Godson 3B1500 Processor User Manual

Volume II

GS464 processor core V1.3

2016 Nian ***4*** Yue

Godson Technology Co., Ltd.

Copyright Notice

Disclaimer

This document only provides periodic information, and the contents can be updated at any time according to the actual situation of the product without notice. Such as

The company does not assume any responsibility for direct or indirect losses caused by improper use of documents.

Godson Technology Co., Ltd.

Loongson Technology Corporation Limited

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

Building No.2, Loongson Industrial Park,

Zhongguancun Environmental Protection Park, Haidian District, Beijing

Phone (Tel): 010-62546668

Fax: 010-62600826

Read the guide

"Longxin 3B1500 Processor User Manual" is divided into the first volume, the second volume, and software programming guide.

Godson 3B1500 Processor User Manual Booklet introduces Godson 3B1500 multi-core processor architecture, mainly including multi-core

Processor architecture and register description;

"Godson 3B1500 Processor User Manual" Book II, a detailed introduction from the perspective of system software developers

GS464 high-performance processor core used;

"Godson 3B1500 Processor User Manual" software programming guide introduces common issues in BIOS and operating system development

problem.

Page 4

revise history

Document update records

Document name:	Godson 3B1500 processor user hands book --Volume II
version number	V1.3
founder:	Chip R & D
Creation date:	2016-04-01

Update history

Serial number	Updated	updater	version number	update content
1	2011-06-17	R & D Center	V1.0	preliminary draft completed
2	2012-05-02	R & D Center	V1.1	Fixed bugs related to 48-bit address space in Chapter 5, "Memory Management" error
3	2015-10-09	Chip R & D Department	V1.2	Revised the description of instructions in Chapter 2 and Chapter 7
4	2016-04-01	The chip research and development department	V1.3	adjusts some instructions according to LS3B1500G

Manual feedback: service@loongson.cn

table of Contents

I

Page 6

II

Page 7

Page 9

Map directory

Table of contents

x

Page 15

XI

Structure overview

GS464 is a general-purpose RISC processor IP that implements the 64-bit MIPS64 instruction set. GS464 instruction pipeline
Each clock cycle takes four instructions to decode, and dynamically sends them to five full-flow functional units. Although the instructions are in
Out-of-order execution is performed under the premise of ensuring dependencies, but the instructions are submitted in the original order of the program to ensure accurate examples
Out and fetch sequentially.

The four-shot superscalar structure makes the instruction and data related issues in the instruction pipeline very prominent. GS464 uses out of order
Execution technology and aggressive storage system design to improve pipeline efficiency.

Out-of-order execution techniques include register renaming techniques, dynamic scheduling techniques, and branch prediction techniques. Register renaming solution
WAR (read-after-write) is related to WAW (write-after-write) and is used for accurate on-site recovery caused by exceptions and false branch prediction,
GS464 performs fixed-point renaming through a 64-item physical register file. Dynamic scheduling based on the number of times the instruction operands are ready
Order instead of the order in which the instructions appear in the program to execute instructions, reducing blocking caused by RAW (read after write), GS464
There is a 16-term fixed-point reservation station and a 16-term floating-point reservation station for out-of-order transmission, and a 64-term
The Reorder queue (referred to as ROQ) implements instructions that are executed out of order and submitted in the order of the program. Branch prediction
Whether the shift instruction successfully jumped to reduce the control-related blocking. The GS464 uses 16-bit branch target address buffers.
Branch Target Buffer (BTB), Branch History Table for 2K entries,
(BHT for short), 9-bit Global Histiry Registor (GHR for short), and return of 4 items
The address stack (Return Address Stack, RAS for short) performs branch prediction. GS464 has two 64-bit full pipeline
Point feature.

GS464 advanced storage system design can effectively improve the efficiency of the pipeline. GS464 Level 1 Cache consists of 64KB
Instruction cache (four-way group associative) and 32KB data cache (two-way group associative). GS464 TLB has 64
The items adopt a fully associative structure, and each item can map an odd page and an even page. The page size is variable from 4KB to 16MB.
GS464 dynamically resolves address dependencies through a 16-item fetch queue and an 8-item fetch queue.
Lai, to achieve out-of-order execution of memory access operations, non-blocking cache, load instruction speculation execution (Load Speculation), etc.

Fetch optimization technology. GS464 supports 128-bit fetch operation, and its virtual address and physical address are 48 bits.

GS464 supports MIPS company's EJTAG debugging specification, adopts standard AXI interface, and its instruction cache implements

Parity check, data cache implements ECC check. The above characteristics can increase the applicable scope of Godson-3.

The basic pipeline of GS464 includes instruction fetch, pre-decoding, decoding, register renaming, scheduling, transmitting, reading registers,

Execution, submission and other 9 levels, each level of the pipeline includes the following operations.

1

Page 17

◆ The instruction pipeline uses the value of the program counter PC to access the instruction cache and instruction TLB. If the instruction cache

And instruction TLB are hit, then four new instructions are fetched into the instruction register IR.

◆ The pre-decoding pipeline stage decodes the branch instruction and predicts the direction of the jump.

◆ The decoding pipeline converts the four instructions in IR into the internal instruction format of GS464 and sends them to the register renaming module.

Piece.

◆ Register renaming pipeline allocates a new physical register for the logical target register and registers the logical source

The device is mapped to the physical register that was most recently assigned to it.

◆ Scheduling pipeline assigns renamed instructions to fixed-point or floating-point reservation stations for execution, and sends them to ROQ at the same time

For sequential submission after execution; in addition, branch instructions and fetch instructions are also sent to the transfer queue and fetch

Store queue.

◆ The launch pipeline selects one of each function from the fixed-point or floating-point reservation station. All operands are ready.

Instructions; instructions not ready for operands when renaming, by listening to the result bus and forward bus, etc.

Wait for its operands to be ready.

◆ The read register pipeline is the instruction issued by the upper-level water stage to read the corresponding source operand from the physical register file

To the corresponding functional components.

◆ The execution pipeline executes the instruction according to the instruction type and writes the calculation result back to the register file; the result bus is also sent to

The station and register renaming tables are reserved to inform the corresponding register values that they are ready for use.

◆ The submission pipeline submits the executed instructions in the order of the programs recorded in the Reorder queue.

Four more instructions can be submitted per shot, and the submitted instructions are sent to the register rename table for confirmation of its purpose.

Device renames the relationship and releases the physical register originally allocated to the same logical register, and sends it to the fetch queue

Allow those committed instructions to be written to the cache or memory.

The above is the pipeline of basic instructions. For some more complex instructions, such as fixed-point multiply-divide instructions, floating-point instructions,

Fetching instructions requires multiple shots during execution. The basic structure of GS464 is shown in the figure below.

2

Page 18

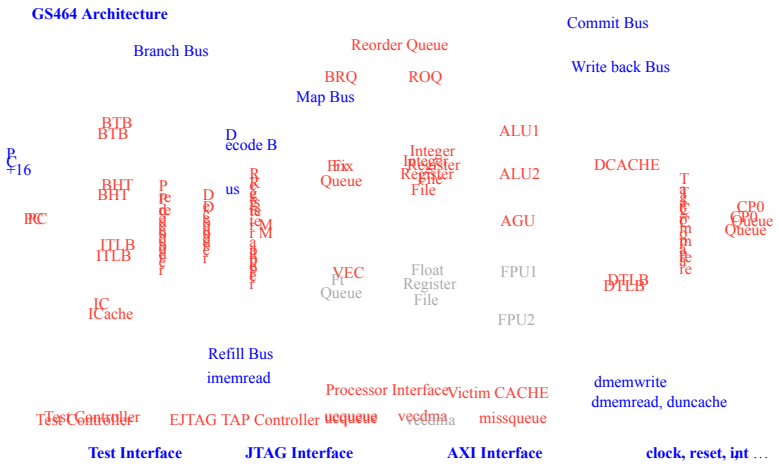


Figure 1-1 Basic structure of GS464 processor

3

Page 20

2. Overview of Godson GS464 processor core instruction set

The Loongson GS464 processor core is compatible with the MIPS64 R2 architecture, which implements all the definitions of the MIPS64 R2 specification. Set of required instructions, as well as some custom extension instructions.

Godson GS464 core processor implemented in the MIPS64 R2 compatible instructions include 2 section 5.1 in. For reasons of space, The detailed definitions of these instructions are not given in this document. Readers do need to know the detailed definition of the relevant directives

MIPS Architecture Specification Volume I and Volume II version 2.50. The implementation-related content in this part of the instructions will be in section 2.2

As explained in the article, the implementation of several other MIPS64 R2 instructions GS464 processor core is not the same as the MIPS architecture specification.

This is also explained in section 2.2 .

Godson custom extension instructions implemented by the processor core GS464 recited in 2. Section 1.3. For reasons of length, this one

The detailed definition of the sub-instruction is not given in this document. Readers do need to know the detailed definitions of the relevant directives.

Instruction System Manual. The Loongson Instruction System Manual is currently only available to authorized customers.

2.1. MIPS64 Compatible Instruction List

According to the instruction function, the MIPS64 compatible instructions implemented by the GS464 processor core can be divided into the following groups:

◆ Fetch instruction

◆ Operation instructions

◆ Branch and jump instructions

◆ Coprocessor instructions

◆ Other instructions

2.1.1. Fetch instruction

The MIPS architecture uses a load / store architecture. All operations are performed on registers, and only fetch instructions can

Access to stored data. Fetch instructions include reading and writing of various widths of data, unsigned reads, non-aligned fetches and atomic fetch

Save, etc.

Table 2-1 CPU instruction set: fetch instruction

Instruction mnemonics	Brief instruction function	ISA compatibility level
LB	Fetch bytes	MIPS32

5

Page 21

Instruction mnemonics	Brief instruction function	ISA compatibility level
LBU	Take unsigned bytes	MIPS32
LH	Take half word	MIPS32
LHU	Take unsigned halfword	MIPS32
LW	Take word	MIPS32
LWU	Take unsigned word	MIPS32
LWL	Take the left part of the word	MIPS32
LWR	Take the right part	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	Double word address	MIPS64
SB	Save byte	MIPS32
SH	Save half a word	MIPS32
SW	Save word	MIPS32
SWL	Left word	MIPS32

SWR	Save the word right	MIPS32
SD	Double word	MIPS64
SDL	Save double word left	MIPS64
SDR	Deposit double word right	MIPS64
SC	Saved under conditions	MIPS32
SCD	Double word	MIPS64

2.1.2. Operation Instructions

Operational instructions perform arithmetic, logic, shift, multiplication, and division operations on register values. Operational instructions include Register instruction format (R-type, operands and operation results are stored in registers) and immediate instruction format (I-type, One of the operands is a 16-bit immediate)

Table 2-2 CPU instruction set: arithmetic instructions (ALU immediate value)

Instruction mnemonics	Brief instruction function	ISA compatibility level
ADDI	Add immediate	MIPS32

6

Instruction mnemonics	Brief instruction function	ISA compatibility level
DADDI	Add double word immediate	MIPS64
ADDIU	Add unsigned immediate	MIPS32
DADDIU	Add unsigned doubleword immediate	MIPS64
SLTI	Less than immediate setting	MIPS32
SLTIU	Unsigned less than immediate setting	MIPS32
ANDI	With immediate	MIPS32
ORI	Or immediate	MIPS32
XORI	XOR immediate	MIPS32
LUI	Take immediate value to high	MIPS32

Table 2-3 CPU instruction set: arithmetic instructions (3 operands)

Instruction mnemonics	Brief instruction function	ISA compatibility level
ADD	plus	MIPS32
DADD	Double word plus	MIPS64
ADDU	Unsigned plus	MIPS32
DADDU	Unsigned double word plus	MIPS64
SUB	Less	MIPS32
DSUB	Double word minus	MIPS64
SUBU	Unsigned minus	MIPS32
DSUBU	Unsigned Double Word Subtraction	MIPS64
SLT	Less than setting	MIPS32
SLTU	Unsigned less than set	MIPS32
AND	versus	MIPS32
OR	or	MIPS32
XOR	XOR	MIPS32

NOR	Or not	MIPS32

Table 2-4 CPU instruction set: arithmetic instructions (2 operands)

Instruction mnemonics	Brief instruction function	ISA compatibility level
CLO	Word leading 1 count	MIPS32
DCLO	Double word leading 1 count	MIPS64

Page 23

CLZ	Word leading 0 count	MIPS32
DCLZ	Double word leading 0 count	MIPS64
WSBH	Halfword byte swap	MIPS32 R2
DSHD	Word and word exchange	MIPS64 R2
SEB	Byte sign extension	MIPS32 R2
SEH	Half-character extension	MIPS32 R2
INS	Bit insertion	MIPS32 R2
EXT	Bit extraction	MIPS32 R2
DINS	Double word 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 CPU instruction set: multiplication and division instructions

Instruction mnemonics	Brief instruction function	ISA compatibility level
MUL	Multiply to general register	MIPS32
MULT	Multiply	MIPS32
DMULT	Doubleword multiplication	MIPS64
MULTU	Unsigned multiplication	MIPS32
DMULTU	Unsigned doubleword multiplication	MIPS64
MADD	Multiply	MIPS32
MADDU	Unsigned multiply-add	MIPS32
MSUB	Multiplication	MIPS32
MSUBU	Unsigned multiplication and subtraction	MIPS32
DIV	except	MIPS32
DDIV	Double word division	MIPS64
DIVU	Unsigned division	MIPS32
DDIVU	Unsigned doubleword division	MIPS64
MFHI	Fetch from register hi to general register	MIPS32
MTHI	Register from general register to hi register	MIPS32

MFLO	Fetch from lo register to general register	MIPS32
MTLO	Store from general register to lo register	MIPS32

Table 2-6 CPU instruction set: shift instruction

Instruction mnemonics	Brief instruction function	ISA compatibility level
SLL	Logical left shift	MIPS32
SRL	Logical right shift	MIPS32
SRA	Arithmetic right shift	MIPS32
SLLV	Variable logical left shift	MIPS32
SRLV	Variable logical shift right	MIPS32
SRAV	Variable arithmetic right shift	MIPS32
ROTR	Cycle right	MIPS32 R2
ROTRV	Variable loop right	MIPS32 R2
DSLL	Double word logic left shift	MIPS64
DSRL	Double word logic shift right	MIPS64
DSRA	Double word arithmetic shift right	MIPS64
DSLLV	Variable doubleword logic left shift	MIPS64
DSRLV	Variable double word logic right shift	MIPS64
DSRAV	Variable double word arithmetic right shift	MIPS64
DSLL32	Double word logic left shift +32	MIPS64
DSRL32	Double word logic right shift +32	MIPS64
DSRA32	Double word arithmetic right shift +32	MIPS64
DROTR	Double word loop right	MIPS64 R2
DROTR32	Double word loop right shift +32	MIPS64 R2
DROTRV	Double word variable right shift	MIPS64 R2

2.1.3. Branch and jump instructions

Branch and jump instructions change the control flow of a program and include the following four types:

◆ PC relative condition branch

◆ PC unconditional jump

◆ Register absolute jump

9

◆ Procedure call

In the MIPS definition, all branch instructions are followed by a delay slot instruction. Likely branch instruction delay slot only Executed when the branch is successful, non-Likely branch instruction delay slot instructions are always executed. Return address of procedure call instruction It is saved in the 31st register by default, and the jump according to the 31st register will be considered as returning from the called procedure.

Table 2-7 CPU instruction set: branch and branch instructions

Instruction mnemonics	Brief instruction function	ISA compatibility level
J	Jump	MIPS32
JAL	Immediate procedure	MIPS32
JR	Jump to the instruction pointed to by the register	MIPS32
JR.HB	Jump to the instruction pointed to by the register	MIPS32 R2
JALR	Register calling process	MIPS32
JALR.HB	Register calling process	MIPS32 R2
BEQ	Jump if equal	MIPS32
BNE	Jump if not waiting	MIPS32
BLEZ	Jump to less than or equal to 0	MIPS32
BGTZ	Greater than 0 jump	MIPS32
BLTZ	Less than 0 jump	MIPS32
BGEZ	Greater than or equal to 0 jump	MIPS32
BLTZAL	Less than 0 call procedure	MIPS32
BGEZAL	Greater than or equal to 0 calling procedure	MIPS32
BEQL	Likely Jump	MIPS32
BNEL	Likely Jump	MIPS32
BLEZL	Less than or equal to 0 then Likely jumps	MIPS32
BGTZL	Greater than 0 Likely jump	MIPS32
BLTZL	Less than 0 Likely jump	MIPS32
BGEZL	Greater than or equal to 0 then Likely jumps	MIPS32
BLTZALL	Less than 0 Likely call procedure	MIPS32
BGEZALL	Greater than or equal to 0 then Likely calls the procedure	MIPS32

10

Page 26

2.1.4. Coprocessor Instructions

Coprocessor instructions complete operations inside the coprocessor. The Godson GS464 processor core has two coprocessors: No. 0 Processor (system processor) and coprocessor number 1 (floating point coprocessor).

Coprocessor 0 (CP0) manages memory and handles exceptions through the registers of CP0. These instructions are listed in Table 2-8 in.

The MIPS architecture specification clearly defines floating-point instructions in coprocessor number 1 (CP1) instructions. Godson The specific implementation of the floating-point instruction in the coprocessor 1 instruction in the GS464 processor core will be the first error **!** No reference source found. Chapters are introduced separately.

Table 2-8 CPU instruction set: CP0 instruction

Instruction mnemonics	Brief instruction function	ISA compatibility level
DMFC0	Doubleword from CP0 register	MIPS64
DMTC0	Write double word to CP0 register	MIPS64
MFC0	Fetch from CP0 register	MIPS32

MTC0	Write to CP0 register	MIPS32
TLBR	Read indexed TLB entries	MIPS32
TLBWI	Write indexed TLB entries	MIPS32
TLBWR	Write random TLB entries	MIPS32
TLBP	Searching for matches in the TLB	MIPS32
CACHE	Cache operation	MIPS32
ERET	Abnormal return	MIPS32
DI	Disable interrupt	MIPS32 R2
EI	Allow interrupts	MIPS32 R2

2.1.5. Other instructions

In MIPS64, in addition to the above-mentioned instructions, there are other instructions, see Ta ɔr details :

Table 2-9 CPU instruction set: special instructions

Instruction mnemonics	Brief instruction function	ISA compatibility level
SYSCALL	System call	MIPS32

11

BREAK	Breakpoint	MIPS32
SYNC	Synchronize	MIPS32
SYNCI	Synchronous instruction cache	MIPS32 R2

Table 2-10 CPU instruction set: exception instruction

Instruction mnemonics	Brief instruction function	ISA compatibility level
TGE	Greater than or equal to fall into	MIPS32
TGEU	Unsigned number is greater than or equal to trapped	MIPS32
TLT	Less than trapped	MIPS32
TLTU	Unsigned number is less than trapped	MIPS32
TEQ	Tantamount to fall into	MIPS32
TNE	Not waiting to fall into	MIPS32
TGEI	Greater than or equal to immediate	MIPS32
TGEIU	Greater than or equal to unsigned immediate	MIPS32
TLTI	Less than immediate	MIPS32
TLTIU	Less than unsigned immediate	MIPS32
TEQI	Equal to immediate	MIPS32
TNEI	Not equal to immediate	MIPS32

Table 2-11 CPU instruction set: conditional move instruction

Instruction mnemonics	Brief instruction function	ISA compatibility level
MOVF	Conditional move when floating point condition is false	MIPS32
MOVN	Conditional move when general register is non-zero	MIPS32
MOVT	Conditional move when floating point condition is true	MIPS32
MOVZ	Conditional move when general register is 0	MIPS32

Table 2-12 CPU instruction set: other instructions

Instruction mnemonics	Brief instruction function	ISA compatibility level
PREF	Prefetch instruction	MIPS32
PREFX	Prefetch instruction	MIPS32
NOP	No operation	MIPS32
SSNOP	Single launch no operation	MIPS32

12

2.2. MIPS64 Compatible Instructions Implementation Instructions

2.2.1. There are instructions for implementation differences

The Loongson GS464 processor checks all MIPS64 R2 instructions, but supports some implementation-related instructions

Redefined, see Table 2-13 .

Table 2-13 Instructions with implementation differences

Instruction mnemonics	Brief instruction function	Specific implementation description
PREF	Prefetch instruction	Treated as NOP instruction processing, no prefetch effect. Software prefetching can be implemented through Load to Register 0.
PREFX	Prefetch instruction	Treated as NOP instruction processing, no prefetch effect. Software prefetching can be implemented through Load to Register 0.
SSNOP	Single launch no operation	Treated as a NOP instruction. All data related and control related (CP0 Hazards) in GS464 are maintained by hardware, No software control is required.
EHB	Isolated execution related	Treated as a NOP instruction. All data related and control related (CP0 Hazards) in GS464 are maintained by hardware, No software control is required.
WAIT	Enter wait state	Treated as NOP instruction processing, without stopping pipeline effect. Avoid using this instruction in code involving low power management The device design is expected to even cause increased power consumption.
RDPGPR	Read shadow register	GS464 does not implement shadow registers, so both the source and destination registers are To the current register set.
WRPGPR	Write shadow register	GS464 does not implement shadow registers, so both the source and destination registers are To the current register set.
SYNC	Synchronize	GS464 only implements the SYNC instruction with stype = 0. When stype is other value, it will report The exception is reserved instructions. This instruction acts as a memory barrier and is used to Ensure that the fetch operation before SYNC is determined to be complete (eg, the data of the store instruction Write to dcache, read and write of uncached is completed, load has retrieved value to register And at the same time ensure that the fetch operation following the SYNC instruction has not yet started. SYNC The instruction requires CU [0], only kernel mode can be used.

13

Page 29

Instruction mnemonics	Brief instruction function	Specific implementation description
SYNCI	Synchronous instruction cache	The Godson 3B1500 processor maintains a level 1 instruction cache and a level 1 data cache by hardware Data consistency between the two, so the implementation of the SYNCI instruction is adjusted. In the existing implementation, the execution effect produced by the SYNCI instruction is equivalent to that of the SYNC instruction. The address information carried by the SYNCI instruction is ignored, so the TIB related example will no longer be triggered. Exceptions, address error exception, and cache error exception. Since the SYNCI instruction can Under the user mode, the software can use this feature when necessary, The SYNC instruction has the same effect as the memory barrier.

14

Page 30

Instruction mnemonics	Brief instruction function	Specific implementation description
		The differences between the CACHE instruction implemented by GS464 and the MIPS64 specification are mainly There are two points: **. 1** , **Index** class **CACHE** address instruction analytically

The index CACHE instruction implemented by GS464 uses the lowest 2 bits of the virtual address as the Select signal for the first cache, instead of the slave as defined in the MIPS64 specification Interception of virtual address.

2 , **CACHE28** , **CACHE29** , **CACHE30** , **CACHE31** instructions Righteousness

In the processor, CACHE28, CACHE29, CACHE30, and CACHE31 refer to The meaning of the command (that is, the Cache instruction of op [4: 2] = 0b111) is different from the MIPS64 specification. These instructions in the Godson processor are Index Store Data operations, while MIPS64 regulations Fanzhong are Fetch and Lock operations.

CACHE cache operation

The types of Cache operations available in the GS464 processor core are listed below:

op [4: 0]	Function description
b00000	Invalid I-Cache row based on index
b01000	Write I-Cache Row Tag Based on Index
b11100	Write I-Cache Row Data Based on Index
b00001	Invalidate based on index and write back to D-Cache row
b00101	Read D-Cache Row Tag by Index
b01001	Write D-Cache Row Tag Based on Index
b10001	Invalid D-Cache line based on hit
b10101	Invalid based on hit and write back to D-Cache line
b11001	Read D-Cache Row Data by Index
b11101	Write D-Cache Row Data by Index
b00011	Invalidate based on index and write back to S-Cache row
b00111	Read S-Cache Row Tag by Index
b01011	Write S-Cache Row Tag Based on Index
b10011	Invalid based on hit and write back to S-Cache line
b11011	Read S-Cache Row Data by Index
b11111	Write S-Cache Row Data Based on Index

15

2.2.2. Disable instructions

The GS464 processor core disables the following instructions, see Table 2-14 :

Table 2-14 Disable instructions

Instruction mnemonics	Brief instruction function	ISA compatibility level
DI	Disable interrupt	MIPS32 R2
EI	Allow interrupts	MIPS32 R2

2.3. Custom Extension Instructions

The custom extended instructions implemented by the Godson GS464 processor are divided into the following categories by function:

◆ Fetch instruction

◆ Multiplication and division instructions

◆ **X86** binary acceleration instructions

◆ Miscellaneous instructions

2.3.1. Custom extended fetch instructions

Table 2-15 Custom extended fetch instructions

Instruction mnemonics	Brief instruction function
GSLE	If the address is less than or equal to the exception
GSGT	Exception if greater than set address error
GSLBLE	Take bytes with out-of-bounds check
GSLBGT	Fetching bytes with lower out-of-bounds check
GSLHLE	Take the halfword of the cross-border check
GSLHGT	Halfword with lower cross-boundary check
GSLWLE	Take the word out of bounds check
GSLWGT	Take word with lower cross-boundary check
GSLDLE	Take double word with cross-boundary check
GSLDGT	Take double word with lower cross-boundary check

Instruction mnemonics	Brief instruction function
GSLQ	Fixed-point quadword
GSLBX	Fetch byte with offset
GSLHX	Halfword with offset
GSLWX	Word fetch with offset
GSLDX	Doubleword with offset
GSSBLE	Store bytes with out-of-bounds check
GSSBGT	Stored bytes with out-of-bounds check
GSSHLE	Bring a halfword of cross-boundary inspection
GSSHGT	Halfword
GSSWLE	Save the word with cross-boundary check
GSSWGT	Saved words with lower cross-boundary check
GSSDLE	Save double word with cross-boundary check
GSSDGT	Double word
GSSQ	Fixed-source quad-word
GSSBX	Stored bytes with offset
GSSHX	Halfword with offset
GSSWX	Stored word with offset
GSSDX	Stored double word with offset

2.3.2. Custom extended operation instructions

Table 2-16 Custom extended multiplication and division instructions

Instruction mnemonics	Brief instruction function
GSMULT	Signed word multiplication, write result to general register
GSDMULT	Signed doubleword multiplication, write result to general register
GSMULTU	Unsigned word multiplication, write result to general register
GSDMULTU	Unsigned doubleword multiplication, write result to general register
GSDIV	Signed word division, quotient write general register
GSDDIV	Signed doubleword division, quotient write general register
GSDIVU	Unsigned word division, quotient write general register

17

Page 33

Instruction mnemonics	Brief instruction function
GSDDIVU	Unsigned double word division, quotient write general register
GSMOD	Signed word division, remainder written to general register
GSDMOD	Signed doubleword division
GSMODU	Unsigned word division, remainder written to general register
GSDMODU	Unsigned double word division, the remainder is written to the general register
GSXOR	XOR of fs and ft
GSNOR	fs and ft logical OR
GSAND	fs and ft logical bitwise AND
GSADDU	fs and ft fixed-point unsigned word addition
GSOR	fs and ft fixed-point logical bit OR
GSADD	fs and ft fixed-point plus
GSDADD	fs and ft fixed-point doubleword addition
GSSEQU	Equal comparison of fs and ft fixed-point numbers
GSSEQ	Equal comparison of fs and ft fixed-point numbers
GSSUBU	fs and ft fixed-point unsigned word subtraction
GSSUB	fs and ft fixed-point subtraction
GSDSUB	fs and ft fixed-point doubleword subtraction
GSSLTU	fs and ft fixed-point unsigned fixed-point numbers less than
GSSLT	fs and ft fixed-point fixed-point numbers less than comparison
GSSLL	fs and ft fixed-point logic left shift word
GSDSLL	fs and ft fixed-point logic double shift left
GSSRL	fs and ft fixed-point logic right shift word
GSDSRL	fs and ft fixed-point logic right-shift doubleword
GSSRA	fs and ft fixed-point arithmetic right shift word
GSDSRA	fs and ft fixed-point arithmetic double shift right
GSSLEU	Comparison of fs and ft fixed-point unsigned fixed-point numbers less than or equal to
GSSLE	Comparison of fs and ft fixed-point fixed-point numbers

2.3.3. Custom Extended X86 Binary Translation Acceleration Instructions

18

Page 34

Table 2-17 Custom extended X86 binary translation acceleration instructions

Instruction mnemonics	Brief instruction function
X86AND	Set only the logical bits of EFLAG and x86
X86OR	Set only the logical bits of EFLAG or
X86XOR	XOR only the logical bit XOR of EFLAG
X86DADD	Only double-word addition of EFLAG in x86 mode
X86ADD	Set only the word addition of EFLAG in x86 mode
X86DADDU	Set x86-only exceptionless doubleword plus
X86ADDU	Set x86-only exception-free word addition
X86DSUB	Set only doubleword subtraction for EFLAG in x86 mode
X86SUB	Set only the subtraction of EFLAG in x86 mode
X86DSUBU	Set x86-only exceptionless doubleword reduction
X86SUBU	Set the exception-free subtraction of only EFLAG in x86 mode
X86INC	In x86 mode, only double-word increment of EFLAG is set. 1
X86DEC	In x86 mode, only double-word decrement of EFLAG is set 1
X86DSLL	Set doubleword left shift of EFLAG only in x86 mode
X86SLL	Set only word shift left of EFLAG in x86 mode
X86DSLL32	In x86 mode, only the shift amount of EFLAG plus 32 double-word logical left shift is set.
X86DSLLV	X86 only double-word variable shift amount of EFLAG left shift
X86SLLV	Set only the word variable shift amount of EFLAG to the left in x86 mode
X86DSRL	Set double-word logical right shift of EFLAG only in x86 mode
X86SRL	Set the word logic right of EFLAG only in x86 mode
X86DSRL32	In x86 mode, only the shift amount of EFLAG plus 32 double-word logic logic is set to the right.
X86DSRLV	Logical right shift of double-word variable shift amount for EFLAG only in x86 mode
X86SRLV	Logic only right shift of the word variable shift amount of EFLAG in x86 mode
X86DSRA	Set doubleword arithmetic right shift of EFLAG only in x86 mode
X86SRA	Set only the word arithmetic right shift of EFLAG in x86 mode
X86DSRA32	In x86 mode, only the shift amount of EFLAG plus 32 double-word logical arithmetic right shift is set.
X86DSRAV	Arithmetically set double-word variable shift amount arithmetic right shift of EFLAG only in x86 mode
X86SRAV	Arithmetically set the word variable shift amount of EFLAG to the right in x86
X86DROTR	Double-word right shift of EFLAG only in x86 mode
X86ROTR	Right-shift only the word loop of EFLAG in x86 mode

Page 35

Instruction mnemonics	Brief instruction function
X86DROTR32	In x86 mode, only the shift amount of EFLAG plus 32 double-word logical loop right shift is set.
X86DROTRV	Double-word cyclic right shift that only sets the variable shift amount of EFLAG in x86 mode

X86ROTRV	Word cycle right shift with only variable shift amount of EFLAG set in x86
X86MFFLAG	Extract the value of the EFLAG flag in x86
X86MTFLAG	Modify the value of the EFLAG flag in x86 mode
X86J	Jump to EFLAG value in x86 mode
X86LOOP	Cycles according to EFLAG value in x86 mode
SETTM	x86 floating-point stack mode set
CLRTM	x86 floating-point stack mode clear
INCTOP	x86 floating-point stack top pointer plus 1
DECTOP	x86 floating-point stack top pointer minus 1
MTTOP	Write x86 floating-point stack top pointer
MFTOP	Read x86 floating-point stack top pointer
SETTAG	Judgment and Set Register

GSXOR	XOR of fs and ft
GSNOR	fs and ft logical OR
GSAND	fs and ft logical bitwise AND
GSADDU	fs and ft fixed-point unsigned word addition
GSOR	fs and ft fixed-point logical bit OR
GSADD	fs and ft fixed-point plus
GSDADD	fs and ft fixed-point doubleword addition
GSSEQU	Equal comparison of fs and ft fixed-point numbers
GSSEQ	Equal comparison of fs and ft fixed-point numbers
GSSUBU	fs and ft fixed-point unsigned word subtraction
GSSUB	fs and ft fixed-point subtraction
GSDSUB	fs and ft fixed-point doubleword subtraction
GSSLTU	fs and ft fixed-point unsigned fixed-point numbers less than
GSSLT	fs and ft fixed-point fixed-point numbers less than comparison
GSSLL	fs and ft fixed-point logic left shift word
GSDSLL	fs and ft fixed-point logic double shift left

GSSRL	fs and ft fixed-point logic right shift word
GSDSRL	fs and ft fixed-point logic right-shift doubleword
GSSRA	fs and ft fixed-point arithmetic right shift word
GSDSRA	fs and ft fixed-point arithmetic double shift right
GSSLEU	Comparison of fs and ft fixed-point unsigned fixed-point numbers less than or equal to
GSSLE	Comparison of fs and ft fixed-point fixed-point numbers

2.3.4. Custom Extended Miscellaneous Instructions

Table 2-18 Custom extended miscellaneous instructions

Instruction mnemonics	Brief instruction function

CAMPV	Query the lookup table and return the content of the hit
CAMPI	Query the lookup table and return the index of the hit
CAMWI	Write lookup table specified item
RAMRI	Read the contents of the specified entry in the lookup table

2.3.5. Custom extended 64-bit multimedia acceleration instructions

Table 2-19 Custom extended 64-bit multimedia acceleration instructions

Instruction mnemonics	Brief instruction function
PADDSH	Four 16-bit signed integers plus signed saturation
PADDUSH	Four 16-bit unsigned integers added, unsigned saturated
PADDH	Four 16 digits plus
PADDW	Two 32 digits plus
PADDSB	Eight 8-bit signed integers added, signed saturated
PADDUSB	Eight 8-bit unsigned integers added, unsigned saturated
PADDB	Eight 8 digits plus
PADDD	64 digits plus
PSUBSH	Four 16-bit signed integers minus, signed saturation

Instruction mnemonics	Brief instruction function
PSUBUSH	Subtraction of four 16-bit unsigned integers, unsigned saturation
PSUBH	Four 16-digit minus
PSUBW	Two 32-digit minus
PSUBSB	Eight 8-bit signed integers minus, signed saturation
PSUBUSB	Eight 8-bit unsigned integers minus, unsigned saturation
PSUBB	Eight 8-digit minus
PSUBD	64-digit minus
PSHUFH	Shuffle four 16 digits
PACKSSWH	32-bit signed integer converted to 16-bit, signed saturation
PACKSSHB	16-bit signed integer converted to 8-bit, signed saturation
PACKUSHB	16-bit signed integer converted to 8-bit, unsigned saturation
PANDN	fs after negation and ft bitwise AND
PUNPCKLHW	Unpacking lower 16 digits
PUNPCKHHW	Unpacking height 16 digits
PUNPCKLBH	Unpacking lower 8 digits
PUNPCKHBH	Unpacking high 8 digits
PINSRH_0	ft lower 16 digits inserted into fs lower 16 digits
PINSRH_1	ft lower 16 digits inserted into fs lower 16 digits
PINSRH_2	ft lower 16 digits inserted into fs lower 16 digits
PINSRH_3	ft lower 16 digits inserted into fs lower 16 digits
PAVGH	Four 16-bit unsigned integers are averaged
PAVGB	Average of eight 8-bit unsigned integers

PMAXSH	The larger of four 16-bit signed integers
PMINSH	Four smaller 16-bit signed integers
PMAXUB	The larger of eight 8-bit unsigned integers
PMINUB	The smaller of eight 8-bit unsigned integers
PCMPEQW	Two 32-digit equal comparisons
PCMPGTW	Two 32-bit signed integers are greater than the comparison
PCMPEQH	Four 16-digit equal comparisons
PCMPGTH	Four 16-bit signed integers greater than the comparison
PCMPEQB	Eight 8-digit equal comparisons
PCMPGTB	Eight 8-bit signed integers greater than the comparison

Page 38

Instruction mnemonics	Brief instruction function
PSLLW	Two 32-bit logical left shifts
PSLLH	Four 16-digit logical left shifts
PMULLH	Multiply four 16-bit signed integers to get the lower 16 bits
PMULHH	Multiply four 16-bit signed integers to get the upper 16 bits
PMULUW	Multiply low 32-bit unsigned integers and store 64-bit results
PMULHUH	Multiply four 16-bit unsigned integers to get the upper 16 bits
PSRLW	Two 32-bit logical right shifts
PSRLH	Four 16-bit logical right shifts
PSRAW	Two 32-bit arithmetic shift right
PSRAH	Four 16-digit arithmetic shift right
PUNPCKLWD	Lower 32-bit arrays are synthesized into 64-bit numbers
PUNPCKHWD	High 32-bit arrays are synthesized into 64-bit numbers
PASUBUB	Subtract eight 8-bit unsigned integers and take the absolute value
PEXTRH	fs 16-bit copy to fd low 16-bit, fd high-order padding 0
PMADDHW	Multiply four 16-bit signed numbers and accumulate the low and high bits
BIADD	Multibyte accumulation
PMOVMSKB	Conditional byte shift
GSXOR	XOR of fs and ft
GSNOR	fs and ft logical OR
GSAND	fs and ft logical bitwise AND
GSADDU	fs and ft fixed-point unsigned word addition
GSOR	fs and ft fixed-point logical bit OR
GSADD	fs and ft fixed-point plus
GSDADD	fs and ft fixed-point doubleword addition
GSSEQU	Equal comparison of fs and ft fixed-point numbers
GSSEQ	Equal comparison of fs and ft fixed-point numbers
GSSUBU	fs and ft fixed-point unsigned word subtraction
GSSUB	fs and ft fixed-point subtraction
GSDSUB	fs and ft fixed-point doubleword subtraction
GSSLTU	fs and ft fixed-point unsigned fixed-point numbers less than
GSSLT	fs and ft fixed-point fixed-point numbers less than comparison
GSSLL	fs and ft fixed-point logic left shift word

twenty three

Page 39

Instruction mnemonics	Brief instruction function
GSDSLL	fs and ft fixed-point logic double shift left
GSSRL	fs and ft fixed-point logic right shift word
GSDSRL	fs and ft fixed-point logic right-shift doubleword
GSSRA	fs and ft fixed-point arithmetic right shift word
GSDSRA	fs and ft fixed-point arithmetic double shift right
GSSLEU	Comparison of fs and ft fixed-point unsigned fixed-point numbers less than or equal to
GSSLE	Comparison of fs and ft fixed-point fixed-point numbers

twenty four

Page 40

3. CP0 control register

This chapter describes the operation of Coprocessor 0 (CP0). The main contents include the registers of CP0.

Definition and CP0 instruction implemented by Godson 3 processor. CP0 register is used to control the status change of the processor and report

The current state of the processor. These registers are read by MFC0 / DMFC0 instructions or by MTC0 / DMTC0 instructions

Come and write. The CP0 register is shown in Table 3-1 .

When the processor is running in core mode or bit 28 (CU0) in the status register (Status register) is set

When set, the CP0 instruction can be used. Otherwise, execution of the CP0 instruction will result in a "coprocessor unavailable exception ".

Table 3-1 CP0 register

Register number		Register name	description
Total number	Child number		
0	0	Index	Writable register for specifying TLB entries to be read / written
1	0	Random	Pseudo-random counter for TLB replacement
2	0	EntryLo0	The contents of the lower half of the TLB entry corresponding to the even virtual page (mainly Physical page number)
3	0	EntryLo1	The contents of the lower half of the TLB entry corresponding to the odd and virtual pages (mainly Physical page number)
4	0	Context	Virtual page translation table (PTE) to the kernel in 32-bit addressing mode
5	0	Page Mask	Set mask value for TLB page size
5	1	Page Grain	Flags support large page addresses
6	0	Wired	Number of fixed-line TLB entries (referring to those not used for random replacement) Low-end TLB entries)
7	0	Hwrena	Hardware register enable
8	0	BadVaddr	Wrong virtual address
9	0	Count	counter
10	0	EntryHi	The upper half of the TLB entry (virtual page number and ASID)
11	0	Compare	Counter comparison
12	0	Status	Processor status register
12	1	IntCtl	Extended interrupt control register
12	2	SRSCtl	Shadow register bank control register
13	0	Cause	Reason for the most recent exception

25

Page 41

14	0	EPC	Exception program counter
15	0	PRid	Processor revision identification number
15	1	EBase	Exception vector base address
16	0	Config	Configuration register
16	1	Config1	Configuration register 1
16	2	Config2	Configuration register 2
16	3	Config3	Configuration register 3
17	0	LLAddr	Link read memory address
18			Keep
19			Keep
20	0	Xcontext	Virtual page translation table (PTE) to the kernel in 64-bit addressing mode
twenty one			Keep

twenty two	0	Diagnose	Enable / disable BTB, RAS and clear ITLB table
twenty three	0	Debug	EJTAG debug register
twenty four	0	DEPC	EJTAG debug exception program counter
25	0/1/2/3	PerfCnt	Performance counter
26	0	ErrCtl	Parity / ECC check control and status
27	0	CacheErr	Cache ECC check control and status
28	0	TagLo	Lower half of CACHE TAG register
28	1	DataLo	Used to interact with and diagnose the cache data queue
29	0	TagHi	The upper half of the CACHE TAG register
29	1	DataHi	Used to interact with and diagnose the cache data queue
30	0	ErrorEPC	Error exception program counter
31	0	DESAVE	Register for Debug exception handling

3.1. Index Register (0, 0)

The Index register is a 32-bit readable / writable register, in which the value of the last six bits is used to index a TLB entry. send
The most significant bit of the register indicates whether the TLB Probe (TLBP) instruction was executed successfully.

The value of the Index register indicates the Read (TLBR) and TLB Index Write (TLBWI) instructions.

Figure 3-1 shows the format of the Index reg ibes the meaning of each field of the Index register.

26

Page 42

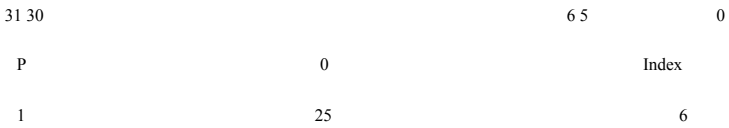


Figure 3-1 Index register

Table 3-2 Description of each field of the Index register

area	description
P	The probe failed. Set when the last TLB probe instruction (TLBP) was unsuccessful
Index	Index values of TLB entries indicating TLB read and TLB index write instructions
0	Reserved. Must be written as 0 and read as 0.

3.2. Random Register (1, 0)

The Random register is a read-only register in which the lower six bits index the entry of the TLB. After each instruction is executed,
Decrement the register value by one. At the same time, the register value floats between an upper and a lower bound. The upper and lower bounds are:

- The lower bound is equal to the number of TLB entries reserved for the operating system (ie the contents of the Wired register).
- The upper bound is equal to the number of entries in the entire TLB minus 1 (maximum 64-1).

The Random register indicates the TLB entry that will be manipulated by the TLB random write instruction. For this purpose, there is no need to read this register. However, this register is readable to verify the correct operation of the processor.

To simplify testing, the Random register is set to the upper bound when the system restarts. In addition, when the Wired register is written,

This register should also be set to the upper bound.

Figure 3-2 shows the format of the Random registe ibes the meaning of each field of the Random register.

Figure 3-2 Random register

Table 3-3 Fields of the Random Register

27

area	description
Random	Random TLB index value
0	Reserved. Must be written as 0 and read as 0.

3.3. EntryLo0 (2,0) and EntryLo1 (3,0) registers

The EntryLo register includes two registers of the same format:

- EntryLo0 for even virtual pages
- EntryLo1 for odd and virtual pages

The EntryLo0 and EntryLo1 registers are both read / write registers. When performing TLB read and write operations, they are split

Do not include the physical page number (PFN) of the parity page in the TLB entry . Figure 3-1 shows the format of these registers.

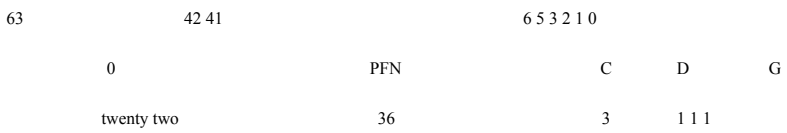


Figure 3-3 EntryLo0 and EntryLo1 registers

The PFN fields of the EntryLo0 and EntryLo1 registers are the upper 36 bits (47:12) of the 48-bit physical address.

Table 3-4 EntryLo register field

area	description
PFN	The page number is the high-order bit of the physical address.
C	Cache consistency properties for TLB pages.
D	Dirty bit. If this bit is set, the page is marked as dirty and writable. Actually this one is soft This file is used as write protection to prevent data from being changed.
V	Effective bit. When this bit is set, it indicates that the TLB entry is valid, otherwise a TLBL or The exception is TLBS.
G	Global bit. When both the G bits in EntryLo0 and EntryLo1 are set to 1, the processor will Ignore ASID when looking up.
0	Reserved. Must be written as 0 and read as 0.

There is only one global bit in each TLB entry. According to EntryLo0 [0] and EntryLo1 [0] in TLB write operation

The value is written.

28

3.4. Context (4,0)

The Context register is a read / write register that contains a pointer to an entry in the page table. The page table is an operation
As a system data structure, store the translation of virtual address to physical address.

When a TLB exception occurs, the CPU loads the TLB from the page table based on the invalidated transition. In general, the operating system
The system uses the Context register to address the mapping of the current page in the page table. Context register is copied from BadVAddr register
Partial information, but the information is arranged in a rocessing by the software TLB exception handler.

Figure 3-4 shows the format of the Context registe s the Context register fields.

63	23 22	4 3	0
PTEBase	B dVPN2	0	
4	19	4	

Figure 3-4 Context register

Table 3-5 Context Register Field

area	description
BadVPN2	This field is written by hardware when the TLB is exceptional. Virtual page Number (VPN).
PTEBase	This field is a read / write field used by the operating system. The value written to this field allows the operating system to copy the Context The register acts as a pointer to the current page table in memory.
0	Reserved. Must be written as 0 and read as 0.

The 19-bit BadVPN2 field contains 31:13 bits of the virtual address that caused the TLB exception; the 12th bit is excluded because
A single TLB entry maps to a parity page pair. For a 4K byte page size, this format can be directly
The addressing PTE table entry is an 8-byte long and organized page table. For other sizes of pages and PTEs, move and block this
The value can generate a suitable address.

3.5. PageMask Register (5, 0)

The PageMask register is a readable and writable register and is used in the process c ıg the TLB; it contains a comparison mask,
You can set different page sizes for each TLB entry, as shown in Table 3-6 . The format of n in Figure 3-5.

TLB read and write operations use this register as a source or destination; when performing virtual-to-real address translation, the corresponding in the TLB

29

The corresponding bits in the PageMask register indicate which of the virtual address bits 24:13 are used for comparison. When the value of the MASK field is not When the values in Table 3-6, the operation of the TLB is undefined. The 0 field is reserved and must be written as 0. It returns 0 when read.

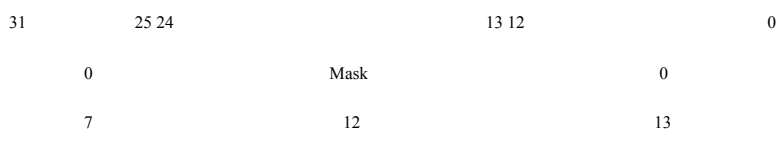


Figure 3-5 PageMask register

Table 3-6 Mask values for different page sizes

Page size	Bit											
	twenty four	twenty three	twenty two	twenty one	20	19	18	17	16	15	14	13
4Kbytes	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	1	1	1	1
256 Kbytes	0	0	0	0	0	0	1	1	1	1	1	1
1 Mbytes	0	0	0	0	1	1	1	1	1	1	1	1
4 Mbytes	0	0	1	1	1	1	1	1	1	1	1	1
16M bytes	1	1	1	1	1	1	1	1	1	1	1	1

3.6. PageGrain Register (5, 1)

The PageGrain register is a readable and writable register. Godson 3 only defines the 29th bit of this register: ELPA (Enable Large Physical Adress), the remaining bits are left at 0.

When ELPA = 1, Loongson 3 supports 48-bit physical address; when ELPA = 0, Loongson 3 supports only 40 bits 理 Address.

The ability to write the ELPA bit depends on the LPA field of the Config3 The ELPA bit is set to zero. The format of the register of FIG . 3 -6 register don

Table 3-7.

Figure 3-6 PageGrain register

30

Table 3-7 PageGrain Register Field

area	description
ELPA	Whether this field is set to indicate whether large physical addresses are supported
0	Reserved. Must be written as 0 and read as 0.

3.7. Wired Register (6, 0)

The Wired register is a readable / writable register whose value specifies the fixed and random tables in the TLB

Boundaries between items, such as

Figure 3-7As shown. Wired entries are fixed and irreplaceable entries. The contents of these entries are not written by the TLB.

Operation modification. The content of the random entry can be modified.

Figure 3-7 Wired register boundaries

The Wired register is set to 0 during a system reset. When writing this register, the Random register value must be set to the upper limit (see

31

Page 47

Figure 3-8Represents the format of the Wired register;

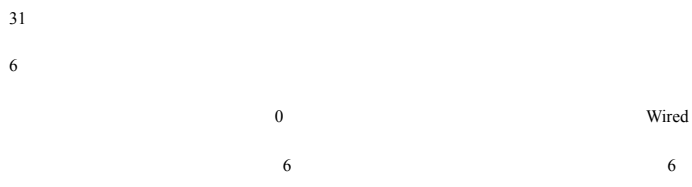


Figure 3-8 Wired register

Table 3-8 Wired register fields

area	description
Wired	TLB fixed entry boundary
0	Reserved. Must be written as 0 and read as 0.

3.8. HWREna register (7, 0)

The HWREna register is a readable and writable register. Godson 3 only defines the Mask field of this register.

Figure 3-9 shows the format of the HWREna register. Table 3-9 describes the correspondence between the Mask domain and the hardware register.

Figure 3-9 HWREna register

Table 3-9 Correspondence between Mask field and hardware register

Hardware register	Bit

	3	2	1	0
CPUnum	0	0	0	1
SYNCI_Step	0	0	1	0
CC	0	1	0	0
CCRes	1	0	0	0

3.9. BadVAddr register (8, 0)

The Bad Virtual Address Register (BadVAddr) is a read-only register that records the most recent TLB or Virtual address with exception of address error. Unless a software reset occurs, except for an NMI or Cache error, the BadVAddr register will

Figure 3-10 The format of the error virtual address register is shown.

Figure 3-10 BadVAddr register

3.10. Count register (9,0) and Compare register (11,0)

Both the Count register and the Compare register are 32-bit read-write registers.

The Count register works as a real-time timer and is incremented every two clock cycles.

The Compare register is used to generate an interrupt at a specific time. This register is written with a value and is constantly updated. Compare with the value in the Count register. Once these two values are equal, an interrupt request is generated. In the Cause register

TI and IP [7] are set. The TI bit in the Cause register is re register is written again.

Figure 3-11 shows the format of the Count regist format of the Compare register.

33

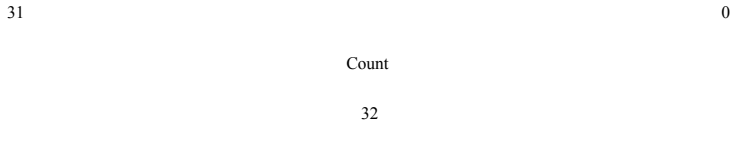


Figure 3-11 Count register

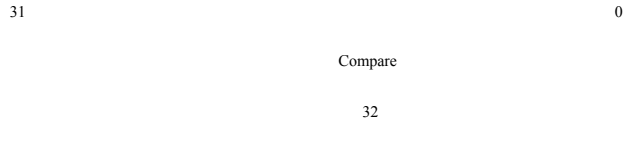


Figure 3-12 Compare register

3.11. EntryHi register (10, 0)

The EntryHi register is used to store the high-order bits of TLB entries when reading and writing TLB.

The EntryHi register can be read by TLB Probe, TLB Write Random, TLB Write Indexed, and TLB Read

Figure 3-13represents the format of the EntryHi register.

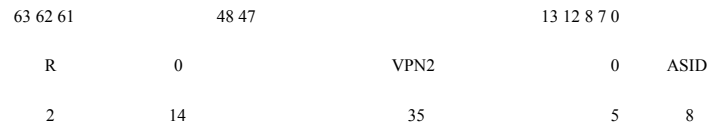


Figure 3-13 EntryHi register

Figure 3-13 shows the fields of the EntryHi register.

Table 3-10 EntryHi register fields

area	description

34

VPN2	Virtual page number divided by 2 (mapped to double page); high order bits of virtual address.
ASID	Address space identification domain. An 8-bit field; used to share the TLB for multiple processes; for the same Virtual page number, each process has a different mapping from other processes.
R	Area bit. (00-> User, 01-> Super User, 11-> Core) Used to match vAddr63 ... 62
0	Reserved. Must be written as 0 and read as 0.

The VPN2 domain contains 61:13 bits of a 64-bit virtual address.

When a TLB Refill, TLB Invalid, or TLB Modified exception occurs, no TLB entry matches

The virtual page number (VPN2) and ASID in the virtual address will be loaded into the EntryHi register.

3.12. Status register (12, 0)

The Status register (SR) is a read-write register that contains the operating mode, interrupt enable, and processor status diagnostics.

The following list describes some of the more important Status register fields ;

Figure 3-14Shows the format of the entire register, including a description of the field. The important fields are:

- The 8-bit interrupt mask (IM) field controls the enabling of eight interrupt conditions. Interrupts must be enabled before they are triggered.

The corresponding bits in the interrupt mask field of the Status register and the interrupt pending field of the Cause register should be set. more

For information, refer to the Interrupt Pending (IP) field of the Cause register.

- The 4-bit coprocessor availability (CU) domain controls the availability of 4 possible coprocessors. Regardless of the CU0 bit

How to set it, CP0 is always available in kernel mode.

Figure 3-14 The format of the Status register is sh[illegible]es the fields of the Status register.

31 28	27 26 25 24 23 22 21 20 19 18 16 15 8 7 6 5 4 3	2	1 0
CU (cu3: cu0)	0 FR 0 PX BEV 0 SR NMI 0 IM7-IM0 KX SX UX KSU ERL EXL IE		
4	1 1 2 1 1 1 1 1 2 8 1 1 1	1	1 1

35

Figure 3-14 Status register

Table 3-11 Status Register Field

area	description
CU	Controls the availability of 4 coprocessor units. Regardless of the CU0 bit setting, CP0 is always available in kernel mode of. 1- Available 0- not available

The initial value of the CU field is 0011

0 Reserved. Must be written as 0 and read as 0.

FR Enable additional floating-point registers s

0- Registers

1 - Registers

PX Enable 64-bit operations in user mode (64-bit operations in the remaining modes do not need to be enabled)

1-enable

0-Not enabled (At this time, whether 64-bit operation is available in user mode requires UX bit judgment)

BEV Control entry address of exception vector

0-normal

1-start

SR 1 indicates a soft reset exception occurred

NMI Whether an NMI exception occurred. Note that software cannot write this bit from 0 to 1

IM Interrupt mask: Controls the enable of each external, internal and software interrupt. If the interrupt is enabled, it will be allowed to trigger,

At the same time, the corresponding bit in the interrupt pending field of the Cause register is set.

0-forbidden

1-allowed

KSU Mode bit

Undefined

36

Page 52

general user

root

core

KX 1-Enable 64-bit Kernel segment access; use XTLB Refill vector.

0-64-bit Kernel segment cannot be accessed; use TLB Refill vector

SX 1-Enable 64-bit Supervisor segment access; use XTLB Refill vector.

0-64-bit supervisor segment not accessible; use TLB Refill vector

UX 1-Enable 64-bit User segment access; use XTLB Refill vector.

0-cannot access 64-bit User segment; use TLB Refill vector

ERL Error level. The processor will reset this bit when a reset occurs, software reset, NMI or Cache error.

normal

error

EXL Exception level. When an exception is not caused by a reset, software reset, or cache error, the processor sets

The bit.

IE Interrupt enable.

Disable all interrupts

Enable all interrupts

Status register mode and access status

The following describes the fields in the Status register for setting the mode and access status:

- Interrupt enable: Interrupt is enabled when the following conditions are met:
  - IE = 1 and
  - EXL = 0 and
  - ERL = 0.

If these conditions are encountered, the setting of the IM bit enables interrupts.

- Operation mode: The following bit fields need to be set when the processor is in normal user, kernel and super user mode.
  - When KSU = 10 2 , EXL = 0 and ERL = 0, the processor is in normal user mode.
  - When KSU = 01 2 , EXL = 0 and ERL = 0, the processor is in superuser mode.

37

Page 53

  - When KSU = 00 2 , or EXL = 1 or ERL = 1, the processor is in kernel mode.
- Kernel address space access: When the processor is in kernel mode, it can access the kernel address space.
- Super user address space access: When the processor is in kernel mode or super user mode, it can access super user

User address space.

- User address space access: The processor can access the user address space in all three operating modes.

Status register reset

At reset, the value of the Status register is 0x30c000e4.

3.13. IntCtl Register (12, 1)

The IntCtl register is a 32-bit register that can be read and written. It manages extended interrupts in the Release2 architecture. Long

Core 3 implements vector interrupts. The VS field of the IntCtl register is used to represent the vector space between the interrupt vectors. 1 domain is not

It can be written and read as 1. The 0 field is reserved and must be written as 0. When read, it returns 0. Where 31:26 bits represent the clock interrupt and

Performance Counter interrupts share HW5.

Figure 3-15 shows the format of the IntCtl regist he correspondence between the VS domain and the vector space.

31	26	25	10	9	5	4	0
1		0		VS		0	
6		16		5		5	

Figure 3-15 IntCtl register

Table 3-12 Correspondence between the encoding of the VS domain and the vector space

coding	Vector space (**16** hex)	Vector space (**10** decimal)
0x00	0x000	0
0x01	0x020	32

0x02	0x040	64
0x04	0x080	128
0x08	0x100	256

38

0x10	0x200	512

3.14. SRSCtl Register (12, 2)

The SRSCtl register is a 32-bit readable and writable register. It governs the shadow register bank in the processor. Thanks Godson No. 3 has only one set of general-purpose registers and no shadow registers, so the shadow of the general-purpose registers is the general-purpose registers themselves. The SRSCtl register in core 3 implements only two fields

Figure 3-16 shows the format of the SRSCtl registe he fields of the SRSCtl register.

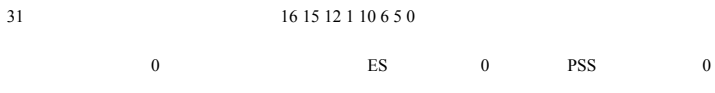


Figure 3-16 SRSCtl register

Table 3-13 SRSCtl Register Fields

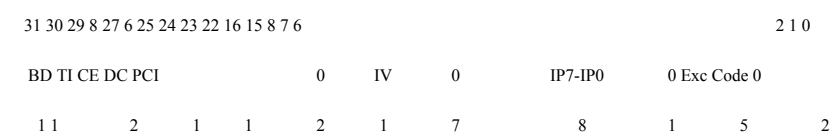
area	description
ESS	Shadow register bank for exceptions. Can only be 0 in Loongson 3
PSS	Previous shadow register bank. Can only be 0 in Loongson 3
0	Reserved. Must be written as 0 and read as 0.

3.15. Cause register (13, 0)

The 32-bit read-write Cause register describes ption occurred.

Figure 3-17 shows the format of this register, a he fields of the Cause register. A 5-digit exception

The code (ExcCode) indicates one of the reasons, as shown in Table 3-13 .



39

Figure 3-17 Cause register

Table 3-14 Cause Register Field

area	description
BD	Indicates whether the last exception used was in a branch delay slot. 1-Delay slot 0-normal
TI	Clock interrupt indication 0-no clock interrupt 1-Time interruption pending processing
CE	Coprocessor unit number when a coprocessor unavailable exception occurs.
DC	Disable count register 0-count register available 1-Count register disabled
PCI	Performance counter interrupt indication 0-no performance counter interrupt 1-Performance counter interrupt pending
IV	Interrupt exception entry 0-use general exception vector (0x180) 1-use special interrupt vector (0x200)
IP	Indicate waiting interruptions. This bit will remain unchanged until the interrupt is removed. IP0 ~ IP1 are soft interrupt bits, which can be set by software With clear. 1-interrupt wait 0-no interrupt
ExcCode	Exception code field (see Table 3-15)
0	Reserved. Must be written as 0 and read as 0.

Table 3-15 ExcCode field of the Cause register

Page 56

Exception code	Mnemonic	description
0	Int	Break
1	Mod	TLB modification exception
2	TLBL	TLB exception (read or fetch)
3	TLBS	TLB exception (stored)
4	AdEL	Address error exception (read or fetch instruction)
5	AdES	Address error exception (stored)

6	IBE	Bus error exception (instruction fetch)
7	DBE	Bus error exception (data reference: read or store)
8	Sys	System call exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor unavailable exception
12	Ov	Arithmetic overflow exception
13	Tr	Trap exception
14	-	Keep
15	FPE	Floating point exception
16	IS	Stack exception
17-18	-	Keep
19	DIB	Debug instruction exception
20	DDBS	Debug stored data exception
twenty one	DDBL	Debug fetch data exception
twenty two	-	Keep
twenty three	WATCH	WATCH exception
24-25	-	Keep
26	DBP	Debug breakpoint exception

41

Page 57

27	DINT	Debug exception
28	DSS	Debug single step exception
29	-	Keep
30	CACHERROR	cache exception
31	-	Keep

3.16. Exception Program Counter register (14, 0)

The Exception Program Counter (EPC) is a read / write register that includes

Continue processing address after exception processing is completed.

For synchronization exceptions, the contents of the EPC register are one of the following:

- Instruction virtual address, which is the direct cause of the exception, or
- Previous branch or jump instruction (when the instruction is in the branch delay slot, the instruction delay bit is in the Cause register

Set) virtual address.

When the EXL bit in the Status register is set, the processor does not write to the EPC register.

Figure 3-18 shows the format of the EPC register.

63

0

EC

64

Figure 3-18 EPC register

3.17. Processor Revision Identifier (PRID) register (15, 0)

The PRID register is a 32-bit read-only register that contains the cal : CP0 implementation

This and revised information. Figure 3-19 shows the format of this registe e fields of this register.

42

31 16 15 8 7 0

0 Imp Rev

16 8 8

Figure 3-19 Processor Revision Identifier register

Table 3-16 PRId Register Field

area	description
IMP	Implementation version number
REV	Revision number
0	Reserved. Must be written as 0 and read as 0.

The low (7: 0) bits of the PRID register can be used as the revision number, and the high (15: 8) bits can be used as the actual number

The current version number. The Godson 3 implementation version is 0x63 and the revision version is 0x03.

The format of the version number is YX, where Y (7: 4 digits) is the major version number, and X (3: 0 digits) is the minor

version number.

The version number can distinguish some processor versions, but there is no guarantee that any changes to the processor will be reflected in the PRID mail

In other words, there is no guarantee that changes to the version number must reflect modifications to the processor. For this reason, the register

The value of is not given, and software cannot rely on the version number in the PRID register to identify the processor.

3.18. EBase Register (15, 1)

The EBase register is a read-write register containing the exception vector base address and a read-only CPU number.

When BEV = 0 in the status register, the base add

Figure 3-20 shows the format of the EBase regist

Table 3-17 describes the fields of the EBase regis...

Figure 3-20 Ebase register

Table 3-17 Ebase Register Field

area	description
1	Cannot be written but read only, read as 1
0	Reserved. Must be written as 0 and read as 0.
Exception Base	Combined with 31 and 30 bits to specify the exception vector base address
CPUNum	In multi-core systems, used to indicate the processor number

3.19. Config register (16, 0)

The Config register specifies various configuration options in the Godson 3 processor; Table 3-18 lists these options.

Some configuration options defined by bits 31: 3 of the Config register are set by hardware at reset and are used only as

The read status bits are included in the Config register for software access. Other configuration options (bits 2: 0 of the Config register)

It is readable / writable and controlled by software. These fields are undefined at reset.

The configuration of the Config register is limited. The Config register should be initialized by software before the cache is used

And the cache should be reinitialized after any changes.

Figure 3-21 shows the format of the Config regist the fields of the Config register. Config hosting

The initial value of the device is 0x00030932.

31 30		16 15 14 13 12 10 9 7 6 4 3 2 0						
M	0	BE	AT		AR	MT	0	VI K0
1	15	4	3		3	1	1	1 3

Figure 3-21 Config register

Table 3-18 Config register field

area	description
0	Reserved. Must be written as 0 and read as 0.
M	Whether the config1 register exists. Set to 1 to indicate presence.
BE	Indicate tail type
	1-big end
	0-little end
AT	Specify the architecture type
	0 – MIPS32
	1 – MIPS64, can only access 32-bit address segments
	2-MIPS64 with access to all address segments
	3-reserved
AR	Specified version
	0 – Release 1
	1 – Release 2
	2-7 – Reserved
MT	Indicate the type of memory management unit
	0-no mapping
	1-Standard TLB
	2-7 – Reserved
VI	Indicates whether the instruction cache is virtual
	0-Instruction cache is not virtual

	1-Instruction cache is virtual
K0	Kseg0 Consistency Algorithm (Cache Consistency Algorithm)
	2 – Uncached
	3 – Cacheable
	Rest-reserved

3.20. Config1 register (16, 1)

The Config1 register specifies the cache configuration of the Godson 3 processor.

The Config1 register is an additional content of the Config register. All contents are read-only and are automatically set during reset.

Home.

Figure 3-22 shows the format of the Config1 registe

Table 3-19 describes the fields of the Config1 register. The initial value of the Config1 register is 0xfee31193.

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

Figure 3-22 Config1 register

Table 3-19 Config1 register field

area	description	
M	Whether the config2 register exists. Set to 1 to indicate presence.	
MMU Size-1	TLB entries reduced by 1	
IS	Icache groups per channel	
	Coding	meaning
	0	64
	1	1
	2	2 6
	3	512
	4	1024
	5	204
		4096
	7	Keep
IL	Icache group size	
	Coding	meaning
	0	No Icache
	1	4 bytes
	2	8 bytes
	3	16 bytes
		32 bytes
	5	64 bytes
	6	128 bytes

	7	Keep
IA	Icache connection Coding meaning	
	0	Direct connection
	1	2 way connected
	2	3-way connected
	3	4-way connection
	4	5 way connected
	5	6-way connected
	6	7-way connected
	7	8-way connected
DS	Dcache each group Coding meaning	
	0	64

47

Page 63

		128
	2	256
	3	512
	4	1024
	5	2048
	6	4096
	7	Keep
DL	Dcache group size Coding meaning	
	0	No Dcache
	1	4 bytes
	2	8 bytes
	3	16 bytes
	4	32 bytes
	5	64 bytes
	6	128 bytes
	7	Keep
DA	Dcache connection Coding meaning	
	0	Direct connection
	1	2 way connected
	2	3-way connected
	3	4-way connection

	4	5 way connected
	5	6-way connected
	6	7-way connected
	7	8 phase
C2	Whether No. 2 coprocessor is implemented	

48

	0-not implemented
	1-implementation
	Whether MDMX ASE is implemented
MD	0-not implemented
	1-implementation
	Whether the performance count register is implemented
PC	0-not implemented
	1-implementation
	Whether the Watch register is implemented
WR	0-not implemented
	1-implementation
	Whether MIPS16e is implemented
CA	0-not implemented
	1-implementation
	Whether EJTAG is implemented
EP	0-not implemented
	1-implementation
	Whether FPU is implemented
FP	0-not implemented
	1-implementation

3.21. Config 2 register (16, 2)

The Config2 register specifies the configuration of the secondary cache in the Godson 3 processor.

As the Config2 register, all contents are read-only a upon reset.

Figure 3-23 shows the format of the Config1 registe he fields of the Config2 register. Config2

The initial value of the register is 0x80001643.

49

Page 65

31 30 28	27 24	23 20	19 16	15 12	11 8 7 4	3 0
M TU	TS	TL	A	SU	S SL	SA
1	4	4	4	4	4	4

Figure 3-23 Config2 register

Table 3-20 Config2 register field

area	description
M	Whether the config3 register exists. Set to 1 to indicate presence.
TU	Three-level cache control or status bits
TS	Level 3 cache per group Coding meaning 0 64 1 128 2 256 3 2 4 024 5 2048 6 4096 7 19 8-15 Reserved
TL	Three levels of cache size Coding meaning 0 No Icache 1 4 bytes 2 8 bytes 3 16 bytes 4 32 bytes

Page 66

	5 64 bytes 6 128 bytes 7 256 bytes 8-15 Reserved
	Three-level cache connection

	coding	meaning
	0	Direct connection
	1	2 way connected
	2	3-way connected
TA	3	4-way connection
	4	5 way connected
	5	6-way connected
	6	7-way connected
	7	8 phase
	8 15	Reserved

SU	Level 2 cache control or status bits
SS	Level 2 cache per group

	Coding	meaning
	0	64
	1	128
	2	256
	3	512
	4	1024
	5	2048
	6	4096
	7	8192
	-15	reserved

SL	Secondary cache size per group

51

Page 67

	Coding	meaning
	0	No Icache
	1	4 bytes
	2	8 bytes
	3	16 bytes
	4	32 bytes
	5	64 bytes
	6	128 bytes
	7	256 bytes
	8-15	Reserved

SA	Two-level cache association method

	Coding	meaning
	0	Direct connection

1	2 way connected
2	3-way connected
3	4-way connection
4	5 way connected
5	6-way connected
6	7-way connected
7	8-way connected

8-15 Reserved

3.22. Config 3 register (16, 3)

The Config3 register marks whether some functions ontents are read-only and are automatically set upon reset.

Figure 3-24 shows the format of the Config1 registe he fields of the Config1 register. Config1

The initial value of the register is 0x000000a0.

31 30	11 10 9 8 7 6 5 4	3 2 1 0
M	0	DSPP LPA VEIC Vint SP 0 MT SM TL
1	20	1 twenty one 1 1 1 1 1 1 1

Figure 3-24 Config3 register

Table 3-21 Config3 register fields

area	description
M	Keep
0	Reserved. Must be written as 0 and read as 0.
DSPP	Whether MIPS DSPASE is implemented 0-not implemented 1-implementation
LPA	Whether large physical addresses are implemented 0-not implemented 1-implementation
VEIC	Whether the external interrupt controller is implemented 0-not implemented 1-implementation
Vint	Whether vector interrupt is implemented 0-not implemented

	1-implementation
SP	Whether small page support is implemented 0-not implemented 1-implementation
MT	Whether MIPS MTASE is implemented 0-not implemented

	1-implementation
SM	Whether SmartMIPS ASE is implemented 0-not implemented 1-implementation
TL	Whether Trace Logic is implemented 0-not implemented 1-implementation

3.23. Load Linked Address (LLAddr) register (17, 0)

The LLAddr register is a 64-bit read-only register. The LLAddr register is used to store the most recent load-linked instructions The address page number of the command, PFN, is cleared when the exception returns (when the eret instruction occurs). On Godson 3 The format of this register is shown in Figure 3-25 .

63	47	11	0
8	12		
0	PFN	0	
16	36	12	

Figure 3-25 LLAddr register

3.24. XContext register (20, 0)

The read-write XContext register contains a pointer to an entry in the operating system page table. When one happens With the exception of one TLB, the operating system loads the TLB from the page table based on the invalidated translation.

The XContext register is used for XTLB refill processing. It handles the loading of TLB entries in a 64-bit address space. For system use. The operating system sets the PTEBase field .

Figure 3-26 shows the format of the XContext register : fields of the XContext register.

63	41 40 39 38	4 3 0
PTEBase	R	dVPN2	0
twenty three	2	35

Figure 3-26 XContext register

The 35-bit BadVPN2 domain contains 47:13 bits in the virtual address that caused the TLB exception, because a single TLB entry maps Shot on a parity page pair, so the 12th bit is not included. When the page size is 4K bytes, this format can Directly addressed PTE entries are 8-byte long and organized page tables. For other pages and PTE sizes, shifted And the mask can get the correct address.

Table 3-22 XContext register fields

area	description
BadVPN2	This field is written by the hardware when a TLB exception occurs. 2.
R	This field contains 63:62 bits of the virtual address. general user root Kernel
0	Reserved. Must be written as 0 and read as 0.
PTEBase	Read / write domain, this value allows the operating system to use the XContext register as a pointer to memory A pointer to the current page table.

3.25. Diagnostic Register (22, 0)

Godson processor's unique 64-bit registers are mainly used to contr nd special operations of the processor . Figure 3-27 The Diagnostic register shows the format of the Diagnostic register. Diagnostic register. Domain.

63	8	7	6	5	4	3 2	1	0
0		W-CAC	W-ISS	S-ISS	S-FET	0	ITLB	BTB	RAS
56		1	1	1	1	1	1	1

Figure 3-27 Diagnostic register

Table 3-23 Diagnostic register fields

area	description
0	Reserved. Must be written as 0 and read as 0.
W-CAC	Remove the limit of wait-cache operation
W-ISS	Remove the limit of wait-issue operation
S-ISS	Remove restrictions on store-issue operations
S-FET	Remove restrictions on store-fetch operations
ITLB	Clear ITLB when writing 1
BTB	Clear BTB on write 1
RAS	Writing a 1 disables the use of RAS.

3.26. Debug Register (23, 0)

The Debug register is a 32-bit read-write register. Debug register contains recent debug exceptions

Or the cause of the exception that occurs in debug mode, it also controls single step interrupts. This register specifies the debug resources

And other internal states. Only the LSNM and SSt fields can be written. When reading the Debug register in non-debug mode

Only the DM bit and EJTAGver field can be read. Godson 3 does not implement the power saving mode when debug exceptions are made. On reset, Debug

The initial value of the register is: 0x02018000.

Figure 3-28 shows the format of the Debug registe ields of the Debug register.

56

31 30 29 28 27 25 twenty four 17 fifteen 14 10 9 8 7 5 4 3 2 1 0

26 18 6

DBD DM NoDCR LSNM 0 CountDM 0 EJTAGver DExcCo NoSS SSt 0 DINT DIB DDBS DD L Bp DSS

1 1 1 1 2 1 7 3 5 1 1 2 1 1 1 1 1

Figure 3-28 Debug register

Table 3-24 Debug Register Fields

area	description
0	Reserved. Must be written as 0 and read as 0.
DBD	Indicates whether the previous debug exception occurred in the delay slot. 0-not

	1 – yes
DM	Indicates whether the processor is in debug mode
	0 – Non-Debug Mode
	1 – Debug Mode
NoDCR	Indicates whether the dseg segment exists
	1-does not exist
	0-exists
LSNM	When the dseg section is present, indicate the addresses where loads / stores are available
	0-dseg segment
	1-system memory
CountDM	Working state of Count register when entering DM
	0 – stopped
	1 – runnig
EJTAGver	EJTAG version
	0-versions 1 and 2.0

57

Page 73

	1 – version 2.5
	2-Version 2.6
	3-Version 3.1
	4-reserved
DexcCode	Indicate the reason for the most recent exception in Debug mode
NoSSt	Indicates whether single step interrupts are supported
	0-supported
	1-not supported
SSt	Single step interrupt enable bit
	0-not available
	1-enable
DINT	Set to indicate that a Debug interrupt exception occurred
	Automatically cleared when entering Debug mode
DIB	When set, indicates that an exception to the Debug instruction interrupt occurred
	Automatically cleared when entering Debug mode
DDBS	When set, indicates that a Debug data interrupt exception occurred
	Automatically cleared when entering Debug mode
DBp	Set to indicate that a Debug breakpoint exception occurred
	Automatically cleared when entering Debug mode
DSS	Set to indicate a Debug single-step exception

Automatically cleared when entering Debug mode

The bit or field in the Debug register is updated only when a debug exception or an exception occurs in debug mode.

3.27. Debug Exception Program Counter register (24, 0)

Debug Exception Program Counter (DEPC) is a 64-bit read / write register, which includes the

Continues to process addresses. This register is updated by hardware in debug exception or debug mode.

For precise debug exceptions and exceptions in precise debug mode, the contents of the DEPC register are one of the following:

- Instruction virtual address, which is the direct cause of the exception, or
- Previous branch or jump instruction (When the instruction is in the branch delay slot, the instruction delay bit DBD is sent in Debug

Virtual address).

Figure 3-29 shows the format of the DEPC register.

63 0

DEPC

64

Figure 3-29 DEPC register

3.28. Performance Counter Register (25, 0/1/2/3)

Godson 3 processor defines four (two sets) performance counters, which are mapped to CP24 register 24

Sel 0, sel 1, sel 2 and sel 3.

When Godson-3 is reset, the initial values assigned to the two control registers of the PerfCnt register are:

PerfCnt, select 0 = 0xc0000000

PerfCnt, select 2 = 0x40000000

The purpose of these four registers is shown in Table 3-25 , and the format o in Figure 3-30.

Same), the definition of the enable bit of the control register is shown in Table 3-26 .

Table 3-25 Performance counter list

Performance counter	sel	Use description
0	select 0	Control register 0
	select 1	Count register 0
1	select 2	Control register 1
	select 3	Count register 1

Each counter is a 64-bit read / write register and is incremented each time a countable event in the associated control domain occurs.

Each counter can count an event independently.

Page 75

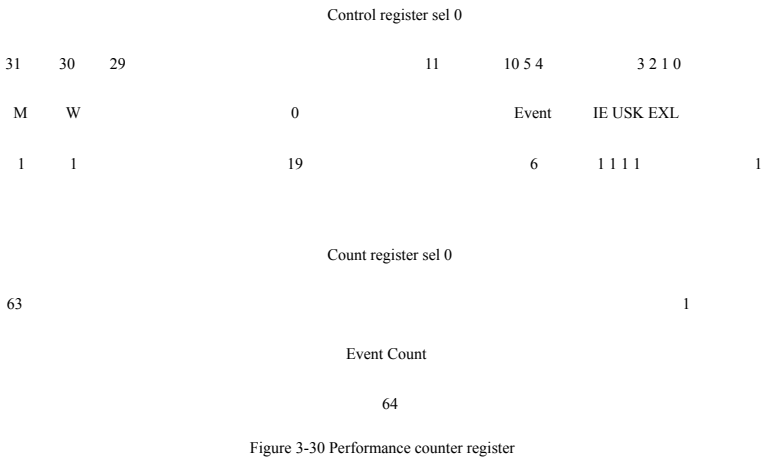


Figure 3-30 Performance counter register

When the first bit (63 bits) of the counter becomes 1 (counter overflow), the counter will trigger an interrupt.

The register PCI bit is set to one (if there are multiple sets of counters, the overflow bi nter overflows

Table 3-26 describes the respective events for Counter 0 and Counter 1.

Table 3-26 Count enable bit definition

Count enable bit	**Count Qualifier (CP0 Status** register field **)**
M	Is there another set of counters
	1 – yes
	0 – no
W	Count register bit width
	0 – 32 bits
	1-64 bits
K	KSU = 0 (kernel mode), EXL = 0, ERL = 0
S	KSU = 1 (Super User Mode), EXL = 0, ERL = 0

60

Page 76

U	KSU = 2 (normal user mode), EXL = 0, ERL = 0
EXL	EXL = 1, ERL = 0

Table 3-27 Counter 0 events

event	signal	description
0000	Cycles	cycle
0001	Brbus.valid	Branch instruction
0010	Jrcount	JR instructions
0011	Jr31count	JR instruction and domain rs = 31
0100	Imemread.valid & imemread_allow	L1 I-cache is missing
0101	Rissuebus0.valid	Alu1 operation has been launched
0110	Rissuebus2.valid	Mem operation launched
0111	Rissuebus3.valid	Falu1 operation has been launched
1000	Brbus_bht	BHT guess instruction
1001	Mreadreq.valid & Mreadreq_allow	Read from main memory
1010	Fxqfull	Fixed transmit queue is full
1011	Roqfull	Reorder queue is full
1100	Cp0qfull	CP0 queue is full
1101	Exbus.ex & excode = 34,35	Tlb refill exception
1110	Exbus.ex & Excode = 0	exception
1111	Exbus.ex & Excode = 63	Internal exception

Table 3-28 Counter 1 events

event	signal	description
0000	Cmtbus.valid	Commit operation
0001	Brbus.brerr	Branch prediction failed
0010	Jrmiss	JR prediction failed
0011	Jr31miss	JR and rs = 31 prediction failed
0100	Dmemread.valid & Dmemread_allow	L1 D-cache is missing
0101	Rissuebus1.valid	Alu2 operation has been launched
0110	Rissuebus4.valid	Falu2 operation has been launched
0111	Duncache_valid & Duncache_allow	Access is not cached

1000	Brbus_bhtmiss	BHT Guess Error
1001	Mwritereq.valid & Mwritereq_allow	Write to main memory
1010	Ftqfull	Floating-pointer queue is full
1011	Brqfull	Branch queue full
1100	Exbus.ex & Op == OP_TLBPI	Itlb is missing
1101	Exbus.ex	Total number of exceptions
1110	Mispec	Missing loading speculation
1111	CP0fwd_valid	CP0 queue load forward

3.29. ECC Register (26, 0)

Godson 3 uses the optional No. 26 ErrCtl register 'or ECC verification.

Figure 3-31 shows the format of the DEPC regist(ıe fields of the ECC register.

62

Page 78

Godson 3B1500 Processor User Manual • Volume 2

Figure 3-31 ECC register

Table 3-29 Fields of the ECC Register

area	description
0	Reserved. Must be written as 0 and read as 0.
ECC	A double-word checksum of the relevant cache

3.30. CacheErr Register (27, 0/1)

Loongson 3 performs the ECC verification of the Loongson 3 in the MIPS64 standard by hardware and software. The hardware is only responsible for checking Check for errors, and save the contents in the control registers such as CacheErr after checking for data errors. Concurrency exceptions are performed by software. Correct errors.

Table 3-30 describes the fields of the CacheErr 1(/s the format of the CacheErr1

Describes the fields of the CacheErr1 register.

63 4 5 twenty one 0

0 ECCWay ECCType

58 4 2

Figure 3-32 CacheErr register

Table 3-30 CacheErr Register Fields

area	description
63	

0	Reserved. Must be written as 0 and read as 0.
ECCWay	Different encodings indicate different errors in the cache when the cache is incorrectly checked
ECCType	00-Instruction cache error
	01-Data cache error
	10-secondary cache error
	11-chip interface bus error

63 0

ECCAddr

64

Figure 3-33 CacheErr1 register

Table 3-31 CacheErr1 register fields

area	description
ECCAddr	Virtual address with parity error

3.31. TagLo (28) and TagHi (29) registers

The TagLo and TagHi registers are 32-bit read / write registers used to save tags and status of the primary / secondary cache. Use the CACHE and MTC0 instructions to write to the Tag register.

Figure 3-34 shows the format of these registers for the first level cache (PCache) op ıe TagLo And the definition of the fields in the TagHi register.

64

Page 80

TagLo register

31	8	7 6	5 4	3 0
PTAG [23: 0]		CS	SCSETI	0
twenty four		2	2	4

TagHi register

31	12	11	0
0		PTAG [35:24]	
20		12	

Figure 3-34 TagLo and TagHi registers (PCache)

Table 3-32 Cache Tag register field (PCache)

area	description
PTAG	Specify the 47:12 bits of the physical address.
CS	Cache row status (0 is invalid, 1 is shared, 2 is exclusive, 3 is dirty).
SCSETI	Group number corresponding to the cache line in the secondary cache (the level of the secondary cache is 0)
0	Reserved. Must be written as 0 and read as 0.

Figure 3-35 shows the format of these registers for secondary cache (SCache) operat gLo

And the definition of the fields in the TagHi register.

TagLo register

31	13	12	11	10	9	7	0
STAG [18: 0]		DIR_STATE	W	CS	BITVECT		0
19		1	1	1	1		8

TagHi register

31	12	11	0
0		STAG [30:19]	
20		12	

Figure 3-35 TagLo and TagHi registers (SCache)

65

Page 81

Table 3-33 Cache Tag register field (SCache)

area	description

STAG	Specify the 47:17 bits of the physical address.
DIR_STATE	Cache line directory status (0 is shared, 1 is exclusive).
W	Whether the cache line is dirty (already written).
CS	Cache row status (0 is invliad, 1 is valid).
BITVECT	For the cache11 instruction, 0 means clear the directory and 1 means don't change the directory. For cache7 instructions, read back the value Is 0.
0	Reserved. Must be written as 0 and read as 0.

3.32. DataLo (28,1) and DataHi (29,1) registers

DataLo and DataHi are read-only registers and are only used to interact with and diagnose the cache data queue. CACHE instruction IndexLoadData operation will read the corresponding data into the DataLo or DataHi register . Figure 3-36DataLo Register and DataHi register format.

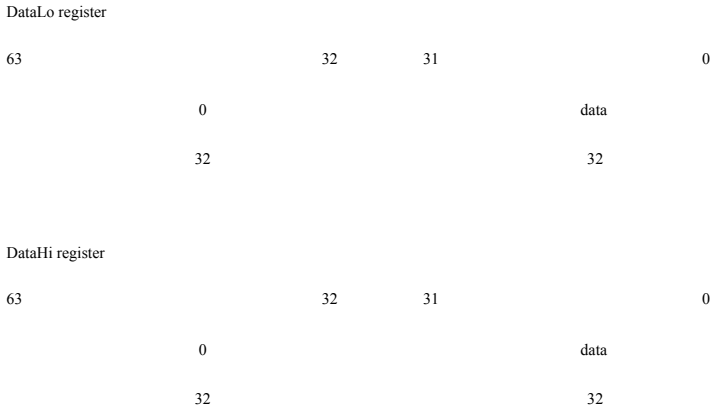


Figure 3-36 DataLo and DataHi registers

3.33. ErrorEPC Register (30, 0)

The ErrorEPC register is similar to the EPC register except for ECC and parity error exceptions. It is used during reset, Software reset, and non-maskable interrupt (NMI) exception store program counter.

ErrorEPC is a read-write register that contains the virtual address of the instruction to resume execution after processing an error. Figure 3-37 shows the format of the ErrorEPC register.

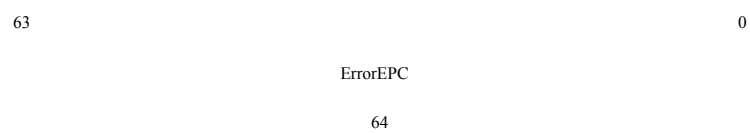


Figure 3-37 ErrorEPC register

3.34. DESAVE register (31, 0)

The DESAVE register is a read-write 64-bit register. His function is a simple staging Register, used to save the value of a general-purpose register during debugging Other contexts.

Figure 3-38 The format of the DESAVE register.

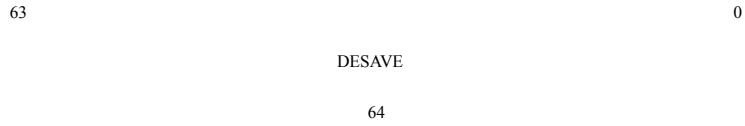


Figure 3-38 DESAVE register

3.35. CP0 instruction

Table 3-34 lists the CP0 instructions defined by the Godson-2 processor.

Table 3-34 CP0 instruction

OpCode	Description	MIPS ISA

DMFC0	Doubleword from CP0 register	III
DMTC0	Write double word to CP0 register	III
MFC0	Fetch from CP0 register	I
MTC0	Write to CP0 register	I
TLBR	Read TLB entries by Index	III
TLBWI	Write TLB entries by Index	III
TLBWR	Write TLB entries by Random	III
TLBP	Find the TLB Pair Index	III
CACHE	Cache operation	III
ERET	Abnormal return	III
DERET	Debug returns	EJTAG
DI	Close interrupt	MIPS32 R2
EI	Enable interrupt	MIPS32 R2
RDHWR	Read hardware register	MIPS32 R2
RDPGPR	Read from shadow register	MIPS32 R2
WRPGPR	Write to shadow register	MIPS32 R2
SDBBP	Software breakpoint	EJTAG

Related

The Loongson processor can handle the pipeline related in the hardware, including the CP0 related and the fetch related, so the CP0 instruction and No NOP instruction is required to correct the instruction sequence.

68

4. Organization and operation of CACHE

Godson-3 uses three separate caches:

First level instruction cache: a total of 64KB, the cache line size is 32 bytes, and a four-way group associative structure is used.

Level 1 data cache: a total of 64KB, a cache line size of 32 bytes, a four-way group associative structure, and write-back Strategy.

Two-level hybrid cache: on-chip cache, which communicates with the processor core through a 128-bit AXI bus, and has a total of four secondary caches Module. Each secondary cache module is globally addressed. The cache line size is 32 bytes and the capacity is 1M. Structure, using a write-back strategy.

4.1. Cache Overview

It takes 3 clock cycles for a fixed-level access to the primary cache, and 4 clock cycles for a floating-point access to the primary cache. period. Each primary cache has its own data path so that two caches can be accessed simultaneously. First level cache The read, write, and refill paths are all 128 bits.

The secondary cache uses a 256-bit data path and is accessed only when the primary cache fails. Secondary The cache and the primary cache cannot be accessed at the same time. When the primary cache fails, access to the secondary cache will increase by at least 14 The cost of failure of each cycle Takes an extra 6 shots). The secondary cache backfills the primary cache at a rate of 128 bits of data per clock cycle.

The primary cache uses the virtual address index and the physical address mark, while the secondary cache uses the physical index and mark. 理 Address. The virtual address index may cause inconsistencies. At present, Godson 2F uses the operating system to solve it. In the future, Solved by hardware.

Godson-3 uses a directory-based cache consistency protocol to ensure that the writes of each processor core and IO can be performed by other The processor core and IO are correctly observed. A directory is maintained in the secondary cache. Cache in each secondary cache Line, the directory uses a 32-bit bit vector to record whether each level cache (including instruction and data cache) owns the cache Backup of the line. Therefore, Godson-3 uses hardware to maintain a first-level instruction cache, a first-level data cache, and a second-level cache. And consistency between HT external devices, the software does not need to flash the cache through the Cache instruction to maintain consistency.

4.1.1. Non-blocking Cache

Godson 3 implements non-blocking cache technology. Non-blocking cache is by allowing cache invalidation fetch operation

69

Multiple cache invalidation or hit fetch operations continue to improve the overall performance of the system.

In a blocking cache design, when a cache failure occurs, the processor will suspend subsequent fetch operations.

At this point, the processor starts a storage cycle, fetches the requested data, fills it into the Cache, and then resumes execution.

This operation process will take more clock cycles, depending on the design of the memory system.

However, in a non-blocking cache design, the cache does not pause on a certain invalidation. Godson 3 supports multiple failures

The next hit, it can support up to 24 cache invalidations.

When the primary cache is invalid, the processor will check the secondary cache to see if the required data is in it.

If it still fails, you need to access the main memory.

The non-blocking Cache structure in Godson 3 can use loop unrolling and software pipeline more effectively. In order to be as large as possible

Use the advantages of Cache to the limit, and execute the corresponding Load operation as early as possible before using the instruction to fetch data.

For those I / O systems that require sequential access, the default setting of Godson 3 is to use blocking Uncached access.

Ask the way.

Godson 3 provides a prefetch instruction, which can prefetch data to one by loading to the fixed point register 0.

Level data cache. In addition, the DSP engine in Godson 3 can prefetch the data in memory or IO into the secondary cache.

4.1.2. Replacement Strategy

Both the primary and secondary caches use random replacement algorithms. However, the secondary cache provides a locking mechanism. By configuration

Lock window register to ensure that up to 4 locked areas are not replaced with secondary cache

Chapter 12).

4.1.3. Cache Parameters

Table 4-1 shows some parameters of the three caches.

Table 4-1 Cache parameters

parameter	Instruction **cache**	Data **Cache**	Secondary **cache**
Cache size	64KB	32KB	1MB (total 4MB)
Connectivity	4-way group connected	2-way group connected	4-way group connected
Replacement strategy	Stochastic method	Stochastic method	Random method (lockable)
Block size	32 bytes	32 bytes	32 bytes

70

Index	Virtual address 13: 5 bits	Virtual address 13: 5 bits	Physical address 16: 5 bit 1
Tag	Physical address 47:12 bits	Physical address 47:12 bits	Physical address 47:12 bits
Write strategy	Not writeable	Write back	Write back
Read strategy	Non-blocking (2 simultaneous)	Non-blocking (24 simultaneous)	Non-blocking (8 simultaneous)
Reading order	Keyword first	Keyword first	Keyword first
Write order	Not writeable	Sequential	
Checking means	Parity	ECC check	ECC check

4.2. First level instruction cache

The size of the first-level instruction cache is 64KB, and the four-way group associative structure is adopted. Cache block size (also commonly called (Cache line) is 32 bytes, which can store 8 instructions. Since Godson 3 uses a 128-bit read path, every

Four clock cycles can take four instructions to the superscalar scheduling unit.

The level 1 instruction cache implements parity. When a parity error occurs when reading the level 1 instruction cache, the hardware will

The corresponding cache line will be invalidated and the correct value will be obtained from the secondary cache. No software required for the entire process

Pre .

4.2.1. Organization of Instruction Cache

Figure 4-1 shows the organizational structure of the first-level instruction cache. The cache uses a four-way group-associated mapping method.

Each group in the group includes 512 index entries. Select the corresponding tag and data according to the index. Read from Cache

After the tag is issued, it is used to compare the converted part of the virtual address to determine the group containing the correct data.

When the first-level instruction Cache is indexed, the four groups will return their corresponding Cache rows with a cache row size of 32.

In bytes, the cache line uses 34 bits as a flag and 1 bit as a valid bit.

1 Which one of the four secondary cache modules a physical address belongs to depends on the routing configuration register of the crossbar.

71

Figure 4-1 Organization of instruction cache

4.2.2. Instruction cache access

The Godson-3 instruction cache uses a four-way associative structure of a virtual address index and a physical address mark. As shown in Figure 4-2 The lower 14 bits of the address are used as the index of the instruction cache. The 13: 5 bits are used to index 512 entries. Each item It contains four 64-bit doublewords, and 4: 3 bits are used to select among them.

When indexing the Cache, the Data in the four blocks and the corresponding physical address Tag are taken from the Cache. At the same time, The high-order address is converted by the instruction TLB (Instruction Translation Look-aside Buffer, ITLB). The changed address is compared with the tags in the four groups taken out. If there is a tag matching it, the group is used. The data. This is called a "first level cache hit". If none of the four groups of tags match, then abort Operation, and start to access the secondary cache. This is called "level 1 cache miss".

Figure 4-2 Instruction cache access

4.3. Primary Data Cache

The data cache has a capacity of 32KB and adopts a four-way group associative structure. Cache block size is 32 bytes, which means that Store 8 words. The read and write data paths of the data cache are both 128 bits.

The data cache uses a virtual address index and a physical address flag. The operating system can resolve what may be caused by virtual addresses Page coloring consistency issues. The data cache is non-blocking, which means that an invalidation in the data cache will not Causes a pause in the pipeline.

The write strategy adopted by the data cache is write-back, that is, the operation of writing data to the primary cache will not cause the secondary cache. And main memory updates. The writeback strategy reduces the amount of traffic from the primary cache to the secondary cache, thereby improving global performance. Only when the data cache line is replaced will the data be written to the secondary cache.

The primary data cache implements ECC check. When a one-bit ECC check error occurs while reading the first-level data cache, The file will automatically correct the results read by the cache and update the contents of the cache to the corrected values. No software required for the entire process Intervention. When a two-bit ECC check error occurs while reading the first-level data cache, an exception will occur and left to the software for processing.

4.3.1. Organization of Data Cache

Figure 4- 3 shows the organizational structure of the data cache. This is a two-way associative cache, which contains 512 The index entry. When indexing the cache, access the Tag and Data in both groups at the same time. Then compare the tags in the two groups with The converted physical address portions are compared to determine which data line is hit.

When indexing the data cache, both groups return their respective cache rows. Cache block size is 32

Byte, the cache line uses 34 bits as the physical flag address, 1 bit as the dirty bit and 2 bits as the status bit (including INV,

SHD and EXC). The INV status indicates that the cache line is invalid, the SHD status indicates that the cache line is readable,

The EXC status indicates that the cache line is readable and writable.

73

Figure 4-3 Organization structure of data cache

4.3.2. Data Cache Access

The Godson-3 data cache uses a two-way group associative structure of a virtual address index and a physical address mark. Figure 4-4 shows

How to resolve the virtual address when accessing the data cache once.

Figure 4-4 Data cache access

As shown in Figure 4-4 , the lower 14 bits of the address are used as an index into the data cache. 13: 5 of which are used as indexes 512

Entries, each of which includes 4 64-bit doublewords. Use 4: 3 bits to select four double words, and 2: 0 bits to select

One of the eight bytes of a doubleword.

Data Cache access invalid instruction (fetch instruction miss or write instruction hit the cache line of SHD state), then

Access the secondary cache. If the secondary cache hits, the cache block retrieved from the secondary cache is sent back to the primary cache.

If the secondary cache fails, the memory is accessed and the secondary cache and data cache are populated with the values retrieved from the internal cache.

74

4.4. Secondary Cache

Godson 3B1500 includes 8 on-chip secondary cache modules. The capacity of each secondary cache module is 512KB.

4MB total. Each cache line is 32 bytes in size. Key features of the secondary cache module include: 128-bit AXI

Interface, four-way group connection, 8 items cache access queue, keyword priority, receive read invalidation request to return data as fast as 8

Shoot, support Cache consistency protocol through the directory, can be used for on-chip multi-core structure (also can be directly connected with uniprocessor IP),

Soft IP level can be configured with the size of a secondary cache module (512KB / 1MB), adopts a four-way group connection structure, and can be moved during operation

Status off, support ECC check, support DMA consistent read and write and pre-fetch read, support 16 secondary cache methods,

Supports locking the secondary cache by window to ensure that the read data returns atomicity.

The secondary cache also maintains a directory for each cache line to record whether each primary cache contains the cache

Backup of the line. The write strategy adopted by the secondary cache is write-back. Write-back strategy reduces bus traffic and therefore improves

Global performance of the system. Data is written to memory only when the secondary cache line is replaced.

The secondary cache implements ECC check. When a one-bit ECC check error occurs when reading the secondary cache, the hardware will automatically

Correct the result read by the cache and update the contents of the cache to the corrected value. No software intervention is required for the entire process. when

When a two-bit ECC check error occurs while reading the secondary data cache, an exception will occur and left to the software for processing.

4.4.1. Organization of the secondary cache

The secondary cache is a hybrid cache, which includes both instructions and data. Secondary cache module supports cache

Consistency agreement. All on-chip secondary caches in Loongson 3 are uniformly addressed, and each secondary cache block has a fixed home

Node. According to the requirements of cache consistency, Godson-3's secondary cache has two roles:

For home, it is Cache for memory. When accessing the secondary cache, four groups of Data and Tag are accessed at the same time.

Compare the four tags taken out with the high-order part of the accessed physical address to determine whether the data still resides in

Cache.

Each Cache line contains a 32-byte data, a 31-bit physical address flag, and a 1-bit Cache status bit (table

Shows whether the corresponding cache line is valid in the secondary cache, 1-bit directory status bit (indicates whether the corresponding cache block is

Exclusive or shared in a level of cache) and 1 W bit (indicating whether the row has been written).

4.4.2. Access to the secondary cache

The secondary cache is accessed only if the primary cache fails. The secondary cache uses a physical address

75

Lead physical address flag. As shown in Figure 4-5 , the lower address is used to index the secondary cache. They are returned in each of the four groups.

From the corresponding Cache line. 16: 5 bits are used as an index into the secondary cache. Each indexed item contains four 64-bit doubles

Word data. Use 4: 3 bits to select among 4 double words. Bits 2: 0 are used to select an 8 byte in a double word.

Figure 4-5 Secondary cache access

4.5. Cache Algorithm and Cache Consistency Properties

Loongson 3 implements the cache algorithm and cache consistency attributes shown in Table 4-2 .

Table 4-2 Consistency attributes of Godson 3 cache

Attribute classification	Consistent code
Keep	0
Keep	1
Uncached	2
Cacheable coherent	3
Keep	4
Keep	5
Keep	6
Uncached Accelerated	7

76

4.5.1. Uncached (Coherence Code 2)

If a page uses a non-cached algorithm, then for a Load or Store operation anywhere in the page,

The processor directly sends a double word, partial double word, word, or partial word read or write request to the main memory without passing any

Level Cache. Non-cached algorithms are implemented in a blocking manner.

4.5.2. Coherent Cache (Coherent Code 3)

A row with this attribute can reside in the cache, and the corresponding store and fetch operations only access the first-level cache.

When the primary cache fails, the processor checks the secondary cache to see if it contains the requested address. If the second level

Cache hit, the data is populated from the secondary cache. If the secondary cache misses, the data is taken from the main memory.

And write it to the secondary cache and primary cache.

Because there are multiple processor cores and IO devices in Godson 3 that can access the main memory, Godson 3 hardware implements

Cache consistency protocol, no need to adopt software to use cache instructions to actively maintain cache consistency .

4.5.3. Uncached Accelerated (Consistency Code 7)

Non-cache-accelerated properties are used to optimize a sequence of the same type done in a continuous address space

Uncached store operation. The optimization method is to set the buffer to collect the storage operation of this attribute. Just slowly

If the punching area is not full, the data of these store operations can be stored in the buffer. The buffer is the same size as a cache line.

Storing data in a buffer is the same as storing it in a cache. When the buffer is full, a block write begins. In Shun

During the collection of sequential storage instructions, if other types of Uncached storage instructions are inserted, the collection is suspended and the buffering is performed.

The data stored in the area is output as a byte write.

The non-cache-accelerated attribute can speed up sequential Uncached access, and it is suitable for fast

Output access.

4.6. Cache Consistency

Godson-3 achieves directory-based cache consistency. The hardware guarantees a level 1 instruction cache, a level 1 data cache,

Data consistency between secondary cache, memory, and IO devices from HT, no software is required to use Cache instruction to

Forcibly flash the cache.

Each cache line in Godson 3 has a fixed host secondary cache module. The directory information of the cache line is

77

Page 93

Godson 3B1500 Processor User Manual • Volume 2

Maintained in the host secondary cache module. The directory utilizes a 32-bit bit vector to record one copy of each cache line backup.

Level Cache (including level 1 instruction cache and level 1 data cache). There are three possible states for each level of cache block:

INV (invalid state), SHD (shared state, readable), and EXC (exclusive state, readable and writable). Three states

The transfer situation is shown in Figure 4-6. When a read or instruction fetch occurs in the primary cache, the processor core moves to the secondary cache

Issue a Reqread request. After receiving the Repread response from the secondary cache module, the primary cache of the processor core

Obtained a cache backup in the SHD state; when the write instruction fails the primary cache, the processor core moves to the secondary cache

The module issues a Reqwrite request. After receiving the Repwrite response from the second-level Cache module, the first level of the processor core

Cache obtained a cache backup of EXC status; when the first-level cache replacement occurs on the processor core,

Reqreplace writes back the secondary cache module, and the secondary cache module informs the processor core to replace the request through the Repreplace response.

The request has been processed. The secondary cache module can invalidate an SHD state by sending a Reqinv request to the processor core.

First level cache backup, the processor core changes the first level cache backup to the INV state and responds to the second level cache through Repinv

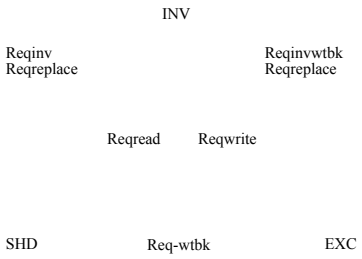
Module; Level 2 Cache module can write a Level 1 EXC status by sending a Reqwtbk request to the processor core

Cache backup, the processor core changes the primary cache backup to the SHD state and responds to the secondary cache through Repwtbk

Module; the secondary cache module can write back and invalidate an EXC status by sending a Reqinvwtbk request to the processor core

State of the first-level cache backup, the processor core changes the first-level cache backup to the INV state and responds with Repinvwtbk

Secondary cache module.



78

Figure 4-6 Godson 3 cache state transition

4.7. Cache directive

GS464 supports 17 kinds of cache instructions, which are aimed at the primary data cache, the primary instruction cache, and the secondary cache. The format of the Cache instruction is: CACHE fmt, offset (base).

The specific instruction list is as follows：

Table 4-3 GS464 cache consistency properties

FMT domain	Meaning of **Cache** instruction	Target **cache**
5'b00000	Invalidate cache block based on index	L1 instruction cache
5'b 01000	Store tags of cache block according to index	L1 instruction cache
5'b 11100	Store the data of the cache block according to the index	L1 instruction cache
5'b 00001	Invalidate cache block based on index and write back	Level 1 data cache
5'b 00101	Get the tag of the cache block according to the index	Level 1 data cache
5'b 01001	Store tags of cache block according to index	Level 1 data cache
5'b 10001	Invalidate cache blocks based on hits	Level 1 data cache
5'b 10101	Invalidate cache block based on hit and write back	Level 1 data cache
5'b 11001	Get the data of the cache block according to the index	Level 1 data cache
	Store the data of the cache block according to the index	Level 1 data cache

5'b 11101		
5'b 00011	Invalidate cache block based on index and write back	Secondary cache
5'b 00111	Get the tag of the cache block according to the index	Secondary cache

79

Page 95

5'b 01011	Store tags of cache block according to index	Secondary cache
5'b 10011	Invalidate cache blocks based on hits	Secondary cache
5'b 10111	Invalidate cache block based on hit and write back	Secondary cache
5'b 11011	Get the data of the cache block according to the index	Secondary cache
5'b 11111	Store the data of the cache block according to the index	Secondary cache

4.7.1. Cache0 instruction

The function of the Cache0 instruction is to invalidate the first-level instruction cache block according to the index. Specifically, the cache index A level 1 instruction cache block equal to address [13: 5] and cache way equal to address [2: 0] is invalid.

4.7.2. Cache8 instruction

The function of the Cache8 instruction is to store the tag of the first-level instruction cache block according to the index. Specifically, it is the CP0 The values in the TAGHi and TAGLo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to address [2: 0] in the tag of the first-level instruction cache block. For the specific content format of TAGHi and TAGLo, see Chapter 3.

4.7.3. Cache28 instruction

The function of the Cache28 instruction is to store the data of the first-level instruction cache block according to the index. Specifically, CP0 The values in the DATAHi and DATALo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to The address [2: 0] of the first-level instruction cache block of the address [4: 3] 64 bits of data. Specific DATAHi and DATALo The content format is described in Chapter 3.

4.7.4. Cache1 instruction

The function of the Cache1 instruction is to invalidate the first-level data cache block according to the index. Specifically, the cache index Level 1 data cache block equal to address [13: 5] and cache way equal to address [2: 0] are invalidated, and the contents inside are lost Discard.

4.7.5. Cache5 instruction

The role of the Cache5 instruction is to fetch the tag of the first-level data cache block according to the index. Specifically, the cache index is equal to address [13: 5], and cache way is equal to address [2: 0].

80

Page 96

In the TAGHi and TAGLo registers of CP0. For the specific content format of TAGHi and TAGLo, see Chapter 3.

4.7.6. Cache9 instruction

The role of the Cache9 instruction is to store the tag of the first-level data cache block according to the index. Specifically, it is the CP0 The values in the TAGHi and TAGLo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to address [2: 0] in the tag of the primary data cache block. For the specific content format of TAGHi and TAGLo, see Chapter 3.

4.7.7. Cache17 instruction

The purpose of the Cache17 instruction is to invalidate a level 1 data cache block based on an address hit. Specifically, if the cache The address targeted by the instruction is hit in the first-level data cache, and the corresponding first-level data cache block is invalidated. The contents are discarded; if the address targeted by the cache instruction does not hit in the primary data cache, no operation is performed.

4.7.8. Cache21 instruction

The role of the Cache21 instruction is to invalidate and write back the first-level data cache block according to the address hit. Specifically, if The address targeted by the cache instruction is hit in the first-level data cache, and the corresponding first-level data cache block is invalidated. If the cache block is dirty, the contents will be sent back to the secondary cache or memory (if there is no secondary cache); If the address targeted by the cache instruction is missed in the primary data cache, no operation is performed.

4.7.9. Cache25 instruction

The function of the Cache25 instruction is to fetch the data of the first-level data cache block according to the index. Specifically, the cache index is equal to address [13: 5], cache way is equal to address [2: 0] address [4: 3] 64 bits are fetched into the DATAHi and DATALo registers of CP0. Specific DATAHi and DATALo The content format is described in Chapter 3.

4.7.10. Cache29 instruction

The function of the Cache29 instruction is to store the data of the first-level data cache block according to the index. Specifically, CP0 The values in the DATAHi and DATALo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to The address [2: 0] in the 64th bit of address [4: 3] of the data of the primary data cache block. Specific DATAHi and DATALo The content format is described in Chapter 3.

4.7.11. Cache3 instruction

The Cache3 instruction is used to invalidate the secondary cache block based on the index. Specifically, the cache index, etc. At address [13: 5], the secondary data cache block with cache way equal to address [2: 0] is invalidated, and the contents are discarded Off.

4.7.12. Cache7 instruction

The role of the Cache7 instruction is to get the tag of the secondary cache block according to the index. Specifically, the cache index

The content of the tag of the secondary data cache block equal to address [13: 5] and cache way equal to address [2: 0] is fetched to CP0

In the TAGHi and TAGLo registers. For the specific content format of TAGHi and TAGLo, see Chapter 3.

4.7.13. Cache11 instruction

The function of the Cache11 instruction is to store the tags of the secondary cache block according to the index. Specifically, it is the CP0

The values in the TAGHi and TAGLo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to

in the tag of the secondary cache block of address [2: 0]. For the specific content format of TAGHi and TAGLo, see Chapter 3.

4.7.14. Cache19 instruction

The purpose of the Cache19 instruction is to invalidate the secondary cache block based on the address hit. Specifically, if the cache instruction

The target address is hit in the secondary data cache, the corresponding secondary data cache block is invalidated, and the contents inside are lost

Discard; if the address targeted by the cache instruction misses in the secondary cache, do nothing .

4.7.15. Cache23 instruction

The role of the Cache23 instruction is to invalidate and write back to the secondary cache block based on the address hit. Specifically, if the cache

The address targeted by the instruction is hit in the secondary cache, the corresponding secondary cache block is invalidated, and if the cache

The block is dirty, and the contents will be sent back to the secondary cache or memory (if there is no secondary cache); if the cache instruction

The target address is not hit in the secondary cache, and no operation is performed.

4.7.16. Cache27 instruction

The role of the Cache27 instruction is to fetch the data of the secondary cache block according to the index. Specifically, the cache

index is equal to address [13: 5], cache way is equal to address [2: 0]

address [4: 3] 64-bit contents are fetched into the DATAHi and DATALo registers of CP0. Specific DATAHi and DATALo

The content format is described in Chapter 3.

4.7.17. Cache31 instruction

The function of the Cache31 instruction is to store the data of the secondary cache block according to the index. Specifically, it is the CP0

The values in the DATAHi and DATALo registers are stored in the cache index equal to address [13: 5], and the cache way is equal to

address [2: 0] in the 64th bit of address [4: 3] of the data of the secondary cache block. Specific DATAHi and DATALo

The content format is described in Chapter 3.

83

Page 100

5. Memory Management

Loongson GS464 processor core provides a full-featured memory management unit (MMU), which uses the on-chip TLB implements the translation from virtual address to physical address.

This section describes the virtual address and physical address space of the processor core, the conversion of virtual address to physical address, TLB Operations to implement these transformations, cache, and system control coprocessor (CP0) that provides a software interface to the TLB register.

5.1. Quick Lookup Table TLB

Mapping virtual addresses to physical addresses is usually implemented by the TLB (there are also virtual address translations without TLB, For example, the CKSEG0 and CKSEG1 kernel address space segments (see Figure 5-5) are not page mapped. The physical address is obtained by subtracting a base address from the virtual address.). The first level of TLB is JTLB, which is also used as data TLB, in addition, the Godson GS464 processor core contains a separate instruction TLB to ease competition with JTLB.

5.2. JTLB

In order to be able to quickly map the virtual address to the physical address, the Godson GS464 processor core uses a larger, Fully associative mapping TLB, JTLB is used for address mapping of instructions and data, using their process numbers and virtual addresses Row index.

JTLB is organized in pairs of odd / even entries, mapping the virtual address space and address space identifiers to the physical address of 256T Address space. By default, JTLB has 64 pairs of even / even entries, allowing 128 pages to be mapped.

There are two mechanisms to help control the size of the mapping space and the replacement strategy for different areas of memory.

First, the page size can be 4KB to 16MB, but it must be increased by 4 times. For CP0 register PageMask The size of the page mapped to the record, and this record is loaded into the TLB while writing a new entry. Godson GS464 Processor cores can support pages of different sizes at the same runtime, allowing the operating system to produce specific purpose mappings:

In video codec processing, the frame buffer can use only one entry for memory mapping.

Second, the Godson GS464 processor core can use a random replacement strategy to select the replacement when the TLB is missing.

Changed TLB entries. The operating system can also reside a certain number of pages in the TLB without being randomly replaced

In addition, this mechanism helps the operating system to improve performance and avoid deadlocks. This mechanism also makes real-time systems more convenient for

Certain critical software provides specific entry points.

85

In addition, for each page, JTLB also maintains the cache consistency property of the page, and each page has a specific bit to

Mark: Does not go through Cache (Uncached), Non-coherent Cache (Cacheable Noncoherent)

Accelerated (Uncached Accelerated).

5.2.1. Instruction TLB

Godson GS464 processor core instruction TLB (ITLB) has 16 entries, which minimizes the capacity of JTLB, and

A large associative array shortens the time critical path during mapping and reduces power. Each ITLB entry can only be mapped

Shoot a page, the page size is specified by the PageMask register. ITLB instruction address mapping and data address mapping can

Parallel execution, which improves performance. When an entry in ITLB becomes invalid, the corresponding entry is looked up from the JTLB, randomly

Select an ITLB entry for replacement. The operation of ITLB is completely transparent to the user. Processor Guarantees ITLB and JTLB

It is the same that when the JTLB is modified by the core state instruction, the ITLB will be automatically cleared.

5.2.2. Hits and failures

If the virtual address matches the virtual address of an entry in the TLB (that is, a TLB hit), the physical page number is changed from

Take out from TLB and connect with offset to form physical address.

If the virtual address is inconsistent with the virtual address of any entry in the TLB (that is, the TLB is invalid), the CPU generates a

Exception, and the software refills the TLB based on the page table stored in memory. Software can rewrite the specified TLB entries,

You can also rewrite any TLB entry using the mechanism provided by the hardware.

5.2.3. Multiple hits

The Loongson GS464 processor checks that the virtual address in the TLB is not the same as the virtual address of an entry.

This detection and disabling mechanism is unlike the design of earlier MIPS processors. Multiple hits are not physically destroyed

TLB, so the detection mechanism for multiple hits is unnecessary. But multiple hits are not defined, so the software

It's important not to let multiple hits happen.

5.3. Processor mode

Godson GS464 processor core has 3 working modes, but unlike other MIPS processors, Godson GS464

The processor core supports only one address mode, one instruction set mode, and one tail mode.

86

5.3.1. Processor operating modes

The processor priority of the following three modes is reduced in order:

- Kernel mode (highest system priority): In this mode, the processor can access and change any register, operation

  The innermost kernel of the operating system runs in kernel mode;

- Management mode: the priority of the processor is reduced, and some less critical parts of the operating system are running in this mode

- User mode (lowest system priority): This mode prevents different users from interfering with each other.

The three modes are switched by the operating system (in kernel mode) by setting the corresponding bits in the status register KSU field.

of. When an error (ERL bit is set) or an exception (EXL bit is set), the processor is forced to switch

Switch to kernel mode. Table 5-1 lists the settings of KSU, EXL, and ERL when the three modes are switched. Empty entries can be

So don't care.

Table 5-1 Working modes of the processor

KSU	ERL	EXL	description
4: 3	2	1	
10	0	0	User mode
01	0	0	Management model
00	0	0	Kernel mode
	0	1	Exception level
	1		Error level

5.3.2. Address mode

Loongson GS464 processor core only supports 64-bit virtual address mode, and the hardware is guaranteed to be compatible with 32-bit address mode

formula.

5.3.3. Instruction Set Mode

The Godson GS464 processor core implements a complete MIPS64R2 instruction set, and also adds some integers and floating points

Directives, see Appendix A and Appendix B for additional directives.

5.3.4. Tail-end mode

The Godson GS464 processor core only works in little-endian mode.

5.4. Address space

This section describes the virtual address space, physical address space, and the method of virtual-to-real address translation through TLB.

5.4.1. Virtual Address Space

The Godson GS464 processor core has three virtual address spaces: user address space, management address space, and kernel address Space, each space is 64 bits, and contains some discontinuous address space segments, the largest segment is 256T (2 48) words Section.

5.4 Section 5) describe these three address spaces.

5.4.2. Physical Address Space

By using a 48-bit address, the physical address space of the processor is 256 T (2 48) bytes. The following sections will detail the virtual Real address translation method.

5.4.3. Virtual and Real Address Translation

When performing virtual-to-real address translation, first compare the virtual address given by the processor with the virtual address stored in the TLB. when The virtual page number (VPN) is equal to the VPN domain of a TLB entry, and if any of the following two conditions hold:

- The global bit of the TLB entry is 1
- The ASID fields of the two virtual addresses are the same.

TLB hit it. If the above conditions are not met, the CPU will generate a TLB failure exception to enable the software to Refill the TLB according to the page table stored in memory.

If the TLB hits, the physical page number will be taken from the TLB and merged with the offset within the page Offset to form a product 理 Address. In-page offset Offset does not go through the TLB during the virtual-to-real address translation.

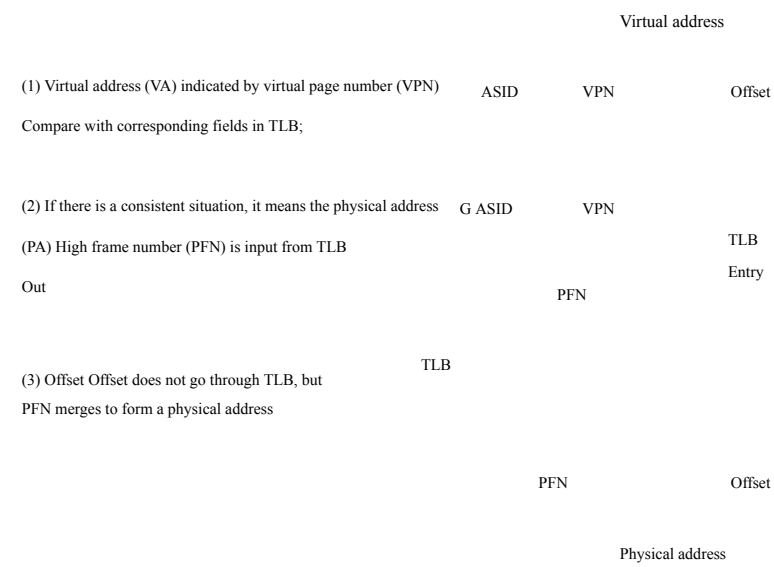


Figure 5-1 shows the virtual-to-real address translation. The virtual address is extended by an 8-bit address space identifier (ASID).

This measure reduces the frequency of TLB refreshes during context switching. The ASID is stored in the CP0 EntryHi register.

The Global bit (G) is in the corresponding TLB entry.

Figure 5- 2 shows the virtual-to-real address translation process in 64-bit mode. This figure shows the maximum page size of 16MB and the minimum page size.

In the case of 4KB.

The upper part of the figure shows the case where the page size is 4K bytes. The offset within the page occupies the

12 bits, the remaining 36 bits in the virtual address is the virtual page number VPN, which is used to index 64G page table entries;

The lower half of the figure shows the case where the page size is 16M bytes, the offset within the page Offset occupies the virtual address

24 bits in the virtual address, and the remaining 24 bits in the virtual address is the virtual page number VPN, which is used to index 16M page table entries.

89

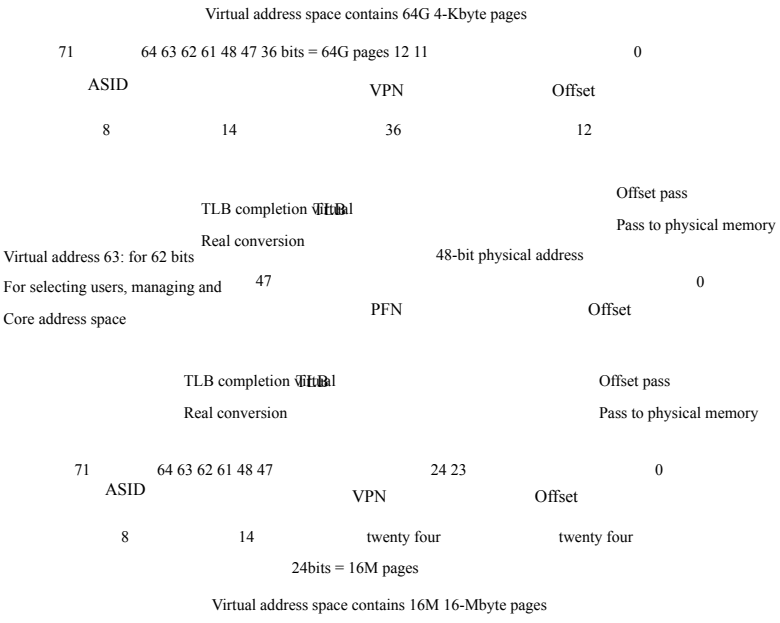


Figure 5-2 64-bit mode virtual address translation

5.4.4. User address space

In user mode, there is only a single, unified virtual address space called the User Segment.

Figure 5-3 shows the user virtual address space, which can be accessed in user mode, management mode, and kernel mode.

The user segment starts at address 0, and the currently active user process resides in this segment (XUSEG). In different modes

Next, the TLB maps XUSEG segments in the same way and controls whether the cache can be accessed.

When the value of the processor's Status register simultaneously meets three conditions: KSU = 10 2 , EXL = 0, ERL = 0, the processing

The processor works in user mode.

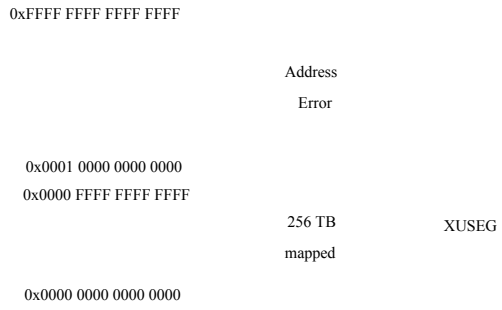


Figure 5-3 User virtual address space overview in user mode

The 63rd to 48th bits of the virtual address in all available user modes must be 0, accessing any 63rd bit

Addresses whose bits are not all 0 to the 40th bit will cause an address error exception. The TLB in the XUSEG address segment is missing and used.

XTLB refill vector. Godson GS464 processor core's XTLB refill vector and TLB refill vector in 32-bit mode

There are the same exception entry addresses.

5.4.5. Managing Address Space

The management mode is designed for a hierarchical operating system. In a layered operating system, the real kernel runs

In kernel mode, the rest of the operating system runs in management mode. Management address space provides management mode

Code and data space. Missing TLBs that manage the address space are handled by the XTLB refill processor.

Both management mode and kernel mode have access to the management address space.

When the value of the processor's Status register simultaneously meets three conditions: KSU = 01 2 , EXL = 0, ERL = 0, the processing

The processor works in management mode. Figure 5- 4 shows the user and management address space overview in management mode.

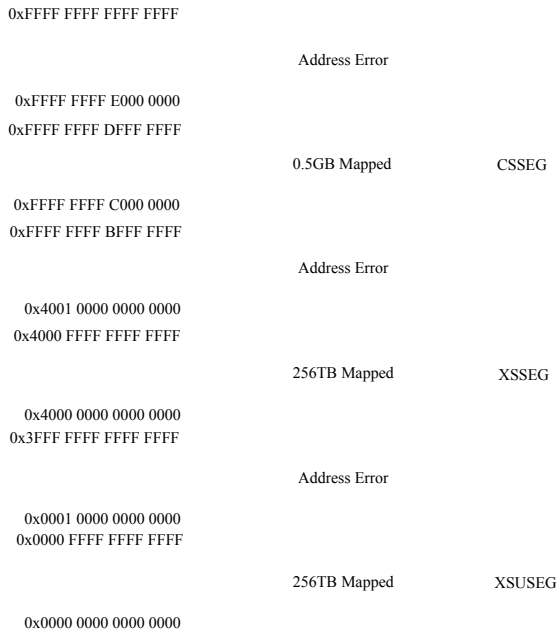


Figure 5-4 User space and management space in management mode

● 64-bit management mode, user address space (XSUSEG)

In management mode, when accessing the user address space and the most significant two bits (bits 63 and 62) of a 64-bit address are

At 00 2 the program uses a virtual address space named XSUSEG. XSUSEG covers the current user address space.

Of all 2 48 (1T) bytes. At this time, the virtual address is extended, and the 8-bit ASID field is added to form a unique address in the system.

Virtual address. This address space starts at 0x0000 0000 0000 0000 and ends at 0x0000 FFFF FFFF FFFF.

● 64-bit management mode, the current management address space (XSSEG)

In management mode, when the most significant two bits (bits 63 and 62) of a 64-bit address are 01 2 , the program uses one

The currently managed virtual address space named XSSEG. At this time, the virtual address is extended, and the 8-bit ASID field is added.

Into a unique virtual address in the system. This address space starts from 0x4000 0000 0000 0000 to 0x4000 FFFF

FFFF FFFF ends.

● 64-bit management mode, independent management of address space (CSSEG)

In management mode, when the most significant two bits (bits 63 and 62) of a 64-bit address are 11 2 , the program uses one

Independently managed virtual address space named CSSEG. Addressing in CSSEG and SSEG in 32-bit mode

Addressing is compatible. At this time, the virtual address is extended, and the 8-bit ASID field is added to form the only virtual address in the system.

site. This address space starts at 0xFFFF FFFF C000 0000 and ends at 0xFFFF FFFF DFFF FFFF.

5.4.6. Kernel Address Space

When the value of the Status register of the processor meets the following conditions: KSU = 00 2 or EXL = 1 or ERL = 1, the processor

Works in kernel mode.

Whenever the processor detects an exception, it enters kernel mode and stays on until the exception return instruction is executed.

(ERET). The ERET instruction restores the processor to the mode it was in before the exception occurred.

According to the high-order bits of the virtual address, the kernel-mode virtual address space is divided into different regions, as shown in Figure 5-5 .

● 64-bit kernel mode, user address space (XKUSEG)

In kernel mode, when accessing user space and the most significant two bits of the 64-bit virtual address are 00 2 , the program uses a

A virtual address space named XKUSEG, XKUSEG covers the current user address space. The virtual address is now

The extension, plus the 8-bit ASID field, forms a unique virtual address in the system.

● 64-bit kernel mode, currently managing the address space (XKSSEG)

In kernel mode , the program uses a name when accessing the management space and the most significant two bits of the 64-bit address are 01 2

The word is the virtual address space of XKSSEG, and XKSSEG is the virtual address space currently managed. The virtual address is now extended,

The 8-bit ASID field is added to form a unique virtual address in the system.

● 64-bit kernel mode, physical address space (XKPHY)

In kernel mode, when the most significant two bits of the 64-bit address are 10 2 , the program uses a virtual name called XKPHY.

In a pseudo-address space, XKPHY is a collection of eight 2 48- byte core physical address spaces. Visit any address from 58th to 58th

93

Page 109

Memory locations with 48 bits other than 0 will cause address errors. Access to XKPHY does not undergo address translation through the TLB.

Instead, use the 47th to 0th bits of the virtual address as the physical address. Bits 61 to 59 of the virtual address control whether or not to pass

The consistency properties of Cache and Cache have the same meaning as the C-bit value of the TLB page described in Table 3-2.

● 64-bit kernel mode, kernel address space (XKSEG)

In kernel mode, when the most significant two bits of a 64-bit address are 11 2 , the program uses one of the following two address spaces:

- Kernel virtual address space XKSEG. At this time, the virtual address is extended, and the 8-bit ASID field is added.

Into a unique virtual address in the system;

- Four 32-bit kernel-compatible address spaces, detailed in the next section.

● 64-bit kernel mode, compatible with address space (CKSEG1: 0, CKSSEG, CKSEG3)

In kernel mode, the most significant two bits of a 64-bit address are 11 2 and all bits 61 to 31 of the virtual address are

When equal to 1, one of the following four 512M byte address spaces used by the program, which one is according to the 30th, 29th

Bit decision:

94

Page 110

0xFFFF FFFF FFFF FFFF	0.5GB Mapped	CKSEG3
0xFFFF FFFF E000 0000		
0xFFFF FFFF C000 0000	0.5GB Mapped	CKSSEG
0xFFFF FFFF A000 0000	0.5GB Unmapped Cached	CKSEG1
0xFFFF FFFF 8000 0000	0.5GB Unmapped Cached	CKSEG0
0xC000 00FF 8000 0000	Address Error	XKSEG
0xC000 0000 0000 0000	Mapped	XKPHY
0x8000 0000 0000 0000	Unmapped	
0x4001 0000 0000 0000	Address Error	XKSSEG
0x4000 0000 0000 0000	256TB Mapped	

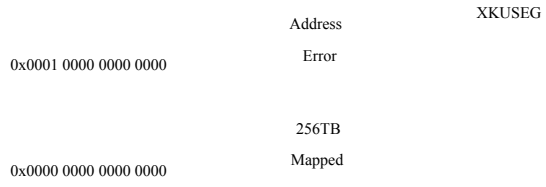


Figure 5-5 User, management, and kernel address space overview in kernel mode

- CKSEG0: This 64-bit virtual address space does not go through TLB and is compatible with KSEG0 in 32-bit mode.

The K0 field of the Config register controls whether the cache and cache consistency attributes are passed.

- CKSEG1: This 64-bit virtual address space does not pass through the TLB or Cache, and is in 32-bit mode.

KSEG1 is compatible.

- CKSSEG: The 64-bit virtual address space is the current management virtual address space, which is the same as that in 32-bit mode.

KSSEG compatible.

- CKSEG3: The 64-bit virtual address space is the kernel virtual address space, and KSEG3 in 32-bit mode

compatible.

5.5. System Control Coprocessor

The system control coprocessor (CP0) is responsible for supporting storage management, virtual-to-real address translation, exception handling, and some privileges operating. Godson GS464 processor core has 26 CP0 registers and a 64-entry TLB, each register is unique

Register number. The following sections give an overview of the registers related to memory management.

5.5.1. TLB entry format

Figure 5-6 shows the format of a TLB entry. Each field in the entry is in EntryHi, EntryLo0, EntryLo1, PageMask.

There are corresponding fields in the registers.

EntryHi, EntryLo0, EntryLo1, and PageMask registers are similar in format to TLB entries. The only one

The same is as a G vhich is not in the EntryHi register, but appears as a reserved field. Figure 5-7 ,

Figure 5-8 ε he fiel in Figure 5-6.

Page 112

255 217 216 205 204 192

0 Mask 0

39 12 13

191 190 189 176 175 141 140 139 136 135 128

R 0 VPN2 G 0 ASID

2 14 35 1 4 8

127 106 105 70 69 67 66 65 64

0 PFN CDV 0

twenty one 36 3 1 1 1

63 42 41 6 5 3 2 1 0

0 PFN CDV 0

twenty one 36 3 1 1 1

Figure 5-6 TLB entry

31 25 24 13 12 0

0 Mask 0

7 12 13

Mask ... page comparison mask

0 …… .. Reserved. Writes must be 0, and returns 0 when read.

Figure 5-7 PageMask register

97

Page 113

63 62 61 48 47 13 12 8 7 0

R Fill VPN2 0 ASID

2 14 35 5 8

VPN2 …. Virtual page number divided by 2 (maps 2 pages).

ASID …. Address space ID field. An 8-bit field used by multiple processes to share the TLB; for the same virtual page number as other processes, each Processes have different mappings.

R …… area. (00-> User, 01-> Management, 11-> Core) The user matches the virtual address 63:62 bits.

Fill ..... Reserved. Read as '0' and write ignored.

0 ..... Reserved. Writes must be 0, and returns 0 when read.

Figure 5-8 EntryHi register

63 42 41 6 5 3 2 1 0

0 PFN CDVG

twenty two 36 3 1 1 1

63 42 41 6 5 3 2 1 0

0 PFN CDVG

twenty two 36 3 1 1 1

PFN … page frame number; high order *bit of* physical address.

C ……. Specify the consistency properties of the TLB page; see Table 3-2.

D ..... dirty position. If the bit value is set to 1, the corresponding page mark is dirty and therefore writable. This bit is actually write-protected and available in software This bit protects data from changes.

V ……. Effective bit. Setting this bit indicates that the TLB entry is valid; otherwise, TLBL / TLBS will be invalidated.

G …… .Global bit. If the corresponding bits in Lo0 and Lo1 are both set to 1, the processor ignores the ASID during a TLB lookup.

0 …… .. Reserved. Writes must be 0, and returns 0 when read.

Figure 5-9 EntryLo0 and EntryLo1 registers

The TLB page consistency attribute bit (C) specifies whether the cache needs to be accessed when the page is accessed. You need to select the consistency property of the cache. Table 5-2Represents the cache consistency property corresponding to the C bit.

Table 5-2 Value of the C bit on the TLB page

C (5: 3) value	**Cache** consistency properties
0	Keep

1	Keep
2	Uncached
3	Noncoherent cache (Cacheable Noncoherent)
4	Keep
5	Keep
6	Keep
7	Uncached Accelerated

5.5.2. CP0 Register

Table 5- 3 lists the CP0 registers related to memory management. Chapter 1 provides a complete description of the CP0 registers.

Table 5-3 CP0 registers related to memory management

Register number	Register name
0	Index
1	Random

Register number	Register name
2	EntryLo0
3	EntryLo1
5	PageMask
6	Wired
10	EntryHi
15	PRID
16	Config
17	LLAddr
28	TagLo
29	TagHi

5.5.3. Virtual address to physical address conversion process

When the virtual address is converted to the physical address, the CPU changes the 8-bit ASID of the virtual address (if the global bit G is not set) and Compare the ASIDs of the TLB entries to see if they match. When comparing ASIDs, you also need to use the page mask (PageMask) The value of 15 matches the upper 15 ~ 27 bits of the virtual address with the virtual page number of the TLB entry. If there is a TLB entry match, it matches from Remove the physical address and access control bits (C, D, and V) from the TLB entry. For a valid address translation, match The V bit of the TLB entry must be set, but the value of the V bit is not taken into account in the match comparison. Figure 5-10 shows the TLB address Conversion process.

100

Page 116

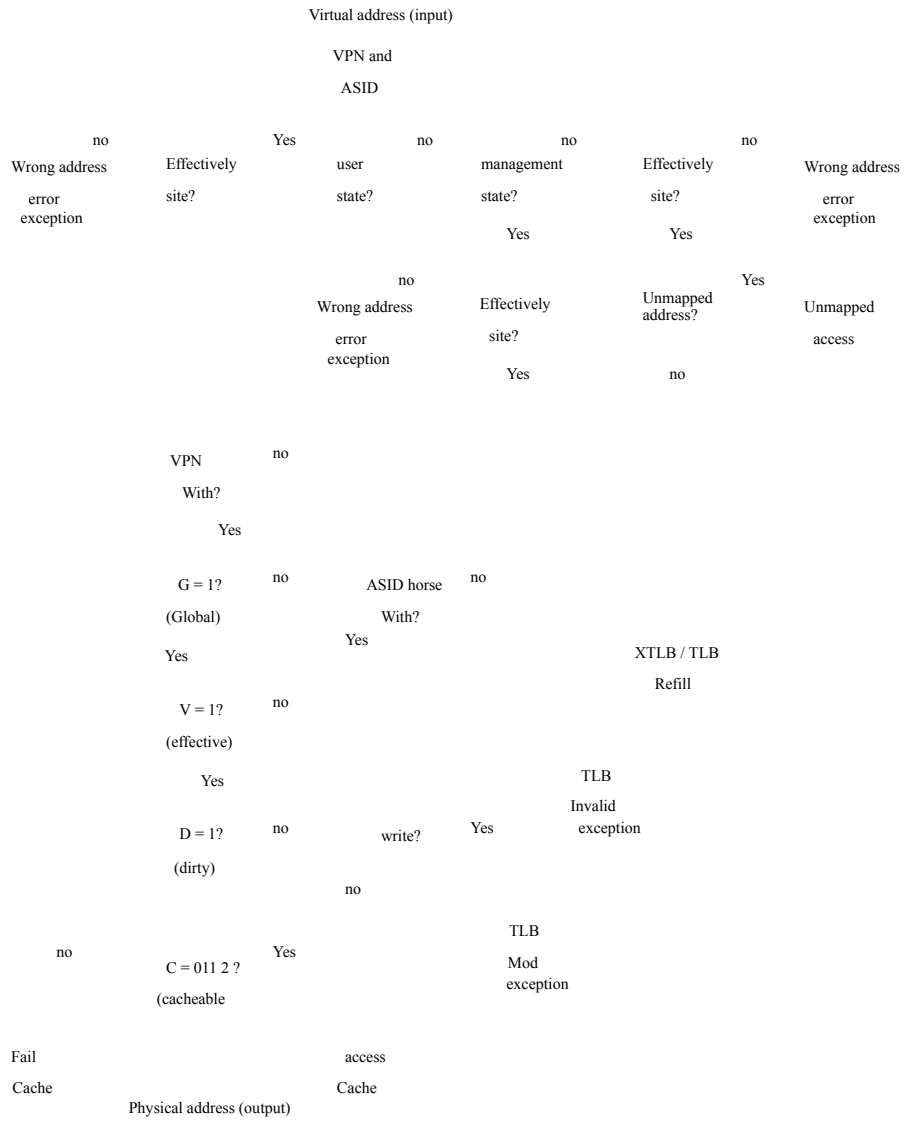


Figure 5-10 TLB address translation

101

5.5.4. TLB Failure

If no TLB entry matches the virtual address, a TLB miss exception is raised. If the access control bit (D And V) indicates that the access is not legal, causing a TLB modification or invalid TLB exception. If the C bit is equal to 011 2 , The retrieved physical address accesses memory through the cache, otherwise it does not pass the cache.

5.5.5. TLB instruction

Table 5- 4 lists all instructions provided by the CPU for operations related to TLB.

Table 5-4 TLB instruction

Opcode	Instruction description
TLBP	Searching for matches in the TLB
TLBR	Read indexed TLB entries
TLBWI	Write indexed TLB entries
TLBWR	Write random TLB entries

5.5.6. Code examples

The first example is how to configure a TLB entry to map a pair of 4KB pages. Most of the kernels of real-time systems are like this Do, this simple kernel MMU is only used for memory protection, so static mapping is sufficient, in all static mapping All TLB exceptions in the system are treated as error conditions (not accessible).

```
1.  mtc0 r0, C0_WIRED                  # make all entries available to random replacement
2.  li r2, (vpn2 << 13) | (asid & 0xff);
3.  mtc0 r2, C0_ENHI                   # set the virtual address
4.  li r2, (epfn << 6) | (coherency << 3) | (Dirty << 2) | Valid << 1 | Global)
5.  mtc0 r2, C0_ENLO0 # set the physical address for the even page
6.  li r2, (opfn << 6) | (coherency << 3) | (Dirty << 2) | Valid << 1 | Global)
7.  mtc0 r2, C0_ENLO1 # set the physical address for the odd page
8.  li r2, 0                           # set the page size to 4KB
9.  mtc0 r2, C0_PAGEMASK
```



```
10. li r2, index_of_some_entry # needed for tlbwi only
11. mtc0 r2, C0_INDEX                  # needed for tlbwi only
```

```
tlbwr                          # or tlbwi
```

A complete virtual storage operating system (such as UNIX), with MMU for memory protection, and main memory and large

Form feed for a mass storage device. This mechanism allows programs to access larger storage devices and is not limited to system physical partitions.

With space. This mechanism that relies on requesting paging requires dynamic page mapping. Dynamic mapping through a series of different types

The MMU is implemented with exceptions, and TLB refilling is the most common exception in such systems. Here is a possible TLB refill exception

control.

```
12. refill_exception:
    13.mfc0 k0, C0_CONTEXT
    14.sra k0, k0,1            # index into the page table
    15. lw k1,0 (k0)           # read page table
    16. lw k0,4 (k0)
    17.sll k1, k1,6
    18. srl k1, k1,6
    19.mtc0 k1, C0_TLBLO0
    20. sll k0, k0,6
    21. srl k0, k0,6
    22.mtc0 k0, C0_TLBLO1
    23. tlbwr                  # write a random entry
    eret
```

This exception control process is very simple, because its frequent execution will affect system performance, this is the TLB refill example

Reasons for allocating independent exception vectors outside. This code assumes that the required mapping has been established in the page table of main memory

Already. If not established, a TLB invalidation exception will occur after the ERET instruction. TLB failure exceptions rarely occur

This is beneficial because it must calculate the expected mapping and may need to read part of the page table from the backing memory.

TLB modification exceptions are used to implement read-only pages and mark pages that need to be modified by the process cleanup code. To protect different processes

There is no mutual interference with the user. The virtual storage operating system usually executes user programs in user mode. The following example shows

How to enter user mode from kernel mode.

```
24. mtc0 r10, C0_EPC                    # assume r10 holds desired usermode
address
    25. mfc0 r1, C0_SR                  # get current value of Status register
    26. and r1, r1, ~ (SR_KSU || SR_ERL)    # clear KSU and ERL field
    27. or r1, r1, (KSU_USERMODE || SR_EXL) # set usermode and EXL bit
    28. mtc0 r1, C0_SR
    eret                                # jump to user mode
```

5.6. Physical Address Space Distribution

The address space of Godson 3 is evenly distributed to each node according to the high-order bits of the address. Upper 4 bits of a 48-bit address [47:44] pair According to the node number where the address space is located, each node has a fixed 44-bit address space. And the 44-bit address within the node The space is further divided into eight 41-bit address spaces. The use of 41-bit space is mainly due to the possibility that a port may access Two HT controllers, each of which requires a 40-bit address space.

6. Processor exception

This chapter introduces Godson GS464 processor core exceptions, including: exception generation and return, exception vector positions and Supported exception types. For each type of exceptions supported, the introduction includes the reasons, handling, and services of the exception.

6.1. Exception creation and return

When the processor starts to handle an exception, the EXL bit of the status register is set to 1, which means that the system is running In kernel mode. After the proper field status is saved, the exception handler usually sets the KSU field of the status register Set to kernel mode and return the EXL bit to 0. When the site state is restored and re-executed, the handler is The KSU field will be restored to the previous value and the EXL bit will be set to 1.

Returning from an exception will also set the EXL bit to zero.

6.2. Exception Vector Position

Cold reset, soft reset, and non-maskable interrupt (NMI) exception vector addresses are dedicated reset exception vector addresses 0xFFFFFFFFBFC00000, this address is neither accessed through the cache nor need address mapping. In addition, EJTAG The entry of the debug interrupt is selected according to whether the ProbeTrap bit in its control register is 0 or 1 0xFFFFFFFFBFC00480 and 0xFFFFFFFFF200200. All other exception vector addresses are in the form of base addresses Add the vector offset. The user can define the base address of the exception vector when the BEV bit in the status register is 0.Table 6-1 Exception vector base address.

Table 6-1 Exception Vector Base Address

exception	BEV = 0	BEV = 1
Reset, Soft Reset, NMI	0xFFFFFFFF BFC00000	
EJTAG Debug (ProbEn = 0)	0xFFFFFFFF BFC00480	
EJTAG Debug (ProbEn = 1)	0xFFFFFFFF FF200200	
Cache Error	0xFFFFFFFF || EBase 31..30 || 1 || EBase 28..12 || 0x000	0xFFFFFFFF BFC00300
Others	0xFFFFFFFF || EBase 31..12 || 0x000	0xFFFFFFFF BFC00200

105

Table 6-2 Lists the exception vector offsets in the Godson GS464 processor core.

Table 6-2 Exception vector offset

exception	Exception vector offset
TLB Refill, EXL = 0	0x000
XTLB Refill, EXL = 0	0x080
Cache error	0x100
Other sharing exceptions	0x180
Interrupted and Cause IV = 1	0x200
Reset, Soft Reset, NMI	None (use base address)

For external interrupts (including clock and performance counter interrupts), the traditional approach is to use a common exception entry Responsibilities are distributed to the corresponding services. Godson GS464 processor core supports Vectored Interrupt mode, this mode It is selected by the IV bit of the Cause register. In vectored interrupt mode, the interrupt priority decreases from IP7 to IP0 and There are special exception entrances. The VS field of the IntCtl register controls the amount of space occupied by these exception handling codes. The entry offset corresponding to the interrupt can be calculated using the following formula (where the vector number starts from zero):

Vector interrupt offset = 0x200 + vector number * IntCtl VS

6.3. Exception priority

The rest of this chapter will followTable 6-3 The order of precedence given in the individual exceptions (for some specific exceptions, (Such as TLB exceptions and instruction / data exceptions are introduced together for convenience). When an instruction generates more than one example at the same time When outside, only the highest priority exception is reported to the processor. Some exceptions are not caused by instructions that were being executed at the time , And some exceptions may be postponed. For more details, see this chapter's separate introduction to each exception.

Table 6-3 Exception priority

Exception priority
Cold reset (highest priority)
Non-maskable interrupt (NMI)

106

Page 122

Address error – fetch

TLB refill – fetch instructions

TLB is invalid – fetch

Cache wrong – fetch

Bus error-fetch

Integer overflow, trap, system call, breakpoint, reserved instruction, coprocessor unavailable, floating point exception

EJTAG interrupt

Address error – data access

TLB refill – data access

TLB is invalid – data access

TLB modification – write data

EJTAG data breakpoint

Cache error – data access

Bus error-data access

Interrupts (lowest priority)

Generally, the exceptions described in the following sections are handled by hardware and then serviced by software.

6.4. Cold reset exception

the reason

When the system is powered on or cold reset for the first time, a cold reset exception is generated. This exception is not maskable.

deal with

The CPU provides a special interrupt vector for this exception:

- Located at 0xBFC0 0000 in 32-bit mode
- Located at 0xFFFF FFFF BFC0 0000 in 64-bit mode

The cold reset vector address belongs to the CPU address space that does not require address mapping and does not access data through the cache.

To handle this exception, it is not necessary to initialize the 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.

107

Page 123

When an exception occurs, the contents of all registers in the CPU are undefined, except for the following register fields:

- The initial value of the Status register is 0x30c000e4, the SR bit is cleared to 0, the ERL bit and

The BEV bit is set to one.

- The initial value of the Config register is 0x80034482.
- The Random register is initialized to its maximum value.
- The Wired register is initialized to 0.
- 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.

service

Cold reset exception services include:

- Initialize all processor registers, coprocessors, caches, and storage systems.
- Perform diagnostic tests.
- Boot the boot operating system.

6.5. NMI exceptions

the reason

NMIn generates a low NMI exception. This exception is not maskable.

deal with

When an NMI exception occurs, the SR bit of the status register is set to 1 to distinguish the cold reset.

NMI exceptions can only be extracted at the boundaries of the instruction. It does not abandon the state of any machine, but retains the state of the processor Status is used for diagnosis. The contents of the Cause register remain unchanged, and the system jumps to the beginning of the NMI exception handler.

The NMI exception preserves all register values except the following:

- ErrorEPC register containing PC value.
- The status register ERL bit is set to 1.
- Status register SR bit with soft reset or NMI set to 1, and cold reset set to 0.
- Status register BEV bit set to 1.
- PC register is reset to 0xFFFF FFFF BFC0 0000

service

NMI exceptions can be used in situations other than "resetting the processor while maintaining the cache and memory contents". E.g, When a power failure is detected, the system can immediately and controllably shut down the system with an NMI exception.

Since the NMI exception occurs in another error exception, it is usually not possible to continue execution after returning from the exception program.

6.6. Address error exception

the reason

Address error exceptions occur when:

- cited illegal address space.

- Reference superuser address space in user mode.
- reference kernel address space in user or supervisor mode.
- taken (Load) or store (Store) a double word, double word, but not aligned with a double word boundary.
- Load (Load, Fetch) or Store (Word), but the word is not aligned with the word boundary
- Take or save a halfword, but the halfword is not aligned with the boundary of the halfword.

This exception is not maskable.

deal with

The shared exception vector is used for address error exceptions. The value of the ExcCode field of the Cause register is set to ADEL or ADES The coded value, along with the BD bit in the EPC and Cause registers, indicates the instruction that caused the exception and the cause of the exception Because it is instruction reference, fetch operation instruction or storage operation instruction.

When an exception occurs, the BadVAddr register holds a virtual address that is not properly aligned, or a protected address space Virtual address.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

At this point, the running process that caused the exception will receive the UNIX SIGSEGV (segment violation) signal. This error It is usually fatal to the process.

Page 125

6.7. TLB exception

Three TLB exceptions can occur:

- When no entry in the TLB matches the address of the mapped address space to be referenced, it will cause the TLB Refill exception.
- When the virtual address reference matches an item in the TLB, but the item is marked as invalid, the TLB has no Effect exception.
- When the virtual address reference of the write memory operation matches an item in the TLB, but the item is not marked as When "dirty" (meaning that the item is not writable), a TLB modification exception occurs.

The following three sections describe these TLB exceptions.

Note: TLB refill vector selection has been introduced earlier in this chapter. For details, see 6.8 " TLB Refill Fill in the exception. "

6.8. TLB refill exception

the reason

When no entry in the TLB matches the reference address in the mapped address space, a TLB refill exception occurs, and the exception is not screenable Shielded.

deal with

For this exception, there are two special exception vectors for the MIPS architecture: one for the 32-bit address space,

The other is for a 64-bit address space. The exception vector offset is 0x000 when the reference address is in a 32-bit address space.

When using addresses in a 64-bit address space, the exception vector offset is 0x080.

When the EXL bit in the status register is set to 0, all address references use these exception vectors. This example peripheral

Set the ExcCode field in the Cause register to TLBL or TLBS encoding. This encoding is related to the EPC register and Cause

The BD of the register together indicates the instruction that caused the exception and the cause of the exception is the instruction reference, fetch instruction, or store

Make instructions.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold that address.

Replace the failed virtual address. The EntryHi register also holds the ASID when the conversion fails. Random register usually holds

A legal place to place the replaced TLB entry. The contents of the EntryLo register are undefined. If an exception raises

If the instruction is not in a branch delay slot, the EPC register holds the address of the instruction that caused the exception; otherwise, the EPC

110

The register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

To service this exception, the contents of the Contex or XContext registers are used as virtual addresses to get some memory bits

These locations contain the physical page address and access control bits for a pair of TLB entries. This pair of TLB items is put in

EntryLo0 / EntryLo1 registers; EntryHi and EntryLo registers are written to the TLB.

The virtual address used to obtain the physical address and access control information may be located on a page that does not reside in the TLB

on. If this happens, the TLB refill handler allows another TLB refill exception to resolve. due to

The EXL bit of the Status register is set to 1. The second TLB refill exception is passed in the common exception vector.

6.9. TLB invalid exception

the reason

TLB is invalid when a virtual address reference matches an entry marked as invalid (the TLB valid bit is cleared)

An exception occurred. This exception is not maskable.

deal with

The common exception vector is used to handle this exception. The value of the ExcCode field of the Cause register is set to TLBL or TLBS,

Together with the BD bit of the EPC register and the Cause register, indicate the instruction that caused the exception and the cause of the exception is the instruction

Reference, fetch operation instruction or store operation instruction.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold that address.

Replace the failed virtual address. The EntryHi register also holds the ASID when the conversion fails. Random register usually holds

A legal place to place the replaced TLB entry. The contents of the EntryLo register are undefined.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction

Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

TLB entries are marked invalid when one of the following occurs:

- The virtual address does not exist
- Virtual address exists, but not in main memory (missing page)
- refer to this page triggered a trap (for example, to maintain a reference bit)

111

After serving the cause of the TLB invalid exception, locate the TLB entry through the TLBP instruction (detect the TLB to find a matching Entry), and then replace the TLB entry with the entry whose flag is valid.

6.10. TLB modification exception

the reason

When the virtual address reference of a write memory operation matches an item in the TLB, but the item is not marked as "dirty", so the When an item is not writable, a TLB modification exception occurs. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the value of the ExcCode field in the Cause register is set to MOD.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold that address. Replace the failed virtual address. The EntryHi register also holds the ASID when the conversion fails. The contents of the EntryLo register are not definite.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

The kernel uses the failed virtual address or page number to identify the corresponding access control information. The identified page may allow or Write access is not allowed; if write access is not allowed, a write protection violation occurs.

If write access is allowed, the kernel marks the page as writable within its own data structure. TLBP instruction The index of the TLB entry that must be changed is placed in the Index register. Contains physical page and access control bits (D bit is set) A word is fetched into the EntryLo register, and then the EntryHi and EntryLo registers are written into the TLB.

6.11. Cache Error Exception

the reason

When the processor fetches or fetches an internal cache check error, a cache error exception occurs. The exception cannot shield.

deal with

The cache error exception entry with an offset of 0x100 is used to handle cache error exceptions. The exception entry base address is now set

112

In the address segment without going through the cache. The value of the ExcCode field of the Cause register is set to CacheErr, along with the EPC register

Together with the BD bit in the Cause register, it indicates the instruction that caused the exception and whether the cause of the exception is an instruction reference or a fetch.

Operation instructions. The CacheErr register records the type of error and its position in the group's associative cache. CacheErr1 register

Record the virtual address or physical address of the error instruction. For details, see Chapter X, Section X, CacheErr and CacheErr1.

description.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction

Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

The Godson GS464 processor checks the cache error and implements the hardware self-correction function. The application program can simply follow the example directly.

Outside return.

If there is an error in the instruction cache, the error cache line will be invalidated; if there is an error in the data cache and only one bit is wrong,

The faulty data will be corrected automatically; if the data cache has errors and there are two errors, the operating system should treat the bits of the error data block

Setting determines the processing method.

6.12. Bus Error Exception

the reason

When the processor reads or updates the data block or reads a double word / single word / half word, it receives an external ERR.

Answer signal, bus error exception occurred. This exception is not maskable.

deal with

The shared interrupt vector is used to handle bus error exceptions. The value of the ExcCode field of the Cause register is set to IBE or DBE,

Together with the BD bit of the EPC register and the Cause register, indicate the instruction that caused the exception and the cause of the exception is the instruction

Reference, fetch operation instruction or store operation instruction.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction

Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

The physical address where the error occurred can be calculated from the information in the CP0 register.

If the value of the ExcCode field in the Cause register is set to IBE encoding (indicating that it is used for guidance), then the result

The virtual address of the instruction where the exception occurred is stored in the EPC register (if the BD bit in the Cause register is set to 1, the instruction

Let the virtual address be the content of the EPC register plus 4).

If the ExcCode field value in the Cause register is set to DBE encoding (indicating that it is a read or store reference), then

The virtual address of the instruction that caused the exception is stored in the EPC register (if the BD bit in the Cause register is set to 1,

The virtual address of this instruction is the content of the EPC register plus 4).

Then, reading and storing the referenced virtual address can be obtained by interpreting this instruction. And the physical address can pass

TLBP instruction and read the contents of EntryLo register to calculate the physical page number. The running exception that caused the exception

A process receives a UNIX SIGBUS (bus error) signal, which is usually fatal to the process.

6.13. Integer overflow exception

the reason

When an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction is executed, causing the complement of the result to overflow, the integer A type overflow exception occurred. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the OV code value.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

The executing process that caused the exception to receive a UNIX SIGFPE / FPE_INTOVE_TRAP Integer overflow) signal. This error is usually fatal to the process.

6.14. Trap exceptions

the reason

When TGE, TGUE, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, TNEI instructions When the condition result is true, a trap exception occurs. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the TR code value.

If the instruction that caused the exception is not an instruction in a branch delay slot, the EPC register holds the location of the instruction Address; otherwise, the EPC register holds the address of the previous branch instruction, and the BD bit of the Cause register is set to 1.

service

The executing process that caused the exception to receive a UNIX SIGFPE / FPE_INTOVE_TRAP (floating point exception / Integer overflow) signal. This error is usually fatal to the process.

6.15. System Call Exceptions

the reason

When the SYSCALL instruction is executed, a system call exception occurs. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the SYS code value.

If the SYSCALL instruction is not in a branch delay slot, the EPC register holds the address of the instruction; otherwise, the Stores the address of the previous branch instruction.

If the SYSCALL instruction is in a split delay slot, the BD bit in the status register is set to 1, otherwise it is cleared to 0.

service

When this exception occurs, control is transferred to the appropriate system routine. Further system call differentiation can be analyzed

Code field of the SYSCALL instruction (bits 25: 6), and the contents of the instruction that loads the address stored in the EPC register.

In order to resume the execution of the process, the contents of the EPC register must be changed so that the SYSCALL instruction will not be executed again OK; this can be done by incrementing the value of the EPC register by 4 before returning.

If the SYSCALL instruction is in a branch delay slot, a more complex algorithm is required, which is beyond the scope of this section.

Stated range.

6.16. Breakpoint Exceptions

the reason

When executing a BREAK instruction, a breakpoint exception occurred. This exception is not maskable.

deal with

115

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the BP code value.

If the BREAK instruction is not in a branch delay slot, the EPC register holds the address of the instruction; otherwise, it saves

The address of the previous branch instruction.

If the BREAK instruction is in a split delay slot, the BD bit in the status register is set to 1, otherwise it is cleared to 0.

service

When this exception occurs, control is transferred to the appropriate system routine. Further differentiation can be analyzed by the BREAK instruction Code field (bits 25: 6), and the contents of the instruction that loads the address stored in the EPC register. If this instruction is in

In the branch delay slot, the contents of the EPC register must be increased by 4 to locate the instruction.

In order to resume the execution of the process, the contents of the EPC register must be changed so that the BREAK instruction will not be executed again; This can be done by incrementing the value of the EPC register by 4 before returning.

If the BREAK instruction is in a branch delay slot, in order to resume the execution of the process, the branch instruction needs to be explained

make.

6.17. Reserved instruction exception

the reason

When trying to execute an instruction not defined in MIPS64 Release2 and not customized by Godson, the instruction example is reserved

Outside happen. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the RI code value.

If the reserved instruction instruction is not in a branch delay slot, the EPC register holds the address of the instruction; otherwise, the

Store the previous branch instruction address.

service

At this point, no instructions are interpreted. The process that was executing that caused the exception to receive UNIX SIGILL / ILL_RESOP_FAULT (illegal instruction / reserved operation error) signal For this process, this error is usually

Is deadly.

116

Page 132

6.18. Coprocessor Unavailability Exception

the reason

Attempting to execute any of the following coprocessor instructions will cause the coprocessor unavailable exception to occur:

- The corresponding coprocessor unit (CP1 or CP2) is not marked as available.
- The CP0 unit is not marked as available, and the process executes CP0 in user or superuser mode

instruction.

This exception is not maskable.

The coprocessor unavailable exception of Godson custom extension instruction triggers the following conditions:

- Custom extended 64-bit multimedia acceleration instructions (Table 2 18), custom extended floating-point fetch instructions (Table 7-5)

A coprocessor unavailable exception is triggered when CP1 is not marked as available.

- The custom extended vector instruction triggers the coprocessor unavailable exception when CP1 is not marked as available.

It should be noted that CVT.D.LD, CVT.LD.D, CVT.UD.D are three custom extended floating-point format conversion instructions.

Makes the coprocessor unavailable exception not triggered even when CP1 is not marked as available.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the CPU code value.

The CE field of the Cause register indicates which of the four coprocessors is referenced. If this instruction is not in a branch delay slot,

The EPC register holds the address of the unusable coprocessor instruction; otherwise, the EPC register holds the previous branch instruction

the address of.

service

There are several situations:

If the process is authorized to access the coprocessor and the coprocessor is marked as available, then the corresponding user state is restored to

The coprocessor executes.

If the process is authorized to access the coprocessor, but the coprocessor does not exist or is faulty, then this needs to be explained / emulated

Coprocessor instructions.

If the BD bit in the Cause register is set, the branch instruction must be interpreted; then the coprocessor instruction is

117

Page 133

simulation. Coprocessor instructions that skip the exception when the exception returns continue execution.

If the process is not authorized to access the coprocessor, then the executing process receives UNIX

SIGILL / ILL_PRIVIN_FAULT (illegal instruction / privileged instruction error) signal. This error is usually fatal.

6.19. Floating-point exceptions

the reason

The floating-point coprocessor uses floating-point exceptions. This exception is not maskable.

deal with

The common exception vector is used to handle this exception, and the ExcCode field of the Cause register is set to the FPE code value.

The contents of the floating-point control / status register indicate the cause of this exception.

service

Clearing the appropriate bit in the floating-point / status register can clear this exception.

6.20. EJTAG exception

EJTAG exceptions are triggered when certain EJTAG-related conditions are met.

6.21. Interrupt exception

the reason

When one of the eight interrupt conditions is triggered, an interrupt exception occurs. The importance of these interruptions depends on the specific system

Now.

By clearing the corresponding bit in the Interrupt-Mask (IM) field in the status register, any of the eight interrupts

One can be masked, and all eight interrupts can be masked at once by clearing the IE bit of the status register.

deal with

The ExcCode field of the Cause register is set to an INT-encoded value. According to the current configuration, the processor uses traditional common

Use exception vector processing or use vector exception mode to select the entry corresponding to the highest priority interrupt number for processing.

The IP field in the Cause register indicates the current interrupt request. More than one interrupt bit may be set at the same time (e.g.

(If the interrupt is triggered and is canceled before the register is read, no bit is even set).

The IP [7] interrupt has three sources. Except for interrupt line 5, the contents of the Count register and the Compare register are equal.

An interrupt is generated when the CP0 performance counter overflows. Clock interrupts and performance counter overflow interrupts are caused in the Cause register

The TI and PCI bits indicate.

If vector interrupt mode is not used, the software needs to query every possible interrupt source to determine the cause

(An interrupt may have multiple sources at the same time).

service

If the interrupt was caused by one of two software exceptions, set the corresponding bit in the Cause register, IP [1:

0], 0 to clear the interrupt condition.

Software interrupts are imprecise. Once a software interrupt is triggered, the program may continue to execute for several times before the exception is processed.

Instructions. The timer interrupt is cleared by writing a value to the Compare register. Clearing of performance counter interrupts

It is implemented by writing 0 to the overflow bit of the counter, namely bit 31.

Cold reset and soft reset will clear all outstanding external interrupt requests, IP [2] to IP [6].

If the interrupt is generated by hardware, then the condition that caused the interrupt pin to be triggered can be cancelled to clear the interrupt condition.

119

Page 136

Godson 3B1500 Processor User Manual • Volume 2

7. Floating-point coprocessor

This chapter describes the characteristics of the Godson 3A processor Floating Point Coprocessor (FPU).

Including programming model, instruction set and instruction format, instruction pipeline, and exceptions. Godson 3A floating-point coprocessor and its related

The system software fully complies with the ANSI / IEEE 754-1985 binary floating-point arithmetic standard.

7.1. Overview

The FPU, as the coprocessor of the CPU, is called CP1 (Coprocessor 1). It expands the instruction set of the CPU to

Complete floating-point arithmetic operations.

The FPU consists of the following two functional units:

- FALU1 unit
- FALU2 unit

The FALU1 module can perform all floating-point operations except floating-point fetches and floating-point and fixed-point data transfers, including

Floating-point addition (subtraction), floating-point multiplication, floating-point multiplication (subtraction), floating-point division, floating-point square root, floating-point invers

Find the inverse after rooting, floating-point and fixed-point conversion, floating-point precision conversion, floating-point comparison, branch judgment, and other simple logic. In

The FALU1 module performs SIMD media operations by expanding and multiplexing the FMT field in the instruction code.

FALU2 executes floating-point multiply-add operations (which can calculate floating-point multiply, add, and floating-point multiply-add instructions), and media

Make. At the same time, Godson 3A's FPU supports the execution of parallel single precision (Paired-Single, referred to as PS) Floating-point instructions. Figure 7-1 illustrates the organization of the functional units in the Godson 3A architecture.

Figure 7-1 Organizational structure of functional units in Godson 3A architecture

The floating-point queue can issue one instruction to the FALU1 unit and one instruction to the FALU2 unit each clock cycle. The floating-point register file provides three dedicated read ports and one dedicated write for each of the FALU1 and FALU2 units port.

7.2. FPU Register

This section describes the FPU register groups and their data organization structure. Godson 3A's FPU Register and MIPS64 FPU register compatible. MIPS64's FPU registers include floating-point registers and floating-point control registers. Where floating point Control registers include FIR (No. 1), FCSR (No. 31), FCCR (No. 25), FEXR (No. 26), FENR (28

No.) etc.

122

7.2.1. Floating-point registers

Godson 3A's floating-point registers follow the R10000 usage, which is slightly different from MIPS64. The Status Control Register When the FR bit is 1, there are 32 64-bit floating-point registers, as shown in the figure below; FR in the Status control register When the bit is 0, R10000 only has 16 32-bit or 64-bit floating-point registers, while MIPS64 means there are 32 32-bit Floating-point registers or 16 64-bit floating-point registers.

63	0	63	0
f1		f0	
f3		f2	
f5		f4	
f7		f6	
f9		f8	
f11		f10	
f13		f12	
f15		f14	
f17		f16	
f19		f18	
f21		f20	
f23		f22	
f25		f24	
f27		f26	
f29		f28	
f31		f30	

Figure 7-2 Floating-point register format

7.2.2. FIR Register (CP1, 0)

FIR is a 32-bit read-only register that contains functions implemented by the floating-point unit, such as processor ID, revision number, etc. information. The initial value of FIR in Godson 3A is 0x00770501.

123

Figure 7-3 shows the format of the FIR regist :s the fields of this register .

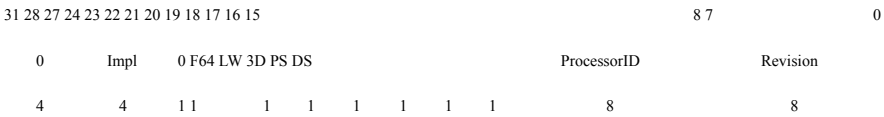


Figure 7-3 FIR register

Table 7-1 FIR Register Field

area	description
0	Reserved. Must be written as 0 and read as 0.
Impl	Implementation-dependent
F64	Whether the floating-point data path is 64-bit 0-32 bits 1-64 bits
L	Whether long-word (64-bit) fixed-point data types are implemented 0-not implemented 1-realized
W	Whether the word (32-bit) fixed-point data type is implemented 0-not implemented 1-realized
3D	Whether MIPS-3D ASE is implemented 0-not implemented 1-realized
PS	Whether floating point data types are implemented 0-not implemented 1-realized
D	Whether double-precision floating-point data types are implemented 0-not implemented 1-realized

area	description
S	Whether single-precision floating-point data types are implemented 0-not implemented 1-realized
ProcessorID	Floating-point processor identification
Revision	The revision number of the floating-point unit

7.2.3. FCSR Register (CP1, 31)

The FCSR register is used to control the operation of the floatir dicate some status. The initial value of FCSR in GS464

0x00000F80 . Figure 7-4 shows the format of the FCSR register. Tabl elds of the FCSR register . its

The E, V, Z, O, U, and I respectively represent unimplemented operations, invalid operations, division by zero, overflow, underflow, and inaccuracy.

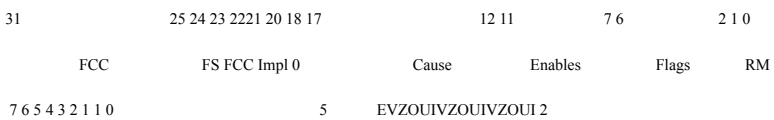


Figure 7-4 FCSR register

Table 7-2 FCSR register fields

area	description
0	Reserved. Must be written as 0 and read as 0.
FCC	Floating-point condition code. Records floating-point comparison results for conditional jumps or branches.
FS	Washed to 0. When this bit is set, the result of the abnormal operation is set to 0 instead of generating an exception.
Impl	Implementation related, GS464 uses FSCR [21] as top_mode, this bit is used to indicate whether to use when decoding X86's TOP register renames floating-point register numbers.
Cause	When an exception occurs for a floating-point operation, the corresponding bit is set.
Enables	Whether to allow exceptions to the corresponding conditions.
Flags	Are there any IEEE floating-point exceptions. (For example, the corresponding bit is not opened in Enables to view this field)
RM	Whether double-precision floating-point data types are implemented 0-not implemented 1-realized

125

Control / Status Register Condition (**CC0**) bit

When a floating-point comparison operation occurs, the result is stored in the CC0 bit, the condition bit. If the comparison is true,

The CC0 bit is set to 1; otherwise it is set to 0. The CC0 bit can only be modified by the floating-point compare instruction and the CTC1 instruction.

Control / status register causes **(Causes)** field

Bits 17:12 of the control / status register are the Causes field. These bits reflect the result of the most recently executed instruction.

The Causes field is a logical extension of the Cause register in coprocessor 0. These bits indicate the

An exception is generated, and an interrupt or exception is generated if the corresponding Enable bit is set. If one

There are more than one exception in each instruction, and each corresponding exception causes the bit to be set.

The Causes field can be overridden by each floating-point operation instruction (excluding Load, Store, and Move operations). Where if

If software emulation is needed to complete, set the unimplemented operation bit (E) of the operation to 1, otherwise it will remain at 0. Other bits follow

Set to 1 or 0 according to the IEEE754 standard to see if the corresponding exception occurs.

When a floating-point exception occurs, no results are stored, and the only state affected is the Causes field.

Control / Status Register Enable **(the Enables)** domain

Whenever the Cause bit and the corresponding Enable bit are both 1 at the same time, a floating-point exception is generated. Such as

If the floating-point operation sets a Cause bit that is allowed to activate (the corresponding enable bit is 1), the processor will immediately generate a

The exception is the same as setting the Cause bit and the Enable bit to 1 with the CTC1 instruction.

There is no corresponding enable bit for unimplemented operation (E). If unimplemented operation is set, it will always generate

An exception is floating point.

Before returning from a floating-point exception, the software must first clear the activated Cause with a CTC1 instruction

Bit to prevent repeated execution of interrupts. Therefore, a program running in user mode will never observe an enabled Cause.

The value of the bit is 1; if the user-mode handler needs to obtain this information, the contents of the Cause bit must be passed to other

Somewhere instead of in the status register.

If the floating-point operation only sets the Cause bit that is not enabled (the corresponding enable bit is 0), no exception occurs.

The default results defined by the IEEE754 standard are written back. In this case, the exception caused by the previous floating-point instruction can

This can be determined by reading the value of the Causes field.

Control / Status Register Flags **(Flags)** Field

The flag is cumulative and indicates that an exception has occurred since the last time it was explicitly reset. If an IEEE754 exception

Is generated, then the corresponding Flag bits are set to 1, otherwise they remain unchanged, so these bits are never set for floating-point operations

Clear. However, we can write a new value to the status register through the CTC1 control instruction to set the Flag bit.

Set or clear.

When a floating-point exception occurs, the Flag bit is not set by the hardware; it is the responsibility of the software handling the floating-point exception to call

These bits are set before the user program.

Control / Status Register Rounding Mode **(RM)** Field

Bits 0 and 1 in the control / status register form the rounding mode (RM) field. As shown in Table 7-3 As shown, FPU

All floating-point operations are rounded according to the rounding method specified by these bits.

Table 7-3 Rounding mode bit decoding

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

7.2.4. FCCR Register (CP1, 25)

The FCCR register is another way to access the FCC fields. Its content is exactly the same as the FCC bit in the FCSR.

The difference is that the FCC bits in this register are continuous. Figure 7-5 shows the format of the FCCR register.

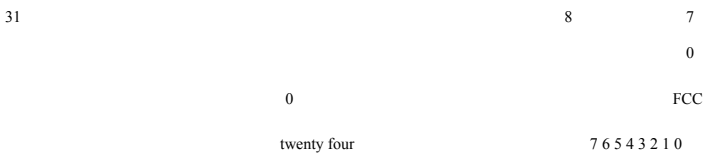


Figure 7-5 FCCR register

7.2.5. FEXR Register (CP1, 26)

The FEXR register is another way to access the Cause and Flags fields. Its contents are the same as the corresponding fields in the FCSR.

127

Exactly the same. Figure 7-6 shows the format of the FEXR register.

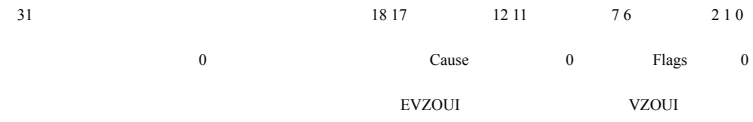


Figure 7-6 FEXR register

7.2.6. FENR Register (CP1, 28)

The FENR register is another way to access the Enable, FS, and RM fields. Its contents correspond to the corresponding words in the FCSR.

The segments are exactly the same. Figure 7-7 shows the format of the FENR register.

Figure 7-7 FENR register

7.3. Floating-point instructions

7.3.1. List of MIPS64 compatible floating-point instructions

GS464 implements all data types of FPU part in MIPS64, including S, D, W, L, and optional PS.

Table 7-4 lists the FPU instructions in the MIPS64 section of GS464.

Table 7-4 FIPS instruction set of MIPS64

OpCode	Description	MIPS ISA
	Arithmetic instructions	
ABS.fmt	Absolute value	MIPS32
ADD.fmt	addition	MIPS32

128

DIV.fmt	division	MIPS32
MADD.fmt	Multiply	MIPS64
MSUB.fmt	Multiplication	MIPS64
MUL.fmt	multiplication	MIPS32
NEG.fmt	Negation	MIPS32
NMADD.fmt	Negation after multiplication and addition	MIPS64
NMSUB.fmt	Negation	MIPS64
RECIP.fmt	Find the reciprocal	MIPS64
RSQRT.fmt	Find the inverse of the square root	MIPS64
SQRT.fmt	Square root	MIPS32
SUB.fmt	Subtraction	MIPS32
	Branch instruction	
BC1F	Floating point jump	MIPS32
BC1FL	Likely Jump when Floating Point Fake	MIPS32
BC1T	Floating point true time jump	MIPS32
BC1TL	Likely Jump when Floating Point True	MIPS32
	Compare instructions	
C.cond.fmt	Compare floating-point values and set flags	MIPS32
	Conversion instruction	
CEIL.L.fmt	Floating point conversion to 64-bit fixed point, rounded up	MIPS64
CEIL.W.fmt	Floating point conversion to 32-bit fixed point, rounded up	MIPS64
CVT.D.fmt	Floating or fixed point conversion to double precision floating point	MIPS32
CVT.L.fmt	Converting floating-point values to 64-bit fixed-point	MIPS64
CVT.PS.S	Convert two floating point values to floating point pair	MIPS64
CVT.S.PL	Converts the low order of a floating point pair to single precision floating point	MIPS64
CVT.S.PL	Converts the high order of a floating point pair to single precision floating point	MIPS64
CVT.S.fmt	Floating point or fixed point to single precision floating point	MIPS32
CVT.W.fmt	Convert floating point value to 32-bit fixed point	MIPS32
FLOOR.L.fmt	Floating-point to 64-bit fixed-point, rounded down	MIPS64
FLOOR.W.fmt	Floating point conversion to 32-bit fixed point, rounded down	MIPS64
PLL.PS	Merges the lower bits of two floating point pairs into a new floating point pair	MIPS64

PLU.PS	Merge the low and high bits of two floating-point pairs into a new floating-point pair	MIPS64
PUL.PS	Merge the high and low bits of two floating point pairs into a new floating-point pair	MIPS64
PUU.PS	Merge the high-order bits of two floating-point pairs into a new floating-point pair	MIPS64
ROUND.L.fmt	Round floating point numbers to 64-bit fixed point	MIPS64
ROUND.W.fmt	Round floating point numbers to 32-bit fixed point	MIPS32
TRUNC.L.fmt	Rounds a floating point number to a 64-bit fixed point in the direction of small absolute value	MIPS64
TRUNC.W.fmt	Round floating-point numbers to 32-bit fixed-point values with a small absolute value	MIPS32

	Fetch instruction	
LDC1	Access doubleword from inside	MIPS32
LDXC1	Access doublewords from within by index	MIPS64
LUXC1	Access doublewords internally by unaligned index	MIPS64
LWC1	Access word from inside	MIPS32
LWXC1	Access word by index	MIPS64
SDC1	Save doubleword to memory	MIPS32
SDXC1	Store doublewords into memory by index	MIPS64
SUXC1	Store doubleword into memory by unaligned index	MIPS64
SWC1	Save word to memory	MIPS32
SWXC1	Store words into memory by index	MIPS64
	MOVE instruction	
CFC1	Read floating-point control register to GPR	MIPS32
CTC1	Write floating-point control register to GPR	MIPS32
DMFC1	Copy doubleword from FPR to GPR	MIPS64
DMTC1	Copy doubleword from GPR to FPR	MIPS64
MFC1	Copy low words from FPR to GPR	MIPS32
MFHC1	Copy high words from FPR to GPR	MIPS32 R2
ALNV.PS	Variable floating point alignment	MIPS64
MOV.fmt	Copy FPR	MIPS32
MOVF.fmt	Copy FPR on floating-point leave	MIPS32
MOVN.fmt	Copy FPR when GPR is not 0	MIPS32
MOVT.fmt	Copy FPR when floating point is true	MIPS32
MOVZ.fmt	Copy FPR when GPR is 0	MIPS32

Page 146

MTC1	Copy low words from GPR to FPR	MIPS32
MTHC1	Copy high words from GPR to FPR	MIPS32 R2

7.3.2. Explanation of MIPS64 compatible floating-point instruction implementation

GS464 is compatible with MIPS64 Release 2 and functionally implements all the provisions of the MIPS64 architecture FPU instructions, but some instructions have minor implementation differences that do not affect compatibility but are important differences. The following two Points deserve the programmer's attention.

(1) Multiply-add, multiply-subtract instructions. After executing MADD.fmt, MSUB.fmt, NMADD.fmt, NMSUB.fmt this With four sets of instructions, the operation result of GS464 is slightly different from that of MIPS64 processor. This is because GS464 is doing multiplication and addition. Only precision rounding is performed at the final result (the so-called fused-mulitply-add), and the MIPS64 processor is performing multiplication. Rounding is performed twice after the operation and after the addition operation. The difference between the two rounding methods results in the final result in some cases. The lowest bit differs by 1.

(2) Single-precision operation instructions. When the FR bit of the Status control register is 0, abs.s, add.s, ceil.wd, ceil.ws, div.s, floor.wd, floor.ws, mul.s, neg.s, round.wd, round.ws, sqrt.s, sub.s, trunc.wd, trunc.ws,

mov.s, cvt.ds, cvt.dw, cvt.sd, cvt.sw, cvt.wd, cvt.ws, movf.s, movn.s, movt.s, movz.s etc

Instructions cannot use odd-numbered registers, but processors with the MIPS64 architecture can. Godson has continued to use this point.

The practices of MIPS R4000 and MIPSR10000 are slightly different from those of MIPS64. (Early MIPS processors

The FR bit indicates whether the floating point register is 16 or 32. The FR bit in MIPS64 indicates whether the floating point register is 32 or 32.

64-bit).

7.3.3. Loongson Custom Extended Floating Point Instruction

Table 7-5 Custom extended floating-point fetch instructions

Instruction mnemonics	Brief instruction function
GSSQC1	Fixed-source quad-word
GSSWLEC1	Store word from floating point register with out-of-bounds check
GSLWXC1	Floating-point word with offset
GSLQC1	Double destination registers take floating-point quadwords
GSLWLEC1	Take word out of bounds check to floating point register

Instruction mnemonics	Brief instruction function
GSLWLC1	Take left of word to floating point register
GSLWRC1	Fetch word right to floating point register
GSLDLC1	Take the left part of the double word to the floating point register
GSLDRC1	Take the right part of a doubleword to a floating point register
GSLWGTC1	Word fetch with lower out-of-bounds check into floating point register
GSLDLEC1	Take doubleword with out-of-bounds check into floating-point register
GSLDGTC1	Take doubleword with lower out-of-bounds check into floating point register
GSLDXC1	Floating-point doubleword with offset
GSSWLC1	Store word from left of floating point register
GSSWRC1	Store word from floating point register to the right
GSSDLC1	Store left part of double word from floating point register
GSSDRC1	Store double word right from floating point register
GSSWGTC1	Store word from floating-point register with lower out-of-bounds check
GSSDLEC1	Store doubleword from floating-point register with out-of-bounds check
GSSDGTC1	Store doubleword from floating-point register with lower out-of-bounds check
GSSWXC1	Floating-point word with offset
GSSDXC1	Floating-point doubleword with offset

Table 7-6 Custom extended floating-point format conversion instructions

Instruction mnemonics	Brief instruction function
CVT.D.LD	Extended double precision to double precision
CVT.LD.D	Double precision converted to extended double precision low bit
CVT.UD.D	Double precision converted to extended double precision high bit

7.4. Floating-point part format

7.4.1. Floating-point format

The FPU can perform both IEEE 32-bit (single-precision) and 64-bit (double-precision) floating-point numbers.

operating. The 32-bit single-precision format includes a 24-bit sign-amplitude decimal field (F + S) and an 8

Exponent field of bits (E); 64-bit double precision format includes a 53-bit sign-amplitude decimal field (F + S)

And an 11-bit exponential field (E); the 64-bit double precision (PS) format contains two single-precision floating-point formats. Respectively, as shown in 7-8 .

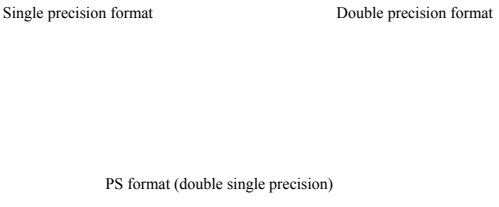


Figure 7-8 Floating point format

As shown in Figure 7-8 , the format of a floating-point number consists of the following three fields:

- Symbol field, S
- Exponential domain with offset, E = E 0 + Bias, E 0 is the exponent without offset
- Decimal field, F = .b 1 b 2 .... b p–1

The range of the index E 0 is an integer between all two including E min and E max , plus the following two guarantees

Residual value:

- E min -1 (used to encode 0 and subnormal numbers)
- E max +1 (used to encode ∞ and NaN [Not a Number])

For single-precision or double-precision formats, each representable non-zero number has a unique encoding corresponding to it.

The value V corresponding to its encoding is determined by the equations in Table 7-7 .

Table 7-7 Formulas for calculating the values of floating-point numbers in single and double precision formats

NO.	formula
(1)	if E0 = Emax + 1 and F ≠ 0, then V = NaN, regardless of s
(2)	if E0 = Emax + 1 and F = 0, then V = (-
(3)	if Emin≤E0≤Emax, then V = (–1) S2 E0 (1.F)
(4)	if E0 = Emin–1 and F ≠ 0, then V = (–1) S2Emin (0.F)

(5) if E0 = Emin–1 and F = 0, then V = (–1) S0

For all floating-point formats, if V is a NaN, then the most significant bit of F determines that this number is Signaling NaN or Quiet NaN: If the most significant bit of F is set, then V is Signaling NaN, otherwise V is

Table 7-8 defines the values of some related parameters in floating-point format; the maxim lues of floating-point are given in Table 7-9.

Table 7-8 Floating-point format parameter values

parameter	format	
	Single precision	Double precision
Emax	+127	+1203
Emin	-126	-1022
Exponential offset	+127	+1023
Exponential bit width	8	11
Integer	Hidden Hidden	
F (decimal place width)	twenty four	53
Format total width	32	64

Table 7-9 Floating point values for maximum and minimum numbers

Types of	value
Single-precision floating-point minimum	1.401129846e--45
Single-precision floating-point minimum regular number	1.17549435e--38
Single-precision floating-point maximum	3.402823347e + 38
Double-precision floating-point minimum	4.940656458412465 4e--324
Double-precision floating-point minimum regular number	2.2250738585072014e--308
Double-precision floating-point maximum	1.797693134862315 7e + 308

7.5. FPU Instruction Pipeline Overview

The FPU provides an instruction pipeline that is parallel to the CPU instruction pipeline. It shares a basic 9-stage pipeline with the CPU Architecture, but according to different floating-point operations, the execution pipeline is subdivided into 2 ~ 6 pipeline stages. Every FPU instruction is One of the two floating-point functional units executes: FALU1 or FALU2. FALU1 can perform all floating-point operations And media operations. FALU2 performs only floating-point addition, subtraction, multiplication, multiply-add operations, and all media operations.

Each FALU unit can receive one instruction per cycle and send one to the floating-point register file. result. In each FALU unit, floating-point addition, subtraction, floating-point multiplication, and floating-point multiplication and addition require 6 execution cycles; fixed-point Format conversion operations with floating point require 4 execution cycles; floating point division requires 4 to 16 executions depending on the operand Cycles; the square root of floating point requires 4 to 31 execution cycles depending on the operand, and 2 floating point operations require 2 executions cycle. In each FALU unit, if two instructions with different execution cycles output results in the same shot, in this kind of

In the case, the instruction with the shorter execution cycle outputs the result to the bus first. Where floating-point except floating-point division and floating-point rooting

Operations and all media operations are fully pipelined. If there are two floating-point division instructions or two floating-point square roots

Order in FALU1, then the FALU1 unit will send a pause signal to the first level, and the FALU1 unit is in

A new instruction cannot be received until the division or square root instruction is written back.

7.6. Floating-point exception handling

This section describes exceptions to floating-point calculations. Floating-point exceptions occur when the FPU cannot handle operands or

When the result of the floating-point calculation is used, the FPU generates a corresponding exception to start the corresponding software trap or set the status flag.

The FPU's control and status register contains an enable bit for each exception. The enable bit determines whether an exception is

No can cause the FPU to initiate an exception trap or set a status flag.

If a trap is started, the FPU keeps the operation started and starts the software exception handling path; if there is no trap

On startup, an appropriate value is written to the FPU destination register and the calculation continues.

The FPU supports five IEEE754 exceptions:

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

And the sixth exception:

- Unimplemented Operation (E)

Unimplemented operation exceptions are used when the FPU cannot perform standard MIPS floating-point structures, including the FPU cannot determine the correct

Of exceptional behavior. This exception indicates the execution of software exception handling. Unimplemented operation exception without enable signal and

Flag bit. When this exception occurs, a corresponding unimplemented exception trap occurs.

The five exceptions of IEEE754 (V, Z, O, U, I) correspond to an exception trap controlled by the user. When 5

When one of the enable bits is set, the corresponding exception trap is allowed to occur. When an exception occurs, cause

The bit is set. If the corresponding Enable bit is not set, the Exception flag is set. If enabled

If the bit is set, then the flag is not set and the FPU generates an exception to the CPU. Subsequent exception handling allows the

An exception trap occurred.

When there is no exception trap signal, the floating-point processor handles it by default, providing a floating-point calculation exception

Substitute value for fruit. Different exception types determine different default values. Table 7-10 lists the FPU for each IEEE example.

Out of default processing.

Table 7-10 Default handling of exceptions

area	description	Rounding mode	Default action
I	Inexact exception	Any	Provide rounded results
		RN	Set the result to 0 according to the sign of the intermediate result
U	Underflow exception	RZ	Set the result to 0 according to the sign of the intermediate result
		RP	Correct the positive underflow to the smallest positive number and the negative underflow to -0

		RM	Correct negative underflow to the smallest negative number and correct positive underflow to +0
		RN	Set the result to infinity based on the sign of the intermediate result
O	Overflow exception	RZ	Set the result to the maximum number based on the sign of the intermediate result
		RP	Correct negative underflow to the largest negative number, and correct positive underflow to + ∞
		RM	Correct positive underflow to the largest integer and negative underflow to -∞
Z	Divide by 0	Any	Provide a corresponding signed infinity number
V	Illegal operation	Any	Provide a Quiet Not a Number (QNaN)

The conditions that cause each exception to be generated by the FPU are described below, and the FPU's Conditioned response.

Inexact exception **(I)**

The FPU generates inexact exceptions when:

- Rounding results are inaccurate
- Rounding result overflow
- The rounding result underflows, and neither the underflow nor inaccurate enable bits are set, and the FS bit is set.

Page 152

Result of trap being enabled: if a non-exact exception trap is enabled, the result register is not modified and Registers are reserved. Because this execution mode affects performance, inexact exception traps are only caught when necessary Enable.

Trap not enabled results: if no other software traps occur, the rounded or overflow result is sent to the target register.

Illegal operation exception **(V)**

When two or one of the operands of an executable operation is illegal, an illegal operation exception is issued. No. notice. If the exception is not caught, MIPS defines the result as a Quiet Not a Number (QNaN). non-Law operations include:

- Addition or subtraction: Subtract infinitely. For example: (+ ∞) + (-∞) or (-∞)-(-∞)
- Multiplication: 0 × ∞, for all positive and negative numbers
- Division: 0/0, ∞ / ∞, for all positive and negative numbers
- When the operand of a comparison operation that does not handle Unordered is Unordered
- Perform floating-point comparison or conversion on an indicator signal NaN
- Any mathematical operation on SNaN (Signaling NaN). When one of the operands is SNaN or two This exception is caused when both are SNaN (MOV operations are not considered mathematical operations, but ABS and NEG are considered as Mathematical operations)
- Square: $\sqrt{X}$, when X is less than 0

Software can simulate exceptions to illegal operations given other source operands. For example, using IEEE 754 to implement software Specific function: X REM Y, here when Y is 0 or X is infinite; or when the floating point number is converted to decimal Overflow occurs when the system is infinity or NaN; or a prior function such as ln (5) or cos -1 (3).

Result of trap being enabled: the value of the source operand is not transmitted.

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

Division by zero exception **(Z)**

When the divisor is 0 in a division operation, the dividend is signaled when the divisor is a finite nonzero data. Profit

Software can be used to generate signed infinite values for other operations to simulate division by zero exceptions, such as: ln (0), sin (π / 2), cos (0),

Or 0 -1 .

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

Trap Disabled: If no trap occurs, the result is a signed infinity.

137

Page 153

Overflow exception **(O)**

When the magnitude of the rounded floating-point result is represented by an unbounded exponent, a value greater than the maximum target pattern has

Limit data, send an exception signal for overflow exception. (This exception sets both inexact exceptions and flags)

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

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

Decide.

Underflow exception **(U)**

Two related events caused an underflow exception:

● A small non-zero result between ± 2 Emin . Because the result is very small, it will cause subsequent occurrences

Overflow exception.

● Use subnormal data (Denormalized Number) to approximate the serious data generated by these two small data

Data is distorted.

IEEE754 allows detecting these events in many different ways, but requires the same method for all operations

Detection. Small data can be detected in one of the following ways:

● After rounding (if a non-zero data is calculated without exponential range, it should be strictly

Between ± 2 Emin)

● Before rounding (if a non-zero data is calculated without an exponent or precision range, it should be

Strictly located between ± 2 Emin)

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

● Distortion of subnormal data (the calculated result is different from the result when the index has no limit)

● Inaccurate data (different calculation results when the results produced are not bounded by exponents and precision ranges)

MIPS structures require precision distortion to be detected as producing inaccurate results.

Trap is enabled: if underflow or imprecise exception is enabled, or if the FS bit is not set, unrealized

The exception is the current operation, the result register is not modified.

Trap is not enabled: if underflow or inexact exceptions are not enabled and the FS bit is set, the final result

The result is determined by the rounding mode and the sign bit of the immediate result.

Unimplemented operation exception **(E)**

FPU control / status register when executing any opcode or operation format instruction reserved for future definition

Unimplemented operations in the processor cause the bits to be set and traps generated. The source operand and destination register remain unchanged, while the instruction is in

138

Simulation in software. Any exception in IEEE754 can be generated from simulation operations, and these exceptions can be reversed.

simulation. In addition, when the hardware fails to perform some rare operations or result conditions, it will also generate unimplemented instruction examples.

outer. These include:

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

Set

Note **:** Subnormal and NaN operations only enter traps in conversion or calculation instructions, and do not enter traps in MOV instructions

trap.

When the trap is enabled: the original operation data is not sent.

Trap Disabled: This trap cannot be disabled.