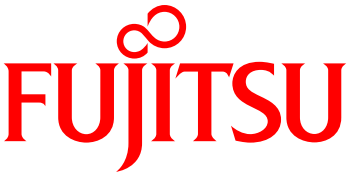

Fujitsu SPARC64™ VII Processor

Fujitsu Limited

Release 1.0. 1 June 2008

Fujitsu Limited
4-1-1 Kamikodanaka
Nakahara-ku, Kawasaki, 211-8588
Japan

Copyright© 2008 Fujitsu Limited, 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan. All rights reserved.

This product and related documentation are protected by copyright and distributed under licenses restricting their use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Fujitsu Limited and its licensors, if any.

The product(s) described in this book may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

SPARC® is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc.

SPARC64 is a registered trademark of SPARC International, Inc., licensed exclusively to Fujitsu Limited.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Sun, Sun Microsystems, the Sun logo, Solaris, and all Solaris-related trademarks and logos are registered trademarks of Sun Microsystems, Inc.

Fujitsu and the Fujitsu logo are trademarks of Fujitsu Limited.

All other company, product names mentioned may be trademarks orregistered trademarks of their respective holders and are used for identification purpose only.

This publication is provided “as is” without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement. This publication could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein; these changes will be incorporated in new editions of the publication. Fujitsu Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

Contents

1 SPARC64 Series

The SPARC64 Series is a SPARC processor developed by Fujitsu for UNIX servers. High reliability technology of the mainframe class and a frequency exceeding 1 GHz have been realized with the SPARC64 V. This processor has been used for Fujitsu PRIMEPOWER servers. The SPARC64 VI has realized high throughput by using the SPARC64 V as a base and by incorporating a two-core x two thread architecture. The throughput of the latest SPARC 64 VII has been improved further by incorporating four-core architecture and by modifying the multi-threading mechanism. These SPARC64 VI and SPARC64 VII processors are used with SPARC Enterprise servers.

2 SPARC64 VII Overview

The SPARC64 VII is the latest processor developed by Fujitsu for the SPARC64 Series. It uses the 65-nm technology of Fujitsu and it has realizes an operating frequency of 2.5 GHz. The chip measures 21.3 mm by 20.9 mm. The chip has four built-in cores with a shared 6 MB L2 cache configuration. The operating power consumption is 135 W.

Fujitsu designed the SPARC64 VII for increased throughput while maintaining the high performance and high reliability that have been realized with the existing SPARC64 VI. For increased throughput, the number of built-in cores has been increased from two to four, and the multi-threading mechanism to be used has been changed from VMT to SMT. The L2 cache is configured to be shared by the four cores, and the throughput has been doubled so that data can be supplied to the four cores. Also, especially with the field of high performance computing (HPC) in mind, an inter-core high-speed synchronization mechanism called hardware barrier has been implemented.

At the same time, by using a bus protocol that is the same as that of the existing SPARC64 VI, each CPU module can be upgraded from SPARC64 VI to SPARC64 VII.

3 SPARC64 VII micro-architecture

This section provides an overview of the micro-architecture of the SPARC64 VII. While the basic structure of the core pipeline of SPARC64 VII is the same as that of the SPARC64 VI, it uses simultaneous multi-threading (SMT) instead of vertical multi-threading (VMT) to implement multi-threading. As shown in Figure 1, two threads can be executed simultaneously on each of the four cores.

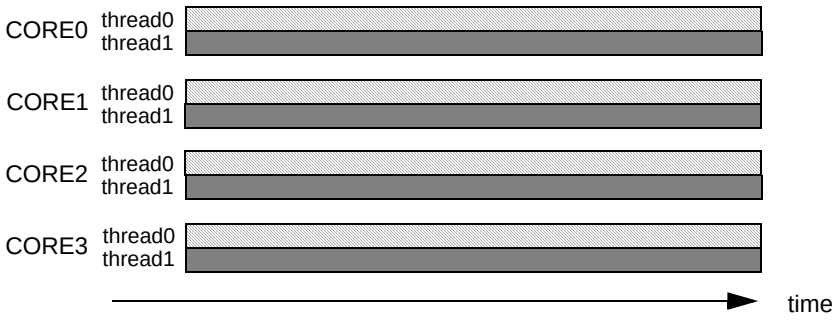

Figure 1 Multi-threading of SPARC64 VII

In SMT design, Fujitsu focused on eliminating interference between threads as much as possible. The chip is configured so that, as a rule, the hardware resources for one thread are isolated from those of the other when both threads are running. In contrast, when either thread is in the idle state, the other thread can use resources of both threads except for some resources. Thus, the chip has been designed to provide increased performance of single-thread operation.

In the structure, both threads share the core pipeline. However, it is controlled so that, even if a pipeline is stalled in one thread, the processing in the other thread is not clogged up. In the instruction fetch stage, instruction decoding stage, or commit stage, either thread is selected in each cycle.

3.1 Details of the micro-architecture

Details of the micro-architecture are outlined below.

As shown in Figure 2, a core of the SPARC64 VII is divided into the instruction fetch block and instruction execution block. The instruction fetch block includes the primary cache dedicated for instructions (L1I cache), and the instruction execution block includes the primary cache dedicated for operands (L1D cache).

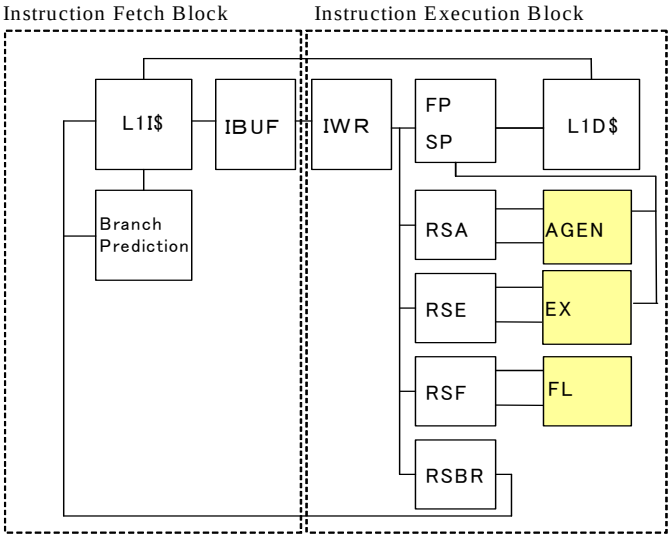

Figure 2 Functional diagram of SPARC64 VII core

3.2 Instruction Fetch Block

The instruction fetch block, which operates independently of the instruction execution block, takes a series of instructions into the instruction buffer (IBUF), which are expected to be executed according to branch prediction. The IBUF has a capacity of 256 bytes, and can store up to 64 instructions. When both threads are running, the IBUF is divided evenly for each thread.

When the instruction execution is stalled, instruction fetch continues until the IBUF becomes full. In contrast, when instruction fetch pauses for some reason such as a cache miss, instructions can be taken from the IBUF and execution can continue as long as the IBUF includes instructions. Instruction fetch can be started in every cycle, and 32 bytes, which comprise eight instructions, are fetched at one time. The throughput of instruction execution is up to four instructions per cycle, while twice the throughput of instruction execution is assured for instruction fetch. The IBUF conceals the latency of the large-capacity primary instruction cache by separating instruction fetch and instruction execution from each other (decoupling).

3.3 Instruction Execution Block

Instruction Decode and Issue

In the Instruction decode and instruction issue stages, the four instructions in the IWR are decoded simultaneously, and resources required for execution (various reservation stations, fetch port and store port, and register update buffer) are determined. Then, whether there are free resources for them is checked. If there are free resources, they are allocated and given instruction identifications (IID) ranging from 0 to 63. Then, the instruction is issued. In other words, the maximum number of in-flight instructions is 64. Meanwhile, when both threads are running, the maximum number of instructions for each thread is 32. In each cycle, an instruction of either thread is decoded and threads are alternately switched.

When an instruction is issued, the IWR is released. For the instruction in any slot of the IWR, there are no restrictions on the allocation of resources such as reservation stations. Also, there are no restrictions on instruction type combinations. Therefore, as long as there are free resources, instructions can be issued. Even if there is no sufficient space for four instructions, as many instructions as possible are issued in program order. As described above, by eliminating stall conditions of instruction issue as much as possible, a high multiplicity level is assured for any binary code.

Instruction Execution

A decoded instruction is registered in a reservation station. The SPARC64 VII has reservation stations for integer operation (reservation stations for execution: RSE) and reservation stations for floating point operation (reservation stations for floating point: RSF). The RSEs and RSFs are divided into two queues for the execution unit. In other words, four reservation stations for operation are provided. They are RSEA, RSEB, RSFA, and RSFB. Each instruction stored in a reservation station is dispatched to the execution unit that corresponds to the reservation station in the order in which source operands are prepared for instructions. Therefore, four operations can be dispatched simultaneously. Basically, the oldest instruction that can be dispatched (oldest ready) is selected from the instructions in a reservation station. However, in cases where a register to be updated by a load instruction is used as a source operand for an operation, the instruction is speculatively dispatched before the result of the load instruction is obtained. Then, in the execution stage, whether the speculative dispatch has been successful is determined. This is called speculative dispatch. Use of speculative dispatch conceals the latency of the pipeline for cache access, increasing the use efficiency of the execution unit.

In addition to the above described RSEs and RSFs, there are other reservation stations, which are reservation stations for branch instructions (reservation station for branch: RSBR) and reservation stations for calculating addresses for load/store instructions (reservation station for address generation: RSA).

Instruction Commit

All results of instructions that are executed out of order are once stored in the GUB and FUB work registers, which not visible to software. To assure the instruction order in a program, registers such as GPR and FPR and memory are updated in program order in the commit stage. In addition, control registers such as the PC are also updated at the same time in the commit stage. As described above, precise interrupt is guaranteed, and processing in execution can always be canceled. The method above is called a synchronous update method, which does not only make it easier to re-execute instructions due to a branch prediction miss, but also contributes to increased RAS as explained in a later chapter. The maximum number of instructions that can be committed at one time is four. The instruction commit stage is shared by the two threads, and either thread is selected in each cycle to perform commit processing.

4 Cache System

The cache memory of the SPARC64 VII has a two-layer structure, consisting of a middle-capacity primary cache (L1 cache) and a high-capacity secondary cache (L2 cache).

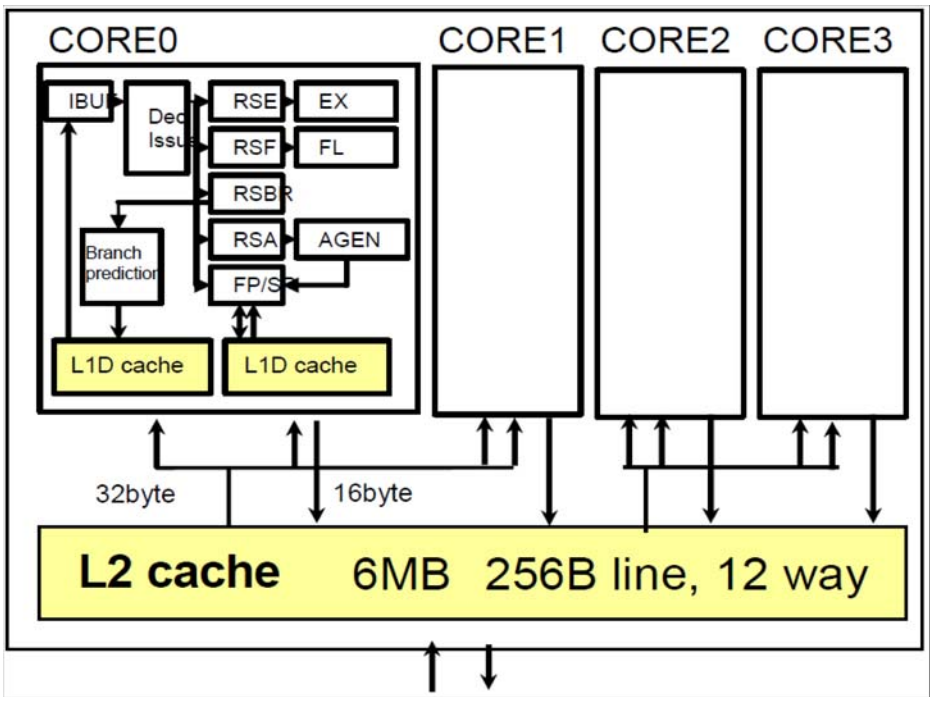

Figure 3 SPARC64 VIIcore and cache

The L1 cache consists of a cache dedicated for instructions (L1I cache) and a cache dedicated for operands (L1D cache). Each of these caches has the capacity of 64 kilobytes, uses the two-way set associative method, and has the block size of 64 bytes. The L1D cache is divided into eight banks on the four-byte address boundaries, and two operands can be accessed at one time. The L1 cache uses virtual addresses for cache indexes and physical addresses for cache tags (virtually indexed physically tagged: VIPT). In the VIPT method, consistency may be lost if the same area of memory is accessed using different virtual addresses because different indexes are used for registration (synonym problem). Through coordination with the L2 cache, the SPARC64 VII resolves the synonym problem with hardware.

The L2 cache has a maximum capacity of 6 megabytes, uses a 12-way set associative method, has a block size of 256 bytes, and is shared by the four cores. By adopting two-bank interleaved structure, 64 bytes of data can be read in each cycle. The bus for sending data that is read from the L2 cache to the L1 cache has a width of 32 bytes per two cores, and the bus for sending data from the L1 cache to L2 cache has a width of 16 bytes per one core.

The cache update policies of L1 cache and L2 cache are both write-back. That is, the store data is written into only one cache hierarchy. In the write-back method, cache missed lines are always loaded on to the cache memory, so that the store operations can complete with updating one cache hierarchy. In the write-back method, it is necessary to bring old data on the memory onto the cache even for store operation when a cache error occurs, but store operation is completed only on the cache when a cache hit occurs. In general, because the frequency of the store operation is quite high, the write-back method has an advantage because it can reduce intercache traffic and memory access traffic.

Meanwhile, because the write-back method keeps the latest data in the cache, if an error occurs in the relevant processor, there is a risk that the error may affect not only the internal operation of the processor, but also the entire system. The SPARC64 VII has powerful RAS functions to cope with this problem.

Also, a new hardware barrier mechanism has been implemented in the SPARC64 VII. The hardware barrier mechanism synchronizes the cores in a CPU chip with each other, and faster synchronization processing can be implemented compared with a conventional synchronization process realized by software. This mechanism is especially useful in the HPC area.

5 Reliability, Availability, Serviceability (RAS) Functions

In the SPARC64 VII, RAS functions comparable to mainframe computers have been implemented. With these RAS functions, errors are reliably detected, their effect is kept within a limited range, recovery processing is tried, error logs are recorded, software is notified, and so forth. In other words, the basics of RAS functions are thoroughly implemented. Through the implementation of the RAS functions, the SPARC64 VII provides high reliability, high availability, high serviceability, and high data integrity as a processor for mission-critical UNIX servers.

5.1 RAS of Internal RAMs

Among the parts of a processor, the error occurrence frequency is highest in RAM. In the SPARC64 VII, because any one-bit error in RAM can automatically be corrected by hardware without intervention by software, it does not affect software.

Type		Error detection method Protection method	Error correction method
L1 Instruction Cache	Data	Parity	Invalidation and reread
	Tag	Parity + Duplication	Rewrite of duplicated data
L1 Data Cache	Data	SECDED ECC	One-bit error correction using ECC
	Tag	Parity + Duplication	Rewrite of duplicated data
L2 Cache	Data	SECDED ECC	One-bit error correction using ECC
	Tag	SECDED ECC	One-bit error correction using ECC
Instruction TLB		Parity	Invalidation
Data TLB		Parity	Invalidation
Branch History		Parity	Recovery from branch prediction failure

SECDED：Single Error Correction Double Error Detection

For the L1 cache, L2 cache, and TLB, degradation can be performed separately in way units. Error occurrence counts are counted for each function unit. When an error occurrence count per unit time exceeds the upper limit, degradation is performed and the relevant way is not used subsequently. Hardware performs degradation automatically. At the same time, it also performs the required operation to assure the continuity of coherency automatically. More specifically, hardware automatically performs the following: 1) operation that writes back to the L2 cache all the dirty lines in the way of the L1D cache to be degraded, and 2) operation that writes back to the memory the dirty lines in the way of the L2 cache to be degraded. The degradation of a way is performed without adversely affecting software, and software operation is free from any effects except for a slowdown of processing speed.

5.2 RAS of Internal Registers and Execution Units

SPARC64 VII also provides error protection for registers and execution units, making assurance doubly sure for data integrity.

Type		Error Detection Method Protection Method
Register	Integer register	SECDED ECC
	Floating-point register	Parity
	PC, PSTATE, etc.	Parity

Type		Error Detection Method Protection Method
Execution Unit	Computation input-output register	Parity
	Addition and subtraction, division, shift, and graphic operation	Parity Prediction
	Multiplication	Parity prediction + residue check

For integer architecture registers, ECC is used from the SPARC64 VII to increase reliability. When an error occurs, the ECC circuit corrects the error. Parity bits have been added to the floating point architecture registers and other registers. Also, the parity prediction circuit, residue check circuit, and other circuits have been added to an execution unit to propagate parity information to output results. In the unlikely event that a parity error is detected, hardware automatically re-executes the instruction to attempt recovery as described below. This function is called instruction retry.

5.3 Synchronous Update Method and Instruction Retry

As shown in the explanation of the instruction execution block, the SPARC64 VII uses the synchronous update method. When an error is detected, all the instructions being executed at this time is canceled. Intermediate results before commitment can be discarded, and only results updated by instructions that have been completed without encountering any errors remain in programmable resources. Therefore, not only can the destruction of programmable resources due to errors be prevented, hardware can also perform an instruction retry after error detection. Even in case of a hang-up, because stalled instructions can be discarded once and then retried from the beginning, there is a possibility of recovery.

Instruction retry is triggered by an error and is automatically started. A retry is performed instruction by instruction to increase the chance of normal execution. When the execution is completed normally, the state automatically returns to the normal execution state. During this period, no software intervention is required, and if the instruction retry succeeds, the error does not affect software. An instruction retry is repeated until the number of retry times reaches the threshold, and when the threshold is exceeded, the occurrence of the error is reported to software by an interrupt.

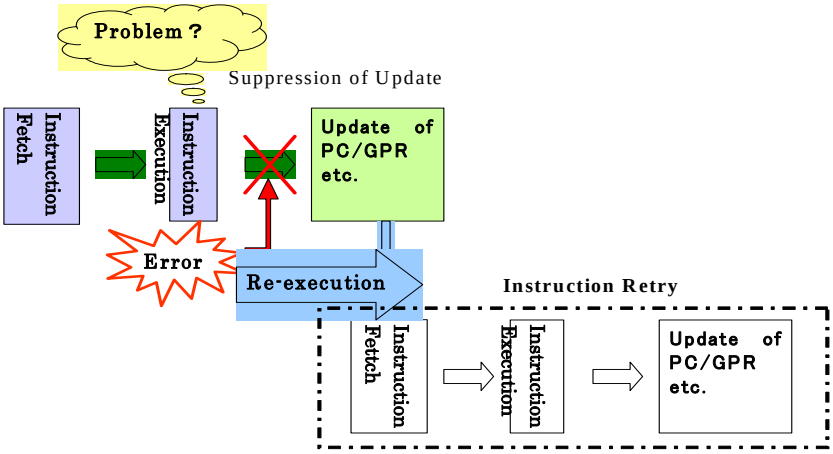

Figure 4 SPARC64 VII Instruction Retry

5.4 Increased Serviceability

The SPARC64 VII has error checking mechanisms in a variety of locations. When an error occurs, the system is notified of the error through a dedicated interface. On receipt of this notification, the system control facility (SCF) firmware collects error logs through the dedicated interface and analyzes them. This series of operations does not affect software and is performed in the background.

With the mechanism described above, a system in which the SPARC64 VII is mounted can identify the location and type of a failure quickly and accurately while continuing the operation. Thus, the system can obtain information useful for preventive maintenance to increase serviceability.

6 Concluding Remarks

SPARC64 VII has realized high throughput with quad-core and SMT design, while maintaining high performance and high reliability of existing SPARC64 VI. Also L2 cache is shared by the four cores, and an inter-core high-speed synchronization mechanism called hardware barrier has been implemented.

SPARC64 VII combines high performance and high throughput. The processor is able to demonstrate its full potential in both business and technical computing fields.

7 Reference

Fujitsu Limited: SPARC64V Processor For UNIX server, Aug 2004