I6500-F Multiprocessing System Datasheet

March 17, 2025

The MIPS® I6500-F Multiprocessing System (MPS) provides a highly scalable foundation for building the many-core designs needed to handle the compute-intensive tasks in emerging safety-critical systems, such as autonomous vehicles, Industrial Internet of Things (IIoT), and robotics. The I6500-F MPS scales to 64 heterogeneous clusters of multi-threaded multi-core MIPS CPUs, and through the MIPS Coherence Manager with AMBA® ACE interface, enables integration with heterogeneous CPU clusters and other accelerators at the system-on-chip (SoC) level.

As such, the I6500-F MPS is a high performance multi-core microprocessor system that provides a best in class power efficiency for use in SoC applications. Each I6500-F CPU core combines multithreading and an efficient dual-issue pipeline to deliver outstanding computational throughput. The I6500-F Coherence Manager (CM) maintains Level 2 (L2) cache and system level coherency between all cores, main memory, and I/O devices. The I6500-F MPS is a configurable and a synthesizable solution. The collection of clusters of cores can be configured with a variable number of cores, I/O coherent interfaces, and L2 cache size. Each of the cores can be configured with Level 1 (L1) cache sizes, number of threads, and single instruction multiple data (SIMD) functionality.

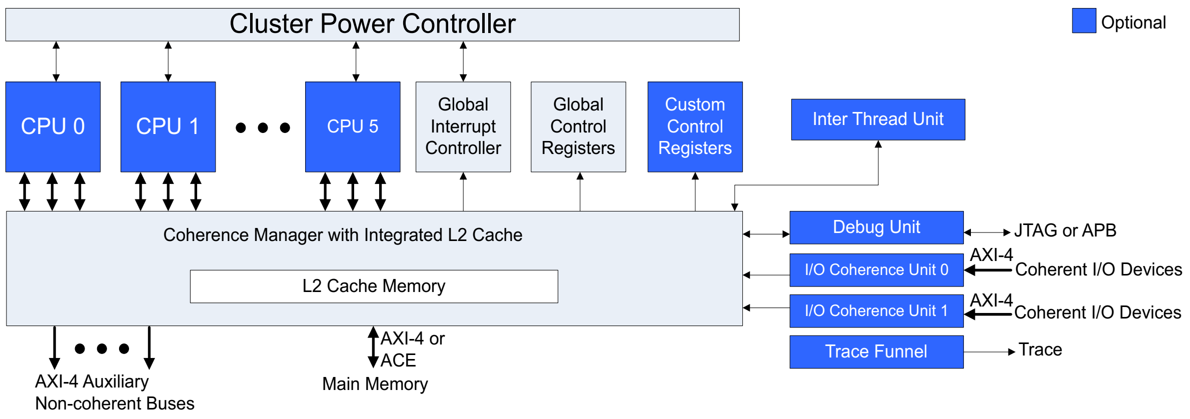
Each I6500-F core implements the Release 6 of the MIPS64 Instruction Set Architecture (ISA) with full hardware multithreading and hardware virtualization support. In addition, the core can be configured with a SIMD engine supporting integer, single and double precision, and floating and fixed point operations.

Highlights of the I6500-F MPS include:

- Multi-Cluster support
- Up to 6 CPU cores per cluster
- PDtrace support
- Coherence Manager (CM3.5) with integrated L2-cache:
- Up to 8 I/O Coherence Units (total of cores + IOCUs must be no greater than 8)
  - Cluster Power Controller (CPC)
  - Global Interrupt Controller (GIC)
  - Global Configuration Registers (GCR)
  - Multiprocessor debug via in-system Debug Unit (DBU)
  - Trace Funnel (TRF)
  - Cluster Inter-thread communication unit (ITU)

Figure 1 shows a block diagram of a single cluster I6500-F Multiprocessing System (MPS).

Figure 1. Block Diagram of Single Cluster I6500-F Multiprocessing System



I6500-F Features

The I6500-F MPS implements the current MIPS64 architecture, including new CPU and system-level features designed for performance, power, and area form factors. The MPS flexibility and features are well suited for a broad range of markets and applications, such as embedded systems, automotive, consumer/mobile, and enterprise class storage, server, and data-plane solutions.

MIPS Architecture

The I6500-F Multiprocessing System has four key architectural features as described in the following subsections.

- MIPS64® Release 6 Architecture
- MIPS SIMD Architecture
- MIPS Virtualization
- MIPS Multithreading
- Coprocessor 0 Privileged Register Set
- Functional Safety

MIPS64® Release 6 Architecture

The MIPS64 architecture incorporates powerful features, standardizing privileged mode instructions, and supporting past ISAs. It also provides a seamless upgrade path from the MIPS32 architecture. MIPS64 is based on a fixed-length, regularly encoded instruction set, and it uses a load/store data model. It is streamlined to support optimized execution of high-level languages.

The MIPS64 Release 6 ISA also supports both compact and delayed branches. This helps the compiler generate dense code while still maintaining backward compatibility. Availability of 31 general-purpose registers enables compilers to further optimize code generation by keeping frequently accessed data in registers.

The MIPS64 Release 6 ISA provides memory management through on-chip configuration registers and enables real-time operating systems and application code to be implemented once and then reused.

MIPS® SIMD Architecture

SIMD (Single Instruction Multiple Data) is an important technology for modern CPU designs because it improves performance by allowing efficient parallel processing of vector operations. The MIPS® SIMD Architecture (MSA) technology incorporates a software-programmable solution into the CPU to handle emerging Coder/Decoders (Codecs) or potentially eliminate dedicated hardware functions in some cases. This programmable solution allows for increased system flexibility. In addition, the MSA is designed to accelerate many compute-intensive applications by enabling generic compiler support, which can automatically vectorize code to enhance performance.

MIPS® Virtualization

To address security, privacy and reliability concerns in a wide range of devices, MIPS has added hardware supported virtualization technology into the I6500-F core. The hardware virtualization support ensures that applications that need to be secure are effectively and reliably isolated from each other, as well as protected from non-secure applications.

MIPS Multithreading

CPU performance depends on minimizing the latency to the system memory. Even with a cache hierarchy, the CPU still stalls while waiting for data. To avoid this scenario, MIPS multithreading provides significant performance improvements by running additional threads concurrently.

This hardware multithreading enables execution of multiple instructions from multiple threads every clock cycle, providing higher utilization and CPU efficiency. In this way, multi-threading is a more area efficient alternative to the use of additional cores and offers a typical 40% performance boost for the execution of two threads simultaneously instead of sequentially.

System Control Coprocessor (CP0) Architecture

In the MIPS architecture, the Coprocessor 0 (CP0) register set implements the Privileged Resource Architecture (PRA), which includes:

- System configuration registers
- Virtual to physical address translation (MMU)
- Exception control system (including interrupt control)
- Processor’s diagnostic capability
- Operating modes (kernel, user, supervisor, and debug)

Configuration information, such as cache size and associativity, and the presence of optional features like a floating point unit, are also available by accessing the CP0 registers. CP0 also contains the state used for identifying and managing exceptions. Exceptions can be caused by a variety of sources, including boundary cases in data, external events, or program errors. Refer to the *MIPS64 Release 6 ISA Specification* for further details.

Functional Safety

The I6500-F IP is designed to support the ASIL-B(D) functional safety standard. In so doing, the I6500-F cluster includes the following fault detection features:

- Fault bus to report detected faults to external fault handling logic
- End-to-end parity protection on address and data buses
- Parity protection of software visible registers in the GCR, GIC, CPC, and ITU blocks
- Programmable transaction time-out detection on memory requests originating from a CPU or IOCU
- SRAM error detection and correction
- Protocol error detection on IOCU and REGTC AXI slave interfaces
- AXI/ACE interface parity protection of address and data compatible with third-party interconnects

System-level Features

- Up to six coherent MIPS64 Release 6 CPU cores
- Multi-Cluster support: Cluster composed of up to 0 - 6 CPUs and 0 - 2 IOCUs (sum being no more than 8 agents) and a Level 2 cache connection to a coherent interconnect. Support for up to 4 clusters.
- Integrated L2 cache controller supporting a 8-way and 16-way set-associativity
  - Inclusive of the L1 data caches
  - 256 KB to 8 MB cache sizes
  - Single bit correction and double bit detection
- CPC to shut down idle cores for power efficiency
- Up to 8 I/O Coherence Units (total of cores + IOCUs must be no greater than 8)
- Virtualization Module Support
- Cache-to-cache data transfers
- Out-of-order data return
- Hardware L2 cache prefetch controller significantly improves performance of workloads such as memory to memory data transfer/copy (memcpy)
- Independent clock ratios on core, memory, and IOCU ports
- SoC system interface supports AXI-4 (Advanced eXtensible Interface rev. 4, also known as AMBA 4 AXI) or ACE (AXI Coherency Extensions) protocol with 48-bit address and 256-bit data paths. This interface can be configured to support up to 96 outstanding requests.
- High bandwidth 128-bit data paths between each core and the Coherence Manager
- Software controlled core level and cluster level power management
- Debug port supporting multi-core debug (JTAG/APB)
- Program and Data trace (PDtrace) mechanism to debug software

CPU Core-Level Features

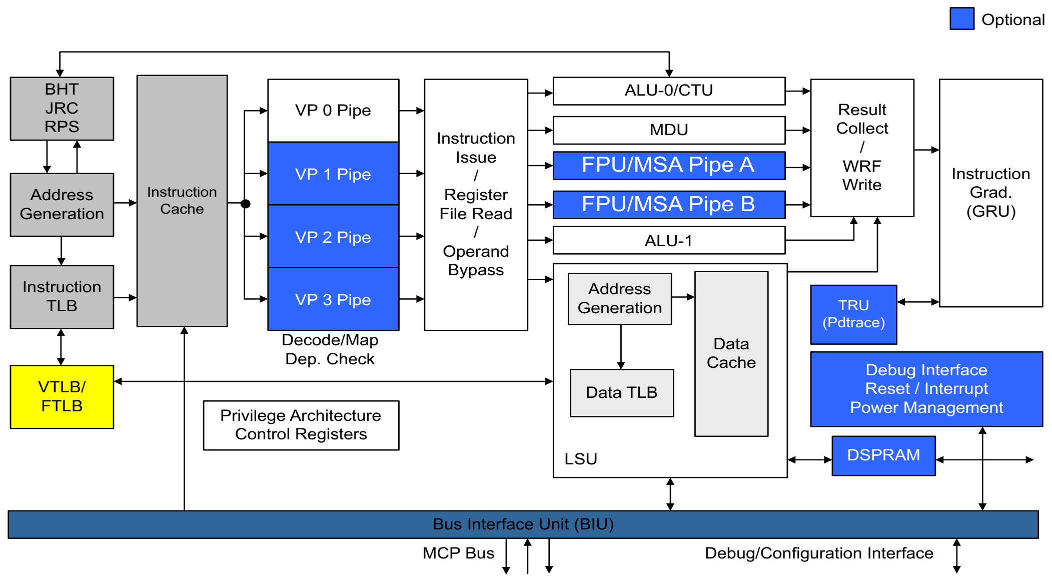
- Full 64-bit Instruction Set Architecture via MIPS64 Release 6
- 48-bit virtual and physical addresses
- Power efficient design
- Dual issue instruction fetch, decode, issue, and graduate
  - Hardware multithreading

- Virtualization support
- L1 caches with Error Correction Code (ECC) protection
- L2 cache support — Implemented as shared L2 in the Coherence Manager
- Programmable Memory Management Unit with large first-level ITLB/DTLB backed by fast on-core second-level variable page size TLB (VTLB) and fixed page size TLB (FTLB)
  - Shared FTLB across all virtual processors (VPs) in a CPU
  - MIPS DVM support through Global Instruction cache and TLB invalidation
- Load and store bonding support
- Unaligned load / store support in hardware
- Program and Data Trace (PDtrace) support for Instructions and Data (Virtual Addresses and Data Values)
- Optional Data Scratch Pad RAM (DSPRAM)
- Optional Inter-Thread Communication Unit (ITU)

I6500-F CPU Core Features

Figure 2 shows a block diagram of a single I6500-F core. The logic blocks in this diagram are described in the following sections.

Figure 2. I6500-F Core Block Diagram



For more information on the I6500-F core in a multiprocessing environment, refer to "Multiprocessing System Features" on page 15.

Instruction Fetch Unit (IFU)

The Instruction Fetch Unit (IFU) fetches instructions from the L1 instruction cache and supplies them to the Execution Unit (EXU). The IFU can fetch up to two instructions at a time from the L1 cache and fill the instruction buffers, which decouple the instruction fetch unit from the issue and execution of the instructions.

Branch Prediction

The IFU employs sophisticated branch prediction that anticipates the branch direction to improve performance and efficiency. The prediction is based on both local and global history of the branch captured in the Branch History Table (BHT) with majority voting. The predictor adapts to the program by self learning. The prediction stops on certain types of instructions, giving software control of the code execution.

Jump Prediction

The IFU has a hardware-based Jump Register Cache (JRC) and Return Prediction Stack (RPS) to predict jump target addresses. This results in faster throughput during subroutine calls and returns.

Level 1 Instruction Cache

The I6500-F L1 instruction cache is configurable as 32 KB or 64 KB in size and is organized as 4-way set-associative. The instruction cache is virtually indexed and physically tagged to allow data accesses and virtual-to-physical address translation to occur in parallel.

This cache is used to fetch two instructions per cycle. To conserve power, a way-prediction mechanism enables only the expected way. The cache is protected by single- and double-bit error detection logic.

Each cache line holds 64 bytes of instructions and the coherency of the cache is maintained by software with hardware assistance.

Execution Unit (EXU)

In the I6500-F core, the Execution Unit (EXU) implements the logic for:

- Instruction Buffer Management
- Instruction Selection and Issue
- Source Operand Read and Bypass
- Integer Execution Units
- FPU / MIPS® SIMD Architecture (MSA) Execution Units
- Result Collection & Instruction Graduation

Instruction Buffer Management and Issue

The fetch unit delivers up to two instructions per cycle to the EXU. The EXU keeps these instructions in a deep instruction buffer. The EXU maintains a separate instruction buffer for each thread (VP).

Up to two instructions may be issued for execution during a clock cycle. The instructions can be issued from the same thread or from different threads. A round-robin priority scheme is used to arbitrate among threads.

Instructions can be concurrently issued to any two of the following EXU functional units:

- 2 Integer Units
- 1 Multiply / Divide Unit
- 1 Branch Unit
- 1 Load Store Unit
- 1 Short Floating Point Pipe
- 1 Long Floating Point Pipe

Source Operand Read and Bypass

The EXU can simultaneously read source operands from the Architectural Register File (ARF) or Working Register File (WRF) for each of the instructions (regardless of thread context). In addition, the EXU implements a fully symmetric operand bypass network to bypass a result from a preceding execution stage.

Integer Execution Units

The EXU has two complete ALUs that perform single-cycle operations including add, subtract, shifts, rotates, bit-wise logical, and several other operations. One of the ALUs assists in resolving conditional branches.

The EXU also contains a dedicated 64x64 integer multiplier and radix 4 SRT divider to speed up compute intensive applications and implements cyclic redundancy code (CRC and its variants) in hardware.

Floating Point / MSA Pipelines

The I6500-F core features an optional 128-bit SIMD engine that implements the MIPS SIMD Architecture (MSA). The engine handles scalar floating point as well as SIMD integer and floating point data types. Floating point operations are IEEE 754-2008 compliant.

The EXU implements two separate pipelines (1 short, 1 long) to execute both floating point and MSA instructions. These two pipelines allow the execution of simple floating point instructions to bypass and execute in parallel with less frequently used complex and iterative instructions. One pipeline executes SIMD logical ops, SIMD integer adds, and FP compares and FP/SIMD stores. The other pipeline executes SIMD integer multiplies, SIMD vector shuffles, FP adds, FP multiplies, and FP divides.

The SIMD unit contains thirty-two 128-bit vector registers shared between SIMD and FPU instructions. Single-precision floating-point instructions use the lower 32 bits of the 128-bit register. Double-precision floating point instructions use the lower 64 bits of the 128-bit register. SIMD instructions use the entire 128-bit register interpreted as multiple vector elements: 16 x 8-bit, 8 x 16-bit, 4 x 32-bit, or 2 x 64 bit vector elements.

SIMD instructions enable:

- Efficient vector parallel arithmetic operations on integer, fixed-point, and floating-point data
- Operations on absolute value operands
- Rounding and saturation options
- Full precision multiply and multiply-add
- Conversions between integer, floating-point, and fixed-point data
- Complete set of vector-level compare and branch instructions with no condition flag
- Vector (1D) and array (2D) shuffle operations
- Typed load and store instructions for endian-independent operation

The SIMD unit is fully synthesizable and operates at the same clock speed as the core.

The exception model is ‘precise’ at all times.

The SIMD unit supports fused floating point multiply-adds as defined by the IEEE Standard for Floating-Point Arithmetic 754-2008. Most FPU and SIMD instructions have one cycle throughput. All floating point denormalized input operands and results are fully supported in hardware.

Result Collection and Graduation

The EXU collects all results from single-cycle, fixed-latency, and variable-latency instructions and pairs them up with associated completion status (such as exceptions and interrupts), and commits the results into the Architectural Register File (ARF). This committing of final results is called the *graduation* of the instruction.

Load Store Unit (LSU)

The Load Store Unit (LSU) moves data between the core and the system memory. It also maintains an L1 data cache to accelerate access to commonly used data by the core. The LSU accepts a single operation per cycle and maintains several buffers to keep the data moving between the EXU and L1 cache and between the L1 cache and the Bus Interface Unit (BIU) at optimal rate.

Level 1 Data Cache

The I6500-F L1 data cache is configurable as 32 KB or 64 KB in size and is organized as 4-way set-associative. The data cache is physically indexed and physically tagged to avoid virtual aliasing.

The L1 data cache is capable of fetching data on both aligned and unaligned memory accesses. In addition, it can combine multiple loads and stores into a single operation using a feature called “instruction bonding” to maximize memory bandwidth.

To conserve power, a way-prediction mechanism enables only the expected way. The cache is protected by single-bit error correction and double-bit error detection logic.

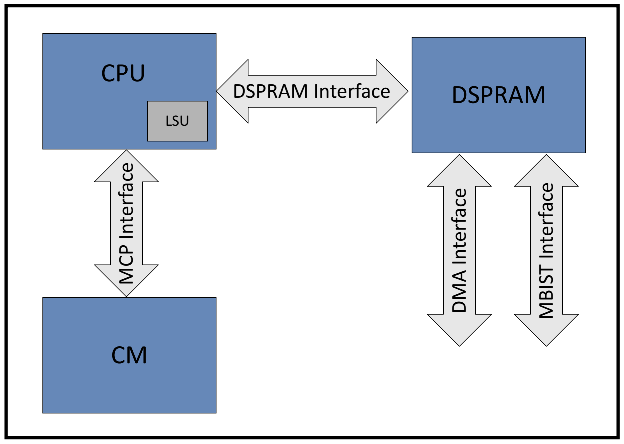
Each cache line holds 64 bytes of data as well as the associated tag and replacement information.

DSPRAM Interface

The I6500-F data scratchpad RAM (DSPRAM) interface provides a connection to on-chip memory or memory mapped registers, which are accessed in parallel to the L1 data cache to minimize access latency. The DSPRAM interface connects the CPU to an external user designed DSPRAM module (a reference design is provided with the I6500-F CPU).

Figure 3 shows a block diagram of the I6500-F DSPRAM and interface.

Figure 3. I6500-F DSPRAM Block Diagram



Features:

- 16-Byte wide data path for both read and write.
- Data can be protected (parity/ECC/none) on 32 bit granularity in the DSPRAM.
- Multi-threaded design, if one thread is blocked the other threads may continue to access the DSPRAM.

Store and Write Buffer

The LSU contains store buffers that decouple the main pipeline from the memory subsystem, allowing the LSU to efficiently schedule cache writes and coherence operations while the main pipeline continues to execute subsequent instructions. After a store instruction graduates in the main pipeline, the LSU takes control and forwards the store data from the store buffer to subsequent load instructions until the data is committed to the cache or main memory.

The store buffers can merge multiple cacheable stores into a single larger write operation, which can take advantage of the 512-bit cache write data-path. This store buffer improves performance by avoiding contention at the cache RAM ports and saves power by reducing the number of RAM accesses. When data from the cache is written back to main memory, an entire cache line is transferred from the cache RAM to the evict buffer in the BIU in a single clock cycle. This frees up the LSU cache pipeline to proceed with subsequent operations, while the BIU streams the write-back data to the CM3.5 as a burst write transaction.

The store buffer also merges multiple uncached-accelerated stores into a single burst-write transaction, to increase the efficiency of the bus and avoid stalling the main pipeline. Gathering of uncached accelerated stores can start on any arbitrary address and can be combined in any order within a 64-byte aligned block of memory.

Memory Management Unit (MMU)

The Memory Management Unit (MMU) translates virtual addresses to physical addresses and provides attribute information for different segments of memory. The I6500-F MMU contains the following Translation Lookaside Buffer (TLB) structures:

- Instruction TLB (ITLB)
- Data TLB (DTLB)

- Variable Page Size Translation Lookaside Buffer (VTLB) per VP
- Fixed Page Size Translation Lookaside Buffer (FTLB) per core

Instruction and Data TLB (ITLB and DTLB)

The ITLB and DTLB (micro TLBs) are fully associative. The micro TLBs are used by the IFU and LSU to perform high speed virtual to physical memory address translation for instruction fetch and data movements respectively.

The ITLB is implemented in the IFU with support for 4 KB, 16 KB, or 64 KB page sizes per entry. The DTLB is implemented in the LSU with support for 4 KB, 16 KB, or 64 KB page sizes per entry. The micro TLB arrays are shared between VPs.

The number of entries varies with the number of VPs present, as listed in Table 1.

Table 1. Entries per VP

TLB Type	VPs	Entries
ITLB	1	6
	2	12
	4	18
DTLB	1	8
	2	14
	4	20

The micro TLBs are managed completely by hardware and are transparent to the software. The micro TLBs are backed up by larger VTLB and FTLB structures. If a virtual address cannot be translated by the micro TLB, the VTLB / FTLB attempts to translate the address in the following clock cycle or when available. If successful, the translation information is copied into the appropriate micro TLB for future use. When Virtualization is in use, the micro TLBs store the full two-level translation from the Guest Virtual Address to Root Physical Address to maintain high performance.

Variable Page Size TLB (VTLB)

The VTLB is a fully associative translation lookaside buffer with 16, 32, or 64 dual-entries per thread that can map variable page sizes from 4 KB to 1 GB.

Fixed Page Size TLB (FTLB)

The FTLB contains 512 dual entries organized as 128 sets and 4-way set-associative. The FTLB page size is configurable at run-time to either 4 KB, 16 KB, or 64 KB.

Fixed TLB translations are shared for all VPs with the same GID (Guest ID) + MMID (Memory Map ID) when the MMID is enabled. Using the 16-bit MMID creates a global address space, allowing the MMU translations to be shared across VPs on a core and global invalidates to be performed across cores. Optionally, the legacy 10-bit ASID can still be used, in which case FTLB translations are not shared across the VPs.

Virtualization Support

The Virtualization Module is a set of extensions to the MIPS64 Architecture for efficient implementation of virtualized systems. This feature provides privileged (root) and unprivileged (guest) operating modes. It supports up to 31 guests.

The guest mode can be enabled by software. The key element is a control program known as a Virtual Machine Monitor (VMM) or Hypervisor. The Hypervisor is in full control of machine resources at all times.

When an operating system (OS) kernel runs within a virtual machine (VM), it becomes a “guest” of the Hypervisor. All operations performed by a guest must be explicitly permitted by the Hypervisor. To ensure that it remains in control, the Hypervisor always runs at a higher level of privilege than a guest operating system kernel.

The Hypervisor manages access to sensitive resources, maintains the expected behavior for each VM, and shares resources between multiple VMs.

In a traditional operating system, the kernel (or supervisor) runs at a higher level of privilege than user applications. The kernel provides a protected virtual-memory environment for each user application, inter-process communications, I/O device sharing, and transparent context switching. The Hypervisor performs these same basic functions in a virtualized system, except that the Hypervisor’s clients are full operating systems rather than user applications.

The virtual machine execution environment created and managed by the Hypervisor consists of the full Instruction Set Architecture (ISA), including all Privileged Resource Architecture (PRA) facilities, and any device-specific or board-specific peripherals and associated registers. It appears to each guest operating system as if it is running on a real machine with full and exclusive control.

Bus Interface (BIU)

The BIU interfaces the instruction and data caches with the CM3.5. This interface implements MIPS Coherence Protocol (MCP) and has three channels that support 128-bit data transfers. The transaction size can vary from 1 byte to 16 bytes for single uncached access or the full 64 bytes for a cache line. The BIU supports full memory coherency, including interventions.

Interrupt Handling

Each I6500-F core supports six hardware interrupts including a timer interrupt and a performance counter interrupt. In addition, it support two software interrupts. These interrupts can be used in any of three interrupt modes, as defined by the MIPS64 Architecture:

- *Interrupt compatibility* mode.
- *Vectored Interrupt (VI)* mode adds the ability to prioritize and vector interrupts to a handler dedicated to that interrupt.
- *External Interrupt Controller (EIC)* mode provides support for an external interrupt controller that handles prioritization and vectoring of interrupts.

Operating Modes

The I6500-F core supports seven modes of operation:

- Two user modes (guest and root) are used for application programs.
- Two supervisor modes (guest and root)
- Two kernel modes (guest and root) are used to handle exceptions and operate system kernel functions, including CP0 management and I/O device accesses.
- Debug mode is used during system bring-up and software development. Refer to Section ""Core Debug Support" on page 14" for more information on debug mode.

I6500-F Core Power Management

The I6500-F core offers several power-management features. It supports low-power design, such as active power management and power-down modes of operation. The I6500-F core is a static design that supports slowing or halting the clocks to reduce system power consumption during idle periods.

Instruction-Controlled Power Management

The Instruction Controlled power-down mode is invoked through execution of the WAIT instruction.

The WAIT instruction puts the processor in a quiescent mode where no instructions are running. When the WAIT instruction is seen by the Instruction Fetch Unit (IFU), subsequent instruction fetches are stopped. However, the internal timer and some of the input pins continue to run. Any interrupt, NMI, or reset condition causes the CPU to exit this mode and resume normal operation.

Core Debug Support

The I6500-F core includes a debug block available for use in software debugging of application and kernel code. For this purpose, in addition to standard user, supervisor, and kernel modes of operation, the I6500-F core provides a Debug mode.

Debug mode is entered when a debug exception occurs and continues until a debug exception return instruction is executed or the CPU is reset. The Debug features include:

- Up to 8 instruction breakpoints
- Up to 4 data breakpoints
- Single-step execution
- Memory and register access
- Program and data trace (PDtrace)

Multiprocessing System Features

The I6500-F Multiprocessing System (MPS) provides multi-cluster support where each cluster consists of up to six I6500-F cores, a Coherence Manager (CM3.5) with integrated L2 cache, up to eight IOCUs, a cluster power controller (CPC), global interrupt controller (GIC), debug unit (DBU), and global configuration registers (GCR). The CM3.5 maintains coherence with the cores' L1 caches by implementing a directory-based coherence protocol that enables both low power and high performance.

The I6500-F extends capability from a single coherent six-core cluster with support I/O coherency to a new set of capabilities that enable more complex systems, such as:

- Multiple coherent clusters of CPUs
- Heterogeneous Multi-processing (CPU + GPU or other coherently designed processing elements)
- Groups of coherent I/O or co-processing functions or clusters

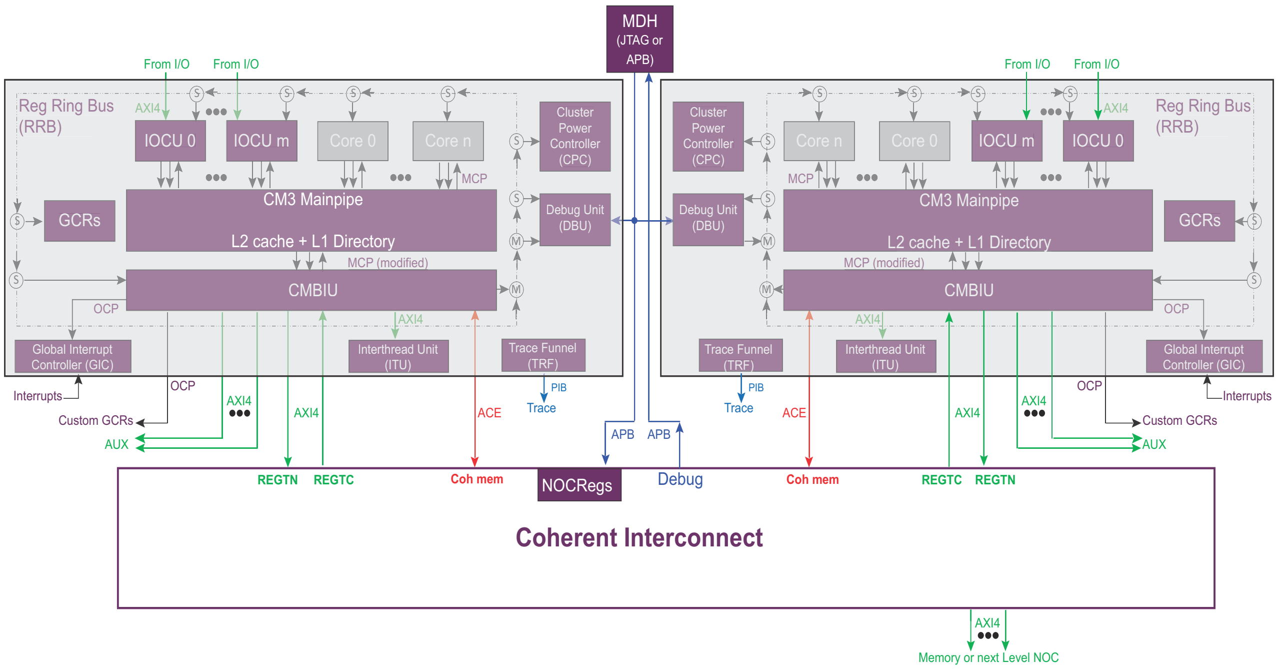
A cluster is composed of up to 0 - 6 CPUs and 0 - 8 IOCUs (sum being no more than 8 agents) and a Level 2 cache connection to a coherent interconnect. An agent is either a CPU, which is included in the cluster, or an external I/O device. The initial I6500-F implementation support is 2 - 4 clusters.

The I6500-F cluster can be configured in one of two modes:

1. It can be configured as a single non-coherent cluster, similar to the I6400. In this case the main memory bus interface from the cluster is AXI-4 (same as the I6400).
2. It can be configured to support multiple coherent clusters. In this case, the main memory bus interface is ACE.

Figure 4 shows a reference design of a cluster integrated with a network.

Figure 4. I6500-F Integrated Cluster with Network



Directory Based Level 1 Cache Coherence

The Coherence Manager (CM3.5) keeps all the L1 data caches coherent with each other by maintaining a directory that tracks the state of each L1 data cache line for each core. The directory uses the same address tags as the Level 2 cache, reducing the power and area required to maintain coherence. All Level 1 data and instruction cache misses are looked up in the directory to determine the state of the line in the L1 data caches as well as the L2 cache. Depending on the request attributes and directory state, the CM3.5 sends intervention requests to cores that have the line in their L1 data cache, reads the data from the L2 Data RAMs, or issues a request to the memory subsystem. The CM3.5 immediately updates the directory state and routes the corresponding data to the requesting core.

With a directory-based coherence architecture, each of the cores do not need to maintain a second copy of the L1 cache tags to “watch” the memory transactions and compare them against its internal cache contents. Instead, that information is maintained by the directory, which shares the L2 address tags.

L1 Instruction Cache Coherence

The Level 1 instruction caches are not coherent, in that the CM3.5 directory does not track their contents. However, L1 instruction cache misses will be looked-up in the CM3.5 directory, and depending on the state, may receive its data from a core’s L1 data cache. This feature reduces the overhead of the software required to maintain L1 instruction cache coherence.

CM3.5 Main Pipeline

The CM3.5 Main Pipeline manages all the data and control flows throughout the CM3.5 and the I6500-F Multiprocessing System.

The main pipeline implements the directory-based coherence architecture and manages a unified and shared L2 cache. Some key features of the L2 cache are:

- 64-byte cache line size
- 8- or 16-way set-associative
- 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, and 8 MB cache-size options
- 1 or 2 cycle tag RAM access
- 2 or 4 cycle data RAM access
- 1 or 2 L2 cache pipelines, each with two memory banks
- Pseudo LRU line-replacement algorithm
- Writeback architecture
- L2 is inclusive of the L1 data caches, that is, it is always a superset of all L1 Data Caches
- Physically Indexed and Physically Tagged
- Non-Blocking architecture (Fully Pipelined)
- 48-Bit Physical Address
- L2 Hardware Prefetcher automatically recognizes workloads, such as `memcopy`, and efficiently prefetches data into the L2 cache
- Hardware can automatically initialize L2 cache upon reset. Hardware can also be programmed to initialize/flush all or part of the L2 cache.
- Cache line locking support
- ECC support (single-bit error correction and double-bit error detection) for Tag and Data arrays
- Parity support on data buses

The CM3.5 main pipeline arbitrates among the requests received from the cores, IOCUs, and L2 hardware prefetcher. It accesses and updates the directory and L2 cache tags, performs reads or writes to the L2 data RAMs as necessary, and issues interventions to manage each core's L1 data caches.

Uncached requests are also handled by the CM3.5 main pipeline, but neither the directory nor L2 cache is accessed. Uncached accesses are decoded based on a programmable address map and routed to the CM's Bus Interface Unit (CMBIU). The programmable address map determines the final target of the request, such as uncached memory or a configuration register in the interrupt controller, power controller, etc.

The CM3.5 main pipeline identifies and resolves conflicting accesses as required.

The CM3.5 includes high performance features for data movement:

- 512-bit wide internal data paths throughout the CM3.5
- Three channel (two of 128-bit wide) system MCP interface to each of the CPU cores and IOCUs
- When configured as multi-cluster, ACE interface to inter-cluster network;
  AXI4 interface when configured as single cluster
- Support for up to 4 non-coherent Auxiliary AXI4 ports

Cluster Power Controller (CPC)

Individual CPUs within the cluster can have their clock, power, or both gated off when they are not in use. This gating is managed by the Cluster Power Controller (CPC). The CPC handles the power shutdown and ramp-up of all cores in the cluster. The CPC can be controlled via software by accessing and changing values in the registers and by hardware through a signal interface.

The CPC also organizes power-cycling of the CM3.5, dependent on the individual core status and shutdown policy. Reset and root-level clock gating of individual CPUs are considered part of this sequencing.

The CPC also controls the clock ratios of the cores, CM3.5, I/O buses, and main memory bus. The CPC allows for the clock ratio of each component to be controlled independently, programmed by means of software commands or hardware signals. The clock ratio can be changed dynamically while the system is fully operating.

Reset Control

The reset input of the system resets the Cluster Power Controller (CPC). Reset sideband signals are required to qualify a reset as system cold, or warm start. Signal settings determine the course of action at deassertion of reset:

- Remain powered down
- Go into clock-off mode
- Power-up and start execution

In case of a system cold start and after reset is released, the CPC powers up the I6500-F CPUs as directed in the CPC cold start configuration. If at least one CPU has been chosen to be powered up on system cold start, the CM3.5 is also powered up.

At a warm start reset, the CPC brings all power domains into their cold start configuration. To ensure power integrity for all domains, the CPC ensures that domain isolation is raised before power is gated off. Domains that were previously powered and are configured to power up at cold start remain powered and go through a reset sequence.

The CM includes memory-mapped registers that can override the default exception vector location. This allows different boot vectors to be specified for each of the VPs, so they can execute unique code if required. Furthermore, signals by the system also determine which VPs on each core start execution up. The CPC implements the capability to bring a core out of reset with no VPs running, letting the system hardware start one or more VPs at a later time.

I/O Coherence Unit (IOCU)

Hardware I/O coherence is provided by the I/O Coherence Unit (IOCU), which maintains I/O coherence of the caches in all coherent CPUs in the cluster.

The IOCU acts as an interface block between the Coherence Manager (CM3.5) and I/O devices. Reads and writes from I/O devices may access the L1 and L2 caches by passing through the IOCU and the CM3.5. Each request from an I/O device may be marked as coherent, or uncached. Coherent requests access the L1 and L2 caches. Uncached requests bypass both the L1 and L2 caches and are routed to main memory.

The IOCU provides an AXI slave interface to the I/O interconnect for I/O devices to read and write system memory.

The IOCU provides several features for easier integration:

- Supports incremental bursts up to 256 beats (128 bits per beat) on I/O side. These requests are split into cache-line sized requests on the CM3.5 side
- Coherent writes are issued to the CM3.5 in the order they were received

Global Interrupt Controller (GIC)

The Global Interrupt Controller handles the distribution of interrupts between and among the CPUs in the cluster. This block has the following features:

- Software interface through relocatable memory-mapped address range
- Configurable number of system interrupts from 8 to 256 in multiples of 8
- Support for different interrupt types:
  - Level-sensitive: active high or low
  - Edge-sensitive: positive-, negative-, or double-edge sensitive
- Ability to mask and control routing of interrupts to a particular CPU
- Support for NMI routing
- Standardized mechanism for sending inter-processor interrupts
- Support for Virtualization of interrupts: each interrupt to be mapped to Guest or Root

Global Configuration Registers (GCR)

The Global Configuration Registers (GCR) are a set of memory-mapped registers that are used to configure and control various aspects of the Coherence Manager and the coherence scheme.

Some of the control options include:

- *Address map* — The base address for the various peripheral blocks, such as the CPC, GCR, User GCRs, and GIC address ranges can be specified
- *Error reporting and control* — Logs information about errors detected by the CM3.5 and controls how errors are handled (ignored, interrupt, etc.)
- *Control Options* — Various features of the CM3.5 can be disabled or configured
- *L2 Cache operations* — Registers used during L2 cache maintenance instructions
- *Mapping registers* — Route requests to one of the non-coherent Auxiliary (AUX) AXI-4 ports
- *Multi-cluster register access* — Allows a CPU of one cluster to access a register on a remote cluster via the REGTC/REGTN AXI4 buses

Custom GCRs

The CM3.5 provides the ability to implement a 64 KB block of custom registers that can be used to control system level functions. These registers are user defined and then instantiated into the design. Two global registers are provided by the CM3.5 to implementation custom registers: the Global Custom Base register, and the Global Custom Status register.

Inter-thread Communication Unit (ITU)

The I6500-F MPS includes an integrated cluster ITU, which provides a gating storage capability for synchronization between threads on all I6500-F CPUs.

The I6500-F ITU includes the following features:

- Memory Mapped ITU Control Registers (ICR) within the ITU Addressable Region
- Entry Cell storage: 64-bit Double-word (Dword)
- Multi Entry Cell storage: 64-bit wide FIFO with build time configurable depth of 2 - 8

Clocking Options

The I6500-F Multiprocessing System has the following clock domains:

- *Reference Clock* — This clock is created by the SOC and used by the I6500-F Multiprocessing System. The reference clock is controlled and scaled by the input clock. This clock drives the CPC.
- *Prescaled clock* — The reference clock can be prescaled by a programmable value of 1:1 (no prescale) to 1:255. This prescaled clock is used a base clock for all cluster components, except the CPC.
- *Cluster clock domain* — This clock drives the CM3.5 (including Coherence Manager, Global Interrupt Controller, IOCU, and L2 cache). This clock can be configured to be the same as Prescale Clock or Prescale / 2.
- *Core-N clock domain* — Each core in the cluster can operate at independent frequency. This clock can be controlled at run time (via CPC).
  - When the CM3.5 is operating at 1:1, the cores can run at 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7 or 1:8 of the prescale clock.
  - When the CM3.5 is operating at 1:2, the cores can run at 1:1, 1:2, 1:4, 1:6, or 1:8 of the prescale clock.
- *System clock domain* — The AXI-4 or ACE port connecting to the SOC and the rest of the memory subsystem may operate at a ratio of the cluster clock domain. The system clock domain can be configured to use an internal clock or an external clock. When configured to use an internal clock, the rate is a ratio of the prescale clock. Supported ratios are 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7, 1:8.
- AUX AXI clock domains — The optional non-coherent AXI-4 port connecting to the SOC may operate at a ratio of the cluster clock domain. Each auxiliary AXI clock domain can be independently configured to use an internal clock or an external clock. When configured to use an internal clock, the rate is a ratio of the pre-scale clock. Supported ratios are 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7, 1:8.
- When configured to use an external asynchronous clock, the AXI interface captures/drives on that clock and there is an asynchronous boundary crossing implemented internal to the cluster.
- *TAP clock domain* — This is a low-speed clock domain for the JTAG TAP controller, controlled by the EJ_TCK pin. It is asynchronous to the Reference Clock.
- *I/O clock domains* — Each port connects the IOCU to the I/O Subsystem. Each IOCU clock may operate at a ratio of the prescale clock domain. Supported ratios are the same as the system clock domain. Similar to the System clock domain, each I/O clock domain can be configured to operate at a ratio of the prescale clock or an asynchronous external clock.

Debug Unit

The Debug Unit (DBU) is an optional component that enables debug using a probe connected through a JTAG scan chain. Alternatively, the DBU can be connected to the system through an APB transactor port. The DBU contains the single TAP controller in the cluster, which can access registers through the cluster. The DBU also contains a RAM to hold instructions and data accessed by the cores while in debug mode.

Features of the debug unit include Hardware Breakpoints and a Fast Debug Channel:

Hardware breakpoints stop the normal operation of the CPU and force the system into debug mode. There are two types of hardware breakpoints implemented in the I6500-F CPU: Instruction breakpoints and Data breakpoints.

Instruction breaks occur on virtual instruction execution addresses and may be qualified by ASID or MMID, VP, GuestID, and Context. Addresses may be single, masked, or ranges.

Data breaks occur on load and store operations based on virtual address, ASID or MMID, VP, GuestID, Context, and data value. Addresses may be single, masked, or ranges. Loads and stores may be aligned or misaligned.

The Fast Debug Channel is a mechanism for efficient bidirectional transfer between a CPU and the debug probe. Data is transferred serially via the TAP interface. Memory-mapped FIFOs buffer the data, isolating software running on the CPU from the actual data transfer. Software can configure the FDC block to generate an interrupt based on the FIFO occupancy level or can operate in a polled mode. Up to 16 virtual channels can travel in each direction.

Inter-CPU Debug Breaks

The MPS includes registers that enable cooperative debugging across all VPs. Programmable registers allow VPs to be placed into debug groups such that whenever one VP within the group enters debug mode, a debug interrupt (DINT) is sent to all VPs within the group, causing them to also enter debug mode and stop executing non-debug mode instructions. This same mechanism can be used to have multiple VPs exit debug mode simultaneously.

PC Sampling

Each VP has hardware to provide periodic sampling of the program counter. Through the DBU, a probe can read addresses that have been executed. The host software can accumulate these executed addresses and provide views of program hot spots, from the module and function level down to the source line and individual instruction levels.

PDtrace

The I6500-F core includes trace support for real-time tracing of instructions, data addresses, data values, and performance counters. A Trace funnel muxes the PDTrace stream from all cores and the CM, and either stores the trace information into an on-chip trace RAM or off-chip memory for post-capture processing by trace regeneration software. Software-only control of trace is possible in addition to probe-based control. The on-chip trace memory may be accessed either through load instructions or the existing JTAG TAP interface, which requires no additional chip pins.

The off-chip trace is managed with the PIB3 (3rd-generation Probe Interface Block) hardware that ships with the product. It provides a selectable trace port width of 8 or 16 pins plus DDR clock. The trace data is streamed on these pins and captured using a compatible probe such as the MIPS Sysprobe SP58ET.

Initial and Possible Configurations

The I6500-F Multiprocessing System can support a variety of configurations. Initially, the following configurations listed in Table 1.2 will be supported. If a different configuration is required, contact your sales account manager for current configurations and to request a new configuration. Each of the initial configurations can include an optional SIMD engine (with integer and floating point data types).

Multi-core offerings are symmetric (same cache size, VPs, SIMD for each core) but different clock ratios per core are supported.

Table 2. I6500-F Configurations

Config Type	Feature Name	Description	Allowed Values
Defined in cm3_config.vh			
Cores	num_cores	Number of cores per cluster	0, 1, 2, 3, 4, 5, 6
L2 cache	l2_cache_size	L2 cache size	256, 512, 1024, 2048 KB
L2 cache misses	l2_missq_override	L2 cache misses in flight (to override default)	8 - 96
L2 data buffer	l2_mem_sdb_override	L2 Store data buffer size (to override default)	8 - 64
L2 prefetch	l2_pref	L2 Prefetch	0, 1 (for absent or present)
Pipes	num_pipes	Number of pipes or scheduler paths in the Coherence Manager	1, 2
ACE requests	l2_num_intv_override	Number of incoming ACE snoop requests	1 - 16
Interrupts	num_irqs	Number of interrupts. Interrupts are as a number of 'slices' where a 'slice' is 8 interrupts.	8 - 256 (in increments of 8)
IOCUs	num_iocus	Number of I/O Coherence Units (IOCUs)	0 - 8

IOCU size	iocu_size	IOCU size information. Chooses the size of the IOCU implementation for
		reads/writes and SDB IDs.
		Small = 4 RDs, 4 WRs, 4 SDB IDs
		Medium = 8 RDs, 8 WRs, 8 SDB IDs
		Large = 16 RDs, 12 WRs, 16 SDB IDs
			Small, Medium, Large

IOCU reads	iocu_num_reads	Number of IOCU reads in flight. N.B: overrides iocu_size.	4, 8, 12, 16, 32
IOCU writes	iocu_num_writes	Number of IOCU writes in flight. N.B: overrides iocu_size.	4, 8, 12, 24
IOCU buffer IDs	iocu_sdb_ids	Number of IOCU Store Data Buffer IDs. N.B: overrides iocu_size.	8 - 32
IOCU width	iocu_user_width	IOCU AxUSER width, routed to MEM port, default = 8	0 - 9
GCRs	ugcr	User Global Configuration Registers (GCRs)	0, 1 (for absent or present)
Relay stages	num_relay_stages_core_mcp_cm	Number of relay stages between cores and CM	0, 1, 2 (0 default)
ITU	cluster_itu	Interthread Communication Unit	0, 1 (for absent or present)

Table 2. I6500-F Configurations

Config Type	Feature Name	Description	Allowed Values
External clock	c_mem_ext_clk_en	External Clock on ACE memory port (only used if ace = 1 and not integrated NoC)	0, 1 (for absent or present)
Aux ports	num_aux	Number of auxiliary ports	0 - 4
Aux port 0	aux0_data_width	Auxiliary Port 0 data width	32, 64, 128, 256, 512
Aux port 1	aux1_data_width	Auxiliary Port 1 data width	32, 64, 128, 256, 512
Aux port 2	aux2_data_width	Auxiliary Port 2 data width	32, 64, 128, 256, 512
Aux port 3	aux3_data_width	Auxiliary Port 3 data width	32, 64, 128, 256, 512
defined in mips_config.vh			
Clusters	num_clusters	Number of coherent clusters. NOTE: With an integrated NoC, this refers to the number of clusters instantiated in mips_soc. However, without an integrated NoC this refers to the number of clusters in the example mips_soc. It does NOT refer to the number of clusters that you may instantiate in your design.	1-4 (for integrated NoC) Generally set to 2 for non-integrated NoC because a dual-cluster mips_soc example is provided.
ACE	ace	MEM port includes AXI Coherency Extensions	0, 1 (for absent, or present)
Virtual Processors	num_vps	Number of VPs per core	1, 2, 4
PDtrace output bus	tru_ext_bus_type	Type of external bus for Pdtrace. PIB: output of Probe Interface Block. TC: 256-bit wide output of Trace Funnel.	PIB or TC
AXI parity	axi_addr_parity	AXI address parity supported.	0, 1 (for absent or present)
AXI parity/byte	axi_addr_perbyte_parity	AXI address parity per-byte	0 - single parity bit, 1 - per byte parity
AXI data parity	axi_data_parity	AXI data parity supported.	0, 1 (for absent or present)
PDtrace	pdtrace	PDtrace unit	0, 1 (for absent or present)
PDtrace memory	tru_mem	PDtrace internal memory size	6, 7, 8
Pdtrace PIB	tru_PIB	PDtrace PIB size (width)	8, 16 bits
defined in sam_core_config.vh			
L1 Instr cache	l1_icache_size	L1 Instruction cache size	32, 64 KB
L1 Data cache	l1_dcache_size	L1 Data cache size	32, 64 KB
FPU	fpu_present	FPU and MSA support	Yes, No
DSPRAM	dspram_size	Core Data Scratchpad RAM (DSPRAM) size	0, 64 KB
VTLB's entries per thread	vtlb_per_thread	Per-thread VTLB dual entries for address translation	16, 32, 64

Latencies and Repeat Rates

This chapter provides the instructions latency and repeat rates for the following instruction types.

Definition of Terms

The terms *latency* and *repeat rate* are defined as follows:

Latency is defined as the minimum time between when an instruction issues, and the time that a subsequent dependent instruction may issue. For example, and ADD instruction has a latency of 1 cycle. Consider the following code sequence:

```
ADD r3, r1, r2
ADD r5, r4, r3
```

In this example the second ADD instruction is dependent on the value placed into r3 by the first ADD instruction. It may issue one cycle after the first ADD instruction issues.

Repeat rate is measured as the minimum issue interval time between independent instructions. For example, a MUL instruction has a latency of 4 cycles and a repeat rate of 1 cycle. Consider the following code sequence:

```
MUL r4, r1, r2
MUH r5, r1, r2
```

The MUL instruction multiplies the r1 and r2 values and places the lower half of the result into r4. The MUH instruction multiples the r1 and r2 values and places the upper half of the result into r5. In this case the MUH can issue one cycle after the MUL instruction issues.

MTC0 Instruction Considerations

Any MTC0 instruction which can potentially change the operating mode (kernel, supervisor, user) or context (memory mapping) should be executed in the delay slot of a JALR.HB instruction to avoid hazards. Instructions following JALR.HB-MTC0 pair will thus be fetched and executed in the new mode. If the mode-changing MTC0 instruction is not placed in delay slot of JALR.HB instruction, it is not guaranteed that the following instruction will be fetched and executed in the new mode or context.

Execution of the MTC0 instruction can change the following register bits:

Status.ERL: Changes the mapping of KUSeg memory segment. If the program is being executed in the KUSeg segment, and the MTC0 instruction that modifies the value of the ERL bit is not placed in the delay slot of a JALR.HB instruction, the instructions following the MTC0 instruction may be fetched from a different memory region.

Status.ERL, Status.EXL, Status.KSU: Changes the mode of operation. If the MTC0 instruction that modified the mode is not placed in the delay slot of JALR.HB instruction, the instructions following the MTC0 instruction may be fetched in kernel mode but executed in the new mode.

Status.KX, Status.SX, Status.UX: These bits determines the access privilege to 64-bit memory segments. If the program is being executed in a 64-bit segment and the MTC0 instruction that modified the value of these bits is not placed in the delay slot of JALR.HB instruction, the instructions following the MTC0 instruction may be fetched incorrectly.

Integer Instruction Latencies and Repeat Rates

Table 3 shows the latency and repeat rates for integer instructions.

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
ADD	Add word	1	1	ALU	2
ADDIU	Add immediate unsigned word	1	1	ALU	2
ADDIUPC	Add immediate to PC (unsigned, non-trapping)	1	1	ALU	2
ADDU	Add unsigned word	1	1	ALU	2
ALIGN	Concatenate two GPRs, and extract a contiguous subset at a byte position. Operates on 32-bit words with a 2-bit byte position field.	1	1	ALU	2
ALUIPC	Aligned add upper immediate to PC	1	1	ALU	2
AND	Bitwise logical AND operation	1	1	ALU	2
ANDI	Bitwise logical AND immediate with a constant	1	1	ALU	2
AUI	Add upper immediate	1	1	ALU	2
AUIPC	Add upper immediate to PC	1	1	ALU	2
B	Unconditional branch	n/a	1	CTU	1
BAL	Branch and link	1	1	CTU	1
BALC	Branch and link compact	1	1	CTU	1
BC	Branch compact	n/a	1	CTU	1
BC1EQZ	Branch if coprocessor 1 equal to zero	n/a	1	CTU	1
BC1NEZ	Branch if coprocessor 1 not equal to zero	n/a	1	CTU	1
BEQ	Branch on equal.	n/a	1	CTU	1
BEQC	Compact branch if GPR values are equal.	n/a	1	CTU	1
BEQZALC	Compact branch-and-link if GPR rt is equal to zero.	1	1	CTU	1
BEQZC	Compact branch if GPR rs is equal to zero.	n/a	1	CTU	1
BGEC	Compact branch if GPR rs is greater than or equal to GPR rt.	n/a	1	CTU	1
BGEUC	Compact branch if GPR rs is greater than or equal to GPR rt, unsigned.	n/a	1	CTU	1
BGEZ	Branch on greater than or equal to zero.	n/a	1	CTU	1
BGEZALC	Compact branch-and-link if GPR rt is greater than or equal to zero.	1	1	CTU	1
BGEZC	Compact branch if GPR rt is greater than or equal to zero.	n/a	1	CTU	1
BGTC	Compact branch if GPR rt is greater than GPR rs (alias for BLTC). Assembly idiom with operands reversed.	n/a	1	CTU	1

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
BGTUC	Compact branch if GPR rt is greater than GPR rs, unsigned (alias for BLTUC). Assembly idiom with operands reversed.	n/a	1	CTU	1
BGTZ	Branch on greater than zero.	n/a	1	CTU	1
BGTZC	Compact branch if GPR rt is greater than zero.	n/a	1	CTU	1
BGTZALC	Compact branch-and-link if GPR rt is greater than zero.	1	1	CTU	1
BITSWAP	Swaps (reverses) bits in each byte. Operates on all 4 bytes of a 32-bit GPR. See DBITSWAP instruction.	1	1	ALU	2
BLEC	Compact branch if GPR rt is less than or equal to GPR rs (alias for BGEC). Assembly idiom with operands reversed.	n/a	1	CTU	1
BLEUC	Compact branch if GPR rt is less than or equal to GPR rt, unsigned (alias for BGEUC). Assembly idiom with operands reversed.	n/a	1	CTU	1
BLEZ	Branch on less than or equal to zero.	n/a	1	CTU	1
BLEZALC	Compact branch-and-link if GPR rt is less than or equal to zero.	1	1	CTU	1
BLEZC	Compact branch if GPR rt is less than or equal to zero.	n/a	1	CTU	1
BLTC	Compact branch if GPR rs is less than GPR rt.	n/a	1	CTU	1
BLTUC	Compact branch if GPR rs is less than GPR rt, unsigned.	n/a	1	CTU	1
BLTZ	Branch on less than zero.	n/a	1	CTU	1
BLTZALC	Compact branch-and-link if GPR rt is less than zero.	1	1	CTU	1
BLTZC	Compact branch if GPR rt is less than zero.	n/a	1	CTU	1
BNE	Branch on not equal.	n/a	1	CTU	1
BNEC	Compact branch if GPR value are not equal.	n/a	1	CTU	1
BNEZALC	Compact branch-and-link if GPR rt is not equal to zero.	1	1	CTU	1
BNEZC	Compact branch if GPR rs is not equal to zero.	1	1	CTU	1
BOVC	Branch on overflow, compact.	1	1	CTU	1
BNVC	Branch on no overflow, compact.	1	1	CTU	1
BREAK	Breakpoint. To cause a breakpoint exception.	1	n/a	CTU	1
CACHE	Perform a cache operation specified by the opcode.	≥8	≥5	LSU	1
CFC1	Move control word from floating point	1	1	ALU	2
CLO	Count number of leading ones in a word.	1	1	ALU	2
CLZ	Count number of leading zeros in a word.	1	1	ALU	2
CTC1	Move control word to floating point	1	1	ALU	2
DADD	Doubleword add. Add two 64-bit integers. Trap on overflow.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
DADDIU	Doubleword add immediate unsigned. Add a constant to a 64-bit integer.	1	1	ALU	2
DADDU	Doubleword add unsigned. Add two 64-bit integers	1	1	ALU	2
DAHI	Doubleword add higher immediate	1	1	ALU	2
DALIGN	Concatenate two GPRs, and extract a contiguous subset at a byte position. Operates on 64-bit double-words with a 3-bit byte position field.	1	1	ALU	2
DATI	Doubleword add top immediate	1	1	ALU	2
DAUI	Doubleword add upper immediate	1	1	ALU	2
DBITSWAP	Swaps (reverses) bits in each byte. Operates on all 8 bytes of a 64-bit GPR. See BITSWAP instruction.	1	1	ALU	2
DCLO	Count leading ones in double-word.	1	1	ALU	2
DCLZ	Count leading zeros in double-word.	1	1	ALU	2
DDIV DMOD	Divide 64-bit integers signed. Modulo 64-bit double-words signed Divide the operands in GPR rs and GPR ft, and place the result into GPR rd. See the DIV instruction. *The integer divide unit can hold up to two DDIV/DMOD instructions. Two DDIV/DMOD instructions may be issued back to back, but a subsequent DDIV/DMOD instruction must wait until the first DDIV/DMOD instruction has generated a result. See the DIV instruction.	≥5	1*	iDIV	1
DDIVU DMODU	Divide 64-bit unsigned integers. Modulo double-words unsigned Divide the unsigned 64-bit operands in GPR rs and GPR rt, and place the result into GPR rd. See the DIVU instruction.	≥5	1* (See DDIV)	iDIV	1
DERET	Return from debug exception.	1	n/a	CTU	1
DEXT	Doubleword extract bit field.	1	1	ALU	2
DEXTM	Doubleword extract bit field middle.	1	1	ALU	2
DEXTU	Doubleword extract bit field upper.	1	1	ALU	2
DI	Disable interrupts. Return the previous value of the CP0 Status register and disable interrupts.	1	n/a	CP0	1
DINS	Doubleword insert bit field. Merge a right-justified bit field from the GPR rs field into the specified GPR rt field.	1	1	ALU	2
DINSM	Doubleword insert bit field middle.	1	1	ALU	2
DINSU	Doubleword insert bit field upper.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
DIV MOD	Divide 32-bit integers signed. Modulo words signed Divide the operands in GPR rs and GPR ft, and place the result into GPR rd. *The integer divide unit can hold up to two DIV/MOD instructions. Two DIV/MOD instructions may be issued back to back, but a subsequent DIV/MOD instruction must wait until the first DIV/MOD instruction has generated a result. See the DDIV instruction.	≥5	1*	iDIV	1
DIVU MODU	Divide 32-bit unsigned integers. Modulo words unsigned Divide the unsigned 32-bit operands in GPR rs and GPR rt, and place the result into GPR rd. See the DDIVU instruction.	≥5	1* (See DIV)	iDIV	1
DLSA	Doubleword load scaled address. Add two values from registers rs and rt. See the LSA instruction.	1	1	ALU	2
DMFC0	Doubleword move from CP0 to GPR.	2	1	CP0	1
DMFC1	Doubleword move from FPR to GPR.	1	1	ALU	2
DMTC0	Doubleword move from GPR to CP0. *Even though there are two ALU's, DMTC0 instructions are serialized, allowing only one instruction at a time.	1	1	ALU*	2
DMTC1	Doubleword move from GPR to FPR.	1	1	ALU	2
DMUH	Multiply double-words signed, high doubleword. Performs a signed 64-bit integer multiplication and places the high 64 bits of the result in the destination register.	4	1	iMUL	1
DMUL	Multiply double-words signed, low doubleword. Performs a signed 64-bit integer multiplication and places the low 64 bits of the result in the destination register.	4	1	iMUL	1
DMUHU	Multiply double-words unsigned, high doubleword. Performs an unsigned 64-bit integer multiplication and places the high 64 bits of the result in the destination register.	4	1	iMUL	1
DMULU	Multiply double-words unsigned, low doubleword. Performs an unsigned 64-bit integer multiplication and places the low 64 bits of the result in the destination register.	4	1	iMUL	1
DROTR	Doubleword rotate right. Logical rotate right of a doubleword by a fixed amount — 0 - 31 bits.	1	1	ALU	2
DROTR32	Doubleword rotate right plus 32. Logical rotate right of a doubleword by a fixed amount — 32 - 63 bits.	1	1	ALU	2
DROTRV	Doubleword rotate right variable. Logical rotate right of a doubleword by a variable number of bits.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
DSBH	Doubleword swap bytes within half-words. Swap the bytes within each halfword of GPR rt and store into GPR rd.	1	1	ALU	2
DSHD	Doubleword swap half-words within double-words. Swap the half-words within each doubleword of GPR rt and store into GPR rd.	1	1	ALU	2
DSLL	Doubleword shift left logical. Logical left-shift of a doubleword by a fixed amount — 0 - 31 bits.	1	1	ALU	2
DSLL32	Doubleword shift left logical plus 32. Logical left-shift of a doubleword by a fixed amount — 32 - 63 bits.	1	1	ALU	2
DSLLV	Doubleword shift left logical variable. Logical left-shift of a doubleword by a variable number of bits.	1	1	ALU	2
DSRA	Doubleword right shift arithmetic. Arithmetic right-shift of a doubleword by a fixed amount — 0 - 31 bits.	1	1	ALU	2
DSRA32	Doubleword right shift arithmetic plus 32. Arithmetic right-shift of a doubleword by a fixed amount — 32 - 63 bits.	1	1	ALU	2
DSRAV	Doubleword shift right arithmetic variable. Arithmetic right-shift of a doubleword by a variable number of bits.	1	1	ALU	2
DSRL	Doubleword shift right logical. Logical right-shift of a doubleword by a fixed amount — 0 - 31 bits.	1	1	ALU	2
DSRL32	Doubleword shift right logical plus 32. Logical right-shift of a doubleword by a fixed amount — 32 - 63 bits.	1	1	ALU	2
DSRLV	Doubleword shift right logical variable. Logical right-shift of a doubleword by a variable number of bits.	1	1	ALU	2
DSUB	Doubleword subtract. Subtract 64-bit integers. Trap on overflow.	1	1	ALU	2
DSUBU	Doubleword subtract unsigned. Subtract unsigned 64-bit integers. Trap on overflow.	1	1	ALU	2
DVP	Disable virtual processor. Disable all virtual processors in a core except the one that issued the instruction.	variable	n/a	CP0	1
EHB	Execute hazard barrier. Stop instruction execute until all execution hazards have been cleared.	1	n/a	ALU	2
EI	Enable interrupts. Return the previous state of the CP0 Status register and enable interrupts.	1	n/a	CP0	1
ERET	Exception return. Return from interrupt, exception, or error trap.	1	n/a	CTU	1
ERETNC	Exception return no clear. Return from interrupt, exception, or error trap without clearing the LL bit.	1	n/a	CTU	1
EVP	Enable virtual processor. Enable all virtual processors in a core except the one that issued the instruction.	2	n/a	CP0	1

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
EXT	Extract bit field. Extract a bit field from GPR rx and store it right-justified into GPT rt.	1	1	ALU	2
INS	Insert bit field. Merge a right-justified bit field from GPR rs into a specified field in GPR rt.	1	1	ALU	2
J	Jump. Branch within the current 256 MByte region.	n/a	1	CTU	1
JAL	Jump and link. Execute a procedure call within the current 256 MByte region.	1	1	CTU	1
JALR	Jump and link register. Execute a procedure call to an instruction address in a register.	1	1	CTU	1
JALR.HB	Jump and link register with hazard barrier. Execute a procedure call to an instruction address in a register and clear all execution and instruction hazards.	n/a	1	CTU	1
JIALC	Jump indexed and link, compact. The jump target is formed by sign extending the offset field of the instruction and adding it to the contents of GPR rt.	n/a	1	CTU	1
JIC	Jump indexed, compact. The branch target is formed by sign extending the offset field of the instruction and adding it to the contents of GPR rt.	n/a	1	CTU	1
JR	Jump register. Execute a branch to an instruction address in a register.	n/a	1	CTU	1
JR.HB	Jump register with hazard barrier. Execute a a branch to an instruction address in a register and clear all execution and instruction hazards.	n/a	n/a	CTU	1
LB	Load byte from memory as a signed value.	≥3	1	LSU	1
LBU	Load byte from memory as an unsigned value.	≥3	1	LSU	1
LD	Load doubleword from memory.	≥3	1	LSU	1
LDC1	Load doubleword from memory to an FPR.	≥3	1	LSU	1
LDPC	Load doubleword PC-relative. Load a doubleword from memory using a PC-relative address.	≥3	1	LSU	1
LH	Load halfword from memory as a signed value.	≥3	1	LSU	1
LHU	Load halfword from memory as an unsigned value.	≥3	1	LSU	1
LL	Load linked word. Load a word from memory for an atomic read-modify-write.	≥3	1	LSU	1
LLD	Load linked doubleword. Load a doubleword from memory for an atomic read-modify-write.	≥3	1	LSU	1
LLDP	Load linked doubleword paired. Load a doubleword paired instruction from memory for an atomic read-modify-write.	≥3	1	LSU	1
LLWP	Load linked word paired. Load a paired word from memory for an atomic read-modify-write.	≥3	1	LSU	1
LSA	Load scaled address. Add two values from registers rs and rt. See DLSA instruction.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
LUI	Load upper immediate. Load a constant into the upper half of a word.	1	1	ALU	2
LW	Load word from memory as a signed value.	≥3	1	LSU	1
LWC1	Load word from memory to an FPR.	≥3	1	LSU	1
LWPC	Load word PC relative. Load a word from memory as a signed value using a PC-relative address.	≥3	1	LSU	1
LWU	Load word from memory as an unsigned value.	≥3	1	LSU	1
LWUPC	Load word unsigned PC relative. Load a word from memory as an unsigned value using a PC-relative address.	≥3	1	LSU	1
MFC0	Move from CP0. Move the contents of a CP0 register to a general register.	2	1	CP0	1
MFC1	Copy a word from an FPR to a GPR.	1	1	ALU	2
MFHC0	Move from high CP0. Move the contents of the upper 32 bits of a CP0 register, extended by 32-bits, to a general register.	2	1	CP0	1
MFHC1	Copy word from high half of an FPR to a GPR.	1	1	ALU	2
MTCO	Move to CP0. Move the contents of the upper 32 bits of a general register to a general register. *Even though there are two ALU's, MTC0 instructions are serialized, allowing only one instruction at a time.	1	1	ALU*	2
MTC1	Move word from a GPR to an FPR.	1	1	ALU	2
MTHC0	Move to high CP0. Move the contents of the upper 32 bits of a CP0 register, extended by 32-bits, to a general register.	1	1	ALU* (See MTCO)	2
MTHC1	Copy word from a GPR to the high half of an FPR.	1	1	ALU	2
MUH	Multiply words signed, high word. Performs a signed 32-bit integer multiplication and places the high 32 bits of the result in the destination register.	4	1	iMUL	1
MUHU	Multiply words unsigned, high word. Performs an unsigned 32-bit integer multiplication and places the high 32 bits of the result in the destination register.	4	1	iMUL	1
MUL	Multiply words signed, low word. Performs a signed 32-bit integer multiplication and places the low 32 bits of the result in the destination register.	4	1	iMUL	1
MULU	Multiply words unsigned, low word. Performs an unsigned 32-bit integer multiplication and places the low 32 bits of the result in the destination register.	4	1	iMUL	1
NAL	No-op and link. Used to read the PC.	1	1	CTU	1
NOP	No operation.	1	1	ALU	2
NOR	NOT OR. Bitwise logical NOT OR.	1	1	ALU	2
OR	OR operation. Bitwise logical OR.	1	1	ALU	2
ORI	OR immediate. Bitwise logical or with a constant.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
PAUSE	Pause. Wait for the LL bit to clear.	n/a	1	LSU	1
PREF	Prefetch. Move data between memory and cache.	n/a	1	LSU	1
RDHWR	Read hardware register. Move the contents of a hardware register to a general purpose register (GPR) if that operation is enabled by privileged software.	2	1	CP0	1
RDPGPR	Read GPR from previous shadow set. Move the contents of a GPR from the previous shadow set to a current GPR.	1	1	ALU	2
ROTR	Rotate word right. Logical right-rotate of a word by a fixed number of bits.	1	1	ALU	2
ROTRV	Rotate word right variable. Logical right-rotate of a word by a variable number of bits.	1	1	ALU	2
SB	Store byte. Store a byte to memory.	n/a	1	LSU	1
SC	Store conditional word. Store a word to memory to complete an atomic read-modify-write.	≥8	≥5	LSU	1
SCD	Store conditional doubleword. Store a doubleword to memory to complete an atomic read-modify-write.	≥8	≥5	LSU	1
SCDP	Store conditional doubleword paired. Store a paired doubleword instruction to memory to complete an atomic read-modify-write.	≥8	≥5	LSU	1
SCWP	Store conditional word paired. Store a paired word instruction to memory to complete an atomic read-modify-write.	≥8	≥5	LSU	1
SD	Store a doubleword to memory.	n/a	1	LSU	1
SDBBP	Software debug break point. Cause a debug breakpoint exception.	1	n/a	CTU	1
SDC1	Store doubleword from FPR to memory	n/a	1	LSU	1
SEB	Sign-extend byte. Sign-extend the least significant byte of GPR rt and store the value into GPR rd.	1	1	ALU	2
SEH	Sign-extend halfword. Sign-extend the least significant halfword of GPR rt and store the value into GPR rd.	1	1	ALU	2
SELEQZ	Select integer GPR value or zero. Condition true only if all bits in GPR rt are zero.	1	1	ALU	2
SELNEZ	Select integer GPR value or non-zero. Condition true only if any bit in GPR rt is non-zero.	1	1	ALU	2
SH	Store halfword to memory.	n/a	1	LSU	1
SIGRIE	Signal reserved instruction exception.	n/a	n/a	--	Y
SLL	Shift word left logical by a fixed number of bits.	1	1	ALU	2
SLLV	Shift word left logical by a variable number of bits.	1	1	ALU	2
SLT	Set on less than. Record the result of a less-than comparison.	1	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
SLTI	Set on less than immediate. Record the result of a less-than comparison with a constant.	1	1	ALU	2
SLTIU	Set on less than immediate unsigned. Record the result of an unsigned less-than comparison with a constant.	1	1	ALU	2
SLTU	Set on less than unsigned. Record the result of a less-than comparison.	1	1	ALU	2
SRA	Shift word right arithmetic. Execute an arithmetic right-shift of a word by a fixed number of bits.	1	1	ALU	2
SRAV	Shift word right arithmetic variable. Execute an arithmetic right-shift of a word by a variable number of bits.	1	1	ALU	2
SRL	Shift word right logical. Execute a logical right-shift of a word by a fixed number of bits.	1	1	ALU	2
SRLV	Shift word right logical variable. Execute a logical right-shift of a word by a variable number of bits.	1	1	ALU	2
SSNOP	Superscalar no-operation. Break superscalar issue.	1	1	ALU	2
SUB	Subtract 32-bit integers. Trap on overflow.	1	1	ALU	2
SUBU	Subtract unsigned 32-bit integers.	1	1	ALU	2
SW	Store word to memory.	n/a	1	LSU	1
SWC1	Store word from FPR to memory	n/a	1	LSU	1
SYNC	Synchronize shared memory. Order loads and stores for shared memory.	≥6	≥5	LSU	1
SYNCI	Synchronize caches to make instruction writes effective.	≥8	≥5	LSU	1
SYSCALL	System call. Cause a system call exception.	1	n/a	CTU	1
TEQ	Trap if equal. Compare GPR’s and do a conditional trap if equal.	n/a	1	ALU	2
TGE	Trap if greater or equal. Compare GPR’s and do a conditional trap on greater or equal condition.	n/a	1	ALU	2
TGEU	Trap if greater or equal unsigned.	n/a	1	ALU	2
TLBINV	TLB invalidate. Invalidates TLB entry based on ASID and index match.	≥8	≥5	LSU	1
TLBINVF	TLB invalidate flush.	≥8	≥5	LSU	1
TLBP	TLB probe. Find a matching TLB entry.	≥8	≥5	LSU	1
TLBR	TLB read. Read an entry from the TLB.	≥8	≥5	LSU	1
TLBWI	TLB write indexed. Write or invalidate a TLB entry indexed by the CP0 Index register.	≥8	≥5	LSU	1
TLBWR	TLB write random. Write a TLB entry indexed by an implementation-defined location.	≥8	≥5	LSU	1
TLT	Trap if less than. Compare GPR’s and trap on condition.	n/a	1	ALU	2

Table 3. I6500-F Integer Instructions — Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate	Unit Type	Number of Units
TLTU	Trap if less than unsigned. Compare GPR's and trap on condition.	n/a	1	ALU	2
TNE	Trap if not equal. Compare GPR's and trap on condition.	n/a	1	ALU	2
WAIT	Wait for event. Enter standby mode.	variable	n/a	ALU	2
WRPGPR	Write to GPR in previous shadow set. Move the contents of a current GPR to a GPR in the previous shadow set.	1	1	ALU	2
WSBH	Word swap bytes within half-words. Swap the bytes within each halfword of GPR rt and store the value into GPR rd.	1	1	ALU	2
XOR	Exclusive OR.	1	1	ALU	2
XORI	Exclusive OR immediate.	1	1	ALU	2

Floating Point Instruction Latencies and Repeat Rates

Table 4 shows the latency and repeat rates for the floating point unit (FPU) instructions.

Table 4. I6500-F Floating Point Latency and Repeat Rates

Instruction	Definition	Latency	Repeat Rate
ABS.fmt	Floating point absolute value	1	1
ADD.fmt	Floating point add	4	1
CEIL.L.fmt	Fixed point ceiling convert to long fixed point	4	1
CEIL.W.fmt	Fixed point ceiling convert to word fixed point	4	1
CLASS.fmt	Scalar floating point class mask	1	1
CMP.cond.fmt	Fixed point compare setting mask	2	1
CVT.D.fmt	Fixed point convert to double floating point	4	1
CVT.L.fmt	Fixed point convert to long fixed point	4	1
CVT.S.fmt	Fixed point convert to single floating point	4	1
CVT.W.fmt	Fixed point convert to word fixed point	4	1
DIV.fmt	Floating point divide	variable	variable
FLOOR.L.fmt	Fixed point floor convert to long fixed point	4	1
FLOOR.W.fmt	Fixed point floor convert to word fixed point	4	1
MADDF.fmt	Floating point fused multiply add	8	1
MAX.fmt	Scalar floating point maximum value	2	1
MAXA.fmt	Scalar floating point maximum value with input arguments	2	1
MIN.fmt	Scalar floating point minimum value	2	1
MINA.fmt	Scalar floating point minimum value with input arguments	2	1

Table 4. I6500-F Floating Point Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
MSUBF.fmt	Floating point fused multiply subtract	8	1
MUL.fmt	Floating point multiply	5	1
NEG.fmt	Floating point negate	1	1
RECIP.fmt	Floating point reciprocal	variable	variable
RINT.fmt	Scalar floating point round to integral floating point value	4	1
ROUND.L.fmt	Floating point round to long fixed point	4	1
ROUND.W.fmt	Floating point round to word fixed point	4	1
RSQRT.fmt	Floating point reciprocal square root	variable	variable
SEL.fmt	Select floating point values with	1	1
SELEQZ.fmt	Select floating point with conditions equal to zero	1	1
SELNEZ.fmt	Select floating point with conditions not equal to zero	1	1
SQRT.fmt	Floating point square root	variable	variable
SUB.fmt	Floating point subtract	4	1
TRUNC.L.fmt	Floating point truncate to long fixed point	4	1
TRUNC.W.fmt	Floating point truncate to word fixed point	4	1

MSA Instruction Latencies and Repeat Rates

The following table shows the latency and repeat rates for the MIPS SIMD Architecture (MSA) instructions.

Table 5. I6500-F MSA Instruction Latency and Repeat Rates

Instruction	Definition	Latency	Repeat Rate
ADD_A.df	Vector add absolute values	2	2
ADDS_A.df	Vector saturated add of absolute values	2	2
ADDS_S.df	Vector saturated add of signed values	2	2
ADDS_U.df	Vector saturated add of unsigned values	2	2
ADDV.df	Vector add	1, 2	2
ADDVI.df	Immediate add	2	2
AND.V	Vector and	1	2
ANDI.B	Immediate and	1	2
ASUB_S.df	Vector absolute values of signed subtract	2	2
ASUB_U.df	Vector absolute values of unsigned subtract	2	2
AVE_S.df	Vector signed average	2	2
AVE_U.df	Vector unsigned average	2	2
AVER_S.df	Vector signed average rounded	2	2
AVER_U.df	Vector unsigned average rounded	2	2
BCLR.df	Vector bit clear	2	2

Table 5. I6500-F MSA Instruction Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
BCLRI.df	Immediate bit clear	2	2
BINSRI.df	Immediate bit insert right	2	2
BINSL.df	Vector bit insert left	2	2
BINSLI.df	Immediate bit insert left	2	2
BINSR.df	Vector bit insert right	2	2
BMNZ.V	Vector move if not zero	1	2
BMNZI.B	Immediate move if not zero	1	2
BMZ.V	Vector move if zero	1	2
BMZI.B	Immediate move if zero	1	2
BNZ.df	Branch if all elements are non zero	1	1
BNZ.V	Branch if any element non zero	1	1
BNEG.df	Vector selected bit position negate	2	2
BNEGI.df	Immediate bit negate	2	2
BSEL.V	Vector bit select	1	2
BSELI.B	Immediate bit select	1	2
BSET.df	Vector bit set	2	2
BSETI.df	Immediate bit set	2	2
BZ.df	Branch if any element zero	1	1
BZ.V	Branch if all elements zero	1	1
CEQ.df	Vector compare equal	2	2
CEQI.df	Immediate compare equal	2	2
CFCMSA	GPR copy from MSA control register	1	1
CTCMSA	GPR copy to MSA control register	1	1
CLE_S.df	Vector compare signed less than or equal	2	2
CLE_U.df	Vector compare unsigned less than or equal	2	2
CLEI_S.df	Immediate compare signed less than or equal	2	2
CLEI_U.df	Immediate compare unsigned less than or equal	2	2
CLT_S.df	Vector compare signed less than	2	2
CLT_U.df	Vector compare unsigned less than	2	2
CLTI_S.df	Immediate compare signed less than or equal	2	2
CLTI_U.df	Immediate compare unsigned less than or equal	2	2
COPY_S.df	Element move to GPR signed	1	1
COPY_U.df	Element move to GPR unsigned	1	1
DIV_S.df	Vector signed divide. See MOD_S instruction	variable	variable
DIV_U.df	Vector unsigned divide. See MOD_U instruction	variable	variable
DOTP_S.df	Vector signed dot product	5	2
DOTP_U.df	Vector unsigned dot product	5	2
DPADD_S.df	Vector signed dot product and add	5	2

Table 5. I6500-F MSA Instruction Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
DPADD_U.df	Vector unsigned dot product and add	5	2
DPSUB_S.df	Vector signed dot product and subtract	5	2
DPSUB_U.df	Vector unsigned dot product and subtract	5	2
FADD.df	Vector FP add	4	2
FCAF.df	Vector FP compare always false	2	2
FCEQ.df	Vector FP compare equal	2	2
FCLASS.df	Vector FP class mask, record class (0, inf, qNaN, etc) of data	2	2
FCLE.df	Vector FP compare less than equal	2	2
FCLT.df	Vector FP compare less than	2	2
FCOR.df	Vector FP compare not equal	2	2
FCNE.df	Vector FP compare not equal	2	2
FCUEQ.df	Vector FP compare not equal	2	2
FCULE.df	Vector FP compare greater than	2	2
FCULT.df	Vector FP compare greater than equal	2	2
FCUN.df	Vector FP compare unordered	2	2
FCUNE.df	Vector FP compare not equal	2	2
FDIV.df	Vector FP divide	variable	variable
FEXDO.df	Vector FP down convert	4	2
FEXP2.df	Vector FP base 2 exponentiation ws is float df, wt is integer of size df	5	2
FEXUPL.df	Vector FP up convert left	4	2
FEXUPR.df	Vector FP up convert right	4	2
FFINT_S.df	Vector FP convert from signed integer	4	2
FFINT_U.df	Vector FP convert from unsigned integer	4	2
FFQL.df	Vector FP convert from fixed point left	4	2
FFQR.df	Vector FP convert from fixed point right	4	2
FILL.df	Replicate and move from GPR	1	2
FLOG2.df	Vector FP base 2 exponentiation ws is float df, wt is integer of size df	2	2
FMADD.df	Vector multiply add	8	2
FMSUB.df	Vector multiply subtract	8	2
FMAX.df	Vector FP maximum	2	2
FMAX_A.df	Vector FP maximum on absolute value	2	2
FMIN.df	Vector FP minimum	2	2
FMIN_A.df	Vector FP minimum on absolute value	2	2
FMUL.df	Vector FP multiply	5	2
FRCP.df	Vector FP reciprocal	variable	variable

Table 5. I6500-F MSA Instruction Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
FRINT.df	Vector FP round to integer value but retain float format	4	2
FRSQRT.df	Vector FP reciprocal-square root	variable	variable
FSAF.df	Vector FP signaling equal	2	2
FSEQ.df	Vector FP signaling equal	2	2
FSLE.df	Vector FP signaling less than equal	2	2
FSLT.df	Vector FP signaling less than	2	2
FSNE.df	Vector FP signaling not equal	2	2
FSOR.df	Vector FP compare not equal	2	2
FSQRT.df	Vector FP square root	variable	variable
FSUB.df	Vector FP sub	4	2
FSUEQ.df	Vector FP signaling not equal	2	2
FSULE.df	Vector FP signaling greater than	2	2
FSULT.df	Vector FP signaling greater than equal	2	2
FSUN.df	Vector FP signaling equal	2	2
FSUNE.df	Vector FP compare not equal	2	2
FTINT_S.df	Vector FP convert to signed integer	4	2
FTINT_U.df	Vector FP convert to unsigned integer	4	2
FTQ.df	Vector FP to fixed point	4	2
FTRUNC_S.df	Vector FP convert to signed integer	4	2
FTRUNC_U.df	Vector FP convert to unsigned integer	4	2
HADD_S.df	Vector signed horizontal add	2	2
HADD_U.df	Vector unsigned horizontal add	2	2
HSUB_S.df	Vector signed horizontal sub	2	2
HSUB_U.df	Vector unsigned horizontal sub	2	2
ILVEV.df	Vector interleave even	1	2
ILVL.df	Vector interleave left	1	2
ILVOD.df	Vector interleave odd	1	2
ILVR.df	Vector interleave right	1	2
INSERT.df	Move from GPR	1	2
INSVE.df	Move from element	1	2
LD.df	Vector load	≥3	2
LDI.df	Immediate load elements	1	2
MADD_Q.df	Vector fixed point madd	5	2
MADDR_Q.df	Vector fixed point multiply rounded and add	5	2
MADDV.df	Vector multiply add	5	2
MAX_A.df	Vector maximum of absolute value	2	2
MAX_S.df	Vector signed maximum	2	2

Table 5. I6500-F MSA Instruction Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
MAX_U.df	Vector unsigned maximum	2	2
MAXI_S.df	Immediate signed maximum	2	2
MAXI_U.df	Intermediate signed maximum	2	2
MIN_A.df	Vector min of absolute value	2	2
MIN_S.df	Vector signed minimum	2	2
MIN_U.df	Vector unsigned minimum	2	2
MINI_S.df	Immediate signed minimum	2	2
MINI_U.df	Immediate unsigned minimum	2	2
MOD_S.df	Vector signed remainder. See DIV_S instruction.	variable	variable
MOD_U.df	Vector unsigned remainder. See DIV_U instruction.	variable	variable
MOVE.V	Vector move	1	2
MSUB_Q.df	Vector fixed point subtract	5	2
MSUBR_Q.df	Vector fixed point multiply rounded and subtracted	5	2
MSUBV.df	Vector multiply subtract	5	2
MUL_Q.df	Vector fixed point multiply	5	2
MULR_Q.df	Vector fixed point multiply rounded	5	2
MULV.df	Vector multiply	5	2
NLOC.df	Vector number of leading ones counted	2	2
NLZC.df	Vector number of leading zeros counted	2	2
NOR.V	Vector NOR	1	2
NORI.B	Immediate NOR	1	2
OR.V	Vector OR	1	2
ORI.B	Immediate OR	1	2
PCKEV.df	Vector pack even	1	2
PCKOD.df	Vector pack odd	1	2
PCNT.df	Vector number of bits set	3	2
SAT_S	Immediate signed saturate to width	3	2
SAT_U	Immediate unsigned saturate to width	3	2
SHF.df	Immediate set shuffle	2	2
SLD.df	Element slide	2	2
SLDI.df	Element slide	2	2
SLL.df	Vector shift left	2	2
SLLI.df	Immediate shift left	2	2
SPLAT.df	Element replicate	1	2
SPLATI.df	Element replicate	1	2
SRA.df	Vector shift right arithmetic	2	2
SRAI.df	Immediate shift right arithmetic	2	2
SRAR.df	Vector shift right arithmetic rounded	2	2

Table 5. I6500-F MSA Instruction Latency and Repeat Rates (Continued)

Instruction	Definition	Latency	Repeat Rate
SRARI.df	Immediate shift right arithmetic rounded	2	2
SRL.df	Vector shift right logical	2	2
SRLI.df	Immediate shift right logical	2	2
SRLR.df	Vector shift right logical rounded	2	2
SRLRI.df	Immediate shift right logical rounded	2	2
ST.df	Vector store	□3	1
SUBS_S.df	Vector signed saturated subtract of signed values	2	2
SUBS_U.df	Vector unsigned saturated subtract of unsigned values	2	2
SUBSUS_U.df	Vector unsigned saturated subtract of signed values	2	2
SUBSUU_S.df	Vector signed saturated subtract of unsigned values	2	2
SUBV.df	Vector subtract	1,2	2
SUBVI.df	Immediate signed saturated subtract of unsigned values	2	2
VSHF.df	Vector shuffle	2	2
XOR.V	Vector XOR	1	2
XORI.B	Immediate XOR	1	2

Revision History

Revision	Date	Description
01.00	March 31, 2017	• RC 1.00 Initial production release of I6500.
01.01	February 26, 2018	• Updated Global Interrupt Controller (GIC) section to add Linux implementation requirement.
01.02	May 31, 2019	• Internal edit of existing material.
01.03	August 30, 2019	• Miscellaneous review comments. • Update to new MIPS template.
01.10	March 17, 2025	• Added Instruction Latency and Repeat Rate section.

Legal Notice