LOONGSON

龙芯 3A2000/3B2000 处理器用户手册

下册

GS464E 处理器核 V1.03

2016 年 6 月

龙芯中科技术有限公司

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

版权声明

免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

龙芯中科技术有限公司

Loongson Technology Corporation Limited
地址：北京市海淀区中关村环保科技示范园龙芯产业园 2 号楼
Building No.2, Loongson Industrial Park,
Zhongguancun Environmental Protection Park, Haidian District, Beijing
电话(Tel)：010-62546668
传真(Fax)：010-62600826

阅读指南

《龙芯 3A2000/3B2000 处理器用户手册》分为上、下两册。

《龙芯 3A2000/3B2000 处理器用户手册》下册，从系统软件开发者角度详细介绍龙芯 3A2000/3B2000 处理器所采用的 GS464E 高性能处理器核。

特殊格式含义介绍

1、本文档中涉及 CP0 控制寄存器某个域的描述时，采用 Reg.Field 的格式，其中 Reg 是该控制寄存器的助记符，Field 是该寄控制存器中待描述域的助记符。例如，EBase.CPUNum 表示 EBase 控制寄存器的 CPUNum 域。

2、本文档中涉及数据内容截取部分的描述，采用[m:n]或 $_{m..n}$ 的格式，表示选取待截取内容的第 n 位至第 m 位。m，n 从 0 开始取值，m≥n。

版本历史

文档更新记录		
	文档名:	龙芯 3A2000/3B2000 处理器用户手册 ——下册
	版本号	V1.03
	创建人:	研发中心芯片研发部
	创建日期 ：	2017-04-07

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2016/03/01	芯片研发部	V1.0	初稿完成。
2	2016/03/29	芯片研发部	V1.01	1. 更新 GSCONFIG 寄存器第 8 位的定义，见 7.34 小节。 2. 新增 2.4.12 小节，关于 DI、EI 指令兼容性说明。
3	2016/06/08	芯片研发部	V1.02	更新 2.指令集概述
4	2017/04/07	芯片研发部	V1.03	1. 删除 8.2.1 节中部分不准确的性能计数事件及调整部分性能计数事件的描述。 2. 调账 2.4.6 小节中关于 SYNC 指令的描述。 3. 调整 2.4.7 小节中关于 SYNCI 指令的描述。

手册信息反馈: service@loongson.cn
也可通过问题反馈网站 http://bugs.loongnix.org/向我司提交芯片产品使用过程中的问题，并获取技术支持。

目 录

图目录

表目录

1 处理器核结构概述

龙芯 GS464E 处理器核（后文简称“GS464E”）是一款实现 LoongsonISA 指令集的通用 RISC 处理器核，是龙芯 GS464 处理器核的性能优化升级版本。GS464E 的指令流水线每个时钟周期取四条指令进行译码，并且动态地发射到六个全流水的功能部件中。指令在保证依赖关系的前提下可以乱序发射执行，所有指令按照程序中的顺序进行提交以保证精确例外和访存顺序执行。

多发射深流水处理器中指令相关和数据相关是影响性能的首要因素，为此 GS464E 采用乱序执行技术和激进的存储系统设计来提高流水线的效率。

乱序执行技术包括寄存器重命名技术、动态调度技术和转移预测技术。寄存器重命名解决 WAR（读后写）和 WAW（写后写）相关，并用于例外和错误转移预测引起的精确现场恢复。GS464E 通过两个 128 项的物理寄存器堆分别对定点和浮点寄存器进行重命名，同时分别采用 16 项、32 项和 32 项物理寄存器堆对 HI/LO 寄存器、DSP Control 寄存器和浮点控制寄存器进行重命名。动态调度根据指令操作数准备好的次序而不是指令在程序中出现的次序来执行指令，减少了 RAW（写后读）相关引起的阻塞。GS464E 采用一个 16 项的定点保留站、一个 24 项的浮点保留站和一个 32 项的访存保留站用于乱序发射，并通过一个 128 项的 Reorder 队列（简称 ROQ）实现乱序执行的指令按照程序的次序提交。转移预测通过预测转移指令是否成功跳转来减少由于控制相关引起的阻塞。GS464E 使用 8K 项的全局转移历史表（Global Branch History Table，简称 GBHT），8K 项的局部转移历史表（Local Branch History Table，简称 LBHT），8K 项的全局选择历史表（Global Branch Select Table，简称 GBSEL），13 位的全局历史寄存器（Global Histiry Registor，简称 GHR），1K 项的转移目标地址缓冲器（Branch Target Buffer，简称 BTB）和 16 项的返回地址栈（Return Address Stack，简称 RAS）进行转移预测，所有分支指令使用一个 24 项 Branch 队列（简称 BRQ）实现分支指令误预测时后续指令的准确撤销。

GS464E 先进的存储系统设计可以有效地提高流水线的效率。GS464E 包含两个全功能的访存部件。每个访存部件可以独立地流水地执行 Load 和 Store 操作。GS464E 通过 64 项的访存队列（CP0 Queue）来动态地解决地址依赖，实现访存操作的乱序执行和非阻塞 Cache。GS464E 采用三级 Cache 存储结构，其中一级 Cache 由 64KB 的指令 Cache 和 64KB 的数据 Cache 组成，均采用 64 字节长度 Cache 行和四路组相联结构，每个处理器核包含私有的 256KB 二级指令数据共享 Cache，采用 64 字节长度 Cache 行和 16 路组相联结构。每四个核共享 4MB 三级 Cache。GS464E 采用两级 TLB 结构，其中一级 TLB 分为 64 项全相联指令 TLB（简称为 ITLB）和 32 项全相联数据 TLB（简称为 DTLB），二级 TLB 包含一个 64 项全相联的可变页大小的 TLB（简称 VTLB）和一个 1024 项 8 路组相联结构的固定页大小 TLB（简称 FTLB），TLB 每项可以映射一个奇页和一个偶页，页大小在 4KB 到 1GB 之间可变。

GS464E 有两个全功能的定点功能部件和两个全功能的浮点功能部件。每个定点部件都可以执行分支指令，并全流水地执行定点乘法操作和所有 DSP 操作。每个浮点部件都可以全流水地执行 64 位双精度浮点乘加操作，并通过浮点指令的 fmt 域的扩展执行 32 位和 64 位的定点指令。

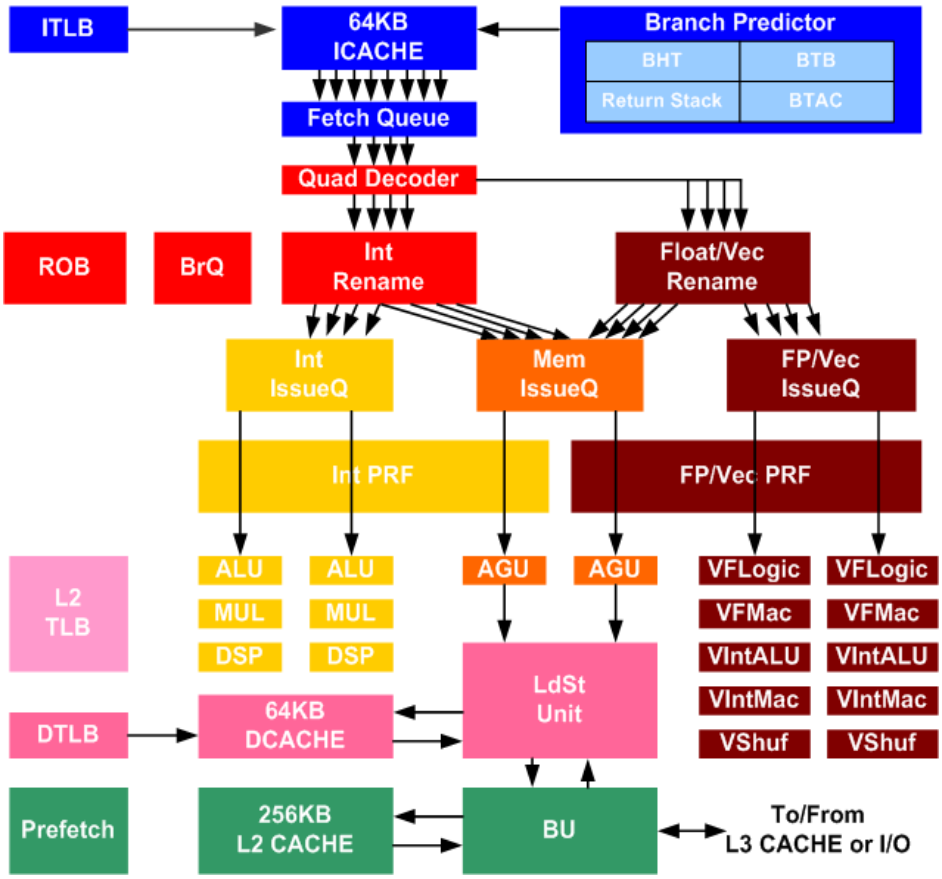
GS464E 支持 MIPS 公司的 EJTAG 调试规范，采用标准的 AXI 接口，其指令 Cache 实现了奇偶校验，数据 Cache 实现了 ECC 校验。

GS464E 的基本流水线包括 PC、取指、预译码、译码 I、译码 II、寄存器重命名、调度、发射、读寄存器、执行、提交 I、提交 II 等 9 级，每一级流水包括如下操作。

- PC 流水级用于生成下一拍取指所需的程序计数器 PC 值。
- 取指流水级用程序计数器 PC 的值去访问指令 Cache 和指令 TLB，如果指令 Cache 和指令 TLB 都命中，则把八条新的指令取到指令寄存器 IR。

- 预译码流水级主要对转移指令进行译码并预测跳转的方向。
- 译码 I 流水级将预译码结果存入指令队列。
- 译码 II 流水级把 IR 中的四条指令转换成处理器的内部指令格式送往寄存器重命名模块。
- 寄存器重命名流水级为逻辑目标寄存器分配一个新的物理寄存器，并将逻辑源寄存器映射到最近分配给该逻辑寄存器的物理寄存器。
- 调度流水级将重命名的指令分配到定点或浮点保留站中等待执行，同时送到 ROQ 中用于执行后的顺序提交；此外，转移指令和访存指令还分别被送往转移队列和访存队列。
- 发射流水级从定点或浮点保留站中为每个功能部件选出一条所有操作数都准备好的指令；在重命名时操作数没准备好的指令，通过侦听结果总线和 forward 总线等待它的操作数准备好。
- 读寄存器流水级为发射的指令从物理寄存器堆中读取相应的源操作数送到相应的功能部件。
- 执行流水级根据指令的类型执行指令并把计算结果写回寄存器堆；结果总线还送往保留站和寄存器重命名表，通知相应的寄存器值已经可以使用了。
- 提交 I 流水级按照 Reorder 队列记录的程序的顺序在已经执行完的指令中选择选择出可以提交的指令，GS464E 最多每拍可以提交四条指令。
- 提交 II 流水级将选择出的提交指令送往寄存器重命名表用于确认它的目的寄存器的重命名关系并释放原来分配给同一逻辑寄存器的物理寄存器，并送往访存队列允许那些提交的存数指令写入 Cache 或内存。

上述是基本指令的流水级，对于一些较复杂的指令，如定点乘除法指令、浮点指令以及访存指令，在执行阶段需要多拍。GS464E 的基本结构如下图所示。



2 指令集概述

GS464E 实现了龙芯指令集（LoongISA）v1.00 版中的 MIPS64 兼容基础部分、MIPS64 DSP 指令（MIPS64 DSP Module）以及 64 位龙芯通用扩展指令（LoongEXT64）。

2.1 MIPS64 兼容通用指令列表

GS464E 实现了所有 MIPS64 规范中所要求必须实现和少量可选实现的通用指令，其实现的 MIPS64 兼容通用指令按功能划分包括如下几类：

- 访存指令
- 运算指令
- 转移指令
- 其它指令
- 协处理器 0 指令

下面对这些指令逐类予以列举。

2.1.1 访存指令

MIPS 体系结构采用 load/store 架构。所有运算都在寄存器上进行，只有访存指令可对主存中的数据进行访问。访存指令包含各种宽度数据的读写、无符号读、非对齐访存和原子访存等。

表 2-1 CPU 指令集：访存指令

指令助记符	指令功能简述	ISA 兼容类别
LB	取字节	MIPS32
LBU	取无符号字节	MIPS32
LH	取半字	MIPS32
LHU	取无符号半字	MIPS32
LW	取字	MIPS32
LWU	取无符号字	MIPS32
LWL	取字左部	MIPS32
LWR	取字右部	MIPS32
LD	取双字	MIPS64
LDL	取双字左部	MIPS64
LDR	取双字右部	MIPS64
LL	取标志处地址	MIPS32
LLD	取标志处双字地址	MIPS64
SB	存字节	MIPS32
SH	存半字	MIPS32
SW	存字	MIPS32
SWL	存字左部	MIPS32
SWR	存字右部	MIPS32

指令助记符	指令功能简述	ISA 兼容类别
SD	存双字	MIPS64
SDL	存双字左部	MIPS64
SDR	存双字右部	MIPS64
SC	满足条件下存	MIPS32
SCD	满足条件下存双字	MIPS64

2.1.2 运算指令

运算型指令完成寄存器值的算术、逻辑、移位、乘法和除法等操作。运算型指令包含了寄存器指令格式（R-型，操作数和运算结果均保存在寄存器中）和立即数指令格式（I-型，其中一个操作数为一个 16 位的立即数）

表 2-2 运算指令：算术指令（ALU 立即数）

指令助记符	指令功能简述	ISA 兼容类别
ADDI	加立即数	MIPS32
DADDI	加双字立即数	MIPS64
ADDIU	加无符号立即数	MIPS32
DADDIU	加无符号双字立即数	MIPS64
SLTI	小于立即数设置	MIPS32
SLTIU	无符号小于立即数设置	MIPS32
ANDI	与立即数	MIPS32
ORI	或立即数	MIPS32
XORI	异或立即数	MIPS32
LUI	取立即数到高位	MIPS32

表 2-3 运算指令：算术指令（3 操作数）

指令助记符	指令功能简述	ISA 兼容类别
ADD	加	MIPS32
DADD	双字加	MIPS64
ADDU	无符号加	MIPS32
DADDU	无符号双字加	MIPS64
SUB	减	MIPS32
DSUB	双字减	MIPS64
SUBU	无符号减	MIPS32
DSUBU	无符号双字减	MIPS64
SLT	小于设置	MIPS32
SLTU	无符号小于设置	MIPS32
AND	与	MIPS32
OR	或	MIPS32
XOR	异或	MIPS32
NOR	或非	MIPS32

表 2-4 运算指令：算术指令（2 操作数）

指令助记符	指令功能简述	ISA 兼容类别
CLO	字前导 1 个数	MIPS32
DCLO	双字前导 1 个数	MIPS64
CLZ	字前导 0 个数	MIPS32
DCLZ	双字前导 0 个数	MIPS64
WSBH	半字中字节交换	MIPS32 R2
DSHD	字中半字交换	MIPS64 R2
DSBH	半字中字节交换	MIPS64 R2
SEB	字节符号扩展	MIPS32 R2
SEH	半字符号扩展	MIPS32 R2
INS	位插入	MIPS32 R2
EXT	位提取	MIPS32 R2
DINS	双字位插入	MIPS64 R2
DINSM	双字位插入	MIPS64 R2
DINSU	双字位插入	MIPS64 R2
DEXT	双字位提取	MIPS64 R2
DEXTM	双字位提取	MIPS64 R2
DEXTU	双字位提取	MIPS64 R2

表 2-5 运算指令：乘、除法指令

指令助记符	指令功能简述	ISA 兼容类别
MUL	乘到通用寄存器	MIPS32
MULT	乘	MIPS32
DMULT	双字乘	MIPS64
MULTU	无符号乘	MIPS32
DMULTU	无符号双字乘	MIPS64
MADD	乘加	MIPS32
MADDU	无符号乘加	MIPS32
MSUB	乘减	MIPS32
MSUBU	无符号乘减	MIPS32
DIV	除	MIPS32
DDIV	双字除	MIPS64
DIVU	无符号除	MIPS32
DDIVU	无符号双字除	MIPS64
MFHI	从 HI 寄存器取数到通用寄存器	MIPS32
MTHI	从通用寄存器存数到 HI 寄存器	MIPS32
MFLO	从 LO 寄存器取数到通用寄存器	MIPS32
MTLO	从通用寄存器存数到 LO 寄存器	MIPS32

表 2-6 运算指令：移位指令

指令助记符	指令功能简述	ISA 兼容类别
SLL	逻辑左移	MIPS32
SRL	逻辑右移	MIPS32
SRA	算术右移	MIPS32
SLLV	可变的逻辑左移	MIPS32
SRLV	可变的逻辑右移	MIPS32
SRAV	可变的算术右移	MIPS32
ROTR	循环右移	MIPS32 R2
ROTRV	可变的循环右移	MIPS32 R2
DSLL	双字逻辑左移	MIPS64
DSRL	双字逻辑右移	MIPS64
DSRA	双字算术右移	MIPS64
DSLLV	可变的双字逻辑左移	MIPS64
DSRLV	可变的双字逻辑右移	MIPS64
DSRAV	可变的双字算术右移	MIPS64
DSLL32	移位量加 32 的双字逻辑左移	MIPS64
DSRL32	移位量加 32 的双字逻辑右移	MIPS64
DSRA32	移位量加 32 的双字算术右移	MIPS64
DROTR	双字循环右移	MIPS64 R2
DROTR32	移位量加 32 的双字循环右移	MIPS64 R2
DROTRV	双字可变的循环右移	MIPS64 R2

2.1.3 跳转和分支指令

跳转和分支指令可改变程序的控制流，包括以下四种类型：

- PC 相对条件分支
- PC 无条件跳转
- 寄存器绝对跳转
- 过程调用

MIPS 架构中，所有转移指令后都紧跟一条延迟槽指令。Likely 转移指令的延迟槽只在转移成功时执行，非 Likely 转移指令延迟槽指令总会得到执行。过程调用指令的返回地址默认保存在第 31 号寄存器中，根据第 31 号寄存器跳转将被认为从被调用过程返回。

表 2-7 跳转和分支指令

指令助记符	指令功能简述	ISA 兼容类别
J	跳转	MIPS32
JAL	立即数调用过程	MIPS32
JR	跳转到寄存器指向的指令	MIPS32
JR.HB	跳转到寄存器指向的指令	MIPS32 R2
JALR	寄存器调用过程	MIPS32
JALR.HB	寄存器调用过程	MIPS32 R2
BEQ	相等则跳转	MIPS32
BNE	不等则跳转	MIPS32

指令助记符	指令功能简述	ISA 兼容类别
BLEZ	小于等于 0 跳转	MIPS32
BGTZ	大于 0 跳转	MIPS32
BLTZ	小于 0 跳转	MIPS32
BGEZ	大于或等于 0 跳转	MIPS32
BLTZAL	小于 0 调用过程	MIPS32
BGEZAL	大于或等于 0 调用过程	MIPS32
BEQL	相等则 Likely 跳转	MIPS32
BNEL	不等则 Likely 跳转	MIPS32
BLEZL	小于或等于 0 则 Likely 跳转	MIPS32
BGTZL	大于 0 则 Likely 跳转	MIPS32
BLTZL	小于 0 则 Likely 跳转	MIPS32
BGEZL	大于或等于 0 则 Likely 跳转	MIPS32
BLTZALL	小于 0 则 Likely 调用过程	MIPS32
BGEZALL	大于或等于 0 则 Likely 调用过程	MIPS32

2.1.4 协处理器 0 指令

处理器通过 0 号协处理器（CP0）寄存器来管理内存和处理异常。

表 2-8 协处理器 0 指令

指令助记符	指令功能简述	ISA 兼容类别
DMFC0	从 CP0 寄存器取双字	MIPS32
DMTC0	往 CP0 寄存器写双字	MIPS32
MFC0	从 CP0 寄存器取字	MIPS32
MTC0	往 CP0 寄存器写字	MIPS32
TLBR	读索引的 TLB 项	MIPS32
TLBWI	写索引的 TLB 项	MIPS32
TLBWR	写随机的 TLB 项	MIPS32
TLBP	在 TLB 中搜索匹配项	MIPS32
CACHE	Cache 操作	MIPS32
ERET	异常返回	MIPS32
DI[1]	禁止中断	MIPS32 R2
EI[2]	允许中断	MIPS32 R2

2.1.5 其它指令

MIPS64 中，除了前面列出上述指令外还有其它一些指令。

表 2-9 其它指令：特殊指令

指令助记符	指令功能简述	ISA 兼容类别
SYSCALL	系统调用	MIPS32
BREAK	断点	MIPS32

[1] 详见 2.4.12 小节说明
[2] 详见 2.4.12 小节说明

指令助记符	指令功能简述	ISA 兼容类别
SYNC	同步	MIPS32
SYNCI	同步指令缓存	MIPS32 R2

表 2-10 其它指令：例外陷入指令

指令助记符	指令功能简述	ISA 兼容类别
TGE	大于或等于陷入	MIPS32
TGEU	无符号数大于或等于陷入	MIPS32
TLT	小于陷入	MIPS32
TLTU	无符号数小于陷入	MIPS32
TEQ	等于陷入	MIPS32
TNE	不等陷入	MIPS32
TGEI	大于或等于立即数陷入	MIPS32
TGEIU	大于或等于无符号立即数陷入	MIPS32
TLTI	小于立即数陷入	MIPS32
TLTIU	小于无符号立即数陷入	MIPS32
TEQI	等于立即数陷入	MIPS32
TNEI	不等于立即数陷入	MIPS32

表 2-11 其它指令：条件移动指令

指令助记符	指令功能简述	ISA 兼容类别
MOVF	条件移动当浮点条件假	MIPS32
MOVT	条件移动当浮点条件真	MIPS32
MOVN	条件移动当通用寄存器非 0	MIPS32
MOVZ	条件移动当通用寄存器为 0	MIPS32

表 2-12 其它指令：空操作

指令助记符	指令功能简述	ISA 兼容类别
PREF	预取指令	MIPS32
PREFX	预取指令	MIPS32
NOP	空操作	MIPS32
SSNOP	单发射空操作	MIPS32

2.2 MIPS64 兼容浮点指令集概览

GS464E 实现的浮点指令兼容 MIPS64 规范[1]，所有浮点指令在浮点协处理器（Floting Point Unit，简称 FPU）中实现。

[1] 并非完全兼容，当 Status.FR=0 时浮点寄存器的格式与 MIPS64 规范(r1~r5)并不一致，详细情况请参看 2.2.2 小节。

2.2.1 FPU 数据类型

浮点协处理器既支持浮点数据类型也支持定点数据类型。

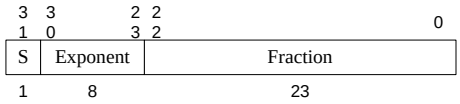
浮点数据类型

浮点协处理器支持的浮点数据类型有：

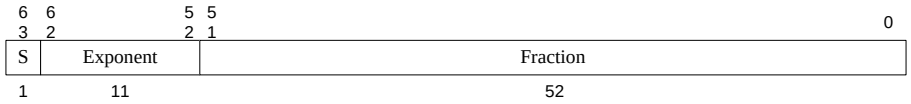
- 32 比特的单精度浮点数（Single-Precisions，S）
- 64 比特的双精度浮点数（Double-Precisions，D）
- 64 比特的成对单精度浮点数（Paired Single-Precisions，PS）

浮点协处理器对单精度浮点数（包括成对单精度浮点数中的每个单精度数）和双精度浮点数的操作符合 ANSI/IEEE 754－1985 二进制浮点运算标准。32 比特的单精度格式包括一个 24 比特的以“符号+幅度”表示的小数域（S＋F）和一个 8 比特的指数域（E）；64 位的双精度格式包括一个 53 比特的“符号+幅度”表示的小数域（S＋F）和一个 11 比特的指数域（E）；64 位双精度（PS）格式包含两个单精度浮点格式。三种类型的数据格式如图 2-1 所示。

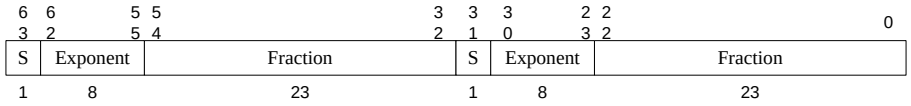
图 2-1 FPU 浮点数据格式



单精度浮点数 (S)



双精度浮点数 (D)



成对单精度浮点数 (PS)

浮点数的格式由以下三个域组成：

- 符号域，S
- 带偏移的指数域，e = E + Bias，E 是不带偏移的指数
- 小数域，F = .$b_1b_2…b_{p-1}$

不带偏移的指数 E 的范围是包括 Emin 和 Emax 在内的所有二者之间的整数，另外再加上以下两个保留值：

- Emin - 1 （用来编码 0 和非规格化数）
- Emax +1 （用来编码∞和 NaN[Not a Number]）

表 2-13 定义了与浮点格式相关的参数的值。

表 2-13 浮点数格式相关参数

参数	单精度	双精度
Emax	+127	+1023

参数	单精度	双精度
Emin	-126	-1022
指数偏移量	+127	+1023
指数位宽	8	11
小数位宽	24	53

对于单精度或者双精度格式来说，每一个可表示的非 0 数都有唯一一种编码与之对应。其编码所对应的数值 V 的计算方式如表 2-14 所示。

表 2-14 浮点数数值 V 的计算方式

E	F	S	b1	V	
$E_{max}+1$	≠0	x	1	SNaN(Signaling NaN)	
		x	0	QNaN(Quiet NaN)	
$E_{max}+1$	0	1	x	-∞	负无穷
		0	x	+∞	正无穷
$[E_{min}, E_{max}]$	x	1	x	$-(2^{E})(1.F)$	负规格化数
		0	x	$+(2^{E})(1.F)$	正规格化数
$E_{min}-1$	≠0	1	x	$-(2^{Emin-1})(0.F)$	负非规格化数
		0	x	$+(2^{Emin-1})(0.F)$	正非规格化数
$E_{min}-1$	0	1	x	-0	负 0
		0	x	+0	正 0

两种类型浮点数的最大值和最小值在表 2-15 中给出。

表 2-15 浮点数最大值和最小值

类型	单精度	双精度
最小数	1.40129846e–45	4.9406564584124654e–324
最小规格化数	1.17549435e–38	2.2250738585072014e–308
最大数	3.40282347e+38	1.7976931348623157e+308

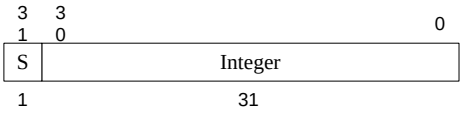
定点数据类型

FPU 支持的定点数据均为有符号整数，分为两种：

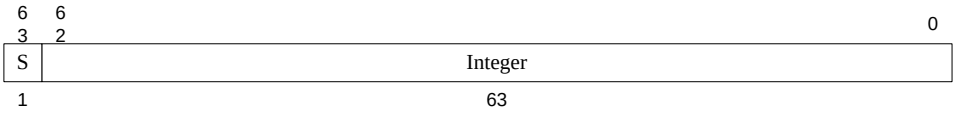
- 32 比特的有符号字整型（Word，W）
- 64 比特的有符号长字整型（Longword，L）

两种定点数据类型的格式如图 2-2 所示。

图 2-2 FPU 定点数据格式



有符号字整型 (W)



有符号长字整型 (L)

2.2.2 浮点寄存器

GS464E 中的浮点寄存器沿袭 MIPS R4000/R10000 处理器用法，与 MIPS64 规范略有不同。在 Status 控制寄存器的 FR 位为 0 时，GS464E 只有 16 个 32 位或 64 位的浮点寄存器，且浮点寄存器号必须为偶数；而 MIPS64 表示有 32 个 32 位的浮点寄存器或 16 个 64 位的浮点寄存器。当 Status 控制寄存器的 FR 位为 1 时，GS464E 使用浮点寄存器的方法与 MIPS64 规范一致，均有 32 个 64 位的浮点寄存器。

2.2.3 浮点控制寄存器

GS464E 中的浮点控制寄存器包括：

- FIR，浮点实现定义寄存器
- FCCR，浮点条件码寄存器
- FEXR，浮点例外寄存器
- FENR，浮点使能寄存器
- FCSR，浮点控制/状态寄存器（常称之为 FCR31）

访问浮点控制寄存器不需要处于核心态。软件通过 CFC1 和 CTC1 指令访问浮点控制寄存器。在上述浮点控制寄存器中，访问 FCCR、FEXR 和 FENR 寄存器实际上访问了 FCSR 的某些域。

浮点实现定义寄存器 (FIR, CP1 Control Register 0)

浮点实现定义寄存器是一个 32 位的只读寄存器，包含了浮点单元实现的功能，如处理器 ID，修订版本号等信息。

图 2-3 说明了 FIR 寄存器的格式；表 2-16 对 FIR 寄存器各域进行了描述。

图 2-3 FIR 寄存器格式

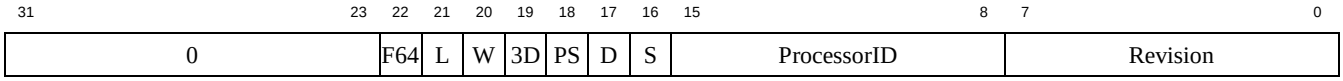


表 2-16 FIR 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..23	只读恒为 0。	0	0
F64	22	恒为 1，表示浮点数据通路是 64 位。	R	0x1
L	21	恒为 1，表示实现了长字(L)定点数据类型。	R	0x1

域名称	位	功能描述	读/写	复位值
W	20	恒为 1，表示实现了字(W)定点数据类型。	R	0x1
3D	19	恒为 0，表示未实现 MIPS 3D ASE。	R	0x0
PS	18	恒为 1，表示实现了成对单精度浮点数据类型。	R	0x1
D	17	恒为 1，表示实现了双精度浮点数据类型。	R	0x1
S	16	恒为 1，表示实现了单精度浮点数据类型。	R	0x1
ProccsorID	15..8	浮点协处理器标识号。	R	0x05
Revision	7..0	浮点协处理器版本号。	R	0x01

浮点控制与状态寄存器 (FCSR, CP1 Control Register 31)

FCSR 寄存器用于控制浮点单元的操作和表示一些状态。

图 2-4 说明了 FIR 寄存器的格式；表 2-17 对 FIR 寄存器各域进行了描述。

图 2-4 FCSR 寄存器格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FCC							FS	FCC	0	TOP	0	0	0	Cause						Enables					Flags					RM		
7	6	5	4	3	2	1		0						E	V	Z	O	U	I	V	Z	O	U	I	V	Z	O	U	I		

表 2-17 FCSR 寄存器域描述

域名称	位	功能描述	读/写	复位值
FCC	31..25, 23	浮点条件码。记录浮点比较结果，用于条件跳转或转移。当一个浮点比较操作发生时，结果被保存在指定的 CC 位，即条件位。如果比较结果为真，则 CC 位被置 1；反之则置 0。	R/W	0x0
FS	24	刷至 0 标识位。当置为 1 时，非规格化结果被置为 0；否则结果出现非规格化数将触发未实现操作例外。	R/W	0x0
0	22	只读恒为 0。	0	0
TOP	21	浮点寄存器 TOP 模式控制位。该位为 1 时，表示浮点指令的寄存器采用 TOP 模式的编码方式；该位为 0 时，表示浮点指令的寄存器仍采用普通的编码方式。	R/W	0x0
0	20..18	只读恒为 0。	0	0
Cause	17..12	这些位反映了最近执行指令的结果。Causes 域是协处理器 0 的 Cause 寄存器的一个逻辑扩充，这些位指示了由上次浮点操作所引起的例外，并且如果相应的使能位（Enable）被设置的话则产生一个中断或者例外。如果一条指令中产生不只一个例外，每一个相应的例外导致位都要被设置。 Causes 域能被每条浮点操作指令所重写（不包括 Load、Store、Move 操作）。其中如果需要软件仿真来完成的则把该操作的未实现操作位（E）置 1，否则保持为 0。其它位则依照 IEEE754 标准看是否相应的例外产生而分别置 1 或者置 0。当一个浮点例外发生，没有结果将被存储，状态唯一受影响的就是 Causes 域。	R/W	0x0

域名称	位	功能描述	读/写	复位值
Enables	11..7	任何时候当 Cause 位和相应的使能位（Enable）同时为 1 时，会产生一个浮点例外。如果浮点操作设置了一个被允许激活（相应使能位为 1）的 Cause 位，则处理器会立即产生一个例外，这和用 CTC1 指令同时设置 Cause 位和 Enable 位为 1 的效果一样。 对于未实现操作(E)来说没有相应的使能位，如果设置了未实现操作，它总是会产生一个浮点例外。 在从一个浮点例外返回之前，软件首先必须用一个 CTC1 指令来清除被激活了的 Cause 位以防止中断的重复执行。因此，在用户态下的运行的程序永远不会观察到被使能的 Cause 位的值为 1；如果用户态的处理程序需要获得该信息，则 Cause 位的内容必须被传递到其它地方而不是在状态寄存器中。 如果浮点操作只设置未被使能（相应的使能位为 0）的 Cause 位，则没有例外发生，同时 IEEE754 标准定义的默认结果被写回。在这种情况下，前一条浮点指令所引起的例外能够通过读 Causes 域的值来确定。	R/W	0x0
Flags	6..2	标志位是累积的，它指示自从上次被明确重置后发生了例外。如果一个 IEEE754 例外被产生，那么相应的 Flag 位被置 1，否则保持不变，因此对于浮点运算来说这些位永不会被清除。但是我们可以通过 CTC1 控制指令写一个新值到状态寄存器中来实现对 Flag 位的设置或清除。	R/W	0x0
RM	1..0	舍入模式控制域。表 2-18 对 RM 的编码做进一步描述。	R/W	0x0

表 2-18 舍入模式(RM)编码

舍入模式 (RM)	助记符	描述
0	RN	把结果向最接近可表示数的方向舍入，当两个最接近可表示数离结果一样接近时，则向最低位为 0 的那个最接近数方向舍入。
1	RZ	向 0 方向舍入：把结果向与之最接近并且在绝对值上不大于它的那个数舍入。
2	RP	向正无穷大方向舍入:把结果向与之最接近并且不小于它的那个数舍入。
3	RM	向负无穷大方向舍入：把结果向与之最接近并且不大于它的那个数舍入。

浮点条件码寄存器 (FCCR, CP1 Register 25)

FCCR 寄存器是访问 FCC 字段的另一种方式，其内容与 FCSR 里的 FCC 位完全相同，不同的是在本寄存器中的 FCC 位是连续的。图 2-5 说明了 FCCR 寄存器的格式。

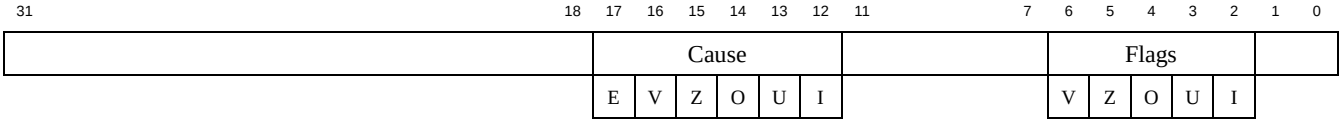
图 2-5 FCCR 寄存器格式

31 … 8	7	6	5	4	3	2	1	0
	FCC							
	7	6	5	4	3	2	1	0

浮点例外寄存器(FCCR, CP1 Register 26)

FEXR 寄存器是访问 Cause 和 Flags 字段的另一种方式，其内容与 FCSR 里的相应字段完全相同。图 2-6 说明了 FEXR 寄存器的格式。

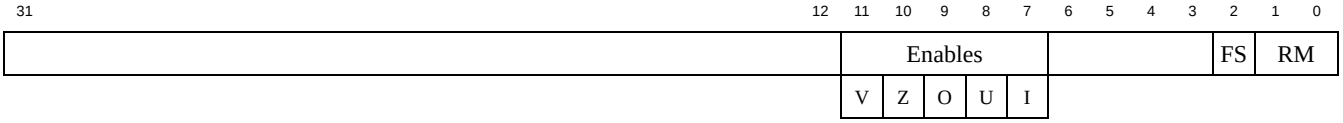
图 2-6 FEXR 寄存器格式



浮点使能寄存器(FCCR, CP1 Register 28)

FENR 寄存器是访问 Enable, FS 和 RM 字段的另一种方式。图 2-7 说明了 FENR 寄存器的格式。

图 2-7 FENR 寄存器格式



2.2.4 浮点例外

浮点例外发生在当 FPU 不能以常规的方式处理操作数或者浮点计算的结果时，FPU 产生相应的例外来启动相应的软件陷阱或者是设置状态标志位。

FPU 的控制和状态寄存器对于每一种例外都包含一个使能位，使能位决定一个例外是否能够导致 FPU 启动一个例外陷阱或者设置一个状态标志。

如果一个陷阱启动，FPU 保持操作开始的状态，启动软件例外处理路径；如果没有陷阱启动，一个适当的值写到 FPU 目标寄存器中，计算继续进行。

FPU 支持五个 IEEE754 例外：

- 不精确 Inexact (I)
- 下溢 Underflow (U)
- 上溢 Overflow (O)
- 除零 Division by Zero (Z)
- 非法操作 Invalid Operation (V)

以及第六个例外：

- 未实现操作 Unimplemented Operation（E）

未实现操作例外用在当 FPU 不能执行标准的 MIPS 浮点结构，包括 FPU 不能决定正确的例外行为的情况。这个例外指示了软件例外处理的执行。未实现操作例外没有使能信号和标志位，当这个例外发生时，一个相应的未实现例外陷阱发生。

IEEE754 的 5 个例外（V，Z，O，U，I）都对应着一个由用户控制的例外陷阱，当 5 个使能位的某一位被设置时，相应的例外陷阱被允许发生。当例外发生时，相应的导致（Cause）位被设置，如果相应的使能（Enable）位没有设置，例外标志（Flag）位被设置。如果使能位被设置，那么标志位不被设置，同时 FPU 产生一个例外给 CPU。随后的例外处理允许该例外陷阱发生。

当没有例外陷阱信号时，浮点处理器采取缺省方式进行处理，提供一个浮点计算例外结果的替代值。不同的例外类型决定了不同的缺省值。列出了 FPU 对于每个 IEEE 例外的默认处理。

表 2-19 浮点例外的默认处理

域	描述	舍入模式	默认操作
I	非精确	任何模式	提供舍入后的结果
U	下溢	RN	根据中间结果的符号把结果置 0
		RZ	根据中间结果的符号把结果置 0
		RP	把正下溢修正为最小正数，把负下溢修正为-0
		RM	把负下溢修正为最小负数,把正下溢修正为+0
O	上溢	RN	根据中间结果的符号把结果置为无穷大
		RZ	根据中间结果的符号把结果置为最大数
		RP	把负下溢修正为最大负数,把正下溢修正为+∞
		RM	把正下溢修正为最大整数,把负下溢修正为-∞
Z	被 0 除	任何模式	提供一个相应的带符号的无穷大数
V	非法操作	任何模式	提供一个 Quiet Not a Number(QNaN)

下面对导致 FPU 产生每种例外的条件进行了描述，并且详细说明了 FPU 对每个例外导致条件的反应。

不精确例外 (I)

FPU 在发生如下的情况时产生不精确例外：

- 舍入结果非精确
- 舍入结果上溢
- 舍入结果下溢，并且下溢和不精确的使能位都没有被设置，而且 FS 位被设置。

陷阱被使能的结果：如果一个非精确例外陷阱被使能，结果寄存器不被修改，并且源寄存器被保留。因为这种执行模式会影响性能，所以不精确例外陷阱只有在必要的时候才被使能。

陷阱不被使能的结果：如果没有其他软件陷阱发生，舍入或者上溢结果被发送到目标寄存器。

非法操作例外 (V)

当一个可执行的操作的两个操作数或其中的一个操作数是非法时，非法操作例外发出信号通知。如果例外没有陷入，MIPS 定义这个结果是一个 Quiet Not a Number（QNaN）。非法操作包括：

- 加法或者减法：无穷相减。例如：(+∞)+(-∞)或者(-∞)-(- ∞)
- 乘法：0×∞，对于所有的正数和负数
- 除法：0/0，∞/∞，对于所有的正数和负数
- 当不处理 Unordered 的比较操作的操作数是 Unordered
- 对一个指示信号 NaN 进行浮点比较或者转换
- 任何对 SNaN（Signaling NaN）的数学操作。当其中一个操作数为 SNaN 或者两个都为 SNaN 时会导致这个例外（MOV 操作不被认为是数学操作，但 ABS 和 NEG 被认为是数学操作）
- 开方：$\sqrt{X}$，当 X 小于 0 时

软件可以模拟其他给定源操作数的非法操作的例外。例如在 IEEE754 中利用软件来实现的特定函数：X REM Y，这里当 Y 是 0 或者 X 是无穷的时候；或者当浮点数转化为十进制时发生上溢，是无穷或者是 NaN；或者先验函数例如：ln(5)或者 $\cos^{-1}(3)$。

陷阱被使能的结果：源操作数的值不被发送。

陷阱不使能的结果：如果没有其他例外发生，QNaN 被发送到目标寄存器中。

除零例外（Z）

除法运算中当除数是 0 被除数是一个有限的非零的数据时，除零例外发出信号通知。利用软件可以对其他操作产生有符号的无穷值时模拟除零例外，如：ln(0)，sin(π/2)，cos(0)，或者 0^{-1}。

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

陷阱不使能的情况：如果没有陷阱发生，结果是有符号的无穷值。

上溢例外（O）

当舍入后的浮点结果的幅度用没有界限的指数来表示时，大于最大的目标模式所表示有限数据，上溢例外发出通知信号。（这个例外同时设置不精确例外和标志位）

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

陷阱不使能的情况：如果没有陷阱发生，最后的结果由舍入模式和中间结果的符号来决定。

下溢例外（U）

两个相关的事件导致了下溢例外：

- 一个很小的在±2Emin 之间的非零结果，由于该结果非常小，因此会导致其后发生下溢例外。
- 用非规格化数（Denormalized Number）来近似表示这两个小数据所产生的严重的数据失真。

IEEE754 允许用多种不同的方法检测这些事件，但对于所有的操作要求用相同的方法来检测。小数据可以用下面的方法的一种来检测：

- 舍入后（如果一个非零的数据，在指数范围没有界限的情况下来计算，应该严格的位于±2Emin 之间）
- 舍入前（如果一个非零的数据，在指数和精度范围没有界限的情况下来计算，应该严格的位于±2Emin 之间）

MIPS 的结构要求微小数据在舍入后检测。精度失真可以用如下方法的一种来检测：

- 非规格化数的失真（当产生的结果与指数没有界限时计算的结果不同）
- 非精确数据（当产生的结果与指数和精度范围没有界限的情况下计算的结果不同）

MIPS 结构要求精度失真被检测为产生非精确结果。

陷阱被使能的情况：如果下溢或者不精确例外被使能，或者 FS 位没有设置，产生未实现操作例外，结果寄存器不被修改。

陷阱不使能的情况：如果下溢或者不精确例外不被使能，而且 FS 位被设置，最后的结果由舍入模式和立即结果的符号位来决定。

未实现操作例外（E）

当执行任何一条为以后定义所保留的操作码或者操作格式指令时，FPU 控制/状态寄存器中的未实现操作导致位被设置并产生陷阱。源操作数和目的寄存器保持不变，同时指令在软件中仿真。IEEE754 中的任何一个例外都能够从仿真操作中产生，这些例外反过来可以被仿真。另外，当硬件不能正确执行一些罕见的操作或者结果条件时，也会产生未实现指令例外。这些包括：

- 非规格化操作数（Denormalized Operand），比较指令除外
- Quite Not a Number 操作数（QNaN），比较指令除外
- 非规格化数或者下溢，而且当下溢或者不精确使能信号被设置同时 FS 位没有被设置

注意：非规格化数和 NaN 操作只在转换或者计算指令中进入陷阱，在 MOV 指令中不进入陷阱。

陷阱被使能的情况：原操作数据不被发送。

陷阱不使能的情况：这个陷阱不能被不使能。

2.2.5 MIPS64 兼容浮点指令列表

GS464E 所实现 MIPS64 兼容浮点协处理器相关指令按功能划分包括如下几类：

- 运算指令
- 分支跳转指令
- 比较指令
- 转换指令
- 移动指令
- 访存指令

下面对这些指令逐类予以列举。

浮点访存指令

表 2-20 浮点分支跳转指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
LDC1	从内存取双字						MIPS32
LDXC1	按索引从内存取双字						MIPS64
LUXC1	按非对齐索引从内存取双字						MIPS64
LWC1	从内存取字						MIPS32
LWXC1	按索引从内存取字						MIPS64
SDC1	存双字到内存						MIPS32
SDXC1	按索引存双字到内存						MIPS64
SUXC1	按非对齐索引存双字到内存						MIPS64
SWC1	存字到内存						MIPS32
SWXC1	按索引存字到内存						MIPS64

浮点运算指令

表 2-21 浮点运算指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
ABS.fmt	绝对值	✓	✓	✓	-	-	MIPS32
ADD.fmt	加法	✓	✓	✓	-	-	MIPS32
DIV.fmt	除法	✓	✓	-	-	-	MIPS32
MADD.fmt	乘加	✓	✓	✓	-	-	MIPS64, MIPS32 R2
MSUB.fmt	乘减	✓	✓	✓	-	-	MIPS64, MIPS32 R2
MUL.fmt	乘法	✓	✓	✓	-	-	MIPS32
NEG.fmt	求反	✓	✓	✓	-	-	MIPS32

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
NMADD.fmt	乘加后求反	✓	✓	✓	-	-	MIPS64, MIPS32 R2
NMSUB.fmt	乘减后求反	✓	✓	✓	-	-	MIPS64, MIPS32 R2
RECIP.fmt	求倒数	✓	✓	-	-	-	MIPS64, MIPS32 R2
RSQRT.fmt	平方根后求倒数	✓	✓	-	-	-	MIPS64, MIPS32 R2
SQRT.fmt	平方根	✓	✓	-	-	-	MIPS32
SUB.fmt	减法	✓	✓	✓	-	-	MIPS32

浮点分支跳转指令

表 2-22 浮点分支跳转指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
BC1F	浮点条件位假时跳转						MIPS32
BC1FL	浮点条件位假时 Likely 跳转						MIPS32
BC1T	浮点条件位真时跳转						MIPS32
BC1TL	浮点条件位真时 Likely 跳转						MIPS32

浮点比较指令

表 2-23 浮点分支跳转指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
C.cond.fmt	比较浮点值并置条件位	✓	✓	✓	-	-	MIPS32

浮点转换指令

表 2-24 浮点分支跳转指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
ALNV.PS	可变浮点对齐	-	-	✓	-	-	MIPS64
CEIL.L.fmt	浮点转换到 64 位定点，向上取整	✓	✓	-	-	-	MIPS64
CEIL.W.fmt	浮点转换到 32 位定点，向上取整	✓	✓	-	-	-	MIPS64
CVT.D.fmt	浮点或定点转换到双精度浮点	✓	-	-	✓	✓	MIPS32
CVT.L.fmt	转换浮点值到 64 位定点	✓	✓	-	-	-	MIPS64
CVT.PS.S	转换两个浮点值到浮点对	✓	-	-	-	-	MIPS64
CVT.S.PL	转换浮点对的低位到单精度浮点	-	-	✓	-	-	MIPS64
CVT.S.PU	转换浮点对的高位到单精度浮点	-	-	✓	-	-	MIPS64
CVT.S.fmt	浮点或定点转换到单精度浮点	-	✓	-	✓	✓	MIPS32
CVT.W.fmt	转换浮点值到 32 位定点	✓	✓	-	-	-	MIPS32
FLOOR.L.fmt	浮点转换到 64 位定点，向下取整	✓	✓	-	-	-	MIPS64
FLOOR.W.fmt	浮点转换到 32 位定点，向下取整	✓	✓	-	-	-	MIPS64

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
PLL.PS	合并两个浮点对的低位为新的浮点对	-	-	✓	-	-	MIPS64
PLU.PS	合并两个浮点对的低位和高位为新的浮点对	-	-	✓	-	-	MIPS64
PUL.PS	合并两个浮点对的高位和低位为新的浮点对	-	-	✓	-	-	MIPS64
PUU.PS	合并两个浮点对的高位为新的浮点对	-	-	✓	-	-	MIPS64
ROUND.L.fmt	把浮点数四舍五入到 64 位定点	✓	✓	-	-	-	MIPS64
ROUND.W.fmt	把浮点数四舍五入到 32 位定点	✓	✓	-	-	-	MIPS32
TRUNC.L.fmt	把浮点数向绝对值小的方向舍入到 64 位定点	✓	✓	-	-	-	MIPS64
TRUNC.W.fmt	把浮点数向绝对值小的方向舍入到 32 位定点	✓	✓	-	-	-	MIPS32

浮点移动指令

表 2-25 浮点分支跳转指令

指令助记符	指令功能简述	fmt					ISA 兼容类别
		S	D	PS	L	W	
CFC1	读浮点控制寄存器到 GPR						MIPS32
CTC1	写浮点控制寄存器到 GPR						MIPS32
DMFC1	从 FPR 复制双字到 GPR						MIPS64
DMTC1	从 GPR 复制双字到 FPR						MIPS64
MFC1	从 FPR 复制低字到 GPR						MIPS32
MFHC1	从 FPR 复制高字到 GPR						MIPS32 R2
MOV.fmt	复制 FPR	✓	✓	✓	-	-	MIPS32
MOVF.fmt	浮点假时复制 FPR	✓	✓	✓	-	-	MIPS32
MOVN.fmt	GPR 不为 0 时复制 FPR	✓	✓	✓	-	-	MIPS32
MOVT.fmt	浮点真时复制 FPR	✓	✓	✓	-	-	MIPS32
MOVZ.fmt	GPR 为 0 时复制 FPR	✓	✓	✓	-	-	MIPS32
MTC1	从 GPR 复制低字到 FPR						MIPS32
MTHC1	从 GPR 复制高字到 FPR						MIPS32 R2

2.3 MIPS64 DSP 指令集概览

GS464 兼容实现了 MIPS64 DSP ASE（r2.34 版）。所实现 DSP 各指令的详细描述请参看《MIPS® Architecture for Programmers VolumeIV-e: The MIPS® DSP Application-Specific Extension to the MIPS64® Architecture》(r2.34)。

2.3.1 MIPS64 DSP ASE 兼容指令列表

GS464E 所实现 MIPS64 DSP ASE 兼容浮点协处理器相关指令按功能划分包括如下几类：

- 运算类指令
- 基于通用寄存器的移位类指令
- 乘法类指令

- 位操作类指令
- 比较-提取类指令
- 累加器操作访问类指令
- DSP 控制寄存器访问类指令
- 带索引寄存器的访存类指令
- 分支指令

下面对这些指令逐类予以列举。

运算类指令

指令助记符	指令功能简述	ISA 兼容类别
ADDQ.PH	向量（最右 2 个）小数半字加	MIPS DSP
ADDQ_S.PH	向量（最右 2 个）小数半字饱和加	MIPS DSP
ADDQ_S.W	有符号字饱和加	MIPS DSP
ADDU_S.QB	向量（最右 4 个）小数字节无符号饱和加	MIPS DSP
ADDUH.QB	向量无符号（最右 4 个）字节加，结果除 2，结果符号扩展	MIPS DSP R2
ADDUH_R.QB	向量无符号（最右 4 个）字节舍入加，结果除 2，结果符号扩展	MIPS DSP R2
ADDU.PH	向量（最右 2 个）小数半字无符号加	MIPS DSP R2
ADDU_S.PH	向量（最右 2 个）小数半字无符号饱和加	MIPS DSP R2
ADDQH.PH	向量（最右 2 个）半字加，结果除 2，结果符号扩展	MIPS DSP R2
ADDQH_R.PH	向量（最右 2 个）半字舍入加，结果除 2，结果符号扩展	MIPS DSP R2
ADDQH.W	向量最右字加，结果除 2，结果符号扩展	MIPS DSP R2
ADDQH_R.W	向量最右字舍入加，结果除 2，结果符号扩展	MIPS DSP R2
SUBQ.PH	向量（最右 2 个）小数半字减	MIPS DSP
SUBQ_S.PH	向量（最右 2 个）小数半字饱和减	MIPS DSP
SUBQ_S.W	有符号字饱和减	MIPS DSP
SUBU.QB	向量（最右 4 个）小数字节无符号减	MIPS DSP
SUBU_S.QB	向量（最右 4 个）小数字节无符号饱和减	MIPS DSP
SUBUH.QB	向量无符号（最右 4 个）字节减，结果除 2，结果符号扩展	MIPS DSP R2
SUBUH_R.QB	向量无符号（最右 4 个）字节舍入减，结果除 2，结果符号扩展	MIPS DSP R2
SUBU.PH	向量（最右 2 个）小数半字无符号减	MIPS DSP R2
SUBU_S.PH	向量（最右 2 个）小数半字无符号饱和减	MIPS DSP R2
SUBQH.PH	向量（最右 2 个）半字减，结果除 2，结果符号扩展	MIPS DSP R2
SUBQH_R.PH	向量（最右 2 个）半字舍入减，结果除 2，结果符号扩展	MIPS DSP R2
SUBQH.W	向量最右字减，结果除 2，结果符号扩展	MIPS DSP R2
SUBQH_R.W	向量最右字舍入减，结果除 2，结果符号扩展	MIPS DSP R2
ADDSC	有符号字加并设置进位	MIPS DSP
ADDWC	有符号字带进位加	MIPS DSP
MODSUB	用索引值做模式减法	MIPS DSP
RADDU.W.QB	（最右）4 字节无符号累加	MIPS DSP
ABSQ_S.QB	向量求取（最右 4 个）字节绝对值，并做饱和操作，结果符号扩展	MIPS DSP R2

指令助记符	指令功能简述	ISA 兼容类别
ABSQ_S.PH	向量求取（最右 2 个）半字绝对值，并做饱和操作，结果符号扩展	MIPS DSP
ABSQ_S.W	向量求取（最右）字绝对值，并做饱和操作，结果符号扩展	MIPS DSP
PRECR.QB.PH	向量整数精度缩减，从（最右两个）半字到 4 个字节	MIPS DSP R2
PRECRQ.QB.PH	向量小数精度缩减，从（最右两个）半字到 4 个字节	MIPS DSP
PRECR_SRA.PH.W	向量右移整数精度缩减，从(最右)字到两个半字，结果符号扩展	MIPS DSP R2
PRECR_SRA_R.PH.W	向量右移整数精度缩减，从(最右)字到两个半字，并做舍入，结果符号扩展	MIPS DSP R2
PRECRQ.PH.W	向量小数精度缩减，从（最右）字到（两个）半字	MIPS DSP
PRECRQ_RS.PH.W	向量小数精度缩减，从（最右）字到（两个）半字，并做饱和与舍入	MIPS DSP
PRECRQU_S.QB.PH	向量小数精度缩减，从（最右两个）半字到 4 个无符号字节	MIPS DSP
PRECEQ.W.PHL	向量精度扩展，从（次右）半字到字，结果符号扩展	MIPS DSP
PRECEQ.W.PHR	向量精度扩展，从（最右）半字到字，结果符号扩展	MIPS DSP
PRECEQU.PH.QBL	向量小数精度扩展，从（次右两个）无符号字节到两个半字	MIPS DSP
PRECEQU.PH.QBR	向量精度扩展，从（最右两个）无符号字节到两个半字	MIPS DSP
PRECEQU.PH.QBLA	向量小数精度扩展，从（最右字的左边交叉两个）无符号字节到两个半字	MIPS DSP
PRECEQU.PH.QBRA	无符号字节到两个半字	MIPS DSP
PRECEU.PH.QBL	向量整数精度扩展，从（次右两个）无符号字节到无符号半字	MIPS DSP
PRECEU.PH.QBR	向量整数精度扩展，从（最右两个）无符号字节到无符号半字	MIPS DSP
PRECEU.PH.QBLA	向量整数精度扩展，从（最右字的左边交叉两个）无符号字节到两个半字	MIPS DSP
PRECEU.PH.QBRA	向量整数精度扩展，从（最右字的右边交叉两个）无符号字节到两个半字	MIPS DSP

基于通用寄存器的移位类指令

指令助记符	指令功能简述	ISA 兼容类别
SHLL.QB	向量逻辑左移（最右 4 个）字节，移位值由立即数指定，结果符号扩展	MIPS DSP
SHLLV.QB	向量逻辑左移（最右 4 个）字节，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHLL.PH	向量逻辑左移（最右 2 个）半字，移位值由立即数指定，结果符号扩展	MIPS DSP
SHLLV.PH	向量逻辑左移（最右 2 个）半字，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHLL_S.PH	向量逻辑饱和左移（最右 2 个）半字，移位值由立即数指定，结果符号扩展	MIPS DSP
SHLLV_S.PH	向量逻辑饱和左移（最右 2 个）半字，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHLL_S.W	向量逻辑饱和左移（最右）字，移位值由立即数指定，结果符号扩展	MIPS DSP

指令助记符	指令功能简述	ISA 兼容类别
SHLLV_S.W	向量逻辑饱和左移（最右）字，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHRL.QB	向量逻辑右移（最右 4 个）字节，移位值由立即数指定，结果符号扩展	MIPS DSP
SHRLV.QB	向量逻辑右移（最右 4 个）字节，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHRL.PH	向量逻辑右移（最右 2 个）半字，移位值由立即数指定，结果符号扩展	MIPS DSP R2
SHRLV.PH	向量逻辑右移（最右 2 个）半字，移位值由寄存器指定，结果符号扩展	MIPS DSP R2
SHRA.QB	向量算术右移（最右 4 个）字节，移位值由立即数指定，结果符号扩展	MIPS DSP R2
SHRA_R.QB	向量算术右移（最右 4 个）字节，移位值由立即数指定，并做舍入，结果符号扩展	MIPS DSP R2
SHRAV.QB	向量算术右移（最右 4 个）字节，移位值由寄存器指定，结果符号扩展	MIPS DSP R2
SHRAV_R.QB	向量算术舍入右移（最右 4 个）字节，移位值由寄存器指定，结果符号扩展	MIPS DSP R2
SHRA.PH	向量算术右移（最右 2 个）半字，移位值由立即数指定，结果符号扩展	MIPS DSP
SHRAV.PH	向量算术右移（最右 2 个）半字，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHRA_R.PH	向量算术舍入右移（最右 2 个）半字，移位值由立即数指定，结果符号扩展	MIPS DSP
SHRAV_R.PH	向量算术舍入右移（最右 2 个）半字，移位值由寄存器指定，结果符号扩展	MIPS DSP
SHRA_R.W	向量算术舍入右移（最右）字，移位值由立即数指定，结果符号扩展	MIPS DSP
SHRAV_R.W	向量算术舍入右移（最右）字，移位值由寄存器指定，结果符号扩展	MIPS DSP

乘法类指令

指令助记符	指令功能简述	ISA 兼容类别
MULEU_S.PH.QBL	向量（次右边两个）字节无符号乘（最右两个）半字，结果为两个半字	MIPS DSP
MULEU_S.PH.QBR	向量（最右边两个）字节无符号乘（最右两个）半字，结果为两个半字	MIPS DSP
MULQ_RS.PH	向量（最右两个）半字饱和且舍入乘，结果为半字	MIPS DSP
MULEQ_S.W.PHL	有符号（次右）半字饱和乘，结果为字	MIPS DSP
MULEQ_S.W.PHR	有符号（最右）半字饱和乘，结果为字	MIPS DSP
DPAU.H.QBL	向量整数（次右两个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP

指令助记符	指令功能简述	ISA 兼容类别
DPAU.H.QBR	向量整数（最右两个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP
DPAU.H.OBL	向量整数（最左 4 个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP
DPAU.H.OBR	向量整数（最右 4 个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP
DPSU.H.QBL	向量整数（最左两个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP
DPSU.H.QBR	向量整数（最右两个）字节无符号乘累加，结果再与 acc 累加	MIPS DSP
DPA.W.PH	向量整数（最右两个）半字乘累加，最后再与 acc 累加	MIPS DSP R2
DPAX.W.PH	向量整数（次右两个）半字乘累加，结果再与 acc 累加	MIPS DSP R2
DPAQ_S.W.PH	向量小数（次右两个）半字乘累加，结果再与 acc 累加	MIPS DSP R2
DPAQX_S.W.PH	向量小数（最右两个）半字乘，结果饱和后累加，再与 acc 累加	MIPS DSP R2
DPAQX_SA.W.PH	向量小数（最右两个）半字乘，结果饱和舍入后累加，再与 acc 累加	MIPS DSP R2
DPS.W.PH	向量整数（最右两个）半字乘累加，结果再与 acc 相减	MIPS DSP R2
DPSX.W.PH	向量整数（次右两个）半字乘累加，结果再与 acc 相减	MIPS DSP R2
DPSQ_S.W.PH	向量小数（次右两个）半字乘累加，结果再与 acc 相减	MIPS DSP
DPSQX_S.W.PH	向量小数（最右两个）半字乘，结果饱和后累加，再与 acc 相减	MIPS DSP R2
DPSQX_SA.W.PH	向量小数（最右两个）半字乘，结果饱和舍入后累加，再与 acc 相减	MIPS DSP R2
MULSAQ_S.W.PH	向量小数（最右两个）半字乘相减，结果再与 acc 累加	MIPS DSP
DPAQ_SA.L.W	向量小数字乘，结果饱和舍入后累加，再与 acc 累加	MIPS DSP
DPAQ_SA.L.PW	向量小数字乘，结果饱和舍入后累加，再与 acc 累加	MIPS DSP
DPSQ_SA.L.W	向量小数字乘，结果饱和舍入后累加，结果再与 acc 累加	MIPS DSP
DPSQ_SA.L.PW	向量小数字乘，结果饱和舍入后累加，结果再与 acc 累加	MIPS DSP
MULSAQ_S.L.PW	向量小数字乘，饱和后相减，结果再与 acc 累加	MIPS DSP
MAQ_S.W.PHL	向量小数（次右）半字乘，结果再与 acc 累加	MIPS DSP
MAQ_S.W.PHR	向量小数（最右）半字乘，结果再与 acc 累加	MIPS DSP
MAQ_SA.W.PHL	向量小数（次右）半字乘，结果求取饱和舍入后再与 acc 累加	MIPS DSP
MAQ_SA.W.PHR	向量小数（最右）半字乘，结果求取饱和舍入后再与 acc 累加	MIPS DSP
MUL.PH	向量（最右 2 个）半字乘，结果低 16 位写入寄存器	MIPS DSP R2
MUL_S.PH	向量（最右 2 个）半字有符号饱和乘，结果低 16 位写入寄存器	MIPS DSP R2
MULQ_S.PH	向量（最右两个）半字饱和乘，结果为半字	MIPS DSP R2
MULQ_S.W	向量最右字饱和乘，结果为字	MIPS DSP R2
MULQ_RS.W	向量最右字饱和且舍入乘，结果为字	MIPS DSP R2
MULSA.W.PH	向量（最右两个）半字乘相减，结果再与 acc 累加	MIPS DSP R2
MADD	32 位有符号定点数乘再与 acc 累加	MIPS32
MADDU	32 位无符号定点数乘再与 acc 累加	MIPS32
MSUB	32 位有符号定点数乘再从 acc 减去	MIPS32
MSUBU	32 位无符号定点数乘再从 acc 减去	MIPS32
MULT	字乘，结果存到 acc 寄存器	MIPS32
MULTU	无符号字乘，结果存到 acc 寄存器	MIPS32

位操作类指令

指令助记符	指令功能简述	ISA 兼容类别
BITREV	半字位翻转，结果 0 扩展	MIPS DSP
INSV	可变位域插入	MIPS DSP
REPL.QB	向量复制立即数（整数）到最右四字节，结果符号扩展	MIPS DSP
REPLV.QB	向量复制字节到最右四字节，结果符号扩展	MIPS DSP
REPL.PH	向量复制立即数（整数）到最右 2 个半，结果符号扩展	MIPS DSP
REPLV.PH	向量复制半字到最右 2 个半字，结果符号扩展	MIPS DSP

比较-提取类指令

指令助记符	指令功能简述	ISA 兼容类别
CMPU.EQ.QB	向量（最右 4 个）无符号字节相等比较，结果置条件位	MIPS DSP
CMPU.LT.QB	向量（最右 4 个）无符号字节小于比较，结果置条件位	MIPS DSP
CMPU.LE.QB	向量（最右 4 个）无符号字节小于等于比较，结果置条件位	MIPS DSP
CMPGDU.EQ.QB	向量（最右 4 个）无符号字节相等比较，结果同时置条件位与通用寄存器	MIPS DSP R2
CMPGDU.LT.QB	向量（最右 4 个）无符号字节小于比较，结果同时置条件位与通用寄存器	MIPS DSP R2
CMPGDU.LE.QB	向量（最右 4 个）无符号字节小于等于比较，结果同时置条件位与通用寄存器	MIPS DSP R2
CMPGU.EQ.QB	向量（最右 4 个）字节相等比较，结果置通用寄存器	MIPS DSP
CMPGU.LT.QB	向量（最右 4 个）字节小于比较，结果置通用寄存器	MIPS DSP
CMPGU.LE.QB	向量（最右 4 个）字节小于等于比较，结果置通用寄存器	MIPS DSP
CMP.EQ.PH	向量（最右 2 个）半字相等比较，结果置条件位	MIPS DSP
CMP.LT.PH	向量（最右 2 个）半字小于比较，结果置条件位	MIPS DSP
CMP.LE.PH	向量（最右 2 个）半字小于等于比较，结果置条件位	MIPS DSP
PICK.QB	基于条件位的（最右四个）字节选择	MIPS DSP
PICK.PH	基于条件位的（最右 2 个）半字选择	MIPS DSP
APPEND	单字左移并且低位拼接	MIPS DSP R2
PREPEND	右移并且高位拼接	MIPS DSP R2
BALIGN	两个寄存器高、低字节拼接	MIPS DSP R2
PACKRL.PH	将源 1 的最右与源 2 的次右半字打包	MIPS DSP R2

累加器操作类指令

指令助记符	指令功能简述	ISA 兼容类别
EXTR.W	累加器右移后，截取字赋给通用寄存器，移位值由立即数指定	MIPS DSP
EXTR_R.W	累加器右移后舍入，截取字赋给通用寄存器，移位值由立即数指定	MIPS DSP
EXTR_RS.W	累加器右移后饱和舍入，截取字赋给通用寄存器，移位值由立即数指定	MIPS DSP
EXTR_S.H	从累加器饱和右移、提取半字到通用寄存器，移位值由立即数指定	MIPS DSP
EXTRV_S.H	从累加器饱和右移、提取半字到通用寄存器，移位值由寄存器指定	MIPS DSP
EXTRV.W	累加器右移后，截取字赋给通用寄存器，移位值由寄存器指定	MIPS DSP

指令助记符	指令功能简述	ISA 兼容类别
EXTRV_R.W	累加器右移后舍入，截取字赋给通用寄存器，移位值由寄存器指定	MIPS DSP
EXTRV_RS.W	累加器右移后饱和舍入，截取字赋给通用寄存器，移位值由寄存器指定	MIPS DSP
EXTP	从累加器的任意位置提取固定长度的数到通用寄存器，长度由立即数指定	MIPS DSP
EXTPV	从累加器的任意位置提取固定长度的数到通用寄存器，长度由寄存器指令	MIPS DSP
EXTPDP	从累加器的任意位置提取固定长度的数到通用寄存器，并减 pos 值，长度由立即数指定	MIPS DSP
EXTPDPV	从累加器的任意位置提取固定长度的数到通用寄存器，并减 pos 值，长度由寄存器指定	MIPS DSP
SHILO	累加器值移位后再写回相同的累加器，移位值由立即数决定	MIPS DSP
SHILOV	累加器值移位后再写回相同的累加器，移位值由寄存器决定	MIPS DSP
MTHLIP	LO 值拷贝到 HI，通用寄存器值拷贝到 LO，pos 值增加 32	MIPS DSP
MFHI	HI 寄存器值移到通用寄存器	MIPS32
MFLO	LO 寄存器值移到通用寄存器	MIPS32
MTHI	通用寄存器值移到 HI 寄存器	MIPS32
MTLO	通用寄存器值移到 LO 寄存器	MIPS32

访问 DSP 控制寄存器类指令

指令助记符	指令功能简述	ISA 兼容类别
WRDSP	读通用寄存器值写到 DSP 控制寄存器	MIPS DSP
RDDSP	读 DSP 控制寄存器值到通用寄存器	MIPS DSP

带索引寄存器的访存类指令

指令助记符	指令功能简述	ISA 兼容类别
LBUX	索引地址取无符号字节	MIPS DSP
LHX	索引地址取半字	MIPS DSP
LWX	索引地址取字	MIPS DSP

分支指令

指令助记符	指令功能简述	ISA 兼容类别
BPOSGE32	DSP 控制寄存器 pos 值大于等于 32 则跳转	MIPS DSP

2.3.2 对 MIPS DSP 指令手册的补充说明

MIPS DSP 指令手册中对于 MTHI、MTLO 指令的定义不合理。在 64 位架构下，执行 MTHI 和 MTLO 指令不需要将通用寄存器的值进行高 32 位的符号扩展后再写入 HI/LO 寄存器。应采用 MIPS 通用指令手册中对于 MTHI、MTLO 指令的定义形式，即 HI ← GPR[rs]，LO ← GPR[rs]。

2.4 MIPS64 兼容指令实现相关定义

本小节对于 GS464E 所实现的 MIPS64 兼容指令中与实现相关部分，或是与 MIPS64 规范存在差异的部分进行描述。

2.4.1 目标为 0 号通用寄存器的 load 指令

除 LDPTE 和 LWPTE 外的所有目标为通用寄存器的 load 指令，当其目标位 0 号通用寄存器时，其执行效果等同于 PREF(hint=0)。不过对于龙芯自定义扩展的 GSLQ 指令，必须同时将其两个目标寄存器置为 0 号通用寄存器才可以。

2.4.2 PREF 指令

PREF 指令只按照 MIPS 规范实现了 hint=0（prefetch for load）和 hint=1（prefetch for store）两种模式。

2.4.3 RDHWR 指令

RDHWR 指令除了按照 MIPS 规范实现了 rd 值为 0、1、2、3、29 外，还实现了 rd 值为 30 和 31。具体定义如下：

- rd=30：读取外部计数器值。该计数器 64 位，每拍增 1，增加频率与芯片内部总线时钟频率一致。该时钟频率的特点是不随着某个处理器核的时钟频率的变化而变化，可认为是一恒定频率。
- rd=31：读取处理器核当前频率与芯片内部总线频率的比例关系（M/N）。其中 M 位于返回值的[15:8]位，N 位于返回值的[7:0]。

2.4.4 PREFX 指令

PREFX 指令共实现了 0、1、26、27、28 五个 hint。具体实现细节如下：

- hint=0、1：此时 PREFX 指令预取功能与 MIPS 规范一致，即一次预取一个 cacheline。区别在于龙芯 GS464E 只将 index 寄存器中低 16 位的值符号扩展后作为索引使用。
- hint=26：按配置连续预取数据进入一级数据缓存，一条预取指令可以从基址开始自动预取 block_num 个间距为 stride 的长度为 block_size 的 block。此时将 index 寄存器中的部分信息用于配置预取地址的生成模式。具体来说，GPR[index]的[15:0]位为与 GPR[base]相加的 16bit 有符号地址偏移量；GPR[index]的[16]位为地址升降序预取标记，为 0 表示地址升序预取，为 1 表示地址降序预取；GPR[index]的[25:20]位为预取的 block_size-1，block_size 的基本单位为 128bit，因此最长的 block 可包含 64×16=1KB 的数据；GPR[index]的[39:32]位为 block_num-1，因此支持最多 256 个 block 的预取；index[59:44]为相邻 block 之间间隔的 stride，是有符号数，单位为字节。
- hint=27：按配置连续预取独占数据进入一级数据缓存。此模式下 index 寄存器中内容的解析与 hint=26 完全一致，区别在于此模式要求取回的数据一定是独占状态。
- hint=28：按配置连续预取数据进入共享缓存。此模式下 index 寄存器中内容的解析与 hint=26 完全一致，区别在域此模式下预取数据存放在共享缓存中。

如果执行了一条 WAIT_CACHE 指令，配置预取会被停下。这类指令包括但不限于 cache 指令，SYNC 和 SYNCI 指令，JRHB 指令。

如果连续执行两条 PREFX（hint=25,26），之前的那条会被新的覆盖。如果连续执行两条 PREFX（hint=27），之前的那条会被新的覆盖。然而，hint=25,26 和 hint=27 这两中模式的预取之间不冲突。

2.4.5 WAIT 指令

WAIT 指令仅支持一种模式，无论软件对指令编码中的 Implementation dependent code 域填入何值，WAIT 指令执行效果一样。

WAIT 指令在处理器流水线中提交后，处理器核将停止取指，进入低功耗模式，该模式将一直持续直至被 NMI、内部时钟中断、内部性能计数器中断或外部硬件中断打断。

需要注意的是，执行 WAIT 指令后，处理器核的一级指令缓存、一级数据缓存和二级牺牲缓存的数据通路仍可以处理来自外部的缓存一致性请求。如果需要彻底关闭处理器核的时钟，软件需要做更多准备工作，已确保处理器核关时钟后不至于造成其它处理器核因一致性请求无响应而死机。

2.4.6 SYNC 指令

本处理器只实现了 stype=0 的 SYNC 指令，stype 为其它值时会报保留指令例外。该指令起到存储栅障(memory barrier)作用，用于保证 SYNC 之前的访存操作已经确定完成（如，store 指令的数据写入 dcache、uncached 的读写已完成、load 已经将值取回至寄存器），同时保证 SYNC 指令后面的访存操作尚未开始执行。

2.4.7 SYNCI 指令

因 GS464E 由硬件维护一级指令缓存和一级数据缓存之间的数据一致性，因此对 SYNCI 指令的实现功能进行调整。现有实现中 SYNCI 指令一方面会等到前面所有的访存操作都执行完毕后才能开始执行（效果与 SYNC 指令等同）；另一方面会 SYNCI 会在执行之后强行让后续指令重新取指，以保证取指部件也一定能看到 SYNCI 指令之前的所有访存操作的执行效果。与此同时 SYNCI 指令携带的地址信息被忽略，因此不再会触发 TIB 相关例外、地址错例外和 Cache 错误例外。

2.4.8 TLBINV 与 TLBINVF 指令

尽管本处理器中 Config4.IE 域的值设置为 0，但处理器中实现了 TLBINVF 指令（按照 MIPS 规范中所定义的 Config4.IE=3 的方式执行），即执行一条 TLBINVF 指令时硬件会将整个 TLB 表项都无效。此外，GS464E 中也实现 TLBINV 指令，但是其执行效果与 MIPS 规范定义不同，而是等效于 TLBINVF 指令。

2.4.9 CACHE 指令

本处理器所实现的 CACHE 指令与 MIPS64 规范存在的区别主要有三点：

1、指令 op 与缓存层次之间的关系

GS464E 中 CACHE 指令 op 域的[1:0]两位与缓存层次之间的对应关系与 MIPS64 规范的定义不同，如表 2-26 所示：

表 2-26 CACHE 指令 op[1:0]与缓存层次对应关系

op[1:0]	GS464E	MIPS64 规范	异同
0b00	一级指令缓存	一级指令缓存	一致
0b01	一级数据缓存	一级数据缓存	一致
0b10	二级牺牲缓存	三级缓存	不一致
0b11	三级共享缓存	二级缓存	不一致

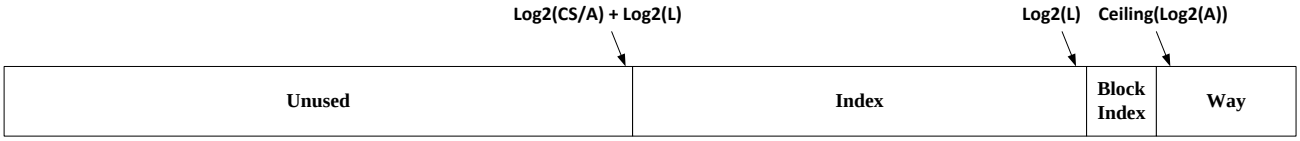
2、Index 类 CACHE 指令的地址解析方式

本处理器实现的 Index 类 Cache 指令的地址解析方式与 MIPS64 规范的定义不同，两者的区别如图 2-8 所示。需要注意的是，正因为 GS464E 采用地址的最低几位来表示待操作的 Cache 路，而二级牺牲缓存与三

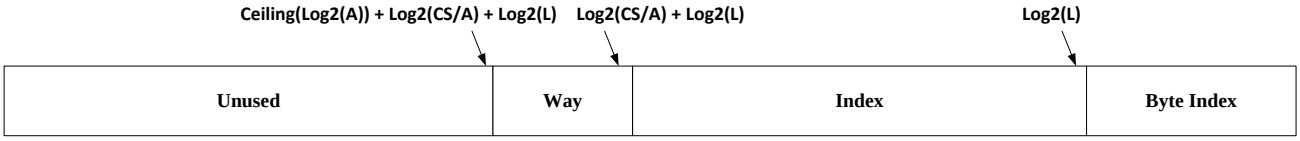
级共享缓存均为 16 路组相联，路选择需要占据地址的最低 4 位，所以对于二级牺牲缓存和三级共享缓存进行的 Index Data Load 和 Index Data Store 操作，只能读取 Cache 行偶数块数据，只能将 Cache 行中奇偶相邻两个块同时填入相同的数据。

图 2-8 Index 类 CACHE 指令的地址解析格式

龙芯3A1500芯片处理器



MIPS64规范



说明：
假设所操作Cache的容量为CS，相联度为A路组相联，Cache行大小为L字节。则
用于选择第几路的地址位数为：Ceiling(Log2(A))
用于索引每一路中第几个Cache行的地址位数为：Log2(CS/A)
用于Cache行内字节定位的地址位数为：Log2(L)

3、CACHE28、CACHE29、CACHE30、CACHE31 指令的含义

GS464E 中，CACHE28、CACHE29、CACHE30 和 CACHE31 指令（即 op[4:2]=0b111 的 Cache 指令）的含义与 MIPS64 规范不同。龙芯处理器中这些指令均为 Index Store Data 操作，而 MIPS64 规范中均为 Fetch and Lock 操作。

4、CACHE15 指令的含义

GS464E 中，CACHE15 指令是一条实现相关的 CACHE 指令，其功能为对 Scache 进行 Fetch and Lock 操作，具体功能定义类似于 MIPS64 规范中的 CACHE31 指令。

2.4.10 MADD.fmt、MSUB.fmt、NMADD.fmt、NMSUB.fmt 指令

在 MIPS64 规范中，如果 FIR.Has2008=0 或者 FCSR.MAC2008=0，则 MADD.fmt、MSUB.fmt、NMADD.fmt 和 NMSUB.fmt 指令对乘法操作后的中间结果舍入后再进行后续加减操作，并对加减之后的最终结果再次舍入。GS464E 尽管不支持 ANSI/IEEE754-2008 二进制浮点运算标准，但在进行浮点乘加类操作时，仅在最后结果处做一次舍入处理，也就是 ANSI/IEEE754-2008 二进制浮点运算标准所定义的 Fuse-Multiply-Add 操作。

2.4.11 EHB、SSNOP 指令

GS464E 指令间所有执行相关均由硬件处理，因此 EHB 和 SSNOP 指令在处理器中均视作 NOP 指令处理。

2.4.12 DI、EI 指令

DI 指令仅在龙芯 3A2000C/3B2000C 款中方可使用，具体定义与 MIPS64 定义兼容。

EI 指令仅在龙芯 3A2000C/3B2000C 款中方可使用，但具体定义与 MIPS64 定义略有差异：EI 执行后可以将 Status 寄存器的 IE 位置为 0，关闭全局终端使能，但是返回值中只有最低 3 位有意义，分别对应 EI 执行前 Status 寄存器的 ERL、EXL 和 IE 位。

2.5 龙芯扩展指令集

GS464E 实现的龙芯扩展指令按功能划分包括如下几类：

- 访存类指令
- 算术与逻辑运算指令
- X86 二进制加速指令
- ARM 二进制加速指令
- 64 位多媒体指令
- 杂项

访存类指令

表 2-27 龙芯扩展访存类指令

指令助记符	指令功能简述	ISA 兼容类别
SETMEM	访存指令扩展前缀	LoongEXT32
LWDIR	32 位页表目录项访问指令	LoongEXT32
LWPTE	32 位页表表项访问指令	LoongEXT32
LDDIR	64 位页表目录项访问指令	LoongEXT64
LDPTE	64 位页表表项访问指令	LoongEXT64
GSLE	如果小于等于置地址错例外	LoongEXT32
GSGT	如果大于置地址错例外	LoongEXT32
GSLWLC1[1]	取字左部到浮点寄存器	LoongEXT32
GSLWRC1[2]	取字右部到浮点寄存器	LoongEXT32
GSLDLC1	取双字左部到浮点寄存器	LoongEXT32
GSLDRC1	取双字右部到浮点寄存器	LoongEXT32
GSLBLE	带上越界检查的取字节	LoongEXT32
GSLBGT	带下越界检查的取字节	LoongEXT32
GSLHLE	带上越界检查的取半字	LoongEXT32
GSLHGT	带下越界检查的取半字	LoongEXT32
GSLWLE	带上越界检查的取字	LoongEXT32
GSLWGT	带下越界检查的取字	LoongEXT32
GSLDLE	带上越界检查的取双字	LoongEXT64
GSLDGT	带下越界检查的取双字	LoongEXT64
GSLWLEC1	带上越界检查的取字到浮点寄存器	LoongEXT32
GSLWGTC1	带下越界检查的取字到浮点寄存器	LoongEXT32

[1] 在 LS3A2000A 和 LS3A2000B 中该指令只能在 Status.FR=1 的情况下才可使用。
[2] 在 LS3A2000A 和 LS3A2000B 中该指令只能在 Status.FR=1 的情况下才可使用。

指令助记符	指令功能简述	ISA 兼容类别
GSLDLEC1	带上越界检查的取双字到浮点寄存器	LoongEXT64
GSLDGTC1	带下越界检查的取双字到浮点寄存器	LoongEXT64
GSLQ	双目标寄存器取定点四字	LoongEXT64
GSLQC1	双目标寄存器取浮点四字	LoongEXT64
GSLBX	带偏移的取字节	LoongEXT32
GSLHX	带偏移的取半字	LoongEXT32
GSLWX	带偏移的取字	LoongEXT32
GSLDX	带偏移的取双字	LoongEXT64
GSLWXC1	带偏移的取浮点字	LoongEXT32
GSLDXC1	带偏移的取浮点双字	LoongEXT32
GSSWLC1	从浮点寄存器存字左部	LoongEXT32
GSSWRC1	从浮点寄存器存字右部	LoongEXT32
GSSDLC1	从浮点寄存器存双字左部	LoongEXT32
GSSDRC1	从浮点寄存器存双字右部	LoongEXT32
GSSBLE	带上越界检查的存字节	LoongEXT32
GSSBGT	带下越界检查的存字节	LoongEXT32
GSSHLE	带上越界检查的存半字	LoongEXT32
GSSHGT	带下越界检查的存半字	LoongEXT32
GSSWLE	带上越界检查的存字	LoongEXT32
GSSWGT	带下越界检查的存字	LoongEXT32
GSSDLE	带上越界检查的存双字	LoongEXT64
GSSDGT	带下越界检查的存双字	LoongEXT64
GSSWLEC1	带上越界检查的从浮点寄存器存字	LoongEXT32
GSSWGTC1	带下越界检查的从浮点寄存器存字	LoongEXT32
GSSDLEC1	带上越界检查的从浮点寄存器存双字	LoongEXT32
GSSDGTC1	带下越界检查的从浮点寄存器存双字	LoongEXT32
GSSQ	双源寄存器存定点四字	LoongEXT64
GSSQC1	双源寄存器存定点四字	LoongEXT64
GSSBX	带偏移的存字节	LoongEXT32
GSSHX	带偏移的存半字	LoongEXT32
GSSWX	带偏移的存字	LoongEXT32
GSSDX	带偏移的存双字	LoongEXT64
GSSWXC1	带偏移的存浮点字	LoongEXT32
GSSDXC1	带偏移的存浮点双字	LoongEXT32

算术与逻辑运算指令

表 2-28 龙芯扩展算术与逻辑运算指令

指令助记符	指令功能简述	ISA 兼容类别
GSANDN	通用寄存器逻辑位非与	LoongEXT32
GSORN	通用寄存器逻辑位非或	LoongEXT32
GSADC.D	带进位双字加	LoongEXT64

指令助记符	指令功能简述	ISA 兼容类别
GSADC.W	带进位字加	LoongEXT32
GSADC.H	带进位半字加	LoongEXT32
GSADC.B	带进位字节加	LoongEXT32
GSSBB.D	带借位双字减	LoongEXT64
GSSBB.W	带借位字减	LoongEXT32
GSSBB.H	带借位半字减	LoongEXT32
GSSBB.B	带借位字节减	LoongEXT32
GSMULT	有符号字乘，结果写通用寄存器	LoongEXT32
GSDMULT	有符号双字乘，结果写通用寄存器	LoongEXT64
GSMULTU	无符号字乘，结果写通用寄存器	LoongEXT32
GSDMULTU	无符号双字乘，结果写通用寄存器	LoongEXT64
GSDIV	有符号字除，商写通用寄存器	LoongEXT32
GSDDIV	有符号双字除，商写通用寄存器	LoongEXT64
GSDIVU	无符号字除，商写通用寄存器	LoongEXT32
GSDDIVU	无符号双字除，商写通用寄存器	LoongEXT64
GSMOD	有符号字除，余数写通用寄存器	LoongEXT32
GSDMOD	有符号双字除，余数写通用寄存器	LoongEXT64
GSMODU	无符号字除，余数写通用寄存器	LoongEXT32
GSDMODU	无符号双字除，余数写通用寄存器	LoongEXT64
GSROTR.H	半字循环右移	LoongEXT32
GSROTR.B	字节循环右移	LoongEXT32
GSROTRV.H	可变移位量半字循环右移	LoongEXT32
GSROTRV.B	可变移位量字节循环右移	LoongEXT32
GSRCR.D	带 CF 位的双字循环右移	LoongEXT64
GSRCR.W	带 CF 位的字循环右移	LoongEXT32
GSRCR.H	带 CF 位的半字循环右移	LoongEXT32
GSRCR.B	带 CF 位的字节循环右移	LoongEXT32
GSDRCR32	移位量加 32 的带 CF 位的双字循环右移	LoongEXT64
GSRCRV.D	可变移位量带 CF 位的双字循环右移	LoongEXT64
GSRCRV.W	可变移位量带 CF 位的字循环右移	LoongEXT32
GSRCRV.H	可变移位量带 CF 位的半字循环右移	LoongEXT32
GSRCRV.B	可变移位量带 CF 位的字节循环右移	LoongEXT32

二进制翻译加速指令(X86)

表 2-29 龙芯扩展 X86 二进制翻译加速指令

指令助记符	指令功能简述	ISA 兼容类别
SETX86FLAG.D	以 x86 方式置 EFLAG 双字模式前缀指令	LoongEXT64
SETX86FLAG.W	以 x86 方式置 EFLAG 字模式前缀指令	LoongEXT32
SETX86FLAG.H	以 x86 方式置 EFLAG 半字模式前缀指令	LoongEXT32
SETX86FLAG.B	以 x86 方式置 EFLAG 字节模式前缀指令	LoongEXT32
X86AND.D	以 x86 方式只设置 EFLAG 的双字逻辑位与	LoongEXT64

指令助记符	指令功能简述	ISA 兼容类别
X86AND.W	以 x86 方式只设置 EFLAG 的字逻辑位与	LoongEXT32
X86AND.H	以 x86 方式只设置 EFLAG 的半字逻辑位与	LoongEXT32
X86AND.B	以 x86 方式只设置 EFLAG 的字节逻辑位与	LoongEXT32
X86OR.D	以 x86 方式只设置 EFLAG 的双字逻辑位或	LoongEXT64
X86OR.W	以 x86 方式只设置 EFLAG 的字逻辑位或	LoongEXT32
X86OR.H	以 x86 方式只设置 EFLAG 的半字逻辑位或	LoongEXT32
X86OR.B	以 x86 方式只设置 EFLAG 的字节逻辑位或	LoongEXT32
X86XOR.D	以 x86 方式只设置 EFLAG 的双字逻辑位异或	LoongEXT64
X86XOR.W	以 x86 方式只设置 EFLAG 的字逻辑位异或	LoongEXT32
X86XOR.H	以 x86 方式只设置 EFLAG 的半字逻辑位异或	LoongEXT32
X86XOR.B	以 x86 方式只设置 EFLAG 的字节逻辑位异或	LoongEXT32
X86ADD.D	以 x86 方式只设置 EFLAG 的双字加	LoongEXT64
X86ADD.W	以 x86 方式只设置 EFLAG 的字加	LoongEXT32
X86ADD.H	以 x86 方式只设置 EFLAG 的半字加	LoongEXT32
X86ADD.B	以 x86 方式只设置 EFLAG 的字节加	LoongEXT32
X86ADDU.D	以 x86 方式只设置 EFLAG 的无例外双字加	LoongEXT64
X86ADDU.W	以 x86 方式只设置 EFLAG 的无例外字加	LoongEXT32
X86SUB.D	以 x86 方式只设置 EFLAG 的双字减	LoongEXT64
X86SUB.W	以 x86 方式只设置 EFLAG 的字减	LoongEXT32
X86SUB.H	以 x86 方式只设置 EFLAG 的半字减	LoongEXT32
X86SUB.B	以 x86 方式只设置 EFLAG 的字节减	LoongEXT32
X86SUBU.D	以 x86 方式只设置 EFLAG 的无例外双字减	LoongEXT64
X86SUBU.W	以 x86 方式只设置 EFLAG 的无例外字减	LoongEXT32
X86INC.D	以 x86 方式只设置 EFLAG 的双字自增 1	LoongEXT64
X86INC.W	以 x86 方式只设置 EFLAG 的字自增 1	LoongEXT32
X86INC.H	以 x86 方式只设置 EFLAG 的半字自增 1	LoongEXT32
X86INC.B	以 x86 方式只设置 EFLAG 的字节自增 1	LoongEXT32
X86DEC.D	以 x86 方式只设置 EFLAG 的双字自减 1	LoongEXT64
X86DEC.W	以 x86 方式只设置 EFLAG 的字自减 1	LoongEXT32
X86DEC.H	以 x86 方式只设置 EFLAG 的半字自减 1	LoongEXT32
X86DEC.B	以 x86 方式只设置 EFLAG 的字节自减 1	LoongEXT32
X86SLL.D	以 x86 方式只设置 EFLAG 的双字左移	LoongEXT64
X86SLL.W	以 x86 方式只设置 EFLAG 的字左移	LoongEXT32
X86SLL.H	以 x86 方式只设置 EFLAG 的半字左移	LoongEXT32
X86SLL.B	以 x86 方式只设置 EFLAG 的字节左移	LoongEXT32
X86DSLL32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑左移	LoongEXT64
X86SLLV.D	以 x86 方式只设置 EFLAG 的双字可变移位量左移	LoongEXT64
X86SLLV.W	以 x86 方式只设置 EFLAG 的字可变移位量左移	LoongEXT32
X86SLLV.H	以 x86 方式只设置 EFLAG 的半字可变移位量左移	LoongEXT32
X86SLLV.B	以 x86 方式只设置 EFLAG 的字节可变移位量左移	LoongEXT32
X86SRL.D	以 x86 方式只设置 EFLAG 的双字逻辑右移	LoongEXT64

指令助记符	指令功能简述	ISA 兼容类别
X86SRL.W	以 x86 方式只设置 EFLAG 的字逻辑右移	LoongEXT32
X86SRL.H	以 x86 方式只设置 EFLAG 的半字逻辑右移	LoongEXT32
X86SRL.B	以 x86 方式只设置 EFLAG 的字节逻辑右移	LoongEXT32
X86DSRL32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑逻辑右移	LoongEXT64
X86SRLV.D	以 x86 方式只设置 EFLAG 的双字可变移位量逻辑右移	LoongEXT64
X86SRLV.W	以 x86 方式只设置 EFLAG 的字可变移位量逻辑右移	LoongEXT32
X86SRLV.H	以 x86 方式只设置 EFLAG 的半字可变移位量逻辑右移	LoongEXT32
X86SRLV.B	以 x86 方式只设置 EFLAG 的字节可变移位量逻辑右移	LoongEXT32
X86SRA.D	以 x86 方式只设置 EFLAG 的双字算术右移	LoongEXT64
X86SRA.W	以 x86 方式只设置 EFLAG 的字算术右移	LoongEXT32
X86SRA.H	以 x86 方式只设置 EFLAG 的半字算术右移	LoongEXT32
X86SRA.B	以 x86 方式只设置 EFLAG 的字节算术右移	LoongEXT32
X86DSRA32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑算术右移	LoongEXT64
X86SRAV.D	以 x86 方式只设置 EFLAG 的双字可变移位量算术右移	LoongEXT64
X86SRAV.W	以 x86 方式只设置 EFLAG 的字可变移位量算术右移	LoongEXT32
X86SRAV.H	以 x86 方式只设置 EFLAG 的半字可变移位量算术右移	LoongEXT32
X86SRAV.B	以 x86 方式只设置 EFLAG 的字节可变移位量算术右移	LoongEXT32
X86ROTR.D	以 x86 方式只设置 EFLAG 的双字循环右移	LoongEXT64
X86ROTR.W	以 x86 方式只设置 EFLAG 的字循环右移	LoongEXT32
X86ROTR.H	以 x86 方式只设置 EFLAG 的半字循环右移	LoongEXT32
X86ROTR.B	以 x86 方式只设置 EFLAG 的字节循环右移	LoongEXT32
X86DROTR32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑循环右移	LoongEXT64
X86ROTL.D	以 x86 方式只设置 EFLAG 的双字循环左移	LoongEXT64
X86ROTL.W	以 x86 方式只设置 EFLAG 的字循环左移	LoongEXT32
X86ROTL.H	以 x86 方式只设置 EFLAG 的半字循环左移	LoongEXT32
X86ROTL.B	以 x86 方式只设置 EFLAG 的字节循环左移	LoongEXT32
X86DROTL32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑循环左移	LoongEXT64
X86RCR.D	以 x86 方式只设置 EFLAG 的带 CF 位的双字循环右移	LoongEXT64
X86RCR.W	以 x86 方式只设置 EFLAG 的带 CF 位的字循环右移	LoongEXT32
X86RCR.H	以 x86 方式只设置 EFLAG 的带 CF 位的半字循环右移	LoongEXT32
X86RCR.B	以 x86 方式只设置 EFLAG 的带 CF 位的字节循环右移	LoongEXT32
X86DRCR32	以 x86 方式只设置 EFLAG 的移位量加 32 的带 CF 位的双字逻辑循环右移	LoongEXT64
X86RCL.D	以 x86 方式只设置 EFLAG 的带 CF 位的双字循环左移	LoongEXT64
X86RCL.W	以 x86 方式只设置 EFLAG 的带 CF 位的字循环左移	LoongEXT32
X86RCL.H	以 x86 方式只设置 EFLAG 的带 CF 位的半字循环左移	LoongEXT32
X86RCL.B	以 x86 方式只设置 EFLAG 的带 CF 位的字节循环左移	LoongEXT32
X86DRCL32	以 x86 方式只设置 EFLAG 的移位量加 32 的带 CF 位的双字逻辑循环左移	LoongEXT64
X86ROTRV.D	以 x86 方式只设置 EFLAG 的可变移位量的双字循环右移	LoongEXT64
X86ROTRV.W	以 x86 方式只设置 EFLAG 的可变移位量的字循环右移	LoongEXT32

指令助记符	指令功能简述	ISA 兼容类别
X86ROTRV.H	以 x86 方式只设置 EFLAG 的可变移位量的半字循环右移	LoongEXT32
X86ROTRV.B	以 x86 方式只设置 EFLAG 的可变移位量的字节循环右移	LoongEXT32
X86ROTLV.D	以 x86 方式只设置 EFLAG 的可变移位量的双字循环左移	LoongEXT64
X86ROTLV.W	以 x86 方式只设置 EFLAG 的可变移位量的字循环左移	LoongEXT32
X86ROTLV.H	以 x86 方式只设置 EFLAG 的可变移位量的半字循环左移	LoongEXT32
X86ROTLV.B	以 x86 方式只设置 EFLAG 的可变移位量的字节循环左移	LoongEXT32
X86RCRV.D	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的双字循环右移	LoongEXT64
X86RCRV.W	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的字循环右移	LoongEXT32
X86RCRV.H	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的半字循环右移	LoongEXT32
X86RCRV.B	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的字节循环右移	LoongEXT32
X86RCLV.D	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的双字循环左移	LoongEXT64
X86RCLV.W	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的字循环左移	LoongEXT32
X86RCLV.H	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的半字循环左移	LoongEXT32
X86RCLV.B	以 x86 方式只设置 EFLAG 的可变移位量的带 CF 位的字节循环左移	LoongEXT32
X86MFFLAG	以 x86 方式提取 EFLAG 标志位的值	LoongEXT32
X86MTFLAG	以 x86 方式修改 EFLAG 标志位的值	LoongEXT32
X86J	以 X86 方式根据 EFLAG 值跳转	LoongEXT32
X86LOOP	以 X86 方式根据 EFLAG 值循环	LoongEXT32
SETTM	x86 浮点栈模式置位	LoongEXT32
CLRTM	x86 浮点栈模式清除	LoongEXT32
INCTOP	x86 浮点栈顶指针加 1	LoongEXT32
DECTOP	x86 浮点栈顶指针减 1	LoongEXT32
MTTOP	写 x86 浮点栈顶指针	LoongEXT32
MFTOP	读 x86 浮点栈顶指针	LoongEXT32
SETTAG	判断并置位寄存器	LoongEXT32
CVT.D.LD	扩展双精度转化为双精度	LoongEXT32
CVT.LD.D	双精度转化为扩展双精度低位	LoongEXT32
CVT.UD.D	双精度转化为扩展双精度高位	LoongEXT32

二进制翻译加速指令(ARM)

表 2-30 龙芯扩展 ARM 二进制翻译加速指令

指令助记符	指令功能简述	ISA 兼容类别
ARMADC	带进位字加，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32
ARMADD	字加，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32
ARMSBC	带进位字减，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32
ARMSUB	字减，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32
ARMAND	字逻辑位与，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMBIC	字逻辑位非与，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMMOV	字移动，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32
ARMMVN	字取反移动，以 ARM 方式条件执行的只设置 EFLAG	LoongEXT32

指令助记符	指令功能简述	ISA 兼容类别
ARMMVLO32	LO 寄存器低 32 位移动至通用寄存器，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMMVHI32	HI 寄存器低 32 位移动至通用寄存器，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMMVACC64	HI 寄存器低 32 位与 LO 寄存器低 32 位拼接成 64 位，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMOR	字逻辑位或，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMORN	字逻辑位非或，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMXOR	字逻辑位异或，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSLL	字左移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSLLV	可变移位量字左移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSRL	字逻辑右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSRLV	可变移位量字逻辑右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSRA	字算术右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMSRAV	可变移位量字算术右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMROTR	字循环右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMROTRV	可变移位量字循环右移，以 ARM 方式条件执行只设置 EFLAG	LoongEXT32
ARMRRX	以 ARM 方式条件执行的只设置 EFLAG 的带进位字循环右移一位	LoongEXT32
ARMFCMP.F32	以 ARM FCMP.F32 指令方式进行浮点比较，置 FCR1 EFLAGS	LoongEXT32
ARMFCMP.F64	以 ARM FCMP.F64 指令方式进行浮点比较，置 FCR1 EFLAGS	LoongEXT32
ARMFCMPE.F32	以 ARM FCMP.F32 指令方式进行浮点比较，置 FCR1 EFLAGS	LoongEXT32
ARMFCMPE.F64	以 ARM FCMP.F64 指令方式进行浮点比较，置 FCR1 EFLAGS	LoongEXT32
ARMMOVE	通用寄存器之间以 ARM 方式根据 EFLAG 条件移动	LoongEXT32
ARMMFHI	HI 以 ARM 方式根据 EFLAG 条件移动至通用寄存器	LoongEXT32
ARMMFLO	LO 以 ARM 方式根据 EFLAG 条件移动至通用寄存器	LoongEXT32
ARMMFFCR	以 ARM 方式根据 EFLAG 条件将 FCR1 EFLAGS 移动至 EFLAGS	LoongEXT32
ARMFMOV.S	浮点寄存器之间以 ARM 方式根据 EFLAG 条件移动单精度数	LoongEXT32
ARMFMOV.D	浮点寄存器之间以 ARM 方式根据 EFLAG 条件移动双精度数	LoongEXT32
ARMJ	以 ARM 方式根据 EFLAG 值跳转	LoongEXT32
GSLDPC	将 PC+8 值取到通用寄存器	LoongEXT32
ARMMFFLAG	以 ARM 方式提取 EFLAG 标志位的值	LoongEXT32
ARMMTFLAG	以 ARM 方式修改 EFLAG 标志位的值	LoongEXT32

64 位多媒体加速指令

表 2-31 龙芯扩展 64 位多媒体加速指令

指令助记符	指令功能简述	ISA 兼容类别
PADDSH	四个 16 位有符号整数加，有符号饱和	LoongEXT32
PADDUSH	四个 16 位无符号整数加，无符号饱和	LoongEXT32
PADDH	四个 16 位数加	LoongEXT32
PADDW	两个 32 位数加	LoongEXT32
PADDSB	八个 8 位有符号整数加，有符号饱和	LoongEXT32

指令助记符	指令功能简述	ISA 兼容类别
PADDUSB	八个 8 位无符号整数加，无符号饱和	LoongEXT32
PADDB	八个 8 位数加	LoongEXT32
PADDD	64 位数加	LoongEXT32
PSUBSH	四个 16 位有符号整数减，有符号饱和	LoongEXT32
PSUBUSH	四个 16 位无符号整数减，无符号饱和	LoongEXT32
PSUBH	四个 16 位数减	LoongEXT32
PSUBW	两个 32 位数减	LoongEXT32
PSUBSB	八个 8 位有符号整数减，有符号饱和	LoongEXT32
PSUBUSB	八个 8 位无符号整数减，无符号饱和	LoongEXT32
PSUBB	八个 8 位数减	LoongEXT32
PSUBD	64 位数减	LoongEXT32
PSHUFH	Shuffle 四个 16 位数	LoongEXT32
PACKSSWH	32 位有符号整数转化成 16 位，有符号饱和	LoongEXT32
PACKSSHB	16 位有符号整数转化成 8 位，有符号饱和	LoongEXT32
PACKUSHB	16 位有符号整数转化成 8 位，无符号饱和	LoongEXT32
PANDN	fs 取非后与 ft 按位与	LoongEXT32
PUNPCKLHW	Unpack 低 16 位数	LoongEXT32
PUNPCKHHW	Unpack 高 16 位数	LoongEXT32
PUNPCKLBH	Unpack 低 8 位数	LoongEXT32
PUNPCKHBH	Unpack 高 8 位数	LoongEXT32
PINSRH_0	ft 低 16 位数插入到 fs 低 0 个 16 位	LoongEXT32
PINSRH_1	ft 低 16 位数插入到 fs 低 1 个 16 位	LoongEXT32
PINSRH_2	ft 低 16 位数插入到 fs 低 2 个 16 位	LoongEXT32
PINSRH_3	ft 低 16 位数插入到 fs 低 3 个 16 位	LoongEXT32
PAVGH	四个 16 位无符号整数取平均值	LoongEXT32
PAVGB	八个 8 位无符号整数取平均值	LoongEXT32
PMAXSH	四个 16 位有符号整数取较大值	LoongEXT32
PMINSH	四个 16 位有符号整数取较小值	LoongEXT32
PMAXUB	八个 8 位无符号整数取较大值	LoongEXT32
PMINUB	八个 8 位无符号整数取较小值	LoongEXT32
PCMPEQW	两个 32 位数相等比较	LoongEXT32
PCMPGTW	两个 32 位有符号整数大于比较	LoongEXT32
PCMPEQH	四个 16 位数相等比较	LoongEXT32
PCMPGTH	四个 16 位有符号整数大于比较	LoongEXT32
PCMPEQB	八个 8 位数相等比较	LoongEXT32
PCMPGTB	八个 8 位有符号整数大于比较	LoongEXT32
PSLLW	两个 32 位数逻辑左移	LoongEXT32
PSLLH	四个 16 位数逻辑左移	LoongEXT32
PMULLH	四个 16 位有符号整数相乘，取结果低 16 位	LoongEXT32
PMULHH	四个 16 位有符号整数相乘，取结果高 16 位	LoongEXT32
PMULUW	低 32 位无符号整数相乘，存 64 位结果	LoongEXT32

指令助记符	指令功能简述	ISA 兼容类别
PMULHUH	四个 16 位无符号整数相乘，取结果高 16 位	LoongEXT32
PSRLW	两个 32 位数逻辑右移	LoongEXT32
PSRLH	四个 16 位数逻辑右移	LoongEXT32
PSRAW	两个 32 位数算术右移	LoongEXT32
PSRAH	四个 16 位数算术右移	LoongEXT32
PUNPCKLWD	低 32 位数组合成 64 位数	LoongEXT32
PUNPCKHWD	高 32 位数组合成 64 位数	LoongEXT32
PASUBUB	八个 8 位无符号整数相减并取绝对值	LoongEXT32
PEXTRH	fs 某 16 位拷贝到 fd 低 16 位，fd 高位补 0	LoongEXT32
PMADDHW	四个 16 位有符号数相乘，低高位分别累加	LoongEXT32
BIADD	多字节累加求和	LoongEXT32
PMOVMSKB	字节符号位提取	LoongEXT32
GSXOR	fs 与 ft 逻辑位异或	LoongEXT32
GSNOR	fs 与 ft 逻辑位或非	LoongEXT32
GSAND	fs 与 ft 逻辑位与	LoongEXT32
GSADDU	fs 与 ft 定点无符号字加	LoongEXT32
GSOR	fs 与 ft 定点逻辑位或	LoongEXT32
GSADD	fs 与 ft 定点字加	LoongEXT32
GSDADD	fs 与 ft 定点双字加	LoongEXT32
GSSEQU	fs 与 ft 定点数相等比较	LoongEXT32
GSSEQ	fs 与 ft 定点数相等比较	LoongEXT32
GSSUBU	fs 与 ft 定点无符号字减	LoongEXT32
GSSUB	fs 与 ft 定点字减	LoongEXT32
GSDSUB	fs 与 ft 定点双字减	LoongEXT32
GSSLTU	fs 与 ft 定点无符号定点数小于比较	LoongEXT32
GSSLT	fs 与 ft 定点定点数小于比较	LoongEXT32
GSSLL	fs 与 ft 定点逻辑左移字	LoongEXT32
GSDSLL	fs 与 ft 定点逻辑左移双字	LoongEXT32
GSSRL	fs 与 ft 定点逻辑右移字	LoongEXT32
GSDSRL	fs 与 ft 定点逻辑右移双字	LoongEXT32
GSSRA	fs 与 ft 定点算术右移字	LoongEXT32
GSDSRA	fs 与 ft 定点算术右移双字	LoongEXT32
GSSLEU	fs 与 ft 定点无符号定点数小于等于比较	LoongEXT32
GSSLE	fs 与 ft 定点定点数小于等于比较	LoongEXT32

杂项指令

表 2-32 龙芯扩展杂项指令

指令助记符	指令功能简述	ISA 兼容类别
CTZ	字尾随 0 个数	LoongEXT32
CTO	字尾随 1 个数	LoongEXT32
DCTZ	双字尾随 0 个数	LoongEXT64

指令助记符	指令功能简述	ISA 兼容类别
DCTO	双字尾随 1 个数	LoongEXT64
CAMPV	查询查找表的 RAM 表项值	LoongEXT32
CAMPI	查询查找表的表项 index	LoongEXT32
CAMWI	填写查找表	LoongEXT32
RAMRI	读取查找表的 RAM 内容	LoongEXT32

3 处理器运行模式

GS464E 的运行模式兼容 MIPS64 规范，包含 2 种运行模式，分别是：调试模式(Debug Mode)和根模式(Root Mode)。其中调试模式主要对应 EJTAG 例外处理程序的运行环境；根模式对应真实主机上操作系统及其上软件的运行环境。其中，根模式又可进一步划分为根-核心模式(Root-Kernel Mode)、根-监管模式(Root-Supervisor Mode)和根-用户模式(Root-User Mode)。所有的运行模式互相独立，也就是说任一时刻，处理器只能存在于某一种运行模式中。

上述 4 种模式中，根-监管模式在实际中很少使用，MIPS 规范亦未对其内涵做充分定义，因此本手册后续描述对于这种模式将只做简单说明。不建议编程人员利用根-监管模式构建软件，如确有需要，请直接参阅 MIPS 规范。

3.1 处理器运行模式定义

表 3-1 给出处理器各模式的判定依据。

表 3-1 处理器模式判定依据

<table>
<tr><th colspan="4">Root CP0</th><th rowspan="2">模式</th></tr>
<tr><th>Debug.DM</th><th>Status.ERL</th><th>Status.EXL</th><th>Status.KSU</th></tr>
<tr><td>1</td><td colspan="3">Don’t care</td><td>调试模式</td></tr>
<tr><td rowspan="4">0</td><td>1</td><td colspan="2">Don’t care</td><td rowspan="2">根-核心模式</td></tr>
<tr><td rowspan="3">0</td><td>1</td><td>Don’t care</td></tr>
<tr><td rowspan="3">0</td><td>00</td><td rowspan="2"></td></tr>
<tr><td>01</td><td>根-监管模式</td></tr>
<tr><td colspan="3">Don’t care</td><td>10</td><td>根-用户模式</td></tr>
<tr><td colspan="3">Don’t care</td><td>11</td><td>无意义</td></tr>
</table>

3.1.1 调试模式

调试模式最有最高优先级。调试模式下软件可以操作所有的处理器资源，包括改变虚实地址映射关系、控制系统环境和进程切换等。

3.1.2 根-核心模式

根-核心模式下软件可以操作所有的处理器资源，包括改变虚实地址映射关系、控制系统环境和进程切换等。处理器上电复位后进入根-核心模式。

3.1.3 根-用户模式

根-用户模式下，软件不允许访问处理器的特权敏感资源，但可以执行非特权指令，使用通用寄存器和浮点寄存器，所有访存落在一个扁平均一虚地址空间上。普通的用户程序均运行在根-用户模式下。

4 内存管理

4.1 基本概念

4.1.1 地址空间

地址空间是指某一特定寻址模式下所能覆盖的所有地址范围。MIPS64 架构中包含一个 64 位地址空间以及一个 32 位地址空间，后者同时映射为前者的一个子集。

4.1.2 段及段大小(SEGBITS)

段是地址空间的一个子集，同一段内的地址空间具有一致的映射方式和访问属性。以 MIPS64 规范的定义为例，其 32 位地址空间被划分为一系列大小为 2^{29} 或 2^{31} 字节的段，其 64 位地址空间理论上可以支持不超过 2^{62} 字节大小的段。实际中无需实现这么大的段，实际实现的段大小(SEGBITS)决定了地址空间被划分为一系列大小为 $2^{SEGBITS}$ 字节的段。

4.1.3 物理地址大小(PABITS)

物理地址大小(PABITS)决定了处理器实际支持的物理地址空间的大小为 2^{PABITS} 字节。

4.1.4 映射地址(Mapped Address)与非映射地址(Unmapped Address)

“映射地址(Mapped Address)”的地址是指该地址需要通过 TLB 进行虚实地址转换。“非映射地址(Unmapped Address)”是指该地址不需要经过 TLB 进行虚实地址转换，虚地址被直接线性映射至物理地址的最低部分。

4.2 主机虚地址空间

4.2.1 主机地址空间划分与访问控制

表 4-1 给出主机地址空间的划分，并对于各地址段的合法性判定和地址映射方式进行了定义。需要注意的时，在主机地址空间中，SEGBITS 始终为 48。

表 4-1 主机地址空间划分与访问控制

段名称	地址范围	合法性判定和地址映射方式		
		用户模式	监管模式	核心模式
kseg3	0xFFFF.FFFF.FFFF.FFFF ~ 0xFFFF.FFFF.E000.0000	非法地址段	非法地址段	映射地址段 TLB 重填例外类型： TLB (Status.KX=0) XTLB (Status.KX=1)。 当 Debug.DM=1 时，有关地址空间的特殊处理请参看 4.2.5 小节

段名称	地址范围	合法性判定和地址映射方式		
		用户模式	监管模式	核心模式
ksseg sseg	0xFFFF.FFFF.DFFF.FFFF ~ 0xFFFF.FFFF.C000.0000	非法地址段	映射地址段 TLB 重填例外类型： TLB (Status.KX=0) XTLB (Status.KX=1)。	映射地址段 TLB 重填例外类型： TLB (Status.KX=0) XTLB (Status.KX=1)。
kseg1	0xFFFF.FFFF.BFFF.FFFF ~ 0xFFFF.FFFF.A000.0000	非法地址段	非法地址段	非映射地址段 请进一步参看 4.2.2 小节
kseg0	0xFFFF.FFFF.9FFF.FFFF ~ 0xFFFF.FFFF.8000.0000	非法地址段	非法地址段	非映射地址段 请进一步参看 4.2.2 小节
	0xFFFF.FFFF.7FFF.FFFF ~ 0xC000.FFFF.8000.0000	非法地址段	非法地址段	非法地址段
xkseg	0xC000.FFFF.7FFF.FFFF ~ 0xC000.0000.0000.0000	非法地址段	非法地址段	Status.KX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。
xkphys	0xBFFF.FFFF.FFFF.FFFF ~ 0x8000.0000.0000.0000	非法地址段	非法地址段	Status.KX=0 时为非法地址段； 否则合法，其内部合法性判定及采用的地址映射请参看 4.2.3 节
	0x7FFF.FFFF.FFFF.FFFF ~ 0x4001.0000.0000.0000	非法地址段	非法地址段	非法地址段
xksseg xsseg	0x4000.FFFF.FFFF.FFFF ~ 0x4000.0000.0000.0000	非法地址段	Status.SX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。	Status.SX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。
	0x3FFF.FFFF.FFFF.FFFF ~ 0x0001.0000.0000.0000	非法地址段	非法地址段	非法地址段
xkuseg xsuseg xuseg	0x0000.FFFF.FFFF.FFFF ~ 0x0000.0000.8000.0000	Status.UX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。	Status.UX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。	Status.UX=0 时为非法地址段； 否则合法，为映射地址段 TLB 重填例外类型： XTLB。

段名称	地址范围	合法性判定和地址映射方式		
		用户模式	监管模式	核心模式
kuseg suseg useg	0x0000.0000.7FFF.FFFF ~ 0x0000.0000.0000.0000	映射地址段 TLB 重填例外类型： TLB (Status.UX=0) XTLB (Status.UX=1)。	映射地址段 TLB 重填例外类型： TLB (Status.KX=0) XTLB (Status.KX=1)。	Status.ERL=1 时为非映射地址段，请进一步参看 4.2.4 小节 Status.UX=0 时为映射地址段，TLB 重填例外类型： TLB (Status.UX=0) XTLB (Status.UX=1)

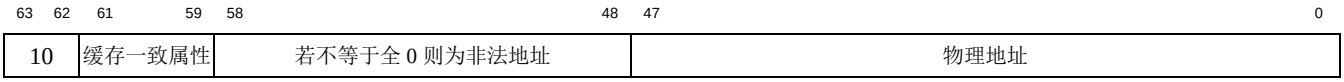
4.2.2 主机地址空间 kseg0 段与 kseg1 段的地址转换、可缓存性与缓存一致属性

主机地址空间的 kseg0 段与 kseg1 段直接映射到物理地址空间的最低 0.5G(2^{29})字节，即虚地址 0xFFFF.FFFF.A000.000 ~ 0xFFFF.FFFF.BFFF.FFFF 映射至物理地址 0x0000.0000.0000.0000 ~ 0x0000.0000.1FFF.FFFF，虚地址 0xFFFF.FFFF.8000.000 ~ 0xFFFF.FFFF.9FFF.FFFF 也映射至物理地址 0x0000.0000.0000.0000 ~ 0x0000.0000.1FFF.FFFF。kseg0 段的缓存一致属性由 Config.K0 域决定，具体定义请参看 110 页 7.28 小节。kseg1 段永远为非缓存属性(Uncached)。

4.2.3 主机地址空间 xkphys 段的地址转换、可缓存性与缓存一致属性

主机地址空间的 xkphys 段采用非映射方式，其包含 8 个子地址段，每个子地址段大小为 2^{48} 字节。xkphys 段的虚地址解析方式如图 4-1 所示。虚地址的[58:48]必须为全 0，否则为非法地址。虚地址的[47:0]不经过 TLB 或其它任何翻译过程。直接作为物理地址。虚地址的[61:59]位用于定义所对应的子地址段的缓存一致属性，其采用的编码方式的定义请参看 83 页表 7-6。

图 4-1 xkphys 段虚地址解析方式



4.2.4 主机地址空间 kuseg 段在 Status.ERL=1 时的地址转换

当 Status.ERL=1 时，kuseg 段为非映射地址段，同时其缓存一致属性为非缓存，类似于 kseg1 段。这一特性是为了在处理 Cache 错例外时，软件在保存上下文时可以利用通用寄存器 R0 作为基址寄存器将其它通用寄存器存入内存，同时由于 Cache 中存在错误，所以访存操作都不再进行缓存。

4.2.5 主机地址空间 kseg3 段在 Debug.DM=1 时的特殊处理

当处理器处于调试模式时（Debug.DM=1），kseg3 段中虚地址 0xFFFF.FFFF.FF20.0000 ~ 0xFFFF.FFFF.FF3F.FFFF 的地址范围将作为特殊的内存地址映射区域——EJTAG 的 dseg 段。有关 EJTAG 中 dseg 段的详细描述请参看第**错误!未找到引用源。**章。

4.2.6 用户模式下 Status.UX=0 时数据访问虚地址的特殊处理

用户模式下，在 64 位 MIPS 处理器上兼容运行 32 位程序时，对于数据访问的虚地址要进行特殊处理。这是因为一个在 32 位 MIPS 处理器上能够得到合法地址的计算过程在 64 位 MIPS 处理器上可能会产生非预期的效果。例如，如下指令序列：

```
la r1, 0x80000000
```

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

在 32 位 MIPS 处理器上执行时，lw 指令的虚地址为 0x80000000 + 0xFFFFFFFC = 0x7FFFFFFF。所得地址仍落在 kuseg 段。但是这段代码在 64 位 MIPS 处理器上执行时，lw 指令的虚地址位 0xFFFFFFFF80000000 + 0xFFFFFFFFFFFFFFFC = 0xFFFFFFFF7FFFFFFC。所得的地址已经不在 kuseg 段，将会导致地址错例外发生。为了保持 64 位处理器对于 32 位程序的兼容，当 Status.UX=0 时，数据访问虚地址的计算进行了特殊处理。具体方式是，地址运算仍采用符号扩展后的两个 64 位数相加，但是将所得的结果的高 32 位丢弃，由结果的第 31 位符号扩展至结果虚地址的 63..32 位。这样特殊处理之后的虚地址结果才用于地址的合法性检查、TLB 映射等操作。正常的取指操作不会涉及该问题，因为 32 位用户模式下合法 PC 的第 31 位一定为 0，不会出现上面所举的违例情况。

4.3 基于 TLB 的虚实地址映射

TLB 是处理器中存放操作系统页表信息的一个临时缓存，用于加速映射地址空间上取指及访存操作的虚实地址转换过程。

4.3.1 TLB 层次结构

GS464E 中实现了两级 TLB。第一级 TLB 是取指和访存部件中各自包含的一个容量较小的全相联查找的 TLB，分别称之为 ITLB 和 DTLB；第二级 TLB 容量较大，包含一个全相联和一个八路组相联的查找表，称之为 JTLB。

JTLB 的所有内容软件可见，表项的装填替换由软件管理。ITLB 和 DTLB 的所有内容软件不可见，其表项的装填替换由硬件维护。在处理器运行时，DTLB 与 JTLB 中存放的页表内容始终为包含关系，该包含关系由硬件自动维护。但是 ITLB 与 JTLB 各自存放的内容无明确包含与互斥关系，也就是说，软件无法通过维护 JTLB 中的内容来确定 ITLB 中存放的信息。在一般运行过程中，ITLB 与 JTLB 中存放信息不维持包含关系对于软件的正确性没有影响。但是，如果操作系统试图修改一项已经存在的页表信息，并且该页表项对应的地址空间上存放有程序代码，那么系统软件必须对 Diag.ITLB 位写 1 来显式地清空 ITLB 的所有内容，以保证 ITLB 中一定不再有该页表项修改前的内容。

ITLB 和 DTLB 均采用全相联查找表结构。ITLB 和 DTLB 中存放的表项信息均来自于 JTLB，其中 DTLB 存放的表项信息与 JTLB 中格式完全一致，而 ITLB 中每一项只存放一个页表而非一对奇偶相邻页表的信息。因为 ITLB 和 DTLB 的状态替换并不需要软件参与，因此其表项的具体格式不在此展开。

4.3.2 JTLB 组织结构

从软件的角度看，JTLB 包含一张全相联查找表和一张多路组相联的查找表，前者因支持不同表项采用不同的页大小而称为可变页大小 TLB(Variable-Page-Size TLB)，简称 VTLB；后者在同一时刻所有表项页大小相同，称为固定页大小 TLB(Fixed-Page-Size TLB)，简称 FTLB。在虚实地址转换过程中，VTLB 和 FTLB 同时查找。相应地，软件需保证 VTLB 和 FTLB 不会出现多项命中的情况，否则处理器行为将不可知。

VTLB 的组织形式及操作方式与 MIPS 传统的全相联 TLB 非常相似，在 GS464E 中为 64 项。如果 GSConfig.VTLBOnly 位置为 1，则禁用 FTLB 的所有功能，仅保留 VTLB。该配置下，龙芯 3A1000 芯片上已有操作系统中关于 TLB 管理部分的无需修改即可在龙芯 3A2000 芯片上正确执行。

FTLB 组织为 8 路组相联结构，每一路包含 128 项，共 1024 项，每一项保存 2 个页表信息，因此最多可存放 2048 个页表信息。在进行查找的过程中，硬件将提取虚地址的[(17+Config4.FTLBPageSize) : (11+Config4.FTLBPageSize)]位作为索引信息，对各路中相同索引位置项的内容进行比较，以决定是否有匹配项。

4.3.3 JTLB 表项

VTLB 和 FTLB 表项的格式基本一致，区别仅在与 VTLB 每个表项均包含 PageMask 信息，而 FTLB 因为是同一页大小故不保持 PageMask 信息。每一个 JTLB 表项包含两个部分：比较部分和物理转换部分。

表项的比较部分包括：

- 页表无效位(EHINV)。当该位为 1 时，该页表项不参与查找匹配。
- 地址空间号(MID)。用以标识该页表项地址处于哪个地址空间。
- 虚拟处理器号及其掩码(VPID, VPMSK)。VPID&VPMSK 是该页表项地址所处的虚拟处理器号。
- 虚地址区域标识(R)和虚页号(VPN2)。在 MIPS 架构中，每一个页表项存放了相邻的一对奇偶相邻页表信息，所以 TLB 页表项中存放虚页号的是系统中虚页号/2 的内容，即虚页号的最低位不予存放，仅用于查找时决定是选择奇数号页还是偶数号页的物理转换信息。
- 地址空间标识(ASID)和全局标志位(G)。地址空间标识用于区分不同进程中的同样的虚地址，操作系统为每个进程分配唯一的 ASID，TLB 在进行查找时除地址信息外一致外，还需要比对 ASID 信息。当操作系统需要在所有进程间共享同一虚拟地址时，可以设置 TLB 页表项中的 G 位，G 位置 1 后 TLB 查找时将不再进行 ASID 是否一致的检查。
- 地址页掩码(Mask)。地址掩码用于控制该页表项中存放的页表大小。GS464E 支持 4KB 至 1GB，以 4 的倍数递增的页大小。对于 VTLB，每一项都存有 Mask 信息，因此不同项可以对应不同的页大小。对于 FTLB，所有项采用统一的 Mask 信息，由 Config4.FTLBPageSize 域决定。

上述各个域的解释可进一步参看 7.19 节 EntryHi 寄存器 (CP0 Register 10, Select 0)、7.4 节 EntryLo0 和 EntryLo1 寄存器 (CP0 Register 2 and 3, Select 0)和 7.7 节 PageMask 寄存器 (CP0 Register 5, Select 0)中的相关描述。

表项的物理转换部分存有一对奇偶相邻页表的物理转换信息，每一个页的转换信息包括：

- 物理页号(PFNX & PFN)。
- 有效位(V)。
- 脏位(D)。页是否可写的控制位，而不是页是否被写入了脏数据的状态位。
- 读禁止位(RI)。
- 执行禁止位(XI)。
- 内核执行保护位(K)，仅在 64 位模式下有意义，32 位模式下请将 GSConfig.KE 恒置为 0 以保证 K 位不产生效果。
- Cache 属性(C)。

上述各个域的解释可进一步参看 7.4 节 EntryLo0 和 EntryLo1 寄存器 (CP0 Register 2 and 3, Select 0)中的相关描述。

4.3.4 TLB 的软件管理

GS464E 仍沿袭了传统 MIPS 架构对于 TLB 的管理方式，即采用软件主导、软硬件相结合的方式管理 TLB。MIPS 规范 release 5 及之后的版本中所新增的硬件页表遍历查找（Hardware Page Table Walking）功能在 GS464E 中不予实现。GS464E 在 MIPS 规范基础之上提供了一套专门用于页表遍历查找的特权访存指令用于加速这一过程，本小节后半部分将对其做简要介绍。

相关例外

TLB 进行虚实地址转换过程由硬件自动完成，但是当 TLB 中没有匹配项，或者尽管匹配但页表项无效或访问非法时，就需要触发例外，交由操作系统内核或其它监管程序，由软件进一步处理，对 TLB 的内容进行维护，或对程序执行的合法性做最后裁定。GS464E 中与 TLB 管理相关的例外有：

1. TLB 重填例外
2. XTLB 重填例外
3. TLB 无效例外
4. TLB 修改例外
5. 不可执行例外
6. 不可读取例外

相关 CP0 寄存器

表 4-2 列举了与 TLB 管理相关的 CP0 寄存器，各寄存器的详细介绍请参看第 7 章中的对应小节。

表 4-2 TLB 管理相关 CP0 寄存器

Reg.	Sel.	寄存器名称	功能定义	索引
0	0	Index	VTLB 与 FTLB 访问指定索引寄存器	第 79 页 7.2 节
1	0	Random	VTLB 与 FTLB 访问随机索引寄存器	第 80 页 7.3 节
2	0	EntryLo0	VTLB 与 FTLB 表项低位内容中与偶数虚页相关部分	第 81 页 7.4 节
3	0	EntryLo1	VTLB 与 FTLB 表项低位内容中与奇数虚页相关部分	第 81 页 7.4 节
4	0	Context	指向内存中页表项的指针	第 84 页 7.5 节
5	0	PageMask	VTLB 页表大小控制	第 86 页 7.7 节
5	1	PageGrain	1KB 小页等页表属性控制	第 87 页 7.8 节
5	5	PWBase	页表基地址寄存器	第 88 页 7.9 节
5	6	PWField	配置各级页表地址索引位置	第 89 页 7.10 节
5	7	PWSize	配置各级页表指针大小	第 90 页 7.11 节
6	0	Wired	控制 VTLB 中固定项数目	第 91 页 7.12 节
6	6	PWCtl	控制多级页表配置	第 92 页 7.13 节
8	0	BadVAddr	记录最新地址相关例外的出错地址	第 94 页 7.15 节
9	7	PGD	页表指针寄存器	第 97 页 7.18 节
10	0	EntryHi	VTLB 与 FTLB 表项高位内容	第 98 页 7.19 节
20	0	XContext	扩展地址模式下页表指针	第 122 页 7.36 节
22	0	Diag	龙芯扩展诊断控制寄存器	第 123 页 7.37 节

相关特权指令

表 4-3 列举了与 TLB 管理相关的特权指令。各指令的详细定义请参考《MIPS® Architecture For Programmers Volume II-A: The MIPS64® Instruction Set》(Rev5.03)以及《龙芯指令集手册（卷 II-a）——自定义通用扩展指令（上册）》(Rev1.00)。

表 4-3 TLB 管理相关特权指令

指令助记符	指令简单描述

指令助记符	指令简单描述
TLBP	在 TLB 中搜索匹配项
TLBR	读索引的 TLB 表项
TLBWI	写索引的 TLB 表项
TLBWR	写随机的 TLB 表项
TLBINVF	将所有 TLB 表项无效
LWDIR	32 位模式下页表目录项装载指令
LWPTE	32 位模式下页表页表项装载指令
LDDIR	64 位模式下页表目录项装载指令
LDPTE	64 位模式下页表页表项装载指令

GS464E 在 EntryHi 寄存器中实现了 EHINV 域。当 EntryHi.EHINV 置为 1 时，执行 TLBWI 指令会将 Index 所指定的 TLB 表项无效。

VTLB 与 FTLB 采用统一的软硬件交互接口，即使用同一套 CP0 寄存器和同样的 TLB 特权指令。这里强调以下需要注意的几点：

- VTLB 可通过 Wired 寄存器保留一些不可随机替换项，而 FTLB 中没有不可随机替换项。
- 当使用 TLBWR 指令随机填入 TLB 表项时。如果此时 Pagemask 寄存器中页大小与 FTLB 的固定页大小不同时，表项被随机填入 VTLB 中；如果相同，表项被随机填入 FTLB 中。
- VTLB 中随机填入的位置由 Random 寄存器的值决定；FTLB 中随机填入的位置由处理器中一个单独的硬件伪随机数生成器决定。
- 当使用 TLBRI、TLBWI 指令读、写 JTLB 时，Index 寄存器中值为 0~63 对应 VTLB 的第 0~63 项，值为 64~1087 对应 FTLB 第 0 路的第 0~127 项、第 1 路的第 0~127 项、……、第 7 路的第 0~127 项。当使用 TLBP 指令进行软件查找时，存放在 Index 寄存器中的结果也遵循上述对应关系。

4.3.5 TLB 初始化与清空

建议使用 TLBINVF 指令将 TLB 所有项无效的方式来初始化或清空 TLB。也可以利用 EntryHi 寄存器中的 EHINV 域，循环执行 TLBWI 指令将 TLB 中所有项逐一无效。

FTLB 中表项的位置与其存放的虚地址之间存在对应关系，因此采用对 TLB 表项中写入某一特定地址的传统初始化 TLB 的方式可能无法保证初始化正确。当 GSConfig.VTLBOnly=0 时，即 FTLB 被使能时，软件必须使用上面介绍的两种方式进行初始化或清空 TLB。传统初始化 TLB 的方式本手册不做介绍，感兴趣的读者可以参考《MIPS® Architecture For Programmers Volume III: The MIPS64® and microMIPS64™ Privileged Resource Architecture》(Rev5.03)的 4.11.3 节。

4.3.6 基于 TLB 的虚实地址转换过程

这里只介绍基于软件可见的 TLB 所进行的虚实地址转换过程。处理器硬件同时对 VTLB 和 FTLB 进行查找，下面以伪码形式介绍时的先查 VTLB 后查 FTLB 的过程仅为描述方便。系统软件需要保证同一虚地址不可在 VTLB 和 FTLB 中出现多项命中。

```
// va -- 待查找虚地址
// mid -- 待查找虚地址所属 MID

/* VTLB 查找过程 */
vtlb_found ← 0
```

```
for i = 0 to 63 step 1
  if ((VTLB[i].EHINV == 0) &&
       (VTLB[i].MID == mid) &&
       ((VTLB[i].VPID & VTLB[i].VPMSK) == (Diag.VPID & Diag.VPMSK)) &&
       (VTLB[i].G || (VTLB[i].ASID == EntryHi.ASID)) &&
       (VTLB[i].R==va[63:62]) &&
       ((VTLB[i].VPN2[34:27] & ~VTLB[i].VPMSK) == (va[47:40] & ~Diag.VPMSK)) &&
       (VTLB[i].VPN2[26:18] == va[39:31]) &&
       ((VTLB[i].VPN2[17:0] & ~VTLB[i].MASK) == (va30..13 & ~VTLB[i].MASK))) then
    if (!vtlb_found) then
      vtlb_found ← 1
      case VTLB[i].MASK
        0b00 0000 0000 0000 0000: vtlb_evenoddbit ← 12 //4KB page
        0b00 0000 0000 0000 0011: vtlb_evenoddbit ← 14 //16KB page
        0b00 0000 0000 0000 1111: vtlb_evenoddbit ← 16 //64KB page
        0b00 0000 0000 0011 1111: vtlb_evenoddbit ← 18 //256KB page
        0b00 0000 0000 1111 1111: vtlb_evenoddbit ← 20 //1MB page
        0b00 0000 0011 1111 1111: vtlb_evenoddbit ← 22 //4MB page
        0b00 0000 1111 1111 1111: vtlb_evenoddbit ← 24 //16MB page
        0b00 0011 1111 1111 1111: vtlb_evenoddbit ← 26 //64MB page
        0b00 1111 1111 1111 1111: vtlb_evenoddbit ← 28 //256MB page
        0b11 1111 1111 1111 1111: vtlb_evenoddbit ← 30 //1GB page
        otherwise: UNDIFINED
      endcase
      if va[vtlb_evenoddbit] == 0 then
        vtlb_pfn ← VTLB[i].PFN0
        vtlb_v ← VTLB[i].V0
        vtlb_c ← VTLB[i].C0
        vtlb_d ← VTLB[i].D0
        vtlb_ri ← VTLB[i].RI0
        vtlb_xi ← VTLB[i].XI0
        vtlb_k ← VTLB[i].K0
      else
        vtlb_pfn ← VTLB[i].PFN1
        vtlb_v ← VTLB[i].V1
        vtlb_c ← VTLB[i].C1
        vtlb_d ← VTLB[i].D1
        vtlb_ri ← VTLB[i].RI1
        vtlb_xi ← VTLB[i].XI1
        vtlb_k ← VTLB[i].K1
      endif
    else
      UNDIFINED
    endif
```

```
  endif
endfor

/* FTLB 查找过程 */
ftlb_found ← 0
case Config4.FTLBPageSize
  0d1 : idx ← va[18:12]; mask ← 0x00000
  0d2 : idx ← va[20:14]; mask ← 0x00003
  0d3 : idx ← va[22:16]; mask ← 0x0000f
  0d4 : idx ← va[24:18]; mask ← 0x0003f
  0d5 : idx ← va[26:20]; mask ← 0x000ff
  0d6 : idx ← va[28:22]; mask ← 0x003ff
  0d7 : idx ← va[30:24]; mask ← 0x00fff
  0d8 : idx ← va[32:26]; mask ← 0x03fff
  0d9 : idx ← va[34:28]; mask ← 0x0ffff
  0d10: idx ← va[36:30]; mask ← 0x3ffff
  otherwise: UNDIFINED
endcase
for set = 0 to 7 step 1
  FTLB[set][idx]
  if ((FTLB[set][idx].EHINV == 0) &&
      (FTLB[set][idx].MID == mid) &&
      ((FTLB[set][idx].VPID & FTLB[set][idx].VPMSK) == (Diag.VPID & Diag.VPMSK)) &&
      (FTLB[set][idx].G || (FTLB[set][idx].ASID == EntryHi.ASID)) &&
      (FTLB[set][idx].R==va[63:62]) &&
      ((FTLB[set][idx].VPN2[34:27]  &  ~FTLB[set][idx].VPMSK)  ==  (va[47:40]  &
  ~Diag.VPMSK)) &&
      (FTLB[set][idx].VPN2[26:18] == va[39:31]) &&
      ((FTLB[set][idx].VPN2[17:0] & ~mask) == (va30..13 & ~mask))) then
    if (!ftlb_found) then
      ftlb_found ← 1
      if va[ftlb_evenoddbit] == 0 then
        ftlb_pfn ← FTLB[set][idx].PFN0
        ftlb_v ← FTLB[set][idx].V0
        ftlb_c ← FTLB[set][idx].C0
        ftlb_d ← FTLB[set][idx].D0
        ftlb_ri ← FTLB[set][idx].RI0
        ftlb_xi ← FTLB[set][idx].XI0
        ftlb_k ← FTLB[set][idx].K0
      else
        ftlb_pfn ← FTLB[set][idx].PFN1
        ftlb_v ← FTLB[set][idx].V1
        ftlb_c ← FTLB[set][idx].C1
        ftlb_d ← FTLB[set][idx].D1
```

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

/* 合法性检查及物理地址生成 */
if (vtlb_found && ftlb_found) then
  UNDIFINED
elseif (vtlb_found) then
  if (!vtlb_v) then
    SignalException(TLBInvalid, reftype)
  endif
  if (vtlb_ri && (reftype == load)) then
    if (PageGrain.IEC) then
      SignalException(TLBRI, reftype)
    else
      SignalException(TLBInvalid, reftype)
    endif
  endif
  if (vtlb_xi && (reftype == fetch)) then
    if (PageGrain.IEC) then
      SignalException(TLBXI, reftype)
    else
      SignalException(TLBInvalid, reftype)
    endif
  endif
  if (!vtlb_d && (reftype == store)) then
    SignalException(TLBModified)
  endif
  PAddr ← vtlb_pfn[35:vtlb_evenoddbit-12] || va[vtlb_evenoddbit-1:0]
elseif (ftlb_found) then
  if (!ftlb_v) then
    SignalException(TLBInvalid, reftype)
  endif
  if (ftlb_ri && (reftype == load)) then
    if (PageGrain.IEC) then
      SignalException(TLBRI, reftype)
    else
      SignalException(TLBInvalid, reftype)
```

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

5 缓存的组织与管理

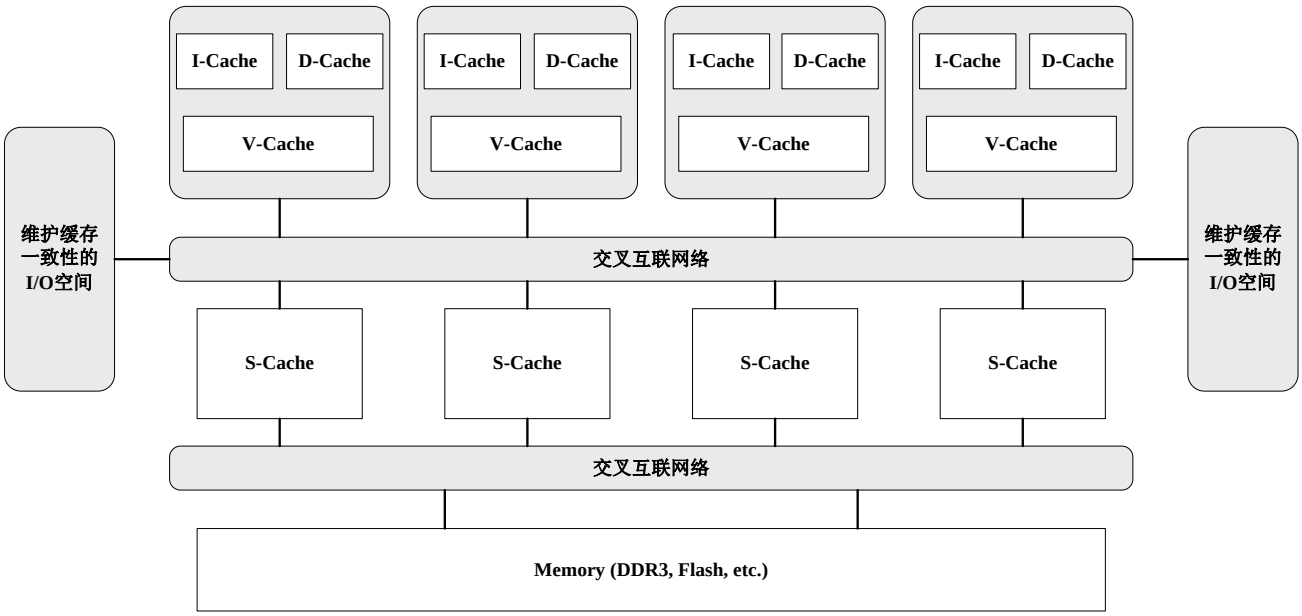
在 MIPS 架构下，处理器中各级缓存(Cache)是核心态软件可见的。对于缓存的某些操作（如缓存初始化、一致性维护等），需要软件参与缓存的管理。本章介绍 GS464E 的缓存组织与管理。本手册中不展开描述处理器缓存的一般性概念，如索引(Index)、标签(Tag)、缓存行(Cacheline)、组相联、缓存访问、缓存命中与缺失、虚地址索引物理地址标签(Virtual Index Physical Tag)的缓存、缓存替换等。若读者对相关概念尚不明晰，请参考计算体系架构的相关书籍，推荐《MIPS 体系结构透视》（第二版）。

5.1 处理器存储层次及各级缓存组织结构

5.1.1 处理器存储层次

龙芯 3A2000 芯片处理器采用含有三级缓存的存储层次，图 5-1 给出了其示意。

图 5-1 龙芯 3A2000 芯片处理器存储层次



根据各级缓存与处理器运算流水线的距离，由近及远，依次为：第一级的指令缓存(Instruction-Cache，I-Cache)和数据缓存(Data-Cache，D-Cache)，第二级的牺牲缓存(Victim-Cache，V-Cache)，第三级的共享缓存(Shared-Cache，S-Cache)。其中 I-Cache、D-Cache 和 V-Cache 为每个处理器核私有，S-Cache 为多核及 I/O 共享。处理器核通过芯片内部及芯片之间的互联网络访问 S-Cache。

I-Cache 中只存放处理器取指部件所需访问的内容，D-Cache 中只存放处理器访存部件所需访问的内容。V-Cache 和 S-Cache 均为混合 Cache，既存放指令也存放数据。

I-Cache 及 D-Cache 中的内容与 V-Cache 中的内容为互斥(exclusive)关系，即同一物理地址的内容存放在 I-Cache 或 D-Cache 中时就不再存放在 V-Cache 中。I-Cache、D-Cache 及 V-Cache 中的内容与 S-Cache 中的内容为包含(inclusive)关系，即同一物理地址的内容，只要其存放在 I-Cache、D-Cache 或 V-Cache 中，就一定在 S-Cache 中也能找到一个相同物理地址的备份。上述互斥与包含的关系均是从软件可观察到的角度进行描述，并不表明任意时刻数据在真实存储介质中的位置关系。

在运行过程中，I-Cache、D-Cache、V-Cache 和 S-Cache 之间的数据一致性由硬件维护。

当取指部件在 I-Cache 或访存部件 D-Cache 中查找不命中时，将首先查找 V-Cache。如果 V-Cache 命中，则将 V-Cache 中命中的缓存行填入 I-Cache 或 D-Cache，从 I-Cache 或 D-Cache 中替换出的缓存行（如果存在的话）将回填至 V-Cache 中取出缓存行的位置。如果 V-Cache 不命中，则进一步查找 S-Cache。S-Cache 的响应返回后，将直接填入 I-Cache 或 D-Cache，从 I-Cache 或 D-Cache 中替换出的缓存行（如果存在的话）将回填至 V-Cache 中取出缓存行的位置。S-Cache 接收到处理器核发来的请求后，所要进行的处理过程涉及缓存一致性的维护，将在 5.3 节中做详细介绍。

表 5-1 列举了各缓存的一些参数。

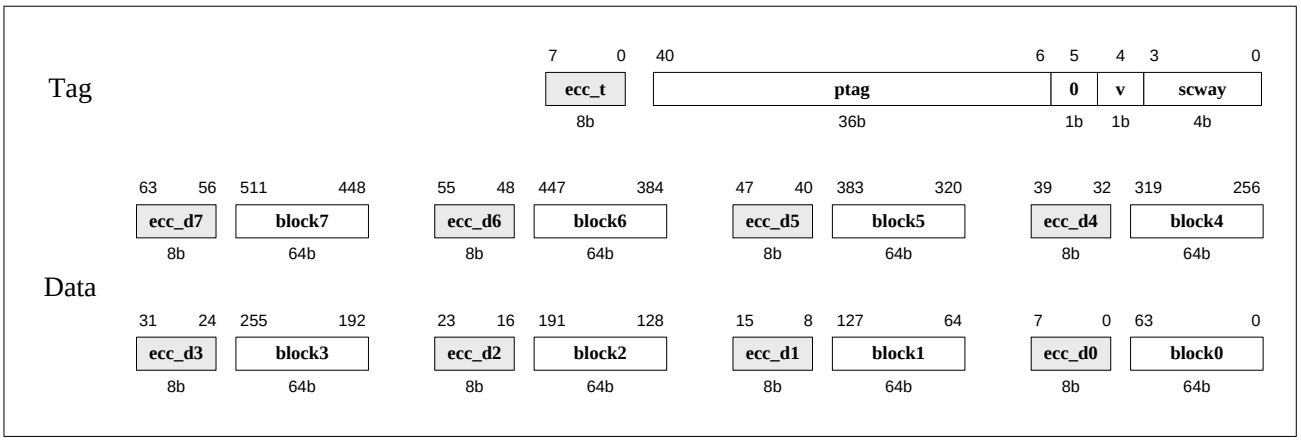
表 5-1 缓存参数

	指令缓存	数据缓存	牺牲缓存	共享缓存
容量	64KB	64KB	256KB	1MB/体 (芯片共 4MB)
相联度	4 路	4 路	16 路	16 路
行大小(line size)	512bit	512bit	512bit	512bit
索引(Index)	虚地址[13:6]	虚地址[13:6]	虚拟地址[13:6]	物理地址中的 10 比特 请参看 57 页 5.1.5 小节
标签(Tag)	物理地址[47:12]	物理地址[47:12]	物理地址[47:12]	物理地址[47:16]
替换策略	随机替换算法	LRU 替换算法	LRU 替换算法	LRU 替换算法
写策略		写回、写分配	写回、写分配	写回、写分配
校验方式	奇偶校验	SEC-DED ECC	SEC-DED ECC	SEC-DED ECC

5.1.2 一级指令缓存(I-Cache)

一级指令缓存的容量为 64KB，采用 4 路组相联结构，随机替换算法。每个缓存行中数据部分长度为 64 字节，分为 8 个 8 字节宽的块(block)，块是数据部分访问的最小单位。指令缓存采用虚地址索引物理地址标签的访问模式。访问时，虚地址的[13:6]位作为缓存行的索引，虚地址的第 5 位及以下的部分用于缓存行内索引，虚地址第 14 位及以上的部分同时进行虚实地址转换，将转换后的物理地址高位与各路中 Tag 读出的内容进行比较，以判定 Cache 是否命中。图 5-2 给出了指令缓存行的结构示意。Tag 中除了存放物理地址的高位(ptag)外，还包括有效位(v)，以及该缓存行位于 S-Cache 中的第几路的信息(scway)。有效位为 1 表示该缓存行上的内容有意义，为 0 表缓存行上无有效内容。

图 5-2 一级指令缓存行结构示意



一级指令缓存采用奇偶校验对缓存行中的 Tag 和 Data 部分进行校验。当一个新的缓存行被更新进入指令缓存时，Data 部分以块(block)作为校验的基本单位，每块生成 8 位校验结果记录下来，Tag 部分将其 0 扩展至 64 比特后，采用相同校验算法生成 8 位校验结果也予以记录。读取缓存时，原始数据和参考校验值被

同时读出，对原始数据重新计算校验值，如果与参考校验值不一致则表明出现了缓存错误，硬件自动将出现错误的缓存行在 I-Cache 中无效掉，并记录相关位置信息，触发例外。若软件无特殊诊断需要，可以直接从例外处理程序返回，处理器恢复执行后将从 V-Cache、S-Cache 或内存中重新取回所需的缓存行内容。需要注意的时，当软件采用 Store Tag 和 Store Data 类 Cache 指令填写指令缓存时，必须同时计算出所填入内容的奇偶校验值，显式地存入 ErrCtl.ECC 域。硬件在执行这类 Cache 指令时，指令缓存中写入的参考校验值来自于 ErrCtl.ECC 域而不是硬件电路自动校验生成结果。该机制主要用于完成某些特殊的诊断。指令缓存的奇偶校验值生成及检测的算法伪码描述如下：

奇偶校验值生成算法：

```
function Parity_Gen(datain63..0, parityout7..0)
endfunction Parity_Gen
```

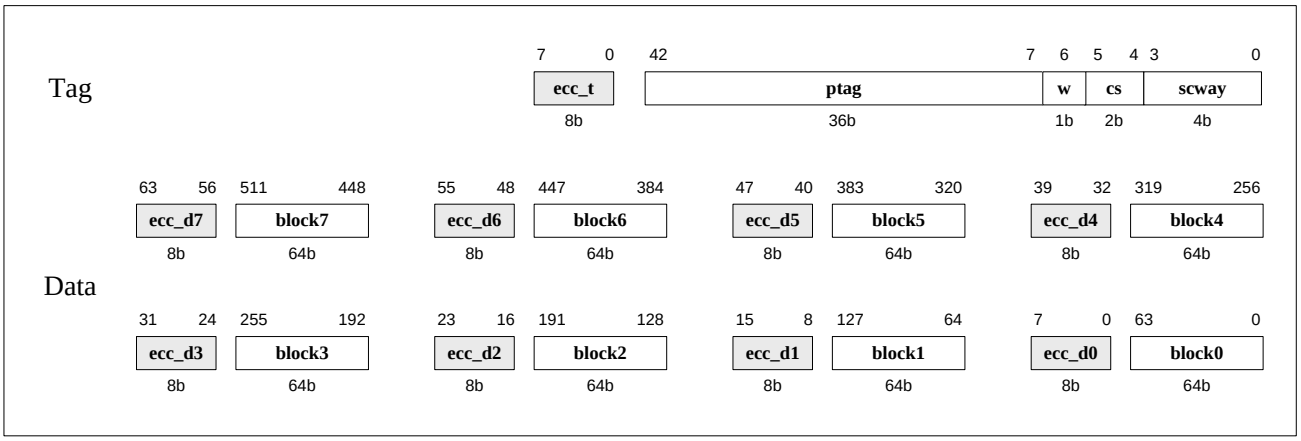
奇偶校验检测算法：

```
function Parity_Check(newparity7..0, refparity7..0, error)
endfunction Parity_Check
```

5.1.3 一级数据缓存(D-Cache)

一级数据缓存的容量为 64KB，采用 4 路组相联结构，LRU 替换算法。每个缓存行中数据部分长度为 64 字节，分为 8 个 8 字节宽的块(block)，块是数据部分访问的最小单位。数据缓存采用虚地址索引物理地址标签的访问模式。访问时，虚地址的[13:6]位作为缓存行的索引，虚地址的第 5 位及以下的部分用于缓存行内索引，虚地址第 14 位及以上的部分同时进行虚实地址转换，将转换后的物理地址高位与各路中 Tag 读出的内容进行比较，以判定 Cache 是否命中。图 5-3 给出了数据缓存行的结构示意。Tag 中除了存放物理地址的高位(ptag)外，还包括缓存行状态信息(cs)，脏标记位(w)，以及该缓存行位于 S-Cache 中的第几路的信息(scway)。cs=0 表示缓存行无效；cs=1 表示缓存行处于共享状态；cs=2 表示缓存行处于独占状态；cs=3 为保留值。w=1 表示该缓存行上有新近写入的数据。

图 5-3 一级数据缓存行结构示意



一级数据缓存采用“纠一检二(SEC-DED)”ECC 校验对缓存行中的 Tag 和 Data 部分进行校验。当一个新的缓存行被更新进入数据缓存时，Data 部分以块(block)作为校验的基本单位，每块生成 8 位校验结果记录下来，对于 Tag 则将除脏标记位(w)外的部分 0 扩展至 64 比特后，采用相同校验算法生成 8 位校验结果也予以记录。需要注意的是，Tag 部分中的脏标记位不参与校验的原因是因为这部分信息与 Tag 的其它部分存放的物理介质不同，并不保存在 SRAM 中。读取缓存时，原始数据和参考校验值被同时读出，对原始数据重新计算校验值后与参考校验值进行比对，如果发现是一比特错时，硬件将自动纠正该错误，将纠正后的值填回 D-Cache，并记录相关位置信息，触发例外；如果出错位数超过一比特，硬件将无法纠正，只能记录

相关位置信息，触发例外。当例外时一位错时，软件如果无特殊诊断需要，可以直接从例外处理程序返回。当出现一位以上错时，通常需要更加彻底地恢复，如软复位。需要注意的时，当软件采用 Store Tag 和 Store Data 类 Cache 指令填写数据缓存时，必须同时计算出所填入内容的 ECC 校验值，显式地存入 ErrCtl.ECC 域。硬件在执行这类 Cache 指令时，数据缓存中写入的参考校验值来自于 ErrCtl.ECC 域而不是硬件电路自动校验生成结果。该机制主要用于完成某些特殊的诊断。数据缓存的 ECC 校验值生成及检测的算法伪码描述如下：

ECC 校验值生成算法：

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

ECC 校验检测算法：

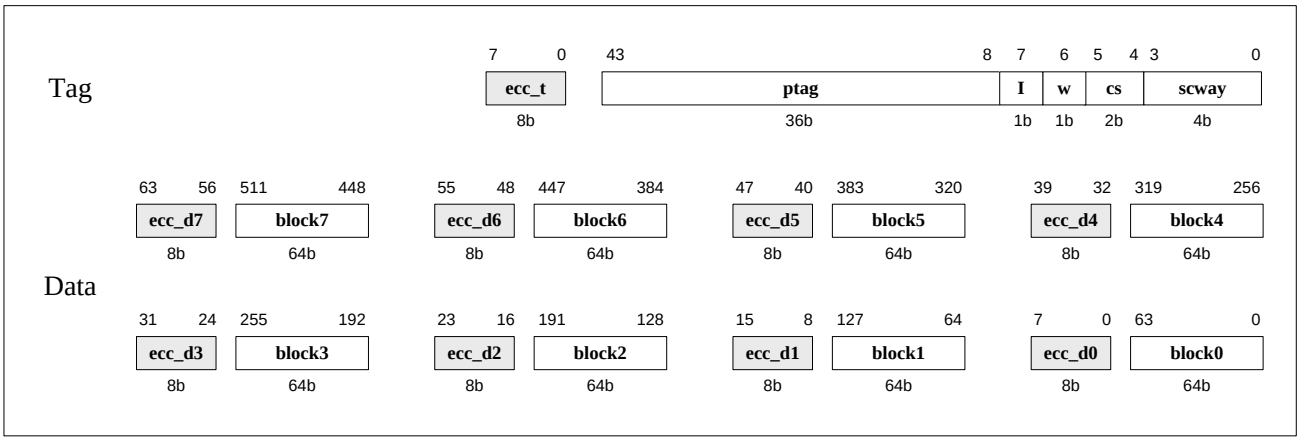
```
function ECC_Check();
endfunction ECC_Check
```

5.1.4 二级牺牲缓存(V-Cache)

二级牺牲缓存的容量为 256KB，采用 16 路组相联结构，LRU 替换算法。牺牲缓存采用虚拟地址索引物理地址标签的访问模式。每个缓存行中数据部分长度为 64 字节，分为 8 个 8 字节宽的块(block)。通常的访问总是读取或写入缓存行中的所有数据部分，仅在执行 Load Data 和 Store Data 类 Cache 指令时以相邻奇偶块作为基本单位，此时物理地址的[5:4]用于表明操作哪一对相邻奇偶块。

访问二级牺牲缓存时，物理地址的[13:6]位作为缓存行的索引，物理地址高位与各路中 Tag 读出的内容进行比较，以判定 Cache 是否命中；如果命中则将命中那一路中对应的缓存块的数据内容读取出来。图 5-4 给出了牺牲缓存行的结构示意。Tag 中除了存放物理地址的高位(ptag)外，还包括缓存行状态信息(cs)，脏标记位(w)，指令标记(I)，以及该缓存行位于 S-Cache 中的第几路的信息(scway)。cs=0 表示缓存行无效；cs=1 表示缓存行处于共享状态；cs=2 表示缓存行处于独占状态；cs=3 为保留值。w=1 表示该缓存行上有新近写入的数据。I=1 表示该缓存行存放指令，I=0 表示该缓存行存放数据。

图 5-4 二级牺牲缓存行结构示意



二级牺牲缓存采用“纠一检二(SEC-DED)”ECC 校验对缓存行中的 Tag 和 Data 部分进行校验。当一个新的缓存行被更新进入牺牲缓存时，Data 部分以块(block)作为校验的基本单位，每块生成 8 位校验结果记录下来；Tag 部分 0 扩展至 64 比特后，采用相同校验算法生成 8 位校验结果也予以记录。读取缓存时，原始数据和参考校验值被同时读出，对原始数据重新计算校验值后与参考校验值进行比对，如果发现是一比特错时，硬件将自动纠正该错误，将纠正后的值填回 V-Cache，并记录相关位置信息，触发例外；如果出错

位数超过一比特，硬件将无法纠正。需要注意的时，当软件采用 Store Tag 和 Store Data 类 Cache 指令填写数据缓存时，必须同时计算出所填入内容的 ECC 校验值，显式地存入 ErrCtl.ECC 域。硬件在执行这类 Cache 指令时，牺牲缓存中写入的参考校验值来自于 ErrCtl.ECC 域而不是硬件电路自动校验生成结果。该机制主要用于完成某些特殊的诊断。牺牲缓存的 ECC 校验值生成及检测的算法与数据缓存一致，请参看 55 页 5.1.3 小节。

5.1.5 三级共享缓存(S-Cache)

三级共享缓存基于目录协议支持缓存一致性。龙芯 3A2000 芯片所有片上 S-Cache 统一编址，每个共享缓存行都有固定的 home 结点。

共享缓存的分体结构

龙芯 3A2000 芯片的 S-Cache 采用分体结构，共分为 4 个体(bank)，通过第一级交叉开关互联网络接收来自处理器核以及维护缓存一致性的 I/O 端口的访问请求。因为龙芯 3A2000 芯片在第一级交叉开关互联网络上采用了软件动态可调整的地址窗口映射机制，所以各个 S-Cache 体所看到的物理地址是通过地址窗口重新映射后的地址，软件在操作 S-Cache 时请务必明确这一点。不同的请求最终落在 4 个 S-Cache 体中的哪一个是通过地址中的两位来决定，具体选择地址的哪两位是软件动态可调整的，由芯片配置寄存器的 SCID_SEL 决定。给出了该配置信息与选择 S-Cache 体的地址位之间的对应关系。相应的，物理地址的哪些位用于缓存行的索引也随着 SCID_SEL 值的变化而改变。

表 5-2 三级共享缓存体选择位与索引地址

SCID_SEL 值	体选择位	索引地址
b0000	PAddr[7:6]	PAddr[17:8]
b0001	PAddr[9:8]	{PAddr[17:10], PAddr[7:6]}
b0010	PAddr[11:10]	{PAddr[17:12], PAddr[9:6]}
b0011	PAddr[13:12]	{PAddr[17:14], PAddr[11:6]}
b0100	PAddr[15:14]	{PAddr[17:16], PAddr[13:6]}
b0101~b1111	PAddr[17:16]	PAddr[15:6]

共享缓存的锁机制

共享缓存单体容量为 1MB，采用 16 路组相联结构。除了采用 LRU 算法选择替换项外，共享缓存还支持缓存锁机制。共有两种锁 Cache 方式：一种是利用 Cache15 指令锁住一个 Cache 行；另一种是利用芯片配置寄存器中的共享缓存锁窗口机制锁住成片的物理地址空间。被锁住的内容一旦存入共享缓存后将不会再被替换出去，除非出现下面两种情况：(1)16 路 S-Cache 中与被锁住 Cache 行同 Index 的所有 Cache 行均处于“锁住”状态，则所有 Cache 行的锁视同无效，仍按照 LRU 算法挑选替换项；(2)软件用 Cache 指令无效被“锁住”的 Cache 行。两种锁机制各有优劣点：采用 Cache15 指令的优点是可以直接使用虚地址进行锁 Cache 操作，且如果数据不在 S-Cache 中会将待锁的 Cache 行取回至 S-Cache 中再锁住，缺点是 Cache 锁住与释放操作均需要逐 Cache 行进行，存在一定的开销；采用锁窗口机制的优点是配置一次（写 3 个锁窗口配置寄存器）就可以锁住一大片连续的地址空间（理论上不超过 S-Cache 容量的 15/16，即 3.75MB），缺点是配置必须采用物理地址信息，需要操作系统内核的特殊支持，而且配置完之后并不能保证数据一定在 S-Cache 中。软件人员可根据应用的具体特性选择合适的 S-Cache 锁机制进行程序优化。有关 Cache15 指令的具体定义请参看 27 页 2.4.9 小节的描述。有关 S-Cache 锁窗口配置寄存器的详细定义请参看《龙芯 3A2000/3B2000 处理器用户手册——上册》第 2.5 小节的描述。

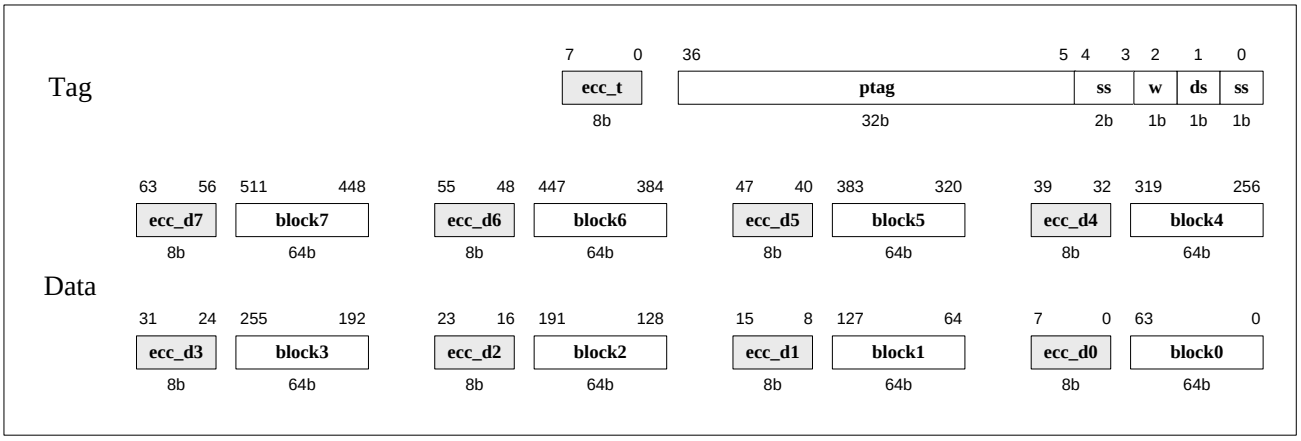
共享缓存的缓存行结构

共享缓存采用物理地址索引物理地址标签的访问模式。每个缓存行中数据部分长度为 64 字节，分为 8

个 8 字节宽的块(block)。通常的访问总是读取或写入缓存行中的所有数据部分，仅在执行 Load Data 和 Store Data 类 Cache 指令时以奇偶相邻的两块作为基本单位，此时物理地址的[5:4]用于表明操作哪一对相邻奇偶块。

访问时，物理地址高位与各路中 Tag 读出的内容进行比较，以判定 Cache 是否命中；如果命中则将命中那一路中对应的缓存块的数据内容读取出来。图 5-5 给出了共享缓存行的结构示意。Tag 中除了存放物理地址的高位(ptag)外，还包括缓存行状态信息(ss)，目录状态信息(ds)，脏位(w)，页着色位(pgc)。ss=1 表示该缓存行有效，ss=0 表示该缓存行无效。ds=1 表示目录脏，ds=0 表示目录干净。w=1 表示该缓存行上有新近写入的数据。表示该缓存行上有新近写入的数据。页着色位用于硬件处理缓存别名问题，详细请参看 63 页 5.4.5 节中的描述。

图 5-5 三级共享缓存行结构示意



共享缓存的校验

三级共享缓存采用“纠一检二(SEC-DED)”ECC 校验对缓存行中的 Tag 和 Data 部分进行校验。当一个新的缓存行被更新进入共享缓存时，Data 部分以块(block)作为校验的基本单位，每块生成 8 位校验结果记录下来；Tag 部分 0 扩展至 64 比特后，采用相同校验算法生成 8 位校验结果也予以记录。读取缓存时，原始数据和参考校验值被同时读出，对原始数据重新计算校验值后与参考校验值进行比对，如果发现是一比特错时，硬件将自动纠正该错误，将纠正后的值填回 S-Cache，并记录相关位置信息，触发例外；如果出错位数超过一比特，硬件将无法纠正。需要注意的时，当软件采用 Store Tag 和 Store Data 类 Cache 指令填写数据缓存时，必须同时计算出所填入内容的 ECC 校验值，显式地存入 ErrCtl.ECC 域。硬件在执行这类 Cache 指令时，牺牲缓存中写入的参考校验值来自于 ErrCtl.ECC 域而不是硬件电路自动校验生成结果。该机制主要用于完成某些特殊的诊断。牺牲缓存的 ECC 校验值生成及检测的算法与数据缓存一致，请参看 55 页 5.1.3 小节。

5.2 缓存算法与缓存一致属性

GS464E 支持三种缓存算法和缓存一致属性：非缓存(Uncached)、一致性缓存(Cacheable Coherent)和非缓存加速(Uncached Accelerated)。非缓存算法对应的一致性算法编码为 0b010，一致性缓存算法对应的一致性算法编码为 0b011，非缓存加速算法对应的一致性算法编码为 0b111。

5.2.1 非缓存算法

当某个地址段或者页采用了非缓存算法时，虚地址落在该地址段或者页上的取指或访存操作，都将由处理器直接向目标地址所在位置直接发起访问请求，所读取或写出的数据不源自或终止于任何一级缓存。

所有采用非缓存算法的访问请求以阻塞方式顺序执行。即当前读请求数据未返回至处理器前，所有后续请求被阻塞发出；写请求数据尚未发送完毕或者已发出的写请求未接收到最终接收方返回的写应答前，所有后续请求被阻塞发出。

5.2.2 一致性缓存算法

当某个地址段或者页采用了非缓存算法时，虚地址落在该地址段或者页上的取指或访存操作，所访问的内容可以驻留在任何一级缓存中。GS464E 由硬件维护缓存一致性，无需软件通过使用 Cache 指令无效、写回缓存中的内容来维护缓存一致性。

5.2.3 非缓存加速算法

非缓存加速算法属性用于优化在一个连续的地址空间中完成的一系列顺序的同一类型的 Uncached 存数操作。该优化方法是通过设置缓冲区来收集这种算法属性的存数操作。只要缓冲区不满，就可以把这些存数操作的数据存入缓冲区中。缓冲区大小和一个 Cache 行一致，为 64 字节。存数操作将数据存储到缓冲区中即算做执行完毕。当缓冲区数据收集满之后，则将其一次性地连续写出。连续写出的数据将直接写入目标地址所在位置，不会停留在任何一级缓存中。在顺序存数指令的数据收集过程中，若有普通类型非缓存存数指令插入，则收集工作中止，缓冲区中已保存的数据按字节写方式输出。非缓存加速算法属性的取指或取数操作的操作效果与具备普通非缓存算法属性的取指或取数操作一致。

非高速缓存加速属性可以加速顺序的 Uncached 访问，它适用于对显示设备存储的快速输出访问。

5.3 缓存一致性

GS464E 实现了基于目录的缓存一致性协议，由硬件保证 I-Cache、D-Cache、V-Cache、S-Cache、内存以及来自 HT 的 IO 设备之间数据的一致性，无需软件利用 Cache 指令来维护缓存一致性。

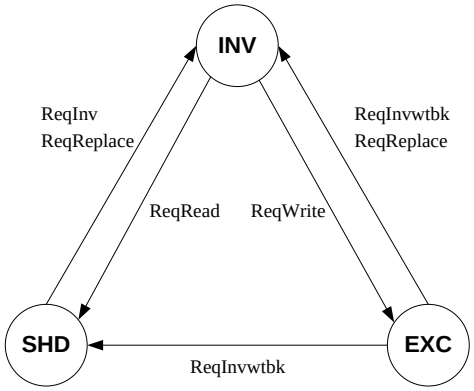
GS464E 中每个 Cache 行都有一个固定的宿主 S-Cache 体。Cache 行的目录信息就在宿主 S-Cache 体中维护。目录利用 64 位的位向量来记录拥有每个 Cache 行备份的一级 Cache（包括 I-Cache 和 D-Cache）。每个一级 Cache 块有三种可能状态：INV（无效状态）、SHD（共享状态，可读）和 EXC（独占状态，可读可写）。三个状态间的转换情况如图 5-6 所示。

当读指令或者取数操作在一级和二级 Cache 中均未命中时，处理器核向三级 S-Cache 发出 Reqread 请求，在得到 S-Cache 返回的 Repread 应答后，处理器核的一级 Cache 获得了一个 SHD 状态的 Cache 行备份。当存数操作在一级和二级 Cache 中均未命中时，处理器核向三级 S-Cache 发出 Reqwrite 请求，在得到 S-Cache 返回的 Repwrite 应答后，处理器核的一级 Cache 获得了一个 EXC 状态的 Cache 行备份。

当处理器核发生 V-Cache 替换时，通过 Reqreplace 写回 S-Cache 模块，S-Cache 通过 Repreplace 应答告知处理器核替换请求已被处理。

S-Cache 可以通过发送 Reqinv 请求至处理器核来无效 I-Cache、D-Cache 或 V-Cache 中一个 SHD 状态的 Cache 行备份，处理器核将相应 Cache 行变为 INV 状态并通过 Repinv 应答 S-Cache。S-Cache 可以通过发送 Reqwtbk 请求至处理器核，写回一个 EXC 状态的 Cache 行备份，处理器核将对应 Cache 行备份变为 SHD 状态并通过 Repwtbk 应答 S-Cache。S-Cache 可以通过发送 Reqinvwtbk 请求至处理器核，以此写回并无效一个 EXC 状态的 Cache 行备份，处理器核将对应 Cache 行备份变为 INV 状态并通过 Repinvwtbk 应答二级 Cache 模块。

图 5-6 一致性协议下缓存状态转换



5.4 缓存管理

5.4.1 CACHE 指令

本处理器实现的 CACHE 指令，分别针对 I-Cache、D-Cache、V-Cache 和 S-Cache。CACHE 指令的格式为：CACHE op, offset(base)。GS464E 的 Cache 指令与 MIPS64 规范存在一些差异，本手册已在 27 页第 2.4.9 小节进行了详细说明。这里再次强调，MIPS 规范中根据 op[1:0]=2 还是=3 来区分所操作对象是第 2 级还是第 3 级 Cache 的规定并不适用于 GS464E。为了尽可能保持软件的向前兼容性，op[1:0]=2 表明操作对象是 S-Cache(第 3 级)，op[1:0]=3 表明操作对象是 V-Cache(第 2 级)。在这一编码形式下，龙芯 3A1000 芯片中使用刷 S-Cache 来维护 Cache 一致性的软件代码在龙芯 3A2000 芯片中仍可以获得同样的 Cache 一致性维护效果。即当软件无效一 S-Cache 行时，由硬件保证同一物理地址在所有处理器核内的 Cache 中均被同时无效掉；当软件无效并写回一 S-Cache 行时，由硬件保证同一物理地址在所有处理器核内的 Cache 中均被同时无效掉，且写回至主存的内容一定包含了最新写入的数据。

根模式下的 CACHE 指令

根模式下运行的核心态软件可以使用所有已实现的 CACHE 指令，具体列表如下：

表 5-3 根模式下 CACHE 指令

op[4:0]	功能描述	目标 Cache
b00000	根据索引无效 Cache 行	I-Cache
b01000	根据索引写 Cache 行 Tag	I-Cache
b11100	根据索引写 Cache 行 Data	I-Cache
b00001	根据索引无效并写回 Cache 行	D-Cache
b00101	根据索引读 Cache 行 Tag	D-Cache
b01001	根据索引写 Cache 行 Tag	D-Cache
b10001	根据命中无效 Cache 行	D-Cache
b10101	根据命中无效并写回 Cache 行	D-Cache
b11001	根据索引读 Cache 行 Data	D-Cache
b11101	根据索引写 Cache 行 Data	D-Cache
b00010	根据索引无效并写回 Cache 行	V-Cache
b00011	根据索引无效并写回 Cache 行	S-Cache
b00111	根据索引读 Cache 行 Tag	S-Cache
b01111	根据地址取回并锁存 Cache 行	S-Cache

op[4:0]	功能描述	目标 Cache
b01011	根据索引写 Cache 行 Tag	S-Cache
b10011	根据命中无效并写回 Cache 行	S-Cache
b11011	根据索引读 Cache 行 Data	S-Cache
b11111	根据索引写 Cache 行 Data	S-Cache

CACHE2 指令主要用于只关闭单个核时，清空这个核的 V-Cache，硬件仅保证按照下列方式使用 CACHE2 时结果的正确性。

首先，在使用 CACHE2 指令清空 V-Cache 时，务必使此过程中所执行的代码均位于 uncache 空间。

其次，在使用 CACHE2 指令清空 V-Cache 时，中间不要再执行其它在 cache 空间上的 load、store 操作。

客模式下的 CACHE 指令使用

客模式下 CACHE 指令的使用受到根模式的控制，具体定义如下：

- GuestCtl0.CG=0 时，使用任何 CACHE 指令都将触发客模式特权敏感指令例外(GPSI)。
- GuestCtl0.CG=1 时，使用 op[4:2]=1, 2, 6, 7 的 CACHE 指令将触发客模式特权敏感指令例外(GPSI)。
- GuestCtl0.CG=1，但 Diag.GCAC=0 时，使用 CACHE0、CACHE1、CACHE3 指令将触发客模式特权敏感指令例外(GPSI)。

5.4.2 缓存初始化

基于硬件的缓存初始化

GS464E 在硬重启期间由硬件将所有缓存的 Tag 部分置为全 0，即将所有缓存的 Cache 行无效。因此除了下面这一种特殊使用情况外，软件无需初始化所有缓存。

导致硬件初始化缓存不安全的特例是：处理器硬重启后，软件直接用 Index Store Tag 类 Cache 指令将某一 Cache 行的 Tag 置为有效，但又并未使用 Index Store Data 类 Cache 指令将该 Cache 行 Data 部分的所有块置为确定的内容。

如果软件确实存在上述操作序列，请务必保证在进行存在安全风险的操作之前基于软件方式对缓存进行初始化。

基于软件的缓存初始化

GS464E 推荐的基于软件的缓存初始化流程如下：

步骤 1：开辟一段内存，填充任意数据。这些数据随后将用于填充缓存的数据部分，以形成正确的校验值。建议使用从 0 地址开始的内存区域。

步骤 2：屏蔽中断，防止初始化过程意外情况的发生。

步骤 3：初始化 I-Cache。

```
mtc0 zero, TagLo; mtc0 zero, TagHi;
mtc0 zero, ErrCtl;   /*64 位全 0 的奇偶校验值为 0x0*/
for (addr=0xffffffff80000000; addr<0xffffffff80010000; addr+=64) {
  /*将 I-Cache 各路逐行置为确定的值*/
  for (way=0; way<4; way+=1) {
```

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

步骤 4：初始化 D-Cache。

```
  mtc0 zero, TagLo; mtc0 zero, TagHi;
  addiu r1, r1, 0x22; mtc0 r1, ErrCtl;   /*64 位全 0 的 ECC 校验值为 0x22*/
  for (addr=0xffffffff80000000; addr<0xffffffff80010000; addr+=64) {
    /*将 D-Cache 各路逐行置为确定的值*/
    for (way=0; way<4; way+=1) {
      Cache_Index_Store_Tag_D(addr+way);
      for (block=0; block<8; block+=1) {
        Cache_Index_Store_Data_D(addr+(block<<3)+way);
      }
    }
  }
```

步骤 5：初始化 V-Cache。

```
  mtc0 zero, TagLo; mtc0 zero, TagHi;
  addiu r1, r1, 0x22; mtc0 r1, ErrCtl;   /*64 位全 0 的 ECC 校验值为 0x22*/
  for (addr=0xffffffff80000000; addr<0xffffffff80010000; addr+=64) {
    /*将 V-Cache 各路逐行置为确定的值*/
    for (way=0; way<16; way+=1) {
      Cache_Index_Store_Tag_V(addr+way);
      for (block_pair=0; block_pair<4; block_pair+=1) {
        Cache_Index_Store_Data_V(addr+(block_pair<<4)+way);
      }
    }
  }
```

步骤 6：初始化 S-Cache[1]。

```
  Scache_init_ok[get_my_CPUNum()]=0; /*需使用 uncached write*/
  mtc0 zero, TagLo; mtc0 zero, TagHi;
  addiu r1, r1, 0x22; mtc0 r1, ErrCtl;   /*64 位全 0 的 ECC 校验值为 0x22*/
  /*0 号处理器初始化 S-Cache Bank 0，1 号处理器初始化 S-Cache Bank 1，依次类推*/
  bank = get_my_CPUNum();
  for (addr=0xffffffff80000000; addr<0xffffffff80040000; addr+=256) {
    /*将 S-Cache 各路逐行置为确定的值*/
    for (way=0; way<16; way+=1) {
```

[1] 需要芯片配置寄存器 SCID_SEL 等于缺省值 0。

```
      Cache_Index_Store_Tag_V(addr+(bank<<6)+way);
      for (block_pair=0; block_pair<4; block_pair+=1) {
        Cache_Index_Store_Data_V(addr+(bank<<6)+(block_pair<<4)+way);
      }
    }
  }
  Scache_init_ok[get_my_CPUNum()]=1; /*需使用 uncached write*/

步骤 7：轮询 Scache_init_ok 向量直至所有项均为 1。/*轮询需使用 uncached read*/

至此缓存初始化完毕。

/*对于步骤 6、7，软件也可以采用其它同步机制，这里只给出一种最简单的方案*/
```

5.4.3 一级指令缓存与一级数据缓存间的一致性维护

对于存在“自修改代码”的应用程序，存在一级指令缓存与一级数据缓存之间的数据一致性问题。GS464E 由硬件维护一级指令缓存与一级数据缓存之间的数据一致性，无需软件使用 CACHE 指令或 SYNCI 指令通过刷回、清空 D-Cache 和 I-Cache 来维护数据一致性。

需要指出的是，GS464E 实现的是弱一致性存储模型。因此软件在修改完代码后，必须使用 jr.hb 或 jalr.hb 指令跳转至被修改代码处执行。jr.hb 和 jalr.hb 除产生跳转外，还将起到栅障(barrier)的作用。从而保证 jr.hb 或 jalr.hb 跳转目标处的 PC 一定看到栅障前写操作的修改效果。

5.4.4 处理器与 DMA 设备间的缓存一致性维护

为了提高处理器的性能，DMA 设备的驱动软件会将需要大量交互的数据放在缓存空间，由此产生了处理器与 DMA 设备间的缓存一致性维护问题。该问题的详细阐述在众多驱动开发书籍中均有提及，本手册不再赘述。在龙芯 3A2000 芯片中，硬件可以维护处理器与连接至 HT 端口上 DMA 设备之间的缓存一致性。因此，当系统中某一 DMA 设备是通过 HT 端口接入访问系统主存时，该设备的驱动软件无需使用 Cache 指令通过刷 S-Cache 来维护数据一致性。这样做可以提升处理器执行效率。

需要指出的是，GS464E 实现的是弱一致性存储模型。因此处理器与 DMA 设备之间仍需要通过存放在非缓存(uncached)区域的信号量或是中断机制来实现栅障的同步效果，以保证消费者在读取数据时确实可以观察到生产者希望其观察到的数据。

5.4.5 缓存别名与页着色

因为 GS464E 的一级指令缓存和一级数据缓存采用的是虚索引物理标签的访问模式，且缓存每一路的大小为 16KB，所以当页大小为 4KB 或 8KB 时，会存在缓存别名问题。解决缓存别名的通行方法是采用“页着色”机制，即保证在任意时刻，一个物理地址至多有一种虚拟地址的“页颜色”。GS464E 通过硬件实现“页着色”机制，不再由软件实现该机制。

在绝大多数情况下，软件可以不考虑缓存别名问题，因为硬件已通过“页着色”保证了不出现缓存别名。唯一的特例是，当软件通过 Index Store Tag 和 Index Store Data 类 Cache 指令直接在各级缓存中凭空制造出有效的 Cache 行，并且将这些 Cache 行的数据同后续的程序使用的情况。此时，软件必须要保证 S-Cache 行 Tag 中的页颜色域(pgc)的内容与该 Cache 行数据的虚地址相匹配。具体来说，S-Cache 的 Tag 中的 pgc 域必须始终等于该 Cache 行虚地址的[13:12]位。

6 处理器例外与中断

6.1 处理器例外

6.1.1 例外优先级

当一条指令同时满足多个例外触发条件时，GS464E 将按照表 6-1 所示例外优先级，优先触发优先级高的例外。

表 6-1 例外优先级

例外	类型
冷复位	异步、复位类
EJTAG 单步执行例外	同步、调试类
EJTAG 调试中断例外	异步、调试类
不可屏蔽中断	异步
EJTAG 指令断点例外	同步、调试类
地址错例外—取指	同步
TLB/XTLB 重填例外—取指	同步
TLB 无效例外—取指	同步
TLB 执行阻止例外	同步
Cache 错例外—取指	同步
EJTAG SDBBP 例外	同步
协处理器不可用例外	同步
保留指令例外	同步
中断	异步
整型溢出例外、陷阱例外、系统调用例外、断点例外、浮点例外、浮点栈例外	同步
EJTAG 精确数据断点例外	同步、调试类
地址错例外—数据访问	同步
TLB/XTLB 重填例外—数据访问	同步
TLB 无效例外—数据访问	同步
TLB 读阻止例外	同步
TLB 修改例外	同步
Cache 错例外—数据访问	同步

6.1.2 例外入口向量位置

冷复位、软复位和不可屏蔽中断的例外入口向量地址均使用是专用的地址 0xFFFF.FFFF.BFC0.0000，这个地址既不通过 Cache 进行存取，也无需地址映射。

EJTAG 调试相关例外的向量地址根据 EJTAG 控制寄存器中的 ProbeTrap 位是 0 还是 1 分别选用 0xFFFF.FFFF.BFC0.0480 和 0xFFFF.FFFF.FF20.0200。

所有其它例外向量地址都采用“基址+偏移”方式定义。当 Status.BEV=0 时，各类例外的基址采用固定配置；当 Status.BEV=0 时，软件可以通过 EBase 寄存器和 GSEBase 寄存器配置例外向量基址。表 6-2 列举了例外向量基址定义，表 6-3 列举了例外偏移定义。

表 6-2 例外向量基址

例外	Status.BEV=0	Status.BEV=1
冷复位、软复位、不可屏蔽中断	0xFFFF.FFFF.BFC0.0000	
EJTAG 调试相关例外(ProbTrap=0)	0xFFFF.FFFF.BFC0.0480	
EJTAG 调试相关例外(ProbTrap=1)	0xFFFF.FFFF.FF20.0200	
Cache 错例外	$EBase_{63..30}$ ‖ 1 ‖ $EBase_{28..12}$ ‖ 0x000	0xFFFF.FFFF.BFC0.0200
其它例外情况	$EBase_{63..12}$ ‖ 0x000	0xFFFF.FFFF.BFC0.0200

表 6-3 例外向量偏移

例外	向量偏移
冷复位、软复位、不可屏蔽中断	无偏移，直接使用基址
各类 EJTAG 调试例外(ProbTrap=0)	无偏移，直接使用基址
各类 EJTAG 调试例外(ProbTrap=1)	无偏移，直接使用基址
TLB 重填例外(Status.EXL=0)	0x000
XTLB 重填例外(Status.EXL=0)	0x080
Cache 错例外	0x100
其它例外情况	0x180
中断(Cause.IV=0)	0x180
中断(IntCtl.VS=0 且 Cause.IV=1)	0x200
中断(Status.BEV=1 且 Cause.IV=1)	0x200
中断 (Cause.IV=1 且 Status.BEV=0 且 IntCtl.VS!=0)	0X200 + (中断向量号 × (IntCtl.VS ‖ 0b00000))

6.1.3 处理器硬件响应例外的通用处理过程

当处理器开始处理某个例外，状态寄存器的 EXL 位是被置为 1 的，这意味着系统运行在内核模式。在保存了适当的现场状态之后，例外处理程序通常将状态寄存器的 KSU 字段设定为内核模式，同时将 EXL 位置回为 0。当恢复现场状态并且重新执行时，处理程序则会把 KSU 字段恢复回上次的值，同时置 EXL 位为 1。

从例外返回也会将 EXL 位置为 0。

6.1.4 冷复位例外

当系统第一次上电或者冷重置时，产生冷重置例外。该例外不可屏蔽。

冷复位例外使用特殊的例外入口向量地址。该地址属于无需地址映射和不通过 Cache 存取数据的 CPU 地址空间，因此处理这个例外不必初始化 TLB 或 Cache。这也意味着即使 Cache 和 TLB 处于不确定状态，

处理器也可以取出并执行指令。

当冷复位例外发生时，处理器将进行一全套复位初始化过程，此时 CPU 中所有寄存器内容是不确定的，但下列寄存器域除外：

- Status 寄存器置初值，其中 SR 位清 0，ERL 位和 BEV 位置 1。
- Config0~Config6 寄存器置初值。
- Random 寄存器初始化为最大值，Wired 寄存器初始化为 0。
- EntryHi、EntryLo0、EntryLo1、PageMask、PageGrain 的相关域置初值。
- ErroEPC 寄存器初始化为 PC 的值。
- Performance Count 寄存器的 Event 位初始化为 0。
- 所有断点和外部中断都被清除。

6.1.5 不可屏蔽中断

不可屏蔽中断由处理器独立的 NMI 中断输入信号触发。该例外不可屏蔽。

不可能屏蔽中断采用的例外入口向量与冷复位一致。因此当发生不可屏蔽中断例外时，Status.NMI 位被置为 1，软件可通过该位区分冷重置。

不可屏蔽中断例外并不抛弃任何机器的状态，而是保留处理器的状态用于诊断。特别的，Cause 寄存器内容保持不变，而系统则跳到不可屏蔽例外入口向量出开始执行处理程序。

不可屏蔽中断例外仅修改下列寄存器：

- Status.ERL 置为 1，Status.SR 置为 0，Status.NMI 置为 1，Status.BEV 置为 1。
- ErroEPC 寄存器初始化为 PC 的值。

6.1.6 中断例外

当未屏蔽的中断到来时，触发中断例外。关于中断的详细描述，请参考 74 页 6.2 节。

控制寄存器 Cause 的 ExcCode 域：

0x00 (Int)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
Cause	IP 域记录待处理的中断。

6.1.7 地址错例外

当发生下列条件时触发地址错例外：

双字 load/store 指令，其访问地址不对齐于双字边界。

字 load/store 指令，其访问地址不对齐于字边界。

半字 load/store 指令，其访问地址不对齐与半字边界。

取指 PC 不对齐于字边界。

在客户模式或监管模式下访问核心模式的地址段。

在客户模式下访问监管模式的地址段。

当 64 位寻址使能未启用时，取指 PC 或 load/store 指令的访问采用 64 位地址，且地址落在 32 位地址空间兼容范围之外。

取指 PC 或 load/store 指令的访问采用 64 位地址，且地址落在未实现的范围之中。

处于核心模式时，所访问的页表项有效且 K 位为 0。

该例外在根模式和客模式下均可以处理。

控制寄存器 Cause 的 ExcCode 域：

0x04 (AdEL)：取指或读数据

AdES (0x05)：写数据

（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。

6.1.8 TLB 重填例外

32 位主机地址空间下，且 Root.Status.EXL=0，访存使用映射地址，该地址在 TLB 中查找无匹配项时触发 TLB 重填例外。请注意该情况区别于在 TLB 中找到了匹配项但是匹配页表项有效位为 0，后者对应的是 TLB 无效例外。为了加速 TLB 重填这一频繁而关键例外的处理效率，TLB 重填例外采用单独的例外入口偏移值，因此该例外在 Root.Cause.ExcCode 域填入的例外编码与 XTLB 重填例外、TLB 无效例外不做区分。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x02 (TLBL)：取指或读数据

0x03 (TLBS)：写数据

（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。
XContext	?
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。
Diag	MID 域置为 0。

6.1.9 XTLB 重填例外

64 位主机地址空间下，且 Root.Status.EXL=0，访存使用映射地址，该地址在 TLB 中查找无匹配项时触发 XTLB 重填例外。请注意该情况区别于在 TLB 中找到了匹配项但是匹配页表项有效位为 0，后者对应的是 TLB 无效例外。为了加速 XTLB 重填这一频繁而关键例外的处理效率，XTLB 重填例外采用单独的例外入口偏移值，因此该例外在 Root.Cause.ExcCode 域填入的例外编码与 TLB 重填例外、TLB 无效例外不做区分。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x02 (TLBL)：取指或读数据

0x03 (TLBS)：写数据

（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。
XContext	BadVPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。
Diag	MID 域置为 0。

6.1.10 TLB 无效例外

当出现下列情况时触发 TLB 无效例外：

访存在主机地址空间下使用映射地址，该地址在 TLB 中找到了匹配项但是匹配页表项有效位为 0，触发该例外。

PageGrain.IEC=0 时

PageGrain.RIE=1，load 操作在主机地址空间下使用映射地址，在 TLB 中找到了匹配且有效的项，但是表项中的 RI 位为 1。

PageGrain.XIE=1，取指在主机地址空间下使用映射地址，在 TLB 中找到了匹配且有效的项，但是表项中的 XI 位为 1。

软件需要注意下面这种情况：当 Root.Status.EXL=1 时，访存所使用的映射地址在 TLB 中找不到匹配项时，所采用的例外入口偏移是普通例外入口偏移 (0x180)，同时 Root.Cause.ExcCode 域填入的例外编码仍是 TLBL (0x2)或 TLBS (0x3)。为了将这种情况与正常的 TLB 无效例外区分开来，只能由例外处理程序使用 TLBP 指令，根据查找结果进行区分。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x02 (TLBL)：取指或读数据

0x03 (TLBS)：写数据

（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。

寄存器	状态更新描述
XContext	BadVPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。
Diag	MID 域置为 0。

6.1.11 TLB 修改例外

store 操作在主机地址空间下映射地址，该地址在 TLB 中找到了匹配且有效的项，但是该页表项的 D 位为 0（意味着该页不可写），触发 TLB 修改例外。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x01 (Mod)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。
XContext	BadVPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。
Diag	MID 域置为 0。

6.1.12 TLB 执行阻止例外

当 Root.PageGrain.IEC=0，且 Root.PageGrain.XIE=1，取指在主机地址空间下使用映射地址，在 TLB 中找到了匹配且有效的项，但是表项中的 XI 位为 1。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x14 (TLBXI)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。
XContext	BadVPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。

寄存器	状态更新描述
Diag	MID 域置为 0。

6.1.13 TLB 读取阻止例外

当 Root.PageGrain.IEC=0，且 Root.PageGrain.RIE=1，load 操作在主机地址空间下使用映射地址，在 TLB 中找到了匹配且有效的项，但是表项中的 RI 位为 1。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x13 (TLBRI)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
BadVAddr	记录触发例外的虚地址。
Context	BadVPN2 域记录触发例外的虚地址的[31..13]位。
XContext	BadVPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。
EntryHi	VPN2 域记录触发例外的虚地址的[47..13]位； R 域记录触发例外的虚地址的[63..62]位。 ASID 域记录触发该例外的操作所属进程的 ASID。
Diag	MID 域置为 0。

6.1.14 Cache 错误例外

当取指或 load/store 操作执行时发现 Cache 的 tag 或 data 出现校验错误时，触发该例外。该例外不可屏蔽。因为该例外涉及的错误在 Cache 中，所以采用专门的例外入口，位于非映射非缓存地址段。该例外入口请参看 65 页 6.1.2 节描述。

该例外仅在根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

无

响应例外时的硬件状态更新过程：

```
CacheErr ← ErrorState
Status.ERL ← 1
if InstructionInBranchDelaySlot then
    ErrorEPC ← PC of the branch/jump
else
    ErrorEPC ← PC of the instruction
endif
if Status.BEV=1 then
    PC ← 0xFFFF.FFFF.BFC0.0200 + 0x100
else
    PC ← 0xFFFF.FFFF || EBase31..30 || 1 || EBase28..12 || 0x100
endif
```

6.1.15 整型溢出例外

当一条 ADD、ADDI、SUB、DADD、DADDI 或 DSUB 指令执行，导致结果的补码溢出时，整型溢出例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x0c (Ov)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

无

6.1.16 陷阱例外

当 TGE、TGUE、TLT、TLTU、TEQ、TNE、TGEI、TGEUI、TLTI、TLTUI、TEQI、TNEI 指令执行，条件结果为真时，触发陷阱例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x0d (Tr)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

无

6.1.17 系统调用例外

当执行 SYSCALL 指令时，触发系统调用例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x08 (Sys)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

无

6.1.18 断点例外

当执行一条 BREAK 指令时，触发断点例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x09 (Bp)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

无

6.1.19 保留指令例外

当执行一条 GS464E 未实现的指令时，触发保留指令例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x0a (RI)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

无

6.1.20 协处理器不可用例外

当下列发生情况时，触发协处理器不可用例外：

- 当不处于调试模式和核心模式时，且 Status.CU0=0，执行 COP0 类指令(opcode=0b010000)、CACHE 类指令(opcode=0b101111)、LWPTE、LWDIR、LDPTE、LDDIR。
- 当 Status.CU1=0 时，执行 COP1 类指令(opcode=0b010001)、COP1X 类指令(opcode=0b010011)、LWC1、SWC1、LDC1、SDC1、MOVF、MOVT、64 位多媒体指令(opcode=0b010010, func=0b000000~0b000011, rs=11000~11111; opcode=0b010010, func=0b001000~0b001110, rs=11000~11101) 、 gsLWLC1 、gsLWRC1、gsLDLC1、gsLDRC1、gsLWLEC1、gsLWGTC1、gsLDLEC1、gsLDGTC1、gsLQC1、gsLWXC1、gsLDXC1、gsSWLC1、gsSWRC1、gsSDLC1、gsSDRC1、gsSWLEC1、gsSWGTC1、gsSDLEC1、gsSDGTC1、gsSQC1、gsSWXC1、gsSDXC1。
- 当 Status.CU2=0 时，执行 COP2 类指令(opcode=0b010010)、LWC2 类指令(opcode=0b110010)、SWC2 类指令(opcode=0b111010)、LDC2 类指令(opcode=0b110110)、SDC2 类指令(opcode=0b111110)，但不包括 SETMEM、gsLBLE、gsLBGT、gsLHLE、gsLHGT、gsLWLE、gsLWGT、gsLDLE、gsLDGT、gsLQ、gsLBX、gsLHX、gsLWX、gsLDX、gsSBLE、gsSBGT、gsSHLE、gsSHGT、gsSWLE、gsSWGT、gsSDLE、gsSDGT、gsSQ、gsSBX、gsSHX、gsSWX、gsSDX、LWPTE、LWDIR、LDPTE、LDDIR、64 位多媒体指令(opcode=0b010010, func=0b000000~0b000011, rs=11000~11111; opcode=0b010010, func=0b001000~0b001110, rs=11000~11101)、gsLWLC1、gsLWRC1、gsLDLC1、gsLDRC1、gsLWLEC1、gsLWGTC1、gsLDLEC1、gsLDGTC1、gsLQC1、gsLWXC1、gsLDXC1、gsSWLC1、gsSWRC1、gsSDLC1、gsSDRC1、gsSWLEC1、gsSWGTC1、gsSDLEC1、gsSDGTC1、gsSQC1、gsSWXC1、gsSDXC1。

需要注意的是：在客模式下，当 Guest.Status.CU1/2=1 但是 Root.Status.CU1/2=0，同样触发协处理器不可用例外直接陷入至根模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x0b (CpU)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
Cause	CE 域记录不可用的协处理器号。

6.1.21 浮点例外

浮点协处理器触发浮点浮点例外。关于浮点例外的详细介绍，请参看 14 页 2.2.4 节。部分浮点例外可以通过配置 FCSR 寄存器的 Enable 域予以屏蔽，详细情况请参看 11 页 2.2.3 节。

控制寄存器 Cause 的 ExcCode 域：

0x0f (FPE)（请参看 105 页表 7-28）

响应例外时的额外硬件状态更新：

寄存器	状态更新描述
FCSR	Cause 域和 Flag 域记录具体的浮点例外类型信息。

6.1.22 浮点栈例外

执行 SETTAG 指令时若源操作数中的内容不满足指定条件，触发浮点栈例外。

该例外可在根模式和客模式下处理。

控制寄存器 Cause 的 ExcCode 域：

0x10 (GSExc)（请参看 105 页表 7-28）

控制寄存器 GSCause 的 GSExcCode 域：

0x00 (IS)（请参看 125 页表 7-43）

6.2 中断

本节描述的中断所涉及的范围包括硬件中断、软件中断、计时器中断和性能计数器溢出中断。“不可屏蔽中断 (NMI)”尽管在名称中包含“中断”字眼，但是它既不受本小节所述的中断系统的控制，也不影响中断系统，所以将它视作单独一种特殊的例外——不可屏蔽中断例外。

6.2.1 中断响应的必要条件

处理器响应中断的必要条件是：

- Status.IE=1，表示全局中断使能开启。
- Debug.DM=0，表示不处于调试模式。
- Staus.ERL=0 且 Status.EXL=0，表示既没有错误也没有例外正在处理。
- 某个中断源产生中断且该中断源未被屏蔽外，

6.2.2 中断模式

本小节所述的内容同时适用于根模式和客模式。当涉及客模式下的虚拟机环境时，本小节所述的“硬件中断”这一概念中的“硬件”并不一定指物理硬件，只是沿袭了 MIPS 规范中一贯的命名风格。

GS464E 支持两种中断模式：

模式一、兼容中断模式

该模式下，处理器支持 2 个软件中断 (SW0~SW1)、6 个硬件中断 (HW0~HW5)、1 个计时器中断和 1 个性能计数器溢出中断。其中计时器中断和性能计数器中断复用 HW5 硬件中断。

软件中断的中断源即为 Cause.IP[1:0]两位，仅可以通过软件对 Cause.IP[1:0]位写 1 进行触发，软件写 0 进行清除。

计时器中断的中断源记录在 Cause.TI 位中，在 Count[31:0]等于 Compare[31:0]的情况下由硬件置 1，软件可通过写 Compare 寄存器间接清除 Cause.TI 位记录的中断。

性能计数器溢出中断的中断源记录在 Cause.PCI 位中，在性能计数器值溢出时（计数器第 47[1]位为 1）由硬件置 1，软件可以通过对相关性能计数值寄存器的第 47 位写 0 间接清除 Cause.PCI 位。

[1] GS464E 中性能计数器实际有效计数位数为 48 位。

硬件中断的中断源来自于处理器外部，由硬件逐拍采样处理器接口上的 6 个中断输入引脚，软件需要反向遍历系统的中断路由路径，清除终端设备或路由路径上的中断状态，以此来清除处理器的硬件中断。

除全局中断使能外，每个中断源各自含有一个中断屏蔽位。各中断请求生成关系如表 6-4 所示。

表 6-4 兼容中断模式下各中断请求生成

中断类型	中断源	中断请求生成
硬件中断、计时器中断或性能计数器溢出中断	HW5	Cause.IP7 & Status.IM7
硬件中断	HW4	Cause.IP6 & Status.IM6
	HW3	Cause.IP5 & Status.IM5
	HW2	Cause.IP4 & Status.IM4
	HW1	Cause.IP3 & Status.IM3
	HW0	Cause.IP2 & Status.IM2
软件中断	SW1	Cause.IP1 & Status.IM1
	SW0	Cause.IP0 & Status.IM0

所有中断采用相同的例外入口偏移，具体采用通用例外入口偏移(0x180)还是特殊例外入口偏移(0x200)由 Cause.IV 决定。详细情况请参看 66 页表 6-3。

中断例外处理程序需查询 Cause.IP 以及 Status.IM 以确定具体中断源。对于同时出现多个有效中断源的情况，软件可通过对查询次序调控来实现中断处理的优先级。

模式二、向量中断模式

该模式在兼容中断模式的基础上为每个中断指定唯一的例外入口向量（计算方式请参看 66 页表 6-3），且为所有中断定义了固定的优先级关系，如表 6-5 所示。

表 6-5 向量中断模式下各中断间优先级关系

优先级	中断类型	中断源	中断请求生成	中断向量号
最高优先级	硬件中断	HW5	Cause.IP7 & Status.IM7	7
		HW4	Cause.IP6 & Status.IM6	6
		HW3	Cause.IP5 & Status.IM5	5
		HW2	Cause.IP4 & Status.IM4	4
		HW1	Cause.IP3 & Status.IM3	3
		HW0	Cause.IP2 & Status.IM2	2
	软件中断	SW1	Cause.IP1 & Status.IM1	1
最低优先级		SW0	Cause.IP0 & Status.IM0	0

在向量中断模式下，计时器中断复用哪个硬件中断源由 IntCtl.IPTI 域定义，请参看 103 页表 7-25。

在向量中断模式下，性能计数器溢出中断复用哪个硬件中断源由 IntCtl.IPPCI 域定义，请参看 103 页表 7-25。

GS464E 未实现影子寄存器，因此向量中断模式下所有中断均对应同一组逻辑通用寄存器(GPR)。

处理器当前采用哪一种中断模式由 Status.BEV、Cause.IV、IntCtl.VS 三个域决定。其对应关系如表 6-6 所示。

表 6-6 中断模式判定

Status.BEV	Cause.IV	IntCtl.VS	中断模式

Status.BEV	Cause.IV	IntCtl.VS	中断模式
1	x[1]	x	兼容中断模式
x	0	x	兼容中断模式
x	x	=0	兼容中断模式
0	1	!=0	向量中断模式

6.2.3 中断处理的补充说明

本手册仅描述了 GS464E 部分的中断系统的组成结构与响应机制。软件人员在设计龙芯 3A2000 芯片中断系统时，需同时参看《龙芯 3A2000/3B2000 处理器户手册——上册》的第 6、7 章。

处理器内部只对外部输入的高电平硬件中断请求予以直接采样记录，不负责电平转换，亦不负责将脉冲式中断信号扩展为电平信号。这部分工作由芯片中的中断控制器完成，请参看《龙芯 3A2000/3B2000 处理器户手册——上册》的第 6、7 章。

视外部硬件中断输入的信号行为，处理器内部直接连接至外部中断输入的中断请求位(Root.Cause.IP[7:2]或 Guest.Cause.IP[7:2])有可能在触发中断请求之后、中断处理程序查询这些请求位之前由 1 变为 0。中断处理程序需要能够处理这种情况。建议不做任何处理直接返回。如为系统诊断而做了特殊处理，请不要影响系统的正常行为。

对于直接受外部硬件影响的硬件中断，软件通常需要清除某一终端设备的中断状态。从处理器发出清除中断的命令序列，到设备接受到命令将自身的中断状态清除，再到处理器中断输入引脚的电平由 1 变 0（中断输入撤销）的过程可能存在不确定的延迟。如果过早地打开中断使能，可能再次采样到同一中断，即所谓的“假中断”现象。软件需要能够正确处理这一情况。建议软件在发出清除设备中断状态的写命令之后，再显式地读取相关设备的中断状态标记[2]，直到读取的状态标记已经清除后，再彻底开启处理器内部或中断路由通路上对应该设备中断的使能。龙芯 3A2000 芯片已保证内部各中断源信号通过中断路由通路传递至处理器中断输入引脚的延时总是小于该中断状态通过数据访问通路返回至处理器的延迟。如果调试时仍发现“假中断”现象，请软件人员进一步查询系统中所涉及的设备芯片、桥片的数据手册和用户手册。

[1] x 表示可以为任意值。

[2] 如果相关设备的中断标记位是“读清除”属性的，则软件在发出查询命令时需慎重。

7 协处理器 0 寄存器

7.1 根协处理器 0 寄存器概览

GS464E 处理器核中根模式上下文中的协处理器 0 寄存器称为“根协处理器 0 寄存器”，具体实现的寄存器列举在表 7-1 中。

表 7-1 根协处理器 0 寄存器一览表

Reg.	Sel.	寄存器名称	功能定义	索引
0	0	Index	VTLB 与 FTLB 访问指定索引寄存器	第 79 页 7.2 节
1	0	Random	VTLB 与 FTLB 访问随机索引寄存器	第 80 页 7.3 节
2	0	EntryLo0	VTLB 与 FTLB 表项低位内容中与偶数虚页相关部分	第 81 页 7.4 节
3	0	EntryLo1	VTLB 与 FTLB 表项低位内容中与奇数虚页相关部分	第 81 页 7.4 节
4	0	Context	指向内存中页表项的指针	第 84 页 7.5 节
4	2	UserLocal	存放用户信息，允许用户态软件通过 RDHWR 指令读取	第 85 页 7.6 节
5	0	PageMask	VTLB 页表大小控制	第 86 页 7.7 节
5	1	PageGrain	1KB 小页等页表属性控制	第 87 页 7.8 节
5	5	PWBase	页表基地址寄存器	第 88 页 7.9 节
5	6	PWField	配置各级页表地址索引位置	第 89 页 7.10 节
5	7	PWSize	配置各级页表指针大小	第 90 页 7.11 节
6	0	Wired	控制 VTLB 中固定项数目	第 91 页 7.12 节
6	6	PWCtl	控制多级页表配置	第 92 页 7.13 节
7	0	HWRENa	RDHWR 指令可访问寄存器使能控制	第 93 页 7.14 节
8	0	BadVAddr	记录最新地址相关例外的出错地址	第 94 页 7.15 节
9	0	Count	处理器时钟计数器	第 95 页 7.16 节
9	6	GSEBase	龙芯扩展例外入口基址寄存器	第 96 页 7.17 节
9	7	PGD	页表指针寄存器	第 97 页 7.18 节
10	0	EntryHi	VTLB 与 FTLB 表项高位内容	第 98 页 7.19 节
11	0	Compare	计时器中断控制	第 100 页 7.20 节
12	0	Status	处理器状态与控制寄存器	第 101 页 7.21 节
12	1	IntCtl	中断系统状态与控制寄存器	第 103 页 7.22 节
12	2	SRSCtl	影子寄存器状态与控制寄存器	第 104 页 7.23 节
13	0	Cause	存放上一次例外原因	第 105 页 7.24 节
14	0	EPC	存放上一次发生例外指令的 PC	第 107 页 7.25 节
15	0	PRId	处理器 ID	第 108 页 7.26 节
15	1	EBase	例外入口基址寄存器	第 109 页 7.27 节
16	0	Config	配置寄存器	第 110 页 7.28 节
16	1	Config1	配置寄存器 1	第 111 页 7.29 节
16	2	Config2	配置寄存器 2	第 112 页 7.30 节

Reg.	Sel.	寄存器名称	功能定义	索引
16	3	Config3	配置寄存器 3	第 113 页 7.31 节
16	4	Config4	配置寄存器 4	第 115 页 7.32 节
16	5	Config5	配置寄存器 5	第 117 页 7.33 节
16	6	GSConfig	龙芯扩展配置寄存器	第 118 页 7.34 节
17	0	LLAddr	存放 Load-Link 指令访问地址	第 121 页 7.35 节
20	0	XContext	扩展地址模式下页表指针	第 122 页 7.36 节
22	0	Diag	龙芯扩展诊断控制寄存器	第 123 页 7.37 节
22	1	GSCause	存放上一次龙芯扩展例外的补充信息	第 125 页 7.38 节
23	0	Debug	EJTAG Debug 寄存器	第 127 页 7.40 节
24	0	DEPC	存放上一次 EJTAG 调试例外的 PC	第 128 页 7.41 节
25	0-7	PerfCnt0-PerfCnt7	处理器核内部性能计数器访问接口	第 129 页 7.42 节
26	0	ErrCtl	Cache Parity/ECC 校验值寄存器	第 131 页 7.43 节
27	0	CacheErr	Cache Parity/ECC 校验状态与控制寄存器	第 132 页 7.44 节
27	1	CacheErr1	Cache Parity/ECC 校验状态与控制寄存器 1	第 134 页 7.45 节
28	0	TagLo	Cache Tag 访问接口低位部分	第 135 页 7.46 节
28	1	DataLo	Cache Data 访问接口低位部分	第 138 页 7.47 节
29	0	TagHi	Cache Tag 访问接口高位部分	第 139 页 7.48 节
29	1	DataHi	Cache Data 访问接口高位部分	第 140 页 7.49 节
30	0	ErrorEPC	存放上一次发生错误的指令的 PC	第 141 页 7.50 节
31	0	DESAVE	EJTAG 调试例外保存寄存器	第 142 页 7.51 节
31	2-7	KScratch1-KScratch6	核心态可访问便签寄存器 1~6	第 143 页 7.52 节

7.2 Index 寄存器 (CP0 Register 0, Select 0)

Index 寄存器是一个 32 位的可读写寄存器，其中存放的索引信息用于 TLBP、TLBR、TLBWI 指令访问 TLB 时使用。

图 7-1 说明了 Index 寄存器的格式；表 7-2 对 Index 寄存器各域进行了描述。

图 7-1 Index 寄存器格式

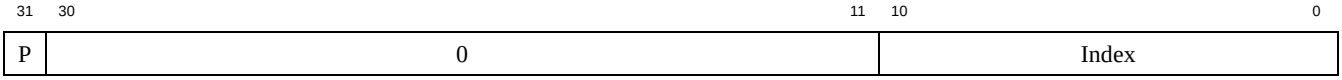


表 7-2 Index 寄存器域描述

域名称	位	功能描述	读/写	复位值
P	31	TLB 查询失败标志。当执行 TLBP 指令未能在 TLB 中找到匹配项时，该位置为 1；否则置为 0。	R	0x0
0	30..11	只读恒为 0。	0	0
Index	10..0	TLB 访问索引。软件通过配置该域，用以指示后续的 TLBR 或 TLBWI 指令读取或写入 TLB 的指定项。 当 TLBP 指令执行时，如果找到匹配项时，则将匹配项的索引值将存放在该域中；当未能找到匹配项时，Index 寄存器的 Index 域的内容可以为任意值。 Index 值的 0..63 用于指示 VTLB 的第 0..63 项。 Index 值的 64..1087 用于指示 FTLB。其中((Index-64) div 128)用以指示访问 FTLB 的哪一路，这一路中具体访问哪一项由((Index-64) mod 128)的值确定。例如，当 Index 值为 798 时，其指示访问第((798-64) div 128 = 5)路的第((798-64) mod 128 = 94)项。	R/W	0x0

编程提示：

Index 域的合理取值范围时 0 ~ 1087，当写入 Index 域的值超过该范围时，处理器结果将不确定。

7.3 Random 寄存器 (CP0 Register 1, Select 0)

Random 寄存器是一个只读寄存器，用于存放 TLBWR 指令访问 TLB 的索引值。Random 寄存器中所存放的索引值每个时钟周期都发生变化，其值变化的上界(含)是 63，变化的下界(含)是 Wired 寄存器中设置的值。Random 寄存器在发生 Reset 例外和 Wired 寄存器被写入时将自动复位为上界，即 63。

当 PageMask 寄存器中所表示的页大小的值与 FTLB 中所配置的页大小不一致时，TLBWR 指令会只操作 VTLB，由当前 Random 寄存器中的值决定写入到 VTLB 的哪一项中。

当 PageMask 寄存器中所表示的页大小的值与 FTLB 中所配置的页大小一致时，TLBWR 指令将只操作 FTLB，写入 FTLB 的哪一路由处理器内部随机决定，不使用 Random 寄存器中的内容。

图 7-2 说明了 Random 寄存器的格式；表 7-3 对 Random 寄存器各域进行了描述。

图 7-2 Random 寄存器格式

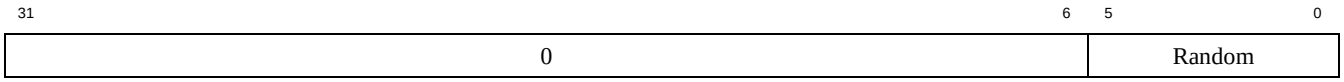


表 7-3 Random 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..6	只读恒为 0。	0	0
Random	5..0	VTLB 写入的随机索引值。	R	0x3f

7.4 EntryLo0 和 EntryLo1 寄存器 (CP0 Register 2 and 3, Select 0)

EntryLo0 和 EntryLo1 寄存器作为 TLBP、TLBR、TLBWI 和 TLBWR 指令访问 TLB 的接口，其中 EntryLo0 寄存器存放偶数页的信息 EntryLo1 寄存器存放奇数页的信息。

EntryLo0 和 EntryLo1 寄存器在使用 DMFC0/DMTC0 和 MFC0/MTC0 这两组指令访问时所呈现的格式存在差别。

图 7-3 说明了 EntryLo0 和 EntryLo1 寄存器在 DMFC0/DMTC0 指令访问时的格式；表 7-4 对这种情况下寄存器各域进行了描述。

图 7-3 EntryLo0 和 EntryLo1 在 DMFC0/DMTC0 指令访问时的寄存器格式

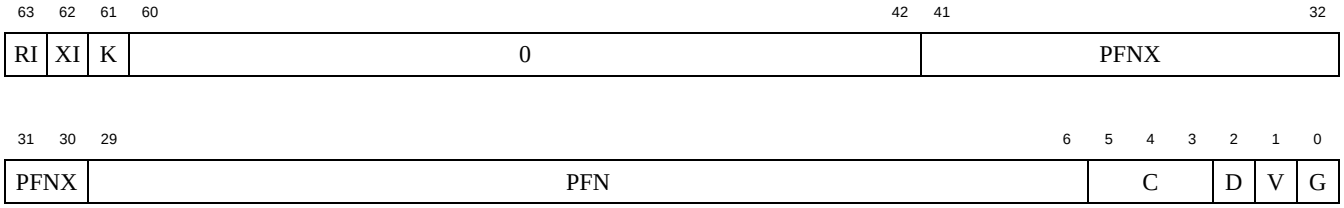


表 7-4 EntryLo0 和 EntryLo1 在 DMFC0/DMTC0 指令访问时的寄存器域描述

域名称	位	功能描述	读/写	复位值
RI	63	读阻止标识位。当某 TLB 表项的 RI 位置为 1 时，当有访问指令试图在该页上进行读取操作时，处理器将触发例外。根据 PageGrain 寄存器 IEC 域，所触发的例外可以是 TLBL invalid 例外或是 TLBRI 例外。 EntryLo0 和 EntryLo1 的 XI 域仅当 $PageGrain_{RIE}$=1 时才可以写入。$PageGrain_{RIE}$=0 时，则无论待写入值为何，EntryLo0 和 EntryLo1 的 RI 域都将读出为 0。	R/W	0x0
XI	62	执行阻止标识位。当某 TLB 表项的 XI 位置为 1 时，在该页上的发生取指，处理器将触发例外。根据 PageGrain 寄存器 IEC 域，所触发的例外可以是 TLBL invalid 例外或是 TLBXI 例外。 EntryLo0 和 EntryLo1 的 XI 域仅当 $PageGrain_{XIE}$=1 时才可以写入。$PageGrain_{XIE}$=0 时，则无论待写入值为何，EntryLo0 和 EntryLo1 的 XI 域都将读出为 0。	R/W	0x0
K	61	内核执行保护位。当处理器处于核心态时，在 K=0 的页上取指，处理器将触发 TLB Invalid 例外。 EntryLo0 和 EntryLo1 的 K 域仅当 GSConfig.KE=1 时才可以写入。GSConfig.KE=0 时，则无论待写入值为何，EntryLo0 和 EntryLo1 的 K 域都将读出为 0。	R/W	0x0
0	60..42	只读恒为 0。	0	0
PFNX	41..30	物理页号扩展部分。当处理器配置为支持大物理地址空间模式时（$Config3_{LPA}$=1 and $PageGrain_{ELPA}$=1），该域内容拼接至 PFN 域的高位形成完整的物理页号，从而支持 48 位物理地址空间。PFNX 域对应物理地址的 47..36 位。 如果处理器配置为不支持大物理地址空间模式时（$PageGrain_{ELPA}$=0[1]），PFNX 域将无法写入且读返回为 0。从而实现对基于 MIPS 规范 Release 1 编写的系统软件的兼容。	R/W	0x0

[1] 龙芯 3A1500 芯片中 $Config_{LPA}$ 恒为 1，因此不可能因为 $Config_{LPA}$=0 而关闭大物理地址空间模式支持。

域名称	位	功能描述	读/写	复位值
PFN	29..6	物理页号基本部分。当处理器配置为支持大物理地址空间模式时（$Config3_{LPA}$=1 and $PageGrain_{ELPA}$=1），该域内容拼接至 PFNX 域的低位形成完整的物理页号，从而支持 48 位物理地址空间。PFN 域对应物理地址的 35..12 位。 当处理器配置为不支持大物理地址空间模式时（$PageGrain_{ELPA}$=0），PFN 域其自身将形成最终的物理页号，从而支持 36 位物理地址空间。	R/W	0x0
C	5..3	该物理页的 Cache 属性。有关 Cache 属性的具体定义及其编码请参看本小节后面给出的表 7-6。	R/W	0x0
D	2	“脏”位。当页表中的脏位置为 1 时，表示该页可以写；否则对于一个脏位置为 0 的页进行写操作将会触发 TLB Mod 例外。	R/W	0x0
V	1	有效位。当页表中的有效位置为 1 时，表示该页可以访问；否则访问一个有效位置为 0 的页将会触发 TLB Invalid 例外。	R/W	0x0
G	0	全局位。当页表项填入 TLB 时，EntryLo0 和 EntryLo1 寄存器的两个 G 位值进行逻辑与，结果作为这一页表项的全局标识位。当页表中的全局标识位为 1 时，TLB 地址匹配查找时将不比较 ASID。 当从 TLB 读出页表项时，EntryLo0 和 EntryLo1 寄存器的两个 G 位同时反映所读出页表项的 G 位信息。	R/W	0x0

图 7-4 说明了 EntryLo0 和 EntryLo1 寄存器在 MFC0/MTC0 指令访问时的格式；表 7-5 对这种情况下寄存器各域进行了描述。需要注意的时，此时无法访问页表的 KE 位。TLB 表项的 KE 位将被写入缺省值 0。

图 7-4 EntryLo0 和 EntryLo1 在 MFC0/MTC0 指令访问时的寄存器格式

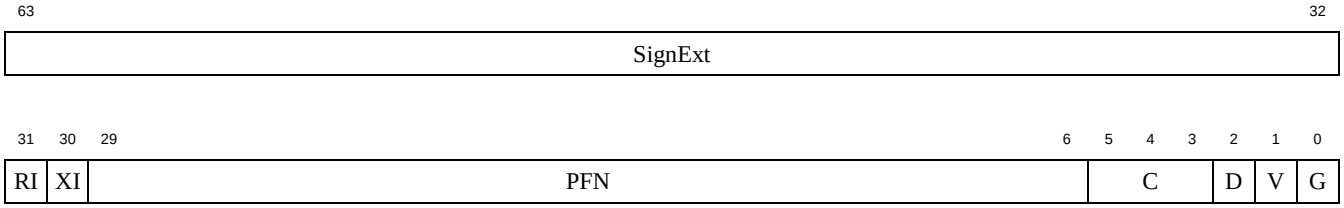


表 7-5 EntryLo0 和 EntryLo1 在 MFC0/MTC0 指令访问时的寄存器域描述

域名称	位	功能描述	读/写	复位值
SignExt	63..32	采用 MTC0 写入时，寄存器中对应此部分的内容忽略，PFNX 域写入 0。 采用 MFC0 读出时，返回至通用寄存器的这部分内容由 RI 位符号扩展得到。	R	0x0
RI	31	读阻止标识位。当某 TLB 表项的 RI 位置为 1 时，当有访问指令试图在该页上进行读取操作时，处理器将触发例外。根据 PageGrain 寄存器 IEC 域，所触发的例外可以是 TLBL invalid 例外或是 TLBRI 例外。 EntryLo0 和 EntryLo1 的 XI 域仅当 $PageGrain_{RIE}$=1 时才可以写入。$PageGrain_{RIE}$=0 时，则无论待写入值为何，EntryLo0 和 EntryLo1 的 RI 域都将读出为 0。	R/W	0x0

域名称	位	功能描述	读/写	复位值
XI	30	执行阻止标识位。当某 TLB 表项的 XI 位置为 1 时，在该页上的发生取指，处理器将触发例外。根据 PageGrain 寄存器 IEC 域，所触发的例外可以是 TLBL invalid 例外或是 TLBXI 例外。 EntryLo0 和 EntryLo1 的 XI 域仅当 $PageGrain_{XIE}$=1 时才可以写入。$PageGrain_{XIE}$=0 时，则无论待写入值为何，EntryLo0 和 EntryLo1 的 XI 域都将读出为 0。	R/W	0x0
PFN	29..6	物理页号基本部分。当处理器配置为支持大物理地址空间模式时（$Config3_{LPA}$=1 and $PageGrain_{ELPA}$=1），该域内容拼接至 PFNX 域的低位形成完整的物理页号，从而支持 48 位物理地址空间。PFN 域对应物理地址的 35..12 位。 当处理器配置为不支持大物理地址空间模式时（$PageGrain_{ELPA}$=0），PFN 域其自身将形成最终的物理页号，从而支持 36 位物理地址空间。	R/W	0x0
C	5..3	该物理页的 Cache 属性。有关 Cache 属性的具体定义及其编码请参看本小节后面给出的表 7-6。	R/W	0x0
D	2	“脏”位。当页表中的脏位置为 1 时，表示该页可以写；否则对于一个脏位置为 0 的页进行写操作将会触发 TLB Mod 例外。	R/W	0x0
V	1	有效位。当页表中的有效位置为 1 时，表示该页可以访问；否则访问一个有效位置为 0 的页将会触发 TLB Invalid 例外。	R/W	0x0
G	0	全局位。当页表项填入 TLB 时，EntryLo0 和 EntryLo1 寄存器的两个 G 位值进行逻辑与，结果作为这一页表项的全局标识位。当页表中的全局标识位为 1 时，TLB 地址匹配查找时将不比较 ASID。 当从 TLB 读出页表项时，EntryLo0 和 EntryLo1 寄存器的两个 G 位同时反映所读出页表项的 G 位信息。	R/W	0x0

编程提示：

当 PageGrain 寄存器的任一域内容被修改前，必须将 EntryLo0 和 EntryLo1 寄存器中的 PFNX 和 PFN 域写入 0，且清空所有 TLB。且上述所有操作必须在非映射地址空间(Unmapped Address Space)进行。如果未按照这里的要求进行操作，处理器行为将不确定。

表 7-6 Cache 属性编码表

页表 C 域编码	Cache 属性[1]
0	保留，强行配置会造成死机
1	保留，强行配置会造成死机
2	Uncached
3	Cached
4	保留，强行配置等效于 Cached
5	保留，强行配置等效于 Cached
6	保留，强行配置将访问 EJTAG dseg 空间，存在死机风险
7	Uncached Accelerated

[1] Uncached、Cacheable Coherent 和 Uncached Accelerated 属性的定义请参看 22 页 5.2 节。

7.5 Context 寄存器 (CP0 Register 4, Select 0)

Context 寄存器是一个可读写寄存器，其中包含由操作系统软件所填入的一些页表基地址高位信息和发生 TLB 例外的出错虚地址的某些位。按照 MIPS 架构最初的设计意图，Context 寄存器中被拼接在一起的信息可形成指向页表中某一项的指针，用于 TLB 例外发生时访问该页表项。Context 寄存器中内容不做任何处理时可访问的页表是一个单级页表结构，页大小为 4K 字节，每个页表项为 16 字节，包含虚地址连续的一个偶数页表项和一个奇数页表项，共 512K 个奇偶对页表项。当页表未采用这种结构时，软件需要对 Context 寄存器的内容进行适当地移位、拼接处理。对于一个采用多级页表的操作系统来说，Context 寄存器仅能用在加速最后一级页表访问的地址生成。

Context 寄存器主要被用在 TLB Refill 例外处理程序中。但是当 XTLB Refill、TLB Invalid 和 TLB Mod 例外发生时，Context 寄存器中的 BadVPN2 域也会被更新，因此软件也可以在相应的例外处理程序中使用 Context 寄存器。

Context 寄存器中的 BadVPN2 域复制了 BadVAddr 寄存器中的部分信息，但并不意味着这部分是完全等价的。当 Address Error 例外发生时，BadVaddr 寄存器会被硬件更新，但 Context 寄存器的 BadVPN2 域不会被硬件更新。

图 7-5 说明了 Context 寄存器的格式；表 7-7 对 Context 寄存器各域进行了描述。

图 7-5 Context 寄存器格式

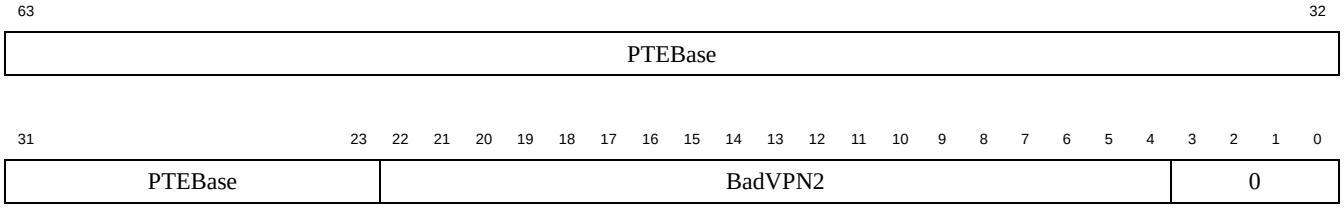


表 7-7 Context 寄存器域描述

域名称	位	功能描述	读/写	复位值
PTEBase	63..23	页表基地址高位。由操作系统软件根据当前页表情况进行配置。	R/W	无
BadVPN2	22..4	当发生 TLB 例外时，存放出错虚地址的 31..13 位。	R	无
0	3..0	只读恒为 0。	0	0

7.6 UserLocal 寄存器 (CP0 Register 4, Selelct 2)

UserLocal 寄存器是一个可读写寄存器，其不用于控制处理器硬件运行，也不会被硬件更改内容。UserLocal 的内容可以通过 RDHWR 指令在用户态读出，其能否读出受 HWREna 寄存器第 29 位控制。

图 7-6 说明了 UserLocal 寄存器的格式；表 7-8 对 UserLocal 寄存器各域进行了描述。

图 7-6 UserLocal 寄存器格式

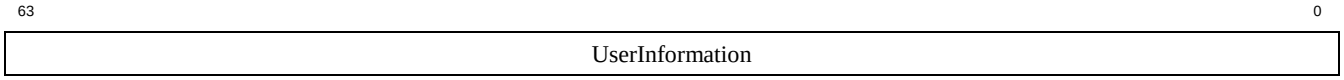


表 7-8 UserLocal 寄存器域描述

域名称	位	功能描述	读/写	复位值
UserInfor-mation	63..0	所存放的信息，不受处理器硬件影响，亦不影响处理器硬件运行。	R/W	无

7.7 PageMask 寄存器 (CP0 Register 5, Select 0)

PageMask 寄存器是个可读写的寄存器，在读写 TLB 的过程使用；它包含一个比较掩码，可为每个 TLB 表项设置不同的页大小。

图 7-7 说明了 PageMask 寄存器的格式；表 7-9 对 PageMask 寄存器各域进行了描述。

图 7-7 PageMask 寄存器格式

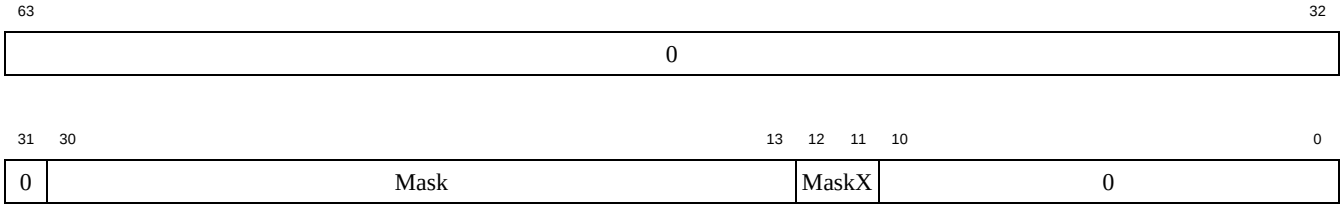


表 7-9 PageMask 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	63..31	只读恒为 0。	0	0
Mask	30..13	当进行虚实地址转换时，Mask 域[17:0]中的每一位分别用于指示虚地址[30:13]位中对应位是否进行比较。1：不比较；0：比较。 龙芯所支持的 Mask 编码及其对应的页大小请参看下面的表 7-10。	R/W	0x3
MaskX	12..11	恒为 3，不支持 1KB 大小的页。	R	0x3
0	10..0	只读恒为 0。	0	0

表 7-10 给出了 GS464E 所支持的 Mask 域编码及其对应的页大小。

表 7-10 Mask 域编码与页大小

Mask 编码	页大小
0x0	4KB
0x3	16KB
0xF	64KB
0x3F	256KB
0xFF	1MB
0x3FF	4MB
0xFFF	16MB
0x3FFF	64MB
0xFFFF	256MB
0x3FFFF	1GB

编程提示：

尽管 GS464E 允许对 PageMask 寄存器填入：0x1、0x7、0x1F、0x7F、0x1FF、0x7FF、0x1FFF、0x7FFF、0x1FFFF，此时页大小为 2^{2n+1}KB (n=0..8)，但处理器对此种配置下的程序运行的正确性不做保证。

7.8 PageGrain 寄存器 (CP0 Register 5, Select 1)

PageGrain 寄存器是一个可读写寄存器。GS464E 只实现其中与 TLB XI/RI 保护位、大物理地址模式控制相关的部分。

图 7-8 说明了 PageGrain 寄存器的格式；表 7-11 对 PageGrain 寄存器各域进行了描述。

图 7-8 PageGrain 寄存器格式

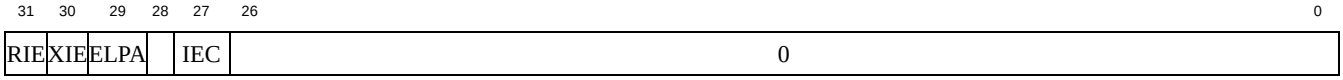


表 7-11 PageGrain 寄存器域描述

域名称	位	功能描述	读/写	复位值
RIE	31	TLB 页表读阻止(Read-Inhibit)功能使能位。 0：关闭该功能。EntryLo0 和 EntryLo1 寄存器的 RI 位将禁止写入，强制为 0； 1：开启该功能。EntryLo0 和 EntryLo1 寄存器的 RI 位可正常使用。	R/W	0x0
XIE	30	TLB 页表执行阻止(Execute-Inhibit)功能使能位。 0：关闭该功能。EntryLo0 和 EntryLo1 寄存器的 XI 位将禁止写入，强制为 0； 1：开启该功能。EntryLo0 和 EntryLo1 寄存器的 XI 位可正常使用。	R/W	0x0
ELPA	29	大物理地址功能使能位。 0：关闭该功能。EntryLo0 和 EntryLo1 寄存器的 PFNX 域将禁止写入，强制为 0； 1：开启该功能。EntryLo0 和 EntryLo1 寄存器的 PFNX 域可正常使用。	R/W	0x0
0	28	只读恒为 0。	0	0
IEC	27	TLB 读阻止和执行阻止例外编码及入口控制位。 0：TLB 读阻止和执行阻止例外编码及入口复用 TLBL 例外的编码与入口； 1：TLB 读阻止例外用 TLBRI 例外编码及入口，TLB 执行阻止例外用 TLBXI 例外编码及入口。	R/W	0x0
0	26..0	只读恒为 0。	0	0

编程提示：

在软件试图修改 PageGrain 任何域之前，必须将所有 TLB 清空，且下面列出的 COP0 寄存器的域必须置为指定值，否则处理器的行为将不确定。

COP0 寄存器域	指定值
EntryLo0PFN, EntryLo1PFN	0
EntryLo0PFNX, EntryLo1PFNX	0

7.9 PWBase 寄存器 (CP0 Register 5, Select 5)

PWBase 寄存器是一个 64 位的可读写寄存器，存放页表基地址的虚拟地址。该寄存器与 PWField、PWSize 和 PWCtl 寄存器配合在一起使用，在 GS464E 中用于为 LDDIR 和 LDPTE 指令的执行提供相关配置信息。LDDIR 和 LDPTE 指令支持多级页表结构的遍历查找，该页表中可包含最多四级目录表和一级页表项表，所支持的页表结构示意请见图 7-9。

图 7-9 PWBase,PWField,PWSize 与 PWCtl 所支持的页表访问过程

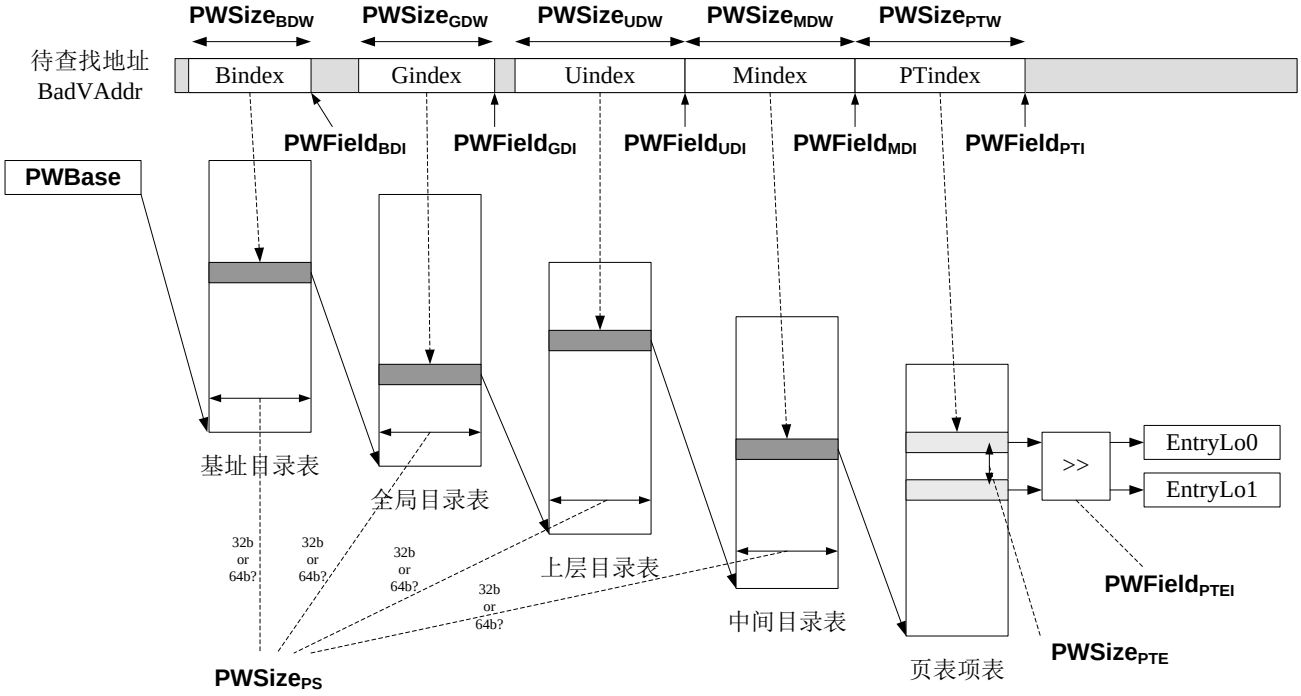


图 7-10 说明了 PWBase 寄存器的格式；表 7-12 对 PWBase 寄存器各域进行了描述。

图 7-10 PWBase 寄存器格式

63	0
PWBase	

表 7-12 PWBase 寄存器域描述

域名称	位	功能描述	读/写	复位值
PWBase	63..0	页表基地址。	R/W	0x0

7.10 PWField 寄存器 (CP0 Register 5, Select 6)

PWField 寄存器与 PWBase、PWSize 和 PWCtl 寄存器配合在一起使用，在 GS464E 中用于为 LDDIR 和 LDPTE 指令的执行提供相关配置信息。LDDIR 和 LDPTE 指令支持多级页表结构的遍历查找，该页表中可包含最多四级目录表和一级页表项表，所支持的页表结构及访问过程请见第 88 页图 7-9。各级页表中存放指向下一级页表或最终页表项的索引值是通过从待查找虚地址(BadVAddr)中截取部分连续比特而得到的。PWField 寄存器用于标识各级页表索引在待查找虚地址(BadVAddr)中截取的起始位置。

图 7-11 说明了 PWField 寄存器的格式；表 7-13 对 PWField 寄存器各域进行了描述。

图 7-11 PWField 寄存器格式

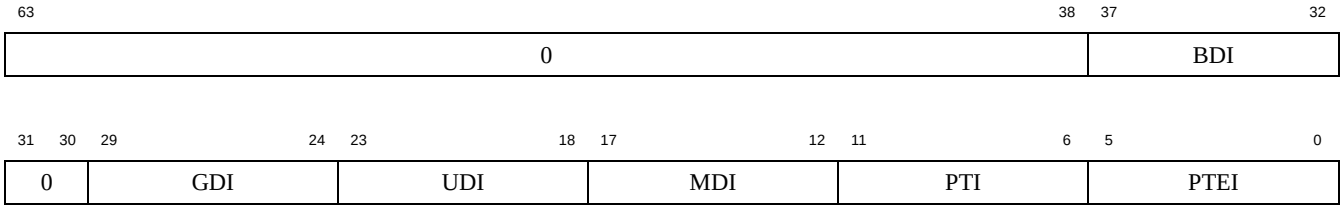


表 7-13 PWField 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	63..38	只读恒为 0。	0	0
BDI	37..32	基址目录表(Base Directory)索引起始位置。 基址目录表是否使用由 $PWCtl_{PWDirExt}$ 控制。基址目录表可以被用来区分不同的页表，譬如用户页表和内核页表可以分开维护。	R/W	0x0
0	31..30	只读恒为 0。	0	0
GDI	29..24	全局目录表(Global Directory)索引起始位置。	R/W	0x0
UDI	23..18	上层目录表(Upper Directory)索引起始位置。	R/W	0x0
MDI	17..12	中间目录表(MIddle Directory)索引起始位置。	R/W	0x0
PTI	11..6	页表项表(Page Table)索引起始位置。	R/W	0x0
PTEI	5..0	页表项移位量。 最终读出的页表项内容将先逻辑右移 PTEI-2 比特，用于移除页表项中仅用于软件而无需放入 TLB 的信息；随后再循环右移 2 比特，将 RI、XI 两比特信息移动至 EntryLo0 或 EntryLo1 的最高两位上。因此 PTEI 域不能填入 0 或 1，否则处理器结果将不确定。	R/W	0x0

7.11 PWSize 寄存器 (CP0 Register 5, Select 6)

PWSize 寄存器与 PWBase、PWField 和 PWCtl 寄存器配合在一起使用，在 GS464E 中用于为 LDDIR 和 LDPTE 指令的执行提供相关配置信息。LDDIR 和 LDPTE 指令支持多级页表结构的遍历查找，该页表中可包含最多四级目录表和一级页表项表，所支持的页表结构及访问过程请见第 88 页图 7-9。各级页表中存放指向下一级页表或最终页表项的索引值是通过从待查找虚地址(BadVAddr)中截取部分连续比特而得到的。PWSize 寄存器用于标识各级页表索引在待查找虚地址(BadVAddr)中截取的连续比特的位数。

$PWSize_{PS}$ 域用于控制目录表中指针的位宽是 32 位还是 64 位。各级目录表根据 PWField 和 PWSize 从待查找虚地址(BadVAddr)中截取的索引值需要乘以目录表中指针的宽度（左移 2 位或左移 3 位）形成真正的访存地址。XTLB Refill 例外处理中只能使用 64 位指针，TLB Refill 例外中可以使用 32 位或 64 位指针。

图 7-12 说明了 PWField 寄存器的格式；表 7-13 对 PWField 寄存器各域进行了描述。

图 7-12 PWSize 寄存器格式

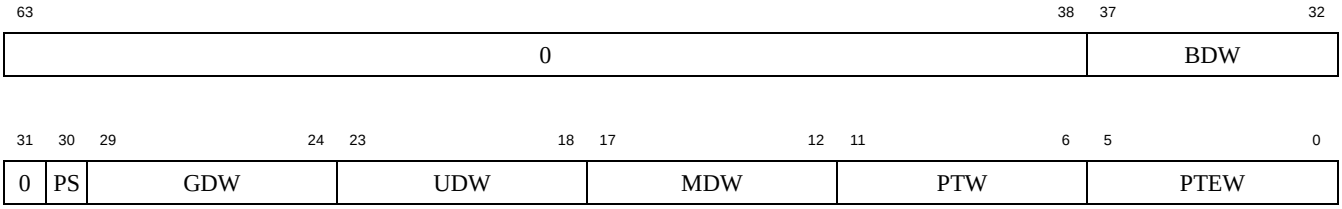


表 7-14 PWSize 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	63..38	只读恒为 0。	0	0
BDW	37..32	基址目录表索引位宽。为 0 表示不需要查找基址目录表。	R/W	0x0
0	31	只读恒为 0。	0	0
PS	30	各级目录表指针位宽。0：32 位指针；1：64 位指针。	R/W	0x0
GDW	29..24	全局目录表索引位宽。为 0 无意义，配置为 0 时处理器结果不确定。	R/W	0x0
UDW	23..18	上层目录表索引位宽。为 0 无意义，配置为 0 时处理器结果不确定。	R/W	0x0
MDW	17..12	中间目录表索引位宽。为 0 无意义，配置为 0 时处理器结果不确定。	R/W	0x0
PTW	11..6	页表项表索引位宽。为 0 无意义，配置为 0 时处理器结果不确定。	R/W	0x0
PTEW	5..0	页表项位宽。用于控制访问页表项索引在左移 3 位（机器默认数据通路宽度为 64 位）后需进一步左移的位数。例如，当每项页表项（单页）宽度是 16 字节时，PTEW 硬设置为 1，访问页表项索引将左移(3+1)位后形成最终的访存地址。	R/W	0x0

7.12 Wired 寄存器 (CP0 Register 6, Select 0)

Wired 寄存器是一个可读写寄存器，用于定义 VTLB 中固定表项与随机替换表项之间的边界。其示意如图 7-13 所示。

图 7-13 VTLB 中固定表项与随机替换表项边界

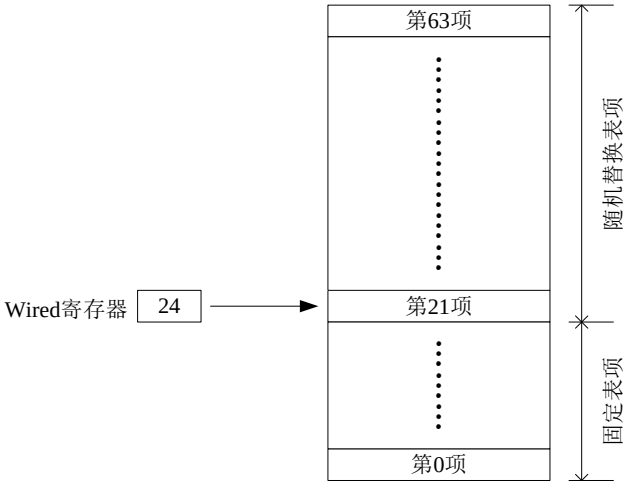


图 7-14 说明了 Wired 寄存器的格式；表 7-15 对 Wired 寄存器各域进行了描述。

图 7-14 Wired 寄存器格式

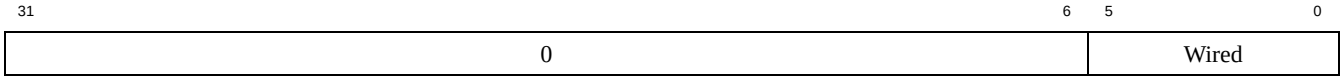


表 7-15 Wired 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..6	只读恒为 0。	0	0
Wired	5..0	TLB 固定表项与随机替换表项之间的边界。	R/W	0x0

7.13 PWCtl 寄存器 (CP0 Register 6, Select 6)

PWCtl 寄存器与 PWBase、PWField 和 PWSize 寄存器配合在一起使用，在 GS464E 中用于为 LDDIR 和 LDPTE 指令的执行提供相关配置信息。LDDIR 和 LDPTE 指令支持多级页表结构的遍历查找，该页表中可包含最多四级目录表和一级页表项表，所支持的页表结构及访问过程请见第 88 页图 7-9。各级页表中存放指向下一级页表或最终页表项的索引值是通过从待查找虚地址(BadVAddr)中截取部分连续比特而得到的。PWSize 寄存器用于标识各级页表索引在待查找虚地址(BadVAddr)中截取的连续比特的位数。

PWCtl 用于控制页表遍历查找中是否使用基址目录表，以及大页的支持。

图 7-15 说明了 PWCtl 寄存器的格式；表 7-16 对 PWCtl 寄存器各域进行了描述。

图 7-15 PWCtl 寄存器格式

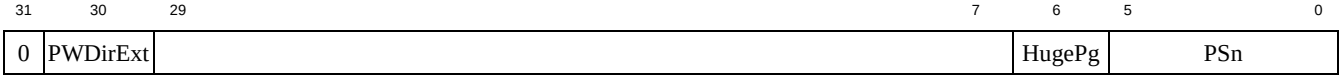


表 7-16 PWCtl 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31	只读恒为 0。	0	0
PWDirExt	30	基址目录(Base Directory)表功能使能位。1：启用；0：禁用。	R/W	0x0
0	29..7	只读恒为 0。	0	0
HugePg	6	为 1 表示目录表中支持大页；为 0 表示目录表中不支持大页。	R/W	0x0
PSn	5..0	用于指出目录表中的表项的 PTEVld 位的位置。	R/W	0x0

7.14 HWREna 寄存器 (CP0 Register 7, Select 0)

HWREna 寄存器中存放一个比特掩码，用于控制 RDHWR 指令在用户态下可以读取哪些硬件寄存器。图 7-16 说明了 HWREna 寄存器的格式；表 7-17 对 HWREna 寄存器各域进行了描述。

图 7-16 HWREna 寄存器格式

31	30	29	28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3	2	1	0
FreqScale	ExtCC	ULR	0	CCRes	CC	SYNCIStep	CPUNum

表 7-17 HWREna 寄存器域描述

域名称	位	功能描述	读/写	复位值
FreqScale	31	RDHWR 31 (处理器核分频系数) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
ExtCC	30	RDHWR 30 (外部 Count 寄存器) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
ULR	29	RDHWR 29 (UserLocal 寄存器) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
0	28..4	只读恒为 0。	0	0
CCRes	3	RDHWR 3 (Count 自增频度) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
CC	2	RDHWR 2 (Count 寄存器) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
SYNCI-Step	1	RDHWR 1 (SYNCI_Step) 使能位。1：允许读取；0：禁止读取。	R/W	0x0
CPUNum	0	RDHWR 0 ($EBase_{CPUNum}$)使能位。1：允许读取；0：禁止读取。	R/W	0x0

7.15 BadVAddr 寄存器 (CP0 Register 8, Select 0)

BadVAddr 寄存器是一个只读寄存器，用于记录最近一次导致发生下列例外的虚地址：

- Address error (AdEL 或 AdES)
- TLB/XTLB Refill
- TLB Invalid (TLBL 或 TLBS)
- TLB Modified

图 7-17 说明了 BadVAddr 寄存器的格式；表 7-18 对 BadVAddr 寄存器各域进行了描述。

图 7-17 BadVAddr 寄存器格式

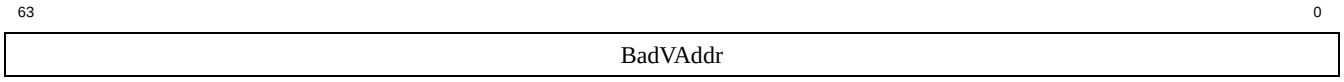


表 7-18 BadVAddr 寄存器域描述

域名称	位	功能描述	读/写	复位值
BadVAddr	63..0	出错的虚地址。	R	无

7.16 Count 寄存器 (CP0 Register 9, Select 0)

Count 寄存器与 Compare 寄存器配合在一起，用于实现一个处理器内部的高精度定时器及定时中断。该计时器自增 1 的频率是处理器核流水线时钟的频率的 1/2。在执行过程中，处理器核流水线时钟频率可能被动态调整，因此 Count 的自增频率也随之变化。

为完成某些功能或诊断目的，软件可以配置 Count 寄存器，例如计时器的复位、同步等操作。

说明了 Count 寄存器的格式；对 Count 寄存器各域进行了描述。

图 7-18 Count 寄存器格式

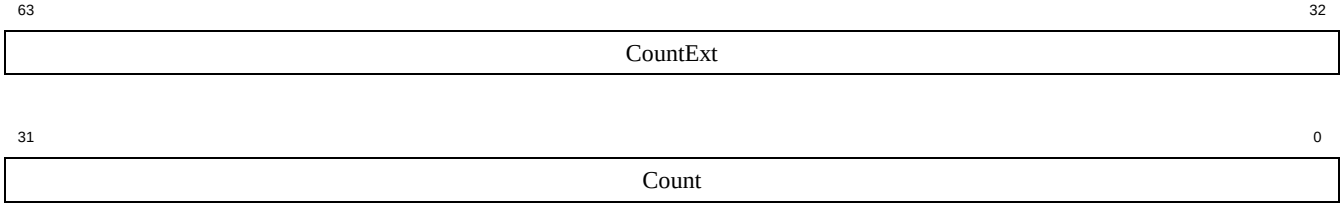


表 7-19 Count 寄存器域描述

域名称	位	功能描述	读/写	复位值
CountExt	63..32	内部计数器高 32 位扩展。	R/W	无
Count	31..0	内部计数器。	R/W	无

编程提示：

本处理器实现的 Count 是一个 64 位的计数器。处理器内部又 Count/Compare 产生的定时中断仍是通过 Count 的低 32 位值与 Compare 寄存器值的比较而触发的。RDHWR (rd=$3)返回 Count 寄存器低 32 位符号扩展至 64 位的结果。MFC0 指令只能读出 Count 寄存器低 32 位符号扩展至 64 位的结果，但是使用 MTC0 指令写 Count 寄存器时，处理器会将源寄存器中的 64 位值全部写入 Count 寄存器中。因此建议：如果软件希望访问 Count 寄存器中完整的 64 位信息，请使用 DMFC0、DMTC0 指令。

7.17 GSEBase 寄存器 (CP0 Register 9, Select 6)

GSEBase 寄存器是一个可读写寄存器，用于配制龙芯扩展例外向量基地址。

图 7-20 说明了 GSEBase 寄存器的格式；表 7-21 对 GSEBase 寄存器各域进行了描述。

图 7-19 GSEBase 寄存器格式

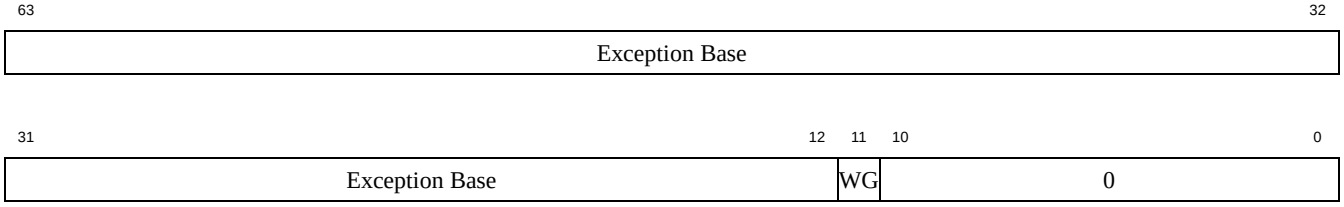


表 7-20 GSEBase 寄存器域描述

域名称	位	功能描述	读/写	复位值
Exception Base	63..12	当 Status.BEV=0 时，逻辑左移 12 位作为龙芯扩展例外入口向量基址。 [63:30]位仅当 WG 位等于 1 时才可以写入，当 WG 位等于 0 时写 EBase 寄存器时[63:30]位保持不变。	R/W	0xffff.ffff.8000.0
WG	11	[63:30]位的写控制位。 1：ExceptionBase[63:30]可写； 0：ExceptionBase[63:30]写入时值保持不变。	R/W	0x0
0	10..0	只读恒为 0。	0	0

7.18 PGD 寄存器 (CP0 Register 9, Select 7)

PGD 寄存器是一个只读寄存器，存放页表的基地址。

图 7-20 说明了 PGD 寄存器的格式；表 7-21 对 PGD 寄存器各域进行了描述。

图 7-20 PGD 寄存器格式

表 7-21 PGD 寄存器域描述

域名称	位	功能描述	读/写	复位值
Page Table Base	63..0	当 BadVaddr 寄存器中第 63 位为 1，则 Page Table Base 的值等于 KScratch6 寄存器中存放的值； 当 BadVaddr 寄存器中第 63 位为 0，则 Page Table Base 的值等于 PWBase 寄存器中存放的值。	R	0x0

7.19 EntryHi 寄存器 (CP0 Register 10, Select 0)

EntryHi 寄存器用于存放 TLB 读、写、查询访问时 TLB 表项的高位信息。

在普通指令执行情况下，$EntryHi_{ASID}$ 域存放由软件填入的当前地址空间标识，与取指或访存操作的虚地址一起参与 TLB 查找。当发生 TLB 例外(TLB Refill, XTLB Refill, TLB Invalid 或 TLB Modifid 例外)时，触发例外的错误地址的相应部分被写入 $EntryHi_R$ 和 $EntryHi_{VPN2}$ 域。当执行 TLBP 指令时，待查找项的虚地址信息和地址空间标识信息存放在 $EntryHi_R$、$EntryHi_{VPN2}$ 和 $EntryHi_{ASID}$ 域，用于进行 TLB 查找。

当执行 TLBR 指令时，从指定 TLB 表项读出的内容也将写入 EntryHi 寄存器的相应域中。由于执行 TLBR 指令会覆盖 $EntryHi_{ASID}$ 域，因此软件必须在执行 TLBR 指令前保存 ASID 域值，并在 TLBR 执行结束后及时的恢复。这点对于 TLB Invalid 和 TLB Modified 例外处理程序，以及其它相关内存管理代码尤为重要。

当执行 TLBWI 和 TLBWR 指令时，待写入项的虚地址信息和地址空间标识信息存放在 $EntryHi_R$、$EntryHi_{VPN2}$ 和 $EntryHi_{ASID}$ 域，用于 TLB 写入。执行 TLBWI 指令时，若将 $EntryHi_{EHINV}$ 域置为 1，可以将指定的 TLB 表项无效。由于 $EntryHi_{EHINV}$ 域也会被 TLBR 指令读出的内容覆盖，因此执行 TLBR 指令时也需要像维护 ASID 域一样维护好 EHINV 域，这对于后续执行的 TLBWI 指令能否正确执行很重要。

图 7-21 说明了 EntryHi 寄存器的格式；表 7-22 对 EntryHi 寄存器各域进行了描述。

图 7-21 EntryHi 寄存器格式

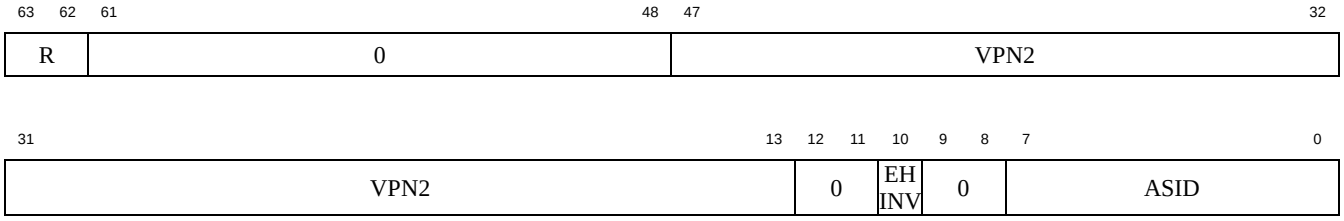


表 7-22 EntryHi 寄存器域描述

域名称	位	功能描述	读/写	复位值
R	63..62	区域标识位，对应虚地址的[63:62]位。 0b00：用户寻址区域(xuseg, user address region)； 0b01：监管寻址区域(xsseg, supervisor address region)； 0b10：保留； 0b11：核心地址区域(xkseg, kernel address region)。	R/W	0x0
0	61..48	只读恒为 0。	0	0
VPN2	47..13	虚页号除 2(映射到双页)，对应虚地址的[47:13]位。	R/W	0x0
0	12..11	只读恒为 0。	0	0
EHINV[1]	10	TLB 无效标记位。 当该位置为 1 时，执行 TLBWI 指令会将对应 TLB 表项无效。 当执行 TLBR 指令时，若读出的 TLB 表项无效，则该位被置为 1。	R/W	0x0
0	9..8	只读恒为 0。	0	0

[1] 尽管 $Config4_{IE}$=0，但 GS464E 仍按照 MIPS 规范在 $Config4_{IE}$>1 情况下的定义实现了 EHINV 域。建议对 GS464E 深度定制的内核等软件使用 $EntryHi_{EHINV}$ 域功能，方便 TLB 的管理。

域名称	位	功能描述	读/写	复位值
ASID	7..0	地址空间标识号。用于让多个进程共享 TLB；通过利用 ASID 进行区分，对于相同的虚页号，可以使不同进程采用不同的映射。	R/W	0x0

7.20 Compare 寄存器 (CP0 Register 11, Select 0)

Compare 寄存器与 Count 寄存器配合在一起，用于实现一个处理器内部的高精度定时器及定时中断。Compare 寄存器所存放的值在写入后保持不变，与 Count 寄存器的低 32 位进行比较，当两者相等时则触发计时器中断，将 $Cause_{IT}$ 置 1。在未使用向量中断模式时，计时器中断将连接至中断线 7 上（$Cause_{IP7}$，硬中断线 5）。在使用向量中断模式时，计时器中断连接至哪个中断线上由 $IntCtl_{IPTI}$ 确定。

软件写 Compare 寄存器时，硬件将自动对 $Cause_{IT}$ 清 0，从而清除计时器中断。

图 7-22 说明了 Compare 寄存器的格式；表 7-23 对 Compare 寄存器各域进行了描述。

图 7-22 Compare 寄存器格式

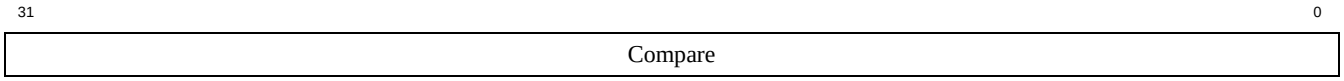


表 7-23 Compare 寄存器域描述

域名称	位	功能描述	读/写	复位值
Compare	31..0	间隔计数比较值。	R/W	0x0

7.21 Status 寄存器 (CP0 Register 12, Select 0)

Status 寄存器是一个可读写寄存器，包含有处理器操作模式、中断使能以及处理器状态诊断信息。

图 7-23 说明了 Status 寄存器的格式；表 7-24 对 Status 寄存器各域进行了描述。

图 7-23 Status 寄存器格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17 16	15 14 13 12 11 10 9 8	7	6	5	4 3	2	1	0
0	CU2	CU1	CU0	RP	FR	0	MX	PX	BEV	0	SR	NMI	0	MID	IM7..IM0	KX	SX	UX	KSU	ERL	EXL	IE

表 7-24 Status 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31	只读恒为 0。	0	0
CU2	30	协处理器 2 可用标识位。1：可用；0：禁用。	R/W	0x0
CU1	29	协处理器 1(浮点协处理器)可用标识位。1：可用；0：禁用。	R/W	0x1
CU0	28	协处理器 0 可用标识位。1：可用；0：禁用。 当处理器处于 Kernel 模式和 Debug 模式时，协处理器 0 总是可以使用，无需考虑 CU0 位是否为 1。	R/W	0x1
RP	27	处理器动态降频使能位。1：开启；0：关闭。	R/W	0x0
FP	26	浮点寄存器模式控制位。 0：16 个浮点寄存器，偶数编号，每个 64 位，单精度存放在低 32 位上； 1：32 个浮点寄存器，连续编号，每个 64 位，单精度存放在低 32 位上。	R/W	0x0
0	25	只读恒为 0。	0	0
MX	24	访问 DSP 资源的使能位。1：可访问；0：禁止访问。	R/W	0x0
PX	23	用户模式下 64 位操作的使能控制位，并不用于使能 64 位地址空间。（其余模式下的 64 位操作无需使能） 1：使能；0：禁用。	R/W	0x1
BEV	22	例外向量入口地址控制。0：正常；1：启动。	R/W	0x1
0	21	只读恒为 0。	0	0
SR	20	用于指示导致进入 reset 例外向量入口是由于软复位(Soft Reset)。 1：是软复位；0：不是软复位(可能是 NMI 或 Reset)。 当该位时 0 时，硬件将忽略软件对该位写 1 的动作，即无法通过软件形成该位从 0→1 的跳变。	R/W	0x0
NMI	19	用于指示导致进入 reset 例外向量入口是由于不可屏蔽中断(NMIt)。 1：是不可屏蔽中断；0：不是不可屏蔽中断(可能是 Reset 或 Soft Reset)。 当该位时 0 时，硬件将忽略软件对该位写 1 的动作，即无法通过软件形成该位从 0→1 的跳变。	R/W	0x0
0	18	只读恒为 0。	0	0

域名称	位	功能描述	读/写	复位值
MID	17..16	龙芯自定义域，用于表示当前默认操作的地址空间是哪一个。取指所落在的地址空间是否受控于 MID 域由 Diag.INST 域决定。访存操作若没有携带 MID 信息将落在 Root.Status.MID 域指定的地址空间上，否则其落在的地址空间由其自身携带的 MID 信息决定。 GS464E 采用 2 位编码(MID)标识出 4 个彼此隔离的全地址空间。MID=0 的地址空间是 MIPS 主机可见的地址空间，MID=1 的地址空间是客模式下虚拟机可见的地址空间，MID 的其它值可由软件分配给其它虚拟机。 MID 域仅在 Diag.VMM=1 时可写，Diag.VMM=0 时无论写何值该域都将被置为 0。	R/W	0x0
IM7..IM0	15..8	中断屏蔽位。每一位分别控制一个外部中断、内部中断或软件中断的使能。 1：使能；0：屏蔽。	R/W	0x0
KX	7	1：能访问 64 位 Kernel 段，Kernel 段访问使用 XTLB Refill 例外向量； 0：不能访问 64 位 Kernel 段，Kernel 段访问使用 TLB Refill 例外向量。	R/W	0x1
SX	6	1：能访问 64 位 Supervisor 段，Supervisor 段访问使用 XTLB Refill 例外向量； 0：不能访问 64 位 Supervisor 段，Supervisor 段访问使用 TLB Refill 例外向量。	R/W	0x1
UX	5	1：能访问 64 位 User 段，User 段访问使用 XTLB Refill 例外向量，允许用户模式下指令操作 64 位数据； 0：不能访问 64 位 User 段，User 段访问使用 TLB Refill 例外向量，不允许用户模式下指令操作 64 位数据。	R/W	0x1
KSU	4..3	处理器模式标识位。 0b00：核心态 (Kernel Mode) 0b01：监管态 (Supervisor Mode) 0b10：用户态 (User Mode) 0b11：保留。	R/W	0x0
ERL	2	错误级。当发生 Reset、Soft Reset、NMI 和 Cache Error 例外时该位被置 1。 0：正常级；1：错误级。 当 ERL 位置为 1 时： • 处理器自动处于核心态 • 所有硬件与软件中断被屏蔽 • ERET 指令将从 ErrorEPC 寄存器读取返回地址 • kuseg 段将视作具备直接映射非缓存属性(unmapped and uncached)	R/W	0x1
EXL	1	例外级。当发生一个不是 Reset、Soft Reset、NMI 和 Cache Error 例外的例外时该位被置 1。0：正常级；1：例外级。 当 EXL 位置为 1 是： • 处理器自动处于核心态 • 所有硬件与软件中断被屏蔽 • TLB/XTLB Refill 例外处理采用通用例外向量入口而非 TLB/XTLB Refill 例外向量入口 • EPC、$Cause_{BD}$ 在发生新的例外时不做更新。	R/W	0x0
IE	0	全局中断使能位。 0：屏蔽所有硬件和软件中断； 1：使能所有硬件和软件中断。	R/W	0x0

7.22 IntCtl 寄存器 (CP0 Register 12, Select 1)

IntCtl 寄存器是一个可读写寄存器，用于对处理器的中断机制进行扩展。

图 7-24 说明了 IntCtl 寄存器的格式；表 7-25 对 IntCtl 寄存器各域进行了描述。

图 7-24 IntCtl 寄存器格式

31 29	28 26	25 10	9 5	4 0
IPTI	IPPCI	0	VS	0

表 7-25 IntCtl 寄存器域描述

域名称	位	功能描述	读/写	复位值
IPTI	31..29	用于在向量中断模式下指示计时器中断合并在哪一个中断线上。 值恒为 7，表示合并在 IP7，硬件中断线 HW5 上。	R	0x7
IPPCI	28..26	用于在向量中断模式下指示性能计数器溢出中断合并在哪一个中断线上。 值恒为 7，表示合并在 IP7，硬件中断线 HW5 上。	R	0x7
0	25..10	只读恒为 0。	0	0
VS	9..5	用于定义向量中断模式下各中断向量入口地址的间距。 **VS 编码** / **向量间距** 0x00 / 0x000 0x01 / 0x020 0x02 / 0x040 0x04 / 0x080 0x08 / 0x100 0x10 / 0x200 除上表所列的其它编码值均被保留。如果对 VS 域配置了被保留的值，处理器结果将不确定。	R/W	0x0
0	4..0	只读恒为 0。	0	0

7.23 SRSCtl 寄存器 (CP0 Register 12, Select 2)

SRSCtl 寄存器用于控制影子寄存器。因 GS464E 只实现了一组通用寄存器，所以通用寄存器的影子就是通用寄存器本身。

图 7-25 说明了 SRSCtl 寄存器的格式；表 7-26 对 SRSCtl 寄存器各域进行了描述。

图 7-25 SRSCtl 寄存器格式

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

表 7-26 SRSCtl 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..30	只读恒为 0。	0	0
HSS	29..26	值为 0，表示只实现一组通用寄存器。	R	0x0
0	25..16	只读恒为 0。	0	0
ESS	15..12	用于例外处理的影子寄存器组的组号。GS464E 只能写入 0，写入其它值处理器行为不确定。	R/W	0x0
0	11..10	只读恒为 0。	0	0
PSS	9..6	前一组影子寄存器的组号。GS464E 只能写入 0，写入其它值处理器行为不确定。	R/W	0x0
0	5..4	只读恒为 0。	0	0
CSS	3..0	值永远为 0，表示用当前的影子寄存器组就是通用寄存器组。	R	0x0

7.24 Cause 寄存器 (CP0 Register 13, Select 0)

Cause 寄存器主要用于描述最近一次例外的原因。除此之外还对软件中断、中断向量进行了控制。除了 IP1..0、DC、IV 和 WP 域外，Cause 寄存器的其它域对于软件均只读。

图 7-26 说明了 Cause 寄存器的格式；表 7-27 对 Cause 寄存器各域进行了描述。

图 7-26 Cause 寄存器格式

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

表 7-27 Cause 寄存器域描述

域名称	位	功能描述	读/写	复位值
BD	31	标识最近发生例外的指令是否处于分支延迟槽。1：在延迟槽中；0：不在延迟槽中	R	0x0
TI	30	计时器中断指示。1：有待处理的计时器中断；0：没有计时器中断。	R	0x0
CE	29..28	发生协处理器不可用例外时记录不可用的协处理器号。	R	0x0
DC	27	Count 寄存器禁止计数控制位。1：停止 Count 计数；0：使能 Count 计数。	R/W	0x0
PCI	26	性能计数器溢出中断指示。 1：有待处理的性能计数器溢出中断；0：没有性能计数器溢出中断。	R	0x0
0	25..24	只读恒为 0。	0	0
IV	23	中断例外向量入口控制位。 1：使用特殊中断向量(0x200)；0：使用通用例外向量(0x180)。	R/W	0x0
0	22..16	只读恒为 0。	0	0
IP7..IP2	15..10	待处理硬件中断标识。每一位对应一个中断线，IP7~IP2 依次对应硬件中断 5~0。 1：该中断线上由待处理的中断；0：该中断线上无中断。	R	0x0
IP1..IP0	9..8	待处理软件中断标识。每一位对应一个软件中断，IP1~IP0 依次对应软件中断 1~0。 软件中断标识位可由软件设置和清除。	R/W	0x0
0	7	只读恒为 0。	0	0
ExcCode	6..2	例外编码。详细描述请见。		
0	1..0	只读恒为 0。	0	0

表 7-28 ExcCode 编码及其对应例外类型

ExcCode	助记符	描述
0x00	Int	中断
0x01	Mod	TLB 修改例外
0x02	TLBL	TLB 例外（读数据或取指令）
0x03	TLBS	TLB 例外（写数据）
0x04	AdEL	地址错例外（读数据或取指令）
0x05	AdES	地址错例外（写数据）

ExcCode	助记符	描述
0x06	IBE	MIPS 规定定义的总线错例外（取指令）。因为 GS464E 并未实现 IBE 例外，故该例外原因编码保留。 软件调试中若发现系统打印“Bus Error”信息请重点查看是否出现其它异常。
0x07	DBE	总线错例外（读数据或写数据）。因为 GS464E 并未实现 DBE 例外，故该例外原因编码保留。 软件调试中若发现系统打印“Bus Error”信息请重点查看是否出现其它异常。
0x08	Sys	系统调用例外。
0x09	Bp	断点例外。 如果在 EJTAG Debug 模式下执行 SDBBP 指令，例外编码 0x9(Bp)将写入 Debug 寄存器的 DExcCode 域。
0x0a	RI	保留指令例外。
0x0b	CpU	协处理器不可用例外。
0x0c	Ov	算出溢出例外。
0x0d	Tr	陷阱例外。
0x0e	MSAFPE	未实现。
0x0f	FPE	浮点例外。
0x10	GSExc	龙芯自定义例外。包括浮点栈例外、虚拟机内存管理例外、虚拟机空间 TLB 例外。软件可以通过查看 GSCause 寄存器的相关域明确具体发生何种例外。
0x11	-	保留
0x12	-	保留
0x13	TLBRI	TLB 读阻止例外
0x14	TLBXI	TLB 执行阻止例外
0x15	MSADis	未实现。
0x16	MDMX	未实现。
0x17	WATCH	未实现。
0x18	MCheck	未实现。
0x19	Thread	未实现。
0x1a	DSPDis	DSP 模块禁用例外
0x1b	GE	未实现。
0x1c	-	保留
0x1d	-	保留
0x1e	-	Cache 错例外。 因为 Cache 错例外采用专用的向量入口地址，所以普通模式下发生 Cache 错例外时 Cause 寄存器的 ExcCode 域并不更新。当处于 Debug 模式时，例外编码 0x1e 将被写入 Debug 寄存器的 DExcCode 域。
0x1f	-	保留

7.25 EPC 寄存器 (CP0 Register 14, Select 0)

EPC 寄存器是一个 64 位可读写寄存器，其包含例外处理完成后继续开始执行的指令的 PC。

在响应同步（精确）例外时，处理器向 EPC 寄存器中写入：

直接触发例外的指令的 PC。

当直接触发例外的指令位于分支延迟槽时，记录该指令前一条分支或跳转指令的 PC，同时 Cause.BD 置为 1。

在响应异步（非精确）例外时，处理器向 EPC 寄存器中写入例外处理完成后继续执行的指令的 PC。

当 Status 寄存器的 EXL 位为 1 时，发生例外时不更新 EPC 寄存器。

说明了 EPC 寄存器的格式；对 EPC 寄存器各域进行了描述。

图 7-27 EPC 寄存器格式

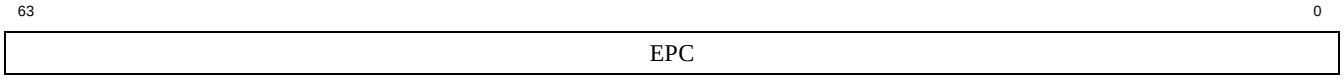


表 7-29 EPC 寄存器域描述

域名称	位	功能描述	读/写	复位值
EPC	63..0	例外程序计数器。	R/W	无

7.26 PRId 寄存器 (CP0 Register 15, Select 0)

PRId 寄存器是一个 32 位只读寄存器，该寄存器包含了标识 MIPS 处理器产商、处理器类型和实现版本的信息。

图 7-28 说明了 PRId 寄存器的格式；表 7-30 对 PRId 寄存器各域进行了描述。

图 7-28 PRId 寄存器格式

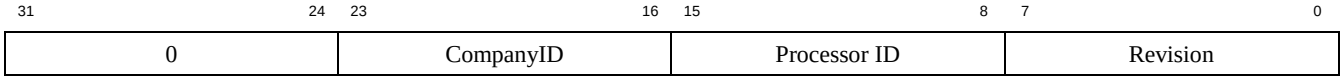


表 7-30 PRId 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..24	只读恒为 0。	0	0
CompanyID	23..16	公司 ID 号。 当 Diag.IDSEL 为 0 时，值为 0x14；当 Diag.IDSEL 为 1 时，值为 0x00。	R	0x14
ProcessorID	15..8	处理器类型号。为 0x63。	R	0x63
Revision	7..0	实现版本号。 当 Diag.IDSEL 为 0 时，值为 0x08；当 Diag.IDSEL 为 1 时，值为 0x05。	R	0x08

7.27 EBase 寄存器 (CP0 Register 15, Select 1)

EBase 寄存器是一个可读写寄存器，包含例外向量基地址和一个只读的 CPU 号。

图 7-29 说明了 EBase 寄存器的格式；表 7-31 对 EBase 寄存器各域进行了描述。

图 7-29 EBase 寄存器格式

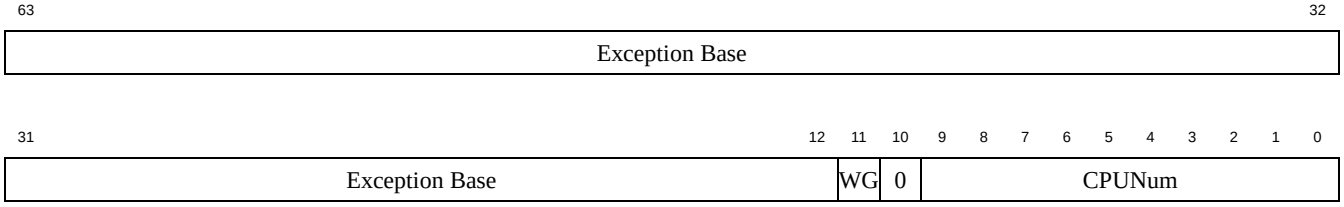


表 7-31 EBase 寄存器域描述

域名称	位	功能描述	读/写	复位值
Exception Base	63..12	当 Status.BEV=0 时，逻辑左移 12 位作为例外入口向量基址。 [63:30]位仅当 WG 位等于 1 时才可以写入，当 WG 位等于 0 时写 EBase 寄存器时[63:30]位保持不变。	R/W	0xffff.ffff.8000.0
WG	11	[63:30]位的写控制位。 1：ExceptionBase[63:30]可以写入； 0：ExceptionBase[63:30]写入时值保持不变。	R/W	0x0
0	10	只读恒为 0。	0	0
CPUNum	9..0	在多核系统中标识当前处理器的索引号。	R	

编程提示：

当使用向量中断模式时(Cause.IV=1)，如果 IntCtl.VS 配置为 0x10，则有第 7 号中断的例外向量偏移量会超过 0xfff 范围。此时软件需要保证 EBase 第 12 位置为 0，让出第 7 号中断向量偏移量的最高位。

7.28 Config 寄存器 (CP0 Register 16, Select 0)

Config 寄存器中定义了一些处理器的配置信息。Config 寄存器中除 K0 域可通过软件读写外，其它域均由硬件在复位期间予以初始化并保持只读状态不变。尽管 GS464E 对 K0 域硬件复位为 0b010（Uncached 属性），仍强烈建议软件在 reset 例外处理程序中对此域进行初始化。

图 7-30 说明了 Config 寄存器的格式；表 7-32 Config 寄存器域描述表 7-32 对 Config 寄存器各域进行了描述。

图 7-30 Config 寄存器格式

31	30 ... 19	18	17	16	15	14 13	12 ... 10	9 ... 6	4	3	2 ... 0
P	0	VEC	QM	CAM	BE	AT	AR	MT	0	VI	K0

表 7-32 Config 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	值为 1，表示存在 Config1 寄存器。	R	0x1
0	30..19	只读恒为 0。	0	0
VEC	18	值为 0，表示不支持龙芯自定义 256 位向量指令。	R	0x0
QM	17	值为 1，表示支持龙芯自定义 128 位访存指令，所涉及指令请参看 29 页表 2-27。	R	0x1
CAM	16	值为 1，表示包含硬件 CAM 部件，支持龙芯自定义 CAM 指令，所涉及指令请参看 37 页表 2-32。	R	0x1
BE	15	值为 0，表示采用小尾端寻址方式。	R	0x0
AT	14..13	值为 2，表示实现 MIPS64 体系结构，可以访问 64 位全地址空间。	R	0x2
AR	12..10	值为 1，兼容 MIPS64 release 5 规范，具体各细节实现的情况软件可通过读取其它配置寄存器或其它寄存器的配置域获得。	R	0x1
MT	9..7	值为 4，MMU 采用 VTLB 和 FTLB 双 TLB 形式。有关双 TLB MMU 信息请参看 4.3 节。	R	0x4
0	6..4	只读恒为 0。	0	0
VI	3	值为 0，指令 Cache 采用虚地址 Index 实地址 Tag 形式。有关 Cache 的组织形式请参看 5.1 节。	R	0x0
K0	2..0	Kseg0 段的 Cache 属性。GS464E 支持的 Cache 属性编码请参看 83 页表 7-6。	R/W	0x2

7.29 Config1 寄存器 (CP0 Register 16, Select 1)

Config1 寄存器用于提供处理器的一些配置信息。Config1 寄存器中的所有域均为只读。

图 7-31 说明了 Config1 寄存器的格式；表 7-33 表 7-32 Config 寄存器域描述对 Config1 寄存器各域进行了描述。

图 7-31 Config1 寄存器格式

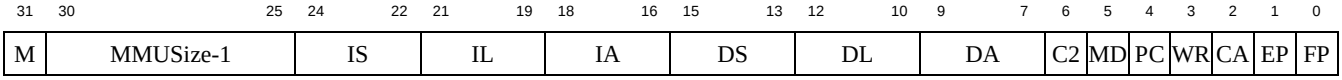


表 7-33 Config1 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	值为 1，表示存在 Config2 寄存器。	R	0x1
MMU Size-1	30..25	值为 63，其与 Config4 寄存器的 VTLBSizeExt 域拼接，形成 10 比特值： $Config4.VTLBSizeExt_{3..0} \| Config1.MMUSize\text{-}1_{5..0}$； 拼接后的值为 63，表示 VTLB 为 64 项。	R	0x3f
IS	24..22	值为 2，表示 I-Cache 每一路包含 256 行。	R	0x2
IL	21..19	值为 5，表示 I-Cache 每行长度为 64 字节。	R	0x5
IA	18..16	值为 3，表示 I-Cache 包含 4 路。	R	0x3
DS	15..13	值为 2，表示 D-Cache 每一路包含 256 行。	R	0x2
DL	12..10	值为 5，表示 D-Cache 每行长度为 64 字节。	R	0x5
DA	9..7	值为 3，表示 D-Cache 包含 4 路。	R	0x3
C2	6	值为 1，表示包含有协处理器 2（COP2）。	R	0x1
MD	5	值为 0，表示未实现 MDMX ASE 指令集。	R	0x0
PC	4	值为 1，表示实现了性能计数器。	R	0x1
WR	3	值为 0，表示未实现 Watch 寄存器。	R	0x0
CA	2	值为 0，表示未实现 MIPS16e 指令集。	R	0x0
EP	1	值为 1，表示实现了 EJTAG。	R	0x1
FP	0	值为 1，表示实现了浮点协处理器。	R	0x1

7.30 Config2 寄存器 (CP0 Register 16, Select 2)

Config2 寄存器用于提供处理器的一些配置信息。Config2 寄存器中的所有域均为只读。

图 7-32 说明了 Config2 寄存器的格式；表 7-34 表 7-32 Config 寄存器域描述对 Config2 寄存器各域进行了描述。

图 7-32 Config2 寄存器格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	0			TS				TL				TA				0				SS				SL				SA			

表 7-34 Config2 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	值为 1，表示存在 Config3 寄存器。	R	0x1
0	30..28	只读恒为 0。	0	0
TS	27..24	值为 2，表示 V-Cache 每一路包含 256 行。	R	0x2
TL	23..20	值为 5，表示 V-Cache 每行长度为 64 字节。	R	0x5
TA	19..16	值为 15，表示 V-Cache 包含 16 路。	R	0xf
0	15..12	只读恒为 0。	0	0
SS	11..8	值为 4，表示 S-Cache 每一路包含 1024 行。	R	0x4
SL	7..4	值为 5，表示 S-Cache 每行长度为 64 字节。	R	0x5
SA	3..0	值为 15，表示 S-Cache 包含 16 路。	R	0xf

7.31 Config3 寄存器 (CP0 Register 16, Select 3)

Config3 寄存器用于提供处理器的一些配置信息。Config3 寄存器中的所有域均为只读。

图 7-33 说明了 Config3 寄存器的格式；表 7-35 表 7-32 Config 寄存器域描述对 Config3 寄存器各域进行了描述。

图 7-33 Config3 寄存器格式

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	BPG	CMGCR	MSAP	BP	BI	SC	PW	VZ	R		R			MCU	R	ISA		ULRI	RXI	DSP2P	DSP	CTXTC	ITL	LPA	VEIC	VInt	SP	CDMM	MT	SM	TL

表 7-35 Config3 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	值为 1，表示存在 Config4 寄存器。	R	0x1
BPG	30	值为 1，表示 TLB 支持超过 256MB 的大页，相应的 PageMask 寄存器为 64 位。	R	0x1
CMGCR	29	值为 0，表示未实现 Coherency Manager memory-mapped Global Configuration Register Space。 GS464E 支持多核，采用自定义的多核一致性管理机制，不采用 MIPS 的多核一致性管理架构，况且 MIPS 并未对其多核一致性管理出具规范。	R	0x0
MSAP	28	值为 0，表示未实现 MIPS 向量模块(SIMD Module)。	R	0x0
BP	27	值为 0，表示未实现 BadInstrP 寄存器。	R	0x0
BI	26	值为 0，表示未实现 BadInstr 寄存器。	R	0x0
SC	25	值为 0，表示未实现段控制(Segment Control)功能。相应地，用于段控制的 SegCtl0、SegCtl1 和 SegCtl2 寄存器未实现。	R	0x0
PW	24	值为 0，表示未实现硬件 TLB 重填机制(Hardware Page Table Walk)。 但是，为了配合龙芯自定义 LWTR、LDTR 指令的执行，MIPS 规范为硬件 TLB 重填机制所定义的 PWBase、PWField 和 PWSize 寄存器仍在 GS464E 中予以定义。并且，PWBase、PWField 和 PWSize 寄存器实现为四组，分别对应与 SpaceID=0, 1, 2, 3 的四个不同地址空间。	R	0x0
VZ	23	值为 1 。	R	0x1
R	22..21	因未实现 MIPS MCU ASE，该域无意义。	R	0x0
R	20..18	因未实现 micrMIPS64 指令集，该域无意义。	R	0x0
MCU	17	值为 0，表示未实现 MIPS MCU ASE。	R	0x0
R	16	因没有同时实现 MIPS64 和 microMIPS64，该域无意义。	R	0x0
ISA	15..14	值为 0，表示只实现了 MIPS64 指令集，没有实现 microMIPS64 指令集。	R	0x0
ULRI	13	值为 1，表示实现了 UserLocal 寄存器。	R	0x1
RXI	12	值为 1，表示在 PageGrain 寄存器中实现了 RIE 和 XIE 位。	R	0x1
DSP2P	11	值为 1，表示实现了 MIPS DSP 模块版本 2。	R	0x1
DSP	10	值为 1，表示实现了 MIPS DSP 模块。	R	0x1

域名称	位	功能描述	读/写	复位值
CTXTC	9	值为 0，表示未实现 ContextConfig 和 XcontextConfig 寄存器。 因不建议运行在 GS464E 上的操作系统采用单级页表结果，因此并不建议在 TLB 例外处理时将 Context 和 Xcontext 寄存器的内容作为页表项的指针。相应地，ContextConfig 和 XcontextConfig 寄存器也不予实现。	R	0x0
ITL	8	值为 0，表示未实现 MIPS IFlowTrace 调试功能。	R	0x0
LPA	7	值为 1，表示支持大物理地址范围。相应地，实现了 PageGrain 寄存器。	R	0x1
VEIC	6	值为 0，表示中断机制中未实现外部中断控制器(EIC)模式。	R	0x0
VInt	5	值为 1，表示中断机制中实现了向量中断(Vectored Interrupts)模式。	R	0x1
SP	4	值为 0，表示不支持 1KB 大小的小页。	R	0x0
CDMM	3	值为 0，表示未实现通用外设内存映射(Common Device Memory Map)机制。	R	0x0
MT	2	值为 0，表示未实现 MIPS 多线程模块(MT Module)。	R	0x0
SM	1	值为 0，表示未实现 SmartMIPS^{TM} ASE。	R	0x0
TL	0	值为 0，表示未实现 Trace 逻辑。	R	0x0

7.32 Config4 寄存器 (CP0 Register 16, Select 4)

Config4 寄存器用于提供处理器的一些配置信息，并用于控制 FTLB 的页大小。

图 7-34 说明了 Config4 寄存器的格式；表 7-36 对 Config4 寄存器各域进行了描述。

图 7-34 Config4 寄存器格式

31	30 29	28	27 24	23 16	15 14	13	12 8	7 4	3 0
M	IE	AE	VTLBSizeExt	KScrExist	MMU ExtDef	0	FTLBPageSize	FTLBWays	FTLBSets

表 7-36 Config4 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	值为 1，表示存在 Config5 寄存器。	R	0x1
IE[1]	30..29	值为 0，表示未实现 TLBINV 和 TLBINVF 指令，未实现 $EntryHi_{EHINV}$ 域的功能。	R	0x0
AE	28	值为 0，表示 $EntryHi_{ASID}$ 域宽度仍为 8 位。	R	0x0
VTLB-SizeExt	27..24	值为 0，其与 Config1 寄存器的 MMUSize-1 域拼接，形成 10 比特值：$Config4.VTLBSizeExt_{3..0} \| Config1.MMUSize\text{-}1_{5..0}$；拼接后的值为 63，表示 VTLB 为 64 项。	R	0x0
KScrExist	23..16	值为 0b11111100，表示 KScratch1~6 寄存器(CP0 Register 31, Selct 2~7)可以被处于核心态的软件访问。	R	0xfc
MMU-ExtDef	15..14	值为 3，用于解释 Config4 寄存器的格式。其中 Config4[3:0]为 FTLBSets，Config4[7:4]为 FTLBWays，Config4[10:8]为 FTLBPageSize，Config4[27:24]为 VTLBSizeExt。	R	0x3
0	13	只读恒为 0。	0	0

[1] Config4.IE 域值设置为 0 是为了保持与 MIPS 规范的兼容性。实际上，GS464E 处理器核实现了 TLBINVF 指令（以 Config4.IE=3 方式执行），即执行一条 TLBINVF 指令时硬件会将整个 TLB 表项都无效。此外，GS464E 处理器核中也实现 TLBINV 指令，但是其执行效果与 MIPS 规范定义不同，而是等效于 TLBINVF 指令。

GS464E 处理器核也实现了 EntryHi.EHINV 域的功能：当 EntryHi.EHINV 位置为 1 时，执行 TLBWI 指令会将对应的项无效；当执行 TLBR 指令时，会将所读取的 TLB 表项的 VPN2 无效位的值更新至 EntryHi.EHINV 位。

<table>
<tr><th>域名称</th><th>位</th><th>功能描述</th><th>读/写</th><th>复位值</th></tr>
<tr><td>FTLB-PageSize</td><td>12..8</td><td>表示 FTLB 所使用的页大小。GS464E 中 FTLB 所支持的页大小及其编码值如下：
<table>
<tr><th>页大小</th><th>编码值</th></tr>
<tr><td>4KB</td><td>1</td></tr>
<tr><td>16KB</td><td>2</td></tr>
<tr><td>64KB</td><td>3</td></tr>
<tr><td>256KB</td><td>4</td></tr>
<tr><td>1MB</td><td>5</td></tr>
<tr><td>4MB</td><td>6</td></tr>
<tr><td>16MB</td><td>7</td></tr>
<tr><td>64MB</td><td>8</td></tr>
<tr><td>256MB</td><td>9</td></tr>
<tr><td>1GB</td><td>10</td></tr>
</table>
软件若向该域写入的值不包含在上面的表格中，则该域的值保持不变。
软件修改该域之前必须清空 FTLB，否则处理器行为不确定。</td><td>R/W</td><td>0x1</td></tr>
<tr><td>FTLB-Ways</td><td>7..4</td><td>值为 6，表示 FTLB 包含 8 路</td><td>R</td><td>0x6</td></tr>
<tr><td>FTLB-Sets</td><td>3..0</td><td>值为 7，表示 FTLB 每一路包含 128 项。</td><td>R</td><td>0x7</td></tr>
</table>

7.33 Config5 寄存器 (CP0 Register 16, Select 5)

Config5 寄存器用于提供处理器的一些配置信息。

图 7-35 说明了 Config5 寄存器的格式；表 7-37 表 7-32 Config 寄存器域描述对 Config5 寄存器各域进行了描述。

图 7-35 Config5 寄存器格式

31	30	29	28	27	26 ... 1	0
0	R	R	R	R	0	NFExists

表 7-37 Config5 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31	只读恒为 0。	0	0
R	30	因为未实现段控制(Segmentation Control)模式，所以该域无意义。	R	0x0
R	29	因为未实现段控制(Segmentation Control)模式，所以该域无意义。	R	0x0
R	28	因为未实现段控制(Segmentation Control)模式，所以该域无意义。	R	0x0
R	27	因为未实现 MIPS 向量模块(SIMD Module)，所以该域无意义。	R	0x0
0	26..1	只读恒为 0。	0	0
NFExists	0	值为 1 。	R	0x1

7.34 GSConfig 寄存器 (CP0 Register 16, Select 6)

GSConfig 寄存器用于对处理器核部分微结构相关的功能进行动态配置。软件可以根据程序的具体特性通过开启或关闭相应的功能，以获得最佳性能。。

图 7-36 说明了 GSConfig 寄存器的格式；表 7-38 对 GSConfig 寄存器各域进行了描述。

图 7-36 GSConfig 寄存器格式

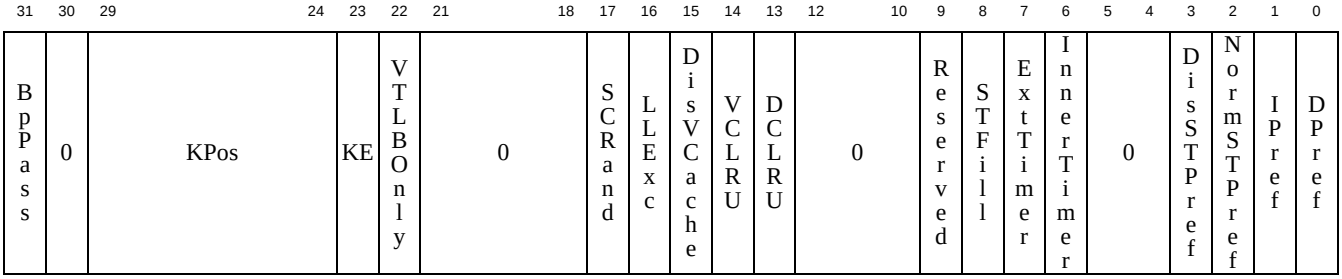


表 7-38 GSConfig 寄存器域描述

域名称	位	功能描述	读/写	复位值
BpPass	31	EJTAG 指令和数据断点响应机制控制。 0：任何满足指令断点和数据断点的条件都会触发例外。 1：EJTAG 指令断点和数据断点例外在例外处理返回重新执行时，自动忽略第一次满足指令断点和数据断点的条件。 当该位置为 0 时，处理器将按照 MIPS EJTAG 规范要求的方式处理指令断点和数据断点，这也就意味着 EJTAG 例外处理程序必须在处理过程中修改指令断点或数据断点的判断条件，以确保当前触发断点例外的指令再 DERET 返回后不会再次触发例外而陷入死循环，同时还要确保程序能够在新的判断条件下及时停止，使得例外处理程序来得及在被观察程序再次执行到待观察断点之前将断点条件设置正确。	R/W	0x0
0	30	只读恒为 0。	0	0
KPos	29..24	指示 K 位在页表项(PTE)中位于哪一比特。	R/W	0x3d
KE	23	TLB 页表内核执行保护功能使能位。 0：关闭该功能。EntryLo0 和 EntryLo1 寄存器的 K 位将禁止写入，强制为 0； 1：开启该功能。EntryLo0 和 EntryLo1 寄存器的 K 位可正常使用。	R/W	0x0
VTLB-Only	22	置为 1 时，处理器将只操作双 TLB 中的 VTLB，等效于 MIPS 传统的单个 CAM 型 TLB，从而保证原有在 GS464 上运行的操作系统内核等底层软件不修改 MMU 部分仍可以在 GS464E 上运行。如果需要使用双 TLB 的 MMU，需将该位置为 0。 软件在修改这一位状态前，应清空 TLB，否则处理器行为将不可知。	R/W	0x1
0	21..18	只读恒为 0。	0	0
SCRand	17	置为 1 时，芯片中的 Cache 一致性维护部件将对该处理器核中 SC/SCD 指令发起的 Cache 访问请求的返回延迟增加 64~128 个时钟周期随机延迟。置为 0 时关闭该功能。	R/W	0x1

域名称	位	功能描述	读/写	复位值
LLExc	16	为 1 时，LL/LLD 指令将发起的 Cache 访问请求必须返回处于独占(Exclusive)状态的 Cache 块； 为 0 时，LL/LLD 指令发起的 Cache 访问请求允许返回处于共享(Shared)状态的 Cache 块。	R/W	0x1
Dis-VCache	15	0：使用 VCache； 1：禁用 VCache。 在从使用 VCache 向禁用 VCache 的过程中，软件需要确保 VCache 中没有任何有效内容。	R/W	0x0
VCLRU	14	配置 VCache 替换算法。1：LRU 替换算法；0：伪随机替换算法。	R/W	0x1
DCLRU	13	配置 DCache 替换算法。1：LRU 替换算法；0：伪随机替换算法。	R/W	0x1
0	12..10	只读恒为 0。	0	0
Reserved	9	该位复位后必须对其写 1，且不再更改为 0。	R/W1	0x0
STFill	8	处理器 store 操作自动写合并功能使能位。1：开启；0：关闭。 软件在修改这一状态前，需使用 SYNC 指令确保处理器中没有尚未执行完毕的访存操作。	R/W	0x1
Ext-Timer	7	置为 1 时，Cause.TI 所记录的时钟中断可以来自于属于本处理器核的外部定时器； 置为 0 时，Cause.TI 所记录的时钟中断不来自于属于本处理器核的外部定时器。 需要指出的是，外部定时器的增长频率不受处理器核变频的影响。 允许软件将 GSConfig 寄存器的 ExtTimer 位和 InnerTimer 位同时置为 1，但需要软件能够正确处理该情况，通常不建议如此配置。	R/W	0x0
Inner-Timer	6	置为 1 时，Cause.TI 所记录的时钟中断可以来自于处理器核内部以 Count/Compare 寄存器实现的定时器； 置为 0 时，Cause.TI 所记录的时钟中断不来自于处理器核内部以 Count/Compare 寄存器实现的定时器。 需要指出的是，以 Count/Compare 寄存器实现的定时器的增长频率会随着处理器核频率的变化而等比例变化。 允许软件将 GSConfig 寄存器的 ExtTimer 位和 InnerTimer 位同时置为 1，但需要软件能够正确处理该情况，通常不建议如此配置。	R/W	0x1
0	5..4	只读恒为 0。	0	0
Dis-STPref	3	置为 1 时，处理器对于数据访存操作中的 store 操作不进行硬件自动预取；置为 0 时，处理器将对 store 操作也进行硬件自动预取，store 预取的性能可进一步通过 $GSConfig_{NormSTPref}$ 进行配置，请见下一项的描述。 该域仅在 $GSConfig_{DPref}$=1 时才有意义，$GSConfig_{DPref}$=0 时修改该域不会引起性能变化。 软件在修改这一状态前，需使用 SYNC 指令确保处理器中没有尚未执行完毕的访存操作。	R/W	0x0

域名称	位	功能描述	读/写	复位值
Norm-STPref	2	置为 1 时，处理器在对 store 操作进行硬件自动预取时，所采用的流控算法与 load 操作等同；置为 0 时，处理器对 store 操作硬件自动预取时，除了采用与 load 操作等同的流控机制外，还将同时监测过去一段时间内 store 操作自动写合并成功(收集满一个 Cache 块)的次数，如果成功次数超过一定阈值，则暂时停止 store 操作的硬件自动预取。 该域仅在 $GSConfig_{DPref}$=1 时才有意义，$GSConfig_{DPref}$=0 时修改该域不会引起性能变化。 软件在修改这一状态前，需使用 SYNC 指令确保处理器中没有尚未执行完毕的访存操作。	R/W	0x0
IPref	1	置为 1 时，处理器对于取指操作进行硬件自动预取；否则，关闭该功能。 软件在修改这一状态前，需使用 SYNC 指令确保处理器中没有尚未执行完毕的访存操作。	R/W	0x1
DPref	0	置为 1 时，处理器对于数据访存操作进行硬件自动预取；否则，关闭该功能。 软件在修改这一状态前，需使用 SYNC 指令确保处理器中没有尚未执行完毕的访存操作。	R/W	0x1

7.35 LLAddr 寄存器 (CP0 Register 17, Select 0)

LLAddr 寄存器是一个 64 位只读寄存器，用于保存最近发生的 Load Linked 指令的物理地址。当例外返回时，LLAddr 寄存器被清零。

图 7-37 说明了 LLAddr 寄存器的格式；表 7-39 对 LLAddr 寄存器各域进行了描述。

图 7-37 LLAddr 寄存器格式

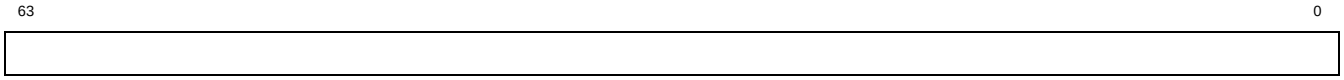


表 7-39 LLAddr 寄存器域描述

域名称	位	功能描述	读/写	复位值
PAddr	63..0	最近发生的 Load Linked 指令的物理地址	R	无

7.36 XContext 寄存器 (CP0 Register 20, Select 0)

XContext 寄存器是一个可读写寄存器，其中包含由操作系统软件所填入的一些页表基地址高位信息和发生 TLB 例外的出错虚地址的某些位。按照 MIPS 架构最初的设计意图，XContext 寄存器中被拼接在一起的信息可形成指向页表中某一项的指针，用于 TLB 例外发生时访问该页表项。XContext 寄存器中内容不做任何处理时可访问的页表是一个单级页表结构，页大小为 4K 字节，每个页表项为 16 字节，包含虚地址连续的一个偶数页表项和一个奇数页表项。当页表未采用这种结构时，软件需要对 XContext 寄存器的内容进行适当地移位、拼接处理。对于一个采用多级页表的操作系统来说，XContext 寄存器仅能用在加速最后一级页表访问的地址生成。

XContext 寄存器主要被用在 XTLB Refill 例外处理程序中。但是当 TLB Refill、TLB Invalid 和 TLB Mod 例外发生时，XContext 寄存器中的 BadVPN2 域和 R 域也会被更新，因此软件也可以在相应的例外处理程序中使用 XContext 寄存器。

XContext 寄存器中的 BadVPN2 域和 R 域复制了 BadVAddr 寄存器中的部分信息，但并不意味着这部分是完全等价的。当 Address Error 例外发生时，BadVaddr 寄存器会被硬件更新，但 Context 寄存器的 BadVPN2 域和 R 域不会被硬件更新。

图 7-38 说明了 XContext 寄存器的格式；表 7-40 对 XContext 寄存器各域进行了描述。

图 7-38 XContext 寄存器格式

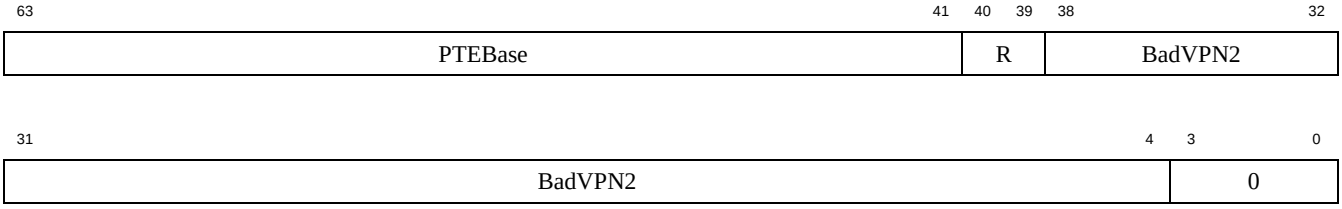


表 7-40 XContext 寄存器域描述

域名称	位	功能描述	读/写	复位值
PTEBase	63..41	页表基地址高位。由操作系统软件根据当前页表情况进行配置。	R/W	无
R	40..39	当发生 TLB 例外时，存放出错虚地址的 63..62 位。 0b00：普通用户区域； 0b01：超级用户区域； 0b10：保留； 0b11：核心区域。	R	无
BadVPN2	38..4	当发生 TLB 例外时，存放出错虚地址的 47..13 位。	R	无
0	3..0	只读恒为 0。	0	0

7.37 Diag 寄存器 (CP0 Register 22, Select 0)

Diag 寄存器是龙芯自定义的寄存器，用于控制虚拟机执行以及一些内部队列和特殊操作。

图 7-39 说明了 Diag 寄存器的格式；表 7-41 表 7-32 Config 寄存器域描述对 Diag 寄存器各域进行了描述。

图 7-39 Diag 寄存器格式

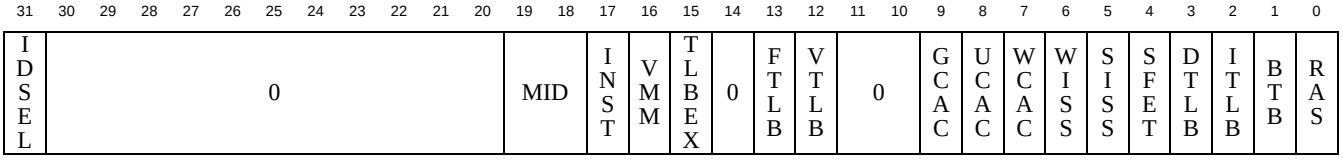


表 7-41 Diag 寄存器域描述

域名称	位	功能描述	读/写	复位值
IDSEL	31	控制 PRId 寄存器的读出值。该位为 0 时，PRId 读出值为 0x00146308；该位为 1 时，PRId 读出值为 0x00006305。后者与 LS3A1000 芯片处理器的 PRId 完全一致。为实现 GS464 上运行的操作系统内核及以上软件在 GS464E 上兼容运行，可以在 PMON 中将该位修改为 1。 需要注意的是：每次硬重启之后，IDSEL 位只能被修改一次。建议这次修改在 PMON 中进行。	R/W	0x0
0	30..20	只读恒为 0。	0	0
MID	19..18	当发生 TLB Refill、TLB Invalid 和 TLB Mod 例外时，存放触发例外的指令所访问的地址空间的 MID。 当使用 TLBWR、TLBWI 指令填写 TLB 时，该域的内容会被写入 TLB 表项中，参与后续的虚实地址映射。 当使用 TLBP、TLBR 指令读取 TLB 时，读出表项中存放的 MID 内容存入此域。在软件读写 PWBase、PWField、PWSize 寄存器时，该域用于就指示访问哪一个空间中的页表配置信息。 MID 域仅在 Diag.VMM=1 时可写，Diag.VMM=0 时无论写何值该域都将被置为 0。	R/W	0x0
INST	17	PC 的虚实地址映射是否使用 SpaceID 信息的标志，0：不使用；1：使用。 该域仅在 Diag 寄存器的 VMM 位置为 1 的情况下才有效，否则该域虽然可以正常读写，但不参与其它任何操作。	R/W	0x0
VMM	16		R/W	0x0
TLBEX	15	置为 1 时，在根-核心模式下执行 TLBR、TLBP、TLBWI、TLBWR、TLBINVF 指令将触发 VMMU 例外。	R/W	0x0
0	14	只读恒为 0。	0	0

域名称	位	功能描述	读/写	复位值
FTLB	13	对该位写入 1 时清空 FTLB。请注意处理器关注的是该位被写入 1 的行为，清空 FTLB 与该位值是否为 1 无关。该位读出值恒为 0。	R0/W	0
VTLB	12	对该位写入 1 时清空 VTLB。请注意处理器关注的是该位被写入 1 的行为，清空 FTLB 与该位值是否为 1 无关。该位读出值恒为 0。	R0/W	0
0	11..10	只读恒为 0。	0	0
GCAC	9		R/W	0x0
UCAC	8	置 1 时，在 Root-User 模式下执行 CACHE0、CACHE1、CACHE3、CACHE15、CACHE21、CACHE23 指令将不触发协处理器例外(CpU)。	R/W	0x0
WCAC	7	置 1 时取消 wait cache 操作的限制。	R/W	0x0
WISS	6	置 1 时取消 wait issue 操作的限制。	R/W	0x0
SISS	5	置 1 时取消 stall issue 操作的限制。	R/W	0x0
SFET	4	置 1 时取消 stall fetch 操作的限制。	R/W	0x0
ITLB	3	对该位写入 1 清空 ITLB。请注意处理器关注的是该位被写入 1 的行为，清空 FTLB 与该位值是否为 1 无关。该位读出值恒为 0。	R0/W	0
ITLB	2	对该位写入 1 清空 ITLB。请注意处理器关注的是该位被写入 1 的行为，清空 FTLB 与该位值是否为 1 无关。该位读出值恒为 0。	R0/W	0
BTB	1	对该位写入 1 清空分支预测中的 BRBTB 和 BTAC。请注意处理器关注的是该位被写入 1 的行为，清空 FTLB 与该位值是否为 1 无关。该位读出值恒为 0。	R0/W	0
RAS	0	置 1 时禁用 RAS 对 jr31 进行分支预测。	R/W	0x0

7.38 GSCause 寄存器 (CP0 Register 22, Select 1)

GSCause 寄存器是一个只读寄存器，其包含了例外发生时与龙芯扩展部分相关的信息，如触发例外指令是否包含前缀，龙芯扩展例外的具体原因。

图 7-40 说明了 GSCause 寄存器的格式；表 7-42 对 GSCause 寄存器各域进行了描述。

图 7-40 GSCause 寄存器格式

31 … 12	11 … 8	7	6 … 2	1	0
0	TLBInst	0	GSExcCode	0	P

表 7-42 GSCause 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..12	只读恒为 0。	0	0
TLBInst	11..8	当在客态下执行 TLB 指令而触发 PSI 例外时，记录具体的 TLB 指令类型。 TLBInst[0]为 1 表示是 TLBP 指令； TLBInst[1]为 1 表示是 TLBR 指令； TLBInst[2]为 1 表示是 TLBWR 指令； TLBInst[3]为 1 表示是 TLBWI、TLBINV 或 TLBINVF 指令。	R	0x0
0	7	只读恒为 0。	0	0
GS-ExcCode	6..2	龙芯扩展例外编码。	R	0x0
0	1	只读恒为 0。	0	0
P	0	1：触发例外的指令携带前缀，指令长 64 比特； 0：触发例外的指令不携带前缀，指令长 32 比特。	R	0x0

表 7-43 GSExcCode 编码及其对应例外类型

ExcCode	助记符	描述
0x00	IS	浮点栈例外
0x01	VMMU	虚拟机内存管理单元例外
0x02	VMTLBL	虚拟机地址空间 TLB 例外（读数据或取指令）
0x03	VMTLBS	虚拟机地址空间 TLB 例外（写数据）
0x04	VMMod	虚拟机地址空间 TLB 修改修改
0x05	VMTLBRI	虚拟机地址空间不可读例外
0x06	VMTLBXI	虚拟机地址空间不可执行例外

7.39 VPID 寄存器 (CP0 Register 22, Select 2)

VPID 寄存器是龙芯自定义的寄存器，用于控制虚拟机执行时的 VPID 信息。

图 7-39 说明了 Diag 寄存器的格式；表 7-41 表 7-32 Config 寄存器域描述对 Diag 寄存器各域进行了描述。

图 7-41 VPID 寄存器格式

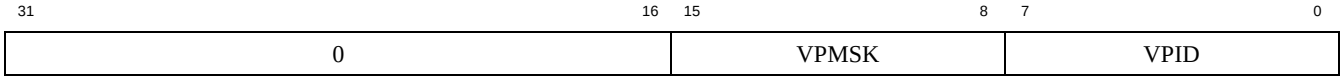


表 7-44 VPID 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..16	只读恒为 0。	0	0
VPMSK	15..8	虚拟机号掩码，用于控制 Diag 寄存器 VPID 域参与虚实地址映射的位数；该掩码同时也控制虚地址高位参与虚实地址映射的实际位数，即可以认为对此时处理器的 SEGBITS 值进行了改变。 若 VPMSK 配置了非法值，处理器行为将不可预测。 该域仅在 Diag 寄存器的 VMM 位置为 1 的情况下才有效，否则该域虽然可以正常读写，但不参与其它任何操作。	R/W	0x0
VPID	7..0	虚拟机号，其有效位数受到 VPMSK 域控制，请见上面一项中的说明。 该域仅在 Diag 寄存器的 VMM 位置为 1 的情况下才有效，否则该域虽然可以正常读写，但不参与其它任何操作。	R/W	0x0

VPMSK 域嵌套表：

VPMSK 合法值	VPID 有效范围	等价 SEGBITS	虚地址有效位范围
0x00	0	48	{Vaddr[63:62], VAddr[47:0]}
0x80	VPID[7]	47	{Vaddr[63:62], VAddr[46:0]}
0xc0	VPID[7:6]	46	{Vaddr[63:62], VAddr[45:0]}
0xe0	VPID[7:5]	45	{Vaddr[63:62], VAddr[44:0]}
0xf0	VPID[7:4]	44	{Vaddr[63:62], VAddr[43:0]}
0xf8	VPID[7:3]	43	{Vaddr[63:62], VAddr[42:0]}
0xfc	VPID[7:2]	42	{Vaddr[63:62], VAddr[47:0]}
0xfe	VPID[7:1]	41	{Vaddr[63:62], VAddr[47:0]}
0xff	VPID[7:0]	40	{Vaddr[63:62], VAddr[47:0]}

7.40 Debug 寄存器 (CP0 Register 23, Select 0)

Debug 寄存器是一个 32 位可读写寄存器。该寄存器包含了最近发生的 EJTAG 调试例外和调试模式下发生的例外的原因，并用于控制单端调试中断。同时该寄存器还对调试模式下的资源进行控制，指明处理器内部的相关状态。关于 Debug 寄存器的描述请参考 MIPS EJTAG 规范。

7.41 DEPC 寄存器 (CP0 Register 24, Select 0)

DEPC 寄存器是一个 64 位可读写寄存器，其包含 EJTAG 调试例外或调试模式下的例外处理完成后继续开始执行的指令的 PC。

在响应例外时，处理器向 DEPC 寄存器中写入：

直接触发例外的指令的 PC。

当直接触发例外的指令位于分支延迟槽时，记录该指令前一条分支或跳转指令的 PC，同时 Cause.BD 和 Debug.DBD 置为 1。

图 7-42 说明了 DEPC 寄存器的格式；表 7-45 对 DEPC 寄存器各域进行了描述。

图 7-42 DEPC 寄存器格式

表 7-45 DEPC 寄存器域描述

域名称	位	功能描述	读/写	复位值
DEPC	63..0	EJTAG 调试例外处理完成后继续开始执行的指令的 PC。	R/W	无

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

PerfCnt 寄存器是一组用于处理器性能事件统计的 CP0 寄存器。每一组性能计数器由一对 Select 号偶-奇相邻的 CP0 寄存器构成，即 Select0~1 构成 PerCnt0、Select2~3 构成 PerCnt1、Select4~5 构成 PerCnt2、Select6~7 构成 PerCnt3。其中每组性能计数器中的偶数号寄存器是控制寄存器（PerfCnt Control Reg），用于定义事件类别、控制计数条件；其中的奇数号寄存器是记录次数值的值寄存器（PerfCnt Counter Reg）。如表 7-46 所示。

表 7-46 PerfCnt 寄存器 Select 分配

性能计数器	Select 值	PerfCnt 寄存器
0	Select 0	PerfCnt Control Register 0
	Select 1	PerfCnt Counter Register 0
1	Select 2	PerfCnt Control Register 1
	Select 3	PerfCnt Counter Register 1
2	Select 4	PerfCnt Control Register 2
	Select 5	PerfCnt Counter Register 2
3	Select 6	PerfCnt Control Register 3
	Select 7	PerfCnt Counter Register 3

GS464E 实现了 PerfCnt0~PerfCnt3 共四组性能计数器，寄存器格式的定义也沿用了 MIPS 规范中给出的范例；但是与 MIPS 规范有所区别的是，GS464E 中的 PerfCnt 寄存器实际上是作为处理器核内部性能计数器的读写接口而非实际的计数器。简言之，软件先通过配置 PerfCtrl 寄存器中的事件信息，将待操作事件与处理器核内部某个具体的性能计数器建立起对应关系，随后软件对于该组 PerfCnt 寄存器的读写实际上作用于被指定的具体的性能计数器。这种设计是为了突破 MIPS 架构下单个处理器可以同时计数的性能事件无法超过 4 个的限制。

图 7-43 说明了 PerfCnt Control 寄存器的格式；表 7-47 对 PerfCnt Control 寄存器各域进行了描述。

图 7-44 说明了 PerfCnt Counter 寄存器的格式；表 7-48 对 PerfCnt Counter 寄存器各域进行了描述。

图 7-43 PerfCnt Control 寄存器格式

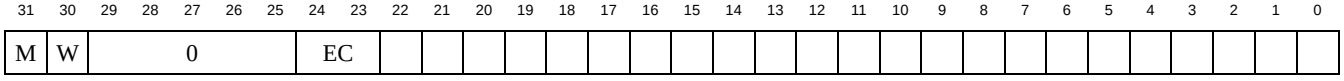


表 7-47 PerfCnt Control 寄存器域描述

域名称	位	功能描述	读/写	复位值
M	31	置为 1 表示实现了下一组性能计数器，否则未实现。	R	0x1/0x0[1]
W	30	恒为 1，表示 PerfCnt Counter 寄存器的位宽是 64 位。	R	0x1
0	29..25	只读恒为 0。	0	0

[1] 对于 Control 0 ~ Control 2，该位复位值为 1；对于 Control3，该位复位值为 0。

域名称	位	功能描述	读/写	复位值
EC	24..23	事件类别。 0：根态事件，指 GuestCtl0.GM=0 时发生的事件。 1：根态介入事件，指 GuestCtl0.GM=1 and !(Root.Status.EXL=0 and Root.Status.ERL=0 and Root.Debug.DM=0)时发生的事件。	R/W	0x0
0	22.15	只读恒为 0。	0	0
Event	14..5	事件号。	R/W	0x0
IE	4	性能计数器溢出中断使能。 0：禁止该性能计数器触发溢出中断。 1：允许该性能计数器触发溢出中断。	R/W	0x0
U	3	用户模式下事件记录使能位。0：禁止记录；1：允许记录。	R/W	0x0
S	2	监管模式下事件记录使能位。0：禁止记录；1：允许记录。	R/W	0x0
K	1	核心模式下事件记录使能位。0：禁止记录；1：允许记录。	R/W	0x0
EXL	0	Status.EXL=1 且 Status.ERL=0 情况下事件记录使能位。0：禁止记录；1：允许记录。	R/W	0x0

图 7-44 PerfCnt Counter 寄存器格式

表 7-48 PerfCnt Counter 寄存器域描述

域名称	位	功能描述	读/写	复位值
Event Count	63..0	性能事件计数器。每当同组中 PerfCnt Control 寄存器定义事件触发时，该计数器值加 1。当该计数器的最高位为 1 时，将 Cause 寄存器的 PCI 位置为 1。 尽管 PerfCnt Control 的 W 位始终定义为 1，64 位宽 Event Count 域每一位均可读写，但是 GS464E 实际的计数范围不超过 48 位。所以读取 Event Count 时，是实际计数器 48 位的数值符号扩展为 64 位的结果；重置计时器值时，只有 Event Count 的低 48 位才能被写入。	R/W	0x0

7.43 ErrCtl 寄存器 (CP0 Register 26, Select 0)

ErrCtl 寄存器是一个软件可读写的寄存器，其作为 Index Load Tag 和 Index Store Tag 类 CACHE 指令与各级 Cache 的 Tag 部分数据的 Parity/ECC 校验值的交互接口，同时也作为 Index Load Data 和 Index Store Data 类 CACHE 指令与各级 Cache 的 Data 部分数据的 Parity/ECC 校验值的交互接口。

图 7-45 说明了 ErrCtl 寄存器的格式；表 7-49 表 7-32 Config 寄存器域描述对 ErrCtl 寄存器各域进行了描述。

图 7-45 ErrCtl 寄存器格式

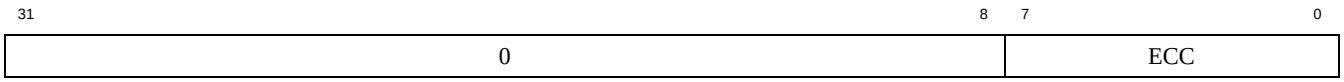


表 7-49 ErrCtl 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..8	只读恒为 0。	0	0
ECC	7..0	待写入或读出的 Tag 或 Data 的 ECC 校验值内容。	R/W	无

7.44 CacheErr 寄存器 (CP0 Register 27, Select 0)

CacheErr 寄存器记录 I-Cache 的 Parity 校验和 D-Cache 的 ECC 校验出错时的诊断信息。GS464E 对于 V-Cache 和 S-Cache 也进行“纠 1 检 2”的 ECC 校验，但是不将校验出错信息存入 CacheErr 寄存器。

CacheErr 寄存器在记录 I-Cache 校验错信息和 D-Cache 校验错信息时的格式存在差异。图 7-46 说明了 CacheErr 寄存器用于 I-Cache 校验错信息时的格式；表 7-50 表 7-32 Config 寄存器域描述对该情况下寄存器各域进行了描述。

图 7-46 CacheErr 寄存器用于 I-Cache 校验错信息时的格式

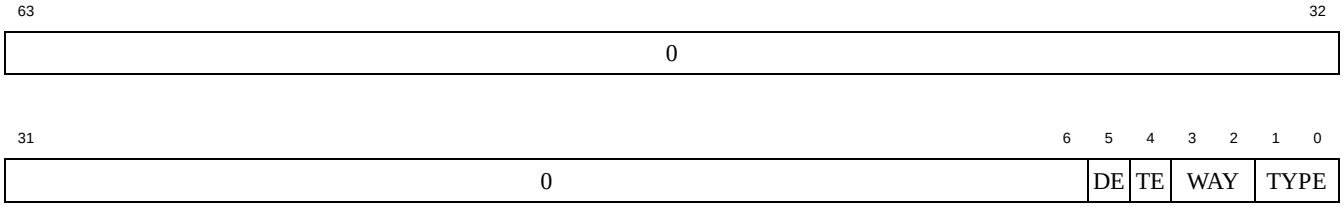


表 7-50 CacheErr 寄存器用于 I-Cache 校验错信息时的域描述

域名称	位	功能描述	读/写	复位值
0	63..6	只读恒为 0。	0	0
DE	5	I-Cache Data 部分奇偶校验出错标识。1：校验出错；0：校验无错。	R	0
TE	4	I-Cache Tag 部分奇偶校验出错标识。1：校验出错；0：校验无错。	R	0
WAY	3..2	记录校验出错在第几路上。	R	0
TYPE	1..0	校验出错类型。0：I-Cache；1：D-Cache；2,3：保留	R	0

图 7-47 说明了 CacheErr 寄存器用于 I-Cache 校验错信息时的格式；表 7-51 对该情况下寄存器各域进行了描述。

图 7-47 CacheErr 寄存器用于 D-Cache 校验错信息时的格式

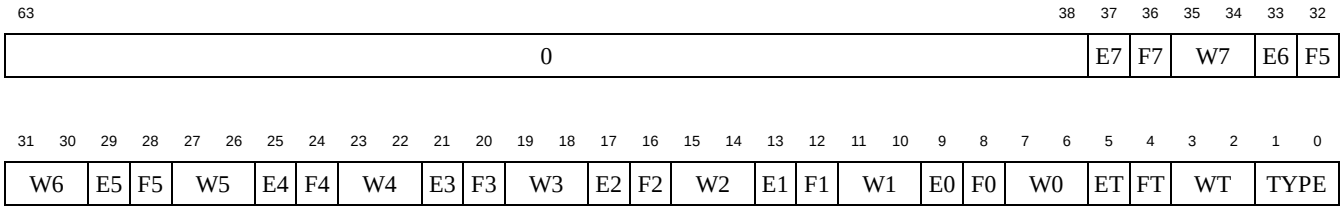


表 7-51 CacheErr 寄存器用于 D-Cache 校验错信息时的域描述

域名称	位	功能描述	读/写	复位值
0	63..38	只读恒为 0。	0	0
E7	37	D-Cache Data Bank 7 校验出错标识。1：出错；0：无错。	R	0
F7	36	D-Cache Data Bank 7 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W7	35..34	D-Cache Data Bank 7 出错在第几路上。	R	0
E6	33	D-Cache Data Bank 6 校验出错标识。1：出错；0：无错。	R	0

域名称	位	功能描述	读/写	复位值
F6	32	D-Cache Data Bank 6 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W6	31..30	D-Cache Data Bank 6 出错在第几路上。	R	0
E5	29	D-Cache Data Bank 5 校验出错标识。1：出错；0：无错。	R	0
F5	28	D-Cache Data Bank 5 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W5	27..26	D-Cache Data Bank 5 出错在第几路上。	R	0
E4	25	D-Cache Data Bank 4 校验出错标识。1：出错；0：无错。	R	0
F4	24	D-Cache Data Bank 4 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W4	23..22	D-Cache Data Bank 4 出错在第几路上。	R	0
E3	21	D-Cache Data Bank 3 校验出错标识。1：出错；0：无错。	R	0
F3	20	D-Cache Data Bank 3 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W3	19..18	D-Cache Data Bank 3 出错在第几路上。	R	0
E2	17	D-Cache Data Bank 2 校验出错标识。1：出错；0：无错。	R	0
F2	16	D-Cache Data Bank 2 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W2	15..14	D-Cache Data Bank 2 出错在第几路上。	R	0
E1	13	D-Cache Data Bank 1 校验出错标识。1：出错；0：无错。	R	0
F1	12	D-Cache Data Bank 1 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W1	11..10	D-Cache Data Bank 1 出错在第几路上。	R	0
E0	9	D-Cache Data Bank 0 校验出错标识。1：出错；0：无错。	R	0
F0	8	D-Cache Data Bank 0 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
W0	7..6	D-Cache Data Bank 0 出错在第几路上。	R	0
ET	5	D-Cache Tag 校验出错标识。1：出错；0：无错。	R	0
FT	4	D-Cache Tag 校验出 2 位及 2 位以上错标识。1：有此类错；0：无此类错。	R	0
WT	3..2	D-Cache Tag 出错在第几路上。	R	0
TYPE	1..0	校验出错类型。0：I-Cache；1：D-Cache；2,3：保留	R	0

7.45 CacheErr1 寄存器 (CP0 Register 27, Select 1)

CacheErr1 寄存器用于记录检查出 I-Cache 校验出错的指令的 PC 值，也用于记录检查出 D-Cache 校验出错的访问的物理地址。需要注意的是，对于 I-Cache 校验错，出错的 Cache 块不一定是 CacheErr1 寄存器中 PC 值所在的 Cache 块；对于 D-Cache 校验错，出错的 Cache 块不一定是 CacheErr1 寄存器中物理地址所在的块。CacheErr1 寄存器所存放的信息可以提取出出错处的 I-Cache/D-Cache Index 值、I-Cache Bank 范围，结合 CacheErr 寄存器中的信息，可以准确定位出错位置。

图 7-48 说明了 CacheErr1 寄存器的格式；表 7-52 表 7-32 Config 寄存器域描述对 CacheErr1 寄存器各域进行了描述。

图 7-48 CacheErr1 寄存器格式

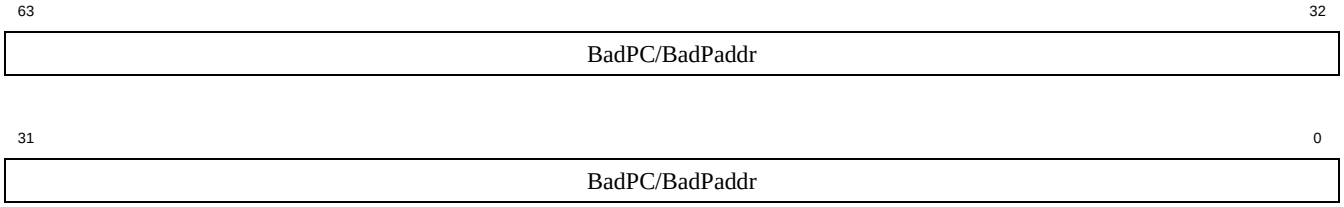


表 7-52 CacheErr1 寄存器域描述

域名称	位	功能描述	读/写	复位值
BadPC	63..0	检查出 I-Cache 校验出错的指令的 PC 值。	R	0
BadPaddr	63..0	检查出 D-Cache 校验出错的访问的物理地址。	R	0

7.46 TagLo 寄存器 (CP0 Register28, Select 0)

TagLo 寄存器是一个软件可读写的寄存器，其与 TagHi 寄存器一起作为 Index Load Tag 和 Index Store Tag 类 CACHE 指令与各级 Cache 的 Tag 部分数据的交互接口，同时也作为 Index Load Data 和 Index Store Data 类 CACHE 指令与各级 Cache 的 Data 部分数据的交互接口。

当 TagLo 用于访问不同 Cache 时，其具体的格式也存在区别。图 7-49 说明了 TagLo 寄存器用于访问 I-Cache Tag 时的格式；表 7-53 表 7-32 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-49 TagLo 寄存器用于访问 I-Cache Tag 时的格式

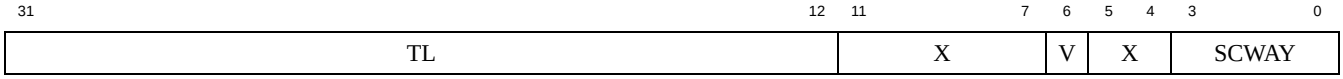


表 7-53 TagLo 寄存器用于访问 I-Cache Tag 时的域描述

域名称	位	功能描述	读/写	复位值
TL	31..12	待写入或读出的 Tag 低位内容，对应物理地址的[31:12]。	R/W	无
X	11..7	可以正常写入、读出，但不参与其它操作。	R/W	无
V	6	待写入或读出的 Cache 块状态。 0：Cache 块无效；1：Cache 块有效。	R/W	无
X	5..4	可以正常写入、读出，但不参与其它操作。	R/W	无
SCWAY	3..0	待写入或读出的 Cache 块在 Scache 中的第几路。	R/W	无

图 7-50 说明了 TagLo 寄存器用于访问 D-Cache Tag 时的格式；表 7-54 对这种情况下寄存器各域进行了描述。

图 7-50 TagLo 寄存器用于访问 D-Cache Tag 时的格式

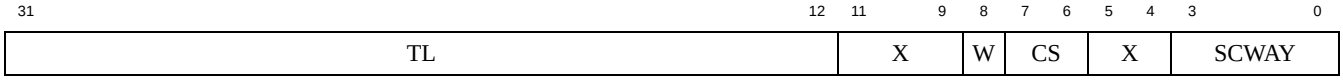


表 7-54 TagLo 寄存器用于访问 D-Cache Tag 时的域描述

域名称	位	功能描述	读/写	复位值
TL	31..12	待写入或读出的 Tag 低位内容，对应物理地址的[31:12]。	R/W	无
X	11..9	可以正常写入、读出，但不参与其它操作。	R/W	无
W	8	待写入或读出的 Cache 块的“脏”标记。1：脏块；0：非脏块。	R/W	无
CS	7..6	待写入或读出的 Cache 块状态。 0：无效块；1：共享块；2：独占块；3：保留，如果使用处理器结果不确定。	R/W	无
X	5..4	可以正常写入、读出，但不参与其它操作。	R/W	无
SCWAY	3..0	待写入或读出的 Cache 块在 Scache 中的第几路。	R/W	无

图 7-51 说明了 TagLo 寄存器用于访问 V-Cache Tag 时的格式；表 7-55 对这种情况下寄存器各域进行

了描述。

图 7-51 TagLo 寄存器用于访问 V-Cache Tag 时的格式

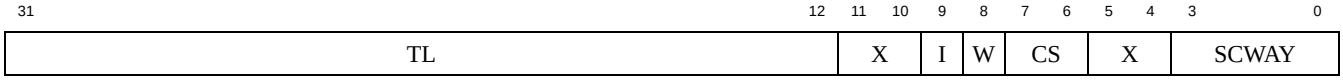


表 7-55 TagLo 寄存器用于访问 V-Cache Tag 时的域描述

域名称	位	功能描述	读/写	复位值
TL	31..12	待写入或读出的 Tag 低位内容，对应物理地址的[31:12]。	R/W	无
X	11..10	可以正常写入、读出，但不参与其它操作。	R/W	无
I	9	待写入或读出的 Cache 块的指令/数据属性。1：指令块；0：数据块。	R/W	无
W	8	待写入或读出的 Cache 块的“脏”标记。1：脏块；0：非脏块。	R/W	无
CS	7..6	待写入或读出的 Cache 块状态。 0：无效块；1：共享块；2：独占块；3：保留，如果使用处理器结果不确定。	R/W	无
X	5..4	可以正常写入、读出，但不参与其它操作。	R/W	无
SCWAY	3..0	待写入或读出的 Cache 块在 Scache 中的第几路。	R/W	无

图 7-52 说明了 TagLo 寄存器用于访问 S-Cache Tag 时的格式；表 7-56 对这种情况下寄存器各域进行了描述。

图 7-52 TagLo 寄存器用于访问 S-Cache Tag 时的格式

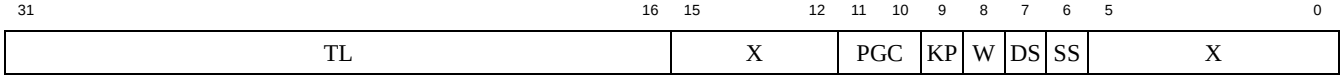


表 7-56 TagLo 寄存器用于访问 S-Cache Tag 时的域描述

域名称	位	功能描述	读/写	复位值
TL	31..16	待写入或读出的 Tag 低位内容，对应物理地址的[31:16]。	R/W	无
X	15..12	可以正常写入、读出，但不参与其它操作。	R/W	无
PGC	11..10	待写入或读出的 Cache 块对应的 PageColor 位。当系统采用 4KB 基本页大小时，两位均有意义，当系统采用 8KB 基本页大小时，仅第 13 位有意义。有关 PageColor 位的详细描述，请参看 5.4.5 节。	R/W	无
KP	9	在写入操作时，置为 0 表示将待写入的 Cache 块对应的目录项清空，置为 1 表示待写入 Cache 块对应的目录项的内容保持不变。读出操作时，该域内容无意义。	R/W	无
W	8	待写入或读出的 Cache 块的“脏”标记。1：脏块；0：非脏块。	R/W	无
DS	7	待写入或读出的 Cache 块对应目录项的状态。1：目录脏；0：目录干净。	R/W	无
SS	6	待写入或读出的 Cache 块的状态。1：有效块；0：无效块。	R/W	无
X	5..0	可以正常写入、读出，但不参与其它操作。	R/W	无

图 7-53 说明了 TagLo 寄存器用于访问各级 Cache 中 Data 部分时的格式；表 7-57 对这种情况下寄存器各域进行了描述。

图 7-53 TagLo 寄存器用于访问各级 Cache Data 时的格式

31	0
DATA	

表 7-57 TagLo 寄存器用于访问各级 Cache Data 时的域描述

域名称	位	功能描述	读/写	复位值
DATA	31..0	待写入或读出的 Cache 块中 Data Bank 低 32 位内容。	R/W	无

编程提示：软件在使用 Index Store Tag 和 Index Store Data 类指令填充 Cache 内容时，请保持各级 Cache 的所填入内容之间满足 GS464E 的 Cache 一致性要求。否则处理器的行为将不确定。

7.47 DataLo 寄存器(CP0 Register 28, Select 1)

在 GS464E 中，DataLo 与 DataHi 寄存器并不作为访问各级 Cache 的 Data 部分的交互接口，仅当 Index Load Data 和 Index Store Data 类 CACHE 指令访问 I-Cache 时，作为 I-Cache 预译码数据部分的交互接口。在其它情况下，DataLo 允许软件读、写，但不参与其它任何操作。

图 7-54 说明了 DataLo 寄存器用于访问 I-Cache 时的格式；表 7-58 表 7-32 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-54 DataLo 寄存器用于 I-Cache 访问时的格式

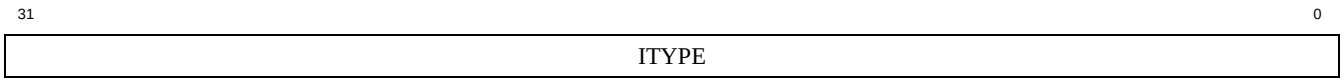


表 7-58 DataLo 寄存器用于 I-Cache 访问时的域描述

域名称	位	功能描述	读/写	复位值
ITYPE	31..0	待写入或读出的 I-Cache 块的预译码信息的[31:0]位。	R/W	无

编程提示：软件在使用 Index Store Tag 和 Index Store Data 类指令填充 Cache 内容时，请保持各级 Cache 的所填入内容之间满足 GS464E 的 Cache 一致性要求。否则处理器的行为将不确定。

7.48 TagHi 寄存器 (CP0 Register 29, Select 0)

TagHi 寄存器是一个软件可读写的寄存器，其与 TagLo 寄存器一起作为 Index Load Tag 和 Index Store Tag 类 CACHE 指令与各级 Cache 的 Tag 部分数据的交互接口，也作为 Index Load Data 和 Index Store Data 类 CACHE 指令与各级 Cache 的 Data 部分数据的交互接口。

图 7-55 说明了 TagHi 寄存器用于访问各级 Cache Tag 时的格式；表 7-59 对这种情况下寄存器各域进行了描述。

图 7-55 TagHi 寄存器用于访问各级 Cache Tag 时的格式

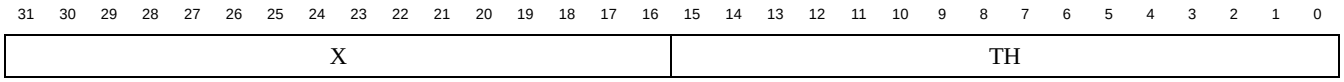


表 7-59 TagHi 寄存器用于访问各级 Cache Tag 时的域描述

域名称	位	功能描述	读/写	复位值
X	31..12	可以正常写入、读出，但不参与其它操作。	R/W	无
TH	15..0	待写入或读出的 Tag 高位内容，对应物理地址的[47:32]。	R/W	无

图 7-56 说明了 TagLo 寄存器用于访问各级 Cache Data 时的格式；表 7-60 对这种情况下寄存器各域进行了描述。

图 7-56 TagHi 寄存器用于访问各级 Cache Data 时的格式

表 7-60 TagHi 寄存器用于访问各级 Cache Data 时的域描述

域名称	位	功能描述	读/写	复位值
DATA	31..0	待写入或读出的 Cache 块中 Data Bank 高 32 位内容。	R/W	无

编程提示：软件在使用 Index Store Tag 和 Index Store Data 类指令填充 Cache 内容时，请保持各级 Cache 的所填入内容之间满足 GS464E 的 Cache 一致性要求。否则处理器的行为将不确定。

7.49 DataHi 寄存器(CP0 Register 29, Select 1)

在 GS464E 中，DataLo 与 DataHi 寄存器并不作为访问各级 Cache 的 Data 部分的交互接口，仅当 Index Load Data 和 Index Store Data 类 CACHE 指令访问 I-Cache 时，作为 I-Cache 预译码数据部分的交互接口。在其它情况下，DataHi 允许软件读、写，但不参与其它任何操作。

图 7-57 说明了 DataHi 寄存器用于访问 I-Cache 时的格式；表 7-61 表 7-32 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-57 DataHi 寄存器用于 I-Cache 访问时的格式

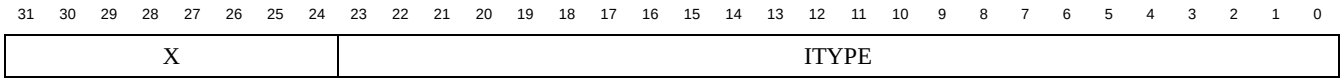


表 7-61 DataHi 寄存器用于 I-Cache 访问时的域描述

域名称	位	功能描述	读/写	复位值
X	31..24	可以正常写入、读出，但不参与其它操作。	R/W	无
ITYPE	23.0	待写入或读出的 I-Cache 块的预译码信息的[55:32]位。	R/W	无

编程提示：软件在使用 Index Store Tag 和 Index Store Data 类指令填充 Cache 内容时，请保持各级 Cache 的所填入内容之间满足 GS464E 的 Cache 一致性要求。否则处理器的行为将不确定。

7.50 ErrorEPC 寄存器 (CP0 Register 30, Select 0)

ErrorEPC 寄存器是一个 64 位可读写寄存器，其作用与 EPC 寄存器类似，只是其只用于存放冷复位、软复位、不可屏蔽中断和 Cache 错例外处理完成后继续执行的指令的 PC，而且 ErrorEPC 寄存器没有类似于 EPC 寄存器的分支延迟槽标识(Cause.BD)。

在响应上述例外时，处理器硬件对 ErrorEPC 寄存器写入：

直接触发例外的指令的 PC。

当直接触发例外的指令位于分支延迟槽时，记录该指令前一条分支或跳转指令的 PC。

图 7-58 说明了 ErrorEPC 寄存器的格式；表 7-62 对 ErrorEPC 寄存器各域进行了描述。

图 7-58 ErrorEPC 寄存器格式

表 7-62 ErrorEPC 寄存器域描述

域名称	位	功能描述	读/写	复位值
ErrorEPC	63..0	例外处理完成后继续开始执行的指令的 PC。	R/W	无

7.51 DESAVE 寄存器 (CP0 Register 31, Select 0)

DESAVE 寄存器是一个 64 位可读写寄存器，用于调试例外处理程序暂存数据。通常调试例外处理程序用 DESAVE 寄存器保存一个通用寄存器，随后利用该通用寄存器作为访存指令的基址寄存器保存当前上下文到指定区域，譬如 dmseg 段。

核心态软件不允许使用 DESAVE 寄存器。

图 7-59 说明了 DESAVE 寄存器的格式；表 7-63 对 DESAVE 寄存器各域进行了描述。

图 7-59 DESAVE 寄存器格式

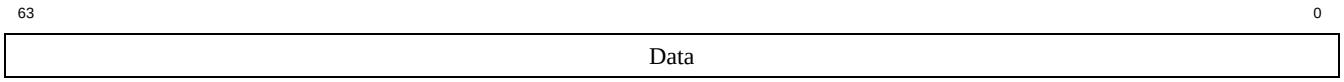


表 7-63 DESAVE 寄存器域描述

域名称	位	功能描述	读/写	复位值
Data	63..0	调试例外处理程序暂存的数据。	R/W	无

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

KScratch1~6 寄存器是一组 64 位可读写寄存器，用于存放核心态软件的暂存数据。

处于调试模式(Debug Mode)的软件不可以访问 KScratch1~6，但是此时软件可以通过 DESAVE 寄存器暂存数据。

图 7-60 说明了 KScratch 寄存器的格式；表 7-64 对 KScratch 寄存器各域进行了描述。

图 7-60 KScratch*n* 寄存器格式

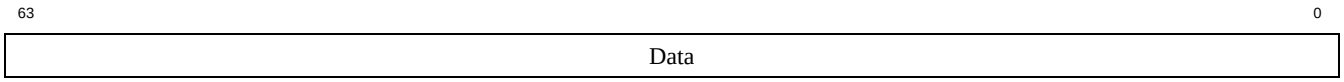


表 7-64 KScratch*n* 寄存器域描述

域名称	位	功能描述	读/写	复位值
Data	63..0	核心态软件暂存数据。	R/W	无

8 处理器性能分析与优化

GS464E 的性能分析与优化工作主要通过硬件集成的性能计数器完成。性能计数器用于统计处理器内部某些事件的发生次数，是芯片硅后性能验证、编译器优化、软件性能调优工作的重要支撑。此外，由于这些性能计数器所记录某些事件的统计意义与芯片运行时的功耗有相关性，性能计数器还可以用来指导芯片的功耗管理。

GS464E 所采用的性能计数器包含两个部分：处理器核性能计数器和共享缓存性能计数器。

8.1 性能计数器组织形式及访问方法

8.1.1 处理器核性能计数器

每个 GS464E 的处理器核实现了 28 组性能计数器，均分在 FETCH 模块、RMAP 模块、ROQ 模块、FIX 模块、FLOAT 模块、MEMORY 模块和 CACHE2MEM 模块，每个模块各自包含 4 组性能计数器。每组性能计数器对应两个物理寄存器，一个用于计数（48 位计数器），一个用于控制计数。每个模块内部的硬件计数器只能用于统计与该模块相关的事件。模块内部的任一组性能计数器均可以统计与该模块相关的任何性能计数事件。

GS464E 处理器核的性能计数器仍通过 MIPS 架构下的 CP0 Performance Counter 寄存器供软件配置和访问，但是访问的形式与传统 MIPS 处理器有些许差异。具体表现为，CP0 中的 Performance Counter 寄存器仅作为一个配置与访问的接口而存在，读、写操作具体传递至哪个硬件计数器是由 Performance Counter 寄存器中配置的事件号动态确立起来的。软件先将奇数号 Performance Counter 寄存器配置上某个事件，处理器内部硬件将该事件与所属模块内的硬件计数器建立起一一映射关系；该映射关系建立后，软件再对偶数号 Performance Counter 寄存器进行读写就将直接操作对所映射的硬件计数器；最终，软件启动该硬件计数器开始进行统计。需要特别说明的是，Performance Counter 每次配置前需要先将所有模块中的性能计数器都初始化为无效状态。具体做法是给在每个模块中配置一个该模块事件允许范围内的最大事件号，并将计数器值置为 0。

8.1.2 共享缓存性能计数器

每个 GS464E 的共享缓存体实现了 4 组 48 位的性能计数器。每组性能计数器对应两个物理寄存器，一个用于计数（48 位计数器），一个用于控制计数。每个性能计数器均可以统计该共享缓存体内发生的任何性能计数事件。

龙芯 3A2000 中四个共享缓存体的性能计数器的访问通过芯片的 confbus 总线进行。四个共享缓存体中共计 16 组性能寄存器采用统一的 confbus 基址 0x3FF0.0000，每个寄存器的具体偏移如表 8-1 所示：

表 8-1 共享缓存性能计数器寄存器地址偏移

寄存器名称	Scache0	Scache1	Scache2	Scache3
PerfCtl0	0x0800	0x0900	0x0a00	0x0b00
PerfCnt0	0x0808	0x0908	0x0a08	0x0b08
PerfCtl1	0x0810	0x0910	0x0a10	0x0b10
PerfCnt1	0x0818	0x0918	0x0a18	0x0b18
PerfCtl2	0x0820	0x0920	0x0a20	0x0b20

寄存器名称	Scache0	Scache1	Scache2	Scache3
PerfCnt2	0x0828	0x0928	0x0a28	0x0b28
PerfCtl3	0x0830	0x0930	0x0a30	0x0b30
PerfCnt3	0x0838	0x0938	0x0a38	0x0b38

使用举例

如果要统计 scache0 收到的 scread 总数、dmaread 总数，那么可以如下操作：

- Step1：将 sc0_perfcnt0 写为 0（sd $0, 0x3ff00808），sc0_perfcnt1 写为 0（sd $0, 0x3ff00818），sc0_perfcnt2 写为 0（sd $0, 0x3ff00828）
- Step2：将 sc0_perfctrl0 写为 1（sd value_1,0x3ff00800），sc_perfctrl1 写为 9（sd value_9, 0x3ff00810），sc_perftrl2 写为 10（sd value_10,0x3ff00820）
- Step3：执行程序
- Step4：读取 sc0_perfcnt0（ld t0, 0x3ff00808），sc0_perfcnt1（ld t1, 0x3ff00818），sc0_perfcnt2（ld t2, 0x3ff00828）；其中 t0 的结果就是 scread 总数，t1+t2 的结果就是 dmaread 总数。

8.2 处理器性能计数事件

GS464E 定义的性能事件分为三大类：

第一类，用于分析程序在指令集层面体现出的特性。具体是统计流水线提交阶段不同类型指令各自的数量，从而得到程序动态执行指令类型的分布情况。

第二类，用于分析程序代码在与处理器微结构交互影响的过程中所体现的性能瓶颈，出发点是为了优化程序。主要是通过统计各类导致流水线阻塞的事件。除了 Cache 不命中次数、队列满次数、分支预测出错次数等基础事件外，GS464E 还新增了一批关于延迟周期数的统计，譬如分支误预测后清空前端流水线导致 regmap 流水级断流的周期数等。

第三类，用于为设计空间探索积累数据，出发点是为了优化微结构。譬如，当前双访存部件采用全功能双端口 RAM 实现，比单端口 RAM 相比开销大很多，因此加入了关于同一拍两条 load 操作 dcache RAM 冲突的次数的统计，在现有结构上这种冲突其实不会造成流水线阻塞，因此并不影响程序的性能。

8.2.1 处理器核性能计数事件定义

GS464E 处理器核的性能计数器事件定义如表 8-2 所示。

表 8-2 处理器核性能计数器事件定义

事件号	事件描述
	FETCH 模块
1	Inst Queue 全空的周期数
2	每周期写入 Inst Queue 的指令数
3	前端流水线阻塞周期数（进入 Inst Queue 的指令数等于 0）
4	进入 Inst Queue 的指令数等于 1
5	进入 Inst Queue 的指令数等于 2
6	进入 Inst Queue 的指令数等于 3
7	进入 Inst Queue 的指令数等于 4
8	进入 Inst Queue 的指令数等于 5

事件号	事件描述
9	进入 Inst Queue 的指令数等于 6
10	进入 Inst Queue 的指令数等于 7
11	进入 Inst Queue 的指令数等于 8
12	cache 空间下因为跨越 cacheline 边界使得进入 InstQueue 的指令数小于 8
13	由于 Inst Queue 满造成的前端流水线阻塞周期数
14	每周期译码指令数
15	每周期来自 Loop Buffer 的译码指令数
16	从 Loop buffer 取指令的 Loop 个数
17	识别出的 Loop 个数（包含可以放入 Loop Buffer 和无法放入 Loop Buffer 的）
18	每周期译码的分支指令数等于 0
19	每周期译码的分支指令数等于 1
20	每周期译码的分支指令数等于 2
21	由于 Icache Miss 造成的前端流水线阻塞
22	由于 BrBTB 未能成功预测 taken branch 造成的前端流水线阻塞
24	Icahe 模块发起且被 missq 接收的 Icache miss 次数
26	ITLB miss 但在 TLB 命中的次数
27	ITLB 被 flush 的次数
	RMAP 模块
64	资源分配被阻塞
65	GR 重命名资源满阻塞
66	GR 重命名资源满假阻塞（待进入指令无需要定点重命名资源的）
67	FR 重命名资源满阻塞
68	FR 重命名资源满假阻塞（待进入指令无需要浮点重命名资源的）
69	FCR 重命名资源满阻塞
70	FCR 重命名资源满假阻塞（待进入指令无需要 FCR 重命名资源的）
71	ACC 重命名资源满阻塞
72	ACC 重命名资源满假阻塞（待进入指令无需要 ACC 重命名资源的）
73	DSPCtrl 重命名资源满阻塞
74	DSPCtrl 重命名资源满假阻塞（待进入指令无需要 DSPCtrl 重命名资源的）
75	BRQ 满阻塞
76	BRQ 满假阻塞（待进入指令无需要进入 BRQ 的）
77	FXQ 满阻塞
78	FXQ 满假阻塞（待进入指令无需要进入 FXQ 的）
79	FTQ 满阻塞
80	FTQ 满假阻塞（待进入指令无需要进入 FTQ 的）
81	MMQ 满阻塞
82	MMQ 满假阻塞（待进入指令无需要进入 MMQ 的）
83	CP0Q 满阻塞
84	CP0Q 满假阻塞（待进入指令无需要进入 CP0Q 的）
85	ROQ 满阻塞
86	完成资源分配阶段的 NOP 类指令数量

事件号	事件描述
87	每周期从 regmap 发射至各发射队列的操作数
88	例外(不含分支误预测)清空流水线的开销（regmap 流水级被例外清空后至首条指令到达 regmap 流水线）
89	分支预测错清空流水线的开销
	ROQ 模块
128	内部流水线时钟
129	每周期提交指令数
130	提交的 ALU 操作
131	提交的 FALU 操作
132	提交的 Memory/CP0/定浮点互换操作
133	提交的 load 操作
134	提交的 store 操作
135	提交的 LL 类操作
136	提交的 SC 类操作
137	提交的非对齐 load 操作
138	提交的非对齐 store 操作
139	所有例外和中断的次数
140	中断的次数
141	从 ROQ 接收到中断信号到产生中断例外
142	从 ROQ 接收到中断信号到中断例外处理程序第一条指令进入 ROQ
143	虚拟机例外次数
144	地址错例外次数
145	TLB 相关例外次数
146	TLB refill 例外次数
147	TLB refill 例外的处理时间（TLB refill 例外清空流水线开始到 TLB refill 例外的 ERET 返回）
148	brq 提交的分支指令
149	brq 提交的 jump register 分支指令
150	brq 提交的 jump and link 分支指令
151	brq 提交的 branch and link 分支指令
152	brq 提交的 bht 分支指令
153	brq 提交的 likely 分支指令
154	brq 提交的 not taken 的分支指令
155	brq 提交的 taken 的分支指令
156	brq 提交的预测错的分支指令
157	brq 提交的预测错的 jump register 分支指令
158	brq 提交的预测错的 jump and link 分支指令
159	brq 提交的预测错的 branch and link 分支指令
160	brq 提交的预测错的 bht 分支指令
161	brq 提交的预测错的 likely 分支指令
162	brq 提交的预测错的 not taken 的分支指令
163	brq 提交的预测错的 taken 的分支指令
	FIX 模块

事件号	事件描述
192	fxq 无发射
193	fxq 发射执行操作数
194	fxq 发射到 FU0 功能部件执行的操作数
195	fxq 发射到 FU1 功能部件执行的操作数
196	FU0 中定点乘法部件处于执行状态
197	FU0 中定点除法部件处于执行状态
198	FU1 中定点乘法部件处于执行状态
199	FU1 中定点除法部件处于执行状态
	FLOAT 模块
256	ftq 无发射
257	ftq 发射执行操作数
258	ftq 发射到 FU3 功能部件执行的操作数
259	ftq 发射到 FU4 功能部件执行的操作数
260	FU3 空闲，FU4 满，但 ftq 只有 FU4 待发射项
261	FU4 空闲，FU3 满，但 ftq 只有 FU3 待发射项
262	每周期发射标量浮点操作数
263	每周期发射的 64 位多媒体加速指令数(指令名含“GS”前缀)
264	每周期发射的 64 位多媒体加速指令数(指令名不含“GS”前缀)
272	FU3 中浮点除法/开方部件处于执行状态
274	FU4 中浮点除法/开方部件处于执行状态
	MEMORY 模块
320	mmq 无发射
321	mmq 发射执行操作数
322	mmq 中，每拍发射 FU2 指令
323	mmq 中，每拍发射 FU5 指令
324	load 发射次数
325	store 发射次数
326	源操作数至少有一个浮点的访存指令数量
327	同时具有定点和浮点操作数的指令的发射次数
329	wait_first 阻塞的周期数
330	SYNC 操作阻塞的周期数
331	stall_issue 阻塞的周期数
332	软件预取操作发射
333	新发射的访存操作堵塞 store 操作写入 dcache 的周期数
334	同一拍两个 load 发生 bank 冲突
337	dcachewrite0 和 1 同时有效的次数
338	执行成功的 SC 类指令次数
339	store 指令 dcahe miss 次数（包括不命中和非 EXC 状态）
340	store 指令因 dcache shared 状态导致的 dcache miss 次数
	CACHE2MEM 模块
341	store 指令 dcache 命中次数

事件号	事件描述
342	load 命中次数
343	fwdbus2 次数
344	fwdbus5 次数
345	fwdbus 总次数，fwdbus2+fwdbus5
346	因 load、store 地址冲突导致的访存操作回滚次数（dwaitstore）
347	因 load、store 地址冲突导致的例外次数（mispec）
348	cp0qhead 因 dcachewrite 不顺利而回滚的次数
349	cp0q 发出 dmemread 请求次数
350	cp0q 发出 duncache 请求次数
351	resbus2 挤占 resbus5 次数，LQ、LQC1 等存在两个 dest 的访存操作
352	软件预取在 L1 Dcache 命中次数
353	store 软件预取在 L1 dcache 命中次数
354	store 软件预取在 L1 dcache miss 次数
355	load 软件预取在 L1 dcache 命中次数
356	load 软件预取在 L1 dcache miss 次数
357	store 软件预取在 L1 dcache 因 share state 不命中的次数
358	specfwdbus2 次数
359	specfwdbus5 次数
360	specfwdbus 次数：specfwdbus2+specfwdbus5
384	数据 load 请求访问 vcache 次数
385	数据 store 请求访问 vcache 次数
386	数据请求访问 vcache 次数
387	指令请求访问 vcache 次数
388	vcache 访问次数
389	软件预取访问 vcache 的次数
390	vcache load 命中次数
391	vcache store 命中次数
392	vcache 数据命中次数
393	vcache 指令命中次数
394	vcache 命中次数
395	vcache 软件配置预取命中次数
396	vcache load 失效次数
397	vcache store 失效次数
398	vcache 数据失效次数
399	vcache 指令失效次数
400	vcache 失效次数
401	vcache 软件配置预取失效次数
402	vcache 被 extreq 操作无效掉有效块的次数
403	vcache 被 wtbk 操作降级有效块的次数
404	vcache 被 INV 操作无效掉有效块的次数
405	vcache 被 INVWTBK 无效掉有效块的次数

事件号	事件描述
406	处理器核对外总线读请求次数
407	处理器核对外总线写请求次数
408	带有写数据的总线写请求次数
409	总线读请求因与总线写请求地址冲突而被阻塞
410	missq 处理的 WTBK 请求数量
411	missq 处理的 INVWTBK 请求数量
412	missq 处理的 INV 请求数量
413	missq 处理的 INV 类（上述 3 个）请求数量
414	refill 的总次数(包括 exreq 和 replace+refill)
415	refill 的针对 icache 的总次数
416	refill 的针对 dcache 的总次数
417	refill(replace+refill)的次数
418	refill 一个 dcache　shared 块的次数
419	refill 一个 dcache　exc 块的次数
420	refill 数据的总次数（replace+refill）
421	refill 指令的总次数（replace+refill）
422	dcache 替换出来一个有效块的次数
423	dcache 替换出来一个 shared 块的次数
424	dcache 替换出来一个 exc 块的次数
425	dcache 替换出来一个 dirty 块的次数
426	icache 替换出有效数据的次数
427	vcache 替换次数
428	vcache 替换出一个有用块的次数
429	vcache 替换出一个 shared 块的次数
430	vcache 替换出一个 exc 块的次数
431	vcache 替换出来一个 dirty 块的次数
432	vcache 替换出有用 dc 块的次数
433	vcache 替换出有用 ic 块的次数
434	累计每一拍未从 scache 返回的 load 请求数量（missq 最多只有 15 项用于处理 scache 请求）
435	累计每一拍未从 scache 返回的 store 请求数量
436	累计每一拍未从 scache 返回的取指请求数量
437	送出的 sc read 的总个数
438	送出的 scread 中 load 的总个数
439	送出的 scread 中 store 的总个数
440	scread 数据访问总个数
441	scread 指令访问总个数
442	scread 非预取总个数
443	scread 非预取数据 load 总个数
444	scread 非预取数据 store 总个数
445	scread 非预取数据访问总个数
446	scread 非预取取指访问总个数

事件号	事件描述
447	送出的 scread 中预取的总个数
448	送出的 scread 中 load 预取的个数
449	送出的 scread 中 store 预取的个数
450	scread 预取数据访问总个数
451	scread 预取指令访问总个数
452	missq 处理的软件预取请求个数
453	missq 发出的 scwrite 个数
454	missq 因 replace 操作发起的 scwrite 个数
455	missq 因 invalid 操作发出的 RESP 类 scwrite 个数
456	missq 因 replace 操作发起的 scwrite 操作，且 replace 有效块
457	missq 真正接受请求个数 miss_en
458	missq 真正接受 load 请求个数 miss_en
459	missq 真正接受 store 请求个数 miss_en
460	missq 真正接受的数据访问的个数
461	missq 真正接受的指令访问的个数
462	missq 占项情况(missq 非空项)
463	missq 普通访问占项情况
464	missq 取指访问占项情况
465	missq 外部请求占项情况
466	missq 预取请求占项情况
467	missq 占项拍数（missq 存在有效项的拍数，即 missq 不空时间）
468	missq 普通访问占项拍数
469	missq 中取指访问占项拍数
470	missq 外部请求占项拍数
471	missq 预取请求占项拍数
472	missq 满计数（missq 不能接受普通访问，missq 有效项不小于 15）
473	load 请求在 missq 中碰到预取的次数
474	load 请求在 missq 中碰到预取 pre_scref 的次数
475	load 请求在 missq 中碰到预取 pre_wait 的次数
476	load 请求在 missq 中碰到预取 pre_rdy 的次数
477	store 请求在 missq 中碰到预取 pre_scref 且 load 操作的次数
478	store 请求在 missq 中碰到预取 pre_rdy 且 state=shard 次数
479	store 请求在 missq 中碰到预取 pre_wait 且 load 操作次数
480	store 请求在 missq 中碰到预取 pre_scref 且 store 操作的次数
481	store 请求在 missq 中碰到预取 pre_rdy 且 state=exc 次数
482	store 请求在 missq 中碰到预取 pre_wait 且 store 操作的次数
483	store 请求在 missq 中碰到预取的次数（包括命中在 store 预取和命中在 load 预取）
484	store 请求在 missq 中碰到有效预取的次数（命中在 store 预取）
485	所有请求在 missq 中碰到预取的次数（load+store）
486	所有请求在 missq 中碰到 pre_scref 预取的次数（laod+store）
487	所有请求在 missq 中碰到 pre_rdy 预取的次数（load+store）

事件号	事件描述
488	所有请求在 missq 中碰到 pre_wait 预取的次数（load+store）
489	取指请求在 missq 中碰到预取的次数
490	取指请求在 missq 中碰到 pre_scref 预取的次数
491	取指请求在 missq 中碰到 pre_rdy 预取的次数
492	取指请求在 missq 中碰到 pre_wait 预取的次数
495	数据和取指在 missq 中碰到 pre_rdy 预取的次数
496	数据和取指在 missq 中碰到 pre_wait 预取的次数
497	硬件 load 预取请求被 Scache cancel 的次数[15]
498	硬件 store 预取请求被 Scache cancel 的次数
499	硬件数据访问预取请求被 Scache cancel 的次数
500	硬件取指预取请求被 Scache cancel 的次数
501	硬件预取请求被 Scache cancel 的次数
502	硬件 load 预取的个数
503	硬件 store 预取的个数
504	硬件数据访问预取的个数
505	硬件取指预取的个数
506	硬件预取的个数
507	tagged 触发的 load 预取个数
508	miss 触发的 load 预取个数
509	tagged 触发的 store 预取个数
510	miss 触发的 store 预取个数
511	tagged 触发的数据访问预取个数
512	miss 触发的数据预取个数
513	tagged 触发的指令预取个数
514	miss 触发的指令预取个数
515	tagged 触发的预取个数
516	miss 触发的预取个数
517	被 missq 接受的 load 预取个数
518	被 missq 接受的 store 预取个数
519	被 missq 接受的数据访问预取个数
520	被 missq 接受的指令预取个数
521	被 missq 接受的预取个数（重复请求不进入 missq）
522	从 scache 回来的有效 load 预取个数
523	从 scache 回来的有效 store 预取个数
524	从 scache 回来的有效数据访问预取个数
525	从 scache 回来的指令预取个数
526	从 scache 回来的有效预取个数（pre_scref->rdy || pre_scref->pre_rdy）
527	能进入 pre_rdy 的 load 预取个数
528	能进入 pre_rdy 的 store 预取个数
529	能进入 pre_rdy 的数据访问预取个数

[15] 被 scache cancel 意味着该数据已经在该处理器核的 Cache 中

事件号	事件描述
530	能进入 pre_rdy 的指令预取个数
531	能进入 pre_rdy 的预取个数（pre_scref -> pre_rdy）
532	累计每一拍处于 pre_rdy 的 load 预取请求个数
533	累计每一拍处于 pre_rdy 的 store 预取请求个数
534	累计每一拍处于 pre_rdy 的数据访问预取请求个数
535	累计每一拍处于 pre_rdy 的指令预取请求个数
536	累计每一拍处于 pre_rdy 的请求个数
537	累计每一拍处于 pre_scref 且被正常 load 请求命中的预取个数
539	累计每一拍处于 pre_scref 且被正常 store 请求命中的预取个数
540	累计每一拍处于 pre_scref 且被正常数据访问请求命中的预取个数
541	累计每一拍处于 pre_scref 且被正常取指请求命中的预取个数
542	累计每一拍处于 pre_scref 且被正常访问命中的预取个数
543	在 pre_scref 状态就被 load 访问 hit 的预取个数
544	在 pre_scref 状态就被 store 访问 hit 的预取个数
545	在 pre_scref 状态就被数据访问 hit 的预取个数
546	在 pre_scref 状态就被取指访问 hit 的预取个数
547	在 pre_scref 状态就被 hit 的预取个数，即从 pre_scref -> rdy 的次数
548	从 pre_scref 状态回到 miss 状态的 load 个数
553	在 pre_wait 状态就被 load 访问 hit 的预取个数
554	在 pre_wait 状态就被 store 访问 hit 的预取个数
555	在 pre_wait 状态就被数据访问 hit 的预取个数
556	在 pre_wait 状态就被取指访问 hit 的预取个数
557	在 pre_wait 状态就被 hit 的预取个数，即从 pre_wait -> pcmiss 的次数
558	因 missq 不能接受正常访问而被替换出去的处于 pre_wait 状态的预取项
559	因 missq 不能接受正常访问而被替换出去的处于 pre_rdy 状态的预取项
560	预取项被 INV 的次数
561	累计每一拍 load 预取占项情况
562	累计这一拍 load 预取是否占项
563	累计每一拍 store 预取占项情况
564	累计这一拍 store 预取是否占项
565	累计每一拍数据预取占项情况
566	累计这一拍数据预取是否占项
567	累计每一拍取指预取占项情况
568	累计这一拍取指预取是否占项
569	累计 load 预取在 pre_scref 和 pre_rdy 命中个数
570	累计 store 预取在 pre_scref 和 pre_rdy 命中个数
571	累计数据预取在 pre_scref 和 pre_rdy 命中个数
572	累计取指预取在 pre_scref 和 pre_rdy 命中个数
573	累计预取在 pre_scref 和 pre_rdy 命中个数

8.2.2 共享缓存性能计数事件定义

表 8-3 共享缓存性能计数器事件定义

事件号	事件描述
0	因 ctrl 中没有开关标记，因此配置为 0 表示没有开。
1	收到的所有 request 请求的个数
2	收到的 cachable 的 request 请求的个数
3	收到的所有非 DMA 操作的 cachable 的 request 请求
4	收到的所有非预取属性的 cachable 的 request 请求
5	收到的所有 cached 的 REQ_READ 请求
6	收到的所有 cached 的 REQ_WRITE 请求
7	收到的所有的非预取属性的 cached 的 REQ_READ
8	收到的所有的非预取属性的 cached 的 REQ_WRITE
9	收到的所有 cached 的 DMAREAD 请求
10	收到的所有 cached 的 DMAWRITE 请求
11	收到的所有 uncached 的 DMAREAD
12	收到的所有 uncached 的 DMAWRITE
13	收到的所有 DMA 请求个数
14	收到的所有类型是 scache_prefx 的请求个数
15	Scache 硬件自己生成的预取被接受的个数
16	收到的所有类型是 store_fill_full 的请求个数
17	收到的所有针对一致性请求的响应
18	收到的所有针对一致性响应并且数据为 dirty 的个数
19	收到的上一级 cache 主动替换的写回
20	收到的上一级主动替换且数据为 dirty 的个数
21	由 cached 访问导致的对 memory 的读请求个数
22	由 cached 访问导致的对 memory 的写请求个数
23	查询 scache 结果为 miss 的次数
24	查询 scache 结果为 miss 的所有 REQREAD 个数
25	查询 scache 结果为 miss 的所有 REQWRITE 个数
26	查询 scache，scache 命中但 pagecolor 不命中的个数
27	REQREAD 查询 scache，结果命中在一个 clean 块
28	REQREAD 查询 scache，结果命中在别人的 EXC 块
29	REQWRITE 查询 scache，结果命中在 clean 块
30	REQWRITE 查询 scache，结果命中在别人的 EXC 块
31	WRITE 查询 scache，结果命中在一个正在被多个核 shared 的块中
32	Cached DMAREAD 查询 scache，结果命中
33	Cached DMAWRITE 查询 scache，结果不命中
34	Cached DMAWRITE 查询 scache，结果命中，并需要对 CPU 发出 INVALIDATION
35	向 CPU 发出的一致性请求中 INV 请求的个数
36	向 CPU 发出的一致性请求中 WTBK 请求的个数
37	向 CPU 发出的一致性请求中 INVWTBK 请求的个数