LOONGSON

Loongson 3A1000 processor user's manual

Part ii

GS464 processor core V1.4

In October 2015

Loongson technology co. LTD

北京市海淀区温泉镇中关村环保科技示范园龙芯产业园2号楼 100095
Loongson Industrial Park, building 2，Zhongguancun environmental protection park
Haidian District，Beijing
www.loongson.cn

Copyright statement

disclaimer

This document only provides periodic information, and the content can be updated at any time according to the actual situation of the product without prior notice. The company shall not be liable for any direct or indirect loss caused by improper use of the documents.

Loongson technology co. LTD

Loongson Technology Corporation Limited
Address: Loongson Industrial Park, Building No. 2, loongson Industrial Park, zhongguancun environmental science and technology demonstration Park, haidian district, Beijing
Zhongguancun Environmental Protection Park, Haidian District, Beijing
Tel: 010-62546668 Fax: 010-62600826

Reading guide

Loongson 3A1000 processor user manual is divided into volume 1 and volume 2.

Loongson 3A1000 processor user's manual is divided into two parts. The first part (chapter 1 ~ chapter 10) introduces loongson 3A1000 multi-core processor architecture and register description, and gives a detailed description of chip system architecture, functions and configuration of main modules, register list and bit domain.The second part (chapter 11 ~ chapter 16) is the system software programming guide, which introduces the common problems in the development of BIOS and operating system.

The second volume of loongson 3A1000 processor user manual introduces in detail the GS464 high-performance processor core adopted by loongson 3A1000 from the perspective of system software developers.

目录

Revision history

Document update record	The document name:	Loongson 3A1000 processor user's manual - part ii
	The version number	V1.4
	The founders:	Research and development center
	Date of creation:	2015-10-08

Update history

The serial number	Updated date	Update one	The version number	Update the content
1	2011-06-24	Research and development center	V1.0	To complete the first draft of the
2	2011-10-18	Research and development center	V1.1	Review and proofread
3	2011-11-24	Research and development center	V1.2	Modify the cover
4	2012-04-28	Research and development center	V1.3	Fixed a 48-bit address space related error in chapter 5, "memory management"
5	2015-10-08	Research and development center	V1.4	Modify the instruction description in chapters 2 and 7

Manual information feedback: service@loongson.cn

1 Summary of structure

The GS464 is a general-purpose RISC processor IP that implements the 64-bit MIPS64 instruction set.GS464's instruction pipeline decoded four instructions per clock cycle and dynamically transmitted them to five all-flowing functional units.Although instructions are executed out of order on the premise of ensuring dependencies, they are delivered in the original order of the program to ensure precise exception and access order.

The superscalar structure of four emission makes the problems related to instruction and data in the instruction pipeline very prominent. GS464 adopts out-of-order execution technology and radical storage system design to improve the efficiency of the pipeline.

The techniques of out-of-order execution include register renaming, dynamic scheduling and transfer prediction. Register renaming solves the WAR (read after write) and WAW (write after write) correlation, and is used for accurate field recovery caused by exception and error transition predictions. GS464 renaming fixed point and floating point registers through a 64-item physical register heap, respectively.Dynamic scheduling according to the instruction operand ready order rather than instructions in a program in order to carry out instructions, reduces the RAW (writing after reading) caused by obstruction, GS464 has a 16 fixed-point retain stand and a 16 floats used in order to launch, and through a 64 item of the Reorder queue (hereinafter referred to as ROQ) to realize random sequence of instructions submitted in accordance with the procedure of the order.Transfer prediction reduces the blocking caused by control correlation by predicting whether the transfer instruction jumps successfully. GS464 USES Branch Target Buffer (BTB) of 16 items, Branch History Table (BHT) of 2K items, Global Histiry Registor (GHR) of 9 bits,

And the Return Address Stack (RAS) of four items.

GS464 advanced storage system design can effectively improve the efficiency of the pipeline.GS464's first-level Cache consists of a 64KB instruction Cache and a 64KB data Cache.GS464 has 64 TLBS in a fully associative structure, each of which maps to an odd page and an even page, with page sizes ranging from 4KB to 16MB. GS464 solves address dependency dynamically by means of the 24 item retrieval Queue and the 8 item retrieval failure Queue, and realizes the out-of-order execution of retrieval operation, non-blocking Cache, Load Speculation and other retrieval optimization techniques. GS464 supports 128-bit memory access, with 48 bit virtual and physical addresses.

The GS464 has two fixed-point features and two floating point features. Each floating point part can execute 64 in full flow

Double-precision floating-point multiplication and addition operations, and through the extension of the FMT domain of floating-point instructions to execute 32-bit and 64-bit fixed-point instructions.

GS464 supports the EJTAG debugging specification of MIPS company, and adopts the standard AXI interface. Its instruction Cache implements parity check and data Cache implements ECC check.The above features can increase the applicability of GS464.

The basic pipeline of GS464 includes taking the index, pre-decoding, decoding, register renaming, scheduling, transmitting, reading register,

There are 9 levels of execution and submission, and each level of flow includes the following operations.

Fetch refers to the pipeline level with the program counter PC value to access the instruction Cache and instruction TLB, if the instruction Cache

When both TLB and TLB are hit, four new instructions are fetched into the instruction register IR.

Predecoding stream level mainly decodes the transfer instruction and predicts the direction of the jump.

The decoding stream level converts the four instructions in the IR into the internal instruction format of GS464 and sends them to the register rename module.

Register renaming flow level allocates a new physical register for the logical target register and maps the logical source register to the physical register most recently assigned to the logical register.

The scheduling flow level will assign the renamed instruction to a fixed point or floating point reservation station for execution, and send it to ROQ for sequential submission after execution.In addition, transfer instruction and access instruction are sent to transfer queue and access queue respectively.

The transmitting stream stage selects an instruction prepared for all operands from a fixed point or floating point reservation station for each feature;An instruction whose operands are not ready at rename time waits for its operands to be ready by listening for the result bus and the forward bus.

The read register flow level is where the transmitted instruction reads the corresponding source operand from the physical register heap to the corresponding functional unit.

The execution flow level executes the instruction according to the type of instruction and writes the result back to the register heap.The resulting bus is also sent to the reserved station and register renaming tables to notify that the corresponding register values are available.

Submit the order of the water level in accordance with the procedures the Reorder queue record submit instruction has been performed, most GS464 per picture can submit four instruction, submitted instructions to register renaming table to confirm the destination register

renaming of relationships and release the originally assigned to the same physical register of logic registers, and sent to fetch queue allows those submitted by the number of instructions or write to the Cache memory.

The above is the flow level of basic instructions. For some more complex instructions, such as fixed-point multiplication and division instructions, floating point instructions and memory access instructions, multi-beat is required in the execution stage.The basic structure of GS464 is shown in the figure below.

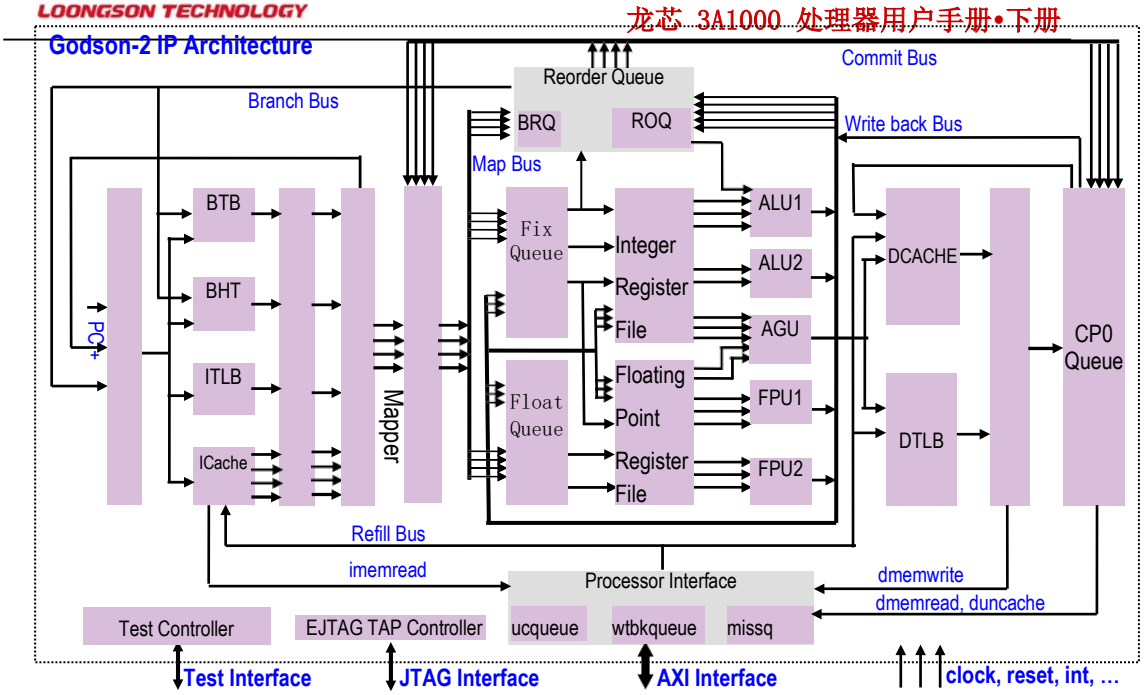


Figure 1-1 basic structure of GS464 processor

2 Overview of core instruction set for loongson GS464 processor

The loongson GS464 processor core is compatible with the MIPS64 R2 architecture, which implements the full set of required instructions defined by the MIPS64 R2 specification, as well as some custom extension instructions.

The MIPS64 R2 compatible instructions implemented by the loongson GS464 processor core are listed in section 2.1.Due to the length, the detailed definition of this part of the instruction is not given in this document.The reader needs to know the detailed definition of the instructions and is advised to refer to volume I and volume II of version 2.50 of the MIPS architecture specification.The implementations-related contents in this part of the instructions will be explained in section 2.2. In addition, the implementation of several MIPS64 R2 instructions GS464 processor core is not consistent with the MIPS architecture specification, which is also explained in section 2.2.

The custom extension instructions implemented by the loongson GS464 processor core are listed in section 2.3.Due to the length, the detailed definition of this part of the instruction is not given in this document.Readers really need to understand the detailed definition of the relevant instructions, it is recommended to consult the loongson instruction system manual.Loongson command system manual is currently available only to authorized customers.

2.1 MIPS64 compatible instruction list

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

- **To fetch instruction**
- **Operation instruction**
- **Branch and jump instructions**
- **Coprocessor instruction**
- **Other instructions**

2.1.1 To fetch instruction

The MIPS architecture USES a load/store architecture. All operations are performed on the register, and only the access instruction can access the data in main memory.Access

instruction includes reading and writing, unsigned reading, unaligned access and atomic access of data of various widths.

Table 2-1 CPU instruction set: memory access instruction

Instruction mnemonics	Command function description	ISA compatibility level
LB	In bytes	MIPS32
LBU	Take the unsigned byte	MIPS32

Instruction mnemonics	Command function description	ISA compatibility level
LH	Take half word	MIPS32
LHU	Take an unsigned half word	MIPS32
LW	Take the word	MIPS32
LWU	Take an unsigned word	MIPS32
LWL	Take words left	MIPS32
LWR	Take the words right	MIPS32
LD	Take double word	MIPS64
LDL	Take the left part of the word	MIPS64
LDR	Take the right part of the word	MIPS64
LL	Take the address of the mark	MIPS32
LLD	Take the double word address of the mark	MIPS64
SB	Remaining bytes	MIPS32
SH	Save half word	MIPS32
SW	characters	MIPS32
SWL	Characters left	MIPS32
SWR	To save words right	MIPS32
SD	Save double word	MIPS64
SDL	Save the left part of the double word	MIPS64
The SDR	Save the right part of the double word	MIPS64
SC	I'm going to save it	MIPS32
SCD	Save two words under the condition	MIPS64

2.1.2 Operation instruction

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

Table 2-2 CPU instruction set: arithmetic instruction (ALU instant number)

Instruction mnemonics	Command function description	ISA compatibility level
ADDI	Add number immediately	MIPS32

DADDI	Add a double word instant number	MIPS64

Instruction mnemonics	Command function description	ISA compatibility level
ADDIU	Add an unsigned immediate number	MIPS32
DADDIU	Add an unsigned word immediate number	MIPS64
SLTI	Less than immediate number setting	MIPS32
SLTIU	Unsigned less than immediate number setting	MIPS32
ANDI	And immediately	MIPS32
ORI	Or the number immediately	MIPS32
XORI	Xor immediate number	MIPS32
LUI	Count immediately to a high position	MIPS32

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

Instruction mnemonics	Command function description	ISA compatibility level
The ADD	add	MIPS32
DADD	Double word plus	MIPS64
ADDU	Unsigned add	MIPS32
DADDU	Unsigned double word plus	MIPS64
SUB	Reduction of	MIPS32
DSUB	Double word cut	MIPS64
SUBU	Unsigned reduction	MIPS32
DSUBU	Unsigned double minus	MIPS64
SLT	Less than the set	MIPS32
SLTU	Unsigned less than setting	MIPS32
The AND	with	MIPS32
The OR	or	MIPS32
XOR	Exclusive or	MIPS32
NOR	Or not	MIPS32

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

Instruction mnemonics	Command function	ISA compatibility

	description	level
CLO	The number of words leading to 1	MIPS32
DCLO	Double word leader 1 number	MIPS64
CLZ	The number of leading characters is 0	MIPS32
DCLZ	The number of words leading to 0	MIPS64
WSBH	Half-word byte swapping	MIPS32 R2
DSHD	Word half word exchange	MIPS64 R2
SEB	Byte symbol extension	MIPS32 R2
SEH	Half-character extensions	MIPS32 R2
INS	An insert	MIPS32 R2
EXT	An extract	MIPS32 R2
DINS	Binary insertion	MIPS64 R2
DINSM	Binary insertion	MIPS64 R2
DINSU	Binary insertion	MIPS64 R2
DEXT	Binary bit extraction	MIPS64 R2
DEXTM	Binary bit extraction	MIPS64 R2
DEXTU	Binary bit extraction	MIPS64 R2

Table 2-5 CPU instruction sets: multiplication and division instructions

Instruction mnemonics	Command function description	ISA compatibility level
The MUL	Multiply to the general register	MIPS32
MULT	take	MIPS32
DMULT	Double word by	MIPS64
MULTU	Unsigned by	MIPS32
DMUL TU	Unsigned double word multiplication	MIPS64
MADD	By adding	MIPS32
MADD U	Unsigned multiplication and addition	MIPS32
MSUB	By reducing	MIPS32
MSUBU	Unsigned multiplication and subtraction	MIPS32
DIV	In addition to	MIPS32
DDIV	Double word except	MIPS64
DIVU	Unsigned except	MIPS32
DDIVU	Unsigned double word division	MIPS64

MFHI	Fetch from the hi register to the universal register	MIPS32
MTHI	Save from the general purpose register to the hi register	MIPS32
MFLO	Take the number from register lo to the general register	MIPS32
MTLO	Save from the general register to the lo register	MIPS32

Table 2-6 CPU instruction set: shift instruction

Instruction mnemonics	Command function description	ISA compatibility level
SSL	The logical left	MIPS32
SRL	Logic moves to the right	MIPS32
SRA	Arithmetic moves to the right	MIPS32
SLLV	Variable logic moves left	MIPS32
SRLV	Variable logical shift to the right	MIPS32
SRAV	Variable arithmetic shift to the right	MIPS32
ROTR	Cycle moves to the right	MIPS32 R2
ROTRV	Variable cycle shift to the right	MIPS32 R2
DSLL	Double word logic moves left	MIPS64
DSRL	Double word logic moves to the right	MIPS64
DSRA	Double word arithmetic shift to the right	MIPS64
DSLLV	Variable double word logic moves left	MIPS64
DSRLV	Variable word logic right shift	MIPS64
DSRAV	Variable double word arithmetic shift to the right	MIPS64
DSLL32	Double word logic moves left +32	MIPS64
DSRL32	Double word logic moves to the right +32	MIPS64
DSRA32	Double word arithmetic shift to the right +32	MIPS64
DROTR	The word cycle moves to the right	MIPS64 R2
DROTR 32	Double word loop moves right +32	MIPS64 R2
DROTRV	Double word variable loop right shift	MIPS64 R2

2.1.3 Branch and jump instructions

Branch and jump instructions can change the control flow of a program, including the

following four types:

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

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

Table 2-7 CPU instruction sets: jump and branch instructions

Instruction mnemonics	Command function description	ISA compatibility level
J	jump	MIPS32
JAL	Immediate count calls the procedure	MIPS32
JR,	Jump to the instruction that the register points to	MIPS32
JR. HB	Jump to the instruction that the register points to	MIPS32 R2
JALR	Register call procedure	MIPS32
JALR. HB	Register call procedure	MIPS32 R2
BEQ	Equal jump	MIPS32
BNE	Unequal jump	MIPS32
BLEZ	Less than or equal to 0 jumps	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 to call the procedure	MIPS32
BGEZAL	Greater than or equal to 0	MIPS32
BEQL	Equal is Likely jump	MIPS32
BNEL	Not equal, Likely jump	MIPS32
BLEZL	Likely jump if less than or equal to 0	MIPS32
BGTZL	Greater than 0 is Likely jump	MIPS32
BLTZL	Likely jump if less than 0	MIPS32

BGEZL	Likely jump if greater than or equal to 0	MIPS32
BLTZALL	Less than 0 is Likely to call the procedure	MIPS32
BGEZALL	If greater than or equal to 0, Likely calls the procedure	MIPS32

2.1.4 Coprocessor instruction

The coprocessor instruction completes the operation inside the coprocessor. The loongson GS464 processor core has two coprocessors: coprocessor no. 0 (system processor) and coprocessor no. 1 (floating point coprocessor).

The zero coprocessor (CP0) manages memory and handles exceptions through the register of CP0. These instructions are listed in table 2-8

In the.

The MIPS architecture specification clearly defines floating point instructions in coprocessor no. 1 (CP1) instructions. The godson

The implementation of floating point instructions in the coprocessor 1 instruction in the GS464 processor core is described separately in chapter 7.

Table 2-8 CPU instruction set: CP0 instruction

Instruction mnemonics	Command function description	ISA compatibility level
DMFC0	Fetch a double word from register CP0	MIPS64
DMTC0	Write a double word to register CP0	MIPS64
MFC0	From the CP0 register	MIPS32
MTC0	Write to register CP0	MIPS32
TLBR	Read the TLB entry of the index	MIPS32
TLBWI	Write the TLB entry of the index	MIPS32
TLBWR	Let me write random TLB terms	MIPS32
TLBP	Search for matches in TLB	MIPS32
The CACHE	Cache operation	MIPS32
ERET	Abnormal return	MIPS32
DI	Disabling interrupts	MIPS32 R2

EI	Allow the interrupt	MIPS32 R2

2.1.5 Other instructions

In MIPS64, in addition to the above instructions listed above, there are some other instructions, as shown in table 2-9 to table 2-12:

Table 2-9 CPU instruction sets: special instructions

Instruction mnemonics	Command function description	ISA compatibility level
The SYSCALL	The system calls	MIPS32
BREAK	The breakpoint	MIPS32
The SYNC	synchronous	MIPS32
SYNCI	Synchronous instruction cache	MIPS32 R2

Table 2-10 CPU instruction sets: exception instructions

Instruction mnemonics	Command function description	ISA compatibility level
TGE	Greater than or equal to entrapped	MIPS32
TGEU	Unsigned Numbers greater than or equal to entrapped	MIPS32
TLT	Less than a	MIPS32

Instruction mnemonics	Command function description	ISA compatibility level
TLTU	The unsigned number is less than entrapped	MIPS32
TEQ	Is equal to a	MIPS32
TNE	Differ in	MIPS32
TGEI	Greater than or equal to immediately number trapped	MIPS32
TGEIU	Greater than or equal to an unsigned immediate number	MIPS32
TLTI	Less than immediately the number is trapped	MIPS32
TLTIU	Less than an unsigned number immediately sinks in	MIPS32
TEQI	Is equal to immediately the number sinks in	MIPS32
TNEI	Not equal to immediately number into	MIPS32

Table 2-11 CPU instruction set: conditional move instructions

Instruction mnemonics	Command function description	ISA compatibility level
MOVF	The condition moves when the floating point condition is false	MIPS32
MOVN	The condition is moved when the universal register is not 0	MIPS32
MOVT	The condition moves when the floating point condition is true	MIPS32
MOVZ	The condition moves when the general register is 0	MIPS32

Table 2-12 CPU instruction sets: other instructions

Instruction mnemonics	Command function description	ISA compatibility level
PREF	Prefetching instructions	MIPS32
PREFX	Prefetching instructions	MIPS32
The NOP	Empty operation	MIPS32
SSNOP	Single launch air operation	MIPS32

2.2 MIPS64 compatible instruction implementation related instructions

2.2.1 There are instructions for implementation differences

The loongson GS464 processor checks all MIPS64 R2 instructions for support, but

redefines some implementation-related instructions, as shown in table 2-13.

There are instructions for implementation differences in table 2-13

Instruction mnemonics	Command function description	Concrete implementation description
PREF	Prefetching instructions	As NOP instruction processing, no prefetch effect. Software prefetch can be achieved by Load to register 0.
PREFX	Prefetching instructions	As NOP instruction processing, no prefetch effect. Software prefetch can be achieved by Load to register 0.
SSNOP	Single launch air operation	Treat as NOP instruction processing. All data and control Hazards in GS464 are maintained by the hardware without software control.
EHB	Isolation execution correlation	Treat as NOP instruction processing. All data and control Hazards in GS464 are maintained by the hardware without software control.
WAIT	Enter a waiting state	As NOP instruction processing, no pipeline stop effect. Avoid using this instruction in code that involves low-power management, which not only fails to achieve what the software design expects, but may even cause increased power consumption.
RDPGPR	Read shadow register	GS464 does not implement a shadow register, so both the source register and the target register belong to the current register group.
WRPGPR	Write shadow register	GS464 does not implement a shadow register, so both the source register and the target register belong to it In the current register group.
The SYNC	synchronous	GS464 only implements the SYNC instruction for stype=0, except for the SYNC instruction reserved for other values of stype. This instruction ACTS as a memory barrier to ensure that the access operation before SYNC has been confirmed (e.g., the data of the store instruction is written to dcache, the read and write of uncached has been completed, and the value has been retrieved to the register by load), and the access operation after SYNC instruction has not been started.The SYNC The instruction requires CU[0] and is available only in kernel mode.

Instruction mnemonics	Command function description	Concrete implementation description
The CACHE	Cache operation	There are two main differences between the CACHE instruction implemented by GS464 and the MIPS64 specification: **1. Address resolution method of Index CACHE instruction** The index CACHE instruction implemented by GS464 takes the lowest two bits of the virtual address as the selection signal of the route to the CACHE, instead of intercepting from the middle of the virtual address as defined in the MIPS64 specification. **2, CACHE28, CACHE29, CACHE30, CACHE31 instructions containing** **The righteous** In processors, CACHE28, CACHE29, CACHE30, and CACHE31 fingersIn processors, the CACHE28, CACHE29, CACHE30, and CACHE31 instructions (that is, the op[4:2]=0b111 Cache instruction) have different meanings from the MIPS64 specification. These instructions in loongson processor are Index Store Data operations, while those in MIPS64 specification are Fetch and Lock operations. These instructions in loongson processor are Index Store Data operations, while those in MIPS64 specification are Fetch and Lock operations. The valid Cache operation types in the GS464 processor core are listed below: **Op [4:0] function description** b10000 Invalid i-cache row based on index b101000 Write the i-cache line Tag according to the index b11100 writes i-cache line Data b00001 according to the index to read d-cache line Tag b01001 according to the index to read d-cache line Tag b10001 according to the index to read d-cache line Tag b10101 according to the indexD-cache line b11001 reads d-cache line Data b11101 reads d-cache line Data b00011 reads s-cache line

		b00111 reads s-cache line Tag b01011 reads s-cache line Tag b10011 reads s-cache line b11011 reads s-cache line Data according to the index b11111 writes the s-cache row Data according to the index

2.2.2 Disable the instruction

GS464 processor core disables the following instructions, as shown in table 2-14:

Table 2-14 disable instructions

Instruction mnemonics	Command function description	ISA compatibility level
DI	Disabling interrupts	MIPS32 R2
EI	Allow the interrupt	MIPS32 R2

2.3 Custom extension instructions

The custom extension instructions implemented by loongson GS464 processor are divided into the following categories according to their functions:

- **To fetch instruction**
- **Multiplication and division instruction**
- **X86 binary acceleration instruction**
- **64 bit multimedia instructions**
- **Miscellaneous instruction**

2.3.1 Custom extended access instructions

Table 2-15 custom extended access instructions

Instruction mnemonics	Command function description
GSLE	The exception is if less than or equal to set the address wrong
GSGT	The exception is if the address is not correct
GSLBLE	Fetch bytes with an out - of - bounds check
GSLBGT	Fetch byte with down - bounds check
GSLHLE	Bring the cross - check half word
GSLHGT	Take down the half word for cross - check
GSLWLE	Bring the word for crossing the line
GSLWGT	Take down the character for cross - check
GSLDLE	Bring the double word for crossing the line
GSLDGT	Take down the cross - border check take double word
GSLQ	The dual target register takes a fixed point quadword
GSLBX	Fetch byte with offset
GSLHX	Take half a word with offset
Instruction	**Command**

mnemonics	function description
GSLWX	An offset fetch
GSLDX	Take two words with offset
GSSBLE	Memory bytes with overbounds checking
GSSBGT	Memory byte with down - bounds check
GSSHLE	Bring the half word with you for cross - check
GSSHGT	Save half a word with down crossing check
GSSWLE	Bring your checkbox with you
GSSWGT	Save the word with down - crossing check
GSSDLE	Bring double characters for cross - check
GSSDGT	Save double characters with down - crossing check
GSSQ	The dual source register stores the fixed point four words
GSSBX	Memory bytes with offset
GSSHX	Half word with offset
GSSWX	Memory with offset
GSSDX	Save two words with offset

2.3.2 Custom extended multiplication and division operation instruction

Table 2-16 custom extended multiplication and division instructions

Instruction mnemonics	Command function description
GSMULT	With a signed word multiplication, the result is written to the universal register
GSDMULT	Signed double word multiplication, the result is written to the universal register
GSMULTU	Unsigned word multiplication, the result is written to the universal register
GSDMULTU	Unsigned double word multiplication, the result is written to the universal register
GSDIV	Signed word division, quotient write universal register
GSDDIV	Signed double word division, quotient write universal register
GSDIVU	Unsigned word division, quotient write universal register
GSDDIVU	Unsigned double word division, quotient write universal register
GSMOD	The remainder of the signed word is written to the universal register
GSDMOD	The remainder is written to the universal register
GSMODU	The remainder of the unsigned word is written to the universal register

Instruction mnemonics	Command function description
GSDMODU	The remainder is written to the universal register

2.3.3 Custom extensions to X86 binary translation acceleration instructions

Table 2-17 custom extensions X86 binary translation acceleration instructions

Instruction mnemonics	Command function description
X86AND	Only the logical bits of EFLAG are set in x86 mode
X86OR	Only the logical bits of EFLAG or are set in x86 mode
X86XOR	Only logical bits of EFLAG are set in x86 mode
X86DADD	In x86 mode, only double-word adders of EFLAG are set
X86ADD	Only word adders of EFLAG are set in x86 mode
X86DADDU	No exception for double-word addition with EFLAG only set in x86 mode
X86ADDU	No exception word addition that sets EFLAG in x86 mode only
X86DSUB	Sets only the double decrement of EFLAG in x86 mode
X86SUB	Only EFLAG subtractions are set in x86 mode
X86DSUBU	No exception for double-word decrement with EFLAG set only in x86 mode
X86SUBU	No exception word subtractions that only set EFLAG in x86 mode
X86INC	The double word with EFLAG in x86 mode is self-increment 1
X86DEC	Only two words that set EFLAG in x86 mode are self-decrement 1
X86DSLL	Only double-word moves left of EFLAG in x86 mode
X86SLL	Only words that set EFLAG in x86 mode move left
X86DSLL32	In x86 mode, only the shift amount of EFLAG plus 32 is set to the left of the doubleword logic
X86DSLLV	In x86 mode, only the two-word variable shift of EFLAG moves to the left
X86SLLV	The variable shift amount of a word that only sets EFLAG in x86 mode moves to the left
X86DSRL	Double-word logic that only sets EFLAG in x86 mode moves to the right
X86SRL	Word logic that sets EFLAG in x86 mode moves only to the right
X86DSRL32	In x86 mode, only the shift amount of EFLAG plus 32 double word logic moves to the right
X86DSRLV	In x86 mode, only the two-word variable shift quantity logic of EFLAG moves to the right
X86SRLV	Logical right shift of the word variable shift that only sets EFLAG in x86 mode
X86DSRA	Double-word arithmetic shift with EFLAG only set in x86 mode

X86SRA	Arithmetic righting of EFLAG words in x86 mode only

Instruction mnemonics	Command function description
X86DSRA32	In x86 mode, only the shift amount of EFLAG plus 32's logical arithmetic shift to the right is set
X86DSRAV	Set EFLAG in x86 mode only for binary variable shift arithmetic shift to the right
X86SRAV	In x86 mode, only EFLAG's word variable shift quantity is set arithmetic right shift
X86DROTR	Double-word loops that only set EFLAG in x86 mode are shifted to the right
X86ROTR	In x86 mode, only EFLAG word loops are set to shift to the right
X86DROTR32	In x86 mode, only the shift amount of EFLAG plus 32 is set to the right of the doubleword logic loop
X86DROTRV	In x86 mode, only the variable shift of EFLAG is set to the right of the double-word loop
X86ROTRV	Word cycles that only set EFLAG's variable shift amount to the right in x86 mode
X86MFFLAG	EFLAG flag bit values are extracted in x86 mode
X86MTFLAG	Modify the value of the EFLAG flag bit in x86 mode
X86J	Jumps on EFLAG values in x86 mode
X86LOOP	Loops in x86 mode based on EFLAG values
SETTM	X86 floating point stack mode Settings
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	Writes the x86 floating point stack top pointer
MFTOP	Read the x86 floating point stack top pointer
SETTAG	Determine the collocation register

2.3.4 Custom extensions for 64-bit multimedia acceleration instructions

Table 2-18 custom extensions for 64-bit multimedia acceleration instructions

Instruction mnemonics	Command function description
PADDSH	Four 16-bit signed integers plus, signed saturation
PADDUSH	Four 16-bit unsigned integers plus, unsigned saturation
PADDH	Four 16-digit Numbers plus
PADDW	Two 32-digit Numbers plus
PADDSB	Eight 8-bit signed integers plus, signed saturation

PADDUSB	Eight 8-bit unsigned integers plus, unsigned saturation
PADDB	Eight eight-digit Numbers plus

Instruction mnemonics	Command function description
PADDD	64 digits add
PSUBSH	Four 16-bit signed integers minus, signed saturation
PSUBUSH	Four 16-bit unsigned integers minus, unsigned saturation
PSUBH	Four 16-digit Numbers minus
PSUBW	Two 32-digit Numbers minus
PSUBSB	Eight 8-bit signed integers minus, signed saturation
PSUBUSB	Eight 8-bit unsigned integers minus, unsigned saturation
PSUBB	Eight eight-digit subtractions
PSUBD	64 digits
PSHUFH	Mix and wash four 16-digit Numbers
PACKSSWH	A 32-bit signed integer is converted to a 16-bit signed saturation
PACKSSHB	A 16-bit signed integer is converted to an 8-bit signed saturation
PACKUSHB	16 - bit signed integer converted to 8 - bit, unsigned saturation
PANDN	Fs and ft are bitwise
PUNPCKLHW	Unpacking is 16 digits low
PUNPCKHHW	Unpacking is 16 digits high
PUNPCKLBH	Unpacking is low by 8 digits
PUNPCKHBH	Unpacking high by 8 digits
PINSRH_0	Ft low 16 bits insert into fs low 0 16 bits
PINSRH_1	Ft low 16 bits insert into fs low 1 16 bits
PINSRH_2	Ft low 16 bits insert into fs low 2 16 bits
PINSRH_3	Ft low 16 bits inserted into fs low 3 16 bits
PAVGH	Four 16-bit unsigned integers are averaged
PAVGB	Eight 8-bit unsigned integers are averaged
PMAXSH	Take the larger value of four 16-bit signed integers
PMINSH	Take the smaller value of four 16-bit signed integers
PMAXUB	Take the larger value of eight 8-bit unsigned integers
PMINUB	Eight 8-bit unsigned integers take smaller values
PCMPEQW	Two 32 digits are equal to each other
PCMPGTW	Two 32-bit signed integers are greater than the comparison
PCMPEQH	Four 16 - digit comparisons are equal

PCMPGTH	Four 16-bit signed integers are greater than the comparison
PCMPEQB	Eight 8-digit Numbers are equally compared

Instruction mnemonics	Command function description
PCMPGTB	Eight 8-bit signed integers are greater than comparisons
PSLLW	Two 32-bit logic moves left
PSLLH	Four 16-digit logic moves left
PMULLH	Multiply four 16-bit signed integers and take the result to be 16 bits lower
PMULHH	Multiply four 16-bit signed integers and take the height of the result to be 16 bits
PMULUW	Multiply the lower 32-bit unsigned integers to store the 64-bit result
PMULHUH	Multiply four 16-bit unsigned integers to get the 16-bit height of the result
PSRLW	Two 32-bit logical moves to the right
PSRLH	Four 16-digit logical moves to the right
PSRAW	Two 32-digit arithmetic moves to the right
PSRAH	Four 16-digit arithmetic shifts to the right
PUNPCKLWD	The lower 32 digits are combined into 64 digits
PUNPCKHWD	The high 32 digits are combined into 64 digits
PASUBUB	Subtract eight 8-bit unsigned integers and take the absolute value
PEXTRH	Fs some 16 bit copy to fd low 16 bit, fd high complement 0
PMADDHW	Four 16-bit signed Numbers are multiplied and the lower and higher levels are added
BIADD	Multibyte accumulation
PMOVMSKB	Conditional byte shift
GSXOR	Fs and ft logical bit or
GSNOR	Fs and ft logical bit or non
GSAND	Fs and ft logical bit 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 character addition
GSDADD	Fs and ft fixed point double word addition
GSSEQU	Fs and ft are equal in number of fixed points
GSSEQ	Fs and ft are equal in number of fixed points
GSSUBU	Fs and ft fixed point unsigned word subtraction
GSSUB	Fs and ft fixed-point subtraction
GSDSUB	Fs and ft fixed point double word subtraction

GSSLTU	Fs and ft fixed points the number of unsigned fixed points is less than the comparison
GSSLT	Fs and ft fixed point number is less than comparison
GSSLL	Fs and ft fixed point logic left - shift words

Instruction mnemonics	Command function description
GSDSLL	Fs and ft fixed point logic left shift double word
GSSRL	Fs and ft fixed point logic right - shift the word
GSDSRL	Fs and ft fixed point logic right shift doublet
GSSRA	Fs and ft fixed point arithmetic right shift word
GSDSRA	Fs and ft fixed point arithmetic right shift doublet
GSSLEU	Fs and ft fixed points the number of unsigned fixed points is less than or equal to comparison
GSSLE	Fs and ft fixed point number is less than or equal to comparison

2.3.5 Custom extension miscellaneous directive

Table 2-19 custom extension miscellaneous instructions

Instruction mnemonics	Command function description
CAMPV	Query the lookup table to return the contents of the hit item
CAMPI	Query the lookup table to return the index of the hit item
CAMWI	Write the lookup table to specify the entry
RAMRI	Read the contents of the specified item in the lookup table

3 CP0 control register

This chapter describes the operation of Coprocessor 0 (CP0 for short), mainly including the register definition of CP0 and the CP0 instruction implemented by loongson 3 processor. The CP0 register is used to control state changes in the processor and report the current state of the processor. These registers are read by the MFC0/DMFC0 instruction or written by the MTC0/ DMTC0 instruction. The CP0 register is shown in table 3-1.

The CP0 instruction can be used when the processor is running in core mode or when bit 28 (CU0) in the Status register is set.Otherwise, executing the CP0 instruction will result in a "coprocessor unavailable exception."

Table 3-1 register CP0

The register no.		Register name	describe
Total no.	The child,		
0	0	The Index	A writable register used to specify TLB table entries that need to be read/written
1	0	The Random	Pseudorandom counter for TLB substitution
2	0	EntryLo0	The contents of the lower half of the TLB table entry corresponding to the even virtual page (mainly Physical page number)
3	0	EntryLo1	The contents of the lower half of the TLB table entry corresponding to the odd virtual page (mainly the physical page number)
4	0	The Context	Virtual page transformation table (PTE) pointing to the kernel in 32-bit addressing mode
5	0	Page Mask	Sets the mask value for the TLB page size
5	1	Page Grain	Whether the flag supports large page addresses
6	0	Wired	Number of fixed - wired TLB table entries (low - end TLB table entries that are not used for random substitution)
7	0	Hwrena	Hardware register enablement
8	0	BadVaddr	Incorrect virtual address
9	0	The Count	counter
10	0	EntryHi	The high half of a TLB table entry (dummy page number and ASID)
11	0	The Compare	Counter comparison
12	0	The Status	Processor status register
12	1	IntCtl	Extended interrupt control register
12	2	SRSCtl	The shadow register group controls the register
13	0	Cause	The reason for the latest exception
14	0	The EPC	Exception program counter
15	0	PRid	Processor revision version id
15	1	EBase	Exception vector base address
16	1	Config1	Configure register 1
16	2	Config2	Configure register 2
16	3	Config3	Configure register 3

17	0	LLAddr	Link read memory address
18			reserve
19			reserve
20	0	Xcontext	Virtual page transformation table (PTE) pointing to the kernel in 64-bit addressing mode
21			reserve
22	0	Diagnose	Enable/disable BTB,RAS and empty ITLB tables
23	0	The Debug	EJTAG debug register
24	0	DEPC	EJTAG debugging exception counters
25	0/1/2/3	PerfCnt	Performance counter
26	0	ErrCtl	Parity/ECC check control and state
27	0	CacheErr	Cache ECC validates control and status
28	0	TagLo	The lower half of the CACHE TAG register
28	1	DataLo	Used to interact with and diagnose cache data queues
29	0	TagHi	The high half of the CACHE TAG register
29	1	DataHi	Used to interact with and diagnose cache data queues
30	0	ErrorEPC	Error exception program counter
31	0	DESAVE	Registers, used to Debug exception handling

3.1 Index register (0,0)

The Index register is a 32-bit read/write register where the last six bits of the value are used to Index TLB table entries. The highest bit of the register indicates whether the TLB probe (TLBP) instruction was executed successfully.

The value of the Index register indicates TLB table entries operated by the TLB read (TLBR) and TLB Index write (TLBWI) instructions. Figure 3-1 shows the format of the Index register, and table 3-2 describes the meaning of each field of the Index register.

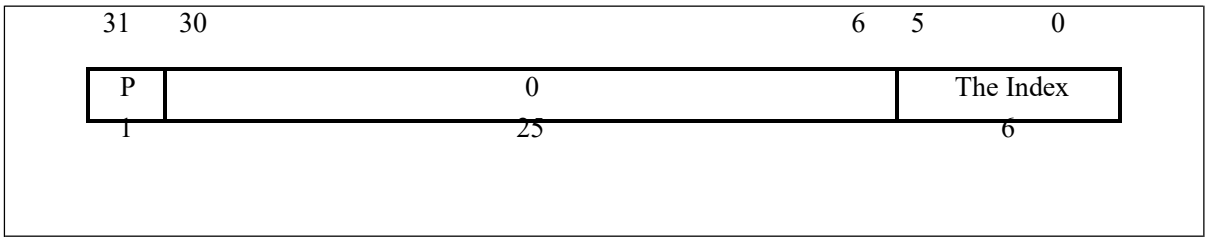


Figure 3-1 Index register

Table 3-2 describes the fields of the Index register

域	描述
P	The probe failed. Set 1 when TLB probe instruction (TLBP) failed last time
The Index	The index value of a TLB table entry that indicates the operation of a TLB read instruction and a TLB index write instruction

0	Retained. You must press 0 to write and return 0 on read.

3.2 Random register (1,0)

The Random register is a read-only register in which the table entries of the TLB are six digits lower. The register value is subtracted 1 for each instruction executed.Meanwhile, the register value floats between an upper bound 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 (that is, the contents of the Wired register).
- The upper bound is going to be the total number of terms of TLB minus 1 (64 minus 1 at most).

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

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

Figure 3-2 shows the format of the Random register, while table 3-3 describes the meaning of the fields of the Random register.

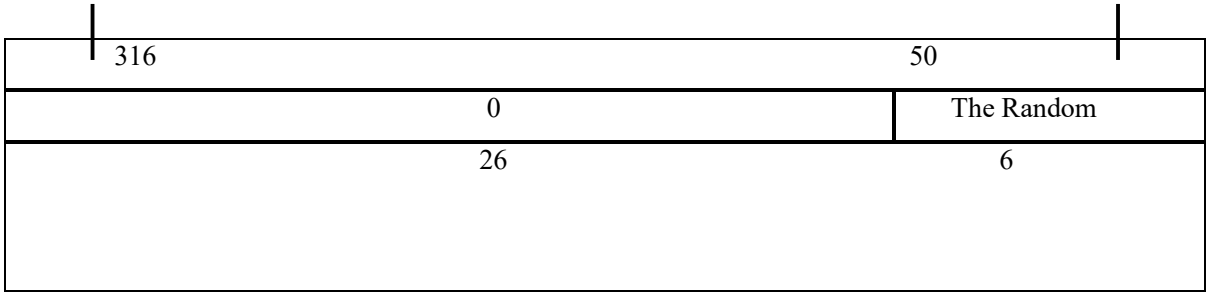


Figure 3-2 Random

register table 3-3 Random

The domain	describe
The Random	Random TLB index values
0	Retained. You must press 0 to write and return 0 on read.

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

The EntryLo register consists of two registers of the same format:

- EntryLo0 is used for even virtual pages
- EntryLo1 is used for odd virtual pages

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

Physical page number (PFN) of the odd and even pages in the item.Figure 3-3 shows the

format of these registers.

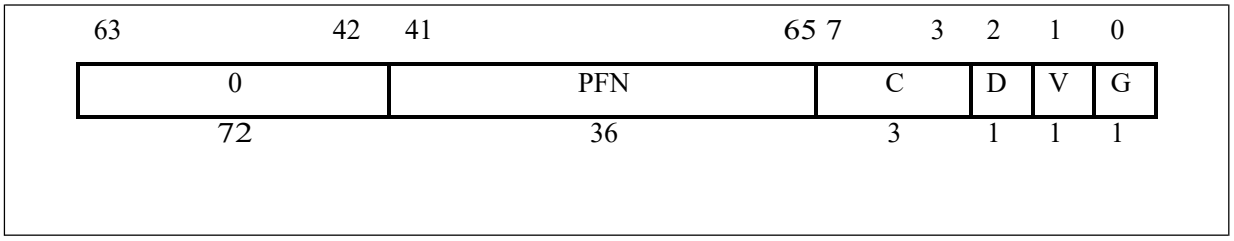


Figure 3-3 EntryLo0 and EntryLo1 registers

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

Table 3-4 EntryLo register fields

The domain	desc ribe
PFN	Page number, which is the high value of the physical address.
C	The Cache consistency property of the TLB page.
D	Dirty bits. If the bit is set, the page is marked as dirty, that is, writable. This bit is actually used in software as write protection against data being changed.
V	Significant bit. When this bit is set, the TLB table entry is valid, otherwise a TLBL or TLBS exception.
G	The global level. When the G bits in EntryLo0 and EntryLo1 are set to 1, the processor will be in TLB Ignore the ASID when searching.
0	Retained. You must press 0 to write and return 0 on read.

There is only one global bit in each TLB table entry, which is written based on the values EntryLo0[0] and EntryLo1[0] in a TLB write operation.

3.4 The Context (4, 0)

The Context register is a read/write register that contains a pointer to an item in the page table. The page table is an operating system data structure that stores the translation of virtual addresses to physical addresses.

When a TLB exception occurs, the CPU loads the TLB from the page table based on the failed conversion.In general, the operating system USES the Context register to address the mapping of the current page in the page table. The Context register copies some of the information in the BadVAddr register, but the information is arranged in a form that the software TLB exception handler can handle.

Figure 3-4 shows the format of the Context register;Table 3-5 describes the context register fields.

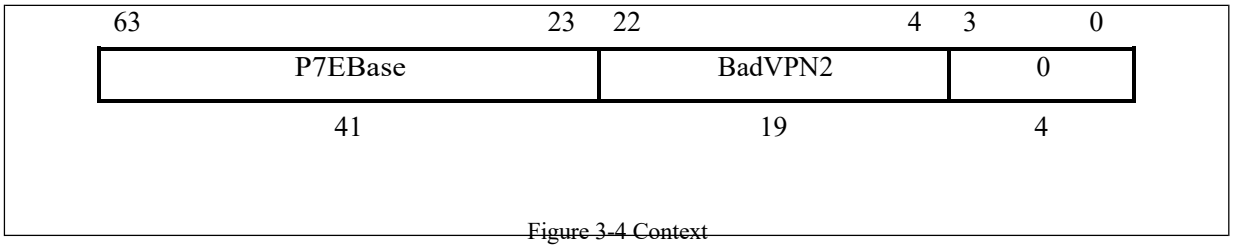


Figure 3-4 Context

The domain	desc ribe

register table 3-5 Context

register field

	(VPN).
PTEBase	This field is the read/write field used by the operating system.The value written to this field allows the operating system to send the Context Memory serves as a pointer to the current page table in memory.
0	Retained. You must press 0 to write and return 0 on read.

The 19-bit BadVPN2 field contains 31:13 bits of the virtual address that caused the TLB exception; Bit 12 is excluded because a single TLB entry maps to a parity page pair.For a page size of 4K bytes, this format can directly address page tables whose PTE table entries are 8 bytes long and organized against each other.For pages of other sizes and ptes, moving and masking this value can produce the appropriate address.

3.5 PageMask register (5,0)

The PageMask register is a read-write register used during TLB read-write; It contains a comparison mask that allows you to set different page sizes for each TLB table entry, as shown in table 3-6.The format of this register is shown in figure 3-5.

TLB read and write operations use this register as a source or destination;When doing virtual and real address translation, the corresponding bit in the TLB corresponding to the PageMask register indicates which bits in the virtual address bit 24:13 are used for comparison.TLB operations are undefined when the values of the MASK field are not the values in table 3-6.The 0 field is reserved and must be written to and returned to 0 on read.

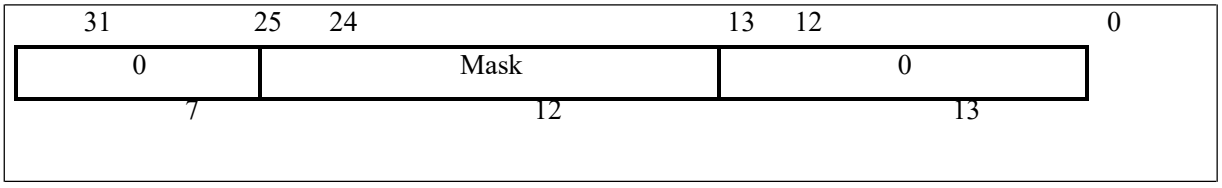


Figure 3-5. PageMask register

Table 3-6 Mask values for different page sizes

Page size	position											
	24	23	22	21	20	19	18	17	16	15	14	13
4 kbytes	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
16 m bytes	1	1	1	1	1	1	1	1	1	1	1	1

3.6 PageGrain register (5,1)

The PageGrain register is a read-write register, and the godson 3 only defines the 29th bit of this register: ELPA (Enable Large Physical Adress), with the remaining bits left at 0.

When ELPA=1, loongson 3 supports 48-bit physical address;When ELPA=0, loongson 3 only supports 40 physical addresses.Whether or not the ELPA bit can be written depends on the LPA field of the Config3 register.When the LPA bit of Config3 is 0, ELPA of PageGrain

The bit is set to 0.The format of the register is shown in figure 3-6, and the register fields are shown in table 3-7.

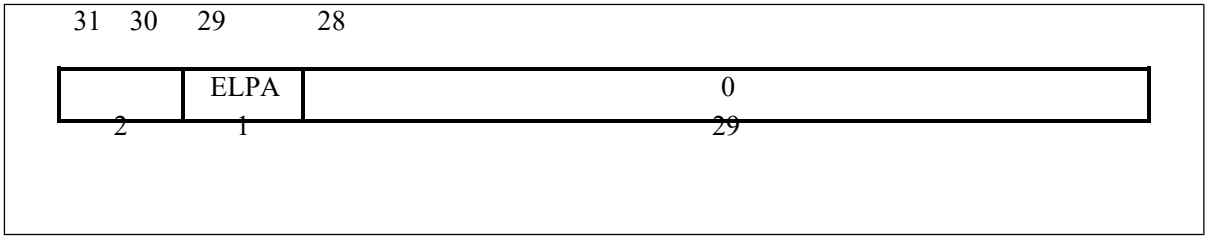


Figure 3-6 PageGrain

register table 3-7 PageGrain

register field

The domain	describe
ELPA	Whether the field is set indicates whether large physical addresses are supported
0	Retained. You must press 0 to write and return 0 on read.

3.7 Wired register (6,0)

The Wired register is a readable/written register whose value specifies the boundaries between fixed and random table items in TLB, as shown in figure 3-7. Wired table entries are fixed, non-replaceable table entries whose contents are not modified by TLB write operations. The contents of random table entries can be modified.

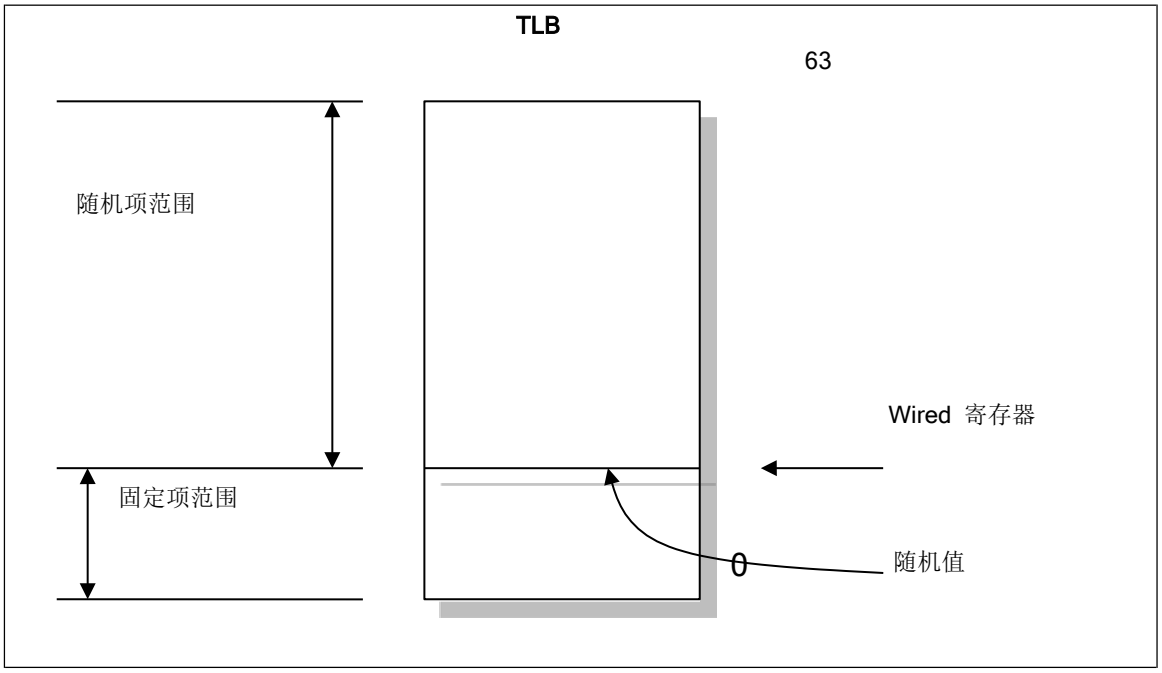


Figure 3-7. Wired register boundaries

The Wired register is set to 0 when the system is reset. When writing to this register, the value of the Random register is set to the upper limit (see the Random register above).

Figure 3-8 shows the format of the Wired register;Table 3-8 describes the domain of this register.

0	Wired
26	6

Figure 3-8 Wired

register table 3-8 Wired

The domain	describe
Wired	TLB fixes table item boundaries
0	Retained. You must press 0 to write and return 0 on read.

3.8 HWREna register (7, 0)

HWREna is a read/write registers, the godson 3 only defines the Mask this register domain, is used to represent instructions

The source hardware register for the RDHWR. The 0 field is reserved and must be written to and returned to 0 on read.

The format of the figure 3-9 shows the HWREna register;Table 3-9 describes the correspondence between the Mask field and the hardware register.

3174	30
0	Mask
28	4

Figure 3-9 HWREna registers

Table 3-9 correspondence between Mask field and hardware register

Hardware register	position			
	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 error virtual address register (BadVAddr) is a read-only register that records the last virtual address that resulted in a TLB or addressing error exception.The BadVAddr register will remain unchanged unless a software reset occurs, with the exception of an NMI or Cache error.Otherwise the register is undefined.

Figure 3-10 shows the format of the error virtual address register.

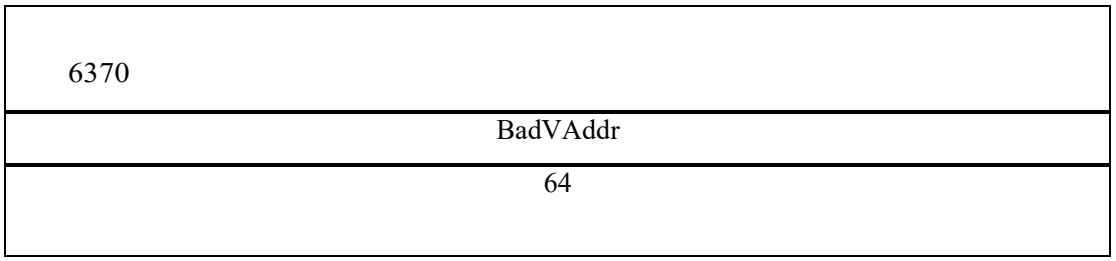


Figure 3-10 BadVAddr register

3.10 The Count register (9,0) and the Compare register (11,0)

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

The Count register works as a real-time timer, incrementing by 1 every two clock cycles.

The Compare register is used to generate an interrupt at a particular moment, which is written to a value and constantly compared to the value in the Count register. Once these two values are equal, an interrupt request is generated. The TI and IP[7] in Cause register are set. The TI bit of Cause register is reset when the Compare register is written again.

Figure 3-11 shows the format of the Count register.Figure 3-12 shows the format of the Compare register.

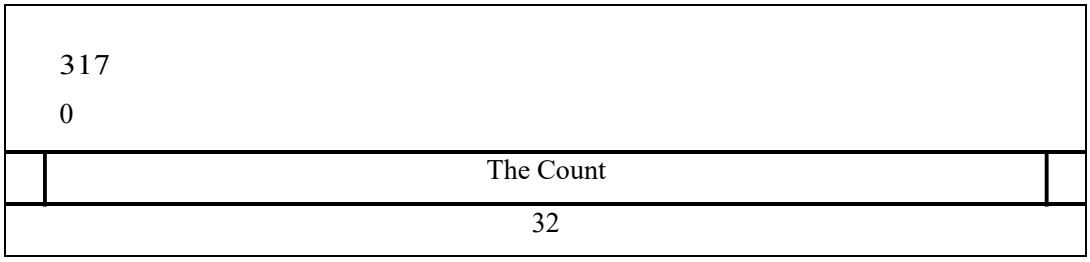


Figure 3-11 Count register

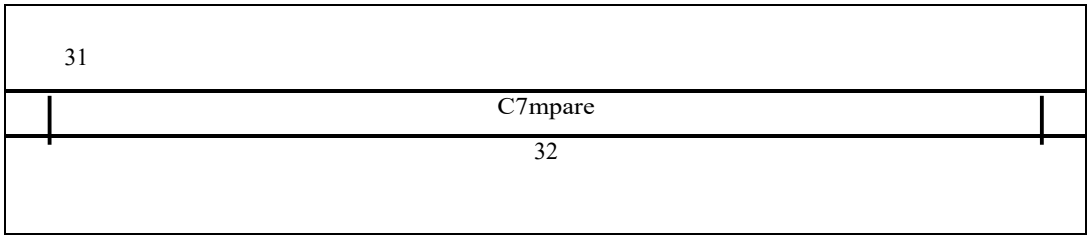


Figure 3-12 the Compare register

3.11 EntryHi register (10,0)

The EntryHi register is used to store the high level of TLB table entries when TLB reads and writes.

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

Instruction access.

Figure 3-13 shows the format of the EntryHi register.Table 3-10 represents the fields of the EntryHi register.

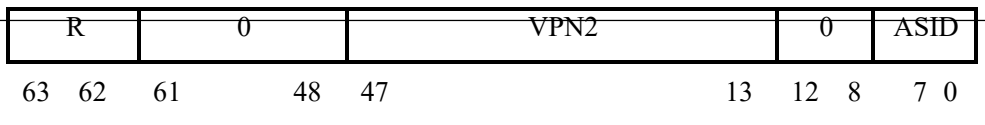


Figure 3-13 EntryHi register

2	14	35	5

Table 3-10 EntryHi register domains

The domain	describe
VPN2	Empty page number divided by 2 (maps to double pages);The high value of the virtual address.
ASID	The address space identifies the domain. An 8-bit field;Used to share TLB between multiple processes;Each process has a different mapping to the other processes for the same virtual page number.
R	Area. (00-> user, 01-> superuser, 11-> core) used to match vAddr63...62
0	Retained. You must press 0 to write and return 0 on read.

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

When a TLB Refill, TLB Invalid, or TLB Modified exception occurs, the virtual page number (VPN2) and ASID in the virtual address that does not match the TLB table entry are loaded into the EntryHi register.

3.12 Status register (12,0)

The Status register (SR) is a read-write register that includes operation mode, interrupt permission, and processor Status diagnosis. The following list describes some of the more important Status register fields;Figure 3-14 shows the format of the entire register, including a description of the domain.Among the important fields are:

The 8-bit interrupt mask (IM) domain controls the enabling of eight interrupt conditions.Interrupts must be enabled before they can be triggered, and the corresponding bits in the interrupt masking field of the Status register and the interrupt pending domain of the Cause register should be set.For more information, refer to the interrupt pending (IP) domain of the Cause register.

The 4-bit coprocessor availability (CU) domain controls the availability of four possible coprocessors.No matter how the CU0 bit is set, CP0 is always available in kernel mode.

Status register format

Figure 3-14 shows the format of the Status register, and table 3-11 describes the Status register field.

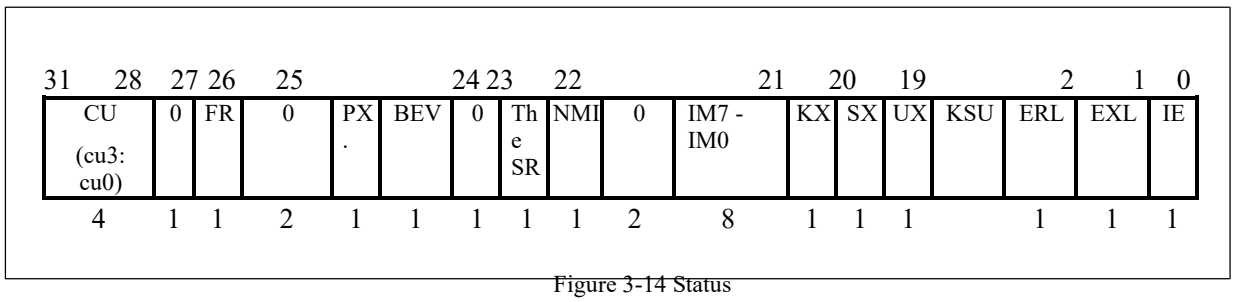


Figure 3-14 Status

register table 3-11 Status

The domain	describe
CU	Controls the availability of four coprocessor units.No matter how the CU0 bit is set, CP0 is always available in kernel mode. 1 - available 0 - is unavailable

FR	ShiNengFuJiaDeFuDianJiC unQi s 0 - 16 GeJiCunQi 1 -□32 GeJiCunQi
PX.	Enable 64-bit operations in user mode (64-bit operations in the remaining modes do not need to be enabled) 1 - can make 0 - not enabled (the UX bit needs to be determined if 64-bit operations are available in user mode at this point)
BEV	Control the entry address of the exception vector 0 Normal - 1 - start
The SR	1 indicates that a soft reset exception occurred
NMI	Whether an NMI exception occurs.Note that the software cannot write this from 0 to 1
IM	Interrupt masking: controls the enabling of each external, internal, and software interrupt. If the interrupt is enabled, it is allowed to trigger while the corresponding bit of the interrupt pending field of the Cause register is set. 0 to ban 1 - allows the
KSU	Mode bits 11 undefined 10 The average user 01 The super user 00 core
KX	1 -- enable 64-bit Kernel segment access; Use the XTLB Refill vector. 0 - cannot access 64-bit Kernel segment; Use the TLB Refill vector
SX	1 -- enable 64 Supervisor visits; Use the XTLB Refill vector. 0 -- cannot access the 64-bit Supervisor segment; Use the TLB Refill vector
UX	1 -- enable 64-bit User segment access; Use the XTLB Refill vector. 0 - cannot access 64-bit User segment; Use the TLB Refill vector
ERL	Error level. The processor resets this bit when a reset, software reset, NMI, or Cache error occurs.
EXL	The exception class. The handler is set when an exception is generated that is not caused by a reset, software reset, or Cache error The bit.
IE	Interrupt enablement.

Status register mode and access Status

The following fields in the Status register are used to set the mode and access the Status:

- **Interrupt enablement: an interrupt is enabled when:**

- And IE = 1

- EXL = 0 and
- ERL = 0.

If these conditions are encountered, the IM bit Settings allow interrupts.

- **Operation mode: the following bit fields need to be set when the processor is in normal user, kernel, and superuser mode.**
  - The processor is in normal user mode when KSU=10, EXL=0, and ERL=0
  - The processor is in superuser mode when KSU=01, EXL=0, and ERL=0
  - The processor is in kernel mode when KSU=00, or EXL=1, or ERL=1
- **Kernel address space access: the kernel address space is accessible when the processor is in kernel mode.**
- **Superuser address space access: when the processor is in kernel or superuser mode, the superuser address space can be accessed.**
- **User address space access: the processor can access the user address space in all three modes of operation.**

Status register reset

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

3.13 IntCtl register (12,1)

The IntCtl register is a read-write 32-bit register. It manages the interruption of expansion in the Release2 system.Loongson 3 implements vector interrupts.The VS field of IntCtl register is used to represent the vector space between interrupt vectors. 1 field not writable, read 1; The 0 field is reserved and must be written to and returned to 0 on read. Of these, 31:26 represent clock interrupts and Performance Counter interrupts sharing HW5.

Figure 3-15 shows the format of the IntCtl register, and table 3-12 describes the correspondence between the VS field and the vector space.

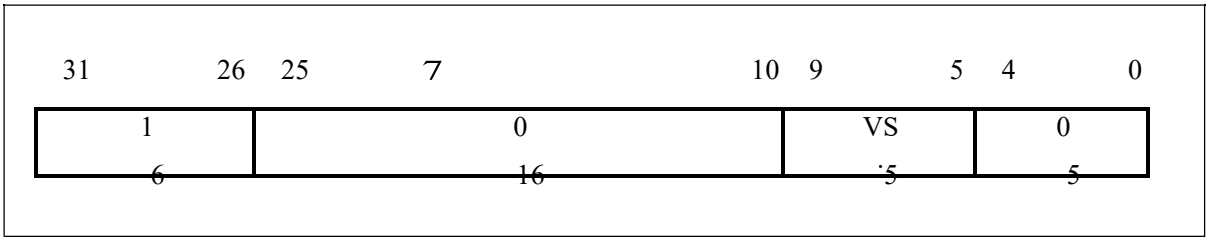


Figure 3-15 IntCtl register

Table 3-12 corresponding relationship between the encoding of VS field and the vector space

coding	Vector space (hexadecimal)	Vector space (base 10)
0 x00	0 x000	0
0 x01	0 x020	32
0 x02	0 x040	64
0 x04	0 x080	128

0 x08	0 x100	256
0 x10	0 x200	512

3.14 SRSCtl register (12,2)

The SRSCtl register is a 32-bit read-write register. It controls the group of shadow registers in the processor.Because loongson 3 has only one set of general purpose registers and no shadow registers, the shadow of the general purpose register is the general purpose register itself, and the SRSCtl register in loongson 3 only implements two domains: ESS and PSS.

Figure 3-16 shows the format of the SRSCtl register, and table 3-13 describes the domain of the SRSCtl register.

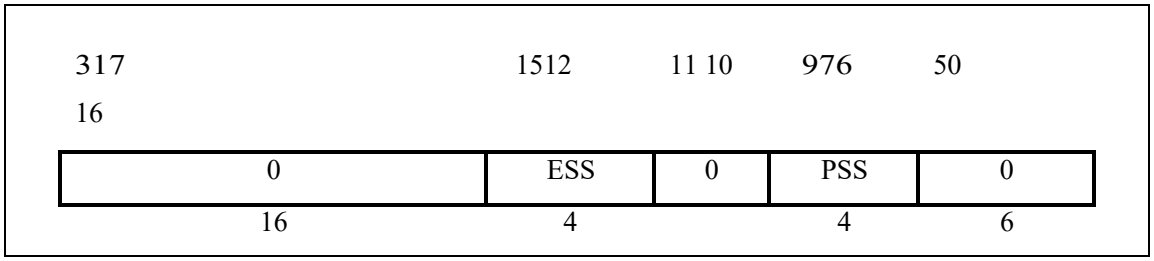


Figure 3-16 SRSCtl

register table 3-13 SRSCtl

The domain	describe
ESS	Shadow register group for exception. It can only be 0 in loongson 3
PSS	The previous shadow register group. It can only be 0 in loongson 3
0	Retained. You must press 0 to write and return 0 on read.

3.15 Cause register (13,0)

The 32-bit read-write Cause register describes the Cause of a recent exception.

Figure 3-17 shows the format of this register, and table 3-14 describes the field of the Cause register.A 5-bit ExcCode

One reason is indicated, as shown in table 3-15.

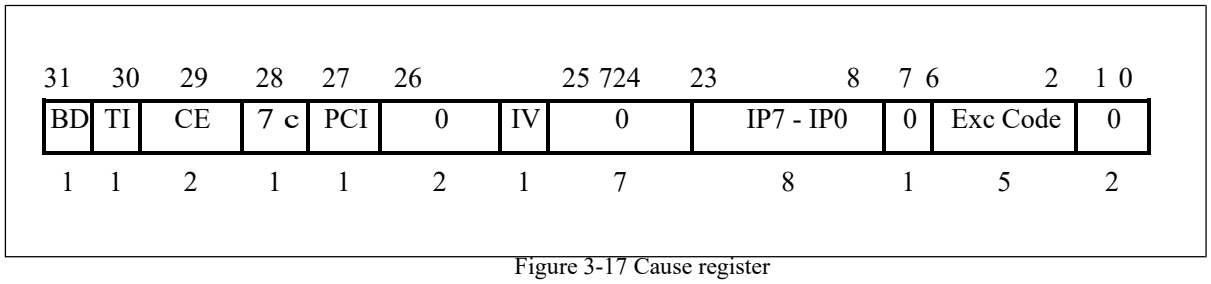


Figure 3-17 Cause register

Table 3-14 Cause register fields

The domain	desc ribe
BD	Indicates whether the last exception adopted is in the branch delay slot. 1 - delay slot 0 - normal

TI	Clock interrupt indicator 0- no clock interrupts 1- there is time to interrupt waiting for processing

CE	The unit number of the coprocessor when an exception occurs that the coprocessor is not available.
DC	Disable counting register The 0- counting register is available 1- the counting register is disabled
PCI	Performance counter interrupt indication 0- no performance counter interrupts 1- there is a performance counter interrupt waiting to be processed
IV	Interrupt exception entry 0- use the generic exception vector (0x180) 1- use special interrupt vector (0x200)
IP	Indicate the interruption of the wait. This bit will remain unchanged until the interrupt is removed. IP0~IP1 are soft interrupt bits that can be set and cleared by software. 1- interrupt waiting 0- no interruption
ExcCode	Exception code fields (see table 3-15)
0	Retained. You must press 0 to write and return 0 on read.

Table 3-15 ExcCode field of Cause register

Exception code	Mnemonic	desc ribe
0	Int	interrupt
1	The Mod	TLB modification is an exception
2	TLBL	TLB exception (read or fetch instructions)
3	TLBS	TLB exception (storage)
4	AdEL	Address error exception (read or fetch instruction)
5	AdES	Address error exception (storage)
6	IBE	Bus error exception (fetch instruction)
7	DBE	Bus error exception (data reference: read or store)
8	Sys	System call exception
9	Bp	Breakpoint exception
10	RI	Exception to reserved instruction
11	The CpU	An exception is not available for the coprocessor
12	Ov	Arithmetic overflow exception
13	The Tr	Trap exceptions
14	-	reserve
15	FPE	Floating-point exception
16	IS	Exception stack
17-18	-	reserve

19	DIB	Debug instruction exception

20	DDBS	Debug saves data exception
21	DDBL	Debug fetch data exception
22	-	reserve
23	WATCH	WATCH the exception
24 and 25	-	reserve
26	DBP	Debug breakpoint exception
27	DINT	Debug Debug exception
28	DSS	Debug single step exception
29	-	reserve
30	CACHERROR	Cache fault exception
31	-	reserve

3.16 Exception Program Counter register (14,0)

The Exception Program Counter (EPC for short) is a read/write register that contains the address of the continuation processing after the Exception processing is completed.

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

- Instruction virtual address, which is the direct cause of the exception, or
- The virtual address of a previous branch or jump instruction (when the instruction is in the branch delay slot, the instruction delay bit is set in the Cause register).

When the EXL bit in the Status register is set to 1, the processor does not write the EPC register.Figure 3-18 shows the format of the EPC register.

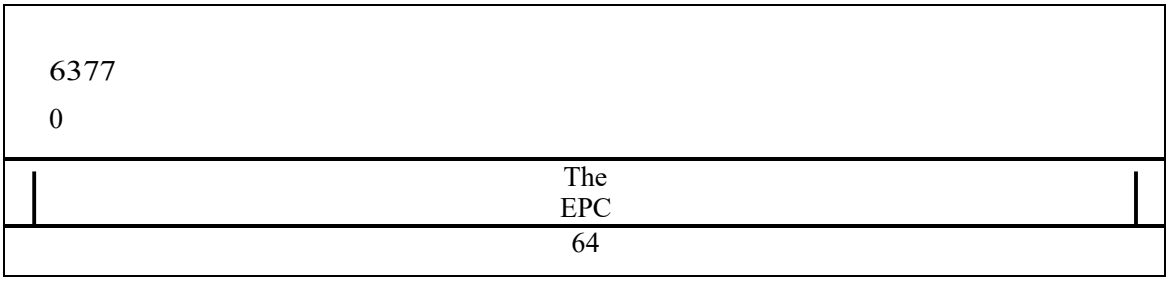


Figure 3-18 EPC register

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

The PRID register is a read-only register of 32 that contains information about the calibration processor and the implementation and revision versions of the CP0 version. Figure 3-19 shows the format of the register;Table 3-16 describes the domain of this register.

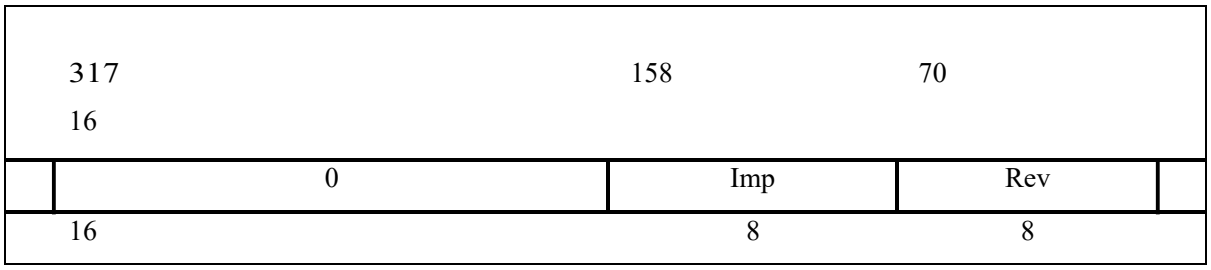


Figure 3-19 Processor Revision Identifier register

Table 3-16 PRId register fields

The domain	describe
IMP	Implementation version number
REV	Revision number
0	Retained. You must press 0 to write and return 0 on read.

The low (7:0 bit) of the PRID register can be used as the revision number, while the high (15:8) bit can be used as the implementation number. Loongson 3 implementation version number 0x63, revision version number 0x03.

The format of the version number is Y.X, where Y (7:4 bit) is the main version number and X (3:0 bit) is the small version number.The version number can distinguish the version of some processors, but there is no guarantee that any changes to the processor will be reflected in the PRID register,

In other words, there is no guarantee that changes to the version number must reflect changes to the processor. For this reason, the value of the register is not given, and the 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 that contains the exception vector base address and a read-only CPU number. When the BEV of the state register is equal to 0, the exception vector base address in the EBase register is used.

Figure 3-20 shows the format of the EBase register, and table 3-17 describes the field of the EBase register.

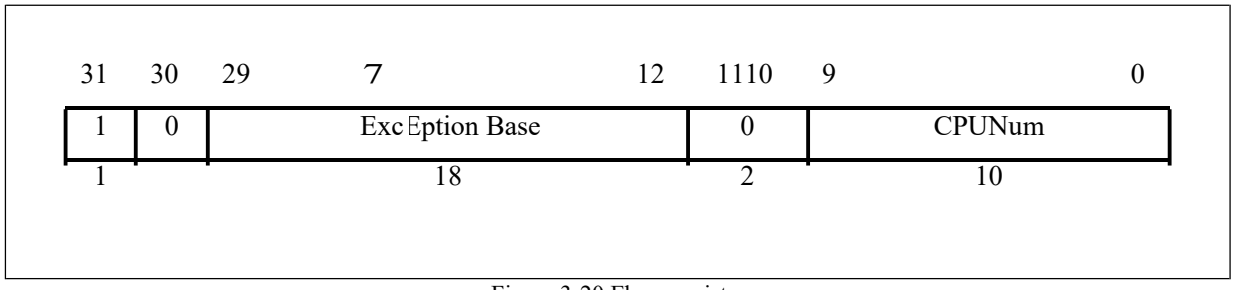


Figure 3-20 Ebase register

Table 3-17 Ebase register fields

The domain	describe
1	Can not write only read, read 1
0	Retained. You must press 0 to write and return 0 on read.
The Exception Base	Jointly specify the base address of the exception vector with 31 bits and 30 bits
CPUNum	In multicore systems, used to specify the processor number

3.19 Config register (16,0)

The Config register provides various configuration options for the loongson 3 processor. These options are listed in table 3-18.

Some of the configuration options defined by bit 31:3 in the Config register are set by the hardware at reset and are included in the Config register as read-only status bits for software access.The other configuration options (bit 2:0 in the Config register) are readable/written and controlled by the software.These fields are not defined at reset time.

The configuration of the Config register is limited. The Config register should be initialized by the software before the Cache is used, and the Cache should be reinitialized after any changes are made.

Figure 3-21 shows the format of the Config register.Table 3-18 describes the fields of the Config register. The initial value of the Config register is 0x00030932.

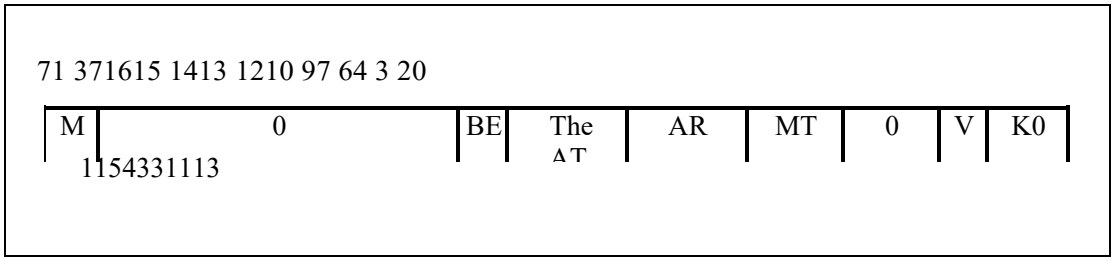


Figure 3-21 Config register

table 3-18 Config register

field

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
M	Whether the config1 register exists, setting 1 means it exists.
BE	Specify the end type 1 -- large tail 0 - small tail
The AT	Specify the architectural type 0 - MIPS32 1 -- MIPS64, which can only access the 32-bit address segment 2 -- MIPS64, which can access all address segments 3 - reserve

AR	Specify the version 0 - Release 1 1 - Release 2 2-7 - remain
MT	Specifies the memory management unit type 0 - no mapping 1 - standard TLB 2-7 - remain
VI	Indicates whether there is a virtual instruction Cache 0 - the instruction Cache is not virtual 1 - the instruction Cache is virtual
K0	Kseg0 consistency algorithm (Cache consistency algorithm) 2 - Uncached 3 - Cacheable Remaining - reserved

3.20 Config1 register (16,1)

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

The Config1 register is used as an additional part of the Config register. All the contents are read-only and are automatically set when reset. Figure 3-22 shows the format of the Config1 register.

Table 3-19 describes the field for the Config1 register. The initial value of the Config1 register is 0xfee37193.

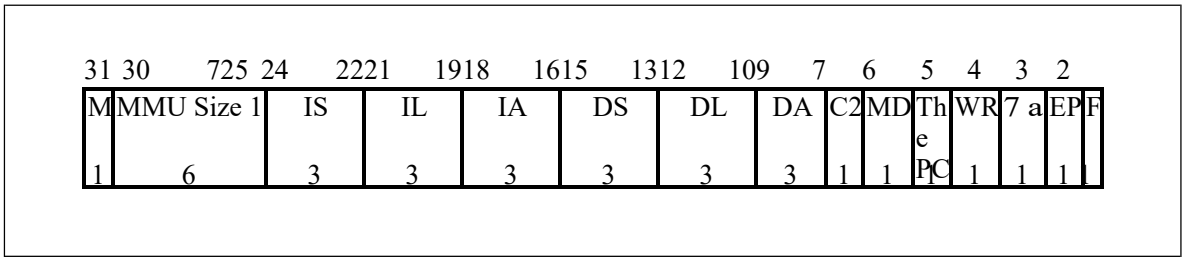


Figure 3-22: Config1 register

Table 3-19 Config1 register field

Domain description	
M	Whether the config2 register exists, setting 1 means it does.
MMU Size 1	The number of TLB table entries minus 1
IS	Icache number of groups per path coding: 0 meaning: 64 1 128 2 756 3 712

	4 1024 5 2048 6 4096 7 reserve
IL	Icache size per group coding meaning 0 No Icache 1 4 bytes
	2 8 bytes 3 16 bytes 4 32 bytes 5 64 bytes 6 128 bytes 7 reserve
IA	Icache connection method coding meaning 0 Directly connected 1 2 way connected 2 3 way connected 3 4-way linked 4 No.5 associative 5 No.6 associative 6 7 road linked 7 8 road linked
DS	Dcache number of groups per path Ed meaning 64 1 128 2 256 3 512 4 1024 5 2048 6 4096 7 reserve
	Dcache size per group coding meaning 0 No Dcache 1 4 bytes

<table>
<tr><td>DL</td><td><table>
<tr><td>2</td><td>8 bytes</td></tr>
<tr><td>3</td><td>16 bytes</td></tr>
<tr><td>4</td><td>32 bytes</td></tr>
<tr><td>5</td><td>647 bytes</td></tr>
<tr><td>6</td><td>128 bytes</td></tr>
<tr><td>7</td><td>reserve</td></tr>
</table></td></tr>
<tr><td>DA</td><td>Dcache connection mode<table>
<tr><td>codin g</td><td>meaning</td></tr>
<tr><td>0</td><td>Directly connected</td></tr>
<tr><td>1</td><td>2 way connected</td></tr>
<tr><td>2</td><td>3 way connected</td></tr>
<tr><td>3</td><td>4-way linked</td></tr>
<tr><td>4</td><td>No.5 associative</td></tr>
<tr><td>5</td><td>No.6 associative</td></tr>
<tr><td>6</td><td>7 road linked</td></tr>
<tr><td>7</td><td>8 road linked</td></tr>
</table></td></tr>
<tr><td>C2</td><td>Coprocessor no. 2 is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>MD</td><td>Whether or not MDMX ASE is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>The PC</td><td>Whether the performance count register is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>WR</td><td>Whether the Watch register is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>The CA</td><td>Whether or not MIPS16e is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>EP</td><td>Whether EJTAG is implemented
0 - unrealized
1 - to achieve</td></tr>
<tr><td>FP</td><td>Whether FPU is implemented
0 - unrealized
1 - to achieve</td></tr>
</table>

3.21 Config 2 register (16,2)

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

The Config2 register ACTS as, all the contents are read-only, and is automatically set when reset.

Figure 3-23 shows the format of the Config1 register.Table 3-20 describes the field for the Config2 register. The initial value of the Config2 register is 0x80001643.

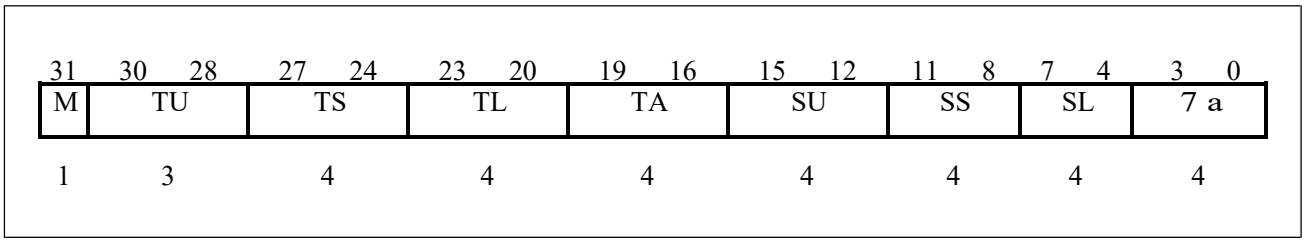


Figure 3-23. Config2 register

Table 3-20 Config2 register fields

<table>
<tr><th>Domain description</th><th></th></tr>
<tr><td>M</td><td>If there is a config3 register, setting 1 means there is one.</td></tr>
<tr><td>TU</td><td>Level 3 cache control or status bit</td></tr>
<tr><td>TS</td><td>The number of groups per path in a three-level cache
<table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>6</td></tr><tr><td>1</td><td>128</td></tr><tr><td>2</td><td>256</td></tr><tr><td>3</td><td>512</td></tr><tr><td>4</td><td>1024</td></tr><tr><td></td><td>2048</td></tr><tr><td>6</td><td>4096</td></tr><tr><td>7</td><td>8192</td></tr><tr><td>8-15</td><td>reserve</td></tr></table></td></tr>
<tr><td>TL</td><td>Three levels of cache size per group
<table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>No Icache</td></tr><tr><td>1</td><td>4 bytes</td></tr><tr><td>2</td><td>8 bytes</td></tr><tr><td>3</td><td>16 bytes</td></tr><tr><td>4</td><td>32 bytes</td></tr><tr><td>5</td><td>64 bytes</td></tr><tr><td>6</td><td>128 bytes</td></tr><tr><td>7</td><td>756 bytes</td></tr><tr><td>8-15</td><td>reserve</td></tr></table></td></tr>
<tr><td>TA</td><td>Three levels of cache connection
<table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>Directly connected</td></tr><tr><td>1</td><td>2 way connected</td></tr><tr><td>2</td><td>3 way connected</td></tr><tr><td>3</td><td>4-way linked</td></tr><tr><td>4</td><td>No.5 associative</td></tr><tr><td>5</td><td>No.6 associative</td></tr><tr><td>6</td><td>7 road linked</td></tr><tr><td>7</td><td>8 road linked</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>7 to 15</td><td>reserve</td></tr></table></td></tr>
<tr><td>SU</td><td>The second level cache control or status bit</td></tr>
<tr><td>SS</td><td>The number of groups per path in the secondary cache</td></tr>
<tr><td></td><td><table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>64</td></tr><tr><td>1</td><td>128</td></tr><tr><td>2</td><td>756</td></tr><tr><td>3</td><td>512</td></tr><tr><td>4</td><td>1024</td></tr><tr><td>5</td><td>2048</td></tr><tr><td>6</td><td>4096</td></tr><tr><td>7</td><td>8192</td></tr><tr><td>8-15</td><td>reserve</td></tr></table></td></tr>
<tr><td>SL</td><td>Level 2 cache size per group<table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>No Icache</td></tr><tr><td>1</td><td>4 bytes</td></tr><tr><td>2</td><td>8 bytes</td></tr><tr><td>3</td><td>16 bytes</td></tr><tr><td>4</td><td>32 bytes</td></tr><tr><td>5</td><td>64 bytes</td></tr><tr><td>6</td><td>128 bytes</td></tr><tr><td>7</td><td>256 7 ytes</td></tr><tr><td>8-15</td><td>reserve</td></tr></table></td></tr>
<tr><td>The SA</td><td>Level 2 cache connection mode<table><tr><td>coding</td><td>meaning</td></tr><tr><td>0</td><td>Directly connected</td></tr><tr><td>1</td><td>2 way connected</td></tr><tr><td>2</td><td>3 way connected</td></tr><tr><td>3</td><td>4-way linked</td></tr><tr><td>4</td><td>No.5 associative</td></tr><tr><td>5</td><td>No.6 associative</td></tr><tr><td>6</td><td>7 road phase</td></tr><tr><td></td><td>8 road linked</td></tr><tr><td>8-15</td><td>reserve</td></tr></table></td></tr>
</table>

3.22 Config 3 register (16,3)

The Config3 register marks whether some functions are implemented or not, and

everything is read-only, which is automatically set when reset.

Figure 3-24 shows the format of the Config1 register.Table 3-21 describes the field for the Config1 register. The initial value of the Config1 register is 0x000000a0.

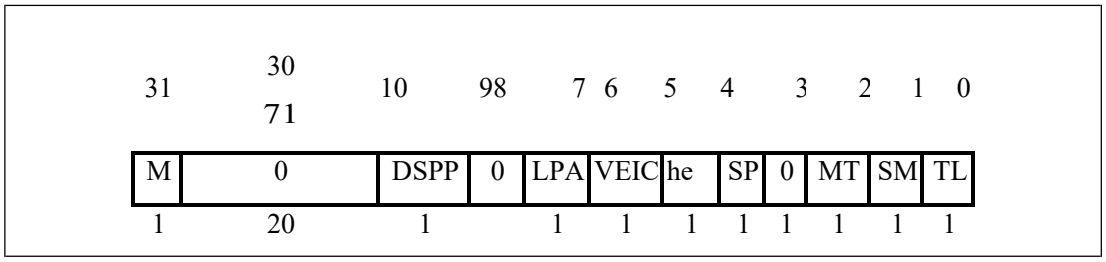


Figure 3-24. Config3 register

Table 3-21: Config3 register field

The domain	describe
M	reserve
0	Retained. You must press 0 to write and return 0 on read.
DSPP	Whether or not MIPS DSPASE is implemented 0 - unrealized 1 - to achieve
LPA	Whether the large physical address is implemented 0 - unrealized 1 - to achieve
VEIC	Whether the external interrupt controller is implemented 0 - unrealized 1 - to achieve
he	Whether the vector interrupt is implemented 0 - unrealized 1 - to achieve
SP	Small page support is implemented or not 0 - unrealized 1 - to achieve
MT	Whether MIPS MTASE is implemented 0 - unrealized 1 - to achieve
SM	Whether SmartMIPS ASE is implemented 0 - unrealized 1 - to achieve
TL	Trace Logic is implemented 0 - unrealized 1 - to achieve

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 address page number PFN of the recently occurred load-linked instruction, and when the exception returns (when the eret instruction occurs), the LLAddr register is cleared. The format of this register in loongson 3 is shown in figure 3-25.

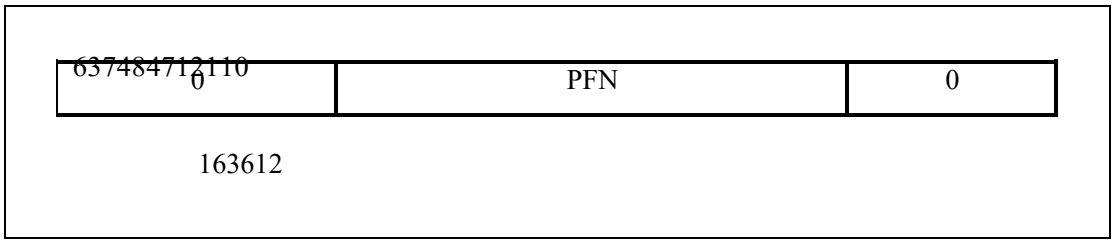


Figure 3-25 LLAddr register

3.24 XContext register (20,0)

The read-write XContext register contains a pointer to a table entry in the operating system page table.When a TLB exception occurs, the operating system loads the TLB from the page table based on the failed conversion.

The XContext register is used for XTLB rescaling processing, handling the TLB TAB loading of the 64-bit address space, and is used by the operating system only. The operating system sets the PTEBase field in the register as needed.

Figure 3-26 shows the format of the XContext register;Table 3-22 describes the domain of the XContext register.

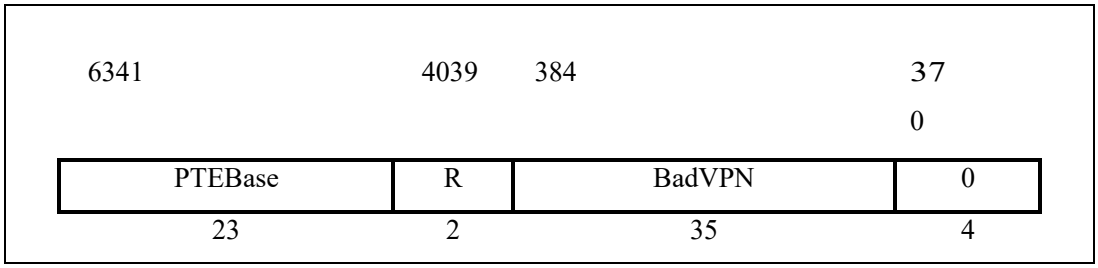


Figure 3-26 XContext register

The 35-bit BadVPN2 domain contains 47:13 bits of the virtual address that caused the TLB exception, and since a single TLB table entry maps to a parity page pair, bit 12 is not included. When the page size is 4K bytes, this format can directly address page tables whose PTE table entries are 8 bytes long and organized by pair.For other pages and PTE sizes, the correct address can be obtained by shifting and masking.

Table 3-22 XContext register fields

The domain	describe
BadVPN2	When a TLB exception occurs, the hardware will write to this field, which contains the virtual page number division for the most recent invalid virtual address 2.

R	The field contains 63:62 bits of the virtual address. 00 super user 01 common user

	11 □内核□□
0	Retained. You must press 0 to write and return 0 on read.
PTEBase	A readable/write field that allows the operating system to use the XContext register as a pointer to memory Pointer to the current page table.

3.25 Diagnostic register (22,0)

Loongson processor special 64 bit register, mainly used to control some internal queues and special operations of the processor.The format of the Diagnostic register is shown in figure 3-27, and the field of the Diagnostic register is described in table 3-23.

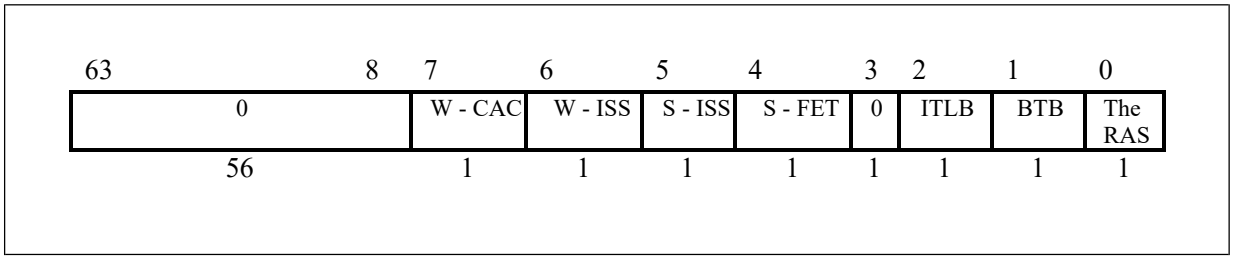


Figure 3-27 Diagnostic register

Table 3-23 Diagnostic register fields

The domain	describe
0	Retained.You must press 0 to write and return 0 on read.
W - CAC	Unrestrict the wait-cache operation
W - ISS	Remove the restriction on the 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	Empty BTB when writing 1
The RAS	RAS is not allowed when writing 1.

3.26 Debug register (23,0)

The Debug register is a 32-bit read-write register. The Debug register contains the reason for the Debug exception that occurred recently or in Debug mode, and it also controls single-step interrupts. This register indicates the debug resource and other internal states.Only the LSNM and SSt fields can be written, and only the DM bit and EJTAGver fields can be read when the debug register is read in non-debug mode.Loongson 3 does not implement the power-saving mode for debug exceptions.When reset, the initial value of the Debug register is: 0x02018000.

Figure 3-28 shows the format of the Debug register, and table 3-24 shows the field of the Debug register.

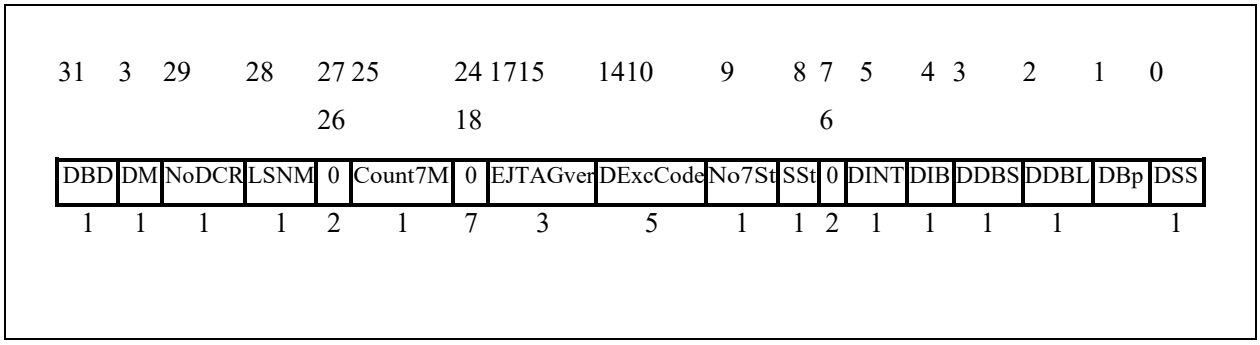


Figure 3-28 Debug register

Table 3-24 fields of the Debug register

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
DBD	Indicates whether the debug exception occurred in the delay slot. 0 - not 1 - is
DM	Indicates whether the processor is in Debug mode 0 -- non-debug Mode 1 -- Debug Mode
NoDCR	Indicates whether the dseg segment exists 1 -- doesn't exist 0 - there
LSNM	When the dseg segment is present, the available addresses of loads/stores are indicated 0 - dseg section 1 -- system memory
CountDM	The working state of the Count register when entering DM 0 - stopped 1 - runnig
EJTAGver	EJTAG version 0 - version 1 and 2.0 1 -- version 2.5 2 -- version 2.6 3 -- version 3.1 4, reserve
DexcCode	Indicates the reason for the last exception in Debug mode
NoSSt	Indicates whether single-step interrupts are supported 0 - support 1 - do not support
SSt	Single-step interrupt enable bit 0 -- not available

	1 - can make
DINT	Setting indicates that a Debug interrupt exception occurred Automatically clears when you enter Debug mode
DIB	Setting indicates that a Debug instruction interrupt exception has occurred Automatically clears when you enter Debug mode
DDBS	Setting indicates that a Debug data interrupt exception has occurred Automatically clears when you enter Debug mode
DBp	Setting indicates that a Debug breakpoint exception occurred Automatically clears when you enter Debug mode
DSS	Setting indicates that a Debug single-step interrupt exception occurs Automatically clears when you enter Debug mode

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

3.27 Debug Exception Program Counter register (24,0)

The Debug exception counter (DEPC) is a 64-bit read/write register that contains the address to continue processing after the exception is processed. This register is updated by the hardware with a debug exception or an exception in debug mode.

For the exact debug exception and the exact debug mode exception, the contents of the DEPC register are one of the following:

- Instruction virtual address, which is the direct cause of the exception, or
- The virtual address of a previous branch or jump instruction (when the instruction is in the branch delay slot, the instruction delay bit DBD is set in the Debug register).

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

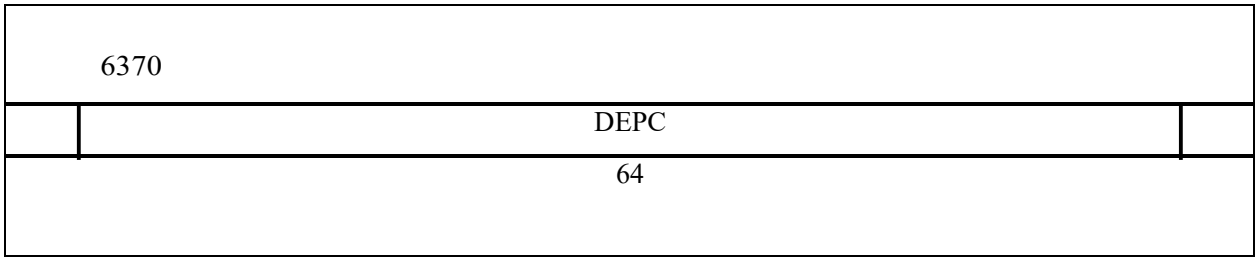


Figure 3-29 DEPC register

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

The loong chip 3 processor defines four (two groups) of performance counters that map to sel 0, sel 1, sel 2, and sel 3 in register CP0, respectively.

When reset, the initial values assigned to the two control registers of 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 of each register is shown in figure 3-30 (the two groups have the same format)

The enable bit definition of the register is shown in table 3-26:

Table 3-25 performance counters list

Performance counter	sel	Purpose to describe
0	The select 0	Control register 0
	Select 1	Counting register 0
1	Select 2	Control register 1
	Select 3	Counting register 1

Each counter is a 64-bit read/write register and increments itself every time a countable event occurs in the associated control domain.Each counter can count an event independently.

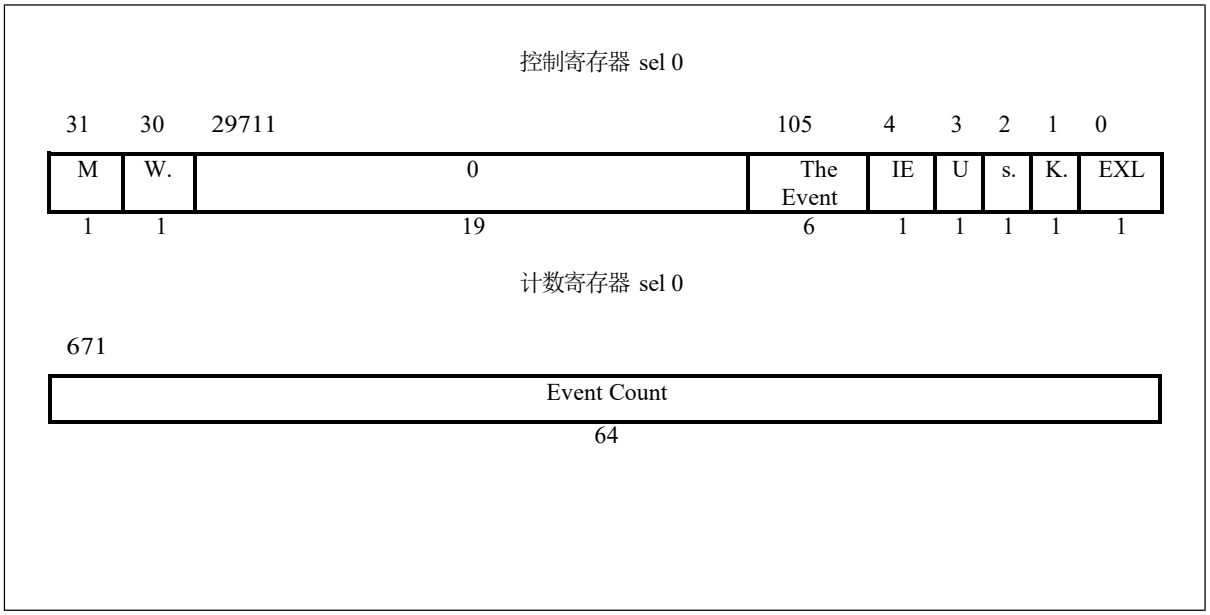


Figure 3-30 performance counter register

When the first digit (63 bits) of the counter becomes 1 (counter overflow), the counter will trigger an interrupt and Cause the PCI bit of the register to be set to 1 (if there are multiple counters, the overflow bit or of multiple counters).After the counter overruns, the count continues regardless of whether the interrupt is notified.Table 3-26 describes the definition of the count enabled bit.Table 3-27 and

Tables 3-28 describe the events for counters 0 and 1 respectively.

Table 3-26 count enabled bit definitions

Count the enabled bits	Count Qualifier(CP0 Status register field)
M	Whether there is another set of counters 1 - is 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 (superuser mode), EXL = 0, ERL = 0
U	KSU = 2 (normal user mode), EXL = 0, ERL = 0
EXL	E to the l is equal to 1, ERL is equal to 0

Table 3-27 counter 0 events

The event	signal	describe
0000	Cycles	cycle
0001	Brbus. Valid	Branch instruction
0010	Jrcount	JR instruction
0011	Jr31count	The JR directive and the field rs=31
0100	Imemread. Valid & imemread_allow	Level I-cache missing
0101	Rissuebus0. Valid	Alu1 operation has been launched
0110	Rissuebus2. Valid	Mem operation has been launched
0111	Rissuebus3. Valid	Operation Falu1 has been launched
1000	Brbus_bht	BHT guesses the instruction
1001	Mreadreq. Valid & Mreadreq_allow	Read from main memory
1010	Fxqfull	Fixed number of times the launch queue is full
1011	Roqfull	The number of times the queue is full
1100	Cp0qfull	The number of times the CP0 queue is full
1101	Exbus. Ex & excode = 34, 35	Tlb refills the exception
1110	Exbus. Ex & Excode = 0	exception
1111	Exbus. Ex & Excode = 63	Internal exception

Table 3-28 counter 1 events

The event	signal	describe
0000	Cmtbus? The valid	Commit operation

0001	Brbus. Brerr	Branch prediction failure
0010	Jrmiss	JR prediction failure
0011	Jr31miss	JR and rs=31 prediction failure
0100	Dmemread. Valid & Dmemread_allow	Level 1 d-cache missing
0101	Rissuebus1. Valid	Alu2 operation has been launched
0110	Rissuebus4. Valid	Operation Falu2 has been launched
0111	Duncache_valid & Duncache_allow	Access uncached
1000	Brbus_bhtmiss	BHT got the wrong guess
1001	Mwritereq. Valid & Mwritereq_allow	Written to main memory
1010	Ftqfull	The number of times a floating-point pointer queue is full
1011	Brqfull	The number of times a branch queue is full
1100	Exbus. Ex & Op = = OP_TLBPI	Lack of Itlb
1101	Exbus. Ex	The total number of exceptions
1110	Mispec	Loading speculative missing
1111	CP0fwd_valid	The CP0 queue is loaded forward

3.29 ECC register (26,0)

Loongson 3 USES the 26 ErrCtl register, which is optional in the MIPS64 standard, for ECC validation.Figure 3-31 shows the format of the DEPC register.The ECC register fields are described in tables 3-29.

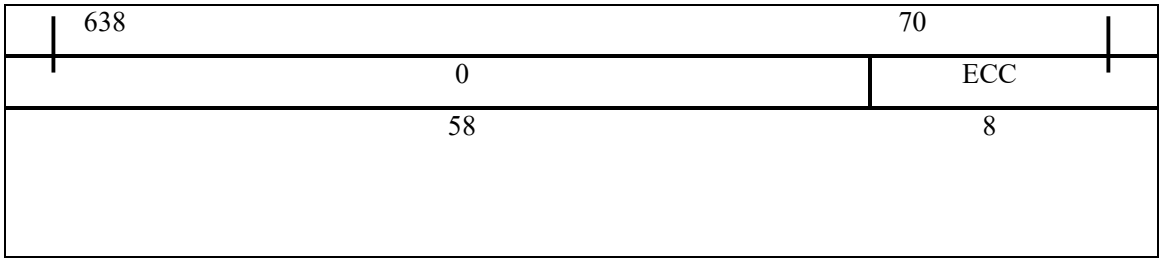


Figure 3-31 ECC register

Table 3-29 ECC register fields

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
ECC	A double-word check code associated with the Cache

3.30 CacheErr register (27, 0/1)

Loongson 3 completes the ECC verification of loongson 3 in the MIPS64 standard by hardware and software. The hardware is only responsible for checking the error. After

checking the data error, the content is saved in the control register such as CacheErr, and the concurrent exception is corrected by the software.

Figure 3-32 shows the format of the CacheErr register,

Table 3-30 describes the fields of the CacheErr register.Figure 3-33 shows the format of the CacheErr1 register, as described in table 3-31

The field of the CacheErr1 register.

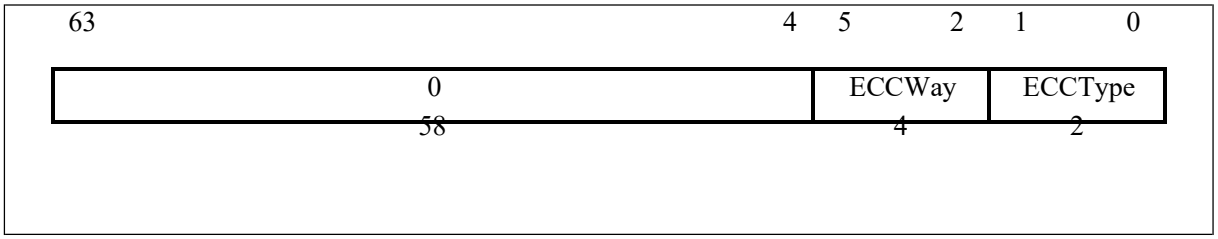


Figure 3-32. CacheErr register

Table 3-30 fields of the CacheErr register

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
ECCWay	Different encodings indicate different errors in the Cache if the Cache check is incorrect
ECCType	00- instruction cache error 01- data cache error 10- level 2 cache error 11- chip interface bus error

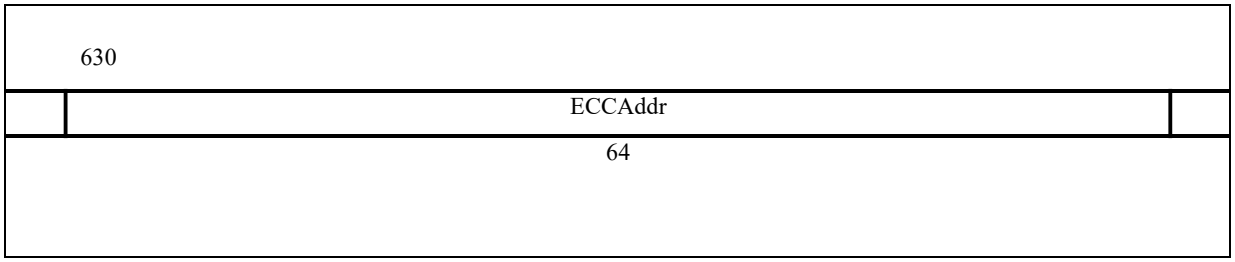


Figure 3-33 CacheErr1 register

Table 3-31 fields of the CacheErr1 register

The domain	describe
ECCAddr	The virtual address at which validation error occurred

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

The TagLo and TagHi registers are 32-bit read/write registers used to hold the labels and state of the level 1 / level 2 Cache The CACHE and MTC0 instructions are written to the Tag register.

Figure 3-34 shows these registers for level The format of the 1 Cache(P-Cache) operation.Table 3-32 lists TagLo and TagHi

Figure 3-34 TagLo and TagHi registers (p-cache)

Table 3-32 Cache Tag register field

The domain	describe
PTAG	Specifies the 39:12 bits of the physical address.
CS	Specifies the state of the Cache.
SCSETI	The group number corresponding to the Cache line in the second-level Cache (the second-level Cache field is 0)
0	Retained. You must press 0 to write and return 0 on read.

3.32 Registers DataLo (28,1) and DataHi (29,1)

DataLo and DataHi are read-only registers used only to interact with and diagnose cache data queues. The IndexLoadTag operation of the CACHE directive reads the corresponding data to the DataLo or DataHi registers. Figure 3-35 lists the formats of the DataLo register and the DataHi register, respectively.

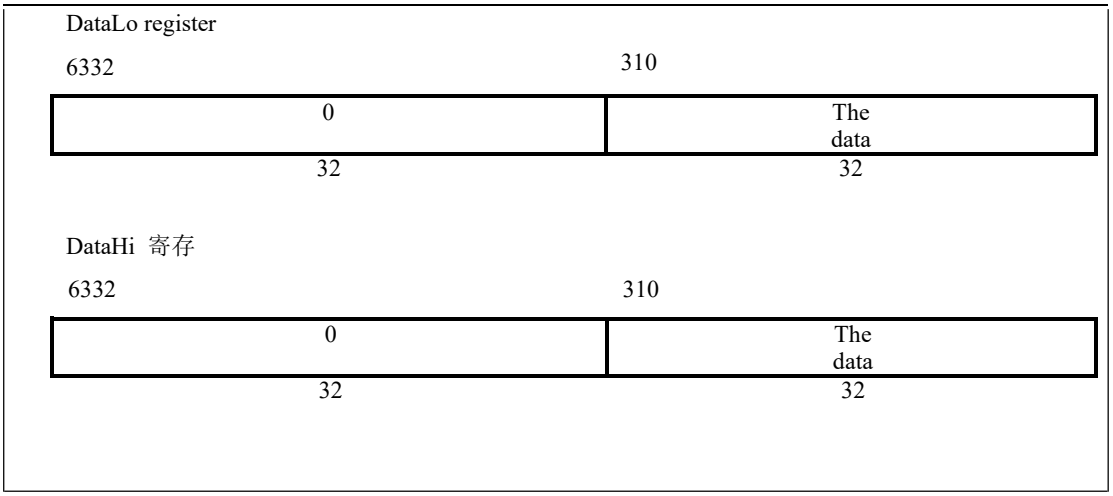


Figure 3-35 DataLo and DataHi registers

3.33 ErrorEPC register (30,0)

The ErrorEPC register is similar to the EPC register except for the exceptions for ECC and parity errors.It is used to store program counters during reset, software reset, and non-masking interrupt (NMI) exceptions.

The ErrorEPC is a read/write register that contains the virtual address where an instruction is restarted after processing an error. Figure 3-36 shows the format of the ErrorEPC register.

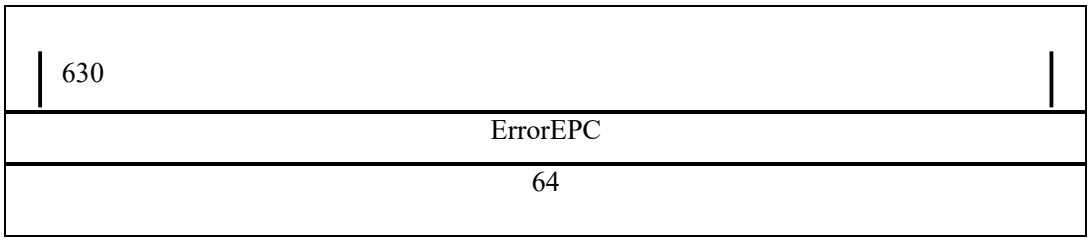


Figure 3-36 ErrorEPC register

3.34 DESAVE register (31,0)

The DESAVE register is a read-write 64-bit register. Its function is a simple register that is used to debug exception handling to save the value of a general-purpose register so that it retains other contexts.

Figure 3-37 shows the format of the DESAVE register.

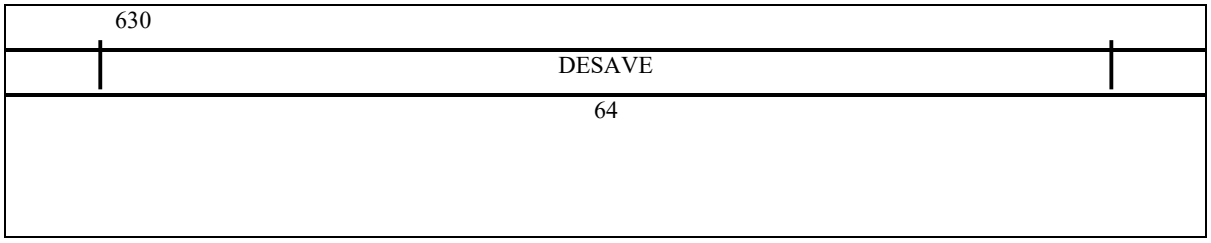


Figure 3-37 DESAVE register

3.35 CP0 instruction

The CP0 instructions defined by the godson-2 processor are listed in table 3-33.

Table 3-33 CP0 instructions

OpCode	The Description	MIPS ISA
DMFC0	Fetch a double word from register CP0	III.
DMTC0	Write a double word to register CP0	III.
MFC0	From the CP0 register	i.
MTC0	Write to register CP0	i.
TLBR	Press Index to read the TLB table entry	III.
TLBWI	Write the TLB table entry by Index	III.

TLBWR	Write the TLB table entry at Random	III.
TLBP	Find the matched index in TLB	III.
The CACHE	Cache operation	III.
ERET	Abnormal return	III.
DERET	The Debug return	EJTAG
DI	Close the interrupt	MIPS32 R2
EI	Open the interrupt	MIPS32 R2
RDHWR	Read hardware register	MIPS32 R2
RDPGPR	Read from the shadow register	MIPS32 R2
WRPGPR	Write to shadow register	MIPS32 R2
SDBBP	Software breakpoints	EJTAG

related

The loongson processor can handle pipeline correlation in hardware, including CP0 correlation and memory access correlation, so CP0 instructions are not required

NOP instruction to correct the instruction sequence.

4 Organization and operation of a CACHE

Loongson 3 USES three separate caches:

Level 1 instruction Cache: a total of 64KB, the Cache line size of 32 bytes, using a four-way group linked structure.

Level 1 data Cache: a total of 64KB, the Cache line size of 32 bytes, using the four-way group linked structure, using the write back strategy.Level 2 hybrid Cache: there are four level 2 Cache modules on the chip that communicate with the processor core via a 128-bit AXI bus.

Each second-level Cache module is globally addressable. The Cache line size is 32 bytes, with a capacity of 1M. The four-way group structure is adopted, and the write-back strategy is adopted.

4.1 Summary of the Cache

A fixed-point access to a level-1 Cache takes three clock cycles, and a floating-point access to a level-1 Cache takes four clock cycles.Each level Cache has its own data path, allowing simultaneous access to both caches.A level 1 Cache has 128 bit read, write, and rescaling paths.

The second-level Cache USES a 256-bit data path that is accessed only when the first-level Cache is invalidated.Level 2 Cache and level 1 Cache cannot be accessed at the same time. When level 1 Cache is disabled, access to level 2 Cache will cost at least 14 cycles. (in loongson 3, the core and level 2 Cache need to communicate via a cross-switch, so an additional 6 beats are required.)The second-level Cache backfills the first-level Cache with 128 bits of data per clock cycle.

The primary Cache USES a virtual address index and physical address flag, while the secondary Cache USES both physical addresses.The virtual address index may cause inconsistencies, which are currently guaranteed by the operating system of loongson 3.

Loongson 3 USES a directory-based Cache consistency protocol to ensure that writes to each processor core and IO are observed correctly by other cores and IO.A directory is maintained in the secondary Cache.For each level 2 Cache line, the directory USES a 32-bit bit vector to record whether each level 1 Cache (including the instruction and data Cache) has a backup of that Cache line.Loongson 3 USES hardware to maintain the consistency between the first-level instruction Cache, the first-level data Cache, the second-level Cache and the HT external devices. The software does not need to use the Cache instruction to maintain the consistency.

4.1.1 Non-blocking Cache

Loong chip 3 implements non-blocking Cache technology.A non-blocking Cache is one of several operations that follow by allowing the Cache to be invalidated

Cache invalidated or hit access operations continue to improve the overall performance of the system.

In a blocking Cache design, when a Cache failure occurs, the processor suspends subsequent accesses.At this point, the processor begins a storage cycle, fetches the requested data, fills it into its Cache, and then resumes execution.Depending on the design of the memory system, this operation will take up a large number of clock cycles.

In a non-blocking Cache design, however, the Cache is not paused on an invalidation.Loongson 3 supports multiple misses. It can support up to 24 Cache failures.

When a level 1 Cache is invalidated, the processor checks the level 2 Cache to see if the desired data is in it, and if the level 2 Cache is still invalidated, it needs to access the main memory.

The non-blocking Cache structure in loongson 3 makes more efficient use of loop unwinding and software flow.In order to maximize the advantages of a Cache, Load should be performed as early as possible before using an instruction to access data.

For I/O systems that require sequential access, the default setting for loongson 3 is blocked Uncached access.Loongson 3 provides the prefetch instruction, which can be used to prefetch the data to the first level by loading it into the zero fixed-point register

The Cache.In addition, the DSP engine in loongson 3 can prefetch the data in memory or IO into the second-level Cache.

4.1.2 Replacement strategy

Both level 1 and level 2 caches use random substitution algorithms.But the second level Cache provides a locking mechanism.By configuring the lock window register, you can ensure that up to four locked areas are not replaced out of the secondary Cache (see chapter 4 of the loggson 3A1000 processor user manual for details).

4.1.3 The parameters of the Cache

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

Table 4-1 Cache parameters

parameter	Instruction Cache	Data Cache	Level 2 Cache
The Cache size	64 KB	64 KB	1MB (total 4MB)
Associative degree	The 4 channels are connected	The 4 channels are connected	The 4 channels are connected
Replacement strategy	Random method	Random method	Random method (lockable)
Block size (line size)	32 bytes	32 bytes	32 bytes
The Index (Index)	Virtual address 13:5 bits	Virtual address 13:5 bits	Physical address 17:5 bits[1]
Mark (Tag)	Physical address 47:12 bits	Physical address 47:12 bits	Physical address 47:12 bits
Write policy	Do not write	Write back method	Write back method
Reading strategies	Non-blocking (2 simultaneous)	Non-blocking (24 simultaneous)	Non-blocking (8 simultaneous)
Read the order	Keyword first	Keyword first	Keyword first
Write the order	Do not write	sequential	
Checking devices	parity	ECC check	ECC check

4.2 First-order instruction Cache

The first-level instruction Cache size is 64KB, using the four-way group structure. The Cache block size (also known as the Cache line) is 32 bytes and can hold eight instructions. Because of the 128-bit read path adopted by loongson 3, four instructions can be sent to the superscalar scheduling unit per clock cycle.

The level 1 instruction Cache implements parity.When a parity error occurs in the read-

level Cache, the hardware automatically invalidates the corresponding Cache line and retrieves the correct value from the second-level Cache.The entire process requires no software intervention.

Which of the four secondary Cache modules a physical address falls into is determined by the routing configuration register of the crossover switch.

4.2.1 The organization of the instruction Cache

Figure 4-1 shows the structure of the first-level instruction Cache.The Cache is mapped in four-way groups, each containing 512 index entries.Select the corresponding Tag and Data according to the Index.After the Tag is read from the Cache, it is compared to the translated portion of the virtual address to determine which group contains the correct data.

When the first-level instruction Cache is indexed, each of the four groups returns its corresponding Cache line, which is 32 bytes in size,

The Cache line takes a 34-bit flag and a 1-bit significant bit.

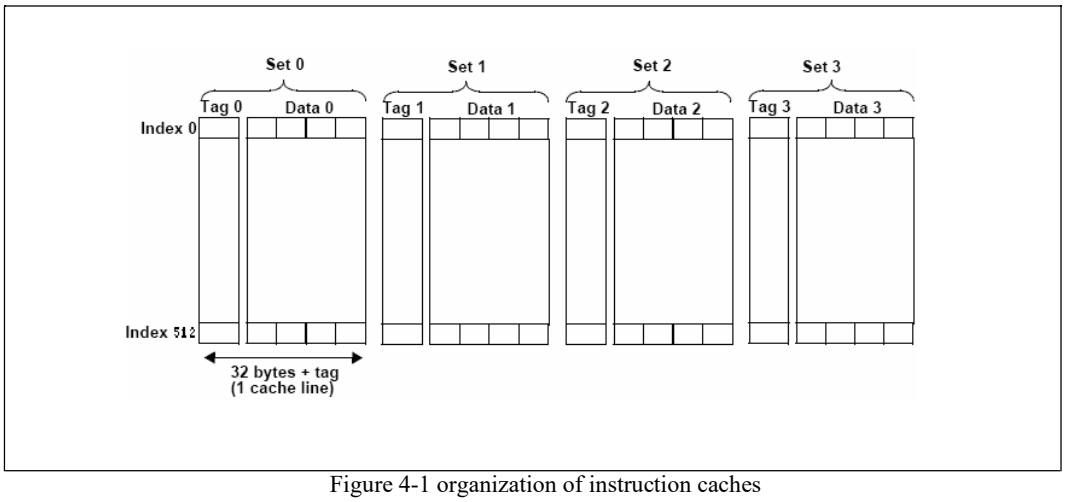


Figure 4-1 organization of instruction caches

4.2.2 Access to the instruction Cache

The loong chip 3 instruction Cache USES the virtual address index and the physical address flag four - way group associative structure.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 items.Each of these items contains four more 64 bits

Use the 4:3 bit to choose between the four words.

When indexing the Cache, the Data in the four blocks and the corresponding physical address Tag are taken from the Cache. Meanwhile, the high-order address is transformed by Instruction Translation look-aside Buffer (ITLB), and the transformed address is compared with the Tags in the four groups. If there is a Tag that matches, the Data in the group will be used.This is called a "level 1 Cache Hit".If none of the four groups of tags matches, the operation is aborted and the second level Cache is accessed.This is called a "level 1 Cache miss".

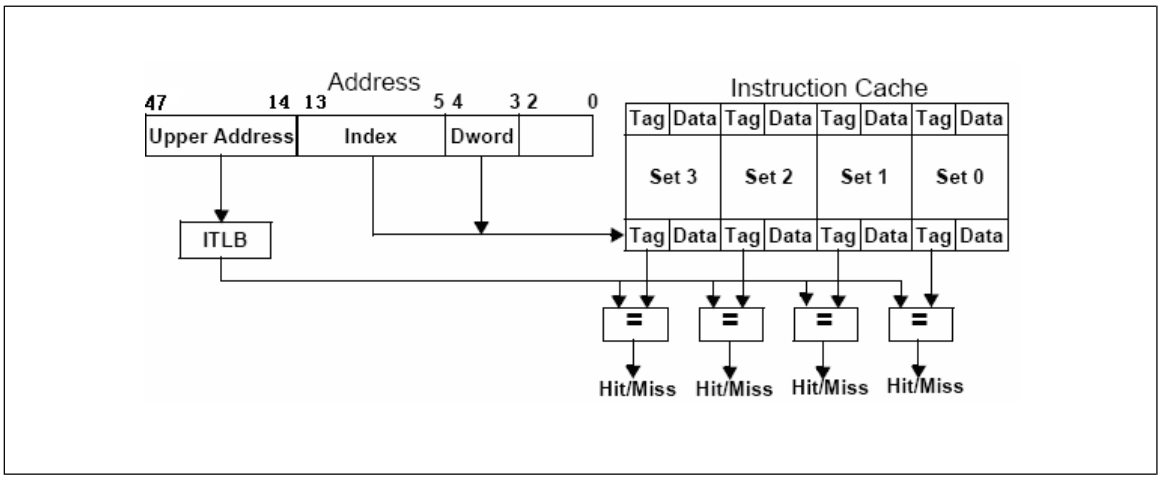


Figure 4-2 instruction Cache access

4.3 Level 1 data Cache

The data Cache has a capacity of 64KB and USES a four-channel structure. The Cache block size is 32 bytes, or 8 words. The read and write data paths in the data Cache are 128 bits.

The data Cache USES the virtual address index, the physical address flag.The operating system needs to resolve page coloring consistency issues that may be caused by virtual addresses.The data Cache is non-blocking, which means that a failure in the data Cache does not cause the pipeline to stop.The write strategy used in a data Cache is write back, that is, write data to a level 1 Cache without causing a level 2 Cache or main memory

The update. The write back policy improves global performance by reducing the traffic from the level 1 Cache to the level 2 Cache.Only in data Cache

When the row is replaced, the data is written to the second level Cache.

Level 1 data Cache implements ECC validation.When an ECC error occurs in the read-level data Cache, the hardware will automatically correct the result of the Cache read and update the contents of the Cache to the corrected value.The entire process requires no software intervention. When a two-bit ECC error occurs in the read level 1 data Cache, an exception will be made for the software to handle.

4.3.1 Organization of data Cache

Figure 4-3 shows the organization of a data Cache.This is a four-way Cache with 512 index entries.When accessing the Cache index, the tags and Data in all four groups are accessed at the same time.The Tag in the four groups is then compared with the transformed physical address portion to determine which data row is hit.

When indexing a data Cache, all four groups return their respective Cache rows. The Cache block size is 32 bytes, and the Cache line USES 34 bits for the physical flag address, 1 bit for the dirty bit, and 2 bits for the status bits (INV, SHD, and EXC). The INV status indicates that the Cache line is invalid, the SHD status indicates that the Cache line is readable, and the EXC status indicates that the Cache line is readable and writable.

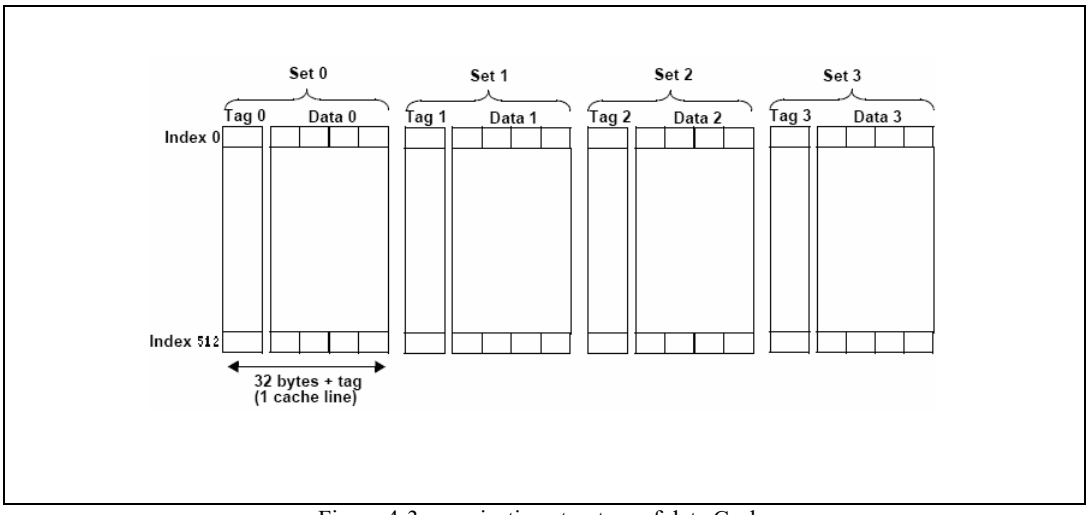


Figure 4-3 organization structure of data Cache

4.3.2 Access to the data Cache

The data Cache of loongson no. 3 adopts a four-way associative structure of virtual address index and physical address flag.Figure 4-4 shows how virtual addresses are decomposed when a data Cache is accessed.

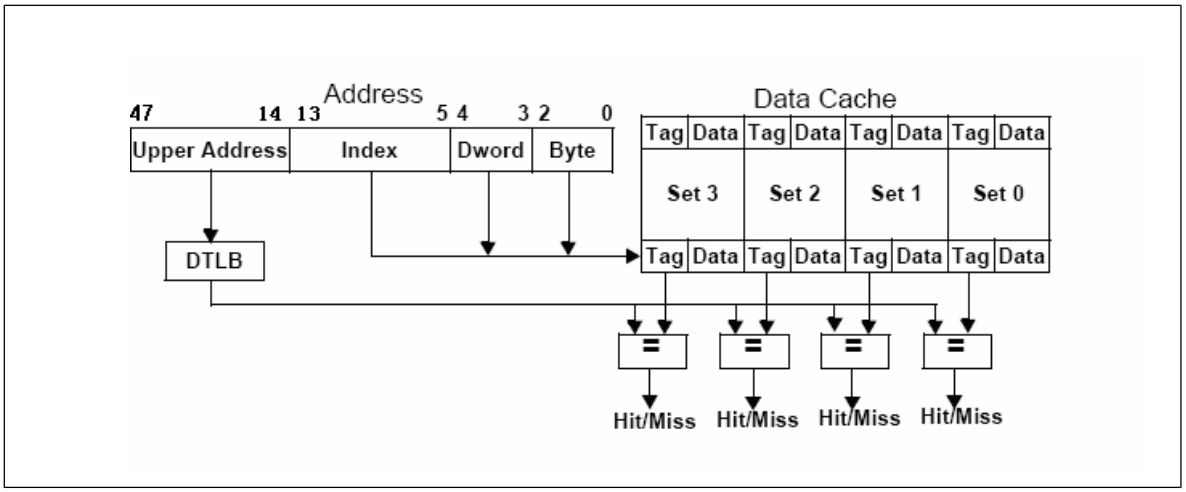


Figure 4-4 data Cache access

As shown in figure 4-4, the lower 14 bits of the address are used as an index to the data Cache.The 13:5 bits are used to index 512 items, among which

Each item also includes four 64-bit doublewords.Four doublewords are selected using 4:3 bits, and the 2:0 bit is used to select one of the eight bytes of a doubleword.

The data Cache accesses the failed instruction (the instruction does not hit the Cache line in SHD state or the instruction hits the Cache line in SHD state), then accesses the second level Cache.If the second level Cache hits, the Cache block retrieved from the second level Cache is sent back to the first level Cache.If the second level Cache fails, the memory is accessed and the second level Cache and data Cache are filled with the values retrieved from memory.

4.4 Level 2 Cache

Loongson 3 includes four on-chip level 2 Cache modules.Each level 2 Cache module has

a capacity of 1MB for a total of 4MB.Each Cache line is 32 bytes in size.The main features of the second-level Cache module include:Four-way group is connected to the 128 AXI interface, and eight Cache access queue, key priority, receives the request to return the data read failure eight fastest racquet, through directory support Cache coherence protocol, can be used for on-chip multi-core structure (also can be directly and single processor IP butt), the size of the soft IP grade level 2 Cache module can be configured (512 KB / 1 MB), adopts four-way group linked structure, runtime dynamically shut, support the ECC check, support DMA and prefetching read consistency, speaking, reading and writing.16 secondary Cache hashes are supported, and secondary caches are locked by window to ensure atomicity of read data return.

The second level Cache also maintains a directory for each Cache line to record whether or not a backup of that Cache line is included in each level Cache.The write strategy for the second level Cache is write back.The write back strategy reduces the bus traffic and improves the global performance of the system. Data is only written to memory if the second level Cache line is replaced.

The second level Cache implements ECC validation.When a single-digit ECC check error occurs in the read-level Cache, the hardware will automatically correct the result of the Cache read and update the contents of the Cache to the corrected value.The entire process requires no software intervention. When a two-bit ECC error occurs in the read secondary data Cache, an exception will be made for the software to handle.

4.4.1 Organization of level 2 caches

A second-level Cache is a mixed Cache that contains both instructions and data.The second level Cache module supports the Cache consistency protocol.In loongson 3, the second-level Cache is uniformly addressed on all chips, and each second-level Cache block has a fixed home node.According to Cache consistency requirements, the loongson 3's second-level Cache has two roles: home for the first-level Cache and Cache for memory.When the second level Cache is accessed, the Data and tags of four groups are accessed at the same time, and the four tags are fetched and the physical Data are accessed respectively

The higher part of the address is compared to determine whether the data still resides in the Cache.

Each Cache line contains a 32 bytes of data, 31 the physical address of the flag, one Cache status bits (representing the corresponding Cache in level 2 Cache is valid), 1 inventory status bits (indicates whether the corresponding Cache block in a level 1 Cache in the exclusive or Shared state) and 1 W (whether said the bank was written).

4.4.2 Access to the second level Cache

The second level Cache is accessed only if the first level Cache is disabled.The second level Cache USES the physical address index physical address flag.As shown in figure 4-5, low order addresses are used to index the second level Cache.All four groups will return their respective Cache rows. 16:5 bits are used as the index of the second level Cache. Each indexed entry contains four 64-bit binary data.Use 4:3 bits to choose between 4 double words. The 2:0 bit is used to select a certain 8 bytes in a double word.

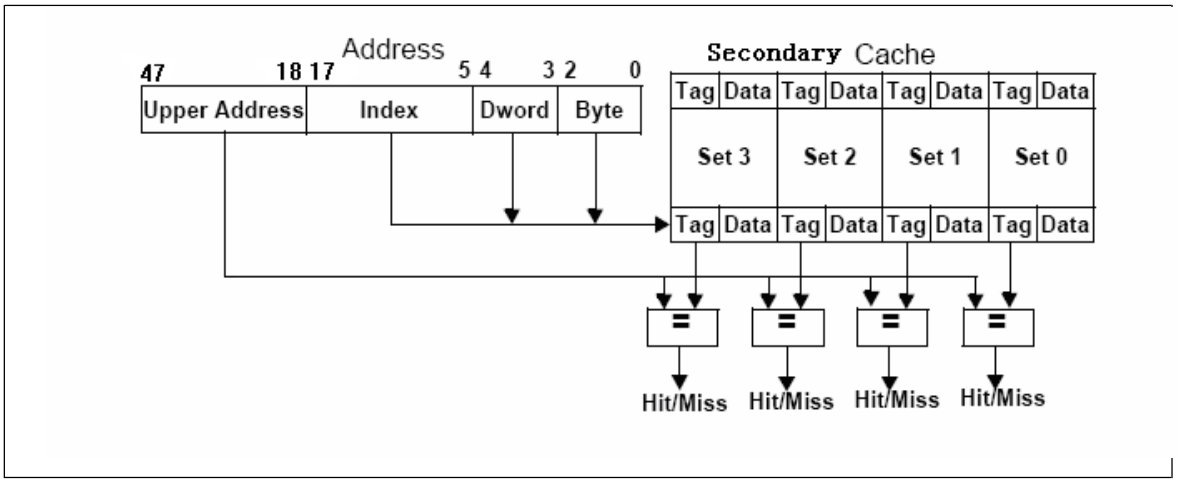


Figure 4-5. Level 2 Cache access

4.5 Cache algorithm and Cache consistency properties

Loongson 3 implements the Cache algorithm and Cache consistency properties shown in table 4-2.

Table 4-2 consistency properties of loongson 3 Cache

Attribute classification	Conformance code
reserve	0
reserve	1
Non-caching (Uncached)	2
Coherent cache (Cacheable coherent)	3

reserve	4
reserve	5
reserve	6
Uncached Accelerated (Uncached Accelerated)	7

4.5.1 Non-caching (Uncached, consistency code 2)

If a page USES a non-cache algorithm, the processor will directly emit a double-word, partial double-word, word, and partial word read or write request to main memory for any Load or Store operation anywhere on the page, without going through any level of Cache.The non-cache algorithm is implemented by blocking.

4.5.2 Coherent cache (Cacheable coherent code 3)

A row with this property can reside in the Cache, and the corresponding save and fetch operations only access the first-level Cache.When a level 1 Cache expires, the processor checks the level 2 Cache to see if it contains the requested address.If the second level Cache hits, the data is populated from the second level Cache.If the second level Cache misses, the data is pulled from main memory and written to the second level Cache and the first level Cache.

Because there are multiple processor cores and IO devices in loongson 3 that can access main memory, the loongson 3 hardware implements the Cache consistency protocol, so it is not necessary to actively maintain the Cache consistency by using the Cache instructions in the software.

4.5.3 Non-caching Accelerated (Uncached Accelerated, consistency code 7)

The non-caching speed up property is used to optimize the same type of Uncached memory operation performed in a sequential sequence in a continuous address space.The optimization method is to collect the memory operation of this property by setting the buffer. As long as the buffer is not satisfied, the data of these memory operations can be put into 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, start block writing. During the collection of sequential stored number instructions, if other types of Uncached stored number instructions are inserted, the collection is suspended and the data saved in the buffer is output in byte write mode.

The non-cache accelerated property speeds up sequential Uncached access, which is suitable for fast output access to the display device store.

4.6 The Cache consistency

Longchip 3 realizes the consistency of the Cache based on the directory. The hardware guarantees the consistency of the data between the first-level instruction Cache, the first-level data Cache, the second-level Cache, the memory and the IO devices from HT, without requiring the software to use the Cache instruction to force the Cache.Each Cache line in loongson 3 has a fixed host level 2 Cache module.The directory information in the Cache line is in host two

Maintained in the level Cache module.The directory USES a 32-bit bit vector to reserve track of the level 1 Cache (including the level 1 instruction Cache and the level 1 data Cache) that has a backup of each Cache line.Each level Cache block has three possible states: INV (invalid state), SHD (Shared state, readable), and EXC (exclusive state, readable and writable).The transitions of the three states are shown in figure 4-6.When the read instruction or fetch instruction fails, the processor core will issue a Reqread request to the second-level Cache module. After receiving the Repread reply from the second-level Cache module, the first-level Cache of the processor core will obtain a Cache backup of SHD status.When the first-level Cache fails, the processor core sends a Reqwrite request to the second-level Cache module. After receiving the Repwrite reply from the second-level Cache module, the first-level Cache of the processor core gets an EXC Cache backup.When a level 1 Cache replacement occurs in the processor core, it is written back to the level 2 Cache module through Reqreplace, and the level 2 Cache module informs the processor that the core replacement request has been processed through the Repreplace reply.The second-level Cache module can invalidate a level-1 Cache backup in SHD state by sending Reqinv request to the processor core. The processor core changes the level-1 Cache backup to INV state and replies to the level-2 Cache module through Repinv.The secondary Cache module can write back an EXC state of the primary Cache backup, the processor, by sending a Reqwtbk request to the processor core

The kernel changes the level 1 Cache backup to EXC state and replies to the level 2 Cache module via Repwtbk.The second-level Cache module can write back and invalidated a level-1 Cache backup in EXC state by sending a Reqinvwtbk request to the processor core, which puts the level-1 Cache backup in INV state and replies to the level-2 Cache module via Repinvwtbk.

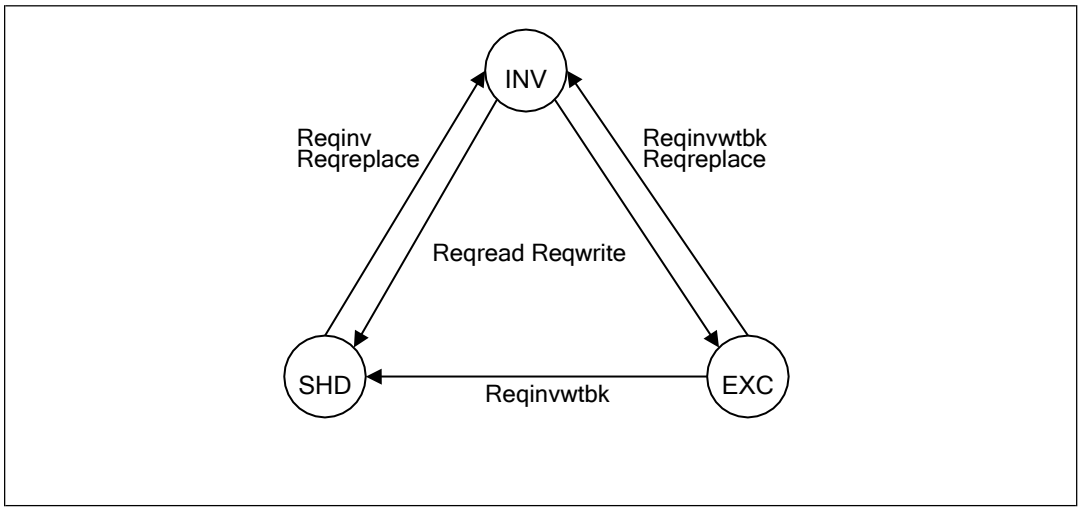


Figure 4-6. Loongson 3's cache state transition

5 Memory management

The logodson GS464 processor core provides a fully functional memory management unit (MMU) that converts virtual addresses to physical addresses using TLB on the chip.

This section describes the virtual and physical address Spaces of the processor core, the translation of virtual addresses to physical addresses, the operation of TLB in implementing these transformations, the Cache, and the system control coprocessor (CP0) register that provides the software interface for TLB.

5.1 Quick lookup of table TLB

Mapping virtual addresses to physical addresses is usually done by TLB (there are also virtual address translation without TLB, such as CKSEG0 and CKSEG1 kernel address space segments (see figure 5-5) without page mapping, where the physical address is obtained by subtracting a base address from the virtual address).. The first level of TLB is JTLB, which also ACTS as data TLB. In addition, the loongson GS464 processor core contains independent instruction TLB to ease the competition for JTLB.

5.1.1 JTLB

In order to be able to quickly map virtual addresses to physical addresses, the loongson GS464 processor core USES a larger, fully associative mapping mechanism called TLB, JTLB for instruction and data address mapping, indexed with their process Numbers and virtual addresses.

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

There are two mechanisms to help control the size of the mapped space and the substitution strategy for different regions of memory, respectively.

First, the page size can be from 4KB to 16MB, but must be quadrupled. The CP0 register PageMask is used to record the size of the mapped page, and this record is loaded into the TLB when a new table entry is written. The logodson GS464 processor core can support pages of different sizes at the same runtime, allowing the operating system to generate purpose-specific mappings: for example, the frame buffer in video codec processing can be memory-mapped using just one table entry.

Second, loongson GS464 processor core can use a random substitution strategy to select TLB table items to be replaced when TLB is missing.The operating system can also host a certain number of pages in TLB without being randomly replaced. This mechanism helps the

operating system to improve performance and avoid deadlocks.This mechanism also makes it easier for real-time systems to provide specific access to a key piece of software.

In addition, JTLB maintains the Cache consistency attribute for each page, and each page is marked with a specific bit: without Cache (Uncached), without coherent Cache (Cacheable Noncoherent), or without Cache Accelerated (Uncached Accelerated).

5.1.2 Instruction TLB

Loongson GS464 processor core instruction TLB (ITLB) has 16 table entries, which minimizes the capacity of JTLB and reduces the time-critical path when mapping through a large associative array, reducing power.Each ITLB table entry can map to only one page, and the page size is specified by the PageMask register. The mapping of ITLB instruction addresses and data addresses can be executed in parallel, thus improving performance. When the table item in ITLB is invalid, the corresponding table item is searched from JTLB, and an ITLB table item is randomly selected for replacement. The operation of ITLB is completely transparent to the user.The processor guarantees that ITLB is consistent with JTLB, and when using the nuclear mindset instruction to modify JTLB, ITLB

Will be automatically cleared.

5.1.3 Hit and miss

If the virtual address matches the virtual address of a table item in the TLB (that is, a TLB hit), the physical page number is taken from the TLB and joined with the offset to form the physical address.

If the virtual address does not match the virtual address of any table entry in the TLB (the TLB fails), the CPU generates an exception and the software refills the TLB based on the in-memory page table.The software can override either a specified TLB table entry or any TLB table entry using a hardware-provided mechanism.

5.1.4 A number of hits

The loongson GS464 processor does not provide any detection and disabling mechanisms when it checks the virtual address in a TLB to match the virtual address of more than one table item, unlike the design of earlier MIPS processors.Multiple hits do not physically break the TLB, so multiple hit detection mechanisms are unnecessary.However, the multiple hit situation is not defined, so the software should control not to allow multiple hits to occur.

5.2 Processor mode

The loongson GS464 processor core has three modes of operation, but unlike other MIPS processors, the loongson GS464 processor core supports only one address mode, one instruction set mode, and one tail mode.

5.2.1 Processor mode

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. The innermost kernel of the operating system runs in kernel mode.**
- **Management mode: the processor's priority is reduced and some less critical parts of the operating system run in this mode;**
- **User mode (lowest system priority) : this mode reserves different users from interfering with each other.**

Switching between the three modes is accomplished by the operating system (in kernel mode) setting the corresponding bit of the KSU domain in the status register.When an error (ERL position bit) or an exception (EXL position bit) occurs, the processor is forced to switch to kernel mode.Table 5-1 lists the Settings of KSU, EXL, and ERL when the three modes are switched. Empty table entries can be ignored.

Table 5-1 processor working mode

KSU 4:3	ERL 2	EXL 1	describe
10	0	0	User mode
01	0	0	Management mode
00	0	0	Kernel mode
	0	1	Exception level
	1		The error level

5.2.2 Mode of address

The core of the loongson GS464 processor only supports 64-bit virtual address mode, and the hardware guarantees compatibility with 32-bit address mode.

5.2.3 Instruction set mode

The core of loongson GS464 processor realizes the complete instruction set of MIPS64R2. In addition, some integer and floating point instructions are added. The added instructions are shown in appendix A and appendix B.

5.2.4 Tail model

The loongson GS464 processor core only works in small tail mode.

5.3 Address space

This section describes virtual address Spaces, physical address Spaces, and methods for virtual and real address translation through TLB.

5.3.1 Virtual address space

The loongson GS464 processor core has three virtual address Spaces: user address space, administrative address space, and kernel address space, each of which is 64-bit and contains discrete address space segments, the largest of which is 256T (2) bytes.48

These three address Spaces are described in sections 5.3.4 through 5.3.6.

5.3.2 Physical address space

By using 48-bit addresses, the processor has a physical address space size of 256T(2) bytes.48The following sections detail methods for virtual and real address translation.

5.3.3 Virtual and real address translation

When translating virtual and real addresses, the virtual address given by the processor is first compared with the virtual address stored in TLB.When the virtual page number (VPN) is equal to the VPN domain of a TLB table entry, and if either of the following is true:

- The Global bit of the TLB table entry is 1
- The ASID fields for both virtual addresses are the same.

TLB hits. If the above conditions are not met, the CPU will generate a TLB invalidation exception to enable the software to refold the TLB based on the in-memory page table.

If the TLB hits, the physical page number will be extracted from the TLB and combined with the Offset in the page to form the physical address.The Offset in the page does not go through TLB in the process of virtual and real address conversion.

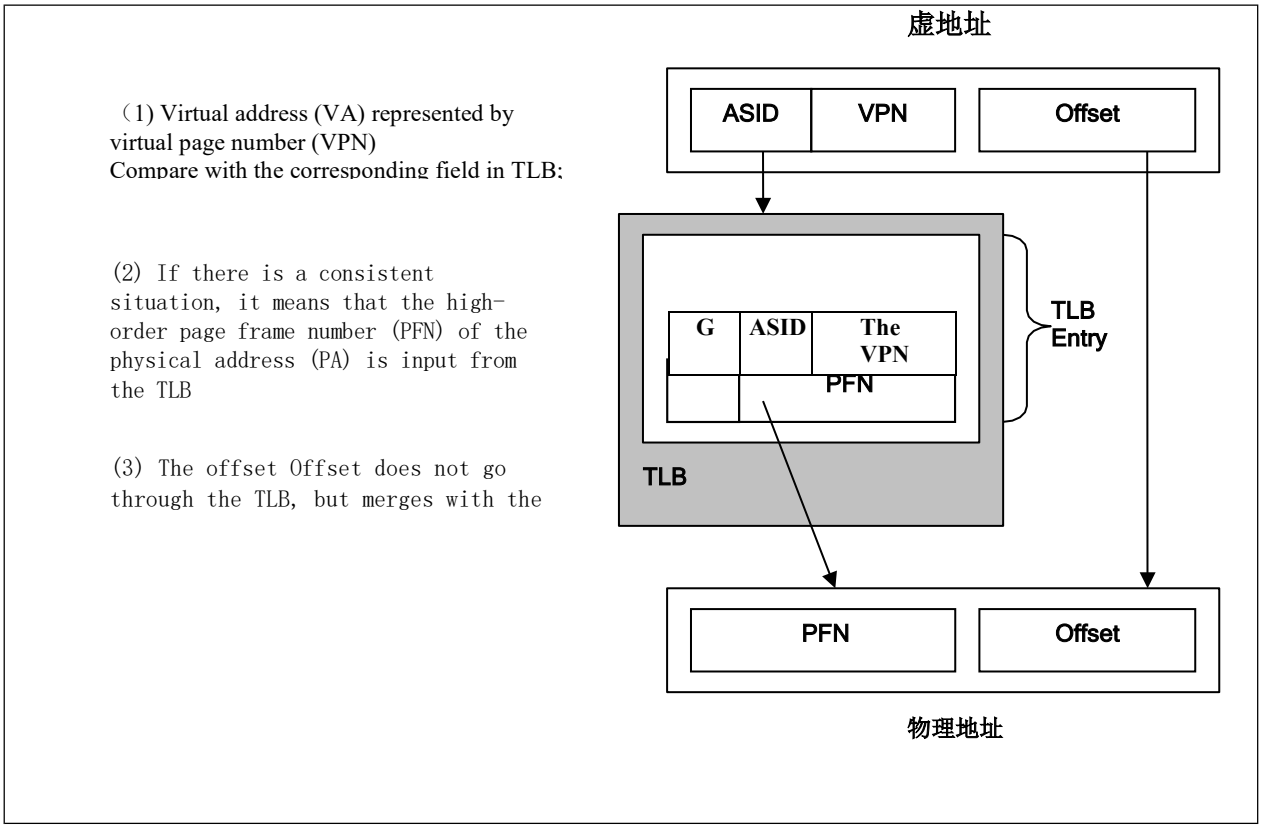


Figure 5-1 overview of virtual and real address translation

As shown in figure 5-1, virtual address translation is extended by an 8-bit address space identifier (ASID), which 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 table entry.

Figure 5-2 shows the real and virtual address translation in 64-bit mode, with a maximum page size of 16MB and a minimum page size of 4KB.

The top half of the figure shows the page size of 4K bytes. Offset in the page occupies 12 bits in the virtual address, and the remaining 36 bits in the virtual address are virtual page number VPN, which is used to index 64 4g page table items.

The bottom half of the figure shows the page size of 16M bytes, the Offset in the page occupies 24 bits in the virtual address, and the remaining 24 bits in the virtual address are virtual page number VPN, which is used to index 16M page table items.

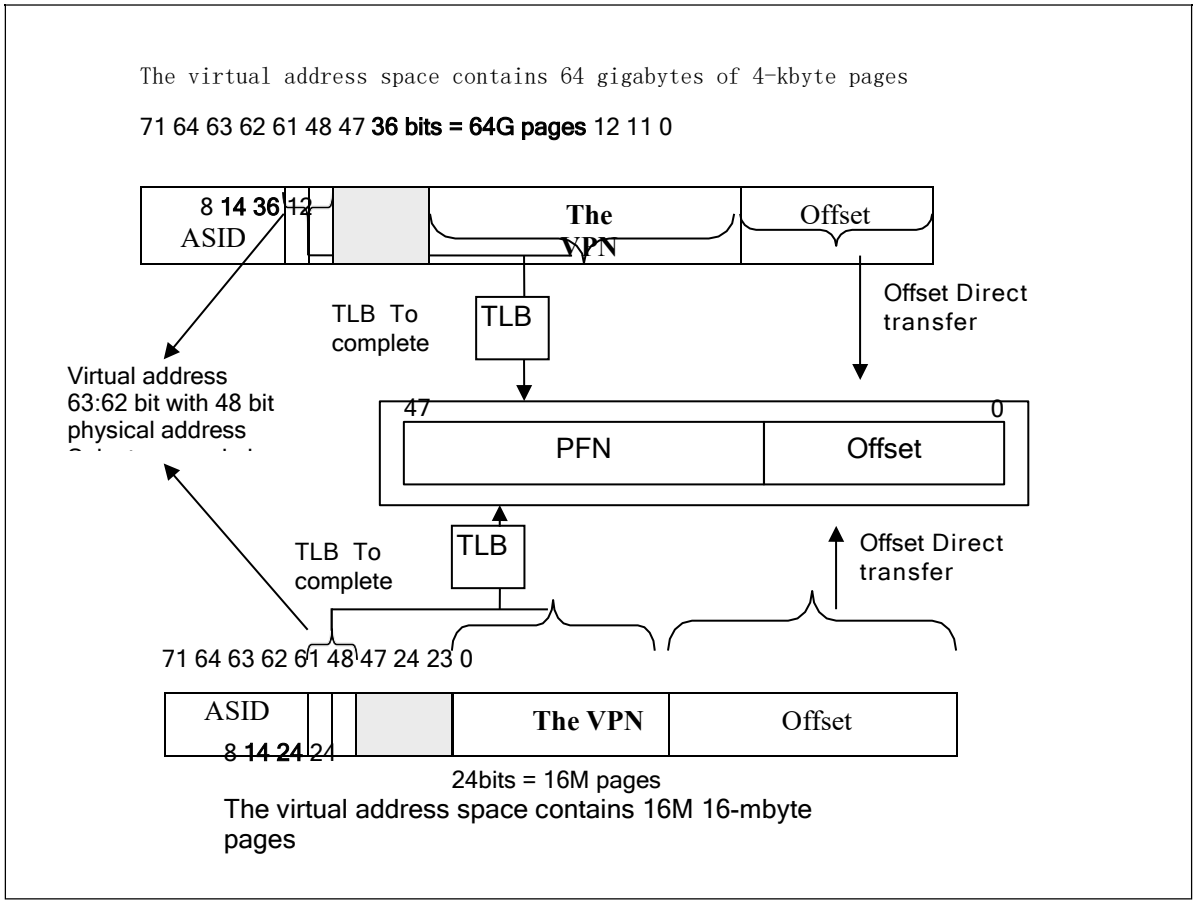


Figure 5-2. 64-bit mode virtual address translation

5.3.4 User address space

In User mode, there is only a single, unified virtual address space called User Segment with a size of 256T (2) bytes and the name XUSEG.48

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

The user segment starts at address 0, where the currently active user process resides (XUSEG).TLB maps the XUSEG segment in the same way in different modes and controls access to the Cache.

When the value of the processor's Status register meets three conditions: KSU=10, EXL=0, and ERL=0, the processor works in user mode.2

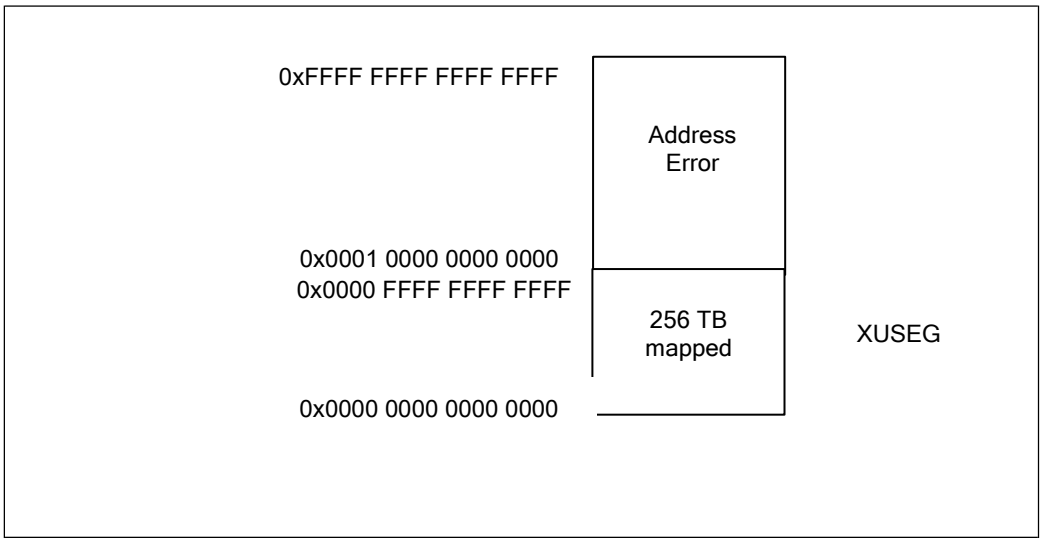


Figure 5-3 overview of user virtual address space in user mode

All available user mode virtual addresses from bit 63 to bit 48 must be 0, access to any address from bit 63 to bit 40 not all 0 will result in an address error exception, in the XUSEG address segment TLB missing using XTLB re fill vector.The XTLB refilling vector of the loongson GS464 processor core has the same exception entry address as the TLB refilling vector in 32-bit mode.

5.3.5 Manage address space

Management mode is designed for hierarchical operating systems. In a hierarchical operating system, the real kernel runs in kernel mode and the rest of the operating system runs in management mode.The managed address space provides code and data space to be accessed by programs in managed mode.The TLB miss that manages the address space is handled by the XTLB refill processor.

The administrative address space is accessible in both administrative and kernel mode.

When the value of the Status register of the processor meets three conditions at the same time: KSU=01, EXL=0, and ERL=0, the processor works in management mode.2Figure 5-4 shows an overview of the user and administrative address space in administrative mode.

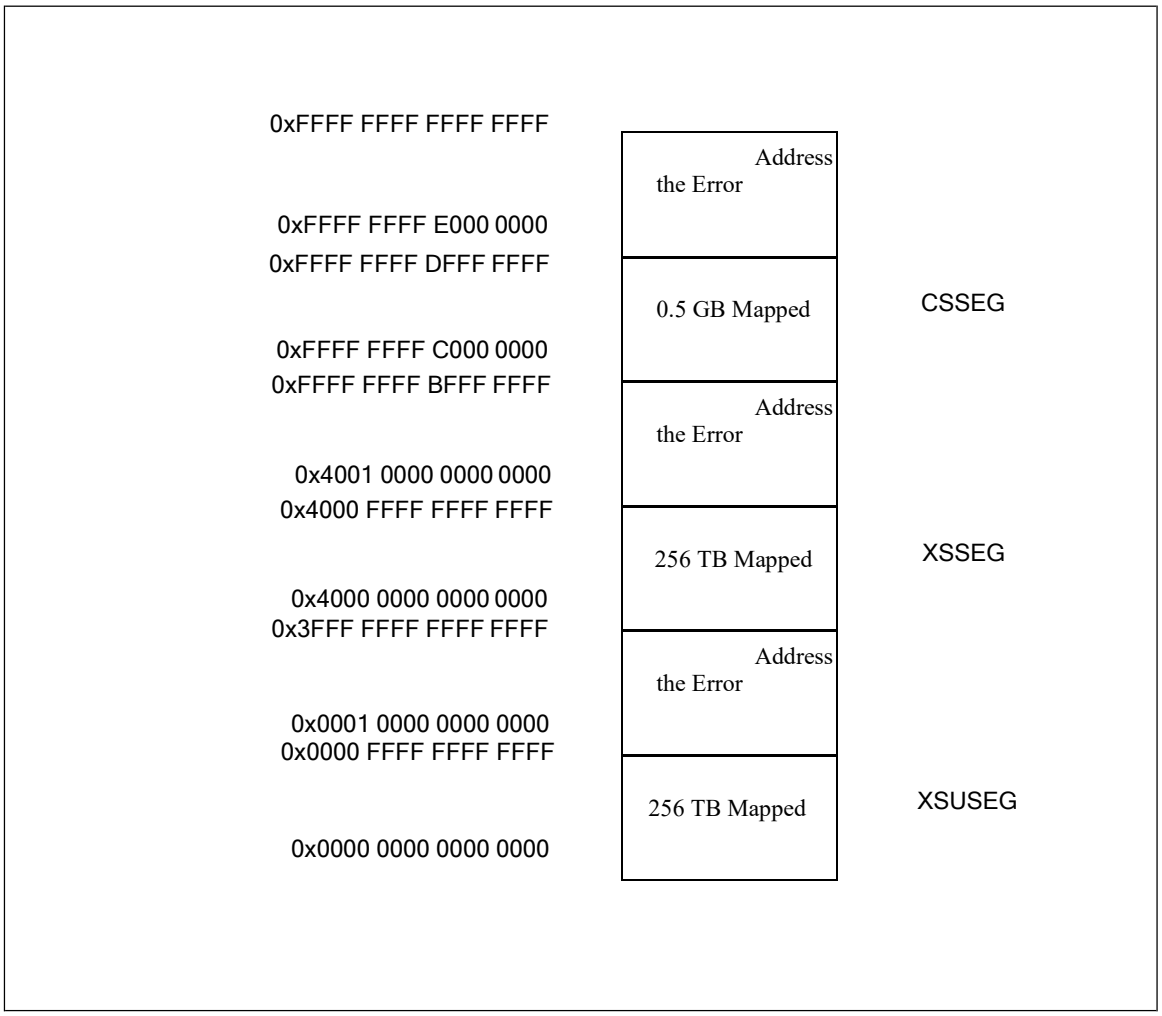


Figure 5-4 user space and management space in management mode

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

In administrative mode, when the user address space is accessed and the top two bits (63rd and 62nd bits) of the 64-bit address are 00, the program USES a virtual address space named XSUSEG, which covers all 2 (1T) bytes of the current user address space.248At this point, the virtual address is extended, plus the 8-bit ASID domain, to form a unique virtual address in the system.This address space starts at 0x0000 0000 0000 0000 and ends at 0x0000 FFFF FFFF FFFF.

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

In administrative mode, when the top two bits of the 64-bit address (63rd and 62nd bits) are 01, the program USES a currently managed virtual address space named XSSEG.2At this point, the virtual address is extended, plus the 8-bit ASID domain, to form a unique virtual address in the system.This address space starts at 0x4000 0000 0000 0000 and ends at 0x4000 FFFF FFFF FFFF.

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

In managed mode, when the top two bits of the 64-bit address (63rd and 62nd bits) are 11, the program manages the virtual address space independently with a name of CSSEG.2Addressing in CSSEG is compatible with addressing in SSEG in 32-bit mode.At this

point, the virtual address is extended, plus the 8-bit ASID domain, to form a unique virtual address in the system.This address space starts at 0xFFFF FFFF C000 0000 and ends at 0xFFFF FFFF DFFF FFFF.

5.3.6 Kernel address space

The processor works in kernel mode when the value of the processor's Status register meets the following conditions: KSU=00 or EXL=1 or ERL=1.2

Every time the processor detects an exception, it enters kernel mode and stays there until the exception return instruction (ERET) is executed.ERET

The instruction restores the processor to the mode in which the exception occurred.

The kernel-mode virtual address space is divided into different regions according to the high virtual address, as shown in figure 5-5.

- 64-bit kernel mode, user address space (XKUSEG)
  In kernel mode, when user space is accessed and the top two bits of the 64-bit virtual address are 00, the program USES a virtual address space named XKUSEG, which overrides the current user address space.2At this point, the virtual address is extended, plus the 8-bit ASID domain, to form a unique virtual address in the system.
- 64-bit kernel mode, current managed address space (XKSSEG)
  In kernel mode, when the administrative space is accessed and the top two bits of the 64-bit address are 01, the program USES a virtual address space named XKSSEG, which is the currently managed virtual address space.2At this point, the virtual address is extended, plus the 8-bit ASID domain, to form a unique virtual address in the system.
- 64-bit kernel mode, physical address space (XKPHY)
  In kernel mode, when the top two bits of a 64-bit address are 10, the program USES a virtual address space called XKPHY, which is a collection of eight two-byte kernel physical address Spaces.248Accessing any storage location whose address is not 0 in bits 58 through 48 will cause an address error.Access to XKPHY does not undergo address shuffling through TLB, but instead USES bits 47 through 0 of the virtual address as the physical address.Bits 61 through 59 of the virtual address control whether it passes through the Cache and the Cache consistency property, which has the same meaning as the c-bit values on the TLB page described in table 3-2.
- 64-bit kernel mode, kernel address space (XKSEG)
  In kernel mode, when the top two bits of a 64-bit address are 11, the program USES one of the following address Spaces:2

  - Kernel virtual address space XKSEG, at this point, the virtual address is expanded, plus the 8-bit ASID domain, forming a unique virtual address in the system;
  - Four 32-bit kernels are compatible with address Spaces, as described in the next section.
- 64-bit kernel mode, compatible with address space (ckseg1:0, CKSSEG, CKSEG3)
  In kernel mode, when the top two bits of 64-bit address are 11 and all bits of the virtual address are 1 from bit 61 to bit 31, the program USES one of the following four 512M byte address Spaces,

which is determined by bit 30 and bit 29:2

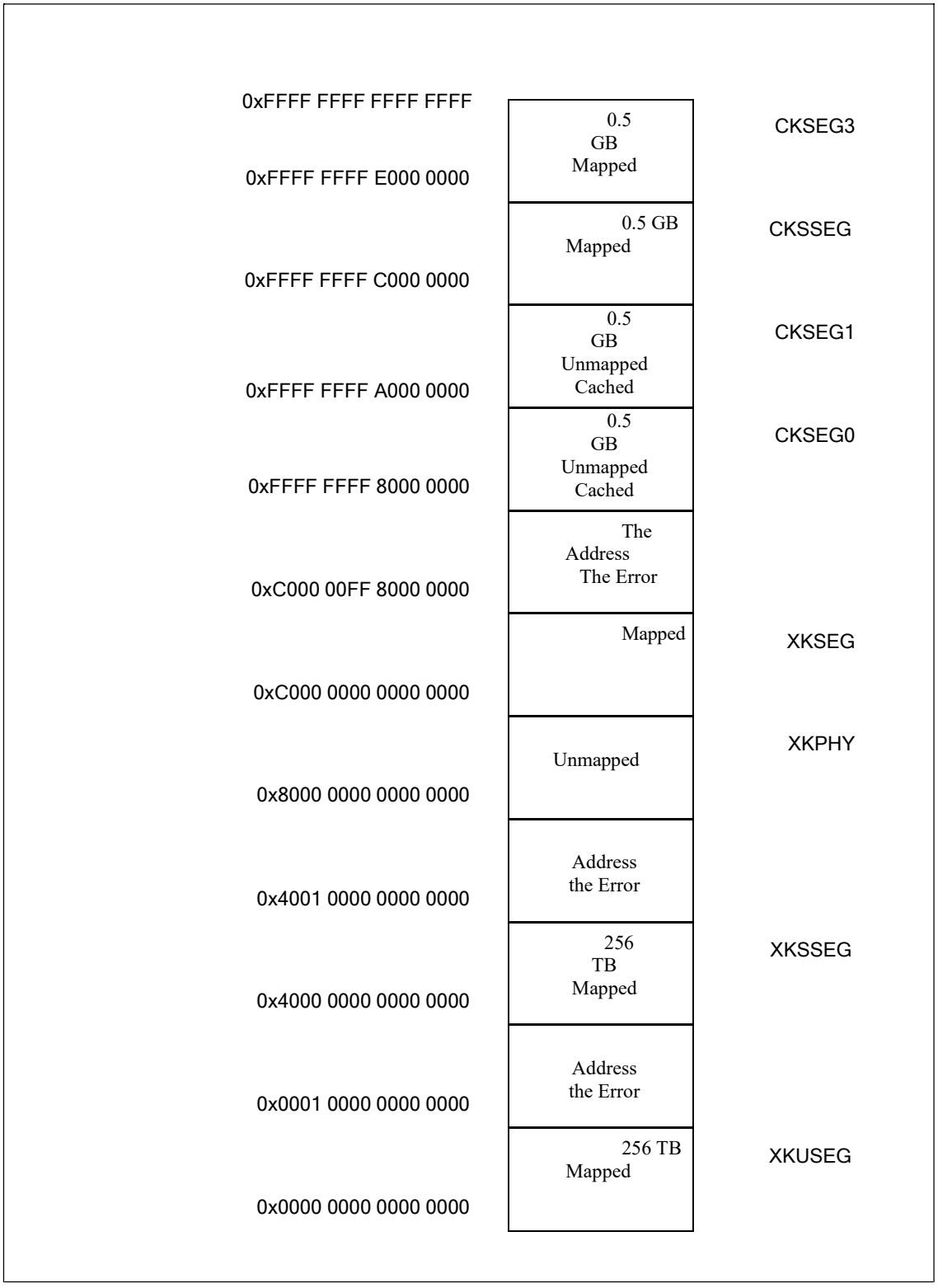


Figure 5-5 overview of user, management, and kernel address space in kernel mode

- CKSEG0: this 64-bit virtual address space does not pass through TLB and is compatible with KSEG0 in 32-bit mode. The K0 field of the Config register controls whether the Cache and the Cache consistency properties,
- CKSEG1: the 64-bit virtual address space does not pass through the TLB nor the Cache, with KSEG1 in 32-bit mode

Compatible.

- CKSSEG: this 64-bit virtual address space is currently managed for the virtual address space, with KSSEG in 32-bit mode

Compatible.

- CKSEG3: this 64-bit virtual address space is the kernel virtual address space and is compatible with KSEG3 in 32-bit mode.

5.4 System control coprocessor

The system control coprocessor (CP0) is responsible for supporting storage management, virtual and real address translation, exception handling, and some privileged operations.The loongson GS464 processor core has 26 registers CP0 and a 64-item TLB, each with a unique register number.The following sections give an overview of registers related to memory management.

5.4.1 Format of TLB table entries

Figure 5-6 shows the format of the TLB table entries, each of which has its own field in the EntryHi, EntryLo0, EntryLo1, PageMask registers.

EntryHi, EntryLo0, EntryLo1, and the PageMask register and TLB entries are formatted similarly.The only difference is that the TLB entry has a Global field (G bit), which is not in the EntryHi register, but appears as a reserved field.Figure 5-7, figure 5-8, and figure 5-9 represent the domains of the TLB term in figure 5-6, respectively.

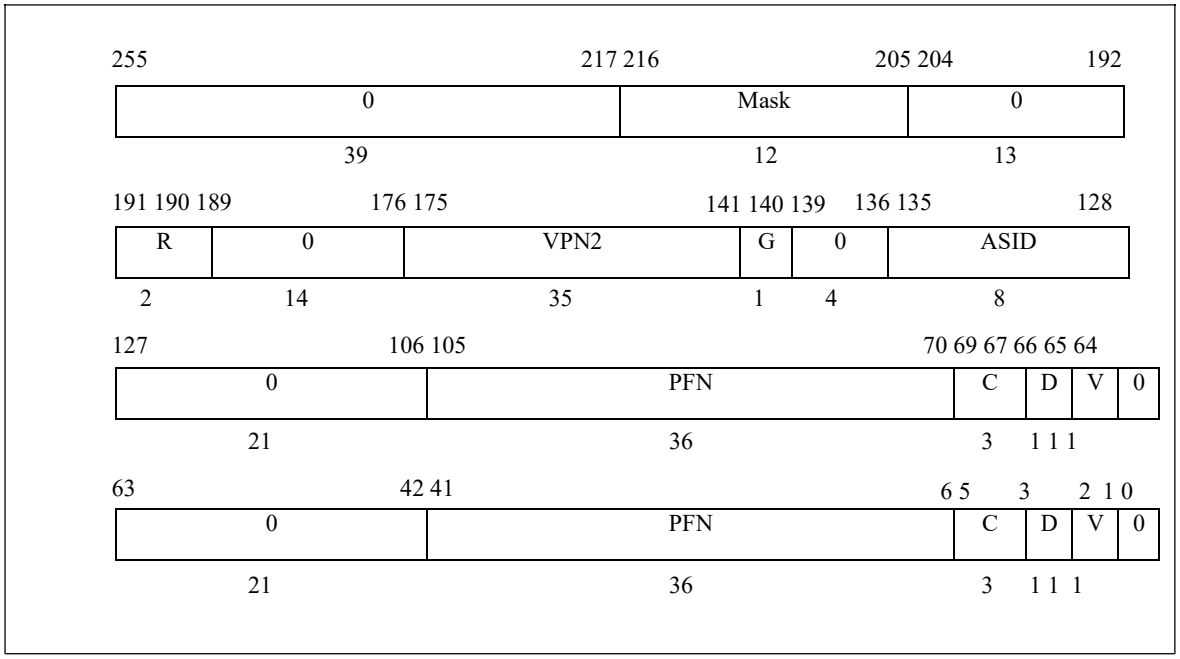
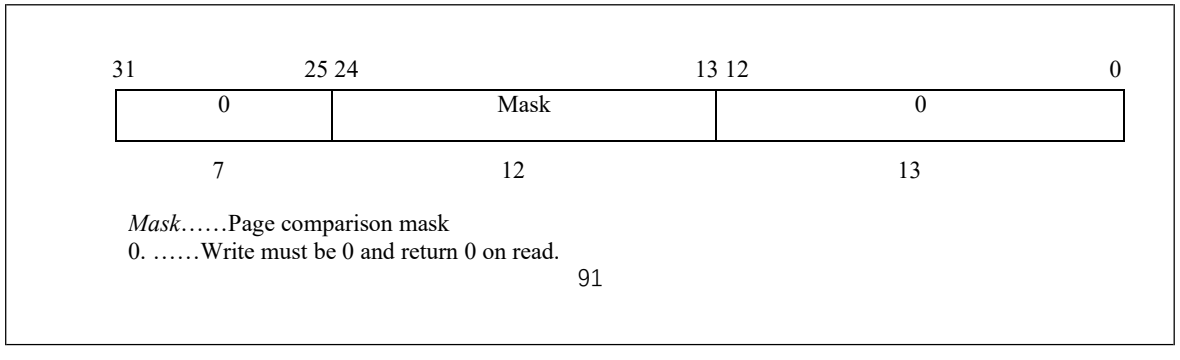


Figure 5-6 TLB table entries



63 62 61 48 47 13 12 8 7 0

R	The Fill	VPN2	0	ASID
2	14	35	5	8

Figure 5-7. PageMask register

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

ASID…. Address space ID field. An 8-bit field for multiple processes to share TLB; For the same virtual page number as other processes, each process has a different mapping.

R……Area. (00-> user, 01-> management, 11-> core) user matches virtual address 63:62 bits.

Fill……. Retained. Read 0, write ignore.

0……Retained. Write must be 0 and return 0 on read.

Figure 5-8 EntryHi register

63 42 41 June 5 3 2 1 0

0	PFN	C	D	V	G
22	36	3	1	1	1

63 42 41 June 5 3 2 1 0

0	PFN	C	D	V	G
2 2	36	3	1	1	1

PFN…Page frame; The high value of a physical address.

C……. Specifies the consistency properties for the TLB page; See table 3-2.

D……. Dirty bits. If the bit value is set to 1, the corresponding page is marked dirty and therefore writable. This bit is actually a write protection bit that the software can use to protect changes to the data.

V……. Significant bit. This bit setting indicates that the TLB table entry is valid; Otherwise, TLBL/TLBS will fail.

G…….global bit. If the corresponding bits in Lo0 and Lo1 are set to 1, the ASID is ignored by the processor during TLB lookup.

0 ..... Retained. Write must be 0 and return 0 on read.

Figure 5-9 EntryLo0 and EntryLo1 registers

TLB page consistency attribute (C) specifies whether the page needs to be accessed through the Cache, and if so, the consistency attribute of the Cache needs to be selected.Table 5-2 shows the Cache consistency attribute for the C bit.

Table 5-2 c-bit values for TLB pages

C (o) values	Cache consistency attribute
0	reserve
1	reserve
2	Non-caching (Uncached)
3	Coherent cache (Cacheable Noncoherent)
4	reserve
5	reserve
6	reserve
7	Uncached Accelerated (Uncached Accelerated)

5.4.2 CP0 register

The CP0 register related to memory management is listed in table 5-3. The CP0 register is fully described in chapter 1.

Table 5-3 memory management-related registers CP0

The register no.	Register names
0	The Index
1	The Random
2	EntryLo0
3	EntryLo1
5	PageMask
6	Wired
10	EntryHi
15	PRID
16	The Config
17	LLAddr
28	TagLo
29	TagHi

5.4.3 The process of converting a virtual address to a physical address

During the virtual address to physical address conversion, the CPU compares the 8-bit ASID of the virtual address (if global bit G is not set) with the ASID of the TLB entry to see if it matches.When comparing asids, we also need to match the virtual address's height of 15-27 bits with the TLB item's virtual page number according to the value of PageMask.If a TLB entry is matched, the physical address and access control bits (C, D, and V) are extracted from the matching TLB entry.For a valid address translation, the V bit of the matching TLB entry must be set, but the V bit value is not considered in the match comparison.Figure 5-10 shows the TLB address translation process.

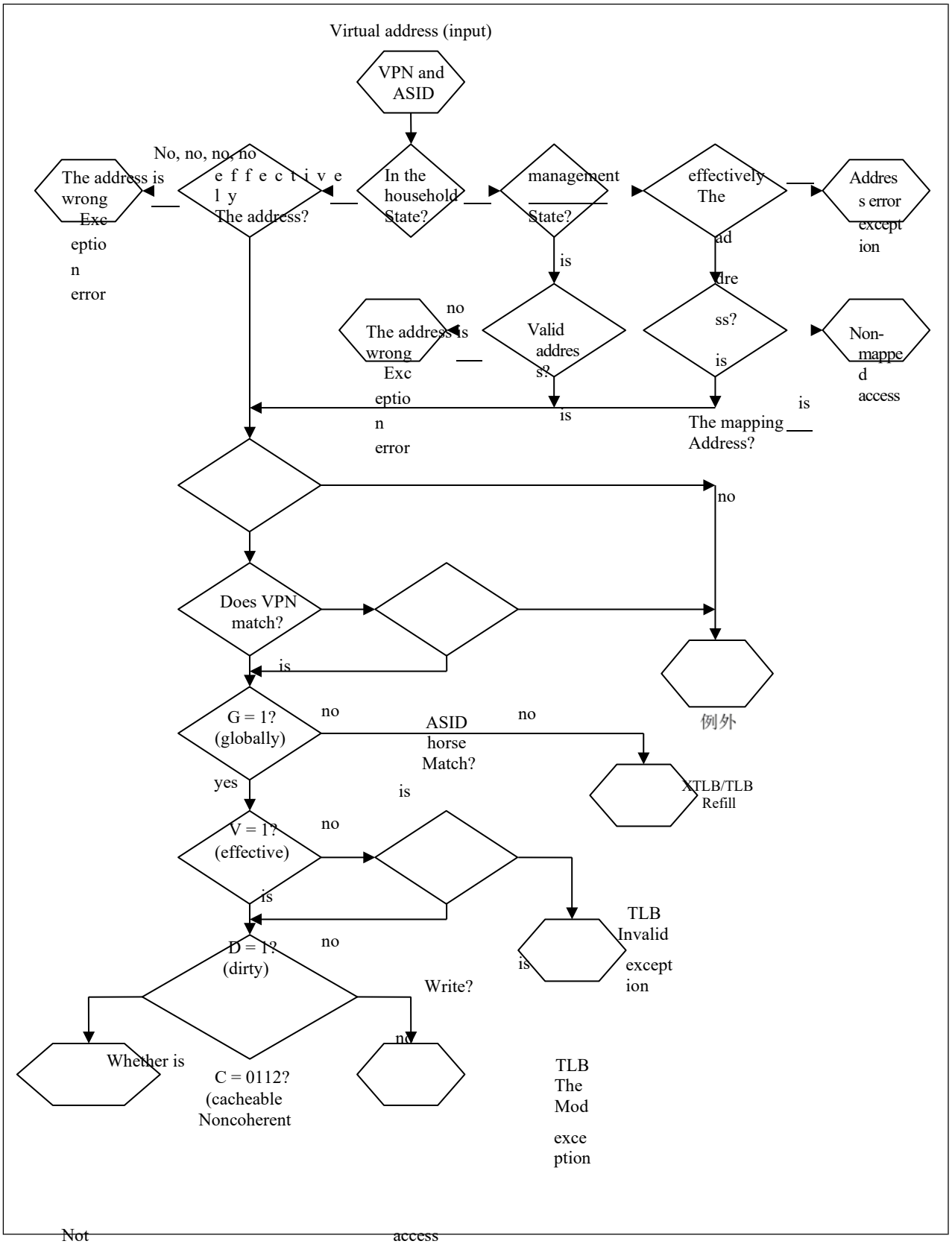

Virtual address (input)
VPN and ASID
In the household State?
management State?
effectively The address?
No, no, no, no effectively
The address is wrong
Exception error
Address error exception
is
no
The address is wrong
Exception error
Valid address?
is
address?
is
Non-mapped access
is
The mapping Address?
no
Does VPN match?
is
例外
G = 1? (globally)
no
ASID horse Match?
no
yes
is
XTLB/TLB Refill
V = 1? (effective)
no
is
TLB Invalid exception
is
D = 1? (dirty)
no
Write?
no
Whether is C = 0112? (cacheable Noncoherent
TLB The Mod exception
Not through The Cache
Physical address (output)
access The Cache

Figure 5-10 TLB address translation

5.4.4 TLB misses

If none of the TLB entries matches the virtual address, a TLB miss exception is thrown.If the access control bits (D and V) indicate that the access is not legitimate, a TLB modification or a TLB invalid exception is raised.If the C bit is equal to 011, the physical address retrieved2

Access memory through the Cache, otherwise not through the Cache.

5.4.5 TLB instruction

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

Table 5-4 TLB instructions

opcode	Instructions described
TLBP	Search for matches in TLB
TLBR	Read the TLB entry of the index
TLBWI	Write the TLB entry of the index
TLBWR	Let me write random TLB terms

5.4.6 The code example

The first example is how to configure TLB table entries to map a pair of 4KB pages.The kernel of a real-time system mostly does this, and this simple kernel MMU is only used for memory protection, so static mapping is sufficient, and all TLB exceptions are treated as error conditions (unreachable) in all statically mapped systems.

1. Mtc0 r0,C0_WIRED# make all entries available to random replacement 2. Li r2,

(vpn2<<13)|(asid & 0xff); 2.

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. *C0_PAGEMASK mtc0 r2*
10. *Li r2, index_of_some_entry# needed for tlbwi only*
11. *Mtc0 r2, C0_INDEX# needed for tlbwi only tlbwr# or*

tlbwi

A full-fledged virtual storage operating system (such as UNIX) that USES the MMU for memory protection and for main storage and bulk storage

Page change of device. This mechanism allows applications to access larger storage devices rather than just the space physically allocated by the system. This page-dependent mechanism requires dynamic page mapping.Dynamic mapping is implemented through a series of different types of MMU exceptions, and TLB refilled is the most common exception to this system.Below is a possible TLB refilling

exception control.

```
12. Refill_exception:
    13. Mfc0 k0, C0_CONTEXT
    14. Sra k0, k0, 1              Into the page table
    15. Lw k1 and 0 (k0)           # read page table
    16. Lw k0, 4 (k0)
    17. SSL k1, k1, 6
    18. SRL k1, k1, 6
    19. Mtc0 k1, C0_TLBLO0
    20. SSL k0, k0, 6
    21. SRL k0, k0, 6
    22. Mtc0 k0, C0_TLBLO1
    23. TLBWR                      # write a random entry
    eret
```

This exception control handling is simple because its frequent execution can affect system performance, which is why TLB rewrites the exception to assign independent exception vectors.This code assumes that the required mapping is already established in the main memory page table. If not, a TLB failure exception occurs after the ERET instruction. TLB failure exceptions are rare, which is beneficial because it must compute the desired mapping and may require partial page tables to be read from backup storage. TLB modification exceptions are used to implement read-only pages and tag processes to clear pages where code needs to be modified. To protect different processes and users from each other, virtual storage operating systems usually execute user programs in user mode.The following example shows how to move from kernel mode to user 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. C0_SR mtc0 r1
Eret# jump to user mode
```

5.5 Physical address space distribution

The address space of loongson 3 is uniformly distributed to each node according to the high position of the address.The height of the 48-bit address is 4 [47:44] corresponding to the empty address

Each node has a fixed 44-bit address space.The 44-bit address space within the node is further divided

For 8 41-bit address Spaces, the use of 41-bit Spaces is mainly due to the fact that one port may be connected to two HT controllers for each

HT needs a 40-bit address space.

6 Processor exception

This chapter describes the core exceptions of the loongson GS464 processor, including the generation and return of exceptions, the location of exception vectors, and the types of exceptions supported.For each type of exception supported, the description includes the reason for the exception, the handling, and the service.

6.1 Exceptions are generated and returned

When the processor starts processing an exception, the EXL bit of the status register is set to 1, which means the system is running in kernel mode.After the appropriate field state is saved, the exception handler typically sets the KSU field of the status register to kernel mode and returns the EXL position to 0.When the field state is restored and reexecuted, the handler restores the KSU field to its previous value, with the EXL bit of 1.

Returning from the exception also puts the EXL position at 0.

6.2 Exception vector position

The vector address for cold reset, soft reset, and unmasked interrupt (NMI) exceptions is the dedicated reset exception vector address 0xffffffffffbfc00000, which is neither accessed through Cache nor mapped.In addition, EJTAG debug interrupts entry 0xffffffffffffffffff200200 and 0xFFFFFFFFFF200200, respectively, according to whether the ProbeTrap bit in its control register is 0 or 1.All other exception vector addresses take the form of a base address plus a vector offset.When the BEV bit in the state register is 0, the user can define the base address of the exception vector, as shown in table 6-1.

Table 6-1 base addresses of exception vectors

exce ption	BEV = 0	BEV = 1
Reset, Soft Reset, NMI	0 XFFFFFFFF BFC00000	
EJTAG Debug (ProbEn = 0)	0 XFFFFFFFF BFC00480	
EJTAG Debug (ProbEn = 1)	0 XFFFFFFFF FF200200	
Cache Error	0 XFFFFFFFF | | EBase31.. 30 | | 1 | | EBase28.. 12 | | 0 x000	0 XFFFFFFFF BFC00300
Others	0 XFFFFFFFF | | EBase31.. 12 | | 0 x000	0 XFFFFFFFF BFC00200

Table 6-2 lists the offsets of the exception vectors in the core of the loongson GS464 processor.

Table 6-2 exception vector offsets

exce ption	Exception vector migration
TLB Refill, EXL = 0	0 x000
XTLB Refill, EXL = 0	0 x080
Cache error	0 x100
Other Shared exceptions	0 x180
The interrupt and CauseIV = 1	0 x200
Reset, Soft Reset, NMI	None (using base address)

For external interrupts (including clock and performance counter interrupts), the traditional approach is to use a common exception entry, which is distributed to the corresponding service by the software.The godson GS464 processor core support vector Interrupt mode (Vectored Interrupt), the model of the register by the Cause IV a choice.In vector interrupt mode, interrupt priority is reduced from IP7 to IP0 and there are special exception entries. The VS field of the IntCtl register controls the space occupied by these exception handling codes, and the entry offset corresponding to each interrupt can be calculated using the following formula (where the vector number starts from zero) :

Vector interrupt offset = 0x200 + vector number * IntCtlVS

6.3 Exception priority

The rest of this chapter covers the exceptions in the order of priority given in table 6-3 (for specific exceptions, such as TLB exceptions and directive/data exceptions, grouped together for convenience).When an instruction simultaneously generates more than one exception, only the highest priority exception is reported to the processor.Some exceptions are not generated by the instruction being executed at the time, and some may be deferred.See this chapter's separate introduction to each exception for more details.

Table 6-3 priority of exceptions

Exception priority order
Cold reset (highest priority)

Unmasked interrupt (NMI)
Wrong address -- pointing
TLB refill - take the finger
TLB is invalid -- take a finger
Cache error - take a pointer
Bus error - take a finger
Integer overflows, traps, system calls, breakpoints, reserved instructions, coprocessors unavailable, floating point exceptions
EJTAG interrupt
Address error - data access
TLB refilling - data access
TLB invalid - data access
TLB modification - write data
EJTAG data breakpoint
Cache error - data access
Bus error - data access
Interrupts (lowest priority)

In general, the exceptions described in the following sections are handled first by the hardware and then by the software.

6.4 Cold reset exception

why

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

To deal with

The CPU provides a special interrupt vector for this exception:

- Located at 0xBFC0 0000 in 32-bit mode
- Located at 0xFFFF FFFF bfc0000 in 64-bit mode

Cold reset vector addresses belong to the CPU address space that does not require address mapping and does not access data through the Cache, so you do not need to initialize TLB or Cache to handle this exception.This also means that the processor can fetch and execute instructions even if the Cache and TLB are in an indeterminate state.

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

- The initial value of the Status register is 0x30c000e4, the SR bit is cleared to 0, and the ERL and BEV bits are set to 1.
- 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 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:

- Initializes all processor registers, coprocessors, caches, and storage systems.
- Perform diagnostic tests.
- Bootstrap the bootstrap operating system.

6.5 NMI exception

why

NMIn for low yields an NMI exception.This exception cannot be blocked.

`

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

The NMI exception can only be extracted at the edge of an instruction. It does not discard any machine state, but rather retains the state of the processor for diagnosis. The Cause register contents remain the same while the system jumps to the beginning of the NMI exception handler.

The NMI exception preserves all register values except the following registers:

- The ErrorEPC register containing the PC value.
- Status register ERL bit set to 1.
- Soft reset or NMI set to 1, cold reset set to 0 status register SR bit.
- The status register BEV bit is set to 1.
- The PC register is reset to 0xFFFF FFFF bfc0000

service

The NMI exception can be used in situations other than "reset the processor while retaining the Cache and memory contents." For example, when a power failure is detected, the system can be shut down immediately and controllably through the NMI exception.

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

6.6 Address error exception

why

The address error exception occurs when:

- Reference invalid address space.
- Refer to the superuser address space in user mode.
- Reference the kernel address space in user or superuser mode.
- Take (Load) or Store (Store) a double word, but the double word does not align with the double word boundary.
- Fetch (Load, Fetch) or Store a word, but the word does not align with the boundary of the word.
- Take or save a half word, but the half word does not

align to the edge of the half word.This exception cannot be

blocked.

To deal with

The common exception vector is used for address error exceptions. The ExcCode field value of the Cause register is set to a ADEL or ADES encoding value, along with the EPC register and the BD bit of the Cause register, to indicate the instruction causing the exception and whether the exception is caused by an instruction reference, fetch or save operation instruction.

When an exception occurs, the BadVAddr register holds incorrectly aligned virtual addresses, or virtual addresses of the protected address space.

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 causing the exception will receive a UNIX SIGSEGV(segment violation) signal, an error that is usually fatal to the process.

6.7 TLB exception

Three TLB exceptions can occur:

- When no item in TLB matches the address of the mapped address space to be referenced, the TLB rewrites the exception.
- A TLB invalidation exception occurs when a virtual address reference matches an item

in a TLB, but that item is marked invalid.

- When the virtual address reference for a write memory operation matches an item in TLB that is not marked as "dirty.

A TLB modification exception occurs.These TLB exceptions are covered in the following three sections.

Note: TLB refilling vector selection is described earlier in this chapter. See section 6.8 "TLB refilling exception" for details.

6.8 TLB refills the exception

why

When no item in TLB matches a reference address in the mapped address space, a TLB refill exception occurs, which is unmasked.

To deal with

For this exception, the MIPS architecture has two special exception vectors: one for the 32-bit address space and one for the 64-bit address space.The exception vector is offset by 0x000 when the reference address is in the 32-bit address space, and by 0x080 when the reference address is in the 64-bit address space.

When the EXL bit in the status register is set to 0, all address references use these exception vectors.This exception sets the value of the ExcCode field in the Cause register to be a TLBL or TLBS encoding.This code, along with the EPC register and BD of Cause register, indicates the instruction causing the exception and whether the exception is caused by an instruction reference, fetch operation instruction, or save operation instruction.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold the virtual address where the address translation failed. The EntryHi register also holds the ASID if the conversion fails. The Random register usually holds the legal location for the TLB item to be replaced. The contents of the EntryLo register are uncertain. If the instruction that caused the exception is not in the branch delay slot, the EPC register holds the address of the instruction that caused the exception.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

To serve this exception, the contents of the Contex or XContext register are used as virtual addresses to obtain certain memory locations that contain a pair of physical page addresses

and access control bits for TLB entries.This TLB pair is put into the EntryLo0/EntryLo1 register; Registers EntryHi and EntryLo are written to TLB.

The virtual address used to obtain the physical address and access control information may be on a page that does not reside in the TLB.If this occurs, the TLB resend handler allows another TLB to resend the exception to resolve it.Since the EXL bit of the Status register is set to 1, the second TLB refold exception is passed with the common exception vector.

6.9 TLB invalid exception

why

A TLB invalid exception occurs when a virtual address reference matches a TLB entry that is marked as invalid (the TLB valid bits are cleared).This exception is unmasked.

To deal with

The common exception vector is used to handle this exception. The ExcCode field value of the Cause register is set to TLBL or TLBS, along with the EPC register and the BD bit of the Cause register, to indicate the instruction causing the exception and whether the exception is caused by an instruction reference, fetch operation instruction, or save operation instruction.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold the virtual address where the address translation failed. The EntryHi register also holds the ASID if the conversion fails. The Random register usually holds the legal location for the TLB item to be replaced. The contents of the EntryLo register are uncertain.

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 TLB entry is marked invalid when one of the following occurs:

- The virtual address does not exist
- Virtual addresses exist, but not in main memory (missing pages)
- Invoking this page raises a trap (for example, maintaining reference bits)

After the cause of the TLB invalid exception is served, the TLB entry is located through the TLBP instruction (which probes the TLB to find a match), and the TLB entry is replaced with one that has a valid tag bit.

6.10 TLB modification is an exception

why

The TLB modification exception occurs when the virtual address reference for the write memory operation matches an item in the TLB, but the item is not marked as "dirty" and therefore cannot be written.This exception cannot be blocked.

To deal with

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

The MOD.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold the virtual address where the address translation failed. The EntryHi register also holds the ASID if the conversion fails. The contents of the EntryLo register are uncertain.

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 virtual page number to identify the appropriate access control information.The identified page may or may not allow write access;If write access is not allowed, a write protection violation occurs.

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

6.11 Cache error exception

why

An exception to the Cache error occurs when an internal Cache check error occurs when the processor fetches a pointer or accesses memory.This exception cannot be blocked.

To deal with

The Cache error exception entry with an offset of 0x100 is used to handle the Cache error exception.At this point, the exception entry base is located in the address segment that is not in the Cache.The ExcCode field value of Cause register is set to CacheErr, along with the EPC register and BD bit of Cause register, to indicate the instruction causing the exception and whether the exception is caused by an instruction reference or an access operation instruction.The CacheErr register records the error type and the position in the group-linked Cache.The CacheErr1 register records the error instruction virtual address or memory physical

address, as described in section 3.30 of the CacheErr and CacheErr1 registers.

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 loongson GS464 processor checks Cache errors to achieve hardware self-correction, and applications can simply return directly from the exception.

If there is an error in the instruction Cache, the error Cache line will be invalid.If there is an error in the data Cache and there is only one error, the error data will be corrected automatically.If there is an error in the data Cache and there are two errors, the operating system should determine the processing method based on the location of the error data block.

6.12 Bus error exception

why

The bus error exception occurs when the processor receives an external ERR completion reply signal when it reads or updates a data block or makes a double-word/single-word/half-word read request.This exception cannot be blocked.

To deal with

Common interrupt vectors are used to handle bus error exceptions. The ExcCode field value of the Cause register is set to IBE or DBE, along with the EPC register and the BD bit of the Cause register, to indicate the instruction causing the exception and whether the exception is caused by an instruction reference, fetch or save operation instruction.

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 (representing fetch instructions), then the instruction virtual address causing the exception is stored in the EPC register (if the BD bit of the Cause register is set to 1, the virtual address of the instruction is the EPC

register contents plus 4).

If the ExcCode field value in the Cause register is set to DBE encoding (representing a read or store reference), then the instruction virtual address that causes the exception to occur is stored in the EPC register (if the BD bit of the Cause register is set to 1, the virtual address of the instruction is the EPC register contents plus 4).

The virtual address to read and store the reference can then be obtained by interpreting the instruction. And the physical address can go through

The TLBP instruction and the reading of the EntryLo register contents calculate the physical page number to obtain. The running process that causes the exception to occur receives a UNIX SIGBUS signal, which is usually fatal to the process.

6.13 The exception is integer overflow

why

The integer overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction is executed, resulting in an overflow of the resulting complement.This exception is unmasked.

To deal with

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

If the instruction causing the exception is not an instruction in the branch delay slot, the EPC register holds the address of the instruction.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 occur receives a UNIX SIGFPE/FPE_INTOVE_TRAP(floating point exception/integer overflow) signal.This error is usually fatal to the process.

6.14 Trap exceptions

why

When TGE, gue, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI,

The trap exception occurs when the TNEI instruction is executed and the condition result is true. This exception is unmasked.To deal with

The common exception vector is used to process this exception, and the ExcCode field of the Cause register is set to the TR encoding value.If the instruction that raised the exception is not an instruction in the branch delay slot, the EPC register holds 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

serv

ice

The executing process causing the exception to occur receives a UNIX SIGFPE/FPE_INTOVE_TRAP (float)

Exception/integer overflow) signal.This error is usually fatal to the process.

6.15 System call exception

why

The system call exception occurs when the SYSCALL directive is

executed.This exception is unmasked.**To deal with**

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

If the SYSCALL instruction is not in the branch delay slot, the EPC register holds the address of the instruction.Otherwise, save the address of the previous branch instruction.

If the SYSCALL instruction is in the delay slot, the BD bit in the status register is set to 1, otherwise the bit is cleared

Service

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

The Code field of the SYSCALL instruction (bit 25:6), and the contents of the instruction loaded into 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 does not occur again

Be performed; This can be done by adding 4 to the value of the EPC register before returning.

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

6.16 Breakpoint exception

why

A breakpoint exception occurs when a BREAK instruction is executed.This exception is unmasked.

To deal with

The common exception vector is used to process this exception, and the ExcCode field of the Cause register is set to the BP encoding value.If the BREAK instruction is not in the branch delay slot, the EPC register holds the address of the instruction.Otherwise,

Save the address of the previous branch instruction.

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

service

When this exception occurs, control is transferred to the appropriate system routine. Further differentiation can be performed by analyzing the Code field of the BREAK instruction (bit 25:6), and the contents of the instruction loaded into the address stored in the EPC register.If the instruction is in the branch delay slot, the contents of the EPC register must be added 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 adding 4 to the value of the EPC register before returning.

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

process.

6.17 Exception to reserved instruction

why

The exception to the reserved instruction occurs when an attempt is made to execute an instruction not defined in MIPS64 Release2 and not customized by loongson.This exception is unmasked.

To deal with

The common exception vector is used to process this exception, and the ExcCode field of the Cause register is set to the RI encoding value.If the reserved instruction instruction is not in the branch delay slot, the EPC register holds the address of this instruction.Otherwise,

Save the previous branch instruction address.

service

At this point, no instructions are interpreted and executed. UNIX SIGILL/ILL_RESOP_FAULT(illegal instruction/reserved operation error) is signaled to the executing process that caused the exception to occur.This error is usually fatal to the process.

6.18 An exception is not available for the coprocessor

why

An attempt to execute any of the following coprocessor instructions will result in a coprocessor unavailable exception:

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

The instructions.

This exception is unmasked.

Loongson custom extension instruction of the coprocessor is not available exception trigger conditions are as follows:

- The custom extended access instruction (table 2-15), the custom extended 64-bit multimedia access instruction (table 2-18), and the custom extended floating point access instruction (table 7-5) trigger coprocessor unavailability exceptions when CP2 is not marked as available.
- Custom extended floating point access instruction (table 7-5) triggers a coprocessor unavailable exception when CP1 is not marked as available.

It is important to note that the three custom extended floating-point format conversion instructions, CVT.D. del.d, and CVT. Ud.d, do not trigger a coprocessor unavailability exception even if CP1 is not marked as available.

To deal with

The common exception vector is used to process this exception, and the ExcCode field of the Cause register is set to the cpu-encoded value. The Cause register's CE field indicates which of the four coprocessors is referenced. If the instruction is not in the branch delay slot, the EPC register holds the address of the non-usable coprocessor instruction.Otherwise, the EPC register holds the address of the previous branch instruction.

service

There are several scenarios as follows:

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

If the process is authorized to access the coprocessor, but the coprocessor does not exist or has a fault, the coprocessor instruction needs to be interpreted/emulated.

If the BD bit in the Cause register is set, the branch instruction must be interpreted.The coprocessor instructions are then simulated.The coprocessor instruction that skipped the exception continues when the exception returns.

If the process is not authorized to access the coprocessor, the executing process receives a UNIX SIGILL/ILL_PRIVIN_FAULT signal.This mistake is usually fatal.

6.19 Floating-point exception

why

Floating point coprocessors use floating point exceptions. This exception is unmasked.

To deal with

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

The contents of the floating point control/status register indicate the reason for this exception.

service

This exception can be cleared by clearing the appropriate bit in the floating point/status register.

6.20 EJTAG exception

EJTAG exceptions are triggered when certain ejtag-related conditions are met.See chapter X for details

6.21 Interrupt exception

why

An interrupt exception occurs when one of the eight interrupt conditions is triggered. The importance of these interrupts depends on the particular system implementation.

Any of the eight interrupts can be masked by clearing the corresponding bit in the interrupt-mask (IM) field in the state register, and all eight interrupts can be masked at once by clearing the IE bit in the state register.

To deal with

The Cause register's ExcCode field is set to an INT encoding value. Depending on the current configuration, the processor USES traditional common exception vector processing or USES the vector exception pattern to select the entry corresponding to the highest priority interrupt number.

The IP domain in the Cause register indicates the current interrupt request. More than one interrupt bit may be set at the same time (if the interrupt is triggered and cancelled before the register is read, no bit may even be set).

IP[7] interrupts have three sources, except for the break line 5, which is generated when the contents of the Count register are equal to the contents of the Compare register or when the CP0 performance counter overflows.Clock interrupts and performance counter overflow interrupts are indicated by TI and PCI bits in the Cause register.

If the vector interrupt pattern is not used, the software needs to query every possible interrupt source to determine the cause of the interrupt (an interrupt may have multiple sources at the same time).

service

If the interrupt is caused by one of the two software exceptions, set the corresponding bit in the Cause register, IP[1:

0], is 0 to clear the interrupt condition.

Software interrupts are imprecise. Once a software interrupt is triggered, the program may continue to execute several instructions before the exception is processed.Clearing of timer interrupts is accomplished by writing a value to the Compare register.The elimination of the performance

counter interrupt is the overflow bit to the counter, namely 31, writing 0 to achieve.

Cold reset and soft reset will clear all outstanding external interrupt requests, IP[2] to IP[6].

If the interrupt is hardware-generated, the interrupt condition can be cleared by undoing the condition that caused the triggered interrupt pin.

7 Floating point coprocessor

This chapter describes the features of the Floting Point Unit (FPU), including the programming model, instruction set and instruction format, instruction pipeline, and exceptions.The loongson 3A1000 floating point coprocessor and its related system software are fully compliant with the ANSI/IEEE 754-1985 binary floating point computing standard.

7.1 An overview of the

As the CPU's Coprocessor, FPU, known as Coprocessor 1, performs floating point arithmetic operations by extending the CPU's instruction set.

FPU consists of the following two functional units:

- FALU1 unit
- FALU2 unit

FALU1 module can be performed in addition to the floating point to fetch and floating-point and fixed-point data transfer all of the floating-point operations, including floating add (subtract) method, the floating-point multiplication, floating point multiply add (subtract), floating-point division, floating point open square root, floating down, floating point open after the root pour, floating-point and fixed-point conversion, floating point precision conversion, floating point comparison, judgment and other simple logic. In addition, the FALU1 module performs SIMD media operations through the extension and reuse of the FMT domain in the instruction encoding.

FALU2 performs floating-point multiplication and addition operations (computable floating-point multiplication, addition, and floating-point multiplication and addition instructions), as well as media instruction operations.Meanwhile, loongson 3A1000's FPU supports the execution of MIPS64 instruction set of Paired - Single (PS) floating point instructions.Figure 7-1 illustrates the organizational structure of the

functional units in the loongson 3A1000 architecture.

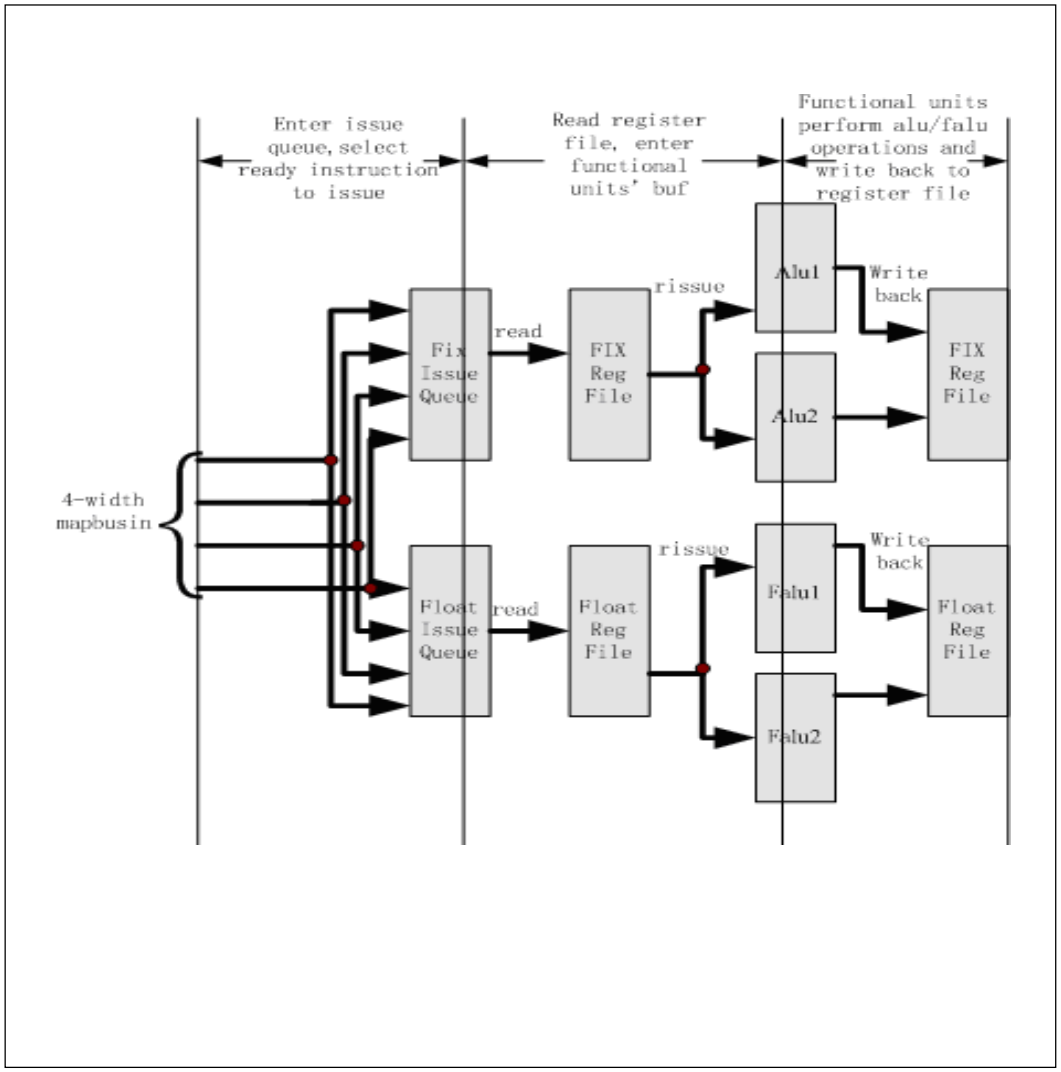


Figure 7-1 organization of functional units in loongson 3A1000 architecture

Floating point queue can emit 1 instruction to FALU1 cell and 1 instruction to FALU2 cell per clock cycle.The floating point register file provides three dedicated read ports for each FALU1 cell and one dedicated write port for each FALU1 cell.

7.2 FPU register

This section describes the FPU register group and their data organization structure.The loongson 3A1000's FPU register is compatible with the MIPS64's FPU register. The FPU register of MIPS64 includes a floating point register and a floating point control register. Floating point control registers include FIR (no. 1), FCSR (no. 31), FCCR (no. 25), FEXR (no. 26), FENR

(28th) etc.

7.2.1 Floating point register

The floating point register of loongson 3A1000 follows the usage of R10000, which is slightly different from MIPS64.Mail in Status control

When the FR bit of the memory is 1, there are 32 64-bit floating point registers, as shown in the figure below.In the Status control register

With FR bit 0, R10000 has only 16 32-bit or 64-bit floating point registers, while MIPS64 has 32 32-bit registers

Bit floating point registers or 16 64 bit floating point registers.

630	630
f1	f0
f3	f2
f5	f4
f7	f6
f9	f8
f11	f10
f13	F12 .
f15	f14
F-17 thunder	f16
f19	f- 18s
f21	f20
mo vie ma ker	ma kin g
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, which contains floating point unit functions, such as processor ID, revision version number and other information. The FIR in loongson 3A1000 has an initial value of 0x00770501.

Figure 7-3 shows the format of the FIR register, and table 7-1 describes the domain of the register.

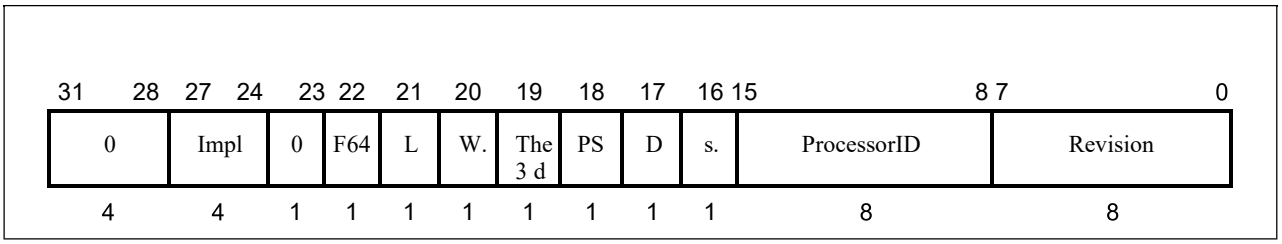


FIG. 7-3 FIR register

Table 7-1 FIR register fields

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
Impl	Implement related
F64	Whether the floating point data path is 64 bits 0 to 32 1-64
L	Whether long word (64 bit) fixed-point data types are implemented 0 - unrealized 1 - have been implemented
W.	Whether the word (32-bit) fixed-point data type is implemented 0 - unrealized 1 - have been implemented
The 3 d	Whether MIPS-3D ASE is implemented 0 - unrealized 1 - have been implemented
PS	Whether floating point is implemented for data types 0 - unrealized 1 - have been implemented
D	Whether the double-precision floating point data type is implemented 0 - unrealized 1 - have been implemented

The domain	describe
s.	Whether a single-precision floating point data type is implemented 0 - unrealized 1 - have been implemented
ProcessorID	Floating point processor identifier
Revision	The revision version number of the floating point unit

7.2.3 FCSR register (CP1, 31)

The FCSR register is used to control the operation of floating point units and to represent some state. The initial value of FCSR in GS464 is 0x00000F80. The format of the FCSR register is shown in figure 7-4, and the field of the FCSR register is described in table 7-2.Among them, E, V, Z, O, U and I respectively represent unrealized operation, invalid operation, division by zero, overflow, underflow and imprecision.

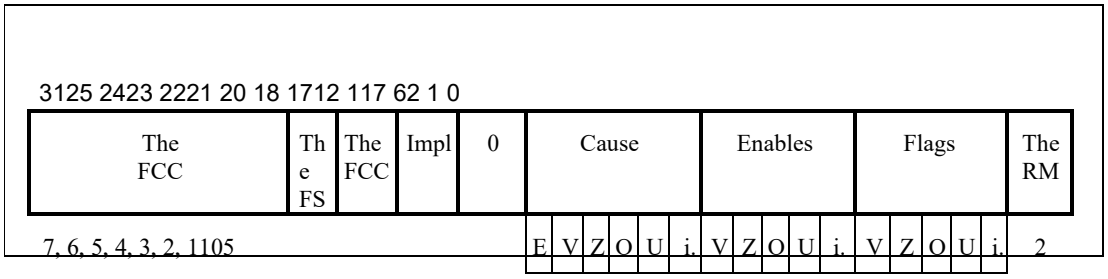


Figure 7-4 FCSR register

Table 7-2 FCSR register fields

The domain	describe
0	Retained. You must press 0 to write and return 0 on read.
The FCC	Floating point condition code. Records floating-point comparison results for conditional jumps or transitions.
The FS	It goes to zero.When this is set, the result of the abnormal operation will be set to 0, instead of producing an exception.
Impl	For implementation correlation, GS464 USES FSCR[21] as top_mode, a bit that indicates when decoding whether to rename floating point register Numbers using the X86 TOP register.
Cause	When an exception to a floating point operation is generated, the corresponding bit is set.
Enables	Whether to allow the corresponding conditions to produce exceptions.
Flags	Whether an IEEE floating point exception is generated.(for example, not opening the corresponding bit in Enables you to view this field)

The RM	Whether the double-precision floating point data type is implemented 0 - unrealized 1 - have been implemented

Control/status register condition (CC0) bit

When a floating point comparison occurs, the result is stored in the CC0 bit, the conditional bit.If the comparison result is true, CC0 bit is set to 1;And the opposite is zero. The CC0 bit can only be modified by the floating point comparison instruction and the CTC1 instruction.

The control/status register Causes the domain

The control/status register bits 17:12 are the Causes field, and these bits reflect the results of the most recent instruction execution. The Causes domain is a logical extension of the Cause register of coprocessor 0. These bits indicate the exception caused by the last floating point operation and generate an interrupt or exception if the corresponding Enable bit (s) is set. If more than one exception is generated in an instruction, each corresponding exception causes the bit to be set.

The Causes domain can be overwritten by every floating point operation instruction (not including Load, Store, Move). Among them, if the software simulation is needed to complete, the unrealized operation bit (E) of this operation is set to 1, otherwise, it remains at 0.The other bits are either 1 or 0 in accordance with the IEEE754 criterion to see if corresponding exceptions are generated.

When a floating point exception occurs, no results will be stored, and the only state affected is the Causes domain.

Control/status register Enables the field

Any time the Cause bit and the corresponding Enable bit are both 1, a floating point exception is generated.If a floating point operation is set to a Cause bit that is allowed to be activated (the corresponding Enable bit is 1), the processor immediately generates an exception, just as it does with the CTC1 instruction setting both the Cause bit and the Enable bit to 1.

There is no corresponding enabling bit for an unimplemented operation (E), which always generates a floating point exception if an unimplemented

operation is set.

Before returning from a floating point exception, the software must first clear the activated Cause bit with a CTC1 instruction to prevent interrupted repeat execution.Therefore, a program running in user mode will never observe that the enabled Cause bit has a value of 1;If the user-state handler needs to get this information, the contents of the Cause bit must be passed somewhere other than 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, and the default result defined by the IEEE754 standard is written back.In this case, the exception caused by the previous floating point instruction can be determined by reading the values of the Causes domain.

Control/status register Flags field

The flag bit is cumulative, indicating that an exception has occurred since it was explicitly reset last time.If an IEEE754 exception is generated, the corresponding Flag bits are set to 1, otherwise they remain unchanged, so these bits will never be cleared for floating point operations.However, we can set or clear the Flag bits by the CTC1 control instruction by writing a new value into the status register.

When a floating point exception occurs, the Flag bit is not set by the hardware;It is the responsibility of the floating-point exception handler to set these bits before invoking the user program.

The rounding mode (RM) field of the control/status register

The zeroth and first bits of the control/status register make up the rounding mode (RM) field.As shown in table 7-3, FPU rounds all floating point operations according to the rounding method specified by these bits.

Table 7-3 rounding mode bit decoding

Rounding mode RM (1-0)	mnemonics	describe
0	RN	Round the result to the direction closest to the number that can be represented. When the two closest to the number that can be represented are equally close to the result, round to the direction closest to the number that has the lowest position of 0.
1	RZ	Rounding to zero: rounds the result to the number closest to it and no greater than it in absolute value.
2	The RP	Rounding to positive infinity: rounding the result to the number closest to it and not less than it
3	The RM	Rounding to negative infinity: rounding 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 field. Its content is exactly the same as the FCC bit in FCSR, except that the FCC bit in this register is continuous. Figure 7-5 shows the format of the FCCR register.

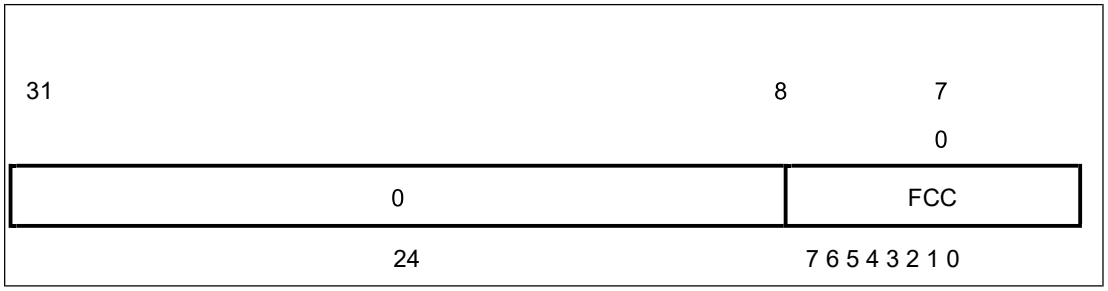


Figure 7-5 FCCR register

7.2.5 FEXR register (CP1, 26)

The FEXR register is another way to access the Cause and Flags fields, which have exactly the same contents as the corresponding fields in FCSR. Figure 7-6 shows the format of the FEXR register.

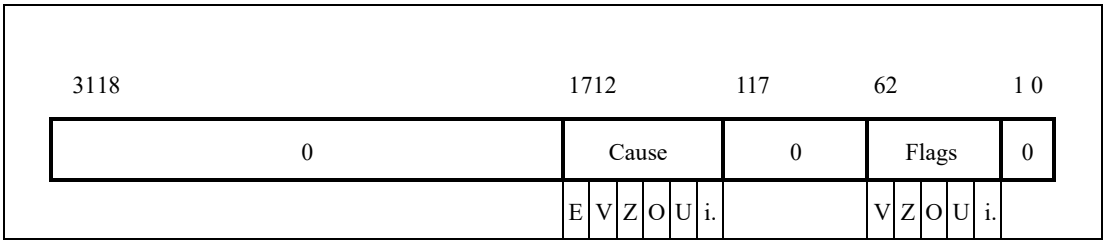


Figure 7-6. FEXR register

7.2.6 FENR register (CP1, 28)

The FENR register is another way to access the Enable, FS, and RM fields, and its content is exactly the same as the corresponding fields in FCSR. Figure 7-7 shows the format of the FENR register.

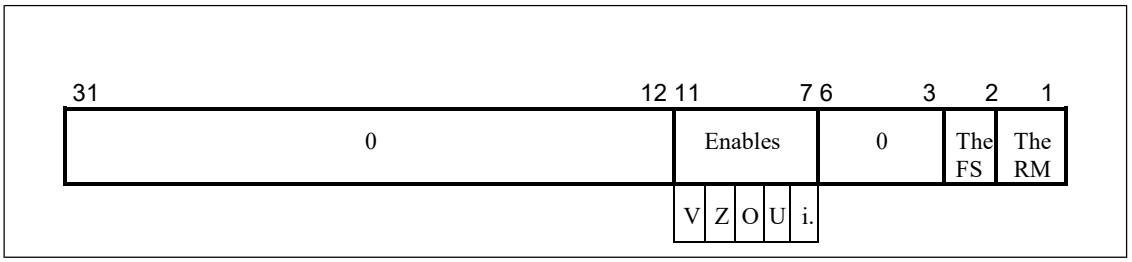


Figure 7-7. FENR register

7.3 Floating-point instructions

7.3.1 MIPS64 compatible floating point instruction list

GS464 implements all data types in the FPU part of MIPS64, including S, D, W, L, and optional PS. Table 7-4 lists the FPU instructions for the MIPS64 part of GS464.

Table 7-4 FPU instruction set of MIPS64

OpCode	The	MIPS ISA

	Description	
Arithmetic instructions		
ABS. FMT	The absolute value	MIPS32
ADD the FMT	add	MIPS32
DIV. FMT	division	MIPS32
MADD. FMT	By adding	MIPS64
MSUB. FMT	By reducing	MIPS64
The MUL. FMT	The multiplication	MIPS32
NEG. FMT	complementation	MIPS32
NMADD. FMT	Multiply and add to get the inverse	MIPS64
NMSUB. FMT	Multiply and subtract to get the inverse	MIPS64
RECIP. FMT	For the bottom	MIPS64
RSQRT. FMT	Take the inverse of the square root	MIPS64
SQRT. FMT	The square root	MIPS32
SUB. FMT	subtraction	MIPS32
Branch jump instruction		
BC1F	Floating point false time jump	MIPS32
BC1FL	Likely jump when floating point false	MIPS32
BC1T	Floating-point true time jump	MIPS32
BC1TL	Likely jump when floating point true	MIPS32
More instructions		
Arthur c. ond. FMT	Compare floating point value juxtaposes flag bits	MIPS32
Transformation instruction		
CEIL. L.f mt	Convert to a 64-bit fixed point and round up	MIPS64
CEIL. W.f mt	Convert to a 32-bit fixed point, and round up	MIPS64
The CVT transmission. D.f mt	Floating-point or fixed-point conversion to double-precision floating-point	MIPS32
The CVT transmission. L.f mt	Converts a floating point value to a 64-bit fixed point	MIPS64

The CVT transmission. PS. S	Converts two floating point values to floating point pairs	MIPS64
The CVT transmission. Supachai panitchpakdi L	Converts the low position of a floating point pair to a single precision floating point	MIPS64
The CVT transmission. Supachai panitchpakdi L	Converts the high point of a floating point pair to a single precision floating point	MIPS64
The CVT transmission. S. mt	Floating-point or fixed-point conversion to single-precision floating-point	MIPS32
The CVT transmission. W.f mt	Converts a floating point value to a 32-bit fixed point	MIPS32
FLOOR. L.f mt	Convert to 64 bit fixed point, round down	MIPS64
FLOOR. W.f mt	Convert to 32-bit fixed point, round down	MIPS64
PS PLL.	Merge two floating point pairs into a new floating point pair	MIPS64
PLU. PS	Merge the low and high values of two floating point pairs into the new floating point pair	MIPS64
PUL. PS	Merge the high and low values of two floating point pairs into the new floating point pair	MIPS64
PUU. PS	Merge two floating point pairs into a new floating point pair	MIPS64

ROUND. L.f mt	Round the floating point number to a 64-bit fixed point	MIPS64
ROUND. W.f mt	Round the floating point to a 32-bit fixed point	MIPS32
TRUNC. L.f mt	Rounds a floating point to a 64-bit point in the direction of a small absolute value	MIPS64
TRUNC. W.f mt	Rounds a floating point to a 32-bit point in the direction of a small absolute value	MIPS32
To fetch instruction		
LDC1	Access binary words from memory	MIPS32
LDXC1	To access binary words from memory by index	MIPS64
LUXC1	Access doublets from memory by unaligned index	MIPS64
LWC1	Access a word from memory	MIPS32
LWXC1	Access the word from memory by index	MIPS64
SDC1	Save the double word into memory	MIPS32
SDXC1	Save double word to memory by index	MIPS64
SUXC1	Save the double word to memory according to the unaligned index	MIPS64
SWC1	Save the word into memory	MIPS32
SWXC1	Save the word to 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 the word from FPR to GPR	MIPS64
DMTC1	Copy the word from GPR to FPR	MIPS64
MFC1	Copy the low word from FPR to GPR	MIPS32
MFHC1	Copy the high word from FPR to GPR	MIPS32 R2
ALNV. PS	Variable floating point alignment	MIPS64
MOV. FMT	Copy FPR	MIPS32
MOVF. FMT	Copy FPR when floating point false	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
MTC1	Copy the low word from GPR to FPR	MIPS32
MTHC1	Copy the high word from GPR to FPR	MIPS32 R2

7.3.2 MIPS64 compatible floating point instruction implementation

GS464 is compatible with MIPS64 Release 2, and functionally implements all FPU instructions specified in the MIPS64 architecture. However, some instructions have subtle but important differences in implementation that do not affect compatibility. The following two points deserve programmers' attention.

Multiply plus, multiply minus instructions. The execution of MADD. FMT, MSUB. FMT, NMADD. FMT, NMSUB. FMT this four set of instructions, GS464 computing results with MIPS64 processor is slightly different, this is because the GS464 when doing multiplication and operations only in the final results do rounding precision (so-called fused - mulitply - add), and MIPS64 processor after to take operation and after operation respectively two rounding, two rounding treatment of different lead to its lowest level in some cases the results are 1.

Single precision operation instruction. When the FR bit of Status control register is 0, abs. S, add.s, ceil. W.d, ceil. W.s, div. S, floor. W.s, floor.

Mov. s, CVT. S, CVT. S, CVT. S.d, CVT. S.w, CVT. W.s, movf. S, movn. S, movt. S, movz. (in early MIPS processors, the FR bit represented whether the floating point register was 16 or 32, and in MIPS64 the FR bit represented whether the floating point register was 32 or 64).

7.3.3 Loongson custom extension floating point instruction

Table 7-5 custom extended floating point access instructions

Instruction mnemonics	Command function description
GSSQC1	The dual source register stores the fixed point four words
GSSWLEC1	Saves a word from a floating point register with an overbounds check
GSLWXC1	A floating-point word with an offset

GSLQC1	The dual target registers fetch floating point characters
GSLWLEC1	Bring the fetch word to the floating point register for overbounds checking
GSLWLC1	Fetch the left part of the word to the floating point register
GSLWRC1	Fetch the right part of the word to the floating point register
GSLDLC1	Take the left part of the word to the floating point register
GSLDRC1	Take the right part of the word to the floating point register

Instruction mnemonics	Command function description
GSLWGTC1	Fetches a word to the floating point register with an overbounds check
GSLDLEC1	Bring the fetch double word to the floating point register with the overbounds check
GSLDGTC1	Fetch a double word to a floating - point register with the down - bounds check
GSLDXC1	Fetch floating point doublet with offset
GSSWLC1	Saves the left part of the word from the floating point register
GSSWRC1	Saves the right part of the word from the floating point register
GSSDLC1	Saves the left part of the word from the floating point register
GSSDRC1	Saves the right part of the word from the floating point register
GSSWGTC1	Saves a word from a floating - point register with a down - bounds check
GSSDLEC1	Save the double word from the floating point register with the overbounds check
GSSDGTC1	Saves a double word from a floating point register with a down - bounds check
GSSWXC1	Store floating point words with offset
GSSDXC1	Store floating point doubleword with offset

Table 7-6 custom extended floating-point format conversion instructions

Instruction mnemonics	Command function description
The CVT transmission. D.L D	Extended double precision to double precision
The CVT transmission. LD. D	Double precision converted to extended double low precision
The CVT transmission. UD. D	Double precision converted to extended double high precision

7.4 Floating point part format

7.4.1 floating point format

FPU can operate on both 32-bit (single-precision) and 64-bit (double-

precision) ieee-compliant floating point Numbers. The 32-bit single-precision format includes a 24-bit decimal field (F+S) represented by the sign-amplitude and an 8-bit exponential field (E). The 64-bit double-precision format includes a 53-bit symbol-amplitude representation of the 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 figure

7-8 Shown below.

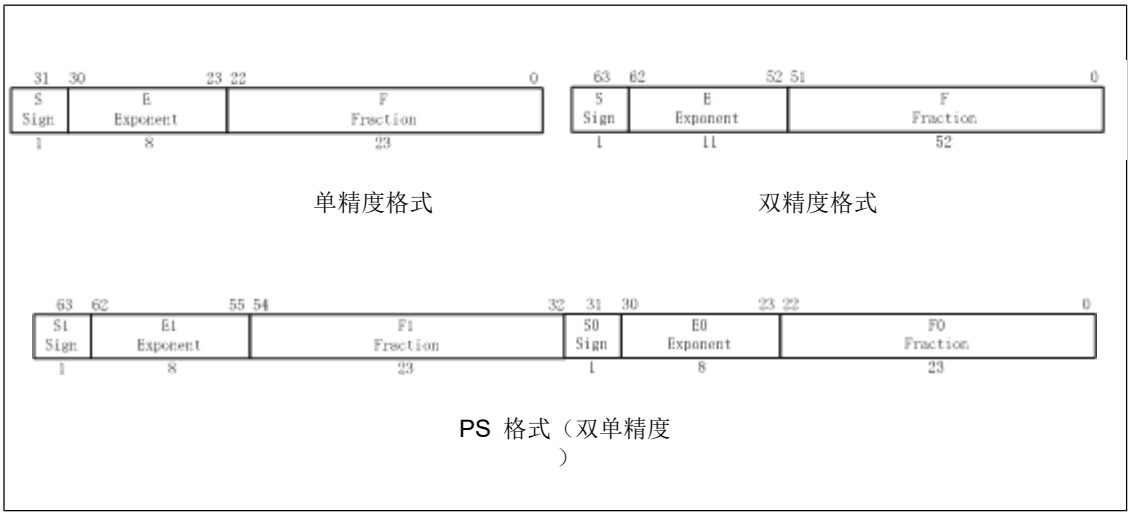


Figure 7-8 floating point format

As shown in figure 7-8, the format for floating point Numbers consists of the following three fields:

- Sign field, S
- The biased exponential field, E = E + Bias, E is the non-biased exponential$_{00}$
- In the decimal field, F=.bbb$_{12}$ $_{p-1}$

The range of exponent E is the integer between all of them, including E and E, plus the following two reserved values:$_{0minmax}$

- E-1 (used to encode 0 and subnormal Numbers)$_{min}$
- E +1 (used to encode infinity and NaN[Not a Number])$_{max}$

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 formulae for calculating the values of floating point Numbers in single-precision and double-precision formats

NO.	The 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= (–1)S∞□
(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 an NaN, then the highest bit of F determines the number to be Signaling

NaN or Quiet NaN: if the highest bit of F is set, then V is Signaling NaN, otherwise V is

Quiet NaN.Tables 7-8 define the values of some related parameters in floating point

format.The maximum and minimum floating point values are given in tables 7-9.

Table 7-8 floating point format parameter values

parameter	format	
	Single precision	double
Emax	+ 127	+ 1203
Emin	- 126.	- 1022.
Exponential offset	+ 127	+ 1023
Exponential bit width	8	11
An integer bit	Hidden	Hidden
F (decimal width)	24	53
Overall format width	32	64

Table 7-9 floating point values for maximum and minimum Numbers

type	value
Single precision floating point minimum number	1.40129846 e - 45
The minimum normal number of a single precision floating point	1.17549435 e - 38
The largest single-precision floating point number	3.40282347 e
The smallest number in a double-precision floating point	4.9406564584124654 e - 324
The minimum normal number for a double-precision floating point	2.2250738585072014 e - 308
The largest number in a double-precision floating point	1.7976931348623157 e+308

7.5 Overview of FPU instruction pipeline

FPU provides an instruction pipeline parallel to the CPU instruction

pipeline. It shares the basic 9-level pipeline architecture with the CPU, but the execution flow level is subdivided into 2-6 flow levels depending on the floating point operation.Each FPU instruction is executed by one of two floating point units: FALU1 or FALU2. FALU1 can perform all floating point operations and media operations. FALU2 performs only floating-point addition and subtraction, multiplication, multiplication and addition, and all media operations.

Each FALU cell can receive one instruction per cycle and send one result to the floating point register file.In each FALU unit, floating-point addition and subtraction, floating-point multiplication, and floating-point multiplication and addition require six cycles of execution.The format conversion operation between fixed point and floating point needs 4 execution cycles.Floating point division takes between 4 and 16 cycles depending on the operand.Depending on the operand, it takes between 4 and 31 cycles to open a square root of floating point, and 2 cycles for other floating point operations

Cycle. In each FALU unit, if two instructions with different execution cycles output the result in the same beat, in this case, the instruction with shorter execution cycle takes priority to output the result to the bus.Floating-point operations and all media operations except floating-point division and floating-point root are fully pipelinized. If there are two floating-point division instructions or two floating-point open square root instructions in FALU1 at the same time, the FALU1 unit will send a stop signal to the previous stream level, and the FALU1 unit will not receive new instructions until the division or open square root instruction is written back.

7.6 Floating point exception handling

This section describes the exceptions to floating point calculations. Floating point exceptions occur when FPU cannot handle operands or the

results of floating point calculations in a normal way, and FPU generates an exception to start the corresponding software trap or to set the status flag bit. The FPU control and status registers contain an enabling bit for each type of exception, which determines whether an exception is

Can cause FPU to start an exception trap or to set a status flag.

If a trap is started, FPU maintains the state at which the operation started and starts the software exception processing path;If no trap is started, an appropriate value is written to the FPU target register and the calculation continues.

FPU supports five IEEE754 exceptions:

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

And the sixth exception:

- Unimplemented Operation (E)

Unimplemented operation exceptions are used when FPU cannot execute the standard MIPS floating point structure, including when FPU cannot determine the correct exception behavior.This exception indicates the execution of the software exception handling. The unimplemented operation exception does not enable the signal and flag bit, and when this exception occurs, a corresponding unimplemented exception trap occurs.

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

trap to occur.

When there are no exception trap signals, the floating-point processor handles them by default, providing a floating-point calculation exception

The substitution value of the fruit. Different exception types determine different default values.Table 7-10 lists the default handling of FPU for each IEEE exception.

Table 7-10 default handling of exceptions

The domain	describe	Rounding mode	The default action
i.	Inexact exception	Any	Provide rounding results
U	Underflow exception	RN	Set the result to zero according to the symbol of the intermediate result
		RZ	Set the result to zero according to the symbol of the intermediate result
		The RP	Correct positive underflow to a minimum positive number and negative underflow to minus 0
		The RM	Correct negative underflow to a minimum negative number and positive underflow to +0
O	Overflow exception	RN	Set the result to infinity according to the symbol of the intermediate result
		RZ	Maximize the result according to the symbol of the intermediate result
		The RP	Correct negative underflow to the maximum negative number and positive underflow to plus infinity
		The RM	Correct positive underflow to the maximum integer and negative underflow to minus infinity
Z	Be zero except	Any	Provides a corresponding signed infinite number
V	Illegal operation	Any	Provide a Quiet Not a Number(QNaN)

The following describes the conditions that lead to each exception in FPU and describes in detail the FPU's response to each exception.

Imprecision exception (I)

The FPU generates an imprecise exception when:

- Rounding results are not accurate
- Rounding results overflow
- Rounding results underflows, and underflows and imprecise enable bits are not set, and the FS bit is set.Trap enabled result: if an imprecise exception trap is enabled, the result register is not modified, and the source

Registers are retained. Because this execution mode affects performance, the imprecise exception trap is enabled only when necessary.

Trap not enabled results: if no other software trap occurs, the rounded or overflowed results are sent to the target register.

Illegal operation exception (V)

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

- Addition or subtraction: infinite subtraction. For example :(+∞)+(-∞) or (-∞)-(-∞).
- Multiplication: 0 times infinity, for all positive and negative Numbers
- Division: 0/0, ∞/∞, for all positive and negative Numbers
- The operand when the Unordered comparison operation is not processed is Unordered
- Perform a floating-point comparison or conversion on an indicator NaN
- Any mathematical operation on SNaN (Signaling NaN).This exception is caused when one of the operands is SNaN or when both operands are SNaN (MOV operations are not considered mathematical operations, but ABS and NEG are considered mathematical operations)
- The square root: X, when X is less than 0

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

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

Trap not enabled result: if no other exception occurs, QNaN is sent to the target register.

Except for zero (Z)

In a division operation, when the divisor is 0 and the dividend is a finite number that is not zero, a signal is sent that the divisor is zero.Software can be used to generate symbolic infinities for other operations, such as ln(0), sin(PI /2), cosine (0), or 0.-1

When the trap is enabled: the result register is not modified and the source register is retained.

Trap failure: if no trap occurs, the result is a signed infinity.

Overflow exception (O)

When the magnitude of the rounded floating point result is represented by an unbounded exponent, the upper overflow exception sends a notification signal when the target mode greater than the maximum represents limited data.(this exception sets both the inexact exception and the flag bit)

When the trap is enabled: the result register is not modified and the source register is retained.

Trap failure: if no trap occurs, the final result is determined by the rounding pattern and the symbol of the intermediate result.

Underflow exception (U)

Two related events led to the overflow exception:

- A very small non-zero result between ± 2, which is very small, leads to a subsequent overflow exception.^{Emin}
- Denormalized Number is used to approximate the serious data distortion generated by these two small data.

IEEE754 allows these events to be detected in many different ways, but the same method is required for all operations. Small data can be detected using one of the following methods:

- After rounding (if a non-zero data is calculated in the case that the exponential range is not bounded, it should be strictly between $\pm 2)^{Emin}$
- Before rounding (if a non-zero data is calculated in the case that there is no limit between the exponent and the precision range, it should be strictly between $\pm 2)^{Emin}$

The structure of MIPS requires small data to be detected after rounding.

Accuracy distortion can be detected by one of the following methods:

- Distortion of subnormal data (when the resulting result is different from the calculated result when there is no bound to the exponent)
- Inexact data (when the resulting result is different from the calculated result when the exponent and precision range are not bounded)

The MIPS structure requires the accuracy distortion to be detected to produce inaccurate results.

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

Trap not enabled: if the overflow or imprecision exception is not enabled, and the FS bit is set, the final result is determined by the rounding mode and the symbol bit of the immediate result.

Unimplemented operation exception (E)

Unimplemented operations in the FPU control/status register cause bits to be set and trap when any opcodes or operation format instructions are executed that are reserved for later definitions.The source operands and

destination registers remain unchanged while the instructions are emulated in the software. Any of the exceptions in IEEE754 can be generated from the simulation operation, which in turn can be simulated. In addition, unimplemented instruction exceptions can occur when the hardware fails to properly perform some rare operation or result condition.These include:

- Denormalized Operand, except for comparison instructions
- Quite Not a Number operand (QNaN), except for the comparison instruction
- Subnormal data is either overflowed, and the overflowed or imprecisely enabled signal is set while the FS bit is not set

buy

Note: subnormal and NaN operations only enter the trap in the transformation or computation instruction, not in the MOV instruction
The trap.

When the trap is enabled: the original operation data is not sent. The trap cannot be disabled.

8 Performance analysis and optimization

This chapter provides some information related to software performance optimization in the loongson 3A1000 architecture, including instruction delay and the interval of instruction loops, extension instructions, instruction flow and storage access processing, etc., for reference by compilers and other software developers.

8.1 Delays and cycle intervals for user instructions

Table 8-1 shows the delays and cycle intervals for all user instructions executed in the alu1/2, MEM, falu1/2 function units, excluding kernel instructions and control instructions.The instruction delay here is the number of beats (one processor cycle per beat) needed for the instruction to be transmitted to the result that it can be used by the next instruction.For example, most ALU instructions have a delay of 2, which means that the result of an ALU instruction can only be used by subsequent instructions after a beat.Therefore, a correlation loop (the next loop depends on the result of the previous loop) in the form of I = I + 1 cannot produce a result per beat.The cycle interval of an instruction refers to the frequency at which the functional unit receives the instruction. 1 means that each beat can receive more than one of the same kind of instruction, and n means that after the functional unit receives one of the same kind of instruction, it can only receive the same kind of instruction after n-1 beats.The command cycle interval of the all-flow feature is 1.

Table 8-1 loongson 3A1000 instruction delay

Instruction type	Perform component	delay	Loop interval
The integer operation			
The add/sub/logical/shift/lui/CMP	ALU1/2,	2	1
The trap/branch	ALU1	2	1
MF/MT HI/LO	ALU1/2,	2	1
(D) MULT (U)	ALU2	5	2

(D) MULT G (U)	ALU2	5	1
(D) DIV (U)	ALU2	2-38	10-76.
(D) DIV G (U)	ALU2	2-38	4-37
(D) MOD G (U)	ALU2	2-38	4-37
The load	MEM	5	1
store	MEM	-	1
Floating point operations			
(D) MTC1 / MFC1 (D)	MEM	5	1
Abs/neg/Arthur c. ond/bc1t/bc1f/move/CVT *	FALU1	3	1
Round/trunc ceil/floor/CVT *	FALU1	5	1
The add/sub/the mul/madd/msub/nmadd/nmsub	FALU1/2,	7	1
Div. S	FALU2	5-11	4-10
Div. D.	FALU2	5-18	4-17
SQRT.. s.	FALU2	5 to 17	4-16
SQRT.. d	FALU2	5-32	4-31
Lwc1, ldc1	MEM	5	1
Instruction type	Perform component	delay	Loop interval
Swc1, sdc1	MEM	-	1

For table 1, there are several additional comments:

The loop interval for load/store operations here does not include LL/SC. LL/SC is waiting for launch operations and can only be launched if they are at the head of the reorder queue and the cp0 queue is empty at this time.

There are no special usage restrictions for the HI/LO register.They are used like other general purpose registers. This table does not contain CTC1/CFC1.They are serialized like many other control instructions.

This table also does not contain multimedia instructions. Because they are done by extending the format of ordinary floating point instructions, they have the same units of functionality and latency as the extended instructions.

8.2 Instruction expansion and usage considerations

Loongson 3A1000 has completed the following command extensions:

Write only a fixed point multiplication and division of the result to the general register. It includes 12 instructions: (D)MULTG, (D)MULTUG, and (D)DIVG

MODG DIVUG (D), (D), (D) MODUG

In the standard MIPS instruction set, multiplication and division require writing two special result registers (HI/LO) in one operation, which are difficult to implement in a RISC pipeline.In

order to use these results, it would have to be taken out of HI/LO and into the general purpose register with an additional instruction.To make matters more complicated, many MIPS processors have some limitations on the use of these instructions due to pipelining issues.These new instructions are faster to execute and easier to use.

Due to the register renaming implementation, on the loongson 3A1000 processor, the relevant instructions in the standard MIPS instruction set involving the HI/LO register operation can only be executed after all the instructions in the processor instruction queue before these instructions are submitted, which will cause the pipeline to stop.However, there is no HI/LO register rename problem in the pipeline when using the above extended instructions, and these instructions will not be blocked.Therefore, when writing programs using assembly instructions, you should try to use the extension instructions above.If you do need to use HI/LO results, such as high 64 bits, after 64-bit multiplication, then the programmer should be aware of the impact of using the standard MIPS multiplication instruction on the pipeline.

Fixed-point operations use floating point data paths:

Floating point data paths are often idle during the execution of fixed-point programs, and these instructions give us the opportunity to take advantage of them to further increase the degree of instruction parallelism.

8.3 The compiler USES prompts

The open source compiler suite GCC currently supports the loongson 3A1000 processor architecture tuning option.In version GCC4.6.0 and above, -march=loongson3A can be used to describe the pipeline of the processor for code scheduling, and the generated code can also make full use of the instruction expansion of loongson 3A1000.In most cases, this option provides a performance boost on the loongson 3A1000 processor.

In addition, during our space exploration of the compiler tuning of the SPEC CPU2000 benchmark set, we concluded that the following GCC compilation options might improve the performance of the godson 3A1000 processor.In the process of fine tuning the program, the following options have some reference significance.

- fdefer - pop	- fcaller - saves
- fno - move - loop invariants	- fno - cprop - registers
- funroll - all - loops	- fno - early - inlining
- ffunction - cse	Floop - optimize
- fno - optimize - register - a move	- fno - peephole
- freorder - blocks	- fno - peephole2
- ftracer	- fprefetch - loop - arrays
- ftree - MAC	- fsched - spec - load - the dangerous
- fno - cse - follow - to	- fschedule - insns2
Fno - math -- errno	- fsignaling - nans
- fno - optimize - (- calls	- fno - strength - reduce
- fno - peel - loops	- fthread - to
- fsingle - precision - constant	- fno - tree - copyrename
- ftree - loop - optimize	- ftree - dominator - opts
- fno - branch - count - reg	- ftree - vect - loop - version

8.4 Instruction stream

Loongson 3A1000 is a multi-emission highly parallel processor. The processing of an essentially serial instruction stream may have a significant impact on program performance. This section discusses issues such as instruction alignment, transfer instructions, and instruction scheduling.

8.4.1 Instruction aligned

In one cycle, loongson 3A1000 can fetch four instructions from a single cached line, but the four instructions cannot cross the boundaries of the cached line.We should properly align the basic

blocks that are routinely executed to avoid crossing the boundaries of the cache line.In addition, if there are transfer instructions in four instructions taken at one time, the efficiency of fetch instruction will also be affected.If the first is a transfer branch instruction, and the transfer prediction is successful, the last two instructions will be discarded.

If the last item is a transfer instruction, even if the transfer is successful, the processor will have to take down another cache line to get the instruction in its delay slot.Loongson 3A1000 can only decode one transfer instruction in one cycle. If there are two transfer instructions in a bundle of instructions, it will need two cycles to complete the decoding, that is to say, the pointing part will be blocked for one cycle.

8.4.2 The processing of transfer instructions

In the loongson 3A1000 processor, an unexpected change in the instruction flow address can waste the time of about 10 instructions."Surprise" can be caused by instructions that transfer successfully, or it can be caused by transfer prediction errors. For the current loongson 3A1000, even a correctly predicted and predicted transfer is slower than sequential code, and it wastes a cycle because the transfer target cache (BTB) cannot give the next correct program counter PC value for a normal conditional transfer.

The compiler can reduce the overhead of the transfer instruction by:

The loongson 3A1000's transfer instruction prediction method is different from other high-performance processors, and different versions have some slight differences.Based on profiling, the compiler can rearrange the code location based on the actual transfer frequency, resulting in a better prediction.

Make the base block as large as possible. A good optimization result is to have an average of 20 instructions between two successful transfers.In order to have at least 20 instructions between them, this requires a loop expansion and the direct inlining of subroutines with fewer than 20 instructions.Loongson 3A1000 implements conditional movement instructions, which can be used to reduce the number of branch instructions.Reorganizing the code by performing profiling also helps with this optimization.

On loongson 3A1000, different transfer instructions make predictions in different ways:

Static prediction

For the likely class transfer instruction and the direct jump instruction. G - share predictor

A 9-bit global history register, GHR, and a schema history table, PHT, with 4K entries.For conditions

Turn the finger. BTB (transfer target cache)

There is a fully associative cache of 16 entries.Used to predict the target address of a register jump instruction.RAS (return address stack)

Four items that are used to predict the target address returned by the function.

Here are some things to note about software:

Special care needs to be taken with the likely class transfer instruction on the loongson 3A1000 processor.Although the likely class transitions

Instructions may be effective for simple static scheduling of sequential scalar processors, but they are not as effective for modern high-performance processors.Because the transfer prediction hardware of modern high-performance processors is relatively complex, they usually have a correct prediction rate of more than 90%.(for example, loongson 3A1000 can correctly predict the direction of a conditional transfer from 85% to 100%, with an average of 95%.) in this case, the compiler should not use the likely class transfer command, which is not too predictive.In fact, we found that GCC with the -mno-branch-likely option generally works better.

The decoding unit is divided into three stream segments, in which the transferred target address is calculated in the third stage.A successful transfer instruction will result in a pause of two cycles. That is to say, if a transfer instruction is taken out at cycle 0, the address of PC+16 is taken out at cycle 1, and the address of PC+32 is taken out at cycle 2, the target address of the transfer instruction will be taken out at cycle 3.So it would be helpful to reduce the number of successful transfer instructions.

The BTB in the godson 3A1000 is only used for register jump instructions (no jr instructions except jr31 and jalr).

A four-item RAS is used to predict the target address of the jr31 directive.The predictive validity of the function return depends on the software that USES the jr31 instruction as the function return instruction.

8.4.3 Increase of instruction flow density

The compiler should make the most of profiling to ensure that the bytes that are called into the instruction Cache are executed.This requires that the target address of the jump instruction be aligned and that code that is rarely executed be removed from the Cache line.

8.4.4 Instruction scheduling

Loongson 3A1000 has a relatively large instruction window for dynamic instruction scheduling. However, due to the limited resources in the processor, the compiler can assist the processor in better scheduling to some extent.Modern compilers (such as GCC) have instruction scheduling support that puts loongson 3A1000's internal component resources and points to

The delay of the let (see table 1) tells the compiler that it can schedule better.

8.5 Memory access

The execution of Load-store instruction has a great influence on the performance of the whole system. These instructions can be executed quickly if the level 1 data cache contains the required content.If the data is only in the secondary cache, it is slightly slower, and if it is only in main memory, there will be a large delay.However, out-of-order execution and non-blocking caches can reduce the performance penalty of these delays.

The loongson 3A1000 consists of four on-chip second-level cache modules, each of which is 1MB in size, for a total of 4MB.Each module is organized as a four-way group. Loongson 3A1000 has a built-in DDR memory controller that minimizes

Memory access latency. The memory frequency and processor operating frequency of loongson 3A1000 system can be configured separately, so as to increase the memory frequency as much as possible and reduce the gap with the processing frequency, which is very conducive to improving the performance of most applications.More information about the memory controller can be found in the relevant section of the 3A1000 processor documentation.

Loongson 3A1000 provides the prefetch instruction, which can be loaded into the zero-point register to prefetch data to the first-level data Cache.In addition, the DSP engine in loong chip 3 can prefetch the data in memory or IO into the second-level Cache. For details, please refer to the relevant parts of the manual.

The compiler should minimize unnecessary storage access. The current logodson 3A1000 processor has a large memory instruction delay (four cycles even for a cache hit), and the instruction window is not large enough to tolerate dozens of cycles of access delay.

The software also pays special attention to data alignment. Aggregates (arrays, some records, subroutine stack frames) should be allocated on the aligned cache line boundaries so that the cache lines can be aligned to the data path and the number of fast cache lines can be reduced.A compile time warning should be generated for items in aggregates (records, normal blocks) that are forced out of alignment (such as GCC's packed property).In loongson 3A1000, normal load/store instructions have alignment requirements, and instructions that do not meet the requirements should be implemented through kernel simulation.For example, taking a word (four bytes) from a non-four-byte aligned address triggers an exception that is handled by the operating system;It usually takes thousands of processor cycles for an operating system to accomplish this task.So the user needs to know that these warnings represent code that may have poor performance. The code that compiles the parameters defaults to the alignment of the parameters.Scalars that are frequently used should reside in registers.

8.6 Other tips

Use all floating point registers. Although the O32 ABI only has 16 available for users, the loongson 3A1000

32 64-bit floating point registers are provided.Using the N32 or N64 ABI can help take advantage of processor performance. Use performance counters. The loongson 3A1000 performance counter can be used to monitor the real-time performance parameters of the

program.compile

Programmers and software developers can analyze the results to improve their code.