

LOONGSON

Loongson 3A3000/3B3000 processor user manual

Part ii

GS464E processor core V1.2

In December 2017

Loongson Zhongke Technology Co. LTD



Copyright statement

The copyright of this document belongs to Loongson Technology Co., LTD., and all rights are reserved. No company or individual may publish, reprint or otherwise distribute any part of this document to third parties without written permission. Otherwise, it will be investigated for legal responsibility.

disclaimer

This document only provides phased information. The contents can be updated according to the actual situation of the product at any time without prior notice. The Company shall not be liable for any direct or indirect losses caused by improper use of documents.

Loongson Zhongke Technology Co. LTD

Loongson Technology Corporation Limited

Address: Loongson Industrial Park, Building No.2, Loongson Industrial Park, Zhongguancun Environmental Protection Technology Demonstration Park, Haidian District, Beijing
Zhongguancun Environmental Protection Park, Haidian District, Beijing

Tel: 010-62546668 Fax: 010-62600826

Reading guide

The User manual of Loongson 3A3000/3B3000 processor is divided into the first and the second volumes.

The second volume of The User's Manual of The 3A300/3B3000 processor introduces in detail the GS464E high-performance processor core used by the 3A300/3B3000 processor from the perspective of system software developers.

Special format meaning introduction

1. When the description of a Field of CP0 control register is involved in this document, the format of reg. Field is adopted, in which Reg is the mnemonic of the control register and Field is the mnemonic of the Field to be described in the storage. For example, Ebase.cpunum represents the CPUNum field of the EBase control register.
2. The description of data content interception in this document is in the format of [m:n] or, indicating that the NTH bit to m of the content to be intercepted is selected $m..n$ position M, n starts at 0, m is greater than or equal to n.

Version history

Document update record	The document name:	Loongson 3A3000/3B3000 processor user manual - part ii		
	The version number	V1.2		
	The founders:	Chip R&d Department		
	Date created:	2017-12-20		
Update history				
The serial number	Updated date	Update one	The version number	Update the content
1	2017/04/07	Chip R&d Department	V1.0	The first draft is complete.
2	2017/11/29	Chip R&d Department	V1.1	1. Fixed an error in the instruction list in Chapter 2. 2. Section 2.3.1 adds the list of "DSP instructions no longer supported by MIPS"
3	2017/12/20	Chip R&d Department	V1.2	Fixed 8 instructions in the instruction list in Chapter 2.

Technical support

You can submit the problems of product use to our company through email or problem feedback website, and obtain technical support. After-sales Service

Email: service@loongson.cn

Question feedback web site: <http://bugs.loongnix.org/>

directory

目录

1	Overview of processor core structure	12
1.1	Speed overview of processor core structure parameters	15
2	Instruction set Overview	5
2.1	MIPS64 compatible General instruction list	5
2.1.1	To fetch instruction	5
2.1.2	Operation instruction	7
2.1.3	Jump and branch instructions	9
2.1.4	Coprocessor 0 instruction	11
2.2	MIPS64 is compatible with floating point instruction set overview	14
2.2.1	FPU data type	14
2.2.2	Floating point register	17
2.2.3	Floating point control register	17
2.2.4	Floating-point exception	17
2.2.5	MIPS64 is compatible with floating point instruction list	21
2.3	An overview of THE MIPS64 DSP instruction set	26
2.3.1	MIPS64 DSP ASE compatible instruction list	27
2.3.2	Supplementary instructions to MIPS DSP instruction manual	47
2.4	MIPS64 compatible instruction implementation definition	47
2.4.1	The load instruction that targets the no. 0 general purpose register	47
2.4.2	PREF instruction	47
2.4.3	RDHWR instruction	47
2.4.4	PREFX instruction	47
2.4.5	WAIT instruction	49
2.4.6	SYNC instructions	49
2.4.7	SYNCI instruction	49
2.4.8	TLBINV and TLBINVF directives	49
2.4.9	CACHE directives	50
2.4.10	Madd.fmt, Msub.fmt, Nmadd.fmt, Nmsub.fMT instruction	52
2.4.11	EHB, SSNOP instructions	53
2.4.12	DI and EI instructions	53
2.5	Loong core expansion command set	53
3	Processor running mode	72
3.1	Processor run mode definition	72
3.1.1	Debug mode	72
3.1.2	The root-core pattern	72
3.1.3	Root-user mode	73
4	Memory management	47
4.1	The basic concept	47
4.1.1	Address space	47
4.1.2	Segment and segment size (SEGBITS)	47
4.1.3	Physical Address size (PABITS)	47
4.1.4	Mapped Address and Unmapped Address	47
4.2	Host virtual address space	47
4.2.1	Host address space division and access control	47

4.2.2	The address translation, cacheability and cache consistency of host address space Kseg0 segment and Kseg1 segment.....	50
4.2.3	The address translation and cacheability of the host address space Xkphys segment are consistent with the cache properties	50
4.2.4	Address translation of the kusEG segment of the host address space when status.eri =1.....	50
4.2.5	Special treatment of host address space kseg3 when debug.dm =1	51
4.2.6	Special handling of data access to virtual addresses when status.ux =0 in user mode	51
4.3	TLB - based virtual and real address mapping	50
4.3.1	TLB hierarchy.....	50
4.3.2	JTLB structure	51
4.3.3	JTLB table item	52
4.3.4	TLB software management.....	53
4.3.5	TLB initialization and clearing.....	53
4.3.6	TLB - based virtual address translation process	54
5	Organization and management of caches	59
5.1	Processor storage hierarchy and cache hierarchy	59
5.1.1	Processor storage hierarchy	59
5.1.2	Level 1 instruction Cache (I-cache).....	61
5.1.3	Level 1 data Cache (D-Cache).....	63
5.1.4	Level 2 Sacrifice Cache (V-Cache)	64
5.1.5	Level 3 Shared Cache (S-Cache).....	66
5.2	The cache algorithm has the same properties as the cache	68
5.2.1	Non-cache algorithm.....	68
5.2.2	Consistent caching algorithm.....	68
5.2.3	Non-cache acceleration algorithm	68
5.3	Cache consistency.....	69
5.4	Cache management.....	70
5.4.1	CACHE directives	70
5.4.2	Cache initialization	67
5.4.3	Maintain consistency between level 1 instruction cache and level 1 data cache	70
5.4.4	Maintain cache consistency between processor and DMA device	70
5.4.5	Cache alias and page coloring	70
6	Processor exceptions and interrupts.....	72
6.1	Processor exception	72
6.1.1	Exception priority	72
6.1.2	Exception entry vector position	72
6.1.3	The processor hardware responds to the exception's generic processing.....	74
6.1.4	Cold reset exception.....	75
6.1.5	Non-masking interrupt.....	76
6.1.6	Interrupt exception.....	76
6.1.7	Wrong address exception.....	76
6.1.8	TLB rewrites the exception.....	78
6.1.9	XTLB rewrites the exception.....	78
6.1.10	TLB is not an exception.....	80
6.1.11	TLB modification exceptions	76
6.1.12	TLB performs blocking exceptions	76
6.1.13	TLB reads prevent exceptions	78
6.1.14	Cache error exception	78
6.1.15	Exception for integer overflow	78

6.1.16	Trap exceptions.....	78
6.1.17	System call exception	78
6.1.18	Breakpoint exception.....	78
6.1.19	Reservation instruction exception.....	79
6.1.20	No exceptions can be made to the coprocessor.....	80
6.1.21	Floating-point exception.....	80
6.1.22	Floating point stack exception	80
6.2	interrupt.....	80
6.2.1	Requirements for interrupt response.....	80
6.2.2	Interrupt mode	80
6.2.3	Additional notes on interrupt handling	84
7	Coprocessor register 0	86
7.1	Root coprocessor 0 register overview.....	86
7.2	Index Register (CP0 Register 0, Select 0)	85
7.3	Random Register (CP0 Register 1, Select 0).....	86
7.4	EntryLo0 and EntryLo1 registers (CP0 Register 2 and 3, Select 0)	87
7.5	Context Register (CP0 Register 4, Select 0).....	90
7.6	UserLocal Register (CP0 Register 4, Select 2).....	91
7.7	PageMask Register (CP0 Register 5, Select 0).....	92
7.8	PageGrain Register (CP0 Register 5, Select 1).....	94
7.9	PWBase Register (CP0 Register 5, Select 5).....	96
7.10	PWField Register (CP0 Register 5, Select 6)	98
7.11	PWSize Register 5, Select 6	100
7.12	Wired Register (CP0 Register 6, Select 0).....	102
7.13	PWctl Register 6, Select 6	103
7.14	Trade characters for a Register (CP0 Register 7, Select 0).....	104
7.15	BadVAddr Register (CP0 Register 8, Select 0).....	105
7.16	Count Register 9, Select 0.....	106
7.17	GSEBase Register (CP0 Register 9, Select 6)	107
7.18	PGD Register (CP0 Register 9, Select 7).....	108
7.19	EntryHi Register (CP0 Register 10, Select 0).....	109
7.20	Compare Register (CP0 Register 11, Select 0)	106
7.21	Status Register (CP0 Register 12, Select 0).....	107
7.22	IntCtl Register (CP0 Register 12, Select 1)	109
7.23	SRSCtl Register (CP0 Register 12, Select 2).....	110
7.24	Cause Register (CP0 Register 13, Select 0).....	111
7.25	EPC Register (CP0 Register 14, Select 0)	113
7.26	PRId Register (CP0 Register 15, Select 0)	114
7.27	EBase Register (CP0 Register 15, Select 1)	115
7.28	Config Register (CP0 Register 16, Select 0)	116
7.29	Config1 Register (CP0 Register 16, Select 1).....	117
7.30	Config2 Register (CP0 Register 16, Select 2).....	118
7.31	Config3 Register (CP0 Register 16, Select 3).....	119
7.32	Config4 Register (CP0 Register 16, Select 4).....	121
7.33	Config5 Register (CP0 Register 16, Select 5).....	123
7.34	GSConfig Register (CP0 Register 16, Select 6)	124
7.35	LLAddr Register (CP0 Register 17, Select 0).....	127
7.36	XContext Register (CP0 Register 20, Select 0).....	128
7.37	Diag Register (CP0 Register 22, Select 0).....	130

7.38	GSCause Register (CP0 Register 22, Select 1).....	131
7.39	VPID Register (CP0 Register 22, Select 2).....	132
7.40	Debug Register (CP0 Register 23, Select 0).....	133
7.41	DEPC Register (CP0 Register 24, Select 0).....	134
7.42	PerfCnt Register (CP0 Register 25, Select 0~7).....	135
7.43	ErrCtl Register (CP0 Register 26, Select 0).....	137
7.44	CacheErr Register (CP0 Register 27, Select 0).....	138
7.45	CacheErr1 Register (CP0 Register 27, Select 1).....	140
7.46	TagLo register (CP0 Register28, Select 0).....	141
7.47	DataLo Register (CP0 Register 28, Select 1).....	144
7.48	TagHi Register (CP0 Register 29, Select 0).....	145
7.49	DataHi Register (CP0 Register 29, Select 1).....	146
7.50	ErrorEPC Register (CP0 Register 30, Select 0).....	147
7.51	DESAVE Register (CP0 Register 31, Select 0).....	148
7.52	KScratch1~6 registers (CP0 Register 31, Select 2~7).....	149
8	Analysis and optimization of processor performance.....	150
8.1	Organization and access method of performance counters.....	150
8.1.1	Processor core performance counter.....	150
8.1.2	Shared cache performance counters.....	150
8.2	Processor performance count events.....	151
8.2.1	Processor core performance count event definition.....	151
8.2.2	Shared cache performance count event definition.....	167

Figure directory

Figure 2-1 FPU floating point data format 11	11
Figure 2-2 FPU fixed-point data format 13	13
Figure 2-3 FIR register format 13	13
Figure 2-4. FCSR register format 14	14
Figure 2-5 FCCR register format 15	15
Figure 2-6 FEXR register format 16	16
Figure 2-7 FENR register format 16	16
Figure 2-8. Address resolution format of the Index class CACHE instruction 34	34
Figure 4-1 Xkphys segment virtual address resolution method 49	49
Figure 5-1 Longson 3A3000 chip processor storage level 59	59
Figure 5-2. Structure of level 1 instruction cache line is 60	60
Figure 5-3 The row structure of the first-level data cache is 61	61
Figure 5-4 Shows the second-level sacrifice cache row structure for 62	62
Figure 5-5 Shows the three-level Shared cache row structure for 64	64
Figure 5-6 Cache state transition under conformance protocol 65	65
Figure 7-1. Index register format 85	85
Figure 7-2. Random register format 86	86
Figure 7-3 Register format 87 for THE DMFCO/DMTCO instruction access for EntryLo0 and EntryLo1	87
Figure 7-4 Register format 88 for EntryLo0 and EntryLo1 when accessed by the MFCO/MTCO instructions	88
Figure 7-5 Context register format 90	90
Figure 7-6 UserLocal register format 91	91
Figure 7-7 PageMask register format 92	92
Figure 7-8 PageGrain register format 93	93
Figure 7-9 Page table access process 94 supported by PWBase, PWField, PWSIZE and PWctl .	94
Figure 7-10. PWBase register format 94	94
Figure 7-11. PWField register format 95	95
Figure 7-12. PWSIZE register format 96	96
Figure 7-13 Fixed table entry and random replacement table entry boundary in VTLB 97 .	97
Figure 7-14 Wired register format 97	97
Figure 7-15 PWctl register format 98	98
Figure 7-16 For trade A register format 99	99
Figure 7-17 BadVAddr register format 100	100
Figure 7-18. Count register format 101	101
Figure 7-19 GSEBase register format 102	102
Figure 7-20 PGD register format 103	103
Figure 7-21. EntryHi register format 104	104
Figure 7-22 Compare register format 106	106
Figure 7-23 Status register format 107	107
Figure 7-24 IntCtl register format 109	109
Figure 7-25 SRSCtl register format 110	110
Figure 7-26 Cause register format 111	111



Figure 7-28 PRId register format 114
Figure 7-29 EBase register format 115
Figure 7-30. Config register format 116
Figure 7-31 Config1 register format 117
Figure 7-32 Config2 register format 118
Figure 7-33 Config3 register format 119
Figure 7-34 Config4 register format 121
Figure 7-35. Config5 register format 123
Figure 7-36 GSConfig register format 124
Figure 7-37. LLAddr register format 127
Figure 7-38 XContext register format 128
Figure 7-39 Diag register format 129
Figure 7-40 GSCause register format 131
Figure 7-41 VPID register format 132
Figure 7-42 DEPC register format 134
Figure 7-43 PerfCnt Control register format 135
Figure 7-44 PerfCnt Counter register format 136
Figure 7-45 ErrCtl register format 137
Figure 7-46 CacheErr register format 138 when used for I-Cache error-checking information
Figure 7-47 CacheErr register format 138 when used for D-Cache error-checking information
Figure 7-48 CacheErr1 register format 140
Figure 7-49 The TagLo register is used to access the I-Cache Tag in format 141
Figure 7-50 TagLo register is used for format 141 when accessing the D-Cache Tag
Figure 7-51 TagLo register in format 142 for accessing the V-Cache Tag
Figure 7-52 The TagLo register is used to access the S-Cache Tag in format 142
Figure 7-53. The TagLo register is in format 143 for accessing each level of Cache Data
Figure 7-54 The DataLo register is used for i-Cache access in format 144
Figure 7-55 Shows the format 145 of the TagHi register used to access the various Cache tags
Figure 7-56 The TagHi register is in format 145 for accessing each level of Cache Data
Figure 7-57 DataHi registers are used for i-Cache access in format 146
Figure 7-58. ErrorEPC register format 147
Figure 7-59 DESAVE register format 148
Figure 7-60 KScratchn register format 149

The table directory

Table 2-1 CPU instruction set: access instruction 5	5
Table 2-2 Operation instruction: arithmetic instruction (ALU immediate number) 6	6
Table 2-3 Operation instruction: arithmetic instruction (3 operands) 6	6
Table 2-4 Operation instruction: arithmetic instruction (2 operands) 7	7
Table 2-5 Operation instruction: multiplication and division instruction 7	7
Table 2-6 Operation instruction: shift instruction 7	7
Table 2-7 Jump and branch instructions 8	8
Table 2-8 Coprocessor instruction 0 9	9
Table 2-9 Other instructions: Special instruction 10	10
Table 2-10 Other instructions: Exceptions fall into instruction 10	10
Table 2-11 Other instructions: Conditional move instruction 10	10
Table 2-12 Other instructions: Prefetch instruction 10	10
Table 2-13 Floating point format related parameters 12	12
Table 2-14 Calculation method of floating point value V 12	12
Table 2-15 Maximum and minimum values of floating point Numbers 12	12
Table 2-16 FIR register fields describe 13	13
Table 2-17 Describes the FCSR register field 14	14
Table 2-18 Codes for rounding mode (RM) 15	15
Table 2-19 Default handling for floating point exceptions: 17	17
Table 2-20 Float branch jump instructions 19	19
Table 2-21 Floating point operation instructions 19	19
Table 2-22 Floating-point branch jump instruction 20	20
Table 2-23 Floating-point branch jump instruction 20	20
Table 2-24 Floating-point branch jump instruction 20	20
Table 2-25 Floating-point branch jump instruction 21	21
Table 2-26 CACHE instruction OP [1:0] corresponds to the CACHE hierarchy 34	34
Table 2-27 Instruction of longson extended access class 35	35
Table 2-28 Instructions for extended arithmetic and logic operations of the Godson 37	37
Table 2-29 Longson extension X86 binary translation acceleration instruction 38	38
Table 2-30 Acceleration instruction of ARM BINARY translation extension	
Table 2-31 Loong core extension 64 bit multimedia instruction 42	42
Table 2-32 Godson expansion miscellaneous instruction 44	44
Table 3-1 Processor mode determination is based on 45	45
Table 4-1 Host address space division and access control	
Table 4-2 TLB management related CPO register 52	52
Table 4-3 TLB manages related privileges instruction 52	52
Table 5-1 Cache parameter 60	60
Table 5-2 Tertiary Shared cache body selection bit and index address 63	63
Table 5-3 CACHE instruction 66 in root mode	66
Table 6-1 Exception priority 71	71
Table 6-2 Exception vector base address 72	72
Table 6-3 Exception vector offset 72	72

Table 6-4	Each interrupt request generates 81 in compatible interrupt mode	81
Table 6-5	Priority relationship among interrupts in vector interrupt mode	81
Table 6-6	Interrupt mode decision	81
Table 7-1	List of coprocessor zero registers	83
Table 7-2	Index register field description	85
Table 7-3	Description of Random register field	86
Table 7-4	The register fields for EntryLo0 and EntryLo1 are described as	87 for DMFCO/DMTCO instruction access
Table 7-5	The register fields for EntryLo0 and EntryLo1 at the time of MFCO/MTCO instruction access are described as	88
Table 7-6	Cache attribute encoding	Table 89
Table 7-7	Context register fields describe	90
Table 7-8	UserLocal register field description	91
Table 7-9	PageMask register field description	92
Table 7-10	Mask domain codes and page sizes	92
Table 7-11	The PageGrain register field is described in	93
Table 7-12	PWBase register fields describe	94
Table 7-13	PWField register fields describe	95
Table 7-14	PWSize register fields describe	96
Table 7-15	Description of Wired register Field	
Table 7-16	PWCtl register fields describe	98
Table 7-17	Trade A register field describes	99
Table 7-18	BadVAddr register fields describe	100
Table 7-19	Count register fields describe	101
Table 7-20	GSEBase register fields describe	102
Table 7-21	PGD register field description	103
Table 7-22	The EntryHi register field is described in	104
Table 7-23	Compare register field describes	106
Table 7-24	Description of the Status register field	107
Table 7-25	IntCtl register field description	109
Table 7-26	SRSCtl register field description	110
Table 7-27	Describes 111 for the Cause register field	
Table 7-28	ExcCode code and its corresponding exception type	111
Table 7-29	EPC register fields describe	113
Table 7-30	PRId register fields describe	114
Table 7-31	EBase register fields describe	115
Table 7-32	Config register field description	116
Table 7-33	Config1 register field description	117
Table 7-34	Config2 register field description	118
Table 7-35	Config3 register field describes	119
Table 7-36	Config4 register field describes	121
Table 7-37	Config5 register field describes	123
Table 7-38	GSConfig register field description	124
Table 7-39	The LLAddr register field describes	127

Table 7-40 XContext register field description 128

Table 7-41 Diag register fields are described in 129



Table 7-42 GSCause register fields describe	131
Table 7-43 GSExcCode codes and their corresponding exception types	131
Table 7-44 VPID register field description	132
Table 7-45 DEPC register field description	134
Table 7-46 PerfCnt register Select assigns	135
Table 7-47 PerfCnt Control register field describes	135
Table 7-48 PerfCnt Counter register field describes	136
Table 7-49 The ErrCtl register field is described	137
Table 7-50 Describes the field 138 when the CacheErr register is used to check the i-Cache error information	
Table 7-51 CacheErr register field description 138 when used for D-Cache error-checking information	
Table 7-52 CacheErr1 register field description	140
Table 7-53 The field description 141 for accessing the I-Cache Tag in the TagLo register	
Table 7-54 The field description 141 is used to access the D-Cache Tag in the TagLo register	
Table 7-55 TagLo register describes the field 142 when accessing the V-Cache Tag	
Table 7-56 The field description for accessing the S-Cache Tag in the TagLo register is	142
Table 7-57 TagLo registers describe the fields 143 when accessing each level of Cache Data	
Table 7-58 The field description of the DataLo register for i-Cache access is	144
Table 7-59 The field description of the TagHi register for accessing the various Cache tags is	145
Table 7-60 Describes the field 145 when the TagHi register is used to access all levels of Cache Data	
Table 7-61 DataHi registers describe	146 fields used for i-Cache access
Table 7-62 Description of the ErrorEPC register fields	147
Table 7-63 DESAVE register field description	148
Table 7-64 KScratchn register fields are described in	149
Table 8-1 Shared cache performance counter register address offset	150
Table 8-2 Processor core performance counter event definition	151
Table 8-3 Shared cache performance counter event definition	160

1 Overview of processor core structure

The Loongson GS464E processor core (hereinafter referred to as "GS464E") is a general-purpose RISC processor core that implements the LoongsonISA instruction set and is a performance optimized and upgraded version of the Loongson GS464 processor core. The instruction pipeline of GS464E takes four instructions for decoding in each clock cycle and dynamically transmits them to six functional parts of the whole flow. Instructions can be executed in random order on the premise of ensuring dependency. All instructions are submitted in the order in the program to ensure accurate exception and access sequence execution.

Instruction correlation and data correlation are the primary factors that affect the performance of multi-emission deep flow processor. Therefore, GS464E adopts out-of-sequence execution technology and radical storage system design to improve the efficiency of pipeline.

The techniques of out-of-order execution include register renaming, dynamic scheduling and transfer prediction. Register renaming addresses the correlation between WAR (read after write) and WAW (write after write) and is used for precise field recovery due to exception and error transition prediction. GS464E renames fixed-point and floating point registers with two 128-item physical register heaps, and renames HI/LO registers, DSP Control registers and floating point Control registers with 16-item, 32-item and 32-item physical register heaps, respectively. Dynamic scheduling executes instructions according to the order in which the instruction operands are prepared, rather than the order in which the instruction appears in the program, reducing raw-related blocking. GS464E USES a 16-item fixed-point reserve station, a 24-item floating-point reserve station and a 32-item access reserve station for out-of-order emission, and implements out-of-order instructions to be submitted according to program order through a 128-item Reorder queue (ROQ). Transition prediction reduces congestion due to control correlation by predicting whether the transition instruction will jump successfully. GS464E USES 8K Global Branch History Table (GBHT), 8K Local Branch History Table (LBHT), 8K Global Branch Select Table (GBSEL), and 13-bit Global History Register, GHR (for short), Branch Target Buffer (BTB) of 1K items and Return Address Stack (RAS) of 16 items are used for transfer prediction. A 24-branch queue (BRQ) is used for all Branch instructions to realize accurate cancellation of subsequent instructions when Branch instructions are mispredicted.

GS464E advanced storage system design can effectively improve the efficiency of the pipeline. The GS464E contains two fully functional accessors. Each accessor can perform Load and Store operations independently and smoothly. GS464E solves address dependency dynamically by means of 64-item access Queue (CP0 Queue), and realizes out-of-order execution of access operations and non-blocking Cache. GS464E adopts three-level Cache structure, in which the first-level Cache consists of 64KB instruction Cache and 64KB data Cache. Each processor core contains private 256KB second-level instruction data Shared Cache, and adopts 64-byte Cache line and 16-way group connection structure. Each of the four cores shares 4MB of three-level Cache. GS464E adopts a two-stage TLB structure, in which the first-level TLB is divided into 64-item fully linked instruction TLB (referred to as ITLB) and 32-item fully linked data TLB (referred to as DTLB), and the second-level TLB contains a 64-item fully linked TLB of variable page size

(VTLB) and a fixed page size TLB (FTLB) with a 1024-item 8-way group linkage structure, each of which can map to an odd and an even page, varying in size from 4KB to 1GB.

The GS464E has two fully functional fixed-point features and two fully functional floating point features. Each fixed-point unit can execute branch instructions and perform fixed-point multiplication and all DSP operations in full flow. Each floating-point component can perform 64-bit double-precision floating-point

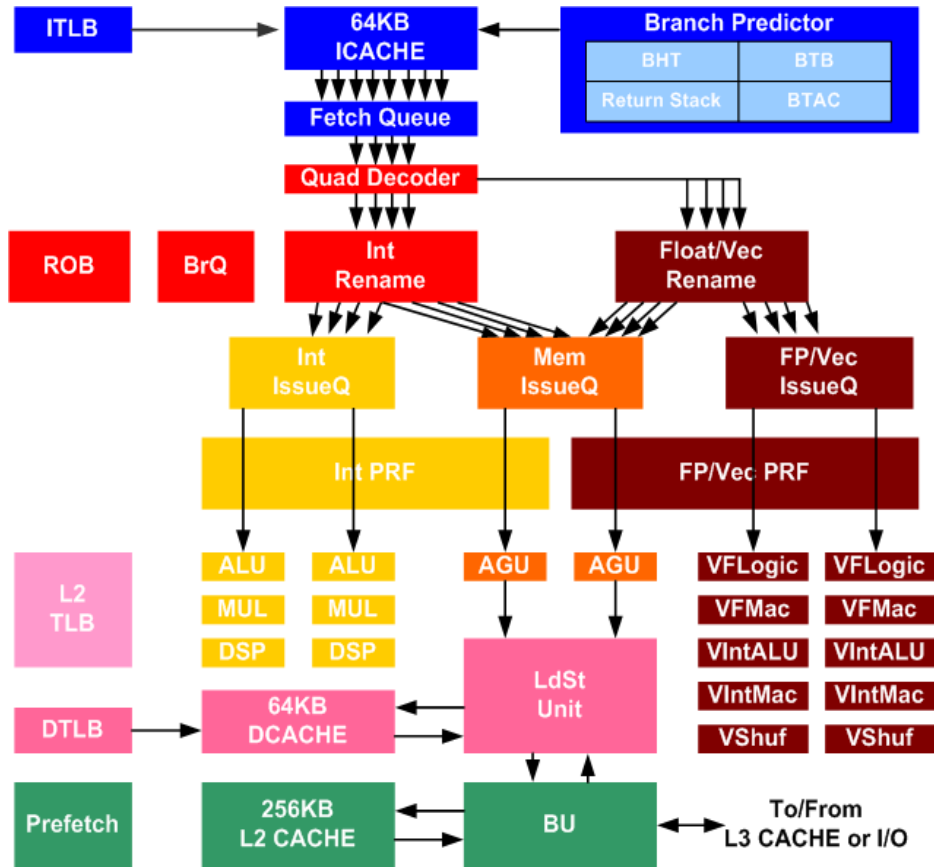
multiplication and addition operations in full stream, and 32-bit and 64-bit fixed-point instructions through the extension of the FMT field of floating-point instructions.

GS464E supports MIPS EJTAG debugging specification and USES standard AXI interface. Its instruction Cache realizes parity and data Cache realizes ECC.

The basic pipeline of GS464E includes 12 levels, including PC, pointing, pre-decoding, decoding I, decoding II, register renaming, scheduling, transmitting, reading register, executing, submitting I, submitting II, etc. Each level of pipeline includes the following operations.

- The PC pipeline level is used to generate the program counter PC value required for the next tap finger.
- To access the instruction Cache and TLB with the value of program counter PC, if both instruction Cache and TLB hit, eight new instructions are fetched to the instruction register IR.
- The pre-decoder pipelining mainly decodes the transfer instruction and predicts the direction of the jump.
- Decoding I Pipelining saves the pre-decoded results to the instruction queue.
- Decoding II pipelining transforms four instructions in IR into the processor's internal instruction format and sends them to the register rename module.
- Register renames the flow-level to assign a new physical register to the logical destination register and map the logical source register to the physical register recently assigned to the logical register.
- The scheduling flow level assigns the renamed instruction to the fixed point or floating point reserve station for execution and sends it to the ROQ for sequential submission after execution. In addition, the transfer instruction and access instruction are sent to the transfer queue and access queue respectively.
- The stream stage selects an instruction prepared by all operands for each feature from a fixed point or floating point reserve station; Instructions whose operands are not ready at rename wait for their operands to be ready by listening for the result bus and the forward bus.
- Read register flow-level is the emitted instruction that reads the corresponding source operands from the physical register heap to the corresponding functional unit.
- The flow-level executes the instruction according to the instruction type and writes the calculated result back to the register heap; The resulting bus is also sent to the reserve station and the register rename table to notify that the corresponding register value is available.
- In accordance with the order of the program recorded in the Reorder queue, the instructions that can be submitted can be selected from the instructions that have been executed. GS464E can submit up to four instructions per beat.
- The commit II pipeline sends the selected commit instruction to the register renaming table to confirm the renaming relationship of its destination register and frees the physical registers originally assigned to the same logical register and sends them to the access queue to allow those committed memory instructions to be written to the Cache or memory.

The above is the flow-level of the basic instruction. For some more complicated instructions, such as fixed-point multiplication and division instruction, floating-point instruction and access instruction, multiple beats are



required in the execution stage. The basic structure of GS464E is shown in the figure below.

1.1 Speed overview of processor core structure parameters

Instruction set	MIPS64r2 compatible LoongISA instruction set
Flow depth	Grade 12
Take refers to the width	8 instructions/cycles
Launch the width	(2 fixed point +2 floating point +2 access)/period
Level 1 instruction Cache specification	Private per core, 4 x 16KB/ channel
Level 1 data Cache specification	Private per core, 4 x 16KB/ channel
Secondary Cache specification	Private per core, 16 channels ×16KB/ channel
Three level Cache specifications	Multi-core sharing, 16 channels ×128KB/ channel
TLB capacity	64 VTLB+2048 FTLB
TLB supports page sizes	4KB (I=1, 2... ⁱ , 10)

2 Instruction set Overview

GS464E realized LoongISA v1.00 version of MIPS64 compatible basic part, MIPS64 DSP instruction (MIPS64 DSP Module) and 64 common extension instruction (LoongEXT64).

2.1 MIPS64 compatible General instruction list

GS464E implements all of the MIPS64 specifications required to implement and a few optional implementations of the generic instructions, its implementation of MIPS64 compatible generic instructions are divided into the following categories by function:

- To fetch instruction
- Operation instruction
- Transfer instruction
- Other instructions
- Coprocessor 0 instruction

These instructions are listed class by class below.

2.1.1 To fetch instruction

MIPS architecture adopts load/ Store architecture. All operations are performed on registers, and only access instructions can access the data in main memory. Access instruction includes read and write, unsigned read, unaligned access and atomic access of various width data.

Table 2-1 CPU instruction set: access instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
LB	In bytes	MIPS32
LBU	Takes an unsigned byte	MIPS32
LH	Take half word	MIPS32
LHU	Take the unsigned half word	MIPS32
LW	Take the word	MIPS32
LWU	Take the unsigned word	MIPS32
LWL	Take words left	MIPS32
LWR	Take the words right	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 two-character address at the sign	MIPS64
SB	Remaining bytes	MIPS32
SH	Save half word	MIPS32
SW	characters	MIPS32

SWL	Characters left	MIPS32
SWR	To save words right	MIPS32

Instruction mnemonic	Instruction function description	ISA Compatible Category
SD	Save double word	MIPS64
SDL	Save the left part of double characters	MIPS64
The SDR	Save the right part of double characters	MIPS64
SC	If I can save it	MIPS32
SCD	Meet the conditions to save double characters	MIPS64

2.1.2 Operation instruction

Operational instructions perform arithmetic, logic, shift, multiplication, and division operations on register values. The operational instruction contains the register instruction format (R-type, where the operands and operation results are stored in the register) and the immediate instruction format (i-type, where one operand is a 16-bit immediate number)

Table 2-2 Operation instruction: arithmetic instruction (ALU immediate number)

Instruction mnemonic	Instruction function description	ISA Compatible Category
ADDI	Add number immediately	MIPS32
DADDI	Add double word number immediately	MIPS64
ADDIU	Add an unsigned immediate number	MIPS32
DADDIU	Add an unsigned double word number immediately	MIPS64
SLTI	Less than immediate Settings	MIPS32
SLTIU	Unsigned less than immediate number setting	MIPS32
ANDI	And immediately	MIPS32
ORI	Or the number immediately	MIPS32
XORI	Xor immediate number	MIPS32
LUI	Take immediate count to high	MIPS32

Table 2-3 Operation instruction: Arithmetic instruction (3 operands)

Instruction mnemonic	Instruction function description	ISA Compatible Category
The ADD	add	MIPS32
DADD	Double word plus	MIPS64
ADDU	Unsigned add	MIPS32
DADDU	Unsigned double word plus	MIPS64
SUB	Reduction of	MIPS32
DSUB	Double word cut	MIPS64
SUBU	Unsigned reduction	MIPS32
DSUBU	Unsigned double word subtraction	MIPS64
SLT	Less than the set	MIPS32
SLTU	Unsigned less than set	MIPS32
The AND	with	MIPS32

The OR	or	MIPS32
XOR	Exclusive or	MIPS32
NOR	Or not	MIPS32

Table 2-4 Operation instruction: Arithmetic instruction (2 operands)

Instruction mnemonic	Instruction function description	ISA Compatible Category
CLO	Word leading 1 number	MIPS32
DCLO	Double word leading 1 number	MIPS64
CLZ	The number of 0 word precursors	MIPS32
DCLZ	Double word leading 0 number	MIPS64
WSBH	Byte exchange in half word	MIPS32 R2
DSHD	Half-word exchange between words	MIPS64 R2
DSBH	Byte exchange in half word	MIPS64 R2
SEB	Byte symbol extension	MIPS32 R2
SEH	Half character extension	MIPS32 R2
INS	An insert	MIPS32 R2
EXT	An extract	MIPS32 R2
DINS	Double word bit insertion	MIPS64 R2
DINSM	Double word bit insertion	MIPS64 R2
DINSU	Double word bit 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 description	ISA Compatible Category
The MUL	Multiply by the general register	MIPS32
MULT	take	MIPS32
DMULT	Double word by	MIPS64
MULTU	Unsigned by	MIPS32
DMULTU	Unsigned double word multiplication	MIPS64
MADD	By adding	MIPS32
MADDU	Unsigned multiplication plus	MIPS32
MSUB	By reducing	MIPS32
MSUBU	Unsigned times minus	MIPS32
DIV	In addition to	MIPS32
DDIV	Double word except	MIPS64
DIVU	Unsigned except	MIPS32
DDIVU	Unsigned double word division	MIPS64
MFHI	Take the number from the HI register to the general register	MIPS32
MTHI	From general purpose register to HI register	MIPS32

MFLO	Fetch from register LO to general purpose register	MIPS32
MTLO	From General Purpose register to LO register	MIPS32

Table 2-6 Operation instruction: shift instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
SSL	The logical left	MIPS32
SRL	Logic moves to the right	MIPS32
SRA	Arithmetic moves to the right	MIPS32
SLLV	Variable logic moves left	MIPS32
SRLV	Variable logic moves to the right	MIPS32
SRAV	Variable arithmetic shift to the right	MIPS32
ROTR	Cycle moves to the right	MIPS32 R2
ROTRV	A variable loop moves to the right	MIPS32 R2
DSLL	Two-word logic moved left	MIPS64
DSRL	Two-word logic right shift	MIPS64
DSRA	Two-word arithmetic shift to the right	MIPS64
DSLLV	Variable two-word logic moves left	MIPS64
DSRLV	Variable two-word logic moves right	MIPS64
DSRAV	Variable two-word arithmetic shift to the right	MIPS64
DSLL32	Shift amount plus 32 double word logic to the left	MIPS64
DSRL32	Shift + 32 double word logic moves right	MIPS64
DSRA32	Shift amount plus 32 double word arithmetic shift to the right	MIPS64
DROTR	Double word loop right shift	MIPS64 R2
DROTR32	Shift amount plus 32 double word cycle right shift	MIPS64 R2
DROTRV	Double-word variable loop right shift	MIPS64 R2

2.1.3 Jump and branch instructions

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

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

In MIPS architecture, all transfer instructions are followed by a delay slot instruction. Delay slots for Likely transfer instructions are only executed when the transfer is successful; delay slots for non-likely transfer instructions are always executed. The return address of a procedure call instruction is saved in register 31 by default, and a jump based on register 31 is considered to be returned from the called procedure.

Table 2-7 Jump and branch instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
-----------------------------	---	--------------------------------

J	jump	MIPS32
JAL	Call the procedure immediately	MIPS32
JR,	Jump to the instruction that the register points to	MIPS32
JR. HB	Jump to the instruction that the register points to	MIPS32 R2
JALR	Register call procedure	MIPS32
JALR. HB	Register call procedure	MIPS32 R2
BEQ	Equal jump	MIPS32
BNE	Unequal jump	MIPS32

Instruction mnemonic	Instruction function description	ISA Compatible Category
BLEZ	Less than or equal to 0 jump	MIPS32
BGTZ	Jump greater than 0	MIPS32
BLTZ	Jump less than 0	MIPS32
BGEZ	A jump greater than or equal to 0	MIPS32
BLTZAL	Less than 0 calls the procedure	MIPS32
BGEZAL	Greater than or equal to 0 calls the procedure	MIPS32
BEQL	Equal is Likely to jump	MIPS32
BNEL	Not Likely to jump	MIPS32
BLEZL	Less than or equal to 0 is Likely to jump	MIPS32
BGTZL	Greater than 0 is Likely to jump	MIPS32
BLTZL	Less than 0 is Likely to jump	MIPS32
BGEZL	Greater than or equal to 0 is Likely to jump	MIPS32
BLTZALL	Less than 0 is Likely to call the procedure	MIPS32
BGEZALL	Greater than or equal to 0 is Likely to call the procedure	MIPS32

2.1.4 Coprocessor 0 instruction

The processor USES the Zero Coprocessor (CP0) register to manage memory and handle exceptions.

Table 2-8 Coprocessor 0 instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
DMFC0	Take a double word from register CP0	MIPS64
DMTC0	Write double characters to register CP0	MIPS64
MFC0	Fetch the word from the CP0 register	MIPS32
MTC0	Write into register CP0	MIPS32
TLBR	Read the TLB entry of the index	MIPS32
TLBWI	Write the TLB entry for the index	MIPS32
TLBWR	Let me write the random TLB terms	MIPS32
TLBP	Search for matches in TLB	MIPS32
TLBINV	Invalid TLB table entry specified	MIPS32
TLBINVF	Invalid all TLB table entries	MIPS32
The CACHE	Cache operation	MIPS32
ERET	Abnormal return	MIPS32
DERET	Debug exception return	MIPS32
DMFGC0	Fetch a double word from the VIRTUAL machine REGISTER CP0	MIPS64
DMTGC0	Write double characters to the VIRTUAL machine CP0 register	MIPS64
MFGC0	Fetch from the VIRTUAL machine CP0 register	MIPS32
MTGC0	Write to the VIRTUAL machine CP0 register	MIPS32
DI ¹	Disabling interrupts	MIPS32 R2
EI ²	Allow the interrupt	MIPS32 R2

Other instructions

In THE MIPS64, there are several other instructions in addition to those listed above.

Table 2-9 Other instructions: Special instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
<i>The SYSCALL</i>	<i>The system calls</i>	<i>MIPS32</i>
<i>BREAK</i>	<i>The breakpoint</i>	<i>MIPS32</i>
<i>HYPSCALL</i>	<i>Virtual machine system call</i>	<i>MIPS32</i>
<i>The SYNC</i>	<i>synchronous</i>	<i>MIPS32</i>
<i>SYNCl</i>	<i>Synchronous instruction cache</i>	<i>MIPS32 R2</i>
<i>RDPGPR</i>	<i>Read shadow register</i>	<i>MIPS32</i>
<i>WRPGPR</i>	<i>Write shadow register</i>	<i>MIPS32</i>
<i>SDBBP</i>	<i>Debugging breakpoints</i>	<i>MIPS32</i>
<i>RDHWR</i>	<i>User mode reads machine state</i>	<i>MIPS32</i>
<i>WAIT</i>	<i>Waiting for instructions</i>	<i>MIPS32</i>

Table 2-10 Other instructions: Exception trapped instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
<i>TGE</i>	<i>Greater than or equal to trapped</i>	<i>MIPS32</i>
<i>TGEU</i>	<i>The unsigned number is greater than or equal to trapped</i>	<i>MIPS32</i>
<i>TLT</i>	<i>Less than a</i>	<i>MIPS32</i>
<i>TLTU</i>	<i>The unsigned number is less than trapped</i>	<i>MIPS32</i>
<i>TEQ</i>	<i>Is equal to a</i>	<i>MIPS32</i>
<i>TNE</i>	<i>Differ in</i>	<i>MIPS32</i>
<i>TGEI</i>	<i>Greater than or equal to the immediate number trapped</i>	<i>MIPS32</i>
<i>TGEIU</i>	<i>Greater than or equal to unsigned immediate number trapped</i>	<i>MIPS32</i>
<i>TLTI</i>	<i>Less than immediate number trapped</i>	<i>MIPS32</i>
<i>TLTIU</i>	<i>Less than an unsigned immediate number is trapped</i>	<i>MIPS32</i>
<i>TEQI</i>	<i>Is equal to the number immediately trapped</i>	<i>MIPS32</i>
<i>TNEI</i>	<i>Does not equal to immediately number into</i>	<i>MIPS32</i>

Table 2-11 Other instructions: conditional move instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
<i>MOVf</i>	<i>Condition moves when floating point condition is false</i>	<i>MIPS32</i>
<i>MOVt</i>	<i>The condition moves when the floating point condition is true</i>	<i>MIPS32</i>
<i>MOVn</i>	<i>Condition moves when the general register is not 0</i>	<i>MIPS32</i>
<i>MOVz</i>	<i>The condition moves when the general register is 0</i>	<i>MIPS32</i>

Table 2-12 Other instructions: Prefetch instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
PREF	Prefetching instructions	MIPS32

Instruction mnemonic	Instruction function description	ISA Compatible Category
PREFX	Prefetching instructions	MIPS32

2.2 MIPS64 is compatible with floating point instruction set overview

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

2.2.1 FPU data type

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

Floating point data type

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

- Single-precisions, S)
- Double-precisions, D)
- 64-bit Paired Single-Precisions (PS)

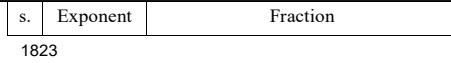
The floating-point coprocessor operates on single-precision floating-point Numbers (including pairs of single-precision floating-point Numbers for each single precision number) and double-precision floating-point Numbers in accordance with the ANSI/IEEE 754-1985 binary floating-point standards. The 32-bit single-precision format includes a 24-bit decimal field (S+F) represented by a "symbol + amplitude" and an 8-bit exponential field (E); 64-bit double format includes a 53-bit "symbol + amplitude"

Represents the decimal domain (S+F) and an 11-bit exponential domain (E); The 64-bit double precision (PS) format contains two single-precision floating point formats.

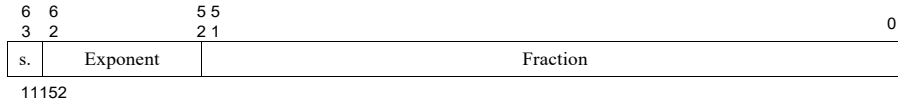
The three types of data formats are shown in Figure 2-1.

Figure 2-1 FPU floating point data format

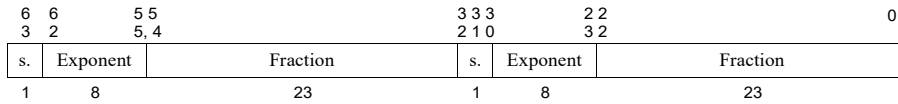
0



Single precision floating point number (S)



Double floating-point number (D)



Paired single precision floating point Numbers (PS)

The floating point format consists of three fields:

- The symbol field, S
- The index field with deviation, $e = e + \text{Bias}$, e is the index without deviation

The decimal region, $F = .bb..._{12} B - p_1$

The range of the exponent E without offset is the integer between all Emin and Emax plus the following two reserved values:

- Emin-1 (used to encode 0 and nonnormalized Numbers)
- Emax +1 (used to encode infinity and NaN[Not a

Number]) Table 2-13 defines the values of the

parameters associated with the floating-point format.

Table 2-13 floating point format parameters

parameter	Single precision	double
Emax	+ 127	+ 1023
Emin	- 126.	- 1022.
Exponential offset	+ 127	+ 1023
Index bits wide	8	11
Small digital wide	24	53

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

Table 2-14 Calculation of floating point values V

E	F	s.	b1	V
$E + 1_{\max}$	Ind	x	1	SNaN (Signaling NaN)

	icates a 0	x	0	QNaN (Quiet NaN)	
E + 1 _{max}	0	1	x	- up	Minus infinity
		0	x	+ up	Is infinite
[E _{min} , E _{max}]	x	1	x	- (2) (1 F) ^E	Negative normalized Numbers
		0	x	+ (2) (1 F) ^E	Normalized Numbers
E - 1 _{min}	Indicates a 0	1	x	. - (2) (0 F) ^{E_{min}-1}	Negative disnormalized Numbers
		0	x	. + (2) (0 F) ^{E_{min}-1}	Positive and non-normalized Numbers
E - 1 _{min}	0	1	x	0	Negative zero
		0	x	+ 0	Is zero

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 values

type	Single precision	double
The minimum number of	1.40129846 e - 45	4.9406564584124654 e - 324
Minimum normalized number	1.17549435 e - 38	2.2250738585072014 e - 308
Maximum number	3.40282347 e	1.7976931348623157 e+308

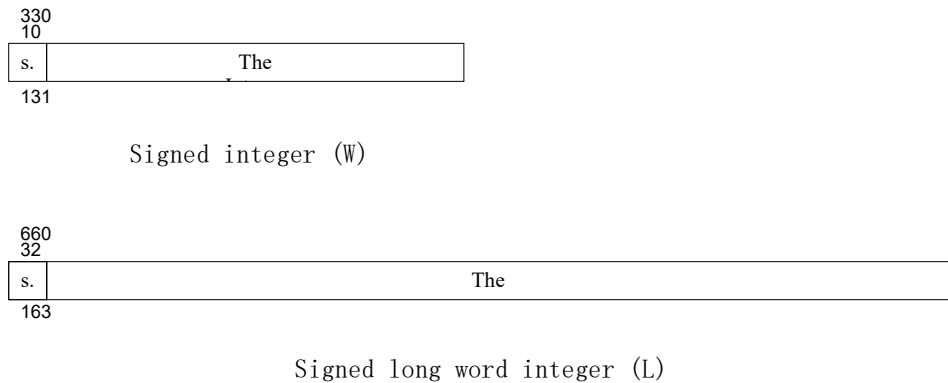
Fixed point data type

Fixed-point data supported by FPU are all signed integers and can be divided into two types:

- 32-bit signed integer (Word, W)

- The format of the two fixed-point data types of 64-bit signed Longword (L) is shown in Figure 2-2.

Figure 2-2 FPU fixed-point data format



2.2.2 Floating point register

Floating-point registers in GS464E inherit the MIPS R4000/R10000 processor usage, which is slightly different from the MIPS64 specification. When the FR bit of the Status control register is 0, GS464E has only 16 32-bit or 64-bit floating-point registers, and the floating-point register Numbers must be even. while

The 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,

The GS464E USES floating-point registers in the same way as the MIPS64 specification, with 32 64-bit floating-point registers.

2.2.3 Floating point control register

Floating point control registers in GS464E include:

- FIR, floating point implementation to define registers
- FCCR, floating point conditional code register
- FEXR, floating point exception register
- FENR, floating point enable register
- FCSR, floating point control/state register (often called FCR31)

Access to floating point control registers does not require a kernel mind. The software accesses the floating point control register through CFC1 and CTC1 instructions. In the floating point control registers mentioned above, the access registers FCCR, FEXR, and FENR actually access some of the domains of FCSR.

Floating point implementation define Register (FIR, CP1 Control Register 0)

The floating-point implementation definition register is a 32-bit read-only register that contains the functions of the floating-point unit implementation, such as processor ID, revision number, and so on.

Figure 2-3 illustrates the FIR register format; Table 2-16 describes each FIR register field.

Figure 2-3 FIR register format

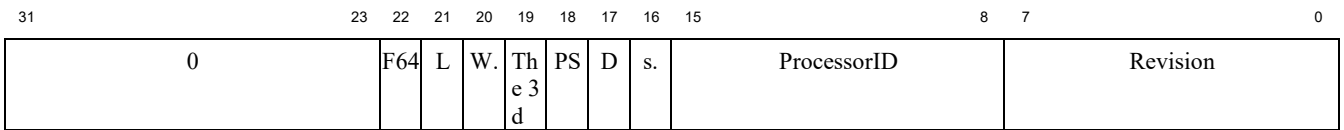


Table 2-16 Description of FIR register fields

Domain name	position	Functional description	Read/write	Reset value

Domain name	position	Functional description	Read/writable	Reset value
0	31.. 23	Read only is always 0.	0	0
F64	22	A constant of 1 indicates that the floating-point data path is 64-bit.	R	0 x1
L	21	Constant is 1, indicating the implementation of long word (L) fixed-point data type.	R	0 x1
W.	20	Constant is 1, indicating the implementation of the word (W) fixed-point data type.	R	0 x1
The 3 d	19	Constant 0 indicates that MIPS 3D ASE has not been implemented.	R	0 x0
PS	18	Constant 1 means that a pair of single-precision floating point data types are implemented.	R	0 x1
D	17	A constant of 1 means that the double precision floating point data type is implemented.	R	0 x1
s.	16	A constant of 1 means that a single precision floating point data type is implemented.	R	0 x1
ProccsorID	15.. 8	Floating-point coprocessor identifier.	R	0 x05
Revision	7.. 0	Floating-point coprocessor version number.	R	0 x01

Floating point Control and Status Register (FCSR, CP1 Control Register 31)

The FCSR register is used to control the operation of the floating point unit and to represent some state.

Figure 2-4 illustrates the FIR register format; Table 2-17 describes each FIR register field.

Figure 2-4. FCSR register format

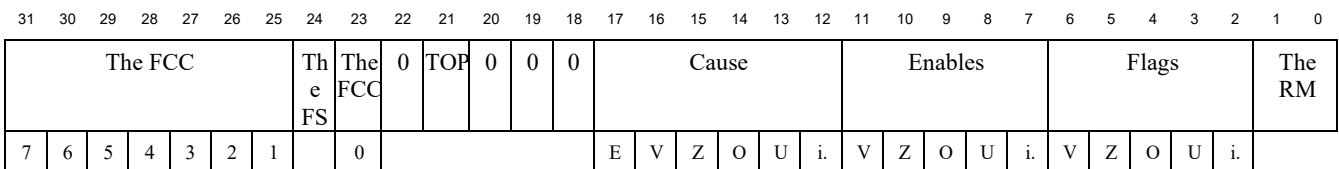


Table 2-17 Describes the FCSR register fields

Domain name	position	Functional description	Read/writable	Reset value
The FCC	31.. 25 23,	Floating point conditional code. Records floating-point comparison results for conditional jumps or transitions. When a floating-point comparison operation is performed When it occurs, the result is saved in the specified CC bit, the conditional bit. If the comparison is true, the CC bit is set to 1; If not, set 0.	R/W	0 x0
The FS	24	Brush to the 0 identification bit. When set to 1, the nonnormalized result is set to 0. Otherwise the result is nonnormalized The number will trigger an unimplemented exception.	R/W	0 x0
0	22	Read only is always 0.	0	0
TOP	21	Floating point register TOP mode control bit. When the bit is 1, the register representing the floating-point instruction takes TOP The way the pattern is encoded; When the bit is 0, the register representing the floating-point instruction is still encoded in the normal way.	R/W	0 x0

0	20.. 18	Read only is always 0.	0	0
---	---------	------------------------	---	---

Domain name	position	Functional description	Read/write	Reset value
Cause	17.. 12	<p>These bits reflect the results of the most recent instruction execution. The Causes field is a logical extension of the coprocessor's 0 Cause register, and these bits indicate the exception caused by the last floating-point operation and produce an interrupt or exception if the corresponding Enable bit is set. If more than one exception is produced in an instruction, each corresponding exception causes the bit to be set.</p> <p>The Causes field can be overridden by every floating-point operation instruction (not including Load, Store, and Move). If software simulation is needed to complete, the unrealized operation bit (E) of the operation will be set to 1; otherwise, it will remain at 0. The other bits are set to 1 or 0 according to the IEEE754 standard depending on whether the corresponding exception is made.</p> <p>When a floating point exception occurs, no result will be stored, and the only state affected is the Causes field.</p>	R/W	0 x0
Enables	11.. 7	<p>Any time the Cause and the corresponding Enable bit are both 1, a floating point exception is generated. If a floating point operation sets a Cause that is allowed to be activated (with the corresponding Enable bit of 1), the processor immediately makes an exception, just as if the CTC1 instruction were used to set the Cause bit and Enable bit to 1 at the same time.</p> <p>There is no enabling bit for unimplemented operation (E), which always produces a floating point exception if set.</p> <p>Before returning from a floating point exception, the software must first clear the activated Cause with a CTC1 instruction to prevent the repeated execution of the interrupt. Thus, a program running in user mode will never observe that the value of the enabled Cause is 1; If the user-mode handler needs to get this information, the contents of the Cause must be passed somewhere other than in the status register.</p> <p>If a floating-point operation sets only the Cause that is not enabled (the corresponding enabled bit is 0), no exceptions occur, and the default result defined by the IEEE754 standard is written back. In this case, the previous floating point Causes the exception to be determined by reading the value of the Causes field.</p>	R/W	0 x0
Flags	6.. 2	<p>The flag bits are cumulative, indicating that an exception has occurred since the last time they were explicitly reset. If an IEEE754 exception is generated, the corresponding Flag bit is set to 1, otherwise it remains unchanged, so the bits are never cleared for floating point operations. But we can write a new value to the state with the CTC1 command</p> <p>Register to set or clear Flag bits.</p>	R/W	0 x0
The RM	1.. 0	Rounding - in mode control domain. Table 2-18 further describes the encoding of RM.	R/W	0 x0

Table 2-18 Rounding mode (RM) encoding

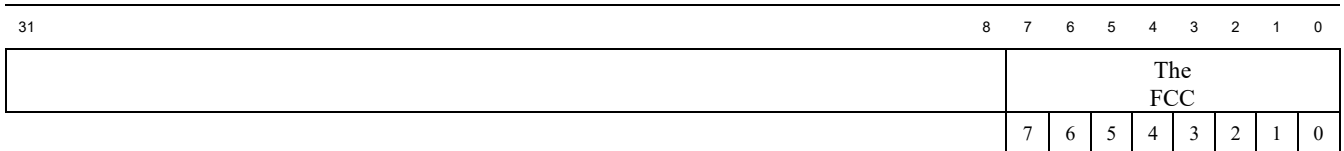
Rounding mode (RM)	mnemonics	describe
0	RN	Rounding the result in the direction closest to the representable number, the result is the same when the two nearest representable Numbers are separated. When approaching, round off to the nearest number direction with the lowest value of 0.
1	RZ	Rounding in the 0 direction: rounding the result to the nearest number and not greater than it in absolute value into the.

2	The RP	Round off in the direction of positive infinity: round off the result to the number closest to and not less than it.
3	The RM	Round off in the direction of negative infinity: round off the result to the number closest to and not greater than it.

Floating point conditional 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 FCSR, except that the FCC bit in this register is continuous. Figure 2-5 illustrates the format of the FCCR register.

Figure 2-5 FCCR register format

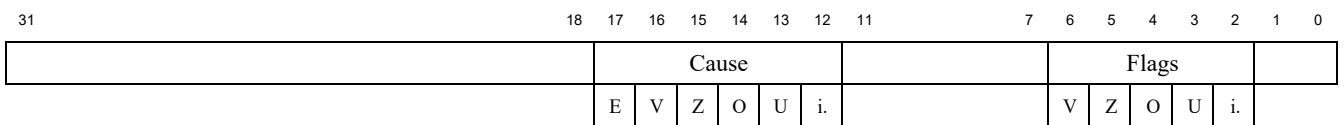


Floating point Exception Register (FCCR, CP1 Register 26)

The FEXR register is another way to access the Cause and Flags fields, and its contents are identical to the corresponding fields in FCSR. Figure 2-6

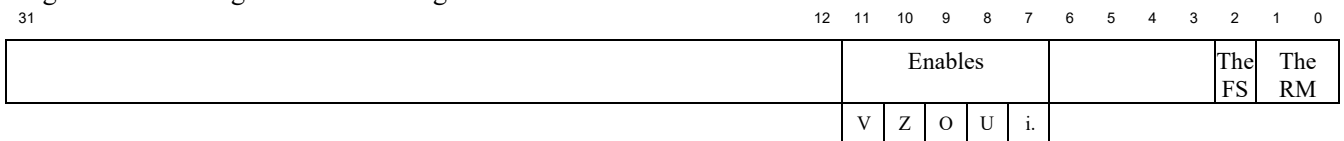
The format of the FEXR register is explained.

Figure 2-6 FEXR register format



Floating-point Enable Register (FCCR, CP1 Register 28)

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



2.2.4 Floating-point exception

Floating point exceptions occur when the FPU does not handle operands or the results of floating point calculations in a regular way, and the FPU produces an exception to start the software trap or to set the status flag bit.

The control and state registers of the FPU contain an enablement bit for each exception that determines whether an exception can cause the FPU

Start an exception trap or set a status flag.

If a trap is started, the FPU reserves the operation in the starting state and starts the software exception processing path. If no trap is started, an appropriate value is written to the FPU target register and the calculation continues.

FPU supports five IEEE754 exceptions:

- Inexact. Inexact (I)
- Underflow Underflow (U)
- Overflow 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 the standard MIPS floating point structure, including when the FPU cannot determine the correct exception behavior. This exception indicates the execution of the software exception handling. An unimplemented exception has no enabling signals and flag bits, and when this exception occurs, a corresponding unimplemented exception trap occurs.

Each of the five exceptions to IEEE754 (V, Z, O, U, I) corresponds to a user-controlled exception trap that is allowed to occur when one of the five enabling bits is set. When an exception occurs, the corresponding Cause bit is set. If the corresponding Enable bit is not set, the exception Flag bit is set. If the enable bit is set, the flag bit is not set, and the FPU produces an exception to the CPU. Subsequent exception handling allows the exception trap to occur.

When there is no exception trap signal, the floating-point processor handles it by default, providing an alternate value for the result of the floating-point calculation. Different exception types determine different default values. Lists the FPU default handling for each IEEE exception.

Table 2-19 Default handling of floating point exceptions

The domain	describe	Rounding mode	The default action
i.	The precise	Any pattern	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
		The RP	Correct positive underflow to a minimum positive number and negative underflow to minus 0
		The RM	Correct the negative underflow to the minimum negative number, and correct the positive underflow to +0
O	The overflow	RN	Set the result to infinity according to the sign of the intermediate result
		RZ	Set the result to the largest number according to the sign of the intermediate result
		The RP	Correct the negative underflow to a maximum negative number, and correct the positive underflow to $+\infty$
		The RM	Correct positive underflow to the largest integer and negative underflow to minus infinity
Z	Be zero except	Any pattern	Provides a corresponding signed infinite number
V	Illegal operation	Any pattern	Provide a Quiet Not a Number(QNaN)

The conditions that cause the FPU to produce each exception are described below, and the FPU's response to each exception's cause condition is detailed. Imprecise exception (I)

The FPU produces an imprecise exception when:

- Rounding results are imprecise
- The rounding result overflows
- Rounding results underflow, and the underflow and imprecise enablement bits are not set, and the FS bits are set.

Trap enabled results: If an imprecise exception trap is enabled, the result register is not modified, and the source register is retained. Because this pattern of execution affects performance, the imprecise exception trap is enabled only when necessary.

Trap unenabled results: If no other software trap occurs, rounding or overloading results are sent to the target

register. Illegal Operation Exception (V)

The illegal operation exception signals when two or one of the operands of an executable operation is illegal. If the exception does Not fall, MIPS defines the result as a Quiet Not a Number (QNaN). Illegal operations include:

- Addition or subtraction: infinite subtraction. For example, $(+\infty)+(-\infty)$ or $(-\infty)-(-\infty)$.
- Multiplication: 0 times infinity, for all positive and negative Numbers
- Division: 0 over 0, infinity over infinity, for all positive and negative Numbers
- The number of comparison operations that do not process Unordered is Unordered
- Performs a floating-point comparison or conversion on an indicator, NaN
- Any mathematical operation for SNaN (Signaling NaN). When one of the operands is SNaN or both of them are SNaN

Causes this exception (MOV operations are not considered mathematical operations, but ABS and NEG are considered mathematical operations)

- Square root of X, when X is less than 0

Software can simulate exceptions to other illegal operations for a given source operand. For example, in IEEE754, software is used to realize a specific function: X REM Y, where Y is 0 or X is infinite; Or overflows when floating point Numbers are converted to decimal, which is infinity or NaN; Or a prior function like ln of 5 or cosine of 3. -1

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

The trap does not enable the result: If no other exception occurs, QNaN is sent to the target register. Except zero (Z)

In a division operation, the exception to zero signals when the divisor is zero and the dividend is a finite non-zero data. Software can be used to simulate all but zero exceptions for other operations that produce signed infinity values, such as ln(0), sin(/2), cos(0), or 0. -1

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

Trap non-enabling: If no trap occurs, the result is a signed infinity. Overflow exception (O)

When the magnitude of the rounded floating-point result is represented by an unbounded exponent, larger than the finite data represented by the maximum target pattern, the overflow exception signals the notification. (This exception sets both imprecise exceptions and flags.)

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

Trap non-enablement: If no trap occurs, the final result is determined by the symbol of the rounding pattern and the intermediate result. Downflow exception (U)

Two related events lead to the downflow exception:

- A small non-zero result between $\pm 2E_{min}$, which is so small that it leads to a downflow exception.
- Severe data distortion approximated by a Denormalized Number from the two small Numbers. IEEE754 allows many different ways to detect these events, but requires the same method for all operations. Little data

It can be detected in one of the following ways:

- After rounding (if a non-zero data is calculated without a bound in the exponential range, it should be strictly between $\pm 2E_{min}$)
- Pre - rounding (if a non-zero data is calculated without boundaries between exponential and precision ranges, it should be strictly within \pm Between 2 emin)

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

- The distortion of a nonnormalized number (when the resulting result is different from that calculated when the

exponent is not bounded)

- Imprecise data (when the resulting results are not bounded by the exponential and precision ranges)

MIPS structures require precision distortion to be detected to produce inaccurate results.

Trap enabled: If an overflow or imprecise exception is enabled, or if the FS bit is not set, an unimplemented exception is generated and the result register is not modified.

Trap not enabled: If overflow or imprecise exceptions are not enabled, and the FS bit is set, the final result is determined by the symbol bit of the rounding mode and the immediate result.

Unimplemented operation Exception (E)

Unimplemented operation in the FPU control/state register when any of the opcodes or operation format instructions reserved for future definitions are executed

Action causes bits to be set and traps are created. The source operands and destination registers remain unchanged while the instructions are simulated in the software. Any exception in IEEE754 can be generated from a simulation operation, and these exceptions can in turn be simulated. In addition, exceptions to unimplemented instructions can occur when the hardware fails to perform some rare operation or result condition correctly. These include:

- Nonnormalized Operand, except for comparison instructions
- Quite Not a Number operand (QNaN), with the exception of comparison instructions
- Nonnormalized Numbers or underflows, and overflows or imprecise enabling signals are set while FS bits are not set

Note: disnormalized Numbers and NaN operations only fall into traps in conversion or computation instructions, not in MOV instructions. Trap enabled: The original operation data is not sent.

Trap unenabled: This trap cannot be unenabled.

2.2.5 MIPS64 is compatible with floating point instruction list

The relevant instructions of MIPS64 compatible floating-point coprocessor implemented by GS464E are divided into the following categories according to their functions:

- Operation instruction
- Branch jump instruction
- More instructions
- Transformation instruction
- Mobile instruction
- To fetch instruction

These instructions are listed class by class below.

Floating-point access instruction

Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
LDC1	Access double words from memory						MIPS32
LDXC1	Access double words from memory by index						MIPS64
LUXC1	Access double words from memory by unaligned index						MIPS64
LWC1	Access words from memory						MIPS32
LWXC1	Access words by index						MIPS64
SDC1	Save double word to memory						MIPS32
SDXC1	Store double words in memory by index						MIPS64
SUXC1	Unaligned index to store double words in memory						MIPS64
SWC1	Save words to memory						MIPS32
SWXC1	Store words into memory by index						MIPS64

Floating
point
instruction

Table 2-21 Floating-point operation instructions

Instruction mnemonic	Instruction function description	FMT	ISA Compatible Category
----------------------	----------------------------------	-----	-------------------------

		s.	D	PS	L	W.	
ABS. FMT	The absolute value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32
ADD the FMT	add	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32
DIV. FMT	division	<input type="checkbox"/>	<input type="checkbox"/>	-	-	-	MIPS32
MADD. FMT	By adding	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS64 MIPS32 R2
MSUB. FMT	By reducing	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS64 MIPS32 R2
The MUL. FMT	The multiplication	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32
NEG. FMT	complementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32
NMADD. FMT	Multiply and add and find the reverse	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS64 MIPS32 R2
NMSUB. FMT	Multiply or subtract and find the reverse	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS64 MIPS32 R2
RECIP. FMT	For the bottom	<input type="checkbox"/>	<input type="checkbox"/>	-	-	-	MIPS64 MIPS32 R2
RSQRT. FMT	Take the inverse of the square root	<input type="checkbox"/>	<input type="checkbox"/>	-	-	-	MIPS64 MIPS32 R2
SQRT. FMT	The square root	<input type="checkbox"/>	<input type="checkbox"/>	-	-	-	MIPS32
SUB. FMT	subtraction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32

Floating point
branch jump
instruction

Table 2-22 Floating-point branch jump instructions

Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
BC1F	Jump when floating point conditional bit is false						MIPS32
BC1FL	Likely jump when floating point conditional bit false						MIPS32
BC1T	Floating point conditional bit true time jump						MIPS32
BC1TL	Floating point conditional bit true time Likely jump						MIPS32

Floating point
comparison
instruction

Table 2-23 Floating-point branch jump instructions

Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
Arthur c. ond. FMT	Compare floating-point values juxtaposed to conditional bits	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	-	-	MIPS32

Floating point
conversion
instruction

Table 2-24 Floating-point branch jump instructions



Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
ALNV. PS	Variable floating-point alignment	-	-	□	-	-	MIPS64
CEIL. L.f mt	Floating-point conversion to a 64-bit fixed point. Round up	□	□	-	-	-	MIPS64
CEIL. W.f mt	Floating-point conversion to a 32-bit point. Round up	□	□	-	-	-	MIPS64
The CVT transmission. D.f mt	To convert to a floating point or fixed point	□	-	-	□	□	MIPS32
The CVT transmission. L.f mt	Converts floating point values to 64-bit fixed points	□	□	-	-	-	MIPS64
The CVT transmission. PS. S	Converts two floating point values to floating point pairs	□	-	-	-	-	MIPS64

Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
The CVT transmission. Supachai panitchpakdi L	Converts a floating-point pair to a single-precision floating-point pair	-	-	□	-	-	MIPS64
CVT's U	Converts the high point of a floating point pair to a single precision floating point	-	-	□	-	-	MIPS64
The CVT transmission. S. mt	To convert to a floating point or fixed point	-	□	-	□	□	MIPS32
The CVT transmission. W.f mt	Converts floating point values to 32-bit points	□	□	-	-	-	MIPS32
FLOOR. L.f mt	Floating-point conversion to a 64-bit fixed point, rounded down	□	□	-	-	-	MIPS64
FLOOR. W.f mt	Floating-point conversion to a 32-bit point, round down	□	□	-	-	-	MIPS64
PS PLL.	Merge the low bits of two floating point pairs to make a new floating point pair	-	-	□	-	-	MIPS64
PLU. PS	Merge the low and high bits of two floating point pairs to make a new floating point pair	-	-	□	-	-	MIPS64
PUL. PS	Merge the high and low bits of two floating point pairs to make a new floating point pair	-	-	□	-	-	MIPS64
PUU. PS	Merge the high bits of two floating point pairs to make a new floating point pair	-	-	□	-	-	MIPS64
ROUND. L.f mt	Round a floating-point number to a 64-bit fixed point	□	□	-	-	-	MIPS64
ROUND. W.f mt	Round a floating-point number to a 32-bit point	□	□	-	-	-	MIPS32
TRUNC. L.f mt	Rounds a floating-point number to a 64-bit fixed point in the direction of a smaller absolute value	□	□	-	-	-	MIPS64
TRUNC. W.f mt	Rounds a float to a 32-bit point in the direction of a smaller absolute value	□	□	-	-	-	MIPS32

Floating point movement instruction

Table 2-25 Float branch jump instructions

Instruction mnemonic	Instruction function description	FMT					ISA Compatible Category
		s.	D	PS	L	W.	
CFC1	Read the floating point control register to GPR	/					MIPS32
CTC1	Write floating-point control registers to GPR	/					MIPS32
DMFC1	Copy double characters from FPR to GPR	/					MIPS64
DMTC1	Copy double word from GPR to FPR	/					MIPS64
MFC1	Copy low characters from FPR to GPR	/					MIPS32
MFHC1	Copy high characters from FPR to GPR	/					MIPS32 R2
MOV. FMT	Copy FPR	□	□	□	-	-	MIPS32
MOV. FMT	Copy FPR when floating point false	□	□	□	-	-	MIPS32
MOVN. FMT	FPR is replicated when GPR is not 0	□	□	□	-	-	MIPS32
MOVT. FMT	Floating point true time copy FPR	□	□	□	-	-	MIPS32
MOVZ. FMT	Copy FPR when GPR is 0	□	□	□	-	-	MIPS32

	MFC1 Copy low characters from GPR to FPR		MIPS32
	MFFC1 Copy high characters from GPR to FPR	龙芯 3A3000/3B3000 处理器用户手册	MIPS32 R2 下册

2.3 An overview of THE MIPS64 DSP instruction set

GS464 compatible implementation of MIPS64 DSP ASE (r2.34 version). For a detailed description of The DSP implemented instructions, refer to The MIPS® Architecture for Programmers VolumeIV-e: The MIPS® DSP Application-specific Extension to The MIPS64® Architecture (R2.34).

2.3.1 MIPS64 DSP ASE compatible instruction list

MIPS64 DSP ASE compatible floating-point coprocessor instructions implemented by GS464E are divided into the following categories according to their functions:

- Operation-class instruction
- Shift class instruction based on general purpose register
- Multiplication class instruction
- Bit operation class instruction
- Compare - Extract class instructions
- Accumulator operations access class instructions
- DSP controls register access class instruction
- Access class instruction with index register
- Branch instruction
- DSP instructions no longer

supported by MIPS are listed on a class-by-class basis.

Operation-class instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
ADDQ.PH	Vector (right 2) decimal half word plus	MIPS DSP
ADDQ.S.PH	Vector (right 2) decimal half - word saturation plus	MIPS DSP
ADDQ.S w.	Sign word saturation plus	MIPS DSP
ADDU.QB	Vector (right 4) decimal byte unsigned plus	MIPS DSP
ADDU.S.QB	Vector (right 4) decimal byte unsigned saturation plus	MIPS DSP
ADDUH.QB	The vector is unsigned (4 right-most) bytes plus, the result is divided by 2, and the result is extended	MIPS DSP R2
ADDUH.R.QB	Vector unsigned (right-most 4) bytes rounded to add, result divided by 2, result symbol extended	MIPS DSP R2
ADDU.PH	Vector (right 2) decimal halfword unsigned plus	MIPS DSP R2
ADDU.S.PH	Vector (right 2) decimal halfword unsigned saturation plus	MIPS DSP R2
ADDQH.PH	Vector (rightmost 2) halfword addition, the result is divided by 2, the result symbol expansion	MIPS DSP R2
ADDQH.R.PH	Vector (rightmost 2) half-word rounding, the result is divided by 2, the result symbol extension	MIPS DSP R2
ADDQH w.	The right-most word of the vector is added, the result is divided by 2, and the result symbol is extended	MIPS DSP R2
ADDQH.R w.	The right-most word of the vector is rounded, the result is divided by 2, and the result is extended	MIPS DSP R2
SUBQ.PH	Vector (right 2) decimal half word minus	MIPS DSP
SUBQ.S.PH	Vector (right 2) decimals half - word saturation minus	MIPS DSP
SUBQ.S w.	Sign word saturation minus	MIPS DSP
SUBU.QB	Vector (right 4) decimal byte unsigned minus	MIPS DSP
SUBU.S.QB	Vector (right 4) decimal byte unsigned saturation minus	MIPS DSP

SUBUH. QB	The vector is unsigned (4 to the right), the result is subtracted, the result is divided by 2, the result is extended	MIPS DSP R2
SUBUH_R. QB	The vector is unsigned (4 right-most) bytes rounded down, the result is divided by 2, and the result is extended	MIPS DSP R2
SUBU. PH	Vector (right 2) decimal halfword unsigned minus	MIPS DSP R2
SUBU_S. PH	Vector (right 2) decimal halfword unsigned saturation minus	MIPS DSP R2
SUBQH. PH	Vector (rightmost 2) half-word subtraction, the result divided by 2, the result symbol expansion	MIPS DSP R2

Instruction mnemonic	Instruction function description	ISA Compatible Category
SUBQH_R. PH	Vector (right-most 2) half-word rounding, result dividing by 2, result symbol expanding	MIPS DSP R2
SUBQH w.	Vector right-most word subtracts, the result divides by 2, the result symbol expands	MIPS DSP R2
SUBQH_R w.	The right-most word of the vector is rounded down, the result is divided by 2, and the result is extended	MIPS DSP R2
ADDSC	A signed word is added and carries	MIPS DSP
ADDWC	A signed word with carry plus	MIPS DSP
MODSUB	Use index values for schema subtraction	MIPS DSP
RADDU.W.Q B	(far right) 4 bytes unsigned accumulation	MIPS DSP
ABSQ_S. QB	The vector takes the (right-most 4) byte absolute value and saturates it, resulting in symbol expansion	MIPS DSP R2
ABSQ_S. PH	Vector take (rightmost 2) the absolute value of half word, and do saturation operation, the result symbol expansion	MIPS DSP
ABSQ_S w.	The vector takes the absolute value of the (rightmost) word and saturates it, resulting in symbol expansion	MIPS DSP
PRECR. QB. PH	Vector integer precision reduced from (rightmost two) halfwords to 4 bytes	MIPS DSP R2
PRECRQ. QB. PH	Vector decimal precision reduced from (rightmost two) half words to 4 bytes	MIPS DSP
PRECR_SRA. PH. W	Vector right shift integer precision reduction from (rightmost) word to two half-words, resulting symbol expansion	MIPS DSP R2
PRECR_SRA_R. PH. W	Vector right shift integer precision reduction from (rightmost) word to two half-words, and do rounding, result character Extension number	MIPS DSP R2
PRECRQ. PH. W	Vector decimal precision reduced from (right) word to (two) half word	MIPS DSP
PRECRQ_RS. PH. W	Vector decimal precision reduced from (right) word to (two) half word, and done with saturation and truncation Into the	MIPS DSP
PRECRQU_S. QB. PH	Vector decimal precision reduced from (rightmost two) halfwords to 4 unsigned bytes	MIPS DSP
PRECEQ. W.P HL	Vector decimal precision extension, from (second right) half - word to word, resulting symbol extension	MIPS DSP
PRECEQ. W.P HR	Vector decimal precision extension, from (right-most) half word to word, resulting symbol extension	MIPS DSP
PRECEQU. PH. QBL	Vector decimal precision extension, from (two to the right) unsigned bytes to two half-words	MIPS DSP
PRECEQU. PH. QBR	Vector decimal precision extension, from (rightmost two) unsigned bytes to two half-words	MIPS DSP
PRECEQU. PH. QBLA	Vector decimal precision extension from unsigned bytes to two A half word	MIPS DSP
PRECEQU. PH. QBRA	Vector decimal precision extension from unsigned bytes to two A half word	MIPS DSP
PRECEU. PH. QBL	Vector integer precision extension, from unsigned bytes to unsigned halfwords	MIPS DSP
PRECEU. PH. QBR	Vector integer precision extension, from (rightmost two) unsigned bytes to unsigned halfwords	MIPS DSP
PRECEU. PH. QBLA	Vector integer precision extension from unsigned bytes to two A half word	MIPS DSP
PRECEU. PH. QBRA	Vector integer precision extension from unsigned bytes to two (the right side of the rightmost word intersects two)	MIPS DSP

	A half word	
--	-------------	--

Shift class instruction based on general purpose register

Instruction mnemonic	Instruction function description	ISA Compatible Category
SHLL. QB	Vector logic shifts left (right-most 4) bytes, the shift value is specified by the immediate number, the resulting symbol extension	MIPS DSP
SHLL. PH	Vector logic moves left (rightmost 2) halfwords, the shift value is specified by the immediate number, the resulting symbol extension	MIPS DSP

Instruction mnemonic	Instruction function description	ISA Compatible Category
SHLLV. QB	Vector logic shifts 4 bytes to the right, the shift value is specified by the register, the resulting symbol extension	MIPS DSP
SHLLV. PH	Vector logic moves left (rightmost 2) halfwords, the shift value is specified by the register, the result symbol extension	MIPS DSP
SHLL_S. PH	Vector logic saturation left shift (right by 2) halfwords, shift value specified by immediate number, result Sign extension	MIPS DSP
SHLL_S w.	Vector logic saturates left-shift (right-most) words with a shift value specified by an immediate number, resulting in symbol expansion show	MIPS DSP
SHLLV_S. PH	Vector logic saturates the left (rightmost 2) halfword, the shift value is specified by the register, the result Sign extension	MIPS DSP
SHLLV_S w.	Vector logic saturates left-shift (right-most) words, shift values specified by registers, resulting in symbol expansion show	MIPS DSP
SHRL. QB	Vector logic moves to the right (right-most 4 bytes), the shift value is specified by the immediate number, the resulting symbol extension	MIPS DSP
SHRL. PH	Vector logic moves right (rightmost 2) halfwords, the shift value is specified by the immediate number, the resulting symbol extension	MIPS DSP R2
SHRLV. QB	Vector logic is shifted to the right of four bytes, the shifted value is specified by the register, the resulting symbol extension	MIPS DSP
SHRLV. PH	Vector logic right shift (rightmost 2) halfword, shift value specified by register, result symbol extension	MIPS DSP R2
SHRA. QB	Vector arithmetic right shift (4 right-most bytes), shift value specified by immediate number, resulting symbol extension	MIPS DSP R2
SHRA_R. QB	Vector arithmetic is right-shifted (right-most four) bytes, with the shift value specified by the immediate number and rounded, Result symbol extension	MIPS DSP R2
SHRAV. QB	Vector arithmetic right shift (4 right-most bytes), shift value specified by register, result symbol extension	MIPS DSP R2
SHRAV_R. QB	Vector arithmetic rounding off to the right (4 rarest) bytes, the shift value specified by the register, the result Sign extension	MIPS DSP R2
SHRA. PH	Vector arithmetic right shift (rightmost 2) halfword, shift value specified by immediate number, resulting symbol extension	MIPS DSP
SHRAV. PH	Vector arithmetic right shift (rightmost 2) halfword, shift value specified by register, result symbol extension	MIPS DSP
SHRA_R. PH	Vector arithmetic rounding off a right-shifted (rightmost 2) halfword, with a shift value specified by an immediate number, results Sign extension	MIPS DSP

SHRAV_R.PH	Vector arithmetic rounding to the right (rightmost 2) halfword, shift value specified by register, result Sign extension	MIPS DSP
SHRA_R.w.	Vector arithmetic rounds right-shifted (right-most) words, with the shift value specified by the immediate number, resulting in symbol expansion show	MIPS DSP
SHRAV_R.w.	Vector arithmetic rounding off right-shifted (right-most) words, shifted values specified by registers, resulting in symbol expansion show	MIPS DSP

Multiplication class instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
MULEU_S. PH. QBL	Vector (two to the right) bytes unsigned times (two to the right) halfwords, resulting in two Half word	MIPS DSP
MULEU_S. PH. QBR	Vector (rightmost two) bytes unsigned times (rightmost two) halfwords, resulting in two Half word	MIPS DSP
MULQ_RS. PH	Vector (rightmost two) halfwords are saturated and rounded, resulting in half-words	MIPS DSP
MULEQ_S. W.P HL	A sign (second right) half-word saturates, resulting in a word	MIPS DSP
MULEQ_S. W.P HR	A signed (right-most) half-word saturates, resulting in a word	MIPS DSP
DPAU. H.Q BL	The vector integer (two to the right) bytes are unsigned multiplied and accumulated, and the result is accumulated with ACC	MIPS DSP
DPAU. H.Q BR	The vector integer (rightmost two) bytes are unsigned multiplied by the sum, and the result is multiplied by the acc	MIPS DSP
DPSU. H.Q BL	Vector integer (the leftmost two) bytes unsigned multiplied by the sum, and the result multiplied by acc	MIPS DSP
DPSU. H.Q BR	The vector integer (rightmost two) bytes are unsigned multiplied by the sum, and the result is multiplied by the acc	MIPS DSP
The DPA. W.P H	Vector integer (rightmost two) half word multiplication sum, finally with ACC sum	MIPS DSP R2
DPAX. W.P H	The vector integer (two to the right) is multiplied by the sum, and the result is multiplied by acc	MIPS DSP R2
DPAQ_S. W.P H	The vector decimals (the rightmost two) are multiplied by half words, and the result is multiplied by ACC	MIPS DSP R2
DPAQX_S. W.P H	The vector decimal (two to the right) is multiplied by half word, the result is accumulated after saturation, and then accumulated with ACC	MIPS DSP R2
DPAQX_SA. W.P H	Vector decimals (right-most two) half-word multiply, the result is summing after saturated rounding, and then cumulative with ACC add	MIPS DSP R2
The DPS. W.P H	Vector integer (rightmost two) half word multiplication sum, the result and ACC subtraction	MIPS DSP R2
DPSX. W.P H	The vector integer (two to the right) is multiplied by the sum, and the result is subtracted from ACC	MIPS DSP R2
DPSQ_S. W.P H	The vector decimals (the right-most two) are multiplied by half words and subtracted from ACC	MIPS DSP
DPSQX_S. W.P H	The vector decimal (two to the right) is multiplied by half word, the result is accumulated after saturation, and then subtracted from ACC	MIPS DSP R2
DPSQX_SA. W.P H	Vector decimals (right-most two) half-word multiply, the results are rounded to saturation and then add up to acc Reduction of	MIPS DSP R2
MULSAQ_S. W.P H	The vector decimals (right two) are multiplied and subtracted, and the result is added to ACC	MIPS DSP
DPAQ_SA. L.W	Vector small number multiplication, the result after saturated rounding, and then summing with ACC	MIPS DSP
DPSQ_SA. L.W	Multiply the vector by a small number, the result is accumulated after saturated rounding, and the result is accumulated with ACC	MIPS DSP
MAQ_S. W.P HL	The vector decimal (to the right) is multiplied by half a word, and the result is added to ACC	MIPS DSP
MAQ_S. W.P HR	The vector decimal (far right) is multiplied by half a word and the result is added to ACC	MIPS DSP
MAQ_SA. W.P HL	Vector decimal (second right) half-word multiplication, the result is to take the saturated rounding and then add with ACC	MIPS DSP

MAQ_SA. W.P HR	Vector decimal (right-most) half-word multiplication, the result is to take the saturated rounding and then add with ACC	MIPS DSP
The MUL. PH	Vector (rightmost 2) halfword multiplication, the result is 16 bits lower written to register	MIPS DSP R2
MUL_S. PH	Vector (rightmost 2) halfwords with sign saturation multiply, the result is a low 16 bit write register	MIPS DSP R2
MULQ_S. PH	Vector (rightmost two) halfword saturation times, the result is half word	MIPS DSP R2
MULQ_S w.	Vector right-most word saturation times, the result is word	MIPS DSP R2
MULQ_RS w.	The right-most word of the vector is saturated and rounded, and the result is word	MIPS DSP R2
MULSA. W.P H	The vector (rightmost two) is multiplied and subtracted, and the result is added to ACC	MIPS DSP R2
MADD	Multiply the 32-bit signed fixed point number by ACC	MIPS32
MADDU	The 32-bit unsigned fixed-point number multiplication is accumulated with ACC	MIPS32
MSUB	The 32-bit signed fixed point number times acc minus	MIPS32
MSUBU	The 32-bit unsigned fixed point number multiplication is subtracted from ACC	MIPS32

Instruction mnemonic	Instruction function description	ISA Compatible Category
MULT	Word multiplication, result saved in ACC register	MIPS32
MULTU	Unsigned word multiplication, the result is saved to acc register	MIPS32

Bit operation class instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
BITREV	The halfbyte is flipped, and the result is 0 extension	MIPS DSP
INSV	Variable bit field insertion	MIPS DSP
REPL. QB	The vector copies the immediate number (integer) to the rightmost four bytes, resulting in symbol extension	MIPS DSP
REPLV. QB	The vector copies the bytes to the rightmost four bytes, resulting in symbol extension	MIPS DSP
REPL. PH	The vector copies the immediate number (integer) to the rightmost two and a half, resulting in symbol extension	MIPS DSP
REPLV. PH	The vector copies the halfword to the rightmost 2 halfwords, resulting in symbol expansion	MIPS DSP

Compare - Extract class instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
CMPU. EQ. QB	Vector (right 4) unsigned byte equal comparison, result set condition bit	MIPS DSP
CMPU. LT. QB	Vector (right-most 4) unsigned bytes less than comparison, result set condition bit	MIPS DSP
CMPU. LE. QB	Vector (right-most 4) unsigned bytes less than or equal to the comparison, the result set condition bit	MIPS DSP
CMPGDU. EQ. QB	Vector (right-most 4) unsigned byte equality comparison, the results are both conditional and universal register	MIPS DSP R2
CMPGDU. LT. QB	Vector (right-most 4) unsigned bytes less than comparison, the results are both conditional and generic register	MIPS DSP R2
CMPGDU. LE. QB	Vector (right-most 4) unsigned bytes less than or equal to the comparison, the result is simultaneously conditional bit and General purpose register	MIPS DSP R2
CMPGU. EQ. QB	Vector (right 4) byte equal comparison, the result of the general purpose register	MIPS DSP
CMPGU. LT. QB	Vector (right-most 4) bytes less than comparison, result in general purpose register	MIPS DSP
CMPGU. LE. QB	Vector (right-most 4) bytes less than or equal to the comparison, the result of the general purpose register	MIPS DSP
CMP. EQ. PH	Vector (rightmost 2) half - word equal comparison, the result set condition bit	MIPS DSP
CMP. LT. PH	Vector (rightmost 2) half character is less than comparison, result sets condition bit	MIPS DSP
CMP. LE. PH	Vector (rightmost 2) half - word less than or equal to the comparison, the result set condition bit	MIPS DSP
PICK the QB	Conditional bit - based (rightmost four) byte selection	MIPS DSP

PICK the PH	Conditional bit - based (rightmost 2) halfword selection	MIPS DSP
APPEND	Words are shifted left and spliced low	MIPS DSP R2
The PREPEND	Right shift and high splice	MIPS DSP R2
BALIGN	Two registers high and low byte splice	MIPS DSP R2
PACKRL. PH	Package the right-most character of source 1 with the second-right character of source 2	MIPS DSP R2

Accumulator operation class instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
EXTR w.	After the accumulator is moved to the right, the truncated word is assigned to the general purpose register. The shift value is specified by the immediate number	MIPS DSP

Instruction mnemonic	Instruction function description	ISA Compatible Category
EXTR_R w.	The accumulator is rounded to the right, the truncated word is assigned to the general purpose register, and the shift value is specified by the immediate number	MIPS DSP
EXTR_RS w.	The accumulator moves right after saturated rounding, intercepting the word to the general purpose register, shifting the value from the immediate number The specified	MIPS DSP
EXTR_S. H	Move right from accumulator saturation, extract half word to general purpose register, shift value specified by immediate number	MIPS DSP
EXTRV_S. H	Move right from accumulator saturation, extract half word to general purpose register, shift value specified by register	MIPS DSP
EXTRV w.	After the accumulator moves to the right, the truncated word is assigned to the general purpose register, and the shifted value is specified by the register	MIPS DSP
EXTRV_R w.	The accumulator is rounded to the right, the truncated word is assigned to the general purpose register, and the shifted value is specified by the register	MIPS DSP
EXTRV_RS w.	The accumulator moves right after saturated rounding, intercepts the word to the general purpose register, and shifts the value from the register The specified	MIPS DSP
EXTP	Extracts a fixed length number from any position of the accumulator to the general purpose register, with the length changed from immediately The number specified	MIPS DSP
EXTPV	The number of fixed lengths is extracted from any position of the accumulator to the general purpose register Implement instruction	MIPS DSP
EXTPDP	Extract the fixed length number from any position of the accumulator to the general purpose register and subtract the POS value, The length is specified by the immediate number	MIPS DSP
EXTPDPV	Extract the fixed length number from any position of the accumulator to the 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 number	MIPS DSP
SHILOV	The accumulator value is shifted and then written back to the same accumulator. The shifted value is determined by the register	MIPS DSP
MTHLIP	Copy LO value to HI, copy general register value to LO, increase POS value by 32	MIPS DSP
MFHI	The HI register value is moved to the General register	MIPS32
MFLO	The LO register value is moved to the General Purpose register	MIPS32
MTHI	The general register value is moved to the HI register	MIPS32
MTLO	The general register value is moved to register LO	MIPS32

Access DSP control register class instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
WRDSP	Read general register values written to DSP control register	MIPS DSP
RDDSP	Read DSP control register value to general register	MIPS DSP

Access class instruction with index register

Instruction mnemonic	Instruction function description	ISA Compatible Category
LBUX	The index address takes an unsigned byte	MIPS DSP
LHX	Index address takes half a word	MIPS DSP
LWX	Index address fetch word	MIPS DSP

Branch instruction

Instruction mnemonic	Instruction function description	ISA Compatible Category
BPOSGE32	DSP control register POS value is greater than or equal to 32 jump	MIPS DSP

DSP instructions no longer supported by MIPS

The instructions listed in this section, although defined in r2.34 of The MIPS® Architecture for Programmers VolumeIV-e: The MIPS® DSP Application-Specific Extension to The MIPS64® Architecture, have been deleted after r2.40. Although the Loongson 3A300/3B3000 processor implements these instructions, it is not recommended for use.

Instruction mnemonic	Instruction function description
ABSQ_S. OB	The vector takes (8 right-most) byte absolute values and saturates them, resulting in symbol expansion
ABSQ_S. PW	Vector take (right 4) absolute value of half word, and do saturation operation, the result sign expansion
ABSQ_S. QH	Vector take (rightmost 2) absolute value of the word, and do saturation operation, the result symbol expansion
ADDQ. PW	The vector 2 to the right plus a small number
ADDQ supachai panitchpakdi W	Vector (right 2) small number saturation plus
ADDQ. QH	Vector (right 4) decimal half word plus
ADDQ S.Q H	Vector (4 to the right) decimal halfword saturation plus
ADDU. OB	Vector (right 8) decimal byte unsigned plus
ADDU S.O B	Vector (right 8) decimal byte unsigned saturation plus
ADDU. QH	Vector (right 4) decimal halfword unsigned plus
ADDU S.Q H	Vector (right 4) decimal halfword unsigned saturation plus
ADDUH. OB	The vector is unsigned (8 right-most) bytes plus, the result is divided by 2, and the result is extended
ADDUH_R. OB	The vector is unsigned (8 right-most) bytes rounded plus, the result is divided by 2, and the result is extended
BPOSGE64	The DSP control register POS value is 64 or higher, then it jumps
CMP. EQ. PW	Vector (rightmost 2) word equal comparison, result set condition bit
CMP. LT. PW	Vector (rightmost 2) word is less than comparison, result sets condition bit
CMP. LE. PW	Vector (rightmost 2) words less than or equal to the comparison, the result set condition bit
CMP. EQ. QH	Vector (right 4) half - word equal comparison, the result set condition bit
CMP. LT. QH	Vector (right-most 4) half character is less than the comparison, the result set condition bit
CMP. LE. QH	Vector (right 4) half - word less than or equal to the comparison, the result set condition bit
CMPGDU. EQ. OB	Vector (rightmost 8) unsigned byte equal comparison, the result of both conditional and general register
CMPGDU. LT. OB	Vector (8 right-most) unsigned bytes less than the comparison, the result is both conditional and general register
CMPGDU. LE. OB	Vector (8 right-most) unsigned bytes less than or equal to the comparison, the result is both conditional and general register
CMPGU. EQ. OB	Vector (rightmost 8) byte equal comparison, the result set general purpose register
CMPGU. LT. OB	Vector (rightmost 8) bytes less than comparison, result in general purpose register
CMPGU. LE. OB	Vector (8 right-most) bytes less than or equal to the comparison, the result of the general purpose register
CMPU. EQ. OB	Vector (rightmost 8) unsigned byte equal comparison, the result set condition bit
CMPU. LT. OB	Vector (8 right-most) unsigned bytes less than comparison, result set condition bit
CMPU. LE. OB	Vector (8 right-most) unsigned bytes less than or equal to the comparison, the result set condition bit

DAPPEND	Right shift and high splice
DBALIGN	Two registers high and low byte splice
DEXTTP	Extracts a fixed length from any position of the accumulator to a general purpose register. The length is specified by the immediate number
DEXTDPDP	Extract the fixed length number from any position of the accumulator to the general purpose register and subtract the POS value. The length is denoted by the immediate number set

Instruction mnemonic	Instruction function description
DEXTDPDV	Extract the fixed length number from any position of the accumulator to the general purpose register and subtract the POS value. The length is indicated by the register set
DEXTPV	Extracts a fixed length number from any position of the accumulator to a general purpose register. The length is determined by the register instruction
DEXTR. L	After the accumulator moves to the right, the double word is intercepted and assigned to the general purpose register. The shift value is specified by the immediate number
DEXTR_R. L	The accumulator is rounded to the right, intercepting the double word and assigning it to the general purpose register. The shift value is specified by the immediate number
DEXTR_RS. L	The accumulator moves to the right after saturated rounding, intercepts the double word and assigns it to the general purpose register. The shift value is specified by the immediate number
DEXTR w.	After the accumulator is moved to the right, the truncated word is assigned to the general purpose register. The shift value is specified by the immediate number
DEXTR_R w.	The accumulator is rounded to the right, the truncated word is assigned to the general purpose register, and the shift value is specified by the immediate number
DEXTR_RS w.	The accumulator moves to the right after saturated rounding, intercepting words to the general purpose register, and shifting values are specified by the immediate number
DEXTR_S. H	Move right from accumulator saturation, extract half word to general purpose register, shift value specified by immediate number
DEXTRV. L	After the accumulator moves to the right, the double word is intercepted and assigned to the general purpose register. The shift value is specified by the register
DEXTRV_R. L	The accumulator is rounded to the right, intercepting the double word and assigning it to the general purpose register. The shift value is specified by the register
DEXTRV_RS. L	The accumulator moves right after saturated rounding, intercepts the double word and assigns it to the general purpose register. The shift value is specified by the register
DEXTRV_S. H	After the accumulator moves to the right, the truncated word is assigned to the general purpose register, and the shifted value is specified by the register
DEXTRV w.	The accumulator is rounded to the right, the truncated word is assigned to the general purpose register, and the shifted value is specified by the register
DEXTRV_R w.	The accumulator moves right after saturated rounding, intercepting words to the general purpose register, and shifting values are specified by the register
DEXTRV_RS w.	Move right from accumulator saturation, extract half word to general purpose register, shift value specified by register
DINSV	Variable bit field insertion
DMADD	Double characters are multiplied by symbols
DMADDU	Double word unsigned multiplication and addition
DMSUB	Double characters are signed multiplied and subtracted
DMSUBU	Double word unsigned times minus
DMTHLIP	LO values are copied to HI, general register values are copied to LO, pos values are increased by 64
The DPA. W.Q H	Vector integers (4 to the right) half-word multiply and sum, and finally sum with ACC
DPAQ_S W.Q H	The vector decimals (4 to the right) are multiplied by the half-word, and the result is multiplied by acc
DPAQ_SA L.P W	Vector small number multiplication, the result after saturated rounding, and then summing with ACC
DPAU. H.O BL	The vector integer (4 leftmost) bytes are unsigned multiplied and accumulated, and the result is then accumulated with ACC
DPAU. H.O BR	Vector integer (4 to the right) bytes unsigned multiplied and accumulated, the result is then accumulated with ACC
The DPS. W.Q H	Vector integers (4 to the right) are multiplied and accumulated, and the result is then subtracted from ACC
DPSQ_S W.Q H	The vector decimals (4 to the right) are multiplied by half words and subtracted from ACC

DPSQ_SA L.P W	Multiply the vector by a small number, the result is accumulated after saturated rounding, and the result is accumulated with ACC
DPSU. H.O BL	The vector integer (4 leftmost) bytes are unsigned multiplied and accumulated, and the result is then accumulated with ACC
DPSU. H.O BR	Vector integer (4 to the right) bytes unsigned multiplied and accumulated, the result is then accumulated with ACC
DSHILO	The accumulator value is shifted and then written back to the same accumulator. The shift value is determined by the immediate number
DSHILOV	The accumulator value is shifted and then written back to the same accumulator. The shifted value is determined by the register
LDX	Base address plus index fetch
MAQ_S. L.P WL	The vector decimal (to the right) is multiplied by acc
MAQ_S. L.P WR	The vector decimal (far right) is multiplied by acc
MAQ_S. W.Q HLL	The vector decimal (leftmost) is multiplied by half a word and the result is added to ACC
MAQ_SA. W.Q HLL	The vector decimal (leftmost) is multiplied by a half-word, and the result is summed with ACC after saturated rounding
MAQ_S W.Q HLR	The vector decimal (second left) is multiplied by half a word, and the result is added to ACC

Instruction mnemonic	Instruction function description
MAQ_SA W.Q HLR	The vector decimal (left) is multiplied by half word, and the result is summed with ACC after saturated rounding
MAQ_S. W.Q HRL	The vector decimal (to the right) is multiplied by half a word, and the result is added to ACC
MAQ_SA. W.Q HRL	Vector decimal (second right) half-word multiplication, the result is to take the saturated rounding and then add with ACC
MAQ_S. W.Q HRR	The vector decimal (far right) is multiplied by half a word and the result is added to ACC
MAQ_SA. W.Q HRR	Vector decimal (right-most) half-word multiplication, the result is to take the saturated rounding and then add with ACC
MULEQ_S. PW. QHL	The left half of a sign is saturated by multiplication, and the result is a word
MULEQ_S. PW. QHR	The right half of a sign is saturated by multiplication, and the result is a word
MULEU_S. QH. With	Vector (4 to the right) bytes unsigned times (4 to the right) halfwords, resulting in two halfwords
MULEU_S. QH. The OBR	Vector (4 right-most) bytes unsigned times (4 right-most) halfwords, resulting in two halfwords
MULQ_RS. QH	The vector (4 right-most) halfwords are saturated and rounded, resulting in half-words
MULSAQ_S L.P W	Multiply the small number of vector, subtract after saturation, and then add the result with ACC
MULSAQ_S W.Q H	The vector decimals (4 to the right) are multiplied and subtracted, and the result is added to ACC
PACKRL. PW	Package the right-most word of source 1 with the right-second word of source 2
PICK the OB	Conditional bit - based (8 right-most) bytes selection
PICK the PW	Conditional bit - based (4 right - most) halfword selection
PICK the QH	Conditional bit - based (rightmost 2) word selection
PRECEQ. L.P WL	Vector decimal precision has signed extension, from (second right) word to double word
PRECEQ. L.P WR	Vector decimal precision signed extension, from (far right) word to double word
PRECEQ. PW. QHL	Vector decimal precision has symbolic extension, from (two to the right) half word to two words
PRECEQ. PW. QHR	Vector decimal precision has symbolic expansion from (rightmost two) half words to two words
PRECEQ. PW. QHLA	Vector decimal precision has a signed extension, from half - word to two - word
PRECEQ. PW. QHRA	Vector decimal precision has a signed extension, from half - word to two - word
PRECEQU. QH. With	Vector decimal precision extension, from (4 left) unsigned bytes to 4 half-words
PRECEQU. QH. The OBR	Vector decimal precision extension, from (4 on the right) unsigned bytes to 4 half-words
PRECEQU. QH. OBLA	Vector decimal precision extension, from unsigned bytes (4 to the left of the right-most word) to 4 half-words
PRECEQU. QH. OBRA	Vector decimal precision extension, from unsigned bytes (4 to the right of the right-most word) to 4 half-words
PRECEU. QH. With	Vector precision extension, from (4 left) unsigned bytes to 4 halfwords
PRECEU. QH. The OBR	Vector precision expansion from (4 on the right) unsigned bytes to 4 halfwords
PRECEU. QH. OBLA	Vector precision expansion from unsigned bytes (4 left crosses of right-most word) to 4 half-words
PRECEU. QH. OBRA	Vector precision expansion from unsigned bytes (4 crosses the right side of the right-most word) to 4 half-words
PRECR. OB. QH	Vector integer precision reduced from (4 right-most) halfwords to 8 bytes
PRECR_SRA. QH. PW	Vector right shift integer precision reduction, from (rightmost 2) words to 4 half words, resulting symbol expansion
PRECR_SRA_R. QH. PW	Vector right shift integer precision reduction from (rightmost 2) words to 4 half words, and do rounding, resulting symbol expansion

PRECRQ. OB. QH	Vector decimal precision reduced from (right-most 4) halfwords to 8 bytes
PRECRQ. PW. L	Vector decimal precision reduced from (2 words to the right) to 4 half words
PRECRQ. QH. PW	Vector decimal precision reduced from (2 words to the right) to (4 words)
PRECRQ_RS. QH. PW	Vector decimal precision is reduced from (right 2) words to (4) half words, and done with saturation and rounding
PRECRQU_S. OB. QH	Vector decimal precision reduced from (right-most 4) halfwords to 8 unsigned bytes
PREPENDD	Double word right shift and high splice
PREPENDW	The word is shifted to the right and spliced high
RADDU L.O B	8 bytes unsigned accumulation

Instruction mnemonic	Instruction function description
REPL. OB	Vector copies immediate number (integer) to 8 bytes, resulting symbol extension
REPL. PW	Vector copies immediate number (integer) to 2 words, resulting symbol extension
REPL. QH	Vector copies immediate Numbers (integers) to 4 halfwords, resulting in symbol expansion
REPLV. OB	Vector copies bytes to 8 bytes, resulting symbol extension
REPLV. PW	Vector copies word to 2 words, resulting symbol extension
REPLV. QH	Vector copy half word to 4 half word, resulting symbol expansion
SHLL. OB	The vector logic shifts 8 bytes to the right, the shift value is specified by the immediate number, and the resulting symbol expands
SHLL. QH	Vector logic moves left (four right-most) halfwords, the shift value is specified by the immediate number, and the resulting symbol expands
SHLL. PW	Vector logic moves left (rightmost 2) halfwords, the shift value is specified by the immediate number, the result symbol expands
SHLLV. OB	The vector logic shifts 8 bytes to the left, the shift value is specified by the register, and the result symbol expands
SHLLV. QH	Vector logic moves left (four right-most) halfwords, the shift value is specified by the register, and the result symbol expands
SHLLV. PW	Vector logic moves left (rightmost 2) halfwords, the shift value is specified by the register, and the result symbol expands
SHLL_S. QH	Vector saturation logic moves left (4 right-most) halfwords, the shift value is specified by the immediate number, the result symbol expands
SHLL_S. PW	Vector saturation logic moves left (rightmost 2) halfwords, the shift value is specified by the immediate number, the result symbol expands
SHLLV_S. QH	Vector saturation logic moves left (four right most) halfwords, the shift value is specified by the register, the result symbol expands
SHLLV_S. PW	Vector saturation logic moves left (rightmost 2) halfwords, the shift value is specified by the register, the result symbol expands
SHRA. OB	Vector arithmetic is shifted to the right (8 right-most) bytes, the shifted value is specified by the immediate number, and the resulting symbol is extended
SHRA. QH	Vector arithmetic is right-shifted (4 right-most) halfwords, the shifted value is specified by the immediate number, and the resulting symbol is extended
SHRA. PW	Vector arithmetic is right-shifted (right-most 2) halfwords, the shifted value is specified by the immediate number, and the resulting symbol is extended
SHRAV. OB	Vector arithmetic is shifted to the right (8 rarest) bytes, the shifted value is specified by the register, and the resulting symbol is extended
SHRAV. QH	Vector arithmetic is right-shifted (four right-most) halfwords, the shifted value is specified by the register, and the resulting symbol is extended
SHRAV. PW	Vector arithmetic is right-shifted (rightmost 2) halfwords, the shifted value is specified by the register, the result symbol is extended
SHRA_R. OB	Vector arithmetic is right-shifted (8 right-most) bytes, with the shift value specified by the immediate number, rounded, and the resulting symbol extended
SHRA_R. QH	Vector arithmetic is right-shifted (4 right-most) halfwords, with the shift value specified by the immediate number, rounded, and the resulting symbol extended
SHRA_R. PW	Vector arithmetic right-shift (rightmost two) words, the value of the shift is specified by the immediate number, and done rounding, the result symbol expansion
SHRAV_R. OB	Vector arithmetic is shifted to the right (8 right-most) bytes, the shifted value is specified by the register, and is rounded, resulting in symbol expansion
SHRAV_R. QH	Vector arithmetic right-shift (4 right-most) halfwords, shift values specified by registers, and done rounding, resulting symbol expansion
SHRAV_R. PW	Vector arithmetic right-shift (rightmost two) words, shift values specified by registers, and done rounding, resulting symbol expansion
SHRL. OB	Vector logic shifts 8 bytes to the right, the shift value is specified by the immediate number, and the resulting symbol expands
SHRL. QH	Vector logic moves right (4 right-most) halfwords, the shift value is specified by the immediate number, the result symbol expands

SHRLV. OB	Vector logic shifts 8 bytes to the right, the shifted value is specified by the register, and the resulting symbol expands
SHRLV. QH	Vector logic moves right (4 right-most) halfwords, the shift value is specified by the register, and the result symbol expands
SUBQ. PW	Vector (2 to the right) small number minus
SUBQ. QH	Vector (4 to the right) decimal half word minus
SUBQ_S. PW	Vector (rightmost 2) small number saturation minus
SUBQ_S. QH	Vector (4 to the right) decimal half word saturation minus
SUBU. OB	Vector (8 right most) decimal bytes unsigned minus
SUBU. QH	Vector (4 to the right) decimal halfword unsigned minus
SUBU_S. OB	Vector (right 8) decimal byte unsigned saturation minus
SUBU_S. QH	Vector (4 to the right) decimal halfword unsigned saturation minus
SUBUH. OB	The vector is unsigned (8 to the right), the result is subtracted, the result is divided by 2, the result is extended

Instruction mnemonic	Instruction function description
SUBUH_R. OB	Vector unsigned (8 right-most) bytes rounded down, result divided by 2, result symbol extended

2.3.2 Supplementary instructions to MIPS DSP instruction manual

The definition of MTHI and MTLO in MIPS DSP instruction manual is unreasonable. In a 64-bit architecture, the execution of the MTHI and MTLO instructions does not require writing to the HI/LO register after a high 32-bit symbol extension of the value of the general purpose register. YingCaiYong MIPS TongYongZhiLingShouCeZhongDuiYu MTHI、MTLO 指令的定义形式，即 $HI \square GPR[rs]$ ， $LO \square GPR[rs]$ 。

2.4 MIPS64 compatible instruction implementation definition

This section describes the implementation-related parts of the MIPS64 compatibility instructions implemented by the GS464E, or those that differ from the MIPS64 specification.

2.4.1 The load instruction that targets the no. 0 general purpose register

All load instructions, except LDPTE and LWPTE, whose target is the universal register, perform as well as PEF(hint=0) when their target bit is the zero universal register. However, for the GSLQ instruction of the godson custom extension, it is necessary to set both of its target registers as general purpose register 0.

2.4.2 PEF instruction

The PEF directive only implements the hint=0 (prefetch for Load) and hint=1 (Prefetch for store) modes according to the MIPS specification.

2.4.3 RDHWR instruction

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

- Rd =30: Reads the external counter value. The counter is 64-bit and increases by 1 per beat at a frequency consistent with the internal bus clock frequency of the chip. The characteristic of the clock frequency is that it does not vary with the clock frequency of a processor core and can be considered as a constant frequency.
- Rd =31: Read the ratio between the current frequency of processor core and the frequency of bus in the chip (M/N). Where M is in the [15:8] bit of the return value and N is in the [7:0] bit of the return value.

2.4.4 PEFX instruction

The PEFX directive implements five hints of 0, 1, 26, 27, and 28. Specific implementation details are as follows:

- Hint =0, 1: Here the PEFX instruction prefetches in accordance with the MIPS specification, that is, it prefetches one cacheline at a time. The difference is that the GS464E only extends the low 16-bit value

symbol in the index register as an index.

- Hint =26: Enter the first-level data cache according to the configuration of continuous prefetch. A prefetch instruction can automatically prefetch the block of block_NUM with the distance of stride as the length of block_size from the base address. At this point, part of the information in the index register is used to configure the generation mode of the prefetch address. Specifically, the [15:0] bit of GPR[index] is the 16bit signed address offset added to GPR[base]; The [16] bit of GPR[index] is the address ascending and descending prefetch mark, 0 represents address ascending prefetch, and 1 represents address descending prefetch. The [25:20] bit of GPR[index] is the pre-taken block_sid-1. The basic unit of block_size is 128bit, so the longest block can contain $64 \times 16 = 1\text{KB}$ of data. The [39:32] bit of GPR[index] is block_num-1, so it supports pre-fetching of up to 256 blocks. Index [59:44] is the stride between adjacent blocks, is the signed number, and the unit is byte.

- Hint =27: The hint is configured to continuously pre-fetch exclusive data into a level 1 data cache. In this mode, the contents of the index register parse and Hint =26 is exactly the same, except that this mode requires that the retrieved data be in an exclusive state.
- Hint =28: Configure the sequential pre-fetch data to enter the Shared cache. Parsing the contents of the index register in this mode is exactly the same as hint=26, except that in this mode the prefetch data is stored in a Shared cache.

If a WAIT_CACHE instruction is executed, the configuration prefetch is stopped. These include but are not limited to cache, SYNC and SYNCI, and JRHB.

If you execute two PREFX (hint=25,26) in a row, the previous one will be covered with the new one. If you execute two PREFX (hint=27) in a row, the previous one will be covered with the new one. However, there is no conflict between the pre-fetch modes of hint=25,26, and hint=27.

2.4.5 WAIT instruction

There is only one mode supported by the WAIT instruction, which executes the same effect no matter what value the software inserts into the Implementation Dependent Code field in the instruction code.

After the WAIT instruction is submitted in the processor pipeline, the processor core stops pointing and goes into low-power mode, which continues until it is interrupted by NMI, internal clock interrupt, internal performance counter interrupt, or external hardware interrupt.

It is important to note that after the WAIT instruction is executed, the data paths of the processor core's first-level instruction cache, first-level data cache, and second-level sacrifice cache can still handle cache conformance requests from outside. If the processor core's clock needs to be turned off completely, the software needs to do more to ensure that it doesn't cause other cores to crash due to unresponsive consistency requests.

2.4.6 SYNC instructions

This processor implements only the SYNC instruction of Stype =0, and stype will leave exceptions for other values. This instruction ACTS as a memory barrier to ensure that the access operation before SYNC has been completed (for example, the data of store instruction has been written to dcache, the read and write of uncached has been completed, and load has retrieved the value to register), and that the access operation after SYNC instruction has not started yet.

2.4.7 SYNCI instruction

Since GS464E is maintained by the hardware for data consistency between the first-level instruction cache and the first-level data cache, the implementation of the SYNCI instruction is adjusted. In the existing implementation, the SYNCI directive, on the one hand, waits until all previous accesses have been executed (the same effect as the SYNC directive). On the other hand, SYNCI will force subsequent instructions to repoint after execution to ensure that the pointing unit can also see the execution effect of all accesses before the SYNCI instruction. At the same time, the address information carried by the SYNCI directive is ignored, so tiB-related exceptions, address error exceptions, and Cache error exceptions are no longer triggered.

2.4.8 TLBINV and TLBINVF directives

Although the config4.ie field in this processor is set to 0, the TLBINVF instruction (which is executed as config4.ie =3 as defined in the MIPS specification) is implemented in the processor, that is, when a TLBINVF instruction is executed, the hardware will invalidate the entire TLB table entry. In addition, the TLBINV directive is also implemented in GS464E, but its execution effect is different from the MIPS specification definition and is equivalent to the TLBINVF directive.

2.4.9 CACHE directives

There are three major differences between the CACHE instructions implemented by this processor and the MIPS64 specification:

1. Relationship between instruction OP and cache hierarchy

The correspondence between the [1:0] bits of the CACHE instruction OP field in GS464E and the CACHE hierarchy is different from the definition of the MIPS64 specification, as shown in the table

2-26:

Table 2-26 CACHE instruction OP [1:0] corresponds to the CACHE hierarchy

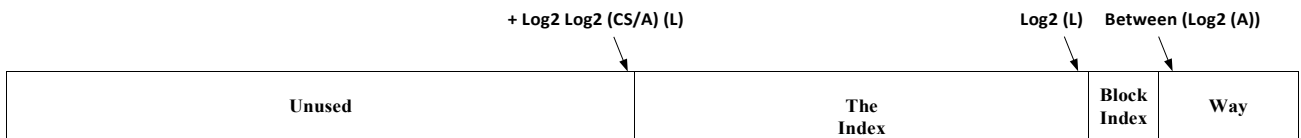
The op (1-0)	GS464E	MIPS64 specification	Similarities and differences between
0 b00	First-order instruction cache	First-order instruction cache	consistent
0 b01	Level 1 data cache	Level 1 data cache	consistent
0 b10	Secondary sacrifice cache	Three levels of cache	Don't agree
0 b11	Three-level Shared cache	The second level cache	Don't agree

2. Address resolution method of Index class CACHE instruction

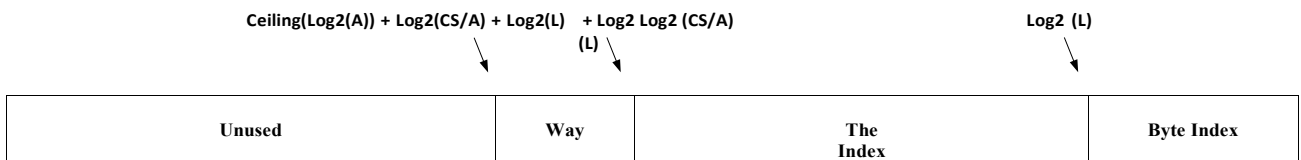
The address resolution of the index-class Cache instruction implemented by this processor is different from the definition of the MIPS64 specification, as shown in Figure 2-8. It is important to note that because of GS464E lowest several addresses are used to indicate to the operation of the Cache, sacrifice and secondary Cache associated with level 3 group Shared Cache are 16 road, road, choose the lowest four need account address, so for the secondary sacrifice Cache and tertiary Shared Cache on the Index Data Load and Index Data Store operation, can only be read from the Cache line even piece of Data, the Cache line only odd-even two adjacent blocks to fill in the same Data at the same time.

Figure 2-8. Address resolution format of the Index class CACHE instruction

Loongson 3A1500 chip processor



MIPS64 specification



Description:

Assume that the capacity of the Cache being operated is CS, the degree of association is a-group association, and the size of the Cache row is L bytes. Then the address number used to select the path is: $\text{Ceiling}(\text{Log}_2(A))$

$\text{Log}_2(\text{CS}/A)$ $\text{Log}_2(L)$ $\text{Log}_2(L)$ $\text{Log}_2(L)$

3. Meaning of CACHE28, CACHE29, CACHE30 and CACHE31 Instructions

In GS464E, the CACHE28, CACHE29, CACHE30, and CACHE31 directives (i.e., the Cache directive of OP [4:2]=0b111) have different meanings from the MIPS64 specification. These instructions in the godson processor are all Index Store Data operations, while in MIPS64 specification, they are all Fetch and Lock operations.

4. Meaning of CACHE15 Instruction

In GS464E, CACHE15 instruction is a CACHE instruction related to implementation. Its function is to Fetch and Lock Scache

Operations, specific functional definitions similar to the CACHE31 directive in the MIPS64 specification.

2.4.10 Madd.fmt, Msub.fmt, Nmadd.fmt, Nmsub.fMT instruction

In THE MIPS64 specification, if FIR.Has2008=0 or FCSR.MAC2008=0, then madd.fmt, msub.fmt, nmadd.fmt

And Nmsub.fMT instruction to round the intermediate result after the multiplication operation, then conduct subsequent addition and subtraction operation, and round the final result after addition and subtraction again. Although GS464E does not support the ANSI/IEEE754-2008 binary floating point standard, the floating-point multiplication plus class operation is performed only once at the final result, which is the Fuse-Multiply-Add operation defined by the ANSI/IEEE754-2008 binary floating point standard.

2.4.11 EHB, SSNOP instructions

All execution correlation between GS464E instructions is handled by the hardware, so EHB and SSNOP instructions are treated as NOP instructions in the processor.

2.4.12 DI and EI instructions

DI instruction can only be used in the 3A3000C/3B3000C model, which is compatible with the MIPS64 definition.

EI instruction can only be used in 3A3000C/3B3000C type, but the specific definition is slightly different from MIPS64 definition: after EI execution, the IE position of the Status register can be set to 0, and the global terminal enable can be turned off. However, only the lowest 3 bits in the return value are meaningful, which correspond to the ERL, EXL and IE bits of the Status register before EI execution.

2.5 Loong core expansion command set

GS464E extension instructions include the following categories according to their functions:

- Access class instruction
- Arithmetic and logical operation instructions
- X86 binary acceleration instruction
- ARM binary acceleration instruction
- 64-bit multimedia instructions
- Misc

ellaneous

access class

instruction

Table 2-27 Instruction of longson extended access class

Instruction mnemonic	Instruction function description	ISA Compatible Category
SETMEM	Access 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 instructions	LoongEXT64
GSLE	Exception if less than or equal to set address error	LoongEXT32
GSGT	Exception if greater than set address error	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

Instruction mnemonic	Instruction function description	ISA Compatible Category
GSLDRC1	Take the right part of the double word to the floating point register	LoongEXT32
GSLBLE	Fetch byte with an out-of-bounds check	LoongEXT32
GSLBGT	The fetch byte with the next out-of-bounds check	LoongEXT32
GSLHLE	Take a half-character with an out-of-bounds check	LoongEXT32
GSLHGT	Take a half word with cross - boundary check below	LoongEXT32
GSLWLE	Take the word for cross - border inspection	LoongEXT32
GSLWGT	Take word with cross - boundary check	LoongEXT32
GSLDLE	Take double characters with cross - boundary check	LoongEXT64
GSLDGT	Take double characters with cross - boundary check below	LoongEXT64
GSLWLEC1	Fetch-word to the floating-point register with an out-of-bounds check	LoongEXT32
GSLWGTC1	Fetch-word to the floating-point register with the down - bound check	LoongEXT32
GSLDLEC1	Take a double word to the floating point register with an out - of - bounds check	LoongEXT64
GSLDGTC1	Take a double word to the floating point register with the next cross check	LoongEXT64
GSLQ	Double target register take - point four word	LoongEXT64
GSLQC1	The dual target register takes a floating point quadword	LoongEXT64
GSLBX	Fetch byte with offset	LoongEXT32
GSLHX	Take half word with offset	LoongEXT32
GSLWX	Take word with offset	LoongEXT32
GSLDX	Take a double word with offset	LoongEXT64
GSLWXC1	Floating-point words with offset	LoongEXT32
GSLDXC1	Floating point double word with offset	LoongEXT32
GSSWLC1	Saves the left part of a word from a floating-point register	LoongEXT32
GSSWRC1	Saves the right part of the word from the floating point register	LoongEXT32
GSSDLC1	Saves the left part of a double word from a floating point register	LoongEXT32
GSSDRC1	Saves the right part of the double word from the floating point register	LoongEXT32
GSSBLE	Carries bytes that are checked out of bounds	LoongEXT32
GSSBGT	Save bytes with down - bound check	LoongEXT32
GSSHLE	Take the cross - border check of the storage half - word	LoongEXT32
GSSHGT	Take the next cross - check of the storage half - word	LoongEXT32
GSSWLE	Carry the word that crosses the line to check	LoongEXT32
GSSWGT	Bring down the word of cross - border check	LoongEXT32
GSSDLE	Take the cross - border check of the save double - character	LoongEXT64
GSSDGT	Bring down the cross - check of the save double - character	LoongEXT64
GSSWLEC1	Saves a word from a floating-point register with an out-of-bounds check	LoongEXT32
GSSWGTC1	Saves a word from a floating-point register with an overbounds check	LoongEXT32
GSSDLEC1	Save double words from the floating point register with an out - of - bounds check	LoongEXT32
GSSDGTC1	Save a double word from a floating-point register with an overbounds check	LoongEXT32
GSSQ	Dual source register saves fixed point four characters	LoongEXT64
GSSQC1	Dual source register saves fixed point four characters	LoongEXT64


GSSBX	Offset bytes	LoongEXT32
GSSHX	Offset memory halfwords	LoongEXT32

Instruction mnemonic	Instruction function description	ISA Compatible Category
GSSWX	Offset memory words	LoongEXT32
GSSDX	Offset memory double word	LoongEXT64
GSSWXC1	Floating point words with offset	LoongEXT32
GSSDXC1	Floating point double word with offset	LoongEXT32

Arithmetic and logical operation instructions

Table 2-28 Loongson extended arithmetic and logic operation instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
GSANDN	General purpose register logical bit non - and	LoongEXT32
GSORN	General purpose register logical bit non or	LoongEXT32
GSADC. D.	Double word with carry	LoongEXT64
GSADC w.	Carry word plus	LoongEXT32
GSADC. H	Carry half word plus	LoongEXT32
GSADC. B	Carry byte plus	LoongEXT32
GSADCU. D.	Unsigned double word plus with carry	LoongEXT64
GSADCU w.	Carry unsigned word addition	LoongEXT32
GSADCU. H	Unsigned halfword with carry	LoongEXT32
GSADCU. B	Unsigned byte plus with carry	LoongEXT32
GSSBC. D.	With borrow double word minus	LoongEXT64
GSSBC w.	Take the debit word minus	LoongEXT32
GSSBC. H	Take the debit half word minus	LoongEXT32
GSSBC. B	Subtract with borrow	LoongEXT32
GSSBCU. D.	Unsigned double word subtraction with debit	LoongEXT64
GSSBCU w.	Minus unsigned word with debit	LoongEXT32
GSSBCU. H	Unsigned half-word subtraction with debit	LoongEXT32
GSSBCU. B	Unsigned byte with debit	LoongEXT32
GSMULT	A sign word is multiplied, and the result is written in the general purpose register	LoongEXT32
GSDMULT	Sign double word multiplication, the result is written to the general purpose register	LoongEXT64
GSMULTU	Unsigned word multiplication, the result is written to the general purpose register	LoongEXT32
GSDMULTU	Unsigned double word multiplication, the result is written to the general purpose register	LoongEXT64
GSDIV	Sign word division, quotient write general register	LoongEXT32
GSDDIV	Sign double - word division, quotient - write general - purpose register	LoongEXT64
GSDIVU	Unsigned word division, quotient write general register	LoongEXT32
GSDDIVU	Unsigned double - word division, quotient write general - purpose register	LoongEXT64
GSMOD	Sign word division, remainder write general purpose register	LoongEXT32
GSDMOD	Sign double - word division, remainder write general - purpose register	LoongEXT64
GSMODU	Unsigned word division, remainder write general purpose register	LoongEXT32

 GSDMODE	Unsigned double - word division, remainder write general - purpose register	LoongEXT64
GSROTR. H	Move the half-word loop to the right	LoongEXT32
GSROTR. B	The byte cycle moves right	LoongEXT32
GSROTRV. H	Variable shift halfword circulates to the right	LoongEXT32

Instruction mnemonic	Instruction function description	ISA Compatible Category
GSROTRV. B	The variable shift byte cycles right	LoongEXT32
GSRRCR. D.	Double word loop with CF bit moves right	LoongEXT64
GSRRCR w.	The word loop with CF bit moves right	LoongEXT32
GSRRCR. H	A half-word loop with CF bits moves to the right	LoongEXT32
GSRRCR. B	The byte loop with CF bits is moved right	LoongEXT32
GSDRCR32	Shift + 32 double word cycle with CF bit shifts to the right	LoongEXT64
GSRRCRV. D.	A double word loop with a variable shift with CF bits shifted right	LoongEXT64
GSRRCRV w.	A word with a variable shift with CF bits circulates to the right	LoongEXT32
GSRRCRV. H	A half-word cycle with a variable shift with CF bits moves right	LoongEXT32
GSRRCRV. B	A byte cycle with a variable shift with CF bits shifts right	LoongEXT32

Binary translation Acceleration instruction (X86)

Table 2-29 Longson extension X86 binary translation acceleration instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
SETX86FLAG. D.	The EFLAG two-word mode prefix instruction is set in x86 mode	LoongEXT64
SETX86FLAG w.	The EFLAG prefix instruction is placed in x86 mode	LoongEXT32
SETX86FLAG. H	EFLAG halfword mode prefix instruction is set in x86 mode	LoongEXT32
SETX86FLAG. B	The EFLAG byte mode prefix instruction is set in x86 mode	LoongEXT32
X86AND. D.	Only the two-word logical bits and of EFLAG are set in x86 mode	LoongEXT64
X86AND w.	Only the logical bits and words of EFLAG are set in x86 mode	LoongEXT32
X86AND. H	Only the half-word logical bits and of EFLAG are set in x86 mode	LoongEXT32
X86AND. B	Only the byte logical bits and bytes of EFLAG are set in x86 mode	LoongEXT32
X86OR. D.	Only the two-word logical bits or of EFLAG are set in x86 mode	LoongEXT64
X86OR w.	Only the word logical bits or of EFLAG are set in x86 mode	LoongEXT32
X86OR. H	Only the half-word logical bits or of EFLAG are set in x86 mode	LoongEXT32
X86OR. B	Only the byte logical bits or of EFLAG are set in x86 mode	LoongEXT32
X86XOR. D.	Only the two-word logical bit xor of EFLAG is set in x86 mode	LoongEXT64
X86XOR w.	Only the logical bit or of EFLAG is set in x86 mode	LoongEXT32
X86XOR. H	Only half-word logical bits or of EFLAG are set in x86 mode	LoongEXT32
X86XOR. B	Only the byte logical bit xor of EFLAG is set in x86 mode	LoongEXT32
X86ADD. D.	Only EFLAG double characters are set in x86 mode	LoongEXT64
X86ADD w.	Only EFLAG characters are set in x86 mode	LoongEXT32
X86ADD. H	Only EFLAG halfwords are set in x86 mode	LoongEXT32
X86ADD. B	Only the bytes of EFLAG are set in x86 mode	LoongEXT32
X86ADC. D.	In x86 mode, only EFLAG's carry double - word addition is set	LoongEXT64
X86ADC w.	Only EFLAG's carry characters are set in x86 mode	LoongEXT32
X86ADC. H	Only EFLAG's carry halfwords are set in x86 mode	LoongEXT32
X86ADC. B	Only the carry bytes of EFLAG are set in x86 mode	LoongEXT32
X86ADDU. D.	Set EFLAG in x86 only with no exception of double characters	LoongEXT64

X86ADDU w.	Set EFLAG in x86 only without exception	LoongEXT32
X86SUB. D.	Only double-word subtractions of EFLAG are set in x86 mode	LoongEXT64

Instruction mnemonic	Instruction function description	ISA Compatible Category
X86SUB w.	Only the word subtractions of EFLAG are set in x86 mode	LoongEXT32
X86SUB. H	Only half - character subtractions of EFLAG are set in x86 mode	LoongEXT32
X86SUB. B	Only the byte reductions of EFLAG are set in x86 mode	LoongEXT32
X86SBC. D.	In x86 mode, only EFLAG's band borrow double-word subtraction is set	LoongEXT64
X86SBC w.	Only EFLAG's loanword subtraction is set in x86 mode	LoongEXT32
X86SBC. H	Only EFLAG's band debit halfwords are set in x86 mode	LoongEXT32
X86SBC. B	Only EFLAG's band borrow bytes are set in x86 mode	LoongEXT32
X86SUBU. D.	Only the double-word subtraction of EFLAG is set in x86 mode without exception	LoongEXT64
X86SUBU w.	Set EFLAG in x86 only with no exception word subtractions	LoongEXT32
X86INC. D.	Only double - character augmentation 1 of EFLAG is set in x86 mode	LoongEXT64
X86INC w.	Only EFLAG's word augmentation of 1 is set in x86 mode	LoongEXT32
X86INC. H	Only half - word augmentation of EFLAG is set in x86 mode	LoongEXT32
X86INC. B	In x86 only the byte augmentation of EFLAG is set to 1	LoongEXT32
X86DEC. D.	Only the two-word autodecrement of EFLAG is set in x86 mode	LoongEXT64
X86DEC w.	Only the characters of EFLAG are subtracted by 1 in x86 mode	LoongEXT32
X86DEC. H	Only EFLAG halfwords are subtracted by 1 in x86 mode	LoongEXT32
X86DEC. B	In x86 only the bytes of EFLAG are subtracted by 1	LoongEXT32
X86SLL. D.	Only EFLAG's two-word left shift is set in x86 mode	LoongEXT64
X86SLL w.	Only the left shift of EFLAG words is set in x86 mode	LoongEXT32
X86SLL. H	Only the left shift of EFLAG halfwords is set in x86 mode	LoongEXT32
X86SLL. B	Only the bytes of EFLAG are set left in x86 mode	LoongEXT32
X86DSLL32	Only the shift amount of EFLAG plus the left shift of 32 two-word logic is set in x86 mode	LoongEXT64
X86SLLV. D.	Only the two-word variable shift of EFLAG is set to the left in x86 mode	LoongEXT64
X86SLLV w.	Only the variable word shift of EFLAG is set left in x86 mode	LoongEXT32
X86SLLV. H	Only the half-word variable shift of EFLAG is set left in x86 mode	LoongEXT32
X86SLLV. B	In x86 only the byte variable shift of EFLAG is set to move left	LoongEXT32
X86SRL. D.	Only EFLAG's two-word logic is set to right shift in x86 mode	LoongEXT64
X86SRL w.	Only the right shift of EFLAG's word logic is set in x86 mode	LoongEXT32
X86SRL. H	Only EFLAG's half-word logic is set to right shift in x86 mode	LoongEXT32
X86SRL. B	Only the byte logic of EFLAG is set right in x86 mode	LoongEXT32
X86DSRL32	Only the shift of EFLAG plus the right shift of 32 double word logic is set in x86 mode	LoongEXT64
X86SRLV. D.	Only the two-word variable shift logical right shift of EFLAG is set in x86 mode	LoongEXT64
X86SRLV w.	Only the logical right shift of EFLAG's variable word shift is set in x86 mode	LoongEXT32
X86SRLV. H	Only the half-word variable shift logical right shift of EFLAG is set in x86 mode	LoongEXT32
X86SRLV. B	Only the byte variable shift logical right shift of EFLAG is set in x86 mode	LoongEXT32
X86SRA. D.	Only EFLAG's two-word arithmetic right-shift is set in x86 mode	LoongEXT64
X86SRA w.	In x86 only the word arithmetic shift of EFLAG is set	LoongEXT32

X86SRA. H	Only the half-word arithmetic right shift of EFLAG is set in x86 mode	LoongEXT32
X86SRA. B	Only the byte arithmetic right shift of EFLAG is set in x86 mode	LoongEXT32
X86DSRA32	Only the shift of EFLAG plus the right shift of 32 two-word logical arithmetic is set in x86 mode	LoongEXT64
X86SRAV. D.	In x86 only the two-word variable shift of EFLAG is set to the arithmetic right shift	LoongEXT64

Instruction mnemonic	Instruction function description	ISA Compatible Category
X86SRAV. w.	In x86 only the variable word shift of EFLAG is set to the arithmetic right shift	LoongEXT32
X86SRAV. H	In x86 only the half-word variable shift of EFLAG is set to the arithmetic right shift	LoongEXT32
X86SRAV. B	In x86 only the byte variable shift of EFLAG is set to the arithmetic right shift	LoongEXT32
X86ROTR. D.	Only the right shift of EFLAG's two-word loop is set in x86 mode	LoongEXT64
X86ROTR. w.	Only EFLAG's word loops are set right in x86 mode	LoongEXT32
X86ROTR. H	Only EFLAG's half-word loop is set to right shift in x86 mode	LoongEXT32
X86ROTR. B	In x86 only the byte cycle of EFLAG is set to move right	LoongEXT32
X86DROTR32	In x86 only the shift amount of EFLAG plus the right shift of 32 two-word logic loops	LoongEXT64
X86ROTL. D.	Only EFLAG double-word loops are set left in x86 mode	LoongEXT64
X86ROTL. w.	Only the left shift of EFLAG's word loop is set in x86 mode	LoongEXT32
X86ROTL. H	Only EFLAG's half-word loop is set left in x86 mode	LoongEXT32
X86ROTL. B	Only the byte loop of EFLAG is set left in x86 mode	LoongEXT32
X86DROTL32	Only the shift amount of EFLAG plus the left shift of 32 two-word logic loop is set in x86 mode	LoongEXT64
X86RCR. D.	In x86 only the CF bits of EFLAG are set to right shift	LoongEXT64
X86RCR. w.	In x86 only the CF bits of EFLAG are set to right shift	LoongEXT32
X86RCR. H	In x86 only the CF bits of EFLAG are set to right shift	LoongEXT32
X86RCR. B	In x86, the byte cycle with CF bits of EFLAG is set right	LoongEXT32
X86DRCR32	Only the shift of EFLAG plus 32 double-word logic with CF bit is set in x86 mode Ring moves to the right	LoongEXT64
X86RCL. D.	In x86 only the CF bits of EFLAG are set to the left of a two-word loop	LoongEXT64
X86RCL. w.	In x86 only the CF bits of EFLAG are set to loop left	LoongEXT32
X86RCL. H	In x86 only the CF bit of EFLAG's half-word loop is set to move left	LoongEXT32
X86RCL. B	In x86 mode, only the byte loops with CF bits of EFLAG are set left	LoongEXT32
X86DRCL32	Only the shift of EFLAG plus 32 double-word logic with CF bit is set in x86 mode Ring left	LoongEXT64
X86ROTRV. D.	In x86 mode, only the variable shift amount of EFLAG is set for a two-word cycle right shift	LoongEXT64
X86ROTRV. w.	In x86, a word cycle that only sets the variable shift amount of EFLAG moves right	LoongEXT32
X86ROTRV. H	In x86 mode, only the variable shift amount of EFLAG is set for a half-word loop right shift	LoongEXT32
X86ROTRV. B	In x86 mode, the byte cycle that sets only the variable shift amount of EFLAG moves right	LoongEXT32
X86ROTLV. D.	In x86 mode, only the variable shift of EFLAG's two-word loop is set to the left	LoongEXT64
X86ROTLV. w.	In x86 mode, only the variable shift amount of EFLAG is set to the left of the word loop	LoongEXT32
X86ROTLV. H	In x86 mode, only the variable shift amount of EFLAG is set to the left of a halfword loop	LoongEXT32
X86ROTLV. B	In x86 mode, only the variable shift amount of EFLAG is set to the left of the byte cycle	LoongEXT32
X86RCRV. D.	In x86 only the variable shift of EFLAG with CF bits is set to the right	LoongEXT64
X86RCRV. w.	In x86 only a CF bit of a word with a variable shift of EFLAG is set right	LoongEXT32

X86RCRV. H	In x86 only a CF bit halfword loop with a variable shift of EFLAG is set right	LoongEXT32
X86RCRV. B	In x86 only the variable shift of EFLAG's byte cycle with CF bits is set to the right	LoongEXT32
X86RCLV. D.	In x86 only a CF bit double-word loop with a variable shift of EFLAG is set to the left	LoongEXT64
X86RCLV w.	In x86 only the CF bits of a word with a variable shift of EFLAG are set left	LoongEXT32
X86RCLV. H	In x86 only a CF bit halfword loop with a variable shift of EFLAG is set left	LoongEXT32
X86RCLV. B	In x86 only the variable shift of EFLAG's byte cycle with CF bits is set to the left	LoongEXT32

Instruction mnemonic	Instruction function description	ISA Compatible Category
X86MFFLAG	The value of the EFLAG bit is extracted in x86 mode	LoongEXT32
X86MTFLAG	Modify the value of the EFLAG flag bit in an x86 manner	LoongEXT32
X86J	Jump in X86 mode based on EFLAG values	LoongEXT32
X86LOOP	The loop is based on EFLAG values in X86 mode	LoongEXT32
SETTM	The x86 floating-point stack mode is set	LoongEXT32
CLRTM	X86 floating-point stack mode cleared	LoongEXT32
INCTOP	Pointer to the top of the x86 floating-point stack plus 1	LoongEXT32
DECTOP	The pointer at the top of the x86 floating-point stack minus 1	LoongEXT32
MTTOP	Write the x86 floating-point top pointer	LoongEXT32
MFTOP	Read the pointer to the top of the x86 floating-point stack	LoongEXT32
SETTAG	Determines the collocation register	LoongEXT32
The CVT transmission. D.L D	Extended double to double precision	LoongEXT32
The CVT transmission. LD. D	Double precision converted to extended double precision low position	LoongEXT32
The CVT transmission. UD. D	Double precision converted to extended double precision high order	LoongEXT32

Binary translation Acceleration instruction (ARM)

Table 2-30 Acceleration instructions for ARM binary translation extension

Instruction mnemonic	Instruction function description	ISA Compatible Category
ARMADC	Only EFLAG is set with carry word and ARM conditional execution	LoongEXT32
ARMADD	Word plus, ARM conditional execution only set EFLAG	LoongEXT32
ARMSBC	With carry subtracting, only EFLAG is set for ARM conditional execution	LoongEXT32
ARMSUB	Subtracting, only EFLAG is set for ARM conditional execution	LoongEXT32
those	Only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMBIC	Only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMMOV	The word moves and only EFLAG is set for ARM mode conditional execution	LoongEXT32
ARMMVN	The word is moved backwards, and only EFLAG is set for ARM conditional execution	LoongEXT32
ARMMVLO32	Low 32 bit LO register moves to general purpose register, ARM mode conditional execution only Buy EFLAG	LoongEXT32
ARMMVHI32	HI register low 32 bit moves to general purpose register, ARM mode conditional execution only Settings EFLAG	LoongEXT32
ARMMVACC64	The low 32 bits of the HI register and the low 32 bits of the LO register are combined into 64 bits in ARM mode Conditional execution sets only EFLAG	LoongEXT32
ARMOR	Only EFLAG is set in ARM mode conditional execution	LoongEXT32

ARMORN	Only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMXOR	Only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMSLL	Move the word left and set only EFLAG in ARM mode conditional execution	LoongEXT32
ARMSLLV	The variable shift quantity word moves left, and only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMSRL	The word logic is shifted to the right, and only EFLAG is set for ARM mode conditional execution	LoongEXT32
ARMSRLV	Variable shift quantity word logic moves right, and only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMSRA	The word arithmetic is shifted to the right, and only EFLAG is set for ARM conditional execution	LoongEXT32
ARMSRAV	The variable shift quantity word arithmetic moves right, and only EFLAG is set in ARM mode conditional execution	LoongEXT32

Instruction mnemonic	Instruction function description	ISA Compatible Category
ARMROTR	The word loops right, and only EFLAG is set for ARM conditional execution	LoongEXT32
ARMROTRV	The variable shifted quantity word circularly moves right, and only EFLAG is set in ARM mode conditional execution	LoongEXT32
ARMRRX	ARM conditional performs a right shift of a carry word loop that sets EFLAG only	LoongEXT32
ARMFCMP. F32	ARM FCMP.f32 instruction is used for floating point comparison and FCR1 EFLAGS are set	LoongEXT32
ARMFCMP. F64	ARM FCMP.f64 instruction is used for floating point comparison and FCR1 EFLAGS are set	LoongEXT32
ARMFCMPE. F32	ARM FCMP.f32 instruction is used for floating point comparison and FCR1 EFLAGS are set	LoongEXT32
ARMFCMPE. F64	ARM FCMP.f64 instruction is used for floating point comparison and FCR1 EFLAGS are set	LoongEXT32
ARMMOVE	The general purpose registers move between each other in ARM mode according to EFLAG conditions	LoongEXT32
ARMMFHI	HI moves to the general register in ARM mode according to EFLAG conditions	LoongEXT32
ARMMFLO	LO moves to the general purpose register in ARM mode according to EFLAG conditions	LoongEXT32
ARMMFFCR	Move FCR1 EFLAGS to EFLAGS by ARM according to EFLAG conditions	LoongEXT32
ARMFMOV. S	Move single precision Numbers between floating point registers in ARM mode according to EFLAG conditions	LoongEXT32
ARMFMOV. D.	Floating point registers move doubles in ARM mode according to EFLAG conditions	LoongEXT32
ARMJ	Jump according to EFLAG value in ARM mode	LoongEXT32
ARMMFFLAG	The value of EFLAG bit is extracted by ARM	LoongEXT32
ARMMTFLAG	Modify the value of the EFLAG bit by ARM	LoongEXT32

64-bit multimedia acceleration instructions

Table 2-31 Loong core extension 64-bit multimedia acceleration instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
PADDSH	Four 16-bit signed integers plus, sign saturation	LoongEXT32
PADDUSH	Four 16-bit unsigned integers plus, unsigned saturation	LoongEXT32
PADDH	Four 16-digit Numbers plus	LoongEXT32
PADDW	Two 32 digits plus	LoongEXT32
PADDSB	Eight 8-bit signed integers plus, sign saturation	LoongEXT32
PADDUSB	Eight 8-bit unsigned integers plus, unsigned saturation	LoongEXT32
PADDDB	Eight 8-digit Numbers plus	LoongEXT32
PADDD	64 digits add	LoongEXT32
PSUBSH	Four 16-bit signed integers minus, sign saturation	LoongEXT32
PSUBUSH	Four 16-bit unsigned integers minus, unsigned saturation	LoongEXT32
PSUBH	Four 16-digit subtractions	LoongEXT32
PSUBW	Two 32-digit subtractions	LoongEXT32
PSUBSB	Eight 8-bit signed integers minus, sign saturation	LoongEXT32

PSUBUSB	Eight 8-bit unsigned integers minus, unsigned saturation	LoongEXT32
PSUBB	Eight 8-digit subtractions	LoongEXT32
PSUBD	64 digits	LoongEXT32
PSHUFH	The Shuffle has four 16-digit digits	LoongEXT32
PACKSSWH	32 bit signed integer converted to 16 bits, sign saturation	LoongEXT32
PACKSSHB	16 bit signed integer converted to 8 bit, sign saturation	LoongEXT32
PACKUSHB	16 bit signed integer converted to 8 bit, unsigned saturation	LoongEXT32
PANDN	Fs is not followed by ft bitwise and	LoongEXT32

Instruction mnemonic	Instruction function description	ISA Compatible Category
PUNPCKLHW	Unpack is low by 16 digits	LoongEXT32
PUNPCKHHW	Unpack is 16 digits high	LoongEXT32
PUNPCKLBH	Unpack is low by eight digits	LoongEXT32
PUNPCKHBH	Unpack is eight digits high	LoongEXT32
PINSRH_0	The ft low 16 bits are inserted into the FS low 0 16 bits	LoongEXT32
PINSRH_1	The ft low 16 bits are inserted into the FS low 16 bits	LoongEXT32
PINSRH_2	The ft low 16 bits are inserted into the FS low 2 16 bits	LoongEXT32
PINSRH_3	The ft low 16 bits are inserted into the FS low 3 16 bits	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 take a larger value	LoongEXT32
PMINSH	Four 16-bit signed integers take smaller values	LoongEXT32
PMAXUB	Eight 8-bit unsigned integers take a larger value	LoongEXT32
PMINUB	Eight 8-bit unsigned integers take smaller values	LoongEXT32
PCMPEQW	Two 32 - digit Numbers equal to compare	LoongEXT32
PCMPGTW	Two 32-bit signed integers are greater than the comparison	LoongEXT32
PCMPEQH	Four 16-digit Numbers are equal for comparison	LoongEXT32
PCMPGTH	Four 16-bit signed integers are greater than the comparison	LoongEXT32
PCMPEQB	Eight 8-digit Numbers are equal for comparison	LoongEXT32
PCMPGTB	Eight 8-bit signed integers are greater than the comparison	LoongEXT32
PSLLW	Two 32-bit logical moves left	LoongEXT32
PSLLH	Four 16-digit logical left shifts	LoongEXT32
PMULLH	Multiply four 16-bit signed integers and you get 16 bits lower	LoongEXT32
PMULHH	Multiply four 16-bit signed integers and take the result 16 bits higher	LoongEXT32
PMULUW	Multiply low 32 - bit unsigned integers and save 64 - bit results	LoongEXT32
PMULHUH	Multiply four 16-bit unsigned integers and take 16 bits higher	LoongEXT32
PSRLW	Two 32-bit logical moves to the right	LoongEXT32
PSRLH	Four 16-digit logical shifts to the right	LoongEXT32
PSRAW	Two 32-bit arithmetic moves to the right	LoongEXT32
PSRAH	Four 16-digit arithmetic moves to the right	LoongEXT32
PUNPCKLWD	The lower 32 digits combine into 64 digits	LoongEXT32
PUNPCKHWD	The higher 32 digits combine into 64 digits	LoongEXT32
PASUBUB	Eight 8-bit unsigned integers subtract and take the absolute value	LoongEXT32
PEXTRH	Fs 16 bits copy to FD low 16 bits, FD high fill 0	LoongEXT32
PMADDHW	The four 16-bit signed Numbers are multiplied, and the low and high Numbers are added together	LoongEXT32
BIADD	Multi-byte summation	LoongEXT32
PMOVMSKB	Byte symbol bit extraction	LoongEXT32
G SXOR	Fs and FT logical heterotopic or	LoongEXT32
G SNOR	Fs and FT logical bits or not	LoongEXT32
G SAND	Fs and FT logical bits and	LoongEXT32

GSADDU	Fs and FT fixed point unsigned word plus	LoongEXT32
--------	--	------------

Instruction mnemonic	Instruction function description	ISA Compatible Category
GSOR	Fs with FT fixed point logical bit or	LoongEXT32
GSADD	Fs and FT fixed point word plus	LoongEXT32
GSDADD	Fs and FT fixed point double word addition	LoongEXT32
GSSEQU	Comparison of FS and FT fixed points	LoongEXT32
GSSEQ	Comparison of FS and FT fixed points	LoongEXT32
GSSUBU	Fs and FT fixed point unsigned word subtraction	LoongEXT32
GSSUB	Fs and FT fixed point word subtraction	LoongEXT32
GSDSUB	Fs and FT fixed point double word subtraction	LoongEXT32
GSSLTU	The number of unsigned fixed points of FS and FT is less than the comparison	LoongEXT32
GSSLT	The number of FS and FT fixed points is less than the comparison	LoongEXT32
GSSLL	Fs and FT fixed point logic left-shift words	LoongEXT32
GSDSLL	Fs and FT fixed point logic left shift double word	LoongEXT32
GSSRL	Fs and FT fixed point logic right shift word	LoongEXT32
GSDSRL	Fs and FT fixed point logic right shift double word	LoongEXT32
GSSRA	Fs and FT fixed point arithmetic right shift word	LoongEXT32
GSDSRA	Fs and FT fixed point arithmetic shift double word to the right	LoongEXT32
GSSLEU	Fs and FT fixed point unsigned fixed point number less than or equal to comparison	LoongEXT32
GSSLE	Fs and FT fixed point number less than or equal to comparison	LoongEXT32

Miscellaneous instructions

Table 2-32 Loong core extension miscellaneous instructions

Instruction mnemonic	Instruction function description	ISA Compatible Category
CTZ	0 number of trailing words	LoongEXT32
The CTO	The number of trailing words	LoongEXT32
DCTZ	The number of 0 tags followed by two characters	LoongEXT64
DCTO	Double word trailing 1 number	LoongEXT64
CAMPV	Query the RAM table item values for the lookup table	LoongEXT32
CAMPI	Query the table entry index of the lookup table	LoongEXT32
CAMWI	Fill in the search form	LoongEXT32
RAMRI	Read the RAM contents of the lookup table	LoongEXT32

3 Processor running mode

GS464E is compatible with THE MIPS64 specification and contains two modes: Debug Mode and Root Mode. Debug mode mainly corresponds to the running environment of EJTAG exception handler. The root mode corresponds to the running environment of the operating system and the software on the real host. Among them, the Root Mode can be further divided into root-kernel Mode, root-Supervisor Mode and root-user Mode. All modes of operation are independent of each other, which means that the processor can only exist in one mode of operation at any one time.

Among the above four modes, the root-regulatory mode is rarely used in practice, and the MIPS specification has not fully defined its connotation. Therefore, this mode will only be briefly explained in the following description of this manual. Programmers are not advised to build software using the root-regulatory pattern, and refer directly to the MIPS specification if necessary.

3.1 Processor run mode definition

Table 3-1 shows the decision basis of each processor mode.

Table 3-1 Processor mode decision basis

Root CP0				model
The Debug. DM	The Status, ERL	Status. EXL	Status. KSU	
1	Don 't care			Debug mode
0	1	Don 't care		The root-core pattern
	0	1	Don 't care	
		0	00	Root-regulatory model
			01	Root-user mode
Don 't care			11	meaningless

3.1.1 Debug mode

Debug mode has the highest priority. In debugging mode, the software can operate all the processor resources, including changing virtual and real address mapping relationship, controlling system environment and process switching, etc.

3.1.2 The root-core pattern

In root-core mode, software can operate all processor resources, including changing virtual and real address mapping relationship, controlling system environment and process switching, etc. The processor is powered back into root-core mode.

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 non-privileged instructions, using general purpose registers and floating point registers, and all accesses fall into a flat uniform virtual address space. Normal user programs run in root-user mode.

4 Memory management

4.1 The basic concept

4.1.1 Address space

The address space is the range of addresses that can be covered by a particular addressing mode. The MIPS64 architecture includes a 64-bit address space and a 32-bit address space that maps to a subset of the former.

4.1.2 Segment and segment size (SEGBITS)

Segment is a subset of address space. The address space within the same segment has a consistent mapping mode and access properties. As defined by the MIPS64 specification, for example, its 32-bit address space is divided into a series of segments of size 2 or 2 bytes, and its 64-bit address space can theoretically support segments no larger than 2 bytes. 293162 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 2-byte segments. SEGBITS

4.1.3 Physical Address size (PABITS)

The physical address size (PABITS) determines the actual size of the physical address space supported by the processor to be 2 bytes. PABITS

4.1.4 Mapped Address and Unmapped Address

When an Address is Mapped, it is an Address that needs to be translated into a virtual Address using a TLB. Unmapped Address refers to an Address that does not need to be transformed into a virtual or real Address through TLB, and that the virtual Address is mapped linearly to the lowest part of the physical Address.

4.2 Host virtual address space

4.2.1 Host address space division and access control

Table 4-1 gives the division of host address space and defines the legitimacy determination and address mapping method of each address segment. It should be noted that in the host address space, SEGBITS are always 48.

Table 4-1 Host address space division and access control

Section names	Address range	Legitimacy determination and address mapping method		
		User mode	Regulatory model	The core model
	0 XFFFF. FFFF. FFFF.			Mapped address segment TLB re-fill exception type: TLB (Status. The KX = 0)

kseg3	<p>FFFF ~ 0 XFFFF. FFFF. E000.0000</p>	Illegal address segment	Illegal address segment	<p>XTLB (Status. The KX = 1). When debug.dm =1 Special handling of address space is requested See 4.2.5 section</p>
-------	--	-------------------------	-------------------------	---

Section names	Address range	Legitimacy determination and address mapping method		
		User mode	Regulatory model	The core model
Ksseg	0 XFFFF. FFFF. DFFF. FFFF ~ 0 XFFFF) FFFF) C000.0000	Illegal address segment	Type of exception: TLB (status.kx =0) XTLB (Status. The KX = 1).	Type of exception: TLB (status.kx =0) XTLB (Status. The KX = 1).
kseg1	0 XFFFF. FFFF. BFFF. FFFF ~ 0 XFFFF) FFFF) A000.0000	Illegal address segment	Illegal address segment	Non-mapped address segment Please refer further to section 4.2.2
kseg0	0 XFFFF. FFFF. 9 FFF. FFFF A 8000.0000 ~ 0 XFFFF) FFFF)	Illegal address segment	Illegal address segment	Non-mapped address segment Please refer further to section 4.2.2
	0 XFFFF. FFFF. 7 FFF. FFFF ~ 0 xc000. FFFF. 8000.0000	Illegal address segment	Illegal address segment	Illegal address segment
xkseg	0 xc000. FFFF. 7 FFF. FFFF ~ 0 xc000. 0000.0000.0000	Illegal address segment	Illegal address segment	Status.KX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.
xkphys	0 XBFFF. FFFF. FFFF. FFFF ~ 0 x8000. 0000.0000.0000	Illegal address segment	Illegal address segment	Status.KX=0 is the illegal address segment; Otherwise legitimate, its internal legitimacy is determined and the address map adopted please See section holdings
	0 x7fff. FFFF. FFFF. FFFF ~ 0 x4001. 0000.0000.0000	Illegal address segment	Illegal address segment	Illegal address segment
Xksseg	0 x4000. FFFF. FFFF. FFFF ~ 0 x4000. 0000.0000.0000	Illegal address segment	Status.SX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.	Status.SX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.
	0 x3fff. FFFF. FFFF. FFFF ~ 0 x0001. 0000.0000.0000	Illegal address segment	Illegal address segment	Illegal address segment

<p>Xkuse g xsuseg xuseg</p>	<p>0 x0000. FFFF. FFFF. FFFF ~ 0 x0000. 0000.8000.0000</p>	<p>Status.UX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.</p>	<p>Status.UX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.</p>	<p>Status.UX=0 is the illegal address segment; Otherwise valid, for mapped address segment TLB re-fill exception type: XTLB.</p>
---	--	--	--	--

Section names	Address range	Legitimacy determination and address mapping method		
		User mode	Regulatory model	The core model
Kuseg suseg useg	0 x0000. 0000.7 FFF. FFFF ~ 0 x0000. 0000.0000.0000	Map address segment TLB re-fill exception type: TLB (status.ux =0) XTLB (status.ux =1).	The exception type is TLB (status.kx =0) XTLB (status.kx =1).	Status.ERL=1 is non-mapped Please refer to the address section for further information 4.2.4 Status.UX=0 is the mapped address segment, and TLB refills in the exception type: TLB (Status. The UX = 0) XTLB (Status. The UX = 1)

4.2.2 The address translation, cacheability and cache consistency of host address space Kseg0 segment and Kseg1 segment

The Kseg0 segment and Kseg1 segment of the host address space are directly mapped to the minimum 0.5G(2) byte of the physical address space, namely the virtual address 29

Ffff.ffff.a000.000 ~ 0xffff.ffff.Bff.ffff radiates to physical address 0x0000.0000.0000.0000~

0x0000.0000.1 fff.ffff, virtual address 0xffff.ffff.8000.000 ~ 0xffff.ffff.9fff.FFff also maps to physical address

0 x0000. 0000.0000.0000 ~ 0 x0000. 0000.1 FFF. FFFF. The cache consistency property of the Kseg0 segment is determined by the Config.k0 domain, as defined in section 7.28 on page 116. Kseg1 section is always a non-cached attribute (Uncached).

4.2.3 The address translation and cacheability of the host address space Xkphys segment are consistent with the cache properties

The Xkphys segment of the host address space is non-mapped and contains 8 sub-address segments, each of which has a size of 2 bytes. 48 The virtual address resolution of xkphys segment is shown in Figure 4-1. The [58:48] of a virtual address must be all 0, otherwise it is an illegal address. The [47:0] of the virtual address is not translated by TLB or any other process. Directly as a physical address. The [61:59] bit of the virtual address is used to define the cache consistent property of the corresponding subaddress segment. See Table 7-6 on page 89 for a definition of the encoding used.

Figure 4-1 Xkphys segment virtual address resolution

63	62	61	59	58	48	47	0
10	Cache consistent properties		Anything less than 0 is an illegal address			Physical address	

4.2.4 Address translation of the kusEG segment of the host address space when status.erl =1

When status. ERL=1, kusEG segment is non-mapped address segment, and its cache consistent attribute is non-cached, similar to kseg1 segment. This feature allows the software to use the GENERAL register R0 as the base address register to store other general registers in memory when handling Cache error exceptions. At the same time, due to errors in the Cache, access operations are no longer cached.

4.2.5 Special treatment of host address space kseg3 when debug.dm =1

When the processor is in Debug mode (debug.dm =1), the address range of virtual address 0xffff.ffff.ff20.0000 ~ 0xffff.ffff.ff3f.ffff will serve as the special memory address mapping region -- the Dseg segment of the EJTAG. See error # for a detailed description of the DSEG section in EJTAG! No reference source was found. Chapter.

4.2.6 Special handling of data access to virtual addresses when status.ux =0 in user mode

In user mode, special processing is required for virtual addresses of data access when running 32-bit programs compatible on a 64-bit MIPS processor. This is because a calculation that yields a valid address on a 32-bit MIPS processor may have unintended effects on a 64-bit MIPS processor. For example, the following sequence of instructions:

```
La r1, 0 x80000000
```

Lw r2, 4 (r1)

When executed on a 32-bit MIPS processor, the virtual address of the LW instruction is $0x80000000 + 0xFFFFFFFF = 0x7FFFFFFF$. The address is still in the KusEG section. But when this code is executed on a 64-bit MIPS processor, the virtual address of the LW instruction is $0xFFFFFFFF80000000$

+ $0xffffffff = 0xffffffff7fffffff$. The obtained address is no longer in kusEG section, which will result in an address error exception. In order to reserve 64-bit processors compatible with 32-bit programs, special processing is done for the calculation of data access virtual addresses when $status.ux = 0$. In this way, the address operation still USES the addition of two 64 bits after symbol expansion, but the high 32 bits of the result are discarded

The 31 bit symbol of the result is extended to 63.. of the result virtual address. 32. The virtual address result after this special treatment is used for address legitimacy checking, TLB mapping, and so on. Normal fingerprinting does not involve this problem, because the 31st bit of a valid PC in 32-bit user mode must be 0, and the violation mentioned above does not occur.

4.3 TLB – based virtual and real address mapping

TLB is a temporary cache in the processor that holds the operating system page table information and is used to speed up the process of virtual and real address translation for pointing and accessing operations in the mapped address space.

4.3.1 TLB hierarchy

Two levels of TLB are implemented in GS464E. The first level TLB is a fully linked look-up TLB with small capacity that is respectively contained in the retrieval and retrieval components, which is called ITLB and DTLB respectively. The second level TLB has a larger capacity and contains a fully linked and an eight-way group linked lookup table, called JTLB.

All content software for JTLB is visible, and the loading and replacement of table items is managed by the software. All content software of ITLB and DTLB is not visible, and the loading replacement of table items is maintained by hardware. While the processor is running, the contents of the DTLB and the page tables stored in the JTLB are always contained, which is maintained automatically by the hardware. However, the contents stored in ITLB and JTLB do not explicitly contain and mutually exclusive relationship, that is, the software cannot determine the information stored in ITLB by maintaining the contents in JTLB. In general operation, the information stored in ITLB and JTLB does not maintain the inclusion relationship, which has no impact on the correctness of the software. However, if the operating system is trying to modify an existing page table information and there is program code in the address space corresponding to the page table entry, the system software must explicitly empty all the contents of ITLB by writing 1 to the `diag.itLB` bit to ensure that the contents of the page table entry are no longer in ITLB before modification.

Both ITLB and DTLB adopt full - phase lookup table structure. The table item information in ITLB and DTLB are all from JTLB, in which the table item information in DTLB is completely consistent with the format in JTLB, while each item in ITLB only holds one page table rather than a pair of odd-even adjacent page tables. Because ITLB and DTLB do not require software to replace the state, the exact format of their table entries is not expanded here.

4.3.2 JTLB structure

From the perspective of software, JTLB contains a fully linked lookup table and a multi-group linked lookup table. The former is called variable-page-size TLB(VTLB for short) because it supports different Page sizes for different table items. The latter is called Fixed Page Size TLB(fixed-Page-size TLB) or FTLB when all table item pages are the same Size at the same time. Both VTLB and FTLB are searched during the conversion process. Accordingly, the software needs to ensure that VTLB and FTLB do 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, with 64 entries in GS464E. If the gsconfig. VTLBOnly position is 1, then all functions of FTLB are disabled, leaving only VTLB. With this configuration, the TLB management part of the existing operating system on the 3A1000 chip can be executed correctly on the 3A3000 chip without modification.

The FTLB is an 8-way group linkage structure, each path contains 128 items, a total of 1024 items, and each item holds 2 page table information, so it can store up to 2048 page table information. During the search, the hardware extracts the $[(17 + \text{Config4.ftlbPagesize}) : (11 + \text{config4.ftlbpagesize})]$ bit of the virtual address as the index information, and compares the contents of the same index position item in each line to determine whether there is a match.

4.3.3 JTLB table item

The format of VTLB and FTLB table entries is basically the same, except that each table entry with VTLB contains PageMask information, while FTLB does not reserve PageMask information because it is the same page size. Each JTLB table entry contains two parts: the comparison part and the physical transformation part.

The comparison portion of table items includes:

- Invalid bit for page table (EHINV). When the bit is 1, the page table entry does not participate in finding a match.
- Address space number (MID). Identifies 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 entry address resides.
- Virtual address area identification (R) and virtual page number (VPN2). In THE MIPS architecture, each page table entry contains a pair of adjacent odd and even 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 least bit of the virtual page number is not stored, and it is only used for the physical transformation information to decide whether to choose the odd page number or even page number in the search.
- Address space identity (ASID) and global identity bit (G). The address space identity is used to distinguish the same virtual address in different processes. The operating system assigns a unique ASID to each process. In addition to the consistent address information, THE TLB also needs to compare the ASID information in the search. When the operating system needs to share the same virtual address among all processes, the G bit in the TLB page table entry can be set. After G position 1, the TLB lookup will not be checked for ASID consistency.
- Address page 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 page size increments of 4. With VTLB, each item has Mask information, so different items can correspond to different page sizes. For FTLB, all items use the unified Mask information, which is determined by Config4.FTLBPageSize domain.

Further descriptions of the above fields can be found in sections 7.19 of the EntryHi Register (CP0 Register 10, Select 0), 7.4 of the EntryLo0 and EntryLo1 registers (CP0 Register 2 and 3, Select 0), and 7.7 of the PageMask Register (CP0 Register 5, Select 0).

The physical transformation section of the table entry contains the physical transformation information of a pair of odd and even adjacent page tables. The transformation information of each page includes:

- Physical page number (PFNX & PFN).
- The significant bit (V).
- Dirty bits (D). The control bit of whether a page is writable, not whether the page is written to the state bit of dirty data.
- Read the forbidden bit (RI).
- Execute the forbidden bit (XI).
- The kernel executes the protection bit (K), which is only meaningful in 64-bit mode. In 32-bit mode, reserve gsconfig.ke constant to 0 to ensure K Bits don't work.
- Cache property (C).

For a further explanation of each of these fields, see the description in Section 7.4 of the EntryLo0 and EntryLo1 registers (CP0 Register 2 and 3, Select 0).

4.3.4 TLB software management

GS464E still follows the traditional MIPS architecture's management mode of TLB, that is, it adopts the software-led and software-combined management mode of TLB. The new Hardware Page Table Walking feature added in release 5 and later was not implemented in GS464E. GS464E provides, on top of the MIPS specification, a set of privileged access instructions for page-table traversal lookup designed to speed up this process, which is briefly described in the second half of this section.

The exception

TLB to the actual situation by the hardware address translation process automatically, but when no match in the TLB, although matching page table entry is invalid or illegal access, you need to trigger the exception, to the operating system kernel, or other regulatory processes, further processing by software, to maintain the content of the TLB, or to the legitimacy of the program execution for final decision. Exceptions related to TLB management in GS464E are:

1. TLB rewrites the exception
2. XTLB rewrites the exception
3. TLB is not an exception
4. TLB modification exceptions
5. Unenforceable exception
6. Unreadable exception

Associated CP0 register

Table 4-2 lists the CP0 registers associated with TLB management. See the corresponding sections in Chapter 7 for a detailed description of each register. **Table 4-2 TLB management related CP0 reg**

Reg.	Sel.	Register name	Function definition	The index
0	0	The Index	VTLB accesses the specified index register with FTLB	Page 85, section 7.2
1	0	The Random	VTLB with FTLB access to random index registers	Page 86, section 7.3
2	0	EntryLo0	VTLB and FTLB table entry low order content associated with even number of virtual pages	Page 87, section 7.4
3	0	EntryLo1	VTLB and FTLB table entry low order content related to odd number of virtual pages	Page 87, section 7.4
4	0	The Context	A pointer to an in-memory page table entry	Page 90, section 7.5
5	0	PageMask	VTLB page table size control	Page 92, section 7.7
5	1	PageGrain	1KB small pages and other page table property control	Page 93, section 7.8
5	5	PWBase	Page table base address register	Page 94, section 7.9
5	6	PWField	Configure the page table address index location for each level	Page 95, section 7.10
5	7	PWSize	Configure the page table pointer size for each level	Page 96, section 7.11
6	0	Wired	Control the number of fixed items in VTLB	Page 97, section 7.12
6	6	PWctl	Control multilevel page table configuration	Page 98, section 7.13
8	0	BadVAddr	Record the error address for the latest address-related exception	Page 100, section 7.15
9	7	PGD	Page table pointer register	Page 103, section 7.18

Relevant privilege instruction

Table 4-3 lists privilege instructions related to TLB management. For The detailed definition of each Instruction, please refer to The MIPS® Architecture For Programmers Volume II-A: The MIPS64® Instruction Set (Rev5.03) and The Godson Instruction Set Manual (Volume II-A) - Custom General Extension Instruction (Volume I) (Rev1.00).

Table 4-3 TLB manages related privilege instructions

Instruction mnemonic	Simple description of instruction
-------------------------	--

Instruction mnemonic	Simple description of instruction
TLBP	Search for matches in TLB
TLBR	Read the TLB table entry for the index
TLBWI	Write the TLB table entry for the index
TLBWR	Write random TLB table entries
TLBINVF	Invalidate all TLB table entries
LWDIR	32 bit mode next page table directory entry load instruction
LWPTE	32 bit mode next page table page table item load instruction
LDDIR	64-bit mode next page table directory entry load instruction
LDPTE	Next page table page table item load instruction in 64-bit mode

GS464E implements the EHINV field in the EntryHi register. When entryhi.ehinv is set to 1, the TLBWI instruction executes Index

The TLB table entry specified is invalid.

VTLB and FTLB use the same hardware and software interface, that is, the same SET of CP0 registers and the same TLB privileged instructions. The following points are highlighted here:

- VTLB retains some non-random substitutable items via the Wired register, while there are no non-random substitutable items in FTLB.
- When the TLBWR instruction is used to randomly populate a TLB table entry. If the page size in Pagemask register is different from the fixed page size of FTLB, the table entries are randomly filled into VTLB. If the same, the table entries are randomly filled into FTLB.
- The randomly filled position in VTLB is determined by the value of the Random register; The location of random entries in the FTLB is determined by a separate hardware pseudo-random number generator in the processor.
- When TLBRI and TLBWI are used to read and write JTLB, the value in the Index register is 0~63 corresponding to the 0~63 item of VTLB, and the value is 64~1087 corresponding to the 0~127 item of FTLB, 0~127 item of FTLB, and..... , the 0th ~127 items of the 7th route. When using the TLBP instruction for software lookup, the results stored in the Index register also follow the above correspondence.

4.3.5 TLB initialization and clearing

It is recommended to initialize or empty TLB using the TLBINVF directive to invalidate all items in the TLB. You can also use the EHINV field in the EntryHi register to loop through the TLBWI instruction, invalidating all items in the TLB one by one.

There is a correspondence between the location of the FTLB table entry and its virtual address, so the traditional TLB initialization method that writes a specific address in the TLB table entry may not ensure proper initialization. When gsconfig.VTLBOnly=0, that is, when FTLB is enabled, the software must initialize or empty the TLB using the two methods described above. Beyond traditional TLB initialises, interested readers can refer to section 4.11.3 of The MIPS® Architecture For Programmers Volume III: The MIPS64® and microMIPS64™ Privileged Resource Architecture (Rev5.03).

4.3.6 TLB - based virtual address translation process

This paper only introduces the virtual and real address translation process based on software visible TLB. Both VTLB and FTLB are searched by processor hardware. The process of checking VTLB first and then FTLB in

```
// VA -- virtual address to be found
// mid -- Virtual address to be found belongs to MID

/* VTLB
ZhaZhaoGuoCheng
*/
```

pseudo-code is described below for the convenience of description. The system software needs to ensure that the same virtual address cannot have multiple hits in VTLB and FTLB.

```

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 [because] & ~ VTLB [I] VPMSK) == (va / 47:40 & ~ Diag.
        VPMSK)) && (VTLB [I] VPN2 == va [39:31] wherefore doth) &&
      ((VTLB [I] VPN2 [17:0] & ~ VTLB [I] MASK) == (va & ~ VTLB [I] MASK))) then 80..13
  If (! 1 case
    VTLB[i].MASK
    vtlb_found) then
    vtlb_found □
      0 0000 0000 0000 0000: vtlb_evenoddbit ← 12 // 4 KB
      b00 page
      0 0000 0000 0000 0011: vtlb_evenoddbit ← 14 // 16 KB
      b00 page
      0 0000 0000 0000 1111: vtlb_evenoddbit ← 16 // 64 KB
      b00 page
      0 0000 0000 0011 1111: vtlb_evenoddbit ← 18 // 256 KB
      b00 page
      0 0000 0000 1111 1111: vtlb_evenoddbit ← 20 // 1 MB
      b00 page
      0 0000 0011 1111 1111: vtlb_evenoddbit ← 22 // 4 MB
      b00 page
      0 0000 1111 1111 1111: vtlb_evenoddbit ← 24 // 16 MB
      b00 page
      0 0011 1111 1111 1111: vtlb_evenoddbit ← 26 // 64 MB
      b00 page
      0 1111 1111 1111 1111: vtlb_evenoddbit ← 28 // 256 MB
      b00 page
      0 1111 1111 1111 1111: vtlb_evenoddbit ← 30 // 1 gb
      b11 page
    Otherwise: UNDEFINED
  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
  The 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

Endif
endfor

/* FTLB
ZhaZhaoGuoCheng
*/ ftlb_found = 0
Case Config4 FTLBPageSize
    0 d1 : independence idx = va[18:12];
    Mask x00000 = 0 0 0 d2 : independence
    idx = va[20:14];    Mask x00003 = 0 0 0
    d3 : independence idx = va[22:16];
    Mask x0000f = 0 0 0 d4 : independence
    idx = va[24:18];    Mask x0003f = 0 0 0
    d5 : independence idx = va[26:20];
    Mask x000ff = 0 0 0 d6 : independence
    idx = va[28:22];    Mask x003ff = 0 0 0
    d7 : independence idx = va[30:24];
    Mask x00fff = 0 0 0 d8 : independence
    idx = va[32:26];    Mask x03fff = 0 0 0
    d9 : independence idx = va[34:28];
    Mask x0ffff = 0 0 0 d10: independence
    idx = va[36:30];    Mask = 0 0 x3ffff
    otherwise: UNDEFINED
endcase
For set = 0 to 7 step 1
    FTLB[SET][IDX]
    If ((FTLB [set] [r]. Independence
        idx EHINV == 0) && (FTLB [set]
        [r]. Independence idx MID ==
        MID) &&
        ((FTLB [set] [R]. Independence idx VPID & FTLB [set] [R]. Independence idx VPMSK) =
        = (Diag. VPID & Diag. VPMSK)) && (FTLB [set] [R]. Independence idx G || (FTLB [set]
        [R]. Independence idx ASID == EntryHi. ASID)) && (FTLB [set] [R]. Independence idx
        R == va [63-62]) &&
        ((FTLB [set] [r]. Independence idx VPN2 [because] & ~ FTLB [set] [r]. Independence
        idx VPMSK) == (va / 47:40 &
        && ~ Diag. VPMSK))
        (FTLB [set] [r]. Independence idx VPN2 == va [39:31] wherefore doth) &&
        ((FTLB [set] [r]. Independence idx VPN2 [17:0] & ~ mask) ==
        (va & to mask))) then the if (! 30..13 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

The 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

  Ftlb_ri □
  FTLB[set][idx].RI1 ftlb_xi □
  FTLB[set][idx].XI1 ftlb_k □
  FTLB[set][idx].K1
Endif
else
  UNDIFINED
Endif
endif
endfor

/* Legitimacy check and physical address generation */
If (vTLB_found && FTLb_found) then
  UNDIFINED
Elseif (vtlb_found) then
  if (! Vtlb_v) then
    Endif SignalException (TLBInvalid,
reftype)
  If (vTLB_RI && (refType == load)) then
    if (pagegrb.IEC) then
      SignalException (TLBRI, reftype)
    else
      Endif SignalException (TLBInvalid,
reftype)
    endif
  If (vTLB_xi && (reftype == fetch)) then
    if (pagegrb.IEC) then
      SignalException (TLBXI, reftype)
    else
      Endif SignalException (TLBInvalid,
reftype)
    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 (pagegrb.IEC) then
        SignalException (TLBRI, reftype)
    else
        SignalException (TLBInvalid reftype)
    Endif
endif
If (FTlb_XI && (refType == fetch)) then
    if (pagegrb.IEC) then
        SignalException (TLBXI,
reftype) else
        Endif SignalException (TLBInvalid,
reftype)
endif
If (! Ftlb_d && (RefType == store))
    then SignalException (TLBModified)
endif
PAddr □ ftlb_pfn[35:ftlb_evenoddbit-12] || va[ftlb_evenoddbit-1:0]
else
    Endif SignalException (TLBMiss,
reftype)

```

5 Organization and management of caches

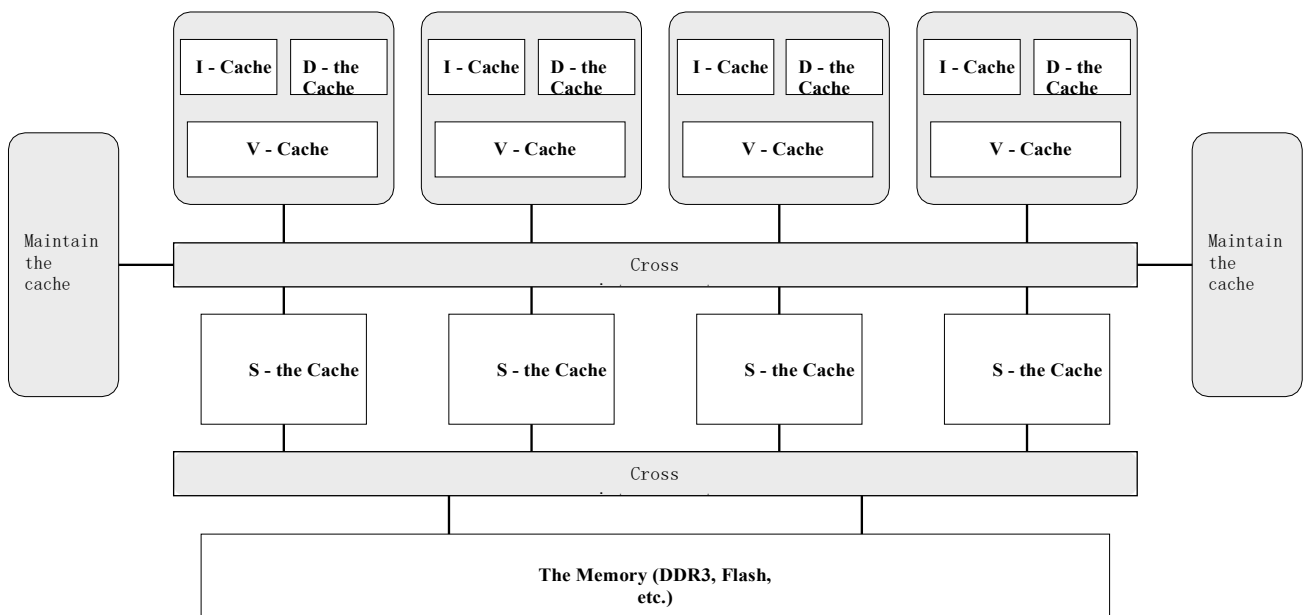
Under THE MIPS architecture, all levels of caches in the processor are visible to the core software. For certain operations of the cache (such as cache initialization, consistency maintenance, etc.), the software is required to participate in the management of the cache. This chapter introduces the cache organization and management of GS464E. In this manual, general concepts describing processor cache are not expanded, such as Index, Tag, Cacheline, group association, cache access, cache hit and miss, Virtual Index Physical Tag cache, cache replacement, etc. If you are not clear about the concepts, please refer to the book on computing Architecture, MIPS Architecture Perspective (2nd edition).

5.1 Processor storage hierarchy and cache hierarchy

5.1.1 Processor storage hierarchy

The loongson 3A3000 chip processor USES a storage level with three levels of cache, which is shown in Figure 5-1.

FIG. 5-1 Storage level of longson 3A3000 chip processor



According to the distance between each level of Cache and the processing pipeline of the processor, the sequence is Instruction-Cache (I -Cache) and Data-Cache (D -Cache) at the first level, Victim-Cache (V -Cache) at the second level, and Shared-Cache (S -Cache) at the third level. Where I-cache, D-Cache, and V-cache are private to each processor core, and S-Cache is Shared between multiple cores and I/O. The processor core accesses the S-Cache through the interconnections between and within the chip.

I-cache only holds what the processor accesses the part that the processor accesses. D-cache only holds what the processor accesses the part that the processor accesses.

Both V-cache and S-cache are hybrid caches that store both instructions and data.

The contents of i-cache and D-cache are exclusive, meaning that the contents of the same physical address are not stored in V-cache when stored in i-cache or D-cache. The contents of the i-cache, D-cache, and V-cache are inclusive of the contents of the S-cache. If the contents of the same physical address are contained in the I-cache, D-cache, or V-cache, a backup of the same physical address must be found in the S-cache. The above relationship between mutual exclusion and inclusion is described from the perspective that can be observed by the software, and does not indicate the location relationship of data in the real storage medium at any time.

Data consistency between I-cache, D-Cache, V-cache, and S-cache is maintained by the hardware during operation.

When a pointing widget is not found in either i-cache or d-cache, the V-cache is first looked up. If a V-cache hit is made, the Cache line hit in v-cache is filled into i-cache or D-cache, and the Cache line replaced from i-cache or D-cache (if it exists) is backfilled to the spot in V-Cache where the Cache line was fetched. If the V-cache does not hit, the s-cache is further looked up. When the response from S-Cache is returned, it is filled directly into i-cache or D-cache, and the cached row (if any) that was replaced from i-cache or D-cache is backfilled to the spot in V-Cache where the cached row was taken. After S-Cache receives a request from the processor core, the processing involved in maintaining Cache consistency is detailed in Section 5.3.

Table 5-1 lists some of the parameters for each cache.

Table 5-1 Cache parameters

	Instruction cache	Data cache	Sacrifice the cache	Shared cache
capacity	64 KB	64 KB	256 KB	2MB/ body (8MB in total)
Associative degree	4 road	4 road	16 road	16 road
Line Size	512 - bit	512 - bit	512 - bit	512 - bit
The Index (Index)	Virtual address [“]	Virtual address [“]	Virtual address [13:6]	11 bits in a physical address See section 5.1.5 on page 63
Label (Tag)	Physical address [47:12]	Physical address [47:12]	Physical address [47:12]	Physical address [47:16]
Replacement strategy	Random replacement algorithm	LRU replacement algorithm	LRU replacement algorithm	LRU replacement algorithm
Write policy		Write back, write assignment	Write back, write assignment	Write back, write assignment
Check way	parity	The SEC - DED ECC	The SEC - DED ECC	The SEC - DED ECC

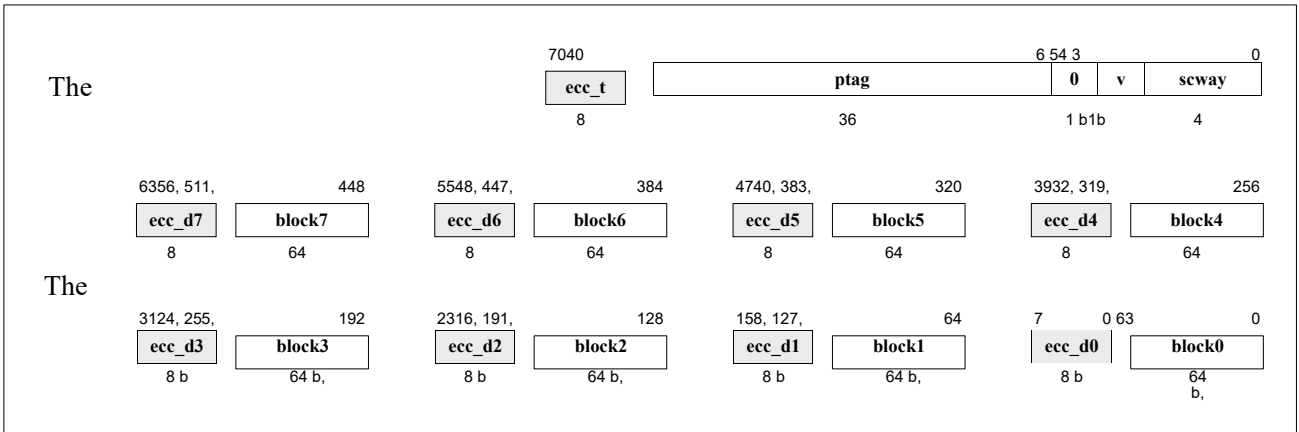
5.1.2 Level 1 instruction Cache (I-cache)

The capacity of first-level instruction cache is 64KB, and 4-way group linkage structure is adopted. The length of the data portion in each cached row is 64

A byte, divided into eight 8-byte wide blocks, is the smallest unit of data part access. The instruction cache USES a virtual address index to the physical address label

The access mode of the visa. During access, the [13:6] bit of the virtual address is used as the index of the cached row, the part below the 5th bit of the virtual address is used to Cache the inline index, and the part above the 14th bit of the virtual address is converted into a virtual and real address at the same time. The converted physical address high bit is compared with the content read out by tags in each channel to determine whether the Cache is hit. Figure 5-2 shows the structure of the instruction cache line. In addition to the high order (PTAG) of the physical address, the Tag contains the significant bit (V) and information about which way the Cache row is located in the S-cache (SCWAY). A significant bit of 1 means that the content on the cached row is meaningful, and no valid content on the cached row of table 0.

Figure 5-2 Shows the line structure of the level 1 instruction cache



The first-level instruction cache USES parity to check the Tag and Data parts in the cache line. When a new cache line is updated into the instruction cache, the Data section takes blocks as the basic unit of validation. Each block generates 8-bit validation results and records them. After the Tag section extends its 0 to 64 bits, the same validation algorithm is used to generate 8-bit validation results and also records them. When the cache is read, the raw data and the reference checksum are

At the same time, read out and recalculate the check value against the original data. If it is inconsistent with the reference check value, it indicates a Cache error. The hardware will automatically invalidate the Cache line with the error in i-cache and record relevant location information, triggering exceptions. If the software has no special diagnostic needs, it can be returned directly from the exception handler, and when the processor resumes execution, it will retrieve the required Cache line contents from V-Cache, S-cache, or memory. It should be noted that when the software fills the instruction Cache with Store Tag and Store Data class Cache instructions, the parity value of the contents must be calculated at the same time, and the errctl.ECC field must be explicitly stored. When the

Parity generation algorithm:

```
Function Parity_Gen(Datain, parityout) endfunction
Parity_Gen 63..07..0
```

Parity detection algorithm:

```
The function Parity_Check (newparity, refparity, error) 7..07..0
Endfunction Parity_Check
```

hardware executes such Cache instructions, the reference checksum written in the instruction Cache comes from the errctl.ECC field rather than the hardware circuit's automatic checksum generation result. This mechanism is mainly used to complete some special diagnosis. The pseudo-code of the algorithm for generating and detecting parity values of instruction cache is described as follows:

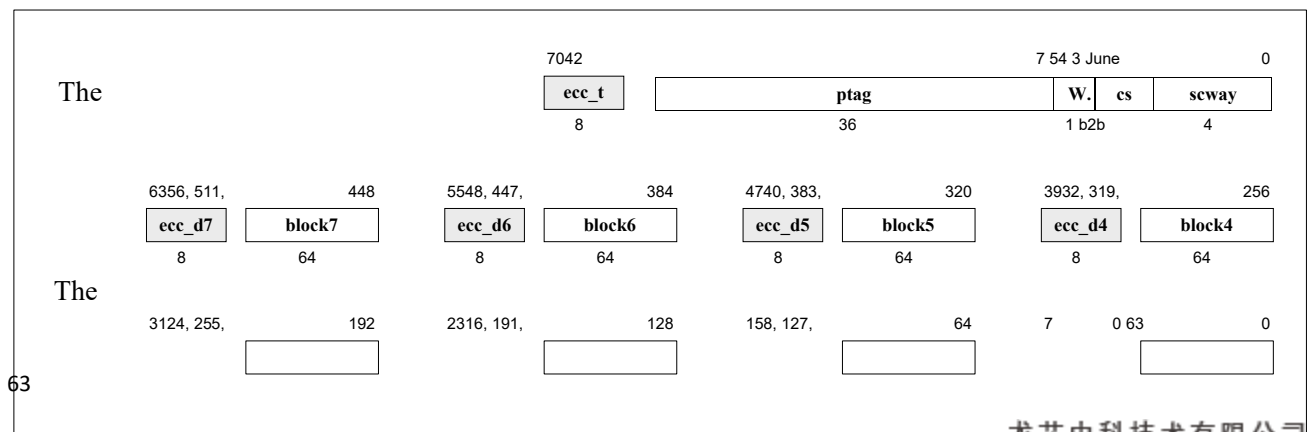
5.1.3 Level 1 data Cache (D-Cache)

The capacity of the first-level data cache is 64KB, and the 4-way group linkage structure and LRU replacement algorithm are adopted. The length of the data section in each cached row is

64 bytes, divided into eight 8-byte wide blocks, blocks are the smallest unit of data part access. The data cache USES a virtual address index physically

The access mode of the address label. During access, the [13:6] bit of the virtual address is used as the index of the cached row, the part below the 5th bit of the virtual address is used to Cache the inline index, and the part above the 14th bit of the virtual address is converted into a virtual and real address at the same time. The converted physical address high bit is compared with the content read out by tags in each channel to determine whether the Cache is hit. Figure 5-3 shows the structure of the data cache rows. In addition to the high (PTAG) position of the physical address, the Tag includes cached row state information (CS), dirty Tag bit (W), and the location of the cached row in the S-cache (SCWAY). Cs =0 means the cache row is invalid; Cs =1 means that the cached rows are in a Shared state; Cs =2 indicates that the cached row is in an exclusive state; Cs =3 is the reserved value. W =1 indicates that there is recently written data on the cache row.

Figure 5-3 Shows the row structure of the level 1 data cache





The first level Data cache USES the "Sec-ded" ECC check to check the Tag and Data parts in the cached row. When a new cache row is updated into the Data cache, the Data section takes blocks (blocks) as the basic unit of verification, and each block generates 8-bit verification results and records them. For Tag, after 0 except dirty Tag bit (W) is extended to 64 bits, the same verification algorithm is used to generate 8-bit verification results and also records them. It should be noted that the dirty Tag bits in the Tag part do not participate in the validation because this part of the information is not stored in SRAM, unlike the physical media stored in other parts of the Tag. When reading the Cache, 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 a bit error is found, the hardware will automatically correct the error, fill the corrected value back into the D-cache, and record relevant location information, triggering an exception. If the error number exceeds one bit, the hardware cannot correct it and can only record it

Related location information, triggering exceptions. When the exception is a bit wrong, the software can return directly from the exception handler if there is no special diagnostic need. When more than one error occurs, a more thorough recovery, such as a soft reset, is usually required. It should be noted that when the software fills the Data Cache with Store Tag and Store Data class Cache instructions, it must simultaneously calculate the ECC check value of the filled content and explicitly Store the ERRctL. ECC field. When the hardware executes such Cache instructions, the reference validation value written in the data Cache comes from the errctl.ECC field rather than the

ECC check value

generation algorithm:

```
Function ECC_Gen();
Endfunction ECC_Gen
```

ECC check and detection algorithm:

```
The function ECC_Check ();
```

hardware circuit's automatic validation result. This mechanism is mainly used to complete some special diagnosis. The pseudo-code of the algorithm for generating and detecting ECC check value of data cache is described as follows:

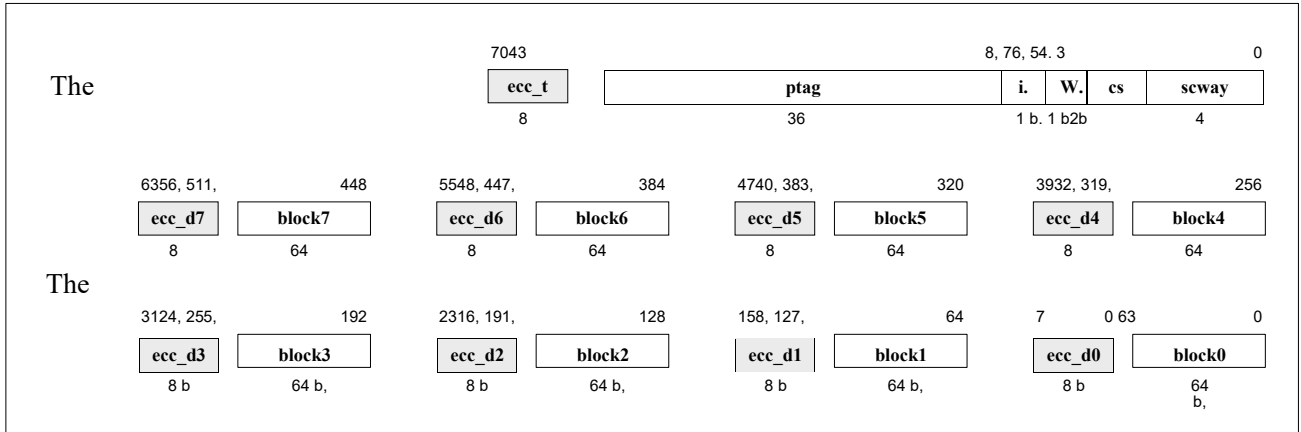
5.1.4 Level 2 Sacrifice Cache (V-Cache)

The capacity of the second-level sacrificed cache is 256KB, which adopts the 16-way group associative structure and LRU replacement algorithm. Sacrificing the cache is the access mode of virtual address index physical address label. The data portion of each cached row is 64 bytes long, divided into eight 8-byte wide blocks. The usual access is always to read or write all the Data parts in the Cache line, only when the Load Data and Store Data class Cache instructions are executed with the adjacent parity block as the basic unit, where the [5:4] of the physical address is used to indicate which pair of adjacent parity blocks are operated on.

When accessing the second-level sacrifice Cache, the [13:6] bit of the physical address is used as the index of the cached row, and the physical address high position is compared with the contents read by tags in each row to determine whether the Cache is hit or not. If it hits, it reads out the data content of the corresponding cache block in the hit path. Figure 5-4 shows the structure of the cached row. In addition to the high (PTAG) position of the physical address, the Tag includes cached line state information (CS), dirty Tag bit (W), instruction Tag (I), and

information about which way the cached line is located in the S-cache (SCWAY). Cs =0 means the cache row is invalid; Cs =1 means that the cached rows are in a Shared state; Cs =2 indicates that the cached row is in an exclusive state; Cs =3 is the reserved value. W =1 indicates that there is recently written data on the cache row. I=1 means the cache line stores instruction, I=0 means the cache line stores data.

Figure 5-4 shows the level 2 sacrifice cache row structure



At the second level, the "Sec-ded" ECC check is used to check the Tag and Data sections in the cached row. When a new cache row is updated into the sacrifice cache, the Data section takes blocks as the basic unit of verification, and each block generates 8-bit verification results and records them. After part 0 of Tag is extended to 64 bits, the same verification algorithm is used to generate 8-bit verification results, which are also recorded. When reading the Cache, 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 a bit error is found, the hardware will automatically correct the error, fill the corrected value back into the V-cache, and record the relevant location information, triggering an exception. If the error

If the digit exceeds one bit, the hardware cannot correct it. It should be noted that when the software fills the Data Cache with Store Tag and Store Data class Cache instructions, it must simultaneously calculate the ECC check value of the filled content and explicitly Store the ERRctL. ECC field. When the hardware executes such Cache instructions, the reference checksum written in the Cache is sacrificed from the errctl.ECC field instead of the hardware circuit's automatic checksum generation result. This mechanism is mainly used to complete some special diagnosis. The algorithm for generating and detecting ECC check values at the expense of the cache is consistent with the data cache, see section 5.1.3 on page 61.

5.1.5 Level 3 Shared Cache (S-Cache)

The three-level Shared cache supports cache consistency based on the directory protocol. Simongson 3A3000 chip s-Cache addresses all the chips uniformly, and each Shared Cache row has a fixed home node.

Split structure of Shared cache

The S-Cache of The Loongson 3A3000 chip is divided into four units (Banks) and receives access requests from the processor core and I/O ports to maintain Cache consistency through the first-level cross-switch interconnection network. Since longson 3A3000 chip adopts the address window mapping mechanism that can be dynamically adjusted by software on the first-level cross-switch interconnection network, the physical address seen by each S-cache body is the address remapped through the address window. Please make this clear when the software operates S-Cache. Which of the four S-cache-bodies the different requests end up in is determined by the two bits in the address that are dynamically adjusted by the software, as determined by the SCID_SEL of the chip configuration register. The correspondence between the configuration information and the address bit of the s-cache body is given. Accordingly, which bits of the physical address are used to cache the row index changes as the SCID_SEL value changes.

Table 5-2 Three levels of Shared cache body selection bit and index address

SCID_SEL value	Select a body	The index address
b0000	PAddr [but]	PAddr [he]
b0001	PAddr [and]	{PAddr [and], PAddr [but]}
b0010	PAddr [10]	{PAddr [went], PAddr [or]}
b0011	PAddr [12]	{PAddr [thou doest, PAddr [but]}
b0100	PAddr [also]	{PAddr [before], PAddr [he]}
B0101 ~ b1111	PAddr [that is]	PAddr [and]

Locking mechanism for Shared cache

The capacity of the Shared cache unit is 2MB, and the 16-way group linkage structure is adopted. In addition to using the LRU algorithm to select replacements, the Shared cache also supports cache locking mechanisms. There are two ways to lock a Cache: one is to lock a Cache line using the Cache15 instruction; The other is to use the Shared cache lock window mechanism in the chip configuration register to lock the physical address space. Once the locked content is stored in the Shared Cache, it will not be replaced again, unless the following two situations occur :(1) all Cache lines in 16-way s-cache that are locked with the locked Cache line and Index are in the "locked" state, then all Cache line locks are deemed invalid, and the replacement item is still selected according to LRU algorithm; (2) Invalid "locked" Cache line using the software Cache instruction. The advantage of using the Cache15 instruction is that it can directly use the virtual address to lock the Cache operation, and if the data is not in s-cache, the Cache line to be locked will be retrieved to S-cache and then locked. The disadvantage is that both the Cache lock and release operations need to be carried out in each Cache line, which has certain overhead.

Using lock window mechanism is to configure the advantages of a (write three window lock configuration register) can lock in a large contiguous address space (not less than 15/16 of the S - Cache capacity in theory, namely 3.75 MB), the disadvantage is that configuration must have a physical address information, need special support, the operating system kernel and configuration does not ensure that the data must be in the S - after the Cache. Software personnel can select the appropriate S-Cache lock mechanism for program optimization according to the specific characteristics of the application. See section 2.4.9 on page 33 for a detailed definition of the Cache15 directive. For a detailed definition of the S-Cache lock window configuration register, see the description in section 2.5 of the Loongson 3A300/3B3000 Processor User manual volume I.

The cache line structure of the Shared cache

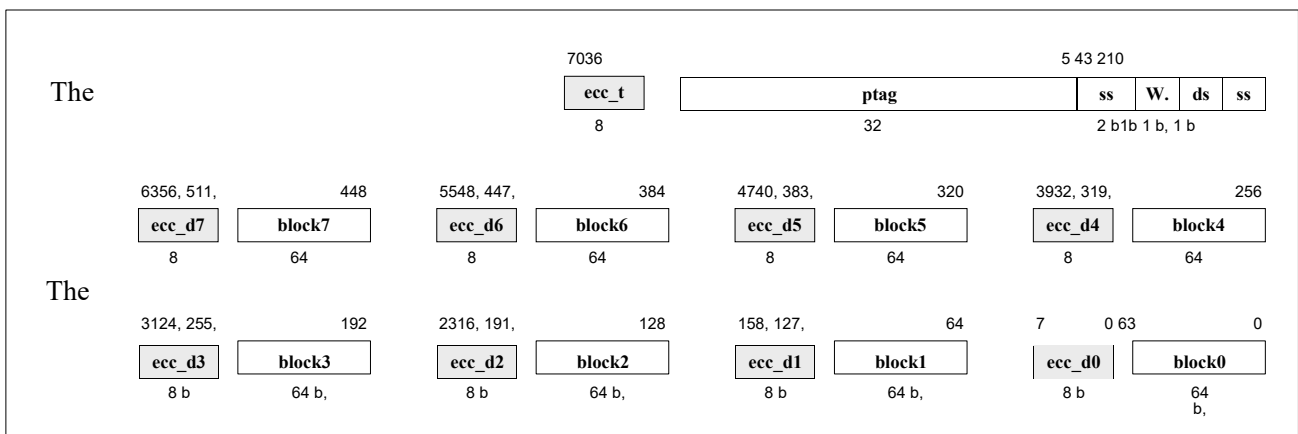
The Shared cache USES the physical address index physical address label access mode. The data portion of each cached row is 64 bytes long, divided into 8 bytes

Eight 8-byte wide blocks. The usual access is always to read or write all the Data parts in the Cache line, only when the Load Data and Store Data class Cache instructions are executed with the two adjacent parity units as the basic unit, at this time the physical address of the [5:4] is used to indicate which pair of adjacent parity blocks to operate.

During access, the physical address high position is compared with the contents read by tags in each path to determine whether the Cache is hit. If it hits, it reads out the data content of the corresponding cache block in the hit path. Figure 5-5 shows the structure of the Shared cache row. In addition to the high position (PTAG) of the physical address, the Tag also includes cache line status information (SS), directory status information (DS), dirty bit (W), 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 indicates that there is recently written data on the cache row. Indicates that there is recently written data on the cache row. Page coloring bits are used by hardware to handle cache aliases, see 69 for details

This is described in section 5.4.5.

Figure 5-5 shows the three-level Shared cache row structure



Shared cache validation

The third-level Shared cache USES the "Sec-ded" ECC check to check the Tag and Data sections in the cached row. When a new cache row is updated into the Shared cache, the Data section takes blocks (blocks) as the basic unit of verification, and each block generates 8-bit verification results and records them. After part 0 of Tag is extended to 64 bits, the same verification algorithm is used to generate 8-bit verification results, which are also recorded. When reading the Cache, 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 a bit error is found, the hardware will automatically correct the error, fill the corrected value back into S-cache, and record the relevant location information, triggering an exception. If the error number exceeds one bit, the hardware will not be able to correct it. It should be noted that when the software fills the Data Cache with Store Tag and Store Data class Cache instructions, it must simultaneously calculate the ECC check value of the filled content and explicitly Store the ERRctL.ECC field. When the hardware executes such Cache instructions, the reference checksum written in the Cache is sacrificed from the errctl.ECC field instead of the hardware circuit's automatic checksum generation result. This mechanism is mainly used to complete some special diagnosis. The algorithm for generating and detecting ECC check values at the expense of the cache is consistent with the data cache, see section 5.1.3 on page 61.

5.2 The cache algorithm has the same properties as the cache

GS464E supports three caching algorithms and cache consistency properties: Uncached, Cacheable cache, and Uncached Accelerated. The consistent algorithm corresponding to the non-cache algorithm is coded as 0b010, the consistent algorithm corresponding to the consistent cache algorithm is coded as 0b011, and the consistent algorithm corresponding to the non-cache acceleration algorithm is coded as 0b111.

5.2.1 Non-cache algorithm

When an address segment or page adopts a non-cache algorithm, the retrieval or memory operation of virtual address on the address segment or page will be directly initiated by the processor to the location of the target address, and the data read or written will not originate or terminate at any level of cache.

All access requests using a non-cached algorithm are executed in a blocking order. That is, before the current read request data is returned to the processor, all subsequent requests are blocked and issued; All subsequent requests are blocked until the write request data has been sent or the issued write request has not received a write reply from the final receiver.

5.2.2 Consistent caching algorithm

When an address segment or page USES a non-cached algorithm, the content accessed by the pointer or fetch operation that falls on the address segment or page can reside in any level of cache. GS464E is maintained by hardware and does not require software to maintain Cache consistency by using invalid Cache instructions and writing back the contents of the Cache.

5.2.3 Non-cache acceleration algorithm

The non-cache acceleration algorithm attribute is used to optimize a sequence of Uncached number operations of the same type completed in a contiguous address space. The optimization method is to collect the memory operation of this algorithm property by setting buffer. Data from these number operations can be stored in the buffer as long as the buffer is not full. The buffer size is the same as a Cache line and is 64 bytes. A save operation completes when the data is stored in a buffer. When the buffer data collection is full, it is written out consecutively at once. Data written out in succession will be written directly to the destination address and will not stay in any level of cache. During the data collection of sequential memory instruction, if a normal type of non-cached memory instruction is inserted, the collection is aborted and the saved data in the buffer is output as byte writes. The operation effect of the finger or number operation of the non-cache acceleration algorithm attribute is the same as

that of the normal non-cache algorithm attribute.

The non-cache acceleration attribute speeds up sequential Uncached access and is suitable for quick output access to display device storage.

5.3 Cache consistency

GS464E implements the directory based Cache consistency protocol, which ensures the consistency of data among I-cache, D-cache, V-cache, S-cache, memory and IO devices from HT by hardware, and does not require software to use Cache instructions to maintain Cache consistency.

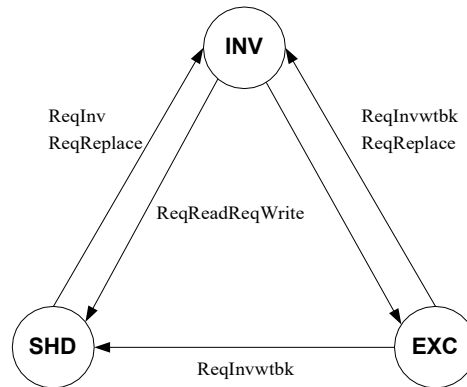
Each Cache line in GS464E has a fixed host S-cache body. The directory information for the Cache rows is maintained in the host S-cache body. The directory USES a 64-bit bit vector to record the first level of caches (including i-cache and D-cache) that have a backup of each Cache row. Each level Cache block has three possible states: INV (invalid state), SHD (Shared state, readable), and EXC (exclusive state, readable, writable). The transitions between the three states are shown in Figure 5-6.

When the read instruction or fetch operation fails in the first and second level caches, the processor core issues a Reqread request to the third level S-Cache. After receiving the Reprread reply returned by S-Cache, the processor core's first level Cache obtains a copy of the Cache line in the state of SHD. When the Cache operation fails in both the first and second level caches, the processor core issues a Reqwrite request to the third level S-Cache. After receiving a Repwrite reply returned by S-cache, the processor core's first level Cache obtains a backup of the Cache line in the state of EXC.

When a V-cache replacement occurs to the processor core, Reqreplace writes back to the S-Cache module, and s-Cache replies via Repreplace to inform the processor that the core replacement request has been processed.

S-cache can be backed up with an invalid SHD Cache line in i-cache, D-cache, or V-cache by sending a Reqinv request to the processor core. The processor core changes the corresponding Cache line to INV and replies to s-cache with Repinv. S-cache can write back an EXC backup of the Cache line by sending a Reqwtbk request to the processor core, which returns the corresponding Cache line backup to the SHD state and replies to s-cache via Repwtbk. S-cache can write back to the processor core by sending a Reqinvwtbk request that invalidates an EXC backup of the Cache line. The processor core returns the backup of the corresponding Cache line to an INV state and replies to a second-level Cache module via Repinvwtbk.

Figure 5-6 Cache state transitions under conformance protocol



5.4 Cache management

5.4.1 CACHE directives

This processor implements CACHE instructions for I- CACHE, D- CACHE, V- CACHE, and S- CACHE. The format of the CACHE instruction is CACHE OP, offset(Base). There are some differences between the GS464E Cache directive and the MIPS64 specification, as detailed in section 2.4.9 on page 33. Again, the MIPS specification does not apply to GS464E if the object being manipulated is level 2 or level 3 Cache based on op[1:0]=2 or =3. To reserve the software as forward-compatible as possible, OP [1:0]=2 indicates that the operation object is S-Cache(level 3), and OP [1:0]=3 indicates that the operation object is V-Cache(level 2). In this code form, software code that USES S-cache brush to maintain Cache consistency on the 3A1000 chip can still achieve the same Cache consistency on the 3A3000 chip. In other words, when the software is invalid in an S-cache line, the hardware guarantees that the same physical address in the Cache of all processor cores will be invalid at the same time. When the software is invalid and written back to an S-cache line, the hardware ensures that the same physical address is invalidated simultaneously in the caches of all processor cores, and that the contents written back to main memory must contain the most recently written data.

CACHE instruction in root mode

Kernel mentality software running in root mode can use all implemented CACHE instructions, the list is as follows:

Table 5-3 CACHE instructions in root mode

Op [4-0]	Function al description	Objective the Cache
b00000	Invalid Cache line based on index	I - Cache
b01000	Writes the Cache line Tag against the index	I - Cache
b11100	Writes the Cache row Data against the index	I - Cache
b00001	Invalid index and writes back to Cache line	D - the Cache
b00101	Read the Cache line Tag against the index	D - the Cache
b01001	Writes the Cache line Tag against the index	D - the Cache
b10001	Invalid Cache line based on hit	D - the Cache
b10101	Invalid hit and writes back to Cache line	D - the Cache

b11001	Read the Cache row Data against the index	D - the Cache
b11101	Writes the Cache row Data against the index	D - the Cache
b00010	Invalid index and writes back to Cache line	V - Cache
b00011	Invalid index and writes back to Cache line	S - the Cache
b00111	Read the Cache line Tag against the index	S - the Cache
b01111	Retrieve and latch the Cache line based on the address	S - the Cache

Op [4-0]	Function al description	Objective the Cache
b01011	Writes the Cache line Tag against the index	S - the Cache
b10011	Invalid hit and writes back to Cache line	S - the Cache
b11011	Read the Cache row Data against the index	S - the Cache
b11111	Writes the Cache row Data against the index	S - the Cache

The CACHE2 instruction is mainly used to clear the V-cache of a single core when it is only closed. The hardware only guarantees the correctness of the result when using CACHE2 in the following way.

First, when using the CACHE2 instruction to clear the V-cache, make sure that all code executed during this process is in the uncache space. Second, when using the CACHE2 instruction to clear the V-cache, do not perform any other load or store operations in the Cache space.

Use of CACHE instructions in guest mode

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

- When guestCTL0.cg =0, the use of any CACHE instruction will trigger the guest mode privileged sensitive instruction exception (GPSI).
- When guestCTL0.cg =1, CACHE instruction with OP [4:2]=1, 2, 6, 7 will trigger the guest mode privileged sensitive instruction exception (GPSI).
- When guestCTL0.cg =1, but Dig.GCAc =0, CACHE0, CACHE1, CACHE3 will trigger the guest mode privileged sensitive instruction exception (GPSI).

5.4.2 Cache initialization

Hardware-based cache initialization

During a hard restart, the hardware sets all the cached Tag parts to all zeros, making all cached Cache lines invalid. Therefore, the software does not need to initialize all caches except for the following special use case.

A special case that makes hardware initialization Cache unsafe is when, after a hard restart of the processor, the software USES the Index Store Tag instruction to make a Cache line valid, but does not use the Index Store Data instruction to make all the blocks of the Data section of that Cache line certain.

If the above sequence of actions does exist in the software, be sure to initialize the cache in a software-like manner before performing an action that poses a security risk.

Software-based cache initialization

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

Step 1: Create a chunk of memory to fill with arbitrary data. This data is then used to populate the cached data portion to form the correct checksum. It is recommended to use a memory area starting at address 0.

Step 2: Mask interrupts to prevent unexpected

situations during initialization. Step 3:

Initialize i-Cache.

```
Mtc0 zero, TagLo;    Mtc0 zero, TagHi;
```

```
Mtc0 zero, ErrCtl;    The parity of /* 64-bit full 0 is 0x0*/
```

```

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 the D-cache.

```

Mtc0 zero, TagLo; Mtc0 zero, TagHi;
Addiu r1, r1, 0 x22; Mtc0 r1, ErrCtl; The ECC check value
for /* 64-bit full 0 is 0x22*/ for (addr= 0xffffffff80000000;
Addr < 0xffffffff80010000; Addr + = 64) {
*/ 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, 0 x22; Mtc0 r1, ErrCtl; The ECC check value
for /* 64-bit full 0 is 0x22*/ for (addr= 0xffffffff80000000;
Addr < 0xffffffff80010000; Addr + = 64) {
*/ 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; /* Use uncached
Write */ MTC0 Zero, TagLo; Mtc0 zero, TagHi;
Addiu r1, r1, 0 x22; Mtc0 r1, ErrCtl; The ECC check value for /* 64-bit full 0 is
0x22*/
Processor no. 0 initializes s-Cache Bank no. 1, and so on */ Bank =
get_my_CPUNum();

```

```
For (addr = 0xffffffff80000000; Addr < 0xffffffff80040000; Addr += 256) {  
    /* for (way=0; Way < 16;  
       Way += 1) {
```

The chip configuration register SCID_SEL is equal to the default value of 0.

```
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; /* Use uncached write*/
```

Step 7: Poll the Scache_init_ok vector until all items are 1. /* Polling should use uncached Read */

At this point, the cache is initialized.

5.4.3 Maintain consistency between level 1 instruction cache and level 1 data cache

For applications with "self-modifying code," there is a problem of data consistency between the first-level instruction cache and the first-level data cache. GS464E is maintained by the hardware for data consistency between the first-level instruction CACHE and the first-level data CACHE without the need for software to use the CACHE instruction or SYNCI instruction to maintain data consistency by brushing back and clearing the D-cache and i-cache.

It should be noted that GS464E implements a weakly consistent storage model. So after the software has modified the code, it must jump to the modified code using the JR. Hb or JALr. hB instructions. In addition to the jump, JR. Hb and jalr. HB act as barriers. This ensures that the PC at the target of the JR. Hb or JALr. HB jump will see the modified front gate write operation.

5.4.4 Maintain cache consistency between processor and DMA device

In order to improve the performance of the processor, the driver software of DMA devices will put the data requiring a large amount of interaction in the cache space, thus causing the cache consistency maintenance problem between the processor and DMA devices. This issue is addressed in detail in many driver development books and is not covered in this manual. In the Loongson 3A3000 chip, the hardware can maintain cache consistency between the processor and DMA devices connected to the HT port. Therefore, when a DMA device in the system accesses system main memory through HT port, the driver software of the device does not need to use Cache instruction to maintain data consistency by swiping S-cache. Doing so can improve processor performance.

It should be noted that GS464E implements a weakly consistent storage model. Therefore, between the processor and DMA equipment, the synchronization effect of the gate barrier still needs to be achieved through the semaphore stored in the uncached area or the interrupt mechanism, so as to ensure that consumers can actually observe the data the producer wants them to observe when they read the data.

5.4.5 Cache alias and page coloring

Because GS464E's first-level instruction cache and first-level data cache use the access mode of virtual index physical labels, and each cache path is 16KB in size, there is a cache alias problem when the page size is 4KB or

8KB. The popular solution to cache aliases is a "page coloring" mechanism, which guarantees that at any given time, a physical address has at most one "page color" of a virtual address. GS464E implements the "page coloring" mechanism through hardware and is no longer implemented by software.

In the vast majority of cases, the software can ignore cache aliases because the hardware is guaranteed by "page coloring" that cache aliases do not occur. The only exception is when the software USES the Index Store Tag and Index Store Data class Cache directives to create valid Cache rows directly from each level of the Cache and USES the Data from those Cache rows with subsequent programs. At this point, the software must ensure that the contents of the page color field (PGC) in the S-cache line Tag match the virtual address of the Cache line data. Specifically, the PGC field in the Tag of S-Cache must always be equal to the [13:12] bit of the virtual address of the Cache line.

6 Processor exceptions and interrupts

6.1 Processor exception

6.1.1 Exception priority

When an instruction meets more than one exception trigger condition at the same time, GS464E will trigger the exception with higher priority according to the example in Table 6-1.

Table 6-1 Exception priorities

exception	type
Cold reset	Asynchronous, reset classes
EJTAG performs the one-step exception	Synchronize and debug classes
EJTAG debug interrupt exception	Asynchronous, debug class
Non-masking interrupt	asynchronous
EJTAG instruction breakpoint exception	Synchronize and debug classes
The wrong address exception - pointing	synchronous
TLB/XTLB refill exception - take finger	synchronous
TLB invalid exception - pointing	synchronous
TLB performs blocking exceptions	synchronous
Cache error exception - pointing	synchronous
EJTAG SDBBP exception	synchronous
No exceptions can be made to the coprocessor	synchronous
Reservation instruction exception	synchronous
interrupt	asynchronous
Integer overflow exception, trap exception, system call exception, breakpoint Exceptions, floating point exceptions, floating point stack exceptions	synchronous
Exception for EJTAG exact data breakpoint	Synchronize and debug classes
The wrong address exception - Data access	synchronous
TLB/XTLB refill exception - Data access	synchronous
Invalid exception for TLB - Data access	synchronous
TLB reads prevent exceptions	synchronous
TLB modification exceptions	synchronous
Cache error exception - Data access	synchronous

6.1.2 Exception entry vector position

Cold reset, soft reset, and non-masking interruptible exception entry vector addresses use the dedicated

address 0xffff.ffff.bfc0.0000, which is neither cache-accessible nor address-mapped.

Vector addresses of EJTAG debugging related exceptions are selected according to whether the ProbeTrap bit in the EJTAG control register is 0 or 1

0 XFFFF. FFFF. BFC0.0480 and 0 XFFFF. FFFF. FF20.0200.

All other exception vector addresses are defined as "base address + offset". When status.BEV=0, the base address of all exceptions adopts fixed configuration; When status.BEV=1, the software can configure the exception vector base address through the EBase register and the GSEBase register. Table 6-2 lists the exception vector base address definitions, and Table 6-3 lists the exception offset definitions.

Table 6-2 exception vector base addresses

exception	Status. BEV = 0	Status. BEV = 1
Cold reset, soft reset, non-masking interrupt	0 XFFFF. FFFF. BFC0.0000	
EJTAG debugging exception (ProbTrap=0)	0 XFFFF. FFFF. BFC0.0480	
EJTAG debugging exception (ProbTrap=1)	0 XFFFF. FFFF. FF20.0200	
Cache fault exception	EBase 1 EBase 0x000 63..3028..12	0 XFFFF. FFFF. BFC0.0200
Other Exceptions	EBase 0 x000 63..12	0 XFFFF. FFFF. BFC0.0200

Table 6-3 exception vector offset

exception	The vector offset
Cold reset, soft reset, non-masking interrupt	No offset, use base address directly
All kinds of 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)	0 x000
XTLB refill exception (status.exl =0)	0 x080
Cache fault exception	0 x100
Other Exceptions	0 x180
Interrupt (Cause. IV = 0)	0 x180
Interrupt (intCTL.vs =0 and cause.IV =1)	0 x200
Interruption (status.bev =1 and cause.iv =1)	0 x200
interrupt (caus.iv =1 and status.bev =0 and intctl.vs! = 0)	0X200 + (interrupt vector no. × (intctl.vs 0b00000))

6.1.3 The processor hardware responds to the exception's generic processing

When the processor starts processing an exception, the EXL bit of the status register is set to 1, meaning that the system is running in kernel mode. After saving the appropriate field state, the exception handler usually sets the KSU field of the state register to kernel mode, while returning the EXL position to 0. When the field state is restored and re-executed, the handler restores the KSU field to its previous value and sets the EXL bit to 1.

Returning from an exception also sets the EXL position to 0.

6.1.4 Cold reset exception

A cold reset exception occurs when the system is powered on or cold reset for the first time. This exception cannot be blocked.

The cold reset exception USES a special exception entry vector address. The address belongs to a CPU that does not require address mapping and does not access data through a Cache

Address space, so processing this exception does not have to initialize TLB or Cache. This also means that even if the Cache and TLB are in an uncertain state,

The processor can also fetch and execute instructions.

When a cold reset exception occurs, the processor goes through a full reset initialization process in which the contents of all registers in the CPU are uncertain, except for the following register fields:

- The Status register is set to the initial value, with SR bit 0, ERL bit and BEV bit 1.
- Initial value of Config0~Config6 register.
- The Random register is initialized to the maximum value, and the Wired register is initialized to 0.
- Initial values of related fields for EntryHi, EntryLo0, EntryLo1, PageMask, PageGrain.
- The ErroEPC register is initialized to the value of the PC.
- The Event bit of the Performance Count register is initialized to 0.
- All breakpoints and external interrupts are cleared.

6.1.5 Non-masking interrupt

Non-masking interrupts are triggered by a processor-independent NMI interrupt input signal. This exception cannot be blocked.

It is not possible to mask interrupts with exception entry vectors consistent with cold reset. Therefore, in the event of an unmasking interrupt exception, the status.NMI bit is set to 1 and the software can distinguish a cold reset by the bit.

The non-masking interrupt exception does not discard the state of any machine, but rather retains the state of the processor for diagnostic purposes. In particular, the Cause register contents remain unchanged, while the system jumps to an unmasked exception entry vector to start executing the handler.

The non-masking interrupt exception only modifies the following registers:

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

6.1.6 Interrupt exception

An interrupt exception is triggered when an unmasked interrupt arrives. For a detailed description of interrupts, refer to section 6.2 on page 80. Control register Cause's ExcCode field:

0x00 (Int) (see table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

register	Status update description
Cause	The IP domain records the interrupts to be processed.

6.1.7 Wrong address exception

The address error exception is triggered when:

- Double-word load/store, whose access address is not aligned at the double-word boundary. The access address of the word load/store is not aligned at the word boundary.
- Half-word load/store instruction, its access address is not aligned with the half-word boundary. Point PC not aligned at word boundary.
- Access the address segment of the core mode in customer mode or regulatory mode. Access the regulatory mode address segment in customer mode.

When 64-bit addressing enablement is not enabled, the access to the PC or Load/Store instruction USES a 64-bit address, and the address falls outside the 32-bit address space compatibility range.

Refers to the PC or load/store instruction access using a 64-bit address, and the address falls in the unimplemented range. In core mode, the page table entry accessed is valid and the K bit is 0.

This exception can be handled in both root and guest modes. Control register Cause's ExcCode field:

0x04 (AdEL) : Finger or read

data AdES (0x05) : write data

(See Table 7-28, page 111)

Additional hardware status updates in response to exceptions:

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

6.1.8 TLB rewrites the exception

In 32-bit host address space, and root.status.exl =0, access memory USES mapped address, which triggers TLB refill exception when no match is found in TLB. Note that this is different from the case where a match is found in TLB but the matching page table entry has a valid bit of 0, which corresponds to an invalid exception for TLB. To speed up the processing efficiency of frequent and critical exceptions such as TLB refill, the TLB refill exception USES a separate exception entry offset, so the exception coded in the root.cause.Exccode field is not distinguished from the XTLB refill exception and the TLB invalid exception.

This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x02 (TLBL) : To pick up or read data

0x03 (TLBS) : Write data

(See Table 7-28, page 111)

Additional hardware status updates in response to exceptions:

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

6.1.9 XTLB rewrites the exception

Under the 64-bit host address space, and `root.status.exl = 0`, access memory USES mapped address, which triggers XTLB refill exception when no match is found in TLB. Note that this is different from the case where a match is found in TLB but the matching page table entry has a valid bit of 0, which corresponds to an invalid exception for TLB. To speed up the processing efficiency of the frequent and critical exception XTLB refill, the XTLB refill exception USES a separate exception entry offset, so the exception encoded in the `root.cause.Exccode` field is not distinguished from the TLB refill exception and the TLB invalid exception.

This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x02 (TLBL) : To pick up or read data

0x03 (TLBS) : Write data

(See Table 7-28, page 111)

Additional hardware status updates in response to exceptions:

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

6.1.10 TLB is not an exception

The TLB invalid exception is triggered when:

The map address is used in the host address space. The match is found in TLB, but the valid bit of the match page table entry is 0. This exception is triggered.

When PageGrain. IEC = 0

Pagegr.rie =1, load operation USES mapped address in the host address space, matching and valid items are found in TLB, but the RI bit in the table entry is 1.

Pagegr.y.xie =1, refers to the use of mapping address in the host address space, and finds a matching and valid item in TLB, but the XI bit in the table entry is 1.

The software needs to pay attention to the following situation: when root.status. EXL=1, when the mapping address used for access cannot find a match in TLB, the exception entry offset adopted is the normal exception entry offset (0x180), and the exception encoding filled in the root.cause.ExcCode field is still TLBL (0x2) or TLBS (0x3). To distinguish this from a normal INVALID TLB exception, only the exception handler can use the TLBP instruction to distinguish based on the lookup results.

This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x02 (TLBL) : To pick up or read data

0x03 (TLBS) : Write data

(See Table 7-28, page 111)

Additional hardware status updates in response to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
The Context	The BadVPN2 domain record triggers the [31..13] bit of an exception's virtual address.

register	Status update description
XContext	BadVPN2 domain record triggers the virtual address of the exception [47..13] bit; The R field record triggers the [63..62] bit of an exception's virtual address.
EntryHi	VPN2 domain record triggers the virtual address of the exception [47..13] bit; The R field record triggers the [63..62] bit of an exception's virtual address. The ASID domain records the ASID of the process to which the exception is triggered.
Diag	The MID field is set to 0.

6.1.11 TLB modification exceptions

The store operation maps the address in the host address space, which finds a matching and valid entry in TLB, but the d-bit of the page table entry is 0 (meaning the page cannot be written), triggering the TLB modification exception.

This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x01 (Mod) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

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

6.1.12 TLB performs blocking exceptions

When root.pagegrb.IEC =0, and root.pagegrb.xie =1, take the mapping address under the host address space, find a matching and valid item in TLB, but the XI bit in the table entry is 1.

This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x14 (TLBXI) (see Table 7-28 on page 111)

Additional hardware status updates in

response to exceptions:

register	Status update description
BadVAddr	Record the virtual address that triggered the exception.
The Context	The BadVPN2 domain record triggers the [31..13] bit of an exception's virtual address.
XContext	BadVPN2 domain record triggers the virtual address of the exception [47..13] bit; The R field record triggers the [63..62] bit of an exception's virtual address.
EntryHi	VPN2 domain record triggers the virtual address of the exception [47..13] bit; The R field record triggers the [63..62] bit of an exception's virtual address. The ASID domain records the ASID of the process to which the exception is triggered.

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

6.1.13 TLB reads prevent exceptions

When root.pagegrb.IEC =0, and root.pagegrb.RIE =1, load operates under the host address space using the mapped address in TLB, but the RI bit in the table entry is 1. This exception is handled only in root mode.

Control register Cause's ExcCode field:

0x13 (TLBRI) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

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

6.1.14 Cache error exception

This exception is triggered when a check error is found in the Cache tag or data during the fetch or load/store operation. This exception cannot be blocked. Because the error involved in this exception is in the Cache, a special exception entry is used in the non-mapped non-cached address segment. This exception entry is described in section 6.1.2 on page 71.

This exception is handled only in root mode.

Control register Cause's ExcCode

field: none

Hardware status update

process in response to

exceptions: CacheErr

ErrorState Status.ERL 1

If InstructionInBranchDelaySlot then

```
ErrorEPC □ PC of the branch/jump
The else
    ErrorEPC □ PC of the instruction
endif
If the Status.BEV = 1 then
    PC □ xFFFF.FFFF.BFC0.0200 + 0 0 x100
The else
    1 || PC □ 0 xFFFF.FFFF || EBase || EBase || 0 x100 endif
31..3028..12
```

6.1.15 Exception for integer overflow

Integer overflow is an exception when an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction executes, resulting in a complement overflow.

This exception can be handled in both root and guest modes. Control register Cause's ExcCode field: 0x0C (Ov) (see Table 7-28, page 111)

Additional hardware status updates in response to exceptions:

There is no

6.1.16 Trap exceptions

When the TGE, TGUE, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTU, TLTUI, TEQI, TNEI instructions are executed,

The trap exception is triggered when the conditional result is true.

This exception can be handled in both root and guest modes. Control register Cause's ExcCode field: 0x0d (Tr) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

There is no

6.1.17 System call exception

The system call exception is triggered when the SYSCALL directive is executed. This exception can be handled in both root and guest modes.

Control register Cause's ExcCode field:

0x08 (Sys) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

There is no

6.1.18 Breakpoint exception

The breakpoint exception is triggered when a BREAK instruction is executed. This exception can be handled in both root and guest modes.

Control register Cause's ExcCode field:

0x09 (Bp) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

There is no

6.1.19 Reservation instruction exception

An exception to the retain instruction is triggered when a GS464E instruction is not implemented. This exception can be handled in both root and guest modes.

Control register Cause's ExcCode field:

0x0a (RI) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

There is no

6.1.20 No exceptions can be made to the coprocessor

The triggering coprocessor is not an exception when:

- When not in debug mode or core mode and status.cu0 =0, execute COP0 class instruction (opcode=0b010000), CACHE class instruction (opcode=0b101111), LWPTE, LWDIR, LDPTE, LDDIR.
- When status.CU1=0, execute COP1 instruction (opcode=0b010001), COP1X instruction (opcode=0b010011), LWC1, SWC1, LDC1, SDC1, MOVF, MOVF, 64-bit multimedia instruction (Opcode =0b010010, func= 0B000000 ~0b000011, RS = 11,000 ~11111; Opcode = 0 b010010, func b001000 = 0 ~ 0 b001110, Rs = 11000 ~ 11101), gsLWLC1GsLWRC1, gsLDLC1, gsLDRCL, gsLWLEC1, gsLWGTC1, gsLDLEC1, gsLWXC1, gsLWXC1, gsLDXC1, gsSWLC1, gsSWRC1, gsSDLC1, gsSDRC1, gsSWLEC1, gsSWGTC1, gsSDLEC1, gsSDGTC1, gsSQC1, gsSWXC1, gsSWXC1.
- When status.CU2=0, COP2 instruction (opcode=0b010010), LWC2 instruction (opcode=0b110010), SWC2 instruction (opcode=0b111010), LDC2 instruction (Opcode =0b110110), SDC2 instruction (Opcode =0b111110) are executed. But does not include SETMEM, gsLBLE, gsLBGT, gsLHGT, gsLHGT, gsLDLE, gsLDGT, gsLQ, gsLBX, gsLHX, gsLWX, gsLDX, gsSBLE, gsSBGT, gsSHGT, gsSWGT, GSSSDLE, gsSQ, gsSBX, gsSHX, gsSWX, gsSDX, LWPTE, LWDIR, LDPTE, LDDIR 64 bit multimedia instruction (Opcode =0b010010, func= 0B000000 ~ 0B000011, RS = 11,000 ~11111; Opcode = 0 b010010, func b001000 = 0 ~ 0 b001110, Rs = 11,000 ~11101), gsLWLC1, gsLWRC1, gsLWRC1, gsLDLEC1, gsLDLEC1, gsLDGTC1, gsLQC1, gsLWXC1, gsLDXC1, gsSWLC1, gsSWRC1, gsSDLC1, gsSDRC1, gsSWLEC1, gsSWGTC1, gsSDLEC1, gsSWGTC1, gsSDLEC1, gsSDGTC1, gsSQC1, gsSWXC1, gsSWXC1.

Note: in Guest mode, when guest.status.cu1/2 =1 but root.status.cu1/2 =0, the trigger coprocessor can't use the exception to directly fall into Root mode.

Control register Cause's ExcCode field:

0x0b (CpU) (see Table 7-28 on page 111)

Additional hardware status updates in response to exceptions:

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

6.1.21 Floating-point exception

The floating-point coprocessor fires the floating-point exception. See section 2.2.4 on page 16 for a detailed introduction to the floating point exception. Some floating point exceptions can be shielded by configuring the Enable field of the FCSR register, see section 2.2.3 on page 13 for details.

Control register Cause's ExcCode field:

0x0F (FPE) (see Table 7-28 on page

111) Additional hardware status updates
in response to exceptions:

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

6.1.22 Floating point stack exception

The floating-point stack exception is triggered when the SETTAG instruction is executed if the contents of the source operand do not meet the specified criteria. This exception can be handled in both root and guest modes.

Control register Cause's ExcCode field:

0X10 (GSExc) (see Table 7-28 on page 111) GSExcCode field for control register

GSCause:

0x00 (IS) (see Table 7-43 on page 131)

6.2 interrupt

The interrupts described in this section cover hardware interrupts, software interrupts, timer interrupts, and performance counter overflow interrupts. Non-masking interrupts (NMI), although it contains the word interrupt in its name, is neither controlled nor affected by the interrupt system described in this section, so it is treated as a separate special exception -- the non-masking interrupt exception.

6.2.1 Requirements for interrupt response

The necessary conditions for a processor to respond to an interrupt are:

- Status.IE=1 indicates that global interrupt enabled is enabled.
- Debug.dm =0, indicating that you are not in Debug mode.
- Staus.ERL=0 and status. EXL=0 indicate that neither error nor exception is being processed.
- An interrupt is caused by an interrupt source that is not masked,

6.2.2 Interrupt mode

What is described in this section applies to both the root and the guest modes. When it comes to virtual machine environments in guest mode, the "hardware" in the concept of "hardware interrupts" described in this section does not necessarily mean physical hardware, but follows the naming conventions of the MIPS specification.

GS464E supports two interrupt modes: mode 1, compatible interrupt mode

In this mode, the processor supports 2 software interrupts (SW0~SW1), 6 hardware interrupts (HW0~HW5), 1 timer interrupt and 1

Total performance counter overflow interrupt. Where timer interrupts and performance counters interrupt multiplexing HW5 hardware interrupts.

The interrupt source of software interrupt is the two digits cause. IP[1:0], which can only be triggered by the software to write 1 of the bit cause. IP[1:0], and the software to write 0

Clean up.

The interrupt source of the timer interrupt is recorded in the cause.ti bit and is set to 1 by hardware when Count[31:0] equals Compare[31:0]. The software can indirectly clear the interrupts of the cause.ti bit record by writing the Compare register.

The interrupt source of the performance counter overflow interrupt is recorded in the cause. PCI bit. When the performance counter value overflow (the 47th bit of the counter is 1), the hardware sets 1. The software can indirectly clear the cause. PCI bit by writing 0 to the 47th bit of the relevant performance counter numerical register.

1

The actual effective number of performance counters in 1 GS464E is 48 bits.

The interrupt source of hardware interrupt comes from the processor, and the 6 interrupt input pins on the processor interface are sampled by the hardware beat by beat. The software needs to reverse traverse the interrupt routing path of the system to clear the interrupt state on the terminal device or routing path, so as to clear the processor's hardware interrupt.

In addition to global interrupt enablement, each interrupt source contains an interrupt mask bit. The generation relationships of each interrupt request are shown in Table 6-4.

Table 6-4 Generation of each interrupt request in compatible interrupt mode

Interrupt type	The 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 the IP5 & Status. IM5
	HW2	Cause. IP4 & Status. IM4
	HW1	Cause the IP3 & Status. IM3
	HW0	Cause. IP2 & Status. IM2
Software interrupt	SW1	Cause. IP1 & Status. IM1
	SW0	Cause. IP0 & Status. IM0

The same exception entry offset is used for all interrupts, whether the general exception entry offset (0x180) or the special exception entry offset (0x200) is cause.IV determined. See Table 6-3 on page 72 for details.

The interrupt exception handler queries cause.ip and status.im to determine the specific interrupt source. In the case of multiple effective interrupt sources, the priority of interrupt processing can be realized by adjusting the order of the query.

Mode two, vector interrupt mode

This mode specifies a unique exception entry vector for each interrupt based on a compatible interrupt mode (see Table 6-3 on page 72 for calculation) and defines a fixed priority relationship for all interrupts, as shown in Table 6-5.

Table 6-5 Priority relationship among interrupts in vector interrupt mode

priority	Interrupt type	The 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 the IP5 & Status. IM5	5

Lowest priority		HW2	Cause. IP4 & Status. IM4	4
		HW1	Cause the IP3 & Status. IM3	3
		HW0	Cause. IP2 & Status. IM2	2
	Software interrupt	SW1	Cause. IP1 & Status. IM1	1
		SW0	Cause. IP0 & Status. IM0	0

In vector interrupt mode, which hardware interrupt source the timer interrupts to reuse is defined by the INTCTL.IPTI domain, see Table 7-25 on page 109. In vector interrupt mode, which hardware interrupt source the performance counter overflows the interrupt multiplexes is defined by the INTCTL.ippCI field, see the table on page 109

7 to 25.

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

Which interrupt mode the processor currently adopts is determined by the status.bev, cause.IV, and INTctl.vs domains. The corresponding relationship is shown in Table 6-6

Shown below.

Table 6-6 Interrupt mode determination

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

Status. BEV	Cause. IV	IntCtl. VS.	Interrupt mode
1	The x1	x	Compatible interrupt mode
x	0	x	Compatible interrupt mode
x	x	= 0	Compatible interrupt mode
0	1	! = 0	Vector interrupt mode

6.2.3 Additional notes on interrupt handling

This manual only describes the structure and response mechanism of the interrupt system of GS464E. When designing the loongson 3A3000 chip interrupt system, the software staff should refer to chapter 6 and 7 of "Loongson 3A3000/3B3000 Processor Household Manual -- Volume I" at the same time.

The processor is only responsible for direct sampling and recording the external input high level hardware interrupt request. It is not responsible for level conversion, nor is it responsible for extending the pulse interrupt signal to the level signal. This work is done by the interrupt controller in the chip. See chapters 6 and 7 of the Loongson 3A3000/3B3000 Processor User Manual, Vol. 1.

Depending on the signal behavior of the external hardware interrupt input, the interrupt request bits internally connected directly to the external interrupt input (root.cause.ip [7:2] or gueve.cause.ip [7:2]) may change from 1 to 0 after the interrupt request is triggered and before the interrupt handler queries these request bits. Interrupt handlers need to be able to handle this situation. It is recommended to return directly without doing any processing. 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, the software usually needs to clear the interrupt state of a terminal device. From the command sequence issued by the processor to clear interrupts, to the device receiving the command to clear its interrupt state, to the level of the processor's interrupt input pin changing from 1 to 0

The process (interrupt input undo) may have an indefinite delay. If interrupt enablement is turned on prematurely, the same interrupt may be re-sampled, a phenomenon known as "spurious interrupt". Software needs to be able to handle this situation correctly. It is recommended that the software, after issuing a write command to clear the device's interrupt state, explicitly read the interrupt state tag of the device concerned until the read status tag has been cleared, and then fully enable the device's interrupt within the processor or on the interrupt routing path. 2 The Longson 3A3000 chip has guaranteed that the delay of each interrupt source signal passing through the interrupt routing path 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 "false interrupt" is still found during debugging, please ask the software personnel to further inquire the data manual and user manual of the device chip and bridge involved in the system.

1, x means it can be anything.

If the interrupt bit of the device is read-clear, the software needs to be careful when issuing query commands.

7 Coprocessor register 0

7.1 Root coprocessor 0 register overview

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

Table 7-1 List of coprocessor 0 registers

Reg.	Sel.	Register name	Function definition	The index
0	0	The Index	VTLB accesses the specified index register with FTLB	Page 85, section 7.2
1	0	The Random	VTLB with FTLB access to random index registers	Page 86, section 7.3
2	0	EntryLo0	VTLB and FTLB table entry low order content associated with even number of virtual pages	Page 87, section 7.4
3	0	EntryLo1	VTLB and FTLB table entry low order content related to odd number of virtual pages	Page 87, section 7.4
4	0	The Context	A pointer to an in-memory page table entry	Page 90, section 7.5
4	2	UserLocal	Store user information that allows user-mode software to read through RDHWR commands	Page 91, section 7.6
5	0	PageMask	VTLB page table size control	Page 92, section 7.7
5	1	PageGrain	1KB small pages and other page table property control	Page 93, section 7.8
5	5	PWBase	Page table base address register	Page 94, section 7.9
5	6	PWField	Configure the page table address index location for each level	Page 95, section 7.10
5	7	PWSize	Configure the page table pointer size for each level	Page 96, section 7.11
6	0	Wired	Control the number of fixed items in VTLB	Page 97, section 7.12
6	6	PWCtl	Control multilevel page table configuration	Page 98, section 7.13
7	0	HWRENa	RDHWR instruction access register enable control	Page 99, section 7.14
8	0	BadVAddr	Record the error address for the latest address-related exception	Page 100, section 7.15
9	0	The Count	Processor clock counter	Page 101, section 7.16
9	6	GSEBase	Loongson extension exception entry base address register	Page 102, section 7.17
9	7	PGD	Page table pointer register	Page 103, section 7.18
10	0	EntryHi	VTLB and FTLB table entries high content	Page 104, section 7.19
11	0	The Compare	Timer interrupt control	Page 106, section 7.20
12	0	The Status	Processor status and control registers	Page 107, section 7.21
12	1	IntCtl	Interrupts system state and control registers	Page 109, section 7.22
12	2	SRSCtl	Shadow register status and control register	Page 110, section

				7.23
13	0	Cause	Store last exception reason	Page 111, section 7.24
14	0	The EPC	Store the PC on which the last exception instruction occurred	Page 113, section 7.25
15	0	PRId	The processor ID	Page 114, section 7.26
15	1	EBase	Exception entry base address register	Page 115, section 7.27
16	0	The Config	Configuration register	Page 116, section 7.28
16	1	Config1	Configure register 1	Page 117, section 7.29
16	2	Config2	Configure register 2	Page 118, section 7.30

Reg.	Sel.	Register name	Function definition	The index
16	3	Config3	Configure Register 3	Page 119, section 7.31
16	4	Config4	Configure register 4	Page 121, section 7.32
16	5	Config5	Configure register 5	Page 123, section 7.33
16	6	GSConfig	Loong chip expansion configuration register	Page 124, section 7.34
17	0	LLAddr	Store the load-Link instruction access address	Page 127, section 7.35
20	0	XContext	Extended address mode next page table pointer	Page 128, section 7.36
22	0	Diag	The loong chip extended diagnostic control register	Page 129, section 7.37
22	1	GSCause	Store additional information about the last add-on exception	Page 131, section 7.38
23	0	The Debug	EJTAG Debug register	Page 133, section 7.40
24	0	DEPC	Holds the PC with the last EJTAG debug exception	Page 134, section 7.41
25	0 to 7	PerfCnt0 - PerfCnt7	Processor core internal performance counter access interface	Page 135, section 7.42
26	0	ErrCtl	Cache Parity/ECC Parity value register	Page 137, section 7.43
27	0	CacheErr	Cache Parity/ECC Parity status and control registers	Page 138, section 7.44
27	1	CacheErr1	Cache Parity/ECC Parity status with control register 1	Page 140, section 7.45
28	0	TagLo	The Cache Tag accesses the lower part of the interface	Page 141, section 7.46
28	1	DataLo	Cache Data access interface low part	Page 144, section 7.47
29	0	TagHi	Cache Tag access interface high part	Page 145, section 7.48
29	1	DataHi	Cache Data access interface high part	Page 146, section 7.49
30	0	ErrorEPC	The PC on which the last wrong instruction was stored	Page 147, section 7.50
31	0	DESAVE	EJTAG debug exception saves register	Page 148, section 7.51
31	2-7	KScratch1 - KScratch6	Core mentality accessible note registers 1~6	Page 149, section 7.52

7.2 Index Register (CP0 Register 0, Select 0)

The Index register is a 32-bit read-write register in which the Index information is used for TLBP, TLBR, TLBWI instruction access

TLB.

Figure 7-1 illustrates the format of the Index register; Table 7-2 describes the Index register fields.

Figure 7-1 Index register format

31	30	11	10	0
P	0	The Index		

Table 7-2 Description of Index register field

Domain name	position	Functional description	Read/write	Reset value
P	31	TLB queries failed flags. When the TLBP instruction fails to find a match in TLB, the position is 1; Otherwise set to 0.	R	0 x0
0	30.. 11	Read only is always 0.	0	0
The Index	10.. 0	TLB accesses the index. The field is configured by the software to instruct subsequent TLBR or TLBWI instructions to read or write the specified item of the TLB. When the TLBP instruction executes, if a match is found, the index value of the match is stored in the field; When no match is found, the contents of the Index field of the Index register can be any value. The Index value of 0.. 63 The number 0.. used to indicate VTLB. A total of 63. The Index value of 64.. 1087 is used to indicate FTLB. Where ((index-64) div 128) is used to indicate which path to visit FTLB, and which path to visit is determined by the value of ((index-64) mod 128). For example, when the Index value is 798, it indicates access to item ((798-64) mod 128 = 94) of the path to ((798-64) div 128 = 5).	R/W	0 x0

Programming tips:

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

7.3 Random Register (CP0 Register 1, Select 0)

The Random register is a read-only register that holds the index value of TLBWR instruction access to TLB. The index value stored in the Random register changes every clock cycle. The upper bound (including) of the value change is 63, and the lower bound (including) of the value change is the value set in the Wired register. The Random register will automatically Reset to the upper bound when the Reset exception occurs and the Wired register is written, i.e. 63.

When the value of the page size represented in the PageMask register is inconsistent with the page size configured in the FTLB, the TLBWR instruction will operate only on THE VTLB, with the value in the current Random register determining which entry is written to the VTLB.

The TLBWR instruction will operate only when the value of the page size represented in the PageMask register is the same as the page size configured in FTLB

FTLB, which routing processor writes to FTLB is internally randomly determined, not using the contents of the Random register. Figure 7-2 illustrates the format of the Random register; Table 7-3 describes each field of the Random register.

Figure 7-2 Random register format

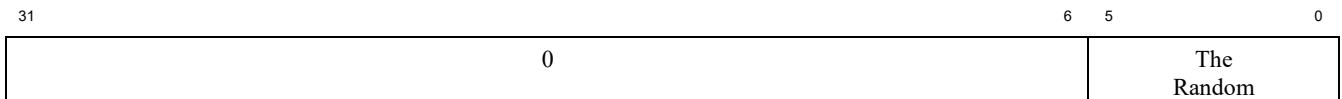


Table 7-3 Description of Random register field

Domain name	position	Functional description	Read/write	Reset value
0	31.. 6	Read only is always 0.	0	0
The Random	5.. 0	The random index value written by VTLB.	R	0 x3f

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

The EntryLo0 and EntryLo1 registers serve as interfaces for TLBP, TLBR, TLBWI, and TLBWR instructions to access the TLB, where EntryLo0 is used

The EntryLo1 register holds information for odd pages.

The EntryLo0 and EntryLo1 registers present a different format when accessed using DMFC0/DMTC0 and MFC0/MTC0 instructions.

Figure 7-3 illustrates the format of the EntryLo0 and EntryLo1 registers when accessed by DMFC0/DMTC0 instructions; Table 7-4 describes the register fields in this case.

Figure 7-3 Register formats for EntryLo0 and EntryLo1 when accessed by DMFC0/DMTC0 instructions

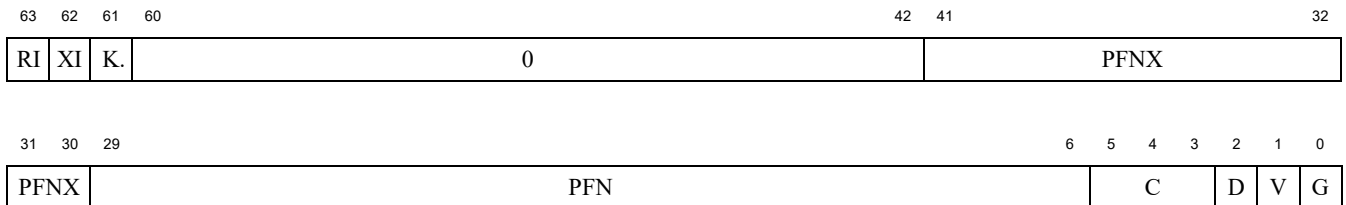


Table 7-4 Description of the register fields of EntryLo0 and EntryLo1 for DMFC0/DMTC0 instruction access

Domain name	position	Functional description	Read/write	Reset value
RI	63	Read prevents identifying bits. When the RI position of a TLB table entry is 1, the processor fires an exception when an access instruction attempts to read on the page. Depending on the PageGrain register IEC field, the triggered exception could be a TLBL invalid exception or a TLBRI exception. The XI fields of EntryLo0 and EntryLo1 can only be written if PageGrain=1. <small>RIE</small> PageGrain = 0 <small>RIE</small> , the RI fields of EntryLo0 and EntryLo1 will be read to 0 regardless of the value to be written.	R/W	0 x0
XI	62	Execution block identification bits. When the XI position of a TLB table entry is 1, the occurrence on the page is indicated and the processor fires the exception. Depending on the PageGrain register IEC field, the triggered exception could be a TLBL invalid exception or a TLBXI exception. The XI fields of EntryLo0 and EntryLo1 can only be written if PageGrain=1. <small>XIE</small> PageGrain = 0 <small>XIE</small> , the XI fields of EntryLo0 and EntryLo1 will be read to 0 regardless of the value to be written.	R/W	0 x0

K.	61	The kernel executes protection bits. If the processor is in kernel mentality, the processor will trigger the TLB Invalid exception by pointing to the page K=0. The K fields for EntryLo0 and EntryLo1 can only be written if gsconfig. KE=1. GSConfig. KE = 0 , the K fields for EntryLo0 and EntryLo1 will be read to 0 regardless of the value to be written.	R/W	0 x0
0	60.. 42	Read only is always 0.	0	0
PFNX	41.. 30	Physical page number extension. When the processor is configured to support the large physical address space pattern (Config3=1 and PageGrain=1), the domain contents are spliced to the high order of the PFN domain to form a complete physical page number, thus supporting the 48-bit physical address space. LPAELPA The PFNX domain corresponds to the 47.. of the physical address. 36. If the processor is configured to not support the large physical address space pattern (PageGrain=0), the PFNX field will ELPA ¹ Cannot write and reads return 0. Thus achieving compatibility with system software written based on Release 1 of the MIPS specification.	R/W	0 x0

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

Domain name	position	Function 1 description	Read/write	Reset value
PFN	29.. 6	Basic section of physical page number. When the processor is configured to support the large physical address space pattern (Config3=1 and PageGrain=1), the field contents are spliced into the lower part of the PFNX field to form a complete physical page number, thus supporting the 48-bit physical address space. LPAELPA The PFN domain corresponds to 35.. of the physical address. 12. When the processor is configured to not support a large physical address space pattern (PageGrain=0), the PFN field itself ELPA The final physical page number is formed to support the 36-bit physical address space.	R/W	0 x0
C	5.. 3	The Cache property of the physical page. See this section for a detailed definition of the Cache property and its encoding That's shown in table 7 minus 6.	R/W	0 x0
D	2	Dirty bits. When the dirty position in the page table is 1, the page can be written; Otherwise for a dirty location it is 0 The TLB Mod exception will be triggered if the page of.	R/W	0 x0
V	1	Significant bit. When the valid position in the page table is 1, it means that the page is accessible; Otherwise access a valid location A page of 0 would trigger the TLB Invalid exception.	R/W	0 x0
G	0	The global level. When a page table entry is filled into a TLB, the EntryLo0 and EntryLo1 register's two G-bit values are logically matched, resulting in the global identity bit for the page table entry. When the global identity bit in the page table is 1, asids are not compared in the TLB address match lookup. When page table entries are read from TLB, both G bits of the EntryLo0 and EntryLo1 registers reflect both reads G bit information for page table entries.	R/W	0 x0

Figure 7-4 illustrates the format of the EntryLo0 and EntryLo1 registers when accessed by the MFC0/MTC0 instructions; Table 7-5 describes the register fields in this case. Note that the KE bit of the page table cannot be accessed at this time. The KE bit of the TLB table entry will be written to the default value 0.

Figure 7-4 Register formats for EntryLo0 and EntryLo1 when accessed by MFC0/MTC0 instructions

63

32

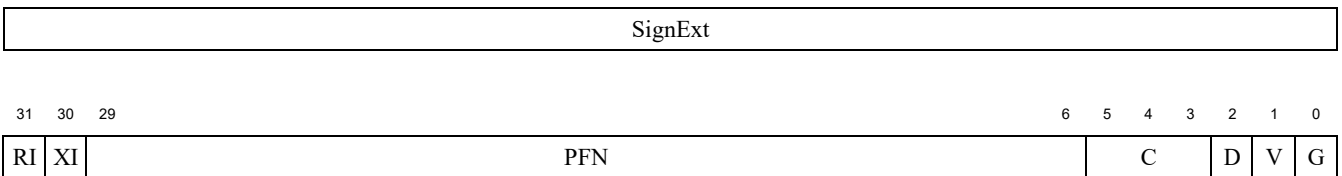


Table 7-5 Description of the register fields for EntryLo0 and EntryLo1 at MFC0/MTC0 instruction access

Domain name	position	Function 1 description	Read/write	Reset value

88

on				
SignExt	63.. 32	When MTC0 is used to write, the contents corresponding to this part in the register are ignored and the PFNX field is written 0. When MFC0 is read out, this part of the contents returned to the general purpose register is extended by the RI bit symbol.	R	0 x0
RI	31	Read prevents identifying bits. When the RI position of a TLB table entry is 1, the processor fires an exception when an access instruction attempts to read on the page. Depending on the PageGrain register IEC field, the triggered exception could be a TLBL invalid exception or a TLBRI exception. The XI fields of EntryLo0 and EntryLo1 can only be written if PageGrain=1. RIE PageGrain = 0 RIE , the RI fields of EntryLo0 and EntryLo1 will be read to 0 regardless of the value to be written.	R/W	0 x0

Domain name	position	Functional description	Read/write	Reset value
XI	30	Execution block identification bits. When the XI position of a TLB table entry is 1, the occurrence on the page is indicated and the processor fires the exception. Depending on the PageGrain register IEC field, the triggered exception could be a TLBL invalid exception or a TLBXI exception. The XI fields of EntryLo0 and EntryLo1 can only be written if PageGrain=1. ^{XIE} PageGrain = 0 ^{XIE} , the XI fields of EntryLo0 and EntryLo1 will be read to 0 regardless of the value to be written.	R/W	0 x0
PFN	29.. 6	Basic section of physical page number. When the processor is configured to support the large physical address space pattern (Config3=1 and PageGrain=1), the field contents are spliced into the lower part of the PFNX field to form a complete physical page number, thus supporting the 48-bit physical address space. ^{LPAELPA} The PFN domain corresponds to 35.. of the physical address. ¹² . When the processor is configured to not support a large physical address space pattern (PageGrain=0), the PFN field itself ^{ELPA} The final physical page number is formed to support the 36-bit physical address space.	R/W	0 x0
C	5.. 3	The Cache property of the physical page. See this section for a detailed definition of the Cache property and its encoding That's shown in table 7 minus 6.	R/W	0 x0
D	2	Dirty bits. When the dirty position in the page table is 1, the page can be written; Otherwise for a dirty location it is 0 The TLB Mod exception will be triggered if the page of.	R/W	0 x0
V	1	Significant bit. When the valid position in the page table is 1, it means that the page is accessible; Otherwise access a valid location A page of 0 would trigger the TLB Invalid exception.	R/W	0 x0
G	0	The global level. When a page table entry is filled into a TLB, the EntryLo0 and EntryLo1 register's two G-bit values are logically matched, resulting in the global identity bit for the page table entry. When the global identity bit in the page table is 1, asids are not compared in the TLB address match lookup. When page table entries are read from TLB, both G bits of the EntryLo0 and EntryLo1 registers reflect both reads G bit information for page table entries.	R/W	0 x0

Programming tips:

Before any field contents of the PageGrain register are modified, the PFNX and PFN fields in the EntryLo0 and EntryLo1 registers must be written to 0, and all TLBS must be cleared. All the above operations must be done in the Unmapped Address Space. If you do not follow the instructions here, the processor behavior is uncertain.

Table 7-6 Cache attribute encoding table

Page table C domain encoding	Cache attribute ¹
0	Reserve, forced configuration will cause a crash
1	Reserve, forced configuration will cause a crash

2	Uncached
3	Cached
4	Reserve, forcing the configuration is equivalent to Cached
5	Reserve, forcing the 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 definitions of the Uncached, Cacheable, and Uncached Accelerated properties, see section 5.2 on page 60.

7.5 Context Register (CP0 Register 4, Select 0)

The Context register is a read-write register that contains some page-table base address high level information filled in by the operating system software and some bits of the error virtual address with the TLB exception. According to the original design intent of the MIPS architecture, the information splited together in the Context register can form a pointer to an item in the page table, which can be accessed when a TLB exception occurs. The page table that can be accessed without any processing of the contents in the Context register is a single-level page table structure. The page size is 4K bytes, and each page table item is 16 bytes. It contains an even page table item and an odd page table item with consecutive virtual address, totaling 512K odd and even page table items. When the page table does not adopt this structure, the software needs to properly shift and concatenate the contents of the Context register. For an operating system with multilevel page tables, the Context register can only be used to speed up address generation for the last level of page table access.

The Context register is primarily used in the TLB Refill exception handler. But when exceptions such as XTLB Refill, TLB Invalid, and TLB Mod occur, the BadVPN2 domain in the Context register is also updated, so the software can also use the Context register in the corresponding exception handler.

The BadVPN2 domain in the Context register copies some of the information in the BadVAddr register, but this does not mean that this part is completely equivalent. When the Address Error exception occurs, the BadVAddr register is updated by the hardware, but the BadVPN2 domain of the Context register is not updated by the hardware.

Figure 7-5 illustrates the format of the Context register; Table 7-7 describes the fields of the Context register.

Figure 7-5 Context register format

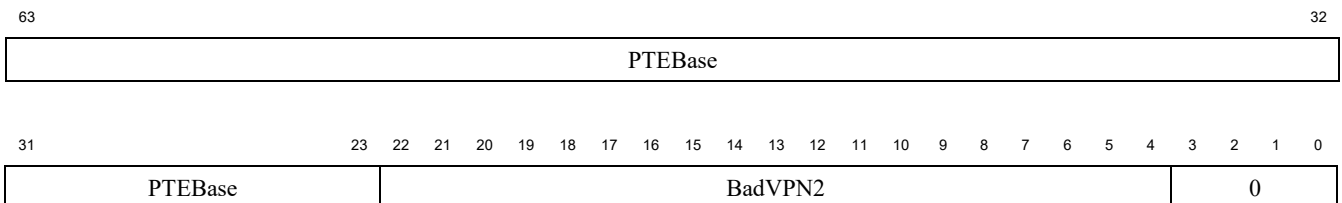


Table 7-7 Description of the Context register fields

Domain name	position	Functional description	Read/write	Reset value
PTEBase	63.. 23	Page table base address high level. Configured by the operating system software according to the current page table.	R/W	There is no
BadVPN2	22.. 4	When a TLB exception occurs, store the error virtual address of 31.. 13.	R	There is no
0	3.. 0	Read only is always 0.	0	0

7.6 UserLocal Register (CP0 Register 4, Select 2)

The UserLocal register is a read-write register that is not used to control the processor's hardware and is not changed by the hardware.

The contents of the UserLocal can be read in user mode via the RDHWR instruction, and whether it can be read is controlled by the HWRENCH a register bit 29. Figure 7-6 illustrates the format of the UserLocal register; Table 7-8 describes the UserLocal register fields.

Figure 7-6 UserLocal register format

63

0

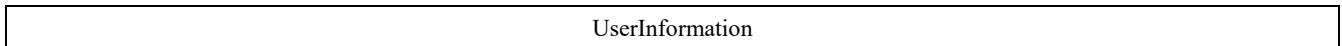


Table 7-8 Description of UserLocal register fields

Domain name	position	Function and description	Read/write	Reset value
UserInformation	63..0	The information stored is not affected by or affected by the processor hardware.	R/W	There is no

7.7 PageMask Register (CP0 Register 5, Select 0)

The PageMask register is a read-write register, used in the process of reading and writing TLB; It contains a comparison mask for each TLB

Table entries set different page sizes.

Figure 7-7 illustrates the format of the PageMask register; The PageMask register fields are described in Table 7-9.

Figure 7-7 PageMask register format

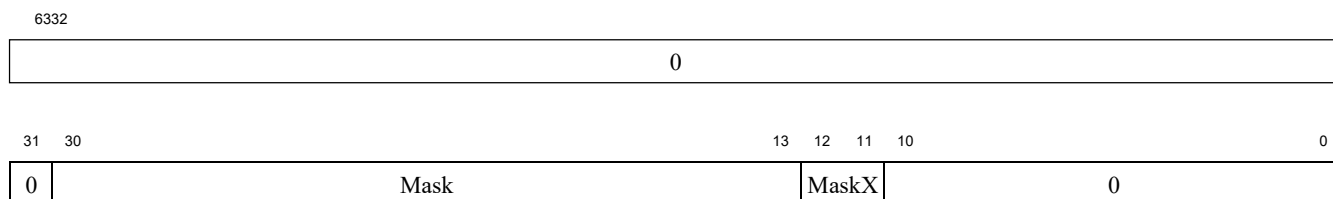


Table 7-9 PageMask register field description

Domain name	position	Functional description	Read/write	Reset value
0	63.. 31	Read only is always 0.	0	0
Mask	30.. 13	When translating between real and virtual addresses, each bit in the Mask field [17:0] is used to indicate whether the corresponding bit in the virtual address [30:13] bit is compared. 1: No comparison; 0: Compare. See Table 7-10 below for the Mask codes supported by longson and their corresponding page sizes.	R/W	0 x3
MaskX	12.. 11	Constant 3, 1KB pages are not supported.	R	0 x3
0	10.. 0	Read only is always 0.	0	0

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

Table 7-10 Mask domain codes and page sizes

Mask encoding	Page size
0 x0	4 KB
0 x3	16 KB
0 xf	64 KB
0 x3f	256 KB
0 XFF	1 MB
0 x3ff	4 MB
0 XFFF	16 MB
0 x3fff	64 MB
0 XFFFF	256 MB

0 x3fff	1 gb
---------	------

Programming tips:

Although GS464E allows the PageMask registers to be filled in: 0x1, 0x7, 0x1F, 0x7F, 0x1FF, 0x7FF, 0x1FFF, 0x7FFF, 0x1FFFF, the page size is 2KB ($n=0..^{2^{n+1}-1}$ 8), but the processor does not guarantee the correctness of the program running under this configuration.

7.8 PageGrain Register (CP0 Register 5, Select 1)

The PageGrain register is a read-write register. The GS464E implements only the parts of it that are related to the TLB XI/RI protection bit and large physical address pattern control.

Figure 7-8 illustrates the format of the PageGrain register; The fields of the PageGrain register are described in Table 7-11.

Figure 7-8 PageGrain register format

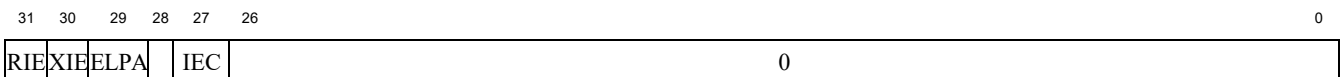


Table 7-11 Description of PageGrain register field

Domain name	position	Functional description	Read/write	Reset value
RIE	31	When I made a detour to the TLB wedge, I stopped in to make a functional detour. 0: Turn off the function. The RI bits of the EntryLo0 and EntryLo1 registers will disable writing and force them to be 0; 1: Enable this function. RI bits of the EntryLo0 and EntryLo1 registers work fine.	R/W	0 x0
XIE	30	When I stopped in to the TLB short wedge function, I made a short detour. 0: Turn off the function. The XI bits of the EntryLo0 and EntryLo1 registers are disabled and forced to be 0; 1: Enable this function. The XI bits of the EntryLo0 and EntryLo1 registers work fine.	R/W	0 x0
ELPA	29	Large physical address function enablement bit. 0: Turn off the function. The PFNX fields of the EntryLo0 and EntryLo1 registers disable writing and force 0; 1: Enable this function. The PFNX fields for the EntryLo0 and EntryLo1 registers work fine.	R/W	0 x0
0	28	Read only is always 0.	0	0
IEC	27	TLB reads block and performs block exception encoding and entry control bits. 0: TLB read block and execute block exception encoding and entrance multiplexing TLBL exception encoding and entrance; 1: TLB read block exception USES TLBRI exception encoding and entry, TLB performs block exception USES TLBXI exception encoding and entry.	R/W	0 x0
0	26.. 0	Read only is always 0.	0	0

Programming tips:

Before the software attempts to modify PageGrain any fields, all TLBS must be cleared and the fields of the COP0 register listed below must be set to the specified value, otherwise the behavior of the processor will be uncertain.

COP0 register field	Specify a value
---------------------	-----------------

EntryLo0PFN, EntryLo1PFN	0
EntryLo0PFNX, EntryLo1PFNX	0

7.9 PWBase Register (CP0 Register 5, Select 5)

The PWBase register is a 64-bit read-write register that holds the virtual address of the page table base address. This register is used in conjunction with the PWField, PWSize, and PWCtl registers and is used in GS464E to provide configuration information for the execution of the LDDIR and LDPTE directives. The LDDIR and LDPTE directives support traversal lookup of multilevel page table structure, which can contain up to four levels of catalog tables and one level of page table entries. The supported page table structure is shown in Figure 7-9.

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

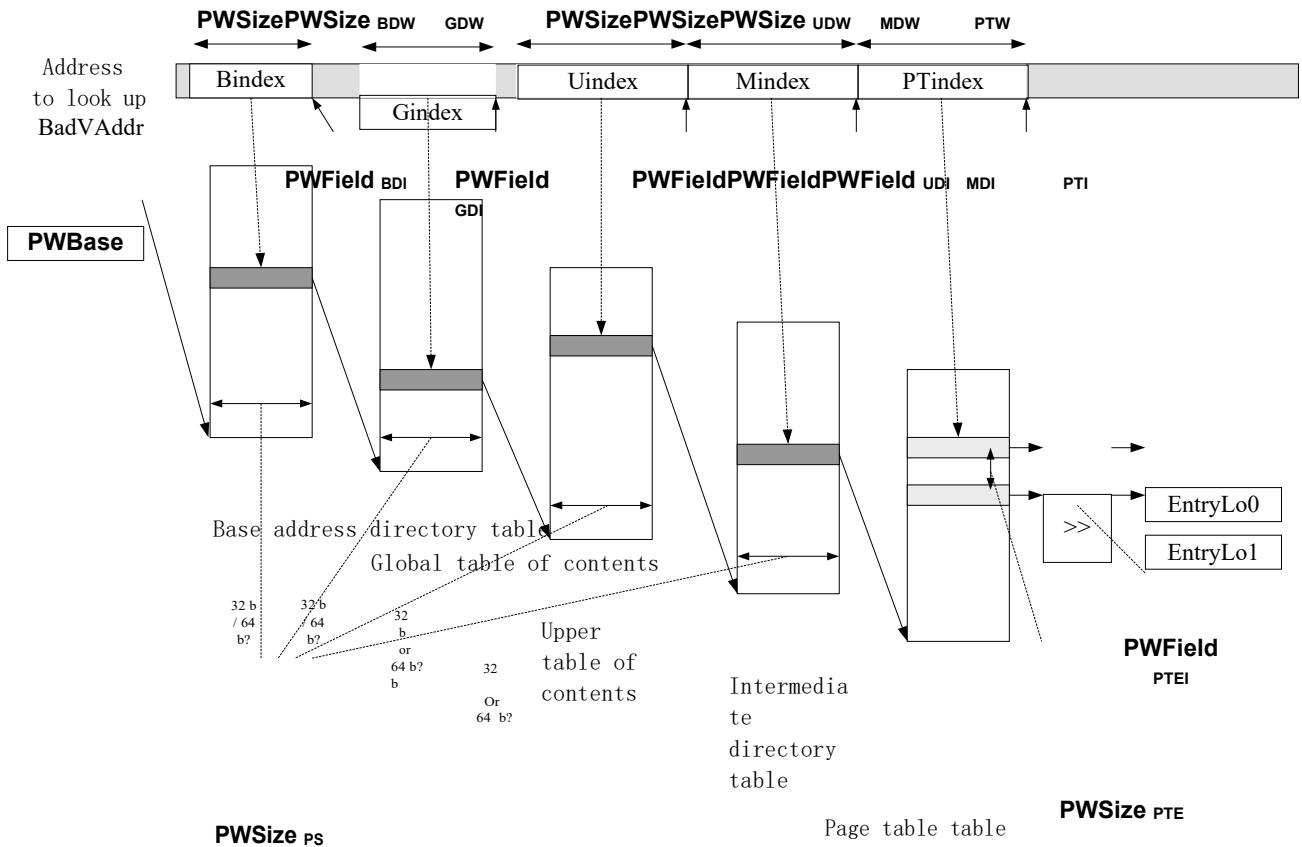


Figure 7-10 illustrates the format of the PWBase register; Table 7-12 describes the PWBase register fields.

Figure 7-10. PWBase register format

63

0

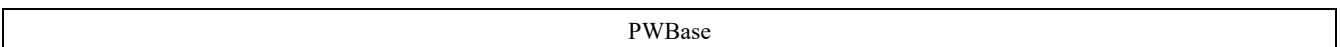


Table 7-12 Description of the PWBase register fields

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

7.10 PWField Register (CP0 Register 5, Select 6)

The PWField register is used in conjunction with the PWBase, PWSize, and PWCtl registers in GS464E to provide configuration information for the execution of the LDDIR and LDPTE directives. The LDDIR and LDPTE directives support traversal lookup of multilevel page table structure, which can contain up to four levels of catalog tables and one level of page table entries. See Figure 7-9 on page 94 for the supported page table structure and access procedure. The index value of each level page table to the next level page table or the final page table entry is obtained by intercepting part of the contiguous bit from the virtual address to be looked up (BadVAddr). The PWField register is used to identify the starting point of the page table index truncated in the pending virtual address (BadVAddr).

Figure 7-11 illustrates the format of the PWField register; Table 7-13 describes the PWField register fields.

Figure 7-11. PWField register format

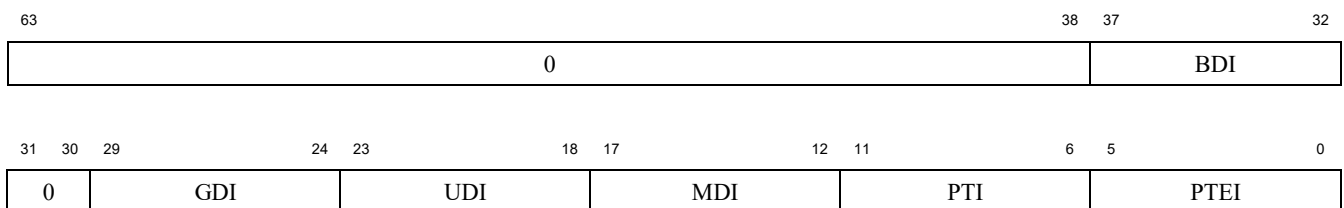


Table 7-13 Description of PWField register fields

Domain name	position	Functional description	Read/write	Reset value
0	63.. 38	Read only is always 0.	0	0
BDI	37.. 32	The Base Directory starts the index. The use of the base directory table is controlled by the PWCtl. $PWDirExt$ The base directory table can be used to distinguish between different page tables, such as the user page table and the kernel page table, which can be maintained separately.	R/W	0 x0
0	31.. 30	Read only is always 0.	0	0
GDI	29.. 24	Global Directory index starting position.	R/W	0 x0
UDI	23.. 18	The Upper Directory starts the index.	R/W	0 x0
MDI	17.. 12	Middle Directory is the starting point of an index.	R/W	0 x0
PTI	11.. 6	Page Table index starting position.	R/W	0 x0
PTEI	5.. 0	Page table item shift amount. The content of the page table entry read out will be logically shifted to the right pTEI-2 bit to remove the information in the page table entry that is only used for software and does not need to be put into TLB. The loop is then rotated 2 bits to the right to move the RI and XI two-bit information to either the highest two bits of EntryLo0 or EntryLo1. Therefore,	R/W	0 x0

		the PTEI field cannot be filled with 0 or 1, otherwise the processor result will be indeterminate.		
--	--	--	--	--

7.11 PWSize Register 5, Select 6

The PWSize registers are used in conjunction with the PWBase, PWField, and PWCtl registers and are used in GS464E to provide configuration information for the execution of the LDDIR and LDPTE directives. The LDDIR and LDPTE directives support traversal lookup of multilevel page table structure, which can contain up to four levels of catalog tables and one level of page table entries. See Figure 7-9 on page 94 for the supported page table structure and access procedure. The index value of each level page table to the next level page table or the final page table entry is obtained by intercepting part of the contiguous bit from the virtual address to be looked up (BadVAddr). The PWSize register is used to identify the number of contiguous bits intercepted by the page table index at each level in the virtual address to be looked up (BadVAddr).

The PWSize field is used to control whether the bit width of the pointer in the directory table is 32 or 64 bits. PS The index value taken from the virtual address to be looked up (BadVAddr) by the various table of contents according to PWField and PWSize needs to be multiplied by the width of the pointer in the table of contents (moved 2 bits left or moved 3 bits left) to form a true access address. Only 64-bit Pointers can be used in the XTLB Refill exception and 32-bit or 64-bit Pointers can be used in the TLB Refill exception.

Figure 7-12 illustrates the format of the PWField register; Table 7-13 describes the PWField register fields.

Figure 7-12. PWSize register format

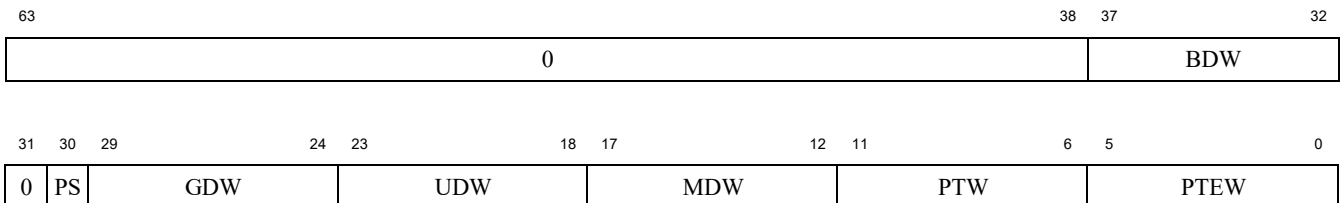


Table 7-14 Description of the PWSize register fields

Domain name	position	Functional description	Read/write	Reset value
0	63.. 38	Read only is always 0.	0	0
BDW	37.. 32	Base directory table index bit width. A value of 0 means that the base address directory table does not need to be looked up.	R/W	0 x0
0	31	Read only is always 0.	0	0
PS	30	Level directory table pointer bit width. 0:32 pointer; 1:64 pointer.	R/W	0 x0
GDW	29.. 24	Global directory table index bit width. 0 is meaningless and the processor result is uncertain when configured as 0.	R/W	0 x0
UDW	23.. 18	Upper directory table index bit width. 0 is meaningless and the processor result is uncertain when configured as 0.	R/W	0 x0
MDW	17.. 12	Intermediate directory table index bit width. 0 is meaningless and the processor result is	R/W	0 x0

		uncertain when configured as 0.		
PTW	11.. 6	Page table entry table index bit width. 0 is meaningless and the processor result is uncertain when configured as 0.	R/W	0 x0
PTEW	5.. 0	Page table item bit width. Used to control the access page table entry index to move 3 bits to the left (machine default data path width is 64 Bits) that need to be moved further left. For example, when the width of each page table entry (single page) is 16 bytes, PTEW is hard set to 1, and accessing the page table entry index moves to the left (3+1) bit to form the final access address.	R/W	0 x0

7.12 Wired Register (CP0 Register 6, Select 0)

The Wired register is a read-write register that defines the boundary between fixed and random replacement table items in VTLB. The diagram is shown in Figure 7-13.

Figure 7-13 Boundary of fixed table items and random replacement table items in VTLB

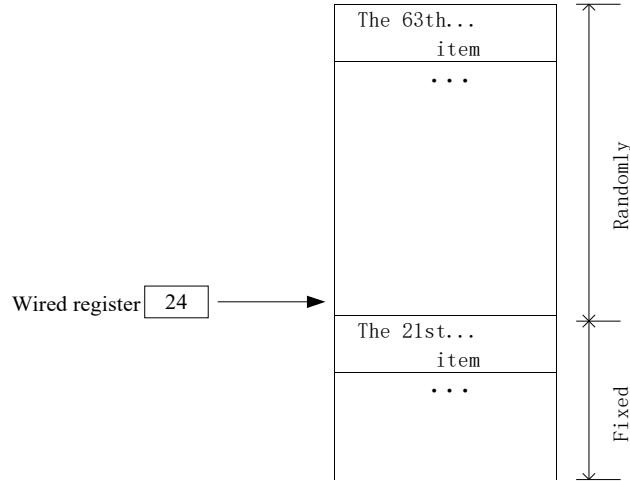


Figure 7-14 illustrates the format of the Wired register; Table 7-15 describes the Wired register fields.

Figure 7-14 Wired register format

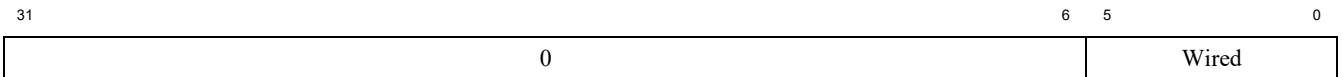


Table 7-15 Description of Wired register field

Domain name	position	Functional description	Read/write	Reset value
0	31.. 6	Read only is always 0.	0	0
Wired	5.. 0	TLB fixes the boundary between a table entry and a random replacement table entry.	R/W	0 x0

7.13 PWCtl Register 6, Select 6

The PWCtl register is used in conjunction with the PWBase, PWField, and PWSIZE registers in GS464E to provide configuration information for the execution of the LDDIR and LDPTE directives. The LDDIR and LDPTE directives support traversal lookup of multilevel page table structure, which can contain up to four levels of catalog tables and one level of page table entries. See Figure 7-9 on page 94 for the supported page table structure and access procedure. The index value of each level page table to the next level page table or the final page table entry is obtained by intercepting part of the contiguous bit from the virtual address to be looked up (BadVAddr). The PWSIZE register is used to identify the number of contiguous bits intercepted by the page table index at each level in the virtual address to be looked up (BadVAddr).

PWCtl controls whether the base directory table is used in page-table traversal lookup, and support for large pages.

Figure 7-15 illustrates the format of the PWCtl register; The PWCtl register fields are described in Table 7-16.

Figure 7-15 PWCtl register format

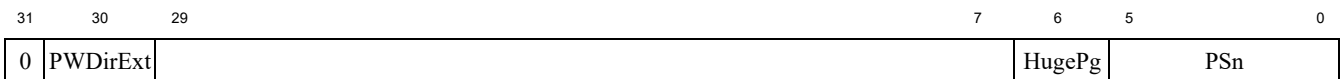


Table 7-16 Description of the PWCtl register fields

Domain name	position	Functional description	Read/write	Reset value
0	31	Read only is always 0.	0	0
PWDirext	30	Base Directory table function enablement bit. 1: Enable; 0: Disabled.	R/W	0 x0
0	29.. 7	Read only is always 0.	0	0
HugePg	6	1 indicates that large pages are supported in the table of contents; A value of 0 indicates that large pages are not supported in the table of contents.	R/W	0 x0
PSn	5.. 0	Used to indicate the location of the PTEVld bit of a table entry in a table of contents.	R/W	0 x0

7.14 Trade characters for a Register (CP0 Register 7, Select 0)

The Htrade A register holds a bit mask for controlling which hardware registers the RDHWR instruction can read in user mode. Figure 7-16 illustrates the format of the HWRENCH a register; Tables 7-17 describe the HWRENCH a register domains.

Figure 7-16 Trade A register format

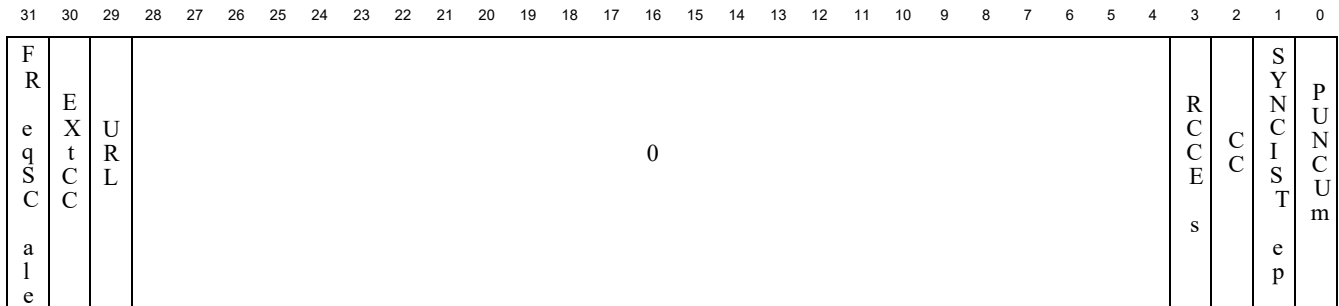


Table 7-17 Trade A describes the register fields

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
FreqScale	31	RDHWR 31 enable bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
ExtCC	30	RDHWR 30 (external Count register) enablement bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
ULR	29	RDHWR 29 (UserLocal register) enable bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
0	28.. 4	Read only is always 0.	0	0
CCRes	3	RDHWR 3 (Count autoincrement) enable bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
CC	2	RDHWR 2 (Count register) enable bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
SYNCI - Step	1	RDHWR 1 (SYNCI_Step) enable bit. 1: Read is allowed; 0: Read disabled.	R/W	0 x0
CPUNum	0	RDHWR 0 (EBase) enable bit. CPUNum 1: Read is allowed; 0: Read disabled.	R/W	0 x0

7.15 BadVAddr Register (CP0 Register 8, Select 0)

The BadVAddr register is a read-only register that records the last virtual address that caused the following exception:

- Address Error (AdEL or AdES)
- TLB/XTLB Refill
- TLB Invalid (TLBL or TLBS)
- TLB Modified

Figure 7-17 illustrates the format of the BadVAddr register; The BadVAddr register fields are described in Table 7-18.

Figure 7-17 BadVAddr register format

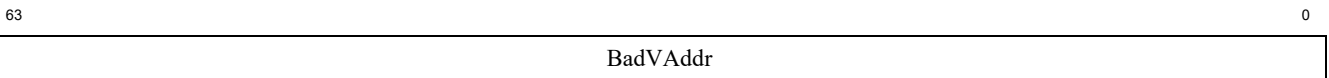


Table 7-18 BadVAddr register field description

Domain name	position	Functional description	Read/write	Reset value
BadVAddr	63..0	The wrong virtual address.	R	There is no

7.16 Count Register 9, Select 0

The Count register is used in conjunction with the Compare register to implement a high-precision timer and timed interrupts within a processor. The timer increases by 1 at 1/2 the frequency of the processor core pipeline clock. During execution, the processor core pipeline clock frequency may be dynamically adjusted, so the self-increasing frequency of Count changes accordingly.

The software can configure the Count register for certain functions or diagnostic purposes, such as timer reset, synchronization, and so on. The format of the Count register is explained. The Count register fields are described.

Figure 7-18 Count register format

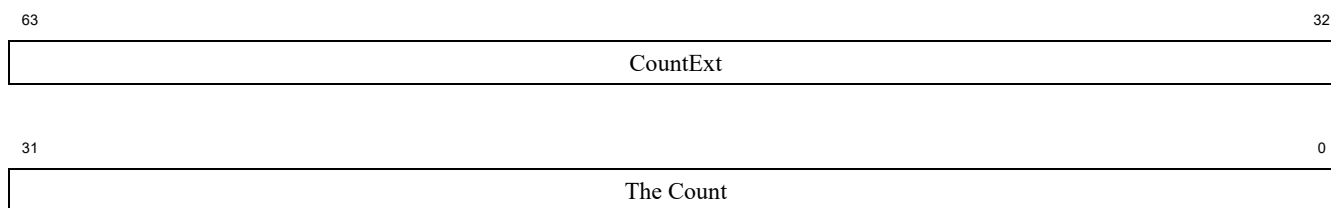


Table 7-19 Count register fields are described

Domain name	position	Functional description	Read/write	Reset value
CountExt	63.. 32	Internal counter high 32 bit extension.	R/W	There is no
The Count	31.. 0	Internal counter.	R/W	There is no

Programming tips:

The processor implements Count as a 64-bit counter. A timed interrupt caused by another Count/Compare inside the processor is still triggered by comparing the low 32-bit value of Count with the value of the Compare register. RDHWR (RD = \$3) returns the result of the Count register's low 32-bit symbol being extended to 64 bits. The MFC0 instruction can only read the result of the Count register's low 32-bit symbol extending to 64 bits, but when the MTC0 instruction is used to write the Count register, the processor writes all the 64-bit values from the source register to the Count register. So a recommendation: If the software wants to access the full 64-bit information in the Count register, use the DMFC0, DMTC0 instructions.

7.17 GSEBase Register (CP0 Register 9, Select 6)

The GSEBase register (GSEBase register) is a read-write register (READ-write register) which is used to prepare the base address of the loongson extension exception vector.

Figure 7-20 illustrates the format of the GSEBase register; Table 7-21 describes the GSEBase register fields.

Figure 7-19 GSEBase register format

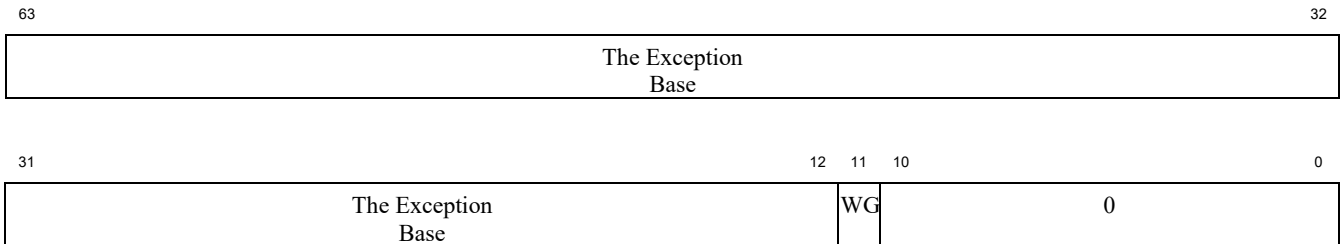


Table 7-20 Description of GSEBase register fields

Domain name	position	Function and description	Read/write	Reset value
The Exception Base	63..12	When status. BEV=0, the logic moves 12 bits to the left as the base address of the entry vector for the godson extension exception. The [63:30] bit can only be written if the WG bit is equal to 1, and the EBase register if the WG bit is equal to 0 The bit [63:30] remains the same.	R/W	0 XFFFF.FFFF 8000.0.
WG	11	The [63:30] write control bit. 1: ExceptionBase[63:30] can be written; 0: ExceptionBase[63:30] remained unchanged when written.	R/W	0 x0
0	10..0	Read only is always 0.	0	0

7.18 PGD Register (CP0 Register 9, Select 7)

The PGD register is a read-only register that holds the base address of the page table.

Figure 7-20 illustrates the format of the PGD register; The PGD register fields are described in Table 7-21.

Figure 7-20 PGD register format

63

0

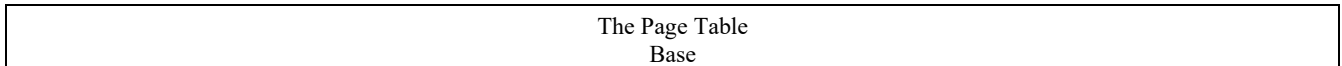


Table 7-21 Description of PGD register fields

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

7.19 EntryHi Register (CP0 Register 10, Select 0)

The EntryHi register is used to store high-level information about TLB table entries during TLB read, write, and query access.

Under normal instruction execution, the EntryHi domain holds the current address space id filled by the software and participates in TLB look-up along with the virtual address of a finger or fetch operation. ASID When a TLB exception occurs (the TLB Refill, XTLB Refill, TLB Invalid, or TLB Modifid exception), the appropriate portion of the error address that triggered the exception is written to the EntryHi and EntryHi fields. RVPN2 When the TLBP instruction is executed, the virtual address information and address space identification information for the items to be looked up are stored in the EntryHi, EntryHi, and EntryHi domains for TLB lookup. RVPN2ASID

When the TLBR instruction is executed, reading from the specified TLB table entry is also written to the corresponding field of the EntryHi register. Because the EXECUTION of the TLBR instruction overwrites the EntryHi domain, the software must save the ASID domain value before executing the TLBR instruction and recover in time after the TLBR execution. ASID This is particularly important for the TLB Invalid and TLB Modified exception handler, and other associated memory management code.

When the TLBWI and TLBWR directives are executed, the virtual address information and address space identification information for the entry to be written are stored in the EntryHi, EntryHi, and EntryHi fields for the TLB write. RVPN2ASID If you set the EntryHi field to 1 when executing the TLBWI instruction, you can invalidate the specified TLB table entry. EHINV Since the EntryHi domain is also overwritten by the CONTENTS of the TLBR instruction readout, it is important to maintain the EHINV domain as well as the ASID domain when executing the TLBR instruction, which is important for subsequent TLBWI instructions to execute correctly. EHINV

Figure 7-21 illustrates the format of the EntryHi register; Table 7-22 describes the EntryHi register fields.

Figure 7-21 In the EntryHi register format

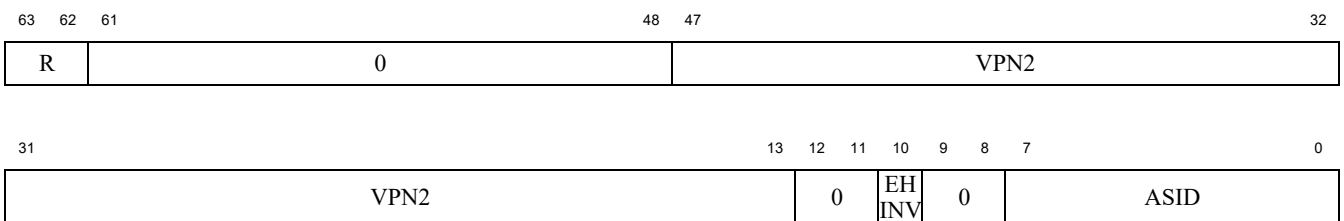


Table 7-22 Description of the EntryHi register field

Domain name	position	Function	Read/write	Reset value
		1 descripti on		

R	63.. 62	An area identifies a bit, corresponding to the [63:62] bit of the virtual address. 0B00: User address region (XUSeg, User Address region); 0B01: Supervisor Address Region (XSSEG, Supervisor Address Region); 0B10: Retention; 0B11: Kernel address region (Xksege, Kernel Address region).	R/W	0 x0
0	61.. 48	Read only is always 0.	0	0
VPN2	47.. 13	The virtual page number is divided by 2(mapped to a double page) and corresponds to the [47:13] bit of the virtual address.	R/W	0 x0
0	12.. 11	Read only is always 0.	0	0
EHINV ¹	10	TLB invalid token bit. When the position is 1, executing the TLBWI instruction invalidates the corresponding TLB table entry. When the TLBR instruction is executed, the bit is set to 1 if the TLB table entry read is invalid.	R/W	0 x0
0	9.. 8	Read only is always 0.	0	0

¹ although Config4=0, GS464E still implements the EHINV domain according to the definition of MIPS specification in the case of Config4>1. ^{IEIE} It is recommended to use the EntryHi domain functionality for software such as the GS464E deeply customized kernel to facilitate TLB management. EHINV

Domain name	position	Function and description	Read/write	Reset value
ASID	7.. 0	Address space identification number. Used to share TLB between multiple processes; By using the ASID to differentiate, for the same Virtual page number, which allows different processes to take different mappings.	R/W	0 x0

7.20 Compare Register (CP0 Register 11, Select 0)

The Compare register is combined with the Count register to implement a high-precision timer and timed interrupts inside a processor. The value stored in the Compare register remains unchanged after writing, compared to the low 32-bit Count register, and when both are equal, the timer is interrupted, causing the Cause to be 1. IT When vector interrupt mode is not in use, timer interrupt will connect to the break 7 (Cause, hard break 5). IP7 When using vector interrupt mode, the interrupt line to which the timer interrupts is determined by IntCtl. IPTI

When the software writes the Compare register, the hardware automatically clears the Cause of 0, thus clearing the timer interrupt. IT

Figure 7-22 illustrates the format of the Compare register; Table 7-23 describes the Compare register fields.

Figure 7-22 Compare register format

31

0

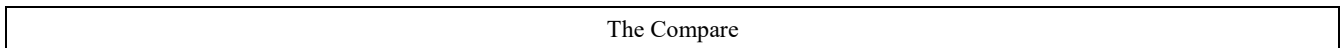


Table 7-23 Compare register field description

Domain name	position	Function and description	Read/write	Reset value
The Compare	31..0	Interval counts compare values.	R/W	0 x0

7.21 Status Register (CP0 Register 12, Select 0)

The Status register is a read-write register that contains processor mode of operation, interrupt enablement, and processor Status diagnostics. Figure 7-23 illustrates the Status register format; Table 7-24 describes the Status register fields.

Figure 7-23 Status register format

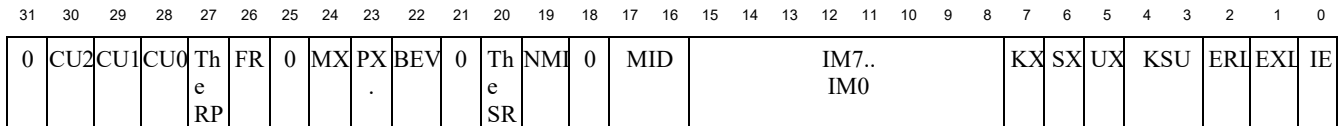


Table 7-24 Description of the Status register fields

Domain name	position	Function description	Read/write	Reset value
0	31	Read only is always 0.	0	0
CU2	30	Coprocessor 2 has identity bits available. 1: Available; 0: Disabled.	R/W	0 x0
CU1	29	Coprocessor 1(floating point coprocessor) can be used to identify bits. 1: Available; 0: Disabled.	R/W	0 x1
CU0	28	The coprocessor 0 is available with identity bits. 1: Available; 0: Disabled. When the processor is in Kernel mode and Debug mode, coprocessor 0 can always be used without consideration CU0 bit is equal to 1.	R/W	0 x1
The RP	27	Processor dynamic frequency down enable bit. 1: Open; 0: Off.	R/W	0 x0
FP	26	The floating point register mode controls the bits. 0:16 floating-point registers, even numbered, each 64-bit, single-precision stored on the lower 32-bit; 1:32 floating point registers, serial Numbers, each 64-bit, single-precision stored in low 32-bit.	R/W	0 x0
0	25	Read only is always 0.	0	0
MX	24	Enable bits to access DSP resources. 1: Accessible; 0: No access.	R/W	0 x0
PX.	23	Enable control bits for 64-bit operations in user mode are not used to enable the 64-bit address space. (The rest of the mode 64-bit operation without enabling) 1: Enabling; 0: Disabled.	R/W	0 x1
BEV	22	Exception vector entry address control. 0: Normal; 1: Boot up.	R/W	0 x1
0	21	Read only is always 0.	0	0
The SR	20	Used to indicate that the entry leading to the Reset exception vector is due to Soft Reset. 1: Soft reset; 0: Not a soft Reset (possibly NMI or Reset). The jump.	R/W	0 x0

NMI	19	Used to indicate that the reset exception vector entry is due to a non-masking interrupt (NMIIt). 1: It is non-masking interrupt; 0: It is not a non-masking interrupt (Reset or Soft Reset). The jump.	R/W	0 x0
0	18	Read only is always 0.	0	0

Domain name	position	Function description	Read/write	Reset value
MID	17.. 16	The loong-son custom field is used to indicate which address space the current default operation is in. Whether the address space in which the reference falls is controlled by the MID domain is determined by the diag.inst domain. If a visit memory operation does not carry MID information, it will fall in the address space specified in the root.status.mid field; otherwise, it will fall in the address space determined by the MID information carried by it. GS464E USES a 2-bit code (MID) to identify the four isolated full address Spaces. MID=0 is the address space visible to MIPS host, MID=1 is the address space visible to virtual machines in guest mode, and other values of MID can be assigned to other virtual machines by software. The MID field can be written only when dig.vmm =1, and will be set to 0 no matter what value is written when dig.vmm =0.	R/W	0 x0
IM7.. IM0	15.. 8	Interrupt mask bit. Each bit controls the enabling of an external interrupt, an internal interrupt, or a software interrupt. 1: Enabling; 0: Masking.	R/W	0 x0
KX	7	1. Can access 64-bit Kernel segment, which USES XTLB Refill exception vector; 0:64-bit Kernel segment cannot be accessed. Kernel segment access USES the EXCEPTION vector TLB Refill.	R/W	0 x1
SX	6	1. Can access the 64-bit Supervisor segment, which USES the XTLB Refill exception vector; 0: The 64-bit Supervisor segment cannot be accessed. The Supervisor segment accesses use the TLB Refill exception vector.	R/W	0 x1
UX	5	1: Can access 64-bit User segment, which USES XTLB Refill exception vector and allows instruction operation of 64-bit data in User mode; 0:64-bit User segment cannot be accessed. User segment access USES the TLB Refill exception vector and does not allow User modules The following instruction operates on 64-bit data.	R/W	0 x1
KSU	4.. 3	Processor mode identifies bits. 0B00: Kernel Mode 0B01: Supervisor Mode 0B10: User Mode 0B11: Reserve.	R/W	0 x0
ERL	2	Error level. This bit is set to 1 when Reset, Soft Reset, NMI, and Cache Error exceptions occur. 0: Normal; 1: Error level. When ERL position is 1: <ul style="list-style-type: none"> The processor is automatically in core mode All hardware and software interrupts are shielded The ERET instruction reads the return address from the ErrorEPC register Kuseg segments will be treated as having directly mapped non-cached attributes (Unmapped and uncached) 	R/W	0 x1

EXL	1	<p>The exception class. This bit is set to 1 when an exception occurs that is not Reset, Soft Reset, NMI, or Cache Error exceptions. 0: Normal; 1: Exception level.</p> <p>When position 1 of EXL is:</p> <ul style="list-style-type: none"> • The processor is automatically in core mode • All hardware and software interrupts are shielded • TLB/XTLB Refill exception processing adopts general exception vector entry instead of TLB/XTLB Refill exception vector entry • EPC, Cause do not update when new exceptions occur. <small>BD</small> 	R/W	0 x0
IE	0	<p>Global interrupt enablement bit.</p> <p>0: Screen all hardware and software interrupts; 1: Enable all hardware and software interrupts.</p>	R/W	0 x0

7.22 IntCtl Register (CP0 Register 12, Select 1)

The IntCtl register is a read-write register that extends the interrupt mechanism of the processor.

Figure 7-24 illustrates the format of the IntCtl register; The IntCtl register fields are described in Table 7-25.

Figure 7-24 IntCtl register format

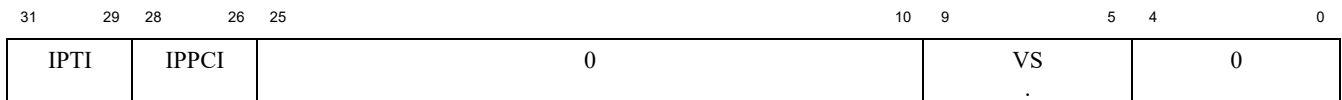


Table 7-25 IntCtl register fields are described

Domain name	position	Function description	Read/write	Reset value												
IPTI	31.. 29	Used in vector interrupt mode to indicate on which interrupt line the timer interrupt merges. The value is constant 7, indicating that it is merged in IP7, and the hardware is disconnected on HW5.	R	0 x7												
IPPCI	28.. 26	Used in vector interrupt mode to indicate on which interrupt line the performance counter overflows the interrupt. The value is constant 7, indicating that it is merged in IP7, and the hardware is disconnected on HW5.	R	0 x7												
0	25.. 10	Read only is always 0.	0	0												
VS.	9.. 5	Used to define the space between interrupt vector entry addresses in vector interrupt mode.	R/W	0 x0												
		<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">VS coding</th> <th style="width: 15%;">Vector space</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0 x00</td><td style="text-align: center;">0 x000</td></tr> <tr><td style="text-align: center;">0 x01</td><td style="text-align: center;">0 x020</td></tr> <tr><td style="text-align: center;">0 x02</td><td style="text-align: center;">0 x040</td></tr> <tr><td style="text-align: center;">0 x04</td><td style="text-align: center;">0 x080</td></tr> <tr><td style="text-align: center;">0 x08</td><td style="text-align: center;">0 x100</td></tr> </tbody> </table>			VS coding	Vector space	0 x00	0 x000	0 x01	0 x020	0 x02	0 x040	0 x04	0 x080	0 x08	0 x100
		VS coding			Vector space											
		0 x00			0 x000											
		0 x01			0 x020											
		0 x02			0 x040											
		0 x04			0 x080											
0 x08	0 x100															
The encoding values are retained except for those listed in the table above. If a reserved value is configured for the VS domain, processor results will be indeterminate.																
0	4.. 0	Read only is always 0.	0	0												

7.23 SRSCtl Register (CP0 Register 12, Select 2)

The SRSCtl register is used to control the shadow register. Since GS464E implements only one set of general purpose registers, the shadow of the general purpose register is the general purpose register itself.

Figure 7-25 illustrates the format of the SRSCtl register; The SRSCtl register fields are described in Table 7-26.

Figure 7-25 SRSCtl register format

31	30	29	26	25	16	15	12	11	10	9	6	5	4	3	0
0	HSS			0	ESS			0	PSS			0	CSS		

Table 7-26 SRSCtl register field description

Domain name	position	Function and description	Read/write	Reset value
0	31.. 30	Read only is always 0.	0	0
HSS	29.. 26	A value of 0 indicates that only one set of general purpose registers is implemented.	R	0 x0
0	25.. 16	Read only is always 0.	0	0
ESS	15.. 12	The group number of the shadow register group used for exception handling. GS464E can only write 0 and other value handler rows Is uncertain.	R/W	0 x0
0	11.. 10	Read only is always 0.	0	0
PSS	9.. 6	The group number of the previous set of shadow registers. GS464E can only write 0, write other values processor behavior is uncertain.	R/W	0 x0
0	5.. 4	Read only is always 0.	0	0
CSS	3.. 0	The value is always 0, indicating that the current shadow register group is the general purpose register group.	R	0 x0

7.24 Cause Register (CP0 Register 13, Select 0)

The Cause register is used primarily to describe the Cause of the most recent exception. In addition, software interrupt and interrupt vector are also controlled. In addition to

IP1.. Outside the DC, IV and WP domains, the other domains of the Cause register are read-only to the software.

Figure 7-26 illustrates the format of the Cause register; Table 7-27 describes the Cause register fields.

Figure 7-26 Cause register format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BD	TI	CE	DC	PCI	0	IV	0				IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0	ExcCode											

Table 7-27 Description of the Cause register field

Domain name	position	Function description	Read/write	Reset value
BD	31	Identifies whether the instruction with the most recent exception is in the branch delay slot. 1: In the delay slot; 0: Not in the delay slot	R	0 x0
TI	30	Timer interruption indication. 1: Interruption of timer to be handled; 0: No timer interrupt.	R	0 x0
CE	29.. 28	Records the coprocessor number that is unavailable when an exception occurs.	R	0 x0
DC	27	The Count register disables the Count control bit. 1: Stop the Count; 0: Enable Count Count.	R/W	0 x0
PCI	26	Performance counter overflow interrupt indication. 1: Performance counter overflow interrupt to be processed; 0: No performance counter overflow interrupt.	R	0 x0
0	25.. 24	Read only is always 0.	0	0
IV	23	Interrupt exception vector entry control bit. 1: Special interrupt vector (0x200) is used; 0: Use the generic exception vector (0x180).	R/W	0 x0
0	22.. 16	Read only is always 0.	0	0
IP7.. IP2	15.. 10	Pending hardware interrupt identification. Each bit corresponds to a middle break, and IP7~IP2 correspond to hardware interrupt 5~0 in turn. 1: The interrupt to be processed on the interrupt line; 0: There is no interrupt on the interrupt line.	R	0 x0
IP1.. IP0	9.. 8	Pending software interrupt identification. Each bit corresponds to a software interrupt, and IP1~IP0 corresponds to software interrupt 1~0 in turn. The software interrupt identification bit can be set and cleared by the software.	R/W	0 x0
0	7	Read only is always 0.	0	0
ExcCode	6.. 2	Exception coding. Please see the detailed description.		
0	1.. 0	Read only is always 0.	0	0

Table 7-28 ExcCode codes and their corresponding exception types

ExcCode	mnemoni cs	desc ribe
0 x00	Int	interrupt
0 x01	The Mod	TLB modification exceptions
0 x02	TLBL	TLB exception (read data or fetch instruction)
0 x03	TLBS	TLB exception (write data)
0 x04	AdEL	Address error exception (read data or fetch instruction)
0 x05	AdES	Address error exception (write data)

ExcCode	mnemonics	desc ribe
0 x06	IBE	MIPS specifies a defined bus error exception (fetch instruction). Because GS464E does not implement the IBE exception, the reason for the exception is retained in the encoding. If the system prints "Bus Error" in the software debugging, please focus on checking whether there are other exceptions.
0 x07	DBE	Bus error exception (read or write data). Because GS464E does not implement the DBE exception, the reason for the exception is encoded. If the system prints "Bus Error" in the software debugging, please focus on checking whether there are other exceptions.
0 x08	Sys	The system call exception.
0 x09	Bp	Break point exception. If SDBBP instructions are executed in EJTAG Debug mode, the exception encoding 0x9(Bp) is written to Debug Register in the DExcCode field.
0 x0a	RI	Preserve instruction exceptions.
0 x0b	The CpU	No exceptions can be made to the coprocessor.
0 x0c	Ov	Calculate the overflow exception.
0 x0d	The Tr	Trap exception.
0 x0e	MSAFPE	Unrealized.
0 x0f	FPE	The floating point exception.
0 x10	GSExc	Loongson custom exception. Includes floating point stack exception, virtual machine memory management exception, virtual machine space TLB example Outside. The software can look at the related fields of the GSCause register to see exactly what exceptions occur.
0 x11	-	reserve
0 x12	-	reserve
0 x13	TLBRI	TLB reads prevent exceptions
0 x14	TLBXI	TLB performs blocking exceptions
0 x15	MSADis	Unrealized.
0 x16	MDMX	Unrealized.
0 x17	WATCH	Unrealized.
0 x18	MCheck	Unrealized.
0 x19	The Thread	Unrealized.
0 x1a	DSPDis	The DSP module disables exceptions
0 x1b	GE	Unrealized.
0 x1c	-	reserve
0 x1d	-	reserve
0 x1e	-	Cache error exception. Because the Cache error exception USES a dedicated vector entry address, the Cause register's ExcCode field is not updated when the Cache error exception occurs in normal mode. When in Debug mode, the exception encoding 0x1e is written to the DExcCode field of the Debug register.
0 x1f	-	reserve

7.25 EPC Register (CP0 Register 14, Select 0)

The EPC register is a 64-bit read-write register that contains a PC that continues to execute instructions after the exception processing has completed. In response to the synchronous (exact) exception, the processor writes to the EPC register:

The PC directly triggers the exception instruction.

When the instruction that directly triggered the exception is in the branch delay slot, record the PC of the previous branch or jump instruction of the instruction, at the same time caused.

Set to 1.

In response to an asynchronous (imprecise) exception, a PC in which the processor writes an instruction to the EPC register that continues execution after the exception processing has completed. The EPC register is not updated when the Status register EXL bit is 1.

The format of EPC register is explained. The EPC register fields are described.

Figure 7-27 EPC register format

63

0

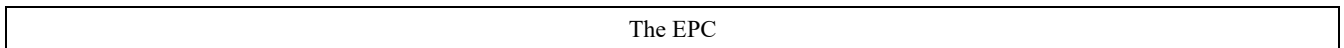


Table 7-29 Description of EPC register fields

Domain name	position	Function and description	Read/write	Reset value
The EPC	63..0	Exception program counter.	R/W	There is no

7.26 PRId Register (CP0 Register 15, Select 0)

The PRId register is a 32-bit read-only register that contains information identifying the MIPS processor manufacturer, processor type, and implementation version.

Figure 7-28 illustrates the format of the PRId register; The PRId register fields are described in Table 7-30.

Figure 7-28 PRId register format

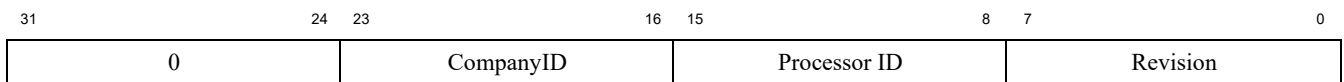


Table 7-30 Description of the PRId register fields

Domain name	position	Function al description	Read /write	Reset value
0	31.. 24	Read only is always 0.	0	0
CompanyID	23.. 16	Company ID number. When dig.idsel is 0, the value is 0x14; When dig.idsel is 1, the value is 0x00.	R	0 x14
ProcessorID	15.. 8	Processor type number. 0 x63.	R	0 x63
Revision	7.. 0	Implementation version number. When dig.idsel is 0, the value is 0x08; When dig.idsel is 1, the value is 0x05.	R	0 x08

7.27 EBase Register (CP0 Register 15, Select 1)

The EBase register is a read-write register that contains the exception vector base address and a read-only CPU number. Figure 7-29 illustrates the format of the EBase register; The EBase register fields are described in Table 7-31.

Figure 7-29 EBase register format

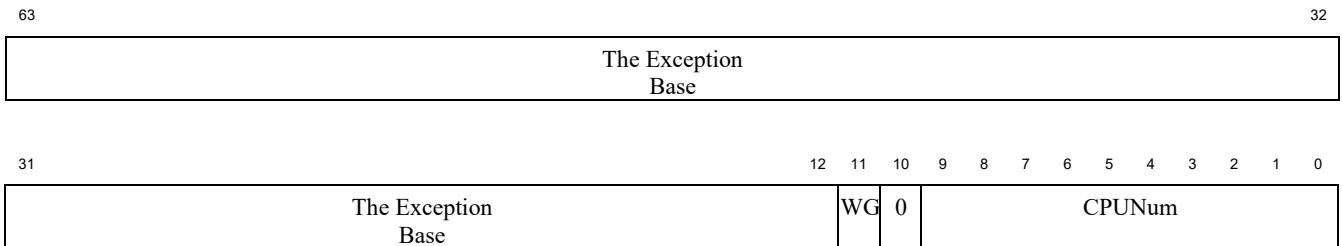


Table 7-31 Description of EBase register fields

Domain name	position	Functional description	Read/write	Reset value
The Exception Base	63.. 12	When status. BEV=0, the logic moves 12 bits to the left as the base address of the exception entry vector. The [63:30] bit can only be written if the WG bit is equal to 1, and the EBase register if the WG bit is equal to 0 The bit [63:30] remains the same.	R/W	0 XFFFF.FFFF 8000.0.
WG	11	The [63:30] write control bit. 1: ExceptionBase[63:30] can be written in; 0: ExceptionBase[63:30] remained unchanged when written.	R/W	0 x0
0	10	Read only is always 0.	0	0
CPUNum	9.. 0	The index number that identifies the current processor on a multicore system.	R	

Programming tips:

When vector interrupt mode is used (caus.iv =1), if intctL.vs is configured to 0x10, the exception vector offset with interrupt Number 7 will exceed the 0xFFf range. At this point, the software needs to ensure that EBase position 12 is 0 and give up the highest bit of the interrupt vector offset of No.7.

7.28 Config Register (CP0 Register 16, Select 0)

The Config register defines some processor configuration information. Config register except K0 field can be read and written by software, other fields are initialized by hardware during reset and reserve the read-only state unchanged. Although GS464E resets the K0 domain hardware to 0B010 (Uncached property), it is strongly recommended that the software initialize this domain in the Reset exception handler.

Figure 7-30 illustrates the format of the Config register; Table 7-32 describes the fields of the Config register.

Figure 7-30. Config register format

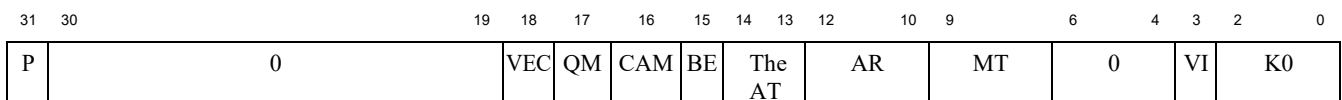


Table 7-32 Description of Config register field

Domain name	position	Functional description	Read/write	Reset value
M	31	A value of 1 indicates that the Config1 register exists.	R	0 x1
0	30.. 19	Read only is always 0.	0	0
VEC	18	A value of 0 indicates that the custom 256-bit vector instruction is not supported.	R	0 x0
QM	17	A value of 1 indicates that 128-bit access memory instruction is supported for loongson customization. See Table 2-27 on page 35 for the instructions involved.	R	0 x1
CAM	16	A value of 1 indicates that it contains hardware CAM components and supports longson custom CAM instructions. Please see the instructions involved Page 44 Table 2-32.	R	0 x1
BE	15	A value of 0 indicates small tail addressing.	R	0 x0
The AT	14.. 13	A value of 2 represents the implementation of the MIPS64 architecture and access to the 64-bit full address space.	R	0 x2
AR	12.. 10	With a value of 1, it is compatible with the MIPS64 Release 5 specification, and the details of the implementation can be read by the software The configuration domain of a configuration register or other register is obtained.	R	0 x1
MT	9.. 7	With a value of 4, MMU adopts VTLB and FTLB double TLB. See double TLB MMU for information Section 4.3.	R	0 x4
0	6.. 4	Read only is always 0.	0	0
VI	3	Value is 0, and the instruction Cache adopts the form of virtual address Index and real address Tag. Please refer to the organization form of the Cache See section 5.1.	R	0 x0
K0	2.. 0	Cache property of Kseg0 segment. See Table 7-6 on page 89 for the Cache attribute encoding supported by GS464E.	R/W	0 x2

7.29 Config1 Register (CP0 Register 16, Select 1)

The Config1 register is used to provide some configuration information for the processor. All fields in the Config1 register are read-only.

Figure 7-31 illustrates the Config1 register format. Table 7-33 Table 7-32 Config register field description Describes each field of the Config1 register.

Figure 7-31 Config1 register format

31	30	25	24	22	21	19	18	16	15	13	12	10	9	7	6	5	4	3	2	1	0
M	MMUSize - 1	IS	IL	IA	DS	DL	DA	C2	MD	The PC	WR	The CA	EP	FP							

Table 7-33 Config1 register field description

Domain name	position	Functional description	Read/write	Reset value
M	31	A value of 1 indicates that the Config2 register exists.	R	0 x1
MMU The Size of 1	30.. 25	The value is 63, which is splice with the VTLBSizeExt field of Config4 register to form a 10-bit value: Config4. VTLBSizeExt Config1. MMUSize - 1; 3..05..0 The value after splicing is 63, indicating that VTLB is 64 items.	R	0 x3f
IS	24.. 22	A value of 2 indicates that the I-cache contains 256 rows per path.	R	0 x2
IL	21.. 19	A value of 5 indicates that the line length of i-cache is 64 bytes.	R	0 x 5
IA	18.. 16	A value of 3 indicates that the I-cache contains 4 channels.	R	0 x3
DS	15.. 13	A value of 2 indicates that the D-cache contains 256 rows per path.	R	0 x2
DL	12.. 10	A value of 5 indicates that d-cache has a row length of 64 bytes.	R	0 x 5
DA	9.. 7	A value of 3 indicates that the D-cache contains four channels.	R	0 x3
C2	6	A value of 1 indicates the inclusion of coprocessor 2 (COP2).	R	0 x1
MD	5	A value of 0 indicates that the MDMX ASE instruction set is not implemented.	R	0 x0
The PC	4	A value of 1 indicates that a performance counter is implemented.	R	0 x1
WR	3	A value of 0 indicates that the Watch register is not implemented.	R	0 x0
The CA	2	A value of 0 represents the unimplemented MIPS16e instruction set.	R	0 x0
EP	1	A value of 1 means that EJTAG is implemented.	R	0 x1
FP	0	A value of 1 indicates that the floating-point coprocessor is implemented.	R	0 x1

7.30 Config2 Register (CP0 Register 16, Select 2)

The Config2 register is used to provide some configuration information for the processor. All fields in the Config2 register are read-only.

Figure 7-32 illustrates the Config2 register format. Table 7-34 Table 7-32 Config2 register fields are described.

Figure 7-32 Config2 register format

31	30	29	28	27	26	25	24	23	22	21	20	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			The SA													

Table 7-34 Config2 register field description

Domain name	position	Functional description	Read/write	Reset value
M	31	A value of 1 indicates that the Config3 register exists.	R	0 x1
0	30.. 28	Read only is always 0.	0	0
TS	27.. 24	A value of 2 indicates that v-Cache contains 256 rows per path.	R	0 x2
TL	23.. 20	A value of 5 indicates that v-Cache has a row length of 64 bytes.	R	0 x 5
TA	19.. 16	A value of 15 indicates that the V-cache contains 16 channels.	R	0 xf
0	15.. 12	Read only is always 0.	0	0
SS	11.. 8	A value of 4 indicates that S-Cache contains 1024 rows per path.	R	0 x4
SL	7.. 4	A value of 5 indicates that s-Cache has a row length of 64 bytes.	R	0 x 5
The SA	3.. 0	A value of 15 indicates that S-cache contains 16 channels.	R	0 xf

7.31 Config3 Register (CP0 Register 16, Select 3)

The Config3 register is used to provide some configuration information for the processor. All fields in the Config3 register are read-only.

Figure 7-33 illustrates the Config3 register format. Table 7-35 Table 7-32 Config register field description Describes each field of THE Config3 register.

Figure 7-33 Config3 register format

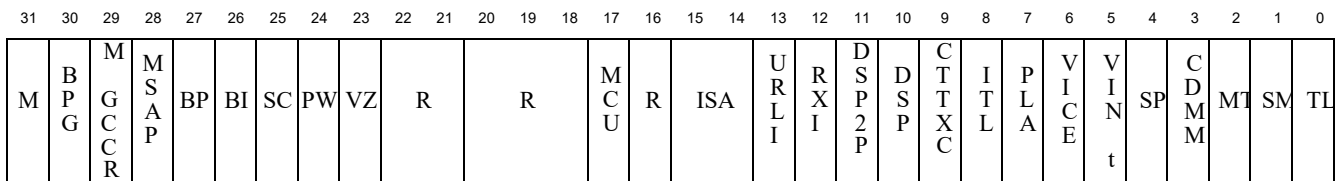


Table 7-35 Config3 register field description

Domain name	position	Functional description	Read/write	Reset value
M	31	A value of 1 indicates the existence of the Config4 register.	R	0 x1
combined	30	A value of 1 indicates that TLB supports large pages over 256MB, and the corresponding PageMask register is 64-bit.	R	0 x1
CMGCR	29	A value of 0 represents unimplemented Coherency Manager memory-Mapped Global Configuration Register Space. GS464E supports multicore, using a custom multicore consistency management mechanism instead of MIPS multicore consistency Sex management framework, and MIPS does not regulate its multi-core conformance management.	R	0 x0
MSAP	28	A value of 0 represents an unimplemented MIPS Vector Module (SIMD Module).	R	0 x0
BP	27	A value of 0 indicates that the BadInstrP register has not been realized.	R	0 x0
BI	26	A value of 0 indicates that the BadInstr register is not implemented.	R	0 x0
SC	25	A value of 0 means the function of Segment Control is not realized. Accordingly, the SegCtl0 for segment control, SegCtl1 and SegCtl2 registers are not implemented.	R	0 x0
PW	24	A value of 0 indicates that the Hardware Page Table Walk is not implemented. However, the PWBase, PWField, and PWSIZE registers defined by the MIPS specification for the hardware TLB refill mechanism are still defined in GS464E to match the execution of the loon-core custom LWTR and LDTR instructions. In addition, PWBase, PWField and PWSIZE registers are implemented into four groups, corresponding to four different address Spaces with SpaceID=0, 1, 2 and 3 respectively.	R	0 x0
VZ	23	The value is 1.	R	0 x1
R	22.. 21	This domain is meaningless because MIPS MCU ASE is not implemented.	R	0 x0
R	20.. 18	This field is meaningless because the micrMIPS64 instruction set is not implemented.	R	0 x0
MCU	17	A value of 0 indicates that MIPS MCU ASE is not implemented.	R	0 x0
R	16	This field is meaningless because both MIPS64 and microMIPS64 are not implemented.	R	0 x0
ISA	15.. 14	A value of 0 indicates that only the MIPS64 instruction set is implemented, and no	R	0 x0

		microMIPS64 instruction set is implemented.		
ULRI	13	A value of 1 indicates that the UserLocal register is implemented.	R	0 x1
RXI	12	A value of 1 indicates that RIE and XIE bits are implemented in the PageGrain register.	R	0 x1
DSP2P	11	A value of 1 indicates that MIPS DSP module version 2 is implemented.	R	0 x1
DSP	10	A value of 1 indicates that the MIPS DSP module is implemented.	R	0 x1

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
CTXTC	9	A value of 0 indicates that the ContextConfig and XcontextConfig registers are not implemented. Since it is not recommended that operating systems running on GS464E adopt single-level page table results, it is not recommended that the contents of the Context and Xcontext registers be used as Pointers to page table entries during TLB exception handling. Correspondingly, the ContextConfig and XcontextConfig registers are also not implemented.	R	0 x0
ITL	8	A value of 0 indicates that MIPS IFlowTrace debugging is not implemented.	R	0 x0
LPA	7	A value of 1 indicates that a large physical address range is supported. Accordingly, the PageGrain register is implemented.	R	0 x1
VEIC	6	A value of 0 indicates that the external interrupt Controller (EIC) mode is not implemented in the interrupt mechanism.	R	0 x0
he	5	A value of 1 indicates that the Vectored Interrupts (Vectored Interrupts) mode is implemented in the interrupt mechanism.	R	0 x1
SP	4	A value of 0 indicates that small pages of 1KB size are not supported.	R	0 x0
CDMM	3	A value of 0 indicates that the Common Device Memory Map mechanism is not implemented.	R	0 x0
MT	2	A value of 0 represents an unimplemented MIPS multithreaded Module (MT Module).	R	0 x0
SM	1	A value of 0 indicates that SmartMIPS ASE is not implemented. TM	R	0 x0
TL	0	A value of 0 indicates that Trace logic is not implemented.	R	0 x0

7.32 Config4 Register (CP0 Register 16, Select 4)

The Config4 register is used to provide some configuration information for the processor and to control the page size of the FTLB.

Figure 7-34 illustrates the Config4 register format. Table 7-36 describes the Config4 register fields.

Figure 7-34 Config4 register format

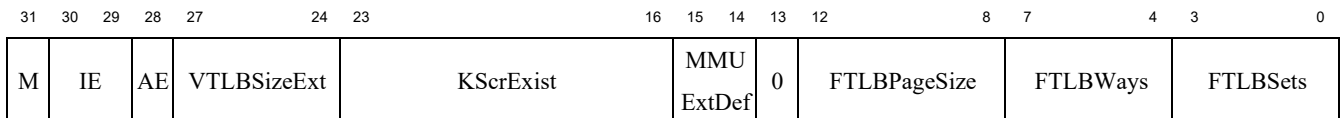


Table 7-36 Config4 register field description

Domain name	position	Function description	Read/write	Reset value
M	31	A value of 1 indicates the existence of the Config5 register.	R	0 x1
IE ¹	30.. 29	A value of 0 indicates that the TLBINV and TLBINVF directives are not implemented and the EntryHi domain is not implemented. <small>EHINV</small>	R	0 x0
AE	28	A value of 0 indicates that the EntryHi field width is still 8 bits wide. <small>ASID</small>	R	0 x0
VTLB - SizeExt	27.. 24	The value is 0, which is splice with the Mmusize-1 field of Config1 register to form a 10-bit value: Config4.VTLBSizeExt Config1.MMUSize - 1; <small>3..05..0</small> The value after splicing is 63, indicating that VTLB is 64 items.	R	0 x0
KScrExist	23.. 16	A value of 0b11111100 indicates that registers KScratch1~6 (CP0 Register 31, Selt 2~7) can be in Kernel mind for software access.	R	0 XFC
MMU - ExtDef	15.. 14	The value is 3 and is used to interpret the Config4 register format. Where Config4[3:0] is FTLBSets, Config4[7:4] FTLBWays, Config4[10:8] is FTLBPageSize, Config4[27:24] is VTLBSizeExt.	R	0 x3
0	13	Read only is always 0.	0	0

The IE field value is set to 0 to maintain compatibility with the MIPS specification. In fact, the GS464E processor core implements the TLBINVF instruction (config4.ie =3), which invalidated the entire TLB table entry by the hardware when executing a TLBINVF instruction. In addition, the TLBINV instruction is also implemented in the GS464E processor core, but its execution effect is different from the MIPS specification definition and is equivalent to the TLBINVF instruction.

The GS464E processor core also implements the entryhi. EHINV field: When the entryhi. EHINV position is 1, TLBWI will disable the entry. When the TLBR instruction is executed, the value of the invalid VPN2 bit for the TLB table entry read is updated to the entryhi.ehinv bit.

Domain name	position	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 encoding values	R/W	0 x1
		4 kb1		
		16 kb2		
		64 kb3		
		256 kb4		
		1 mb5		
		4 mb6		
		16 mb7		
		64 mb8		
		256 mb9		
	1 gb10			
		If the value the software writes to the field is not included in the table above, the value of the field remains the same. The software must clear FTLB before modifying the domain, otherwise the processor's behavior is uncertain.		
FTLB - Ways	7.. 4	A value of 6 indicates that FTLB contains 8 channels	R	0 x6
FTLB - Sets.	3.. 0	A value of 7 means that FTLB contains 128 items in each path.	R	0 x7

7.33 Config5 Register (CP0 Register 16, Select 5)

The Config5 register is used to provide some configuration information for the processor.

Figure 7-35 illustrates the Config5 register format. Table 7-37 Table 7-32 Config register field description Describes each field of THE Config5 register.

Figure 7-35 Config5 register format

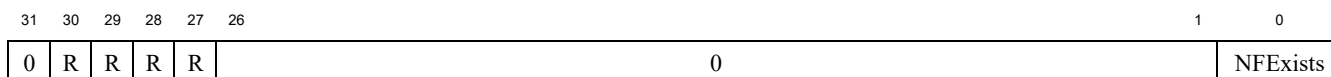


Table 7-37 Config5 register field description

Domain name	position	Functional description	Read/write	Reset value
0	31	Read only is always 0.	0	0
R	30	Since the Segmentation Control pattern is not implemented, the field is meaningless.	R	0 x0
R	29	Since the Segmentation Control pattern is not implemented, the field is meaningless.	R	0 x0
R	28	Since the Segmentation Control pattern is not implemented, the field is meaningless.	R	0 x0
R	27	Because the MIPS Vector Module (SIMD Module) is not implemented, this field is meaningless.	R	0 x0
0	26.. 1	Read only is always 0.	0	0
NFExists	0	The value is 1.	R	0 x1

7.34 GSConfig Register (CP0 Register 16, Select 6)

The GSConfig register is used to dynamically configure the microstructure-related functions of the processor core portion. The e software can turn on or off the corresponding functions according to the specific characteristics of the program to achieve the best performance.

Figure 7-36 illustrates the format of the GSConfig register; Table 7-38 describes the fields of the GSConfig register.

Figure 7-36 GSConfig register format

31	30	29	24	23	22	21	18	17	16	15	14	13	12	10	9	8	7	6	5	4	3	2	1	0
B P P A s s	0	KPos	KE	V T L B O N l y	0	S C R A n d	E L X c	D I S V C A c h e	V C L R U	D , C , L , R , U	0	R E s e R V E D	S T F I l l	E X t I m e r	i N , N , e R , T I m e r	0	D I S S T P R e f	N O R M S T P R e f	I P R e f	D P R e f				

Table 7-38 Description of GSConfig register fields

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
BpPass	31	EJTAG instruction and data breakpoint response mechanism control. 0: Any condition that satisfies an instruction breakpoint and a data breakpoint triggers an exception. 1: EJTAG instruction breakpoint and data breakpoint exceptions Automatically ignore the condition that the instruction breakpoint and data breakpoint were met the first time when the exception handler returns for re-execution. When the location of 0, the processor will be treated in accordance with the requirements for MIPS EJTAG specification instruction breakpoints and data breakpoints, which means the EJTAG exception handler must change in processing instruction breakpoint or data breakpoints judgement conditions, in order to ensure the current trigger breakpoints exception instruction DERET again after return will not trigger the exception again into dead circulation, and to ensure that the program can new judgment conditions in time to stop, make an exception The handler has time to set the breakpoint condition correctly before the observed program executes to the observed breakpoint again.	R/W	0 x0
0	30	Read only is always 0.	0	0
KPos	29.. 24	Indicates which bit K is located in the page table entry (PTE).	R/W	0 x3d
KE	23	The TLB page table kernel performs protection enabled bits. 0: Turn off the function. The K bits of the EntryLo0 and EntryLo1 registers will disable writing and force them to be 0;	R/W	0 x0

		1: Enable this function. The K bits of the EntryLo0 and EntryLo1 registers work fine.		
VTLB - Only	22	When set to 1, the processor will only operate the VTLB in the double TLB, which is equivalent to the traditional single CAM-type TLB of MIPS, so as to ensure that the original operating system kernel and other underlying software running on GS464 can still run on GS464E without modifying the MMU part. If you want an MMU with a double TLB, set the position to 0. The software should clear the TLB before modifying the bit state, otherwise the processor behavior will be unknowable.	R/W	0 x1
0	21.. 18	Read only is always 0.	0	0
SCRand	17	When set to 1, the Cache consistency component in the chip will initiate the SC/SCD instruction in the processor core The return delay of a Cache access request is increased by 64 to 128 random clock cycles. Turn off the function when set to 0.	R/W	0 x1

Domain name	position	Functional description	Read/write	Reset value
LLExc	16	When is 1, the LL/LLD instruction will initiate a Cache access request that must be returned in an Exclusive state Cache block; At 0, the LL/LLD instruction initiates a Cache access request that allows the Shared Cache to be returned Block.	R/W	0 x1
Dis - VCache	15	0: Use VCache; 1: Disable VCache. In moving from using VCache to disabling VCache, the software needs to ensure that nothing is valid in VCache The content.	R/W	0 x0
VCLRUI	14	Configure the VCache replacement algorithm. 1: LRU replacement algorithm; 0: Pseudo-random replacement algorithm.	R/W	0 x1
DCLRUI	13	Configure the DCache replacement algorithm. 1: LRU replacement algorithm; 0: Pseudo-random replacement algorithm.	R/W	0 x1
0	12.. 10	Read only is always 0.	0	0
Reserved	9	The bit must be written to 1 after it is reset and no longer changed to 0.	R/W1	0 x0
STFill	8	Processor Store operation auto write merge function enable bit. 1: Open; 0: Off. Before the software can modify this state, it needs to use the SYNC instruction to ensure that there are no pending accesses in the processor.	R/W	0 x1
Ext - Timer	7	When set to 1, the clock interrupt recorded by caus.ti may come from an external timer belonging to the processor core; When set to 0, the clock interrupt recorded by caus.ti does not come from an external timer belonging to the processor core. It should be pointed out that the increase frequency of external timer is not affected by processor core frequency conversion. Allows software to set the ExtTimer and InnerTimer bits of the GSConfig register to 1 simultaneously, but requires soft This is handled correctly and is generally not recommended.	R/W	0 x0
Inner - Timer	6	When set to 1, clock interrupts recorded by caus.ti can come from inside the processor core with Count/Compare The register realizes the timer; When set to 0, the clock interrupt recorded by caus.ti does not come from the timer implemented in the Count/Compare register inside the processor core. It should be noted that the timer implemented with the Count/Compare register increases in frequency proportionally with the processor core frequency. Allows software to set the ExtTimer and InnerTimer bits of the GSConfig register to 1 simultaneously, but requires soft This is handled correctly and is generally not recommended.	R/W	0 x1
0	5.. 4	Read only is always 0.	0	0
Dis - STPref	3	When set to 1, the processor does not automatically prefetch the store operation in the data access operation; When set to 0, the processor will also perform hardware automatic prefetching of store operation, and the performance of Store prefetching can be further configured through GSConfig, as described in the next item. NormSTPref This domain is only meaningful when GSConfig=1, and modifying it when GSConfig=0 does not cause a performance change. DPrefDPref Before the software can modify this state, it needs to use the SYNC instruction to ensure that there are no pending accesses in the processor Operation.	R/W	0 x0

Domain name	position	Functional description	Read/write	Reset value
Norm- STPref	2	<p>When set to 1, when the processor performs hardware automatic prefetching of the store operation, the flow control algorithm adopted is the same as the LOAD operation; Set to 0, the processor to store operation hardware automatic prefetching, in addition to adopt with the load operation flow control mechanism, store over a period of time will be at the same time monitoring operation automatically write merger success turns a Cache block (collecting) the number of times, if success number exceeds a certain threshold, the automatic prefetching to suspend hardware store operation.</p> <p>This domain is only meaningful when GSConfig=1, and modifying it when GSConfig=0 does not cause a performance change. <small>DPrefDPref</small></p> <p>Before the software can modify this state, it needs to use the SYNC instruction to ensure that there are no pending accesses in the processor Operation.</p>	R/W	0 x0
IPref	1	<p>When set to 1, the processor performs hardware automatic prefetching for finger operation. Otherwise, turn it off.</p> <p>Before the software can modify this state, it needs to use the SYNC instruction to ensure that there are no pending accesses in the processor.</p>	R/W	0 x1
DPref	0	<p>When set to 1, the processor performs hardware automatic prefetch for data access. Otherwise, turn it off.</p> <p>Before the software can modify this state, it needs to use the SYNC instruction to ensure that there are no pending accesses in the processor.</p>	R/W	0 x1

7.35 LLAddr Register (CP0 Register 17, Select 0)

The LLAddr register is a 64-bit read-only register that holds the physical address of the recently occurred Load Linked instruction. The LLAddr register is cleared when the exception returns.

Figure 7-37 illustrates the format of the LLAddr register; The LLAddr register fields are described in Table 7-39.

Figure 7-37. LLAddr register format

63

0



Table 7-39 Description of the LLAddr register field

Domain name	position	Functional description	Read /write	Reset value
PAddr	63.. 0	The physical address of the recently occurred Load Linked directive	R	There is no

7.36 XContext Register (CP0 Register 20, Select 0)

The XContext register is a read-write register that contains some page-table base address high-level information filled in by the operating system software and some bits of the error virtual address where the TLB exception occurs. According to the original design intent of the MIPS architecture, the information splited together in the XContext register can form a pointer to an item in the page table, which can be accessed when a TLB exception occurs. The page table accessible without any processing of the contents of the XContext register is a single-level page table structure, with a page size of 4K bytes and each page table entry of 16 bytes, containing an even page table entry and an odd page table entry with consecutive virtual addresses. When the page table does not have this structure, the software needs to properly shift and concatenate the contents of the XContext register. For an operating system with multilevel page tables, the XContext register can only be used to speed up address generation for the last level of page table access.

The XContext register is primarily used in the XTLB Refill exception handler. However, when exceptions such as TLB Refill, TLB Invalid, and TLB Mod occur, the BadVPN2 and R fields in the XContext register are also updated, so the software can also use the XContext register in the corresponding exception handler.

The BadVPN2 and R domains in the XContext register copy some of the information in the BadVAddr register, but this does not mean that this part is completely equivalent. When the Address Error exception occurs, the BadVaddr register will be updated by the hardware, but the BadVPN2 and R fields of the Context register will not be updated by the hardware.

Figure 7-38 illustrates the format of the XContext register; The XContext register fields are described in Table 7-40.

Figure 7-38 XContext register format

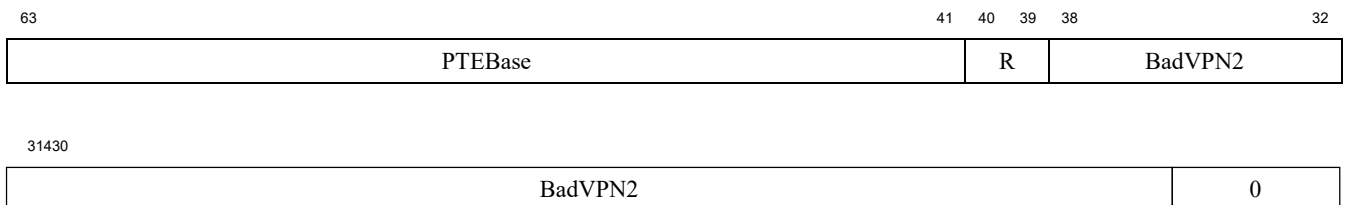


Table 7-40 Description of XContext register fields

Domain name	position	Function and description	Read/write	Reset value
PTEBase	63.. 41	Page table base address high level. Configured by the operating system software according to the current page table.	R/W	There is no
R	40.. 39	When a TLB exception occurs, store the error virtual address 63.. A 62 - bit. 0B00: General user area; 0B01: Superuser area; 0B10: Retention; 0B11: Core area.	R	There is no
BadVPN2	38.. 4	When a TLB exception occurs, store the error virtual address of 47.. 13.	R	There is no

0	3..0	Read only is always 0.	0	0
---	------	------------------------	---	---

7.37 Diag Register (CP0 Register 22, Select 0)

The Diag register is a proprietary register that controls the execution of the virtual machine and some internal queues and special operations.

Figure 7-39 illustrates the format of the Diag register; Table 7-41 Table 7-32 Config register field description describes each field of the Diag register

Above.

Figure 7-39 Diag register format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I D S E L	0										MID	I N S T	V M M	T B E L X	0	F T L B	V T L B	0	G C C A	U C C A	W C A C	W I S S	S I S S	S T F E	D T L B	I T L B	B T B	R A S			

Table 7-41 Description of Diag register fields

Domain name	position	Functional description	Read/write	Reset value
IDSEL	31	Controls the readout value of the PRId register. When the bit is 0, the PRId readout value is 0x00146308; When the bit is 1, the PRId read value is 0x00006305. The latter is identical to the PRId of the LS3A1000 chip processor. This bit can be modified to 1 in PMON to enable the operating system kernel and above running on GS464 to be compatible with GS464E. It is important to note that the IDSEL bit can only be modified once after each hard restart. It is suggested that this revision be made in PMON.	R/W	0 x0
0	30.. 20	Read only is always 0.	0	0
MID	19.. 18	Stores the MID of the address space accessed by the directive that triggered the exception when exceptions such as TLB Refill, TLB Invalid, and TLB Mod occur. When TLBWR and TLBWI are used to fill in THE TLB, the contents of the field will be written into the TLB table entry to participate in the subsequent virtual and real address mapping. When TLB is read using TLBP and TLBR instructions, the MID content stored in the read table entry is stored in this field. This field is used when the software reads and writes the PWBase, PWField, PWSize registers to indicate which space the page table configuration information is to be accessed. The MID field can be written only when dig.vmm =1, and will be set to 0 no matter what value is written when dig.vmm =0.	R/W	0 x0
INST	17	Whether the virtual and real address mapping of PC USES the flag of SpaceID information, 0: Not used; 1: Use. This field is only valid if the VMM position of the Diag register is 1, otherwise the field can read and write normally but does not participate in any other operations.	R/W	0 x0

The VMM	16		R/W	0 x0
TLBEX	15	When set to 1, TLBR, TLBP, TLBWI, TLBWR, TLBINVF are executed in root-core mode The instruction will trigger the VMMU exception.	R/W	0 x0
0	14	Read only is always 0.	0	0

Domain name	position	Function al description	Read /write	Reset value
FTLB	13	Empty FTLB when writing 1 to the bit. Note that the processor is concerned with the behavior of the bit being written to 1, clearing FTLB is independent of whether the bit value is 1. The bit readout value is always 0.	R0 / W	0
VTLB	12	Clear VTLB when writing 1 to the bit. Note that the processor is concerned with the behavior of the bit being written to 1, clearing FTLB is independent of whether the bit value is 1. The bit readout value is always 0.	R0 / W	0
0	11.. 10	Read only is always 0.	0	0
GCAC	9		R/W	0 x0
UCAC	8	When set to 1, CACHE0, CACHE1, CACHE3, The CACHE15, CACHE21, and CACHE23 instructions will not trigger the coprocessor exception (CpU).	R/W	0 x0
WCAC	7	Set 1 to unrestrict the wait Cache operation.	R/W	0 x0
WISS	6	Set 1 to cancel the wait Issue operation.	R/W	0 x0
SISS	5	Cancel the stall Issue operation restriction when setting 1.	R/W	0 x0
SFET	4	Cancel the stall fetch operation restriction when set 1.	R/W	0 x0
ITLB	3	Write 1 to empty ITLB for that bit. Note that the processor is concerned with the bit being written to 1, emptying FTLB It doesn't matter if the bit value is 1. The bit readout value is always 0.	R0 / W	0
ITLB	2	Write 1 to empty ITLB for that bit. Note that the processor is concerned with the bit being written to 1, emptying FTLB It doesn't matter if the bit value is 1. The bit readout value is always 0.	R0 / W	0
BTB	1	To this bit, BRBTB and BTAC in the 1 empty-branch prediction are written. Note that the processor focuses on that bit Emptying FTLB does not matter whether the bit value is 1 or not. The bit readout value is always 0.	R0 / W	0
The RAS	0	When set 1, RAS is disabled for branch prediction of JR31.	R/W	0 x0

7.38 GSCause Register (CP0 Register 22, Select 1)

The GSCause register is a read-only register that contains information related to the loong-core extension when an exception occurs, such as whether the trigger exception instruction contains a prefix, and the specific reason for the loong-core extension exception.

Figure 7-40 illustrates the format of the GSCause register; The GSCause register fields are described in Table 7-42.

Figure 7-40 GSCause register format

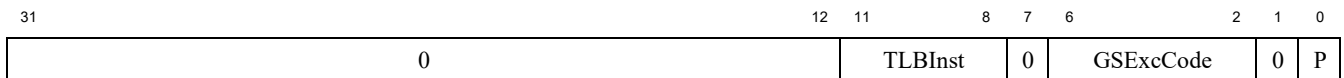


Table 7-42 Describes the GSCause register fields

Domain name	position	Function and description	Read/write	Reset value
0	31.. 12	Read only is always 0.	0	0
TLBInst	11.. 8	When the PSI exception is triggered by TLB instructions executed in guest state, the specific TLB instruction type is recorded. TLBInst[0] 1 means TLBP instruction; TLBInst[1] is a TLBR instruction. TLBInst[2] 1 means TLBWR instruction; A value of 1 for TLBInst[3] indicates a TLBWI, TLBINV, or TLBINVF instruction.	R	0 x0
0	7	Read only is always 0.	0	0
GS - ExcCode	6.. 2	Loongson extension exception coding.	R	0 x0
0	1	Read only is always 0.	0	0
P	0	1: The instruction that triggers the exception carries a prefix with a command length of 64 bits; 0: The instruction that triggers the exception has no prefix and is 32 bits long.	R	0 x0

Table 7-43 GSExcCode codes and corresponding exception types

ExcCode	mnemonics	description
0 x00	IS	Floating point stack exception
0 x01	VMMU	Virtual machine memory management unit exception
0 x02	VMTLBL	Virtual machine address space TLB exception (read data or fetch instructions)
0 x03	VMTLBS	Virtual machine address space TLB exception (write data)
0 x04	VMMMod	Virtual machine address space TLB modification modification
0 x05	VMTLBRI	Virtual machine address space is not readable exception
0 x06	VMTLBXI	No exceptions can be made to the virtual machine address space

7.39 VPID Register (CP0 Register 22, Select 2)

The VPID register is a self-defined register used to control the VPID information of the virtual machine.

Figure 7-39 illustrates the format of the Diag register; Table 7-41 Table 7-32 Config register field description describes each field of the Diag register

Above.

Figure 7-41 VPID register format

31	16	15	8	7	0
0	VPMSK			VPID	

Table 7-44 VPID register field description

Domain name	position	Functional description	Read/write	Reset value																																												
0	31.. 16	Read only is always 0.	0	0																																												
VPMSK	15.. 8	<p>The virtual machine mask is used to control the number of bits of virtual and real address mapping in the VPID domain of Diag register. The mask also controls the actual bits of virtual address high position participating in virtual address mapping, that is, it can be considered that the SEGBITS value of the processor at this time has been changed.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 20%;">VPMSK</td> <td style="width: 20%;">VPID equivalent</td> <td style="width: 20%;"></td> <td style="width: 40%; text-align: center;">Virtual address valid bit range</td> </tr> <tr> <td></td> <td>Valid value valid range SEGBITS</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0 x00048 {Vaddr (63-62), Vaddr [47:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0 x80vpid [7] 47 {Vaddr (63-62), Vaddr [46:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>46 {0 xc0vpid [but] Vaddr (63-62), Vaddr [45:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>45 {0 xe0vpid [then] Vaddr (63-62), Vaddr [44:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>44 {0 xf0vpid [17] Vaddr (63-62), Vaddr [43:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>43 {0 xf8vpid [and] Vaddr (63-62), Vaddr [42:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0 xfcvpid [when] {42 Vaddr (63-62), Vaddr [47:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0 xfevpid [7:1] {41 Vaddr (63-62), Vaddr [47:0]}</td> <td></td> <td></td> </tr> <tr> <td></td> <td>0 xffvpid [away] {40 Vaddr (63-62), Vaddr [47:0]}</td> <td></td> <td></td> </tr> </table> <p>If VPMSK is configured with an invalid value, processor behavior is unpredictable. This field is only valid if the VMM position of the Diag register is 1, otherwise the field can read and write normally but does not participate in any other operations.</p>	VPMSK	VPID equivalent		Virtual address valid bit range		Valid value valid range SEGBITS				0 x00048 {Vaddr (63-62), Vaddr [47:0]}				0 x80vpid [7] 47 {Vaddr (63-62), Vaddr [46:0]}				46 {0 xc0vpid [but] Vaddr (63-62), Vaddr [45:0]}				45 {0 xe0vpid [then] Vaddr (63-62), Vaddr [44:0]}				44 {0 xf0vpid [17] Vaddr (63-62), Vaddr [43:0]}				43 {0 xf8vpid [and] Vaddr (63-62), Vaddr [42:0]}				0 xfcvpid [when] {42 Vaddr (63-62), Vaddr [47:0]}				0 xfevpid [7:1] {41 Vaddr (63-62), Vaddr [47:0]}				0 xffvpid [away] {40 Vaddr (63-62), Vaddr [47:0]}			R/W	0 x0
VPMSK	VPID equivalent		Virtual address valid bit range																																													
	Valid value valid range SEGBITS																																															
	0 x00048 {Vaddr (63-62), Vaddr [47:0]}																																															
	0 x80vpid [7] 47 {Vaddr (63-62), Vaddr [46:0]}																																															
	46 {0 xc0vpid [but] Vaddr (63-62), Vaddr [45:0]}																																															
	45 {0 xe0vpid [then] Vaddr (63-62), Vaddr [44:0]}																																															
	44 {0 xf0vpid [17] Vaddr (63-62), Vaddr [43:0]}																																															
	43 {0 xf8vpid [and] Vaddr (63-62), Vaddr [42:0]}																																															
	0 xfcvpid [when] {42 Vaddr (63-62), Vaddr [47:0]}																																															
	0 xfevpid [7:1] {41 Vaddr (63-62), Vaddr [47:0]}																																															
	0 xffvpid [away] {40 Vaddr (63-62), Vaddr [47:0]}																																															
VPID	7.. 0	<p>A virtual machine number whose significant digits are controlled by the VPMSK domain, as described in the preceding item.</p> <p>This field is only valid if the VMM position of the Diag register is 1, otherwise the field can read and write normally but does not participate in any other operations.</p>	R/W	0 x0																																												

7.40 Debug Register (CP0 Register 23, Select 0)

The Debug register is a 32-bit read-write register. This register contains the reasons for the EJTAG debug exceptions that occurred recently and for exceptions in debug mode, and is used to control single-end debug interrupts. At the same time, the register also controls the resources in debug mode, indicating the internal state of the processor. For a description of the Debug register refer to the MIPS EJTAG specification.

7.41 DEPC Register (CP0 Register 24, Select 0)

The DEPC register is a 64-bit read-write register that contains the EJTAG debug exception or the PC that continues to execute instructions after the exception is processed in debug mode.

In response to an exception, the processor writes to the DEPC register: the PC that directly triggers the instruction of the exception.

When the instruction that directly triggered the exception is in the branch delay slot, record the PC of the previous branch or jump instruction of the instruction, at the same time caus.bd

Debug.dbd is set to 1.

Figure 7-42 illustrates the FORMAT of the DEPC register; The DEPC register fields are described in Table 7-45.

Figure 7-42 DEPC register format

63

0



Table 7-45 Description of DEPC register field

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
DEPC	63.. 0	EJTAG continues to execute instructions after completion of the exception processing on the PC.	R/W	There is no

7.42 PerfCnt Register (CP0 Register 25, Select 0~7)

The PerfCnt register is a set of CP0 registers for processor performance event statistics. Each set of performance counters is composed of a pair of even and odd-adjacent CP0 registers with Select number, namely, Select0~1 produces a PerCnt0, Select2~3 produces a PerCnt1, Select4~5 produces a PerCnt2, and Select6~7 produces a PerCnt3. The even-number registers in each set of performance counters are Control registers (PerfCnt Control Reg) used to define event categories and Control counting conditions. The odd-number register is the value register (PerfCnt Counter Reg) for recording secondary values. See Table 7-46.

Table 7-46 PerfCnt register Select allocation

Performance counter	Select the value	PerfCnt register
0	The Select 0	PerfCnt Control Register 0
	Select 1	The Register 0 PerfCnt Counter
1	Select 2	PerfCnt Control Register 1
	Select 3	The Register 1 PerfCnt Counter
2	Select 4	PerfCnt Control Register 2
	Select 5	The Register 2 PerfCnt Counter
3	Select 6	PerfCnt Control Register 3
	The Select 7	PerfCnt Counter Register 3

GS464E implements four sets of PerfCnt0~PerfCnt3 performance counters, and the register format definition follows the example given in the MIPS specification. Unlike the MIPS specification, however, the PerfCnt register in GS464E is actually a read/write interface to the processor core's internal performance counters rather than the actual counters. In short, the software first establishes a relationship between the event to be operated on and a specific performance counter inside the processor core by configuring the event information in the PerfCnt register, and then reads and writes to the PerfCnt register of that group actually act on the specific performance counter specified. This is designed to break the MIPS architecture limit of four performance events that a single processor can count at the same time.

Figure 7-43 illustrates the format of the PerfCnt Control register; The PerfCnt Control register fields are described in Table 7-47. Figure 7-44 illustrates the format of the PerfCnt Counter register; The PerfCnt Counter register fields are described in Table 7-48.

Figure 7-43 PerfCnt Control register format

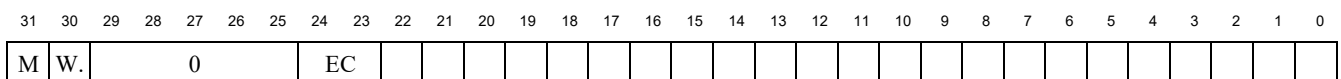


Table 7-47 PerfCnt Control register field description

Domain name	position	Function 1 description	Read/write	Reset value
M	31	A value of 1 indicates that the next set of performance counters is	R	0 x1/0 x0

		implemented, otherwise it is not.		1
W.	30	A constant of 1 means PerfCnt Counter register has a bit width of 64 bits.	R	0x1
0	29..25	Read only is always 0.	0	0

¹ For Control 0 ~ Control 2, the bit reset value is 1; For Control3, the bit reset value is 0.

Domain name	position	Functional description	Read/write	Reset value
EC	24.. 23	Event category. 0: Root state event, refers to the event occurring when guestTL0.GM =0. 1: Root state mediates the event, referring to guestTL0.GM =1 and! (root.status. EXL=0 and root.status. ERL=0 and root.debug.dm =0)	R/W	0 x0
0	22.15	Read only is always 0.	0	0
The Event	14.. 5	Event number.	R/W	0 x0
IE	4	Performance counter overflow interrupt enablement. 0: This performance counter is not allowed to trigger an overflow interrupt. 1: Allows this performance counter to trigger an overflow interrupt.	R/W	0 x0
U	3	Event logging enablement bit in user mode. 0: Prohibit recording; 1: Recording is allowed.	R/W	0 x0
s.	2	Event recording enablement bit in supervisory mode. 0: Prohibit recording; 1: Recording is allowed.	R/W	0 x0
K.	1	Event logging enablement bit in core mode. 0: Prohibit recording; 1: Recording is allowed.	R/W	0 x0
EXL	0	Status.EXL=1 and status. ERL=0. 0:Prohibit recording; 1:Recording is allowed	R/W	0 x0

Figure 7-44 PerfCnt Counter register format

63

0

Event Count

Table 7-48 PerfCnt Counter register fields are described

Domain name	position	Functional description	Read/write	Reset value
Event Count	63.. 0	Performance event counters. The counter value is incremented by 1 each time the PerfCnt Control register in the same group defines an event trigger. When the highest bit of the counter is 1, the PCI position of the Cause register is 1. Although the W bit of PerfCnt Control is always defined as 1,64 bit wide Event Count field where every bit can be read and written, the actual Count range for GS464E does not exceed 48 bits. So when the Event Count is read, it's the result of the actual counter's 48-bit numeric symbol expanding to 64 bits; When the timer value is reset, only the low of Event Count is present 48 bits to be written.	R/W	0 x0

7.43 ErrCtl Register (CP0 Register 26, Select 0)

ErrCtl register is a register that can be read and written by the software. It ACTS as an interactive interface between the Index Load Tag and Index Store Tag class CACHE instructions and the Parity/ECC check value of Tag part data of all levels of the CACHE. It is also an interactive interface between the Index Load Data and the Index Store Data class CACHE instruction and the Parity/ECC check value of the Data part of all levels of the CACHE.

Figure 7-45 illustrates the format of the ErrCtl register; Table 7-49 Table 7-32 Config register field description describes the ErrCtl register fields.

Figure 7-45 ErrCtl register format

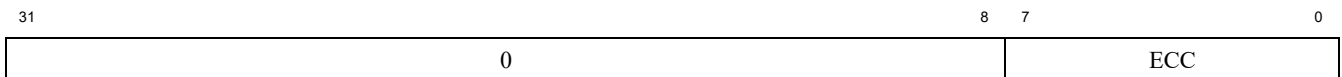


Table 7-49 Description of ErrCtl register fields

Domain name	position	Function description	Read /write	Reset value
0	31.. 8	Read only is always 0.	0	0
ECC	7.. 0	The CONTENTS of the ECC check value of the Tag or Data to be written or read.	R/W	There is no

7.44 CacheErr Register (CP0 Register 27, Select 0)

The CacheErr register records the Parity check of I-Cache and ECC check of D-Cache. GS464E for

V-cache and S-Cache also perform ECC "check 1, check 2" checks, but do not store error checks into the CacheErr register.

The CacheErr register has a different format for recording i-cache and D-cache errors. Figure 7-46 illustrates the format of the CacheErr register for i-Cache error checking. Table 7-50 Table 7-32 Config Register fields description describes each register field in this case.

Figure 7-46 Shows the format of the CacheErr register when used for i-Cache error-checking information

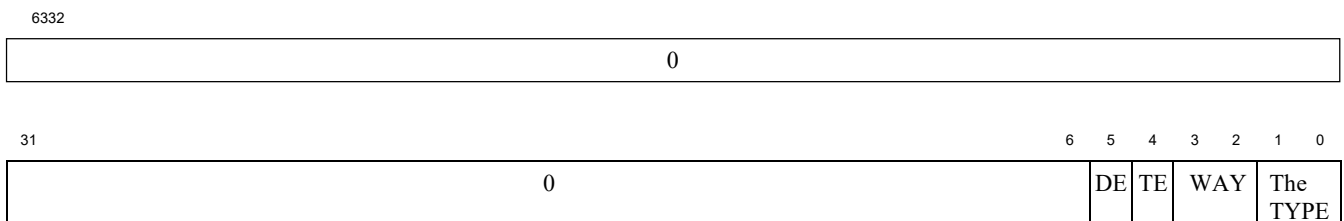


Table 7-50 Field description of the CacheErr register when used for I-Cache error-checking information

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
0	63..6	Read only is always 0.	0	0
DE	5	Error flag for parity in the I-Cache Data section. 1: Check error; 0: Check no error.	R	0
TE	4	Partial parity error flags on the I-Cache Tag. 1: Check error; 0: Check no error.	R	0
WAY	3..2	On which route did the calibration error occur?	R	0
The TYPE	1..0	Check the error type. Zero: I - Cache; 1: D - the Cache; 2, 3: reservations	R	0

Figure 7-47 illustrates the format of the CacheErr register for i-Cache error checking. Table 7-51 describes the register fields in this case.

Figure 7-47 Shows the format of the CacheErr register when used for D-Cache error-checking

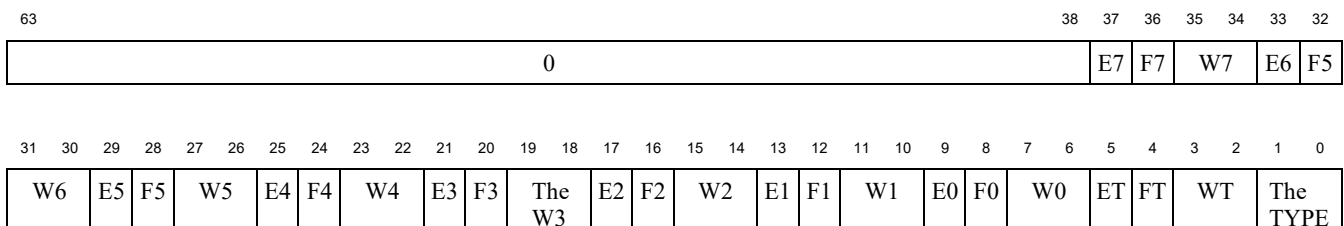


Table 7-51 Describes the fields of the CacheErr register when it is used for d-Cache error-

checking

Domain name	position	Function description	Read/write	Reset value
0	63.. 38	Read only is always 0.	0	0
E7	37	D-cache Data Bank 7 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F7	36	D-cache Data Bank 7 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W7	35.. 34	On which route did the d-Cache Data Bank 7 error occur?	R	0
E6	33	D-cache Data Bank 6 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0

Domain name	position	Function and description	Read/write	Reset value
F6	32	D-cache Data Bank 6 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W6	31.. 30	On which route did the d-Cache Data Bank 6 error occur?	R	0
E5	29	D-cache Data Bank 5 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F5	28	D-cache Data Bank 5 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W5	27.. 26	On which route did the d-Cache Data Bank 5 error occur?	R	0
E4	25	D-cache Data Bank 4 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F4	24	D-cache Data Bank 4 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W4	23.. 22	On which route did the d-Cache Data Bank 4 error occur?	R	0
E3	21	D-cache Data Bank 3 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F3	20	D-cache Data Bank 3 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
The W3	19.. 18	On which route did the d-Cache Data Bank 3 error occur?	R	0
E2	17	D-cache Data Bank 2 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F2	16	D-cache Data Bank 2 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W2	15.. 14	On which route did the d-Cache Data Bank 2 error occur?	R	0
E1	13	D-cache Data Bank 1 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F1	12	D-cache Data Bank 1 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W1	11.. 10	On which route did the d-Cache Data Bank 1 error occur?	R	0
E0	9	D-cache Data Bank 0 verifies the error id. 1: Make a mistake; 0: No mistake.	R	0
F0	8	D-cache Data Bank 0 verifies the 2-bit or more error identifier. 1: There are such mistakes; 0: No such mistakes.	R	0
W0	7.. 6	On which route did the D-Cache Data Bank 0 error occur?	R	0
ET	5	D-Cache Tag check error flag. 1: Make a mistake; 0: No mistake.	R	0
FT	4	The D-Cache Tag verifies two or more incorrect identifiers. 1: There are such mistakes; 0: No such mistakes.	R	0
WT	3.. 2	Where did the D-Cache Tag error occur?	R	0
The TYPE	1.. 0	Check the error type. Zero: I - Cache; 1: D - the Cache; 2, 3: reservations	R	0

7.45 CacheErr1 Register (CP0 Register 27, Select 1)

The CacheErr1 register is used to record the PC value of an instruction that checks for an i-cache check error, as well as the physical address of an access that checks for a D-cache check error. It should be noted that for i-cache error, the Cache block in error is not necessarily the Cache block with PC value in CacheErr1 register. For a D-cache error, the Cache block in error is not necessarily the block with the physical address in the CacheErr1 register. The information stored in CacheErr1 register can be used to extract the i-cache/D-cache Index value and the i-Cache Bank range at the wrong location. Combined with the information in CacheErr register, the error location can be accurately located.

Figure 7-48 illustrates the format of the CacheErr1 register; Table 7-52 Table 7-32 Config Register field description describes each field of CacheErr1 register.

Figure 7-48 CacheErr1 register format

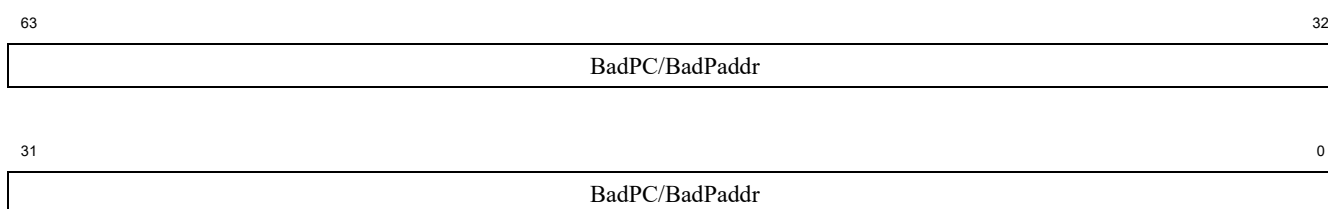


Table 7-52 CacheErr1 register field description

Domain name	position	Function and description	Read/write	Reset value
BadPC	63..0	Check the PC value of the instruction with an i-cache check error.	R	0
BadPaddr	63..0	Check the physical address of the access to which the D-cache check error occurred.	R	0

7.46 TagLo register (CP0 Register28, Select 0)

The TagLo register is a register that can be read and written by software. Together with the TagHi register, it ACTS as an interactive interface between the Index Load Tag and Index Store Tag CACHE instructions and the Tag Data of all levels of the CACHE, as well as the interface between the Index Load Data and Index Store Data instructions and the Data of all levels of the CACHE.

When TagLo is used to access different caches, the exact format is also different. Figure 7-49 illustrates the TagLo register for access

I-cache Tag format; Table 7-53 Table 7-32 Config Register fields description describes the register fields in this case. Figure 7-49 Shows the format of the TagLo register used to access the I-Cache Tag

31	12	11	7	6	5	4	3	0
TL		X	V	X	SCWAY			

Table 7-53 TagLo registers describe the fields used to access the I-Cache Tag

Domain name	position	Function and description	Read/write	Reset value
TL	31..12	Tag low-level content to be written or read, corresponding to the physical address [31:12].	R/W	There is no
X	11..7	Can write and read normally, but do not participate in other operations.	R/W	There is no
V	6	The state of a Cache block to be written or read. 0: Cache block invalid; 1: Cache block valid.	R/W	There is no
X	5..4	Can write and read normally, but do not participate in other operations.	R/W	There is no
SCWAY	3..0	Where in Scache is the Cache block to be written or read.	R/W	There is no

Figure 7-50 illustrates the format of the TagLo register when accessing the D-Cache Tag. Table 7-54 describes the register fields in this case.

Figure 7-50 Shows the format of the TagLo register used to access the D-Cache Tag

31	12	11	9	8	7	6	5	4	3	0
TL		X	W.	CS	X	SCWAY				

Table 7-54 TagLo registers describe the fields used to access the D-Cache Tag

Domain name	position	Function and description	Read/write	Reset value
TL	31..12	Tag low-level content to be written or read, corresponding to the	R/W	There is

		physical address [31:12].		no
X	11..9	Can write and read normally, but do not participate in other operations.	R/W	There is no
W.	8	The "dirty" mark of a Cache block to be written or read. 1: Dirty lump; 0: Non-dirty block.	R/W	There is no
CS	7..6	The state of a Cache block to be written or read. 0: Invalid block; 1: Shared block; 2: Exclusive block; 3: Reserved, if using the processor the result is uncertain.	R/W	There is no
X	5..4	Can write and read normally, but do not participate in other operations.	R/W	There is no
SCWAY	3..0	Where in Scache is the Cache block to be written or read.	R/W	There is no

Figure 7-51 illustrates the format of the TagLo register when accessing the V-Cache Tag. Table 7-55 shows the register fields in this case

The
descript
ion.

Figure 7-51 TagLo register is used to access the V-Cache Tag format

31	12 11 10 9 8 7 6 5 4 3	0
TL	X i W. CS X	SCWAY

Table 7-55 TagLo registers describe the fields used to access the V-Cache Tag

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
TL	31..12	Tag low-level content to be written or read, corresponding to the physical address [31:12].	R/W	There is no
X	11..10	Can write and read normally, but do not participate in other operations.	R/W	There is no
i.	9	The instruction/data property of a Cache block to be written or read. 1: Instruction block; 0: Data block.	R/W	There is no
W.	8	The "dirty" mark of a Cache block to be written or read. 1: Dirty lump; 0: Non-dirty block.	R/W	There is no
CS	7..6	The state of a Cache block to be written or read. 0: Invalid block; 1: Shared block; 2: Exclusive block; 3: Reserved, if using the processor the result is uncertain.	R/W	There is no
X	5..4	Can write and read normally, but do not participate in other operations.	R/W	There is no
SCWAY	3..0	Where in Scache is the Cache block to be written or read.	R/W	There is no

Figure 7-52 illustrates the format of the TagLo register for accessing the S-Cache Tag. Table 7-56 describes the register fields in this case.

Figure 7-52 Shows the format of the TagLo register used to access the S-Cache Tag

31	16 15	12 11 10 9 8 7 6 5	0
TL	X	PGC KP W. DS SS	X

Table 7-56 TagLo registers describe the fields used to access the S-Cache Tag

Domain name	position	Functiona l descripti on	Read /wri te	Reset value
TL	31..16	Tag low-level content to be written or read, corresponding to the physical address [31:16].	R/W	There is no
X	15..12	Can write and read normally, but do not participate in other operations.	R/W	There is no

	11-10	The PageColor bit of the Cache block to be written or read. When the system USES 4KB base page size, both digits are significant, while when the system USES 8KB base page size, only the 13th digit is significant. The relevant PageColor For a detailed description of bits, see section 5.4.5.	R/W	There is no
KP	9	On a write operation, setting 0 means clearing the directory entry of the Cache block to be written, and setting 1 means waiting The contents of the directory entry written to the Cache block remain unchanged. The field contents are meaningless when read out.	R/W	There is no
W.	8	The "dirty" mark of a Cache block to be written or read. 1: Dirty lump; 0: Non-dirty block.	R/W	There is no
DS	7	The state of the directory entry corresponding to the Cache block to be written or read. 1: Dirty directory; 0: The catalog is clean.	R/W	There is no
SS	6	The state of the Cache block to be written or read. 1: Effective block; 0: Invalid block.	R/W	There is no
X	5..0	Can write and read normally, but do not participate in other operations.	R/W	There is no

Figure 7-53 illustrates the format of the TagLo register when it is used to access the Data portion of the Cache at each level. Table 7-57 describes the register fields in this case.

Figure 7-53 The TagLo register is used for accesses
Format for each level of Cache Data

0

31



Table 7-57 TagLo registers describe the fields used to access each level of Cache Data

Domain name	position	Function and description	Read/write	Reset value
The DATA	31..0	The low 32-bit contents of the Data Bank in a Cache block to be written or read.	R/W	There is no

Programming tip: Maintain all levels of the Cache as the software fills the contents of the Cache using the Index Store Tag and Index Store Data instructions

To satisfy the Cache consistency requirements of GS464E. Otherwise the behavior of the processor will be uncertain.

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 parts of the various levels of caches, but only as interactive interfaces to the pre-encoded Data parts of the I-cache when the Index Load Data and Index Store Data class Cache instructions access the I-cache. In other cases, DataLo allows the software to read and write without participating in anything else.

Figure 7-54 illustrates the format of the DataLo register used to access the I-cache; Table 7-58 Table 7-32 Config register fields description describes the register fields in this case.

Figure 7-54 The DataLo register is in the format used for i-Cache access



Table 7-58 Describes the fields the DataLo registers are used for i-Cache access

Domain name	position	Functional description	Read /write	Reset value
ITYPE	31.. 0	The [31:0] bit of the pre-decoded information of the I-cache block to be written or read.	R/W	There is no

Programming tip: Maintain all levels of the Cache as the software fills the contents of the Cache using the Index Store Tag and Index Store Data instructions

To satisfy the Cache consistency requirements of GS464E. Otherwise the behavior of the processor will be uncertain.

7.48 TagHi Register (CP0 Register 29, Select 0)

The TagHi register is a register that can be read and written by software. Together with the TagLo register, it ACTS as an interactive interface between the Index Load Tag and Index Store Tag CACHE instructions and the Tag Data of all levels of the caches, as well as the interface between the Index Load Data and Index Store Data instructions and the Data of all levels of the caches.

Figure 7-55 illustrates the format of the TagHi register for accessing the Cache tags at each level. Table 7-59 describes the register fields in this case.

Figure 7-55 Shows the format of the TagHi register used to access the various Cache tags

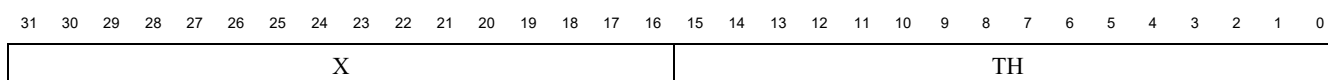


Table 7-59 Field descriptions of the TagHi registers used to access the various Cache tags

Domain name	position	Functional description	Read /write	Reset value
X	31.. 12	Can write and read normally, but do not participate in other operations.	R/W	There is no
TH	15.. 0	Tag high-level content to be written or read, corresponding to the physical address [47:32].	R/W	There is no

Figure 7-56 illustrates the format of the TagLo register for accessing each level of Cache Data. Table 7-60 describes the register fields in this case.

Figure 7-56 Shows the format of the TagHi register used to access each level of Cache Data



Table 7-60 Describes the fields used to access each level of Cache Data in the TagHi register

Domain name	position	Functional description	Read /write	Reset value
The DATA	31.. 0	The 32-bit height of the Data Bank in the Cache block to be written or read.	R/W	There is no

Programming tip: Maintain all levels of the Cache as the software fills the contents of the Cache using the Index Store Tag and Index Store Data instructions

To satisfy the Cache consistency requirements of GS464E. Otherwise the behavior of the processor will be uncertain.

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 parts of the various levels of caches, but only as interactive interfaces to the pre-encoded Data parts of the I-cache when the Index Load Data and Index Store Data class Cache instructions access the I-cache. In other cases, DataHi allows the software to read and write without participating in anything else.

Figure 7-57 illustrates the format of the DataHi register used to access i-cache; Table 7-61 Table 7-32 Config Register fields description describes the register fields in this case.

Figure 7-57 DataHi register in the format used for i-Cache access

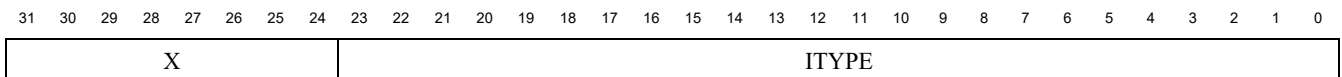


Table 7-61 DataHi registers describe the fields used for i-Cache access

Domain name	position	Functional description	Read/write	Reset value
X	31.. 24	Can write and read normally, but do not participate in other operations.	R/W	There is no
ITYPE	23.0	The [55:32] bit of the pre-decoded information of the I-cache block to be written or read.	R/W	There is no

Programming tip: Maintain all levels of the Cache as the software fills the contents of the Cache using the Index Store Tag and Index Store Data instructions

To satisfy the Cache consistency requirements of GS464E. Otherwise the behavior of the processor will be uncertain.

7.50 ErrorEPC Register (CP0 Register 30, Select 0)

The ErrorEPC register is a 64-bit read-write register that functions similar to the EPC register except that it is only used to store the INSTRUCTIONS that continue to execute after cold reset, soft reset, non-blocking interrupt, and Cache error exception processing have been completed, and the ErrorEPC register does not have an EPCLike branch delay slot identifier (Cause.BD).

In response to the above exception, the processor hardware writes to the ErrorEPC register: the PC that directly triggers the exception's instruction.

When an instruction that directly triggers an exception is in a branch delay slot, record the PC of the previous branch or jump instruction of the instruction. Figure 7-58 illustrates the format of the ErrorEPC register; Table 7-62 describes the fields of the ErrorEPC register.

Figure 7-58. ErrorEPC register format

63

0



Table 7-62 Description of ErrorEPC register fields

Domain name	position	Functional description	Read/write	Reset value
ErrorEPC	63.. 0	The PC of the instruction that continues to execute after the exception processing completes.	R/W	There is no

7.51 DESAVE Register (CP0 Register 31, Select 0)

The DESAVE register is a 64-bit read-write register used to debug exception handlers' temporary data. Typically, debugging exception handlers save a general purpose register with the DESAVE register and then use the general purpose register as a base address register for access instructions to save the current context to a specified area, such as a DMSEG segment.

The DESAVE register is not allowed in kernel mind software.

Figure 7-59 illustrates the format of the DESAVE register; Table 7-63 describes the DESAVE register fields.

Figure 7-59 DESAVE register format

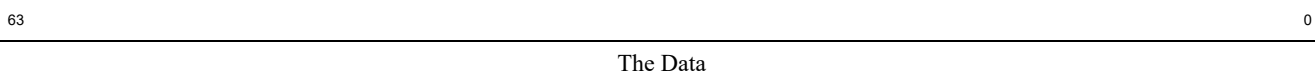


Table 7-63 Description of DESAVE register fields

Domain name	position	Functional description	Read/write	Reset value
The Data	63.. 0	Debug data stored by exception handlers.	R/W	There is no

7.52 KScratch1~6 registers (CP0 Register 31, Select 2~7)

The KScratch1~6 registers are a set of 64-bit read-write registers used to hold scratch1 ~6 temporary data for core mental software.

Software in Debug Mode cannot access Kscratch1-6, but at this point the software can scratch1-6 through the DESAVE register.

Figure 7-60 illustrates the format of the KScratch register; Table 7-64 describes the KScratch register fields.

Figure 7-60 KScratchn register format

63

0

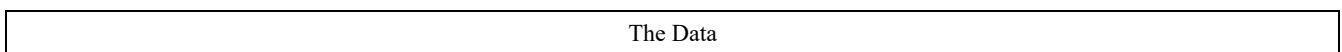


Table 7-64 KScratchn register fields are described

Domain name	position	Functional description	Read /write	Reset value
The Data	63.. 0	Kernel mentality software temporary storage data.	R/W	There is no

8 Analysis and optimization of processor performance

The performance analysis and optimization of GS464E is mainly accomplished through the performance counter of hardware integration. The performance counter is used to count the occurrence times of some events within the processor, which is an important support for the performance verification, compiler optimization and software performance tuning after silicon. In addition, since the statistical significance of certain events recorded by these performance counters is related to the power consumption of the chip at run time, performance counters can also be used to guide the power management of the chip.

The performance counter used in GS464E consists of two parts: the processor core performance counter and the Shared cache performance counter.

8.1 Organization and access method of performance counters

8.1.1 Processor core performance counter

Each PROCESSOR core of GS464E implements 28 sets of performance counters, divided equally into FETCH module, RMAP module, ROQ module, FIX module, FLOAT module, MEMORY module and CACHE2MEM module. Each module contains 4 sets of performance counters respectively. Each set of performance counters corresponds to two physical registers, one for counting (48-bit counters) and one for controlling counting. Hardware counters within each module can only be used to count events related to that module. Any set of performance counters within a module can count any performance count event associated with that module.

The Performance Counter of the GS464E processor core is still configured and accessed by the software through the CP0 Performance Counter register under the MIPS architecture, but in a slightly different form from the traditional MIPS processor. Specifically, the Performance Counter register in CP0 exists only as an interface for configuration and access. The hardware Counter to which read and write operations are delivered is dynamically determined by the event number configured in the Performance Counter register. The software will first configure an event on the Performance Counter register with an odd number, and the internal hardware of the processor will establish a one-to-one mapping relationship between this event and the hardware Counter in the module it belongs to. After the mapping relationship is established, the software reads and writes the even number Performance Counter register, then it will directly operate the mapped hardware Counter. Finally, the software starts the hardware counter to count. In particular, Performance Counter needs to initialize the Performance counters in all modules as invalid before each configuration. This is done by configuring each module with the maximum event number allowed for that module event and setting the counter value to 0.

8.1.2 Shared cache performance counters

Each GS464E Shared cache body implements four sets of 48-bit performance counters. Each set of performance counters corresponds to two physical registers, one for counting (48-bit counters) and one for controlling counting. Each performance counter can count any performance count events that occur within the Shared cache.

Performance counters of the four Shared cache bodies in the Loongson 3A3000 are accessed via the Chip's

ConfBus bus. A total of 16 sets of performance registers in the four Shared cache bodies adopt the unified Confbus base address 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	0 x0800	0 x0900	0 x0a00	0 x0b00
PerfCnt0	0 x0808	0 x0908	0 x0a08	0 x0b08
PerfCtl1	0 x0810	0 x0910	0 x0a10	0 x0b10
PerfCnt1	0 x0818	0 x0918	0 x0a18	0 x0b18
PerfCtl2	0 x0820	0 x0920	0 x0a20	0 x0b20

Register name	Scache0	Scache1	Scache2	Scache3
PerfCnt2	0 x0828	0 x0928	0 x0a28	0 x0b28
PerfCtl3	0 x0830	0 x0930	0 x0a30	0 x0b30
PerfCnt3	0 x0838	0 x0938	0 x0a38	0 x0b38

Using an example

If you want to count the total number of Sread 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), and sc0_perfcnt2 as 0 (sd \$0, 0x3ff00828)
- Step2: write sc0_perfctrl0 as 1 (sd value_1,0x3ff00800), sc_perfctrl1 as 9 (sd value_9, 0x3ff00810), and sc_perfctrl2 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); The result of T0 is sread total, and that of T1 + T2 is DMaread total.

8.2 Processor performance count events

The performance events defined by GS464E fall into three broad categories:

The first is used to analyze the characteristics of a program at the instruction set level. Specifically, the number of different types of instructions in pipeline submission phase is counted so as to obtain the distribution of instruction types in dynamic execution of the program.

The second category is used to analyze performance bottlenecks that occur when program code interacts with processor microstructures, with the goal of optimizing the program. Mainly by counting the various events that cause the pipeline to block. In addition to basic events such as Cache misses, queue full times, and branch prediction error times, GS464E also added a batch of statistics on delay cycles, such as the number of cycles in regmap pipelines-level disconnection caused by clearing the front-end pipeline after branch misprediction.

The third category is used to accumulate data for design space exploration, and the starting point is to optimize the microstructure. For example, the current dual-access memory component is implemented with full-function dual-port RAM, which is much more expensive than single-port RAM. Therefore, statistics are added on the number of collisions between the two load operations dcache RAM in the same stroke. In the existing structure, such collisions do not actually cause pipeline blocking, so it does not affect program performance.

8.2.1 Processor core performance count event definition

The performance counter event definition for the GS464E processor core is shown in Table 8-2.

Table 8-2 Processor core performance counter event definitions

Event no.	Event description
The FETCH module	
1	The number of cycles the Inst Queue is completely empty

2	Number of instructions written to Inst Queue per cycle
3	Number of front line blocking cycles (number of instructions entering Inst Queue equals 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 no.	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 cache space the number of instructions to InstQueue is less than 8 because they cross the cacheline boundary
13	The number of front line blocking cycles due to a full Inst Queue
14	Number of decoded instructions per period
15	Number of decoded instructions per cycle from the Loop Buffer
16	The number of loops to fetch the instruction from the Loop Buffer
17	Number of identified loops (both available and unavailable)
18	The number of branch instructions decoded per period is equal to 0
19	The number of branch instructions decoded per period is equal to 1
20	The number of branch instructions decoded per period is equal to 2
21	Front-end pipeline block due to Icache Miss
22	BrBTB failed to predict the front-end pipeline blockage caused by Taken Branch
24	The number of Icache misses initiated by the Icahe module and received by MISSq
26	Number of ITLB Misses but hits in TLB
27	Number of times ITLB was flushed
The RMAP module	
64	Resource allocation is blocked
65	GR renames resource full blocked
66	GR renaming resource full false block (when the incoming instruction does not require a fixed point renaming resource)
67	FR rename resource full block
68	FR renaming resource full false block (no floating-point renaming resource required for incoming instructions)
69	FCR renames resource full block
70	FCR renaming resource full false block (no need for FCR renaming resource when entering instruction)
71	ACC rename resource full block
72	ACC rename resource full false block (there is no need for ACC rename resource to enter instruction)
73	DSPCtrl renames resource full block
74	DSPCtrl rename resource full false block (there is no need for DSPCtrl to rename resource when entering command)
75	BRQ full block
76	BRQ full false blocking (there is no need to enter BRQ in order to enter)
77	FXQ full block
78	FXQ full false block (there is no need to enter FXQ in the instruction to be entered)
79	FTQ full block
80	FTQ full false blocking (there is no need to enter FTQ in order to enter)
81	MMQ full block
82	MMQ full false block (no need to enter MMQ instruction)
83	CP0Q full block
84	CP0Q full false block (no need to enter CP0Q in order to enter)

85	ROQ full block
86	Number of NOP class instructions to complete the resource allocation phase

Event no.	Event description
87	Operands emitted from the RegMap to each transmitter queue per cycle
88	Exception (not including branch error prediction) The cost of cleaning the assembly line (after the Regmap assembly line is cleared by exception until the first instruction reaches the Regmap assembly line)
89	The branch mispredicted the cost of clearing the pipeline
ROQ module	
128	Internal pipeline clock
129	Number of instructions per cycle
130	Committed ALU operation
131	The FALU operation is committed
132	Committed Memory/CP0/ floating point swap operation
133	The load operation submitted
134	Committed Store operations
135	Committed LL class operations
136	Committed SC class operations
137	Committed non-aligned load operation
138	Committed non-aligned Store operations
139	Number of exceptions and interrupts
140	Number of interrupts
141	From the time the interrupt signal is received by ROQ to the time the interrupt exception is generated
142	The first instruction from the interrupt signal received by the ROQ to the interrupt exception handler enters the ROQ
143	Number of virtual machine exceptions
144	Number of wrong address exceptions
145	TLB related exception number
146	Number of TLB Refill exceptions
147	Processing time of THE TLB Refill exception (the TLB Refill exception starts the cleanup line and returns the ERET of the TLB Refill exception)
148	Branch instructions submitted by BRQ
149	The jump Register branch instruction submitted by BRQ
150	BRQ submits jump and link branch instructions
151	BRQ submits branch and Link branch instructions
152	BRQ submits BHT branch instructions
153	Likely branch instructions submitted by BRQ
154	Branch instructions from NOT Taken submitted by BRQ
155	Branch instructions from THE TAKEN submitted by BRQ
156	BRQ submits branch instructions that predict incorrectly
157	BRQ submits an incorrect jump Register branch instruction
158	BRQ submits an incorrect prediction for the jump and link branch instruction
159	BRQ submitted branch and Link branch instructions that predicted wrong
160	BRQ submits BHT branch instructions that predict incorrectly
161	A likely branch of a BRQ submission that predicts something wrong
162	Branch instructions not taken submitted by BRQ that predicted wrong

163	The branch instructions of the TAKEN that BRQ submitted were wrong predictions
FIX module	

Event no.	Event description
192	No launch FXQ
193	FXQ emits execution operands
194	The operands emitted by FXQ to the FU0 feature
195	The operands that FXQ emits to the FU1 feature to perform
196	In FU0, the fixed-point multiplication component is in execution state
197	In FU0, the fixed-point division component is in execution state
198	In FU1, the fixed-point multiplication component is in the execution state
199	The fixed-point division component in FU1 is in execution state
FLOAT module	
256	No launch FTQ
257	FTQ emits execution operands
258	Operands performed by FTQ emitted to FU3 features
259	The operands that FTQ transmits to the FU4 feature to perform
260	FU3 is free and FU4 is full, but FTQ only has FU4 to launch
261	FU4 is free and FU3 is full, but FTQ only has FU3 to launch
262	Emits a scalar floating point operand per period
263	Number of 64-bit multimedia acceleration instructions per cycle (instruction names with "GS" prefix)
264	Number of 64-bit multimedia acceleration instructions emitted per cycle (instruction names without "GS" prefix)
272	The floating - point division/root in FU3 is in the execution state
274	The floating - point division/root in FU4 is in the execution state
The MEMORY module	
320	No launch MMQ
321	MMQ transmits execute operands
322	In MMQ, FU2 instructions are emitted per beat
323	In MMQ, FU5 instructions are emitted per beat
324	Load times
325	Store launches
326	The source operand has at least one floating-point access instruction number
327	The number of times an instruction is emitted with both fixed - point and floating - point operands
329	Number of blocks for wait_first
330	The number of cycles the SYNC operation blocks
331	Number of cycles stall_issue blocks
332	The software pre-fetch operation is launched
333	The number of times a newly emitted access operation has blocked a store operation from writing to dCache
334	There is a bank conflict between two Loads in the same beat
337	The number of times dcachewrite0 and 1 are both valid
338	Number of successful SC class instructions executed
339	Number of Store dCAhe Misses (including misses and non-EXC states)
340	The number of Dcache misses caused by the dcache Shared status of the store instruction
CACHE2MEM module	

341	Store dcache hits
-----	-------------------

Event no.	Event description
342	Load hit times
343	Fwdbus2 number
344	Fwdbus5 number
345	Total number of FWDBUS, FWdbus2 + FWdbus5
346	Number of callback operations caused by load and store address conflicts (DWaitStore)
347	Number of exceptions caused by load and store address conflicts (MISPEC)
348	The number of times cp0qhead was rolled back due to a dcacheWrite failure
349	Cp0q dMEMread request times
350	Number of Duncache requests issued by CP0Q
351	Resbus2 occupies resbus5 times, LQ, LQC1, and so on have two dest accesses
352	Software prefetch hit times in L1 Dcache
353	Store prefetch hit times in L1 Dcache
354	Number of Misses in L1 Dcache prefetched by store software
355	Load prefetch hits in L1 Dcache
356	Number of misses in L1 Dcache prefetched by LOAD software
357	The number of times store software prefetches share State in L1 Dcache
358	Specfwdbus2 number
359	Specfwdbus5 number
360	Specfwdbus frequency: SPECFWdbus2 + SpecfWdbus5
384	Number of times vCache was accessed by a data load request
385	Number of data Store requests to vCache
386	Number of times the data request accessed vCache
387	Number of times the instruction requested vcache access
388	Number of Vcache visits
389	The number of times the software prefetches vCache
390	Number of vcache load hits
391	Number of vcache Store hits
392	Number of Vcache data hits
393	Number of vcache directive hits
394	Number of Vcache hits
395	The Vcache software configuration prefetch hit times
396	Number of vcache load failures
397	Number of vCache store failures
398	Vcache data invalidation times
399	Vcache instruction invalidation times
400	Vcache invalidation times
401	Vcache software configuration prefetch invalidation times
402	The number of times that valid blocks are invalid dropped by vCache under extreQ operation
403	The number of times vcache was degraded by a WTBK operation
404	The number of times an INV operation has invalidated a vcache block

405	The number of times vcache was invalidated by INVWTBK and dropped a valid block
-----	---

Event no.	Event description
406	Number of processor core read requests to the external bus
407	Number of processor core write requests to the external bus
408	Number of bus write requests with write data
409	The bus read request is blocked because of a conflict with the bus write request address
410	Number of WTBK requests processed by MISSq
411	Missq handles the number of INVWTBK requests
412	Number of INV requests processed by MISSq
413	The number of INV class requests (the three above) that MISSq handles
414	Total number of refill (including exreq and replace+ Refill)
415	Total number of times for icache of Refill
416	Total number of times against dCache of Refill
417	The number of times the refill (replace + refill)
418	Number of times a DCache Shared block refill
419	Number of exc blocks of a Dcache refill
420	Total number of refill data (replace+ Refill)
421	Total number of refill instructions (replace+ Refill)
422	The number of times a valid block is replaced by dcache
423	The number of times a Shared block is replaced by dcache
424	The number of times dcache replaces an exC block
425	Number of times dcache replaces a dirty block
426	The number of times icache replaces 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	Number of times vcache replaces a dirty block
432	The number of times vcache replaces a useful DC block
433	The number of times vcache replaces a useful IC block
434	Accumulate the number of load requests not returned from scache per beat (missq has only 15 items at most for processing scache requests)
435	Add up the number of store requests not returned from scache per beat
436	Add up the number of finger requests not returned from scache per beat
437	The total number of SC Reads sent
438	The total number of load sent in scread
439	Total number of stores sent in Scream
440	Total number of scread data access
441	Total number of scread directive accesses
442	Scream non-prefetched total number
443	Scream total number of non-prefetched data Load
444	Total number of scread non-prefetched data Store
445	Total number of non-prefetch data access in Scream

446	Scread total number of non-prefetchable index visits
-----	--

Event no.	Event description
447	The total number of sread prefetches sent out
448	Number of load prefetch in sread sent
449	Number of store prefetch sent in Sread
450	Total number of sread prefetch data access
451	Total number of sread prefetch instruction accesses
452	Number of software prefetch requests processed by MISSq
453	The number of SCwrite emitted by MISSq
454	Number of SCwrite initiated by MISSq due to replace operation
455	The number of RESP class SCwrite issued by MISSq because of invalid operation
456	Missq is a SCwrite operation initiated by the replace operation and the replace is a valid block
457	Missq actually accepts the number of requests miss_en
458	Missq really accepts the number of load requests MISS_en
459	Missq actually accepts the number of store requests miss_en
460	The number of data accesses that MISSq actually accepts
461	The number of instruction accesses that MISSq actually receives
462	Item occupancy condition of MISSq (NON-null item of MISSq)
463	Missq normal access possession condition
464	Missq refers to the access possession condition
465	Missq external request occupancy condition
466	Missq prefetch request occupancy condition
467	The number of beats occupied by MISSq (the number of beats with valid items in MISSq, i.e., the time when MISSq is not empty)
468	Missq ordinary access to the number of item beats
469	Number of beats in MISSq to refer to the item
470	Missq external request item number of beats
471	Missq prefetch request number of item beats
472	Missq full count (MISSq cannot accept normal access, missq valid item is not less than 15)
473	The number of times the LOAD request encounters a prefetch in MISSq
474	Number of times the LOAD request encounters a prefetch pre_scref in MISSq
475	The number of times the LOAD request encounters a prefetch pre_wait in MISSq
476	The number of times the LOAD request encounters a prefetch pre_rdy in MISSq
477	The number of times the store request encounters a prefetch pre_scref and load operation in MISSq
478	Store request encounters prefetch pre_rdy and state=shard times in MISSq
479	The store request encounters prefetch pre_wait and load operations in MISSq
480	The number of times a store request encounters a prefetch pre_scref and a store operation in MISSq
481	Store request encounters prefetch pre_rdy and state=exc times in MISSq
482	The number of times a store request encounters a prefetch pre_wait and store operation in MISSq
483	Number of times store request encounters prefetch in MISSq (including hit prefetch in Store and hit prefetch in Load)
484	Number of times the store request encounters a valid prefetch in MISSq (hit prefetch in Store)
485	Number of times all requests are prefetched in MISSq (load+store)

486	Number of times all requests are prefetched in MISSq by hitting pre_screF (LAOD +store)
487	The number of times all requests are prefetched in MISSq by pre_rdy (load+store)

Event no.	Event description
488	Number of times all requests are prefetched in MISSq with pre_wait (load+store)
489	The number of times a finger request encounters a prefetch in MISSq
490	Number of times the fetch request encounters prefetch in MISSq by prescref
491	The number of times the finger request encounters prefetch pre_rdy in MISSq
492	The number of times the fetch request encounters pre_wait prefetch in MISSq
495	The number of times the data and fetchfinger are prefetched by PRE_rdy in MISSq
496	The number of times the data and fetching point encounters pre_wait prefetch in MISSq
497	Number of times the hardware Load prefetch request is cancelled by Scache ¹⁵
498	Number of times the hardware Store prefetch request is cancelled by Scache
499	Number of times the hardware data access prefetch request is cancelled by Scache
500	Hardware fetch refers to the number of Scache cancels for a prefetch request
501	Number of times the hardware prefetch request is cancelled by Scache
502	The number of hardware load prefetches
503	Number of hardware Store prefetches
504	The number of hardware data access prefetches
505	Hardware fetch refers to the number of prefetches
506	Number of hardware prefetches
507	Tagged triggered load pre-fetch number
508	The number of load prefetches triggered by MISS
509	Number of store pre-fetches triggered by Tagged
510	The number of store prefetches triggered by Miss
511	Number of pre-fetches of data access triggered by TAGGED
512	The number of pre-fetching data triggered by MISS
513	Number of pre-fetches of tagged triggered instructions
514	The number of prefetch instructions triggered by MISS
515	Number of pre-fetches tagged triggers
516	The number of prefetches triggered by MISS
517	The number of load prefetches accepted by MISSq
518	The number of store prefetches accepted by Missq
519	The number of data access prefetches accepted by MISSq
520	The number of prefetches of instructions accepted by MISSq
521	Number of prefetters accepted by MISSq (repeat requests do not enter MISSq)
522	The number of valid load prefetches back from scache
523	Number of valid store prefetches returned from scache
524	Number of valid data access prefetches returned from scache
525	Number of prefetches of instructions coming back from scache
526	Effective prefetch from scache (pre_scref->rdy pre_scref->pre_rdy)
527	The number of load prefetters that can enter PRE_rdy
528	The number of prefetchings from the store that can enter pre_rdy
529	The number of data access prefetchings that can enter PRE_rDY

Event no.	Event description
530	Prefetching number of instructions that can enter PRE_rdy
531	Number of prefetchings that can enter pre_rdy (pre_scref-> pre_rdy)
532	Accumulate the number of load prefetch requests per beat in PRE_rdy
533	Accumulate the number of store prefetch requests in PRE_Rdy per beat
534	Add up the number of data access prefetch requests in PRE_Rdy per beat
535	Add up the number of prefetch requests per beat in PRE_Rdy
536	Add up the number of pre_rdy requests per beat
537	Accumulate the number of prefetches per beat that are in pre_scref and hit by the normal load request
539	Accumulate the number of prefetches per beat that are in pre_scref and hit by normal store requests
540	Accumulate the number of prefetches per beat that are in pre_scref and hit by the normal data access request
541	Accumulate the number of prefetches per beat that are in pre_scref and are hit by normal fetches
542	Accumulate the number of prefetches per beat that are in pre_scref and hit by normal access
543	The number of prefetches hit is accessed by load in the pre_scref state
544	The number of prefetches that are hit by store access in the pre_scref state
545	The number of prefetches that are hit by data access in the pre_scref state
546	In the prescref state the number of prefetches to access the hit is referred to
547	The number of prefetches hit in the pre_scref state, that is, the number of rdy from pre_scref->
548	Number of loads from the pre_scref state back to the MISS state
553	The number of prefetches of hits that are accessed by load in the pre_wait state
554	The number of prefetches that are store-accessed hits in the pre_wait state
555	The number of prefetchings that are hit by data access in the pre_wait state
556	In the pre_wait state, the number of prefetchings to the hit is referred to
557	The number of prefetchings hit in pre_wait state, that is, the number of pmisses from pre_wait ->
558	Prefetch item in pre_wait state that is replaced because MISSq cannot accept normal access
559	Prefetch item in pre_rdy state that is replaced because MISSq cannot accept normal access
560	The number of times the prefetch item is INV
561	Accumulate the prefetching of each load
562	Accumulates whether this load prefetches an item
563	Accumulates the prefetching item of each auction store
564	Whether the total prefetch of this auction store occupies the item
565	Accumulate the prefetching item of each beat
566	Whether to prefetch the item of the accumulative auction data
567	Accumulates the precapture of each beat
568	Whether the cumulative prefetching refers to the item of prefetching
569	Cumulative load prefetch number of hits in pre_scref and PRE_RDY
570	Cumulative store prefetch number of hits in pre_scref and PRE_Rdy
571	Cumulative number of prefetched data hits in PRE_scref and PRE_RDY
572	Cumulative number of prefetches in prescref and PRE_RDY
573	Cumulative number of prefetches in PRE_SCREF and PRE_RDY

8.2.2 Shared cache performance count event definition

Table 8-3 Shared cache performance counter event definitions

Event no.	Event description
0	Since there is no switch marker in CTRL, a configuration of 0 indicates that it is not on.
1	Number of requests received for all requests
2	Number of cachable requests received
3	Receive all non-DMA operations of cachable's request request
4	Received all non - fetching attributes of the Cachable request request
5	All REQ_READ requests received for cached
6	All REQ_WRITE requests received for cached
7	REQ_READ for cached received for all non-prefetched properties
8	REQ_WRITE for all non-prefetched properties received from cached
9	All DMAREAD requests received for cached
10	All DMAWRITE requests received for cached
11	Received all of uncached's DMAREAD
12	Received all DMAWRITE of Uncached
13	Number of ALL DMA requests received
14	All the types you'll receive will be the number of requests for scache_prefix
15	The number of prefetches accepted by the Scache hardware itself
16	All received types are the number of requests for STORE_fill_full
17	All responses received for conformance requests
18	The number of all responses received for consistency with dirty data
19	A write back to the previous cache active replacement received
20	The number of higher-level active substitutions received with dirty data
21	Number of read requests to Memory caused by cached access
22	Number of write requests to Memory caused by cached access
23	The number of times the scache result is Miss
24	The query scache result is the number of all reqreads of Miss
25	Query the scache result as the number of all REqwrites of Miss
26	Query scache for the number of scache hits but pagecolor does not hit
27	REQREAD queries scache, resulting in a hit in a clean block
28	REQREAD queries scache, resulting in a hit in someone else's EXC block
29	REQWRITE queries scache, and the result hits the clean block
30	REQWRITE query scache, result hit in someone else's EXC block
31	A WRITE query scache results in a block that is being Shared by multiple cores
32	Cached DMAREAD queries scache and hits
33	Cached DMAWRITE queried scache, did not hit
34	Cached DMAWRITE queries scache with an INVALIDATION of the CPU
35	Number of INV requests made to the CPU for consistency
36	Number of WTBK requests made to the CPU for consistency
37	The number of INVWTBK requests made to the CPU for consistency

