Loongson 3A3000 / 3B3000 Processor User Manual

Volume II

GS464E processor core V1.2

2017 Nian ***12*** Yue

Loongson Zhongke Technology Co., Ltd.

Copyright Notice

The copyright of this document belongs to Loongson Zhongke Technology Co., Ltd. and reserves all rights. Without written permission, no company or individual shall

Any part of the file is open, reprinted or otherwise distributed to third parties. Otherwise, it will be held accountable.

Disclaimer

This document only provides periodic information, and the content can be updated at any time according to the actual situation of the product without notice. If due to improper use of docume

The company does not assume any responsibility for the direct or indirect losses caused.

Loongson Zhongke Technology Co., Ltd.

Loongson Technology Corporation Limited

Address: Building 2, Longxin Industrial Park, Zhongguancun Environmental Protection Technology Demonstration Park, Haidian District, Beijing

Building No. 2, Loongson Industrial Park,

Zhongguancun Environmental Protection Park, Haidian District, Beijing

Telephone (Tel): 010-62546668

Fax: 010-62600826

Reading guide

"Godson 3A3000 / 3B3000 Processor User Manual" is divided into two volumes.

Volume 2 of the "Loongson 3A3000 / 3B3000 Processor User Manual" introduces Loongson 3A3000 / 3B3000 processing in detail from the perspective of system software developers The GS464E high-performance processor core used in the processor.

Introduction to the meaning of special formats

1. In this document, the description of a certain field of the CP0 control register is in the format of Reg.Field, where Reg is the help of the control register. Notation, Field is the mnemonic of the field to be described in this register. For example, EBase.CPUNum represents the CPUNum of the EBase control register area.

2. The description of the data content interception part in this document adopts the format of [m: n] or m..n , which means that the content from the nth to the mth is selected Bit. m and n take values from 0, m≥n.

Page 4

Version history

Document update record

Document name:	Loongson 3A3000 / 3B3000 Processor User Manual --Volume II
version number	V1.2
founder:	Chip R & D Department
Creation Date:	2017-12-20

Update history

Serial number	Updated	updater	version number	update content
1	2017/04/07	Chip R & D Department	V1.0	The first draft is completed.
2	2017/11/29	Chip R & D Department	V1.1	1. Fix the errors in the instruction list in Chapter 2. 2. The 2.3.1 section adds a list of "DSP instructions no longer supported by MIPS"
3	2017/12/20	Chip R & D Department	V1.2	Correct the errors of 8 instructions in the instruction list in Chapter 2.

Technical Support

You can submit questions about product use to our company via email or the problem feedback website and obtain technical support.

After-sales service email:service@loongson.cn

Question feedback URL:http://bugs.loongnix.org/

Page 5

table of Contents

Page 8

Figure catalog

Table directory

1 Overview of the processor core structure

Godson GS464E processor core (hereinafter referred to as "GS464E") is a general-purpose RISC processor core that implements the LoongsonISA instruction set,
It is the optimized and upgraded version of Godson GS464 processor core. The instruction pipeline of GS464E takes four instructions per clock cycle to decode,
And dynamically launched into six full-flow functional parts. Instructions can be executed out of order without guaranteeing the dependency relationship, all instructions
Submit in accordance with the order in the procedure to ensure precise exceptions and memory access order.

In the multi-launch deep pipeline processor, instruction correlation and data correlation are the primary factors affecting performance. For this reason, GS464E uses out-of-order executi
The aggressive storage system is designed to improve the efficiency of the pipeline.

Out-of-order execution technologies include register renaming technology, dynamic scheduling technology, and branch prediction technology. Register renaming to resolve WAR (after
Write) is related to WAW (write after write) and is used for accurate on-site recovery due to exception and error transfer prediction. GS464E passed two 128 items
The physical register file renames the fixed-point and floating-point registers respectively, while using 16, 32 and 32 physical register file pairs respectively
Rename HI / LO register, DSP Control register and floating point control register. Dynamic scheduling based on the order in which the instruction operands are prepared
Instead of the order in which the instructions appear in the program, the instructions are executed, reducing the blocking caused by RAW (read after write). GS464E uses one
16-item fixed-point reserve station, a 24-item floating-point reserve station and a 32-item access reserve station are used for out-of-order transmission and pass a 128-item
The Reorder queue (abbreviated as ROQ) realizes that the instructions executed out of order are submitted in the order of the program. Branch prediction by predicting whether the branch inst
Jump successfully to reduce blockage due to control related. GS464E uses 8K items of global branch history table (Global Branch History
Table, referred to as GBHT), 8K item Local Branch History Table (referred to as LBHT), 8K item global selection
Global Branch Select Table (GBSEL for short), 13-bit global history register (Global Histiry Registor, simplified
Called GHR), 1K item branch target address buffer (Branch Target Buffer, BTB for short) and 16 item return address stack (Return
Address Stack (RAS) for branch prediction, all branch instructions use a 24-item Branch Queue (BRQ) for branching
Accurate cancellation of subsequent instructions when an instruction is mispredicted.

GS464E advanced storage system design can effectively improve the efficiency of the pipeline. GS464E contains two full-function memory access parts. each
Each memory access component can independently execute Load and Store operations. GS464E is dynamically accessed through the 64-item memory access queue (CP0 Queue)
Address dependence is solved to achieve out-of-order execution of memory access operations and non-blocking cache. GS464E adopts three-level cache storage structure, of which one level
Cache is composed of 64KB instruction Cache and 64KB data Cache, both of which use 64-byte length Cache lines and four-way group associative structure.
Each processor core contains a private 256KB secondary instruction data sharing cache, which uses a 64-byte-length Cache line and 16-way set associative structure.
Every four cores share 4MB level 3 cache. GS464E adopts a two-level TLB structure, of which the first-level TLB is divided into 64 fully-linked instructions TLB (simplified
It is called ITLB) and 32 fully connected data TLBs (referred to as DTLB). The secondary TLB contains a 64 fully connected TLB with variable page size
(Referred to as VTLB) and a fixed page size TLB (FTLB) of a 1024-item 8-way group associative structure, each item of the TLB can map an odd
Page and an even page, the page size is variable from 4KB to 1GB.

GS464E has two full-function fixed-point functions and two full-function floating-point functions. Each fixed-point component can execute branch instructions
Order, and perform fixed-point multiplication and all DSP operations in full flow. Each floating-point component can fully perform 64-bit double-precision floating-point multiplication
Add operations and execute 32-bit and 64-bit fixed-point instructions through the expansion of the fmt field of floating-point instructions.

GS464E supports the EJTAG debugging specification of MIPS, adopts the standard AXI interface, and its instruction cache implements parity check.
Data Cache implements ECC verification.

The basic pipeline of GS464E includes PC, instruction fetch, pre-decoding, decoding I, decoding II, register renaming, scheduling, transmitting, and reading register
There are 12 levels such as controller, execution, submission I, submission II, etc., and each level of pipeline includes the following operations.

- The PC pipeline stage is used to generate the program counter PC value required for the next tap.
- Use the value of the program counter PC to access the instruction cache and instruction TLB, if the instruction cache and instruction TLB are both taken
  Hit, then take eight new instructions to the instruction register IR.

1

Page 15

Loongson **3A3000 / 3B3000** Processor User Manual • Next

- The pre-decoding pipeline stage mainly decodes the branch instruction and predicts the direction of the jump.
- The decoding I pipeline stage stores the pre-decoding result in the instruction queue.
- The decoding II pipeline stage converts the four instructions in the IR into the processor's internal instruction format and sends it to the register renaming module.
- The register renaming pipeline assigns a new physical register to the logical target register and maps the logical source register to the nearest
  The physical register assigned to this logical register.
- The dispatch pipeline assigns the renamed instructions to fixed-point or floating-point reservation stations for execution, and sends them to ROQ for execution.
  Submit in order; In addition, the transfer instruction and the memory access instruction are sent to the transfer queue and the memory access queue, respectively.
- The launch pipeline stage selects from the fixed-point or floating-point reservation station for each functional unit an instruction with all operands prepared;
  When the operand is not ready for the name, wait for its operand to be ready by listening to the result bus and the forward bus.
- The read register pipeline stage reads the corresponding source operand from the physical register file for the issued instruction and sends it to the corresponding functional unit.
- The execution pipeline stage executes the instruction according to the type of instruction and writes the calculation result back to the register file; the result bus is also sent to the re
  The device renames the table to inform that the corresponding register value is ready for use.
- Submit I Pipeline level selects the instructions that can be submitted among the instructions that have been executed according to the order of the programs recorded in the Reorder
  Order, GS464E can submit up to four instructions per shot.
- The commit II pipeline stage sends the selected commit instruction to the register renaming table to confirm the renaming relationship of its destination register and
  Release the physical register originally allocated to the same logical register, and send it to the memory access queue to allow those submitted memory instructions to be written
  Cache or memory.

The above is the pipeline level of basic instructions.
Multiple shots are required during the execution phase. The basic structure of GS464E is shown in the figure below.

2

Page 16

1.1 A quick overview of the processor core structure parameters

Instruction Set	MIPS64r2 compatible LoongISA instruction set
Pipeline level depth	12 levels
Fetch width	8 instructions / cycle
Launch width	(2 fixed point + 2 floating point + 2 memory access) / cycle
First order **cache** specification	Each core is private, 4 channels × 16KB / channel
Level 1 data **cache** specifications	Each core is private, 4 channels × 16KB / channel
Secondary **Cache** specifications	Private per core, 16 channels × 16KB / channel
Three-level **cache** specifications	Multi-core sharing, 16 channels × 128KB / channel
TLB capacity	64 items VTLB + 2048 items FTLB
TLB supports page size	4 i KB (i = 1, 2, ……, 10)

3

Loongson **3A3000 / 3B3000** Processor User Manual • Next

2 Instruction Set Overview

GS464E implements the MIPS64 compatible basic part, MIPS64 DSP instruction (MIPS64) in LoongISA instruction set v1.00 DSP Module) and the 64-bit Loongson General Extended Instructions (LoongEXT64).

2.1 MIPS64 compatible general instruction list

GS464E implements all the general instructions that must be implemented and a small number of optional implementations required in the MIPS64 specification. The general instructions are divided into the following categories according to their functions:

- Fetch instruction
- Arithmetic instruction
- Transfer instruction
- Other instructions
- Coprocessor 0 instruction

These instructions are listed below by category.

2.1.1 Memory access instruction

The MIPS architecture uses a load / store architecture. All operations are performed on the register, only the memory access instruction can access the data in the main memory OK visit. The memory access instruction includes reading and writing of various width data, unsigned reading, unaligned memory access and atomic memory access.

Table **2-1 CPU** instruction set: memory access instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
LB	Fetch bytes	MIPS32
LBU	Take unsigned byte	MIPS32
LH	Take half word	MIPS32
LHU	Take unsigned halfword	MIPS32
LW	Fetch word	MIPS32
LWU	Take unsigned word	MIPS32
LWL	Take the left part of the word	MIPS32
LWR	Take the right part of the word	MIPS32
LD	Take double word	MIPS64
LDL	Take the left part of the double word	MIPS64
LDR	Take the right part of the double word	MIPS64
LL	Take the address of the sign	MIPS32
LLD	Take the double word address of the mark	MIPS64
SB	Save byte	MIPS32
SH	Half word	MIPS32
SW	Save word	MIPS32
SWL	Save the left part	MIPS32
SWR	Save the right part	MIPS32

5

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SD	Save double word	MIPS64
SDL	Save double word left	MIPS64
SDR	Save double word right	MIPS64
SC	Save under the conditions	MIPS32
SCD	Save double word	MIPS64

2.1.2 Operation instructions

Operational instructions complete the arithmetic, logic, shift, multiplication, and division of register values. Operational instructions include register instruction lattice (R-type, operands and operation results are stored in registers) and immediate instruction format (I-type, where one operand is a 16-bit Immediate)

Table **2-2** Operation instructions: arithmetic instructions (**ALU** immediate data)

Instruction mnemonic	Instruction function brief	**ISA** compatible category
ADDI	Add immediate	MIPS32
DADDI	Add double word immediately	MIPS64
ADDIU	Add unsigned immediate	MIPS32
DADDIU	Add unsigned double word immediately	MIPS64
SLTI	Less than immediate setting	MIPS32
SLTIU	Unsigned less than immediate setting	MIPS32
ANDI	And immediate	MIPS32
ORI	Or immediate	MIPS32
XORI	XOR immediate	MIPS32
LUI	Take immediate to high	MIPS32

Table **2-3** Operation instructions: arithmetic instructions (**3** operands)

Instruction mnemonic	Instruction function brief	**ISA** compatible category
ADD	plus	MIPS32
DADD	Double word plus	MIPS64
ADDU	Unsigned plus	MIPS32
DADDU	Unsigned double word plus	MIPS64
SUB	Less	MIPS32
DSUB	Double word minus	MIPS64
SUBU	Unsigned minus	MIPS32
DSUBU	Unsigned double word subtraction	MIPS64
SLT	Less than setting	MIPS32
SLTU	Unsigned less than setting	MIPS32
AND	versus	MIPS32
OR	or	MIPS32
XOR	XOR	MIPS32
NOR	NOR	MIPS32

Table **2-4** Operation instructions: arithmetic instructions (**2** operands)

Instruction mnemonic	Instruction function brief	**ISA** compatible category
CLO	Leading 1 number	MIPS32
DCLO	Leading 1 word	MIPS64
CLZ	Leading 0 number	MIPS32
DCLZ	Leading double words	MIPS64
WSBH	Byte swap in halfword	MIPS32 R2
DSHD	Half-word swap	MIPS64 R2
DSBH	Byte swap in halfword	MIPS64 R2
SEB	Byte sign extension	MIPS32 R2
SEH	Half-character extension	MIPS32 R2
INS	Bit insertion	MIPS32 R2
EXT	Bit extraction	MIPS32 R2
DINS	Double word insertion	MIPS64 R2
DINSM	Double word insertion	MIPS64 R2
DINSU	Double word insertion	MIPS64 R2
DEXT	Double word bit extraction	MIPS64 R2
DEXTM	Double word bit extraction	MIPS64 R2
DEXTU	Double word bit extraction	MIPS64 R2

Table **2-5** Operation instructions: multiplication and division instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
MUL	Multiply to general register	MIPS32
MULT	Multiply	MIPS32
DMULT	Double word multiplication	MIPS64
MULTU	Unsigned multiplication	MIPS32
DMULTU	Unsigned double word multiplication	MIPS64
MADD	Multiply-add	MIPS32
MADDU	Unsigned multiply-add	MIPS32
MSUB	Multiplication and subtraction	MIPS32
MSUBU	Unsigned multiplication and subtraction	MIPS32
DIV	except	MIPS32
DDIV	Double word division	MIPS64
DIVU	Unsigned division	MIPS32
DDIVU	Unsigned double word division	MIPS64
MFHI	Fetch data from HI register to general register	MIPS32
MTHI	Store data from general register to HI register	MIPS32
MFLO	Get data from LO register to general register	MIPS32
MTLO	Store data from general register to LO register	MIPS32

Table **2-6** Operation Instructions: Shift Instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SLL	Logical shift left	MIPS32
SRL	Logical shift right	MIPS32
SRA	Arithmetic shift right	MIPS32
SLLV	Variable logical shift left	MIPS32
SRLV	Variable logical shift right	MIPS32
SRAV	Variable arithmetic shift right	MIPS32
ROTR	Rotate right	MIPS32 R2
ROTRV	Variable loop right shift	MIPS32 R2
DSLL	Double word logical shift left	MIPS64
DSRL	Double-word logical shift right	MIPS64
DSRA	Double-word arithmetic shift right	MIPS64
DSLLV	Variable double word logical shift left	MIPS64
DSRLV	Variable double word logical shift right	MIPS64
DSRAV	Variable double word arithmetic shift right	MIPS64
DSLL32	Shift left plus 32 double-word logical shift left	MIPS64
DSRL32	Shift right plus 32 double-word logical shift right	MIPS64
DSRA32	Double-word arithmetic shift right by shift amount plus 32	MIPS64
DROTR	Double word rotation right	MIPS64 R2
DROTR32	Double-word shift plus 32 shift right	MIPS64 R2
DROTRV	Double word variable circular right shift	MIPS64 R2

2.1.3 Jump and branch instructions

Jump and branch instructions can change the control flow of the program, including the following four types:

- PC relative conditional branch
- PC unconditional jump
- Register absolute jump
- Procedure call

In the MIPS architecture, all transfer instructions are followed by a delay slot instruction. Likely the delay slot of the transfer instruction is only executed when the transfer is successful, Non-Likely branch instruction delay slot instructions are always executed. The return address of the procedure call instruction is stored in register No. 31 by default, according to Register No. 31 will be considered to return from the called procedure.

Table **2-7** Jump and branch instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
J	Jump	MIPS32
JAL	Immediate call procedure	MIPS32
JR	Jump to the instruction pointed to by the register	MIPS32
JR.HB	Jump to the instruction pointed to by the register	MIPS32 R2
JALR	Register call process	MIPS32
JALR.HB	Register call process	MIPS32 R2
BEQ	Jump if equal	MIPS32
BNE	Jump if not equal	MIPS32

Instruction mnemonic	Instruction function brief	**ISA** compatible category
BLEZ	Less than or equal to 0	MIPS32
BGTZ	Greater than 0 jump	MIPS32
BLTZ	Less than 0 jump	MIPS32
BGEZ	Greater than or equal to 0	MIPS32
BLTZAL	Less than 0 calling procedure	MIPS32
BGEZAL	Greater than or equal to 0 calling procedure	MIPS32
BEQL	Likely jumps Likely	MIPS32
BNEL	If not equal, Likely jump	MIPS32
BLEZL	Less than or equal to 0 then Likely jump	MIPS32
BGTZL	If it is greater than 0, Likely jumps	MIPS32
BLTZL	If less than 0, Likely jumps	MIPS32
BGEZL	Greater than or equal to 0, Likely jump	MIPS32
BLTZALL	If less than 0, Likely calls the procedure	MIPS32
BGEZALL	Greater than or equal to 0 then Likely calls the procedure	MIPS32

2.1.4 Coprocessor **0** instruction

The processor manages memory and handles exceptions through the No. 0 coprocessor (CP0) register.

Table **2-8** Coprocessor **0** instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
DMFC0	Fetch double word from CP0 register	MIPS64
DMTC0	Write double word to CP0 register	MIPS64
MFC0	Get word from CP0 register	MIPS32
MTC0	Write to CP0 register	MIPS32
TLBR	Read indexed TLB entries	MIPS32
TLBWI	Indexed TLB entry	MIPS32
TLBWR	Write random TLB entries	MIPS32
TLBP	Search for matches in TLB	MIPS32
TLBINV	Invalid specified TLB entry	MIPS32
TLBINVF	Invalid all TLB entries	MIPS32
CACHE	Cache operation	MIPS32
ERET	Exception return	MIPS32
DERET	Debug exception return	MIPS32
DMFGC0	Fetch double word from CP0 register of virtual machine	MIPS64
DMTGC0	Write double words to the CP0 register of the virtual machine	MIPS64
MFGC0	Fetch word from virtual machine CP0 register	MIPS32
MTGC0	Write to the CP0 register of the virtual machine	MIPS32
DI 1	Disable interrupt	MIPS32 R2
EI 2	Allow interrupt	MIPS32 R2

1 See section 2.4.12 for details
2 See section 2.4.12 for details

2.1.5 Other instructions

In MIPS64, there are other instructions besides those listed above.

Table **2-9** Other instructions: special instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SYSCALL	System call	MIPS32
BREAK	Breakpoint	MIPS32
HYPCALL	Virtual machine system call	MIPS32
SYNC	Synchronize	MIPS32
SYNCI	Synchronous instruction cache	MIPS32 R2
RDPGPR	Read shadow register	MIPS32
WRPGPR	Write shadow register	MIPS32
SDBBP	Debug breakpoint	MIPS32
RDHWR	Read the machine state in user mode	MIPS32
WAIT	Waiting for instruction	MIPS32

Table **2-10** Other instructions: Exception caught instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
TGE	Greater than or equal to	MIPS32
TGEU	Unsigned number is greater than or equal to trap	MIPS32
TLT	Less than trapped	MIPS32
TLTU	Unsigned number is less than trapped	MIPS32
TEQ	Equal to falling into	MIPS32
TNE	Wait for	MIPS32
TGEI	Greater than or equal to immediate	MIPS32
TGEIU	Greater than or equal to unsigned immediate	MIPS32
TLTI	Less than immediate	MIPS32
TLTIU	Immediate number less than unsigned	MIPS32
TEQI	Equal to immediate	MIPS32
TNEI	Not equal to immediate	MIPS32

Table **2-11** Other Instructions: Conditional Move Instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
MOVF	Conditional movement when floating-point conditional false	MIPS32
MOVT	Conditional movement when floating-point condition is true	MIPS32
MOVN	Conditional move when general register is not 0	MIPS32
MOVZ	Conditional move when general register is 0	MIPS32

Table **2-12** Other instructions: Prefetch instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
PREF	Prefetch instruction	MIPS32

10

Instruction mnemonic	Instruction function brief	**ISA** compatible category
PREFX	Prefetch instruction	MIPS32

2.2 Overview of **MIPS64** compatible floating point instruction set

The floating-point instructions implemented by GS464E are compatible with MIPS64 specification 1 , all floating-point instructions are in Floating Point Unit (Floting Point Unit, abbre
FPU).

2.2.1 FPU data type

The floating-point coprocessor supports both floating-point and fixed-point data types.

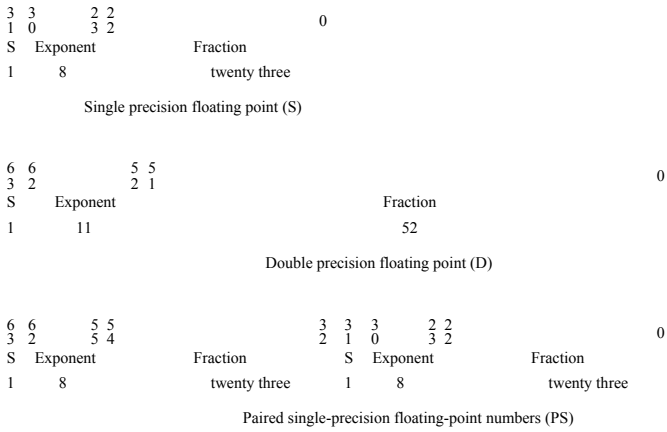
Floating-point data type

The floating-point data types supported by the floating-point coprocessor are:

- 32-bit single-precision floating point (Single-Precisions, S)
- 64-bit double-precision floating-point number (Double-Precisions, D)
- 64-bit paired single-precision floating point numbers (Paired Single-Precisions, PS)

Floating-point coprocessor for single-precision floating-point numbers (including each single-precision floating-point number in pairs) and double-precision floating-point number
Comply with ANSI / IEEE 754-1985 binary floating point arithmetic standard. The 32-bit single-precision format includes a 24-bit "symbol + amplitude"
Represented decimal field (S + F) and an 8-bit exponential field (E); 64-bit double-precision format includes a 53-bit "sign + amplitude"
Represented decimal field (S + F) and an 11-bit exponential field (E); 64-bit double precision (PS) format contains two single-precision floating-point formats.
The three types of data formats are shown in Figure 2-1 .

Figure **2-1 FPU** floating point data format



The format of floating-point numbers consists of the following three fields:

- Symbol field, S
- Index field with offset, e = E + Bias, E is the index without offset

1 is not completely compatible, Status.FR = 0 when the floating-point register format MIPS64 specification (r1 ~ r5) are not the same, the details see 2 .2.2 section.
11

Page 25

Loongson **3A3000 / 3B3000** Processor User Manual • Next

- Decimal field, F = .b 1 b 2 … b p–1

The range of the index E without offset is an integer between Emin and Emax, plus the following two reservations
value:

- Emin-1 (used to encode 0 and denormalized numbers)
- Emax +1 (used to encode ∞ and NaN [Not a Number])

Table 2-13 defines the values of the parameters related to the floating point format.

Table **2-13** Parameters related to floating point format

parameter	Single precision	Double precision
Emax	+127	+1023
Emin	-126	-1022
Exponential offset	+127	+1023
Exponential width	8	11
Decimal width	twenty four	53

For single-precision or double-precision formats, each non-zero number that can be represented has a unique code corresponding to it. Corresponding to its encoding
The calculation method of the numerical value V is shown in Table 2-14 .

Table **2-14** Calculation method of floating point value **V**

E	≠0	S	b1	V	
E max +1		x	1	SNaN (Signaling NaN)	
		x	0	QNaN (Quiet NaN)	
E max +1	0	1	x	-∞	Negative infinity
		0	x	+ ∞	Positive infinity
[E min , E max]	x ≠ 0	1	x	-(2 E) (1.F)	Negative normalized number
		0	x	+ (2 E) (1.F)	Positive normalized number
E min -1		1	x	-(2 Emin-1) (0.F)	Negative denormalized number
		0	x	+ (2 Emin-1) (0.F)	Positive denormalized number
E min -1	0	1	x	-0	Negative 0
		0	x	+0	Positive 0

The maximum and minimum values of the two types of floating point numbers are given in Table 2-15 .

Table **2-15** Maximum and minimum floating point numbers

Types of	Single precision	Double precision
Minimum number	1.40129846e–45	4.9406564584124654e–324
Normalized number	1.17549435e–38	2.2250738585072014e–308
Maximum number	3.40282347e + 38	1.7976931348623157e + 308

Fixed-point data type

The fixed-point data supported by the FPU are signed integers, which are divided into two types:

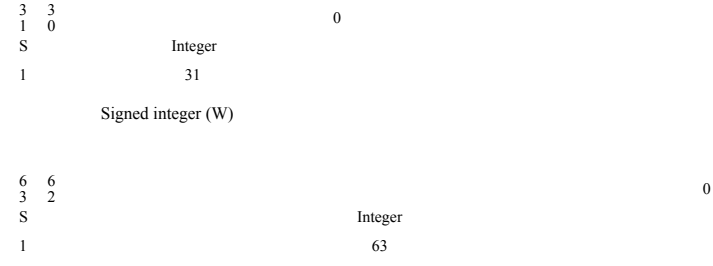
- 32-bit signed word integer (Word, W)

12

- 64-bit signed longword integer (Longword, L)

The format of the two fixed-point data types is shown in Figure 2-2 .

Figure **2-2 FPU** fixed-point data format



Signed integer (W)

Signed long integer (L)

2.2.2 Floating-point registers

The floating-point registers in the GS464E follow the MIPS R4000 / R10000 processor usage, and are slightly different from the MIPS64 specification. Control in Status When the FR bit of the control register is 0, the GS464E only has 16 32-bit or 64-bit floating-point registers, and the floating-point register number must be an even number; and MIPS64 means there are 32 32-bit floating-point registers or 16 64-bit floating-point registers. When the FR bit of the Status control register is 1, GS464E uses floating-point registers in accordance with the MIPS64 specification, each with 32 64-bit floating-point registers.

2.2.3 Floating point control register

The floating-point control registers in GS464E include:

- FIR, floating-point implementation-defined register
- FCCR, floating point condition code register
- FEXR, floating-point exception register
- FENR, floating-point enable register
- FCSR, floating point control / status register (often called FCR31)

Access to floating-point control registers does not need to be in the core state. The software accesses the floating-point control register through the CFC1 and CTC1 instructions. Float al In the point control register, access to the FCCR, FEXR, and FENR registers actually accesses certain fields of the FCSR.

Floating point implementation definition register **(FIR, CP1 Control Register 0)**

The floating-point implementation definition register is a 32-bit read-only register that contains the functions implemented by the floating-point unit, such as processor ID, revision Number and other information.

Figure 2-3 illustrates the format of the FIR re; bes the fields of the FIR register.

ter format

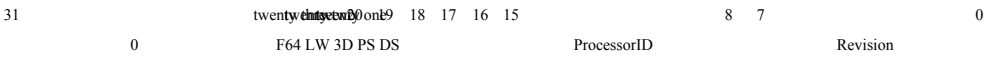


Table **2-16 FIR** register field description

Domain name	Bit	Functional description	Read / write	reset value
13				

Domain name	Bit	Functional description	Read / write	reset value
0	31..23	Read-only constant is 0.	0	0
F64	twenty two	Constant is 1, indicating that the floating-point data path is 64 bits.	R	0x1
L	twenty one	Constant is 1, indicating that the long-word (L) fixed-point data type is implemented.	R	0x1
W	20	Constant is 1, indicating that the word (W) fixed-point data type is implemented.	R	0x1
3D	19	Constant is 0, indicating that MIPS 3D ASE is not implemented.	R	0x0
PS	18	Constant is 1, indicating that a pair of single-precision floating-point data types are implemented.	R	0x1
D	17	Constant is 1, indicating that a double-precision floating-point data type is implemented.	R	0x1
S	16	Constant is 1, indicating that a single-precision floating-point data type is implemented.	R	0x1
ProccsorID	15..8	Floating-point coprocessor identification number.	R	0x05
Revision	7..0	Floating-point coprocessor version number.	R	0x01

Floating point control and status register **(FCSR, CP1 Control Register 31)**

The FCSR register is used to control the oper; ıt unit and indicate some states.

Figure 2-4 illustrates the format of the FIR re; bes the fields of the FIR register.

ster format

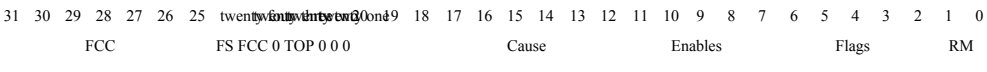


Table **2-17 FCSR** register field description

Domain name	Bit	Functional description	Read / write	reset value
FCC	31..25 , twenty three	Floating point condition codes. Record floating point comparison results for conditional jumps or transitions. When a floating point comparison operation When it occurs, the result is stored in the specified CC bit, the condition bit. If the comparison result is true, the CC bit Set to 1; otherwise set to 0.	R / W	0x0
FS	twenty four	Swipe to the 0 flag. When set to 1, the denormalized result is set to 0; otherwise the result is denormalized The number will trigger an exception for unimplemented operations.	R / W	0x0
0	twenty two	Read-only constant is 0.	0	0
TOP	twenty one	Floating register TOP mode control bit. When this bit is 1, it means that the register of floating-point instruction adopts TOP Mode encoding mode; when this bit is 0, it indicates that the register of floating-point instruction still uses the ordinary encoding mode.	R / W	0x0
0	20..18	The read-only constant is 0.	0	0

Domain name	Bit	Functional description	Read / write reset value
Cause	17..12	These bits reflect the result of the most recently executed instruction. The Causes field is in the Cause register of coprocessor 0 A logical extension, these bits indicate the exception caused by the last floating-point operation, and if the corresponding If the enable bit (Enable) is set, an interrupt or exception is generated. If generated in an instruction There is more than one exception, and each corresponding exception causes the bit to be set. The Causes field can be rewritten by each floating-point operation instruction (excluding Load, Store, and Move operations). its If software simulation is needed to complete, set the unimplemented operation bit (E) of this operation to 1, otherwise keep Is 0. The other bits are set to 1 or 0 respectively according to the IEEE754 standard to see if a corresponding exception occurs. When a floating-point exception occurs, no result will be stored, and the only affected state is the Causes field. At any time when the Cause bit and the corresponding enable bit (Enable) are both 1, a floating-point instance will be generated outer. If the floating point operation sets a Cause bit that is allowed to activate (the corresponding enable bit is 1), The processor will immediately generate an exception, which is to set the Cause and Enable bits at the same time with the CTC1 instruction 1 has the same effect.	R / W 0x0
Enables	11..7	There is no corresponding enable bit for unimplemented operation (E), if unimplemented operation is set, it always A floating-point exception is generated. Before returning from a floating point exception, the software must first use a CTC1 instruction to clear the activation To prevent repeated execution of interrupts. Therefore, programs running in user mode will never Observe that the value of the enabled Cause bit is 1; if the user-mode handler needs to obtain this information, then The contents of the Cause bit must be passed elsewhere rather than in the status register.	R / W 0x0
Flags	6..2	If the floating-point operation only sets the Cause bit which is not enabled (the corresponding enable bit is 0), there is no exception The default results defined by the IEEE754 standard are written back. In this case, the previous floating-point The exception caused by the order can be determined by reading the value in the Causes field. The flag bit is cumulative and indicates that an exception has occurred since the last time it was explicitly reset. If an IEEE754 An exception is generated, then the corresponding Flag bit is set to 1, otherwise it remains unchanged, so for floating-point operations Say these bits will never be cleared. But we can write a new value to the state through the CTC1 control instruction Register to set or clear the Flag bit.	R / W 0x0
RM	1..0	Rounding mode control field. Table 2-18 further describes the RM encoding.	R / W 0x0

Table **2-18** Rounding mode **(RM)** encoding

Rounding mode **(RM)**	Mnemonic	description
0	RN	Round the result to the direction closest to the representable number, when the two closest representable numbers are the same When approaching, round to the nearest number with the lowest bit being 0.
1	RZ	Round towards 0: round the result to the number closest to it and not greater than it in absolute value Into.
2	RP	Round towards positive infinity: round the result to the number closest to it and not less than it.
3	RM	Round towards negative infinity: round the result to the number closest to it and not greater than it.

Floating point condition code register **(FCCR, CP1 Register 25)**

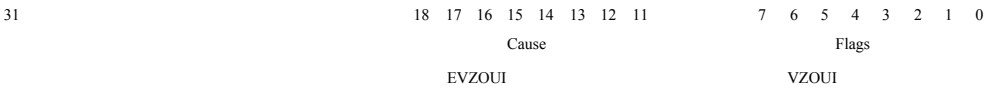
The FCCR register is another way to access the FCC field. Its content is exactly the same as the FCC bit in the FCSR. The difference is that The FCC bits in the memory are continuous. Figure 2-5 illustrates the format of the FCCR register.

Figure **2-5 FCCR** register format

Floating-point exception register **(FCCR, CP1 Register 26)**

The FEXR register is another way to access the Cause and Flags fields, and its contents are exactly the same as the corresponding fields in the FCSR. Figure 2-6 Clarified the format of the FEXR register.

Figure **2-6 FEXR** register format



Floating-point enable register **(FCCR, CP1 Register 28)**

The FENR register is another way to access the Enable, FS and RM fields . Figure 2-7 illustrates the format of the FENR register.

Figure **2-7 FENR** register format

2.2.4 Floating point exception

Floating-point exceptions occur when the FPU cannot process operands or the results of floating-point calculations in the usual way, and the FPU generates corresponding exceptions. Start the corresponding software trap or set the status flag.

The control and status registers of the FPU contain an enable bit for each exception. The enable bit determines whether an exception can cause the FPU Start an exception trap or set a status flag.

If a trap is started, the FPU keeps the state where the operation started, and the software exception processing path is started; if no trap is started, an appropriate When the value is written to the FPU target register, the calculation continues.

The FPU supports five IEEE754 exceptions:

- Inexact (I)
- Underflow (U)
- Overflow (O)
- Division by Zero (Z)
- Invalid Operation (V)

And the sixth exception:

- Unimplemented operation (E)

Unimplemented operation exceptions are used when the FPU cannot perform standard MIPS floating-point structures, including when the FPU cannot determine the correct exception be condition. This exception indicates the execution of software exception processing. The unimplemented operation exception has no enable signal and flag bit. When this exception occurs, A corresponding unimplemented exception trap occurs.

16

The five exceptions of IEEE754 (V, Z, O, U, I) all correspond to an exception trap controlled by the user. When one of the five enable bits When the bit is set, the corresponding exception trap is allowed to occur. When an exception occurs, the corresponding Cause bit is set, if the corresponding enable The Enable bit is not set, and the Exception flag is set. If the enable bit is set, the flag bit is not set and the FPU An exception is generated for the CPU. Subsequent exception handling allows this exception trap to occur.

When there is no exception trap signal, the floating-point processor takes the default method of processing and provides a substitute value for the floating-point calculation exception res Different exception types determine different default values. Lists the FPU's default handling for each IEEE exception.

Table **2-19** Default handling of floating-point exceptions

area	description	Rounding mode	Default action
I	Imprecise	Any mode	Provide rounded results
U	Underflow	RN	Set the result to 0 according to the sign of the intermediate result
		RZ	Set the result to 0 according to the sign of the intermediate result
		RP	Correct positive underflow to the smallest positive number, and negative underflow to -0
		RM	Correct negative underflow to the smallest negative number, and positive underflow to +0
O	Overflow	RN	Set the result to infinity according to the sign of the intermediate result
		RZ	Set the result to the maximum number according to the sign of the intermediate result
		RP	Correct negative underflow to the largest negative number, and positive underflow to + ∞
		RM	Correct positive underflow to the largest integer and negative underflow to -∞

Z	Divide by 0	Any mode	Provide a corresponding signed infinity number
V	Illegal operation	Any mode	Provide a Quiet Not a Number (QNaN)

The conditions that cause each exception to the FPU are described below, and the FPU response to each exception-causing condition is described in detail.

Inexact exception **(I)**

The FPU produces an inaccurate exception when the following occurs:

- Rounding result is not precise
- Rounding result overflow
- The rounding result underflows, and neither the underflow nor inaccurate enable bits are set, and the FS bit is set.

Trap enabled result: If a non-exact exception trap is enabled, the result register is not modified and the source register is retained.
Because this execution mode affects performance, imprecise exception traps are only enabled when necessary.

The result of the trap not being enabled: if no other software trap occurs, the rounding or overflow result is sent to the destination register.

Illegal operation exception **(V)**

When two or one of the operands of an executable operation is illegal, an illegal operation exception is signaled. in case
The exception is not caught, MIPS defines this result as a Quiet Not a Number (QNaN). Illegal operations include:

- Addition or subtraction: infinite subtraction. For example: (+ ∞) + (-∞) or (-∞)-(-∞)
- Multiplication: 0 × ∞, for all positive and negative numbers
- Division: 0/0, ∞ / ∞, for all positive and negative numbers
- When the operand of the comparison operation that does not handle Unordered is Unordered
- Floating point comparison or conversion of an indicator signal NaN
- Any mathematical operation on SNaN (Signaling NaN). When one of the operands is SNaN or both are SNaN

Caused this exception (MOV operations are not considered to be mathematical operations, but ABS and NEG are considered to be mathematical operations)

- Prescription: √X, when X is less than 0

The software can simulate the exception of illegal operations of other given source operands. For example, a specific function implemented by software in IEEE754: X
REM Y, here when Y is 0 or X is infinite; or overflow occurs when the floating-point number is converted to decimal, which is infinite or NaN;
Or a priori function such as ln (5) or cos -1 (3).

The result of the trap being enabled: the value of the source operand is not sent.

The result of trap disabling: If no other exception occurs, QNaN is sent to the destination register.

Except for zero exception **(Z)**

In division operations, when the divisor is 0 and the dividend is a finite non-zero data, the division by zero exception is signaled. Using software can
When other operations produce signed infinite values, the analog exception to zero is simulated, such as: ln (0), sin (π / 2), cos (0), or 0 -1 .

Trap is enabled: the result register is not modified, the source register is retained.

Trap is not enabled: If no trap occurs, the result is a signed infinite value.

Overflow exception **(O)**

When the magnitude of the floating-point result after rounding is expressed by an exponent without bounds, the limited data that is greater than the maximum target mode indicates an ov
Exceptional notification signal. (This exception sets both inexact exceptions and flags)

Trap is enabled: the result register is not modified, the source register is retained.

Trap disabled condition: if no trap occurs, the final result is determined by the rounding mode and the sign of the intermediate result.

Underflow exception **(U)**

Two related events led to underflow exceptions:

- A very small non-zero result between ± 2Emin, because the result is very small, it will cause an underflow exception afterwards.
- Denormalized number (Denormalized Number) is used to approximate the serious data distortion caused by these two small data.

IEEE754 allows many different methods to detect these events, but requires the same method to detect all operations. Small data
You can use one of the following methods to detect:

- After rounding (if a non-zero data is calculated without exponential range, it should be strictly within ± 2Emin between)
- Before rounding (if a non-zero data is calculated without exponential and precision limits, it should be strictly within ±

Between 2Emin)

The structure of MIPS requires small data to be detected after rounding. Accuracy distortion can be detected by one of the following methods:

- Distortion of non-normalized numbers (when the result produced is different from the exponent without calculation)
- Inaccurate data (when the result produced is different from the calculation result when there is no boundary between the index and the accuracy range)

The MIPS structure requires that precision distortion be detected to produce inaccurate results.

Trap is enabled: If an underflow or inaccurate exception is enabled, or the FS bit is not set, an unimplemented operation exception is generated.
If the register is not modified.

Trap is not enabled: if an underflow or inaccurate exception is not enabled and the FS bit is set, the final result is determined by the rounding mode and
The sign of the immediate result is determined.

Unimplemented operation exception **(E)**

When executing any operation code or operation format instruction reserved for later definition, the unimplemented operation in the FPU control / status register

The action bit is set and a trap is generated. The source operand and destination register remain unchanged, while the instructions are simulated in software. IEEE 754
Any exception can be generated from the simulation operation, and these exceptions can in turn be simulated. Also, when the hardware cannot perform some rare
The operation or result condition will also produce an unimplemented instruction exception. These include:

- Denormalized operand, except for comparison instructions
- Quite Not a Number operand (QNaN), except for comparison instructions
- Unnormalized number or underflow, and when the underflow or inaccurate enable signal is set and the FS bit is not set

Note: Non-normalized numbers and NaN operations only enter traps in conversion or calculation instructions, and do not enter traps in MOV instructions.

Trap is enabled: the original operation data is not sent.

Trap is not enabled: this trap cannot be disabled.

2.2.5 **MIPS64** compatible floating point instruction list

The instructions related to the MIPS64 compatible floating-point coprocessor implemented by GS464E include the following categories according to their functions:

- Arithmetic instruction
- Branch jump instruction
- Compare instructions
- Conversion instructions
- Move instruction
- Fetch instruction

These instructions are listed below by category.

Floating-point fetch instruction

Table **2-20** Floating-point branch jump instructions

Instruction mnemonic	Instruction function brief	**fmt** **SD PS LW**	**ISA** compatible category
LDC1	Access double words from within		MIPS32
LDXC1	Access double words from within by index		MIPS64
LUXC1	Access double words from within by unaligned index		MIPS64
LWC1	Access word		MIPS32
LWXC1	Access words by index		MIPS64
SDC1	Save double word to memory		MIPS32
SDXC1	Save doubleword to memory by index		MIPS64
SUXC1	Save doublewords to memory by unaligned index		MIPS64
SWC1	Save word to memory		MIPS32
SWXC1	Save words to memory by index		MIPS64

Floating point instruction

Table **2-21** Floating-point operation instructions

Instruction mnemonic	Instruction function brief	**fmt**	**ISA** compatible category

Page 33

		fmt					
		SD	**PS**	**LW**			
ABS.fmt	Absolute value	✓	✓	✓	-	-	MIPS32
ADD.fmt	addition	✓	✓	✓	-	-	MIPS32
DIV.fmt	division	✓	✓	-	-	-	MIPS32
MADD.fmt	Multiply-add	✓	✓	✓	-	-	MIPS64, MIPS32 R2
MSUB.fmt	Multiplication and subtraction	✓	✓	✓	-	-	MIPS64, MIPS32 R2
MUL.fmt	multiplication	✓	✓	✓	-	-	MIPS32
NEG.fmt	Negate	✓	✓	✓	-	-	MIPS32
NMADD.fmt	Invert after multiply-add	✓	✓	✓	-	-	MIPS64, MIPS32 R2
NMSUB.fmt	Multiply and subtract	✓	✓	✓	-	-	MIPS64, MIPS32 R2
RECIP.fmt	Find the reciprocal	✓	✓	-	-	-	MIPS64, MIPS32 R2
RSQRT.fmt	Find the reciprocal after the square root	✓	✓	-	-	-	MIPS64, MIPS32 R2
SQRT.fmt	Square root	✓	✓	-	-	-	MIPS32
SUB.fmt	Subtraction	✓	✓	✓	-	-	MIPS32

Floating-point branch instruction

Table **2-22** Floating-point branch jump instructions

Instruction mnemonic	Instruction function brief	**fmt**					**ISA** compatible category
		SD	**PS**	**LW**			
BC1F	Jump when floating-point condition bit is false						MIPS32
BC1FL	Likely jump when floating-point condition bit is false						MIPS32
BC1T	Jump when the floating-point condition bit is true						MIPS32
BC1TL	Likely jump when floating-point condition bit is true						MIPS32

Floating-point comparison instructions

Table **2-23** Floating-point branch jump instructions

Instruction mnemonic	Instruction function brief	**fmt**					**ISA** compatible category
		SD	**PS**	**LW**			
C.cond.fmt	Compare floating-point values and concatenate condition bits	✓	✓	✓	-	-	MIPS32

Floating-point conversion instruction

Table **2-24** Floating-point branch jump instructions

Instruction mnemonic	Instruction function brief	**fmt**					**ISA** compatible category
		SD	**PS**	**LW**			
ALNV.PS	Variable floating point alignment	-	-	✓	-	-	MIPS64
CEIL.L.fmt	Floating point conversion to 64-bit fixed point, rounded up	✓	✓	-	-	-	MIPS64
CEIL.W.fmt	Floating point conversion to 32-bit fixed point, rounded up	✓	✓	-	-	-	MIPS64
CVT.D.fmt	Floating or fixed point conversion to double precision floating point	-	✓	✓	-	-	MIPS32
CVT.L.fmt	Convert floating-point values to 64-bit fixed-point	✓	✓	-	-	-	MIPS64
CVT.PS.S	Convert two floating-point values to floating-point pairs	✓	-	-	-	-	MIPS64

20

Page 34

Instruction mnemonic	Instruction function brief	**fmt**					**ISA** compatible category
		SD	**PS**	**LW**			
CVT.S.PL	Converts the low-order bits of floating-point pairs to single-precision floating-point	-	-	✓	-	-	MIPS64

CVT.S.PU	Convert high-order bits of floating-point pairs to single-precision floating-point	-	-	✓	-	MIPS64
CVT.S.fmt	Floating or fixed point conversion to single precision floating point	✓	-	✓	✓	MIPS32
CVT.W.fmt	Convert floating-point values to 32-bit fixed-point	✓	✓	-	-	MIPS32
FLOOR.L.fmt	Floating point conversion to 64-bit fixed point, round down	✓	✓	-	-	MIPS64
FLOOR.W.fmt	Floating point conversion to 32-bit fixed point, round down	✓	✓	-	-	MIPS64
PLL.PS	Merge the lower bits of two floating-point pairs into a new floating-point pair	-	-	✓	-	MIPS64
PLU.PS	Combine the low and high bits of two floating-point pairs into a new floating-point pair	-	-	✓	-	MIPS64
PUL.PS	Combine the high and low bits of two floating-point pairs into a new floating-point pair	-	-	✓	-	MIPS64
PUU.PS	Merge the high-order bits of two floating-point pairs into a new floating-point pair	-	-	✓	-	MIPS64
ROUND.L.fmt	Round floating-point numbers to 64-bit fixed point	✓	✓	-	-	MIPS64
ROUND.W.fmt	Round floating-point numbers to 32-bit fixed-point	✓	✓	-	-	MIPS32
TRUNC.L.fmt	Round floating-point numbers to 64-bit fixed points with small absolute values	✓	✓	-	-	MIPS64
TRUNC.W.fmt	Round floating-point numbers to 32-bit fixed points with small absolute values	✓	✓	-	-	MIPS32

Floating-point move instruction

Table **2-25** Floating-point branch jump instructions

Instruction mnemonic	Instruction function brief	**fmt**				**ISA** compatible category
		SD	**PS**	**LW**		
CFC1	Read floating point control register to GPR					MIPS32
CTC1	Write floating point control register to GPR					MIPS32
DMFC1	Copy double words from FPR to GPR					MIPS64
DMTC1	Copy double words from GPR to FPR					MIPS64
MFC1	Copy low words from FPR to GPR					MIPS32
MFHC1	Copy high words from FPR to GPR					MIPS32 R2
MOV.fmt	Copy FPR	✓	✓	✓	-	MIPS32
MOVF.fmt	FPR when floating-point fake	✓	✓	✓	-	MIPS32
MOVN.fmt	Copy FPR when GPR is not 0	✓	✓	✓	-	MIPS32
MOVT.fmt	Floating point real time copy FPR	✓	✓	✓	-	MIPS32
MOVZ.fmt	Copy FPR when GPR is 0	✓	✓	✓	-	MIPS32
MTC1	Copy low words from GPR to FPR					MIPS32
MTHC1	Copy high words from GPR to FPR					MIPS32 R2

2.3 Overview of **MIPS64 DSP** instruction set

GS464 is compatible with MIPS64 DSP ASE (version 2.34). For a detailed description of the DSP instructions implemented, please refer to the "MIPS® Architecture for Programmers VolumeIV-e: The MIPS® DSP Application-Specific Extension to the MIPS64® Architecture "(r2.34).

Loongson **3A3000 / 3B3000** Processor User Manual • Next

2.3.1 MIPS64 DSP ASE compatible instruction list

The instructions related to the MIPS64 DSP ASE compatible floating-point coprocessor implemented by GS464E include the following categories according to their functions:

- Operation instruction
- Shift instruction based on general register
- Multiplication instructions
- Bit manipulation instructions
- Compare-extract instruction
- Accumulator operation access instruction
- DSP control register access instruction
- Memory access instruction with index register
- Branch instruction
- DSP instructions no longer supported by MIPS

These instructions are listed below by category.

Operation instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
ADDQ.PH	Vector (2 rightmost) decimal halfword plus	MIPS DSP
ADDQ_S.PH	Vector (2 rightmost) decimal halfword saturation plus	MIPS DSP
ADDQ_S.W	Signed word saturation plus	MIPS DSP
ADDU.QB	Vector (4 rightmost) small unsigned plus	MIPS DSP
ADDU_S.QB	Vector (4 rightmost) small unsigned saturation addition	MIPS DSP
ADDUH.QB	Vector unsigned (4 rightmost) bytes are added, the result is divided by 2, and the result is sign extended	MIPS DSP R2
ADDUH_R.QB	Vector unsigned (4 rightmost) bytes are rounded and added, the result is divided by 2, and the result is sign extended	MIPS DSP R2
ADDU.PH	Vector (2 rightmost) decimal halfword unsigned plus	MIPS DSP R2
ADDU_S.PH	Vector (2 rightmost) decimal halfword unsigned saturation plus	MIPS DSP R2
ADDQH.PH	Vector (the rightmost 2) halfword addition, the result is divided by 2, the sign extension of the result	MIPS DSP R2
ADDQH_R.PH	Vector (the rightmost 2) half-word rounding and addition, the result is divided by 2, the result is sign expansion	MIPS DSP R2
ADDQH.W	The rightmost word of the vector is added, the result is divided by 2, and the sign of the result is expanded	MIPS DSP R2
ADDQH_R.W	The rightmost word of the vector is rounded and added, the result is divided by 2, and the sign of the result is expanded	MIPS DSP R2
SUBQ.PH	Vector (2 rightmost) decimal half-word minus	MIPS DSP
SUBQ_S.PH	Vector (2 rightmost) decimal halfword saturation minus	MIPS DSP
SUBQ_S.W	Signed word saturation minus	MIPS DSP
SUBU.QB	Vector (4 rightmost) unsigned minus numbers	MIPS DSP
SUBU_S.QB	Vector (4 rightmost) small unsigned saturation minus	MIPS DSP
SUBUH.QB	Vector unsigned (4 rightmost) bytes minus, result divided by 2, result sign extended	MIPS DSP R2
SUBUH_R.QB	Vector unsigned (4 rightmost) bytes are rounded and subtracted, the result is divided by 2, and the result is sign extended	MIPS DSP R2
SUBU.PH	Vector (2 rightmost) decimal halfword unsigned minus	MIPS DSP R2
SUBU_S.PH	Vector (2 rightmost) decimal halfword unsigned saturation minus	MIPS DSP R2
SUBQH.PH	Vector (the rightmost 2) half word subtraction, the result is divided by 2, the sign of the result is expanded	MIPS DSP R2

twenty two

Page 36

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SUBQH_R.PH	Vector (the rightmost 2) halfword rounding minus, the result is divided by 2, the result sign expansion	MIPS DSP R2
SUBQH.W	The rightmost word of the vector is subtracted, the result is divided by 2, and the sign of the result is expanded	MIPS DSP R2
SUBQH_R.W	The rightmost word of the vector is rounded down, the result is divided by 2, and the sign of the result is expanded	MIPS DSP R2
ADDSC	Signed word add and set carry	MIPS DSP
ADDWC	Signed word with carry plus	MIPS DSP
MODSUB	Use index value for pattern subtraction	MIPS DSP
RADDU.W.QB	(Far right) 4-byte unsigned accumulation	MIPS DSP
ABSQ_S.QB	Vector finds (the rightmost 4) byte absolute value, and do saturation operation, the sign expansion of the result	MIPS DSP R2
ABSQ_S.PH	The vector finds (the rightmost 2) half-word absolute values, and performs a saturation operation, resulting in sign expansion	MIPS DSP
ABSQ_S.W	The vector finds the absolute value of the (rightmost) word and performs a saturation operation, resulting in sign expansion	MIPS DSP
PRECR.QB.PH	Vector integer precision reduction, from (rightmost two) halfwords to 4 bytes	MIPS DSP R2
PRECRQ.QB.PH	Vector decimal precision reduction, from (rightmost two) halfwords to 4 bytes	MIPS DSP
PRECR_SRA.PH.W	Vector right shift integer precision reduction, from (rightmost) word to two halfwords, resulting in sign expansion	MIPS DSP R2
PRECR_SRA_R.PH.W	The vector is shifted to the right by integer precision reduction, from the (rightmost) word to two halfwords, and rounded, resulting in Number extension	MIPS DSP R2
PRECRQ.PH.W	Vector decimal precision reduction, from (rightmost) word to (two) halfwords	MIPS DSP
PRECRQ_RS.PH.W	Decrease the precision of vector decimals, from (rightmost) word to (two) halfwords, and do saturation and truncation Enter	MIPS DSP
PRECRQU_S.QB.PH	Vector decimal precision reduction, from (rightmost two) halfwords to 4 unsigned bytes	MIPS DSP
PRECEQ.W.PHL	Vector decimal precision expansion, from (second right) halfword to word, resulting sign expansion	MIPS DSP
PRECEQ.W.PHR	Vector decimal precision expansion, from (rightmost) halfword to word, resulting sign expansion	MIPS DSP
PRECEQU.PH.QBL	Vector decimal precision extension, from (second right) unsigned byte to two halfword	MIPS DSP
PRECEQU.PH.QBR	Vector decimal precision expansion, from (the rightmost two) unsigned bytes to two halfwords	MIPS DSP
PRECEQU.PH.QBLA	Vector decimal precision expansion, from (the left of the rightmost word crosses two) unsigned bytes to two Half word	MIPS DSP
PRECEQU.PH.QBRA	Vector decimal precision expansion, from (the rightmost word crosses two) unsigned bytes to two Half word	MIPS DSP
PRECEU.PH.QBL	Vector integer precision expansion, from (second right) unsigned byte to unsigned halfword	MIPS DSP

PRECEU.PH.QBR	Vector integer precision expansion, from (the rightmost two) unsigned bytes to unsigned half words	MIPS DSP
PRECEU.PH.QBLA	Vector integer precision expansion, from (crossing the left of the rightmost word) two unsigned bytes to two Half word	MIPS DSP
PRECEU.PH.QBRA	Vector integer precision expansion, from (the rightmost word crosses two) unsigned bytes to two Half word	MIPS DSP

Shift instruction based on general register

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SHLL.QB	Vector logical shift left (4 rightmost) bytes, the shift value is specified by the immediate value, the result symbol Expand	MIPS DSP
SHLL.PH	Vector logical shift left (two rightmost) halfwords, the shift value is specified by the immediate value, the result symbol Expand	MIPS DSP

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SHLLV.QB	Vector logical shift left (4 rightmost) bytes, the shift value is specified by the register, the result symbol Expand	MIPS DSP
SHLLV.PH	Vector logical shift left (2 rightmost) halfwords, shift value is specified by register, result sign Expand	MIPS DSP
SHLL_S.PH	Vector logic saturation left shift (the rightmost 2) halfword, the shift value is specified by the immediate value, the result Sign extension	MIPS DSP
SHLL_S.W	Vector logic saturation shift left (rightmost) word, the shift value is specified by the immediate value, the result sign exhibition	MIPS DSP
SHLLV_S.PH	Vector logic saturation shift left (two rightmost) halfwords, the shift value is specified by the register, the result Sign extension	MIPS DSP
SHLLV_S.W	Vector logic saturation left shift (rightmost) word, the shift value is specified by the register, the result symbol is expanded exhibition	MIPS DSP
SHRL.QB	Vector logical right shift (4 rightmost) bytes, the shift value is specified by the immediate value, the result symbol Expand	MIPS DSP
SHRL.PH	Vector logical shift right (2 rightmost) halfwords, shift value is specified by immediate value, result sign Expand	MIPS DSP R2
SHRLV.QB	Vector logical shift right (4 rightmost) bytes, the shift value is specified by the register, the result symbol Expand	MIPS DSP
SHRLV.PH	Vector logical shift right (2 rightmost) halfwords, shift value is specified by register, result sign Expand	MIPS DSP R2
SHRA.QB	Vector arithmetic right shift (4 rightmost) bytes, shift value is specified by immediate value, result sign Expand	MIPS DSP R2
SHRA_R.QB	Vector arithmetic right shift (4 rightmost) bytes, the shift value is specified by the immediate and rounded, Result sign extension	MIPS DSP R2
SHRAV.QB	Vector arithmetic right shift (4 rightmost) bytes, the shift value is specified by the register, the result symbol Expand	MIPS DSP R2
SHRAV_R.QB	Vector arithmetic rounds to the right (4 rightmost) bytes, the shift value is specified by the register, the result Sign extension	MIPS DSP R2
SHRA.PH	Vector arithmetic shift right (2 rightmost) halfwords, the shift value is specified by the immediate value, the result sign Expand	MIPS DSP
SHRAV.PH	Vector arithmetic right shift (2 rightmost) halfwords, shift value is specified by register, result sign Expand	MIPS DSP
SHRA_R.PH	Vector arithmetic rounds to the right (two rightmost) halfwords, the shift value is specified by the immediate value, the result Sign extension	MIPS DSP
SHRAV_R.PH	Vector arithmetic rounds to the right (two rightmost) halfwords, the shift value is specified by the register, the result Sign extension	MIPS DSP
SHRA_R.W	Vector arithmetic rounds to the right (the rightmost) word, the shift value is specified by the immediate exhibition	MIPS DSP
SHRAV_R.W	Vector arithmetic rounds to the right (the rightmost) word, the shift value is specified by the register, and the result sign is expanded exhibition	MIPS DSP

twenty four

Page 38

Multiplication instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
MULEU_S.PH.QBL	Vector (two next to the right) byte unsigned multiplication (two rightmost) halfword, the result is two Half word	MIPS DSP
MULEU_S.PH.QBR	Vector (two rightmost) byte unsigned multiplication (two rightmost) halfwords, the result is two Half word	MIPS DSP
MULQ_RS.PH	Vector (two rightmost) halfwords are saturated and rounded, the result is halfwords	MIPS DSP
MULEQ_S.W.PHL	Signed (second right) halfword saturation multiplication, the result is word	MIPS DSP
MULEQ_S.W.PHR	Signed (rightmost) halfword saturation multiplication, the result is a word	MIPS DSP
DPAU.H.QBL	Vector integer (two times right) bytes unsigned multiply and accumulate, and the result is accumulated with acc	MIPS DSP
DPAU.H.QBR	Vector integer (two rightmost) bytes are accumulated by unsigned multiplication, and the result is accumulated with acc	MIPS DSP
DPSU.H.QBL	Vector unsigned multiply and accumulate (two leftmost) bytes, accumulate with acc	MIPS DSP
DPSU.H.QBR	Vector integer (two rightmost) bytes are accumulated by unsigned multiplication, and the result is accumulated with acc	MIPS DSP
DPA.W.PH	Vector integer (two rightmost) halfword multiply accumulate, and then accumulate with acc	MIPS DSP R2
DPAX.W.PH	Vector integer (two times to the right) halfword multiply accumulate, and then accumulate with acc	MIPS DSP R2
DPAQ_S.W.PH	Vector decimal (two rightmost) halfwords are accumulated and the result is accumulated with acc	MIPS DSP R2
DPAQX_S.W.PH	Vector decimal (two times to the right) halfword multiply, accumulate after saturation, then accumulate with acc	MIPS DSP R2
DPAQX_SA.W.PH	Vector decimals (two rightmost) are multiplied by halfwords, the result is accumulated after rounding, and then accumulates with acc plus	MIPS DSP R2
DPS.W.PH	Vector integer (two rightmost) halfword multiply-accumulate, the result is then subtracted from acc	MIPS DSP R2
DPSX.W.PH	Vector integer (two times right) halfword multiply-accumulate, and the result is then subtracted from acc	MIPS DSP R2
DPSQ_S.W.PH	Vector decimal (two rightmost) halfword multiply and accumulate, the result is then subtracted from acc	MIPS DSP
DPSQX_S.W.PH	Vector decimal (two times right) halfword multiplied, the result is accumulated after saturation, and then subtracted from acc	MIPS DSP R2
DPSQX_SA.W.PH	Vector decimals (two rightmost) are multiplied by halfwords, the results are saturated and rounded up, and then accumulate Less	MIPS DSP R2
MULSAQ_S.W.PH	Vector decimal (two rightmost) half-word multiplication and subtraction, the result is then accumulated with acc	MIPS DSP
DPAQ_SA.LW	Vector multiplying small numbers, accumulating after saturation and rounding, then accumulating with acc	MIPS DSP
DPSQ_SA.LW	Vector multiplying small numbers, accumulating after saturation and rounding, and accumulating with acc	MIPS DSP
MAQ_S.W.PHL	Vector decimal (second right) halfword multiplication, and the result is accumulated with acc	MIPS DSP
MAQ_S.W.PHR	Vector decimal (rightmost) halfword multiplication, and the result is accumulated with acc	MIPS DSP
MAQ_SA.W.PHL	Vector decimal (second right) halfword multiplication, and the result is saturated and rounded up, then accumulated with acc	MIPS DSP
MAQ_SA.W.PHR	Vector decimal (rightmost) halfword multiplication, and the result is rounded up and then accumulated with acc	MIPS DSP
MUL.PH	Vector (the rightmost 2) halfword multiplication, and the lower 16 bits are written to the register	MIPS DSP R2
MUL_S.PH	Vector (the rightmost 2) halfword signed saturation multiplication, the result is lower 16 bits written to the register	MIPS DSP R2
MULQ_S.PH	Vector (two rightmost) halfword saturation multiplication, the result is halfword	MIPS DSP R2
MULQ_S.W	The rightmost word of the vector is saturated multiplied, and the result is the word	MIPS DSP R2
MULQ_RS.W	The rightmost word of the vector is saturated and rounded and multiplied, the result is the word	MIPS DSP R2
MULSA.W.PH	Vector (two rightmost) halfwords are multiplied and subtracted, and the result is accumulated with acc	MIPS DSP R2
MADD	32-bit signed fixed-point multiply and accumulate with acc	MIPS32
MADDU	32-bit unsigned fixed-point number multiply and accumulate with acc	MIPS32
MSUB	32-bit signed fixed-point number multiplied and then subtracted from acc	MIPS32
MSUBU	32-bit unsigned fixed-point number multiplied and then subtracted from acc	MIPS32

25

Page 39

Instruction mnemonic	Instruction function brief	**ISA** compatible category
MULT	Word multiplication, the result is stored in the acc register	MIPS32

MULTU	Unsigned word multiplication, the result is stored in the acc register	MIPS32

Bit manipulation instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
BITREV	Halfword bit flip, result 0 expansion	MIPS DSP
INSV	Variable bit field insertion	MIPS DSP
REPL.QB	Vector copy the immediate value (integer) to the rightmost four bytes, the sign extension of the result	MIPS DSP
REPLV.QB	Vector copies bytes to the rightmost four bytes, resulting in sign expansion	MIPS DSP
REPL.PH	The vector copies the immediate value (integer) to the rightmost two and a half, and the result is sign extended	MIPS DSP
REPLV.PH	Vector copies halfwords to the rightmost halfwords, resulting in sign expansion	MIPS DSP

Compare - extract instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
CMPU.EQ.QB	Vector (4 rightmost) unsigned byte equality comparison, result set condition	MIPS DSP
CMPU.LT.QB	Vector (4 rightmost) unsigned bytes are less than the comparison, and the result is set to conditional	MIPS DSP
CMPU.LE.QB	Vector (4 rightmost) unsigned bytes are less than or equal to the comparison, the result is set to condition	MIPS DSP
CMPGDU.EQ.QB	Vector (the rightmost 4) unsigned byte equality comparison, the result is set conditional and general register	MIPS DSP R2
CMPGDU.LT.QB	Vector (the rightmost 4) unsigned bytes are less than the comparison, and the result is set to condition bit and general register	MIPS DSP R2
CMPGDU.LE.QB	Vector (the rightmost 4) unsigned bytes are less than or equal to the comparison, the result is set to the condition bit at the same time General register	MIPS DSP R2
CMPGU.EQ.QB	Vector (the rightmost 4) bytes are compared for equality, and the result is set in the general register	MIPS DSP
CMPGU.LT.QB	The vector (the rightmost 4 bytes) is less than the comparison, and the result is set in the general register	MIPS DSP
CMPGU.LE.QB	Vector (the rightmost 4 bytes) is less than or equal to the comparison, the result is set to the general register	MIPS DSP
CMP.EQ.PH	Vector (the rightmost 2) halfwords are compared for equality, and the result is set to conditional	MIPS DSP
CMP.LT.PH	Vector (the rightmost 2) halfword is less than the comparison, the result is set to conditional	MIPS DSP
CMP.LE.PH	Vector (the rightmost 2) halfword is less than or equal to the comparison, the result is set to condition	MIPS DSP
PICK.QB	Conditional bit-based (fourth rightmost) byte selection	MIPS DSP
PICK.PH	Conditional bit-based (2 rightmost) halfword selection	MIPS DSP
APPEND	Word shift left and stitch low	MIPS DSP R2
PREPEND	Move right and stitch high	MIPS DSP R2
BALIGN	Two registers high and low byte stitching	MIPS DSP R2
PACKRL.PH	Pack the rightmost half of source 1 and the second right halfword of source 2	MIPS DSP R2

Accumulator operation instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
EXTR.W	After the accumulator shifts to the right, the intercepted word is assigned to the general purpose register, and the shift value is specified by the immediate	MIPS DSP

Page 40

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Instruction mnemonic	Instruction function brief	**ISA** compatible category
EXTR_R.W	The accumulator is shifted to the right and rounded, and the intercepted word is assigned to the general-purpose register. The shift value is specified by the immedi	MIPS DSP
EXTR_RS.W	The accumulator is shifted to the right after saturation and rounded, and the truncated word is assigned to the general register. Designated	MIPS DSP
EXTR_S.H	Saturate right shift from accumulator, extract halfword to general register, shift value is specified by immediate	MIPS DSP
EXTRV_S.H	Right shift from accumulator saturation, extract halfword to general register, shift value is specified by register	MIPS DSP
EXTRV.W	After the accumulator is shifted to the right, the intercepted word is assigned to the general register, and the shift value is specified by the register	MIPS DSP
EXTRV_R.W	The accumulator is shifted to the right and rounded, and the intercepted word is assigned to the general-purpose register. The shift value is specified by the register	MIPS DSP
EXTRV_RS.W	The accumulator is shifted to the right after saturation and rounded, and the intercepted word is assigned to the general register. The shift value is determined by th Designated	MIPS DSP
EXTP	Extract a fixed-length number from any position of the accumulator to a general-purpose register Number specification	MIPS DSP
EXTPV	Extract a fixed-length number from any position of the accumulator to a general-purpose register Instructions	MIPS DSP
	Extract a fixed-length number from any position of the accumulator to a general-purpose register, and subtract the pos value,	

EXTPDP	The length is specified by the immediate	MIPS DSP
EXTPDPV	Extract a fixed-length number from any position of the accumulator to a general-purpose register and subtract the pos value, The length is specified by the register	MIPS DSP
SHILO	The accumulator value is shifted and then written back to the same accumulator, the shift value is determined by the immediate value	MIPS DSP
SHILOV	The accumulator value is shifted and written back to the same accumulator, the shift value is determined by the register	MIPS DSP
MTHLIP	The LO value is copied to HI, the general register value is copied to LO, and the pos value is increased by 32	MIPS DSP
MFHI	HI register value moved to general register	MIPS32
MFLO	LO register value moved to general register	MIPS32
MTHI	General register value moved to HI register	MIPS32
MTLO	The general register value is moved to the LO register	MIPS32

Instruction to access **DSP** control register

Instruction mnemonic	Instruction function brief	**ISA** compatible category
WRDSP	Read general register value and write to DSP control register	MIPS DSP
RDDSP	Read DSP control register value to general register	MIPS DSP

Memory access instruction with index register

Instruction mnemonic	Instruction function brief	**ISA** compatible category
LBUX	Index address takes unsigned byte	MIPS DSP
LHX	Index address takes halfword	MIPS DSP
LWX	Index address fetch	MIPS DSP

Branch instruction

Instruction mnemonic	Instruction function brief	**ISA** compatible category
BPOSGE32	Jump if DSP control register pos value is greater than or equal to 32	MIPS DSP

DSP instructions no longer supported by **MIPS**

The instructions listed in this section are in the "MIPS® Architecture for Programmers Volume IV-e: The MIPS® DSP Application-Specific Extension to the MIPS64® Architecture "is defined in r2.34, but from r2.40,
These instructions related to 64-bit data operations have been deleted. Although Loongson 3A3000 / 3B3000 processor implements these instructions, it is not recommended User use.

Instruction mnemonic	Instruction function brief
ABSQ_S.OB	The vector finds the (rightmost 8) byte absolute value, and performs a saturation operation, resulting in sign expansion
ABSQ_S.PW	The vector finds (the rightmost 4) half-word absolute values, and performs a saturation operation, resulting in sign expansion
ABSQ_S.QH	Vector finds (the rightmost 2) absolute value of the word, and performs saturation operation, the sign expansion of the result
ADDQ.PW	Vector (2 rightmost) small numbers plus
ADDQ.S.PW	Vector (2 rightmost) small numbers saturated plus
ADDQ.QH	Vector (4 rightmost) decimal halfword plus
ADDQ.S.QH	Vector (4 rightmost) decimal halfword saturation plus
ADDU.OB	Vector (8 rightmost) small number section unsigned plus
ADDU.S.OB	Vector (8 rightmost) small unsigned saturation plus
ADDU.QH	Vector (4 rightmost) decimal halfword unsigned plus
ADDU.S.QH	Vector (4 rightmost) decimal halfword unsigned saturation plus
ADDUH.OB	Vector unsigned (8 rightmost) bytes are added, the result is divided by 2, and the result is sign extended
ADDUH_R.OB	Vector unsigned (8 rightmost) bytes are rounded and added, the result is divided by 2, and the result is sign extended
BPOSGE64	Jump if DSP control register pos value is greater than or equal to 64
CMP.EQ.PW	Vector (the rightmost 2) words are compared for equality, and the result is set to conditional
CMP.LT.PW	Vector (the rightmost 2) words are less than the comparison, the result is set to conditional
CMP.LE.PW	Vector (the rightmost 2) words are less than or equal to the comparison, the result is set to conditional
CMP.EQ.QH	Vector (4 rightmost) half-word equal comparisons, result set condition
CMP.LT.QH	Vector (the rightmost 4) halfword is less than the comparison, the result is set to conditional
	Vector (4 rightmost) halfwords are less than or equal to the comparison, the result is set to conditional

CMP.LE.QH CMPGDU.EQ.OB	Vector (8 rightmost) unsigned byte equality comparison, the result is set to condition bit and general register
CMPGDU.LT.OB	Vector (8 rightmost) unsigned bytes are less than the comparison, the result is set to condition bit and general register at the same time
CMPGDU.LE.OB	Vector (8 rightmost) unsigned bytes are less than or equal to the comparison, the result is set to the condition bit and the general register
CMPGU.EQ.OB	Vector (8 rightmost) bytes are compared for equality, and the result is placed in the general register
CMPGU.LT.OB	The vector (rightmost 8) bytes are less than the comparison, and the result is set in the general register
CMPGU.LE.OB	Vector (the rightmost 8) bytes are less than or equal to the comparison, the result is set to the general register
CMPU.EQ.OB	Vector (8 rightmost) unsigned byte equality comparison, result set condition
CMPU.LT.OB	Vector (8 rightmost) unsigned bytes are less than the comparison, the result is set to conditional
CMPU.LE.OB	Vector (8 rightmost) unsigned bytes are less than or equal to the comparison, the result is set to conditional
DAPPEND	Move right and stitch high
DBALIGN	Two registers high and low byte stitching
DEXTP	Extract a fixed-length number from any position in the accumulator to a general-purpose register, the length is specified by the immediate
DEXTPDP	Extract a fixed-length number from any position of the accumulator to a general-purpose register, and subtract the pos value, the length is indicated by the immedi set

Instruction mnemonic	Instruction function brief
DEXTPDPV	Extract a fixed-length number from any position of the accumulator to a general-purpose register, and subtract the pos value. The length is determined by the regis set
DEXTPV	Extract a fixed-length number from any position of the accumulator to a general-purpose register, the length is controlled by the register instruction
DEXTR.L	After the accumulator shifts to the right, the double word is intercepted and assigned to the general-purpose register, the shift value is specified by the immediate v
DEXTR_R.L	The accumulator is shifted to the right and rounded, and the double word is intercepted and assigned to the general register. The shift value is specified by the imm
DEXTR_RS.L	The accumulator is shifted to the right after saturation and rounded, and the double word is intercepted and assigned to the general register.
DEXTR.W	After the accumulator shifts to the right, the intercepted word is assigned to the general purpose register, and the shift value is specified by the immediate
DEXTR_R.W	The accumulator is shifted to the right and rounded, and the intercepted word is assigned to the general-purpose register. The shift value is specified by the immed
DEXTR_RS.W	The accumulator is shifted to the right after saturation and rounded, and the intercepted word is assigned to the general-purpose register.
DEXTR_S.H	Saturate right shift from accumulator, extract halfword to general register, shift value is specified by immediate
DEXTRV.L	After the accumulator shifts to the right, the double word is intercepted and assigned to the general register, and the shift value is specified by the register
DEXTRV_R.L	The accumulator is shifted to the right and rounded, and the double word is intercepted and assigned to the general-purpose register. The shift value is specified by
DEXTRV_RS.L	The accumulator is shifted to the right after saturation and rounded, and the double word is intercepted and assigned to the general-purpose register.
DEXTRV_S.H	After the accumulator is shifted to the right, the intercepted word is assigned to the general register, and the shift value is specified by the register
DEXTRV.W	The accumulator is shifted to the right and rounded, and the intercepted word is assigned to the general-purpose register. The shift value is specified by the register
DEXTRV_R.W	The accumulator is shifted to the right after saturation and rounded, and the intercepted word is assigned to the general-purpose register. The shift value is specifie
DEXTRV_RS.W	Right shift from accumulator saturation, extract halfword to general register, shift value is specified by register
DINSV	Variable bit field insertion
DMADD	Double sign signed multiply-add
DMADDU	Double-word unsigned multiply-add
DMSUB	Double sign signed multiplication
DMSUBU	Double word unsigned multiplication and subtraction
DMTHLIP	The LO value is copied to HI, the general register value is copied to LO, and the pos value is increased by 64
DPA.W.QH	Vector integer (4 rightmost) halfwords accumulate and accumulate with acc
DPAQ_S.W.QH	Vector decimal (the rightmost 4) halfword multiply and accumulate, and the result accumulates with acc
DPAQ_SA.L.PW	Vector multiplying small numbers, accumulating after saturation and rounding, then accumulating with acc
DPAU.H.OBL	Vector integer (leftmost 4) bytes unsigned multiply and accumulate, and the result accumulates with acc
DPAU.H.OBR	Vector integer (4 rightmost) bytes unsigned multiply and accumulate, and the result accumulates with acc
DPS.W.QH	Vector integer (4 rightmost) halfword multiply-accumulate, the result is then subtracted from acc
DPSQ_S.W.QH	Vector decimal (the rightmost 4) halfword multiply and accumulate, and the result is subtracted from acc
DPSQ_SA.L.PW	Vector multiplying small numbers, accumulating after saturation and rounding, and accumulating with acc
DPSU.H.OBL	Vector integer (leftmost 4) bytes unsigned multiply and accumulate, and the result accumulates with acc
DPSU.H.OBR	Vector integer (4 rightmost) bytes unsigned multiply and accumulate, and the result accumulates with acc
DSHILO	The accumulator value is shifted and then written back to the same accumulator, the shift value is determined by the immediate value
DSHILOV	The accumulator value is shifted and written back to the same accumulator, the shift value is determined by the register
LDX	Base address plus index access
MAQ_S.L.PWL	Vector decimal (second right) word multiplication, and the result is accumulated with acc
MAQ_S.L.PWR	Vector decimal (rightmost) word multiply, accumulate result with acc
MAQ_S.W.QHLL	Vector decimal (leftmost) halfword multiplication, and the result is accumulated with acc

MAQ_SA.W.QHLL	Vector decimal (leftmost) halfword multiplication, and the result is rounded to saturation and then accumulated with acc
MAQ_S.W.QHLR	Vector decimal (second left) halfword multiplication, and the result is accumulated with acc

Instruction mnemonic	Instruction function brief
MAQ_SA.W.QHLR	Vector fractional (second left) halfword multiplication, the result is rounded to saturation and then accumulated with acc
MAQ_S.W.QHRL	Vector decimal (second right) halfword multiplication, and the result is accumulated with acc
MAQ_SA.W.QHRL	Vector decimal (second right) halfword multiplication, and the result is saturated and rounded and then accumulated with acc
MAQ_S.W.QHRR	Vector decimal (rightmost) halfword multiplication, and the result is accumulated with acc
MAQ_SA.W.QHRR	Vector decimal (rightmost) halfword multiplication, and the result is rounded up and then accumulated with acc
MULEQ_S.PW.QHL	Saturated multiplication of the half word of the left half of the sign, the result is the word
MULEQ_S.PW.QHR	Saturated multiplication of the signed half-word in the right half, the result is the word
MULEU_S.QH.OBL	Vector (4 times to the right) byte unsigned multiplication (4 times to the right) halfword, the result is two halfwords
MULEU_S.QH.OBR	Vector (4 rightmost) byte unsigned multiplication (4 rightmost) halfwords, the result is two halfwords
MULQ_RS.QH	Vector (4 rightmost) halfwords are saturated and rounded and multiplied, resulting in halfwords
MULSAQ_S.L.PW	Vector small numbers are multiplied and subtracted after saturation, and the result is accumulated with acc
MULSAQ_S.W.QH	Vector decimals (4 rightmost) are multiplied and subtracted by halfwords, and the result is accumulated with acc
PACKRL.PW	Pack the rightmost word of source 1 and the second right word of source 2
PICK.OB	(8 rightmost) byte selection based on condition bits
PICK.PW	Conditional bit-based (4 rightmost) halfword selection
PICK.QH	Conditional bit-based (2 rightmost) word selection
PRECEQ.L.PWL	Vector decimal precision has sign extension, from (second right) word to double word
PRECEQ.L.PWR	Vector decimal precision has sign extension, from (rightmost) word to double word
PRECEQ.PW.QHL	Vector decimal precision has sign extension, from (second right) halfword to two words
PRECEQ.PW.QHR	Vector decimal precision has sign extension, from (rightmost two) halfword to two words
PRECEQ.PW.QHLA	Vector decimal precision has sign extension, from (crossing two on the left of the rightmost word) halfword to two words
PRECEQ.PW.QHRA	The vector decimal precision has sign extension, from (the rightmost word crosses two) halfword to two words
PRECEQU.QH.OBL	Vector decimal precision extension, from (4 left) unsigned bytes to 4 halfwords
PRECEQU.QH.OBR	Vector decimal precision extension, from (4 right) unsigned bytes to 4 halfwords
PRECEQU.QH.OBLA	Vector decimal precision extension, from (crossing 4 left of the rightmost word) unsigned byte to 4 halfwords
PRECEQU.QH.OBRA	Vector decimal precision expansion, from (crossing 4 to the right of the rightmost word) unsigned byte to 4 halfwords
PRECEU.QH.OBL	Vector precision expansion, from (4 left) unsigned bytes to 4 halfwords
PRECEU.QH.OBR	Vector precision expansion, from (4 right) unsigned bytes to 4 halfwords
PRECEU.QH.OBLA	Vector precision expansion, from (crossing 4 left of the rightmost word) unsigned byte to 4 halfwords
PRECEU.QH.OBRA	Vector precision expansion, from (crossing 4 to the right of the rightmost word) unsigned bytes to 4 halfwords
PRECR.OB.QH	Vector integer precision reduction, from (rightmost 4) halfwords to 8 bytes
PRECR_SRA.QH.PW	Vector right shift integer precision reduction, from (the rightmost 2) words to 4 halfwords, resulting sign expansion
PRECR_SRA_R.QH.PW	The vector is shifted to the right by integer precision reduction, from (the rightmost 2) words to 4 halfwords, and rounded, resulting in sign expansion
PRECRQ.OB.QH	Vector decimal precision reduction, from (rightmost 4) halfword to 8 bytes
PRECRQ.PW.L	Vector decimal precision reduction, from (the rightmost 2) words to 4 halfwords
PRECRQ.QH.PW	Vector decimal precision reduction, from (rightmost 2) words to (4) halfwords
PRECRQ_RS.QH.PW	Decrease precision of vector decimal, from (rightmost 2) words to (4) halfwords, and do saturation and rounding
PRECRQU_S.OB.QH	Vector decimal precision reduction, from (rightmost 4) halfwords to 8 unsigned bytes
PREPENDD	Double-word shift to the right and stitch high
PREPENDW	Word shift right and stitch high
RADDU.L.OB	8-byte unsigned accumulation

Instruction mnemonic	Instruction function brief
REPL.OB	Vector copy immediate value (integer) to 8 bytes, resulting sign expansion
REPL.PW	Vector copy immediate number (integer) to 2 words, sign expansion of result
REPL.QH	Vector copy immediate (integer) to 4 halfwords, sign expansion of result
REPLV.OB	Vector copies bytes to 8 bytes, resulting sign expansion
REPLV.PW	Vector copy word to 2 words, result sign expansion
REPLV.QH	Vector copy halfword to 4 halfwords, resulting sign expansion
SHLL.OB	Vector logical shift left (8 rightmost) bytes, the shift value is specified by the immediate value, and the sign extension of the result
SHLL.QH	Vector logical shift left (4 rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHLL.PW	Vector logical shift left (two rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHLLV.OB	Vector logical shift left (8 rightmost) bytes, the shift value is specified by the register, and the result sign expansion
SHLLV.QH	Vector logical shift left (4 rightmost) halfwords, shift value is specified by register, result sign expansion
SHLLV.PW	Vector logical shift left (two rightmost) halfwords, the shift value is specified by the register, and the sign expansion of the result
SHLL_S.QH	Vector saturation logic shifts to the left (4 rightmost) halfwords, the shift value is specified by the immediate value, and the result is sign extended
SHLL_S.PW	Vector saturation logic shifts to the left (2 rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHLLV_S.QH	Vector saturation logic shifts to the left (4 rightmost) halfwords, the shift value is specified by the register, and the sign expansion of the result
SHLLV_S.PW	Vector saturation logic shifts to the left (2 rightmost) halfwords, the shift value is specified by the register, and the sign expansion of the result
SHRA.OB	Vector arithmetic right shift (8 rightmost) bytes, the shift value is specified by the immediate value, and the sign extension of the result
SHRA.QH	Vector arithmetic right shift (4 rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHRA.PW	Vector arithmetic right shift (2 rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHRAV.OB	Vector arithmetic right shift (8 rightmost) bytes, the shift value is specified by the register, and the sign extension of the result
SHRAV.QH	Vector arithmetic right shift (4 rightmost) halfwords, the shift value is specified by the register, and the sign extension of the result
SHRAV.PW	Vector arithmetic shift right (2 rightmost) halfwords, shift value is specified by register, result sign expansion
SHRA_R.OB	Vector arithmetic right shift (8 rightmost) bytes, the shift value is specified by the immediate value, and rounded, the result is sign extended
SHRA_R.QH	Vector arithmetic right shift (4 rightmost) halfwords, the shift value is specified by the immediate value, and rounded, the result is sign extended
SHRA_R.PW	Vector arithmetic shift right (two rightmost) words, the shift value is specified by the immediate number, and rounded, the result sign expansion
SHRAV_R.OB	Vector arithmetic right shift (8 rightmost) bytes, the shift value is specified by the register, and rounding is performed, and the result is sign extended
SHRAV_R.QH	Vector arithmetic right shift (4 rightmost) halfwords, the shift value is specified by the register, and rounding is performed, and the result is sign extended
SHRAV_R.PW	Vector arithmetic shift right (two rightmost) words, the shift value is specified by the register, and rounded, the sign expansion of the result
SHRL.OB	Vector logical right shift (8 rightmost) bytes, the shift value is specified by the immediate value, and the sign extension of the result
SHRL.QH	Vector logical right shift (4 rightmost) halfwords, the shift value is specified by the immediate value, and the sign extension of the result
SHRLV.OB	Vector logical shift right (8 rightmost) bytes, the shift value is specified by the register, and the sign extension of the result
SHRLV.QH	Vector logical shift right (4 rightmost) halfwords, shift value is specified by register, result sign expansion
SUBQ.PW	Vector (2 rightmost) small numbers minus
SUBQ.QH	Vector (4 rightmost) decimal half-word minus
SUBQ_S.PW	Vector (2 rightmost) small numbers
SUBQ_S.QH	Vector (4 rightmost) decimal halfword saturation minus
SUBU.OB	Vector (8 rightmost) unsigned minus numbers
SUBU.QH	Vector (4 rightmost) decimal halfword unsigned minus
SUBU_S.OB	Vector (8 rightmost) small unsigned saturation minus
SUBU_S.QH	Vector (4 rightmost) decimal halfword unsigned saturation minus
SUBUH.OB	Vector unsigned (8 rightmost) bytes minus, result divided by 2, result sign extended

31

Instruction mnemonic	Instruction function brief
SUBUH_R.OB	Vector unsigned (8 rightmost) bytes are rounded down, the result is divided by 2, and the result is sign extended

2.3.2 Supplement to the **MIPS DSP** instruction manual

The definition of MTHI and MTLO instructions in the MIPS DSP instruction manual is unreasonable. Under 64-bit architecture, execute MTHI and MTLO
The instruction does not need to sign-extend the value of the general-purpose register to the HI / LO register after sign extension. MIPS General Instruction Manual should be used
The definition forms of MTHI and MTLO instructions in HI are HI ← GPR [rs], LO ← GPR [rs].

2.4 **MIPS64** compatible instruction implementation related definitions

This section deals with the implementation-related parts of the MIPS64 compatible instructions implemented by GS464E, or the parts that differ from the MIPS64 specification

Describe it separately.

2.4.1 **Load** instruction with the target of general register **0**

All targets except LDPTE and LWPTE are load instructions of general registers. When the target bit is general register 0, its execution
The effect is equivalent to PREF (hint = 0). However, for Godson's custom extended GSLQ instruction, both of its target registers must be set to 0
No. general register.

2.4.2 **PREF** instruction

The PREF instruction only implements two modes of hint = 0 (prefetch for load) and hint = 1 (prefetch for store) according to the MIPS specification.

2.4.3 **RDHWR** instruction

The RDHWR instruction not only implements rd values of 0, 1, 2, 3, and 29 according to the MIPS specification, but also implements rd values of 30 and 31. specific
The definition is as follows:

- rd = 30: read the external counter value. The counter is 64 bits, incremented by 1 per beat, the increase frequency is consistent with the internal bus clock frequency of the chip.
  The characteristic of the clock frequency is that it does not change with the change of the clock frequency of a certain processor core, and it can be considered as a constant frequer
- rd = 31: Read the proportional relationship (M / N) between the current frequency of the processor core and the internal bus frequency of the chip. Where M is in the return value [15:
  Bit, N is located in [7: 0] of the return value.

2.4.4 **PREFX** instruction

The PREFX instruction implements five hints of 0, 1, 26, 27, and 28. The specific implementation details are as follows:

- hint = 0, 1: At this time, the prefetch function of the PREFX instruction is consistent with the MIPS specification, that is, one cacheline is prefetched at a time. The difference is the dra
  The core GS464E only uses the sign expansion of the lower 16 bits of the index register as an index.
- hint = 26: Continuously prefetch data into the first-level data cache according to the configuration, a prefetch instruction can automatically prefetch from the base address
  block_num blocks with stride length and block_size. At this time, part of the information in the index register is used for configuration
  Set the prefetch address generation mode. Specifically, the [15: 0] bits of GPR [index] are the 16-bit signed addresses added to GPR [base]
  Offset; the [16] bit of GPR [index] is the address pre-fetch mark in ascending-descending order, 0 indicates pre-fetch in ascending address order, 1 indicates descending address
  Prefetching; The [25:20] bits of GPR [index] are the prefetched block_size-1, the basic unit of block_size is 128bit, so the longest
  block can contain 64 × 16 = 1KB of data; the [39:32] bits of GPR [index] is block_num-1, so it supports up to 256 blocks
  Prefetching; index [59:44] is the stride between adjacent blocks, which is a signed number in bytes.

- hint = 27: Continuously prefetch exclusive data according to the configuration and enter the first-level data cache. In this mode, the content of the index register is analyzed and
  hint = 26 is exactly the same, the difference is that this mode requires that the data retrieved must be exclusive.
- hint = 28: Continuously prefetch data into the shared cache according to the configuration. In this mode, the analysis of the contents of the index register is exactly the same as hint = 2
  Consistent, the difference is that the prefetched data is stored in the shared cache in this mode.

If a WAIT_CACHE instruction is executed, configuration prefetching will be stopped. Such instructions include but are not limited to cache instructions, SYNC
And SYNCI instruction, JRHB instruction.

If you execute two PREFX continuously (hint = 25,26), the previous one will be overwritten by the new one. If two PREFX are executed consecutively (hint = 27),
The previous one will be covered by the new one. However, there is no conflict between prefetching of the two modes hint = 25, 26 and hint = 27.

2.4.5 **WAIT** instruction

The WAIT instruction only supports one mode, no matter what value the Software enters into the Implementation dependent code field in the instruction encoding, WAIT
The instruction execution effect is the same.

After the WAIT instruction is submitted in the processor pipeline, the processor core will stop fetching instructions and enter a low-power mode, which will continue until
Interrupted by NMI, internal clock interrupt, internal performance counter interrupt or external hardware interrupt.

It should be noted that after executing the WAIT instruction, the data of the first-level instruction cache, first-level data cache and second-level victim cache of the processor core
The channel can still handle cache coherency requests from outside. If you need to turn off the clock of the processor core completely, the software needs to do more preparations
It has been ensured that the processor core does not cause other processor cores to freeze due to unresponsiveness of consistency requests after the clock is turned off.

2.4.6 **SYNC** instruction

This processor only implements the SYNC instruction with stype = 0, and the reserved instruction exception will be reported when stype is other values. This instruction acts as a storag
(memory barrier) function, used to ensure that the memory access operation before SYNC has been determined to be completed (for example, the data of the store instruction is written to dcac
Uncached read and write is completed, load has retrieved the value to the register), and at the same time ensure that the memory access operation after the SYNC instruction has not yet begun

2.4.7 **SYNCI** instruction

Because the GS464E maintains the data consistency between the first-level instruction cache and the first-level data cache by hardware, the implementation of the SYNCI instruction Can be adjusted. In the existing implementation, on the one hand, the SYNCI instruction will wait for all previous memory access operations to be completed before it can be executed (effect (Same as SYNC instruction); on the other hand, SYNCI will force subsequent instructions to fetch instructions after execution to ensure that the fetching parts are also certain You can see the execution effect of all memory access operations before the SYNCI instruction. At the same time, the address information carried by the SYNCI instruction is ignored, so TIB related exceptions, address error exceptions and cache error exceptions will no longer be triggered.

2.4.8 **TLBINV** and **TLBINVF** instructions

Although the value of the Config4.IE field in this processor is set to 0, the TLBINVF instruction (in accordance with the MIPS specification) is implemented in the processor The defined Config4.IE = 3 is executed), that is, the hardware will invalidate the entire TLB entry when executing a TLBINVF instruction. In addition, The TLBINV instruction is also implemented in GS464E, but its execution effect is different from the definition of the MIPS specification, but is equivalent to the TLBINVF instruction.

2.4.9 **CACHE** instruction

There are three main differences between the CACHE instruction implemented by this processor and the MIPS64 specification:

1. The relationship between instruction **op** and cache hierarchy

The correspondence between the [1: 0] two bits of the op field of the CACHE instruction in the GS464E and the cache hierarchy is different from the definition of the MIPS64 specificat As shown in 2-26 :

Table **2-26** Correspondence between **CACHE** instruction **op [1: 0]** and cache hierarchy

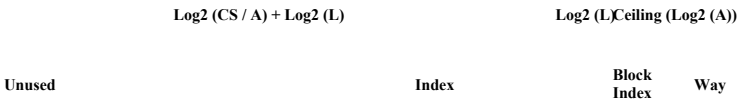
op [1: 0]	**GS464E**	**MIPS64** specification	Similarities and differences
0b00	Level 1 instruction cache	Level 1 instruction cache	Unanimous
0b01	Level 1 data cache	Level 1 data cache	Unanimous
0b10	Second-level sacrifice cache	Level 3 cache	Inconsistent
0b11	Level 3 shared cache	Secondary cache	Inconsistent

2 , **Index** class **CACHE** address instruction analytically

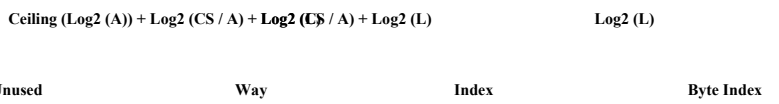
The address resolution method of the Index cache instruction implemented by this processor is different from the definition of the MIPS64 specification. The difference between the two As shown. It should be noted that because the GS464E uses the lowest bits of the address to represent the cache path to be operated, the second level sacrifices the cache and the third The level shared caches are all associated with 16 way groups, and the way selection needs to occupy the lowest 4 bits of the address, so for the level 2 victim cache and level 3 shared cache The Index Data Load and Index Data Store operations of the row can only read the even block data of the Cache row, and only the parity of the Cache row can be adjacent Both blocks are filled with the same data at the same time.

FIG. **2-8 Index** class **CACHE** address resolution instruction format

Loongson **3A1500** chip processor



MIPS64 specification



Description:
Assume that the capacity of the operated Cache is CS, the degree of association is A-way group association, and the cache line size is L bytes. then
The number of address bits used to select the number of channels is: Ceiling (Log2 (A))
The number of address bits used to index the number of Cache lines in each way is: Log2 (CS / A)
The number of address bits used for byte positioning in the cache line is: Log2 (L)

3 , the **meaning of CACHE28** , **CACHE29** , **CACHE30** , **CACHE31** instructions

In GS464E, CACHE28, CACHE29, CACHE30 and CACHE31 instructions (ie op [4: 2] = 0b111 Cache instruction) The meaning is different from the MIPS64 specification. The instructions in the Godson processor are Index Store Data operations, while the MIPS64 specification is Fetch and Lock operation.

. 4 , **CACHE15** meaning instruction

In GS464E, the CACHE15 instruction is an implementation-related CACHE instruction whose function is to Fetch and Lock the Scache

Operation, the specific function definition is similar to the CACHE31 instruction in the MIPS64 specification.

2.4.10 **MADD.fmt** , **MSUB.fmt** , **NMADD.fmt** , **NMSUB.fmt** instructions

In the MIPS64 specification, if FIR.Has2008 = 0 or FCSR.MAC2008 = 0, then MADD.fmt, MSUB.fmt, NMADD.fmt

And the NMSUB.fmt instruction round the intermediate result after the multiplication operation and then perform subsequent addition and subtraction operations, and round the final result aft
Into. GS464E Although it does not support the ANSI / IEEE754-2008 binary floating-point arithmetic standard, when performing floating-point multiply-add operations, only the last
The result is rounded, which is the Fuse-Multiply-Add operation defined by the ANSI / IEEE754-2008 binary floating-point arithmetic standard.

2.4.11 **EHB** and **SSNOP** instructions

All execution correlations between GS464E instructions are handled by hardware, so EHB and SSNOP instructions are regarded as NOP instructions in the processor Management.

2.4.12 **DI** and **EI** instructions

The DI instruction can only be used in Loongson 3A3000C / 3B3000C. The specific definition is compatible with the MIPS64 definition.

The EI instruction can only be used in the Godson 3A3000C / 3B3000C, but the specific definition is slightly different from the MIPS64 definition: after the EI implementation To set the IE position of the Status register to 0, turn off the global terminal enable, but only the lowest 3 bits of the return value are meaningful, corresponding to the EI execution The ERL, EXL, and IE bits of the Status register before the line.

2.5 Godson extended instruction set

The Godson extended instructions implemented by GS464E are divided into the following categories according to functions:

- Fetch instruction
- Arithmetic and logical operation instructions
- X86 binary acceleration instructions
- ARM binary accelerated instructions
- 64-bit multimedia commands
- Miscellaneous

Fetch instruction

Table **2-27** Loongson extended memory access instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SETMEM	Fetch instruction extension prefix	LoongEXT32
LWDIR	32-bit page table directory entry access instruction	LoongEXT32
LWPTE	32-bit page table entry access instruction	LoongEXT32
LDDIR	64-bit page table directory entry access instruction	LoongEXT64
LDPTE	64-bit page table entry access instruction	LoongEXT64
GSLE	Exception if less than or equal to wrong address	LoongEXT32
GSGT	Exception if greater than wrong address	LoongEXT32
GSLWLC1 1	Take the left part of the word to the floating point register	LoongEXT32
GSLWRC1 2	Take the right part of the word to the floating point register	LoongEXT32
GSLDLC1	Take the left part of the double word to the floating-point register	LoongEXT32

1 In LS3A2000A and LS3A2000B, this instruction can only be used when Status.FR = 1.
2 In LS3A2000A and LS3A2000B, this instruction can only be used when Status.FR = 1.

Instruction mnemonic	Instruction function brief	**ISA** compatible category
GSLDRC1	Take the right part of the double word to the floating point register	LoongEXT32
GSLBLE	Fetch bytes with out-of-bounds check	LoongEXT32
GSLBGT	Fetch byte with lower bound check	LoongEXT32
GSLHLE	Take half-word with cross-border inspection	LoongEXT32
GSLHGT	Take half word with cross-border check	LoongEXT32
GSLWLE	Take the word out of check	LoongEXT32
GSLWGT	Take the word with cross-border inspection	LoongEXT32
GSLDLE	Take double words with cross-border inspection	LoongEXT64
GSLDGT	Take double word with cross-border check	LoongEXT64
GSLWLEC1	Fetch word with floating-point check to floating-point register	LoongEXT32
GSLWGTC1	Fetch word with floating point check to floating point register	LoongEXT32
GSLDLEC1	Take double word with floating-point check to floating point register	LoongEXT64
GSLDGTC1	Take double word to floating-point register with lower out-of-bounds check	LoongEXT64
GSLQ	Double target register takes fixed-point four words	LoongEXT64
GSLQC1	Double target register fetches floating point four words	LoongEXT64
GSLBX	Fetch byte with offset	LoongEXT32
GSLHX	Halfword with offset	LoongEXT32
GSLWX	Fetch word with offset	LoongEXT32
GSLDX	Double word with offset	LoongEXT64
GSLWXC1	Floating-point word with offset	LoongEXT32
GSLDXC1	Floating double word with offset	LoongEXT32
GSSWLC1	Save the left part of the word from the floating-point register	LoongEXT32
GSSWRC1	Save the right part of the word from the floating-point register	LoongEXT32
GSSDLC1	Save double word left from floating point register	LoongEXT32
GSSDRC1	Save double word right from floating point register	LoongEXT32
GSSBLE	Stored bytes with out-of-bounds check	LoongEXT32
GSSBGT	Stored bytes with lower bound check	LoongEXT32
GSSHLE	Save half-word with cross-border inspection	LoongEXT32
GSSHGT	Save half word with cross-border check	LoongEXT32
GSSWLE	Bring the deposit with cross-border inspection	LoongEXT32
GSSWGT	Save the word with cross-border inspection	LoongEXT32
GSSDLE	Save double word with cross-border inspection	LoongEXT64
GSSDGT	Save double word with cross-border check	LoongEXT64
GSSWLEC1	Save word from floating-point register with out-of-bounds check	LoongEXT32
GSSWGTC1	Save word from floating-point register with lower out-of-bounds check	LoongEXT32
GSSDLEC1	Save double word from floating-point register with cross-border check	LoongEXT32
GSSDGTC1	Save double word from floating-point register with lower out-of-bounds check	LoongEXT32
GSSQ	Dual source registers store fixed-point four words	LoongEXT64
GSSQC1	Dual source registers store fixed-point four words	LoongEXT64
GSSBX	Byte with offset	LoongEXT32
GSSHX	Halfword with offset	LoongEXT32

Instruction mnemonic	Instruction function brief	**ISA** compatible category
GSSWX	Store word with offset	LoongEXT32
GSSDX	Double word with offset	LoongEXT64
GSSWXC1	Floating-point word with offset	LoongEXT32
GSSDXC1	Floating double word with offset	LoongEXT32

Arithmetic and logical operation instructions

Table **2-28** Loongson extended arithmetic and logic operation instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
GSANDN	NOR of general register logic bits	LoongEXT32
GSORN	NOR of general logic bits	LoongEXT32

GSADC.D	Carry double word plus	LoongEXT64
GSADC.W	Carry word plus	LoongEXT32
GSADC.H	Carry halfword plus	LoongEXT32
GSADC.B	Carry carry plus	LoongEXT32
GSADCU.D	Unsigned double word plus with carry	LoongEXT64
GSADCU.W	Unsigned word with carry plus	LoongEXT32
GSADCU.H	Unsigned halfword plus with carry	LoongEXT32
GSADCU.B	Unsigned byte plus carry	LoongEXT32
GSSBC.D	Double word subtraction with borrow	LoongEXT64
GSSBC.W	With Borrow Word Subtraction	LoongEXT32
GSSBC.H	Half-word subtraction with borrow	LoongEXT32
GSSBC.B	With borrowed bytes minus	LoongEXT32
GSSBCU.D	Unsigned double word subtraction with borrow	LoongEXT64
GSSBCU.W	Unsigned word subtraction with borrow	LoongEXT32
GSSBCU.H	Unsigned halfword subtraction with borrow	LoongEXT32
GSSBCU.B	Unsigned byte subtraction with borrow	LoongEXT32
GSMULT	Signed word multiplication, result is written to general register	LoongEXT32
GSDMULT	Signed double word multiplication, write result to general register	LoongEXT64
GSMULTU	Unsigned word multiplication, the result is written to the general register	LoongEXT32
GSDMULTU	Unsigned double word multiplication, the result is written to the general register	LoongEXT64
GSDIV	Signed word division, quotient write general register	LoongEXT32
GSDDIV	Signed double word division, quotient write general register	LoongEXT64
GSDIVU	Unsigned word division, quotient to write general register	LoongEXT32
GSDDIVU	Unsigned double word division, quotient write general register	LoongEXT64
GSMOD	Divide signed words, write remainder to general register	LoongEXT32
GSDMOD	Signed double word division, write remainder to general register	LoongEXT64
GSMODU	Unsigned word division, remainder write to general register	LoongEXT32
GSDMODU	Unsigned double word division, write remainder to general register	LoongEXT64
GSROTR.H	Half word cycle right	LoongEXT32
GSROTR.B	Byte rotation right	LoongEXT32
GSROTRV.H	Variable shift amount half-word right shift	LoongEXT32

37

Page 51

Instruction mnemonic	Instruction function brief	**ISA** compatible category
GSROTRV.B	Variable shift amount byte rotate right	LoongEXT32
GSRCR.D	Double word with CF bit shift right	LoongEXT64
GSRCR.W	Word with CF bit rotate right	LoongEXT32
GSRCR.H	Half word with CF bit rotate right	LoongEXT32
GSRCR.B	Byte shift with CF bit rotate right	LoongEXT32
GSDRCR32	The double word with CF bit plus 32 is shifted right by the shift amount	LoongEXT64
GSRCRV.D	Variable shift amount Double CF with CF bit shift right	LoongEXT64
GSRCRV.W	Variable shift with CF bit word shift right	LoongEXT32
GSRCRV.H	Variable shift amount Half-word with CF bit shift right	LoongEXT32
GSRCRV.B	Variable shift with CF bit shift right	LoongEXT32

Binary translation acceleration instruction **(X86)**

Table **2-29** Godson extended **X86** binary translation acceleration instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
SETX86FLAG.D	Set EFLAG double word mode prefix instruction in x86	LoongEXT64
SETX86FLAG.W	Set the EFLAG word mode prefix instruction in x86	LoongEXT32
SETX86FLAG.H	Set EFLAG halfword mode prefix instruction in x86	LoongEXT32
SETX86FLAG.B	Set EFLAG byte mode prefix instruction in x86	LoongEXT32
X86AND.D	Set only double-word logical bits of EFLAG in x86 mode	LoongEXT64
X86AND.W	In x86 mode, only set the EFLAG word logic bit and	LoongEXT32
X86AND.H	Set only half-word logical bits of EFLAG with x86	LoongEXT32
X86AND.B	Only set the EFLAG byte logical bit and	LoongEXT32

X86OR.D	Set only double-word logical bits of EFLAG in x86 mode	LoongEXT64
X86OR.W	Only set the word logic bit of EFLAG in x86 mode or	LoongEXT32
X86OR.H	Set only half-word logical bits of EFLAG in x86 mode	LoongEXT32
X86OR.B	Only set the EFLAG byte logical bit or in x86 mode	LoongEXT32
X86XOR.D	XOR only sets the double-word logical bit XOR of EFLAG	LoongEXT64
X86XOR.W	Only set XOR of word logical bit of EFLAG in x86 mode	LoongEXT32
X86XOR.H	Only set the XOR of half-word logical bit of EFLAG in x86 mode	LoongEXT32
X86XOR.B	XOR only sets the logical logical bit XOR of EFLAG	LoongEXT32
X86ADD.D	Set only double word plus of EFLAG in x86 mode	LoongEXT64
X86ADD.W	Only set the word addition of EFLAG in x86 mode	LoongEXT32
X86ADD.H	Set only half-word plus of EFLAG in x86 mode	LoongEXT32
X86ADD.B	Set only the bytes of EFLAG plus in x86 mode	LoongEXT32
X86ADC.D	In x86 mode only set EFLAG's carry double word plus	LoongEXT64
X86ADC.W	Only set EFLAG's carry word plus in x86 mode	LoongEXT32
X86ADC.H	Only set EFLAG's carry halfword plus in x86	LoongEXT32
X86ADC.B	In x86 mode only set EFLAG's carry byte plus	LoongEXT32
X86ADDU.D	Set only EFLAG's no exception double word plus in x86	LoongEXT64
X86ADDU.W	In x86 mode, only set the EFLAG without exception word plus	LoongEXT32
X86SUB.D	Only set the double word subtraction of EFLAG in x86 mode	LoongEXT64

Page 52

Instruction mnemonic	Instruction function brief	**ISA** compatible category
X86SUB.W	Only set the word subtraction of EFLAG in x86 mode	LoongEXT32
X86SUB.H	Set only half-word subtraction of EFLAG in x86 mode	LoongEXT32
X86SUB.B	In x86 mode only set EFLAG's byte minus	LoongEXT32
X86SBC.D	Only set EFLAG's double word subtraction with borrow in x86 mode	LoongEXT64
X86SBC.W	Only set EFLAG subtraction with borrow word in x86 mode	LoongEXT32
X86SBC.H	Only set EFLAG with borrowed halfword subtraction in x86 mode	LoongEXT32
X86SBC.B	In x86 mode only set EFLAG with borrowed bytes minus	LoongEXT32
X86SUBU.D	Set only EFLAG's no exception double word subtraction in x86 mode	LoongEXT64
X86SUBU.W	Set only EFLAG's no exception word subtraction in x86 mode	LoongEXT32
X86INC.D	Only set the double word increment of EFLAG in x86 mode 1	LoongEXT64
X86INC.W	Only set EFLAG word increment by 1 in x86	LoongEXT32
X86INC.H	Set only half-word increment of EFLAG by x86	LoongEXT32
X86INC.B	Only set EFLAG byte increment by 1 in x86	LoongEXT32
X86DEC.D	Only set the double word decrement of EFLAG by 1 in x86	LoongEXT64
X86DEC.W	Only set EFLAG word decrement by 1 in x86	LoongEXT32
X86DEC.H	Set only half-word decrement of EFLAG by 1 in x86	LoongEXT32
X86DEC.B	Only set EFLAG byte decrement by 1 in x86	LoongEXT32
X86SLL.D	Set only double-word left shift of EFLAG in x86 mode	LoongEXT64
X86SLL.W	Set only the left shift of EFLAG in x86 mode	LoongEXT32
X86SLL.H	Set only half-word shift of EFLAG in x86 mode	LoongEXT32
X86SLL.B	Set only left shift of EFLAG in x86 mode	LoongEXT32
X86DSLL32	Set the shift amount of EFLAG plus the double-word logical left shift of 32 in x86 mode	LoongEXT64
X86SLLV.D	Set the double-word variable shift amount of EFLAG to the left by x86	LoongEXT64
X86SLLV.W	Set the variable shift amount of the word EFLAG only to the left in x86 mode	LoongEXT32
X86SLLV.H	Set only half-word variable shift amount of EFLAG left shift in x86 mode	LoongEXT32
X86SLLV.B	In x86 mode, only set EFLAG byte variable shift amount left shift	LoongEXT32
X86SRL.D	Set the double-word logical right shift of EFLAG only in x86 mode	LoongEXT64
X86SRL.W	Set only the right logical shift of EFLAG in x86 mode	LoongEXT32
X86SRL.H	Set only half-word logical right shift of EFLAG in x86 mode	LoongEXT32
X86SRL.B	Set the logical right shift of EFLAG's bytes only in x86 mode	LoongEXT32
X86DSRL32	In x86 mode, only set the shift amount of EFLAG plus 32 double-word logic right shift	LoongEXT64
X86SRLV.D	Set only double-word variable shift logic right shift of EFLAG in x86 mode	LoongEXT64
X86SRLV.W	Set only the right shift of the word variable shift amount of EFLAG in x86 mode	LoongEXT32
X86SRLV.H	Set only half-word variable shift amount logical right shift of EFLAG in x86 mode	LoongEXT32
X86SRLV.B	Set only the right shift of EFLAG's byte variable shift logically in x86	LoongEXT32
	Set only double-word arithmetic right shift of EFLAG in x86 mode	

X86SRA.D		LoongEXT64
X86SRA.W	Set only the word arithmetic right shift of EFLAG in x86 mode	LoongEXT32
X86SRA.H	Set only half-word arithmetic right shift of EFLAG in x86 mode	LoongEXT32
X86SRA.B	Set only EFLAG's byte arithmetic right shift in x86 mode	LoongEXT32
X86DSRA32	In x86 mode, only set the shift amount of EFLAG plus 32 double-word logical arithmetic right shift	LoongEXT64
X86SRAV.D	Set the double-word variable shift arithmetic right shift of EFLAG only in x86 mode	LoongEXT64

Instruction mnemonic	Instruction function brief	**ISA** compatible category
X86SRAV.W	Arithmetic right shift of only variable word shift amount of EFLAG in x86 mode	LoongEXT32
X86SRAV.H	Set only half-word variable shift arithmetic right shift of EFLAG in x86 mode	LoongEXT32
X86SRAV.B	Arbitrary right shift of EFLAG byte variable shift amount in x86 mode	LoongEXT32
X86ROTR.D	Set only double-word circular right shift of EFLAG in x86 mode	LoongEXT64
X86ROTR.W	Set only the EFLAG word rotation right in x86 mode	LoongEXT32
X86ROTR.H	Set only half-word rotation right of EFLAG in x86 mode	LoongEXT32
X86ROTR.B	Set only EFLAG byte rotation right by x86	LoongEXT32
X86DROTR32	In x86 mode, only set the shift amount of EFLAG plus 32 double-word logical circular right shift	LoongEXT64
X86ROTL.D	Set the double word rotation of EFLAG left by x86	LoongEXT64
X86ROTL.W	Set the word rotation of EFLAG only to the left in x86 mode	LoongEXT32
X86ROTL.H	Set only half-word rotation left of EFLAG in x86 mode	LoongEXT32
X86ROTL.B	Set only EFLAG byte rotation left in x86	LoongEXT32
X86DROTL32	In x86 mode, only set the shift amount of EFLAG plus 32 double-word logic circular left shift	LoongEXT64
X86RCR.D	In x86 mode, only the double word with CF bit of EFLAG is set to rotate right	LoongEXT64
X86RCR.W	In x86 mode, only the word with CF bit of EFLAG is set to rotate right	LoongEXT32
X86RCR.H	In x86 mode, only the half word with CF bit of EFLAG is set to rotate right	LoongEXT32
X86RCR.B	Set only the byte with CF bit of EFLAG to rotate right by x86	LoongEXT32
X86DRCR32	In x86 mode, only set the shift amount of EFLAG plus 32 double word logic cycle with CF bit Ring right	LoongEXT64
X86RCL.D	In x86 mode, only the double word with CF bit of EFLAG is set to rotate left	LoongEXT64
X86RCL.W	In x86 mode, only set the word with CF bit of EFLAG to rotate left	LoongEXT32
X86RCL.H	In x86 mode, only the half word with CF bit of EFLAG is set to rotate left	LoongEXT32
X86RCL.B	Set only the bytes with CF bit of EFLAG to rotate left by x86	LoongEXT32
X86DRCL32	In x86 mode, only set the shift amount of EFLAG plus 32 double word logic cycle with CF bit Ring shift left	LoongEXT64
X86ROTRV.D	Double-word circular right shift with only variable displacement of EFLAG set in x86 mode	LoongEXT64
X86ROTRV.W	Set the variable shift amount of EFLAG only in x86 mode	LoongEXT32
X86ROTRV.H	Set the variable shift amount of EFLAG only by half-word circular right shift in x86 mode	LoongEXT32
X86ROTRV.B	Set the variable shift amount of EFLAG by x86 to rotate the bytes to the right	LoongEXT32
X86ROTLV.D	Double-word cyclic left shift with only variable displacement of EFLAG set in x86 mode	LoongEXT64
X86ROTLV.W	In x86 mode, only the variable shift amount of EFLAG is set to rotate left	LoongEXT32
X86ROTLV.H	In x86 mode, only the half-word of the variable shift amount of EFLAG is set to rotate left	LoongEXT32
X86ROTLV.B	In x86 mode, only the variable shift amount of EFLAG is set to rotate left	LoongEXT32
X86RCRV.D	Double-word cyclic right shift with CF bit to set only variable shift amount of EFLAG in x86 mode	LoongEXT64
X86RCRV.W	In x86 mode, only the variable shift amount of EFLAG is set, and the word with CF bit is rotated right.	LoongEXT32
X86RCRV.H	In x86 mode, only the variable shift amount of EFLAG is set, and the half word with CF bit is rotated right	LoongEXT32
X86RCRV.B	In x86 mode, only the variable shift amount of EFLAG is set, and the byte with CF bit is rotated right.	LoongEXT32
X86RCLV.D	In x86 mode, only the variable shift amount of EFLAG is set, and the double word with CF bit is rotated left	LoongEXT64
X86RCLV.W	In the x86 mode, only the variable shift amount of EFLAG is set, and the word with CF bit is rotated left.	LoongEXT32
X86RCLV.H	In the x86 mode, only the variable shift amount of EFLAG is set, and the half word with CF bit is rotated left	LoongEXT32
X86RCLV.B	In x86 mode, only the variable shift amount of EFLAG is set, and the byte with CF bit is rotated left	LoongEXT32

Instruction mnemonic	Instruction function brief	**ISA** compatible category
X86MFFLAG	Extract the value of the EFLAG flag in x86	LoongEXT32
X86MTFLAG	Modify the value of the EFLAG flag in x86	LoongEXT32
X86J	Jump based on EFLAG value in X86 mode	LoongEXT32
X86LOOP	Cycle according to EFLAG value in X86 mode	LoongEXT32
SETTM	Set in x86 floating point stack mode	LoongEXT32
CLRTM	x86 floating-point stack mode clear	LoongEXT32
INCTOP	x86 floating point stack top pointer plus 1	LoongEXT32
DECTOP	x86 floating point stack top pointer minus 1	LoongEXT32
MTTOP	Write x86 floating point stack top pointer	LoongEXT32
MFTOP	Read x86 floating point stack top pointer	LoongEXT32
SETTAG	Judge and set register	LoongEXT32
CVT.D.LD	Convert double precision to double precision	LoongEXT32
CVT.LD.D	Conversion of double precision to extended double precision low	LoongEXT32
CVT.UD.D	Conversion of double precision to extended double precision high	LoongEXT32

Binary translation acceleration instruction **(ARM)**

Table **2-30** Godson extended **ARM** binary translation acceleration instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
ARMADC	With carry word plus, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMADD	Word plus, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSBC	With carry word subtraction, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSUB	Word subtraction, only set EFLAG conditionally executed in ARM mode	LoongEXT32
ARMAND	Word logic bit AND, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMBIC	Word logic bit NOT AND, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMMOV	Word movement, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMMVN	Word reversal moves, only conditional execution in ARM mode sets EFLAG	LoongEXT32
ARMMVLO32	The lower 32 bits of the LO register are moved to the general purpose register, and the conditional execution in the ARM mode only sets EFLAG	LoongEXT32
ARMMVHI32	The lower 32 bits of the HI register are moved to the general purpose register, and the conditional execution in the ARM mode only sets EFLAG	LoongEXT32
ARMMVACC64	The lower 32 bits of the HI register and the lower 32 bits of the LO register are concatenated into 64 bits, in ARM mode Conditional execution only sets EFLAG	LoongEXT32
ARMOR	Word logic bit OR, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMORN	Word logic bit is not OR, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMXOR	XOR of word logic bit, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSLL	Word left shift, conditional execution in ARM mode only set EFLAG	LoongEXT32
ARMSLLV	Variable shift amount word shift left, conditional execution in ARM mode only set EFLAG	LoongEXT32
ARMSRL	Word logic shifts to the right, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSRLV	Variable shift quantity word logical shift right, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSRA	Word arithmetic shifts to the right, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMSRAV	Variable shift word arithmetic right shift, conditional execution in ARM mode only set EFLAG	LoongEXT32

41

Page 55

Instruction mnemonic	Instruction function brief	**ISA** compatible category
ARMROTR	Word cycle right shift, conditional execution in ARM mode only set EFLAG	LoongEXT32
ARMROTRV	Variable shift amount word rotates to the right, conditional execution in ARM mode only sets EFLAG	LoongEXT32
ARMRRX	The conditional execution in ARM mode only sets EFLAG's carry word to rotate right by one bit	LoongEXT32
ARMFCMP.F32	Use ARM FCMP.F32 instruction to perform floating point comparison and set FCR1 EFLAGS	LoongEXT32
ARMFCMP.F64	Use ARM FCMP.F64 instruction to compare floating point, set FCR1 EFLAGS	LoongEXT32
ARMFCMPE.F32	Use ARM FCMP.F32 instruction to perform floating point comparison and set FCR1 EFLAGS	LoongEXT32
ARMFCMPE.F64	Use ARM FCMP.F64 instruction to compare floating point, set FCR1 EFLAGS	LoongEXT32
ARMMOVE	Move between general registers in ARM mode according to EFLAG conditions	LoongEXT32
ARMMFHI	HI moves to general register in ARM mode according to EFLAG condition	LoongEXT32

ARMMFLO	LO moves to general register in ARM mode according to EFLAG condition	LoongEXT32
ARMMFFCR	Move FCR1 EFLAGS to EFLAGS according to EFLAG conditions in ARM mode	LoongEXT32
ARMFMOV.S	Move single-precision numbers between floating-point registers in ARM mode according to EFLAG conditions	LoongEXT32
ARMFMOV.D	Move double-precision numbers between floating-point registers in ARM mode according to EFLAG conditions	LoongEXT32
ARMJ	Jump based on EFLAG value in ARM mode	LoongEXT32
ARMMFFLAG	Extract the value of EFLAG flag in ARM mode	LoongEXT32
ARMMTFLAG	Modify the value of EFLAG flag in ARM mode	LoongEXT32

64 -bit multimedia acceleration instructions

Table **2-31** Godson extended **64** -bit multimedia acceleration instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
PADDSH	Four 16-bit signed integer additions, signed saturation	LoongEXT32
PADDUSH	Four 16-bit unsigned integer addition, unsigned saturation	LoongEXT32
PADDH	Four 16-digit plus	LoongEXT32
PADDW	Two 32-digit plus	LoongEXT32
PADDSB	Eight 8-bit signed integer additions, signed saturation	LoongEXT32
PADDUSB	Eight 8-bit unsigned integer additions, unsigned saturation	LoongEXT32
PADDB	Eight 8-digit plus	LoongEXT32
PADDD	64 digit plus	LoongEXT32
PSUBSH	Four 16-bit signed integer subtraction, signed saturation	LoongEXT32
PSUBUSH	Four 16-bit unsigned integers, unsigned saturation	LoongEXT32
PSUBH	Four 16-digit minus	LoongEXT32
PSUBW	Two 32-bit minus	LoongEXT32
PSUBSB	Eight 8-bit signed integer subtraction, signed saturation	LoongEXT32
PSUBUSB	Eight 8-bit unsigned integers minus, unsigned saturation	LoongEXT32
PSUBB	Eight 8-digit minus	LoongEXT32
PSUBD	64 digit minus	LoongEXT32
PSHUFH	Shuffle four 16 digits	LoongEXT32
PACKSSWH	Convert 32-bit signed integer to 16-bit, signed saturation	LoongEXT32
PACKSSHB	16-bit signed integer converted to 8-bit, signed saturation	LoongEXT32
PACKUSHB	16-bit signed integer converted to 8-bit, unsigned saturation	LoongEXT32
PANDN	fs is negated and ft bitwise and	LoongEXT32

Instruction mnemonic	Instruction function brief	**ISA** compatible category
PUNPCKLHW	Unpack lower 16 digits	LoongEXT32
PUNPCKHHW	Unpack high 16 digits	LoongEXT32
PUNPCKLBH	Unpack lower 8 digits	LoongEXT32
PUNPCKHBH	Unpack high 8 digits	LoongEXT32
PINSRH_0	The lower 16 digits of ft are inserted into the lower 16 digits of fs	LoongEXT32
PINSRH_1	The lower 16 digits of ft are inserted into the lower 16 digits of fs	LoongEXT32
PINSRH_2	The lower 16 digits of ft are inserted into the lower 16 digits of fs	LoongEXT32
PINSRH_3	The lower 16 digits of ft are inserted into the lower 16 digits of fs	LoongEXT32
PAVGH	Four 16-bit unsigned integers are averaged	LoongEXT32
PAVGB	Eight 8-bit unsigned integers are averaged	LoongEXT32
PMAXSH	Four 16-bit signed integers, whichever is greater	LoongEXT32
PMINSH	Four 16-bit signed integers take the smaller value	LoongEXT32
PMAXUB	Eight 8-bit unsigned integers, whichever is greater	LoongEXT32
PMINUB	Eight 8-bit unsigned integers take the smaller value	LoongEXT32
PCMPEQW	Two 32-digit equal comparisons	LoongEXT32
PCMPGTW	Two 32-bit signed integers are greater than the comparison	LoongEXT32
PCMPEQH	Four 16-digit equality comparisons	LoongEXT32
PCMPGTH	Four 16-bit signed integers are greater than the comparison	LoongEXT32
PCMPEQB	Eight 8-digit equality comparisons	LoongEXT32
PCMPGTB	Eight 8-bit signed integers are greater than the comparison	LoongEXT32
PSLLW	Two 32-digit logical shifts to the left	LoongEXT32

PSLLH	Four 16-digit logical shift left	LoongEXT32
PMULLH	Multiply four 16-bit signed integers and take the lower 16 bits	LoongEXT32
PMULHH	Multiply four 16-bit signed integers and take the result as high 16 bits	LoongEXT32
PMULUW	Multiply the lower 32-bit unsigned integers and store the 64-bit result	LoongEXT32
PMULHUH	Multiply four 16-bit unsigned integers and take the result as high 16 bits	LoongEXT32
PSRLW	Two 32-bit logical shift right	LoongEXT32
PSRLH	Four 16-digit logical shift right	LoongEXT32
PSRAW	Two 32-digit arithmetic shift right	LoongEXT32
PSRAH	Four 16-digit arithmetic shift right	LoongEXT32
PUNPCKLWD	The lower 32-bit array synthesizes 64 digits	LoongEXT32
PUNPCKHWD	The upper 32-bit array synthesizes 64 digits	LoongEXT32
PASUBUB	Eight 8-bit unsigned integers are subtracted and take the absolute value	LoongEXT32
PEXTRH	fs 16 bits are copied to fd lower 16 bits, fd higher bits are filled with 0	LoongEXT32
PMADDHW	Four 16-bit signed numbers are multiplied and the low and high bits are accumulated separately	LoongEXT32
BIADD	Multi-byte cumulative summation	LoongEXT32
PMOVMSKB	Byte sign bit extraction	LoongEXT32
GSXOR	logical OR of fs and ft	LoongEXT32
GSNOR	fs and ft logical bit NOR	LoongEXT32
GSAND	fs and ft logical bitwise AND	LoongEXT32
GSADDU	fs and ft fixed-point unsigned word plus	LoongEXT32

Instruction mnemonic	Instruction function brief	**ISA** compatible category
GSOR	fs and ft fixed-point logical OR	LoongEXT32
GSADD	fs and ft fixed-point words plus	LoongEXT32
GSDADD	fs and ft fixed-point double word addition	LoongEXT32
GSSEQU	fs and ft fixed-point number equal comparison	LoongEXT32
GSSEQ	fs and ft fixed-point number equal comparison	LoongEXT32
GSSUBU	fs and ft fixed-point unsigned word subtraction	LoongEXT32
GSSUB	fs and ft fixed-point subtraction	LoongEXT32
GSDSUB	fs and ft fixed-point double word subtraction	LoongEXT32
GSSLTU	fs and ft fixed-point unsigned fixed-point number is less than comparison	LoongEXT32
GSSLT	fs and ft fixed-point fixed-point number is less than comparison	LoongEXT32
GSSLL	fs and ft fixed-point logical shift left	LoongEXT32
GSDSLL	fs and ft fixed-point logical double shift left	LoongEXT32
GSSRL	fs and ft fixed-point logical shift right	LoongEXT32
GSDSRL	fs and ft fixed-point logical shift right double word	LoongEXT32
GSSRA	fs and ft fixed-point arithmetic shift right	LoongEXT32
GSDSRA	fs and ft fixed-point arithmetic right shift double word	LoongEXT32
GSSLEU	fs and ft fixed-point unsigned fixed-point number less than or equal to the comparison	LoongEXT32
GSSLE	fs and ft fixed-point fixed-point number is less than or equal to the comparison	LoongEXT32

Miscellaneous instructions

Table **2-32** Godson extended miscellaneous instructions

Instruction mnemonic	Instruction function brief	**ISA** compatible category
CTZ	Trailing 0 digits	LoongEXT32
CTO	Trailing 1 number	LoongEXT32
DCTZ	Double word trailing 0 number	LoongEXT64
DCTO	Double word trailing 1 number	LoongEXT64
CAMPV	Query the RAM entry value of the lookup table	LoongEXT32
CAMPI	Query the index of the lookup table	LoongEXT32
CAMWI	Fill in the lookup table	LoongEXT32
RAMRI	Read the RAM contents of the lookup table	LoongEXT32

44

Page 58

3 processor operating mode

The operation mode of GS464E is compatible with the MIPS64 specification and includes 2 operation modes, namely: Debug Mode and Root Mode
(Root Mode). Among them, the debug mode mainly corresponds to the operating environment of the EJTAG exception handler; the root mode corresponds to the operating system on the real
The operating environment of the software on it. Among them, the root mode can be further divided into root-kernel mode (Root-Kernel Mode), root-supervision mode
(Root-Supervisor Mode) and Root-User Mode. All operating modes are independent of each other, which means that at any moment,
The processor can only exist in a certain operating mode.

Among the above four models, the root-regulatory model is rarely used in practice, and the MIPS specification does not fully define its connotation.
The following description will only briefly explain this model. It is not recommended that programmers use the root-supervision model to build software, if it is really necessary, please directl
Refer to the MIPS specification.

3.1 Definition of processor operating mode

Table 3-1 gives the judgment basis of each mode of the processor.

Table **3-1** Basis of processor mode judgment

Root CP0				mode
Debug.DM	**Status.ERL**	**Status.EXL**	**Status.KSU**	
1		Don't care		Debug mode
0	1	Don't care		Root-core model
	0	1	Don't care	
		0	00	
			01	Root-supervision model
			10	Root-user mode
	Don't care		11	Meaningless

3.1.1 Debug mode

The debug mode has the highest priority. In debug mode, the software can operate all processor resources, including changing the mapping between virtual and real addresses,
Control system environment and process switching, etc.

3.1.2 Root - Core Mode

In the root-core mode, the software can operate all processor resources, including changing the mapping between virtual and real addresses, controlling the system environment and proc
Change. The processor enters root-core mode after power-on reset.

3.1.3 root - user mode

In root-user mode, the software does not allow access to the processor's privileged sensitive resources, but can execute unprivileged instructions, use general-purpose registers and
For floating-point registers, all memory accesses fall on a flat, uniform virtual address space. Ordinary user programs run in root-user mode.

45

4 memory management

4.1 Basic concepts

4.1.1 Address space

The address space refers to all address ranges that can be covered in a particular addressing mode. MIPS64 architecture contains a 64-bit address space
As well as a 32-bit address space, the latter is simultaneously mapped as a subset of the former.

4.1.2 Segment and segment size **(SEGBITS)**

A segment is a subset of the address space, and the address space in the same segment has a consistent mapping mode and access attributes. According to the MIPS64 specification
For example, its 32-bit address space is divided into a series of segments with a size of 2 29 or 2 31 bytes, and its 64-bit address space can theoretically support
Over 2 62- byte segments. There is no need to implement such a large segment in practice. The actual segment size (SEGBITS) determines that the address space is divided into
A series of segments with a size of 2 SEGBITS bytes.

4.1.3 Physical address size **(PABITS)**

The physical address size (PABITS) determines the size of the physical address space that the processor actually supports is 2 PABITS bytes.

4.1.4 Mapping address **(Mapped Address)** and non-mapped address **(Unmapped Address)**

The address of "Mapped Address" means that the address needs to be translated by virtual and real addresses through TLB. "Unmapped address
(Unmapped Address) "means that the address does not need to undergo virtual and real address translation through TLB, and the virtual address is directly linearly mapped to the physical addr
The lowest part.

4.2 Host virtual address space

4.2.1 Host address space division and access control

Table 4-1 shows the division of the host address space, and defines the legality determination and address mapping method of each address segment. Need note
Intentionally, SEGBITS is always 48 in the host address space.

Table **4-1** Host address space division and access control

Segment name	Address range	Legality determination and address mapping		
		User mode	Regulatory model	Core mode
kseg3	0xFFFF.FFFF.FFFF.FFFF ~ 0xFFFF.FFFF.E000.0000	Illegal address segment	Illegal address segment	Mapped address segment TLB refill exception type: TLB (Status.KX = 0) XTLB (Status.KX = 1). When Debug.DM = 1, related For special treatment of address space, please refer to See section 4.2.5

Segment name	Address range	Legality determination and address mapping		
		User mode	Regulatory model	Core mode
ksseg sseg	0xFFFF.FFFF.DFFF.FFFF ~ 0xFFFF.FFFF.C000.0000	Illegal address segment	Mapped address segment TLB refill exception type: TLB (Status.KX = 0) XTLB (Status.KX = 1).	Mapped address segment TLB refill exception type: TLB (Status.KX = 0) XTLB (Status.KX = 1).
kseg1	0xFFFF.FFFF.BFFF.FFFF ~	Illegal address segment	Illegal address segment	Unmapped address segment Please see further 4 .2.2 section

kseg0	0xFFFF.FFFF.A000.0000 ~ 0xFFFF.FFFF.9FFF.FFFF	Illegal address segment	Illegal address segment	Unmapped address segment Please see further 4 .2.2 section
	0xFFFF.FFFF.8000.0000 ~ 0xC000.FFFF.7FFF.FFFF	Illegal address segment	Illegal address segment	Illegal address segment
xkseg	0xC000.0000.0000.0000 ~ 0xBFFF.FFFF.FFFF.FFFF	Illegal address segment	Illegal address segment	Status.KX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.
xkphys	0x8000.0000.0000.0000 ~ 0x7FFF.FFFF.FFFF.FFFF	Illegal address segment	Illegal address segment	Status.KX = 0 is illegal Address segment Otherwise legal, its internal legality Determine and use the address mapping please See section 4.2.3
	0x4001.0000.0000.0000 ~ 0x4000.FFFF.FFFF.FFFF	Illegal address segment	Illegal address segment	Illegal address segment
xksseg xsseg	0x4000.0000.0000.0000 ~ 0x3FFF.FFFF.FFFF.FFFF	Illegal address segment	Status.SX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.	Status.SX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.
	0x0001.0000.0000.0000 ~	Illegal address segment	Illegal address segment	Illegal address segment
xkuseg xsuseg xuseg	0x0000.FFFF.FFFF.FFFF ~ 0x0000.0000.8000.0000	Status.UX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.	Status.UX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.	Status.UX = 0 is illegal Address segment Otherwise legal, it is the mapped address segment TLB refill exception type: XTLB.

Segment name	Address range	Legality determination and address mapping		
		User mode	Regulatory model	Core mode
kuseg suseg useg	0x0000.0000.7FFF.FFFF ~ 0x0000.0000.0000.0000	Mapped address segment TLB refill exception type: TLB (Status.UX = 0) XTLB (Status.UX = 1).	Mapped address segment TLB refill exception type: TLB (Status.KX = 0) XTLB (Status.KX = 1).	Status.ERL = 1 is unmapped Address section, please see further Section 4.2.4 Status.UX = 0 when mapping Address segment, TLB refill exception type: TLB (Status.UX = 0) XTLB (Status.UX = 1)

4.2.2 host address space **kseg0** segment **kseg1** address translation segment, cacheability and cache-coherent properties

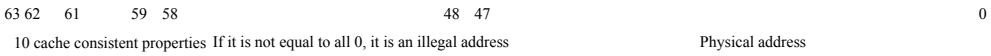
The kseg0 segment and kseg1 segment of the host address space are directly mapped to the lowest 0.5G (2 29) bytes of the physical address space , that is, the virtual address 0xFFFF.FFFF.A000.000 ~ 0xFFFF.FFFF.BFFF.FFFF is mapped to the physical address 0x0000.0000.0000.0000 ~ 0x0000.0000.1FFF.FFFF, virtual addresses 0xFFFF.FFFF.8000.000 ~ 0xFFFF.FFFF.9FFF.FFFF are also mapped to physical addresses 0x0000.0000.000(0.0000.1FFF.FFFF. The cache consistency attribute of the kseg0 segment is determined by the Config.K0 field, which is specifically defined Please refer to sec ge 116 . The kseg1 segment is always Uncached.

4.2.3 Address translation, **cacheability** and cache consistency attributes of the **xkphys** segment of the host address space

The xkphys segment of the host address space uses an unmapped method, which contains 8 sub-address segments, each of which is 2 48 bytes in size. xkphys The segment virtual address resolution method is shown in Figure 4-1 . The [58:48] of the virtual address must be all 0s, otherwise it is an illegal address. The virtual address of [47: 0] does nc

TLB or any other translation process. Directly as a physica ;9] bits of the virtual address are used to define the cache consistency of the corresponding sub-address segment
Refer to Table 7-6 on page 89 for the definition of the attri method used .

Figure **4-1** rtual address resolution method



4.2.4 host address space **kuseg** segment **Status.ERL = 1** NAT when

When Status.ERL = 1, the kuseg segment is an unmapped address segment, and its cache consistency attribute is uncached, similar to the kseg1 segment. This
The feature is that when handling Cache error exceptions, the software can use the general register R0 as the base address register to save other
Use registers to store in memory. At the same time, due to errors in Cache, memory access operations are no longer cached.

4.2.5 Special handling of the host address space **kseg3** segment when **Debug.DM = 1**

When the processor is in debug mode (Debug.DM = 1), the virtual address 0xFFFF.FFFF.FF20.0000 in the kseg3 segment
The address range of 0xFFFF.FFFF.FF3F.FFFF will be used as a special memory address mapping area-the dseg segment of EJTAG. About EJTAG
For a detailed description of the dseg section, please refer to the first error **!** No reference source found. chapter.

4.2.6 Special handling of data access virtual address when **Status.UX = 0 in** user mode

In user mode, when a 32-bit program is compatible with a 64-bit MIPS processor, special processing is required for the virtual address of data access.
This is because a calculation process that can obtain a legal address on a 32-bit MIPS processor may produce an unpredictable result on a 64-bit MIPS processor.
Period effect. For example, the following sequence of instructions:

```
la r1, 0x80000000
```



```
lw r2, -4 (r1)
```

When executed on a 32-bit MIPS processor, the virtual address of the lw instruction is 0x80000000 + 0xFFFFFFFC = 0x7FFFFFFF. Income
The address is still in the kuseg segment. But when this code is executed on a 64-bit MIPS processor, the virtual address bit of the lw instruction is 0xFFFFFFFF80000000
+ 0xFFFFFFFFFFFFFFFC = 0xFFFFFFFF7FFFFFFC. The resulting address is no longer in the kuseg segment, which will result in an incorrect address exception
Health. In order to maintain the compatibility of 64-bit processors for 32-bit programs, when Status.UX = 0, the calculation of the data access virtual address has been carried out in a special p
Management. The specific method is that the address operation still uses the two 64-bit numbers after sign expansion, but the higher 32 bits of the resulting result are discarded by
The 31st sign of the result is extended to 63..32 bits of the resulting virtual address. The virtual address result after such special processing is only used for address legality check
Check, TLB mapping and other operations. The normal instruction fetch operation will not involve this problem, because the 31st bit of a legal PC in 32-bit user mode must be 0,
There will be no violations mentioned above.

4.3 Virtual and Real Address Mapping Based on **TLB**

TLB is a temporary cache for storing the operating system page table information in the processor, which is used to accelerate the fetching and accessing operations on the mapped addre
Virtual and real address conversion process.

4.3.1 TLB hierarchy

Two levels of TLB are implemented in GS464E. The first level of TLB is a fully associative search with a smaller capacity included in the fetch and fetch parts
The TLBs are called ITLB and DTLB respectively; the second-level TLB has a larger capacity and contains a lookup table that is fully connected and an eight-way group,
Call it JTLB.

All content software of JTLB is visible, and the filling and replacement of table entries is managed by the software. All content software of ITLB and DTLB is not visible, its
The filling and replacement of the table items is maintained by the hardware. When the processor is running, the contents of the page tables stored in DTLB and JTLB are always in an inclusiv
The relationship is automatically maintained by the hardware. However, the contents of ITLB and JTLB are not explicitly included and mutually exclusive, that is to say, the software cannot c
After maintaining the contents of the JTLB to determine the information stored in the ITLB. In the general operation process, the information stored in ITLB and JTLB does not maintain the p
Inclusive relations have no effect on the correctness of the software. However, if the operating system attempts to modify an existing page table information, and the page table
The program code is stored in the address space corresponding to the item, then the system software must write 1 to the Diag.ITLB bit to explicitly clear all the contents of ITLB
Content to ensure that there is no longer any content in the ITLB before the modification of the page table entry.

Both ITLB and DTLB use a fully associative lookup table structure. The entry information stored in ITLB and DTLB comes from JTLB, of which DTLB
The information stored in the table entries is exactly the same as the format in JTLB, and each entry in ITLB only stores one page table instead of a pair of odd and even adjacent page tables.
Because the status replacement of ITLB and DTLB does not require software to participate, the specific format of its table entries is not expanded here.

4.3.2 JTLB organization structure

From a software perspective, JTLB contains a fully associative lookup table and a multi-way group associative lookup table. The former supports the use of different entries
Different page sizes are called variable-page-size TLB (Variable-Page-Size TLB), referred to as VTLB; the latter at the same time all the page pages

Same small, called fixed-page-size TLB (Fixed-Page-Size TLB), referred to as FTLB. In the process of virtual and real address translation, VTLB and FTLB
Find at the same time. Accordingly, the software needs to ensure that VTLB and FTLB will not have multiple hits, otherwise the processor behavior will be unknowable.

The organization and operation of VTLB are very similar to the MIPS traditional fully-linked TLB, which is 64 items in GS464E. in case
If the GSConfig.VTLBOnly bit is 1, all functions of FTLB are disabled and only VTLB is retained. In this configuration, Godson 3A1000 chip
The TLB management part of the existing operating system can be correctly executed on the Godson 3A3000 chip without modification.

The FTLB organization is an 8-way group-linked structure, each way contains 128 items, a total of 1024 items, each item stores 2 page table information, so the most
Can store 2048 page table information. During the search process, the hardware will extract the virtual address of [(17 + Config4.FTLBPageSize):
(11 + Config4.FTLBPageSize)] bits are used as index information to compare the contents of the same index position items in each way to determine whether there is a match
Matching.

4.3.3 JTLB entries

The format of VTLB and FTLB entries is basically the same, the only difference is that each entry with VTLB contains PageMask information, while FTLB
Because it is the same page size, PageMask information is not maintained. Each JTLB entry contains two parts: a comparison part and a physical conversion part.

The comparison part of the table entry includes:

- Page table invalid bit (EHINV). When this bit is 1, the page table entry does not participate in finding matches.
- Address space number (MID). Used to identify which address space the page table entry address is in.
- Virtual processor number and its mask (VPID, VPMSK). VPID & VPMSK is the virtual processor number where the page table entry address is located.
- Virtual address area identification (R) and virtual page number (VPN2). In the MIPS architecture, each page table entry stores an adjacent pair of parity neighbors
  Page table information, so the virtual page number stored in the TLB page table entry is the content of the virtual page number / 2 in the system, that is, the lowest bit of the virtual
  It is only used to find out whether to select the physical conversion information of odd-numbered pages or even-numbered pages when searching.
- Address space identification (ASID) and global flag (G). The address space identifier is used to distinguish the same virtual address in different processes, operation
  The system assigns a unique ASID to each process. In addition to the address information, the TLB needs to match the ASID letter when searching.
  interest. When the operating system needs to share the same virtual address among all processes, you can set the G bit and G position in the TLB page table entry
  1 After the TLB search, the ASID consistency check will no longer be performed.
- Address page mask (Mask). The address mask is used to control the size of the page table stored in the page table entry. GS464E supports 4KB to 1GB,
  The page size increases in multiples of 4. For VTLB, each item has Mask information, so different items can correspond to different pages
  size. For FTLB, all items use unified Mask inf

For the explanation of the above fields, please refer t

Related to EntryLo1 register (CP0 Register 2 and 3, Select
description.

The physical conversion part of the table entry stores a pair of parity adjacent page table physical conversion information. The conversion information of each page includes:

- Physical page number (PFNX & PFN).
- Significant bit (V).
- Dirty bit (D). The control bit of whether the page is writable, not the status bit of whether the page is written with dirty data.
- Read prohibit bit (RI).
- Execution inhibit bit (XI).
- The kernel execution protection bit (K) is only meaningful in 64-bit mode. In 32-bit mode, please set GSConfig.KE to 0 constantly to ensure K
  Bit has no effect.
- Cache attribute (C).

For the explanation of the above fields, please refe
Related description.

4.3.4 TLB software management

GS464E still follows the traditional MIPS architecture for the management of TLB, that is, the software-led and software-hardware combination management
TLB. Hardware Page Table Walking function added in MIPS specification release 5 and later
Not implemented in GS464E. GS464E provides a set of privileged access instructions specifically for page table traversal search based on the MIPS specification
It is used to speed up this process and will be briefly introduced in the second half of this section.

Related exception

Page 65

TLB virtual and real address conversion process is automatically completed by the hardware, but when there is no match in the TLB, or the page table entry is invalid despite the match Or when the access is illegal, an exception needs to be triggered, which is handed over to the operating system kernel or other supervisory program, and further processed by the software. Perform maintenance, or make a final ruling on the legality of program execution. The exceptions related to TLB management in GS464E are:

1. TLB refill exception
2. XTLB refill exception
3. TLB invalid exception
4. TLB modification exception
5. Unexecutable exception
6. Unreadable exception

Related **CP0** registers

Table 4- 2 lists the CP0 registers related to TLB management. For detailed description of each register : refer to the corresponding section in Chapter 7.

Table **4-2 CP0** registers related to **TLB** management

Reg.	**Sel.**	Register name	Function definition	in
0	0	Index	VTLB and FTLB access the specified index register	Page 85, S
1	0	Random	VTLB and FTLB access random index register	Page 86 S
2	0	EntryLo0	VTLB and FTLB entries in the low-order content related to even virtual pages	Page 87, S
3	0	EntryLo1	VTLB and FTLB entries in the low-order content related to odd virtual pages	Page 87, S
4	0	Context	Pointer to page table entry in memory	Page 90, S
5	0	PageMask	VTLB page table size control	Page 92, S
5	1	PageGrain	1KB small page and other page table attribute control	Page 93, S
5	5	PWBase	Page table base address register	Page 94, S
5	6	PWField	Configure page table address index positions at all levels	Page 95, S
5	7	PWSize	Configure page table pointer size at all levels	Page 96 S
6	0	Wired	Control the number of fixed items in VTLB	Page 97 S
6	6	PWCtl	Control multi-level page table configuration	Page 98 S
8	0	BadVAddr	Record the wrong address of the latest address related exception	Page 100
9	7	PGD	Page table pointer register	Page 103 S
10	0	EntryHi	High-level content of VTLB and FTLB entries	Page 104 S
20	0	XContext	Page table pointer in extended address mode	Page 128 S
twenty two	0	Diag	Godson Extended Diagnostic Control Register	Page 129, S

Related privileged instructions

Table 4-3 lists the privileged instructions related to TLB management. For detailed definition of each instruction, please refer to "MIPS® Architecture For Programmers Volume II-A: The MIPS64® Instruction Set "(Rev5.03) and" Godson Instruction Set Manual (Volume II-a)-from Define General Extended Instructions (Volume 1) (Rev1.00).

Table **4-3 TLB** management related privileged instructions

Instruction mnemonic	Instruction brief description

Page 66

Instruction mnemonic	Instruction brief description
TLBP	Search for matches in TLB
TLBR	Read indexed TLB entries
TLBWI	Write indexed TLB entries
TLBWR	Write random TLB entries
TLBINVF	Invalidate all TLB entries

LWDIR	Load instruction for page table directory entry in 32-bit mode
LWPTE	Page table page table entry load instruction in 32-bit mode
LDDIR	Load instruction of page table directory entry in 64-bit mode
LDPTE	Page table page table entry load instruction in 64-bit mode

GS464E implements the EHINV field in the EntryHi register. When EntryHi.EHINV is set to 1, the execution of the TLBWI instruction will index
The specified TLB entry is invalid.

VTLB and FTLB use a unified software and hardware interaction interface, that is, use the same set of CP0 registers and the same TLB privileged instructions. Here
Emphasize the following points to be noted:

- VTLB can retain some non-randomly replaceable items through the Wired register, but there is no non-randomly replaceable items in FTLB.
- When the TLBWR instruction is used to fill in TLB entries randomly. If the page size in the Pagemask register and the fixed page of FTLB at this time
  When the sizes are different, the entries are randomly filled in VTLB; if they are the same, the entries are randomly filled in FTLB.
- The randomly filled position in the VTLB is determined by the value of the Random register; the randomly filled position in the FTLB is determined by a single
  Independent hardware pseudo-random number generator decision.
- When using the TLBRI and TLBWI instructions to read and write JTLB, the value in the Index register is 0 ~ 63, which corresponds to items 0 ~ 63 of VTLB.
  The value is 64 ~ 1087, which corresponds to FTLB No. 0 item 0 ~ 127, No. 1 item 0 ~ 127, ..., No. 7 item 0 ~ 127
  item. When using the TLBP instruction to perform a software search, the results stored in the Index register also follow the above correspondence.

4.3.5 **TLB** initialization and clearing

It is recommended to use the TLBINVF instruction to initialize or clear the TLB by invalidating all TLB entries. You can also use the EntryHi register
In the EHINV field in the cyclic execution of the TLBWI instruction, all items in the TLB are invalid one by one.

There is a correspondence between the position of the entry in the FTLB and the virtual address it stores, so a specific address is written to the TLB entry
The traditional way of initializing TLB may not guarantee correct initialization. When GSConfig.VTLBOnly = 0, that is, when FTLB is enabled, the software
The software must use the two methods described above to initialize or clear the TLB. The traditional way of initializing TLB is not introduced in this manual, I am interested
Readers can refer to "MIPS® Architecture For Programmers Volume III: The MIPS64® and microMIPS64 ™
Section 4.11.3 of Privileged Resource Architecture (Rev5.03).

4.3.6 Based **TLB** actual situation of address conversion process

Here only introduces the process of virtual and real address conversion based on software-visible TLB. Processor hardware performs both VTLB and FTLB
For the search, the process of checking VTLB first and then FTLB when introduced in the form of pseudocode is only for the convenience of description. System software needs to ensure the
The address cannot have multiple hits in VTLB and FTLB.

```
// va- virtual address to be found
// mid-the MID of the virtual address to be found

/ * VTLB search process * /
vtlb_found ← 0
53
```


Loongson **3A3000 / 3B3000** Processor User Manual • Next

```
for i = 0 to 63 step 1
  if ((VTLB [i] .EHINV == 0) &&
       (VTLB [i] .MID == mid) &&
       ((VTLB [i] .VPID & VTLB [i] .VPMSK) == (Diag.VPID & Diag.VPMSK)) &&
       (VTLB [i] .G || (VTLB [i] .ASID == EntryHi.ASID)) &&
       (VTLB [i] .R == va [63:62]) &&
       ((VTLB [i] .VPN2 [34:27] & ~ VTLB [i] .VPMSK) == (va [47:40] & ~ Diag.VPMSK)) &&
       (VTLB [i] .VPN2 [26:18] == va [39:31]) &&
       ((VTLB [i] .VPN2 [17: 0] & ~ VTLB [i] .MASK) == (va 30..13 & ~ VTLB [i] .MASK))) then
    if (! vtlb_found) then
      vtlb_found ← 1
      case VTLB [i] .MASK
        0b00 0000 0000 0000 0000: vtlb_evenoddbit ← 12 // 4KB page
        0b00 0000 0000 0000 0011: vtlb_evenoddbit ← 14 // 16KB page
        0b00 0000 0000 0000 1111: vtlb_evenoddbit ← 16 // 64KB page
        0b00 0000 0000 0011 1111: vtlb_evenoddbit ← 18 // 256KB page
        0b00 0000 0000 1111 1111: vtlb_evenoddbit ← 20 // 1MB page
        0b00 0000 0011 1111 1111: vtlb_evenoddbit ← 22 // 4MB page
        0b00 0000 1111 1111 1111: vtlb_evenoddbit ← 24 // 16MB page
        0b00 0011 1111 1111 1111: vtlb_evenoddbit ← 26 // 64MB page
```

```
        0b00 1111 1111 1111 1111: vtlb_evenoddbit ← 28 // 256MB page
        0b11 1111 1111 1111 1111: vtlb_evenoddbit ← 30 // 1GB page
        otherwise: UNDIFINED
      endcase
      if va [vtlb_evenoddbit] == 0 then
        vtlb_pfn ← VTLB [i] .PFN0
        vtlb_v ← VTLB [i] .V0
        vtlb_c ← VTLB [i] .C0
        vtlb_d ← VTLB [i] .D0
        vtlb_ri ← VTLB [i] .RI0
        vtlb_xi ← VTLB [i] .XI0
        vtlb_k ← VTLB [i] .K0
      else
        vtlb_pfn ← VTLB [i] .PFN1
        vtlb_v ← VTLB [i] .V1
        vtlb_c ← VTLB [i] .C1
        vtlb_d ← VTLB [i] .D1
        vtlb_ri ← VTLB [i] .RI1
        vtlb_xi ← VTLB [i] .XI1
        vtlb_k ← VTLB [i] .K1
      endif
    else
      UNDIFINED
    endif
54
```

Page 68

Loongson **3A3000 / 3B3000** Processor User Manual • Next

```
  endif
endfor

/ * FTLB search process * /
ftlb_found ← 0
case Config4.FTLBPageSize
  0d1: idx ← va [18:12]; mask ← 0x00000
  0d2: idx ← va [20:14]; mask ← 0x00003
  0d3: idx ← va [22:16]; mask ← 0x0000f
  0d4: idx ← va [24:18]; mask ← 0x0003f
  0d5: idx ← va [26:20]; mask ← 0x000ff
  0d6: idx ← va [28:22]; mask ← 0x003ff
  0d7: idx ← va [ 30:24 ]; mask ← 0x00fff
  0d8: idx ← va [32:26]; mask ← 0x03fff
  0d9: idx ← va [34:28]; mask ← 0x0ffff
  0d10: idx ← va [36:30]; mask ← 0x3ffff
  otherwise: UNDIFINED
endcase
for set = 0 to 7 step 1
  FTLB [set] [idx]
  if ((FTLB [set] [idx] .EHINV == 0) &&
       (FTLB [set] [idx] .MID == mid) &&
       ((FTLB [set] [idx] .VPID & FTLB [set] [idx] .VPMSK) == (Diag.VPID & Diag.VPMSK)) &&
       (FTLB [set] [idx] .G || (FTLB [set] [idx] .ASID == EntryHi.ASID)) &&
       (FTLB [set] [idx] .R == va [63:62]) &&
       ((FTLB [set] [idx] .VPN2 [34:27] & ~ FTLB [set] [idx] .VPMSK) == (va [47:40] &
~ Diag.VPMSK)) &&
       (FTLB [set] [idx] .VPN2 [26:18] == va [39:31]) &&
       ((FTLB [set] [idx] .VPN2 [17: 0] & ~ mask) == (va 30..13 & ~ mask))) then
    if (! ftlb_found) then
      ftlb_found ← 1
      if va [ftlb_evenoddbit] == 0 then
        ftlb_pfn ← FTLB [set] [idx] .PFN0
```

```
        ftlb_v ← FTLB [set] [idx] .V0
        ftlb_c ← FTLB [set] [idx] .C0
        ftlb_d ← FTLB [set] [idx] .D0
        ftlb_ri ← FTLB [set] [idx] .RI0
        ftlb_xi ← FTLB [set] [idx] .XI0
        ftlb_k ← FTLB [set] [idx] .K0
      else
        ftlb_pfn ← FTLB [set] [idx] .PFN1
        ftlb_v ← FTLB [set] [idx] .V1
        ftlb_c ← FTLB [set] [idx] .C1
        ftlb_d ← FTLB [set] [idx] .D1
55
```

Page 69

Loongson **3A3000 / 3B3000** Processor User Manual • Next

```
        ftlb_ri ← FTLB [set] [idx] .RI1
        ftlb_xi ← FTLB [set] [idx] .XI1
        ftlb_k ← FTLB [set] [idx] .K1
      endif
    else
      UNDIFINED
    endif
  endif
endfor

/ * Legality check and physical address generation * /
if (vtlb_found && ftlb_found) then
  UNDIFINED
elseif (vtlb_found) then
  if (! vtlb_v) then
    SignalException (TLBInvalid, reftype)
  endif
  if (vtlb_ri && (reftype == load)) then
    if (PageGrain.IEC) then
      SignalException (TLBRI, reftype)
    else
      SignalException (TLBInvalid, reftype)
    endif
  endif
  if (vtlb_xi && (reftype == fetch)) then
    if (PageGrain.IEC) then
      SignalException (TLBXI, reftype)
    else
      SignalException (TLBInvalid, reftype)
    endif
  endif
  if (! vtlb_d && (reftype == store)) then
    SignalException (TLBModified)
  endif
  PAddr ← vtlb_pfn [35: vtlb_evenoddbit-12] || va [vtlb_evenoddbit-1: 0]
elseif (ftlb_found) then
  if (! ftlb_v) then
    SignalException (TLBInvalid, reftype)
  endif
  if (ftlb_ri && (reftype == load)) then
    if (PageGrain.IEC) then
      SignalException (TLBRI, reftype)
    else
      SignalException (TLBInvalid, reftype)
56
```

Page 70

Loongson **3A3000 / 3B3000** Processor User Manual • Next

```
    endif
  endif
  if (ftlb_xi && (reftype == fetch)) then
    if (PageGrain.IEC) then
      SignalException (TLBXI, reftype)
    else
      SignalException (TLBInvalid, reftype)
    endif
  endif
  if (! ftlb_d && (reftype == store)) then
    SignalException (TLBModified)
  endif
  PAddr ← ftlb_pfn [35: ftlb_evenoddbit-12] || va [ftlb_evenoddbit-1: 0]
else
  SignalException (TLBMiss, reftype)
endif
```

57

Page 72

Loongson **3A3000 / 3B3000** Processor User Manual • Next

5 Cache organization and management

Under the MIPS architecture, all levels of cache in the processor are visible to the core software. For some operations of the cache (such as cache initial Software, consistency maintenance, etc.), software needs to participate in cache management. This chapter introduces the cache organization and management of GS464E. This manual does n

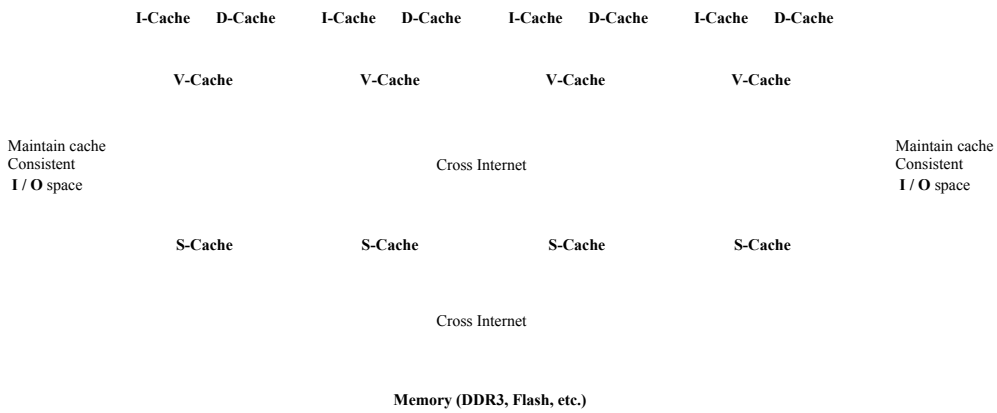
Describe the general concepts of processor cache, such as index, tag, cache line, group association, cache access, cache hit
And missing, virtual address index physical address tag (Virtual Index Physical Tag) cache, cache replacement, etc. If readers are not familiar with related concepts
For clarity, please refer to the relevant books on computing architecture and recommend "MIPS Architecture Perspective" (Second Edition).

5.1 Processor storage hierarchy and cache organization structure at all levels

5.1.1 Processor storage hierarchy

The Loongson 3A3000 chip processor uses a storage hierarchy with three levels of cache, as shown in Figure 5-1 .

Figure **5-1** Godson **3A3000** chip processor storage hierarchy



According to the distance between the caches at all levels and the processor pipeline, from near to far, the order is: the first level instruction cache (Instruction-Cache, I-Cache) and data cache (Data-Cache, D-Cache), the second level of victim cache (Victim-Cache, V-Cache), the third level of shared cache Save (Shared-Cache, S-Cache). Among them, I-Cache, D-Cache and V-Cache are private to each processor core, and S-Cache is multi-core and I / O shared. The processor core accesses the S-Cache through the internal network of the chip and between the chips.

The I-Cache only stores the content that the processor access component needs to access, and the D-Cache only stores the content that the processor access component needs to access. Both V-Cache and S-Cache are hybrid caches, which store both instructions and data.

The contents of I-Cache and D-Cache are exclusive to the contents of V-Cache, that is, the contents of the same physical address are stored in I-Cache or D-Cache will no longer be stored in V-Cache. Contents in I-Cache, D-Cache and V-Cache and those in S-Cache The content is inclusive, that is, the content of the same physical address, as long as it is stored in I-Cache, D-Cache or V-Cache, A copy of the same physical address can also be found in S-Cache. The relationship between the above mutual exclusion and inclusion is carried out from the observable angle of the software The description does not indicate the positional relationship of the data in the real storage medium at any time.

During operation, the data consistency between I-Cache, D-Cache, V-Cache and S-Cache is maintained by hardware.

When the fetching component finds a miss in the I-Cache or the fetching component D-Cache, it will first search for the V-Cache. If the V-Cache hits, Then fill the cache line hit in V-Cache into I-Cache or D-Cache, and replace the cache line from I-Cache or D-Cache (if there is In case), backfill to the location of the V-Cache to take out the cache line. If the V-Cache does not hit, the S-Cache is further searched. S-Cache After the response is returned, it will be directly filled into the I-Cache or D-Cache, and the cache line replaced from the I-Cache or D-Cache (if it exists) Backfill to the location in the V-Cache where the cache line is fetched. After the S-Cache receives the request from the processor core, the processing process to be performed involves The maintenance of cache consistency will be described in detail in Section 5.3 .

Table 5-1 lists some parameters of each cache.

Table **5-1** Cache parameters

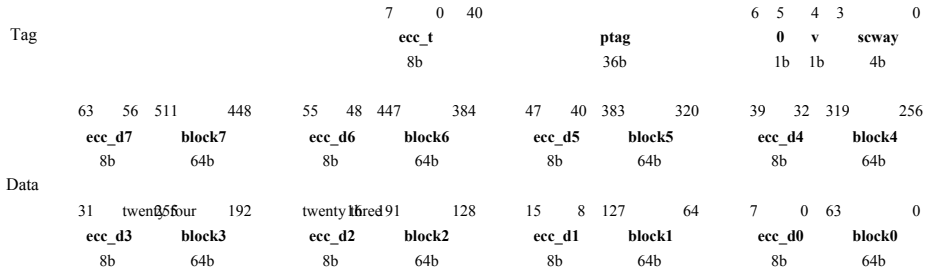
	Instruction cache	Data cache	Sacrifice cache	Shared cache
capacity	64KB	64KB	256KB	2MB / body (chip total 8MB)
Degree of association	4 way	4 way	16 Road	16 Road
Line size **(line size)**	512bit	512bit	512bit	512bit
Index **(Index)**	Virtual address [13: 6]	Virtual address [13: 6]	Virtual address [13: 6]	11 bits in the p Please refer to : page 63
Tag **(Tag)**	Physical address [47:12]	Physical address [47:12]	Physical address [47:12]	Physical ad
Replacement strategy	Random replacement algorithm	LRU replacement algorithm	LRU replacement algorithm	LRU replacement algorithm
Write strategy		Write back, write allocation	Write back, write allocation	Write back, write allocation

Verification method	Parity check	SEC-DED ECC	SEC-DED ECC	SEC-DED ECC

5.1.2 First-level instruction cache **(I-Cache)**

The capacity of the first-level instruction cache is 64KB, using a 4-way set associative structure and a random replacement algorithm. The length of the data part in each cache line is 64 Bytes are divided into 8 8-byte wide blocks. The block is the smallest unit of data part access. Instruction cache uses virtual address index physical address label Signed access mode. During access, the [13: 6] bits of the virtual address are used as the index of the cache line, and the fifth and lower parts of the virtual address are used in the cache line Index, the 14th and above parts of the virtual address are converted into virtual and real addresses at the same time. The content is compared to determine whether the cache hit . Figure 5-2 shows the structure of the instruction cache line. In addition to storing physical addresses in Tag In addition to the high bit (ptag), it also includes the valid bit (v) and the information of the cache line where the cache line is located in the S-Cache (scway). The valid bit is 1 means the The content on the cache line is meaningful. 0 means there is no valid content on the cache line.

Figure **5-2** Structure diagram of the first-level instruction cache line



The first-level instruction cache uses parity to verify the Tag and Data parts of the cache line. When a new cache line is updated into the index When caching, the Data part uses blocks as the basic unit of verification. Each block generates an 8-bit verification result and records it, and the Tag part expands it by 0. After expanding to 64 bits, the 8-bit check result generated by the same check algorithm is also recorded. When reading the cache, the original data and the reference check value are

At the same time, read out, recalculate the check value of the original data, if it is inconsistent with the reference check value, it indicates that a cache error has occurred, the hardware The current cache line is invalid in the I-Cache, and the relevant location information is recorded, triggering an exception. If the software does not require special diagnosis, you can directly Return from the exception handler. After the processor resumes execution, the required cache line content will be retrieved from the V-Cache, S-Cache, or memory. need It should be noted that when the software uses the Store Tag and Store Data Cache instructions to fill the instruction cache, it must also calculate the filled in The parity value of the content is explicitly stored in the ErrCtl.ECC field. When the hardware executes this type of Cache instruction, the reference check written in the instruction cache The value comes from the ErrCtl.ECC field instead of the hardware circuit automatic check generation result. This mechanism is mainly used to complete some special diagnosis. Instruction The stored algorithm parity codes for generating and detecting parity values are described as follows:

Parity value generation algorithm:

```
function Parity_Gen (datain 63..0 , parityout 7..0 )
endfunction Parity_Gen
```

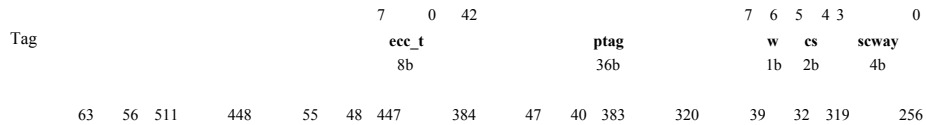
Parity check algorithm:

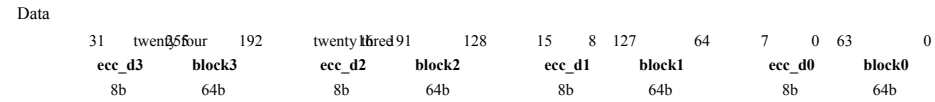
```
function Parity_Check (newparity 7..0 , refparity 7..0 , error)
endfunction Parity_Check
```

5.1.3 First-level data cache **(D-Cache)**

The capacity of the first-level data cache is 64KB, using a 4-way set associative structure and LRU replacement algorithm. The length of the data portion in each cache line is 64 bytes, divided into 8 8-byte wide blocks (block), the block is the smallest unit of data access. The data cache uses a virtual address index to physically The access mode of the address tag. When accessing, the [13: 6] bits of the virtual address are used as the index of the cache line, and the fifth and lower parts of the virtual address are used fo In-line index, the virtual and real addresses are converted at the 14th bit and above of the virtual address at the same time. The content is compared to determine whether the cache hit. Figure 5-3 shows the structure of the data cache line. In addition to storing physical In addition to the high-order bit (ptag) of the address, it also includes the cache line status information (cs), the dirty tag bit (w), and the information of the number of the cache line in the S-Ca 息 (scway). cs = 0 means the cache line is invalid; cs = 1 means the cache line is in a shared state; cs = 2 means the cache line is in an exclusive state; cs = 3 is Keep the value. w = 1 indicates that there is newly written data on the cache line.

Figure **5-3** Schematic diagram of the primary data cache line structure





The first-level data cache uses "SEC-DED" ECC check to verify the Tag and Data parts of the cache line. When one
When the new cache line is updated into the data cache, the Data section uses blocks as the basic unit of verification, and each block generates an 8-bit verification result.
After recording, for Tag, the part 0 except the dirty mark bit (w) is expanded to 64 bits, and the same check algorithm is used to generate an 8-bit check result.
Be recorded. It should be noted that the dirty tag bit in the Tag part does not participate in the verification because this part of the information is stored with other parts of the Tag
The physical media is different and is not stored in SRAM. When reading the cache, the original data and the reference check value are read out at the same time.
After recalculating the check value and comparing it with the reference check value, if a one-bit error is found, the hardware will automatically correct the error and
Fill the value back into the D-Cache, and record the relevant location information, triggering an exception; if the number of errors exceeds one bit, the hardware will not be able to correct it, or
61

Page 75

Relevant location information, trigger an exception. When one bit is wrong during the exception, the software can directly return from the exception handler if there is no special diagnosis.
When there is more than one bit error, it usually needs more complete recovery, such as soft reset. When you need to pay attention, when the software uses Store Tag and Store
When the Data type Cache instruction fills the data cache, the ECC check value of the filled content must be calculated at the same time and explicitly stored in the ErrCtl.ECC field.
When the hardware executes this type of Cache instruction, the reference check value written in the data cache comes from the ErrCtl.ECC field instead of the hardware circuit automatically c
Verify the results. This mechanism is mainly used to complete some special diagnosis. The algorithm pseudocode for generating and detecting the ECC check value of the data cache is descrit
under:

ECC check value generation algorithm:

```
function ECC_Gen ();
endfunction ECC_Gen
```

ECC check detection algorithm:

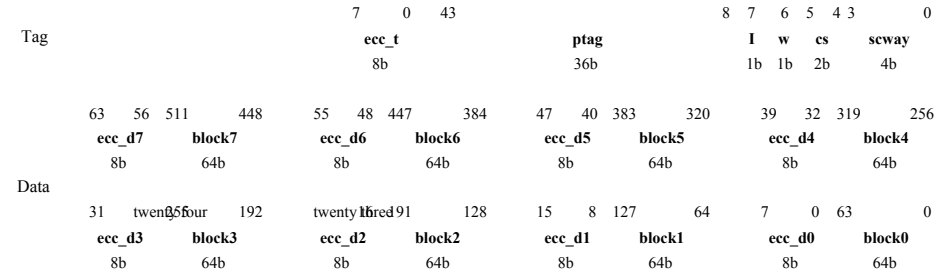
```
function ECC_Check ();
endfunction ECC_Check
```

5.1.4 two victim cache **(V-Cache)**

The capacity of the secondary sacrificial cache is 256KB, using a 16-way set associative structure and LRU replacement algorithm. Sacrificing cache using virtual address index
Manage the access mode of the address label. The data portion of each cache line is 64 bytes long and is divided into 8 8-byte wide blocks. Usual visit
Q Always read or write all the data in the cache line, only when the Load Data and Store Data class Cache instructions are executed, the adjacent parity
The block is used as the basic unit. At this time, [5: 4] of the physical address is used to indicate which pair of adjacent parity blocks are operated.

When accessing the secondary sacrificial cache, the [13: 6] bits of the physical address are used as the index of the cache line, the high bits of the physical address and the content read by
Compare to determine whether the cache hits; if it hits, read the data content of the corresponding cache block in the way. Figure 5-4
The structure of the cache line is sacrificed. In addition to the high-order bits (ptag) of the physical address, the Tag also includes cache line status information (cs), dirty tags
Count bit (w), instruction mark (I), and the information (scway) of the cache line in the S-Cache. cs = 0 means the cache line is invalid; cs = 1
Indicates that the cache line is in a shared state; cs = 2 indicates that the cache line is in an exclusive state; cs = 3 is a reserved value. w = 1 means there is a recent write on this cache line
The data entered. I = 1 indicates that the cache line stores instructions, and I = 0 indicates that the cache line stores data.

Figure **5-4** Structure diagram of the secondary sacrificial cache line



The second-level victim cache uses the "SEC-DED" ECC check to verify the Tag and Data parts of the cache line. When one
When the new cache line is updated into the sacrificial cache, the Data section uses blocks as the basic unit of verification, and each block generates an 8-bit verification result.
Recorded; after the Tag part 0 is extended to 64 bits, the same verification algorithm is used to generate an 8-bit verification result and recorded. When reading the cache, the original
The original data and the reference check value are read out at the same time, and the original data is recalculated and compared with the reference check value. If it is found to be a compariso
When there is an error, the hardware will automatically correct the error, fill the corrected value back into the V-Cache, and record the relevant location information to trigger an exception; if a

If the number of digits exceeds one bit, the hardware cannot correct it. It should be noted that when the software uses Store Tag and Store Data Cache instructions to fill in
When data is cached, the ECC check value of the filled content must be calculated at the same time and explicitly stored in the ErrCtl.ECC field. The hardware is performing this type of Cach
When instructing, the reference check value written in the sacrificing cache comes from the ErrCtl.ECC field instead of the hardware :ic check generation result. The main mech
To be used to complete some special diagnosis. Algorithms and Data cache coherent victim cache ECC check value generation and dc 5 .1.3
Section.

5.1.5 Three-level shared cache **(S-Cache)**

The three-level shared cache supports cache consistency based on the directory protocol. All the on-chip S-Cache addressing of Loongson 3A3000 chip, each shared
The cache line has a fixed home node.

Split structure of shared cache

Loongson 3A3000 chip S-Cache adopts split structure, which is divided into 4 banks, which are received through the first-level crossbar interconnection network
Access requests from processor cores and I / O ports that maintain cache coherency. Because Loongson 3A3000 chip is in the first level crossbar internet
The software uses a dynamically adjustable address window mapping mechanism, so the physical address seen by each S-Cache body is re-addressed through the address window
The newly mapped address must be clear when the software operates the S-Cache. Which of the 4 S-Cache bodies does the different requests end up in?
The two are determined by the two bits in the address, which two bits of the specific address are dynamically adjustable by the software, by the chip configuration register
SCID_SEL decides. The corresponding relationship between the configuration information and the address bits of the S-Cache body is given. Correspondingly, which physical addresses
The index of the bit used for the cache line also changes as the value of SCID_SEL changes.

Table **5-2** Three-level shared cache body selection bit and index address

SCID_SEL value	Body selection	Index address
b0000	PAddr [7: 6]	PAddr [18: 8]
b0001	PAddr [9: 8]	{PAddr [18:10], PAddr [7: 6]}
b0010	PAddr [11:10]	{PAddr [18:12], PAddr [9: 6]}
b0011	PAddr [13:12]	{PAddr [18:14], PAddr [11: 6]}
b0100	PAddr [15:14]	{PAddr [18:16], PAddr [13: 6]}
b0101 ~ b1111	PAddr [18:17]	PAddr [16: 6]

Shared cache lock mechanism

The shared cache has a single capacity of 2MB and adopts a 16-way group associative structure. In addition to using the LRU algorithm to select alternatives, the shared cache also supp
Support cache lock mechanism. There are two ways to lock the cache: one is to use the Cache15 instruction to lock a cache line; the other is to use the chip
The shared cache lock window mechanism in the configuration register locks the sliced physical address space. Once the locked content is stored in the shared cache, it will not
It is replaced again, unless the following two situations occur: (1) All Cache lines in the 16-way S-Cache that have the same Index as the locked Cache line are
In the "locked" state, the locks of all Cache lines are deemed invalid, and the replacement is still selected according to the LRU algorithm; (2) The software uses the Cache instruction
The cache line whose effect is "locked". The two locking mechanisms have their own advantages and disadvantages: the advantage of using the Cache15 instruction is that it can be directly us
Lock Cache operation, and if the data is not in S-Cache, the Cache row to be locked will be retrieved into S-Cache and then locked, the disadvantage is that Cache
Both lock and release operations need to be performed on a cache-by-cache line, and there is a certain overhead; the advantage of using the lock window mechanism is that it is configured onc
Window configuration register) can lock a large continuous address space (theoretically no more than 15/16 of the S-Cache capacity, or 3.75MB),
The disadvantage is that the configuration must use physical address information, which requires special support from the operating system kernel, and the data cannot be guaranteed after conf
In S-Cache. Software personnel can s ate S-Cache lock mechanism for program optimization according to the specific characteristics of the application. About Cache15 means
For the specific definition of the orde he description in section 2.4.9 on page 33 . For the detailed definition of the S-Cache lock window configuration register, please refer to
3A3000 / 3B3000 Processor User Ma described in Section 2.5.

Cache line structure of shared cache

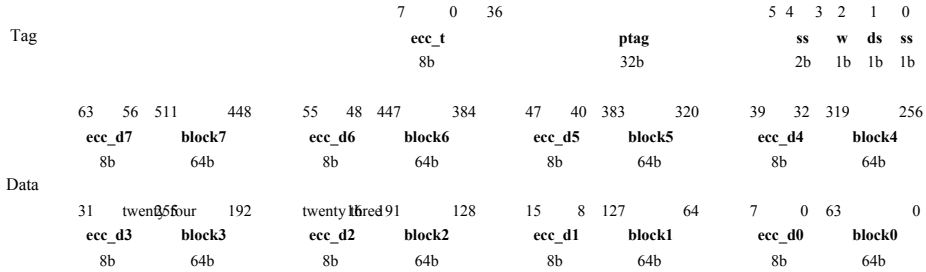
The shared cache adopts the access mode of physical address index and physical address label. The length of the data part in each cache line is 64 bytes, divided into 8

8-byte wide blocks. The usual access is always to read or write all the data part of the cache line, only in the implementation of Load Data and Store
The Data class Cache instruction uses two adjacent blocks of parity as the basic unit. At this time, the physical address of [5: 4] is used to indicate which pair of adjacent parity is operated.
Piece.

When accessing, the high address of the physical address is compared with the content read by the tag in each way to determine whether the cache hits; if it hits, it will hit
The data content of the corresponding cache block in that way is read out. Figure 5-5 shows the structure of the shared cache line. In addition to storing physical

In addition to the high-order address (ptag), it also includes cache line status information (ss), directory status information (ds), dirty bit (w), and page coloring bit (pgc). ss = 1 means
The cache line is valid, ss = 0 means the cache line is invalid. ds = 1 means the directory is dirty, ds = 0 means the directory is clean. w = 1 means there is new on this cache line
Recently written data. Indicates that there is newly written data on this cache line. Page coloring bits are used by the hardware to handle cache aliasing issues, please see 6 9 for details
Page 5.4.5 description.

Figure **5-5** Schematic diagram of the three-level shared cache line



Shared cache verification

The three-level shared cache uses the "SEC-DED" ECC check to verify the Tag and Data parts of the cache line. When one
When the new cache line is updated into the shared cache, the Data section uses blocks as the basic unit of verification, and each block generates an 8-bit verification result.
Recorded; after the Tag part 0 is extended to 64 bits, the same verification algorithm is used to generate an 8-bit verification result and recorded. When reading the cache, the original
The original data and the reference check value are read out at the same time, and the original data is recalculated and compared with the reference check value. If it is found to be a compariso
When there is an error, the hardware will automatically correct the error, fill the corrected value back into the S-Cache, and record the relevant location information to trigger an exception; if a
If the number of digits exceeds one bit, the hardware cannot correct it. It should be noted that when the software uses Store Tag and Store Data Cache instructions to fill in
When data is cached, the ECC check value of the filled content must be calculated at the same time and explicitly stored in the ErrCtl.ECC field. The hardware is performing this type of Cach
When instructing, the reference check value written in the sacrificing cache comes from the ErrCtl.ECC field instead of the hardware :ic check generation result. The main mech
To be used to complete some special diagnosis. Algorithms and Data cache coherent victim cache ECC check value generation and dc 5 .1.3
Section.

5.2 Cache algorithm and cache consistent attributes

GS464E supports three cache algorithms and cache coherent attributes: Uncached, Coherent Cache (Cacheable Coherent) and Non-Cache
Cache acceleration (Uncached Accelerated). The consistency algorithm code corresponding to the non-cache algorithm is 0b010, and the consistency cache algorithm corresponds to the consis
The performance algorithm code is 0b011, and the consistency algorithm code corresponding to the non-cache acceleration algorithm is 0b111.

5.2.1 Non-cache algorithm

When an address segment or page uses a non-cache algorithm, the fetching or fetching operation of the virtual address on the address segment or page will be caused by
The processor directly initiates an access request directly to the location of the target address, and the data read or written does not originate or terminate in any level one cache.

64

Page 78

All access requests using non-cache algorithms are executed sequentially in a blocking manner. That is, before the current read request data is returned to the processor, all subsequent requests
The request is blocked; the write request data has not yet been sent or the write request has not been received before the final write response returned by the receiver, all
Subsequent requests are blocked.

5.2.2 Consistency cache algorithm

When an address segment or page uses a non-cache algorithm, the fetch or fetch operation of the virtual address on the address segment or page is accessed
Content can reside in any level one cache. GS464E maintains cache coherence by hardware, no software is required to be invalid by using Cache instruction,
Write back the contents of the cache to maintain cache consistency.

5.2.3 Non-cache acceleration algorithm

The non-cache acceleration algorithm attribute is used to optimize a series of sequential Uncached memory of the same type completed in a continuous address space
数 操作。 Number operation. The optimization method is to collect data of this algorithm attribute by setting a buffer. As long as the buffer is not full, you can put these
The data stored in the operation is stored in the buffer. The buffer size is consistent with one cache line, which is 64 bytes. Store data operation to store data to buffer
The district is considered to be completed. When the buffer data is full, it will be written out in one go. Continuously written data will be written directly
The location of the target address will not stay in any level one cache. In the data collection process of the sequential store instruction, if there is a common type of non-slow
If the save number instruction is inserted, the collection will be stopped, and the data saved in the buffer will be output in byte write mode. Non-cache acceleration algorithm attribute retrieval

The operation effect of the finger or fetch operation is the same as the finger fetch or fetch operation with the attributes of ordinary non-cache algorithms.

The non-cache acceleration attribute can speed up sequential Uncached access, which is suitable for fast output access to display device storage.

5.3 Cache consistency

GS464E implements a directory-based cache coherence protocol, and the hardware guarantees I-Cache, D-Cache, V-Cache, S-Cache, and memory And the consistency of data between IO devices from HT, no software is required to use the Cache instruction to maintain cache consistency.

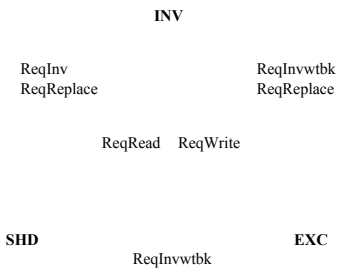
Each cache line in GS464E has a fixed host S-Cache body. The directory information of the Cache line is in the host S-Cache body maintain. The directory uses 64-bit bit vectors to record the first-level cache (including I-Cache and D-Cache) with a backup of each Cache line. each There are three possible states of a first-level cache block: INV (invalid state), SHD (shared state, readable), and EXC (exclusive state, readable) Writable). The transition between the three states is shown in Figure 5-6 .

When the read instruction or the fetch operation fails in both the first-level and second-level cache, the processor core issues a Reqread request to the third-level S-Cache, After receiving the Repread response from the S-Cache, the first-level cache of the processor core obtains a SHD state cache line backup. when When the storage operation fails in both the first-level and second-level caches, the processor core issues a Reqwrite request to the third-level S-Cache to obtain the S-Cache After the returned Repwrite response, the first-level cache of the processor core obtains an EXC state cache line backup.

When V-Cache replacement occurs on the processor core, write back to the S-Cache module through Reqreplace, and S-Cache responds to the report through Repreplace Know that the processor core replacement request has been processed.

S-Cache can invalidate an SHD state in I-Cache, D-Cache or V-Cache by sending a Reqinv request to the processor core Cache line backup, the processor core changes the corresponding Cache line to INV state and responds to S-Cache through Repinv. S-Cache can be sent by Reqwtbk request to the processor core, write back an EXC state cache line backup, the processor core will change the corresponding cache line backup to SHD state And reply to S-Cache through Repwtbk. S-Cache can write Reqinvwtbk request to the processor core to write back and invalidate one Cache line backup in EXC state, the processor core changes the corresponding Cache line backup to INV state and responds to the secondary cache through Repinvwtbk Module.

Figure **5-6** Cache state transition under the consistency protocol



5.4 Cache management

5.4.1 CACHE instruction

The CACHE instructions implemented by this processor are for I-Cache, D-Cache, V-Cache and S-Cache respectively. CACHE rmat It is: CACHE op, offset (base). There are some differences between the Cache instruction of GS464E and the MIPS64 specification, t been on page 33 2.4.9 The subsections are explained in detail. Here again, the MIPS specification distinguishes whether the operated object is level 2 or not p [1: 0] = 2 or = 3 The provisions of Level 3 Cache do not apply to GS464E. In order to maintain the forward compatibility of the software as much as possible, op [1: 0] = 2 indicates that the operation object is S-Cache (level 3), op [1: 0] = 3 indicates that the operation object is V-Cache (level 2). In this coding format, Godson 3A1000 chip uses Software code that uses S-Cache to maintain Cache consistency can still achieve the same Cache consistency maintenance effect in Godson 3A3000 chip fruit. That is, when the software invalidates an S-Cache line, the hardware guarantees that the same physical address is invalidated at the same time in the cache in all processor cores; When the software is invalid and writes back an S-Cache line, the hardware guarantees that the same physical address is invalidated in the cache in all processor cores And the content written back to the main memory must contain the latest written data.

CACHE instruction in root mode

The core state software running in root mode can use all the implemented CACHE instructions, the specific list is as follows:

Table **5-3 CACHE** instructions in root mode

op [4: 0]	Functional description	Target **Cache**
	Invalid cache line based on index	

b00000		I-Cache
b01000	Write Cache line Tag based on index	I-Cache
b11100	Write Cache data according to index	I-Cache
b00001	According to the invalid index and write back to the Cache line	D-Cache
b00101	Read Cache line Tag based on index	D-Cache
b01001	Write Cache line Tag based on index	D-Cache
b10001	According to hit invalid cache line	D-Cache
b10101	According to the invalid hit and write back the Cache line	D-Cache
b11001	Read Cache row Data according to index	D-Cache
b11101	Write Cache data according to index	D-Cache
b00010	According to the invalid index and write back to the Cache line	V-Cache
b00011	According to the invalid index and write back to the Cache line	S-Cache
b00111	Read Cache line Tag based on index	S-Cache
b01111	Retrieve and latch the cache line according to the address	S-Cache

op [4: 0]	Functional description	Target **Cache**
b01011	Write Cache line Tag based on index	S-Cache
b10011	According to the invalid hit and write back the Cache line	S-Cache
b11011	Read Cache row Data according to index	S-Cache
b11111	Write Cache data according to index	S-Cache

The CACHE2 command is mainly used to clear the V-Cache of this core when only a single core is shut down.
The correctness of the result at CACHE2.

First of all, when using the CACHE2 instruction to clear the V-Cache, make sure that the code executed in this process is located in the uncache space.

Secondly, when using the CACHE2 instruction to clear the V-Cache, do not perform other load and store operations in the cache space.

Use of **CACHE** instruction in guest mode

The use of the CACHE instruction in guest mode is controlled by root mode, which is defined as follows:

- When GuestCtl0.CG = 0, using any CACHE instruction will trigger the guest mode privilege sensitive instruction exception (GPSI).
- When GuestCtl0.CG = 1, the CACHE instruction with op [4: 2] = 1, 2, 6, 7 will trigger the guest mode privilege sensitive instruction exception (GPSI).
- GuestCtl0.CG = 1, but Diag.GCAC = 0, using the CACHE0, CACHE1, CACHE3 instructions will trigger the guest mode feature
  Power-sensitive instruction exception (GPSI).

5.4.2 Cache initialization

Hardware-based cache initialization

GS464E sets all cached Tag parts to all 0s during hardware restart, that is, invalidates all cached cache lines. Therefore except
In addition to the following special use case, the software does not need to initialize all caches.

The special case that causes the hardware initialization cache to be unsafe is that after the processor hard restarts, the software directly uses the Index Store Tag class Cache instruction to
The Tag of a Cache line is set to be valid, but all the blocks in the Data section of the Cache line are not used by the Index Store Data class Cache instruction
Set to certain content.

If the software does exist the above sequence of operations, please ensure that the cache is based on the software before performing operations that pose a security risk.
Line initialization.

Software-based cache initialization

The software-based cache initialization process recommended by GS464E is as follows:

Step 1 : Open up a section of memory and fill it with arbitrary data. This data will then be used to fill the cached data portion to form the correct check value. Suggest
Use the memory area starting at address 0 .

Step 2 : Mask interrupts to prevent unexpected situations during the initialization process.

Step 3 : Initialize the I-Cache .

```
mtc0 zero, TagLo; mtc0 zero, TagHi;

mtc0 zero, ErrCtl; / * 64 bits are all 0 parity value 0x0 * /
for (addr = 0xffffffff80000000; addr <0xffffffff80010000; addr + = 64) {
  / * Set each line of I-Cache to a certain value line by line * /
  for (way = 0; way <4; way + = 1) {
```

67

Page 81


```
    Cache_Index_Store_Tag_I (addr + way);
    for (block = 0; block <8; block + = 1) {
      Cache_Index_Store_Data_I (addr + (block << 3) + way);
    }
  }
}
```

Step 4 : Initialize D-Cache .

```
mtc0 zero, TagLo; mtc0 zero, TagHi;
addiu r1, r1, 0x22; mtc0 r1, ErrCtl; / * 64 bits are all 0 the ECC check value 0x22 * /
for (addr = 0xffffffff80000000; addr <0xffffffff80010000; addr + = 64) {
  / * Set D-Cache line by line to a determined value line by line * /
  for (way = 0; way <4; way + = 1) {
    Cache_Index_Store_Tag_D (addr + way);
    for (block = 0; block <8; block + = 1) {
      Cache_Index_Store_Data_D (addr + (block << 3) + way);
    }
  }
}
```

Step 5 : Initialize V-Cache .

```
mtc0 zero, TagLo; mtc0 zero, TagHi;
addiu r1, r1, 0x22; mtc0 r1, ErrCtl; / * 64 bits are all 0 the ECC check value 0x22 * /
for (addr = 0xffffffff80000000; addr <0xffffffff80010000; addr + = 64) {
  / * Set each line of V-Cache to a certain value line by line * /
  for (way = 0; way <16; way + = 1) {
    Cache_Index_Store_Tag_V (addr + way);
    for (block_pair = 0; block_pair <4; block_pair + = 1) {
      Cache_Index_Store_Data_V (addr + (block_pair << 4) + way);
    }
  }
}
```

Step 6 : Initialize S-Cache [1] .

```
Scache_init_ok [get_my_CPUNum ()] = 0; / * require uncached write * /
mtc0 zero, TagLo; mtc0 zero, TagHi;
addiu r1, r1, 0x22; mtc0 r1, ErrCtl; / * 64 bits are all 0 the ECC check value 0x22 * /
/ * No. 0 processor initializes S-Cache Bank 0 , No. 1 processor initializes S-Cache Bank 1 , and so on * /
bank = get_my_CPUNum ();
for (addr = 0xffffffff80000000; addr <0xffffffff80040000; addr + = 256) {
  / * Set each line of S-Cache to a certain value line by line * /
  for (way = 0; way <16; way + = 1) {
```

1 need SCID_SEL chip configuration register equal to the default value of zero.

68

Page 82


```
    Cache_Index_Store_Tag_V (addr + (bank << 6) + way);
    for (block_pair = 0; block_pair <4; block_pair + = 1) {
```

```
        Cache_Index_Store_Data_V (addr + (bank << 6) + (block_pair << 4) + way);
      }
    }
  }
  Scache_init_ok [get_my_CPUNum ()] = 1; / * require uncached write * /
```

Step 7 : Polling the Scache_init_ok vector until all entries are 1 . / * Polling requires uncached read * /

At this point, the cache initialization is complete.

/ * For steps 6 and 7 , the software can also use other synchronization mechanisms, only one of the simplest solutions is given here * /

5.4.3 Consistency maintenance between first-level instruction cache and first-level data cache

For applications with "self-modifying code", there is a data consistency problem between the first-level instruction cache and the first-level data cache. GS464E
The hardware maintains the data consistency between the first-level instruction cache and the first-level data cache, without the need for software to use CACHE instructions or SYNCI instruc
By refreshing and clearing D-Cache and I-Cache to maintain data consistency.

It should be pointed out that the GS464E implements a weakly consistent storage model. Therefore, after modifying the code, the software must use jr.hb or jalr.hb
The instruction jumps to the modified code for execution. In addition to the jump, jr.hb and jalr.hb will also act as a barrier. Thus ensuring jr.hb
Or the PC at the jump target of jalr.hb must see the modification effect of the write operation before the barrier.

5.4.4 Cache consistency maintenance between the processor and the **DMA** device

In order to improve the performance of the processor, the driver software of the DMA device puts the data that needs a lot of interaction in the cache space, resulting in a place
The problem of maintaining cache consistency between the processor and the DMA device. The detailed description of this problem is mentioned in many drive development books, this manu
Repeat again. In the Godson 3A3000 chip, the hardware can maintain the cache coherence between the processor and the DMA device connected to the HT port.
Therefore, when a DMA device in the system accesses the system main memory through the HT port, the device driver software does not need to use the Cache
Let S-Cache be used to maintain data consistency. Doing so can improve the execution efficiency of the processor.

It should be pointed out that the GS464E implements a weakly consistent storage model. Therefore, the processor and the DMA device still need to be stored in
The semaphore or interrupt mechanism of the non-cached (uncached) area to achieve the synchronization effect of the barrier to ensure that consumers can indeed read data
Observe the data that the producer wants to observe.

5.4.5 Cache aliases and page coloring

Because the first-level instruction cache and the first-level data cache of GS464E use the access mode of virtual index physical tags, and each way of the cache is large
The small size is 16KB, so when the page size is 4KB or 8KB, there will be a cache alias problem. The common way to solve the cache alias is to use "page
"Coloring" mechanism, which ensures that at any time, a physical address has at most one "page color" of a virtual address. GS464E is implemented by hardware
The "page coloring" mechanism is no longer implemented by software.

In the vast majority of cases, the software can ignore the cache alias problem, because the hardware has ensured that there is no cache through "page coloring"
name. The only special case is when the software directly controls the cache at all levels through the Index Store Tag and Index Store Data Cache instructions
Create effective Cache lines, and use the data in these Cache lines with subsequent programs. At this time, the software must guarantee S-Cache
The content of the page color field (pgc) in the row Tag matches the virtual address of the Cache row data. Specifically, the pgc field in the Tag of S-Cache
Must always be equal to [13:12] bits of the virtual address of the cache line.

70

6 processor exceptions and interrupts

6.1 processor exception

6.1.1 Exception priority

When an instruction meets multiple exception triggering conditions at the same time, GS464E will follow the exception priority shown in Table 6-1 , the priority trigger priority is high Exception.

Table **6-1** Exception Priority

exception	Types of
Cold reset	Asynchronous, reset
EJTAG single step exception	Synchronization and debugging
EJTAG debug interrupt exception	Asynchronous, debugging
Non-maskable interrupt	asynchronous
EJTAG instruction breakpoint exception	Synchronization and debugging
Address error exception-fetch instructions	Synchronize
TLB / XTLB refill exception-fetch instructions	Synchronize
TLB invalid exception—fetch instruction	Synchronize
TLB execution block exception	Synchronize
Cache error exception-fetch instructions	Synchronize
EJTAG SDBBP exception	Synchronize
Coprocessor unavailable exception	Synchronize
Exceptions to reserved orders	Synchronize
Interrupt	asynchronous
Integer overflow exception, trap exception, system call exception, breakpoint	Synchronize

Exception, floating point exception, floating point stack exception	
EJTAG precise data breakpoint exception	Synchronization and debugging
Address error exception-data access	Synchronize
TLB / XTLB refill exception-data access	Synchronize
TLB invalid exception-data access	Synchronize
TLB read block exception	Synchronize
TLB modification exception	Synchronize
Cache error exception-data access	Synchronize

6.1.2 Exceptional entry vector position

The exception entry vector addresses for cold reset, soft reset, and non-maskable interrupts all use the dedicated address 0xFFFF.FFFF.BFC0.0000, which This address is neither accessed through Cache nor address mapping.

The vector addresses of EJTAG debugging related exceptions are selected according to whether the ProbeTrap bit in the EJTAG control register is 0 or 1. 0xFFFF.FFFF.BFC0.0480 and 0xFFFF.FFFF.FF20.0200.

All other exception vector addresses are defined by "base address + offset". When Status.BEV = 0, the base addresses of various exceptions are fixed Configuration; when Status.BEV = 0, the software can configure the exception vector base address through the EBase register and the GSEBase register. Table 6-2 List For the exception vector base address definition, Table 6-3 lists the exception offset definition.

Table **6-2** Exception vector base address

exception	**Status.BEV = 0**	**Status.BEV = 1**
Cold reset, soft reset, non-maskable interrupt	0xFFFF.FFFF.BFC0.0000	
EJTAG debugging related exceptions (ProbTrap = 0)	0xFFFF.FFFF.BFC0.0480	
EJTAG debugging related exceptions (ProbTrap = 1)	0xFFFF.FFFF.FF20.0200	
Cache error exception	EBase 63..30 || 1 || EBase 28..12 || 0x000	0xFFFF.FFFF.BFC0.0200
Other exceptions	EBase 63..12 || 0x000	0xFFFF.FFFF.BFC0.0200

Table **6-3** Exception Vector Offset

exception	Vector offset
Cold reset, soft reset, non-maskable interrupt	No offset, use base address directly
Various EJTAG debugging exceptions (ProbTrap = 0)	No offset, use base address directly
Various EJTAG debugging exceptions (ProbTrap = 1)	No offset, use base address directly
TLB refill exception (Status.EXL = 0)	0x000
XTLB refill exception (Status.EXL = 0)	0x080
Cache error exception	0x100
Other exceptions	0x180
Interrupt (Cause.IV = 0)	0x180
Interrupt (IntCtl.VS = 0 and Cause.IV = 1)	0x200
Interrupt (Status.BEV = 1 and Cause.IV = 1)	0x200
Interrupt (Cause.IV = 1 and Status.BEV = 0 and IntCtl.VS! = 0)	0X200 + (interrupt vector number × (IntCtl.VS || 0b00000))

6.1.3 General processing procedure of processor hardware response exception

When the processor starts to handle an exception, the EXL bit of the status register is set to 1, which means that the system is running in kernel mode. in After saving the appropriate on-site state, the exception handler usually sets the KSU field of the status register to kernel mode and sets the EXL bit Set back to 0. When the on-site state is restored and re-executed, the processing program will restore the KSU field back to the previous value and set the EXL bit at the same time Is 1.

Returning from an exception will also set the EXL position to 0.

6.1.4 Cold reset exception

When the system is first powered on or cold reset, a cold reset exception is generated. This exception cannot be masked.

Cold reset exceptions use special exception entry vector addresses. This address belongs to the CPU that does not require address mapping and does not access data through the cache Address space, so it is not necessary to initialize TLB or Cache to handle this exception. This also means that even if Cache and TLB are in an uncertain state,

The processor can also fetch and execute instructions.

When a cold reset exception occurs, the processor will perform a full set of reset initialization process, at this time the contents of all registers in the CPU are uncertain, Except for the following register fields:

- The Status register is set to its initial value, with the SR bit cleared and the ERL bit and BEV bit set.
- Config0 ~ Config6 registers are set to initial values.
- The Random register is initialized to the maximum value, and the Wired register is initialized to 0.
- The relevant fields of EntryHi, EntryLo0, EntryLo1, PageMask, PageGrain are set to initial values.
- The ErroEPC register is initialized to the value of PC.
- The Event bit of the Performance Count register is initialized to 0.
- All breakpoints and external interrupts are cleared.

6.1.5 Non-maskable interrupt

The non-maskable interrupt is triggered by the processor's independent NMI interrupt input signal. This exception cannot be masked.

It is impossible to mask the exception entry vector used by the interrupt in accordance with the cold reset. Therefore, when a non-maskable interrupt exception occurs, the Status.NMI bi Set to 1, the software can distinguish cold reset by this bit.

The non-maskable interrupt exception does not discard the state of any machine, but reserves the state of the processor for diagnosis. In particular, the Cause register The content remains unchanged, and the system jumps to the unmaskable exception entry vector out and starts executing the processing program.

Non-maskable interrupt exceptions only modify the following registers:

- Status.ERL is set to 1, Status.SR is set to 0, Status.NMI is set to 1, and Status.BEV is set to 1.
- The ErroEPC register is initialized to the value of PC.

6.1.6 Interrupt exception

When an unmasked interrupt arrives, an interrupt exception is triggered. A detailed description of inte 0 6 Jie .2.

Control register **Cause** of **ExcCode**

0x00 (Int) (see Table 7-28 on pa

Additional hardware status updates ons:

register	Status update description
Cause	The IP domain records pending interrupts.

6.1.7 Address error exception

The address error exception is triggered when the following conditions occur:

The double word load / store instruction, its access address is not aligned on the double word boundary.

For word load / store instructions, the access address is not aligned to the word boundary.

For halfword load / store instructions, the access addresses are not aligned with halfword boundaries.

Fetch means that the PC is not aligned to the word boundary.

Access the address segment of the core mode in the client mode or supervision mode.

Access the supervision mode address segment in the client mode.

When the 64-bit addressing enable is not enabled, the fetch instruction PC or load / store instruction access uses a 64-bit address, and the address falls in the 32-bit address space Compatible outside the scope.

Fetch instructions PC or load / store instructions use 64-bit addresses, and the addresses fall within the unimplemented range.

When in core mode, the accessed page table entry is valid and the K bit is 0.

This exception can be handled in both root mode and guest mode.

Control register **Cause** of **ExcCode** domain:

0x04 (AdEL): fetch or read data

AdES (0x05): write

(Please refer to Tabl

Additional hardware st e to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.

6.1.8 **TLB** refill exception

In the 32-bit host address space, and Root.Status.EXL = 0, the memory access uses the mapped address, which is touched when there is no match in the TLB Send TLB refill exception. Please note that this situation is different from finding a match in the TLB but the effective page table entry is 0, the latter corresponds to TLB invalid exception. In order to speed up the processing efficiency of the frequent and critical exception of TLB refill, TLB refill exception uses a separate exception entry bias The value is shifted, so the exception code entered in the Root.Cause.ExcCode field is not distinguished from the XTLB refill exception and TLB invalid exception.

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** domain:

0x02 (TLBL): fetch or read data

0x03 (TLBS): write

(Please refer to Tabl

Additional hardware st e to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.
XContext	?
	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception;
EntryHi	The R field records the [63..62] bits of the virtual address that triggered the exception.
	The ASID field records the ASID of the process to which the operation that triggered the exception belongs.
Diag	The MID field is set to 0.

6.1.9 **XTLB** refill exception

Under the 64-bit host address space, and Root.Status.EXL = 0, the mapped address is used for memory access. This address is touched when there is no match in the TLB. Issue XTLB refill exception. Please note that this situation is different from finding a match in the TLB but the effective bit of the matching page table entry is 0, which corresponds to the TLB invalid exception. In order to speed up the processing efficiency of this frequent and critical exception of XTLB refill, XTLB refill exception uses a separate exception The entry offset value, so the exception code filled in the Root.Cause.ExcCode field with the exception code and TLB refill exception, TLB invalid exception is not made Minute.

Loongson **3A3000 / 3B3000** Processor User Manual • Next

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** domain:

0x02 (TLBL): fetch or read data

0x03 (TLBS): write

(Please refer to Tabl

Additional hardware st e to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.
XContext	BadVPN2 domain records the [47..13] bits of the virtual address that triggered the exception;
	The R field records the [63..62] bits of the virtual address that triggered the exception.
	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception;

EntryHi	The R field records the [63..62] bits of the virtual address that triggered the exception. The ASID field records the ASID of the process to which the operation that triggered the exception belongs.
Diag	The MID field is set to 0.

6.1.10 TLB invalid exception

The TLB invalid exception is triggered when the following conditions occur:

Use the mapped address in the host address space. The address found a match in the TLB but the effective bit of the matching page table entry is 0. Touch Made the exception.

PageGrain.IEC = 0

PageGrain.RIE = 1, the load operation uses the mapped address under the host address space, and a matching and valid entry is found in the TLB, but The RI bit in the table entry is 1.

PageGrain.XIE = 1, which means that the mapped address is used under the host address space, and a matching and valid entry is found in the TLB, but the entry The XI bit in is 1.

The software needs to pay attention to the following situation: when Root.Status.EXL = 1, the mapped address used for fetching cannot find a match in the TLB , The exception entry offset used is the normal exception entry offset (0x180), and the exception code filled in the Root.Cause.ExcCode field is still TLBL (0x2) or TLBS (0x3). To distinguish this situation from normal TLB invalid exceptions, it can only be used by exception handlers The TLBP instruction distinguishes according to the search result.

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** domain:

0x02 (TLBL): fetch or read data

0x03 (TLBS): write

(Please refer to Tabl

Additional hardware st e to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.

register	Status update description
XContext	BadVPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception.
EntryHi	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception. The ASID field records the ASID of the process to which the operation that triggered the exception belongs.
Diag	The MID field is set to 0.

6.1.11 TLB modification exception

The store operation maps an address under the host address space. The address finds a matching and valid entry in the TLB, but the D bit of the page table entry 0 (meaning that the page is not writable), triggering TLB modification exception.

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** d

0x01 (Mod) (Please refer to Table

Additional hardware status updates in :

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.
XContext	BadVPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception.
EntryHi	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception. The ASID field records the ASID of the process to which the operation that triggered the exception belongs.
Diag	The MID field is set to 0.

6.1.12 TLB execution blocking exception

When Root.PageGrain.IEC = 0 and Root.PageGrain.XIE = 1, take the instruction to use the mapped address under the host address space, in the TLB A matching and valid entry was found, but the XI bit in the entry is 1.

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** dom

0x14 (TLBXI) (please refer to Table 7

Additional hardware status updates in res

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.
XContext	BadVPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception.
EntryHi	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception. The ASID field records the ASID of the process to which the operation that triggered the exception belongs.

76

Page 90

Loongson **3A3000 / 3B3000** Processor User Manual • Next

register	Status update description
Diag	The MID field is set to 0.

6.1.13 TLB read blocking exception

When Root.PageGrain.IEC = 0, and Root.PageGrain.RIE = 1, the load operation uses the mapped address in the host address space, in the TLB A matching and valid entry was found in the table, but the RI bit in the entry is 1.

This exception is only handled in root mode.

Control register **Cause** of **ExcCode** dom

0x13 (TLBRI) (Please refer to Table 7

Additional hardware status updates in res

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
Context	The BadVPN2 domain records the [31..13] bits of the virtual address that triggered the exception.
XContext	BadVPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception.
EntryHi	The VPN2 domain records the [47..13] bits of the virtual address that triggered the exception; The R field records the [63..62] bits of the virtual address that triggered the exception. The ASID field records the ASID of the process to which the operation that triggered the exception belongs.
Diag	The MID field is set to 0.

6.1.14 Cache error exception

When the instruction fetch or load / store operation is executed, it is found that a check error occurs in the tag or data of the cache, and this exception is triggered. The exception is not sc shield. Because the ı the exception is in the Cache, a special exception entry is used, located in the unmapped non-cached address segment. The exception entry Please refer to the d tion 6.1.2 on page 71 .

This exceptio ł in root mode.

Control register **Cause** of **ExcCode** domain:

no

The hardware status update process in response to an exception:

```
CacheErr ← ErrorState
Status.ERL ← 1
if InstructionInBranchDelaySlot then
    ErrorEPC ← PC of the branch / jump
else
    ErrorEPC ← PC of the instruction
endif
if Status.BEV = 1 then
```

```
        PC ← 0xFFFF.FFFF.BFC0.0200 + 0x100
    else
        PC ← 0xFFFF.FFFF || EBase 31..30 || 1 || EBase 28..12 || 0x100
    endif
```

77

Page 91

Loongson **3A3000** / **3B3000** Processor User Manual • Next

6.1.15 Integer overflow exception

When an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction is executed, resulting in an overflow of the complement of the result, the integer overflows exception.

This exception can be handled in root mode and guest mode.

Control register **Cause** of **ExcCode**

0x0c (Ov) (see Table 7-28 on pa

Additional hardware status updates ns:

no

6.1.16 Trap exception

When TGE, TGUE, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, TNEI instructions are executed, When the condition result is true, a trap exception is triggered.

This exception can be handled in root mode and guest mode.

Control register **Cause** of **ExcCod**

0x0d (Tr) (Please refer to Table

Additional hardware status update ons:

no

6.1.17 System call exception

When the SYSCALL instruction is executed, the system call exception is triggered.

This exception can be handled in root mode and guest mode.

Control register **Cause** of **ExcCode**

0x08 (Sys) (see Table 7-28 on pa

Additional hardware status updates i ns:

no

6.1.18 Breakpoint exception

When a BREAK instruction is executed, the breakpoint exception is triggered.

This exception can be handled in root mode and guest mode.

Control register **Cause** of **ExcCode**

0x09 (Bp) (see Table 7-28 on pa

Additional hardware status updates ns:

no

6.1.19 Reserved instruction exception

When an instruction not implemented by GS464E is executed, the exception of the reserved instruction is triggered.

This exception can be handled in root mode and guest mode.

78

Page 92

Control register **Cause** of **ExcCod**

0x0a (RI) (see Table 7-28 on pa

Additional hardware status updates ons:

no

6.1.20 Coprocessor unavailable exception

When the following conditions occur, the trigger coprocessor unavailable exception:

- When not in debug mode and core mode, and Status.CU0 = 0, execute COP0 type instructions (opcode = 0b010000), CACHE Class instructions (opcode = 0b101111), LWPTE, LWDIR, LDPTE, LDDIR.
- When Status.CU1 = 0, execute COP1 instruction (opcode = 0b010001), COP1X instruction (opcode = 0b010011), LWC1 SWC1, LDC1, SDC1, MOVF, MOVT, 64-bit multimedia instructions (opcode = 0b010010, func = 0b000000 ~ 0b000011, rs = 11000 ~ 11111; opcode = 0b010010, func = 0b001000 ~ 0b001110, rs = 11000 ~ 11101), gsLWLC1, gsLWRC1, gsLDLC1, gsLDRC1, gsLWLEC1, gsLWGTC1, gsLDLEC1, gsLDGTC1, gsLQC1 gsLWXC1, gsLDXC1, gsSWLC1, gsSWRC1, gsSDLC1, gsSDRC1, gsSWLEC1, gsSWGTC1 gsSDLEC1, gsSDGTC1, gsSQC1, gsSWXC1, gsSDXC1.
- When Status.CU2 = 0, execute COP2 type instruction (opcode = 0b010010), LWC2 type instruction (opcode = 0b110010), SWC2 Type instruction (opcode = 0b111010), LDC2 type instruction (opcode = 0b110110), SDC2 type instruction (opcode = 0b111110), but Does not include SETMEM, gsLBLE, gsLBGT, gsLHLE, gsLHGT, gsLWLE, gsLWGT, gsLDLE, gsLDGT, gsLQ, gsLBX, gsLHX, gsLWX, gsLDX, gsSBLE, gsSBGT, gsSHLE, gsSHGT, gsSWLE, gsSWGT, gsSDLE, gsSDGT, gsSQ, gsSBX, gsSHX, gsSWX, gsSDX, LWPTE, LWDIR, LDPTE, LDDIR, 64-bit multimedia instructions (opcode = 0b010010, func = 0b000000 ~ 0b000011, rs = 11000 ~ 11111; opcode = 0b010010, func = 0b001000 ~ 0b001110, rs = 11000 ~ 11101), gsLWLC1, gsLWRC1, gsLDLC1, gsLDRC1, gsLWLEC1 gsLWGTC1, gsLDLEC1, gsLDGTC1, gsLQC1, gsLWXC1, gsLDXC1, gsSWLC1, gsSWRC1 gsSDLC1, gsSDRC1, gsSWLEC1, gsSWGTC1, gsSDLEC1, gsSDGTC1, gsSQC1, gsSWXC1 gsSDXC1.

It should be noted that in the guest mode, when Guest.Status.CU1 / 2 = 1 but Root.Status.CU1 / 2 = 0, the coprocessor is not triggered Exceptions can be directly dealt with in root mode.

Control register **Cause** of **ExcCode** d

0x0b (CpU) (see Table 7-28 on pag

Additional hardware status updates in :

register	Status update description
Cause	The CE domain records the unavailable coprocessor number.

6.1.21 Floating point exception

The floating-point coprocessor triggers a floating-point floating-point exception. For m g-point exceptions, see 16 2 Jie .2.4. Some floating point exceptions It can be masked by configuring the Enable field of the FCSR register. For details, see sectio

Control register **Cause** of **ExcCode** (

0x0f (FPE) (see Table 7-28 on pa

Additional hardware status updates i is:

register	Status update description
FCSR	The Cause and Flag fields record specific floating-point exception type information.

6.1.22 Exception for floating point stack

When the SETTAG instruction is executed, if the content in the source operand does not meet the specified conditions, the floating point stack exception is triggered.

This exception can be handled in root mode and guest mode.

Control register **Cause** of **ExcCode** dom

0x10 (GSExc) (Please refer to Table 7

The **GSExcCode** field of the contr

0x00 (IS) (see Table 7-43 on pa

6.2 Interrupt

The scope of interrupts described in this section includes hardware interrupts, software interrupts, timer interrupts and performance counter overflow interrupts. "Not screenable "Negative interrupt (NMI)" although it contains the word "interrupt" in the name, it is neither controlled nor affected by the interrupt system described in this section Interrupt the system, so treat it as a special exception-non-maskable interrupt exception.

6.2.1 Necessary conditions for interrupt response

The necessary conditions for the processor to respond to the interrupt are:

- Status.IE = 1, indicating that global interrupt enable is enabled.
- Debug.DM = 0 means it is not in debug mode.
- Staus.ERL = 0 and Status.EXL = 0, indicating that neither errors nor exceptions are being processed.
- An interrupt source generates an interrupt and the interrupt source is not masked,

6.2.2 Interrupt mode

The content described in this section applies to both root mode and guest mode. When referring to the virtual machine environment in guest mode, the “hard "Hardware" in the concept of "software interrupt" does not necessarily refer to physical hardware, but just follows the consistent naming style in the MIPS specification.

GS464E supports two interrupt modes:

Mode one, compatible interrupt mode

In this mode, the processor supports 2 software interrupts (SW0 ~ SW1), 6 hardware interrupts (HW0 ~ HW5), 1 timer interrupt and 1 A performance counter overflow interrupt. Among them timer interrupt and performance counter interrupt multiplex HW5 hardware interrupt.

The interrupt source of software interrupt is Cause.IP [1: 0] two bits. You can only trigger the Cause.IP [1: 0] bit by software to write 1 and the software to write 0 To clear.

The interrupt source of the timer interrupt is recorded in the Cause.TI bit. When the Count [31: 0] is equal to Compare [31: 0], it is set to 1 by the hardware. The software can indirectly clear the interrupt recorded by the Cause.TI bit by writing the Compare register.

Performance counter overflow interrupt in the interrupt source Cause.PCI recording bit, when the value of the performance counter overflow (47 Counter [1] bit 1) Set by hardware, the software can indirectly clear the Cause.PCI bit by writing 0 to the 47th bit of the related performance count register.

[1] The actual effective counting digits of the performance counter in GS464E is 48 digits.

The interrupt source of the hardware interrupt comes from outside the processor. The hardware samples the 6 interrupt input pins on the processor interface step by step. Reversely traverse the interrupt routing path of the system to clear the interrupt status on the terminal device or routing path to clear the hardware interrupt of the processor.

In addition to the global interrupt enable, each interrupt source contains an interrupt mask bit. The generation relationship of each interrupt request is shown in Table 6-4 .

Table **6-4** Interrupt request generation in compatible interrupt mode

Type of interrupt	Interrupt source	Interrupt request generation
Hardware interrupt, timer interrupt or performance counter overflow interrupt	HW5	Cause.IP7 & Status.IM7
Hardware interrupt	HW4	Cause.IP6 & Status.IM6
	HW3	Cause.IP5 & Status.IM5
	HW2	Cause.IP4 & Status.IM4
	HW1	Cause.IP3 & Status.IM3
	HW0	Cause.IP2 & Status.IM2
Software interrupt	SW1	Cause.IP1 & Status.IM1
	SW0	Cause.IP0 & Status.IM0

All interrupts use the same exception entry offs ıe general exception entry offset (0x180) or the special exception entry offset (0x200) Decided by Cause.IV. For details, see Table 6-3 on p

The interrupt exception handler needs to query .IM to determine the specific interrupt source. For multiple simultaneous interrupt sources In case, the software can realize the priority of interrupt processing by regulating the query order.

Mode two, vector interrupt mode

This mode specifies a unique exception entry vector for each interrupt based on the compatible interrupt mode (see Table ıe calculation method), And a fixed priority relationship is defined for all interrupts, as shown in Table 6-5 .

Table **6-5** Priority relationship between interrupts in vector interrupt mode

priority	Type of interrupt	Interrupt source	Interrupt request generation	Interrupt vector number
Highest priority	Hardware interrupt			

	HW5	Cause.IP7 & Status.IM7	7
	HW4	Cause.IP6 & Status.IM6	6
	HW3	Cause.IP5 & Status.IM5	5
	HW2	Cause.IP4 & Status.IM4	4
	HW1	Cause.IP3 & Status.IM3	3
	HW0	Cause.IP2 & Status.IM2	2
Software interrupt	SW1	Cause.IP1 & Status.IM1	1
Lowest priority	SW0	Cause.IP0 & Status.IM0	0

In vectored interrupt mode, which hardware interrupt source multiplexed by the timer interrupt is defined by the IntC to Table 7-25 on page 109 .

In vectored interrupt mode, the performance counter overflow interrupt multiplex which hardware interrupt source is CI field, please refer to the table on page 1 7-25 .

GS464E does not implement shadow registers, so all interrupts in vector interrupt mode correspond to the same set of logical general registers (GPR).

The interrupt mode currently used by the processor is determined by the three fields Status.BEV, Cause.IV, and IntCtl.VS. The corresponding relationship is shown in Table 6-6 As shown.

Table **6-6** Interrupt mode judgment

Status.BEV	**Cause.IV**	**IntCtl.VS**	Interrupt mode

Status.BEV	**Cause.IV**	**IntCtl.VS**	Interrupt mode
1	x 1	x	Compatible with interrupt mode
x	0	x	Compatible with interrupt mode
x	x	= 0	Compatible with interrupt mode
0	1	! = 0	Vector interrupt mode

6.2.3 Supplementary explanation of interrupt handling

This manual only describes the structure and response mechanism of the GS464E part of the interrupt system. Software staff designing Loongson 3A3000 chip When shutting down the system, please refer to Chapters 6 and 7 of the "Loongson 3A3000 / 3B3000 Processor User Manual-Volume 1".

The processor only directly samples and records the externally input high-level hardware interrupt request, and is not responsible for level conversion or pulse control The burst interrupt signal is expanded to a level signal. This part of the work is completed by the interrupt controller in the chip, please refer to "Godson 3A3000 / 3B3000 Chapters 6 and 7 of the User Manual for Managers-Volume One.

Depending on the signal behavior of the external hardware interrupt input, the processor is directly connected to the interrupt request bit of the external interrupt input (Root.Cause.IP [7: Or Guest.Cause.IP [7: 2]) It is possible to change from 1 to 0 after the interrupt request is triggered and before the interrupt handler queries these request bits. Break The management program needs to be able to handle this situation. It is recommended to return without doing anything. If special treatment is done for system diagnosis, please do not affect The normal behavior of the system.

For hardware interrupts directly affected by external hardware, software usually needs to clear the interrupt status of a certain terminal device. Clear from the processor In addition to the interrupted command sequence, when the device receives the command to clear its own interrupt status, and then the level of the processor interrupt input pin changes from 1 (Interrupt input cancellation) The process may have an uncertain delay. If the interrupt enable is turned on prematurely, the same interrupt may be sampled again, This is the so-called "false interruption" phenomenon. The software needs to be able to handle this situation correctly. It is recommended that the software issue a write command to clear the d After that, explicitly read the interrupt status flags of related devices 2Until the read status flag has been cleared, then turn on the processor completely Or interrupt the routing channel corresponding to the device interrupt enable. Loongson 3A3000 chip has ensured that the internal interrupt source signals are transmitted through the interrupt The delay delivered to the interrupt input pin of the processor is always less than the delay of the interrupt state returning to the processor through the data access path. If debugging The "false interruption" phenomenon is still found, please ask the software staff to further inquire the data manual and user manual of the device chips and bridges involved in the system.

1 x means it can be any value.
2 If the interrupt flag bit of the related device is of "read clear" attribute, the software needs to be cautious when issuing the query command.

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7 Coprocessor **0** register

7.1 Overview of root coprocessor **0** registers

The coprocessor 0 register in the root mode context of the GS464E processor core is called "root coprocessor 0 register". The registers are listed in Table 7-1 .

Table **7-1** List of root coprocessor **0** registers

Reg.	Sel.	Register name	Function definition	in
0	0	Index	VTLB and FTLB access the specified index register	Page 85, S
1	0	Random	VTLB and FTLB access random index register	Page 86 S
2	0	EntryLo0	VTLB and FTLB entries in the low-order content related to even virtual pages	Page 87, S
3	0	EntryLo1	VTLB and FTLB entries in the low-order content related to odd virtual pages	Page 87, S
4	0	Context	Pointer to page table entry in memory	Page 90, S
4	2	UserLocal	Store user information, allow user mode software to read through RDHWR instruction	Page 91 S
5	0	PageMask	VTLB page table size control	Page 92, S
5	1	PageGrain	1KB small page and other page table attribute control	Page 93, S
5	5	PWBase	Page table base address register	Page 94, S
5	6	PWField	Configure page table address index positions at all levels	Page 95, S
5	7	PWSize	Configure page table pointer size at all levels	Page 96 S
6	0	Wired	Control the number of fixed items in VTLB	Page 97 S
6	6	PWCtl	Control multi-level page table configuration	Page 98 S
7	0	HWRENa	RDHWR instruction can access register enable control	Page 99 S
8	0	BadVAddr	Record the wrong address of the latest address related exception	Page 100
9	0	Count	Processor clock counter	Page 101 S
9	6	GSEBase	Godson extended exception entry base address register	Page 102 S
9	7	PGD	Page table pointer register	Page 103 S
10	0	EntryHi	High-level content of VTLB and FTLB entries	Page 104 S
11	0	Compare	Timer interrupt control	Page 106 S
12	0	Status	Processor status and control register	Page 107 S
12	1	IntCtl	Interrupt system status and control register	Page 109, S
12	2	SRSCtl	Shadow register status and control register	Page 110 S
13	0	Cause	Store the reason for the last exception	Page 111 S
14	0	EPC	PC where the last exception occurred	Page 113 S
15	0	PRId	Processor ID	Page 114 S
15	1	EBase	Exception entry base address register	Page 115 S
16	0	Config	Configuration register	Page 116 S
16	1	Config1	Configuration register 1	Page 117, S
16	2	Config2	Configuration register 2	Page 118 S

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Reg.	Sel.	Register name	Function definition	inc
16	3	Config3	Configuration register 3	Page 119,
16	4	Config4	Configuration register 4	Page 121,
16	5	Config5	Configuration register 5	Page 123,
16	6	GSConfig	Godson extended configuration register	Page 124 S
17	0	LLAddr	Store Load-Link instruction access address	Page 127,
20	0	XContext	Page table pointer in extended address mode	Page 128 S
twenty two	0	Diag	Godson Extended Diagnostic Control Register	Page 129,
twenty two	1	GSCause	Store the supplementary information of the last Godson expansion exception	Page 131,
twenty three	0	Debug	EJTAG Debug register	Page 133,
twenty four	0	DEPC	The PC that stored the last EJTAG debugging exception	Page 134,
25	0-7	PerfCnt0-PerfCnt7	Access interface for processor core internal performance counter	Page 135,
26	0	ErrCtl	Cache Parity / ECC check value register	Page 137,
27	0	CacheErr	Cache Parity / ECC check status and control register	Page 138,
27	1	CacheErr1	Cache Parity / ECC check status and control register 1	Page 140 S
28	0	TagLo	Cache Tag access interface low part	Page 141,
28	1	DataLo	Cache Data access interface low part	Page 144,
29	0	TagHi	Cache Tag access interface high part	Page 145,
29	1	DataHi	Cache Data access interface high part	Page 146,
30	0	ErrorEPC	The PC that stored the last error instruction	Page 147,
31	0	DESAVE	EJTAG debug exception save register	Page 148,
31	2-7	KScratch1-KScratch6	Core state accessible notes register 1 ~ 6	Page 149,

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.2 **Index** register **(CP0 Register 0, Select 0)**

The Index register is a 32-bit readable and writable register, where the index information stored is used for TLBP, TLBR, TLBWI instruction access Used in TLB.

Figure 7-1 illustrates the format of the Index re ribes the fields of the Index register.

Figure **7-1 Index** register format

Table **7-2 Index** register field description

Domain name	Bit	Functional description	Read / write	reset value
P	31	TLB query failure flag. When the TLBP instruction fails to find a match in the TLB, the position is 1; Otherwise set to 0.	R	0x0
0	30..11	The read-only constant is 0.	0	0
Index	10..0	TLB access index. The software configures this field to instruct subsequent TLBR or TLBWI instructions to read or Write the specified item of TLB. When the TLBP instruction is executed, if a match is found, the index value of the match will be stored in this field; When no match is found, the contents of the Index field of the Index register can be any value. Index value 0..63 is used to indicate VTLB item 0..63. The Index value of 64..1087 is used to indicate FTLB. Where ((Index-64) div 128) is used to indicate access to FTLB Which way to go and which item to visit in this way is determined by the value of ((Index-64) mod 128). For example, when Index When the value is 798, it indicates access to the ((798-64) mod 128 = 94) item of the ((798-64) div 128 = 5) way.	R / W	0x0

Programming tips:

The reasonable value range of the Index field is 0 ~ 1087. When the value written to the Index field exceeds this range, the processor result will be undefined.

85

Page 99

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.3 **Random** register **(CP0 Register 1, Select 0)**

The Random register is a read-only register used to store the index value of the TLBWR instruction to access the TLB. Random register
The put index value changes every clock cycle, the upper bound (inclusive) of the value change is 63, and the lower bound (inclusive) of the change is set in the Wired register value. The Random register will automatically reset to the upper bound, 63, when a Reset exception occurs and the Wired register is written.

When the value of the page size indicated in the PageMask register is inconsistent with the page size configured in the FTLB, the TLBWR instruction will only operate As a VTLB, the value in the current Random register determines which item is written to VTLB.

When the value of the page size indicated in the PageMask register is the same as the page size configured in the FTLB, the TLBWR instruction will only operate FTLB, which routing processor writes to FTLB is randoı ally, and does not use the contents of the Random register.

Figure 7-2 illustrates the format of the Random reg ibes the fields of the Random register.

Figure **7-2 Random** register format

Table **7-3 Random** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31..6	Read-only constant is 0.	0	0
Random	5..0	Random index value written by VTLB.	R	0x3f

86

Page 100

7.4 **EntryLo0** and **EntryLo1** registers **(CP0 Register 2 and 3, Select 0)**

The EntryLo0 and EntryLo1 registers are used as the interface for TLBP, TLBR, TLBWI and TLBWR instructions to access the TLB, among which EntryLo0 The register stores even page information EntryLo1 register stores odd page information.

The format of the EntryLo0 and EntryLo1 registers when accessed using the two sets of instructions DMFC0 / DMTC0 and MFC0 / MTC0 There are differences.

Figure 7-3 illustrates the format of the EntryLo0 and EntryLo1 registers when accessed by the DMFC0 / DM e 7-4 shows this case The fields of the following registers are described.

FIGS **7-3 EntryLo0** and **EntryLo1** in **DMFC0 / DMTC0** instruction format accessible register

Table **7-4 EntryLo0** and **EntryLo1** in **DMFC0 / DMTC0** Register Field Descriptions access instruction

Domain name	Bit	Functional description	Read / write	reset value
RI	63	Read block flag. When the RI bit of a TLB entry is 1, when an access instruction attempts to proceed on this page When reading, the processor will trigger an exception. According to the IEC field of the PageGrain register, the triggered exception can be So it is TLBL invalid exception or TLBRI exception. The XI fields of EntryLo0 and EntryLo1 can only be written when PageGrain RIE = 1. PageGrain RIE = 0 At this time, regardless of the value to be written, the RI fields of EntryLo0 and EntryLo1 will be read as 0.	R / W	0x0
XI	62	Execute block flag. When the XI position of a TLB entry is 1, the instruction fetch occurs on this page, the processor An exception will be triggered. According to the IEC field of the PageGrain register, the exception triggered can be TLBL invalid The exception is TLBXI. The XI fields of EntryLo0 and EntryLo1 can only be written when PageGrain XIE = 1. PageGrain XIE = 0 , The XI field of EntryLo0 and EntryLo1 will be read as 0 regardless of the value to be written.	R / W	0x0
K	61	Kernel execution protection bit. When the processor is in the core state, fetch the instruction on the K = 0 page, the processor will trigger the TLB The exception is Invalid. The K fields of EntryLo0 and EntryLo1 can only be written when GSConfig.KE = 1. GSConfig.KE = 0 At this time, regardless of the value to be written, the K fields of EntryLo0 and EntryLo1 will be read as 0.	R / W	0x0
0	60..42	Read-only constant is 0.	0	0
		Physical page number extension. When the processor is configured to support large physical address space mode (Config3 LPA = 1 and		

PFNX	41..30	PageGrain ELPA = 1), the content of this field is spliced into the high bits of the PFN field to form a complete physical page number, thus supporting 48-bit physical address space. The PFNX domain corresponds to 47..36 bits of the physical address. If the processor is configured not to support the large physical address space mode (PageGrain ELPA = 0 [1]), the PFNX domain will Unable to write and read returns 0. In order to realize the combination of the system software written based on the MIPS specification Release 1 Content.	R / W	0x0

1 The Config LPA in the Loongson 3A1500 chip is always 1, so it is impossible to turn off the large physical address space mode support because Config LPA = 0.

Domain name	Bit	Functional description	Read / write	reset value
PFN	29..6	The basic part of the physical page number. When the processor is configured to support large physical address space mode (Config3 LPA = 1 and PageGrain ELPA = 1), the content of this field is spliced into the lower bits of the PFNX field to form a complete physical page number, thereby supporting Supports 48-bit physical address space. The PFN field corresponds to the 35..12 bits of the physical address. When the processor is configured not to support the large physical address space mode (PageGrain ELPA = 0), the PFN domain itself The final physical page number will be formed to support the 36-bit physical address space.	R / W	0x0
C	5..3	The Cache attribute of the physical page. For the specific definition and coding of the Cache attribute, please refer to the following section Table 7-6 .	R / W	0x0
D	2	"Dirty" bit. When the dirty position in the page table is 1, it means that the page can be written; otherwise it is 0 for a dirty position The write operation of the page will trigger TLB Mod exception.	R / W	0x0
V	1	Valid bit. When the valid position in the page table is 1, it means that the page can be accessed; otherwise, a valid position is accessed A page of 0 will trigger a TLB Invalid exception.	R / W	0x0
G	0	Global bit. When the page table entry is filled in the TLB, the two G-bit values of the EntryLo0 and EntryLo1 registers are used for logic Compiled and the result is used as the global identification bit of this page entry. When the global flag in the page table is 1, the TLB ground The address matching will not compare ASID when searching. When reading the page table entry from the TLB, the two G bits of the EntryLo0 and EntryLo1 registers simultaneously reflect the read G-bit information of page table entry.	R / W	0x0

Figure 7-4 illustrates the format of the EntryLo0 and EntryLo1 registers when accessed by the MFC0 / ıble 7-5 is sent in this case
Each field of the memory is described. If you need to pay attention, you cannot access the KE bit of the page ; KE bit of the TLB entry will be written to the default value

FIGS **7-4 EntryLo0** and **EntryLo1** in **MFC0 / MTC0** register access instruction format

Table **7-5 EntryLo0** and **EntryLo1** in **MFC0 / MTC0** Register Field Descriptions access instruction

Domain name	Bit	Functional description	Read / write	reset value
SignExt	63..32	When using MTC0 to write, the content of this part in the register is ignored, and the PFNX field is written with 0. When MFC0 is used for reading, the part returned to the general register is obtained by sign extension of the RI bit.	R	0x0
RI	31	Read block flag. When the RI bit of a TLB entry is 1, when an access instruction attempts to proceed on this page When reading, the processor will trigger an exception. According to the IEC field of the PageGrain register, the triggered exception can be So it is TLBL invalid exception or TLBRI exception. The XI fields of EntryLo0 and EntryLo1 can only be written when PageGrain RIE = 1. PageGrain RIE = 0 At this time, regardless of the value to be written, the RI fields of EntryLo0 and EntryLo1 will be read as 0.	R / W	0x0

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Domain name	Bit	Functional description	Read / write	reset value
XI	30	Execute block flag. When the XI position of a TLB entry is 1, the instruction fetch occurs on this page, the processor An exception will be triggered. According to the IEC field of the PageGrain register, the exception triggered can be TLBL invalid The exception is TLBXI. The XI fields of EntryLo0 and EntryLo1 can only be written when PageGrain XIE = 1. PageGrain XIE = 0 , The XI field of EntryLo0 and EntryLo1 will be read as 0 regardless of the value to be written.	R / W	0x0
PFN	29..6	The basic part of the physical page number. When the processor is configured to support large physical address space mode (Config3 LPA = 1 and PageGrain ELPA = 1), the content of this field is spliced into the lower bits of the PFNX field to form a complete physical page number, thereby supporting Supports 48-bit physical address space. The PFN field corresponds to the 35..12 bits of the physical address. When the processor is configured not to support the large physical address space mode (PageGrain ELPA = 0), the PFN domain itself The final physical page number will be formed to support the 36-bit physical address space.	R / W	0x0
C	5..3	The Cache attribute of the physical page. For the specific definition and coding of the Cache attribute, please refer to the following section Table 7-6 .	R / W	0x0
D	2	"Dirty" bit. When the dirty position in the page table is 1, it means that the page can be written; otherwise it is 0 for a dirty position The write operation of the page will trigger TLB Mod exception.	R / W	0x0
V	1	Valid bit. When the valid position in the page table is 1, it means that the page can be accessed; otherwise, a valid position is accessed A page of 0 will trigger a TLB Invalid exception.	R / W	0x0
G	0	Global bit. When the page table entry is filled in the TLB, the two G-bit values of the EntryLo0 and EntryLo1 registers are used for logic Compiled and the result is used as the global identification bit of this page entry. When the global flag in the page table is 1, the TLB ground The address matching will not compare ASID when searching. When reading the page table entry from the TLB, the two G bits of the EntryLo0 and EntryLo1 registers simultaneously reflect the read G-bit information of page table entry.	R / W	0x0

Programming tips:

Before any field content of the PageGrain register is modified, the PFNX and PFN fields in the EntryLo0 and EntryLo1 registers must be changed Write 0, and clear all TLB. And all the above operations must be performed in the unmapped address space. If not Follow the requirements here and the processor behavior will be undefined.

Table **7-6 Cache** attribute coding table

Page table **C** field encoding	**Cache** attribute [1]
0	Keep, forced configuration will cause a crash
1	Keep, forced configuration will cause a crash
2	Uncached
3	Cached
4	Reserved, forced configuration is equivalent to Cached
5	Reserved, forced configuration is equivalent to Cached
6	Reserved, forced configuration will access the EJTAG dseg space, there is a risk of crash
7	Uncached Accelerated

[1] For the definition of Uncached, Cacheable Coherent and Uncached Accelerated attributes, pleas ction 5.2 on page 60 .
89

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.5 **Context** register **(CP0 Register 4, Select 0)**

The Context register is a readable and writable register, which contains some high-level information and sending of the base address of the page table filled in by the operating system so Some bits of the virtual address of the error that caused the TLB exception. According to the original design intent of the MIPS architecture, the letters that are stitched together in the Context The information can form a pointer to an item in the page table and is used to access the page table item when a TLB exception occurs. Do not do anything in the Context register The page table that can be accessed in a reasonable time is a single-level page table structure, the page size is 4K bytes, each page table entry is 16 bytes, including a continuous one of virtual One even page table entry and one odd page table entry, a total of 512K parity page table entries. When the page table does not use this structure, the software needs to The contents of the register are appropriately shifted and spliced. For an operating system that uses a multi-level page table, the Context register can only be used

Accelerate address generation for page table access at the last level.

The Context register is mainly used in TLB Refill exception handlers. But when XTLB Refill, TLB Invalid and TLB Mod When an exception occurs, the BadVPN2 field in the Context register is also updated, so the software can also be used in the corresponding exception handler Context register.

The BadVPN2 field in the Context register copies part of the information in the BadVAddr register, but this does not mean that this part is completely equal Price. When an Address Error exception occurs, the BadVaddr register will be updated by hardware, but the BadVPN2 field of the Context register will not Updated by hardware.

Figure 7-5 illustrates the format of the Context reg ribes the fields of the Context register.

Figure **7-5 Context** register format

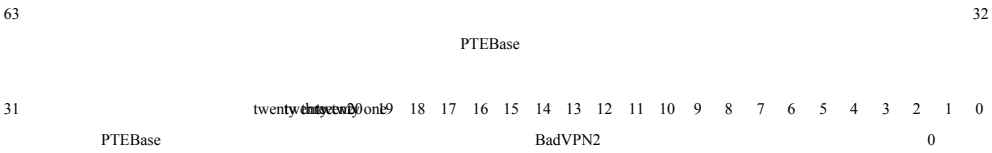


Table **7-7 Context** register field description

Domain name	Bit	Functional description	Read / write	reset value
PTEBase	63..23	page table base address high. It is configured by the operating system software according to the current page table	R / W	no
BadVPN2	22..4	When a TLB exception occurs, the 31..13 bits of the erroneous virtual address are stored.	R	no
0	3..0	Read-only constant is 0.	0	0

90

Page 104

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.6 UserLocal Register (CP0 Register 4, Selelct 2)

The UserLocal register is a readable and writable register. It is not used to control the operation of the processor hardware, nor will it be changed by the hardware. The content of UserLocal can be read in the user mode by t on. Whether it can be read is controlled by bit 29 of the HWREna register.

Figure 7-6 illustrates the format of the UserLocal reg ibes the fields of the UserLocal register.

Figure **7-6 UserLocal** register format

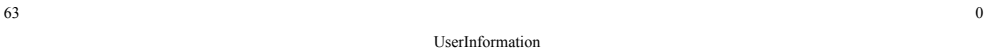


Table **7-8 UserLocal** register domain description

Domain name	Bit	Functional description	Read / write	reset value
UserInfor -mation	63..0	The stored information is not affected by the processor hardware, nor does it affect the processor hardware operation.	R / W	no

91

7.7 **PageMask** Register **(CP0 Register 5, Select 0)**

The PageMask register is a readable and writable register that is used during the reading and writing of the TLB; it contains a comparison mask for each TLB Table entries set different page sizes.

Figure 7-7 illustrates the format of the PageMask reg ibes the fields of the PageMask register.

Figure **7-7 PageMask** register format

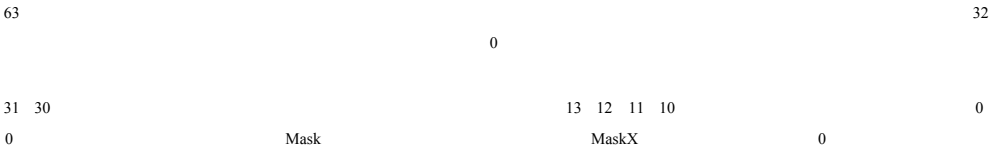


Table **7-9 PageMask** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
0	63..31	The read-only constant is 0.	0	0
Mask	30..13	When performing virtual-to-real address translation, each bit in the Mask field [17: 0] is used to indicate the virtual address [30:13] bits Whether the corresponding bit is compared. 1: No comparison; 0: Comparison. Please refer to Table 7-10 below for the Mask codes supported by Loongson and their corresponding page sizes .	R / W	0x3
MaskX	12..11	Constant is 3, 1KB pages are not supported.	R	0x3
0	10..0	Read-only constant is 0.	0	0

Table 7-10 shows the Mask field codes supported by GS464E and their corresponding page sizes.

Table **7-10 Mask** field encoding and page size

Mask encoding	Page size
0x0	4KB
0x3	16KB
0xF	64KB
0x3F	256KB
0xFF	1MB
0x3FF	4MB

0xFFF 0x3FFF	16MB 64MB
0xFFFF	256MB
0x3FFFF	1GB

Programming tips:

Although GS464E allows to fill in the PageMask register: 0x1, 0x7, 0x1F, 0x7F, 0x1FF, 0x7FF, 0x1FFF, 0x7FFF, 0x1FFFF, the page size is 2 2n + 1 KB (n = 0..8), but the processor does not guarantee the correctness of the program running in this configuration.

7.8 **PageGrain** Register **(CP0 Register 5, Select 1)**

The PageGrain register is a readable and writable register. GS464E only realizes the control mode with TLB XI / RI protection bit and large physical address System-related parts.

Figure 7-8 illustrates the format of the PageGrain reg es the fields of the PageGrain register.

Figure **7-8 PageGrain** register format

31	30	29	28	27	26	0
RIE	XIE	ELPA	IEC		0	

Table **7-11 PageGrain** register field description

Domain name	Bit	Functional description	Read / write	reset value
RIE	31	TLB Page Table Read-Inhibit function enable bit. 0: Disable this function. The RI bits of the EntryLo0 and EntryLo1 registers will be forbidden to write and forced to 0; 1: Turn on this function. The RI bits of the EntryLo0 and EntryLo1 registers can be used normally.	R / W	0x0
XIE	30	TLB page table execution inhibit (Execute-Inhibit) function enable bit. 0: Disable this function. The XI bit of EntryLo0 and EntryLo1 registers will be forbidden to write and forced to 0; 1: Turn on this function. The XI bits of the EntryLo0 and EntryLo1 registers can be used normally.	R / W	0x0
ELPA	29	Big physical address function enable bit. 0: Disable this function. The PFNX fields of the EntryLo0 and EntryLo1 registers will be forbidden to write and forced to 0; 1: Turn on this function. The PFNX fields of the EntryLo0 and EntryLo1 registers can be used normally.	R / W	0x0
0	28	Read-only constant is 0.	0	0
IEC	27	TLB read block and execute block exception codes and entry control bits. 0: TLB read blocking and execution blocking exception coding and entry multiplexing TLBL exception coding and entry; 1: TLB read block exceptions use TLBRI exception codes and entries, TLB execute block exceptions use TLBXI exceptions Coding and entrance.	R / W	0x0
0	26..0	Read-only constant is 0.	0	0

Programming tips:

Before the software attempts to modify any fields of PageGrain, all TLBs must be cleared, and the fields of the COP0 register listed below must be cleared. Set to the specified value, otherwise the processor's behavior will be undefined.

COP0 register field	Specified value
EntryLo0PFN, EntryLo1PFN	0
EntryLo0PFNX, EntryLo1PFNX	0

Page 107

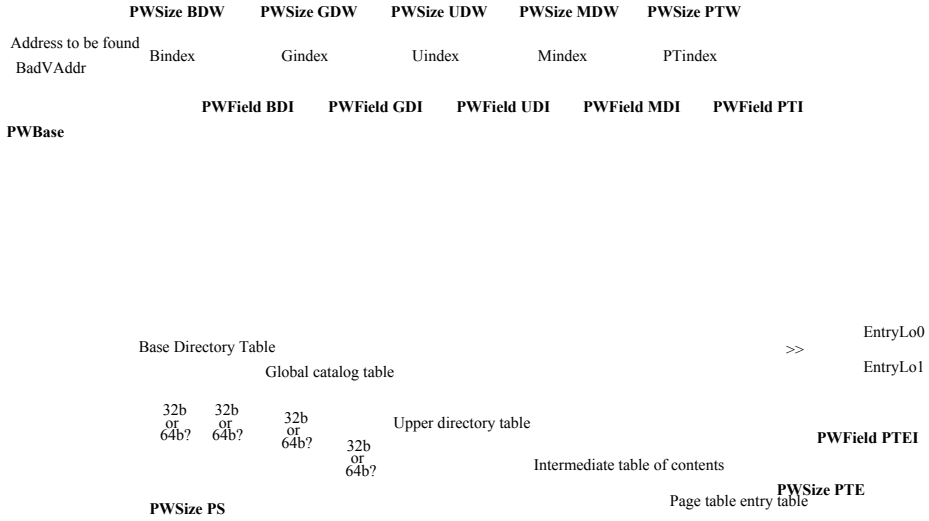
Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.9 **PWBase** Register **(CP0 Register 5, Select 5)**

The PWBase register is a 64-bit readable and writable register that stores the virtual address of the base address of the page table. This register and PWField, PWSize Used in conjunction with the PWCtl register and used in GS464E to provide configuration information for the execution of LDDIR and LDPTE instructions.

LDDIR and LDPTE instructions support multi-level page table structure traversal search, the page table can contain up to four levels of directory table and a page table entry table, See Figure 7-9 for a schematic of the supported page table structure .

Figure **7-9** Page table access process supported by **PWBase, PWField, PWSize** and **PWCtl**



GS464E uses MID to construct four 64-bit address spaces isolated from each other, in order to use LDDIR and
The LDPTE instruction accelerates page table traversal search, and the PWBase, PWField, PWSize, and PWCtl registers are implemented as hardware independent
Two groups. Which group the software and hardware actually operate is controlled by the MID field of the Diag register. When Diag.MID = 0, operate the first group; when
When Diag.MID! = 0, operate the second group.

Figure 7-10 illustrates the format of the PWBase reg es the fields of the PWBase register.

Figure **7-10 PWBase** register format

Table **7-12 PWBase** register field description

Domain name	Bit	Functional description	Read / write	reset value
PWBase	63..0	Page table base address.	R / W	0x0

Page 108

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.10 **PWField** register **(CP0 Register 5, Select 6)**

The PWField register is used together with the PWBase, PWSize, and PWCtl registers. It is used in the GS464E for LDDIR and
The execution of the LDPTE instruction provides relevant configuration information. The LDDIR and LDI port multi-level page table structure traversal search, the page ta
Contains up to four levels of directory tables and one level of page table entry tables. For the supported pag l access process, see Figure 7-9 on page 94 . Stored in page table
The index value pointing to the next-level page table or the final page table entry is obtained by interceptin bits from the virtual address (BadVAddr) to be searched.

The PWField register is used to identify the starting position of the page table index at all levels intercepted in the virtual address to be searched (BadVAddr).

GS464E uses MID to construct four 64-bit address spaces isolated from each other, in order to use LDDIR and

The LDPTE instruction accelerates page table traversal search, and the PWBase, PWField, PWSize, and PWCtl registers are implemented as hardware independent

Two groups. Which group the software and hardware actually operate is controlled by the MID field of the Diag register. When Diag.MID = 0, operate the first group; when

When Diag.MID! = 0, operate the second group.

Figure 7-11 illustrates the format of the PWField reg es the fields of the PWField register.

Figure **7-11 PWField** register format

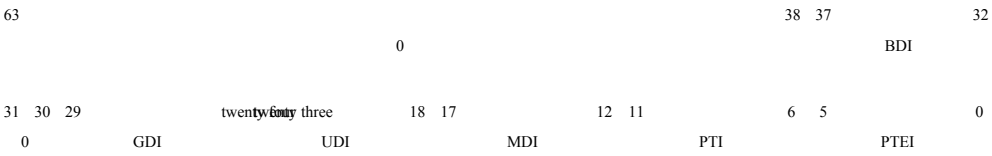


Table **7-13 PWField** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
0	63..38	The read-only constant is 0.	0	0
BDI	37..32	The starting position of the base directory index (Base Directory) index. Whether the base directory table is used is controlled by PWCtl PWDirExt . The base directory table can be used to distinguish different page tables, For example, user page tables and kernel page tables can be maintained separately.	R / W	0x0
0	31..30	Read-only constant is 0.	0	0
GDI	29..24	The starting position of the Global Directory index.	R / W	0x0
UDI	23..18	The starting position of the upper directory table (Upper Directory) index.	R / W	0x0
MDI	17..12	The starting position of the MIddle Directory index.	R / W	0x0
PTI	11..6	Page table entry table (Page Table) index starting position.	R / W	0x0
PTEI	5..0	Page table entry shift amount. The content of the page table entry that is read out will be logically shifted to the right by PTEI-2 bits first. It is used to remove the page table entry. Without putting the information into the TLB; then recirculate the right shift by 2 bits to move the RI and XI two bits of information to The top two digits of EntryLo0 or EntryLo1. Therefore, the PTEI field cannot be filled with 0 or 1, otherwise the processor The results will be uncertain.	R / W	0x0

95

Page 109

7.11 PWSize register (CP0 Register 5, Select 6)

The PWSize register is used together with the PWBase, PWField, and PWCtl registers. It is used in the GS464E for LDDIR and

The execution of the LDPTE instruction provides relevant configuration information. The LDDIR and LDI port multi-level page table structure traversal search, the page ta

Contains up to four levels of directory tables and one level of page table entry tables. For the supported pag l access process, see Figure 7-9 on page 94 . Stored in page table

The index value pointing to the next-level page table or the final page table entry is obtained by interceptin bits from the virtual address (BadVAddr) to be searched.

The PWSize register is used to identify the number of consecutive bits intercepted in the virtual address (BadVAddr) of the page table index at each level.

The PWSize PS field is used to control whether the bit width of the pointer in the directory table is 32 bits or 64 bits. The table of contents at all levels is based on PWField and PWSize

Find the index value intercepted in the virtual address (BadVAddr) needs to be multiplied by the width of the pointer in the directory table (shift 2 bits to the left or shift 3 bits to the left) to for

Fetch address XTLB Refill exception handling can only use 64-bit pointers, and TLB Refill exceptions can use 32-bit or 64-bit pointers.

GS464E uses MID to construct four 64-bit address spaces isolated from each other, in order to use LDDIR and

The LDPTE instruction accelerates page table traversal search, and the PWBase, PWField, PWSize, and PWCtl registers are implemented as hardware independent

Two groups. Which group the software and hardware actually operate is controlled by the MID field of the Diag register. When Diag.MID = 0, operate the first group; when

When Diag.MID! = 0, operate the second group.

Figure 7-12 illustrates the format of the PWField reg es the fields of the PWField register.

Figure **7-12 PWSize** register format

0	BDW

31	30	29	twenty four	twenty three	18	17	12	11	6	5	0
0	PS	GDW		UDW		MDW		PTW		PTEW	

Table **7-14 PWSize** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	63..38	The read-only constant is 0.	0	0
BDW	37..32	The index bit width of the base directory table. 0 means no need to search the base directory table.	R / W	0x0
0	31	Read-only constant is 0.	0	0
PS	30	The pointers of directory tables at various levels are wide. 0: 32-bit pointer; 1: 64-bit pointer.	R / W	0x0
GDW	29..24	Index width of the global catalog table. 0 is meaningless. When configured to 0, the processor result is undefined.	R / W	0x0
UDW	23..18	The index of the upper directory table is wide. 0 is meaningless. When configured to 0, the processor result is undefined.	R / W	0x0
MDW	17..12	The index bit width of the intermediate table of contents. 0 is meaningless. When configured to 0, the processor result is undefined.	R / W	0x0
PTW	11..6	The page table entry table index is wide. 0 is meaningless. When configured to 0, the processor result is undefined.	R / W	0x0
PTEW	5..0	Page table item width. Used to control the index of the page table entry shifted to the left by 3 bits (machine default data path width is 64 Digits). For example, when the width of each page table entry (single page) is 16 bytes, PTEW Hard set to 1, the index of the access page table entry will be shifted left (3 + 1) bits to form the final access address.	R / W	0x0

7.12 **Wired** Register **(CP0 Register 6, Select 0)**

The Wired register is a readable and writable register used to define the boundary between the fixed entry and the random replacement entry in the VTLB. Its indication is as follows As shown in Figure 7-13 .

Figure **7-13 VTLB** fixed entry and random replacement entry boundary

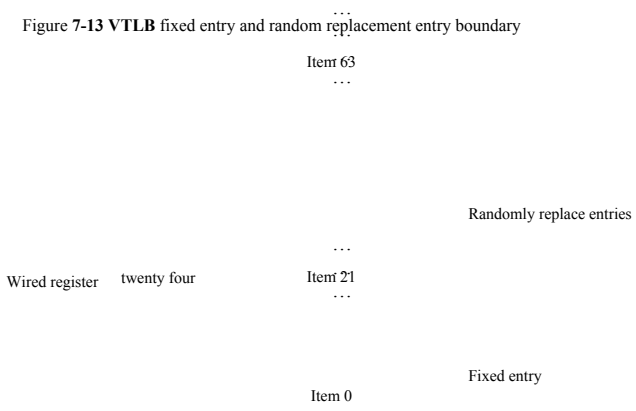


Figure 7-14 illustrates the format of the Wired re ges the fields of the Wired register.

Figure **7-14 Wired** register format

31	6	5	0
0		Wired	

Table **7-15 Wired** register domain description

Domain name	Bit	Functional description	Read / write	reset value
0	31..6	Read-only constant is 0.	0	0
Wired	5..0	TLB fixed the boundary between the entry and the random replacement entry.	R / W	0x0

97

Page 111

7.13 **PWCtl** register **(CP0 Register 6, Select 6)**

The PWCtl register is used together with the PWBase, PWField, and PWSize registers, and is used in the GS464E for LDDIR and

The execution of the LDPTE instruction provides relevant configuration information. The LDDIR and LDI ıport multi-level page table structure traversal search, the page ta
Contains up to four levels of directory tables and one level of page table entry tables. For the supported pag l access process, see Figure 7-9 on page 94 . Stored in page table
The index value pointing to the next-level page table or the final page table entry is obtained by interceptin bits from the virtual address (BadVAddr) to be searched.

The PWSize register is used to identify the number of consecutive bits intercepted in the virtual address (BadVAddr) of the page table index at each level.

PWCtl is used to control whether the base directory table is used in page table traversal search, and the support of large pages.

GS464E uses MID to construct four 64-bit address spaces isolated from each other, in order to use LDDIR and
The LDPTE instruction accelerates page table traversal search, and the PWBase, PWField, PWSize, and PWCtl registers are implemented as hardware independent
Two groups. Which group the software and hardware actually operate is controlled by the MID field of the Diag register. When Diag.MID = 0, operate the first group; when
When Diag.MID! = 0, operate the second group.

Figure 7-15 illustrates the format of the PWCtl reg ıes the fields of the PWCtl register.

Figure **7-15 PWCtl** register format

31	30	29	7	6	5	0
0	PWDirExt			HugePg	PSn	

Table **7-16 PWCtl** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31	Read-only constant is 0.	0	0
PWDirExt	30	The base directory table (Base Directory) table function enable bit. 1: enable; 0: disable.	R / W	0x0
0	29..7	Read-only constant is 0.	0	0
HugePg	6	1 means large pages are supported in the catalog table; 0 means large pages are not supported in the catalog table.	R / W	0x0
PSn	5..0	Used to indicate the location of the PTEVld bit of an entry in the directory table.	R / W	0x0

98

Page 112

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.14 HWREna register (CP0 Register 7, Select 0)

A bit mask is stored in the HWREna register to contr ers can be read by the RDHWR instruction in user mode.

Figure 7-16 illustrates the format of the HWREna reg es the fields of the HWREna register.

Figure **7-16 HWREna** register format

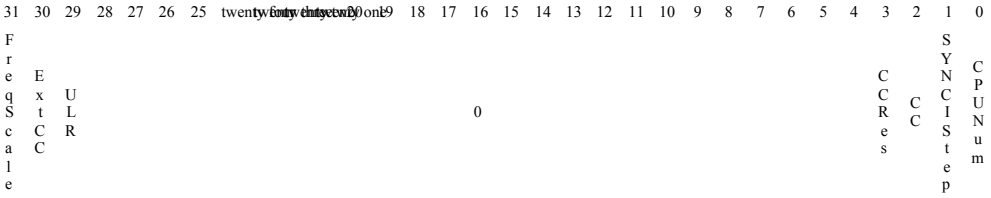


Table **7-17 HWREna** register field description

Domain name	Bit	Functional description	Read / write	reset value
FreqScale	31	RDHWR 31 (processor core frequency division factor) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
ExtCC	30	RDHWR 30 (external Count register) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
ULR	29	RDHWR 29 (UserLocal register) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
0	28..4	Read-only constant is 0.	0	0
CCRes	3	RDHWR 3 (Count self-increasing frequency) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
CC	2	RDHWR 2 (Count register) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
SYNCI-Step	1	RDHWR 1 (SYNCI_Step) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0
CPUNum	0	RDHWR 0 (EBase CPUNum) enable bit. 1: Read is allowed; 0: Read is prohibited.	R / W	0x0

99

Page 113

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.15 BadVAddr register (CP0 Register 8, Select 0)

The BadVAddr register is a read-only register used to record the virtual address that caused the following exception to occur most recently:

- Address error (AdEL or AdES)

- TLB / XTLB Refill
- TLB Invalid (TLBL or TLBS)
- TLB Modified

Figure 7-17 illustrates the format of the BadVAddr reg es the fields of the BadVAddr register.

Figure **7-17 BadVAddr** register format

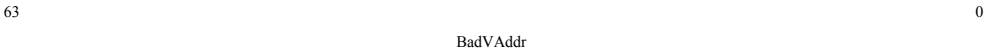


Table **7-18 BadVAddr** register field description

Domain name	Bit	Functional description	Read / write	reset value
BadVAddr	63..0	The virtual address of the error.	R	no

7.16 **Count** register **(CP0 Register 9, Select 0)**

The Count register and the Compare register are used together to implement a high-precision timer and timer interrupt in the processor. The
The frequency of the timer incrementing by 1 is 1/2 of the frequency of the processor core pipeline clock. During execution, the processor core pipeline clock frequency may be Dynamic adjustment, so the self-increasing frequency of Count also changes accordingly.

To complete certain functions or diagnostic purposes, the software can configure the Count register, such as timer reset, synchronization and other operations.

The format of the Count register is explained; each field of the Count register is described.

Figure **7-18 Count** register format

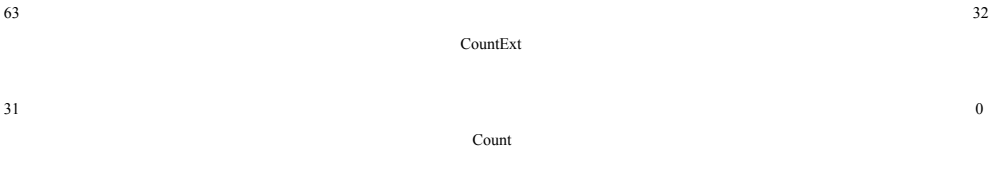


Table **7-19 Count** register field description

Domain name	Bit	Functional description	Read / write	reset value
CountExt	63..32	The upper 32 bits of the internal counter are extended.	R / W	no
Count	31..0	Internal counter.	R / W	no

Programming tips:

The Count implemented by this processor is a 64-bit counter. The timer interrupt generated by Count / Compare within the processor is still through

Triggered by the comparison of the lower 32-bit value of Count and the value of the Compare register. RDHWR (rd = $ 3) returns the sign of the lower 32 bits of the Count register Expanded to 64-bit results. The MFC0 instruction can only read the result of the sign extension of the lower 32 bits of the Count register to 64 bits, but using the MTC0 instruction When writing to the Count register, the processor writes all 64-bit values in the source register to the Count register. It is therefore recommended that if the software hopes To access the complete 64-bit information in the Count register, please use the DMFC0 and DMTC0 instructions.

7.17 GSEBase register (CP0 Register 9, Select 6)

The GSEBase register is a readable and writable regis ıfigure the base address of Godson extended exception vector.

Figure 7-20 illustrates the format of the GSEBase reg es the fields of the GSEBase register.

Figure **7-19 GSEBase** register format

Table **7-20 GSEBase** register field description

Domain name	Bit	Functional description	Read / write	reset value
Exception Base	63..12	When Status.BEV = 0, the logic shifts 12 bits to the left as the base address of Godson extended exception entry vector. [63:30] Bits can only be written when the WG bit is equal to 1, when writing to the EBase register when the WG bit is equal to 0 [63:30] The bit remains unchanged.	R / W	0xffff.ffff .8000.0
WG	11	[63:30] Bit write control bit. 1: ExceptionBase [63:30] can be written; 0: ExceptionBase [63:30] value remains unchanged when writing.	R / W	0x0
0	10..0	Read-only constant is 0.	0	0

102

Page 116

7.18 **PGD** Register **(CP0 Register 9, Select 7)**

The PGD register is a read-only register that sto ıe page table.

Figure 7-20 illustrates the format of the PGD re; ɔes the fields of the PGD register.

Figure **7-20 PGD** register format

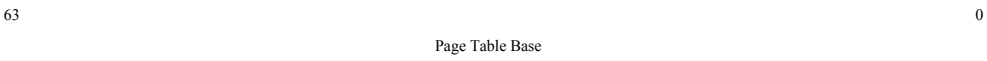


Table **7-21 PGD** register field description

Domain name	Bit	Functional description	Read / write	reset value
Page Table Base	63..0	When the 63rd bit in the BadVaddr register is 1, the value of the Page Table Base is equal to the KScratch6 register The value stored in; When the 63rd bit in the BadVaddr register is 0, the value of the Page Table Base is equal to the PWBase register The stored value.	R	0x0

103

Page 117

7.19 EntryHi register (CP0 Register 10, Select 0)

The EntryHi register is used to store the high-order information of TLB entries during TLB read, write, and query access.

In the case of ordinary instruction execution, the EntryHi ASID field stores the current address space identifier filled in by the software, and the virtual address fetch or fetch operation Address together to participate in TLB lookup. When a TLB exception (TLB Refill, XTLB Refill, TLB Invalid or TLB Modifid exception) occurs,
The corresponding part of the error address that triggered the exception is written into the EntryHi R and EntryHi VPN2 fields. When the TLBP instruction is executed, the virtual ground of th The address information and address space identification information are stored in the EntryHi R , EntryHi VPN2 and EntryHi ASID fields for TLB search.

When the TLBR instruction is executed, the content read from the specified TLB entry will also be written to the corresponding field of the EntryHi register. Since the implementation o The instruction will cover the EntryHi ASID field, so the software must save the ASID field value before executing the TLBR instruction, and after the TLBR execution is completed and Recovery. This is especially important for TLB Invalid and TLB Modified exception handlers, and other related memory management code.

When the TLBWI and TLBWR instructions are executed, the virtual address information and address space identification information of the item to be written are stored in EntryHi R , EntryHi VPN2 and EntryHi ASID fields are used for TLB writing. When the TLBWI instruction is executed, if the EntryHi EHINV field is set to 1, you can set the The specified TLB entry is invalid. Since the EntryHi EHINV field is also covered by the content read by the TLBR instruction, it is also necessary to execute the TLBR instruction It is important to maintain the EHINV domain as well as t subsequent execution of the TLBWI instruction.

Figure 7-21 illustrates the format of the EntryHi reg es the fields of the EntryHi register.

Figure **7-21 EntryHi** register format

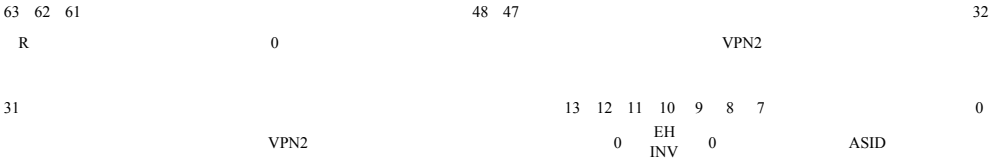


Table **7-22 EntryHi** register field description

Domain name	Bit	Functional description	Read / write	reset value
R	63..62	The area identification bit corresponds to the [63:62] bits of the virtual address. 0b00: user address region (xuseg, user address region); 0b01: supervisory address region (xsseg, supervisor address region); 0b10: reserved; 0b11: kernel address region (xkseg, kernel address region).	R / W	0x0
0	61..48	Read-only constant is 0.	0	0
VPN2	47..13	The virtual page number divided by 2 (mapped to double pages) corresponds to the [47:13] bits of the virtual address.	R / W	0x0
0	12..11	Read-only constant is 0.	0	0
EHINV [1]	10	TLB invalid flag. When this bit is 1, executing the TLBWI instruction will invalidate the corresponding TLB entry. When the TLBR instruction is executed, if the TLB entry read is invalid, this bit is set to 1.	R / W	0x0
0	9..8	Read-only constant is 0.	0	0

1 While CONFIG4 IEs = 0, but still in accordance with the MIPS specification GS464E CONFIG4 IEs > 1 defined in the case of realizing a EHINV domain. It is recommended to customize GS464E in deptl The core and other software use EntryHi EHINV domain function to facilitate the management of TLB.

Page 118

Domain name	Bit	Functional description	Read / write	reset value
ASID	7..0	Address space identification number. Used to allow multiple processes to share TLB; by using ASID to distinguish, for the same The virtual page number can make different processes use different mappings.	R / W	0x0

105

7.20 **Compare** register **(CP0 Register 11, Select 0)**

The Compare register and the Count register are used together to implement a high-precision timer and timer interrupt in the processor.
The value stored in the Compare register remains unchanged after writing, and compares with the lower 32 bits of the Count register, and triggers when the two are equal
The timer is interrupted, and Cause IT is set to 1. When the vectored interrupt mode is not used, the timer interrupt will be connected to interrupt line 7 (Cause IP7 , hard in
Disconnection 5). When using the vectored interrupt mode, the interrupt line to which the timer interrupt is connected is determined by IntCtl IPTI .

When the software writes the Compare register, the h lly clear Cause IT to clear the timer interrupt.

Figure 7-22 illustrates the format of the Compare reg es the fields of the Compare register.

Figure **7-22 Compare** register format

Table **7-23 Compare** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
Compare	31..0	Interval count comparison value.	R / W	0x0

7.21 **Status** register **(CP0 Register 12, Select 0)**

The Status register is a readable and writable regi ;or operating modes, interrupt enable, and processor status diagnostic information.

Figure 7-23 illustrates the format of the Status reg ıes the fields of the Status register.

Figure **7-23 Status** register format

31	30	29	28	27	26	25	twenty four	twenty three	twenty two	twenty one	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CU2	CU1	CU0	RP	FR	0	MX	PX	BEV	0	SR	NMI	0	MID					IM7..IM0				KX	SX	UX	KSU	ERL	EXL	IE	

Table **7-24 Status** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31	Read-only constant is 0.	0	0
CU2	30	Coprocessor 2 available flags. 1: available; 0: disabled.	R / W	0x0
CU1	29	Coprocessor 1 (floating point coprocessor) is available for identification. 1: available; 0: disabled.	R / W	0x1
CU0	28	Coprocessor 0 is available. 1: available; 0: disabled. When the processor is in Kernel mode and Debug mode, coprocessor 0 is always available, no need to consider whether the CU0 bit is 1.	R / W	0x1
RP	27	Processor dynamic down-conversion enable bit. 1: On; 0: Off.	R / W	0x0
FP	26	Floating register mode control bits. 0: 16 floating-point registers, even numbers, each 64 bits, single precision is stored in the lower 32 bits; 1: 32 floating-point registers, consecutively numbered, each 64 bits, single precision is stored in the lower 32 bits.	R / W	0x0
0	25	Read-only constant is 0.	0	0
MX	twenty four	Enable bit to access DSP resources. 1: accessible; 0: prohibited access.	R / W	0x0
PX	twenty three	The enable control bit for 64-bit operation in user mode is not used to enable the 64-bit address space. (In the remaining modes 64-bit operation does not need to be enabled) 1: enable; 0: disable.	R / W	0x1
BEV	twenty two	Exception vector entry address control. 0: normal; 1: start.	R / W	0x1

0	twenty one	Read-only constant is 0.	0	0
SR	20	Used to indicate that the entry into the reset exception vector was caused by a soft reset (Soft Reset). 1: It is a soft reset; 0: It is not a soft reset (may be NMI or Reset). When this bit is 0, the hardware will ignore the action of software writing 1 to this bit, that is, the bit cannot be formed by software from 0 → 1 Jump.	R / W	0x0
NMI	19	Used to indicate that the entry into the reset exception vector was due to a non-maskable interrupt (NMIt). 1: It is a non-maskable interrupt; 0: It is not a non-maskable interrupt (may be Reset or Soft Reset). When this bit is 0, the hardware will ignore the action of software writing 1 to this bit, that is, the bit cannot be formed by software from 0 → 1 Jump.	R / W	0x0
0	18	Read-only constant is 0.	0	0

Domain name	Bit	Functional description	Read / write	reset value
MID	17..16	The Godson custom field is used to indicate which address space is currently the default operation. Fetch the address where the address falls Whether the time is controlled by the MID field is determined by the Diag.INST field. If the memory access operation does not carry the MID information, it will fall on The address space specified by the Root.Status.MID field, otherwise it falls in the address space carried by the MID Information decision. The GS464E uses a 2-bit code (MID) to identify four isolated full address spaces. Address with MID = 0 is empty Is the address space visible to the MIPS host, and the address space with MID = 1 is the address visible to the virtual machine in guest mode Space, other values of MID can be assigned to other virtual machines by software. The MID field can only be written when Diag.VMM = 1. When Diag.VMM = 0, the field will be set to 0 no matter what value is written.	R / W	0x0
IM7..IM0	15..8	Interrupt mask bit. Each bit controls the enable of an external interrupt, internal interrupt, or software interrupt. 1: enable; 0: shield.	R / W	0x0
KX	7	1: Can access 64-bit Kernel segment, Kernel segment access uses XTLB Refill exception vector; 0: 64-bit Kernel segment cannot be accessed. Kernel segment access uses TLB Refill exception vector.	R / W	0x1
SX	6	1: Can access 64-bit Supervisor segment, Supervisor segment access uses XTLB Refill exception vector; 0: The 64-bit Supervisor segment cannot be accessed. Supervisor segment access uses the TLB Refill exception vector.	R / W	0x1
UX	5	1: Can access 64-bit User segment, User segment access uses XTLB Refill exception vector, allowing user mode Instruction to operate on 64-bit data; 0: Cannot access 64-bit User segment. User segment access uses TLB Refill exception vector, user mode is not allowed Instruction, operate 64-bit data.	R / W	0x1
KSU	4..3	Processor mode flag. 0b00: Kernel Mode 0b01: Supervisor Mode 0b10: User Mode 0b11: reserved.	R / W	0x0
ERL	2	Error level. This bit is set when Reset, Soft Reset, NMI, and Cache Error exceptions occur. 0: normal level; 1: error level. When the ERL position is 1: • The processor is automatically in the core state • All hardware and software interrupts are masked • The ERET instruction will read the return address from the ErrorEPC register • The kuseg segment will be considered as having unmapped and uncached attributes	R / W	0x1
EXL	1	Exceptional level. This bit occurs when an exception that is not an exception to Reset, Soft Reset, NMI, and Cache Error occurs Set to 1. 0: Normal level; 1: Exception level. When the EXL position is 1, it is: • The processor is automatically in the core state • All hardware and software interrupts are masked • TLB / XTLB Refill exception handling uses a general exception vector entry instead of TLB / XTLB Refill exception Volume entrance • EPC and Cause BD are not updated when new exceptions occur.	R / W	0x0
IE	0	Global interrupt enable bit. 0: shield all hardware and software interrupts; 1: Enable all hardware and software interrupts.	R / W	0x0

108

Page 122

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.22 **IntCtl** register **(CP0 Register 12, Select 1)**

The IntCtl register is a readable and writable regi rocessor's interrupt mechanism.

Figure 7-24 illustrates the format of the IntCtl reg es the fields of the IntCtl register.

Figure **7-24 IntCtl** register format

Table **7-25 IntCtl** register field description

Domain name	Bit	Functional description	Read / write	reset value
IPTI	31..29	Used to indicate on which interrupt line the timer interrupt is merged in the vector interrupt mode. The constant value is 7, which means it is merged on IP7 and hardware interrupt line HW5.	R	0x7
IPPCI	28..26	It is used to indicate on which interrupt line the performance counter overflow interrupt is merged in the vector interrupt mode. The constant value is 7, which means it is merged on IP7 and hardware interrupt line HW5.	R	0x7
0	25..10	Read-only constant is 0.	0	0
VS	9..5	It is used to define the interval of each interrupt vector entry address in vector interrupt mode. **VS** encoding / Vector spacing 0x00 / 0x000 0x01 / 0x020 0x02 / 0x040 0x04 / 0x080 0x08 / 0x100 0x10 / 0x200 All other code values listed in the above table are reserved. If the reserved value is configured for the VS field, the processor results Will be unsure.	R / W	0x0
0	4..0	Read-only constant is 0.	0	0

109

Page 123

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.23 **SRSCtl** register **(CP0 Register 12, Select 2)**

The SRSCtl register is used to control the shadow register. Because GS464E only implements a set of general registers, the shadow of general registers It is the general register itself.

Figure 7-25 illustrates the format of the SRSCtl reg ıes the fields of the SRSCtl register.

Figure **7-25 SRSCtl** register format

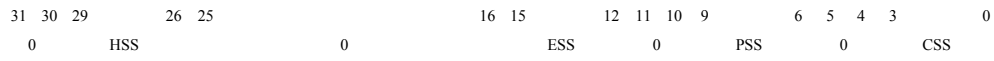


Table **7-26 SRSCtl** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31..30	Read-only constant is 0.	0	0
HSS	29..26	The value is 0, which means that only a set of general registers are implemented.	R	0x0
0	25..16	The read-only constant is 0.	0	0
ESS	15..12	The group number of the shadow register bank used for exception handling. GS464E can only write 0, write other value processor line Not sure.	R / W	0x0
0	11..10	Read-only constant is 0.	0	0
PSS	9..6	The group number of the previous group of shadow registers. GS464E can only write 0, the processor behavior is uncertain when writing other values.	R / W	0x0
0	5..4	Read-only constant is 0.	0	0
CSS	3..0	The value is always 0, indicating that the current shadow register set is the general register set.	R	0x0

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.24 **Cause** register **(CP0 Register 13, Select 0)**

The Cause register is mainly used to describe the cause of the last exception. In addition, software interrupts and interrupt vectors are controlled. apart from Outside the IP1..0, DC, IV, and WP domains, the other gister are read-only for the software.

Figure 7-26 illustrates the format of the Cause reg ıes the fields of the Cause register.

Figure **7-26 Cause** register format

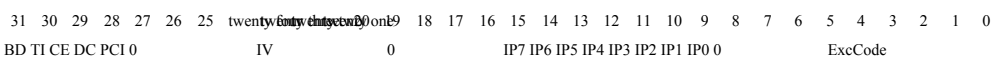


Table **7-27 Cause** register field description

Domain name	Bit	Functional description	Read / write	reset value
BD	31	Identifies whether the instruction with the most recent exception is in a branch delay slot. 1: in the delay slot; 0: not in the delay slot	R	0x0

TI	30	Timer interrupt indication. 1: There is a timer interrupt pending; 0: No timer interrupt.	R	0x0
CE	29..28	Record the unavailable coprocessor number when an unavailable coprocessor exception occurs.	R	0x0
DC	27	The Count register prohibits counting control bits. 1: Stop Count counting; 0: Enable Count counting.	R / W	0x0
PCI	26	Performance counter overflow interrupt indication. 1: Performance counter overflow interrupt pending; 0: No performance counter overflow interrupt.	R	0x0
0	25..24	Read-only constant is 0.	0	0
IV	twenty three	Interrupt exception vector entry control bit. 1: Use special interrupt vector (0x200); 0: Use general exception vector (0x180).	R / W	0x0
0	22..16	Read-only constant is 0.	0	0
IP7..IP2	15..10	Pending hardware interrupt identification. Each bit corresponds to an interrupt line, and IP7 ~ IP2 in turn correspond to hardware interrupts 5 ~ 0. 1: There is a pending interrupt on this interrupt line; 0: No interrupt on this interrupt line.	R	0x0
IP1..IP0	9..8	Pending software interrupt indicator. Each bit corresponds to a software interrupt, and IP1 ~ IP0 in turn correspond to software interrupts 1 ~ 0. The software interrupt flag can be set and cleared by software.	R / W	0x0
0	7	Read-only constant is 0.	0	0
ExcCode	6..2	Exception coding. See detailed description.		
0	1..0	Read-only constant is 0.	0	0

Table **7-28 ExcCode** encoding and its corresponding exception types

ExcCode	Mnemonic	description
0x00	Int	Interrupt
0x01	Mod	TLB modification exception
0x02	TLBL	TLB exception (read data or fetch instruction)
0x03	TLBS	TLB exception (write data)
0x04	AdEL	Address error exception (read data or fetch instructions)
0x05	AdES	Address error exception (write data)

111

Page 125

ExcCode	Mnemonic	description
0x06	IBE	The bus error exception (instruction fetch) defined by MIPS regulations. Because GS464E does not implement the IBE exception, so The exception reason code is reserved. If you find that the system prints "Bus Error" during software debugging, please check if there are other abnormalities.
0x07	DBE	The exception is bus errors (read data or write data). Because the GS464E does not implement the DBE exception, the exception The reason code is reserved. If you find that the system prints "Bus Error" during software debugging, please check if there are other abnormalities.
0x08	Sys	System call exception.
0x09	Bp	The breakpoint exception. If the SDBBP instruction is executed in EJTAG Debug mode, the exception code 0x9 (Bp) will be written to Debug The DExcCode field of the register.
0x0a	RI	The exception to the reserved instructions.
0x0b	CpU	Coprocessor is not available exception.
0x0c	Ov	Work out the overflow exception.
0x0d	Tr	The exception is the trap.
0x0e	MSAFPE	Not realized.
0x0f	FPE	Floating point exception.
0x10	GSExc	Godson custom exception. Including floating-point stack exceptions, virtual machine memory management exceptions, and virtual machine space TLB exam outer. The software can clarify what kind of exception occurs by looking at the relevant field of the GSCause register.
0x11	-	Keep
0x12	-	Keep
0x13	TLBRI	TLB read block exception
0x14	TLBXI	TLB execution block exception
0x15	MSADis	Not realized.
0x16	MDMX	Not realized.
0x17	WATCH	Not realized.
0x18	MCheck	Not realized.

0x19	Thread	Not realized.
0x1a	DSPDis	DSP module disable exception
0x1b	GE	Not realized.
0x1c	-	Keep
0x1d	-	Keep
0x1e	-	Cache error exception. Because the Cache error exception uses a dedicated vector entry address, a Cache error occurs in normal mode The ExcCode field of the Cause register is not updated when outside. When in Debug mode, the exception code is 0x1e Will be written to the DExcCode field of the Debug register.
0x1f	-	Keep

7.25 EPC register (CP0 Register 14, Select 0)

The EPC register is a 64-bit readable and writable register that contains the PC that continues to execute after the exception processing is completed.

In response to a synchronous (exact) exception, the processor writes to the EPC register:

PC that directly triggers the exception command.

When the instruction that directly triggers the exception is in the branch delay slot, record the PC of the previous branch or jump instruction of the instruction and Cause.BD Set to 1.

In response to an asynchronous (non-precise) exception, the processor writes to the EPC register the PC that continues to execute instructions after the exception processing is completed

When the EXL bit of the Status register is 1, the EPC register is not updated when an exception occurs.

The format of the EPC register is explained; each field of the EPC register is described.

Figure **7-27 EPC** register format

Table **7-29 EPC** register domain description

Domain name	Bit	Functional description	Read / write	reset value
EPC	63..0	Exception program counter.	R / W	no

113

Page 127

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.26 PRId register (CP0 Register 15, Select 0)

The PRId register is a 32-bit read-only register, which contains the identification of MIPS processor manufacturer, processor type and implementation version Information.

Figure 7-28 illustrates the format of the PRId re[illegible]es the fields of the PRId register.

Figure **7-28 PRId** register format

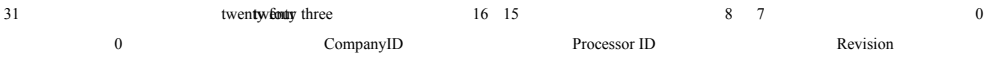


Table **7-30 PRId** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31..24	Read-only constant is 0.	0	0
CompanyID	23..16	Company ID number. When Diag.IDSEL is 0, the value is 0x14; when Diag.IDSEL is 1, the value is 0x00.	R	0x14
ProcessorID	15..8	Processor type number. It is 0x63.	R	0x63
Revision	7..0	Implementation version number. When Diag.IDSEL is 0, the value is 0x08; when Diag.IDSEL is 1, the value is 0x05.	R	0x08

114

Page 128

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.27 EBase register (CP0 Register 15, Select 1)

The EBase register is a readable and writable regi :eption vector base address and a read-only CPU number.

Figure 7-29 illustrates the format of the EBase reg)es the fields of the EBase register.

Figure **7-29 EBase** register format

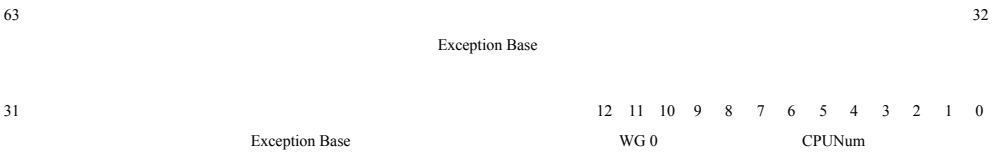


Table **7-31 EBase** register field description

Domain name	Bit	Functional description	Read / write	reset value
Exception Base	63..12	When Status.BEV = 0, the logic shifts 12 bits to the left as the base of the exception entry vector. [63:30] Bits can only be written when the WG bit is equal to 1, when writing to the EBase register when the WG bit is equal to 0 [63:30] The bit remains unchanged.	R/W	0xffff.ffff .8000.0
WG	11	[63:30] Bit write control bit. 1: ExceptionBase [63:30] can be written; 0: ExceptionBase [63:30] value remains unchanged when writing.	R / W	0x0
0	10	Read-only constant is 0.	0	0
CPUNum	9..0	The index number that identifies the current processor in a multi-core system.	R	

Programming tips:

When using vectored interrupt mode (Cause.IV = 1), if IntCtl.VS is configured as 0x10, there is an exception vector offset for interrupt No.
Will exceed the 0xfff range. At this time, the software needs to ensure that the 12th position of EBase is 0, yielding the highest bit of the offset of No. 7 interrupt vector.

7.28 Config register (CP0 Register 16, Select 0)

The configuration information of some processors is defined in the Config register. In the Config register, except for the K0 field that can be read and written by software, all other fields
It is initialized by the hardware during reset and remains read-only. Although GS464E resets the hardware of K0 domain to 0b010 (Uncached
Property), it is still strongly recommended that the softw

Figure 7-30 illustrates the format of the Config reg
Description.

Figure **7-30 Config** register format

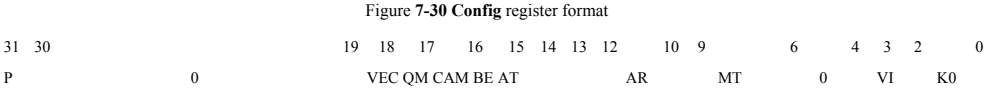


Table **7-32 Config** register field description

Domain name	Bit	Functional description	Read / write	reset value
M	31	A value of 1 indicates that the Config1 register exists.	R	0x1
0	30..19	Read-only constant is 0.	0	0
VEC	18	The value is 0, which means that Godson does not support custom 256-bit vector instru		0x0
QM	17	If the value is 1, it indicates that Godson supports 128-bit memory access instructions. I	ıs, please see	0x1
CAM	16	The v ıat it contains hardware CAM components and supports Goc ommands. For the related commands, please refer to Table	R	0x1
BE	15	A val little-endian addressing mode is used.	R	0x0
AT	14..13	A value of 2 means that the MIPS64 architecture is implemented and 64-bit full address space can be accessed.	R	0x2
AR	12..10	The value is 1, compatible with the MIPS64 release 5 specification, the specific details of the implementation of the software can be read by other Obtain the configuration field of the configuration register or other registers.	R	0x1
MT	9..7	The value is 4, MMU adopts the double TLB form of VTLB and FTLB. For information on dual TLB MMU, please refer to Section 4.3 .	R	0x4
0	6..4	Read-only constant is 0.	0	0
VI	3	When the value is 0, the instruction cache takes the form of virtual address Index and real address tag. For the organization form of Cache, please See section 5.1 .	R	0x0
K0	2..0	Cache attribute of Kseg0 segment. For the Cache attribute encoding supported by C r to Table 7-6 on page 89 .	RW	0x2

116

7.29 **Config1** register **(CP0 Register 16, Select 1)**

The Config1 register is used to provide some config

Figure 7-31 illustrates the format of the Config1 reg

Described.

Figure **7-31 Config1** register format

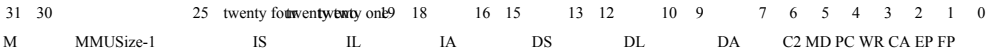


Table **7-33 Config1** register field description

Domain name	Bit	Functional description	Read / write	reset value
M	31	A value of 1 means that there is a Config2 register.	R	0x1
MMU Size-1	30..25	The value is 63, which is concatenated with the VTLBSizeExt field of the Config4 register to form a 10-bit value: Config4.VTLBSizeExt 3..0 || Config1.MMUSize-1 5..0 ; The spliced value is 63, which means that the VTLB is 64 items.	R	0x3f
IS	24..22	The value is 2, indicating that each way of I-Cache contains 256 lines.	R	0x2
IL	21..19	The value is 5, which means that the length of each line of I-Cache is 64 bytes.	R	0x5
IA	18.	The value of 3 is 3, indicating that the I-Cache contains 4 channels.	R	0x3
DS	15..13	The value is 2, indicating that each way of D-Cache contains 256 lines.	R	0x2
DL	12..10	The value is 5, which means that the length of each line of D-Cache is 64 bytes.	R	0x5
DA	9..7	A value of 3 means that D-Cache contains 4 channels.	R	0x3
C2	6	A value of 1 indicates that coprocessor 2 (COP2) is included.	R	0x1
MD	5	A value of 0 means that the MDMX ASE instruction set is not implemented.	R	0x0

PC	4	A value of 1 indicates that a performance counter has been implemented.	R	0x1
WR	3	A value of 0 means that the Watch register is not implemented.	R	0x0
CA	2	A value of 0 means that the MIPS16e instruction set is not implemented.	R	0x0
EP	1	A value of 1 indicates that EJTAG is implemented.	R	0x1
FP	0	A value of 1 indicates that a floating-point coprocessor is implemented.	R	0x1

7.30 **Config2** register **(CP0 Register 16, Select 2)**

The Config2 register is used to provide some config

Figure 7-32 illustrates the format of the Config2 reg

Described.

Figure **7-32 Config2** register format

31	30	29	28	27	26	25	twenty four	twenty three	twenty two	twenty one	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M		0			TS				TL			TA			0				SS				SL				SA			

Table **7-34 Config2** register domain description

Domain name	Bit	Functional description	Read / write	reset value
M	31	A value of 1 indicates that there is a Config3 register.	R	0x1
0	30..28	Read-only constant is 0.	0	0
TS	27..24	The value is 2, which means that each way of V-Cache contains 256 lines.	R	0x2
TL	23.20	value is 5, which means that the length of each line of V-Cache is 64 bytes.	R	0x5
TA	19..16	The value is 15, indicating that the V-Cache contains 16 channels.	R	0xf
0	15..12	The read-only constant is 0.	0	0
SS	11..8	A value of 4 means that each channel of S-Cache contains 1024 lines.	R	0x4
SL	7..4	A value of 5 means that each line of S-Cache is 64 bytes long.	R	0x5
SA	3..0	A value of 15 means that the S-Cache contains 16 channels.	R	0xf

118

Page 132

Loongson 3A3000 / 3B3000 Processor User Manual • Next

7.31 Config3 register (CP0 Register 16, Select 3)

The Config3 register is used to provide some config

Figure 7-33 illustrates the format of the Config3 reg

Described.

Figure 7-33 Config3 register format

31	30	29	28	27	26	25	twenty four	twenty three	twenty two	twenty one	20..19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	BPG	CMGCR	MSAP	BP	BI	SC	PW	VZ	R		R		MCU	R	ISA		ULRI	RXI	DSP2P	DSP	CTXTC	ITL	LPA	VEIC	VInt	SP	CDMM	MT	SM	TL

Table 7-35 Config3 register field description

Domain name	Bit	Functional description	Read / write	reset value
M	31	A value of 1 means that there is a Config4 register.	R	0x1
BPG	30	A value of 1 means that the TLB supports large pages larger than 256MB, and the corresponding PageMask register is 64 bits.	R	0x1
CMGCR	29	A value of 0 means that the Coherency Manager memory-mapped Global Configuration is not implemented Register Space. GS464E supports multi-core, uses a custom multi-core consistency management mechanism, and does not use MIPS for multi-core consistency Management framework, and MIPS has not issued a specification for its multi-core consistency management.	R	0x0
MSAP	28	A value of 0 means that the MIPS vector module (SIMD Module) is not implemented.	R	0x0
BP	27	A value of 0 means that the BadInstrP register is not implemented.	R	0x0
BI	26	A value of 0 means that the BadInstr register is not implemented.	R	0x0
SC	25	A value of 0 indicates that the Segment Control function is not implemented. Correspondingly, SegCtl0 for segment control, The SegCtl1 and SegCtl2 registers are not implemented.	R	0x0
PW	twenty four	A value of 0 means that the hardware TLB refill mechanism (Hardware Page Table Walk) is not implemented. However, in order to cooperate with the execution of Loongson's custom LWTR and LDTR instructions, the MIPS specification sets the emphasis on hardware TLB. The PWBase, PWField and PWSize registers defined by the filling mechanism are still defined in GS464E, and Moreover, the PWBase, PWField and PWSize registers are implemented in four groups, corresponding to SpaceID = 0, 1, 2, respectively 3 four different address spaces.	R	0x0
VZ	twenty three	A value of 1 indicates that hardware virtualization is supported .	R	0x1
R	22..21	This field is meaningless because MIPS MCU ASE is not implemented.	R	0x0
R	20..18	This field is meaningless because the micrMIPS64 instruction set is not implemented.	R	0x0
MCU	17	A value of 0 means that MIPS MCU ASE is not implemented.	R	0x0
R	16	Since MIPS64 and microMIPS64 are not implemented at the same time, this field is meaningless.	R	0x0
ISA	15..14	The value is 0, which means that only the MIPS64 instruction set is implemented, and the microMIPS64 instruction set is not implemented.	R	0x0
ULRI	13	A value of 1 indicates that the UserLocal register is implemented.	R	0x1
RXI	12	A value of 1 indicates that the RIE and XIE bits are implemented in the PageGrain register.	R	0x1
DSP2P	11	A value of 1 indicates that version 2 of the MIPS DSP module is implemented.	R	0x1
DSP	10	A value of 1 indicates that the MIPS DSP module is implemented.	R	0x1

119

Page 133

Domain name	Bit	Functional description	Read / write	reset value
CTXTC	9	A value of 0 means that the ContextConfig and XcontextConfig registers are not implemented. Because the operating system running on GS464E is not recommended to use single-level page table results, it is not recommended to use TLB In exception handling, the contents of the Context and Xcontext registers are used as pointers to page table entries. Correspondingly, The ContextConfig and XcontextConfig registers are also not implemented.	R	0x0
ITL	8	A value of 0 means that the MIPS IFlowTrace debugging function is not implemented.	R	0x0
LPA	7	A value of 1 indicates that a large physical address range is supported. Accordingly, the PageGrain register is implemented.	R	0x1
VEIC	6	A value of 0 means that the external interrupt controller (EIC) mode is not implemented in the interrupt mechanism.	R	0x0
VInt	5	The value is 1, indicating that the vectored interrupt (Vectored Interrupts) mode is implemented in the interrupt mechanism.	R	0x1
SP	4	A value of 0 means that small pages with a size of 1KB are not supported.	R	0x0
CDMM	3	A value of 0 means that the common device memory map (Common Device Memory Map) mechanism is not implemented.	R	0x0
MT	2	A value of 0 means that the MIPS multi-threaded module (MT Module) is not implemented.	R	0x0
SM	1	A value of 0 means that SmartMIPS TM ASE is not implemented .	R	0x0
TL	0	A value of 0 means that Trace logic is not implemented.	R	0x0

7.32 Config4 register (CP0 Register 16, Select 4)

The Config4 register is used to provide some config ıe processor and to control the page size of FTLB.

Figure 7-34 illustrates the format of the Config4 reg ıes the fields of the Config4 register.

Figure **7-34 Config4** register format

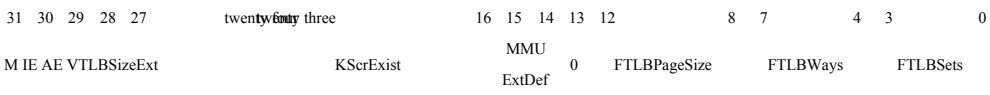


Table **7-36 Config4** register domain description

Domain name	Bit	Functional description	Read / write	reset value
M	31	A value of 1 indicates that there is a Config5 register.	R	0x1
IE [1]	30..29	The value is 0, which means that the TLBINV and TLBINVF instructions are not implemented and the function of EntryHi.EHINV domain is not implemented .	R	0x0
AE	28	A value of 0 means that the EntryHi ASID field width is still 8 bits.	R	0x0
VTLB-SizeExt	27..24	The value is 0, which is concatenated with the MMUSize-1 field of the Config1 register to form a 10-bit value: Config4.VTLBSizeExt 3..0 || Config1.MMUSize-1 5..0 ; The spliced value is 63, which means that the VTLB is 64 items.	R	0x0
KScrExist	23..16	The value is 0b11111100, indicating that the KScratch1 ~ 6 registers (CP0 Register 31, Selct 2 ~ 7) can be Core state software access.	R	0xfc
MMU-ExtDef	15..14	The value is 3, used to explain the format of the Config4 register. Where Config4 [3: 0] is FTLBSets, Config4 [7: 4] It is FTLBWays, Config4 [10: 8] is FTLBPageSize, Config4 [27:24] is VTLBSizeExt.	R	0x3
0	13	Read-only constant is 0.	0	0

[1] The Config4.IE field value is set to 0 to maintain compatibility with the MIPS specification. In fact, the GS464E processor core implements the TLBINVF instruction (with Config4.IE = 3 Execution), that is, when executing a TLBINVF instruction, the hardware will invalidate the entire TLB entry. In addition, the TL464V instruction is also implemented in the GS464E processor core, However, its execution effect is different from the definition of the MIPS specification, but is equivalent to the TLBINVF instruction.

The GS464E processor core also implements the function of EntryHi.EHINV domain: when the EntryHi.EHINV bit is 1, executing the TLBWI instruction will invalidate the corresponding item; When the TLBR instruction is executed, the value of the VPN2 invalid bit of the read TLB entry will be updated to the EntryHi.EHINV bit.

Domain name	Bit	Functional description	Read / write	reset value
FTLB-PageSize	12..8	Represents the page size used by FTLB. The page sizes and encoding values supported by FTLB in GS464E are as follows: Page size / Coded value 4KB 1 16KB 2 64KB 3 256KB 4 1MB 5 4MB 6 16MB 7 64MB 8 256MB 9 1GB 10 If the value written by the software to this field is not included in the above table, the value of this field remains unchanged. The software must clear the FTLB before modifying this field, otherwise the processor behavior is undefined.	R / W	0x1
FTLB-Ways	7..4	A value of 6 means that FTLB contains 8 channels	R	0x6
FTLB-Sets	3..0	A value of 7 means that each channel of FTLB contains 128 items.	R	0x7

122

Page 136

7.33 **Config5** register **(CP0 Register 16, Select 5)**

The Config5 register is used to provide some config

Figure 7-35 illustrates the format of the Config5 reg

Described.

Figure **7-35 Config5** register format

Table **7-37 Config5** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31	Read-only constant is 0.	0	0
R	30	Since the Segmentation Control mode is not implemented, this field is meaningless.	R	0x0
R	29	Since the Segmentation Control mode is not implemented, this field is meaningless.	R	0x0
R	28	Since the Segmentation Control mode is not implemented, this field is meaningless.	R	0x0
R	27	Because the MIPS Vector Module (SIMD Module) is not implemented, this field is meaningless.	R	0x0
0	26..1	Read-only constant is 0.	0	0
NFExists	0	A value of 1 indicates that the Nested Fault feature is supported .	R	0x1

123

Page 137

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.34 GSConfig register (CP0 Register 16, Select 6)

The GSConfig register is used to dynamically configure the functions related to the microstructure of the processor core. Software can be based on the specific characteristics of the prog
Get the best performance by turning on or off the correspon

Figure 7-36 illustrates the format of the GSConfig reg es the fields of the GSConfig register.

Figure **7-36 GSConfig** register format

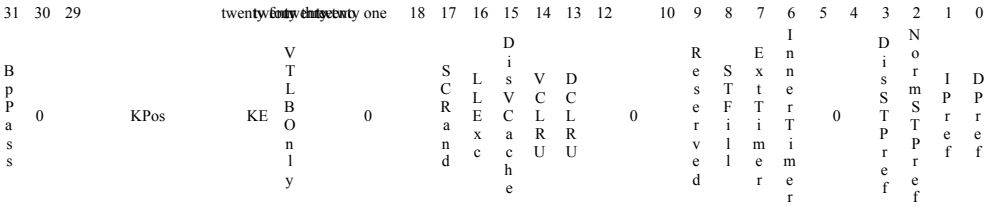


Table **7-38 GSConfig** register field description

Domain name	Bit	Functional description	Read / write	reset value
BpPass	31	EJTAG command and data breakpoint response mechanism control. 0: Any condition that meets the instruction breakpoint and data breakpoint will trigger an exception. 1: EJTAG instruction breakpoints and data breakpoint exceptions are automatically ignored for the first time when exception processing returns to re-execution Meet the conditions of instruction breakpoint and data breakpoint. When this bit is 0, the processor will process instruction breakpoints and data in the manner required by the MIPS EJTAG specification Breakpoint, which means that the EJTAG exception handler must modify the instruction breakpoint or data during processing The judgment condition of the breakpoint to ensure that the current instruction that triggers the breakpoint exception will not be triggered again after DERET returns Exception and fall into an endless loop, but also to ensure that the program can be stopped in a timely manner under the new judgment conditions, making the exception The processing program has time to set the breakpoint condition correctly before the observed program executes to the breakpoint to be observed again.	R / W	0x0
0	30	Read-only constant is 0.	0	0
KPos	29..24	indicates where the K bit is located in the page table entry (PTE).	R / W	0x3d
KE	twenty three	TLB page table kernel execution protection function enable bit. 0: Disable this function. The K bits of the EntryLo0 and EntryLo1 registers will be forbidden to write and forced to 0;	R / W	0x0
		1: Turn on this function. The K bits of the EntryLo0 and EntryLo1 registers can be used normally.		
VTLB-Only	twenty two	When set to 1, the processor will only operate VTLB in the dual TLB, which is equivalent to the traditional single CAM type of MIPS TLB, so as to ensure that the underlying software such as the operating system kernel running on GS464 does not modify the MMU part Can still run on GS464E. If you need to use a dual TLB MMU, you need to set this bit to 0. The software should clear the TLB before modifying this bit state, otherwise the processor behavior will be unknowable.	R / W	0x1
0	21..18	Read-only constant is 0.	0	0
SCRand	17	When set to 1, the Cache consistency maintenance component in the chip will initiate the SC / SCD instruction on the processor core. The return delay of the cache access request is increased by a random delay of 64 to 128 clock cycles. Set to 0 to turn off this function.	R / W	0x1

124

Page 138

Domain name	Bit	Functional description	Read / write	reset value
LLExc	16	When 1, the Cache access request initiated by the LL / LLD instruction must return to the exclusive state Cache block; When it is 0, the Cache access request initiated by the LL / LLD instruction allows returning the cache in the shared state Piece.	R / W	0x1
Dis- VCache	15	0: Use VCache; 1: Disable VCache. In the process from using VCache to disabling VCache, the software needs to ensure that there is nothing valid in VCache content.	R / W	0x0
VCLRU	14	Configure the VCache replacement algorithm. 1: LRU replacement algorithm; 0: pseudo-random replacement algorithm.	R / W	0x1
DCLRU	13	Configure the DCache replacement algorithm. 1: LRU replacement algorithm; 0: pseudo-random replacement algorithm.	R / W	0x1
0	12..10	Read-only constant is 0.	0	0
Reserved	9	This bit must be written to 1 after reset, and it will no longer be changed to 0.	R / W1	0x0
STFill	8	Processor store operation automatic write merge function enable bit. 1: On; 0: Off. Before modifying this state, the software needs to use the SYNC instruction to ensure that there is no unfinished memory in the processor operating.	R / W	0x1
Ext- Timer	7	When set to 1, the clock interrupt recorded by Cause.TI can come from an external timer belonging to the processor core; When set to 0, the clock interrupt recorded by Cause.TI does not come from the external timer belonging to the processor core. It should be pointed out that the increase frequency of the external timer is not affected by the frequency conversion of the processor core. Allow the software to set the ExtTimer and InnerTimer bits of the GSConfig register to 1 at the same time, but software The software can handle the situation correctly, and it is generally not recommended to do so.	R / W	0x0
Inner- Timer	6	When set to 1, the clock interrupt recorded by Cause.TI can come from the processor core with Count / Compare Register implemented timer; When set to 0, the clock interrupt recorded by Cause.TI does not come from the processor core and is sent by Count / Compare The timer implemented by the memory. It should be pointed out that the increase frequency of the timer implemented by the Count / Compare register will increase with the processor core The frequency changes in proportion to the change. Allow the software to set the ExtTimer and InnerTimer bits of the GSConfig register to 1 at the same time, but software The software can handle the situation correctly, and it is generally not recommended to do so.	R / W	0x1
0	5..4	Read-only constant is 0.	0	0
Dis- STPref	3	When set to 1, the processor does not automatically prefetch hardware for store operations in data access operations; set to 0 At this time, the processor will also automatically perform hardware prefetching for store operations, and the performance of store prefetching can be further passed GSConfig NormSTPref for configuration, please see the description of the next item. This field is meaningful only when GSConfig DPref = 1 , and modifying this field when GSConfig DPref = 0 will not cause performance Variety. Before modifying this state, the software needs to use the SYNC instruction to ensure that there is no unfinished memory in the processor operating.	R / W	0x0

Domain name	Bit	Functional description	Read / write	reset value
Norm- STPref	2	When set to 1, the flow control algorithm and load used by the processor when performing hardware automatic prefetch for store operations The operation is equivalent; when set to 0, the processor automatically prefetches the store operation hardware, except for the load operation. In addition to the equivalent flow control mechanism, it will also monitor the automatic write and merge success of store operations in the past period (collect The number of times a cache block is full), if the number of successes exceeds a certain threshold, the hardware operation of the store operation is temporarily stopped Pieces are automatically prefetched. This field is meaningful only when GSConfig DPref = 1 , and modifying this field when GSConfig DPref = 0 will not cause performance Variety. Before modifying this state, the software needs to use the SYNC instruction to ensure that there is no unfinished memory in the processor operating.	R / W	0x0

IPref	1	When set to 1, the processor performs hardware automatic prefetch for the fetch operation; otherwise, the function is turned off. Before modifying this state, the software needs to use the SYNC instruction to ensure that there is no unfinished memory in the processor operating.	R, W	0x1
DPref	0	When set to 1, the processor performs hardware automatic prefetch for data access operations; otherwise, the function is turned off. Before modifying this state, the software needs to use the SYNC instruction to ensure that there is no unfinished memory in the processor operating.	R, W	0x1

7.35 LLAddr register (CP0 Register 17, Select 0)

The LLAddr register is a 64-bit read-only register used to store the physical address of the most recent Load Linked instruction. When the exception returns On return, the LLAddr register is cleared.

Figure 7-37 illustrates the format of the LLAddr reg es the fields of the LLAddr register.

Figure **7-37 LLAddr** register format

63 0

Table **7-39 LLAddr** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
PAddr	63..0	The physical address of the most recent Load Linked instruction	R	no

127

7.36 XContext register (CP0 Register 20, Select 0)

The XContext register is a readable and writable register, which contains some high-level information of the page table base address and Some bits of the virtual address where the TLB exception occurred. According to the original design intent of the MIPS architecture, the XContext registers are stitched together The information can form a pointer to an item in the page table, which is used to access the page table item when a TLB exception occurs. XContext register does not do The page table accessible during any processing is a single-level page table structure, the page size is 4K bytes, each page table entry is 16 bytes, including Continue an even page table entry and an odd page table entry. When the page table does not adopt this structure, the software needs to enter the content of the XContext register The line is appropriately shifted and spliced. For an operating system that uses a multi-level page table, the XContext register can only be used to speed up the last Address generation for the level page table access.

The XContext register is mainly used in XTLB Refill exception handlers. But when TLB Refill, TLB Invalid and TLB Mod When an exception occurs, the BadVPN2 and R fields in the XContext register will also be updated, so the software can also use the corresponding exception handler The XContext register is used in the sequence.

The BadVPN2 and R fields in the XContext register copy part of the information in the BadVAddr register, but this does not mean that Is completely equivalent. When an Address Error exception occurs, the BadVaddr register will be updated by hardware, but the BadVPN2 of the Context register The domain and R domain will not be updated by hardware

Figure 7-38 illustrates the format of the XContext reg es the fields of the XContext register.

Figure **7-38 XContext** register format

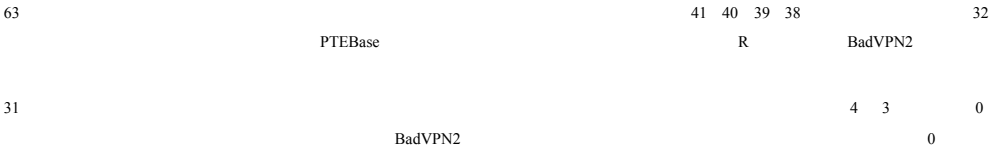


Table **7-40 XContext** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
PTEBase	63..41	page table base address high. It is configured by the operating system software according to the current page table	R / W	no
R	40..39	When a TLB exception occurs, the 63..62 bits of the erroneous virtual address are stored. 0b00: common user area; 0b01: Super user area; 0b10: reserved; 0b11: Core area.	R	no
BadVPN2	38..4	When a TLB exception occurs, the 47..13 bits of the erroneous virtual address are stored.	R	no

0	3..0	Read-only constant is 0.	0	0

128

7.37 **Diag** Register **(CP0 Register 22, Select 0)**

The Diag register is a self-defined register of G and special operations.

Figure 7-39 illustrates the format of the Diag re Diag register

Narrate.

Figure **7-39 Diag** register format

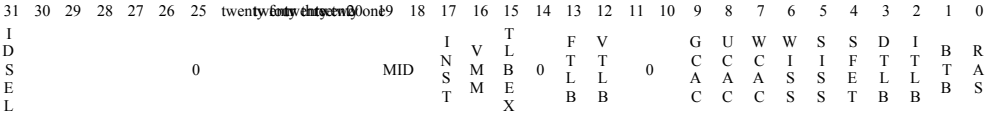


Table **7-41 Diag** Register Field Description

Domain name	Bit	Functional description	Read / write	Reset value
IDSEL	31	Controls the read value of the PRId register. When this bit is 0, the read value of PRId is 0x00146308; this bit is 1 At this time, the read value of PRId is 0x00006305. The latter is completely consistent with the PRId of the LS3A1000 chip processor. In order to achieve the compatible operation of the operating system kernel and the above software running on GS464 on GS464E, you can Modify this bit to 1 in PMON. It should be noted that after each hard restart, the IDSEL bit can only be modified once. It is recommended that this modification In PMON.	R / W	0x0
0	30..20	Read-only constant is 0.	0	0
MID	19..18	When exceptions to TLB Refill, TLB Invalid, and TLB Mod occur, they are accessed by the instruction that triggered the exception The MID of the address space. When the TLBWR and TLBWI instructions are used to fill in the TLB, the content of this field will be written into the TLB entry, Participate in subsequent virtual and real address mapping. When using the TLBP and TLBR instructions to read the TLB, the content of the MID stored in the read table entry is stored in this field. When the software reads and writes the PWBase, PWField, and PWSize registers, this field is used to indicate which one to access Page table configuration information in the space. The MID field can only be written when Diag.VMM = 1. When Diag.VMM = 0, the field will be set to 0 no matter what value is written.	R / W	0x0
INST	17	Whether the PC's virtual and real address mapping uses the SpaceID information flag, 0: not used; 1: used. This field is only valid when the VMM bit of the Diag register is 1, otherwise the field can be positive Often read and write, but do not participate in any other operations.	R / W	0x0
VMM	16	Loongson virtual machine address mapping enhanced mode enable bit. 1: On; 0: Off. After the enhanced address mapping mode of Loongson virtual machine is enabled, MID and VPID functions can be used. If there is a change in the mapping content of the page table entry with MID = 0 before and after modifying this bit, you need to modify this bit Before clearing all TLBs, clearing the TLB can be done through the FTLB, VTLB and ITLB fields of the Diag register Write 1 is completed, you can also use the TLBINVF instruction to complete.	R / W	0x0
TLBEX	15	When set to 1, execute TLBR, TLBP, TLBWI, TLBWR, TLBINVF in root-core mode The instruction will trigger the VMMU exception. The setting of this domain does not affect the Guest-Kernel mode Executing the above TLB instruction in Guest-Kernel mode will still trigger the PSI exception.	R / W	0x0
0	14	Read-only constant is 0.	0	0

129

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Domain name	Bit	Functional description	Read / writ	Reset value
FTLB	13	Writing 1 to this bit clears FTLB. Please note that the processor is concerned about the behavior of this bit being written to 1, cleared FTLB has nothing to do with whether the value of this bit is 1. The read value of this bit is always 0.	R0 / W	0
VTLB	12	Write 1 to this bit to clear VTLB. Please note that the processor is concerned about the behavior of this bit being written to 1, cleared FTLB has nothing to do with whether the value of this bit is 1. The read value of this bit is always 0.	R0 / W	0
0	11..10	Read-only constant is 0.	0	0
GCAC	9	When set to 1, executing CACHE0, CACHE1, CACHE3 instructions in Guest mode will not trigger PSI exception.	R / W	0x0
UCAC	8	When set to 1, the Root-User mode and the Guest-User execution mode CACHE0, CACHE1, CACHE3, The CACHE15, CACHE21, CACHE23 instructions will not trigger the coprocessor exception (CpU).	R / W	0x0
WCAC	7	Set to 1 to cancel the wait cache operation limit.	R / W	0x0
WISS	6	Set to 1 to cancel the wait issue operation restriction.	R / W	0x0
SISS	5	Set to 1 to cancel the restriction of stall issue operations.	R / W	0x0
SFET	4	Set to 1 to cancel the restriction of stall fetch operations.	R / W	0x0
ITLB	3	Writing 1 to this bit clears ITLB. Please note that the processor is concerned about the behavior of this bit being written to 1, clear FTLB It does not matter whether the bit value is 1. The read value of this bit is always 0.	R0 / W	0
ITLB	2	Writing 1 to this bit clears ITLB. Please note that the processor is concerned about the behavior of this bit being written to 1, clear FTLB It does not matter whether the bit value is 1. The read value of this bit is always 0.	R0 / W	0
BTB	1	Write 1 to this bit to clear BRBTB and BTAC in branch prediction. Please note that the processor is concerned about this bit The behavior of being written as 1, clearing FTLB has nothing to do with whether the value of this bit is 1. The read value of this bit is always 0.	R0 / W	0
RAS	0	Set to 1 to disable RAS for branch prediction for jr31.	R / W	0x0

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.38 GSCause register (CP0 Register 22, Select 1)

The GSCause register is a read-only register, which contains information related to the Godson expansion part when an exception occurs, such as triggering an exception Whether the order contains a prefix, the specific reason for expansion.

Figure 7-40 illustrates the format of the GSCause reg es the fields of the GSCause register.

Figure **7-40 GSCause** register format

31	12	11	8	7	6	2	1	0
0		TLBInst		0	GSExcCode		0	P

Table **7-42 GSCause** register field description

Domain name	Bit	Functional description	Read / write	reset value
0	31..12	Read-only constant is 0.	0	0
TLBInst	11..8	When the TLB instruction is executed in the guest state and the PSI exception is triggered, the specific TLB instruction type is recorded. TLBInst [0] is 1 means TLBP instruction; TLBInst [1] is 1 means TLBR instruction; TLBInst [2] is 1 means TLBWR instruction; When TLBInst [3] is 1, it means TLBWI, TLBINV or TLBINVF instruction.	R	0x0
0	7	Read-only constant is 0.	0	0
GS-ExcCode	6..2	Godson extended exception code.	R	0x0
0	1	Read-only constant is 0.	0	0
P	0	1: The instruction that triggered the exception carries a prefix, and the instruction is 64 bits long; 0: The instruction that triggers the exception does not carry a prefix, and the instruction is 32 bits long.	R	0x0

Table **7-43 GSExcCode** codes and their corresponding exception types

ExcCode	Mnemonic	description
0x00	IS	Floating-point stack exception
0x01	VMMU	Virtual machine memory management unit exception
0x02	VMTLBL	Virtual machine address space TLB exception (read data or fetch instruction)
0x03	VMTLBS	Virtual machine address space TLB exception (write data)
0x04	VMMod	Modify the virtual machine address space TLB
0x05	VMTLBRI	Virtual machine address space unreadable exception
0x06	VMTLBXI	The virtual machine address space cannot execute exceptions

7.39 VPID register (CP0 Register 22, Select 2)

The VPID register is a self-defined register of (

Figure 7-39 illustrates the format of the Diag re Diag register Narrate.

Figure **7-41 VPID** register format

31	16	15	8	7	0
0		VPMSK		VPID	

Table **7-44 VPID** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
0	31..16	Read-only constant is 0.	0	0
		Virtual machine number mask, used to control the number of digits in the VPID field of the Diag register to participate in the mapping of virtual and real addresses It also controls the actual number of bits in the high-order virtual address to participate in the virtual-real address mapping The SEGBITS value has been changed.		

VPMSK Legal value	VPID Effective range	Equivalent SEGBITS	Virtual address valid bit range
0x00	0	48	{Vaddr [63:62], VAddr [47: 0]}

		0x80	VPID [7]	47	{Vaddr [63:62], VAddr [46: 0]}		
		0xc0	VPID [7: 6]	46	{Vaddr [63:62], VAddr [45: 0]}		
VPMSK	15..8	0xe0	VPID [7: 5]	45	{Vaddr [63:62], VAddr [44: 0]}	R / W	0x0
		0xf0	VPID [7: 4]	44	{Vaddr [63:62], VAddr [43: 0]}		
		0xf8	VPID [7: 3]	43	{Vaddr [63:62], VAddr [42: 0]}		
		0xfc	VPID [7: 2]	42	{Vaddr [63:62], VAddr [47: 0]}		
		0xfe	VPID [7: 1]	41	{Vaddr [63:62], VAddr [47: 0]}		
		0xff	VPID [7: 0]	40	{Vaddr [63:62], VAddr [47: 0]}		
		If an illegal value is configured for VPMSK, the processor behavior will be unpredictable.					
		This field is only valid when the VMM bit of the Diag register is 1, otherwise the field can be normal					
		Read and write, but do not participate in any other operations.					
		The effective number of virtual machine numbers is controlled by the VPMSK domain, please see the description in the above item.					
VPID	7..0	This field is only valid when the VMM bit of the Diag register is 1, otherwise the field can be normal				R / W	0x0
		Read and write, but do not participate in any other operations.					

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.40 Debug register (CP0 Register 23, Select 0)

The Debug register is a 32-bit readable and writable register. This register contains the most recent EJTAG debug exception and debug mode issue
The cause of the exception is used to control single-ended debugging interrupts. At the same time, this register also controls the resources in the debug mode, indicating the processor
Related internal status. For the description of the Debug register, please refer to the MIPS EJTAG specification.

133

Page 147

7.41 **DEPC** register **(CP0 Register 24, Select 0)**

The DEPC register is a 64-bit readable and writable register, which contains EJTAG debugging exceptions or exception processing in debug mode to continue The PC that started executing the instruction.

In response to an exception, the processor writes to the DEPC register:

PC that directly triggers the exception command.

When the instruction that directly triggers the exception is in the branch delay slot, record the PC of the previous branch or jump instruction of the instruction and Cause.BD And Debug.DBD is set to 1.

Figure 7-42 illustrates the format of the DEPC reg ›es the fields of the DEPC register.

Figure **7-42 DEPC** register format

63 0

DEPC

Table **7-45 DEPC** register field description

Domain name	Bit	Functional description	Read / write	reset value
DEPC	63..0	PC with instructions to continue execution after EJTAG debug exception handling is completed.	R / W	no

134

Page 148

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.42 PerfCnt register (CP0 Register 25, Select 0 ~ 7)

The PerfCnt register is a set of CP0 registers used for processor performance event statistics. Each set of performance counters consists of a pair of Select numbers-Odd adjacent CP0 register structure, namely Select0 ~ 1 constitute PerCnt0, Select2 ~ 3 constitute PerCnt1, Select4 ~ 5 constitute PerCnt2, Select6 ~ 7 constitute PerCnt3. The even number register in each group of performance counters is a control register (PerfCnt Control Reg), which is used to Define the event type and control the counting conditions; the odd number register is the value register (PerfCnt Counter Reg) for recording the number of times. Such as Table 7-46 shows.

Table **7-46 Select** allocation of **PerfCnt** register

Performance counter	**Select** value	**PerfCnt** register
0	Select 0	PerfCnt Control Register 0
	Select 1	PerfCnt Counter Register 0
1	Select 2	PerfCnt Control Register 1
	Select 3	PerfCnt Counter Register 1
2	Select 4	PerfCnt Control Register 2
	Select 5	PerfCnt Counter Register 2
3	Select 6	PerfCnt Control Register 3
	Select 7	PerfCnt Counter Register 3

GS464E implements four sets of performance counters in PerfCnt0 ~ PerfCnt3. The register format definition also follows the MIPS specification. Example; but the difference from the MIPS specification is that the PerfCnt register in GS464E is actually used as the internal performance count of the processor core The reading and writing interface of the device is not the actual counter. In short, the software first configures the event information in the PerfCtrl register to A specific performance counter inside the processor core establishes a corresponding relationship, and then the software actually reads and writes the group of PerfCnt registers. For the specific performance counters specified. This design is to break through the performance events that a single processor can simultaneously count under the MIPS architecture More than 4 limits.

Figure 7-43 illustrates the format of the PerfCnt Control regis the fields of the PerfCnt Control register.

Figure 7-44 illustrates the format of the PerfCnt Counter regi ; the fields of the PerfCnt Counter register.

Figure **7-43 PerfCnt Control** register format

31	30	29	28	27	26	25	twenty-four	twenty-three	twenty-two	twenty-one	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MW				0			EC																							

Table **7-47 PerfCnt Control** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
M	31	Set to 1 indicates that the next set of performance counters is implemented, otherwise it is not implemented	R	0x1 / 0x0 1
W	30	Constant is 1, indicating that the bit width of the PerfCnt Counter register is 64 bits.	R	0x1
0	29..25	Read-only constant is 0.	0	0

1 For Control 0 ~ Control 2, this bit reset value is 1; for Control3, this bit reset value is 0.

Page 149

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Domain name	Bit	Functional description	Read / write	reset value
EC	24..23	Event category. 0: root state event, refers to the event that occurs when GuestCtl0.GM = 0. 1: Root state intervention event, means GuestCtl0.GM = 1 and! (Root.Status.EXL = 0 and Root.Status.ERL = 0 and Root.Debug.DM = 0). 2: Guest event refers to GuestCtl0.GM = 1 and Root.Status.EXL = 0 and Root.Status.ERL = 0 and Event that occurs when Root.Debug.DM = 0.	R / W	0x0

		3: Guest event + root intervention event. Refers to the event that occurs when GuestCtl0.GM = 1		
0	22.15	Read-only constant is 0.	0	0
Event	14..5	Event number.	R / W	0x0
IE	4	The performance counter overflow interrupt is enabled. 0: Disable the performance counter from triggering the overflow interrupt. 1: Allow the performance counter to trigger an overflow interrupt.	R / W	0x0
U	3	Event record enable bit in user mode. 0: prohibit recording; 1: allow recording.	R / W	0x0
S	2	Event recording enable bit in supervision mode. 0: prohibit recording; 1: allow recording.	R / W	0x0
K	1	Event recording enable bit in core mode. 0: prohibit recording; 1: allow recording.	R / W	0x0
EXL	0	In the case of Status.EXL = 1 and Status.ERL = 0, the event record enable bit. 0: prohibit recording; 1: allow recording.	R / W	0x0

Figure **7-44 PerfCnt Counter** register format

Table **7-48 PerfCnt Counter** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
Event Count	63..0	Performance event counter. Whenever the event defined in the PerfCnt Control register in the same group is triggered, the counter value plus 1. When the highest bit of the counter is 1, set the PCI bit of the Cause register to 1. Although the W bit of PerfCnt Control is always defined as 1, every bit in the 64-bit wide Event Count field can be read and written, But the actual counting range of GS464E does not exceed 48 digits. So when reading Event Count, it is the actual count The sign of the 48-bit value of the register is expanded to a 64-bit result; when the timer value is reset, only the low of Event Count Only 48 bits can be written.	R / W	0x0

7.43 **ErrCtl** register **(CP0 Register 26, Select 0)**

The ErrCtl register is a software readable and writable register, which is used as the Index Load Tag and Index Store Tag class CACHE instruction and The interactive interface of Parity / ECC check value of Tag part data of Cache at all levels, and also serves as Index Load Data and Index Store Data The interactive interface between the CACHE-like inst

Figure 7-45 illustrates the format of the ErrCtl re

description.

Figure **7-45 ErrCtl** register format

Table **7-49 ErrCtl** Register Field Description

Domain name	Bit	Functional description	Read / write	reset value
0	31..8	Read-only constant is 0.	0	0
ECC	7..0	The content of the ECC check value of the Tag or Data to be written or read.	R / W	no

137

Page 151

7.44 **CacheErr** register **(CP0 Register 27, Select 0)**

The CacheErr register records the diagnostic information when I-Cache's Parity check and D-Cache's ECC check are in error. GS464E for V-Cache and S-Cache also perform "correct 1 check 2" ECC check, but do not store check error information in the CacheErr register.

There is a difference in the format of the CacheErr register when record ure 7-46 illustrates The format of the CacheErr register used for I-Cache check error information case Each domain is described.

Figure **7-46 The format of the CacheErr** register used for **I-Cache** check error information

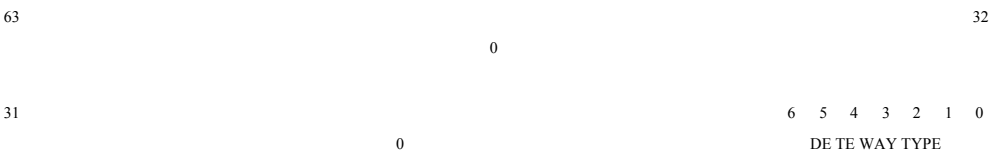


Table **7-50** Field description when the **CacheErr** register is used for **I-Cache** check error information

Domain name	Bit	Functional description	Read / write	reset value
0	63..6	Read-only constant is 0.	0	0
DE	5	I-Cache Data part parity error flag. 1: check error; 0: check error-free.	R	0
TE	4	I-Cache Tag part parity error identification. 1: check error; 0: check error-free.	R	0
WAY	3..2	The record verification error is on the way.	R	0
TYPE	1..0	Verify the error type. 0: I-Cache; 1: D-Cache; 2, 3: reserved	R	0

Figure 7-47 illustrates the format of the CacheErr register used for I-Cache check error in ıters the register fields in this case Described.

Figure **7-47 The format of the CacheErr** register used for **D-Cache** check error information

W6 E5 F5 W5 E4 F4 W4 E3 F3 W3 E2 F2 W2 E1 F1 W1 E0 F0 W0 ET FT WT TYPE

Table **7-51** Field description when the **CacheErr** register is used for **D-Cache** check error information

Domain name	Bit	Functional description	Read / write	reset value
0	63..38	The read-only constant is 0.	0	0
E7	37	D-Cache Data Bank 7 checks the error flag. 1: Error; 0: No error.	R	0
F7	36	D-Cache Data Bank 7 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W7	35..34	D-Cache Data Bank 7 is on the wrong way.	R	0
E6	33	D-Cache Data Bank 6 Check the error flag. 1: Error; 0: No error.	R	0

Domain name	Bit	Functional description	Read / write	reset value
F6	32	D-Cache Data Bank 6 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W6	31..30	D-Cache Data Bank 6 is on the wrong way.	R	0
E5	29	D-Cache Data Bank 5 Check the error flag. 1: Error; 0: No error.	R	0
F5	28	D-Cache Data Bank 5 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W5	27..26	D-Cache Data Bank 5 is on the wrong way.	R	0
E4	25	D-Cache Data Bank 4 Check the error flag. 1: Error; 0: No error.	R	0
F4	twenty four	D-Cache Data Bank 4 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W4	23..22	D-Cache Data Bank 4 is on the wrong way.	R	0
E3	twenty one	D-Cache Data Bank 3 Check the error flag. 1: Error; 0: No error.	R	0
F3	20	D-Cache Data Bank 3 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W3	19..18	D-Cache Data Bank 3 is on the wrong way.	R	0
E2	17	D-Cache Data Bank 2 verifies the error flag. 1: Error; 0: No error.	R	0
F2	16	D-Cache Data Bank 2 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W2	15..14	D-Cache Data Bank 2 is on the wrong way.	R	0
E1	13	D-Cache Data Bank 1 Check the error flag. 1: Error; 0: No error.	R	0
F1	12	D-Cache Data Bank 1 verifies 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W1	11..10	D-Cache Data Bank 1 is on the wrong way.	R	0
E0	9	D-Cache Data Bank 0 verification error flag. 1: Error; 0: No error.	R	0
F0	8	D-Cache Data Bank 0 checks out 2 or more error flags. 1: There is such an error; 0: There is no such error.	R	0
W0	7..6	D-Cache Data Bank 0 is on the wrong way.	R	0
ET	5	D-Cache Tag check error flag. 1: Error; 0: No error.	R	0
FT	4	D-Cache Tag verifies 2 or more wrong identifications. 1: There is such an error; 0: There is no such error.	R	0
WT	3..2	D-Cache Tag is on the wrong way.	R	0
TYPE	1..0	Verify the error type. 0: I-Cache; 1: D-Cache; 2, 3: reserved	R	0

Page 153

7.45 CacheErr1 register (CP0 Register 27, Select 1)

The CacheErr1 register is used to record the PC value of the instruction that detects an I-Cache check error, and also used to record the D-Cache check
The physical address of the erroneous access. It should be noted that for the I-Cache check error, the Cache block in error is not necessarily the CacheErr1 register
The Cache block where the PC value is located; for D-Cache checksum errors, the Cache block in error is not necessarily the physical address in the CacheErr1 register
In the block. The information stored in the CacheErr1 register can extract the I-Cache / D-Cache Index value and I-Cache Bank range of the error,
Combined with the information in the CacheErr register, the

Figure 7-48 illustrates the format of the CacheErr1 regi
The domain is described.

Figure **7-48 CacheErr1** register format

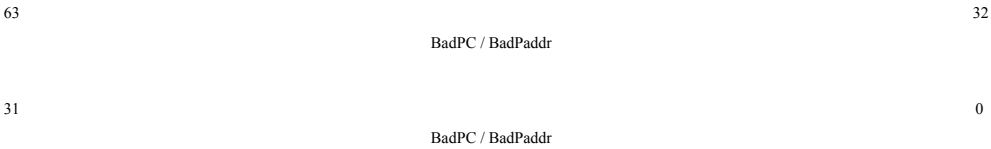


Table **7-52 CacheErr1** register domain description

Domain name	Bit	Functional description	Read / write	reset value
BadPC	63..0	Check the PC value of the instruction with I-Cache check error.	R	0
BadPaddr	63..0	Check out the physical address of the D-Cache verification error access.	R	0

Page 154

7.46 TagLo register (CP0 Register28, Select 0)

The TagLo register is a software readable and writable register, which together with the TagHi register serves as Index Load Tag and Index Store Tag
The CACHE-like instruction interacts with the Tag part of the Cache at all levels, and also serves as Index Load Data and Index Store Data
The interactive interface between the CACHE-like instruction and the Data part of the Cache at all levels.

When TagLo is used to access diffi gLo register for access

I-Cache Tag format ; Table 7-53 Table 7

Figure

Table **7-53** Description of the fields when the **TagLo** register is used to access the **I-Cache Tag**

Domain name	Bit	Functional description	Read / write	reset value
TL	31..12	The low-order content of the tag to be written or read corresponds to [31:12] of the physical address.	R / W	no
X	11..7	Can write and read normally, but does not participate in other operations.	R / W	no
V	6	The status of the cache block to be written or read. 0: Cache block is invalid; 1: Cache block is valid.	R / W	no
X	5..4	Can write and read normally, but does not participate in other operations.	R / W	no
SCWAY	3..0	The number of Cache blocks to be written or read in Scache.	R / W	no

Figure 7-50 illustrates the format when the TagLo register is used to access the D-C rforms the fields of the register in this case Description.

Figure **7-50** The format when the **TagLo** register is used to access the **D-Cache Tag**

Table **7-54** Field Description of **TagLo** Register Used to Access **D-Cache Tag**

Domain name	Bit	Functional description	Read / write	reset value
TL	31..12	The low-order content of the tag to be written or read corresponds to [31:12] of the physical address.	R / W	no
X	11..9	Can write and read normally, but does not participate in other operations.	R / W	no
W	8	The "dirty" mark of the Cache block to be written or read. 1: dirty block; 0: non-dirty block.	R / W	no
CS	7..6	The status of the cache block to be written or read. 0: Invalid block; 1: Shared block; 2: Exclusive block; 3: Reserved, if the processor result is uncertain.	R / W	no
X	5..4	Can write and read normally, but does not participate in other operations.	R / W	no
SCWAY	3..0	The number of Cache blocks to be written or read in Scache.	R / W	no

Figure 7-51 illustrates the format when the TagLo register is used to access the V-C rforms the register fields in this case

Description.

Figure **7-51 TagLo** register format used to access **V-Cache Tag**

Table **7-55** Field Description of **TagLo** Register Used to Access **V-Cache Tag**

Domain name	Bit	Functional description	Read / write	reset value
TL	31..12	The low-order content of the tag to be written or read corresponds to [31:12] of the physical address.	R / W	no
X	11..10	Can write and read normally, but does not participate in other operations.	R / W	no
I	9	The instruction / data attributes of the Cache block to be written or read. 1: instruction block; 0: data block	R / W	no
W	8	The "dirty" mark of the Cache block to be written or read. 1: dirty block; 0: non-dirty block.	R / W	no
CS	7..6	The status of the cache block to be written or read. 0: Invalid block; 1: Shared block; 2: Exclusive block; 3: Reserved, if the processor result is uncertain.	R / W	no
X	5..4	Can write and read normally, but does not participate in other operations.	R / W	no
SCWAY	3..0	The number of Cache blocks to be written or read in Scache.	R / W	no

Figure 7-52 illustrates the format when the TagLo register is used to access the S describes the fields of the register in this case description.

Figure **7-52 TagLo** register format used to access **S-Cache Tag**

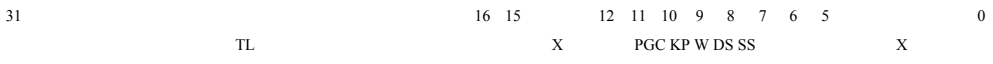


Table **7-56** Field Description of **TagLo** Register Used to Access **S-Cache Tag**

Domain name	Bit	Functional description	Read / write	reset value
TL	31..16	The low content of the tag to be written or read corresponds to [31:16] of the physical address.	R / W	no
X	15..12	Can write and read normally, but does not participate in other operations.	R / W	no
PGC	11..10	The PageColor bit corresponding to the Cache block to be written or read. When the system uses 4KB basic page size, two Bits are meaningful. When the system uses the 8KB basic page size, only the 13th bit is meaningful. About PageColor Bit detailed description, see 5. 4.5 of 5 Day.	R / W	no
KP	9	When writing, set to 0 to clear the directory entry corresponding to the Cache block to be written, and set to 1 to indicate The content of the directory entry written to the Cache block remains unchanged. During the read operation, the content of this field is meaningless.	R / W	no
W	8	The "dirty" mark of the Cache block to be written or read. 1: dirty block; 0: non-dirty block.	R / W	no
DS	7	The status of the directory entry corresponding to the Cache block to be written or read. 1: The directory is dirty; 0: The directory is clean.	R / W	no
SS	6	The status of the Cache block to be written or read. 1: valid block; 0: invalid block.	R / W	no
X	5..0	Can write and read normally, but does not participate in other operations.	R / W	no

Figure 7-53 illustrates the format when the TagLo register is used to access the Data portion of le 7-57 registers for this case Each domain is described.

142

Page 156

Figure **7-53** The format of the **TagLo** register used to access **Cache Data** at all levels

Table **7-57** Description of the fields when the **TagLo** register is used to access **Cache Data** at all levels

Domain name	Bit	Functional description	Read / write	reset value
DATA	31..0	The lower 32 bits of the Data Bank in the Cache block to be written or read.	R / W	no

Programming Tip: When the software uses Index Store Tag and Index Store Data instructions to fill Cache content, please keep Cache at all levels The content filled in meets the cache consistency requirements of GS464E. Otherwise, the processor's behavior will be undefined.

143

Page 157

7.47 **DataLo** Register **(CP0 Register 28, Select 1)**

In GS464E, the DataLo and DataHi registers are not used as interactive interfaces to access the Data portion of Cache at all levels, only when Index Load Data and Index Store Data CACHE instructions access the I-Cache as an interactive interface for the I-Cache pre-decoding data part. In other cases, DataLo allows software to read and write, but does not participate in

Figure 7-54 illustrates the format when the DataLo register is used to access tl
In this case, the register fields are described.

Figure **7-54 DataLo** register format for **I-Cache** access

Table **7-58** Field Description of **DataLo** Register Used for **I-Cache** Access

Domain name	Bit	Functional description	Read / write	reset value
ITYPE	31..0	The [31: 0] bits of the pre-decoding information of the I-Cache block to be written or read.	R / W	no

Programming Tip: When the software uses Index Store Tag and Index Store Data instructions to fill Cache content, please keep Cache at all levels The content filled in meets the cache consistency requirements of GS464E. Otherwise, the processor's behavior will be undefined.

144

Page 158

Loongson 3A3000 / 3B3000 Processor User Manual • Next

7.48 TagHi Register (CP0 Register 29, Select 0)

The TagHi register is a software readable and writable register, which together with the TagLo register serves as Index Load Tag and Index Store Tag CACHE instruction and the interactive interface of the Tag part data of each level of Cache, also used as Index Load Data and Index Store Data The interactive interface of the CACHE instruction and the Data part of the Cache at all le

Figure 7-55 illustrates the format when the TagHi register is used to access various l le 7-59 performs the register fields in this case Description.

FIG **7-55 TagHi** register access levels **Cache Tag** format when

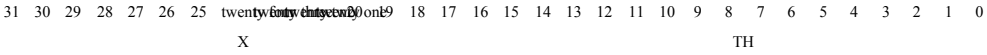


Table **7-59** Field Description of **TagHi** Register Used to Access **Cache Tags** at All Levels

Domain name	Bit	Functional description	Read / write	reset value
X	31..12	can write and read normally, but does not participate in other operations.	R / W	no
TH	15..0	The high content of the tag to be written or read corresponds to the physical address [47:32].	R / W	no

Figure 7-56 illustrates the format when the TagLo register is used to access Cache Dat) enters the register fields in this case Described.

FIG **7-56 TagHi** register access levels **Cache Data** format when

Table **7-60** Field description when **TagHi** register is used to access various levels of **Cache Data**

Domain name	Bit	Functional description	Read / write	reset value
DATA	31..0	The upper 32 bits of the Data Bank in the Cache block to be written or read.	R / W	no

Programming Tip: When the software uses Index Store Tag and Index Store Data instructions to fill Cache content, please keep Cache at all levels The content filled in meets the cache consistency requirements of GS464E. Otherwise, the processor's behavior will be undefined.

145

Page 159

Loongson 3A3000 / 3B3000 Processor User Manual • Next

7.49 DataHi Register (CP0 Register 29, Select 1)

In GS464E, the DataLo and DataHi registers are not used as interactive interfaces to access the Data portion of Cache at all levels, only when Index

Load Data and Index Store Data CACHE instructions access the I-Cache as an interactive interface for the I-Cache pre-decoding data part. In other cases, DataHi allows the software to read and write, but does not particip

Figure 7-57 illustrates the format of the DataHi register used to access the I-

In this case, the register fields are described.

Figure **7-57 DataHi** register format used for **I-Cache** access

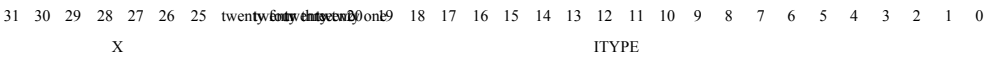


Table **7-61** Field description when the **DataHi** register is used for **I-Cache** access

Domain name	Bit	Functional description	Read / write	reset value
X	31..24	can write and read normally, but does not participate in other operations.	R / W	no
ITYPE	23.0	The [55:32] bits of the pre-decoding information of the I-Cache block to be written or read.	R / W	no

Programming Tip: When the software uses Index Store Tag and Index Store Data instructions to fill Cache content, please keep Cache at all levels The content filled in meets the cache consistency requirements of GS464E. Otherwise, the processor's behavior will be undefined.

146

Page 160

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.50 ErrorEPC register (CP0 Register 30, Select 0)

The ErrorEPC register is a 64-bit readable and writable register. Its function is similar to the EPC register, except that it is only used to store cold reset, Soft reset, non-maskable interrupt, and Cache error exceptions are processed after the completion of the instruction PC, and the ErrorEPC register is not Identify the branch delay slot (Cause.BD) in the EPC register.

In response to the above exception, the processor hardware writes to the ErrorEPC register:

PC that directly triggers the exception command.

When the instruction that directly triggers the exceptio slot, record the PC of the previous branch or jump instruction of the instruction.

Figure 7-58 illustrates the format of the ErrorEPC reg es the fields of the ErrorEPC register.

Figure **7-58 ErrorEPC** register format

Table **7-62 ErrorEPC** register field description

Domain name	Bit	Functional description	Read / write	reset value
ErrorEPC	63..0	PC that continues to execute instructions after the exception processing is completed.	R / W	no

7.51 DESAVE register (CP0 Register 31, Select 0)

The DESAVE register is a 64-bit readable and writable register used to temporarily store data for debugging exception handlers. Usually debug exception handlers Use the DESAVE register to save a general register, and then use the general register as the base register of the fetch instruction to save the current Text to a specified area, such as the dmseg segment.

The core state software does not allow the use of the I

Figure 7-59 illustrates the format of the DESAVE reg es the fields of the DESAVE register.

Figure **7-59 DESAVE** register format

Table **7-63 DESAVE** register field description

Domain name	Bit	Functional description	Read / write	reset value
Data	63..0	Debug the data temporarily stored in the exception handler.	R / W	no

148

Page 162

Loongson **3A3000 / 3B3000** Processor User Manual • Next

7.52 KScratch1 ~ 6 registers (CP0 Register 31, Select 2 ~ 7)

The KScratch1 ~ 6 registers are a group of 64-bit readable and writable registers, which are used to store the temporary storage data of the core state software.

The software in Debug Mode cannot access KScratch1 ~ 6, but the software can temporarily access the DESAVE register

Save the data.

Figure 7-60 illustrates the format of the KScratch regi es the fields of the KScratch register.

Figure **7-60 KScratch *n*** register format

Table **7-64 KScratch *n*** register domain description

Domain name	Bit	Functional description	Read / write	reset value
Data	63..0	Core state software temporarily stores data.	R / W	no

149

Page 163

8 processor performance analysis and optimization

GS464E's performance analysis and optimization work is mainly done through hardware integrated performance counters. Performance counters are used to count the internal processor The number of occurrences of certain events is an important support for post-silicon performance verification, compiler optimization, and software performance tuning. In addition, due to this The statistical significance of some events recorded by these performance counters is related to the power consumption of the chip during operation. The performance counter can also be used Power management.

The performance counter used by GS464E includes two parts: processor core performance counter and shared cache performance counter.

8.1 Organization form and access method of performance counter

8.1.1 Processor core performance counter

Each GS464E processor core implements 28 sets of performance counters, which are divided into FETCH module, RMAP module, ROQ module, FIX Module, FLOAT module, MEMORY module and CACHE2MEM module, each module contains 4 sets of performance counters. Performance per group The counter corresponds to two physical registers, one for counting (48-bit counter) and one for controlling counting. Hardware count inside each module The device can only be used to count events related to this module. Any group of performance counters inside the module can count any performance related to the module Count events.

The performance counter of the GS464E processor core is still available for software configuration and access through the CP0 Performance Counter register under the MIPS architectur Q, but the form of access is slightly different from the traditional MIPS processor. Specifically, the Performance Counter register in CP0 is only As a configuration and access interface, the specific hardware counter to which the read and write operations are passed is registered by the Performance Counter The event number configured in the device is dynamically established. The software first configures the odd number Performance Counter register on an event, the processor The internal hardware establishes a one-to-one mapping relationship between the event and the hardware counter in the module to which it belongs; after the mapping relationship is establishe Reading and writing the Performance Counter register will directly operate on the mapped hardware counter; eventually, the software starts the hardware counter Start statistics. In particular, the Performance Counter needs to be initialized in all modules before each configuration. Turned into an invalid state. The specific method is to configure a maximum event number within the allowable range of the module event in each module, and set the counter value Set to 0.

8.1.2 Shared cache performance counter

Each GS464E shared cache volume implements 4 sets of 48-bit performance counters. Each group of performance counters corresponds to two physical registers, one One is for counting (48-bit counter), and one is for controlling counting. Each performance counter can count any occurrences in the shared cache Can count events.

The performance counters of the four shared buffers in Loongson 3A3000 are accessed through the chip's confbus. In four shared caches A total of 16 sets of performance registers use a uniform confbus base address of 0x3FF0.0000. The specific offset of each register is shown in Table 8-1 :

Table **8-1** Shared cache performance counter register address offset

Register name	**Scache0**	**Scache1**	**Scache2**	**Scache3**
PerfCtl0	0x0800	0x0900	0x0a00	0x0b00
PerfCnt0	0x0808	0x0908	0x0a08	0x0b08
PerfCtl1	0x0810	0x0910	0x0a10	0x0b10
PerfCnt1	0x0818	0x0918	0x0a18	0x0b18
PerfCtl2	0x0820	0x0920	0x0a20	0x0b20

150

Page 164

Register name	**Scache0**	**Scache1**	**Scache2**	**Scache3**
PerfCnt2	0x0828	0x0928	0x0a28	0x0b28

PerfCtl3	0x0830	0x0930	0x0a30	0x0b30
PerfCnt3	0x0838	0x0938	0x0a38	0x0b38

Examples of use

If you want to count the total number of scread and dmaread received by scache0, you can do the following:

- Step1: write sc0_perfcnt0 as 0 (sd $ 0, 0x3ff00808), sc0_perfcnt1 as 0 (sd $ 0, 0x3ff00818), sc0_perfcnt2 is written as 0 (sd $ 0, 0x3ff00828)
- Step2: Write sc0_perfctrl0 as 1 (sd value_1,0x3ff00800), sc_perfctrl1 as 9 (sd value_9, 0x3ff00810), sc_perftrl2 is written as 10 (sd value_10, 0x3ff00820)
- Step3: execute the program
- Step4: Read sc0_perfcnt0 (ld t0, 0x3ff00808), sc0_perfcnt1 (ld t1, 0x3ff00818), sc0_perfcnt2 (ld t2, 0x3ff00828); where the result of t0 is the total number of scread, and the result of t1 + t2 is the total number of dmaread.

8.2 Processor performance count events

The performance events defined by GS464E are divided into three categories:

The first type is used to analyze the characteristics of the program at the instruction set level. Specifically, the statistics of the different types of instructions in the pipeline submission sta Number, and thus the distribution of the types of instructions dynamically executed by the program.

The second category is used to analyze the performance bottlenecks reflected in the interaction between the program code and the processor microstructure. The starting point is to optim 化 程序。 The process. Mainly through statistics of various events leading to pipeline blockage. In addition to Cache misses, queue full times, and branch predictions
In addition to basic events such as the wrong number of times, GS464E also added a batch of statistics on the number of delay cycles, such as clearing the front-end pipeline guide after the bra The number of cycles leading to regmap flow-level interruption, etc.

The third category is used to accumulate data for design space exploration. The starting point is to optimize the microstructure. For example, the current dual-access memory component Can be implemented with dual-port RAM, which is much more expensive than single-port RAM, so dcache RAM is added for two load operations on the same shoot
Statistics of the number of conflicts. In the existing structure, such conflicts will not cause pipeline blocking, so it does not affect the performance of the program.

8.2.1 Definition of processor core performance count events

The performance counter events of GS464E processor core are defined as shown in Table 8-2 .

Table **8-2** Definition of processor core performance counter events

Event number	Event description
	FETCH module
1	Inst Queue full empty number of cycles
2	Number of instructions written to Inst Queue per cycle
3	The number of front-end pipeline blocking cycles (the number of instructions entering the Inst Queue is equal to 0)
4	The number of instructions entering the Inst Queue is equal to 1
5	The number of instructions entering the Inst Queue is equal to 2
6	The number of instructions entering the Inst Queue is equal to 3
7	The number of instructions entering the Inst Queue is equal to 4
8	The number of instructions entering the Inst Queue is equal to 5

Event number	Event description
9	The number of instructions entering the Inst Queue is equal to 6
10	The number of instructions entering the Inst Queue is equal to 7
11	The number of instructions entering the Inst Queue is equal to 8
12	In the cache space, the number of instructions entering the InstQueue is less than 8 because of crossing the boundary of the cacheline
13	Number of blocked front-end pipeline cycles due to full Inst Queue
14	Decoding instructions per cycle
15	Number of decoding instructions from Loop Buffer per cycle
16	Number of loops to fetch instructions from loop buffer
17	The number of Loops identified (including those that can be put into Loop Buffer and those that cannot be put into Loop Buffer)
18	The number of branch instructions decoded per cycle is equal to 0
19	The number of branch instructions decoded per cycle is equal to 1
20	The number of branch instructions decoded per cycle is equal to 2
twenty one	Blocked front-end pipeline due to Icache Miss
twenty two	BrBTB failed to predict the front-end pipeline blockage caused by the taken branch

twenty four	The number of Icache misses initiated by the Icahe module and received by missq
26	ITLB miss but the number of hits in TLB
27	Number of times ITLB was flushed
	RMAP module
64	Resource allocation is blocked
65	GR rename resource full blocking
66	GR Rename resource full false block
67	FR Rename resource full blocking
68	FR rename resource full false block
69	FCR rename resource full blocking
70	FCR rename resource full false blocking (no FCR renaming resource required for entry instruction)
71	ACC rename resource full blocking
72	ACC rename resource full false blocking (no ACC rename resource required for entry instruction)
73	DSPCtrl renamed resource full block
74	DSPCtrl rename resource full false block (no DSPCtrl rename resource is required for the entry command)
75	BRQ full blocking
76	BRQ full false blocking (no need to enter BRQ for pending instruction)
77	FXQ full blocking
78	FXQ full false blocking (no need to enter FXQ for entry command)
79	FTQ full blocking
80	FTQ full false blocking (no need to enter the FTQ for the entry instruction)
81	MMQ full blocking
82	MMQ full false blocking (no need to enter MMQ for the entry command)
83	CP0Q full blocking
84	CP0Q full false blocking (no need to enter CP0Q for the entry instruction)
85	ROQ full blocking
86	Number of NOP instructions that completed the resource allocation phase

Event number	Event description
87	Number of operations transmitted from regmap to each transmission queue per cycle
88	Exception (excluding branch misprediction) the overhead of clearing the pipeline (after the regmap pipeline is cleared by exception until the first instruction reaches the regmap pipe
89	Branch prediction error and empty pipeline overhead
	ROQ module
128	Internal pipeline clock
129	Number of instructions submitted per cycle
130	ALU operations submitted
131	FALU operation submitted
132	Submitted Memory / CP0 / fixed floating point swap operation
133	Submitted load operation
134	Submitted store operation
135	Submitted LL operations
136	Submitted SC operations
137	Submitted unaligned load operation
138	Unaligned store operations submitted
139	All exceptions and interruptions
140	Number of interruptions
141	From ROQ receiving interrupt signal to generating interrupt exception
142	From ROQ received interrupt signal to the first instruction of interrupt exception handler to enter ROQ
143	Virtual machine exceptions
144	Number of wrong address exceptions
145	Number of exceptions related to TLB
146	TLB refill exceptions
147	TLB refill exception processing time (from the beginning of TLB refill exception clearing pipeline to the ERET return of TLB refill exception)
148	Branch instructions submitted by brq
149	Jump register branch instruction submitted by brq

150	Jump and link branch instructions submitted by brq
151	Branch and link branch instructions submitted by brq
152	bht branch instruction submitted by brq
153	Like branch instruction submitted by brq
154	Not taken branch instruction submitted by brq
155	Take branch instruction submitted by brq
156	brq commits the wrong predicted branch instruction
157	Jump register branch instruction with wrong prediction submitted by brq
158	Jump and link branch instruction with wrong prediction submitted by brq
159	brq submits the wrong predicted branch and link branch instruction
160	brq commits a wrong predicted bht branch instruction
161	Likely branch instruction with wrong prediction submitted by brq
162	brq submitted incorrectly predicted not taken branch instructions
163	Branch instruction submitted by brq with wrong prediction
	FIX module

153

Page 167

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Event number	Event description
192	fxq no emission
193	fxq emission execution operand
194	fxq is transmitted to the number of operations performed by the FU0 function
195	fxq is transmitted to the number of operations performed by the FU1 function
196	The fixed-point multiplication part in FU0 is in the execution state
197	The fixed-point division component in FU0 is in the execution state
198	The fixed-point multiplication part in FU1 is in the execution state
199	The fixed-point division component in FU1 is in the execution state
	FLOAT module
256	ftq no launch
257	ftq launch execution operand
258	ftq is transmitted to the number of operations performed by the FU3 function
259	ftq The number of operations performed by the FU4 feature
260	FU3 is idle, FU4 is full, but ftq has only FU4 to be launched
261	FU4 is idle, FU3 is full, but ftq has only FU3 to be launched
262	Emit scalar floating point operands per cycle
263	The number of 64-bit multimedia acceleration instructions issued per cycle (the instruction name includes the "GS" prefix)
264	The number of 64-bit multimedia acceleration instructions issued per cycle (the instruction name does not include the "GS" prefix)
272	Floating point division / prescription part in FU3 is in execution state
274	Floating point division / prescription component in FU4 is in execution state
	MEMORY module
320	mmq no emission
321	mmq launch execution operand
322	In mmq, FU2 command is sent per beat
323	In mmq, FU5 command is sent per shot
324	load launches
325	store launches
326	The source operand has at least one floating point fetch instruction number
327	The number of launches for instructions with both fixed-point and floating-point operands
329	wait_first number of blocked cycles
330	Number of cycles blocked by SYNC operation
331	stall_issue Number of blocked cycles
332	Software prefetch operation launch
333	The number of cycles of newly launched memory access operations that block store operations writing to dcache
334	Bank conflict in two loads on the same shoot
337	The number of times dcachewrite0 and 1 are valid at the same time
338	Number of successfully executed SC instructions
339	Store instruction dcahe miss times (including miss and non-EXC state)
340	The number of dcache misses caused by the dcache shared state of the store instruction

CACHE2MEM module

341	Store instruction dcache hit count

Event number	Event description
342	load hits
343	fwdbus2 times
344	fwdbus5 times
345	fwdbus total times, fwdbus2 + fwdbus5
346	Number of rollbacks for memory access operations due to load and store address conflicts (dwaitstore)
347	Exceptions due to load and store address conflicts (mispec)
348	The number of cp0qhead rollbacks due to unsuccessful dcachewrite
349	cp0q number of dmemread requests
350	cp0q Duncache requests
351	resbus2 occupies resbus5 times, there are two dest memory access operations for LQ, LQC1, etc.
352	Number of software prefetch hits on L1 Dcache
353	Store software prefetch in L1 dcache hits
354	Store software prefetches in L1 dcache miss times
355	Load software prefetch hits on L1 dcache
356	Load software prefetches in L1 dcache miss times
357	Store software prefetches the number of misses in L1 dcache due to share state
358	specfwdbus2 times
359	specfwdbus5 times
360	specfwdbus times: specfwdbus2 + specfwdbus5
384	Number of data load requests to access vcache
385	Number of data store requests to access vcache
386	Number of data requests to access vcache
387	Instruction requests to access vcache times
388	vcache visits
389	The number of times software prefetches access vcache
390	vcache load hits
391	vcache store hits
392	vcache data hits
393	vcache command hit count
394	vcache hits
395	vcache software configuration prefetch hits
396	vcache load failure times
397	vcache store failure times
398	Number of vcache data failures
399	vcache instruction failure times
400	vcache failure times
401	vcache software configuration prefetch failure times
402	vcache is invalidated by extreq operation the number of valid blocks
403	vcache The number of valid blocks degraded by wtbk operations
404	vcache is invalidated by INV operation the number of valid blocks
405	vcache is invalidated by INVWTBK the number of valid blocks

Event number	Event description
406	Number of read requests from the processor core to the external bus
407	Number of write requests from the processor core to the external bus
408	Number of bus write requests with write data
409	Bus read request is blocked due to conflict with bus write request address
410	Number of WTBK requests processed by missq
411	Number of INVWTBK requests processed by missq
412	Number of INV requests processed by missq
413	Number of INV class (3 above) requests processed by missq
414	The total number of refills (including exreq and replace + refill)
415	The total number of refills for icache
416	The total number of refills for dcache
417	The number of refill (replace + refill)
418	The number of times to refill a dcache shared block
419	The number of times to refill a dcache exc block
420	The total number of refill data (replace + refill)
421	The total number of refill instructions (replace + refill)
422	The number of times dcache replaces a valid block
423	The number of times dcache replaces a shared block
424	The number of times that dcache replaces an exc block
425	The number of times that dcache replaces a dirty block
426	icache replaces the number of valid data
427	vcache replacement times
428	The number of times vcache replaces a useful block
429	The number of times vcache replaces a shared block
430	The number of times vcache replaces an exc block
431	The number of times vcache replaces a dirty block
432	The number of times vcache replaces useful dc blocks
433	The number of times vcache replaces useful ic blocks
434	Accumulate the number of load requests that are not returned from scache per shot (missq has only 15 items at most for processing scache requests)
435	Accumulate the number of store requests not returned from scache per beat
436	Accumulate the number of fetch requests not returned from scache per beat
437	The total number of sc reads sent
438	The total number of loads in the sent scread
439	The total number of stores in the sent scread
440	Total number of scread data access
441	The total number of scread instruction access
442	scread total number of non-prefetches
443	scread total number of non-prefetched data loads
444	scread total number of non-prefetched data stores
445	scread total number of non-prefetch data access
446	scread non-prefetch refers to the total number of visits

156

Page 170

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Event number	Event description
447	The total number of prefetches in the sent scread
448	The number of load prefetches in the sent scread
449	The number of store prefetches in the sent scread
450	The total number of scread prefetch data access
451	scread prefetch instruction access total number
452	The number of software prefetch requests processed by missq
453	The number of scwrites issued by missq
454	The number of scwrites initiated by missq due to the replace operation
455	The number of RESP class scwrite issued by missq due to invalid operation
456	missq The scwrite operation initiated by the replace operation, and replace the valid block
457	missq really accepts the number of requests miss_en
458	missq really accepts load requests miss_en

459	missq really accepts store request number miss_en
460	missq The number of data accesses actually accepted
461	missq The number of instruction accesses actually accepted
462	Missq occupancy (missq is not empty)
463	Missq general access account
464	missq fetch access account
465	Missq external request account
466	Missq prefetch request account items
467	missq accounted for the number of beats (missq has the number of valid entries, ie missq is not empty time)
468	missq common access accounted for the number of beats
469	missq refers to the number of shots accessed
470	missq external requests accounted for the number of beats
471	missq prefetch requests accounted for the number of entries
472	Missq full count (missq cannot accept ordinary access, missq valid item is not less than 15)
473	The number of times load requests encounter prefetches in missq
474	The number of times the load request encountered pre_scref in missq
475	The number of times load request encounters prefetch pre_wait in missq
476	The number of times load request encounters prefetch pre_rdy in missq
477	Store request encounters prefetch pre_scref in missq and the number of load operations
478	Store request encountered prefetching pre_rdy and state = shard times in missq
479	Store request encountered prefetch pre_wait in missq and load operation times
480	Store request encounters prefetch pre_scref in missq and the number of store operations
481	Store request encountered prefetch pre_rdy and state = exc times in missq
482	Store request encountered prefetch pre_wait in missq and the number of store operations
483	The number of times store requests encounter prefetches in missq (including hits in store prefetches and hits in load prefetches)
484	The number of times store requests encountered valid prefetches in missq (hits store prefetches)
485	The number of times all requests encounter prefetch in missq (load + store)
486	The number of times all requests encounter pre_scref prefetch in missq (laod + store)
487	The number of times all requests encounter pre_rdy prefetching in missq (load + store)

157

Page 171

Loongson **3A3000 / 3B3000** Processor User Manual • Next

Event number	Event description
488	The number of times all requests encounter pre_wait in fetch (load + store)
489	The number of times fetch requests encounter prefetches in missq
490	The number of times the fetch request encounters pre_scref in missq
491	The number of times fetch requests encounter pre_rdy prefetches in missq
492	The number of times the fetch request meets pre_wait in missq
495	The number of times data and fetch instructions encountered pre_rdy prefetch in missq
496	The number of times data and fetch instructions encountered pre_wait in missq
497	Number of times the hardware load prefetch request was canceled by Scache 15
498	Number of times the hardware store prefetch request was canceled by Scache
499	The number of times the hardware data access prefetch request was canceled by Scache
500	Hardware fetch refers to the number of times the prefetch request was canceled by Scache
501	The number of times the hardware prefetch request was canceled by Scache
502	Number of hardware load prefetches
503	Number of hardware store prefetches
504	Number of hardware data access prefetches
505	Hardware fetch refers to the number of prefetches
506	Number of hardware prefetches
507	The number of load prefetches triggered by tagged
508	The number of load prefetches triggered by miss
509	The number of store prefetches triggered by tagged
510	The number of store prefetches triggered by miss
511	The number of data access prefetches triggered by tagged
512	Number of data prefetches triggered by miss
513	Number of prefetched instructions triggered by tagged
514	Number of instruction prefetches triggered by miss

515	The number of prefetches triggered by tagged
516	Number of prefetches triggered by miss
517	Number of load prefetches accepted by missq
518	Number of store prefetches accepted by missq
519	Number of data access prefetches accepted by missq
520	Number of instruction prefetches accepted by missq
521	Number of prefetches accepted by missq (repeat requests do not enter missq)
522	The number of effective load prefetches returned from scache
523	Number of valid store prefetches returned from scache
524	Number of valid data access prefetches returned from scache
525	Number of instruction prefetches returned from scache
526	The number of effective prefetches returned from scache (pre_scref-> rdy || pre_scref-> pre_rdy)
527	The number of load prefetches that can enter pre_rdy
528	The number of prefetches that can enter the pre_rdy store
529	Number of data access prefetches that can enter pre_rdy

15 Being canceled by scache means that the data is already in the cache of the processor core

Event number	Event description
530	Number of instructions that can enter pre_rdy
531	The number of prefetches that can enter pre_rdy (pre_scref-> pre_rdy)
532	Accumulate the number of load prefetch requests in pre_rdy per beat
533	Accumulate the number of store prefetch requests in pre_rdy per beat
534	Accumulate the number of data access prefetch requests in pre_rdy per beat
535	Accumulate the number of instruction prefetch requests in pre_rdy per beat
536	Accumulate the number of requests in pre_rdy per beat
537	Accumulate the number of prefetches that each beat is in pre_scref and is hit by a normal load request
539	Accumulate the number of prefetches that are in pre_scref and hit by normal store requests for each beat
540	Accumulate the number of prefetches that are in pre_scref and hit by normal data access requests per beat
541	Accumulate the number of prefetches that are in pre_scref and hit by the normal fetch request
542	Accumulate the number of prefetches in each beat that are in pre_scref and hit by normal access
543	The number of hit prefetches that are accessed by load in the pre_scref state
544	The number of hits prefetched by the store in the pre_scref state
545	The number of prefetches hit by data access in the pre_scref state
546	In pre_scref state, it is taken to refer to the number of prefetches hit
547	The number of prefetches that were hit in the pre_scref state, that is, the number of times from pre_scref-> rdy
548	The number of loads from the pre_scref state to the miss state
553	The number of hit prefetches that are accessed by load in the pre_wait state
554	The number of hit prefetches accessed by the store in the pre_wait state
555	The number of prefetches hit by data access in the pre_wait state
556	The number of prefetches that were fetched to access hits in the pre_wait state
557	The number of prefetches that were hit in the pre_wait state, that is, the number of times from pre_wait-> pcmiss
558	Prefetch items in pre_wait state that were replaced because missq cannot accept normal access
559	The prefetch item in pre_rdy state that was replaced because missq cannot accept normal access
560	The number of times the prefetch item is INV
561	Accumulated load prefetch account for each beat
562	Accumulate whether this load prefetch accounted for the item
563	Accumulated store prefetched items per beat
564	Accumulate whether this store prefetch accounted for the item
565	Accumulated data pre-fetched items per beat
566	Accumulate whether the prefetch of this beat data accounts for the item
567	Accumulated prefetched items per beat
568	Accumulated this beat refers to whether the prefetch accounted for
569	Accumulated load prefetch in the number of pre_scref and pre_rdy hits
570	Accumulated store prefetches in pre_scref and pre_rdy hits
571	Accumulated data prefetches the number of hits in pre_scref and pre_rdy

572	Accumulative fetching refers to the number of prefetch hits in pre_scref and pre_rdy
573	Accumulated prefetches in pre_scref and pre_rdy hits

159

Loongson **3A3000 / 3B3000** Processor User Manual • Next

8.2.2 Definition of shared cache performance count events

Table **8-3** Definition of shared cache performance counter events

Event number	Event description
0	Since there is no switch mark in ctrl, the configuration of 0 means no switch.
1	The number of all request requests received
2	Number of cachable request requests received
3	All cachable request requests received for all non-DMA operations
4	All cachable request requests with non-prefetched attributes received
5	REQ_READ requests for all cached received
6	REQ_WRITE requests for all cached received
7	REQ_READ of all cached non-prefetched attributes received
8	REQ_WRITE of all cached non-prefetched attributes received
9	All cached DMAREAD requests received
10	All cached DMAWRITE requests received
11	All uncached DMAREAD received
12	All uncached DMAWRITE received
13	Number of all DMA requests received
14	Number of requests received with all types being scache_prefx
15	The number of prefetches generated by the Scache hardware itself is accepted
16	Number of requests received for all types of store_fill_full
17	All responses received for consistency requests
18	The number of all received responses for consistency and the data is dirty
19	The write-back of the active replacement of the upper-level cache received
20	The number of received upper-level active replacements and the data is dirty
twenty one	Number of read requests for memory caused by cached access
twenty two	Number of write requests to memory due to cached access
twenty three	The number of times the query scache result is miss
twenty four	Query the number of all REQREADs whose scache result is miss
25	Query the number of all REQWRITE results of scache miss
26	Query scache, the number of scache hits but pagecolor misses
27	REQREAD query scache, the result hits a clean block
28	REQREAD query scache, the result hits someone else's EXC block
29	REQWRITE query scache, the result is in the clean block
30	REQWRITE query scache, the result hits someone else's EXC block
31	WRITE query scache, the result is hit in a block being shared by multiple cores
32	Cached DMAREAD query scache, result hit
33	Cached DMAWRITE query scache, the result does not hit
34	Cached DMAWRITE query scache, the result hit, and need to issue INVALIDATION to CPU
35	Number of INV requests in consistency requests issued to the CPU
36	The number of WTBK requests in the consistency request to the CPU
37	Number of INVWTBK requests in consistency requests to the CPU

160

161