129
VSS 130
SysAD56 131
Status3 132
SysAD25 133
SysAD57 134
VCCCore 135
VSSCore 136
Status2 137
SysAD26 138
SysAD58 139
TestModeB 140
VCC 141
VSS 142
SysAD27 143
SysAD59 144
ColdResetB 145
SysAD28 146
VCC 147
VSS 148
SysAD60 149
ResetB 150
SysAD29 151
SysAD61 152
VCC 153
VSS 154
NC 155
NC 156
52 NC
51 NC
50 VCC
49 VSS
48 SysAD5
47 SysAD37
46 WrRdyB
45 Status1
44 SysAD6
43 SysAD38
42 VCC
41 VSS
40 SysCmd3
39 SysAD7
38 SysAD39
37 SysCmd4
36 VCC
35 VSS
34 SysADC0
33 SysADC4
32 SysCmd5
31 SysAD8
30 VCC
29 VSS
28 VCCCore
27 VSSCore
26 SysAD40
25 SysCmd6
24 SysAD9
23 SysAD41
22 VCC
21 VSS
20 SysCmd7
19 SysAD10
18 SysAD42
17 SysCmd8
16 VCC
15 VSS
14 SysAD11
13 SysAD43
12 SysCmdP
11 SysAD12
10 VCCCore
9 VSSCore
8 SysAD44
7 Status0
6 SysAD13
5 SysAD45
4 VCC
3 VSS
2 NC
1 NC

Figure 70: R4200LP - 208-pin PQFP Physical Pin Location

R4200PC 179-pin PGA Package Pinout

Pin	R4200PC Function	Pin	R4200PC Function	Pin
C10	SysAD40	K1	Vcc	T10
C11	SysAD41	K2	SysAD16	T11
C12	SysAD10	K3	IntB2	T12
C13	SysCmd8	K16	VssP	T13
C14	SysCmdP	K17	VccP	T14
C15	SysAD44	K18	Vss	T15
C16	SysAD13	L1	Vss	T16
C17	TClock0	L2	SysAD48	T17
C18	Vss	L3	IntB1	T18
D1	Vss	L16	SysADC3	U1
D2	SysAD35	L17	SysADC7	U2
D3	SysCmd1	L18	Vcc	U3
D16	TClock1	M1	Vcc	U4
D17	SysAD14	M2	SysAD17	U5
D18	Vcc	M3	SysAD49	U6
E1	SysAD2	M16	SysAD31	U7
E2	SysCmd0	M17	BypassPLLB	U8
E3	SysAD3	M18	Vss	U9
E16	JTMS	N1	Vss	U10
E17	SysAD46	N2	IntB0	U11
E18	SysAD15	N3	SysAD50	U12
F1	Vcc	N16	SysAD62	U13
F2	N/C	N17	SysAD63	U14
F3	SysAD34	N18	Vcc	U15
F16	JTDO	P1	SysAD18	U16
F17	SysAD47	P2	ValidInB	U17
F18	Vss	P3	SysAD19	U18
G1	Vss	P16	SyncOut	V1
G2	SysAD1	P17	MasterOut	V2
G3	SysAD33	P18	Vss	V3
G16	JTDI	R1	Vcc	V4
G17	SysADC1	R2	SysAD51	V5
G18	Div2	R3	ValidOutB	V6
H1	Vcc	R16	RClock1	V7
H2	SysAD32	R17	SysAD30	V8
H3	IntB4	R18	Vss	V9
H16	SysADC5	T1	Vss	V10
H17	JTCK	T2	SysAD20	V11
H18	Vss	T3	SysAD52	V12
J1	Vss	T4	SysAD21	V13
J2	SysAD0	T5	RdRdyB	V14
J3	IntB3	T6	SysAD54	V15
J16	SyncIn	T7	SysAD55	V16
J17	MasterClock	T8	SysADC2	V17
J18	Vcc	T9	N/C	V18

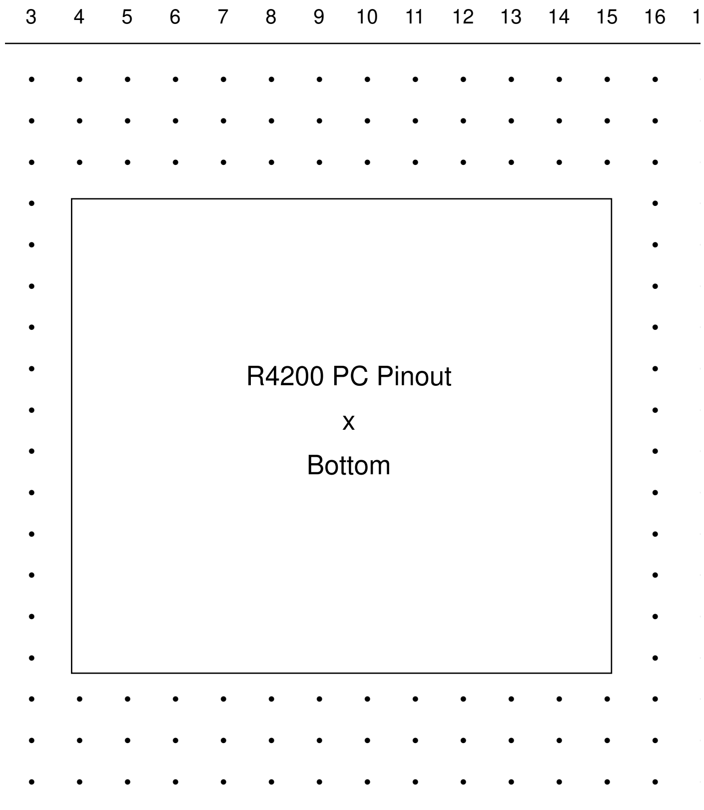


Figure 71:R4200PC - 179-pin PGA Physical Pin Location

DIFFERENCES FROM THE R4000

For a detailed list of differences amomg the R4x00 series processors, please refer to “Designing in common for R4200 (VRX), R4600 (Orion), R4000PC and R4400PC processors”, available from MIPS Technologies, Inc.

The R4200 processor implements the MIPS III instruction set, as does the R4000. As with any two hardware implementations of the same architecture, there are some differences between the R4200 and the R4000. While the R4200 has tried to stay as close to the R4000 implementation as possible both from a system design and software viewpoint, there are some visible differences. This section highlights the differences between the R4200 and the R4000.

The primary differences between the R4200 and the R4000 are in cache handling. This is primarily because the R4200 processor provides no support for second level caches. The R4200 also provides no support for multiprocessing, which has an impact on system design.

SOFTW ARE VISIBLE DIFFERENCES

The functional differences between the R4000 and the R4200 processor that are visible to software are contained in the Coprocessor 0 implementation.

Cache Ops

Because the R4200 processor does not support secondary cache, all of the references to secondary cache in the R4000 implementation do not apply to the R4200. Designating either SD or SI (secondary data or instruction cache) for any cache op on the R4200 is undefined. All writeback operations will send data from primary cache to main memory. Cache op number seven (7) is undefined for the R4200.

Cache Op Index Store Tag will use parity bit from TagLo register bit 0 instead of computed parity if CE bit (R4000 section) of cp0 Status register is set.

The data cache Dirty bit, W bit in R4000, will be cleared for cache op Hit Write Back.

Status Register

The R4200 has a slightly different status register from the R4000. The bits that perform identical functionalities are left in the same location on the R4200 as they are in the R4000.

One bit of the Status Register that was previously always zero (i.e. unused) is now used. This is the Instruction Trace Support *ITS* bit.

The status register’s CH bit, the Cp0 Condition bit used for the Branch on Co-Processor 0 *BC0T/BC0F* instructions, is writable only by software in the R4200. But, in R4000, this bit can be set or cleared by the hardware dependent on secondary cache operations, which the R4200 does not support.

Configuration Register

The R4200 configuration register implements a subset of the fields on the R4000, since the R4200 does not have as many options. The exact meaning of bits within the configuration register may be different.

Unimplemented Operation Exception and other cause bits

The R4000 defines the Unimplemented Operation Exception (E) in FSR supersedes all other cause bits. The Unimplemented Exception handler is expected to ignore and clear all cause bits. Thus, during E exception, all other cause bits are undefined or don’t care.

The R4200 has stricter definition than R4000. It sets no other FP exception cause bits when there is an Unimplemented Operation Exception.

Integer Divide-by-Zero

When dividing an integer by zero, MIPS ISA specifies the result to be undefined. Under this illegal operation, the R4200 returns a different value in Register LO than the R4000:

Here is what the R4000 returns in Hi and Lo for all cases of dividend:

Dividend	Lo	Hi
>= 0	0xFFFFFFFF	dividend

< 0	0x00000001	dividend

and the R4200 returns:

Dividend	Lo	Hi
>= 0	0x7FFFFFFF	dividend
< 0	0x80000001	dividend

Non-restartable Cache Parity Error exception

The R4200 processes the Cache Parity Error exception (DCPE) in the WB stage.

When a cache data parity error occurs on a load with a cache hit, the R4200 will not prevent the cache data with a bad parity from writing into the register file in WB stage. The exception is still precise, since both the EPC and CacheErr registers are updated with the correct virtual address pointing to the offending load instruction. The exception handler can still determine the cause of exception and its origin. But, the program cannot be restarted since the bad cache data has clobbered the destination register. Note that CacheErr exception for cache-ops, uncached access, and cache-miss remains as Dc stage exception and exact.

SYSTEM DESIGN DIFFERENCES

Processor Initialization

The R4000 had a fairly complicated scan based processor initialization scheme.The R4200 has far fewer modes; those that are not available via software are hardwired as pins on the package. The R4200 also requires only 16 cycles of Reset* assertion during the reset sequence.

System Interface

The R4200 uses the same SysAD bus protocol as the R4000; however, since the R4200 processor does not support multiprocessing or secondary cache, the R4200 only implements a subset of the full SysAD bus definition. Any transactions that reference multi-processing or secondary cache features are undefined.

The R4200 offers an optional enhancement over the R4000 SysAD protocol. This enhancement can increase the performance of uncached memory accesses, important in graphics applications. However in order to avail of this enhancement, the interfacing ASICs must be able to handle the increased traffic on the SysAD bus. Some ASICs, designed originally for the R4000 are known not to be compatible with these enhancements. Details of the optional SysAD differences in R4000 and R4200 are listed in a separate document “Designing in common for R4200 (VRX), R4600 (Orion), R4000PC and R4400PC processors”, available from MIPS Technologies, Inc.

For compatibility with existing R4000 ASICs, the default mode of operation does not use the enhancement mentioned above. The enhancement is set at boot time using the DataRate* pin.

The R4200 will support two data rates for write transactions: DDx and Dxx. The rate to be used by the R4200 is programmed via the *DataRate** pin. A DataRate* signal of ‘0’ indicates a DDx datarate and R4200 SysAD enhancement functional. A DataRate* signal of ‘1’ indicates a Dxx datarate and R4200 SysAD enhancement disabled.

The R4200 has fixed sizes for instruction cache lines (8 words) and data cache lines (4 words). Instruction blocks will be written to the memory system as an eight word transaction. (The writing of instruction cache lines to memory can be done via the Hit Write Back Cache Op).

The R4200 handles data coming onto the processor as fast as it can be delivered by the SysAD bus. There will never be a stream of data presented to the R4200 processor exceeding four doublewords, and the R4200 can handle a data pattern of DDDD.

In order to conserve power, the R4200 processor will provide only a single T clock and a single R clock.

RP Bit Effect on System Interface

On the R4000, *SClock* and *TClock* were unaffected by the RP bit. On the R4200, setting the RP bit will reduce the clock frequency of *SClock* and *TClock* by a factor of four. If there are any timing dependent features in an external agent that would be affected by this (e.g., DRAM refresh counters), the external agent must provide software a mechanism to accommodate the frequency change.

OTHER DIFFERENCES

Cache Size

The R4200 will implement a 16kB, direct mapped, virtually indexed instruction cache with a cache line of eight words (thirty-two bytes). The data cache is 8kB, direct mapped, virtually indexed with a cache line of four words (sixteen bytes). This implies that any software routines for initializing, invalidating, or flushing cache must take this difference into account.

TLB

TLB entries

The R4200 will implement a thirty-two way, fully associative TLB in which each entry maps two Page Frame Numbers, one even and one odd. While the structure of this is identical to the R4000, the number of entries is reduced from forty-eight entries on the R4000.

Interactions between ITM & TLB Ops

When a software TLB instruction is accessing the JTLB during the same cycle as an Instruction TLB Miss (ITM) stall, the R4000 behaves in an undefined manner, and can actually generate a bogus TLB Invalid Exception, especially when the TLB software read/write operation (tlbwi, tlbwr, tlbr) is accessing a completely different entry than that accessed on behalf of the Instruction TLB miss.The R4200 has corrected this bogus TLB behavior.

Floating Point Coprocessor

Floating-point Datapath

The R4200 integrates the operation of Coprocessor 1, the floating point coprocessor, into the main pipeline and datapath of the integer processor. This means that the integer pipe will stall for any multicycle floating point instructions. This is invisible to software and has no effect on integer processing.

Variable latencies

Any multicycle instruction which experiences a source exception (i.e. an exception with its source operands) will not cause a stall. Instead, it will issue a default result on that cycle or report a trapped exception on the next cycle, depending on the trap enable flags. In addition, certain trivial calculations (such as 0 * 0) will be performed with less latency than non-trivial cases of the same operation. The R4000, on the other hand, always has the same latency for each particular instruction whether an exception occurs or not.

Cvt.[s,d].l instruction

To convert a 64-bit integer to a single- or double-precision floating point number, R4000 initiates a FP unimplemented operation exception when bits 63...52 of an integer are all Ones or all Zeros.

Since it has a unified 64-bit datapath for both integer and floating-point operations, the R4200 can process more bits out of a 64-bit integer in the long integer to floating-point convert instruction, without added cost to the design. The R4200 raises a FP unimplemented operation exception only when bits 63...55 of a 64-bit integer are all Ones or all Zeros.

RP Bit Effect on *PClock*

On the R4000, setting the RP bit is described as reducing the *PClock* frequency by a factor of two through sixteen, although in fact it is non-functional. On the R4200, this bit will reduce *PClock* frequency by a factor of four.

Pipeline

The R4200 uses a simpler pipeline than the R4000. This pipe is a five stage pipe that strongly resembles the pipeline used on the R3000. The R4200 is not superpipelined. This is not visible in any functional way, but may impact how different code sequences may be optimized. This also means that there are fewer stalls due to the basic pipeline. This increases the CPI component due to cache misses, meaning that the system design of the memory system is even more crucial to the performance of the R4200 processor.

Interrupts

The R4200 dedicates bit 15 of the Cause Register, which is bit 7 of the Interrupt Pending field, to the timer interrupt caused when the Compare and Count registers match. Because of this, the pin *Int**[5] has been removed. Writing to Interrupt register bit 5 via the system interface will have no effect. The R4000 allowed the user to select whether bit 15 was dedicated to the timer interrupt or to Interrupt register bit 5. Interrupt register bit 5 has been eliminated on the R4200.

Kernel Physical Address Segment Organization

The R4200 implements only two cache coherency algorithms, uncached and cacheable non-coherent. This, and the fact that the R4200 implements only 33 physical address bits as compared to the R4000's 36-bit physical addresses, is reflected in the virtual address map in the organization of the Kernel Physical Address Space segment. The R4200 breaks this segment further into two valid address spaces, one per cache coherency algorithm, while the rest of the address space is unavailable. The R4000 processor, in contrast, has more valid addresses in this segment. Refer to Figure 48 for the organization of this virtual address segment for the R4200 processor.

JTAG

When in Shift-IR and Shift-DR modes, IEEE Std1149.1-1990 on page 5-11 in Table 5-2 states that JTDO should be active. The R4200 implements the JTAG controller per IEEE Std1149.1-1990. But, R4000 implements an older version of the standard and does not drive JTDO under these two modes.

CHANGES TO THIS DOCUMENT

Rev. 1.1 (June '93)

On P.17 and P.18, Figure 49 and Table 7 have been updated to show that the ByPassPLL input signal has been defined as active low (now ByPassPLL* or ByPassPLLB).

On P.27, in the packaging information for the 208-pin PQFP package, the following pin-assignments have been corrected following a change in the design:

Pin 129: VCC (was VSS)

Pin 130: VSS (was VCC)

Pin 176: ByPassPLLB (was ByPassPLL)

On P.28, the above changes to the pin-assignments are reflected in the pin-out diagram.

A new page 29 and 30 have been added to show the pin assignments and pin-out diagram for the 179-pin PGA package which has strong compatibility with the R4000PC.

Rev. 1.2 (July '93)

On P.3, Figure 7 was corrected to show Instruction Virtual Address calculation (IVA) commencing on phase 2 of the RF stage.

On P.4, the activity of calculating the virtual branch target address is stated under the RF stage rather than the EX stage.

On P.4, an explanation of hardware interlocking on a load delay slot was included.

On P.4, the priority of pipeline stalls and exceptions was reordered, reflecting the processor design.

On P.7, last line, left-hand column, text was changed to state that instruction fetches are initiated if there is a jump or branch instruction in the EX stage, not the RF stage.

On P.12, the implementation identifier in the PRId register was added.

On P.13, the IB and DB fields of the Configuration register had been described as readable and writable. They are in fact hardwired and are readable only. The reference was removed.

On P.14, Figure 41 was changed to correctly describe the TagLo register fields.

On P.16, Figure 48 was changed to show the correct Kernel Mode Virtual Address Space. The previous figure had shown that of the R4000.

On P.16, Table 2 and 3 were changed to coreect a number of the exception vectors which had been stated incorrectly.

On P.22 in the last line of the page, the word invalidate as partof the term “external invalidate request” was changed to “write” to correctly describe the transaction taking place in Figure 64.

On P.24, Figure 69 and the adjacent text was changed to give a better example of datarate being controlled by the setting of ther DataRate* signal. The diagram now shows a processor block write instead of a block read response.

On P.25, the description of the operation of the phase locked loop was extended to include operation during reduced power mode.

On P.25, the Bootstrap Eception Vector values were corrected.

On P.31, a reference to a new additional document “Designing in common for R4x00 series processors” is included.

On P.32, an optional enhancement to the SysAD protocol is described in accordance with a change in the design.

On P.32, the description of an Instruction Cache Line writeback has been corrected.

Rev . 1.3. (October ‘93)

On P.17, RdRdy* and WrRdy* signals are corrected to show them as inputs to the processor, not outputs.

On P. 17, P.18 and P.25, the BigEndian signal is corrected from “active low (BigEndian*) to normal.

On P.18, P.25, P.27 and P.29, the BypassPLL* signal name spelling is corrected from “ByPassPLL*”.

On P.18, Table 6 has been corrected to read that Rclock leads MasterOut by 90 degrees and Tclock is aligned with MasterOut.

Rev . 1.4. (August ‘94)

On P.4, two new interlock conditions “DCPI” and “CP0I”, and one new exception condition “DCPE” are added to Figure 10 “Pipeline interlocks and exceptions”.

On P.9, the text has been amended to include all seven page sizes in the TLB, not just 4K and 16M bytes as in early versions.

On P.10, Fig. 23, C field, all values besides C=010 now are defined as cached.

On P.10, the last paragraph has been amended to include all seven page sizes in the TLB, not just 4K and 16M bytes as in early versions.

On P.11, Fig. 25, CP0 Mask Register has been amended to include all seven page sizes in the TLB, not just 4K and 16M bytes as in early versions.

On P.12, Fig. 31, a description of the IE field in the Status register has been included.

On P.13, Fig. 36, the description of the Paddr field has been modified.

On P.13, Fig.37, the positions of the R and W fields has been corrected.

On P.17, Table 4 has been updated.

On P.32, a paragraph outlining differences between the TLB in the R4200 and R4000 was eliminated.

Table 10: Load and Store Instruction Summary

MIPS ISA					
LB LBU	rt, offset (rs) rt, offset (rs)	Load Byte Load Byte Unsigned	SB	rt, offset (rs)	Store Byte
LH LHU	rt, offset (rs) rt, offset (rs)	Load Halfword Load Halfword Unsigned	SH	rt, offset (rs)	Store Halfword
LW	rt, offset (rs)	Load Word	SW	rt, offset (rs)	Store Word
LWL	rt, offset (rs)	Load Word Left	SWL	rt, offset (rs)	Store Word Left
LWR	rt, offset (rs)	Load Word Right	SWR	rt, offset (rs)	Store Word Right
MIPS ISA Extensions					
LD	rt, offset (rs)	Load Double	SD	rt, offset (rs)	Store Double-word
LDL	rt, offset (rs)	Load Double Left	SDL	rt, offset (rs)	Store Double-word Left
LDR	rt, offset (rs)	Load Double Right	SDR	rt, offset (rs)	Store Double-word Right
LL	rt, offset (rs)	Load and Link	SC	rt, offset (rs)	Store Conditional
LLD	rt, offset (rs)	Load and Link Double-word	SCD	rt, offset (rs)	Store Double-word Conditional
LWU	rt, offset (rs)	Load Word Unsigned	Sync		Sync

Table 11: Arithmetic Instructions

MIPS ISA					
ADD	rd, rs, rt	Add	ADDI	rt, rs, immediate	Add Immediate
ADDU	rd, rs, rt	Add Unsigned	ADDIU	rt, rs, immediate	Add Immediate unsigned
SUB	rd, rs, rt	Subtract	SUBU	rd, rs, rt	Subtract Unsigned
MULT	rs, rt	Multiply	MULTU	rs, rt	Multiply Unsigned
DIV	rs, rt	Divide	DIVU	rs, rt	Divide Unsigned
MFHI	rd	Move From Multiply Hi Register	MFLO	rd	Move From Multiply Lo Register
MTHI	rd	Move To Multiply Hi Register	MTLO	rd	Move To Multiply Lo Register
AND	rd, rs, rt	Bitwise AND	ANDI	rs, rt, immediate	Bitwise AND Immediate
OR	rd, rs, rt	Bitwise OR	ORI	rs, rt, immediate	Bitwise OR Immediate
XOR	rd, rs, rt	Bitwise Exclusive	XORI	rs, rt, immediate	Bitwise Exclusive Immediate
NOR	rd, rs, rt	Bitwise NOR	LUI	rt, immediate	Load upper Immediate
SLT	rd, rs, rt	Set on Less Than	SLTI	rt, rs, immediate	Set on Less Than Immediate
SLTU	rd, rs, rt	Set on Less Than Unsigned	SLTIU	rt, rs, immediate	Set on Less Than Immediate Unsigned
SLL	rd, rt, sa	Shift Left Logical	SRL	rd, rt, sa	Shift Right Logical
SLLV	rd, rt, rs	Shift Left Logical Variable	SRLV	rd, rt, rs	Shift Right Logical Variable
SRA	rd, rt, sa	Shift Right Arithmetic	SRAV	rd, rt, rs	Shift Right Arithmetic Variable
MIPS ISA Extensions					
DADD	rd, rs, rt	Double Add	DADDI	rt, rs, immediate	Double Add Immediate
DADDU	rd, rs, rt	Double Add Unsigned	DADDIU	rt, rs, immediate	Double Add Immediate unsigned
DSUB	rd, rs, rt	Double Subtract	DSUBU	rd, rs, rt	Double Subtract Unsigned
DMULT	rs, rt	Double Multiply	DMULTU	rs, rt	Double Multiply Unsigned
DDIV	rs, rt	Double Divide	DDIVU	rs, rt	Double Divide Unsigned
DSLL	rd, rt, sa	Double Shift Left Logical	DSRL	rd, rt, sa	Double Shift Right Logical
DSLLV	rd, rt, rs	Double Shift Left Logical Variable	DSRLV	rd, rt, rs	Double Shift Right Logical Variable
DSRA	rd, rt, sa	Double Shift Right Arithmetic	DSRAV	rd, rt, rs	Double Shift Right Arithmetic Variable
DSLL32	rd, rt, sa	Double Shift Left Logical+32	DSRL32	rd, rt, sa	Double Shift Right Logical+32
			DSRA32	rd, rt, sa	Double shift Right Arithmetic+32

Table 12: Jump and Branch Instruction Summary

MIPS ISA					
J	target	Jump	JAL	target	Jump And Link
JR	rs	Jump Register	JALR	rs, rd	Jump And Link Register
BEQ	rs, rt, offset	Branch on Equal	BNE	rs, rt, offset	Branch on Not Equal
BLEZ	rs, offset	Branch on Less than or Equal Zero	BGEZ	rs, offset	Branch on Greater than or Equal Zero
BLTZ	rs, offset	Branch on Less than Zero	BGTZ	rs, offset	Branch on Greater than Zero
BLTZAL	rs, offset	Branch on Less than Zero And Link	BGEZAL	rs, offset	Branch on Greater than or Equal Zero And Link
MIPS ISA Extensions					
BEQL	rs, rt, offset	Branch on Equal Likely	BNEL	rs, rt, offset	Branch on Not Equal Likely
BLEZL	rs, offset	Branch on Less Than or Equal to Zero Likely	BGEZL	rs, offset	Branch on Greater Than or Equal to Zero Likely
BLTZL	rs, offset	Branch on Less Than Zero Likely	BGTZL	rs, offset	Branch on Greater Than Zero Likely
BLTZALL	rs, offset	Branch on Less Than Zero And Link Likely	BGEZL	rs, offset	Branch on Greater Than or Equal to Zero And Link Likely

Table 13: Special Instruction Summary

MIPS ISA			
SYSCALL	System Call	BREAK	Break Point

Table 14: Exception Instruction Summary

MIPS ISA Extensions			
TGE rs, rt	Trap if Greater Than or Equal	TGEI rs, Immediate	Trap if Greater Than or Equal Immediate
TGEU rs, rt	Trap if Greater Than or Equal Unsigned	TGEIU rs, Immediate	Trap if Greater Than or Equal Immediate Unsigned
TLT rs, rt	Trap if Less Than	TLTI rs, Immediate	Trap if Less Than Immediate
TLTU rs, rt	Trap if Less Than Unsigned	TLTIU rs, Immediate	Trap if Less Than Immediate Unsigned
TEQ rs, rt	Trap if Equal	TEQI rs, Immediate	Trap if Equal Immediate
TNE rs, rt	Trap if Not Equal	TNEIU rs, Immediate	Trap if Not Equal Immediate Unsigned

Table 15: Coprocessor Instruction Summary

MIPS ISA			
LWCz rt, offset(rs)	Load Word to Coprocessor z	SWCz rt, offset(rs)	Store Word from Coprocessor z
MTCz rt, rd	Move To Coprocessor z	MFCz rt, rd	Move From Coprocessor z
CTCz rt, rd	Move Control To Coprocessor z	CFCz rt, rd	Move Control From Coprocessor z
BCzT offset	Branch on Coprocessor True	BCzF offset	Branch on Coprocessor False
COPz cofun	Coprocessor z Operation		
MIPS ISA Extensions			
LDCz rt, offset(rs)	Load Double to Coprocessor z	SDCz rt, offset(rs)	Store Double from Coprocessor z
DMTCz rt, rd	Double Move To Coprocessor z	DMFCz rt, rd	Double Move From Coprocessor z
BCzTL offset	Branch on Coprocessor True Likely	BCzFL offset	Branch on Coprocessor False Likely

Table 16: System Control Coprocessor (CP0) Instruction Summary

MIPS ISA			
MTC0 rt, rd	Move to CP0	MFC0 rt, rd	Move From CP0
TLBR	Read Indexed TLB Entry	TLBWI	Write Indexed TLB Entry
TLBP	Probe TLB for Matching Entry	TLBWR	Write Random TLB Entry
ERET	Return from Exception	CACHE sub, offset(rs)	Cache Operation

Table 17: R4200 Cache Op Instruction Summary

MIPS ISA Extensions			
CACHE 0x0, offset(rs)	op = 0:	Index Invalidate; I-Cache	Set the cache state of the cache line to invalid.
CACHE 0x1, offset(rs)	op = 1:	Index Write-back Invalidate; D-Cache	Examines the cache state, if valid dirty, then that block is written back to main memory. Afterward, set the cache state of the cache line to invalid.
CACHE 0x4, offset(rs)	op = 4:	Index Load Tag; I-Cache	Read the content of the tag for the cache line at the specified index and place it into the TagLo and TagHi CP0 registers, ignoring parity errors.
CACHE 0x5, offset(rs)	op = 5:	Index Load Tag; D-Cache	
CACHE 0x8, offset(rs)	op = 8:	Index Store Tag; I-Cache	Write the tag for the cache line at the specified index from with the contents of the TagLo and TagHi CP0 registers.
CACHE 0x9, offset(rs)	op = 9:	Index Store Tag; D-Cache	
CACHE 0xD, offset(rs)	op = D:	Create Dirty Line; D-Cache	This operation is used to avoid loading data needlessly from main memory when writing new contents into an entire cache line. If the cache line does not contain the specified address, and the line is dirty, it will be written back to main memory. In all cases, set the cache line tag to the specified physical address, and set the cache state to valid dirty.
CACHE 0x10, offset(rs)	op = 10:	Hit Invalidate; I-Cache	If the cache line contains the specified address, mark the cache line invalid.
CACHE 0x11, offset(rs)	op = 11:	Hit Invalidate; D-Cache	
CACHE 0x14, offset(rs)	op = 14:	Fill; I-Cache	Fill the instruction cache line from main memory.
CACHE 0x15, offset(rs)	op = 15:	Hit Write-back Invalidate; D-Cache	If the cache line contains the specified address, and it is dirty, the data will be written back to main memory. Then, then cache line is set to invalid.
CACHE 0x18, offset(rs)	op = 18:	Hit Write-back; I-Cache	If the cache line contains the specified address, and it is dirty, write back the data to memory, and clear the dirty bit. For instruction cache, the cache line will be written back to main memory unconditionally.
CACHE 0x19, offset(rs)	op = 19:	Hit Write-back; D-Cache	