

LOONGSON

龙芯 GS264 处理器核用户手册

V1.0

2018 年 7 月

龙芯中科技术有限公司

自主决定命运, 创新成就未来

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

本档版权归龙芯中科技术有限公司所有，并保留一切权利。未经书面许可，任何公司和个人不得将此文档中的任何部分公开、转载或以其他方式散发给第三方。否则，必将追究其法律责任。

免责声明

本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 GS264 处理器核用户手册》用于介绍龙芯 2K1000 芯片中所集成的 GS264 处理器核的相关软件特性。

特殊格式含义介绍

- 1、本文中涉及 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， $m \geq n$ 。

版本历史

文档更新记录	文档名	龙芯 GS264 处理器核用户手册		
	版本号	V1.0		
	创建人	研发中心芯片研发部		
	创建日期	2017-02-28		
更新历史				
序号	更新日期	更新人	版本号	更新内容
1	2017/02/28	芯片研发部	V0.01	初稿。
2	2017/04/25	芯片研发部	V0.02	<ol style="list-style-type: none"> 1) 1.1.7 小节对最大物理地址的概念进行了明确。 2) 1.1.8 小节删去了 KE 位相关的说明。 3) 1.1.11 小节调整关于 GS464 和 GS464E 的描述。 4) 2.3.4 小节调整关于低功耗模式和 CP0.Status.RP 置 1 之间容易引起歧义的描述。 5) 7.31 小节中补充 ITLB 位的说明。 6) 7.39 小节中关于 8KB 页的相关内容删去。 7) 增加了第 8 章, 对于处理器核内部的结构级时钟门控进行介绍。 8) 增加了 2.2.6 小节浮点运算实现细节说明。
3	2017/07/05	芯片研发部	V0.03	<ol style="list-style-type: none"> 1) 增加独立的“浮点指令 E 例外判定说明”小节 2) perfcnt 中 30 号事件 D-Cache 访问次数计数不准确, 删去。
4	2018/06/29	芯片研发部	V1.0	<ol style="list-style-type: none"> 1) 文档名从《龙芯 2K1000 芯片技术参考书——下册》更改为《龙芯 GS264 处理器核用户手册》 2) 将第 1 章中所有针对 2H 中 GS464 的描述替换为 3A1000 中 GS464 的描述。 3) 删除原 1.1.1 小节。 4) 原 1.1.8 小节增加了一级 ITLB 和一级 DTLB 是否需要软件维护其与二级 TLB 之间的一致性。 5) 2.3.4 小节补充了对于从 WAIT 休眠状态中恢复处理的时延的说明。 6) 更新了原 1.1.2 小节和 7.20 小节有关 PRID 的数值。 7) 删除了所有涉及“根模式”的内容, 避免读者误解 GS264 实现了硬件虚拟化。 8) 修正了 2.3.5 和 2.3.6 小节关于 SYNC 和 SYNCI 指令的描述。 9) 调整 5.4.4 小节的内容表述。

手册信息反馈: service@loongson.cn

目 录

1	快速了解 GS264	1
1.1	GS264@2K1000、GS464@3A1000、GS464E@3A2000 主要差异点概述	1
1.1.1	PRID	1
1.1.2	指令	1
1.1.3	浮点指令兼容性问题	1
1.1.4	向量指令兼容性问题	1
1.1.5	SEGBITS	2
1.1.6	最大物理地址有效位数(PABITS)	2
1.1.7	TLB	2
1.1.8	Cache	2
1.1.9	Cache Tag 的读写格式	2
1.1.10	Cache Store Tag 和 Cache Store Data 时写入的 ECC 的来源	3
1.1.11	关于 StoreFill 功能	3
2	指令集概述	5
2.1	MIPS64 兼容通用指令列表	5
2.1.1	访存指令	5
2.1.2	运算指令	6
2.1.3	跳转和分支指令	8
2.1.4	协处理器 0 指令	9
2.1.5	其它指令	9
2.2	MIPS64 兼容浮点指令集概览	10
2.2.1	FPU 数据类型	10
2.2.2	浮点寄存器	13
2.2.3	浮点控制寄存器	13
2.2.4	浮点例外	16
2.2.5	MIPS64 兼容浮点指令列表	18
2.2.6	浮点运算实现细节说明	21
2.2.7	浮点指令 E 例外判定说明	30
2.3	MIPS64 兼容指令实现相关定义	34
2.3.1	目标为 0 号通用寄存器的 load 指令	34
2.3.2	PREF 指令	34
2.3.3	PREFX 指令	34
2.3.4	WAIT 指令	34
2.3.5	SYNC 指令	34
2.3.6	SYNCI 指令	34
2.3.7	CACHE 指令	35
2.3.8	MADD.fmt、MSUB.fmt、NMADD.fmt、NMSUB.fmt 指令	35
2.3.9	EHB、SSNOP 指令	35
2.4	龙芯扩展指令集	36
2.5	SIMD 扩展指令集	44
2.5.1	MSA 兼容指令列表	44
2.5.2	MSA 兼容部分与 MIPS 规范的细节差异	44

3	处理器运行模式	47
3.1	处理器运行模式定义	47
3.1.1	调试模式	47
3.1.2	核心模式	47
3.1.3	用户模式	47
4	内存管理	49
4.1	基本概念	49
4.1.1	地址空间	49
4.1.2	段及段大小(SEGBITS)	49
4.1.3	物理地址大小(PABITS)	49
4.1.4	映射地址(Mapped Address)与非映射地址(Unmapped Address)	49
4.2	主机虚地址空间	49
4.2.1	主机地址空间划分与访问控制	49
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	51
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	51
4.2.4	主机地址空间 kuseg 段在 Status.ERL=1 时的地址转换	51
4.2.5	主机地址空间 kseg3 段在 Debug.DM=1 时的特殊处理	51
4.2.6	用户模式下 Status.UX=0 时数据访问虚地址的特殊处理	51
4.3	基于 TLB 的虚实地址映射	52
4.3.1	ITLB	52
4.3.2	TLB 组织结构	52
4.3.3	TLB 的软件管理	53
4.3.4	TLB 初始化与清空	54
4.3.5	基于 TLB 的虚实地址转换过程	54
5	缓存的组织与管理	57
5.1	处理器存储层次及各级缓存组织结构	57
5.1.1	处理器存储层次	57
5.1.2	一级指令缓存(I-Cache)	58
5.1.3	一级数据缓存(D-Cache)	59
5.1.4	二级共享缓存(S-Cache)	59
5.2	缓存算法与缓存一致属性	60
5.2.1	非缓存算法	60
5.2.2	一致性缓存算法	60
5.2.3	非缓存加速算法	60
5.3	缓存一致性	60
5.4	缓存管理	61
5.4.1	CACHE 指令	61
5.4.2	缓存初始化	62
5.4.3	一级指令缓存与一级数据缓存间的一致性维护	63
5.4.4	处理器与设备 DMA 间的缓存一致性维护	63
5.4.5	缓存别名与页着色	64
6	处理器例外与中断	65
6.1	处理器例外	65
6.1.1	例外优先级	65

6.1.2	例外入口向量位置	65
6.1.3	处理器硬件响应例外的通用处理过程.....	66
6.1.4	冷复位例外	66
6.1.5	不可屏蔽中断	67
6.1.6	中断例外	67
6.1.7	地址错例外	67
6.1.8	TLB 重填例外.....	68
6.1.9	XTLB 重填例外.....	68
6.1.10	TLB 无效例外.....	69
6.1.11	TLB 修改例外.....	69
6.1.12	TLB 执行阻止例外.....	70
6.1.13	TLB 读取阻止例外.....	70
6.1.14	Cache 错误例外.....	71
6.1.15	整型溢出例外	71
6.1.16	陷阱例外	71
6.1.17	系统调用例外	71
6.1.18	断点例外	72
6.1.19	保留指令例外	72
6.1.20	协处理器不可用例外	72
6.1.21	浮点例外	73
6.1.22	浮点栈例外	73
6.2	中断	73
6.2.1	中断响应的必要条件	73
6.2.2	中断模式	73
6.2.3	中断处理的补充说明	75
7	协处理器 0 寄存器	76
7.1	协处理器 0 寄存器概览	76
7.2	Index 寄存器 (CP0 Register 0, Select 0).....	78
7.3	Random 寄存器 (CP0 Register 1, Select 0).....	79
7.4	EntryLo0 和 EntryLo1 寄存器 (CP0 Register 2 and 3, Select 0).....	80
7.5	Context 寄存器 (CP0 Register 4, Select 0).....	83
7.6	UserLocal 寄存器 (CP0 Register 4, Select 2).....	84
7.7	PageMask 寄存器 (CP0 Register 5, Select 0).....	85
7.8	PageGrain 寄存器 (CP0 Register 5, Select 1).....	86
7.9	Wired 寄存器 (CP0 Register 6, Select 0)	87
7.10	HWREna 寄存器 (CP0 Register 7, Select 0)	88
7.11	BadVAddr 寄存器 (CP0 Register 8, Select 0)	89
7.12	Count 寄存器 (CP0 Register 9, Select 0).....	90
7.13	EntryHi 寄存器 (CP0 Register 10, Select 0)	91
7.14	Compare 寄存器 (CP0 Register 11, Select 0).....	92
7.15	Status 寄存器 (CP0 Register 12, Select 0)	93
7.16	IntCtl 寄存器 (CP0 Register 12, Select 1).....	95
7.17	SRSCtl 寄存器 (CP0 Register 12, Select 2)	96
7.18	Cause 寄存器 (CP0 Register 13, Select 0).....	97

7.19	EPC 寄存器 (CP0 Register 14, Select 0)	99
7.20	PRId 寄存器 (CP0 Register 15, Select 0)	100
7.21	EBase 寄存器 (CP0 Register 15, Select 1).....	101
7.22	Config 寄存器 (CP0 Register 16, Select 0).....	102
7.23	Config1 寄存器 (CP0 Register 16, Select 1).....	103
7.24	Config2 寄存器 (CP0 Register 16, Select 2).....	104
7.25	Config3 寄存器 (CP0 Register 16, Select 3).....	105
7.26	Config4 寄存器 (CP0 Register 16, Select 4).....	107
7.27	Config5 寄存器 (CP0 Register 16, Select 5).....	108
7.28	GSConfig 寄存器 (CP0 Register 16, Select 6).....	109
7.29	LLAddr 寄存器 (CP0 Register 17, Select 0)	110
7.30	XContext 寄存器 (CP0 Register 20, Select 0).....	111
7.31	Diag 寄存器 (CP0 Register 22, Select 0)	112
7.32	GSCause 寄存器 (CP0 Register 22, Select 1)	113
7.33	Debug 寄存器 (CP0 Register 23, Select 0).....	114
7.34	DEPC 寄存器 (CP0 Register 24, Select 0).....	115
7.35	PerfCnt 寄存器 (CP0 Register 25, Select 0~7)	116
7.36	ErrCtl 寄存器 (CP0 Register 26, Select 0)	120
7.37	CacheErr 寄存器 (CP0 Register 27, Select 0).....	121
7.38	CacheErr1 寄存器 (CP0 Register 27, Select 1).....	122
7.39	TagLo 寄存器 (CP0 Register 28, Select 0)	123
7.40	DataLo 寄存器 (CP0 Register 28, Select 1)	125
7.41	TagHi 寄存器 (CP0 Register 29, Select 0).....	126
7.42	DataHi 寄存器 (CP0 Register 29, Select 1)	127
7.43	ErrorEPC 寄存器 (CP0 Register 30, Select 0)	128
7.44	DESAVE 寄存器 (CP0 Register 31, Select 0).....	129
7.45	KScratch1~2 寄存器 (CP0 Register 31, Select 2~3).....	130

图目录

图 2-1 FPU 浮点数据格式.....	11
图 2-2 FPU 定点数据格式.....	12
图 2-3 FIR 寄存器格式.....	13
图 2-4 FCSR 寄存器格式.....	14
图 2-5 FCCR 寄存器格式.....	15
图 2-6 FEXR 寄存器格式.....	15
图 2-7 FENR 寄存器格式.....	15
图 2-8 Index 类 CACHE 指令的地址解析格式.....	35
图 4-1 xkphys 段虚地址解析方式.....	51
图 5-1 龙芯 2K1000 芯片处理器存储层次.....	57
图 5-2 一致性协议下缓存状态转换.....	61
图 7-1 Index 寄存器格式.....	78
图 7-2 Random 寄存器格式.....	79
图 7-3 EntryLo0 和 EntryLo1 在 DMFC0/DMTC0 指令访问时的寄存器格式.....	80
图 7-4 EntryLo0 和 EntryLo1 在 MFC0/MTC0 指令访问时的寄存器格式.....	81
图 7-5 Context 寄存器格式.....	83
图 7-6 UserLocal 寄存器格式.....	84
图 7-7 PageMask 寄存器格式.....	85
图 7-8 PageGrain 寄存器格式.....	86
图 7-9 TLB 中固定表项与随机替换表项边界.....	87
图 7-10 Wired 寄存器格式.....	87
图 7-11 HWREna 寄存器格式.....	88
图 7-12 BadVAddr 寄存器格式.....	89
图 7-13 Count 寄存器格式.....	90
图 7-14 EntryHi 寄存器格式.....	91
图 7-15 Compare 寄存器格式.....	92
图 7-16 Status 寄存器格式.....	93
图 7-17 IntCtl 寄存器格式.....	95
图 7-18 SRSCtl 寄存器格式.....	96
图 7-19 Cause 寄存器格式.....	97
图 7-20 EPC 寄存器格式.....	99
图 7-21 PRId 寄存器格式.....	100
图 7-22 EBase 寄存器格式.....	101
图 7-23 Config 寄存器格式.....	102
图 7-24 Config1 寄存器格式.....	103
图 7-25 Config2 寄存器格式.....	104
图 7-26 Config3 寄存器格式.....	105
图 7-27 Config4 寄存器格式.....	107
图 7-28 Config5 寄存器格式.....	108
图 7-29 GSConfig 寄存器格式.....	109
图 7-30 LLAddr 寄存器格式.....	110
图 7-31 XContext 寄存器格式.....	111

图 7-32 Diag 寄存器格式	112
图 7-33 GSCause 寄存器格式	113
图 7-34 DEPC 寄存器格式	115
图 7-35 PerfCnt Control 寄存器格式	116
图 7-36 PerfCnt Counter 寄存器格式	117
图 7-37 ErrCtl 寄存器格式	120
图 7-38 CacheErr 寄存器用于 I-Cache 校验错信息时的格式	121
图 7-39 CacheErr 寄存器用于 D-Cache 校验错信息时的格式	121
图 7-40 CacheErr1 寄存器格式	122
图 7-41 TagLo 寄存器用于访问 I-Cache Tag 时的格式	123
图 7-42 TagLo 寄存器用于访问 D-Cache Tag 时的格式	123
图 7-43 TagLo 寄存器用于访问 S-Cache Tag 时的格式	124
图 7-44 TagLo 寄存器用于访问各级 Cache Data 时的格式	124
图 7-45 DataLo 寄存器用于 I-Cache 访问时的格式	125
图 7-46 TagHi 寄存器用于访问各级 Cache Tag 时的格式	126
图 7-47 TagHi 寄存器用于访问各级 Cache Data 时的格式	126
图 7-48 DataHi 寄存器用于 I-Cache 访问时的格式	127
图 7-49 ErrorEPC 寄存器格式	128
图 7-50 DESAVE 寄存器格式	129
图 7-51 KScratchn 寄存器格式	130

表目录

表 2-1 CPU 指令集：访存指令	5
表 2-2 运算指令：算术指令（ALU 立即数）	6
表 2-3 运算指令：算术指令（3 操作数）	6
表 2-4 运算指令：算术指令（2 操作数）	7
表 2-5 运算指令：乘、除法指令	7
表 2-6 运算指令：移位指令	8
表 2-7 跳转和分支指令	8
表 2-8 协处理器 0 指令	9
表 2-9 其它指令：特殊指令	9
表 2-10 其它指令：例外陷入指令	10
表 2-11 其它指令：条件移动指令	10
表 2-12 其它指令：空操作	10
表 2-13 浮点数格式相关参数	11
表 2-14 浮点数数值 V 的计算方式	12
表 2-15 浮点数最大值和最小值	12
表 2-16 FIR 寄存器域描述	13
表 2-17 FCSR 寄存器域描述	14
表 2-18 舍入模式(RM)编码	15
表 2-19 浮点例外的默认处理	16
表 2-20 浮点分支跳转指令	19
表 2-21 浮点运算指令	19
表 2-22 浮点分支跳转指令	19
表 2-23 浮点分支跳转指令	20
表 2-24 浮点分支跳转指令	20
表 2-25 浮点分支跳转指令	20
表 2-26 龙芯扩展访存类指令	36
表 2-27 龙芯扩展算术与逻辑运算指令	37
表 2-28 龙芯扩展 X86 二进制翻译加速指令	38
表 2-29 龙芯扩展 ARM 二进制翻译加速指令	41
表 2-30 龙芯扩展 64 位多媒体加速指令	42
表 2-31 龙芯扩展杂项指令	44
表 3-1 处理器模式判定依据	47
表 4-1 主机地址空间划分与访问控制	49
表 4-2 TLB 管理相关 CPO 寄存器	53
表 4-3 TLB 管理相关特权指令	54
表 5-1 缓存参数	58
表 5-2 CACHE 指令	61
表 6-1 例外优先级	65
表 6-2 例外向量基址	66
表 6-3 例外向量偏移	66
表 6-4 兼容中断模式下各中断请求生成	74
表 6-5 向量中断模式下各中断间优先级关系	74

表 6-6 中断模式判定	75
表 7-1 根协处理器 0 寄存器一览表	76
表 7-2 Index 寄存器域描述	78
表 7-3 Random 寄存器域描述	79
表 7-4 EntryLo0 和 EntryLo1 在 DMFC0/DMTC0 指令访问时的寄存器域描述	80
表 7-5 EntryLo0 和 EntryLo1 在 MFC0/MTC0 指令访问时的寄存器域描述	81
表 7-6 Cache 属性编码表	82
表 7-7 Context 寄存器域描述	83
表 7-8 UserLocal 寄存器域描述	84
表 7-9 PageMask 寄存器域描述	85
表 7-10 Mask 域编码与页大小	85
表 7-11 PageGrain 寄存器域描述	86
表 7-12 Wired 寄存器域描述	87
表 7-13 HWREna 寄存器域描述	88
表 7-14 BadVAddr 寄存器域描述	89
表 7-15 Count 寄存器域描述	90
表 7-16 EntryHi 寄存器域描述	91
表 7-17 Compare 寄存器域描述	92
表 7-18 Status 寄存器域描述	93
表 7-19 IntCtl 寄存器域描述	95
表 7-20 SRSCtl 寄存器域描述	96
表 7-21 Cause 寄存器域描述	97
表 7-22 ExcCode 编码及其对应例外类型	97
表 7-23 EPC 寄存器域描述	99
表 7-24 PRId 寄存器域描述	100
表 7-25 EBase 寄存器域描述	101
表 7-26 Config 寄存器域描述	102
表 7-27 Config1 寄存器域描述	103
表 7-28 Config2 寄存器域描述	104
表 7-29 Config3 寄存器域描述	105
表 7-30 Config4 寄存器域描述	107
表 7-31 Config5 寄存器域描述	108
表 7-32 GSConfig 寄存器域描述	109
表 7-33 LLAddr 寄存器域描述	110
表 7-34 XContext 寄存器域描述	111
表 7-35 Diag 寄存器域描述	112
表 7-36 GSCause 寄存器域描述	113
表 7-37 GSExcCode 编码及其对应例外类型	113
表 7-38 DEPC 寄存器域描述	115
表 7-39 PerfCnt 寄存器 Select 分配	116
表 7-40 PerfCnt Control 寄存器域描述	116
表 7-41 PerfCnt Counter 寄存器域描述	117
表 7-42 处理器核性能计数器事件定义	117
表 7-43 ErrCtl 寄存器域描述	120

表 7-44 CacheErr 寄存器用于 I-Cache 校验错信息时的域描述.....	121
表 7-45 CacheErr 寄存器用于 D-Cache 校验错信息时的域描述	121
表 7-46 CacheErr1 寄存器域描述.....	122
表 7-47 TagLo 寄存器用于访问 I-Cache Tag 时的域描述.....	123
表 7-48 TagLo 寄存器用于访问 D-Cache Tag 时的域描述	123
表 7-49 TagLo 寄存器用于访问 S-Cache Tag 时的域描述.....	124
表 7-50 TagLo 寄存器用于访问各级 Cache Data 时的域描述.....	124
表 7-51 DataLo 寄存器用于 I-Cache 访问时的域描述.....	125
表 7-52 TagHi 寄存器用于访问各级 Cache Tag 时的域描述	126
表 7-53 TagHi 寄存器用于访问各级 Cache Data 时的域描述	126
表 7-54 DataHi 寄存器用于 I-Cache 访问时的域描述	127
表 7-55 ErrorEPC 寄存器域描述.....	128
表 7-56 DESAVE 寄存器域描述.....	129
表 7-57 KScratchn 寄存器域描述	130

1 快速了解 GS264

本章是写给那些对龙芯软件特性比较熟悉的软件人员的。通过对比的方式，将 GS264 与龙芯 3A1000 中的 GS464、与龙芯 3A2000¹/3A3000 中的 GS464E 的差异标明出来，方便软件人员进行软件移植。

先提供一个总体的印象，GS264 在用户指令上像 GS464E 多一些，删去了 DSP，增加了 128 位 SIMD；在特权态部分上像 GS464 更多，例如 TLB 的组织形式、performance counter 访问形式、counter，尤其是没有 GS464E 里面的虚拟化支持部分。

1.1 GS264@2K1000、GS464@3A1000、GS464E@3A2000 主要差异点

概述

1.1.1 PRID

GS264@2K1000	0x00146101、0x00146103
GS464@3A1000	0x00006303、0x00006305
GS464E@3A2000	0x00146308、0x0014630C
GS464E@3A3000	0x00146309、0x0014630D

1.1.2 指令

GS264@2K1000	MIPS64R3、LoongEXT64、SIMD 指令
GS464@3A1000	MIPS64R3、自定义通用指令（如 lq、sq、MMX）、CAM 指令
GS464E@3A2000	MIPS64R3、DSP 指令、LoongEXT64、CAM 指令、虚拟机扩展指令

DSP 指令、CAM 指令、SIMD 指令、虚拟机扩展指令都可以通过 config~config5 寄存器和 gsconfig 的相关标志位进行判定。

但是，GS264 和 GS464E 实现的 LoongEXT64 指令并不完全一致。GS464E 中的 LoongEXT64 包含 SETMEM、LWDIR、LWPTE、LDDIR、LDPTE 这 5 条指令，但是 GS264 并没有支持。

GS264 没有 DSP 指令，但是这其实对应用来说没有什么影响，因为 MIPS 的 DSP 指令的加速效果实在有限，我们推荐使用 SIMD 指令进行相关应用的加速。

1.1.3 浮点指令兼容性问题

GS264@2K1000 中的浮点运算部件完全沿用 GS464E@3A2000 中的版本，所以在一些边界情况下与 MIPS 规范并不一致。如有需要请阅读 2.2.6 节“[浮点运算实现细节](#)”了解详细情况。

1.1.4 向量指令兼容性问题

GS264@2K1000 中的向量指令的 MIPS 部分并未进行过 MIPS 官方的兼容性测试，所以可能存在兼容性问题。如果发现 2K1000 上运行结果与同样程序在 MIPS 平台上不一致，可以告知我们。

¹ 本章后面的内容，3A2000 和 3A3000 没有区别，所以后面只写 3A2000，3A3000 的其实是一样的。

1.1.5 SEGBITS

GS264@2K1000	40 位
GS464@3A1000	48 位
GS464E@3A2000	48 位

1.1.6 最大物理地址有效位数(PABITS)

GS264@2K1000	40 位
GS464@3A1000	48 位
GS464E@3A2000	48 位

1.1.7 TLB

GS264@2K1000	64 项 CAM, 支持最大页为 1GB, 支持 RI、XI, 没有 EHINV 位
	有一级 ITLB, 需要软件维护其与 TLB 的一致性
	无一级 DTLB
GS464@3A1000	64 项 CAM, 支持最大页为 16MB, 不支持 RI、XI, 没有 EHINV 位
	有一级 ITLB, 需要软件维护其与 TLB 的一致性
	无一级 DTLB
GS464E@3A2000	64 项 CAM (VTLB) +1024 项 Cache (FTLB), 支持最大页为 1GB, 支持 RI、XI, 有 EHINV 位
	有一级 ITLB, 需要软件维护其与 TLB 的一致性
	有一级 DTLB, 由硬件维护其与 TLB 的一致性

1.1.8 Cache

GS264@2K1000	有一级指令 Cache, 8KB×4 路, 每行 64 字节, 随机替换
	有一级数据 Cache, 8KB×4 路, 每行 64 字节, 随机替换
	没有私有二级 Cache
	有共享二级 Cache, 每个体 64KB×8 路, 每行 64 字节, LRU 替换 2K1000 中二级 Cache 有 2 个体, 地址固定按 Cache 行交织
GS464@3A1000	有一级指令 Cache, 16KB×4 路, 每行 32 字节, 随机替换
	有一级数据 Cache, 16KB×4 路, 每行 32 字节, 随机替换
	没有私有二级 Cache
	有共享二级 Cache, 每个体 128KB×4 路, 每行 32 字节, 随机替换 3A1000 中二级 Cache 只有 1 个体, 无所谓 Cache 体的地址映射问题
GS464E@3A2000	有一级指令 Cache, 16KB×4 路, 每行 64 字节, 随机替换
	有一级数据 Cache, 16KB×4 路, 每行 64 字节, LRU 替换
	有私有二级 Cache, 16KB×16 路, 每行 64 字节, LRU 替换
	有共享三级 Cache, 每个体 64KB×16 路, 每行 64 字节, LRU 替换 3A2000 中三级 Cache 有 4 个体, Cache 体的地址映射可软件动态配置

1.1.9 Cache Tag 的读写格式

GS264@2K1000 的 Cache Tag 的读写格式与 GS464E@3A2000 的读写格式基本是一样的, 只是因为两

者支持的物理地址位数不一样多导致在 ptag 的长度上有差异。具体情况请看 7.39 节 TagLo 寄存器 (CP0 Register28, Select 0)和 7.41 节 TagHi 寄存器 (CP0 Register 29, Select 0)中的说明。

1.1.10 Cache Store Tag 和 Cache Store Data 时写入的 ECC 的来源

GS264@2K1000	软件无需预先计算，由硬件自动计算并填入
GS464@3A1000	对于 ICache 和 Dcache，软件需预先计算好并写入 CP0.ErrCtl 寄存器的 ECC 域
GS464E@3A2000	对于 ICache、Dcache 和 Vcache，软件需预先计算好并写入 CP0.ErrCtl 寄存器的 ECC 域

1.1.11 关于 StoreFill 功能

GS264@2K1000	有此功能，软件可通过 CP0.GSCONFIG[8]开启/关闭
GS464@3A1000	无此功能
GS464E@3A2000	有此功能，软件可通过 CP0.GSCONFIG[8]开启/关闭

2 指令集概述

GS264 实现了龙芯指令集 (LoongISA) v1.00 版中的 MIPS64 兼容基础部分、64 位龙芯通用扩展指令 (LoongEXT64) 和龙芯 128 位向量扩展指令 (LoongSIMD128)。有关指令的详细定义请参看《MIPS® Architecture For Programmers Volume II-A: The MIPS64® Instruction Set》(r5.03)。

2.1 MIPS64 兼容通用指令列表

GS264 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

指令助记符	指令功能简述	ISA 兼容类别
SWR	存字右部	MIPS32
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	可变的字逻辑左移	MIPS64
DSRLV	可变的字逻辑右移	MIPS64
DSRAV	可变的字算术右移	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

指令助记符	指令功能简述	ISA 兼容类别
BNE	不等则跳转	MIPS32
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

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
SYNC	同步	MIPS32
SYNCHI	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

GS264 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(Floating Point Unit, 简称 FPU)中实现。

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

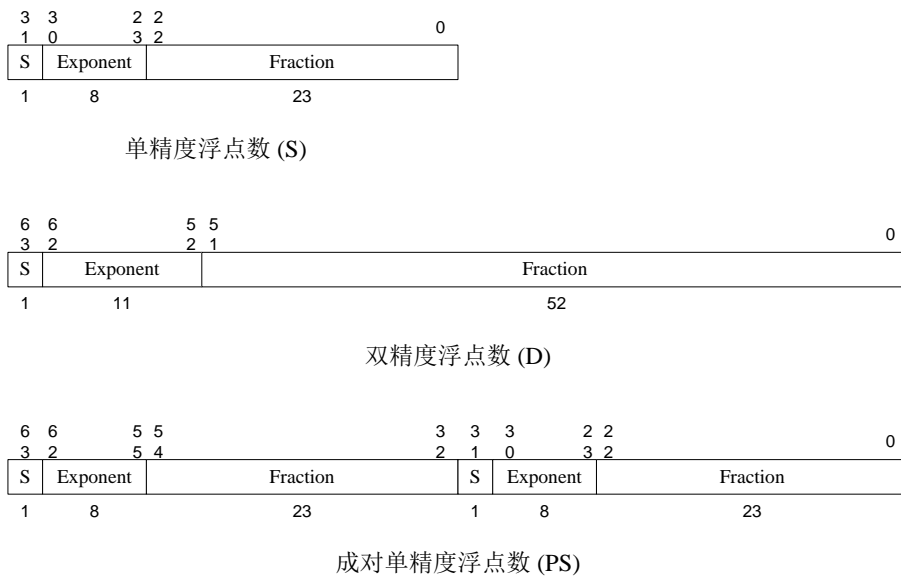
浮点协处理器支持的浮点数据类型有：

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。

- 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



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, $e = E + \text{Bias}$, E 是不带偏移的指数
- 小数域, $F = .b_1b_2\dots b_{p-1}$

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

- $E_{\min} - 1$ (用来编码 0 和非规格化数)
- $E_{\max} + 1$ (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 浮点数据格式相关参数

参数	单精度	双精度
E_{\max}	+127	+1023
E_{\min}	-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	
$E_{\max}+1$	$\neq 0$	x	1	SNaN(Signaling NaN)	
		x	0	QNaN(Quiet NaN)	
$E_{\max}+1$	0	1	x	$-\infty$	负无穷
		0	x	$+\infty$	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
$E_{\min}-1$	$\neq 0$	1	x	$-(2^{E_{\min}-1})(0.F)$	负非规格化数
		0	x	$+(2^{E_{\min}-1})(0.F)$	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

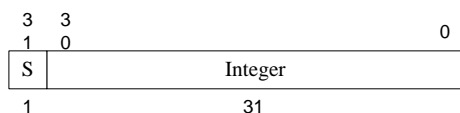
定点数据类型

FPU 支持的定点数据均为有符号整数，分为两种：

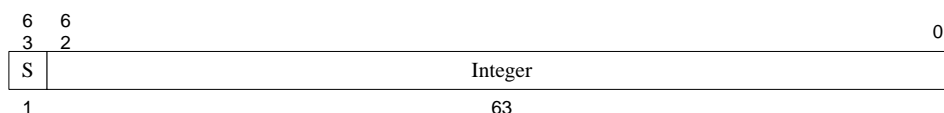
- 32 比特的有符号字整型 (Word, W)
- 64 比特的有符号长字整型 (Longword, L)

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

GS264 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，GS264 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，GS264 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

GS264 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

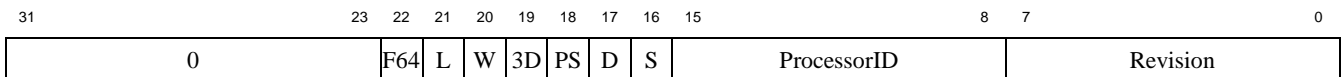


表 2-16 FIR 寄存器域描述

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
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

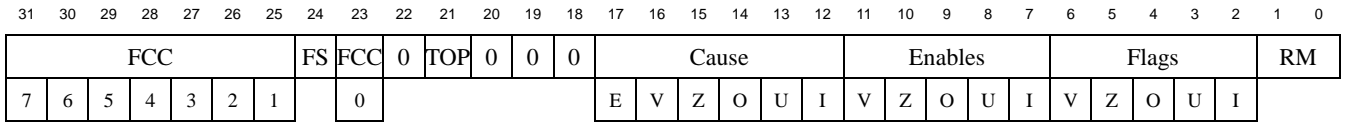


表 2-17 FCSR 寄存器域描述

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
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	0x1f

域名称	位	功能描述	读/写	复位值
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

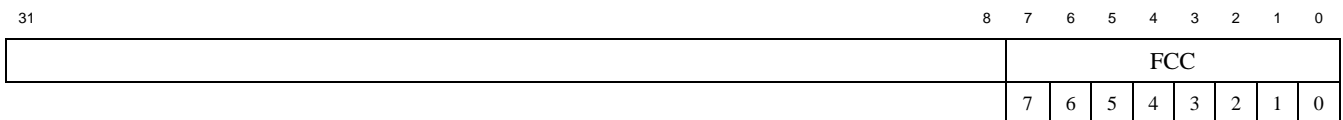
表 2-18 舍入模式(RM)编码

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。

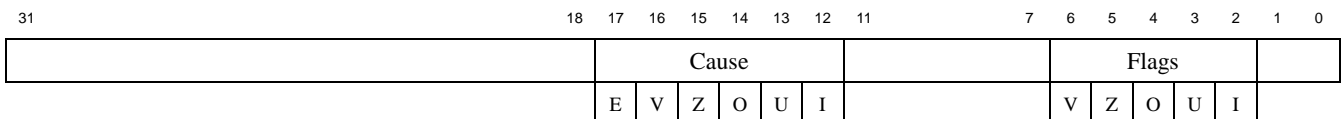
图 2-5 FCCR 寄存器格式



浮点例外寄存器(FEXR, 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。

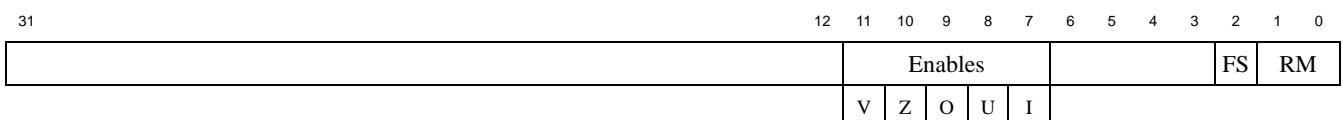
图 2-6 FEXR 寄存器格式



浮点使能寄存器(FENR, 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：如果一个不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： $(+\infty)+(-\infty)$ 或者 $(-\infty)-(-\infty)$
- 乘法： $0 \times \infty$ ，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： $\ln(0)$ ， $\sin(\pi/2)$ ， $\cos(0)$ ，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 $\pm 2E_{min}$ 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 $\pm 2E_{min}$ 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 $\pm 2E_{min}$ 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

GS264 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
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	

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
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
MOV.f.fmt	浮点假时复制 FPR	✓	✓	✓	-	-	MIPS32
MOVN.fmt	GPR 不为 0 时复制 FPR	✓	✓	✓	-	-	MIPS32
MOV.f.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

2.2.6 浮点运算实现细节说明

GS264 上的浮点部件主要在浮点例外判定条件方面与 MIPS 规范或是 MIPS 已经 release 的 IP 存在不一致，所以本小节按照指令功能逐一列举 GS264 上浮点部件的例外判定条件。

浮点转换为定点 (F2I)

例外	判定条件	缺省结果
E	输入是非规格化数 (subnormal)	不关注其结果
V	(1) 输入是 inf (2) 输入是 SNaN (3) 输入是 QNaN (4) 如果是转换为无符号字(W), round 后值超出 $[2^{32}-1, 0]$ 的; 如果是转换为无符号双字(L), round 后值超出 $[2^{64}-1, 0]$ 的; 如果是转换为有符号字(W), round 后值超出 $[2^{31}-1, -2^{31}]$; 如果是转换为有符号双字(L), round 后值超出 $[2^{63}-1, -2^{63}]$ 。	如果是浮点转为无符号整数: (1) input 为负数, 缺省结果为 0; (2) input 为 nan, 缺省结果为 0; (3) input 为正数且不是 nan 时, 缺省结果为最大无符号正整数。 如果是浮点转为有符号整数: 缺省结果总是最大有符号正整数。
Z	无	-
O	无	-
U	无	-
I	输入不是 subnormal, snan, qnan, inf, 且不报 V, 且 round 之后的结果不 exact 时	缺省结果取 round 之后的值即可。

注意:

1. F2I 要么报 E, 要么报 V, 要么报 I, 不会同时报其中的两个或者三个。
2. F2I 操作中没有涉及到 fs, 这是因为 fs 只对 subnormal 结果起作用, 而 F2I 的结果是 integer, 并没有 subnormal 的结果出现

定点转浮点 (I2F)

例外	判定条件	缺省结果
E	无	-
V	无	-
Z	无	-
O	无	-
U	无	-
I	当转换成浮点以后的结果（round 之后的）和输入的整数值不相等的时候。	根据 rounding 模式 round 后的值

注意：

1. fs 对 I2F 没有影响，这是因为输入是整数，结果不可能是 tiny 非 0 值。

浮点加法、乘法、乘加、乘减、负乘加、负乘减（FMAF）

为了描述简便，FMAF 各种操作这样表示： $a+c$ 、 $a-c$ 、 $a\times b$ 、 $a\times b+c$ 、 $a\times b-c$ 、 $-(a\times b+c)$ 、 $-(a\times b-c)$ 。

例外	判定条件	缺省结果
E	<p>(1) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，a 或 b 或 c 是 QNaN 时。</p> <p>(2) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，c 是 subnormal 时。</p> <p>(3) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，a 是 subnormal，b 不是 0 或 ∞ 时。</p> <p>(4) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，b 是 subnormal，a 不是 0 或 ∞ 时。</p> <p>(5) 在不满足 E 例外判定条件 (1) ~ (4)、V 例外判定条件 (1) ~ (4) 的前提下，fs=0 或 enable_U=1 或 enable_I=1 时，出现了 exponent_underflow 的情况(round 后的结果的值在$(-2^{E_{\min}}, 2^{E_{\min}})$ 之间)，且结果不是 0。</p>	
V	<p>(1) a 或 b 或 c 是 SNaN。</p> <p>(2) $a\times b$ 为 $0\times\infty$ 或 $\infty\times 0$ 时。</p> <p>(3) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，a 是 ∞, b 不是 0 或 qnan，(也就是 $a*b=\infty$)，c 是 ∞ 时，若是 $(-\infty) + (+\infty)$，$(-\infty) - (-\infty)$，$(+\infty) + (-\infty)$，$(+\infty) - (+\infty)$。</p> <p>(4) 在不满足 V 例外判定条件 (1) 和 (2) 的前提下，b 是 ∞, a 不是 0 或 qnan，(也就是 $a*b=\infty$)，c 是 ∞ 时，若是 $(-\infty) + (+\infty)$，$(-\infty) - (-\infty)$，$(+\infty) + (-\infty)$，$(+\infty) - (+\infty)$。</p>	<p>当精度为 Double 时，缺省结果是 0x7ff7_ffff_ffff_ffff；</p> <p>当精度为 Single 时，缺省结果是 0x7fbf_ffff。</p>
Z	无	-

例外	判定条件	缺省结果
O	round 后的结果的绝对值（看成 exponent 无界，precision 有界）超过了目标格式能表达的最大有限数，且不是 inf_fmaf*时。	根据 rounding mode: (1) 如果是 round_to_zero、结果为负且 round_to_+inf 或者结果为正且 round_to_-inf 时，当结果为负时，则缺省结果为目标格式能表示的最小有限数；当结果为正时，则缺省结果为目标格式能表示的最大有限数。 (2) 不在(1)情况中的，当结果为正时，缺省结果为+∞；当结果为负时，缺省结果为-∞。
U	fs=1 且 enable_U=0 且 enable_I=0 时，出现了 exponent_underflow 的情况(结果不是 0)。	根据 rounding mode: (1) 如果是结果为负且 round_to_-inf 或结果为正且 round_to_+inf，则缺省结果是目标格式能表示的绝对值最小的有限数，符号与结果一致。 (2) 不在(1)情况中的，当结果为正时，缺省结果为+0；当结果为负时，缺省结果为-0。
I	(1)fs=1 且 enable_U=0 且 enable_I=0 时，出现了 exponent_underflow 的情况(结果不是 0)。 (2) overflow 时。 (3) round 后的结果与 round 前的值不相等，且不是 underflow，不是 inf_fmaf*时。	如果是由于 O 或者 U 引起的，则取根据 U 和 I 的 default 处理得到的值； 如果是由于其它情况产生的 Inexact，则取 round 后的值。

* inf_fmaf 指的是：c 是 ∞，或者(a 是 ∞,b 不是 0、qnan、snan)，或者 (b 是 ∞,a 不是 0、nan、snan)，并且都不是 (-∞) + (+∞)，(-∞) - (-∞)，(+∞) + (-∞)，(+∞) - (+∞)。也就是有效的带 ∞ 的操作，最后得到 -∞ 或者 +∞ 的结果。

注意：

1. FMAF 中的各种操作的各种例外，要么报 V，要么报 E，要么报 O(和 I)，要么报 U (和 I)，要么报 I，不会多个一块儿报出来。可以看出来，应该是 V 的优先级比 E 高。
2. 当 fs=1 且 enable_U=0 且 enable_I=0 时，出现了 exponent_underflow 的情况(结果不是 0)而报 U 和 I 例外时，结果是根据 rounding mode 得到的值，有可能不是+0/-0。这里并没有严格遵循 MIPS 规范中当 fs=1 时，就把 subnormal 结果或者极小非 0 值变为 0 的规定。

浮点比较

例外	判定条件	缺省结果
E	无	-

例外	判定条件	缺省结果
V	如果是 quiet compare, 当输入有 snan 时, 报 V; 如果是 signaling compare, 当输入中有 nan 时(不管是 snan 还是 qnan), 都报 V。	-
Z	无	-
O	无	-
U	无	-
I	无	-

注意:

1. MIPS 规范中 C.cond.fmt 可以报 E, 但我们的实现中并没有情况报 E 的。即使输入中有 subnormal 的数时, 还是正常根据 subnormal 的值进行比较。

除法

例外	判定条件	缺省结果								
E	(1) 输入有任一个是 subnormal。 (2) 输入有任一个是 QNaN。 (3) 当报 U 时, fs=0 或 enable_U=1 或 enable_I=1。	QNaN 7fbf...f/7ff7ff...f								
V	(1) 输入有任一个是 SNaN。 (2) 被除数、除数都是 0。 (3) 被除数、除数都是 inf。	QNaN 7fbf...f/7ff7ff...f								
Z	被除数非 0 且不为 inf、QNaN、SNaN, 除数为 0。	缺省结果取相应符号的 inf。								
O	在被除数不是 inf、snan、qnan, 除数不是 0 的情况下, 单精度结果的 $\text{exp} \geq 255$ 或者双精度结果的 $\text{exp} \geq 2047$ 。	根据 Round 模式: <table border="1"> <tr> <td>Nearest</td> <td>+inf/-inf</td> </tr> <tr> <td>Zero</td> <td>目标格式可以表示的绝对值最大的有限数</td> </tr> <tr> <td>+Inf</td> <td>正数: +inf 负数: 绝对值最大的有限数 (负的)</td> </tr> <tr> <td>-Inf</td> <td>正数: 最大的有限数 负数: -inf</td> </tr> </table>	Nearest	+inf/-inf	Zero	目标格式可以表示的绝对值最大的有限数	+Inf	正数: +inf 负数: 绝对值最大的有限数 (负的)	-Inf	正数: 最大的有限数 负数: -inf
Nearest	+inf/-inf									
Zero	目标格式可以表示的绝对值最大的有限数									
+Inf	正数: +inf 负数: 绝对值最大的有限数 (负的)									
-Inf	正数: 最大的有限数 负数: -inf									
U	在除数不是 inf、snan、qnan, 被除数不是 0 的情况下, 结果的 $\text{exp} \leq 0$ 。	根据 Round 模式: <table border="1"> <tr> <td>Nearest</td> <td>+0/-0</td> </tr> <tr> <td>Zero</td> <td>+0/-0</td> </tr> <tr> <td>+Inf</td> <td>正数: 正的可表示的最小的数 (偏移指数为 1) 负数: -0</td> </tr> <tr> <td>-Inf</td> <td>正数: +0 负数: 绝对值最小的有限数, 负的 (偏移指数为 1)</td> </tr> </table>	Nearest	+0/-0	Zero	+0/-0	+Inf	正数: 正的可表示的最小的数 (偏移指数为 1) 负数: -0	-Inf	正数: +0 负数: 绝对值最小的有限数, 负的 (偏移指数为 1)
Nearest	+0/-0									
Zero	+0/-0									
+Inf	正数: 正的可表示的最小的数 (偏移指数为 1) 负数: -0									
-Inf	正数: +0 负数: 绝对值最小的有限数, 负的 (偏移指数为 1)									

例外	判定条件	缺省结果
I	在被除数不是 inf、0，除数不是 inf、0 的情况下，产生： (1) Overflow； (2) Underflow 且 enables_U=0 且 enable_I=0 且 fs=1； (3) 最终得到的结果与无穷精度的结果不一样。	

注意：

1. 当 fs=1 且出现结果的 exp=0 时，不报 E!（不管此时是否有 E 的（1）-（3）条件满足，都是不报 E 的）。也就是出现 subnormal 结果时一定会报 U，若是此时 enable_U=1 或 enable_I=1，按 E(5)来说，是要报 E 的，但是此时就是不报 E！
2. 被除数是 inf，且不满足 E(1)-(3)，不报 V 时，结果为+inf/-inf。
3. 被除数是 0 或者除数是 inf，且不满足 E(1)-(3),不报 V，不报 Z 时，结果为+0/-0。
4. 除法在第一拍进行检测从而不用进行后续运算直接结束的条件有：E(1)-(3)，V，Z，0，U，被除数是 0，除数是 inf，被除数是 inf。

浮点开方

例外	判定条件	缺省结果
E	(1) 输入是 subnormal。 (2) 输入是 QNaN	QNaN 7ff7fff...fff/7fbff..ff
V	(1) 输入是 SNaN。 (2) 符号为负（非 0），包括 -inf。	QNaN 7ff7fff...fff/7fbff..ff
Z	无	-
O	无	-
U	无	-
I	无穷精度结果值和最终的结果值不相等的时候。	

开方后求倒数

总的原则是：当开方报了 E 或 V 时，最后结果就报 E 或 V，结果为 QNaN。当开方没有报 E 或 V 时，除法是直接结束的，则结果的例外和除法一样，值也是除法处理后的值。当开方没有报 E 或 V 时，除法没有直接结束，最后的结果是除法的例外 | 开方的例外。

需要注意的有：对于开方后求倒数的操作，其中的 O，U，Z 都一定是由于算除法引起的。当出现了 O/U 时，此时的 I 也一定是由除法引起的，和之前开方的 I 没有关系。只有 E,V,Z,O,U 都不出现时，此时的 I 才看开方的 I 以及除法的 I。

log2

例外	判定条件	缺省结果
E	(1) fs=0 且输入是 subnormal。 (2) 输入是 QNaN。	
V	输入是 SNaN。	QNaN，符号与输入一致。

例外	判定条件	缺省结果
Z	(1) 输入是 0。 (2) fs=1 且输入是 subnormal。	$-\infty$
O	无	
U	无	
I	无	

注意：

1. $\log_2(+\infty/-\infty)=+\infty$

exp2

例外	判定条件	缺省结果								
E	(1) 输入是 subnormal。 (2) 输入是 QNaN。 (3) (fs=0 或 enable_U=1 或 enable_I=1)且出现了 underflow，详见 U 的判定条件 (2)。									
V	输入是 SNaN。									
Z	无									
O	输入不是 nan、0、inf、subnormal，且结果超过了目标格式可以表示的最大有限数。	根据 Round 模式： <table border="1" style="margin-left: 20px;"> <tr> <td>Nearest</td> <td>+inf/-inf</td> </tr> <tr> <td>Zero</td> <td>目标格式可以表示的绝对值最大的有限数</td> </tr> <tr> <td>+Inf</td> <td>正数：+inf 负数：绝对值最大的有限数（负的）</td> </tr> <tr> <td>-Inf</td> <td>正数：最大的有限数 负数：-inf</td> </tr> </table>	Nearest	+inf/-inf	Zero	目标格式可以表示的绝对值最大的有限数	+Inf	正数：+inf 负数：绝对值最大的有限数（负的）	-Inf	正数：最大的有限数 负数：-inf
Nearest	+inf/-inf									
Zero	目标格式可以表示的绝对值最大的有限数									
+Inf	正数：+inf 负数：绝对值最大的有限数（负的）									
-Inf	正数：最大的有限数 负数：-inf									
U	(1) 输入不是 nan、0、inf、subnormal， fs=1 且 rounding 之前的结果是极小非 0 值（也就是把 rounding 前的结果看成是无限精度、指数范围也是无限的，这个结果在 $(-2^{E_{\min}}, 2^{E_{\min}})$ 之间）。 (2) 输入不是 nan、0、inf、subnormal， fs=0 且 rounding 后的结果是极小非 0 值（也就是 rounding 后的结果，看成是精度有限、指数范围无限的，在 $(-2^{E_{\min}}, 2^{E_{\min}})$ 之间）。	如果是条件 (1) 引起的，结果是 +0/-0。 条件 (2) 会触发 E 例外。								
I	(1) 同 U 的判定条件 (1)。 (2) 同 O 的判定条件。 (3) underflow 且 enable_U=0 且 enable_I=0。									

注意：

1. exp2 可能同时报 E, U, I。
2. 输入是 inf，不报例外，结果为相应符号的 inf。
3. 输入是 0，不报例外，结果为相应符号的 0。
4. 出现了 U(1)的情况时，结果为相应符号的 0。

精度提升的浮点转浮点 (H2S、S2D)

例外	判定条件	缺省结果
E	(1) 输入是 subnormal。 (2) 输入是 QNaN。	当输入是 NaN 时, 结果是 QNaN
V	输入是 SNaN。	当输入是 NaN 时, 结果是 QNaN
Z	无	
O	无	
U	无	
I	无	

注意:

1. fs 位对其没有影响。
2. 输入为 $+\infty/-\infty$ 时结果也为 $+\infty/-\infty$ 。

精度降低的浮点转浮点 (S2H、D2S)

例外	判定条件	缺省结果
E	(1) 输入是 QNaN。 (2) 结果是目标格式的极小非 0 值 (即输入是 single/double 格式下的 subnormal, 或者是在 single/double 格式下可以表示但是在半精度格式/single 格式下由于太小而不能表示的输入)且 fs=0 或 enable_U=1 或 enable_I=1	当输入是 NaN 时, 结果是 QNaN
V	输入是 SNaN。	当输入是 NaN 时, 结果是 QNaN
Z	无	
O	输入超出了目标格式能表达的最大有限数, 且不是 $+\infty/-\infty$ 。	根据 rm 进行 round 后得到的值。 如果是 round to nearest, 或者 正数 round to $+\infty$, 或者负数 round to $-\infty$, 结果为相应符号的 $+\infty/-\infty$; 其他的情况, 结果为相应符号的该格式能表示的最大、最小值。
U	结果是目标格式下的极小非 0 值 且 fs=1	相应符号的+0/-0
I	(1) 结果与输入的值不等。 (2) Overflow。 (3) Underflow 且 enable_U=0 且 enable_I=0。	如果是由于 O 产生的 I, 结果就是 O 的 default value; 如果是由于 U 产生的 I, 结果就是 U 的 default value; 否则, 就是根据 rm 进行 round 后的结果。

注意:

1. S2H/D2S 最多只能报一种例外 (除了 O 和 I,或 U 和 I)。
2. 输入为 $+\infty/-\infty$ 时结果也为 $+\infty/-\infty$ 。

D2E

不发生任何例外。与 fs 无关。

Nan → 7fffbfff...fff

Inf → sign, ffff, 000...000

0 → sign, 0000...000

Subnormal → sign, 16 个 0, fract, 11 个 0

E2D

例外	判定条件	缺省结果
E	(1) 输入是 QNaN。 (2) 输入不为 0, round 后的结果为极小非 0 值, 且 (fs=0 或 enable_U=1 或 enable_I=1)	当输入是 NaN 时, 结果是 QNaN
V	(1) 输入是 SNaN。 (2) opa[63]=1, opa[78:64]=15 个 0。 (3) opa[63]=0, opa[78:64]不全为 0。	情况 (1) 时, 结果是 QNaN。 情况 (2) 和 (3) 时, 或者报 U, 或者报 E。报 U 时同 U 的缺省结果。
Z	无	
O	输入不是 inf、不是 nan, round 后的结果 (绝对值) 超过了 double 可以表示的最大有限数。	按 rm 进行 round 得到相应的结果, 如果是 round to nearest, 正数 round to +inf, 负数 round to -inf, 结果是 +∞/-∞; 其他情况, 结果是 double 格式能表示的最大的有限数 {sign, 7fefff...fff}
U	输入不是 nan、不是 0, fs=1 且 round 后的结果为极小非 0 值。	+0/-0
I	输入不是 inf、不是 nan, 且 (1) 结果与输入的值不等。 (2) Overflow。 (3) Underflow 且 enable_U=0 且 enable_I=0。	如果是由于 O 引起, 取 O 的 default 值; 如果是由于 U 引起, 取 U 的 default 值; 如果是由于结果 inexact 引起, 结果取 round 后的值。

浮点转小数 (F2Q)

例外	判定条件	缺省结果
E	输入是 subnormal。	
V	输入是 NaN。	0
Z	无	
O	round 后的结果超过了 [-1.0, 0.111...11] 的范围。(包括输入为 inf 时)	正数: 7ffff...ff 负数: 10000...00
U	无	
I	(1) Overflow。 (2) round 后结果和输入并不相等。	如果是由于 overflow, 则取 overflow 的值; 否则, 取根据 rm 进行 rounding 之后的值。

小数转浮点 (Q2F)

不会发生任何例外，fs 位对其没有影响。

浮点数转浮点格式的整数 (FRINT)

例外	判定条件	缺省结果
E	(1) 输入是 subnormal。 (2) 输入是 QNaN。	当输入是 NaN 时，结果是 QNaN
V	输入是 SNaN。	当输入是 NaN 时，结果是 QNaN
Z	无	
O	无	
U	无	
I	结果和输入值不相等。	根据 rm 进行 round 后的值。

注意：

- fs 位对其没有影响。
- 输入为 $+\infty/-\infty$ 时结果也为 $+\infty/-\infty$ 。

取最大值最小值 (MAX、MIN)

例外	判定条件	缺省结果
E	无	
V	opa 或者 opb 是 NaN	opb
Z	无	
O	无	
U	无	
I	无	

注意：

- 当 opa 和 opb 都是 0 时，结果取 opb。
- fs 位对其没有影响。

绝对值取最大值最小值 (MAXABS、MINABS)

例外	判定条件	缺省结果
E	opa 或者 opb 是 subnormal	
V	opa 或者 opb 是 NaN	opb
Z	无	
O	无	
U	无	
I	无	

注意：

1. 当 opa 和 opb 都是 0 时，结果取 opb。
2. fs 位对其没有影响。

ABS/NEG

例外	判定条件	缺省结果
E	opa 或者 opb 是 subnormal	
V	opa 或者 opb 是 NaN	QNaN
Z	无	
O	无	
U	无	
I	无	

注意：

1. 除了输入是 nan 时，其他结果都是只将符号取反或者将符号变成 0。
2. fs 位对其没有影响。

其它

其它指令都不会触发 EVZUI 例外，fs 位对其也没有任何影响。

2.2.7 浮点指令 E 例外判定说明

首先指出的是，在每类指令的说明中如果并没有提到 fs 位，那么说明 fs 位对该指令是否报 E 例外是不产生影响的。

F2I

ROUND.L.fmt	TRUNC.L.fmt	CEIL.L.fmt	FLOOR.L.fmt
ROUND.W.fmt	TRUNC.W.fmt	CEIL.W.fmt	FLOOR.W.fmt
CVT.W.S	CVT.W.D	CVT.L.S	CVT.L.D
FTINT_S.df	FTINT_U.df	FTRUNC_S.df	FTRUNC_U.df
VFROUNDdf.	VFCEIL.df	VFFLOOR.df	
VFROUNDDO_W	VFTRUNCDO_W	VFCEILDO_W	VFFLOORDO_W
VFTINTUPL_D	VFTINTUPR_D	VFTRUNCUPL_D	VFTRUNCUPR_D
VFCEILUPL_D	VFCEIUPR_D	VFFLOORUPL_D	VFFLOORUPR_D
VFROUNDUPL_D	VFROUNDUPR_D	VFTINTDO_W	

F2I 的指令只有在输入是非规格化数的时候报 E 例外。在其他情况下都不会报 E 例外。

所有的指令都不受 fs 位的影响，也就是 fs 等于 0 或者等于 1 的时候，指令的行为都是一样的。

I2F

CVT.S.W	CVT.S.L	CVT.D.W	CVT.D.L
---------	---------	---------	---------

FFINT_S.df	FFINT_U.df	VFFINTDO_W	
VFFINTUPL_D	VFFINTUPR_D		

I2F 的指令不会报 E 例外。

Fmaf

ADD.fmt	SUB.fmt	MUL.fmt	MADD.fmt
MSUB.fmt	NMADD.fmt	NMSUB.fmt	
FADD.df	FSUB.df	FMSUB.df	FMUL.df
FMADD.df	VFHADD.df	VFHSUB.df	VFADDSUB.df
VFSUBADD.df	VFMADD.df	VFMSUB.df	VFNMADD.df
VFNMSUB.df	VFMADDSUB.df	VFMSUBADD.df	

Fmaf 指令报 E 例外的条件较为繁杂，参考 2K1000 技术参考书中 2.2.6 节中的说明。

浮点比较

C.cond.fmt 和 MIPS SIMD 指令手册中 Table 4.5 中的所有指令。不会报 E 例外。

Fdiv

DIV.fmt	RECIP.fmt	FRCP.df	FDIV.df
---------	-----------	---------	---------

浮点除法指令在以下的三种情况下会报 E 例外：

- (1) 输入有任一个是 subnormal。
- (2) 输入有任一个是 QNaN。

(3) 当结果出现 Underflow condition 时，fs=0 或 enable_U=1 或 enable_I=1，报 E。(特别需要注意的是，这里面包含了一种特殊情况，在这种特殊情况下，不报 E。当 fs=1 且 $\text{exp}+\text{Bia}(\text{fmt})=0$ 时，不报 E。)

结果出现 Underflow condition 的条件是：

在除数不是 inf、snan、qnan，被除数不是 0 的情况下，结果化为 $1.\dots \times 2^{\text{exp}}$ 的形式，其中 $\text{exp}+\text{Bias}(\text{fmt}) \leq 0$ 。Bias(single)=127，Bias(double)=1023。

除了上述 3 种情况以外，其他情况下都不报 E 例外。

Fsqrt

SQRT.fmt	FSQRT.df
----------	----------

浮点开方指令在如下两种情况下会报 E 例外：

- (1) 输入是 subnormal。
- (2) 输入是 QNaN

在其他情况下都不报 E 例外。

Frsqrt

RSQRT.fmt	FRSQRT.df
-----------	-----------

在如下两种情况下会报 E 例外：

- (1) 输入是 subnormal。
- (2) 输入是 QNaN

在其他情况下都不报 E 例外。

FLOG2.df

该指令在如下两种情况下会报 E 例外：

- (1) fs=0 且输入是 subnormal。
- (2) 输入是 QNaN。

在其他情况下都不会报 E 例外。

FEXP2.df

该指令在如下三种情况下会报 E 例外：

- (1) 输入是 subnormal。
- (2) 输入是 QNaN。
- (3) fs=0 或 enable_U=1 或 enable_I=1 时，结果出现了 underflow condition。

其中，(3) 中结果出现了 underflow condition 指的是：

当输入不是 nan、0、inf、subnormal 时，fs=0 且 rounding 后的结果是极小非 0 值。

Rounding 后的结果是极小非 0 值，指的是结果化为 $1.\dots \times 2^{\text{exp}}$ 的形式时，结果的值在 $(-2E_{\text{min}}, 2E_{\text{min}})$ 之间)。E_min=-126 when single, E_min=-1022 when double。

在其他情况下都不会报 E 例外。

H2S、S2D

H2S:

FEXUPL_W	FEXUPR_W
----------	----------

S2D:

CVT.D.S	FEXUPL_D	FEXUPR_D
---------	----------	----------

报 E 例外的情况如下：

- (1) 输入是 subnormal。
- (2) 输入是 QNaN。

在其他情况下都不会报 E 例外。

S2H、D2S

S2H: FEXDO_H

D2S:

CVT.S.D	FEXDO_W
---------	---------

报 E 例外的情况如下:

(1) 输入是 QNaN。

(2) 结果是目标格式的极小非 0 值 (即输入是 single/double 格式下的 subnormal, 或者是在 single/double 格式下可以表示但是在半精度格式/single 格式下由于太小而不能表示的输入) 且 fs=0 或 enable_U=1 或 enable_I=1)。

在其他情况下都不会报 E 例外。

FTQ.df

该指令只有当输入是 subnormal 时, 报 E 例外, 其他时候都不报 E 例外。

Q2F

FFQL.df	FFQR.df
---------	---------

该类指令不会报任何例外。

FRINT

FRINT.df			
VFRINT_RM00.df	VFRINT_RM01.df	VFRINT_RM10.df	VFRINT_RM11.df

报 E 例外的情况如下:

(1) 输入是 subnormal。

(2) 输入是 QNaN。

在其他情况下都不会报 E 例外。

MAX、MIN

FMAX.df	FMIN.df	VFHMAX.df	VFHMIN.df
---------	---------	-----------	-----------

该类指令不会报 E 例外。

MAXABS、MINABS

FMAX_A.df	FMIN_A.df
-----------	-----------

该类指令只有当输入中有 subnormal 时, 报 E 例外, 其他时候都不报 E 例外。

ABS/NEG

ABS.fmt	VFABS.df	NEG.fmt	VFNEG.df
---------	----------	---------	----------

该类指令只有当输入中有 subnormal 时，报 E 例外，其他时候都不报 E 例外。

2.3 MIPS64 兼容指令实现相关定义

本小节对于 GS264 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。

2.3.1 目标为 0 号通用寄存器的 load 指令

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达到预取的效果，否则仍然会触发各种访存相关例外。

2.3.2 PREF 指令

PREF 指令不会产生任何预取效果，视同 NOP 指令处理。

2.3.3 PREFX 指令

PREFX 指令不会产生任何预取效果，视同 NOP 指令处理。

2.3.4 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¹或外部硬件中断打断。处理器核处于低功耗模式且时钟关闭的情况下，能够自动开启时钟处理来自外部的缓存一致性请求，一致性请求处理完毕后，时钟会再度关闭，整个过程处理器核仍处于低功耗模式，也就是说不会进行取指。

执行 WAIT 指令进入休眠模式后可以被中断唤醒，该唤醒过程在硬件上至多引入 1~2 个处理器时钟周期的延迟。由于被唤醒的处理器一定会处理唤醒 WAIT 的中断，所以软件看到的从 WAIT 休眠状态唤醒到 WAIT 后面的第一条指令真正被执行的时间延迟包含了软件处理唤醒 WAIT 的中断的时间。

2.3.5 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。

2.3.6 SYNCI 指令

因 GS264 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。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

¹ 仅当 CP0.Status.RP=0 时，才有可能被性能计数器溢出中断唤醒。

例外。现有实现中 SYNCI 指令产生的执行效果与类似于 X86 架构下的串行化指令，即该指令只有等到前面所有的指令都产生执行效果后才开始执行，且只有该指令执行完毕后，其后续指令才能取入到流水线中执行。

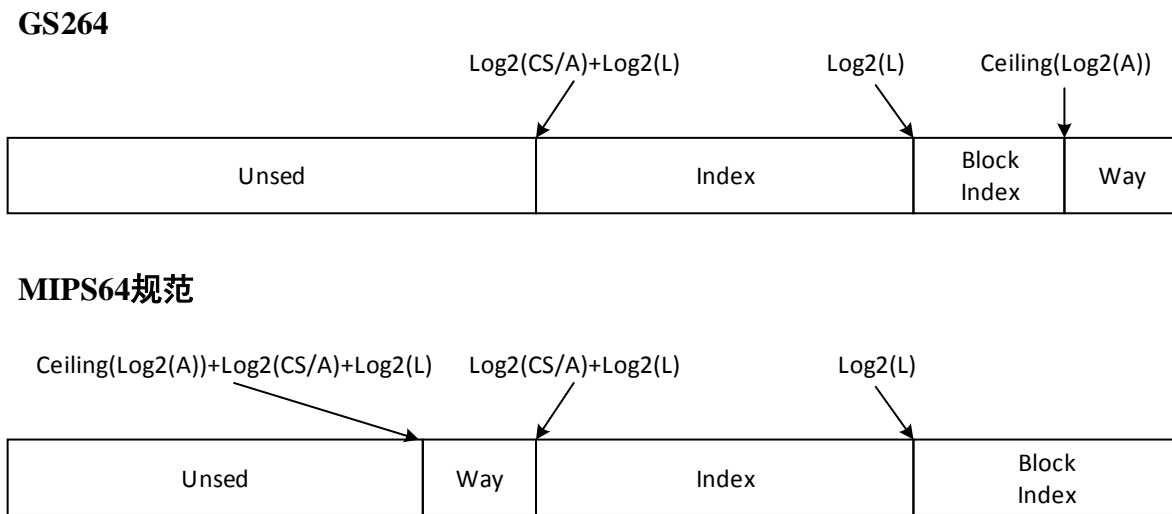
2.3.7 CACHE 指令

本处理器所实现的 CACHE 指令与 MIPS64 规范存在的区别主要有两点：

1、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。

图 2-8 Index 类 CACHE 指令的地址解析格式



说明：

假设所操作的Cache的容量为CS，A路组相联，行大小为L字节，则
 用于选择第几路的地址位数为： $\text{Ceiling}(\text{Log}_2(A))$
 用于索引每一路中第几个Cache行的地址位数为： $\text{Log}_2(CS/A)$
 用户Cache行内字节定位的地址位数为： $\text{Log}_2(L)$

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

GS264 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。GS264 中这些指令均为 Index Store Data 操作，而 MIPS64 规范中均为 Fetch and Lock 操作。

2.3.8 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。GS264 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。

2.3.9 EHB、SSNOP 指令

GS264 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 龙芯扩展指令集

GS264 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 2-26 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
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	取字左部到浮点寄存器	LoongEXT32
GSLWRC1	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
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

指令助记符	指令功能简述	ISA 兼容类别
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 2-27 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
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

指令助记符	指令功能简述	ISA 兼容类别
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 2-28 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
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

指令助记符	指令功能简述	ISA 兼容类别
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 的移位置加 32 的双字逻辑左移	LoongEXT64
X86SLLV.D	以 x86 方式只设置 EFLAG 的双字可变移位置左移	LoongEXT64
X86SLLV.W	以 x86 方式只设置 EFLAG 的字可变移位置左移	LoongEXT32
X86SLLV.H	以 x86 方式只设置 EFLAG 的半字可变移位置左移	LoongEXT32
X86SLLV.B	以 x86 方式只设置 EFLAG 的字节可变移位置左移	LoongEXT32
X86SRL.D	以 x86 方式只设置 EFLAG 的双字逻辑右移	LoongEXT64
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 的移位置加 32 的双字逻辑逻辑右移	LoongEXT64
X86SRLV.D	以 x86 方式只设置 EFLAG 的双字可变移位置逻辑右移	LoongEXT64
X86SRLV.W	以 x86 方式只设置 EFLAG 的字可变移位置逻辑右移	LoongEXT32
X86SRLV.H	以 x86 方式只设置 EFLAG 的半字可变移位置逻辑右移	LoongEXT32
X86SRLV.B	以 x86 方式只设置 EFLAG 的字节可变移位置逻辑右移	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

指令助记符	指令功能简述	ISA 兼容类别
X86DSRA32	以 x86 方式只设置 EFLAG 的移位置加 32 的双字逻辑算术右移	LoongEXT64
X86SRAV.D	以 x86 方式只设置 EFLAG 的双字可变移位置算术右移	LoongEXT64
X86SRAV.W	以 x86 方式只设置 EFLAG 的字可变移位置算术右移	LoongEXT32
X86SRAV.H	以 x86 方式只设置 EFLAG 的半字可变移位置算术右移	LoongEXT32
X86SRAV.B	以 x86 方式只设置 EFLAG 的字节可变移位置算术右移	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 的移位置加 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 的移位置加 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 的移位置加 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 的移位置加 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 的可变移位置量的双字循环右移	LoongEXT64
X86ROTRV.W	以 x86 方式只设置 EFLAG 的可变移位置量的字循环右移	LoongEXT32
X86ROTRV.H	以 x86 方式只设置 EFLAG 的可变移位置量的半字循环右移	LoongEXT32
X86ROTRV.B	以 x86 方式只设置 EFLAG 的可变移位置量的字节循环右移	LoongEXT32
X86ROTLV.D	以 x86 方式只设置 EFLAG 的可变移位置量的双字循环左移	LoongEXT64
X86ROTLV.W	以 x86 方式只设置 EFLAG 的可变移位置量的字循环左移	LoongEXT32
X86ROTLV.H	以 x86 方式只设置 EFLAG 的可变移位置量的半字循环左移	LoongEXT32
X86ROTLV.B	以 x86 方式只设置 EFLAG 的可变移位置量的字节循环左移	LoongEXT32
X86RCRV.D	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的双字循环右移	LoongEXT64
X86RCRV.W	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的字循环右移	LoongEXT32
X86RCRV.H	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的半字循环右移	LoongEXT32
X86RCRV.B	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的字节循环右移	LoongEXT32
X86RCLV.D	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的双字循环左移	LoongEXT64
X86RCLV.W	以 x86 方式只设置 EFLAG 的可变移位置量的带 CF 位的字循环左移	LoongEXT32

指令助记符	指令功能简述	ISA 兼容类别
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 2-29 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
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

指令助记符	指令功能简述	ISA 兼容类别
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
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 2-30 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
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

指令助记符	指令功能简述	ISA 兼容类别
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
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

指令助记符	指令功能简述	ISA 兼容类别
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 2-31 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
DCTO	双字尾随 1 个数	LoongEXT64

2.5 SIMD 扩展指令集

2.5.1 MSA 兼容指令列表

GS264 实现了 MIPS MSA1.0 的所有指令，详细内容可以参考《MIPS® Architecture for Programmers Volume IV-j: The MIPS64® SIMD Architecture Module》。

2.5.2 MSA 兼容部分与 MIPS 规范的细节差异

1. MSA 规范中浮点操作大多数规定兼容 IEEE754-2008，但是 GS264 的向量浮点完全复用标量浮点

运算的部件，所以关于例外等细节的定义自成体系，具体情况请参看 2.2.6 节浮点运算实现细节说明和 2.2.7 节浮点指令 E 例外判定说明。

2. MSA 规范中浮点操作中 SNaN、QNaN 的定义按照 IEEE754-2008，但是 GS264 的实现按照 IEEE754-1985，所以恰好是相反的。
3. MSA 规范中浮点操作支持 NaN 传递，但是 GS264 不支持。
4. 对于 FEXDO.H 指令，MSA 规范中其结果不收 MSACSR1.FS 的影响；但是 GS264 的实现中，如果结果是 tiny non-zero 且 MSACSR1.FS=1 时，结果为 0。
5. 对于 FMADD.df、FMSUB.df 指令，如果禁止触发 V 例外但是 V 例外产生条件又满足时，在 MSA 规范中如果乘法操作的一个操作数是+0 或-0，另一个操作数是+ ∞ 或- ∞ 时，结果是 0x7FC0.0000 或 0x7FF8.0000.0000.0000；但是 GS264 的实现中，结果为 0x7FBF.FFFF 或 0x7FF7.FFFF.FFFF.FFFF。
6. 对于 FTINT_S.df 和 FTRUNC_S.df 指令，在 MSA 规范中，如果操作数是 NaN 则结果为 0，如果操作数是超过范围的正数（包括+inf）则结果为最大的有符号整数，如果操作数是超过范围的负数（包括-inf）则结果为最小的有符号整数；但是 GS264 实现中以上三种情况结果都是最大的有符号整数。

3 处理器运行模式

GS264 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)和正常模式(Func Mode)。其中调试模式主要对应 EJTAG 例外处理程序的运行环境；正常模式对应真实主机上操作系统及其上软件运行环境。其中，正常模式又可进一步划分为核心模式(Kernel Mode)、监管模式(Supervisor Mode)和用户模式(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，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

CP0				模式
Debug.DM	Status.ERL	Status.EXL	Status.KSU	
1	Don't care			调试模式
0	1	Don't care		核心模式
	0	1	Don't care	
		0	00	监管模式
			01	用户模式
Don't care			11	无意义

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 核心模式

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核心模式。

3.1.3 用户模式

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一虚地址空间上。普通的用户程序均运行在用户模式下。

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 40。

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 ~ 0xC000.00FF.8000.0000	非法地址段	非法地址段	非法地址段
xkseg	0xC000.00FF.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 ~ 0x4000.0100.0000.0000	非法地址段	非法地址段	非法地址段
xksseg xsseg	0x4000.00FF.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 ~ 0x0000.0100.0000.0000	非法地址段	非法地址段	非法地址段
xkuseg xsuseg xuseg	0x0000.00FF.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(2²⁹)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 102 页 7.22 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 2⁴⁰ 字节。xkphys 段的虚地址解析方式如图 4-1 所示。虚地址的[58:40]必须为全 0, 否则为非法地址。虚地址的[39:0]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 82 页表 7-6。

图 4-1 xkphys 段虚地址解析方式

63 62 61	59 58	40 39	0
10	缓存一致属性	若不等于全 0 则为非法地址	物理地址

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 MIPS EJTAG 规范。

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 $0x80000000 + 0xFFFFF7FC = 0x7FFFFFFF$ 。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 $0xFFFFF7FC + 0xFFFFF7FC = 0xFFFFF7FC$ 。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 ITLB

为了加快访问速度，GS264 在取指部件中包含的一个容量较小的全相联查找的 TLB，称之为 ITLB。ITLB 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，ITLB 与 TLB 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 TLB 中的内容来确定 ITLB 中存放的信息。在一般运行过程中，ITLB 与 TLB 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 CP0.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 采用全相联查找表结构，其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 的内容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 TLB 组织结构

TLB 的组织形式及操作方式采用 MIPS 传统的全相联 TLB 方式，在 GS264 中为 64 项。每一个 TLB 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：

- 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。GS264 支持 4KB 至 1GB，以 4 的倍数递增的页大小。

上述各个域的解释可进一步参看 7.13 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)。
- 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。

4.3.3 TLB 的软件管理

GS264 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 GS264 中不予实现。

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。GS264 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	TLB 访问指定索引寄存器	第 78 页 7.2 节
1	0	Random	TLB 访问随机索引寄存器	第 79 页 7.3 节
2	0	EntryLo0	TLB 表项低位内容中与偶数虚页相关部分	第 80 页 7.4 节
3	0	EntryLo1	TLB 表项低位内容中与奇数虚页相关部分	第 80 页 7.4 节
4	0	Context	指向内存中页表项的指针	第 83 页 7.5 节
5	0	PageMask	TLB 页表大小控制	第 85 页 7.7 节
5	1	PageGrain	1KB 小页等页表属性控制	第 86 页 7.8 节
6	0	Wired	控制 VTLB 中固定项数目	第 87 页 7.9 节

Reg.	Sel.	寄存器名称	功能定义	索引
8	0	BadVAddr	记录最新地址相关例外的出错地址	第 89 页 7.11 节
10	0	EntryHi	VTLB 与 FTLB 表项高位内容	第 91 页 7.13 节
20	0	XContext	扩展地址模式下页表指针	第 111 页 7.30 节
22	0	Diag	龙芯扩展诊断控制寄存器	第 112 页 7.31 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)。

表 4-3 TLB 管理相关特权指令

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

4.3.4 TLB 初始化与清空

TLB 的初始化与清空采用传统 MIPS TLB 的操作方式。本手册不再进行详细描述，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。

4.3.5 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。系统软件需要保证同一虚地址不可在 TLB 中出现多项命中。

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

/* TLB 查找过程 */
tlb_found ← 0
for i = 0 to 63 step 1
  if ((TLB[i].G || (TLB[i].ASID == EntryHi.ASID)) &&
      (TLB[i].R==va[63:62]) &&
      (TLB[i].VPN2[26:18] == va[39:31]) &&
      ((TLB[i].VPN2[17:0] & ~TLB[i].MASK) == (va30..13 & ~TLB[i].MASK))) then
    if (!tlb_found) then
      tlb_found ← 1
      case TLB[i].MASK
        0b00 0000 0000 0000 0000: tlb_evenoddbit ← 12 //4KB page
        0b00 0000 0000 0000 0011: tlb_evenoddbit ← 14 //16KB page
        0b00 0000 0000 0000 1111: tlb_evenoddbit ← 16 //64KB page
```

```

0b00 0000 0000 0011 1111: tlb_evenoddbit ← 18 //256KB page
0b00 0000 0000 1111 1111: tlb_evenoddbit ← 20 //1MB page
0b00 0000 0011 1111 1111: tlb_evenoddbit ← 22 //4MB page
0b00 0000 1111 1111 1111: tlb_evenoddbit ← 24 //16MB page
0b00 0011 1111 1111 1111: tlb_evenoddbit ← 26 //64MB page
0b00 1111 1111 1111 1111: tlb_evenoddbit ← 28 //256MB page
0b11 1111 1111 1111 1111: tlb_evenoddbit ← 30 //1GB page
otherwise: UNDIFINED
endcase
if va[tlb_evenoddbit] == 0 then
    tlb_pfn ← TLB[i].PFN0
    tlb_v ← TLB[i].V0
    tlb_c ← TLB[i].C0
    tlb_d ← TLB[i].D0
    tlb_ri ← TLB[i].RI0
    tlb_xi ← TLB[i].XI0
else
    tlb_pfn ← TLB[i].PFN1
    tlb_v ← TLB[i].V1
    tlb_c ← TLB[i].C1
    tlb_d ← TLB[i].D1
    tlb_ri ← TLB[i].RI1
    tlb_xi ← TLB[i].XI1
endif
else
UNDIFINED
endif
endif
endfor

/* 合法性检查及物理地址生成 */
if (tlb_found) then
    if (!tlb_v) then
        SignalException(TLBInvalid, reftype)
    endif
    if (tlb_ri && (reftype == load)) then
        if (PageGrain.IEC) then
            SignalException(TLBRI, reftype)
        else
            SignalException(TLBInvalid, reftype)
        endif
    endif
    if (tlb_xi && (reftype == fetch)) then
        if (PageGrain.IEC) then

```

```
    SignalException(TLBXI, reftype)
  else
    SignalException(TLBInvalid, reftype)
  endif
endif
if (!tlb_d && (reftype == store)) then
  SignalException(TLBModified)
endif
PAddr ← tlb_pfn[35:tlb_evenoddbit-12] || va[tlb_evenoddbit-1:0]
else
  SignalException(TLBMiss, reftype)
endif
```

5 缓存的组织与管理

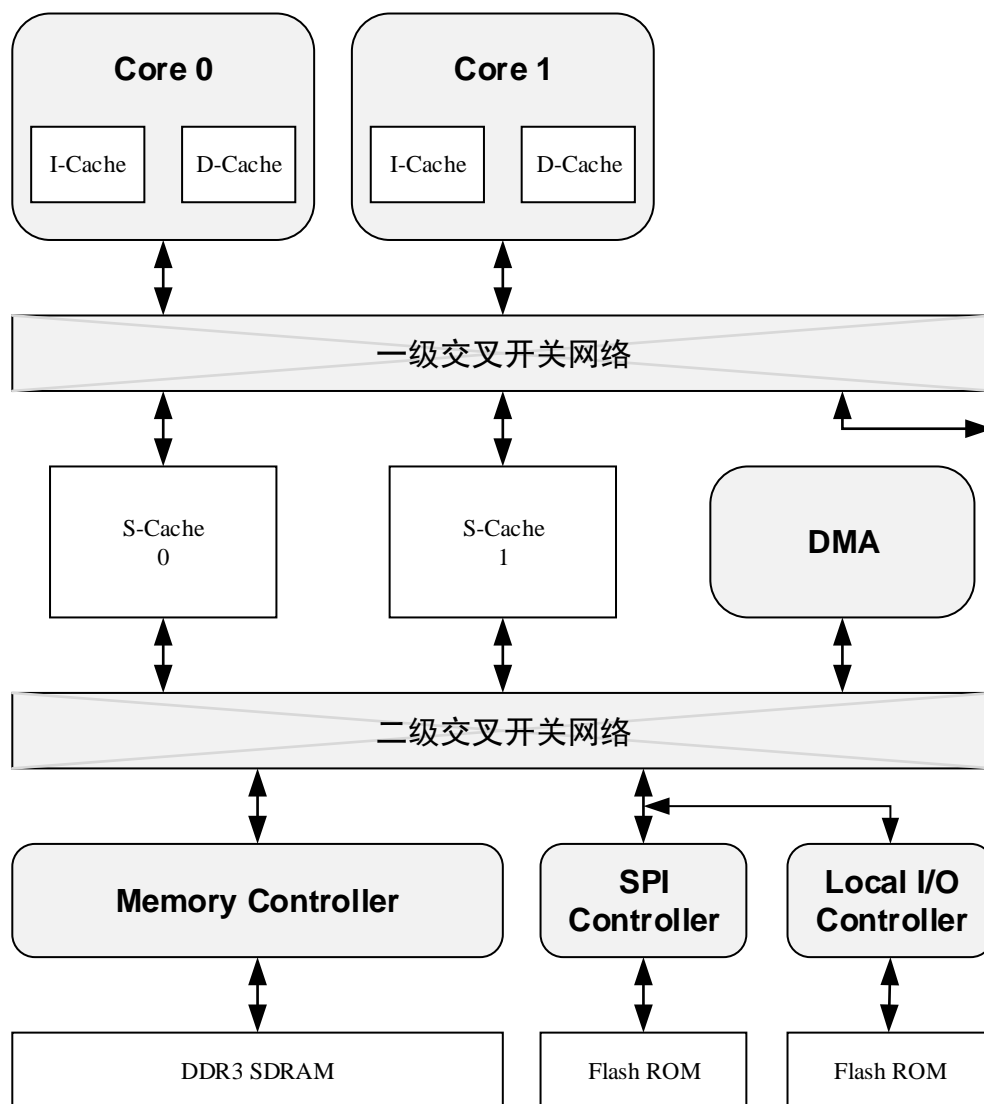
在 MIPS 架构下，处理器中各级缓存(Cache)是核心态软件可见的。对于缓存的某些操作（如缓存初始化、一致性维护等），需要软件参与缓存的管理。本章介绍 GS264 的缓存组织与管理。本手册中不展开描述处理器缓存的一般性概念，如索引(Index)、标签(Tag)、缓存行(Cacheline)、组相联、缓存访问、缓存命中与缺失、虚地址索引物理地址标签(Virtual Index Physical Tag)的缓存、缓存替换等。若读者对相关概念尚不明晰，请参考计算体系架构的相关书籍，推荐《MIPS 体系结构透视》（第二版）。

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 2K1000 芯片处理器采用含有两级缓存的存储层次，图 5-1 给出了其示意。

图 5-1 龙芯 2K1000 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)，第二级的共享缓存(Shared-Cache, S-Cache)。其中 I-Cache 和 D-Cache 为每个处理器核私有，S-Cache 为多核共享。处理器核通过一级交叉开关网络访问 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。S-Cache 为混合 Cache，既存放指令也存放数据。

I-Cache 和 D-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 或 D-Cache 中，就一定在 S-Cache 中也能找到一个相同物理地址的备份。此处提到的包含的关系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 和 S-Cache 之间的数据一致性由硬件维护。

当取指部件在 I-Cache 或访存部件在 D-Cache 中查找不命中时，将查找 S-Cache。S-Cache 的响应返回后，将填入 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 S-Cache 中该行对应的位置。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	共享缓存
容量	32KB	32KB	512KB/体 (芯片共 1MB)
相联度	4 路	4 路	8 路
行大小(line size)	512bit	512bit	512bit
索引(Index)	虚地址[12:6]	虚地址[12:6]	物理地址[16:7]
标签(Tag)	物理地址[39:12]	物理地址[39:12]	物理地址[39:16]
替换策略	随机替换算法	随机替换算法	LRU 替换算法
写策略		写回、写分配	写回、写分配
校验方式	奇偶校验	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 32KB，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[12:6]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 12 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。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 表示缓存行上无有效内容。

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 S-Cache 或内存中重新取回所需的缓存行内容。

5.1.3 一级数据缓存(D-Cache)

一级数据缓存的容量为 32KB，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。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[12:6]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 12 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。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。

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，软件若无特殊诊断需要，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。

5.1.4 二级共享缓存(S-Cache)

二级共享缓存基于目录协议支持缓存一致性。龙芯 2K1000 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 2K1000 芯片的 S-Cache 采用分体结构，共分为 2 个体(bank)，通过第一级交叉开关互连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。龙芯 2K1000 芯片在第一级交叉开关互连网络上采用了固定地址窗口映射机制，所以不同的请求最终落在 2 个 S-Cache 体中的哪一个是通过物理地址中的第 6 位来决定。每个 Scache 体中 Cache 行内的寻址仍用物理地址的[5:0]位，但是每一路 Cache 索引的地址采用的是物理地址的[16:7]位。

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 指令时以块作为基本单位，此时物理地址的[5:3]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。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 64 页 5.4.5 节中的描述。

共享缓存的校验

二级共享缓存采用“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。

5.2 缓存算法与缓存一致属性

GS264 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。GS264 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在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

GS264 实现了基于目录的缓存一致性协议，由硬件保证 I-Cache、D-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。

GS264 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 8 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 5-2 所示。

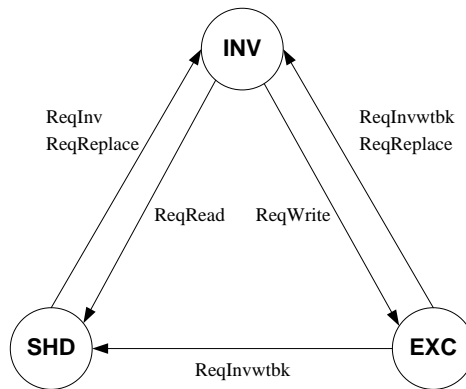
当读指令或者取数操作在一级 Cache 中未命中时，处理器核向二级 S-Cache 发出 Reqread 请求，在得到 S-Cache 返回的 Repread 应答后，处理器核的一级 Cache 获得了一个 SHD 状态的 Cache 行备份。当存数操

作在一级 Cache 中未命中时，处理器核向二级 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 I-Cache 或 D-Cache 替换时，通过 Reqreplace 写回 S-Cache 模块，S-Cache 通过 Replace 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 或 D-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 S-Cache 模块。

图 5-2 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 和 S-Cache。CACHE 指令的格式为：CACHE op, offset(base)。GS264 的 Cache 指令与 MIPS64 规范存在一些差异，本手册已在 35 页第 2.3.7 小节进行了详细说明。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。

CACHE 指令

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 5-2 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

op[4:0]	功能描述	目标 Cache
b11001	根据索引读 Cache 行 Data	D-Cache
b11101	根据索引写 Cache 行 Data	D-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
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

5.4.2 缓存初始化

基于硬件的缓存初始化

GS264 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

GS264 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;
for (addr=0xffffffff80000000; addr<0xffffffff80008000; 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;
```

```
for (addr=0xffffffff80000000; addr<0xffffffff80008000; 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 S-Cache。

```
Scache_init_ok[get_my_CPUNum()]=0; /*需使用 uncached write*/
mtc0 zero, TagLo; mtc0 zero, TagHi;
/*0 号处理器初始化 S-Cache Entity 0, 1 号处理器初始化 S-Cache Entity 1 */
bank = get_my_CPUNum();
for (addr=0xffffffff80000000; addr<0xffffffff80100000; addr+=128) {
    /*将 S-Cache 各路逐行置为确定的值*/
    for (way=0; way<8; way+=1) {
        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 6: 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 5、6，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。GS264 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，GS264 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 处理器与设备 DMA 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。对于龙芯 2K1000 芯片，若 PRId=0x00146101，必须通过软件维护处理器与设备 DMA 之间的缓存一致性；若 PRId=0x00146103，可以利用硬件维护处理器与设备 DMA 之间的缓存一致性。

5.4.5 缓存别名与页着色

因为 GS264 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 8KB，所以当页大小小于 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”。GS264 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，GS264 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、向量浮点例外、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=1 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。

使用 ERET 指令从例外返回也会将 EXL 位自动置为 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 73 页 6.2 节。

控制寄存器 Cause 的 ExcCode 域：

0x00 (Int) (请参看 97 页表 7-22)

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。

在用户模式或监管模式下访问核心模式的地址段。

在用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内。

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 97 页表 7-22)

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 CP0.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 CP0.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。

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 97 页表 7-22)

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。

6.1.9 XTLB 重填例外

64 位主机地址空间下，且 CP0.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 CP0.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。

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 97 页表 7-22)

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。

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 CP0.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 CP0.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。

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 97 页表 7-22)

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。

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。

控制寄存器 Cause 的 ExcCode 域：

0x01 (Mod) (请参看 97 页表 7-22)

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。

6.1.12 TLB 执行阻止例外

当 CP0.PageGrain.IEC=0, 且 CP0.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。

控制寄存器 Cause 的 ExcCode 域:

0x14 (TLBXI) (请参看 97 页表 7-22)

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。

6.1.13 TLB 读取阻止例外

当 CP0.PageGrain.IEC=0, 且 CP0.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。

控制寄存器 Cause 的 ExcCode 域:

0x13 (TLBRI) (请参看 97 页表 7-22)

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。

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。

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。

控制寄存器 Cause 的 ExcCode 域:

0x0c (Ov) (请参看 97 页表 7-22)

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。

控制寄存器 Cause 的 ExcCode 域:

0x0d (Tr) (请参看 97 页表 7-22)

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。

控制寄存器 Cause 的 ExcCode 域:

0x08 (Sys) (请参看 97 页表 7-22)

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。

控制寄存器 Cause 的 ExcCode 域:

0x09 (Bp) (请参看 97 页表 7-22)

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 GS264 未实现的指令时, 触发保留指令例外。

控制寄存器 Cause 的 ExcCode 域:

0x0a (RI) (请参看 97 页表 7-22)

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)。
- 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、gsSWGTC1、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。

控制寄存器 Cause 的 ExcCode 域:

0x0b (CpU) (请参看 97 页表 7-22)

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 16 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 13 页 2.2.3 节。

控制寄存器 Cause 的 ExcCode 域:

0x0f (FPE) (请参看 97 页表 7-22)

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。

控制寄存器 Cause 的 ExcCode 域:

0x10 (GSExc) (请参看 97 页表 7-22)

控制寄存器 GSCause 的 GSExcCode 域:

0x00 (IS) (请参看 113 页表 7-37)

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

GS264 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 47¹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写 0 间接清除 Cause.PCI 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
最低优先级	软件中断	SW1	Cause.IP1 & Status.IM1	1
		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 95 页表 7-19。

¹ GS464E 中性能计数器实际有效计数位数为 48 位。

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 95 页表 7-19。

GS264 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
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 GS264 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 2K1000 芯片中断系统时，需同时参看《龙芯 2K1000 处理器用户手册》第 7、8 两章。

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 2K1000 处理器用户手册》第 8 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(CP0.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 2K1000 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 协处理器 0 寄存器概览

GS264 处理器核中实现的协处理器 0 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	TLB 访问指定索引寄存器	第 78 页 7.2 节
1	0	Random	TLB 访问随机索引寄存器	第 79 页 7.3 节
2	0	EntryLo0	TLB 表项低位内容中与偶数虚页相关部分	第 80 页 7.4 节
3	0	EntryLo1	TLB 表项低位内容中与奇数虚页相关部分	第 80 页 7.4 节
4	0	Context	指向内存中页表项的指针	第 83 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 84 页 7.6 节
5	0	PageMask	VTLB 页表大小控制	第 85 页 7.7 节
5	1	PageGrain	1KB 小页等页表属性控制	第 86 页 7.8 节
6	0	Wired	控制 VTLB 中固定项数目	第 87 页 7.9 节
7	0	HWRENa	RDHWR 指令可访问寄存器使能控制	第 88 页 7.10 节
8	0	BadVAddr	记录最新地址相关例外的出错地址	第 89 页 7.11 节
9	0	Count	处理器时钟计数器	第 90 页 7.12 节
10	0	EntryHi	TLB 表项高位内容	第 91 页 7.13 节
11	0	Compare	计时器中断控制	第 92 页 7.14 节
12	0	Status	处理器状态与控制寄存器	第 93 页 7.15 节
12	1	IntCtl	中断系统状态与控制寄存器	第 95 页 7.16 节
12	2	SRSCtl	影子寄存器状态与控制寄存器	第 96 页 7.17 节
13	0	Cause	存放上一次例外原因	第 97 页 7.18 节
14	0	EPC	存放上一次发生例外指令的 PC	第 99 页 7.19 节
15	0	PRId	处理器 ID	第 100 页 7.20 节
15	1	EBase	例外入口基址寄存器	第 101 页 7.21 节
16	0	Config	配置寄存器	第 102 页 7.22 节
16	1	Config1	配置寄存器 1	第 103 页 7.23 节
16	2	Config2	配置寄存器 2	第 104 页 7.24 节
16	3	Config3	配置寄存器 3	第 105 页 7.25 节
16	4	Config4	配置寄存器 4	第 107 页 7.26 节
16	5	Config5	配置寄存器 5	第 108 页 7.27 节
16	6	GSConfig	龙芯扩展配置寄存器	第 109 页 7.28 节
17	0	LLAddr	存放 Load-Link 指令访问地址	第 110 页 7.29 节
20	0	XContext	扩展地址模式下页表指针	第 111 页 7.30 节
22	0	Diag	龙芯扩展诊断控制寄存器	第 112 页 7.31 节
22	1	GSCause	存放上一次龙芯扩展例外的补充信息	第 113 页 7.32 节
23	0	Debug	EJTAG Debug 寄存器	第 114 页 7.33 节
24	0	DEPC	存放上一次 EJTAG 调试例外的 PC	第 115 页 7.34 节

Reg.	Sel.	寄存器名称	功能定义	索引
25	0-7	PerfCnt0-PerfCnt7	处理器核内部性能计数器访问接口	第 116 页 7.35 节
26	0	ErrCtl	Cache Parity/ECC 校验值寄存器	第 120 页 7.36 节
27	0	CacheErr	Cache Parity/ECC 校验状态与控制寄存器	第 121 页 7.37 节
27	1	CacheErr1	Cache Parity/ECC 校验状态与控制寄存器 1	第 122 页 7.38 节
28	0	TagLo	Cache Tag 访问接口低位部分	第 123 页 7.39 节
28	1	DataLo	Cache Data 访问接口低位部分	第 125 页 7.40 节
29	0	TagHi	Cache Tag 访问接口高位部分	第 126 页 7.41 节
29	1	DataHi	Cache Data 访问接口高位部分	第 127 页 7.42 节
30	0	ErrorEPC	存放上一次发生错误的指令的 PC	第 128 页 7.43 节
31	0	DESAVE	EJTAG 调试例外保存寄存器	第 129 页 7.44 节
31	2-3	KScratch1-KScratch2	核心态可访问便签寄存器 1~2	第 130 页 7.45 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

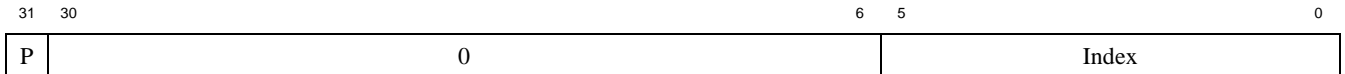


表 7-2 Index 寄存器域描述

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	无
0	30..6	只读恒为 0。	0	0
Index	5..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 TLB 的第 0..63 项。	R/W	无

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当处理器有指令提交时将自增 1，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。

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

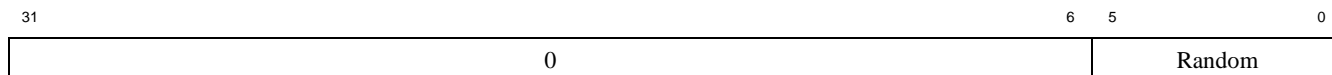


表 7-3 Random 寄存器域描述

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

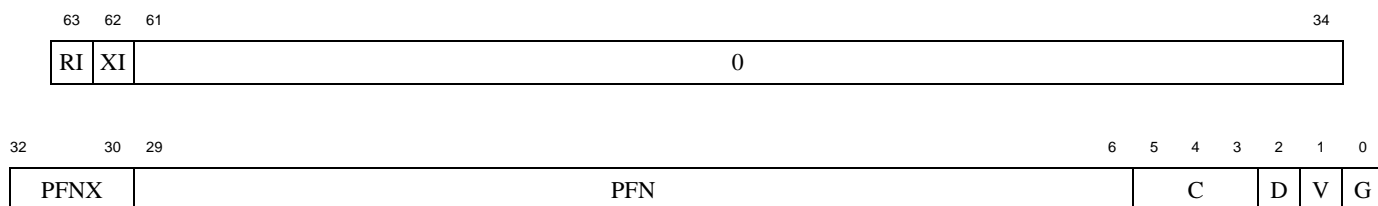


表 7-4 EntryLo0 和 EntryLo1 在 DMFC0/DMTC0 指令访问时的寄存器域描述

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 域都将读出于 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 域都将读出于 0。	R/W	0x0
0	61..33	只读恒为 0。	0	0
PFNX	32..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 (Config _{3LPA} =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 39..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 ⁷),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

⁷ 龙芯 2K1000 芯片的 GS264 处理器核的 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。

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

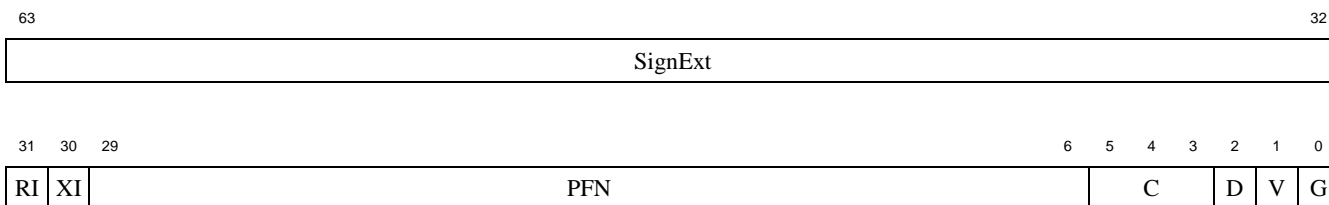


表 7-5 EntryLo0 和 EntryLo1 在 MFC0/MTC0 指令访问时的寄存器域描述

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 ⁸
0	保留，强行配置等效于 Uncached
1	保留，强行配置等效于 Uncached
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

⁸ 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

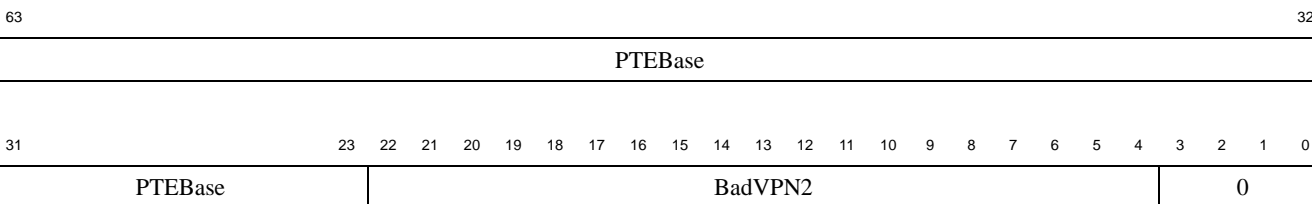


表 7-7 Context 寄存器域描述

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, Seletct 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

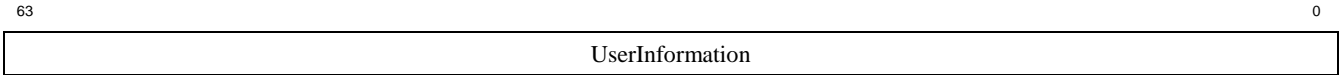


表 7-8 UserLocal 寄存器域描述

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

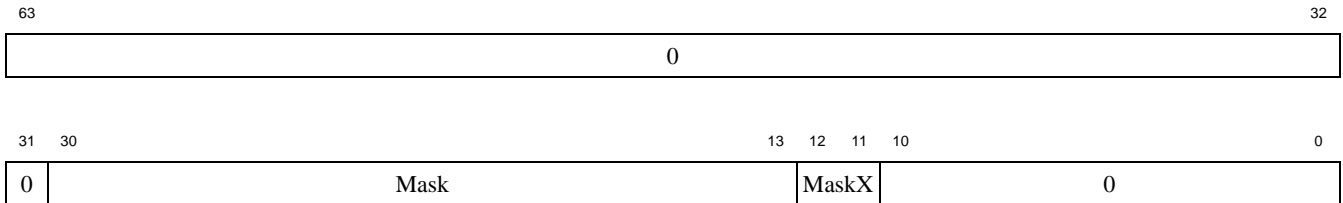


表 7-9 PageMask 寄存器域描述

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 0, 不支持 1KB 大小的页。	R	0x0
0	10..0	只读恒为 0。	0	0

表 7-10 给出了 GS264 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 GS264 允许软件对 PageMask 寄存器写入: 0x1、0x7、0x1F、0x7F、0x1FF、0x7FF、0x1FFF、0x7FFF、0x1FFFF 这些值, 但实际 PageMask 寄存器中记录的值依次为: 0x0、0x3、0xF、0x3F、0xFF、0x3FF、0xFFF、0x3FFF、0xFFFF。

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

PageGrain 寄存器是一个可读写寄存器。GS264 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

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

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 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 TLB 中固定表项与随机替换表项之间的边界。其示意如图 7-9 所示。

图 7-9 TLB 中固定表项与随机替换表项边界

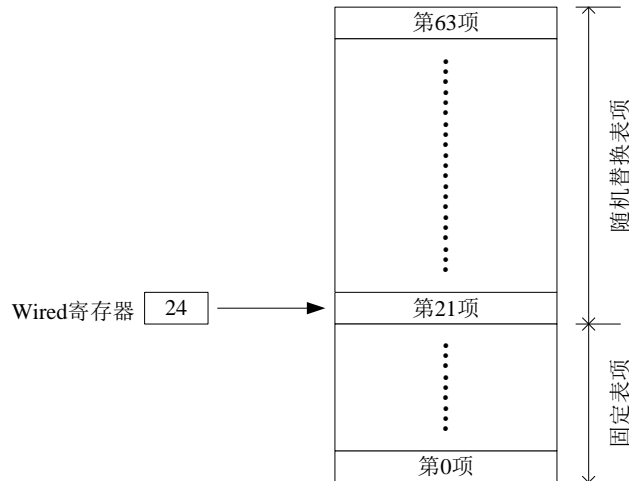


图 7-10 说明了 Wired 寄存器的格式；表 7-12 对 Wired 寄存器各域进行了描述。

图 7-10 Wired 寄存器格式

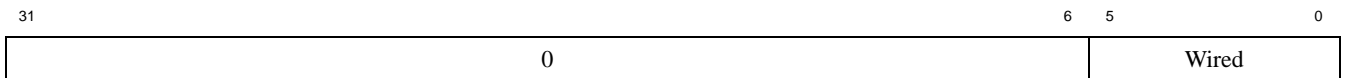


表 7-12 Wired 寄存器域描述

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

7.10 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 7-11 说明了 HWREna 寄存器的格式；表 7-13 对 HWREna 寄存器各域进行了描述。

图 7-11 HWREna 寄存器格式

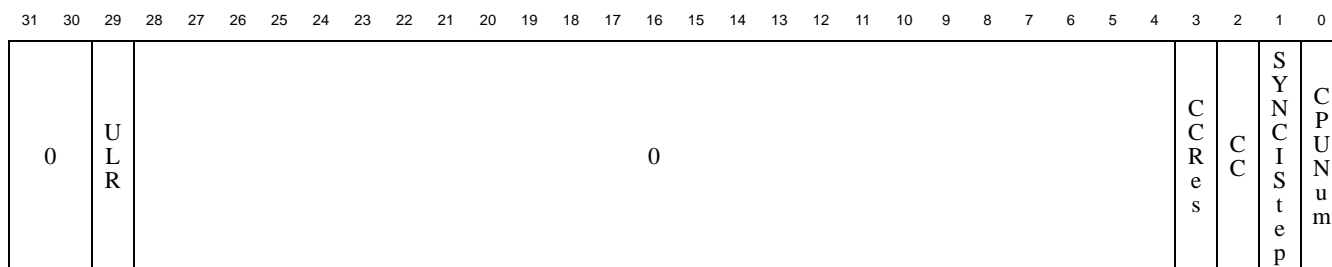


表 7-13 HWREna 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..30	只读恒为 0。	0	0
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
SYNCL-Step	1	RDHWR 1 (SYNCL_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

7.11 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 7-12 说明了 BadVAddr 寄存器的格式；表 7-14 对 BadVAddr 寄存器各域进行了描述。

图 7-12 BadVAddr 寄存器格式

63

0

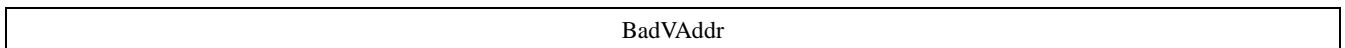


表 7-14 BadVAddr 寄存器域描述

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

7.12 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 7-13 Count 寄存器格式

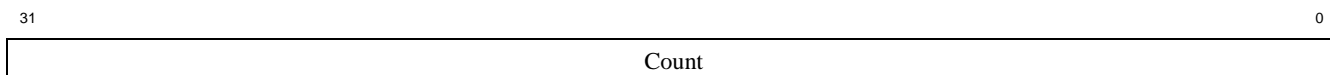


表 7-15 Count 寄存器域描述

域名称	位	功能描述	读/写	复位值
Count	31..0	内部计数器。	R/W	无

7.13 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 Modified 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。

图 7-14 说明了 EntryHi 寄存器的格式；表 7-16 对 EntryHi 寄存器各域进行了描述。

图 7-14 EntryHi 寄存器格式

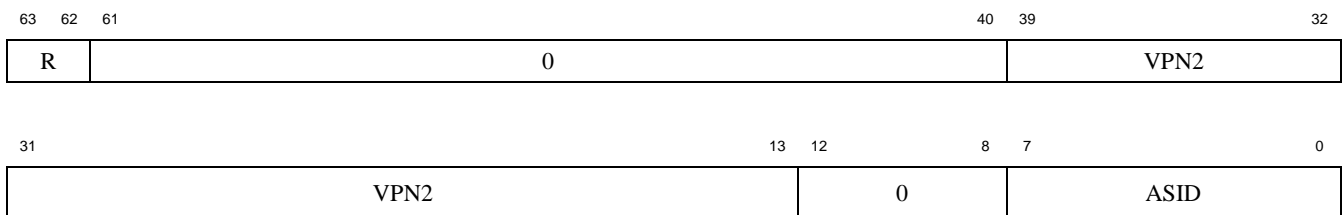


表 7-16 EntryHi 寄存器域描述

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	0x3
0	61..40	只读恒为 0。	0	0
VPN2	39..13	虚页号除 2(映射到双页)，对应虚地址的[39:13]位。	R/W	0x7FC.000
0	12..8	只读恒为 0。	0	0
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

7.14 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定时器中断，将 Cause_{IT} 置 1。

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

图 7-15 说明了 Compare 寄存器的格式；表 7-17 对 Compare 寄存器各域进行了描述。

图 7-15 Compare 寄存器格式



表 7-17 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

7.15 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 7-16 说明了 Status 寄存器的格式；表 7-18 对 Status 寄存器各域进行了描述。

图 7-16 Status 寄存器格式

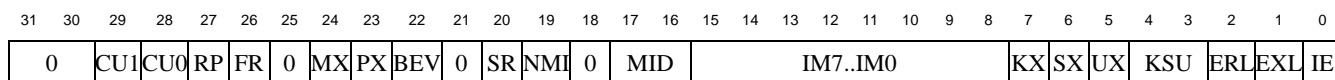


表 7-18 Status 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..30	只读恒为 0。	0	0
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	0x0
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	处理器结构级门控时钟使能位。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(NMI)。 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 CP0.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。 GS264 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	0x2
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： <ul style="list-style-type: none"> 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： <ul style="list-style-type: none"> 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

7.16 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 7-17 说明了 IntCtl 寄存器的格式；表 7-19 对 IntCtl 寄存器各域进行了描述。

图 7-17 IntCtl 寄存器格式

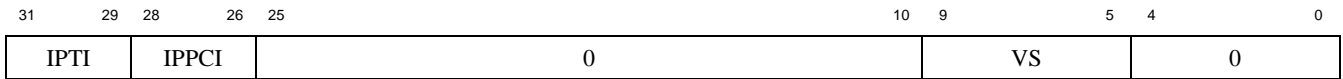


表 7-19 IntCtl 寄存器域描述

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 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 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。	R/W	0x0														
		<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">VS 编码</th> <th style="width: 50%;">向量间距</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0x00</td><td style="text-align: center;">0x000</td></tr> <tr><td style="text-align: center;">0x01</td><td style="text-align: center;">0x020</td></tr> <tr><td style="text-align: center;">0x02</td><td style="text-align: center;">0x040</td></tr> <tr><td style="text-align: center;">0x04</td><td style="text-align: center;">0x080</td></tr> <tr><td style="text-align: center;">0x08</td><td style="text-align: center;">0x100</td></tr> <tr><td style="text-align: center;">0x10</td><td style="text-align: center;">0x200</td></tr> </tbody> </table>			VS 编码	向量间距	0x00	0x000	0x01	0x020	0x02	0x040	0x04	0x080	0x08	0x100	0x10	0x200
		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。																		
0	4..0	只读恒为 0。	0	0														

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

SRSCtl 寄存器用于控制影子寄存器。因 GS264 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 7-18 说明了 SRSCtl 寄存器的格式；表 7-20 对 SRSCtl 寄存器各域进行了描述。

图 7-18 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 7-20 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。GS264 中该域只读。	R	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。GS264 中该域只读。	R	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

7.18 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 7-19 说明了 Cause 寄存器的格式；表 7-21 对 Cause 寄存器各域进行了描述。

图 7-19 Cause 寄存器格式

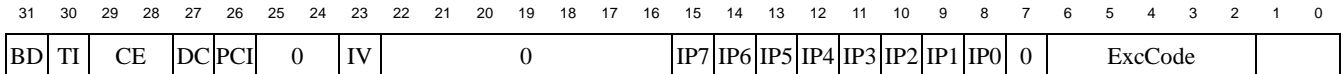


表 7-21 Cause 寄存器域描述

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：该中断线上有待处理的中断；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 7-22 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 GS264 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 GS264 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	向量指令中的浮点例外
0x0f	FPE	浮点例外。
0x10	GSExc	龙芯自定义例外，包括浮点栈例外。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	向量模块禁用例外。
0x16	MDMX	未实现。
0x17	WATCH	未实现。
0x18	MCheck	未实现。
0x19	Thread	未实现。
0x1a	DSPDis	未实现。
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

7.19 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 7-20 EPC 寄存器格式

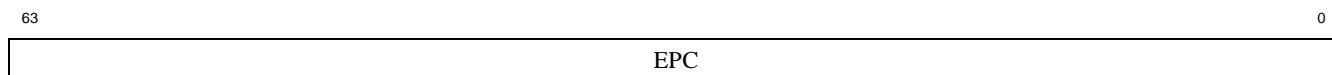


表 7-23 EPC 寄存器域描述

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

7.20 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 7-21 说明了 PRId 寄存器的格式；表 7-24 对 PRId 寄存器各域进行了描述。

图 7-21 PRId 寄存器格式

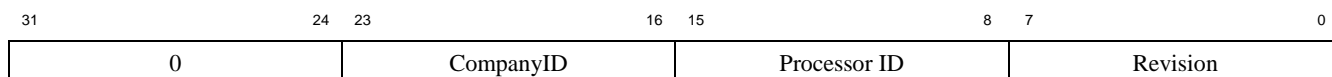


表 7-24 PRId 寄存器域描述

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。值为 0x14。	R	0x14
ProcessorID	15..8	处理器类型号。值为 0x61。	R	0x61
Revision	7..0	实现版本号。值为 0x01。	R	0x01

7.21 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 7-22 说明了 EBase 寄存器的格式；表 7-25 对 EBase 寄存器各域进行了描述。

图 7-22 EBase 寄存器格式

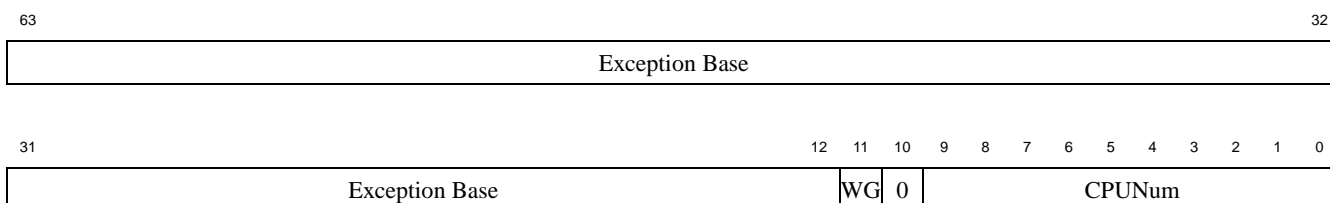


表 7-25 EBase 寄存器域描述

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 0xffff 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。

7.22 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 GS264 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 7-23 说明了 Config 寄存器的格式；表 7-26 Config 寄存器域描述表 7-26 对 Config 寄存器各域进行了描述。

图 7-23 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	SIMD 128	QM	CAM	BE	AT	AR	MT	0	VI	K0					

表 7-26 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
SIMD128	18	值为 1，表示支持 128 位向量指令。	R	0x1
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 36 页表 2-26。	R	0x1
CAM	16	值为 0，表示不包含硬件 CAM 部件，即不支持龙芯自定义 CAM 指令。	R	0x0
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 1，MMU 采用标准的 CAM TLB 形式。	R	0x1
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。GS264 支持的 Cache 属性编码请参看 82 页表 7-6。	R/W	0x2

7.23 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 7-24 说明了 Config1 寄存器的格式；表 7-27 表 7-26 Config 寄存器域描述对 Config1 寄存器各域进行了描述。

图 7-24 Config1 寄存器格式

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

表 7-27 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 Config1 寄存器的 MMUSize-1 域拼接，形成 10 比特值： Config4.VTLBSizeExt _{3..0} Config1.MMUSize-1 _{5..0} ； 拼接后的值为 63，表示 TLB 为 64 项。	R	0x3f
IS	24..22	值为 2，表示 I-Cache 每一路包含 128 行。	R	0x1
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 128 行。	R	0x1
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 0，表示不包含协处理器 2 (COP2)。	R	0x0
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

7.24 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 7-25 说明了 Config2 寄存器的格式；表 7-28 表 7-26 Config 寄存器域描述对 Config2 寄存器各域进行了描述。

图 7-25 Config2 寄存器格式

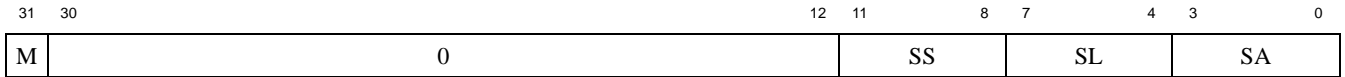


表 7-28 Config2 寄存器域描述

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..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 7，表示 S-Cache 包含 8 路。	R	0x7

7.25 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 7-26 说明了 Config3 寄存器的格式；表 7-29 表 7-26 Config 寄存器域描述对 Config3 寄存器各域进行了描述。

图 7-26 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 7-29 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。 GS264 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。	R	0x0
MSAP	28	值为 1，表示实现了向量模块(SIMD Module)。	R	0x1
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)。	R	0x0
VZ	23	值为 0，表示不支持硬件虚拟化。	R	0x0
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 0，表示未实现 MIPS DSP 模块版本 2。	R	0x0
DSP	10	值为 0，表示未实现 MIPS DSP 模块。	R	0x0
CTXTC	9	值为 0，表示未实现 ContextConfig 和 XcontextConfig 寄存器。 因不建议运行在 GS264 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

域名称	位	功能描述	读/写	复位值
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™ ASE。	R	0x0
TL	0	值为 0，表示未实现 Trace 逻辑。	R	0x0

7.26 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 7-27 说明了 Config4 寄存器的格式；表 7-30 对 Config4 寄存器各域进行了描述。

图 7-27 Config4 寄存器格式

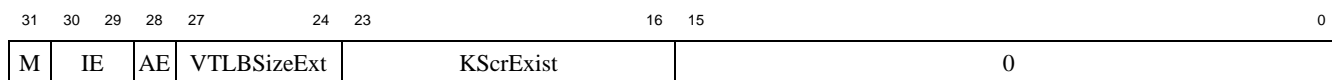


表 7-30 Config4 寄存器域描述

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	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-1 _{5..0} ； 拼接后的值为 63，表示 TLB 为 64 项。	R	0x0
KScrExist	23..16	值为 0b00001100，表示 KScratch1~2 寄存器(CP0 Register 31, Selct 2~3)可以被处于核心态的软件访问。	R	0x0c
0	15..0	只读恒为 0。	0	0

7.27 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 7-28 说明了 Config5 寄存器的格式；表 7-31 表 7-26 Config 寄存器域描述对 Config5 寄存器各域进行了描述。

图 7-28 Config5 寄存器格式

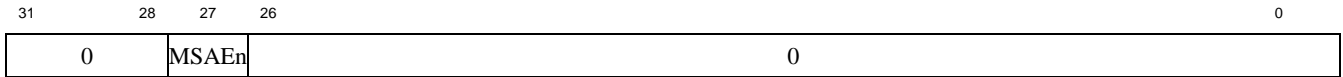


表 7-31 Config5 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31	只读恒为 0。	0	0
MSAEn	27	MIPS 向量模块(SIMD Module)使能位。0: 禁用; 1: 可用。	R/W	0x0
0	26..0	只读恒为 0。	0	0

7.28 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 7-29 说明了 GSConfig 寄存器的格式；表 7-32 对 GSConfig 寄存器各域进行了描述。

图 7-29 GSConfig 寄存器格式

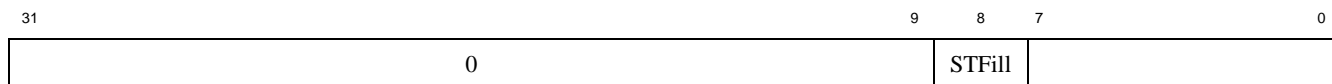


表 7-32 GSConfig 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..9	只读恒为 0。	0	0
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
0	7..0	只读恒为 0。	0	0

7.29 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 7-30 说明了 LLAddr 寄存器的格式；表 7-33 对 LLAddr 寄存器各域进行了描述。

图 7-30 LLAddr 寄存器格式

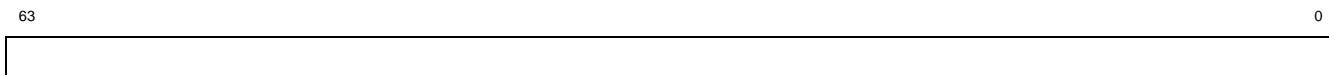


表 7-33 LLAddr 寄存器域描述

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

7.30 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 7-31 说明了 XContext 寄存器的格式；表 7-34 对 XContext 寄存器各域进行了描述。

图 7-31 XContext 寄存器格式

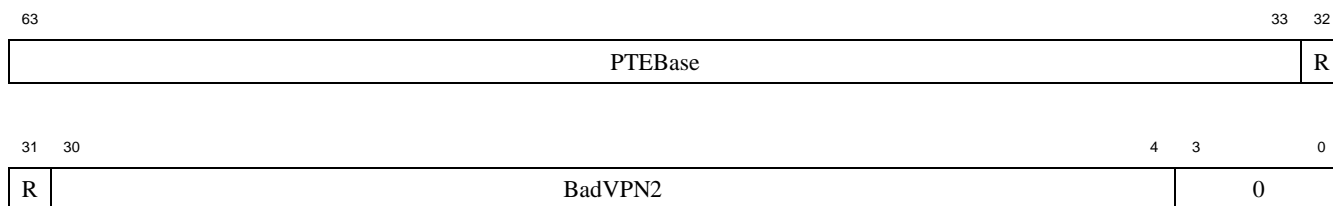


表 7-34 XContext 寄存器域描述

域名称	位	功能描述	读/写	复位值
PTEBase	63..33	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	32..31	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	30..4	当发生 TLB 例外时，存放出错虚地址的 39..13 位。	R	无
0	3..0	只读恒为 0。	0	0

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

Diag 寄存器是龙芯自定义的寄存器，用于控制处理器核内部部分微结构的操作特性。

图 7-32 说明了 Diag 寄存器的格式；表 7-35 表 7-26 Config 寄存器域描述对 Diag 寄存器各域进行了描述。

图 7-32 Diag 寄存器格式

31	9	8	7	3	2	1	0
			UCAC	0	IT LB	0	RAS

表 7-35 Diag 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..9	只读恒为 0。	0	0
UCAC	8	置 1 时，在 User 模式下执行 CACHE0、CACHE1、CACHE3、CACHE21、CACHE23 指令将不触发协处理器例外(CpU)。	R/W	0x0
0	7..3	只读恒为 0。	0	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
0	1	只读恒为 0。	0	0
RAS	0	置 1 时禁用 RAS 对 jr31 进行分支预测。	R/W	0x0

7.32 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 7-33 说明了 GSCause 寄存器的格式；表 7-36 对 GSCause 寄存器各域进行了描述。

图 7-33 GSCause 寄存器格式

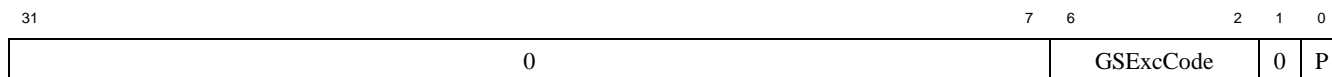


表 7-36 GSCause 寄存器域描述

域名称	位	功能描述	读/写	复位值
0	31..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 7-37 GSExcCode 编码及其对应例外类型

ExcCode	助记符	描述
0x00	IS	浮点栈例外

7.33 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。

7.34 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.DBID 置为 1。

图 7-34 说明了 DEPC 寄存器的格式；表 7-38 对 DEPC 寄存器各域进行了描述。

图 7-34 DEPC 寄存器格式

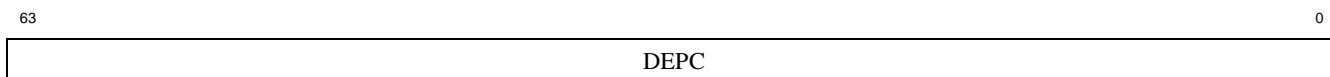


表 7-38 DEPC 寄存器域描述

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

7.35 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 7-39 所示。

表 7-39 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

图 7-35 说明了 PerfCnt Control 寄存器的格式；表 7-40 对 PerfCnt Control 寄存器各域进行了描述。

图 7-36 说明了 PerfCnt Counter 寄存器的格式；表 7-41 对 PerfCnt Counter 寄存器各域进行了描述。

图 7-35 PerfCnt Control 寄存器格式

31	30	29													15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	W	0	Event												IE	U	S	K	EXL											

表 7-40 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 ⁹
W	30	恒为 1，表示 PerfCnt Counter 寄存器的位宽是 64 位。	R	0x1
0	29..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

⁹ 对于 Control 0~Control 2，该位复位值为 1；对于 Control3，该位复位值为 0。

图 7-36 PerfCnt Counter 寄存器格式

63

0

Event Count

表 7-41 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。	R/W	0x0

GS264 处理器核的性能计数器事件定义如表 7-42 所示。

表 7-42 处理器核性能计数器事件定义

事件号	事件描述
1	提交的指令数
2	提交的分支指令数
3	提交的非 likely 类分支指令数
4	提交的 likely 类分支指令数
5	提交的 JR \$31 指令数
6	提交的 JR.HB 和 JALR.HB 指令数
7	提交的非 return 非 barrier 间接跳转指令数 ¹⁰
8	提交的分支指令中出错的条数
9	提交的非 likely 类分支指令中出错的条数
10	提交的 JR \$31 指令中出错的条数
11	提交的非 return 非 barrier 间接跳转指令中出错的条数 ¹¹
12	发射的浮点操作数
13	发射的向量操作数
14	提交的 load 指令数
15	提交的 store 指令数
16	发出的 uncache 访存次数
17	发射的定点部件操作数
18	SC 指令的执行次数
19	SC 指令执行失败的次数
20	软件可见 TLB 访问的次数
21	因地址非对齐造成的 ADE 例外次数
22	TLB 重填例外次数
23	由取指触发的 TLB 重填例外次数

¹⁰ 不包括 JR \$31、JR.HB、JALR.HB 的 JR 类指令

¹¹ 不包括 JR \$31、JR.HB、JALR.HB 的 JR 类指令

事件号	事件描述
24	由 Load 操作触发的 TLB 重填例外次数
25	由 Store 操作触发的 TLB 重填例外次数
26	ITLB 重填次数
28	I-Cache 访问次数
29	I-Cache 访问不命中的次数
31	D-Cache 访问不命中的次数
34	多核一致性请求造成的 D-Cache 替换次数
35	多核一致性请求造成的 D-Cache 替换中数据为脏的次数
64	因 I-Cache 不命中造成的前端流水线清空的次数
65	因 I-Cache 不命中造成的前端流水线断流的周期数
66	因分支预测 taken 造成的前端流水线清空的次数
67	因分支预测 taken 造成的前端流水线断流的周期数
68	前端送到后端的指令数
69	因为 roq 满而被阻塞在 map 阶段的周期数
70	因为 brq 满而被阻塞在 map 阶段的周期数
71	因为 cp0q 满而被阻塞在 map 阶段的周期数
72	因为 ftq 满而被阻塞在 map 阶段的周期数
73	因为 fxq 满而被阻塞在 map 阶段的周期数
74	因为 grmt 满而被阻塞在 map 阶段的周期数
75	因为 frmt 满而被阻塞在 map 阶段的周期数
76	因为长延迟 load 非执行完而被阻塞在 map 阶段的周期数 ¹²
77	0 号发射总线发射的操作数
78	1 号发射总线发射的操作数
79	2 号发射总线发射的操作数
80	3 号发射总线发射的操作数
81	因为分支误预测造成的流水级停顿的周期数 ¹³
82	因为例外造成的流水线停顿的周期数 ¹⁴
83	STALLF 内部例外发生的次数
84	因 STALLF 内部例外造成的流水线停顿的周期数
85	MISPEC 内部例外发生的次数
86	因 MISPEC 内部例外造成的流水线停顿的周期数
87	对核外发起的 cached 读请求的次数
88	对核外发起的 cached 读请求的累计延迟 ¹⁵
89	软件预取对核外发起的请求次数
90	store fill 成功填满一项的次数
91	因为 missq 满导致 cache miss 请求被阻塞的周期数
92	对核外发起的 uncached 读请求的次数

¹² roq 头和 cp0q 头是同一条 load 指令，且此时 roq、brq、cp0q、fxq、ftq 有任一队列满

¹³ 停顿指 brbus_brerr 发出清空流水线后至首条指令进入 roq

¹⁴ 停顿指 exbus_ex 发出清空流水线后至首条指令进入 roq

¹⁵ 如果同时有多个请求在等待结果，那么每个周期所有请求各自均计延迟

事件号	事件描述
93	对核外发起的 uncached 读请求的延迟
94	对核外发起的 uncached 写请求的次数
95	对核外发起的 uncached 写请求的延迟
96	对核外发起的 uncached acc 请求的次数
97	对核外发起的 uncached acc 请求中凑满一个 cacheline 大小的次数
98	因为 ucq 满导致 uncache 请求被阻塞的周期数
99	因为 wtbuf 满导致被阻塞的周期数
100	因为 wtqid 队列满导致被阻塞的周期数
101	对核外发起的 cached 写请求的次数
102	对核外发起的 cached 写请求的累计延迟 ¹⁶

¹⁶ 延迟指从写地址请求发出到对应的 bvalid 回来

7.36 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。

图 7-37 说明了 ErrCtl 寄存器的格式；表 7-43 表 7-26 Config 寄存器域描述对 ErrCtl 寄存器各域进行了描述。

图 7-37 ErrCtl 寄存器格式

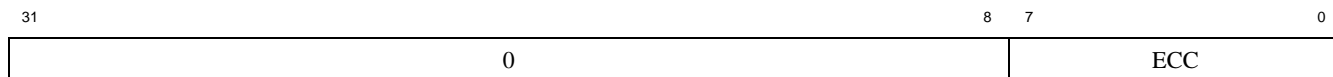


表 7-43 ErrCtl 寄存器域描述

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	软件读写的 ECC 值。	R/W	无

7.37 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。GS264 对于 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 7-38 说明了 CacheErr 寄存器用于 I-Cache 校验错信息时的格式；表 7-44 表 7-26 Config 寄存器域描述对该情况下寄存器各域进行了描述。

图 7-38 CacheErr 寄存器用于 I-Cache 校验错信息时的格式

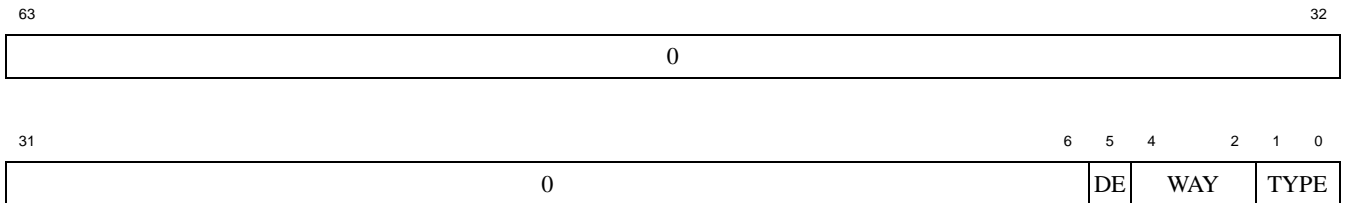


表 7-44 CacheErr 寄存器用于 I-Cache 校验错信息时的域描述

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 奇偶校验出错标识。1: Data 部分校验出错；0: Tag 部分校验出错。	R	0
WAY	4..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 7-39 说明了 CacheErr 寄存器用于 I-Cache 校验错信息时的格式；表 7-45 对该情况下寄存器各域进行了描述。

图 7-39 CacheErr 寄存器用于 D-Cache 校验错信息时的格式

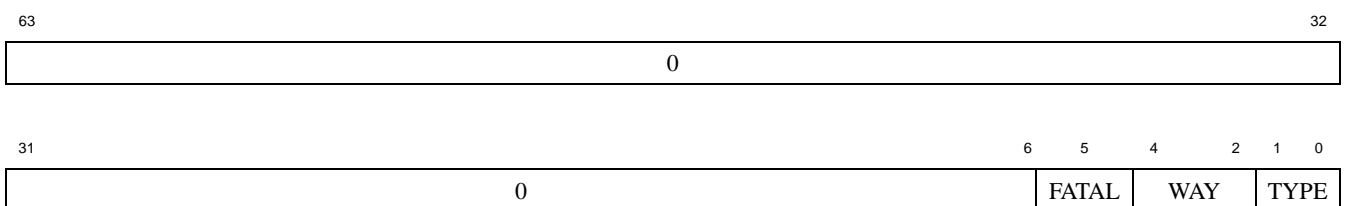


表 7-45 CacheErr 寄存器用于 D-Cache 校验错信息时的域描述

域名称	位	功能描述	读/写	复位值
0	63..6	只读恒为 0。	0	0
FATAL	5	D-Cache 校验出错标识。1: 2 比特及以上错；0: 1 比特错（硬件自动可纠正）。	R	0
WAY	4..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

7.38 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 7-40 说明了 CacheErr1 寄存器的格式；表 7-46 表 7-26 Config 寄存器域描述对 CacheErr1 寄存器各域进行了描述。

图 7-40 CacheErr1 寄存器格式

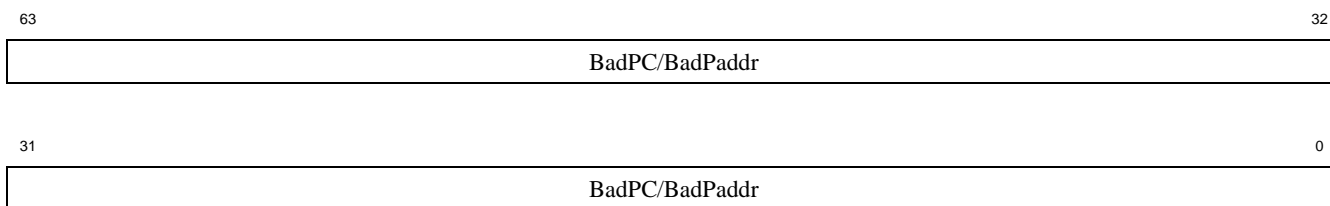


表 7-46 CacheErr1 寄存器域描述

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

7.39 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 7-41 说明了 TagLo 寄存器用于访问 I-Cache Tag 时的格式;表 7-47 表 7-26 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-41 TagLo 寄存器用于访问 I-Cache Tag 时的格式

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

表 7-47 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 7-42 说明了 TagLo 寄存器用于访问 D-Cache Tag 时的格式;表 7-48 对这种情况下寄存器各域进行了描述。

图 7-42 TagLo 寄存器用于访问 D-Cache Tag 时的格式

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

表 7-48 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 7-43 说明了 TagLo 寄存器用于访问 S-Cache Tag 时的格式; 表 7-49 对这种情况下寄存器各域进行了描述。

图 7-43 TagLo 寄存器用于访问 S-Cache Tag 时的格式

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

表 7-49 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。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 7-44 说明了 TagLo 寄存器用于访问各级 Cache 中 Data 部分时的格式; 表 7-50 对这种情况下寄存器各域进行了描述。

图 7-44 TagLo 寄存器用于访问各级 Cache Data 时的格式

31	DATA	0
----	------	---

表 7-50 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 GS264 的 Cache 一致性要求。否则处理器的行为将不确定。

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

在 GS264 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，DataLo 允许软件读、写，但不参与其它任何操作。

图 7-45 说明了 DataLo 寄存器的格式；表 7-51 表 7-26 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-45 DataLo 寄存器用于 I-Cache 访问时的格式

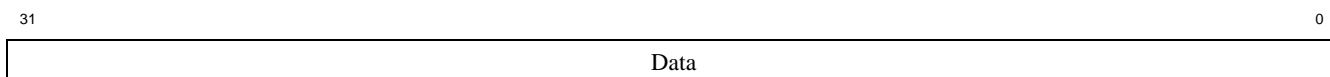


表 7-51 DataLo 寄存器用于 I-Cache 访问时的域描述

域名称	位	功能描述	读/写	复位值
Data	31..0	32 位数值。	R/W	无

7.41 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 7-46 说明了 TagHi 寄存器用于访问各级 Cache Tag 时的格式;表 7-52 对这种情况下寄存器各域进行了描述。

图 7-46 TagHi 寄存器用于访问各级 Cache Tag 时的格式

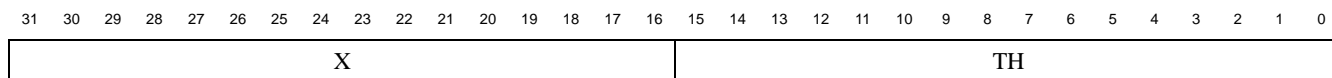


表 7-52 TagHi 寄存器用于访问各级 Cache Tag 时的域描述

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 7-47 说明了 TagLo 寄存器用于访问各级 Cache Data 时的格式;表 7-53 对这种情况下寄存器各域进行了描述。

图 7-47 TagHi 寄存器用于访问各级 Cache Data 时的格式

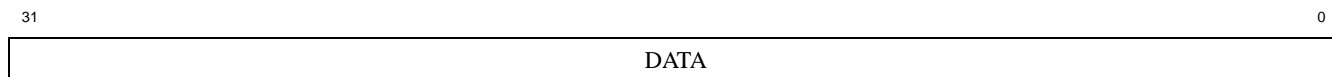


表 7-53 TagHi 寄存器用于访问各级 Cache Data 时的域描述

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 GS264 的 Cache 一致性要求。否则处理器的行为将不确定。

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

在 GS264 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，DataHi 允许软件读、写，但不参与其它任何操作。

图 7-48 说明了 DataHi 寄存器用于访问 I-Cache 时的格式；表 7-54 表 7-26 Config 寄存器域描述对这种情况下寄存器各域进行了描述。

图 7-48 DataHi 寄存器用于 I-Cache 访问时的格式

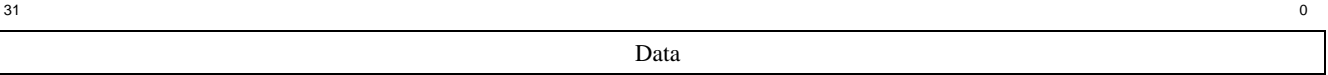


表 7-54 DataHi 寄存器用于 I-Cache 访问时的域描述

域名称	位	功能描述	读/写	复位值
Data	31..0	32 位数值。	R/W	无

7.43 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 7-49 说明了 ErrorEPC 寄存器的格式；表 7-55 对 ErrorEPC 寄存器各域进行了描述。

图 7-49 ErrorEPC 寄存器格式

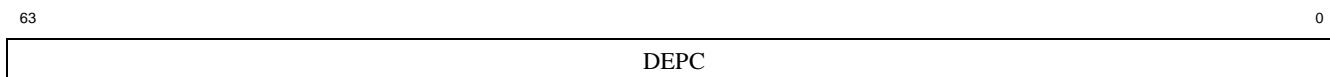


表 7-55 ErrorEPC 寄存器域描述

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

7.44 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 7-50 说明了 DESAVE 寄存器的格式；表 7-56 对 DESAVE 寄存器各域进行了描述。

图 7-50 DESAVE 寄存器格式

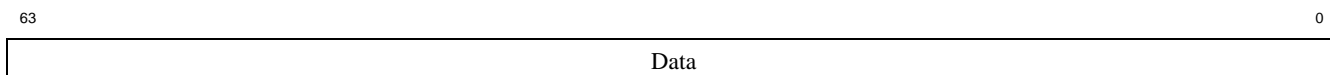


表 7-56 DESAVE 寄存器域描述

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

7.45 KScratch1~2 寄存器 (CP0 Register 31, Select 2~3)

KScratch1~2 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 KScratch1~2，但是此时软件可以通过 DESAVE 寄存器暂存数据。

图 7-51 说明了 KScratch 寄存器的格式；表 7-57 对 KScratch 寄存器各域进行了描述。

图 7-51 KScratch n 寄存器格式

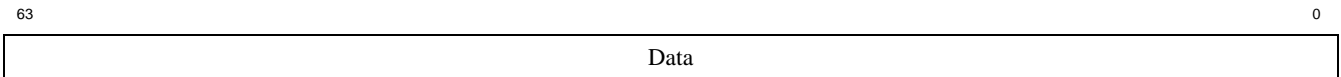


表 7-57 KScratch n 寄存器域描述

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