LOONGSON

龙芯 3A1000 处理器用户手册

下册

GS464 处理器核 V1.4

2015 年 10 月

龙芯中科技术有限公司

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

www.loongson.cn

版权声明

免责声明

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

龙芯中科技术有限公司

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

阅读指南

《龙芯 3A1000 处理器用户手册》分为上册和下册。

《龙芯 3A1000 处理器用户手册》上册分为两部分，第一部分（第 1 章~第 10 章）介绍龙芯 3A1000 多核处理器架构与寄存器描述，对芯片系统架构、主要模块的功能与配置、寄存器列表及位域进行详细说明；第二部分（第 11 章~第 16 章）是系统软件编程指南，对 BIOS 和操作系统开发过程中的常见问题进行专题介绍。

《龙芯 3A1000 处理器用户手册》下册，从系统软件开发者角度详细介绍龙芯 3A1000 所采用的 GS464 高性能处理器核。

修订历史

文档更新记录	文档名:	龙芯 3A1000 处理器用户手册 ——下册
	版本号	V1.4
	创建人:	研发中心
	创建日期 ：	2015-10-08

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2011-06-24	研发中心	V1.0	初稿完成
2	2011-10-18	研发中心	V1.1	审核、校对
3	2011-11-24	研发中心	V1.2	修改封面
4	2012-04-28	研发中心	V1.3	修正第 5 章“内存管理”48 位地址空间相关错误
5	2015-10-08	研发中心	V1.4	修改第 2 章和第 7 章中的指令描述

手册信息反馈: service@loongson.cn

目 录

1 结构概述.....12
2 龙芯 GS464 处理器核指令集概述.....15
2.1 MIPS64 兼容指令列表.....15
2.1.1 访存指令.....15
2.1.2 运算指令.....16
2.1.3 分支和跳转指令.....19
2.1.4 协处理器指令.....20
2.1.5 其它指令.....21
2.2 MIPS64 兼容指令实现相关说明.....22
2.2.1 存在实现差异的指令.....22
2.2.2 禁用指令.....24
2.3 自定义扩展指令.....25
2.3.1 自定义扩展访存指令.....25
2.3.2 自定义扩展乘除运算指令.....26
2.3.3 自定义扩展 X86 二进制翻译加速指令.....27
2.3.4 自定义扩展 64 位多媒体加速指令.....28
2.3.5 自定义扩展杂项指令.....31
3 CP0 控制寄存器.....32
3.1 Index 寄存器(0，0).....33
3.2 Random 寄存器(1，0).....34
3.3 EntryLo0 (2，0)以及 EntryLo1 (3，0)寄存器.....34
3.4 Context (4，0).....35
3.5 PageMask 寄存器(5，0).....36
3.6 PageGrain 寄存器(5，1).....36
3.7 Wired 寄存器(6，0).....37
3.8 HWREna 寄存器 (7，0).....38
3.9 BadVAddr 寄存器(8，0).....38
3.10 Count 寄存器(9，0)以及 Compare 寄存器(11，0).....39
3.11 EntryHi 寄存器(10，0).....39
3.12 Status 寄存器(12，0).....40
3.13 IntCtl 寄存器 (12，1).....42
3.14 SRSCtl 寄存器 (12，2).....43
3.15 Cause 寄存器(13，0).....43
3.16 Exception Program Counter 寄存器(14，0).....45
3.17 Processor Revision Identifier (PRID)寄存器(15，0).....45
3.18 EBase 寄存器(15，1).....46
3.19 Config 寄存器(16，0).....47

3.20 Config1 寄存器(16，1) 48
3.21 Config 2 寄存器(16，2) 50
3.22 Config 3 寄存器(16，3) 52
3.23 Load Linked Address (LLAddr)寄存器(17，0) 54
3.24 XContext 寄存器(20，0) 54
3.25 Diagnostic 寄存器(22，0) 55
3.26 Debug 寄存器(23，0) 55
3.27 Debug Exception Program Counter 寄存器(24，0) 57
3.28 Performance Counter 寄存器(25，0/1/2/3) 57
3.29 ECC 寄存器(26，0) 60
3.30 CacheErr 寄存器(27，0/1) 60
3.31 TagLo(28)和 TagHi (29)寄存器 61
3.32 DataLo (28，1)和 DataHi (29，1) 寄存器 62
3.33 ErrorEPC 寄存器(30，0) 63
3.34 DESAVE 寄存器(31，0) 63
3.35 CP0 指令 63
4 CACHE 的组织和操作 65
4.1 Cache 概述 65
4.1.1 非阻塞 Cache 65
4.1.2 替换策略 66
4.1.3 Cache 的参数 66
4.2 一级指令 Cache 66
4.2.1 指令 Cache 的组织 67
4.2.2 指令 Cache 的访问 67
4.3 一级数据 Cache 68
4.3.1 数据 Cache 的组织 68
4.3.2 数据 Cache 的访问 68
4.4 二级 Cache 69
4.4.1 二级 Cache 的组织 69
4.4.2 二级 Cache 的访问 70
4.5 Cache 算法和 Cache 一致性属性 70
4.5.1 非高速缓存(Uncached，一致性代码 2) 71
4.5.2 一致性高速缓存(Cacheable coherent，一致性代码 3) 71
4.5.3 非高速缓存加速 (Uncached Accelerated，一致性代码 7) 71
4.6 Cache 一致性 71
5 内存管理 73
5.1 快速查找表 TLB 73
5.1.1 JTLB 73

5.1.2 指令 TLB 73
5.1.3 命中和失效 74
5.1.4 多项命中 74
5.2 处理器模式 74
5.2.1 处理器工作模式 74
5.2.2 地址模式 75
5.2.3 指令集模式 75
5.2.4 尾端模式 75
5.3 地址空间 75
5.3.1 虚拟地址空间 75
5.3.2 物理地址空间 75
5.3.3 虚实地址转换 75
5.3.4 用户地址空间 77
5.3.5 管理地址空间 78
5.3.6 内核地址空间 80
5.4 系统控制协处理器 82
5.4.1 TLB 表项的格式 82
5.4.2 CP0 寄存器 84
5.4.3 虚拟地址到物理地址的转换过程 84
5.4.4 TLB 失效 85
5.4.5 TLB 指令 86
5.4.6 代码例子 86
5.5 物理地址空间分布 87
6 处理器例外 88
6.1 例外的产生及返回 88
6.2 例外向量位置 88
6.3 例外优先级 89
6.4 冷重置例外 90
6.5 NMI 例外 91
6.6 地址错误例外 92
6.7 TLB 例外 92
6.8 TLB 重填例外 93
6.9 TLB 无效例外 94
6.10 TLB 修改例外 94
6.11 Cache 错误例外 95
6.12 总线错误例外 96
6.13 整型溢出例外 97
6.14 陷阱例外 97

6.15 系统调用例外.....98
6.16 断点例外.....98
6.17 保留指令例外.....99
6.18 协处理器不可用例外.....99
6.19 浮点例外.....100
6.20 EJTAG 例外.....101
6.21 中断例外.....101
7 浮点协处理器.....103
7.1 概述.....103
7.2 FPU 寄存器.....104
7.2.1 浮点寄存器.....104
7.2.2 FIR 寄存器（CP1，0）.....105
7.2.3 FCSR 寄存器（CP1，31）.....107
7.2.4 FCCR 寄存器（CP1，25）.....109
7.2.5 FEXR 寄存器（CP1，26）.....109
7.2.6 FENR 寄存器（CP1，28）.....110
7.3 浮点指令.....110
7.3.1 MIPS64 兼容浮点指令列表.....110
7.3.2 MIPS64 兼容浮点指令实现相关说明.....113
7.3.3 龙芯自定义扩展浮点指令.....113
7.4 浮点部件格式.....114
7.4.1 浮点格式.....114
7.5 FPU 指令流水线概述.....116
7.6 浮点例外处理.....117
8 性能分析和优化.....122
8.1 用户指令的延迟和循环间隔.....122
8.2 指令扩充和使用注意事项.....123
8.3 编译器使用提示.....124
8.4.1 指令对齐.....124
8.4.2 转移指令的处理.....125
8.4.3 指令流密度的提高.....126
8.4.4 指令调度.....126
8.5 存储器访问.....126
8.6 其他提示.....127

图目录

图 3-1 Index 寄存器 33
图 3-2 Random 寄存器 34
图 3-3 EntryLo0 和 EntryLo1 寄存器 35
图 3-4 Context 寄存器 35
图 3-5 PageMask 寄存器 36
图 3-6 PageGrain 寄存器 37
图 3-7 Wired 寄存器界限 37
图 3-8 Wired 寄存器 38
图 3-9 HWREna 寄存器 38
图 3-10 BadVAddr 寄存器 38
图 3-11 Count 寄存器 39
图 3-12 Compare 寄存器 39
图 3-13 EntryHi 寄存器 39
图 3-14 Status 寄存器 40
图 3-15 IntCtl 寄存器 42
图 3-16 SRSCtl 寄存器 43
图 3-17 Cause 寄存器 43
图 3-18 EPC 寄存器 45
图 3-19 Processor Revision Identifier 寄存器 46
图 3-20 Ebase 寄存器 46
图 3-21 Config 寄存器 47
图 3-22 Config1 寄存器 48
图 3-23 Config2 寄存器 51
图 3-24 Config3 寄存器 53
图 3-25 LLAddr 寄存器 54
图 3-26 XContext 寄存器 54
图 3-27 Diagnostic 寄存器 55
图 3-28 Debug 寄存器 56
图 3-29 DEPC 寄存器 57
图 3-30 性能计数器寄存器 58
图 3-31 ECC 寄存器 60
图 3-32 CacheErr 寄存器 61

图 3-33 CacheErr1 寄存器 61
图 3-34 TagLo 和 TagHi 寄存器(P-Cache) 62
图 3-35 DataLo 和 DataHi 寄存器 62
图 3-36 ErrorEPC 寄存器 63
图 3-37 DESAVE 寄存器 63
图 4-1 指令 Cache 的组织 67
图 5-1 虚实地址转换概览 76
图 5-2 64 位模式虚拟地址转换 77
图 5-3 用户模式下用户虚拟地址空间概况 78
图 5-4 管理模式下用户空间和管理空间 79
图 5-5 内核模式下的用户、管理、内核地址空间概况 81
图 5-6 TLB 表项 82
图 5-7 PageMask 寄存器 82
图 5-8 EntryHi 寄存器 83
图 5-9 EntryLo0 和 EntryLo1 寄存器 83
图 5-10 TLB 地址转换 85
图 7-1 龙芯 3A1000 体系结构中功能单元的组织构成 104
图 7-2 浮点寄存器格式 105
图 7-3 FIR 寄存器 106
图 7-4 FCSR 寄存器 107
图 7-5 FCCR 寄存器 109
图 7-6 FEXR 寄存器 110
图 7-7 FENR 寄存器 110
图 7-8 浮点格式 115

表目录

表 2-1 CPU 指令集：访存指令.....15
表 2-2 CPU 指令集：算术指令（ALU 立即数）.....16
表 2-3 CPU 指令集：算术指令（3 操作数）.....17
表 2-4 CPU 指令集：算术指令（2 操作数）.....17
表 2-5 CPU 指令集：乘法和除法指令.....18
表 2-6 CPU 指令集：移位指令.....18
表 2-7 CPU 指令集：跳转和分支指令.....20
表 2-8 CPU 指令集：CP0 指令.....21
表 2-9 CPU 指令集：特殊指令.....21
表 2-10 CPU 指令集：异常指令.....21
表 2-11 CPU 指令集：条件移动指令.....22
表 2-12 CPU 指令集：其它指令.....22
表 2-13 存在实现差异的指令.....22
表 2-14 禁用指令.....25
表 2-15 自定义扩展访存指令.....25
表 2-16 自定义扩展乘除运算指令.....26
表 2-17 自定义扩展 X86 二进制翻译加速指令.....27
表 2-18 自定义扩展 64 位多媒体加速指令.....28
表 2-19 自定义扩展杂项指令.....31
表 3-1 CP0 寄存器.....32
表 3-2 Index 寄存器各域描述.....33
表 3-3 Random 寄存器各域.....34
表 3-4 EntryLo 寄存器域.....35
表 3-5 Context 寄存器域.....35
表 3-6 不同页大小的掩码（Mask）值.....36
表 3-7 PageGrain 寄存器域.....37
表 3-8 Wired 寄存器域.....38
表 3-9 Mask 域和硬件寄存器的对应关系.....38
表 3-10 EntryHi 寄存器域.....39
表 3-11 Status 寄存器域.....40
表 3-12 VS 域的编码与向量空间对应关系.....42
表 3-13 SRSCtl 寄存器的域.....43

表 3-14 Cause 寄存器域 43
表 3-15 Cause 寄存器的 ExcCode 域 44
表 3-16 PRId 寄存器域 46
表 3-17 Ebase 寄存器域 46
表 3-18 Config 寄存器域 47
表 3-19 Config1 寄存器域 48
表 3-20 Config2 寄存器域 51
表 3-21 Config3 寄存器域 53
表 3-22 XContext 寄存器域 54
表 3-23 Diagnostic 寄存器域 55
表 3-24 Debug 寄存器的域 56
表 3-25 性能计数器列表 58
表 3-26 计数使能位定义 58
表 3-27 计数器 0 事件 59
表 3-28 计数器 1 事件 59
表 3-29 ECC 寄存器的域 60
表 3-30 CacheErr 寄存器的域 61
表 3-31 CacheErr1 寄存器的域 61
表 3-32 Cache Tag 寄存器域 62
表 3-33 CP0 指令 63
表 4-1 Cache 参数 66
表 4-2 龙芯 3 号 Cache 的一致性属性 70
表 5-1 处理器的工作模式 74
表 5-2 TLB 页的 C 位的值 84
表 5-3 内存管理相关的 CP0 寄存器 84
表 5-4 TLB 指令 86
表 6-1 例外向量基址 88
表 6-2 例外向量偏移 89
表 6-3 例外优先顺序 89
表 7-1 FIR 寄存器域 106
表 7-2 FCSR 寄存器域 107
表 7-3 舍入模式位解码 109
表 7-4 MIPS64 的 FPU 指令集 110
表 7-5 自定义扩展浮点访存指令 113

表 7-6 自定义扩展浮点格式转换指令 114
表 7-7 计算单精度和双精度格式的浮点数的值的公式 115
表 7-8 浮点格式参数值 116
表 7-9 最大数和最小数的浮点值 116
表 7-10 例外的默认处理 118

1 结构概述

GS464 是一款实现 64 位 MIPS64 指令集的通用 RISC 处理器 IP。GS464 的指令流水线每个时钟周期取四条指令进行译码，并且动态地发射到五个全流水的功能部件中。虽然指令在保证依赖关系的前提下进行乱序执行，但指令的提交还是按照程序原来的顺序以保证精确例外和访存顺序执行。

四发射的超标量结构使得指令流水线中指令和数据相关问题十分突出，GS464 采用乱序执行技术和激进的存储系统设计来提高流水线的效率。

乱序执行技术包括寄存器重命名技术、动态调度技术和转移预测技术。寄存器重命名解决 WAR（读后写）和 WAW（写后写）相关，并用于例外和错误转移预测引起的精确现场恢复，GS464 分别通过 64 项的物理寄存器堆进行定点和浮点寄存器的重命名。动态调度根据指令操作数准备好的次序而不是指令在程序中出现的次序来执行指令，减少了 RAW（写后读）相关引起的阻塞，GS464 有一个 16 项的定点保留站和一个 16 项的浮点保留站用于乱序发射，并通过一个 64 项的 Reorder 队列（简称 ROQ）实现乱序执行的指令按照程序的次序提交。转移预测通过预测转移指令是否成功跳转来减少由于控制相关引起的阻塞，GS464 使用 16 项的转移目标地址缓冲器（Branch Target Buffer，简称 BTB），2K 项的转移历史表（Branch History Table，简称 BHT），9 位的全局历史寄存器（Global Histiry Registor，简称 GHR），和 4 项的返回地址栈（Return Address Stack，简称 RAS）进行转移预测。

GS464 先进的存储系统设计可以有效地提高流水线的效率。GS464 的一级 Cache 由 64KB 的指令 Cache 和 64KB 的数据 Cache 组成，均采用四路组相联的结构。GS464 的 TLB 有 64 项，采用全相联结构，每项可以映射一个奇页和一个偶页，页大小在 4KB 到 16MB 之间可变。GS464 通过 24 项的访存队列以及 8 项的访存失效队列（Miss Queue）来动态地解决地址依赖，实现访存操作的乱序执行、非阻塞 Cache、取数指令猜测执行（Load Speculation）等访存优化技术。GS464 支持 128 位的访存操作，其虚地址和物理地址均为 48 位。

GS464 有两个定点功能部件和两个浮点功能部件。每个浮点部件都可以全流水地执行 64 双精度的浮点乘加操作，并通过浮点指令的 fmt 域的扩展执行 32 位和 64 位的定点指令。

GS464 支持 MIPS 公司的 EJTAG 调试规范，采用标准的 AXI 接口，其指令 Cache 实现了奇偶校验，数据 Cache 实现了 ECC 校验。上述特点可以增加 GS464 的适用范围。

GS464 的基本流水线包括取指、预译码、译码、寄存器重命名、调度、发射、读寄存器、

执行、提交等 9 级，每一级流水包括如下操作。

- **取指流水级**用程序计数器 PC 的值去访问指令 Cache 和指令 TLB，如果指令 Cache 和指令 TLB 都命中，则把四条新的指令取到指令寄存器 IR。
- **预译码流水级**主要对转移指令进行译码并预测跳转的方向。
- **译码流水级**把 IR 中的四条指令转换成 GS464 的内部指令格式送往寄存器重命名模块。
- **寄存器重命名流水级**为逻辑目标寄存器分配一个新的物理寄存器，并将逻辑源寄存器映射到最近分配给该逻辑寄存器的物理寄存器。
- **调度流水级**将重命名的指令分配到定点或浮点保留站中等待执行，同时送到 ROQ 中用于执行后的顺序提交；此外，转移指令和访存指令还分别被送往转移队列和访存队列。
- **发射流水级**从定点或浮点保留站中为每个功能部件选出一条所有操作数都准备好的指令；在重命名时操作数没准备好的指令，通过侦听结果总线和 forward 总线等待它的操作数准备好。
- **读寄存器流水级**为发射的指令从物理寄存器堆中读取相应的源操作数送到相应的功能部件。
- **执行流水级**根据指令的类型执行指令并把计算结果写回寄存器堆；结果总线还送往保留站和寄存器重命名表，通知相应的寄存器值已经可以使用了。
- **提交流水级**按照 Reorder 队列记录的程序的顺序提交已经执行完的指令，GS464 最多每拍可以提交四条指令，提交的指令送往寄存器重命名表用于确认它的目的寄存器的重命名关系并释放原来分配给同一逻辑寄存器的物理寄存器，并送往访存队列允许那些提交的存数指令写入 Cache 或内存。

上述是基本指令的流水级，对于一些较复杂的指令，如定点乘除法指令、浮点指令以及访存指令，在执行阶段需要多拍。GS464 的基本结构如下图所示。

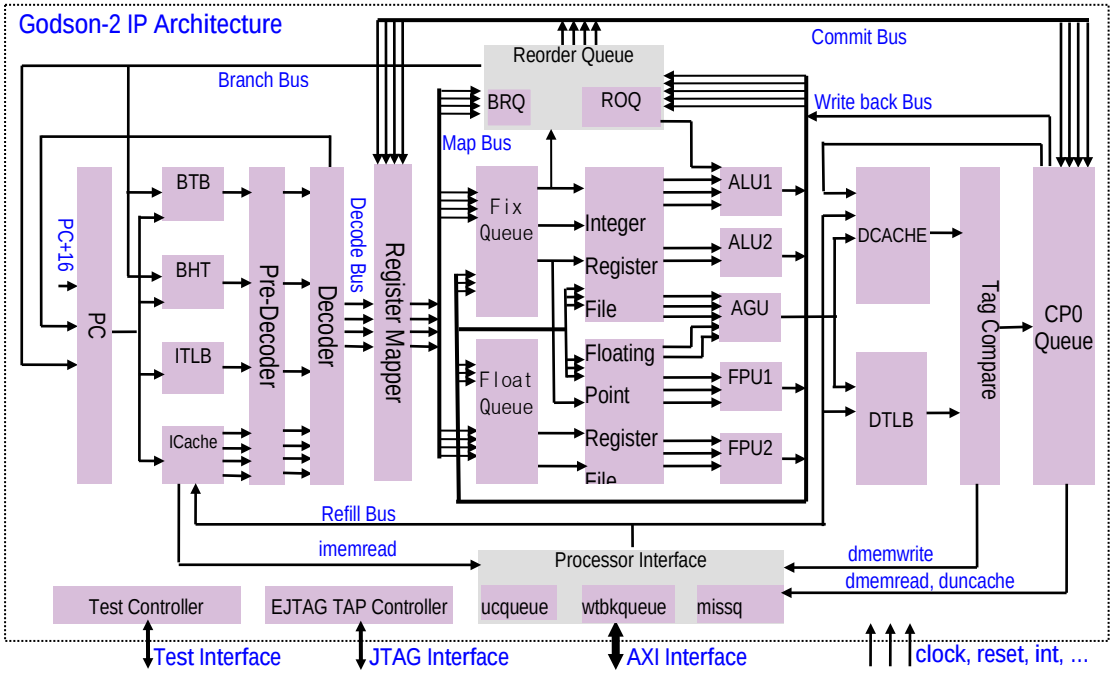


图 1-1　GS464 处理器的基本结构

2 龙芯 GS464 处理器核指令集概述

龙芯 GS464 处理器核兼容 MIPS64 R2 体系结构，其实现了 MIPS64 R2 规范所定义的全套必需指令，同时还有一部分自定义的扩展指令。

龙芯 GS464 处理器核所实现的 MIPS64 R2 兼容指令列举在 2.1 节中。囿于篇幅的原因，这部分指令的详细定义不在本文档中给出。读者确需了解相关指令的详细定义，建议查阅 MIPS 体系结构规范 2.50 版本的卷 I 和卷 II。这部分指令中涉及实现相关的内容将在 2.2 节中阐述，另有若干条 MIPS64 R2 指令 GS464 处理器核的实现与 MIPS 体系结构规范并不一致，也在 2.2 节中一并予以说明。

龙芯 GS464 处理器核所实现的自定义扩展指令列举在 2.3 节中。囿于篇幅的原因，这部分指令的详细定义不在本文档中给出。读者确需了解相关指令的详细定义，建议查阅《龙芯指令系统手册》。《龙芯指令系统手册》目前只提供给授权客户。

2.1 MIPS64 兼容指令列表

根据指令功能，GS464 处理器核所实现的 MIPS64 兼容指令可分为如下几组：

- **访存指令**
- **运算指令**
- **分支和跳转指令**
- **协处理器指令**
- **其它指令**

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 兼容级别
LB	取字节	MIPS32
LBU	取无符号字节	MIPS32

指令助记符	指令功能简述	ISA 兼容级别
LH	取半字	MIPS32
LHU	取无符号半字	MIPS32
LW	取字	MIPS32
LWU	取无符号字	MIPS32
LWL	取字左部	MIPS32
LWR	取字右部	MIPS32
LD	取双字	MIPS64
LDL	取双字左部	MIPS64
LDR	取双字右部	MIPS64
LL	取标志处地址	MIPS32
LLD	取标志处双字地址	MIPS64
SB	存字节	MIPS32
SH	存半字	MIPS32
SW	存字	MIPS32
SWL	存字左部	MIPS32
SWR	存字右部	MIPS32
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 CPU 指令集：算术指令（ALU 立即数）

指令助记符	指令功能简述	ISA 兼容级别
ADDI	加立即数	MIPS32
DADDI	加双字立即数	MIPS64

指令助记符	指令功能简述	ISA 兼容级别
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 CPU 指令集：算术指令（3 操作数）

指令助记符	指令功能简述	ISA 兼容级别
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 CPU 指令集：算术指令（2 操作数）

指令助记符	指令功能简述	ISA 兼容级别
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	半字字节交换	MIPS32 R2
DSHD	字半字交换	MIPS64 R2
SEB	字节符号扩展	MIPS32 R2
SEH	半字符号扩展	MIPS32 R2
INS	位插入	MIPS32 R2
EXT	位提取	MIPS32 R2
DINS	双字位插入	MIPS64 R2
DINSM	双字位插入	MIPS64 R2
DINSU	双字位插入	MIPS64 R2
DEXT	双字位提取	MIPS64 R2
DEXTM	双字位提取	MIPS64 R2
DEXTU	双字位提取	MIPS64 R2

表 2-5 CPU 指令集：乘法和除法指令

指令助记符	指令功能简述	ISA 兼容级别
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 CPU 指令集：移位指令

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

2.1.3 分支和跳转指令

分支和跳转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 定义中，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 CPU 指令集：跳转和分支指令

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

协处理器指令完成协处理器内部的操作。龙芯 GS464 处理器核有两个协处理器：0 号协处理器（系统处理器）和 1 号协处理器（浮点协处理器）。

0 号协处理器（CP0）通过 CP0 的寄存器来管理内存和处理异常。这些指令列在表 2-8 中。

MIPS 体系结构规范对 1 号协处理器（CP1）指令中的浮点指令进行了明确定义。龙芯

GS464 处理器核中关于协处理器 1 指令中浮点指令的具体实现将在第 7 章单独介绍。

表 2-8 CPU 指令集：CP0 指令

指令助记符	指令功能简述	ISA 兼容级别
DMFC0	从 CP0 寄存器取双字	MIPS64
DMTC0	往 CP0 寄存器写双字	MIPS64
MFC0	从 CP0 寄存器取	MIPS32
MTC0	往 CP0 寄存器写	MIPS32
TLBR	读索引的 TLB 项	MIPS32
TLBWI	写索引的 TLB 项	MIPS32
TLBWR	写随机的 TLB 项	MIPS32
TLBP	在 TLB 中搜索匹配项	MIPS32
CACHE	Cache 操作	MIPS32
ERET	异常返回	MIPS32
DI	禁止中断	MIPS32 R2
EI	允许中断	MIPS32 R2

2.1.5 其它指令

MIPS64 中，除了前面列出上述指令外还有其它一些指令，详见表 2-9 至表 2-12：

表 2-9 CPU 指令集：特殊指令

指令助记符	指令功能简述	ISA 兼容级别
SYSCALL	系统调用	MIPS32
BREAK	断点	MIPS32
SYNC	同步	MIPS32
SYNCI	同步指令缓存	MIPS32 R2

表 2-10 CPU 指令集：异常指令

指令助记符	指令功能简述	ISA 兼容级别
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

指令助记符	指令功能简述	ISA 兼容级别
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 CPU 指令集：条件移动指令

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

表 2-12 CPU 指令集：其它指令

指令助记符	指令功能简述	ISA 兼容级别
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 兼容指令实现相关说明

2.2.1 存在实现差异的指令

龙芯 GS464 处理器核对所有 MIPS64 R2 指令都作了支持，但对一些与实现相关的指令作了重定义，见表 2-13。

表 2-13 存在实现差异的指令

指令助记符	指令功能简述	具体实现描述
PREF	预取指令	视作 NOP 指令处理，无预取效果。 软件预取可通过 Load 到 0 号寄存器实现。
PREFX	预取指令	视作 NOP 指令处理，无预取效果。 软件预取可通过 Load 到 0 号寄存器实现。
SSNOP	单发射空操作	视作 NOP 指令处理。 GS464 中所有数据相关和控制相关（CP0 Hazards）由硬件维护，无需软件控制。
EHB	隔离执行相关	视作 NOP 指令处理。 GS464 中所有数据相关和控制相关（CP0 Hazards）由硬件维护，无需软件控制。
WAIT	进入等待状态	视作 NOP 指令处理，无停止流水线效果。 避免在涉及低功耗管理的代码处使用该指令，不仅无法实现软件设计预期，甚至可能引起功耗增加。
RDPGPR	读影子寄存器	GS464 没有实现影子寄存器，所以源寄存器与目标寄存器都属于当前寄存器组。
WRPGPR	写影子寄存器	GS464 没有实现影子寄存器，所以源寄存器与目标寄存器都属于当前寄存器组。
SYNC	同步	GS464 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。SYNC 指令需要 CU[0]，只有内核态可以使用。

<table>
<tr><th>指令助记符</th><th>指令功能简述</th><th>具体实现描述</th></tr>
<tr><td>CACHE</td><td>cache 操作</td><td>
GS464 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
GS464 所实现的 index 类 CACHE 指令采用虚地址的最低 2 位作为第几路 Cache 的选择信号，而不是像 MIPS64 规范中定义的从虚地址的中间截取。
2、CACHE28、CACHE29、CACHE30、CACHE31 指令的含义
处理器中，CACHE28、CACHE29、CACHE30 和 CACHE31 指令（即 op[4:2]=0b111 的 Cache 指令）的含义与 MIPS64 规范不同。龙芯处理器中这些指令均为 Index Store Data 操作，而 MIPS64 规范中均为 Fetch and Lock 操作。

GS464 处理器核中有效的 Cache 操作类型列举如下：
<table>
<tr><th>op[4:0]</th><th>功能描述</th></tr>
<tr><td>b00000</td><td>根据索引无效 I-Cache 行</td></tr>
<tr><td>b01000</td><td>根据索引写 I-Cache 行 Tag</td></tr>
<tr><td>b11100</td><td>根据索引写 I-Cache 行 Data</td></tr>
<tr><td>b00001</td><td>根据索引无效并写回 D-Cache 行</td></tr>
<tr><td>b00101</td><td>根据索引读 D-Cache 行 Tag</td></tr>
<tr><td>b01001</td><td>根据索引写 D-Cache 行 Tag</td></tr>
<tr><td>b10001</td><td>根据命中无效 D-Cache 行</td></tr>
<tr><td>b10101</td><td>根据命中无效并写回 D-Cache 行</td></tr>
<tr><td>b11001</td><td>根据索引读 D-Cache 行 Data</td></tr>
<tr><td>b11101</td><td>根据索引写 D-Cache 行 Data</td></tr>
<tr><td>b00011</td><td>根据索引无效并写回 S-Cache 行</td></tr>
<tr><td>b00111</td><td>根据索引读 S-Cache 行 Tag</td></tr>
<tr><td>b01011</td><td>根据索引写 S-Cache 行 Tag</td></tr>
<tr><td>b10011</td><td>根据命中无效并写回 S-Cache 行</td></tr>
<tr><td>b11011</td><td>根据索引读 S-Cache 行 Data</td></tr>
<tr><td>b11111</td><td>根据索引写 S-Cache 行 Data</td></tr>
</table>
</td></tr>
</table>

2.2.2 禁用指令

GS464 处理器核禁用下列指令，见表 2-14：

表 2-14 禁用指令

指令助记符	指令功能简述	ISA 兼容级别
DI	禁止中断	MIPS32 R2
EI	允许中断	MIPS32 R2

2.3 自定义扩展指令

龙芯 GS464 处理器实现的自定义扩展指令按功能划分包括如下几类：

- **访存指令**
- **乘除运算指令**
- **X86 二进制加速指令**
- **64 位多媒体指令**
- **杂项指令**

2.3.1 自定义扩展访存指令

表 2-15 自定义扩展访存指令

指令助记符	指令功能简述
GSLE	如果小于等于置地址错例外
GSGT	如果大于置地址错例外
GSLBLE	带上越界检查的取字节
GSLBGT	带下越界检查的取字节
GSLHLE	带上越界检查的取半字
GSLHGT	带下越界检查的取半字
GSLWLE	带上越界检查的取字
GSLWGT	带下越界检查的取字
GSLDLE	带上越界检查的取双字
GSLDGT	带下越界检查的取双字
GSLQ	双目标寄存器取定点四字
GSLBX	带偏移的取字节
GSLHX	带偏移的取半字

指令助记符	指令功能简述
GSLWX	带偏移的取字
GSLDX	带偏移的取双字
GSSBLE	带上越界检查的存字节
GSSBGT	带下越界检查的存字节
GSSHLE	带上越界检查的存半字
GSSHGT	带下越界检查的存半字
GSSWLE	带上越界检查的存字
GSSWGT	带下越界检查的存字
GSSDLE	带上越界检查的存双字
GSSDGT	带下越界检查的存双字
GSSQ	双源寄存器存定点四字
GSSBX	带偏移的存字节
GSSHX	带偏移的存半字
GSSWX	带偏移的存字
GSSDX	带偏移的存双字

2.3.2 自定义扩展乘除运算指令

表 2-16 自定义扩展乘除运算指令

指令助记符	指令功能简述
GSMULT	有符号字乘，结果写通用寄存器
GSDMULT	有符号双字乘，结果写通用寄存器
GSMULTU	无符号字乘，结果写通用寄存器
GSDMULTU	无符号双字乘，结果写通用寄存器
GSDIV	有符号字除，商写通用寄存器
GSDDIV	有符号双字除，商写通用寄存器
GSDIVU	无符号字除，商写通用寄存器
GSDDIVU	无符号双字除，商写通用寄存器
GSMOD	有符号字除，余数写通用寄存器
GSDMOD	有符号双字除，余数写通用寄存器
GSMODU	无符号字除，余数写通用寄存器

指令助记符	指令功能简述
GSDMODU	无符号双字除，余数写通用寄存器

2.3.3 自定义扩展 X86 二进制翻译加速指令

表 2-17 自定义扩展 X86 二进制翻译加速指令

指令助记符	指令功能简述
X86AND	以 x86 方式只设置 EFLAG 的逻辑位与
X86OR	以 x86 方式只设置 EFLAG 的逻辑位或
X86XOR	以 x86 方式只设置 EFLAG 的逻辑位异或
X86DADD	以 x86 方式只设置 EFLAG 的双字加
X86ADD	以 x86 方式只设置 EFLAG 的字加
X86DADDU	以 x86 方式只设置 EFLAG 的无例外双字加
X86ADDU	以 x86 方式只设置 EFLAG 的无例外字加
X86DSUB	以 x86 方式只设置 EFLAG 的双字减
X86SUB	以 x86 方式只设置 EFLAG 的字减
X86DSUBU	以 x86 方式只设置 EFLAG 的无例外双字减
X86SUBU	以 x86 方式只设置 EFLAG 的无例外字减
X86INC	以 x86 方式只设置 EFLAG 的双字自增 1
X86DEC	以 x86 方式只设置 EFLAG 的双字自减 1
X86DSLL	以 x86 方式只设置 EFLAG 的双字左移
X86SLL	以 x86 方式只设置 EFLAG 的字左移
X86DSLL32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑左移
X86DSLLV	以 x86 方式只设置 EFLAG 的双字可变移位量左移
X86SLLV	以 x86 方式只设置 EFLAG 的字可变移位量左移
X86DSRL	以 x86 方式只设置 EFLAG 的双字逻辑右移
X86SRL	以 x86 方式只设置 EFLAG 的字逻辑右移
X86DSRL32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑逻辑右移
X86DSRLV	以 x86 方式只设置 EFLAG 的双字可变移位量逻辑右移
X86SRLV	以 x86 方式只设置 EFLAG 的字可变移位量逻辑右移
X86DSRA	以 x86 方式只设置 EFLAG 的双字算术右移
X86SRA	以 x86 方式只设置 EFLAG 的字算术右移

指令助记符	指令功能简述
X86DSRA32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑算术右移
X86DSRAV	以 x86 方式只设置 EFLAG 的双字可变移位量算术右移
X86SRAV	以 x86 方式只设置 EFLAG 的字可变移位量算术右移
X86DROTR	以 x86 方式只设置 EFLAG 的双字循环右移
X86ROTR	以 x86 方式只设置 EFLAG 的字循环右移
X86DROTR32	以 x86 方式只设置 EFLAG 的移位量加 32 的双字逻辑循环右移
X86DROTRV	以 x86 方式只设置 EFLAG 的可变移位量的双字循环右移
X86ROTRV	以 x86 方式只设置 EFLAG 的可变移位量的字循环右移
X86MFFLAG	以 x86 方式提取 EFLAG 标志位的值
X86MTFLAG	以 x86 方式修改 EFLAG 标志位的值
X86J	以 x86 方式根据 EFLAG 值跳转
X86LOOP	以 x86 方式根据 EFLAG 值循环
SETTM	x86 浮点栈模式置位
CLRTM	x86 浮点栈模式清除
INCTOP	x86 浮点栈顶指针加 1
DECTOP	x86 浮点栈顶指针减 1
MTTOP	写 x86 浮点栈顶指针
MFTOP	读 x86 浮点栈顶指针
SETTAG	判断并置位寄存器

2.3.4 自定义扩展 64 位多媒体加速指令

表 2-18 自定义扩展 64 位多媒体加速指令

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

指令助记符	指令功能简述
PADDD	64 位数加
PSUBSH	四个 16 位有符号整数减，有符号饱和
PSUBUSH	四个 16 位无符号整数减，无符号饱和
PSUBH	四个 16 位数减
PSUBW	两个 32 位数减
PSUBSB	八个 8 位有符号整数减，有符号饱和
PSUBUSB	八个 8 位无符号整数减，无符号饱和
PSUBB	八个 8 位数减
PSUBD	64 位数减
PSHUFH	混洗四个 16 位数
PACKSSWH	32 位有符号整数转化成 16 位，有符号饱和
PACKSSHB	16 位有符号整数转化成 8 位，有符号饱和
PACKUSHB	16 位有符号整数转化成 8 位，无符号饱和
PANDN	fs 取非后与 ft 按位与
PUNPCKLHW	拆包低 16 位数
PUNPCKHHW	拆包高 16 位数
PUNPCKLBH	拆包低 8 位数
PUNPCKHBH	拆包高 8 位数
PINSRH_0	ft 低 16 位数插入到 fs 低 0 个 16 位
PINSRH_1	ft 低 16 位数插入到 fs 低 1 个 16 位
PINSRH_2	ft 低 16 位数插入到 fs 低 2 个 16 位
PINSRH_3	ft 低 16 位数插入到 fs 低 3 个 16 位
PAVGH	四个 16 位无符号整数取平均值
PAVGB	八个 8 位无符号整数取平均值
PMAXSH	四个 16 位有符号整数取较大值
PMINSH	四个 16 位有符号整数取较小值
PMAXUB	八个 8 位无符号整数取较大值
PMINUB	八个 8 位无符号整数取较小值
PCMPEQW	两个 32 位数相等比较
PCMPGTW	两个 32 位有符号整数大于比较
PCMPEQH	四个 16 位数相等比较
PCMPGTH	四个 16 位有符号整数大于比较
PCMPEQB	八个 8 位数相等比较

指令助记符	指令功能简述
PCMPGTB	八个 8 位有符号整数大于比较
PSLLW	两个 32 位数逻辑左移
PSLLH	四个 16 位数逻辑左移
PMULLH	四个 16 位有符号整数相乘，取结果低 16 位
PMULHH	四个 16 位有符号整数相乘，取结果高 16 位
PMULUW	低 32 位无符号整数相乘，存 64 位结果
PMULHUH	四个 16 位无符号整数相乘，取结果高 16 位
PSRLW	两个 32 位数逻辑右移
PSRLH	四个 16 位数逻辑右移
PSRAW	两个 32 位数算术右移
PSRAH	四个 16 位数算术右移
PUNPCKLWD	低 32 位数组合成 64 位数
PUNPCKHWD	高 32 位数组合成 64 位数
PASUBUB	八个 8 位无符号整数相减并取绝对值
PEXTRH	fs 某 16 位拷贝到 fd 低 16 位，fd 高位补 0
PMADDHW	四个 16 位有符号数相乘，低高位分别累加
BIADD	多字节累加
PMOVMSKB	条件字节移动
GSXOR	fs 与 ft 逻辑位异或
GSNOR	fs 与 ft 逻辑位或非
GSAND	fs 与 ft 逻辑位与
GSADDU	fs 与 ft 定点无符号字加
GSOR	fs 与 ft 定点逻辑位或
GSADD	fs 与 ft 定点字加
GSDADD	fs 与 ft 定点双字加
GSSEQU	fs 与 ft 定点数相等比较
GSSEQ	fs 与 ft 定点数相等比较
GSSUBU	fs 与 ft 定点无符号字减
GSSUB	fs 与 ft 定点字减
GSDSUB	fs 与 ft 定点双字减
GSSLTU	fs 与 ft 定点无符号定点数小于比较
GSSLT	fs 与 ft 定点定点数小于比较
GSSLL	fs 与 ft 定点逻辑左移字

指令助记符	指令功能简述
GSDSLL	fs 与 ft 定点逻辑左移双字
GSSRL	fs 与 ft 定点逻辑右移字
GSDSRL	fs 与 ft 定点逻辑右移双字
GSSRA	fs 与 ft 定点算术右移字
GSDSRA	fs 与 ft 定点算术右移双字
GSSLEU	fs 与 ft 定点无符号定点数小于等于比较
GSSLE	fs 与 ft 定点定点数小于等于比较

2.3.5 自定义扩展杂项指令

表 2-19 自定义扩展杂项指令

指令助记符	指令功能简述
CAMPV	查询查找表，返回命中项的内容
CAMPI	查询查找表，返回命中项的索引
CAMWI	写查找表指定项
RAMRI	读查找表指定项的内容

3 CP0 控制寄存器

本章描述协处理器 0（Coprocessor 0，简称 CP0）的操作，主要内容包括 CP0 的寄存器定义以及龙芯 3 号处理器实现的 CP0 指令。CP0 寄存器用于控制处理器的状态改变并报告处理器的当前状态。这些寄存器通过 MFC0/DMFC0 指令来读或者通过 MTC0/ DMTC0 指令来写。CP0 寄存器如表 3-1 所示。

当处理器运行在核心模式时或状态寄存器（Status 寄存器）中的第 28 位（CU0）被设置时，可以使用 CP0 指令。否则，执行 CP0 指令将产生“协处理器不可用例外”。

表 3-1 CP0 寄存器

寄存器号		寄存器名字	描述
总号	子号		
0	0	Index	可写的寄存器，用于指定需要读/写的 TLB 表项
1	0	Random	用于 TLB 替换的伪随机计数器
2	0	EntryLo0	TLB 表项低半部分中对应于偶虚页的内容（主要是物理页号）
3	0	EntryLo1	TLB 表项低半部分中对应于奇虚页的内容（主要是物理页号）
4	0	Context	32 位寻址模式下指向内核的虚拟页转换表（PTE）
5	0	Page Mask	设置 TLB 页大小的掩码值
5	1	Page Grain	标志是否支持大页地址
6	0	Wired	固定连线的 TLB 表项数目（指不用于随机替换的低端 TLB 表项）
7	0	Hwrena	硬件寄存器使能
8	0	BadVaddr	错误的虚地址
9	0	Count	计数器
10	0	EntryHi	TLB 表项的高半部分内容（虚页号和 ASID）
11	0	Compare	计数器比较
12	0	Status	处理器状态寄存器
12	1	IntCtl	扩充中断控制寄存器
12	2	SRSCtl	影子寄存器组控制寄存器
13	0	Cause	最近一次例外的原因
14	0	EPC	例外程序计数器
15	0	PRid	处理器修订版本标识号
15	1	EBase	例外向量基地址

16	1	Config1	配置寄存器 1
16	2	Config2	配置寄存器 2
16	3	Config3	配置寄存器 3
17	0	LLAddr	链接读内存地址
18			保留
19			保留
20	0	Xcontext	64 位寻址模式下指向内核的虚拟页转换表（PTE）
21			保留
22	0	Diagnose	使能/禁用 BTB,RAS 以及清空 ITLB 表
23	0	Debug	EJTAG 调试寄存器
24	0	DEPC	EJTAG 调试例外程序计数器
25	0/1/2/3	PerfCnt	性能计数器
26	0	ErrCtl	Parity/ECC 校验控制及状态
27	0	CacheErr	Cache ECC 校验控制及状态
28	0	TagLo	CACHE TAG 寄存器的低半部分
28	1	DataLo	用于和 cache 数据队列交互和诊断
29	0	TagHi	CACHE TAG 寄存器的高半部分
29	1	DataHi	用于和 cache 数据队列交互和诊断
30	0	ErrorEPC	错误例外程序计数器
31	0	DESAVE	暂存器，用于 Debug 异常处理

3.1 Index 寄存器(0，0)

Index 寄存器是个 32 位可读/写的寄存器，其中最后六位的值用于索引 TLB 的表项。寄存器的最高位表示 TLB 探测(TLBP)指令执行是否成功。

Index 寄存器的值指示 TLB 读(TLBR)和 TLB 索引写(TLBWI)指令操作的 TLB 表项。

图 3-1 表示 Index 寄存器的格式，表 3-2 描述了 Index 寄存器各域的含义。

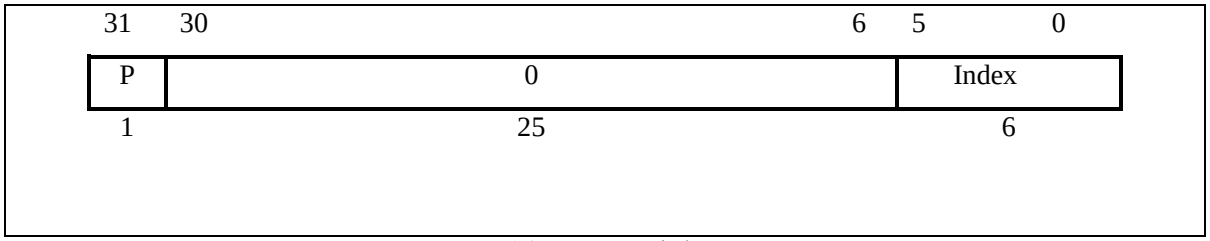


图 3-1 Index 寄存器

表 3-2 Index 寄存器各域描述

域	描述

P	探测失败。上一次 TLB 探测指令（TLBP）没有成功时置 1
Index	指示 TLB 读指令和 TLB 索引写指令操作的 TLB 表项的索引值
0	保留。必须按 0 写入，读时返回 0。

3.2 Random 寄存器(1，0)

Random 寄存器是个只读寄存器，其中低六位索引 TLB 的表项。每执行完一条指令，该寄存器值减 1。同时，寄存器值在一个上界和一个下界之间浮动，上下界具体是：

- 下界等于保留给操作系统专用的 TLB 项数（即 Wired 寄存器的内容）。
- 上界等于整个 TLB 的项数减 1（最大为 64-1）。

Random 寄存器指示将由 TLB 随机写指令操作的 TLB 项。从这个目的来说，无需读此寄存器。但该寄存器是可读的，以验证处理器相应的操作是否正确。

为了简化测试，Random 寄存器在系统重起时置为上界。另外，当 Wired 寄存器被写时，该寄存器也要置为上界。

图 3-2 表示 Random 寄存器的格式，而表 3-3 描述 Random 寄存器各域的含义。

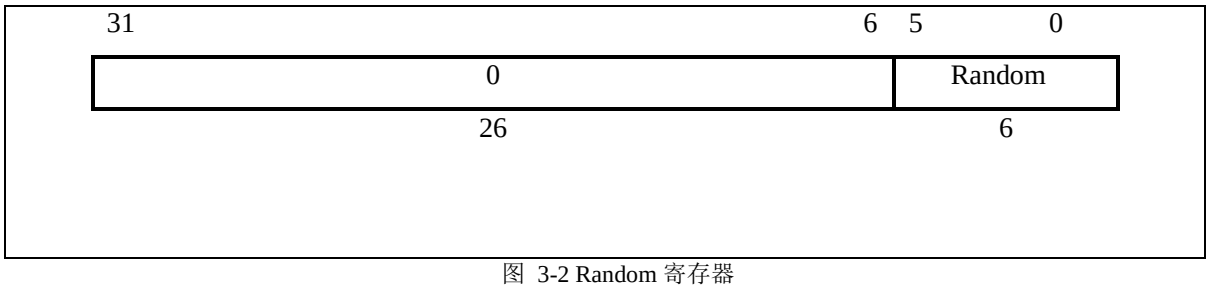


图 3-2 Random 寄存器

表 3-3 Random 寄存器各域

域	描述
Random	随机 TLB 索引值
0	保留。必须按 0 写入，读时返回 0。

3.3 EntryLo0 (2，0)以及 EntryLo1 (3，0)寄存器

EntryLo 寄存器包括两个相同格式的寄存器：

- EntryLo0 用于偶虚页
- EntryLo1 用于奇虚页

EntryLo0 和 EntryLo1 寄存器都是可读/写寄存器。当执行 TLB 读和写操作时，它们分别包括 TLB 项中奇偶页的物理页号（PFN）。图 3-3 表示这些寄存器的格式。

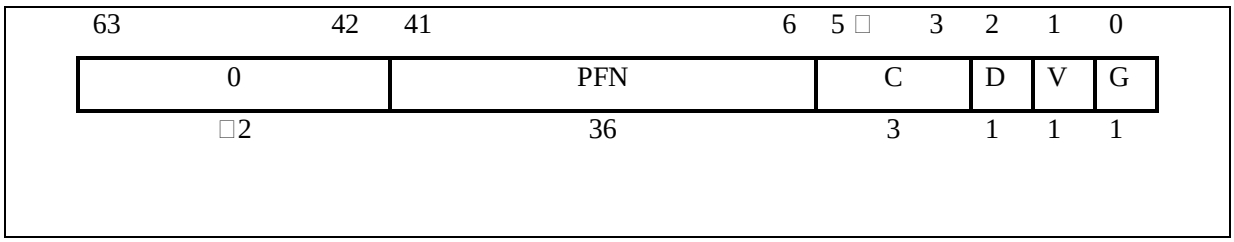


图 3-3 EntryLo0 和 EntryLo1 寄存器

EntryLo0 和 EntryLo1 寄存器的 PFN 域是 48 位物理地址中的高 36 位（47：12）。

表 3-4 EntryLo 寄存器域

域	描述
PFN	页号，是物理地址的高位。
C	TLB 页的 Cache 一致性属性。
D	脏位。如果该位被设置，页面则标记为脏，也就是可写的。实际上这一位在软件中作为防止数据被更改的写保护使用。
V	有效位。当该位被设置时，说明 TLB 表项是有效的，否则将产生一个 TLBL 或 TLBS 例外。
G	全局位。当 EntryLo0 和 EntryLo1 中的 G 位都被设置为 1 时，处理器将在 TLB 查找时忽略 ASID。
0	保留。必须按 0 写入，读时返回 0。

在每个 TLB 表项中只有一个全局位，在 TLB 写操作中根据 EntryLo0[0]和 EntryLo1[0]的值写入。

3.4 Context （4，0）

Context 寄存器是一个读/写寄存器，它包含指向页表中某一项的指针。该页表是一个操作系统数据结构，存储虚拟地址到物理地址的转换。

当 TLB 例外发生时，CPU 将根据失效的转换从页表中加载 TLB。一般情况下，操作系统使用 Context 寄存器寻址页表中当前页的映射。Context 寄存器复制 BadVAddr 寄存器中的部分信息，但是该信息被安排成一种利于软件 TLB 例外处理程序处理的形式。

图 3-4 显示了 Context 寄存器的格式；表 3-5 描述了上下文寄存器字段。

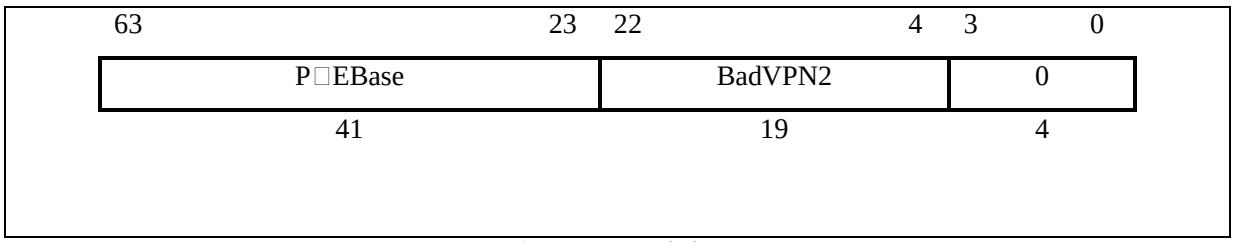


图 3-4 Context 寄存器

表 3-5 Context 寄存器域

域	描述
BadVPN2	当 TLB 例外时这一字段被硬件写。它包含最近不能进行有效转换的虚地址的虚页号

	(VPN)。
PTEBase	这一字段是操作系统使用的读/写字段。该字段写入的值允许操作系统将 Context 寄存器作为一个指向内存中当前页表的指针。
0	保留。必须按 0 写入，读时返回 0。

19 位的 BadVPN2 字段包含导致 TLB 例外的虚地址的 31:13 位；第 12 位被排除是因为一个单一的 TLB 项映射到一个奇偶页对。对于一个 4K 字节的页尺寸，这一格式可以直接寻址 PTE 表项为 8 字节长且按对组织的页表。对于其它尺寸的页和 PTE，移动和屏蔽这个值可以产生合适的地址。

3.5 PageMask 寄存器(5，0)

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，如表 3-6。该寄存器的格式如图 3-5。

TLB 读写操作使用该寄存器作为一个源或目的；当进行虚实地址转换时，TLB 中对应于 PageMask 寄存器的相应位指示虚地址位 24:13 中哪些位用于比较。当 MASK 域的值不是表 3-6 中的值时，TLB 的操作为未定义。0 域为保留，必须按 0 写入，读时返回 0。

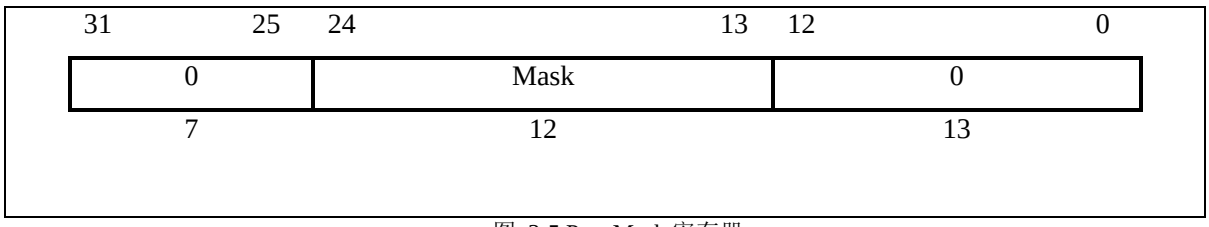


图 3-5 PageMask 寄存器

表 3-6 不同页大小的掩码（Mask）值

页大小	位											
	24	23	22	21	20	19	18	17	16	15	14	13
4Kbytes	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	1	1	1	1
256 Kbytes	0	0	0	0	0	0	1	1	1	1	1	1
1 Mbytes	0	0	0	0	1	1	1	1	1	1	1	1
4 Mbytes	0	0	1	1	1	1	1	1	1	1	1	1
16M bytes	1	1	1	1	1	1	1	1	1	1	1	1

3.6 PageGrain 寄存器(5，1)

PageGrain 寄存器是个可读写的寄存器，龙芯 3 号只定义了这个寄存器的第 29 位：ELPA（Enable Large Physical Adress），其余的位保留为 0。

当 ELPA=1 时，龙芯 3 号支持 48 位物理地址；当 ELPA=0 时，龙芯 3 号只支持 40 位物理地址。

能否写 ELPA 位依赖于 Config3 寄存器的 LPA 域。当 Config3 的 LPA 位为 0 时，PageGrain 的 ELPA 位被置 0。该寄存器的格式如图 3-6，寄存器的域见表 3-7。

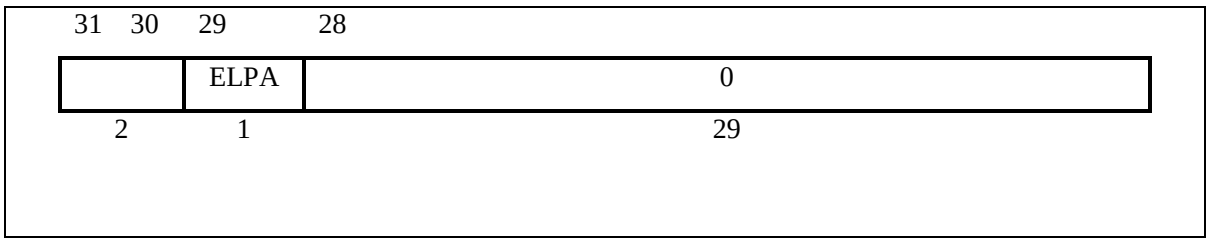


图 3-6 PageGrain 寄存器

表 3-7 PageGrain 寄存器域

域	描述
ELPA	这个域是否置位，表示是否支持大物理地址
0	保留。必须按 0 写入，读时返回 0。

3.7 Wired 寄存器(6，0)

Wired 寄存器是一个可读/写的寄存器，该寄存器的值指定了 TLB 中固定表项与随机表项之间的界限，如图 3-7 所示。Wired 表项是固定的、不可替换的表项，这些表项的内容不会被 TLB 写操作修改。而随机表项的内容可以被修改。

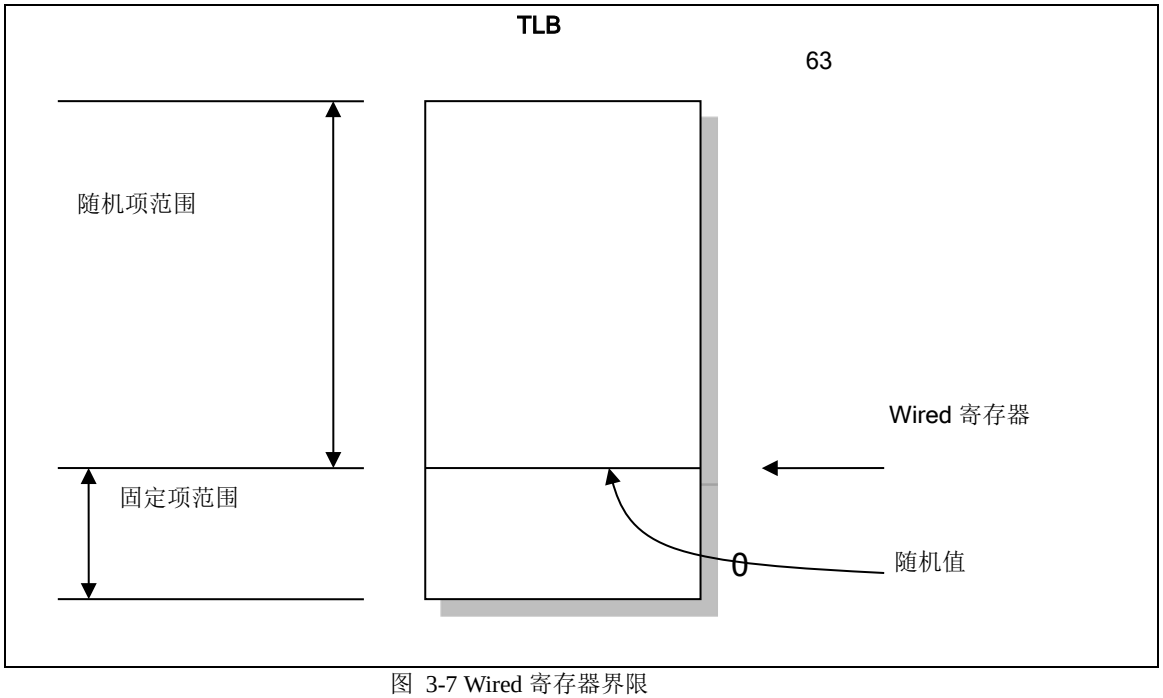


图 3-7 Wired 寄存器界限

Wired 寄存器在系统复位时置 0。写该寄存器的同时，Random 寄存器值要置为上限值（参阅前面的 Random 寄存器）。

图 3-8 表示 Wired 寄存器的格式；表 3-8 描述了该寄存器的域。

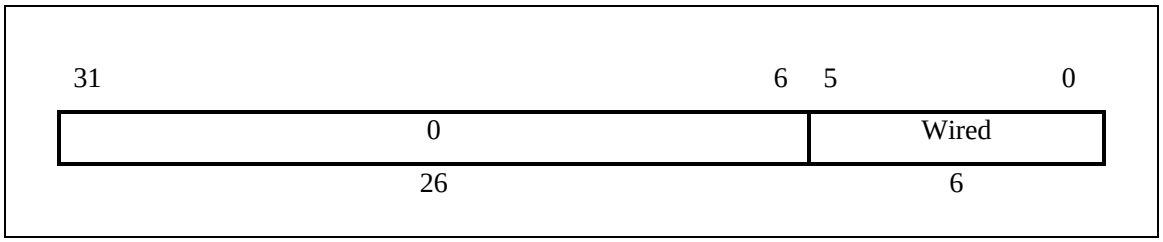


图 3-8 Wired 寄存器

表 3-8 Wired 寄存器域

域	描述
Wired	TLB 固定表项边界
0	保留。必须按 0 写入，读时返回 0。

3.8 HWREna 寄存器（7，0）

HWREna 寄存器是一个可读写寄存器，龙芯 3 号只定义了这个寄存器的 Mask 域，用于表示指令 RDHWR 的源硬件寄存器。0 域为保留，必须按 0 写入，读时返回 0。

图 3-9 显示了 HWREna 寄存器的格式；表 3-9 描述的 Mask 域和硬件寄存器的对应关系。

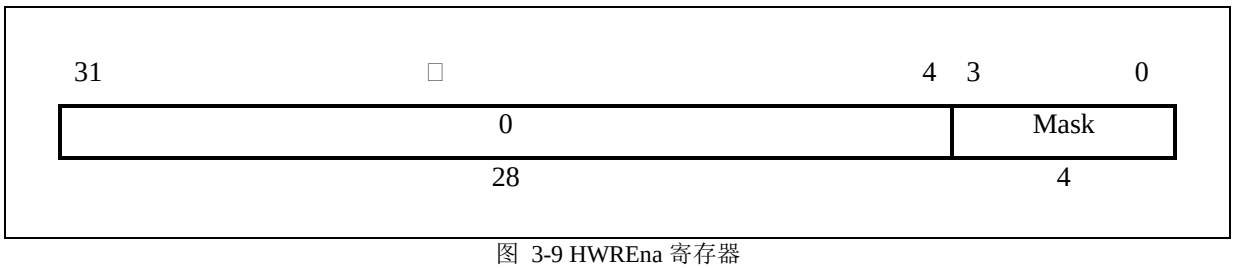


图 3-9 HWREna 寄存器

表 3-9 Mask 域和硬件寄存器的对应关系

硬件寄存器	位			
	3	2	1	0
CPUnum	0	0	0	1
SYNCI_Step	0	0	1	0
CC	0	1	0	0
CCRes	1	0	0	0

3.9 BadVAddr 寄存器(8，0)

错误虚地址寄存器（BadVAddr）是一个只读寄存器，它记录了最近一次导致 TLB 或寻址错误例外的虚拟地址。除非发生软件复位，NMI 或 Cache 错误例外，BadVAddr 寄存器将一直保持不变。否则这个寄存器就是未定义的。

图 3-10 显示了错误虚地址寄存器的格式。

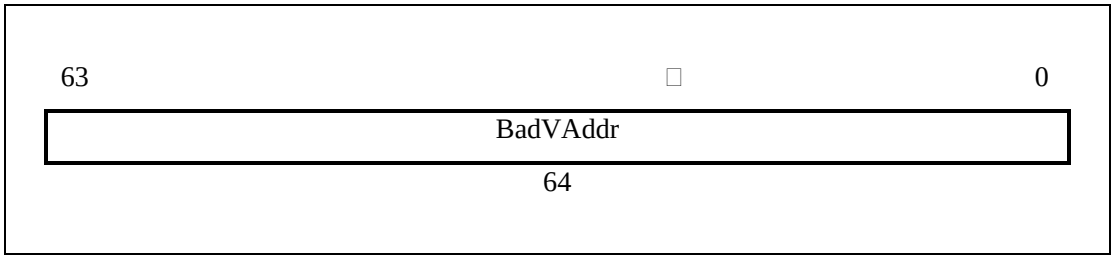


图 3-10 BadVAddr 寄存器

3.10 Count 寄存器(9，0)以及 Compare 寄存器(11，0)

Count 寄存器和 Compare 寄存器都是 32 位读写寄存器。

Count 寄存器作为一个实时的定时器工作，每两个时钟周期增 1。

Compare 寄存器用来在特定的时刻生成一个中断，该寄存器被写入一个值，并且不断地与 Count 寄存器中的值比较。一旦这两个值相等，便生成一个中断请求。Cause 寄存器里的 TI 和 IP[7]被设置。当 Compare 寄存器被再次写时 Cause 寄存器的 TI 位才被清零。

图 3-11 显示了 Count 寄存器的格式。图 3-12 显示了 Compare 寄存器的格式。

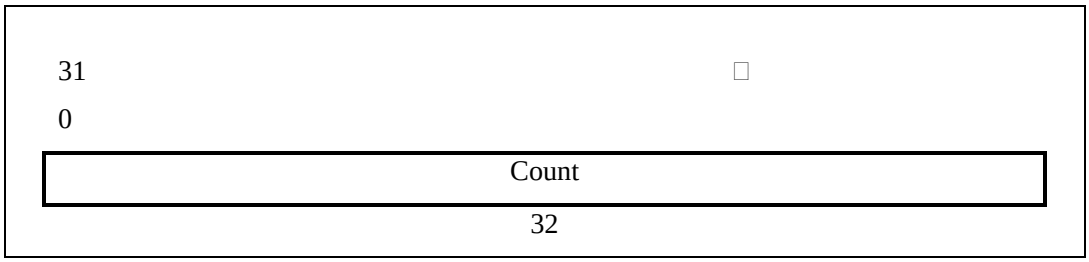


图 3-11 Count 寄存器

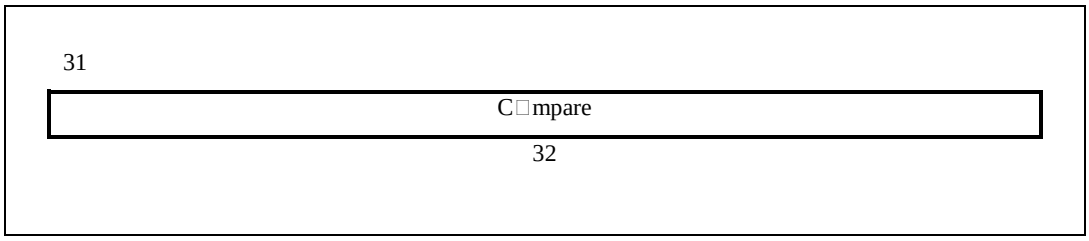


图 3-12 Compare 寄存器

3.11 EntryHi 寄存器(10，0)

EntryHi 寄存器用于 TLB 读写时存放 TLB 表项的高位。

EntryHi 寄存器可以被 TLB Probe, TLB Write Random, TLB Write Indexed, 和 TLB Read Indexed 指令访问。

图 3-13 表示 EntryHi 寄存器的格式。

表 3-10 表示 EntryHi 寄存器的域。

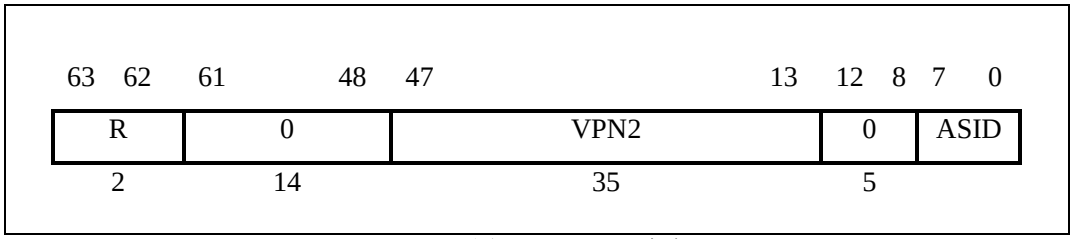


图 3-13 EntryHi 寄存器

表 3-10 EntryHi 寄存器域

域	描述
VPN2	虚页号除 2（映射到双页）；虚拟地址的高位。
ASID	地址空间标识域。一个 8 位的域；用于让多个进程共享 TLB；对于相同的虚页号，每个进程都与其他进程有不同的映射。
R	区域位。（00->用户，01->超级用户，11->核心）用于匹配 vAddr63...62
0	保留。必须按 0 写入，读时返回 0。

VPN2 域包含 64 位虚拟地址的 61:13 位。

当一个 TLB Refill，TLB Invalid，或 TLB Modified 例外发生时，没有匹配 TLB 表项的虚拟地址中虚拟页号（VPN2）和 ASID 将被加载到 EntryHi 寄存器。

3.12 Status 寄存器(12，0)

Status 寄存器(SR) 是一个读写寄存器，它包括操作模式，中断允许和处理器状态诊断。下面列表描述了一些更重要的 Status 寄存器字段；图 3-14 显示了整个寄存器的格式，包括域的描述。其中重要的域有：

- 8 位的中断屏蔽(IM)域控制 8 个中断条件的使能。中断在被触发之前必须被使能，在 Status 寄存器的中断屏蔽域和 Cause 寄存器的中断待定域相应的位都应该被置位。更多的信息，请参考 Cause 寄存器的中断待定（IP）域。
- 4 位的协处理器可用性（CU）域控制 4 个可能的协处理器的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。

Status 寄存器格式

图 3-14 显示了 Status 寄存器的格式，表 3-11 描述了 Status 寄存器的域。

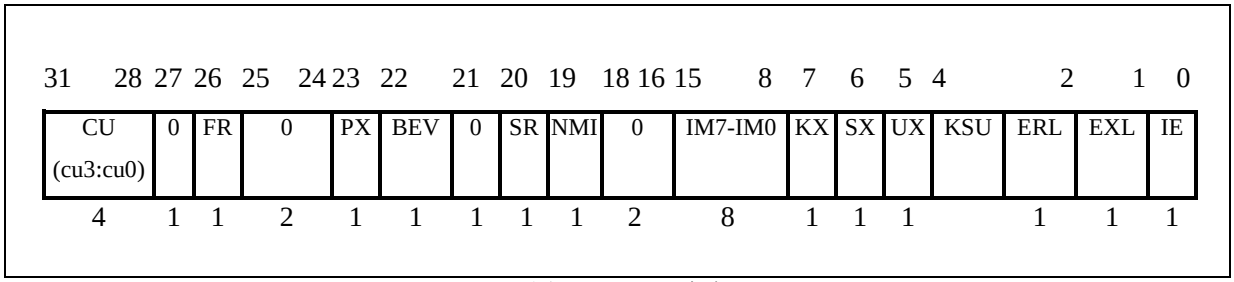


图 3-14 Status 寄存器

表 3-11 Status 寄存器域

域	描述
CU	控制 4 个协处理器单元的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。 1- 可用 0- 不可用 CU 域的初值是 0011
0	保留。必须按 0 写入，读时返回 0。

FR	使能附加的浮点寄存器 s 0 -□16 个寄存器 1 -□32 个寄存器
PX	使能用户模式下的 64 位操作（其余模式下的 64 位操作无需使能） 1 – 使能 0 – 未使能（此时用户模式下 64 位操作是否可用需要判断 UX 位）
BEV	控制例外向量的入口地址 0 – 正常 1 – 启动
SR	1 表示有软复位例外发生
NMI	是否发生 NMI 例外。注意，软件不能把这位由 0 写为 1
IM	中断屏蔽：控制每一个外部、内部和软件中断的使能。如果中断被使能，将允许它触发，同时 Cause 寄存器的中断 pending 字段相应的位被置位。 0－禁止 1－允许
KSU	模式位 11 未定义 10 普通用户 01 超级用户 00 核心
KX	1 – 使能 64 位 Kernel 段访问；使用 XTLB Refill 向量。 0 – 不能访问 64 位的 Kernel 段；使用 TLB Refill 向量
SX	1 – 使能 64 位 Supervisor 段访问；使用 XTLB Refill 向量。 0 – 不能访问 64 位的 Supervisor 段；使用 TLB Refill 向量
UX	1 – 使能 64 位 User 段访问；使用 XTLB Refill 向量。 0 – 不能访问 64 位的 User 段；使用 TLB Refill 向量
ERL	错误级。当发生复位，软件复位，NMI 或 Cache 错误时处理器将重置此位。 0 正常 1 错误
EXL	例外级。当一个不是由复位，软件复位或 Cache 错误引发的例外产生时，处理器将设置该位。
IE	中断使能。 0 禁用所有中断 1 使能所有中断

Status 寄存器模式和访问状态

下面描述 Status 寄存器中用于设置模式和访问状态的域：

- **中断使能：**当符合以下条件时，中断被使能：
  - IE = 1 且

- EXL = 0 且
- ERL = 0。

如果遇到这些条件，IM 位的设置允许中断。

- **操作模式：**当处理器处于普通用户、内核和超级用户模式时需要设置下述位域。
  - 当 KSU = 10_2, EXL = 0 和 ERL = 0 时处理器处于普通用户态模式下。
  - 当 KSU = 01_2, EXL = 0 和 ERL = 0 时处理器处于超级用户态模式下。
  - 当 KSU = 00_2, or EXL = 1 或者 ERL = 1 时处理器处于内核态模式下。
- **内核地址空间访问：** 当处理器处在内核模式时，可以访问内核地址空间。
- **超级用户地址空间访问：**当处理器处在内核模式或超级用户模式时，可以访问超级用户地址空间。
- **用户地址空间访问：**处理器在这三种操作模式下都可以访问用户地址空间。

Status 寄存器复位

复位时，Status 寄存器的值是 0x30c000e4。

3.13 IntCtl 寄存器（12，1）

IntCtl 寄存器是一个可以读写的 32 位寄存器。它管理 Release2 体系中扩充的中断。龙芯 3 号实现了向量中断。用 IntCtl 寄存器的 VS 域来表示中断向量之间的向量空间。1 域不可写，读出为 1；0 域为保留，必须按 0 写入，读时返回 0。其中 31:26 位代表时钟中断和 Performance Counter 中断共用 HW5。

图 3-15 显示了 IntCtl 寄存器的格式，表 3-12 描述了 VS 域与向量空间的对应关系。

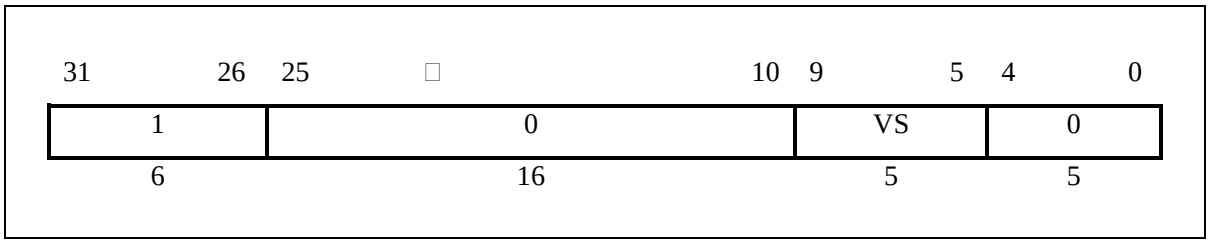


图 3-15 IntCtl 寄存器

表 3-12 VS 域的编码与向量空间对应关系

编码	向量空间（16 进制）	向量空间（10 进制）
0x00	0x000	0
0x01	0x020	32
0x02	0x040	64
0x04	0x080	128
0x08	0x100	256
0x10	0x200	512

3.14 SRSCtl 寄存器（12，2）

SRSCtl 寄存器是一个 32 位可读写寄存器。它掌管处理器中的影子寄存器组。由于龙芯 3 号只有一组通用寄存器，没有影子寄存器，所以通用寄存器的影子为通用寄存器本身，龙芯 3 号中的 SRSCtl 寄存器只实现两个域：ESS 和 PSS。

图 3-16 显示了 SRSCtl 寄存器的格式，表 3-13 描述了 SRSCtl 寄存器的域。

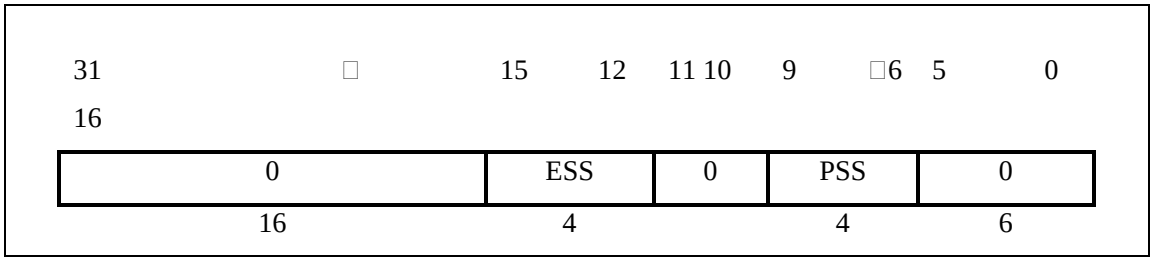


图 3-16 SRSCtl 寄存器

表 3-13 SRSCtl 寄存器的域

域	描述
ESS	用于例外的影子寄存器组。在龙芯 3 号中只能为 0
PSS	前一个影子寄存器组。在龙芯 3 号中只能为 0
0	保留。必须按 0 写入，读时返回 0。

3.15 Cause 寄存器(13，0)

32 位的可读写 Cause 寄存器描述最近一个例外发生的原因。

图 3-17 显示了这一寄存器的格式，表 3-14 描述了 Cause 寄存器的域。一个 5 位例外码（ExcCode）指出了原因之一，如表 3-15 所示。

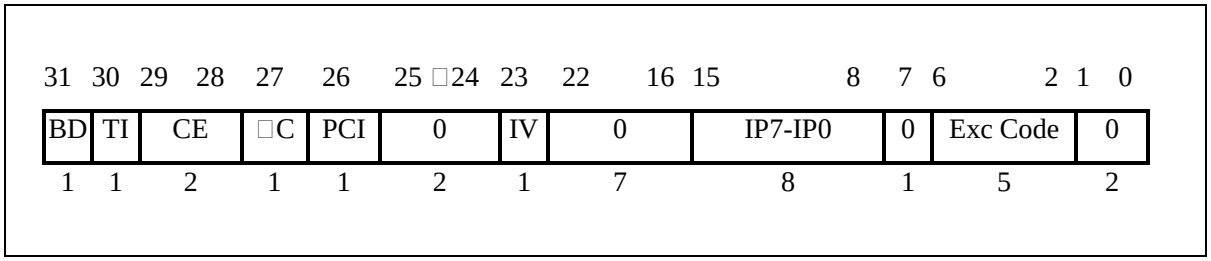


图 3-17 Cause 寄存器

表 3-14 Cause 寄存器域

域	描述
BD	指出最后采用的例外是否在分支延时槽中。 1－延时槽 0－正常
TI	时钟中断指示 0－没有时钟中断 1－有时间中断等待处理

CE	当发生协处理器不可用例外时协处理器的单元编号。
DC	禁用计数寄存器 0－计数寄存器可用 1－计数寄存器禁用
PCI	性能计数器中断指示 0－没有性能计数器中断 1－有性能计数器中断待处理
IV	中断例外入口 0－使用通用例外向量（0x180） 1－使用特殊中断向量（0x200）
IP	指出等待的中断。该位将保持不变直到中断撤除。IP0~IP1 是软中断位，可由软件设置与清除。 1－中断等待 0－没有中断
ExcCode	例外码域 (见表 3-15)
0	保留。必须按 0 写入，读时返回 0。

表 3-15 Cause 寄存器的 ExcCode 域

例外代码	Mnemonic	描述
0	Int	中断
1	Mod	TLB 修改例外
2	TLBL	TLB 例外（读或者取指令）
3	TLBS	TLB 例外（存储）
4	AdEL	地址错误例外（读或者取指令）
5	AdES	地址错误例外（存储）
6	IBE	总线错误例外（取指令）
7	DBE	总线错误例外（数据引用：读或存储）
8	Sys	系统调用例外
9	Bp	断点例外
10	RI	保留指令例外
11	CpU	协处理器不可用例外
12	Ov	算术溢出例外
13	Tr	陷阱例外
14	-	保留
15	FPE	浮点例外
16	IS	栈例外
17-18	-	保留
19	DIB	Debug 指令例外

20	DDBS	Debug 存数据例外
21	DDBL	Debug 取数据例外
22	-	保留
23	WATCH	WATCH 例外
24－25	-	保留
26	DBP	Debug 断点例外
27	DINT	Debug 调试例外
28	DSS	Debug 单步例外
29	-	保留
30	CACHERROR	cache 错例外
31	-	保留

3.16 Exception Program Counter 寄存器(14，0)

例外程序计数器（Exception Program Counter，简称 EPC）是一个读/写寄存器，它包括例外处理结束后的继续处理地址。

对于同步例外，EPC 寄存器的内容是下面之一：

- 指令虚地址，这是导致例外的直接原因，或者
- 之前的分支或者跳转指令（当指令在分支延时槽中，指令延时位在 Cause 寄存器中被置位）的虚地址。

当 Status 寄存器中的 EXL 位被置 1 时，处理器不写 EPC 寄存器。

图 3-18 显示了 EPC 寄存器的格式。

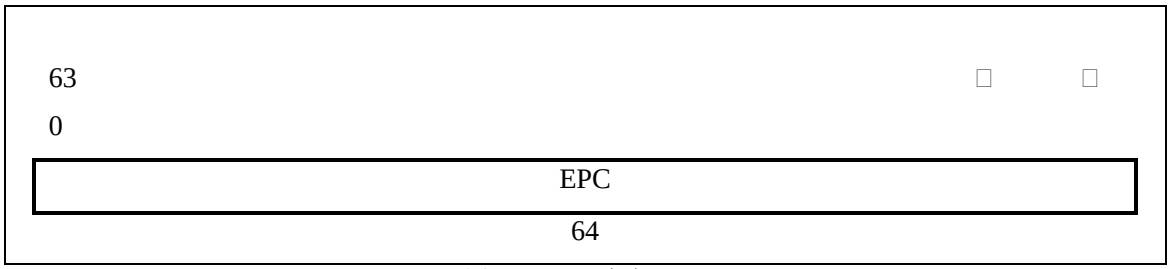


图 3-18 EPC 寄存器

3.17 Processor Revision Identifier (PRID)寄存器(15，0)

PRID 寄存器是个 32 的只读寄存器，该寄存器包含了标定处理器和 CP0 版本的实现版本和修订版本的信息。图 3-19 表示了该寄存器的格式；表 3-16 描述了该寄存器的域。

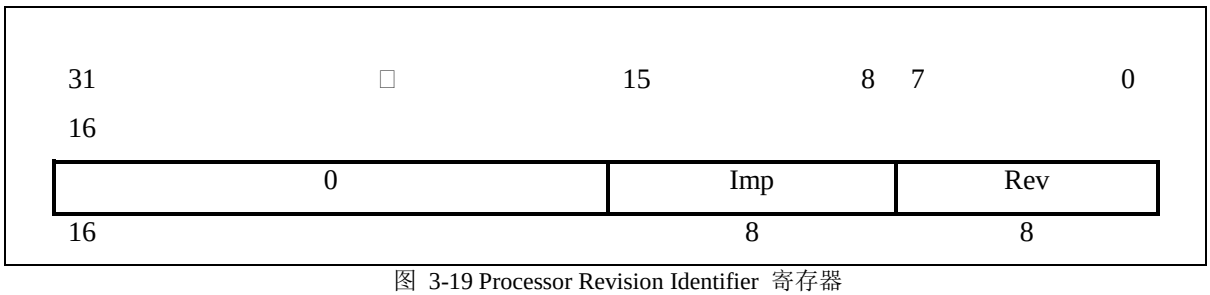


图 3-19 Processor Revision Identifier 寄存器

表 3-16 PRId 寄存器域

域	描述
IMP	实现版本号
REV	修订版本号
0	保留。必须按 0 写入，读时返回 0。

PRID 寄存器的低位（7：0 位）可用作修订版本的号码，而高位（15：8）位可用作实现版本的号码。龙芯 3 号 实现版本号为 0x63，修订版本号为 0x03。

版本号码的表示格式为 Y.X，其中 Y（7：4 位）为主要版本号，而 X（3：0 位）为小版本号。

版本号码可以区分一些处理器的版本，但不能保证处理器的任何改动要体现在 PRID 寄存器中，换句话说，不能保证版本号的改动必须体现处理器的修改。因为这个原因，寄存器的值没有给出，而软件也不能依赖 PRID 寄存器中的版本号来标识处理器。

3.18 EBase 寄存器(15，1)

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。

当状态寄存器的 BEV=0 时，使用 EBase 寄存器中的例外向量基址。

图 3-20 显示了 EBase 寄存器的格式，

表 3-17 描述了 EBase 寄存器的域。

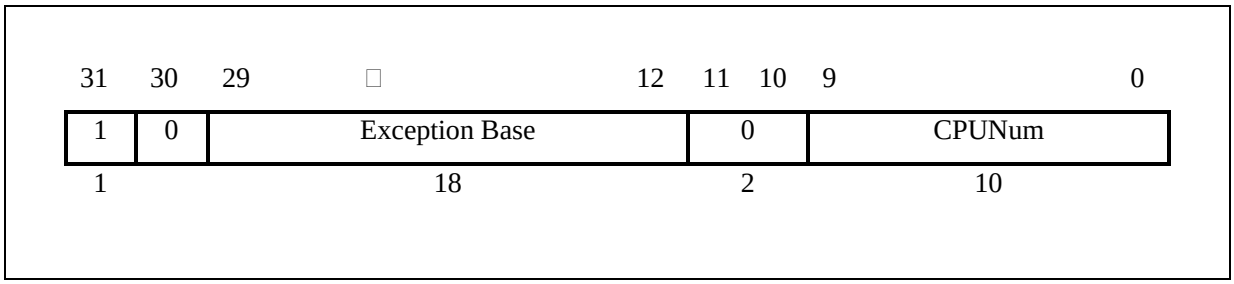


图 3-20 Ebase 寄存器

表 3-17 Ebase 寄存器域

域	描述
1	不可写只可读，读出为 1
0	保留。必须按 0 写入，读时返回 0。
Exception Base	与 31 位 30 位联合指明例外向量基址

CPUNum	在多核系统中，用于指明处理器号

3.19 Config 寄存器(16，0)

Config 寄存器规定了龙芯 3 处理器中各种配置选择项；表 3-18 列出了这些选项。

由 Config 寄存器的位 31:3 所定义的一些配置选项，在复位时由硬件设置，而且作为只读状态位包括在 Config 寄存器中，用于软件的访问。其他配置选项（Config 寄存器的位 2:0）是可读/写的并且由软件所控制。在复位时这些域是没有定义的。

Config 寄存器的配置是受限的。Config 寄存器在 Cache 被使用之前应该由软件来初始化，并且，在做了任何改变后 Cache 应该重新初始化。

图 3-21 表示了 Config 寄存器的格式；表 3-18 描述了 Config 寄存器的域。Config 寄存器的初值为 0x00030932。

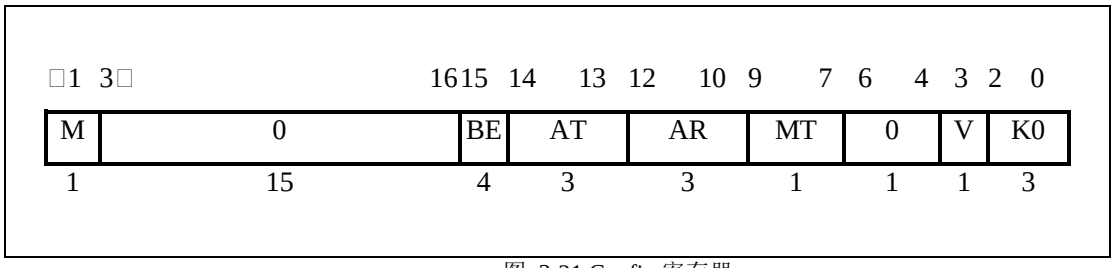


图 3-21 Config 寄存器

表 3-18 Config 寄存器域

域	描述
0	保留。必须按 0 写入，读时返回 0。
M	是否存在 config1 寄存器，置 1 表示存在。
BE	指明尾端类型 1 – 大尾端 0 – 小尾端
AT	指明体系结构类型 0 – MIPS32 1 – MIPS64，只能访问 32 位地址段 2 – MIPS64，可以访问所有地址段 3 – 保留
AR	指明版本 0 – Release 1 1 – Release 2 2-7 – 保留
MT	指明内存管理单元类型 0 – 无映射 1 – 标准 TLB 2-7 – 保留

VI	指明是否存在虚拟指令 Cache 0 – 指令 Cache 不是虚拟的 1 – 指令 Cache 是虚拟的
K0	Kseg0 一致性算法（Cache 一致性算法） 2 – Uncached 3 – Cacheable 其余 – 保留

3.20 Config1 寄存器(16，1)

Config1 寄存器规定了龙芯 3 号处理器的 cache 配置。

Config1 寄存器作为 Config 寄存器的附加内容，所有内容都是只读的，在复位时自动设置。

图 3-22 表示了 Config1 寄存器的格式；

表 3-19 描述了 Config1 寄存器的域。Config1 寄存器的初值为 0xfee37193。

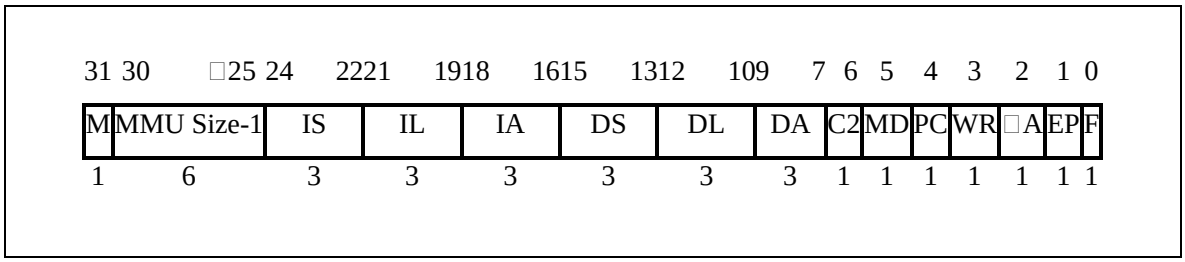


图 3-22 Config1 寄存器

表 3-19 Config1 寄存器域

域	描述
M	是否存在 config2 寄存器，置 1 表示存在。
MMU Size-1	TLB 表项数减 1
IS	Icache 每路组数 编码 / 含义 0 / 64 1 / 128 2 / □56 3 / □12 4 / 1024 5 / 2048 6 / 4096 7 / 保留
IL	Icache 每组大小 编码 / 含义 0 / 无 Icache 1 / 4 bytes

<table>
<tr><td></td><td>
<table>
<tr><td>2</td><td>8 bytes</td></tr>
<tr><td>3</td><td>16 bytes</td></tr>
<tr><td>4</td><td>32 bytes</td></tr>
<tr><td>5</td><td>64 bytes</td></tr>
<tr><td>6</td><td>128 bytes</td></tr>
<tr><td>7</td><td>保留</td></tr>
</table>
</td></tr>
<tr><td>IA</td><td>Icache 相联方式
<table>
<tr><td>编码</td><td>含义</td></tr>
<tr><td>0</td><td>直接相联</td></tr>
<tr><td>1</td><td>2 路相联</td></tr>
<tr><td>2</td><td>3 路相联</td></tr>
<tr><td>3</td><td>4 路相联</td></tr>
<tr><td>4</td><td>5 路相联</td></tr>
<tr><td>5</td><td>6 路相联</td></tr>
<tr><td>6</td><td>7 路相联</td></tr>
<tr><td>7</td><td>8 路相联</td></tr>
</table>
</td></tr>
<tr><td>DS</td><td>Dcache 每路组数
<table>
<tr><td>编</td><td>含义</td></tr>
<tr><td></td><td>64</td></tr>
<tr><td>1</td><td>128</td></tr>
<tr><td>2</td><td>256</td></tr>
<tr><td>3</td><td>512</td></tr>
<tr><td>4</td><td>1024</td></tr>
<tr><td>5</td><td>2048</td></tr>
<tr><td>6</td><td>4096</td></tr>
<tr><td>7</td><td>保留</td></tr>
</table>
</td></tr>
<tr><td>DL</td><td>Dcache 每组大小
<table>
<tr><td>编码</td><td>含义</td></tr>
<tr><td>0</td><td>无 Dcache</td></tr>
<tr><td>1</td><td>4 bytes</td></tr>
<tr><td>2</td><td>8 bytes</td></tr>
<tr><td>3</td><td>16 bytes</td></tr>
<tr><td>4</td><td>32 bytes</td></tr>
<tr><td>5</td><td>64□bytes</td></tr>
<tr><td>6</td><td>128 bytes</td></tr>
<tr><td>7</td><td>保留</td></tr>
</table>
</td></tr>
<tr><td>DA</td><td>Dcache 相联方式
<table>
<tr><td>编码</td><td>含义</td></tr>
<tr><td>0</td><td>直接相联</td></tr>
<tr><td>1</td><td>2 路相联</td></tr>
<tr><td>2</td><td>3 路相联</td></tr>
</table>
</td></tr>
</table>

	3 4 路相联 4 5 路相联 5 6 路相联 6 7 路相联 7 8 路相联
C2	2 号协处理器是否实现 0－未实现 1－实现
MD	MDMX ASE 是否实现 0－未实现 1－实现
PC	性能计数寄存器是否实现 0－未实现 1－实现
WR	Watch 寄存器是否实现 0－未实现 1－实现
CA	MIPS16e 是否实现 0－未实现 1－实现
EP	EJTAG 是否实现 0－未实现 1－实现
FP	FPU 是否实现 0－未实现 1－实现

3.21 Config 2 寄存器(16，2)

Config2 寄存器规定了龙芯 3 号处理器中二级 cache 的配置。

Config2 寄存器作为，所有内容都是只读的，在复位时自动设置。

图 3-23 表示了 Config1 寄存器的格式；表 3-20 描述了 Config2 寄存器的域。Config2 寄存器的初值为 0x80001643。

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

图 3-23 Config2 寄存器

表 3-20 Config2 寄存器域

<table>
<tr><th>域</th><th>描述</th></tr>
<tr><td>M</td><td>是否存在 config3 寄存器，置 1 表示存在。</td></tr>
<tr><td>TU</td><td>三级 cache 控制或状态位</td></tr>
<tr><td>TS</td><td>三级 cache 每路组数
<table><tr><td>编码</td><td>含义</td></tr><tr><td>0</td><td>6</td></tr><tr><td>1</td><td>128</td></tr><tr><td>2</td><td>256</td></tr><tr><td>3</td><td>512</td></tr><tr><td>4</td><td>1024</td></tr><tr><td></td><td>2048</td></tr><tr><td>6</td><td>4096</td></tr><tr><td>7</td><td>8192</td></tr><tr><td>8-15</td><td>保留</td></tr></table></td></tr>
<tr><td>TL</td><td>三级 cache 每组大小
<table><tr><td>编码</td><td>含义</td></tr><tr><td>0</td><td>无 Icache</td></tr><tr><td>1</td><td>4 bytes</td></tr><tr><td>2</td><td>8 bytes</td></tr><tr><td>3</td><td>16 bytes</td></tr><tr><td>4</td><td>32 bytes</td></tr><tr><td>5</td><td>64 bytes</td></tr><tr><td>6</td><td>128 bytes</td></tr><tr><td>7</td><td>□56 bytes</td></tr><tr><td>8-15</td><td>保留</td></tr></table></td></tr>
<tr><td>TA</td><td>三级 cache 相联方式
<table><tr><td>编码</td><td>含义</td></tr><tr><td>0</td><td>直接相联</td></tr><tr><td>1</td><td>2 路相联</td></tr><tr><td>2</td><td>3 路相联</td></tr><tr><td>3</td><td>4 路相联</td></tr><tr><td>4</td><td>5 路相联</td></tr><tr><td>5</td><td>6 路相联</td></tr><tr><td>6</td><td>7 路相联</td></tr><tr><td>7</td><td>8 路相联</td></tr><tr><td>□-15</td><td>保留</td></tr></table></td></tr>
<tr><td>SU</td><td>二级 cache 控制或状态位</td></tr>
<tr><td>SS</td><td>二级 cache 每路组数</td></tr>
</table>

	编码 / 含义 0 / 64 1 / 128 2 / □56 3 / 512 4 / 1024 5 / 2048 6 / 4096 7 / 8192 8-15 / 保留
SL	二级 cache 每组大小 编码 / 含义 0 / 无 Icache 1 / 4 bytes 2 / 8 bytes 3 / 16 bytes 4 / 32 bytes 5 / 64 bytes 6 / 128 bytes 7 / 256 □ytes 8-15 / 保留
SA	二级 cache 相联方式 编码 / 含义 0 / 直接相联 1 / 2 路相联 2 / 3 路相联 3 / 4 路相联 4 / 5 路相联 5 / 6 路相联 6 / 7 路相 / 8 路相联 8-15 / 保留

3.22 Config 3 寄存器(16，3)

Config3 寄存器标记了一些功能是否实现，所有内容都是只读的，在复位时自动设置。

图 3-24 表示了 Config1 寄存器的格式；表 3-21 描述了 Config1 寄存器的域。Config1 寄存器的初值为 0x000000a0。

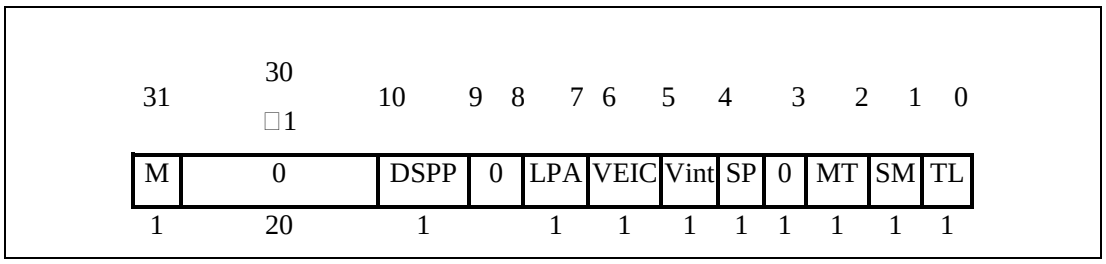


图 3-24 Config3 寄存器

表 3-21 Config3 寄存器域

域	描述
M	保留
0	保留。必须按 0 写入，读时返回 0。
DSPP	MIPS DSPASE 是否实现 0－未实现 1－实现
LPA	大物理地址是否实现 0－未实现 1－实现
VEIC	外部中断控制器是否实现 0－未实现 1－实现
Vint	向量中断是否实现 0－未实现 1－实现
SP	小页面支持是否实现 0－未实现 1－实现
MT	MIPS MTASE 是否实现 0－未实现 1－实现
SM	SmartMIPS ASE 是否实现 0－未实现 1－实现
TL	Trace Logic 是否实现 0－未实现 1－实现

3.23 Load Linked Address (LLAddr)寄存器(17, 0)

LLAddr 寄存器为 64 位只读寄存器。LLAddr 寄存器用于存放最近发生的 load-linked 指令的地址页号 PFN，当例外返回时（eret 指令发生时），LLAddr 寄存器被清零。在龙芯 3 号中该寄存器的格式如图 3-25 所示。

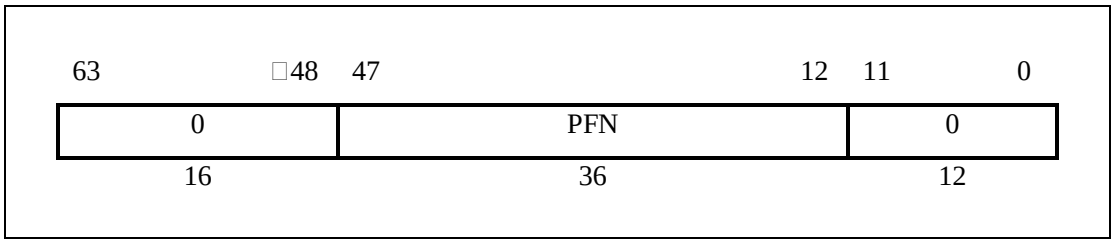


图 3-25 LLAddr 寄存器

3.24 XContext 寄存器(20, 0)

可读写的 XContext 寄存器包含了一个指向操作系统页表中一个表项的指针。当发生一个 TLB 例外时，操作系统根据失效的转换从页表加载 TLB。

XContext 寄存器用于 XTLB 重填处理，处理 64 位地址空间的 TLB 表项加载，并仅供操作系统使用。操作系统根据需要设置寄存器中的 PTEBase 域。

图 3-26 显示了 XContext 寄存器的格式；表 3-22 描述了 XContext 寄存器的域。

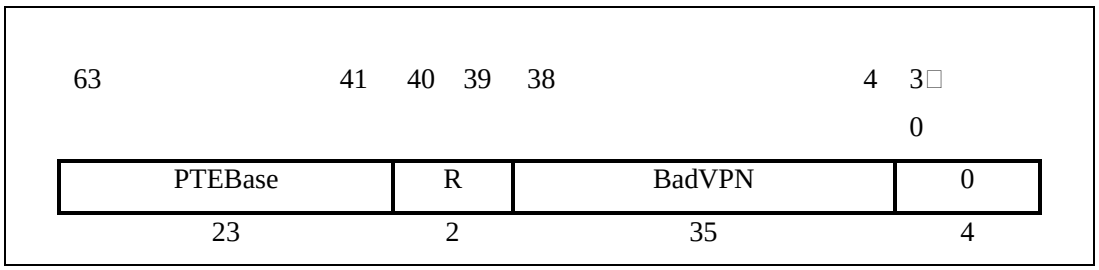


图 3-26 XContext 寄存器

35 位的 BadVPN2 域包含着引起 TLB 例外的虚拟地址中 47：13 位，因为单一 TLB 表项映射到一个奇偶页对，所以第 12 位并没有被包括在内。当页面大小是 4K 字节时，这一格式可以直接寻址 PTE 表项为 8 字节长且按对组织的页表。对于其他的页面和 PTE 大小，经过移位和掩码可以得到正确的地址。

表 3-22 XContext 寄存器域

域	描述
BadVPN2	当发生 TLB 例外时硬件将写这个域，它包含了最近无效虚地址的虚页号除 2。
R	该域包含虚地址的 63:62 位。 00 普通用户 01 超级用户

	11 □□□□
0	保留。必须按 0 写入，读时返回 0。
PTEBase	可读/写域，该值允许操作系统使用 XContext 寄存器作为一个指向内存中当前页表的指针。

3.25 Diagnostic 寄存器(22，0)

龙芯处理器特有的 64 位寄存器，主要用于控制处理器的一些内部队列和特殊操作。图 3-27 Diagnostic 寄存器显示了 Diagnostic 寄存器的格式，表 3-23 描述了 Diagnostic 寄存器的域。

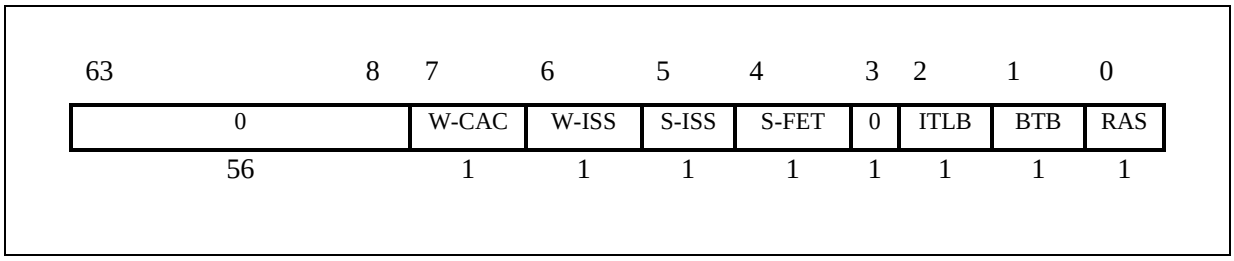


图 3-27 Diagnostic 寄存器

表 3-23 Diagnostic 寄存器域

域	描述
0	保留。必须按 0 写入，读时返回 0。
W-CAC	取消 wait-cache 操作的限制
W-ISS	取消 wait-issue 操作的限制
S-ISS	取消 store-issue 操作的限制
S-FET	取消 store-fetch 操作的限制
ITLB	写入 1 时清空 ITLB
BTB	写入 1 时清空 BTB
RAS	写入 1 时禁止使用 RAS。

3.26 Debug 寄存器(23，0)

Debug 寄存器是一个 32 位的可读写寄存器。Debug 寄存器包含最近发生的 debug 例外或者在 debug 模式下发生的例外的原因，它还控制单步中断。这个寄存器指明了 debug 资源和其他的内部状态。只有 LSNM 域和 SSt 域可写，在非 debug 模式下读取 Debug 寄存器时只能读取 DM 位和 EJTAGver 域。龙芯 3 号没实现 debug 例外时的省电模式。复位时，Debug 寄存器的初始值为：0x02018000。

图 3-28 表示了 Debug 寄存器的格式，表 3-24 为 Debug 寄存器的域。

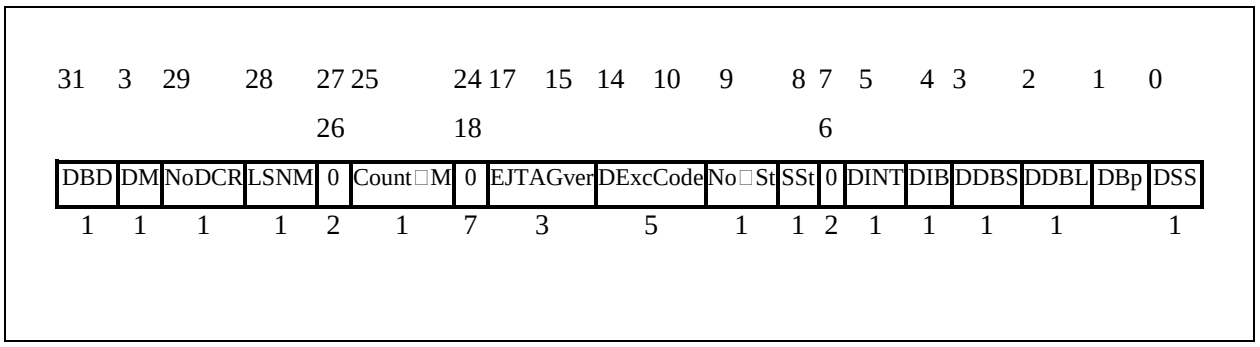


图 3-28 Debug 寄存器

表 3-24 Debug 寄存器的域

域	描述
0	保留。必须按 0 写入，读时返回 0。
DBD	指明上一条 debug 例外是否发生在延迟槽中。 0 – 非 1 – 是
DM	指明处理器是否处于 Debug 模式 0 – Non-Debug Mode 1 – Debug Mode
NoDCR	指明 dseg 段是否存在 1 – 不存在 0 – 存在
LSNM	当 dseg 段存在时，指明 loads/stores 可用的地址 0 – dseg 段 1 – 系统内存
CountDM	进入 DM 时 Count 寄存器的工作状态 0 – stopped 1 – runnig
EJTAGver	EJTAG 版本 0 – 版本 1 和 2.0 1 – 版本 2.5 2 – 版本 2.6 3 – 版本 3.1 4 – 保留
DexcCode	指明最近一次在 Debug 模式下的例外原因
NoSSt	指明是否支持单步中断 0 – 支持 1 – 不支持
SSt	单步中断使能位 0 – 不可用

	1 – 使能
DINT	置位表示有 Debug 中断例外发生 当进入 Debug 模式后自动清零
DIB	置位表示有 Debug 指令中断例外发生 当进入 Debug 模式后自动清零
DDBS	置位表示有 Debug 数据中断例外发生 当进入 Debug 模式后自动清零
DBp	置位表示有 Debug 断点例外发生 当进入 Debug 模式后自动清零
DSS	置位表示有 Debug 单步中断例外发生 当进入 Debug 模式后自动清零

Debug 寄存器中的位或域，只有在 debug 例外或 debug 模式下例外发生时才被更新。

3.27 Debug Exception Program Counter 寄存器(24, 0)

Debug 例外程序计数器（DEPC）是一个 64 位的读/写寄存器，它包括例外处理结束后的继续处理地址。此寄存器由硬件在 debug 例外或 debug 模式下的例外时更新。

对于精确的 debug 例外和精确的 debug 模式下的例外，DEPC 寄存器的内容是下面之一：

- 指令虚地址，这是导致例外的直接原因，或者
- 之前的分支或者跳转指令（当指令在分支延时槽中，指令延时位 DBD 在 Debug 寄存器中被置位）的虚地址。

图 3-29 显示了 DEPC 寄存器的格式。

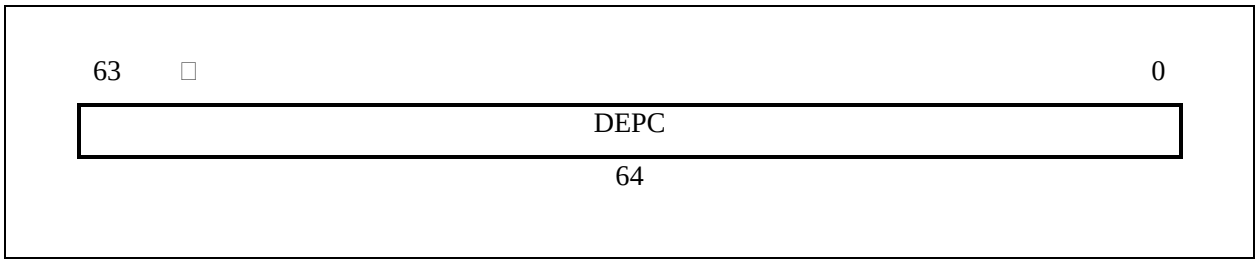


图 3-29 DEPC 寄存器

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

龙芯 3 号处理器定义了四个（两组）性能计数器，他们分别映射到 CP0 寄存器的 24 号的 sel 0，sel 1，sel 2 和 sel 3。

龙芯 3 号在复位时，为 PerfCnt 寄存器的两个控制寄存器赋的初始值分别为：

PerfCnt，select 0 =0xc0000000

PerfCnt，select 2 =0x40000000

这四个寄存器的用途如表 3-25 所示，每种寄存器的格式如图 3-30 所示（两组格式相同），控制寄存器的使能位定义由表 3-26 所示：

表 3-25 性能计数器列表

性能计数器	sel	用途描述
0	select 0	控制寄存器 0
	select 1	计数寄存器 0
1	select 2	控制寄存器 1
	select 3	计数寄存器 1

每个计数器都是 64 位的读/写寄存器，并且在每次关联控制域中可数事件发生时自增。每个计数器都可以独立对一种事件计数。

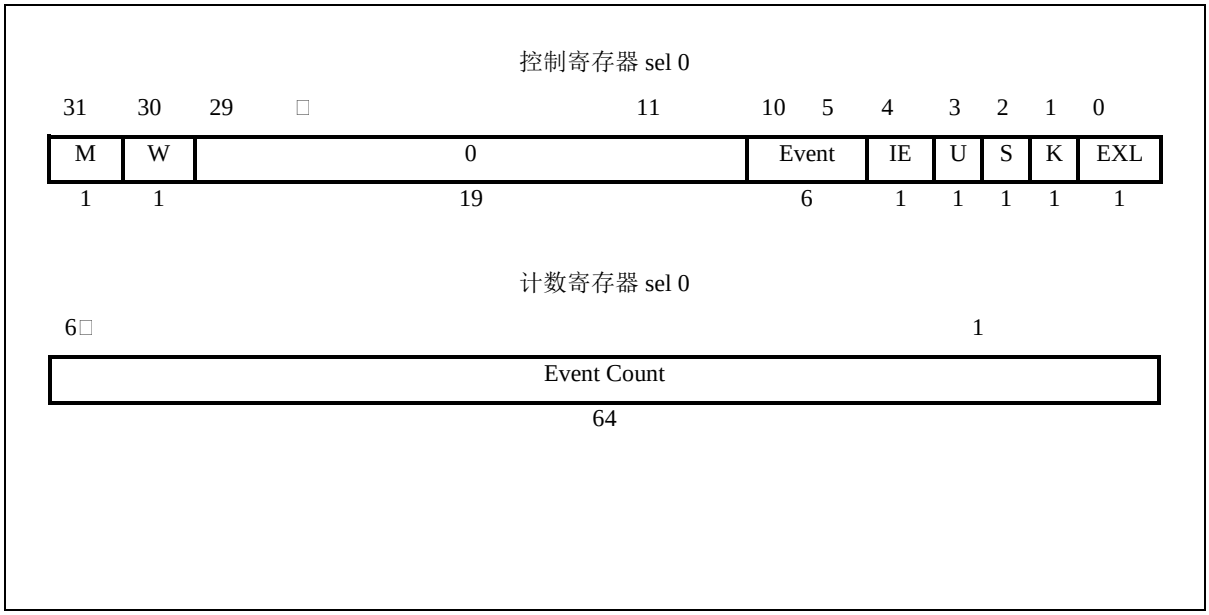


图 3-30 性能计数器寄存器

当计数器的首位（63 位）变成 1（计数器溢出）时，计数器将触发一个中断，Cause 寄存器 PCI 位被置一（若有多组计数器，则多组计数器的溢出位或）。在计数器溢出后无论中断是否被告知，计数都将继续。表 3-26 描述计数使能位的定义。表 3-27 和

表 3-28 描述计数器 0 和计数器 1 各自的事件。

表 3-26 计数使能位定义

计数使能位	Count Qualifier(CP0 Status 寄存器域)
M	是否存在另一组计数器 1 – 是 0 – 否
W	计数寄存器位宽 0 – 32 bits 1 – 64 bits

K	KSU = 0 (内核模式), EXL = 0, ERL = 0
S	KSU = 1 (超级用户模式), EXL = 0, ERL = 0
U	KSU = 2 (普通用户模式), EXL = 0, ERL = 0
EXL	EXL = 1, ERL = 0

表 3-27 计数器 0 事件

事件	信号	描述
0000	Cycles	周期
0001	Brbus.valid	分支指令
0010	Jrcount	JR 指令
0011	Jr31count	JR 指令并且域 rs=31
0100	Imemread.valid& imemread_allow	一级 I-cache 缺失
0101	Rissuebus0.valid	Alu1 操作已发射
0110	Rissuebus2.valid	Mem 操作已发射
0111	Rissuebus3.valid	Falu1 操作已发射
1000	Brbus_bht	BHT 猜测指令
1001	Mreadreq.valid& Mreadreq_allow	从主存中读
1010	Fxqfull	固定发射队列满的次数
1011	Roqfull	重排队列满的次数
1100	Cp0qfull	CP0 队列满的次数
1101	Exbus.ex & excode=34,35	Tlb 重填例外
1110	Exbus.ex & Excode=0	例外
1111	Exbus.ex & Excode=63	内部例外

表 3-28 计数器 1 事件

事件	信号	描述
0000	Cmtbus?.valid	提交操作
0001	Brbus.brerr	分支预测失败
0010	Jrmiss	JR 预测失败
0011	Jr31miss	JR 且 rs=31 预测失败
0100	Dmemread.valid& Dmemread_allow	一级 D-cache 缺失
0101	Rissuebus1.valid	Alu2 操作已发射

0110	Rissuebus4.valid	Falu2 操作已发射
0111	Duncache_valid& Duncache_allow	访问未缓存
1000	Brbus_bhtmiss	BHT 猜测错误
1001	Mwritereq.valid& Mwritereq_allow	写到主存
1010	Ftqfull	浮点指针队列满的次数
1011	Brqfull	分支队列满的次数
1100	Exbus.ex & Op==OP_TLBPI	Itlb 缺失
1101	Exbus.ex	例外总数
1110	Mispec	载入投机缺失
1111	CP0fwd_valid	CP0 队列向前加载

3.29 ECC 寄存器(26，0)

龙芯 3 号将 MIPS64 标准里可选的 26 号 ErrCtl 寄存器用于 ECC 校验。

图 3-31 显示了 DEPC 寄存器的格式。表 3-29 描述了 ECC 寄存器的域。

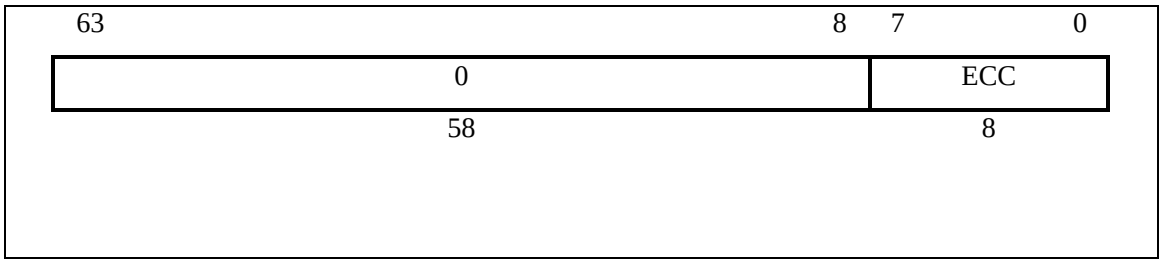


图 3-31 ECC 寄存器

表 3-29 ECC 寄存器的域

域	描述
0	保留。必须按 0 写入，读时返回 0。
ECC	相关 Cache 的一个双字校验码

3.30 CacheErr 寄存器(27，0/1)

龙芯 3 号将 MIPS64 标准里可龙芯 3 号的 ECC 校验由软硬件共同完成，硬件只负责检查错误，检查到数据错误后把内容保存在 CacheErr 等控制寄存器中，并发例外由软件进行纠正错误。

图 3-32 显示了 CacheErr 寄存器的格式，

表 3-30 描述了 CacheErr 寄存器的域。图 3-33 现实了 CacheErr1 寄存器的格式，表 3-31 描述了 CacheErr1 寄存器的域。

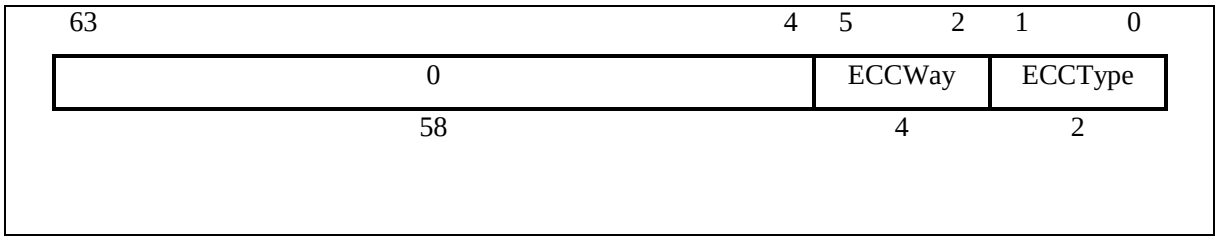


图 3-32 CacheErr 寄存器

表 3-30 CacheErr 寄存器的域

域	描述
0	保留。必须按 0 写入，读时返回 0。
ECCWay	不同编码在 Cache 校验错的情况下表示 Cache 的不同错
ECCType	00－指令 cache 错 01－数据 cache 错 10－二级 cache 错 11－芯片接口总线错

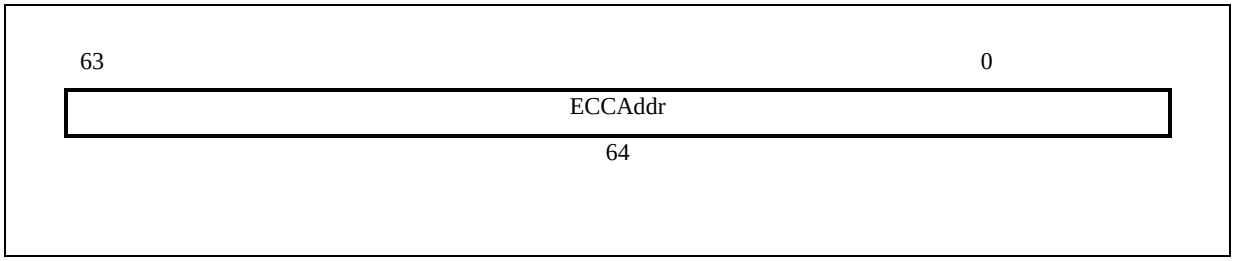


图 3-33 CacheErr1 寄存器

表 3-31 CacheErr1 寄存器的域

域	描述
ECCAddr	发生校验错的虚地址

3.31 TagLo（28）和 TagHi （29）寄存器

TagLo 和 TagHi 寄存器是 32 位读/写寄存器，用于保存一级/二级 Cache 的标签和状态，使用 CACHE 和 MTC0 指令往 Tag 寄存器写。

图 3-34 显示了这些寄存器用于一级 Cache（P-Cache）操作的格式。表 3-32 列出了 TagLo 和 TagHi 寄存器中域的定义。

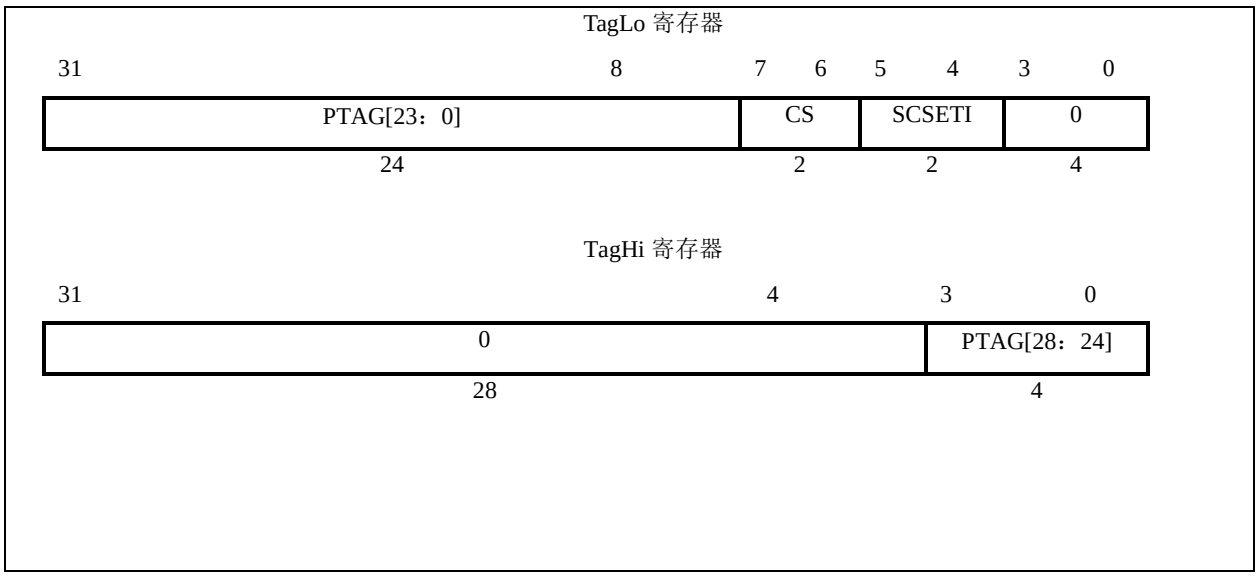


图 3-34 TagLo 和 TagHi 寄存器(P-Cache)

表 3-32 Cache Tag 寄存器域

域	描述
PTAG	指定物理地址的 39:12 位。
CS	指定 Cache 的状态。
SCSETI	对应 Cache 行在二级 Cache 的组号（二级 Cache 该域为 0）
0	保留。必须按 0 写入，读时返回 0。

3.32 DataLo（28，1）和 DataHi （29，1） 寄存器

DataLo 和 DataHi 是只读寄存器，只用于和 cache 数据队列交互和诊断。CACHE 指令的 IndexLoadTag 操作将读取相应数据到 DataLo 或 DataHi 寄存器。图 3-35 分别列出了 DataLo 寄存器和 DataHi 寄存器的格式。

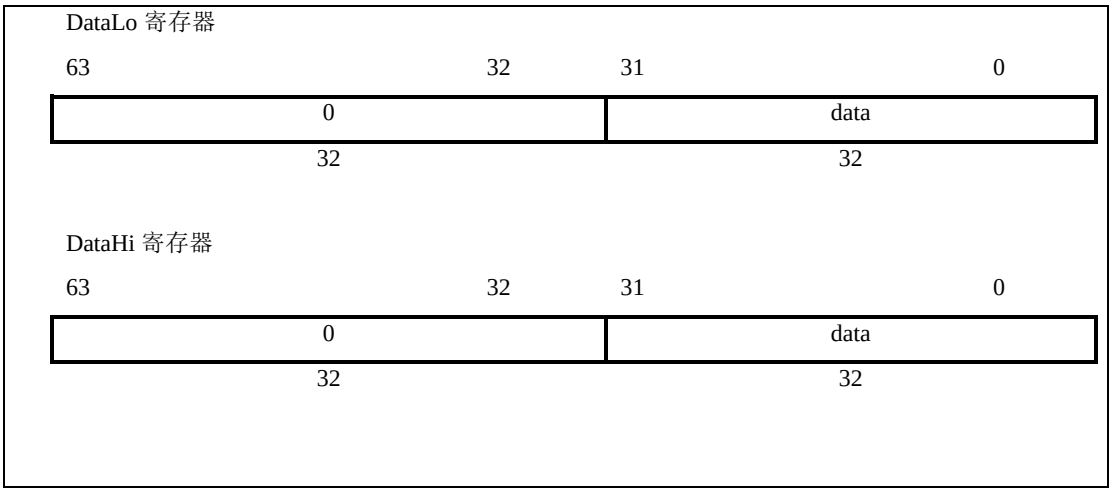


图 3-35 DataLo 和 DataHi 寄存器

3.33 ErrorEPC 寄存器(30，0)

除了用于 ECC 和奇偶错误例外外，ErrorEPC 寄存器与 EPC 寄存器类似。它用于在复位、软件复位、和不可屏蔽中断（NMI）例外时存储程序计数器。

ErrorEPC 是一个读写寄存器，它包括处理一个错误后指令重新开始执行的虚拟地址。图 3-36 显示了 ErrorEPC 寄存器的格式。

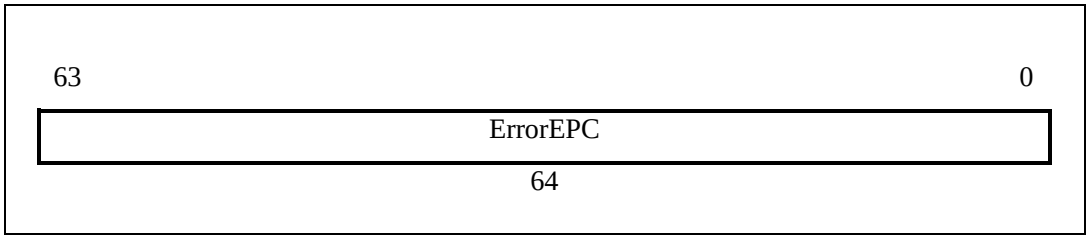


图 3-36 ErrorEPC 寄存器

3.34 DESAVE 寄存器(31，0)

DESAVE 寄存器是一个可读写的 64 位寄存器。他的功能是一个简单的暂存器，用于 debug 异常处理时保存一个通用寄存器的值，以便这个通用寄存器保留其他的上下文。

图 3-37 为 DESAVE 寄存器的格式。

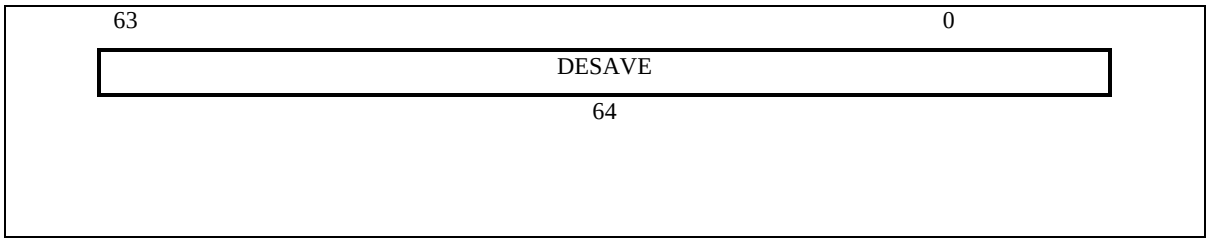


图 3-37 DESAVE 寄存器

3.35 CP0 指令

表 3-33 列出了 Godson-2 处理器定义的 CP0 指令。

表 3-33 CP0 指令

OpCode	Description	MIPS ISA
DMFC0	从 CP0 寄存器取双字	III
DMTC0	往 CP0 寄存器写双字	III
MFC0	从 CP0 寄存器取	I
MTC0	往 CP0 寄存器写	I
TLBR	按 Index 读 TLB 表项	III
TLBWI	按 Index 写 TLB 表项	III
TLBWR	按 Random 写 TLB 表项	III
TLBP	查找在 TLB 配对索引	III

CACHE	Cache 操作	III
ERET	异常返回	III
DERET	Debug 返回	EJTAG
DI	关闭中断	MIPS32 R2
EI	开启中断	MIPS32 R2
RDHWR	读取硬件寄存器	MIPS32 R2
RDPGPR	从影子寄存器中读取	MIPS32 R2
WRPGPR	写到影子寄存器	MIPS32 R2
SDBBP	软件断点	EJTAG

相关

龙芯处理器能够处理硬件中的流水线相关，包括 CP0 相关和访存相关，因此 CP0 指令并不需要 NOP 指令来校正指令序列。

4 CACHE 的组织和操作

龙芯 3 号使用了三种独立的 Cache：

一级指令 Cache：共 64KB，Cache 行大小 32 字节，采用四路组相联的结构。

一级数据 Cache：共 64KB，Cache 行大小 32 字节，采用四路组相联的结构，采用写回策略。

二级混合 Cache：片上 Cache，通过 128 位 AXI 总线和处理器核通讯，共有 4 个二级 Cache 模块。每个二级 Cache 模块全局编址，Cache 行大小 32 字节，1M 容量，采用四路组相联的结构，采用写回策略。

4.1 Cache 概述

定点访问一次一级 Cache 需要 3 个时钟周期，浮点访问一次一级 Cache 需要 4 个时钟周期。每个一级 Cache 都有它们自己的数据通路，从而可以同时访问两个 Cache。一级 Cache 的读、写和重填通路都是 128 位。

二级 Cache 使用的是 256 位的数据通路，它只有在一级 Cache 失效时才被访问。二级 Cache 和一级 Cache 不能同时访问，当一级 Cache 失效时，访问二级 Cache 会增加至少 14 个周期的失效代价（在龙芯 3 号中处理器核和二级 Cache 需要通过交叉开关通讯，因此还需要额外的 6 拍）。二级 Cache 以每个时钟周期 128 位数据的速度对一级 Cache 进行回填。

一级 Cache 采用虚地址索引和物理地址标志，而二级 Cache 的索引和标志采用的都是物理地址。虚地址索引可能会引起不一致问题，目前龙芯 3 号使用操作系统保证不会引起不一致问题。

龙芯 3 号采用了基于目录的 Cache 一致性协议，保证每个处理器核和 IO 的写能被其它处理器核及 IO 正确观察到。在二级 Cache 中维护了一个目录。对每个二级 Cache 中的 Cache 行，目录用 32 位的位向量记录每个一级 Cache（包括指令和数据 Cache）是否拥有该 Cache 行的备份。龙芯 3 号利用硬件维护了一级指令 Cache、一级数据 Cache、二级 Cache 以及 HT 外部设备之间的一致性，软件不需要通过 Cache 指令刷 Cache 来维护一致性。

4.1.1 非阻塞 Cache

龙芯 3 号实现了非阻塞 Cache 技术。非阻塞 Cache 是通过允许 Cache 失效访存操作后面的多个 Cache 失效或命中的访存操作继续进行，来提高系统的整体性能。

在一个阻塞 Cache 的设计中，当发生某个 Cache 失效时，处理器将暂停后续的访存操作。此时，处理器开始一个存储周期，取出被请求的数据，将其填入 Cache 中，然后再恢复执行。这个操作过程会占用较多的时钟周期，具体多少取决于存储器系统的设计。

然而在非阻塞 Cache 设计中，Cache 并不会在某个失效上暂停。龙芯 3 号支持多重失效下的命中，它最多可以支持 24 次 Cache 失效。

当一级 Cache 失效时，处理器会检查二级 Cache，看所需数据是否在其中，若二级 Cache 仍然失效，则需要访问主存储器。

龙芯 3 号中的非阻塞 Cache 结构能更有效的使用循环展开和软件流水。为了尽可能最大限度地发挥 Cache 的优势，在使用访存数据的指令之前，应尽可能早的执行相应的 Load 操作。

针对那些需要顺序存取的 I/O 系统，龙芯 3 号的默认设置是采用阻塞式的 Uncached 访问方式。

龙芯 3 号提供了预取指令，可以通过 load 到 0 号定点寄存器的方式来将数据预取到一级数据 Cache。此外龙芯 3 号中的 DSP 引擎可以将内存或 IO 中的数据预取到二级 Cache 中。

4.1.2 替换策略

一级 Cache 和二级 Cache 均采用随机替换算法。但二级 Cache 提供了锁机制。通过配置锁窗口寄存器，可以确保最多 4 个被锁住的区域不被替换出二级 Cache（具体配置方法见《龙芯 3A1000 处理器用户手册》上第 4 章）。

4.1.3 Cache 的参数

表 4-1 给出了三个 Cache 的一些参数

表 4-1 Cache 参数

参数	指令 Cache	数据 Cache	二级 Cache
Cache 大小	64KB	64KB	1MB（共 4MB）
相联度	4 路组相联	4 路组相联	4 路组相联
替换策略	随机法	随机法	随机法（可锁定）
块大小(line size)	32 字节	32 字节	32 字节
索引(Index)	虚地址 13:5 位	虚地址 13:5 位	物理地址 17:5 位 [1]
标志(Tag)	物理地址 47:12 位	物理地址 47:12 位	物理地址 47:12 位
写策略	不可写	写回法	写回法
读策略	非阻塞（2 个同时）	非阻塞(24 个同时)	非阻塞（8 个同时）
读顺序	关键字优先	关键字优先	关键字优先
写顺序	不可写	顺序式	
校验手段	奇偶校验	ECC 校验	ECC 校验

4.2 一级指令 Cache

一级指令 Cache 大小是 64KB，采用的是四路组相联结构。Cache 块大小（通常也被称作 Cache 行）为 32 字节，可以存放 8 条指令。由于龙芯 3 号采用 128 位的读通路，所以每个时钟周期可以取四条指令送到超标量调度单元。

一级指令 Cache 实现了奇偶校验。当读一级指令 Cache 发生了奇偶校验错时，硬件会自动将相应的 Cache 行会被置成无效，并从二级 Cache 中获取正确的值。整个过程无需软件干预。

[1] 一个物理地址具体落在 4 个二级 Cache 模块中哪一个根据交叉开关的路由配置寄存器决定。

4.2.1 指令 Cache 的组织

图 4-1 给出了一级指令 Cache 的组织结构。该 Cache 采用四路组相联的映射方式，其中每组包括 512 个索引项。根据索引(Index)选择相应的标志(Tag)和数据(Data)。从 Cache 读出 Tag 后，它被用来和虚地址中的被转换的部分进行比较，从而确定包含正确数据的组。

当一级指令 Cache 被索引时，四个组都会返回它们相应的 Cache 行，Cache 行大小为 32 字节，Cache 行采用了 34 位作为标志和 1 位作为有效位。

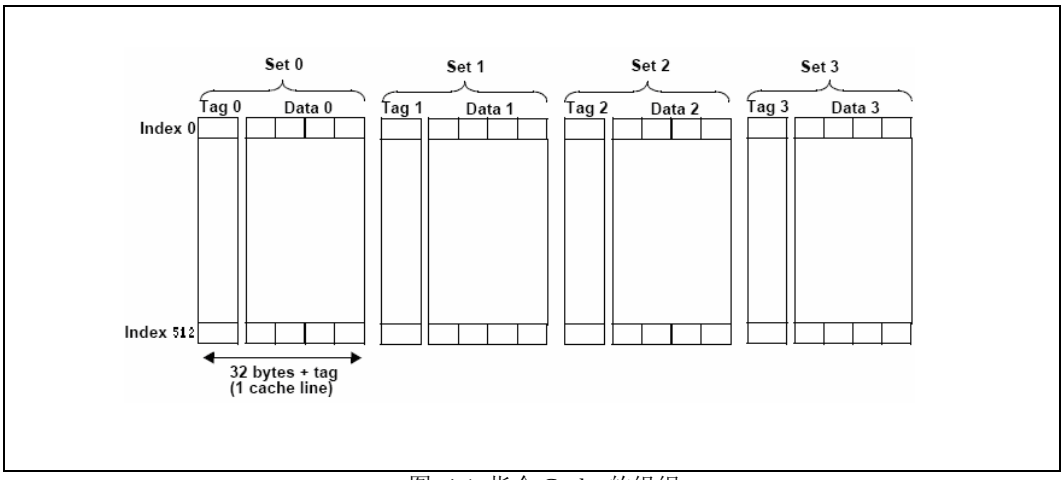


图 4-1 指令 Cache 的组织

4.2.2 指令 Cache 的访问

龙芯 3 号指令 Cache 采用虚地址索引和物理地址标志的四路组相联结构。如图 4-2 所示，地址的低 14 位被用作指令 Cache 的索引。其中 13:5 位用于索引 512 个项。其中每个项中又包含四个 64 位的双字，使用 4:3 位在这四个双字中进行选择。

当对 Cache 索引时，从 Cache 中取出四个块中的 Data 和相应的物理地址 Tag，同时，高位地址通过指令 TLB(Instruction Translation Look-aside Buffer，简称 ITLB)进行转换，将转换后的地址与取出的四个组中的 Tags 进行比较，若存在一个 Tag 与其匹配，则使用该组中的数据。这就被称为一次“一级 Cache 命中(Hit)”。若四组的 Tag 都不与其匹配，那么中止操作，并开始访问二级 Cache。这就被称为“一级 Cache 失效(miss)”。

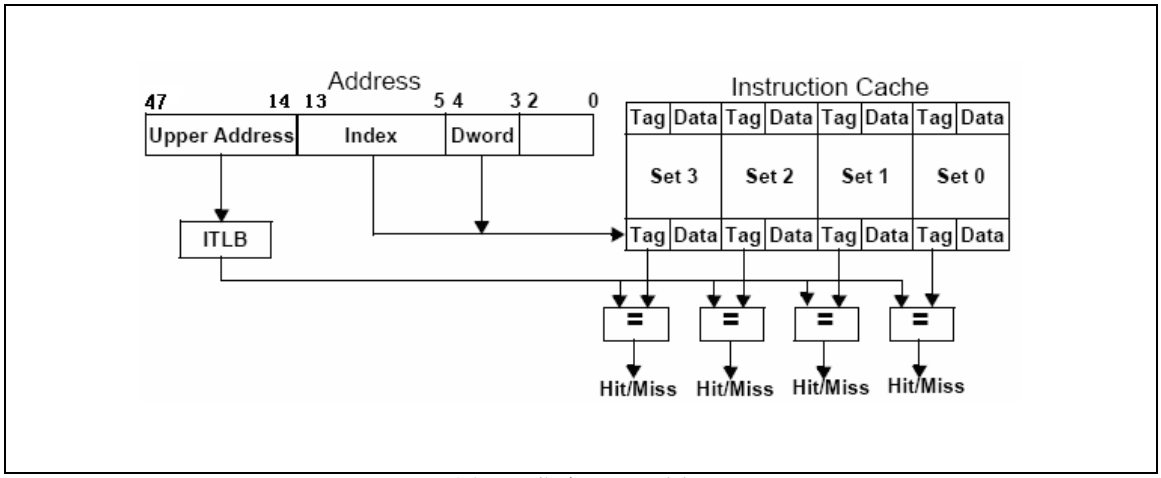


图 4-2 指令 Cache 访问

4.3 一级数据 Cache

数据 Cache 的容量为 64KB，采用四路组相联的结构。Cache 块大小为 32 字节，即可以存放 8 个字。数据 Cache 的读写数据通路都是 128 位。

数据 Cache 使用的是虚地址索引，物理地址标志。操作系统需要解决可能由虚地址引起的页着色一致性问题。数据 Cache 是非阻塞的，也就意味着，数据 Cache 中的一次失效不会引起流水线的停顿。

数据 Cache 采用的写策略是写回法，即写数据到一级 Cache 的操作不会引起二级 Cache 和主存的更新。写回策略减少了一级 Cache 到二级 Cache 的通信量，从而提高了全局性能。只有在数据 Cache 行被替换出去时，数据才会被写到二级 Cache 中。

一级数据 Cache 实现了 ECC 校验。当读一级数据 Cache 发生了一位 ECC 校验错时，硬件会自动校正 Cache 读出的结果并将 Cache 中的内容更新为校正后的值。整个过程无需软件干预。当读一级数据 Cache 发生了二位 ECC 校验错时，将会发生例外留待软件处理。

4.3.1 数据 Cache 的组织

图 4-3 给出了数据 Cache 的组织结构。这是一个四路组相联的 Cache，其中含有 512 个索引项。当对 Cache 索引时，同时访问四个组中的 Tag 和 Data。然后将四个组中的 Tag 与转换后的物理地址部分进行比较，从而确定命中哪一个数据行。

当索引数据 Cache 时，四个组中都会返回它们各自相应的 Cache 行。Cache 块大小为 32 字节，Cache 行使用了 34 位作为物理标志地址，1 位作为脏位和 2 位作为状态位（包括 INV，SHD 和 EXC 三种状态）。INV 状态表示该 Cache 行无效，SHD 状态表示该 Cache 行可读，EXC 状态表示该 Cache 行可读可写。

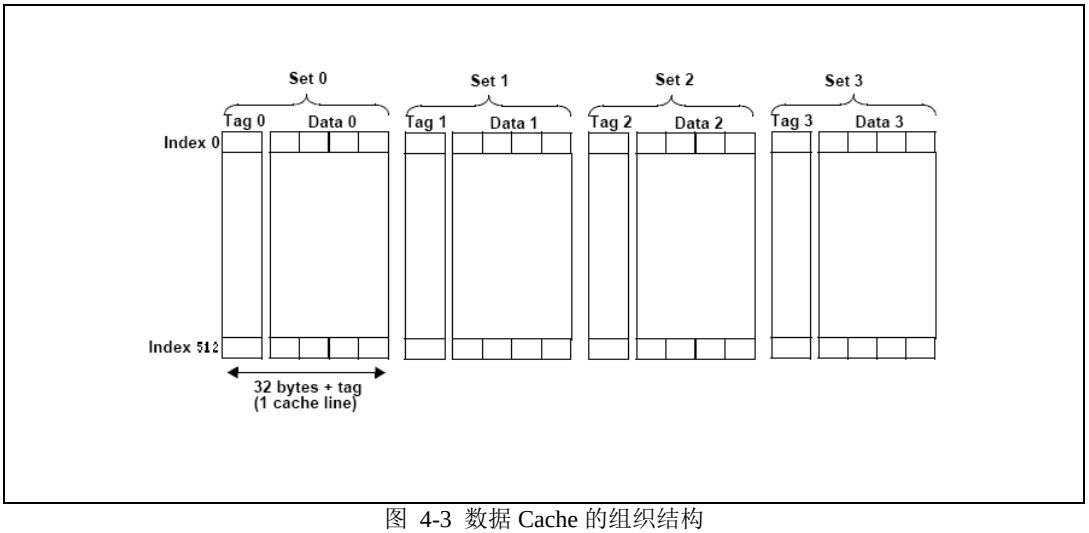


图 4-3 数据 Cache 的组织结构

4.3.2 数据 Cache 的访问

龙芯 3 号数据 Cache 采用虚地址索引和物理地址标志的四路组相联结构。图 4-4 给出了访问一次数据 Cache 时，虚地址如何被分解。

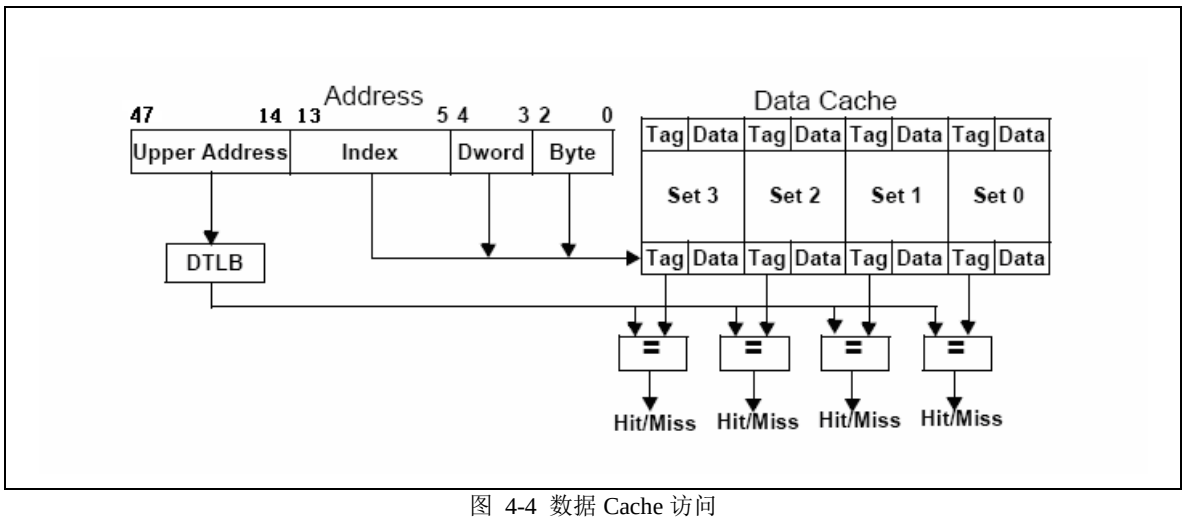


图 4-4 数据 Cache 访问

如图 4-4 所示，地址的低 14 位用作对数据 Cache 的索引。其中 13:5 位用作索引 512 个项，其中每个项又包括 4 个 64 位的双字。使用 4:3 位对四个双字进行选择，2:0 位用作选择一个双字的八个字节中的某一个字节。

数据 Cache 访问失效的指令（访存指令不命中或写指令命中 SHD 状态的 Cache 行），则访问二级 Cache。如果二级 Cache 命中，则将从二级 Cache 取回的 Cache 块送回一级 Cache。如果二级 Cache 失效，则访问内存，用从内存取回的值填充二级 Cache 和数据 Cache。

4.4 二级 Cache

龙芯 3 号中包括了 4 个片上二级 Cache 模块。每个二级 Cache 模块的容量为 1MB，共 4MB。每个 Cache 行大小为 32 字节。二级 Cache 模块的主要特征包括：采用 128 位 AXI 接口，四路组相连，8 项 Cache 访问队列，关键字优先，接收读失效请求到返回数据最快 8 拍，通过目录支持 Cache 一致性协议，可用于片上多核结构（也可直接和单处理器 IP 对接），软 IP 级可配置二级 Cache 模块的大小（512KB/1MB），采用四路组相联结构，运行时可动态关闭，支持 ECC 校验，支持 DMA 一致性读写和预取读，支持 16 种二级 Cache 散列方式，支持按窗口锁二级 Cache，保证读数据返回原子性。

二级 Cache 还维护了每个 Cache 行的目录，以记录每个一级 Cache 中是否包含该 Cache 行的备份。二级 Cache 采用的写策略是写回法。写回策略减少了总线的通信量，从而提高了系统的全局性能。只有在二级 Cache 行被替换出去时，数据才会被写到内存中。

二级 Cache 实现了 ECC 校验。当读二级 Cache 发生了一位 ECC 校验错时，硬件会自动校正 Cache 读出的结果并将 Cache 中的内容更新为校正后的值。整个过程无需软件干预。当读二级数据 Cache 发生了二位 ECC 校验错时，将会发生例外留待软件处理。

4.4.1 二级 Cache 的组织

二级 Cache 是混合 Cache，其中既包括指令也包括数据。二级 Cache 模块支持 Cache 一致性协议。龙芯 3 号中所有片上二级 Cache 统一编址，每个二级 Cache 块都有固定的 home 结点。根据 Cache 一致性的要求，龙芯 3 号的二级 Cache 具有两方面的角色：对一级 Cache 来说是 home，对于内存来说是 Cache。当访问二级 Cache 时，同时访问四组的 Data 和 Tag，将取出的四个 Tag 分别和访问的物理

地址高位部分进行比较，来确定数据是否还驻留在 Cache 中。

每个 Cache 行包含一个 32 字节的数据，31 位的物理地址标志，1 位 Cache 状态位（表示相应的 Cache 行在二级 Cache 中是否有效），1 位目录状态位（表示相应的 Cache 块是否在某个一级 Cache 中处于独占或共享状态）和 1 位 W 位（表示该行是否被写过）。

4.4.2 二级 Cache 的访问

只有在一级 Cache 失效的情况下，才访问二级 Cache。二级 Cache 采用的是物理地址索引物理地址标志。如图 4-5 所示，低位地址用来索引二级 Cache。四个组中都会返回它们各自相应的 Cache 行。16:5 位被用作二级 Cache 的索引。每个被索引项都含有 4 个 64 位的双字数据。使用 4:3 位在 4 个双字中进行选择。2:0 位用于选择一个双字中的某 8 个字节。

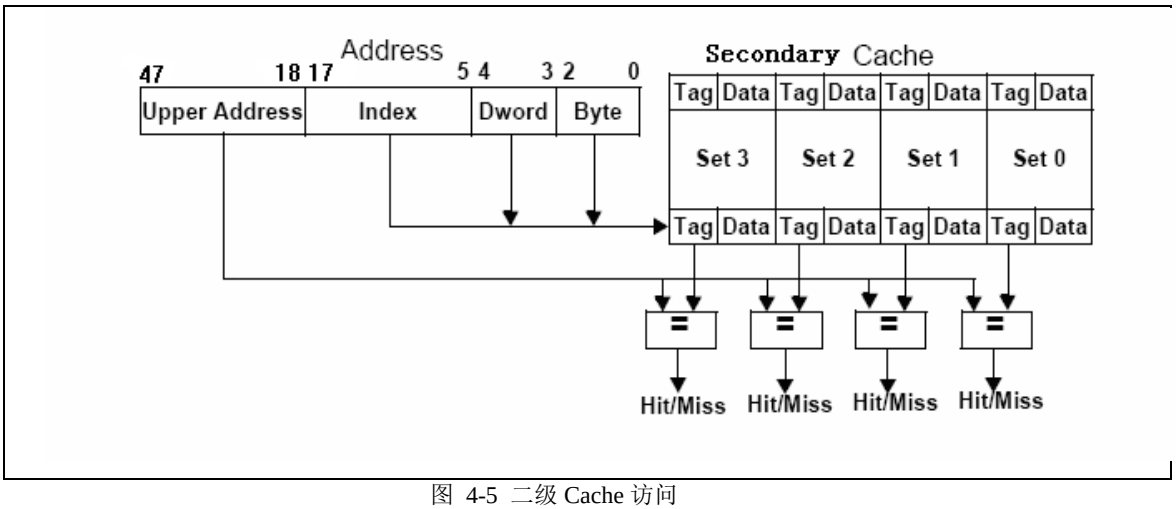


图 4-5 二级 Cache 访问

4.5 Cache 算法和 Cache 一致性属性

龙芯 3 号实现表 4-2 所示的 Cache 算法和 Cache 一致性属性。

表 4-2 龙芯 3 号 Cache 的一致性属性

属性分类	一致性代码
保留	0
保留	1
非高速缓存（Uncached）	2
一致性高速缓存（Cacheable coherent）	3
保留	4
保留	5
保留	6
非高速缓存加速（Uncached Accelerated）	7

4.5.1 非高速缓存(Uncached，一致性代码 2)

如果某个页采用非高速缓存算法时，那么对于在该页中任何位置的 Load 或 Store 操作，处理器都直接发射一个双字，部分双字，字，部分字的读或写请求给主存，而不通过任何一级 Cache。非高速缓存算法采用阻塞的方式实现。

4.5.2 一致性高速缓存(Cacheable coherent，一致性代码 3)

一个具有该属性的行可以驻留在 Cache 中，相应的存数和取数操作都只访问一级 Cache。当一级 Cache 失效时，处理器会检查二级 Cache，看是否有包含所请求的地址。如果二级 Cache 命中，则从二级 Cache 中填充数据。如果二级 Cache 不命中，则从主存中取出数据，并将其写入二级 Cache 和一级 Cache。

由于龙芯 3 号中存在多个处理器核及 IO 设备可以访问主存，因此龙芯 3 号硬件实现了 Cache 一致性协议，无需通过软件采用 Cache 指令来主动维护 Cache 的一致性。

4.5.3 非高速缓存加速 (Uncached Accelerated，一致性代码 7)

非高速缓存加速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。缓冲区和一个 Cache 行一样大小。把数据存储到缓冲区中就和存储到 Cache 中一样。当缓冲区满的时候，开始进行块写。在顺序存数指令的收集过程中，若有其他类型 Uncached 存数指令插入，则收集工作中止，缓冲区中保存的数据按字节写方式输出。

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。

4.6 Cache 一致性

龙芯 3 号实现了基于目录的 Cache 一致性，硬件保证了一级指令 Cache、一级数据 Cache、二级 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 指令来强制刷 Cache。

龙芯 3 号中每个 Cache 行都有一个固定的宿主二级 Cache 模块。Cache 行的目录信息就在宿主二级 Cache 模块中维护。目录利用 32 位的位向量来记录拥有每个 Cache 行备份的一级 Cache（包括一级指令 Cache 和一级数据 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）。三个状态的转移情况如图 4-6。当读指令或者取指发生一级 Cache 失效时，处理器核向二级 Cache 模块发出 Reqread 请求，在得到二级 Cache 模块送回的 Repread 应答后，处理器核的一级 Cache 获得了一个 SHD 状态的 Cache 备份；当写指令发生一级 Cache 失效时，处理器核向二级 Cache 模块发出 Reqwrite 请求，在得到二级 Cache 模块送回的 Repwrite 应答后，处理器核的一级 Cache 获得了一个 EXC 状态的 Cache 备份；当处理器核发生一级 Cache 替换时，通过 Reqreplace 写回二级 Cache 模块，二级 Cache 模块通过 Repreplace 应答告知处理器核替换请求已被处理。二级 Cache 模块可以通过发送 Reqinv 请求至处理器核来无效一个 SHD 状态的一级 Cache 备份，处理器核将该一级 Cache 备份变为 INV 状态并通过 Repinv 应答二级 Cache 模块；二级 Cache 模块可以通过发送 Reqwtbk 请求至处理器核来写回一个 EXC 状态的一级 Cache 备份，处理器

核将该一级 Cache 备份变为 EXC 状态并通过 Repwtbk 应答二级 Cache 模块；二级 Cache 模块可以通过发送 Reqinvwtbk 请求至处理器核来写回并无效一个 EXC 状态的一级 Cache 备份，处理器核将该一级 Cache 备份变为 INV 状态并通过 Repinvwtbk 应答二级 Cache 模块。

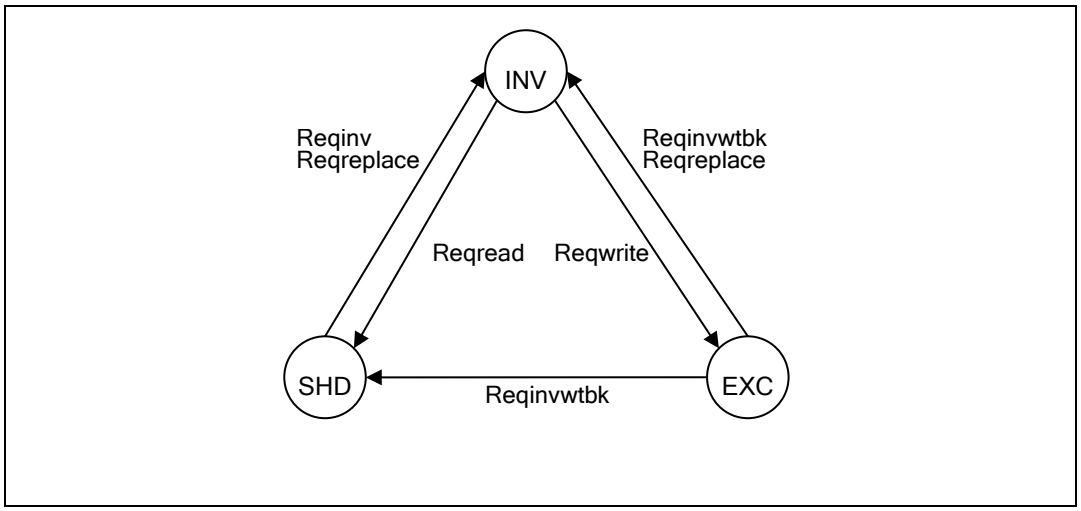


图 4-6 龙芯 3 号 cache 状态转换

5 内存管理

龙芯 GS464 处理器核提供了一个功能完备的内存管理单元（MMU），它利用片上的 TLB 实现虚拟地址到物理地址的转换。

本章节描述了处理器核的虚拟地址和物理地址空间，虚拟地址到物理地址的转换，TLB 在实现这些转换时的操作，Cache，以及为 TLB 提供软件接口的系统控制协处理器（CP0）寄存器。

5.1 快速查找表 TLB

把虚拟地址映射成物理地址通常是由 TLB 来实现的（也有不经过 TLB 的虚拟地址转换，比如 CKSEG0 和 CKSEG1 内核地址空间段（见图 5-5）就是不进行页面映射的，其中的物理地址是由虚拟地址减去一个基址得到的。）。第一级的 TLB 是 JTLB，同时也作为数据 TLB，另外，龙芯 GS464 处理器核包含独立的指令 TLB 以缓解对 JTLB 的竞争。

5.1.1 JTLB

为了能够快速地进行虚拟地址到物理地址的映射，龙芯 GS464 处理器核采用了较大的，全相联映射机制的 TLB，JTLB 用于指令和数据的地址映射，用它们的进程号和虚拟地址进行索引。

JTLB 按奇/偶表项成对组织，把虚拟地址空间和地址空间标识符映射到 256T 的物理地址空间。在默认的情况下，JTLB 有 64 对奇/偶表项，允许 128 页进行映射。

有两个机制分别用来协助控制映射空间的大小和内存不同区域的替换策略。

第一，页的大小可以是 4KB 到 16MB，但必须是按 4 倍递增。CP0 寄存器 PageMask 用于记录映射的页的大小，并且这个记录在写一个新的表项的同时载入 TLB 中。龙芯 GS464 处理器核可以在同一运行时刻支持不同大小的页，允许操作系统产生特定目的的映射：例如，视频编解码处理中帧缓冲区就可以只用一个表项来进行内存映射。

第二，龙芯 GS464 处理器核在 TLB 缺失的时候可以采用随机替换的策略来选择要被替换的 TLB 表项。操作系统也可以把一定数量的页面驻留在 TLB 中，而不致于被随机替换出去，这种机制有利于使操作系统提高性能，避免死锁。这种机制也使实时系统比较方便地为某一关键软件提供特定入口。

此外对每个页来说，JTLB 还维护该页面的 Cache 一致性属性，每个页都有特定的位来标记：不经过 Cache（Uncached），非一致性 Cache（Cacheable Noncoherent），或者是非 Cache 加速（Uncached Accelerated）。

5.1.2 指令 TLB

龙芯 GS464 处理器核的指令 TLB（ITLB）有 16 个表项，它最小化了 JTLB 的容量，并通过一个大的相联阵列缩短了映射时的时间关键路径，降低了功率。每个 ITLB 表项只能映射一页，页面大小由 PageMask 寄存器来指定。ITLB 指令地址的映射和数据地址的映射能并行执行，从而提高了性能。当 ITLB 中的表项失效时，从 JTLB 中查找相应的表项，随机选择一个 ITLB 表项进行替换，ITLB 的操作对用户是完全透明的。处理器保证 ITLB 与 JTLB 的一致，当使用核心态指令修改 JTLB 时，ITLB

将被自动清空。

5.1.3 命中和失效

如果虚拟地址与 TLB 中某个表项的虚拟地址一致（即 TLB 命中），则物理页面号就从 TLB 中取出，并和偏移连接组成物理地址。

如果虚拟地址与 TLB 中任何表项的虚拟地址都不一致（即 TLB 失效），则 CPU 产生一个异常，并由软件根据内存中存放的页表重新填写 TLB。软件既可以重写指定的 TLB 表项，也可以使用硬件提供的机制重写任意一个 TLB 表项。

5.1.4 多项命中

龙芯 GS464 处理器核对 TLB 中虚地址不只与一个表项的虚地址一致的情况没有提供任何的探测和禁用机制，这一点不像早期的 MIPS 处理器的设计。多项命中并不会物理地破坏 TLB，因此多项命中的探测机制是不必要的。但是多项命中的情况并没有被定义，因此软件要控制不要让多项命中的情况发生。

5.2 处理器模式

龙芯 GS464 处理器核有 3 种工作模式，但是与其它 MIPS 处理器不同，龙芯 GS464 处理器核只支持一种地址模式，一种指令集模式和一种尾端模式。

5.2.1 处理器工作模式

以下三种模式的处理器优先级依次降低：

- **内核模式**（最高的系统优先级）：在这种模式下处理器可以访问和改变任何寄存器，操作系统最内层的内核运行在内核模式；
- **管理模式**：处理器的优先级降低，操作系统的一些不太关键的部分运行在该模式；
- **用户模式**（最低的系统优先级）：该模式下使不同的用户间不致互相干扰。

三种模式的切换是由操作系统（在内核模式）置位状态寄存器 KSU 域的相应位来实现的。当出现一个错误（ERL 位置位）或出现一个例外（EXL 位置位）时，处理器被强制切换到内核模式。表 5-1 列出了三种模式切换时 KSU、EXL 和 ERL 的置位情况，空的表项可以不必关心。

表 5-1 处理器的工作模式

KSU 4:3	ERL 2	EXL 1	描述
10	0	0	用户模式
01	0	0	管理模式
00	0	0	内核模式
	0	1	例外级别
	1		错误级别

5.2.2 地址模式

龙芯 GS464 处理器核只支持 64 位的虚拟地址模式，并且硬件保证兼容 32 位的地址模式。

5.2.3 指令集模式

龙芯 GS464 处理器核实现了完备的 MIPS64R2 指令集， 另外还增加了一些整型和浮点指令，增加的指令见附件 A 和附录 B。

5.2.4 尾端模式

龙芯 GS464 处理器核只工作在小尾端模式。

5.3 地址空间

本节叙述的是虚拟地址空间，物理地址空间，和经过 TLB 进行虚实地址转换的方法。

5.3.1 虚拟地址空间

龙芯 GS464 处理器核有三个虚拟地址空间：用户地址空间、管理地址空间和内核地址空间，每个空间都是 64 位的，并且包含一些不连续的地址空间段，最大的段为 256T(2^{48})字节。

5.3.4 节到 5.3.6 节分别描述了这三种地址空间。

5.3.2 物理地址空间

通过使用 48 位地址，处理器的物理地址空间大小为 256T(2^{48})字节。以下小节将详述虚实地址转换的方法。

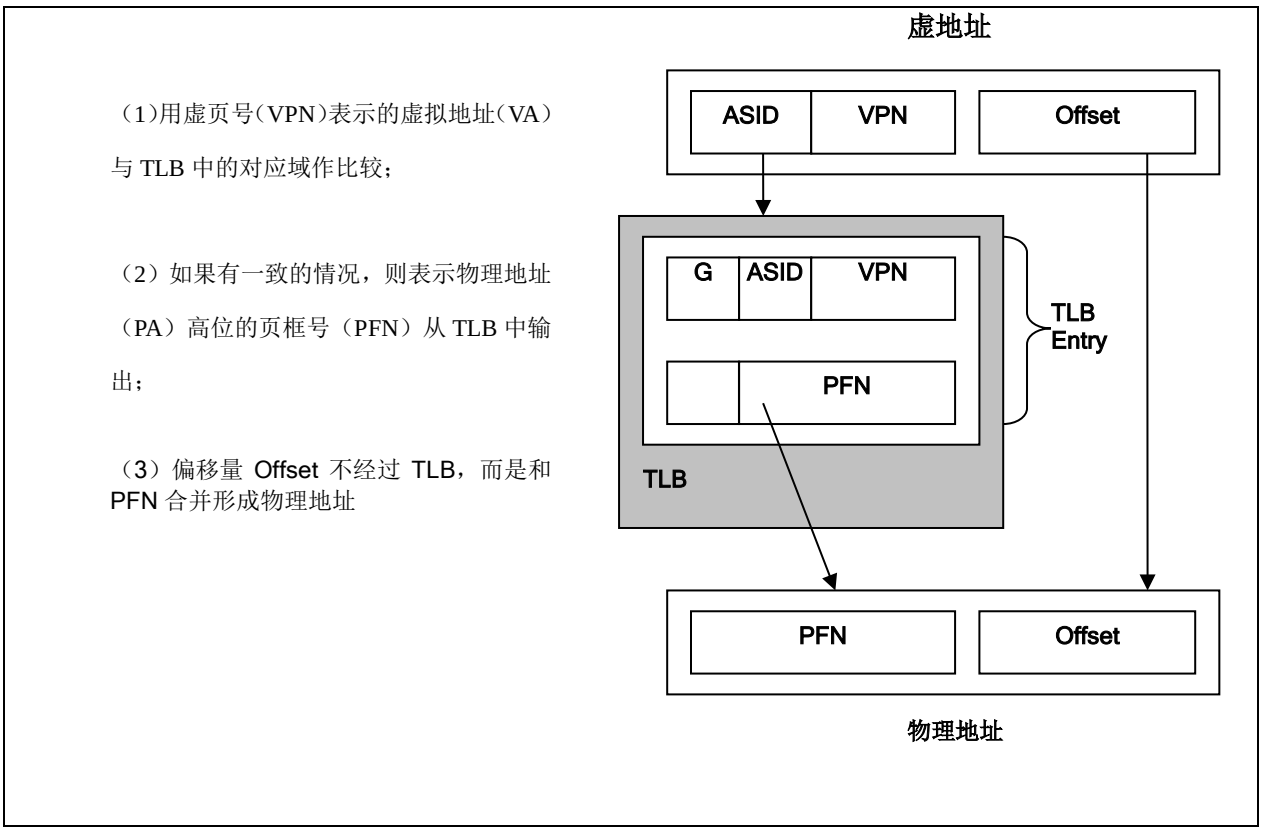
5.3.3 虚实地址转换

进行虚实地址转换时，首先比较处理器给出的虚拟地址和 TLB 中存放的虚拟地址。当虚页号（VPN）等于某个 TLB 表项的 VPN 域，并且如果下面两种情况中的任何一种成立：

- TLB 表项的 Global 位为 1
- 两个虚拟地址的 ASID 域一样。

TLB 就命中了。如果不满足以上条件，那么 CPU 会产生 TLB 失效异常，以使软件能够根据内存中存放的页表重新填写 TLB。

如果 TLB 命中了，则物理页号将从 TLB 中取出，并与页内偏移量 Offset 合并，形成物理地址。页内偏移量 Offset 在虚实地址转换的过程中不经过 TLB。



（1）用虚页号（VPN）表示的虚拟地址（VA）与 TLB 中的对应域作比较；

（2）如果有一致的情况，则表示物理地址（PA）高位的页框号（PFN）从 TLB 中输出；

（3）偏移量 Offset 不经过 TLB，而是和 PFN 合并形成物理地址

图 5-1 虚实地址转换概览

图 5-1 所示为虚实地址转换，虚拟地址被一个 8 位的地址空间标识符（ASID）扩展了，该措施降低了上下文切换时进行 TLB 刷新的频率。ASID 存放在 CP0 EntryHi 寄存器中。Global 位（G）在相应的 TLB 表项中。

图 5-2 显示了 64 位模式的虚实地址转换过程，这个图显示了最大页面 16MB 和最小页面 4KB 的情况。

图的上半部分显示了页面大小为 4K 字节的情况，页内偏移量 Offset 占用虚拟地址中的 12 位，虚拟地址中剩下的 36 位为虚页号 VPN，用于索引 64G 个页表表项；

图的下半部分显示了页面大小为 16M 字节的情况，页内偏移量 Offset 占用虚拟地址中的 24 位，虚拟地址中剩下的 24 位为虚页号 VPN，用于索引 16M 个页表表项。

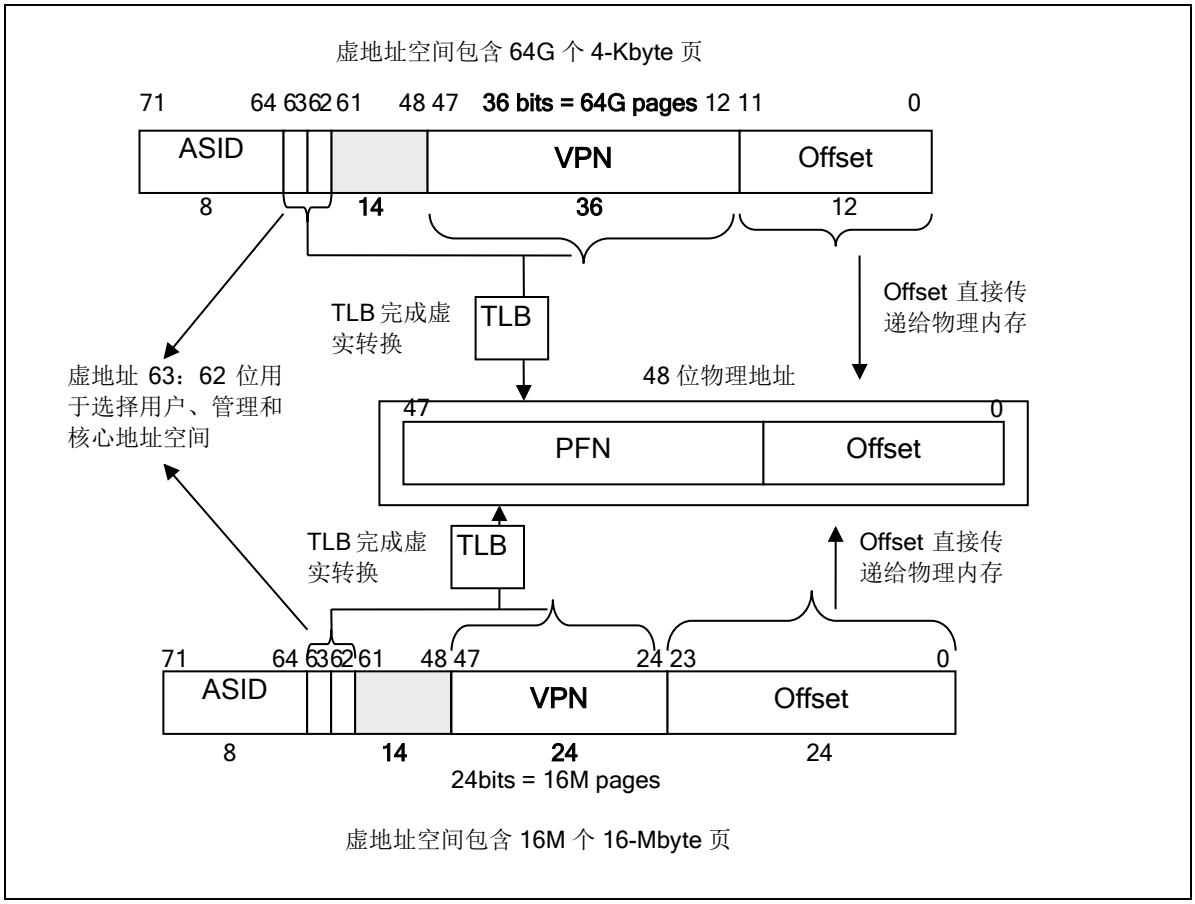


图 5-2 64 位模式虚拟地址转换

5.3.4 用户地址空间

在用户模式下，只有一个称为用户段（User Segment）的单独、统一的虚拟地址空间，其大小为 256T（2^{48}）字节，名字为 XUSEG。

图 5-3 显示了用户虚拟地址空间，可以在用户模式、管理模式、内核模式下访问。

用户段从地址 0 处开始，当前活动的用户进程驻留在该段（XUSEG）中。在不同模式下，TLB 对 XUSEG 段的映射处理方式都一样，并控制是否可以访问 Cache。

当处理器的 Status 寄存器的值同时满足三个条件：KSU=10_2、EXL=0、ERL=0 时，处理器工作在用户模式下。

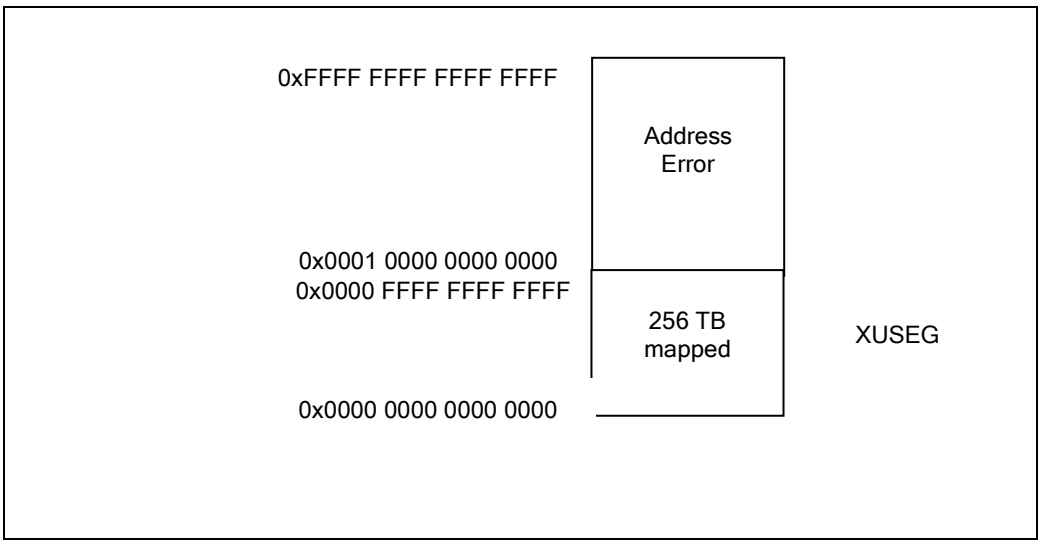


图 5-3 用户模式下用户虚拟地址空间概况

所有可用的用户模式下虚拟地址的第 63 位到第 48 位必须都为 0，访问任何一个第 63 位到第 40 位不全为 0 的地址都将导致地址错误异常，在 XUSEG 地址段的 TLB 缺失使用 XTLB 重填向量。龙芯 GS464 处理器核的 XTLB 重填向量与 32 位模式下 TLB 的重填向量有相同的例外入口地址。

5.3.5 管理地址空间

管理模式是为分层结构的操作系统设计的。在分层结构的操作系统中，真正的内核运行在内核模式下，操作系统的其余部分运行在管理模式下。管理地址空间提供了管理模式下程序访问的代码和数据空间。管理地址空间的 TLB 缺失由 XTLB 重填处理器来处理。

管理模式和内核模式都可访问管理地址空间。

当处理器的 Status 寄存器的值同时满足三个条件：KSU=01_2、EXL=0、ERL=0 时，处理器工作在管理模式下。图 5-4 显示了管理模式下的用户和管理地址空间概况。

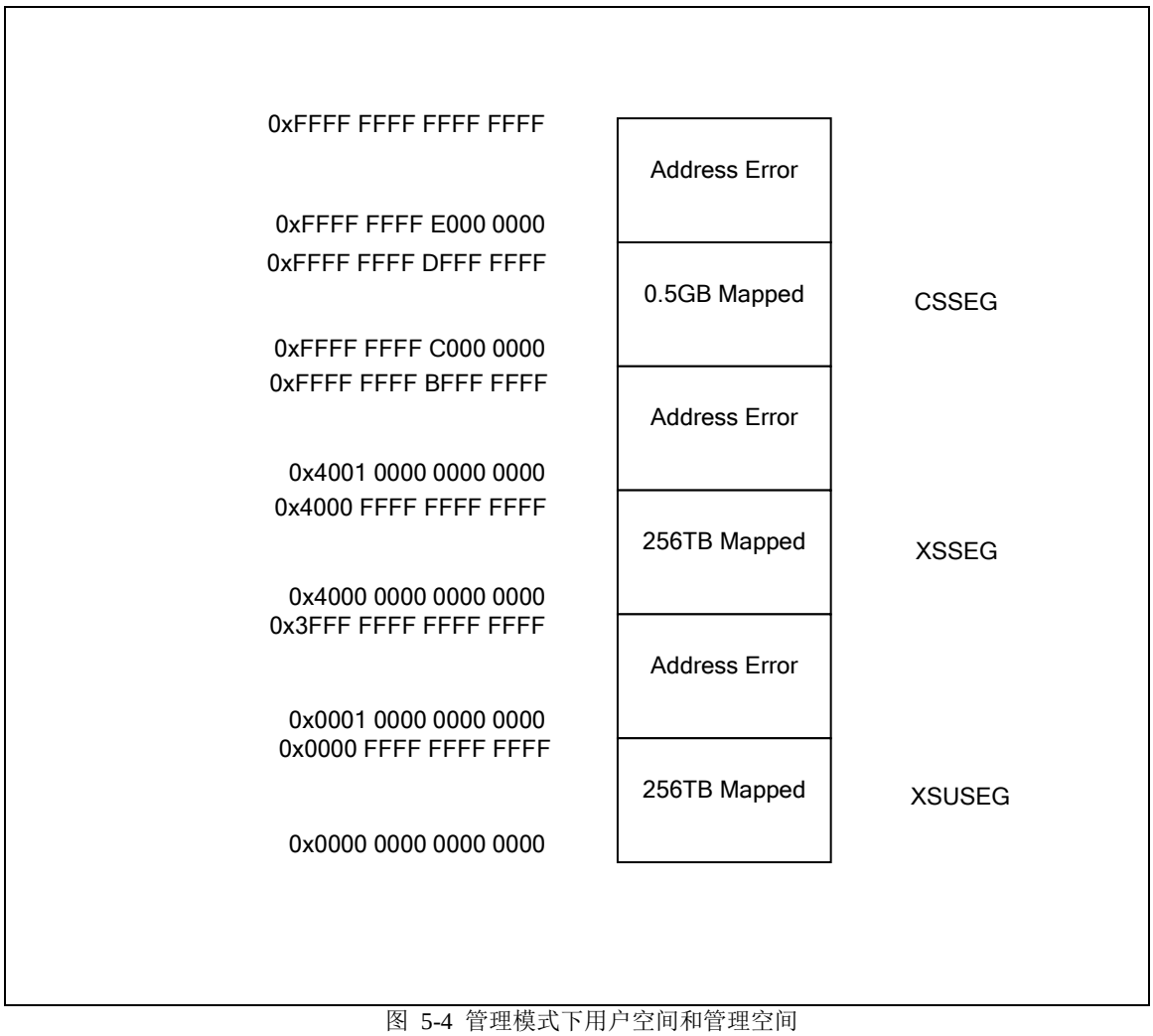


图 5-4 管理模式下用户空间和管理空间

- 64 位管理模式，用户地址空间（XSUSEG）

在管理模式下，当访问用户地址空间并且 64 位地址的最高两位（第 63 和第 62 位）为 00_2 时，程序使用一个名字为 XSUSEG 的虚拟地址空间，XSUSEG 覆盖了当前用户地址空间的全部 2^{48}（1T）字节。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0x0000 0000 0000 0000 开始，到 0x0000 FFFF FFFF FFFF 结束。

- 64 位管理模式，当前管理地址空间（XSSEG）

在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 01_2 时，程序使用一个名字为 XSSEG 的当前管理虚拟地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0x4000 0000 0000 0000 开始，到 0x4000 FFFF FFFF FFFF 结束。

- 64 位管理模式，独立管理地址空间（CSSEG）

在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 11_2 时，程序使用一个名字为 CSSEG 的独立管理虚拟地址空间。在 CSSEG 中的寻址与 32 位模式下在 SSEG 中的寻址是兼容的。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0xFFFF FFFF C000 0000 开始，到 0xFFFF FFFF DFFF FFFF 结束。

5.3.6 内核地址空间

当处理器的 Status 寄存器的值满足下述条件：KSU=00_2 或 EXL=1 或 ERL=1 时，处理器工作在内核模式下。

每当处理器检测到一个例外时便进入内核模式，并一直保持到执行例外返回指令（ERET）。ERET 指令将处理器恢复到例外发生前所在的模式。

根据虚拟地址高位的不同，内核模式虚拟地址空间被分为不同的区域，如图 5-5 所示。

- 64 位内核模式，用户地址空间（XKUSEG）

在内核模式下，当访问用户空间并且 64 位虚拟地址的最高两位为 00_2 时，程序使用一个名字为 XKUSEG 的虚拟地址空间，XKUSEG 覆盖了当前用户地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式，当前管理地址空间（XKSSEG）

在内核模式下，当访问管理空间并且 64 位地址的最高两位为 01_2 时，程序使用一个名字为 XKSSEG 的虚拟地址空间，XKSSEG 是当前管理虚拟地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式，物理地址空间（XKPHY）

在内核模式下，当 64 位地址的最高两位为 10_2 时，程序使用一个名字为 XKPHY 的虚拟地址空间，XKPHY 是八个 2^{48} 字节的内核物理地址空间的集合。访问任何地址第 58 到第 48 位不为 0 的存储单元都将引起地址错误。对 XKPHY 的访问不经过 TLB 进行地址变换，而是将虚拟地址的第 47 到第 0 位作为物理地址。虚拟地址的第 61 到第 59 位控制是否通过 Cache 和 Cache 的一致性属性，与表 3-2 描述的 TLB 页的 C 位值含义相同。

- 64 位内核模式，内核地址空间（XKSEG）

在内核模式下，当 64 位地址的最高两位为 11_2 时，程序使用以下两个地址空间之一：

  - 内核虚拟地址空间 XKSEG，此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址；
  - 四个 32 位内核兼容地址空间，下一小节详述。

- 64 位内核模式，兼容地址空间（CKSEG1：0，CKSSEG，CKSEG3）

在内核模式下，64 位地址的最高两位为 11_2，并且虚拟地址的第 61 到第 31 位所有位都等于 1 时，程序使用的以下四个 512M 字节地址空间中的一个，具体哪一个根据第 30、29 位决定：

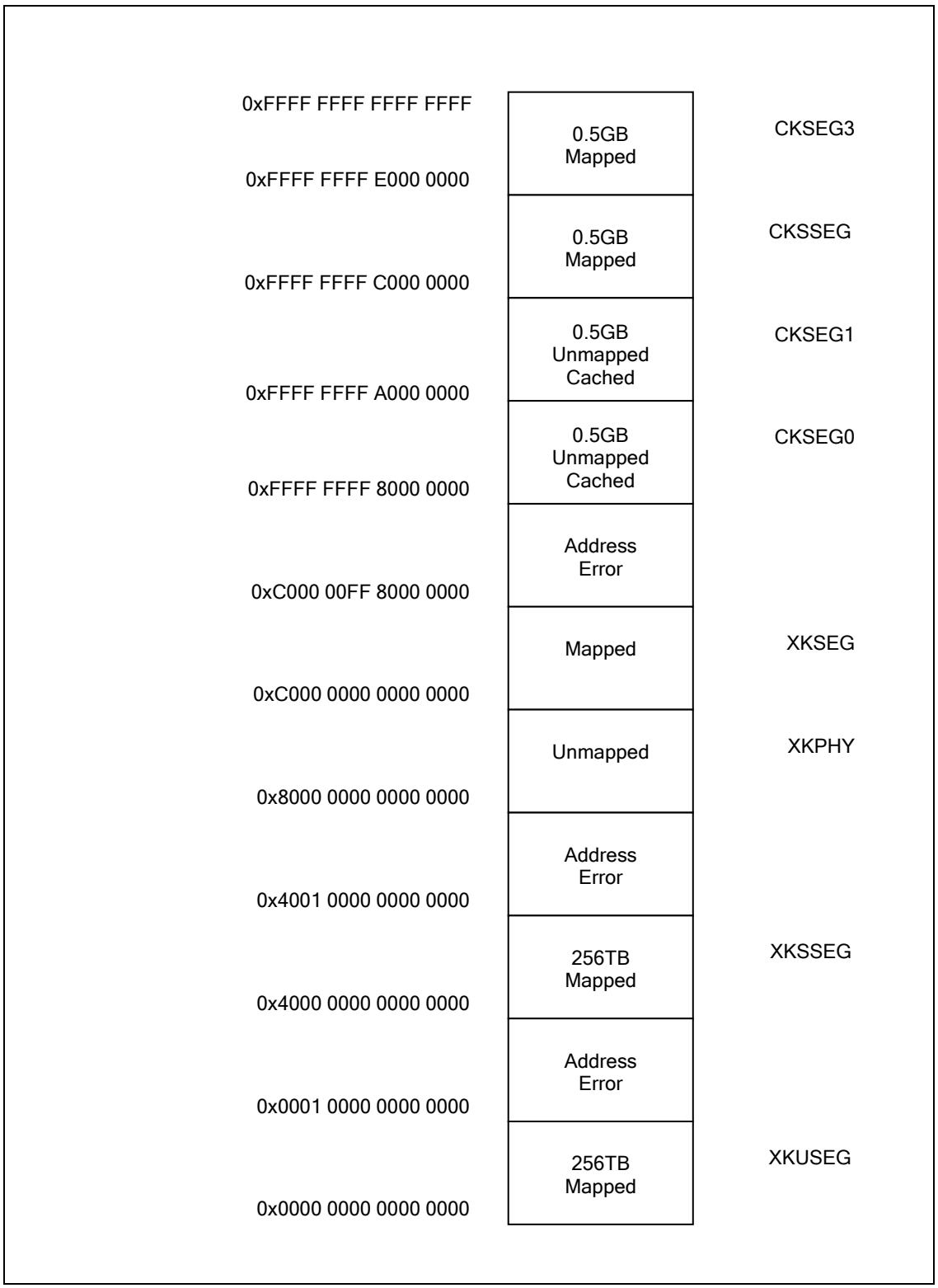


图 5-5 内核模式下的用户、管理、内核地址空间概况

■ CKSEG0：该 64 位虚拟地址空间不经过 TLB，与 32 位模式下的 KSEG0 兼容。Config 寄存器的 K0 域控制是否通过 Cache 和 Cache 的一致性属性，

■ CKSEG1：该 64 位虚拟地址空间不经过 TLB 也不经过 Cache，与 32 位模式下的 KSEG1 兼容。

■ CKSSEG：该 64 位虚拟地址空间为当前管理虚拟地址空间，与 32 位模式下的 KSSEG

兼容。

- CKSEG3：该 64 位虚拟地址空间为内核虚拟地址空间，与 32 位模式下的 KSEG3 兼容。

5.4 系统控制协处理器

系统控制协处理器(CP0)负责支持存储管理，虚实地址转换，例外处理，以及一些特权操作。龙芯 GS464 处理器核有 26 个 CP0 寄存器和一个 64 项的 TLB，每个寄存器都有唯一的寄存器号。下面的章节将给出与内存管理相关的寄存器的概述。

5.4.1 TLB 表项的格式

图 5-6 表示 TLB 表项的格式，项中的每个域在 EntryHi，EntryLo0，EntryLo1，PageMask 寄存器中都有相应的域。

EntryHi，EntryLo0，EntryLo1，以及 PageMask 寄存器和 TLB 项的格式类似。唯一的不同就是 TLB 项有一个 Global 域（G 位），EntryHi 寄存器中没有，而作为保留域出现。图 5-7、图 5-8 和图 5-9 分别表示了在图 5-6 TLB 项的各个域。

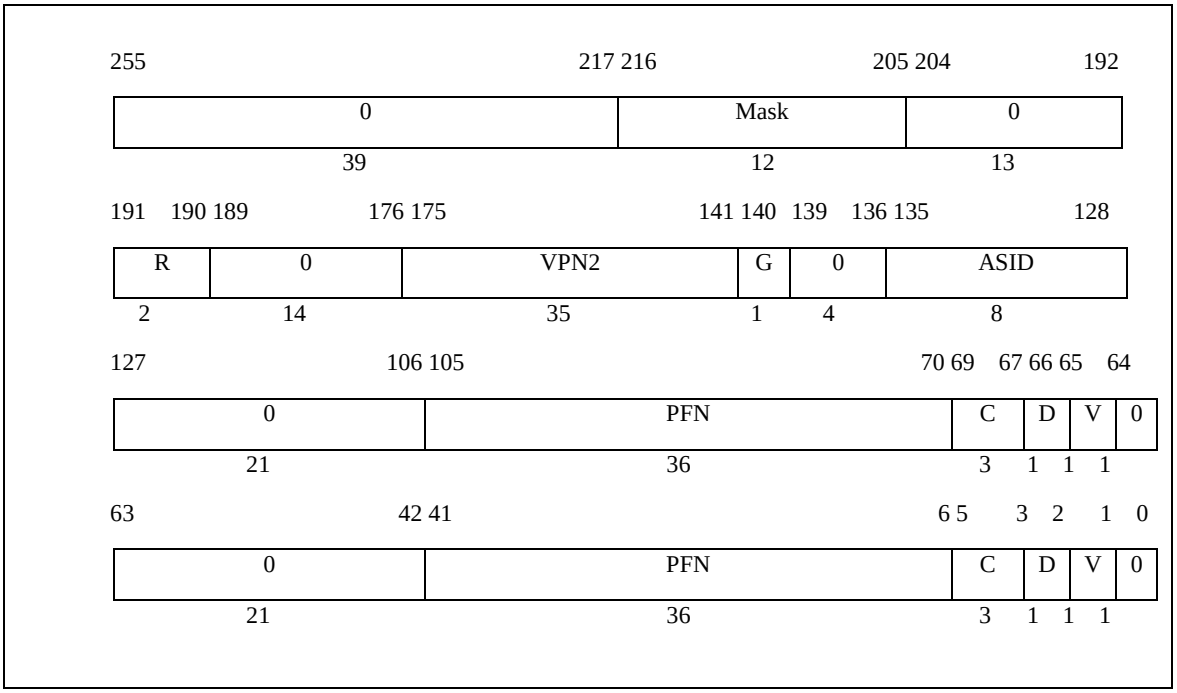


图 5-6 TLB 表项

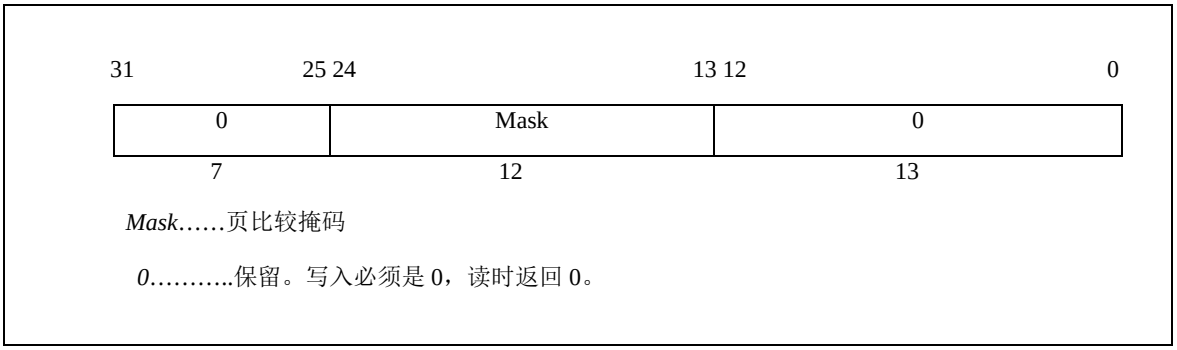
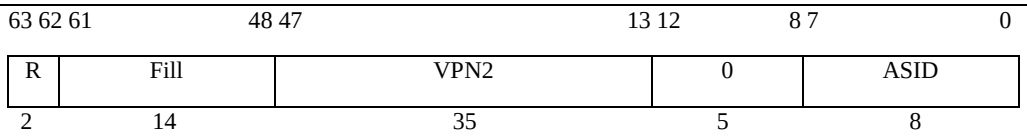


图 5-7 PageMask 寄存器



VPN2….虚页号除 2（映射 2 个页）。

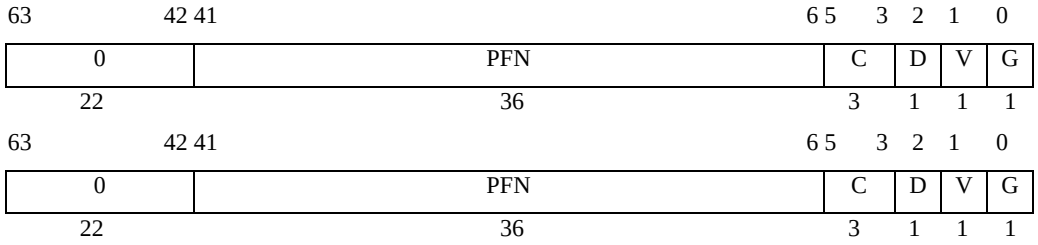
ASID….地址空间 ID 域。一个 8 位域，用于让多个进程共享 TLB；对于与其它进程同样的虚页号，每个进程有不同的映射。

R……区域。（00->用户，01->管理，11->核心）用户匹配虚地址 63:62 位。

Fill…….保留。读为 0，写忽略。

0……保留。写入必须是 0，读时返回 0。

图 5-8 EntryHi 寄存器



PFN…页框号；物理地址的高位。

C…….指定 TLB 页的一致性属性；见表 3-2。

D…….脏位。如该位值设置为 1，则对应的页标记为脏，因而可写。该位实际上是写保护位，软件可用此位保护数据的改动。

V…….有效位。该位设置表示 TLB 表项是有效的；否则，将产生 TLBL/TLBS 失效。

G…….全局位。如果 Lo0 和 Lo1 中对应的位都设置为 1，则在 TLB 查找时处理器忽略 ASID。

0……..保留。写入必须是 0，读时返回 0。

图 5-9 EntryLo0 和 EntryLo1 寄存器

TLB 页一致性属性位（C）指定访问该页时是否需要通过 Cache，如果通过 Cache，则需要选择 Cache 的一致性属性。表 5-2 表示 C 位对应的 Cache 一致性属性。

表 5-2 TLB 页的 C 位的值

C(5:3) 值	Cache一致性属性
0	保留
1	保留
2	非高速缓存（Uncached）
3	非一致性高速缓存（Cacheable Noncoherent）
4	保留
5	保留
6	保留
7	非高速缓存加速（Uncached Accelerated）

5.4.2 CP0 寄存器

表 5-3 列出了与内存管理相关的 CP0 寄存器，第 1 章对 CP0 寄存器进行了完备的描述。

表 5-3 内存管理相关的 CP0 寄存器

寄存器号	寄存器名
0	Index
1	Random
2	EntryLo0
3	EntryLo1
5	PageMask
6	Wired
10	EntryHi
15	PRID
16	Config
17	LLAddr
28	TagLo
29	TagHi

5.4.3 虚拟地址到物理地址的转换过程

在虚地址到物理地址转换时，CPU 将虚地址的 8 位 ASID（如果全局位 G 没有设置）和 TLB 项的 ASID 进行比较，看是否匹配。在比较 ASID 的同时还需要根据页掩码（PageMask）的值将虚地址的高 15~27 位和 TLB 项的虚页号进行匹配比较。如果有 TLB 项匹配，从匹配的 TLB 项中取出物理地址和访问控制位（C，D 和 V）。对一个有效的地址转换来说，匹配的 TLB 项的 V 位必须设置，但是在匹配比较时不考虑 V 位的值。图 5-10 表示了 TLB 地址转换过程。

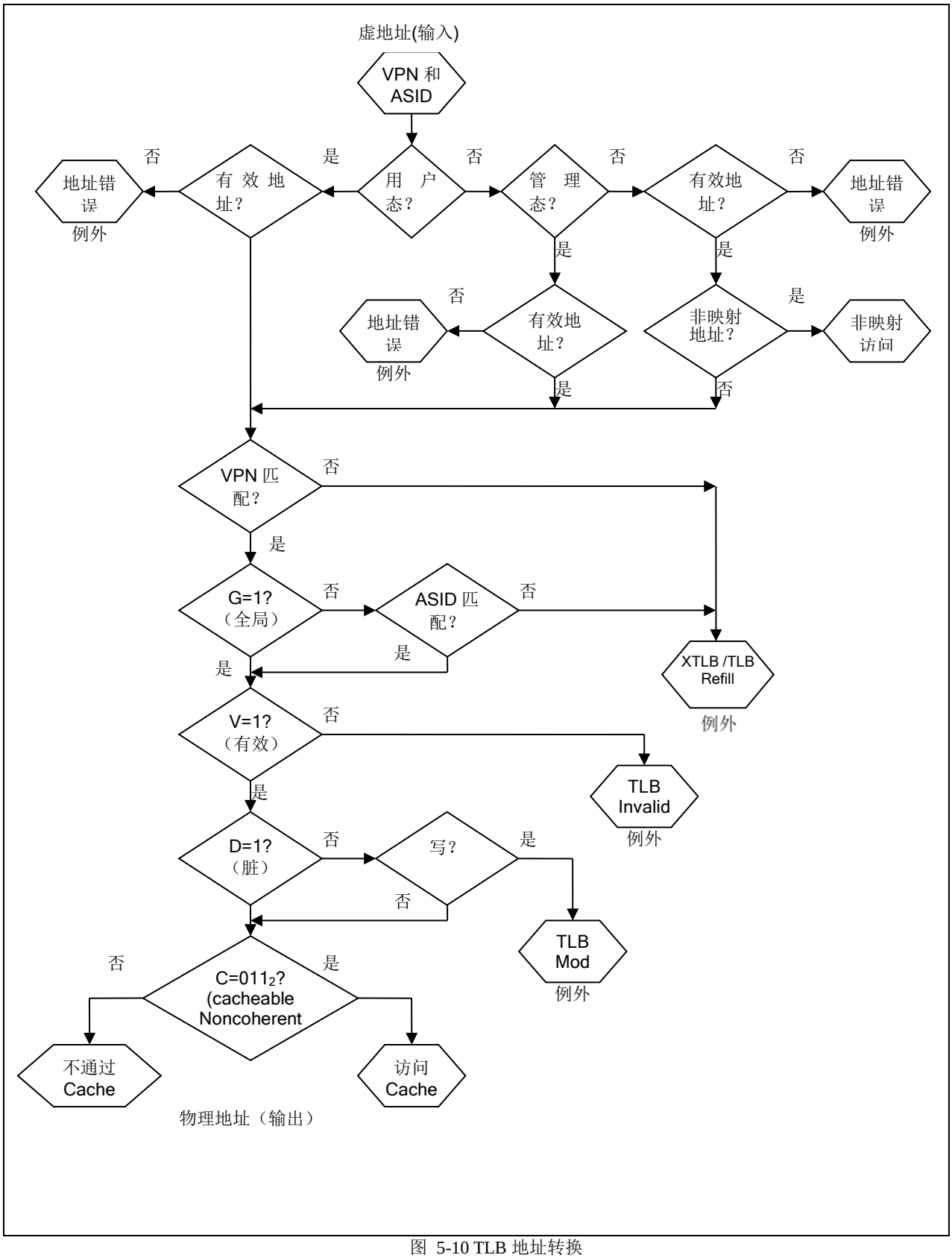


图 5-10 TLB 地址转换

5.4.4 TLB 失效

如果没有任何一 TLB 项匹配虚地址，引发一个 TLB 不命中例外。如果访问控制位(D 和 V)指示访问不是合法的，引发一个 TLB 修改或者 TLB 无效例外。如果 C 位等于 011_2，被检索到的物理地址

通过 Cache 访问内存，否则不通过 Cache。

5.4.5 TLB 指令

表 5-4 列出了所有的 CPU 所提供的用于和 TLB 操作有关的指令。

表 5-4 TLB 指令

操作码	指令描述
TLBP	在 TLB 中搜索匹配项
TLBR	读索引的 TLB 项
TLBWI	写索引的 TLB 项
TLBWR	写随机的 TLB 项

5.4.6 代码例子

第一个例子是如何配置 TLB 表项来映射一对 4KB 的页面。实时系统的内核大多都这么做，这种简单的内核 MMU 只用于进行内存保护，所以静态映射就足够了，在所有静态映射的系统中所有 TLB 例外都被作为是错误条件（不可访问）。

```
1.  mtc0 r0,C0_WIRED                # make all entries available to random replacement
2.  li r2, (vpn2<<13)|(asid & 0xff);
3.  mtc0 r2, C0_ENHI                # set the virtual address
4.  li r2, (epfn<<6)|(coherency<<3)|(Dirty<<2)|Valid<<1|Global)
5.  mtc0 r2, C0_ENLO0               # set the physical address for the even page
6.  li r2, (opfn<<6)|(coherency<<3)|(Dirty<<2)|Valid<<1|Global)
7.  mtc0 r2, C0_ENLO1               # set the physical address for the odd page
8.  li r2, 0                        # set the page size to 4KB
9.  mtc0 r2,C0_PAGEMASK
10. li r2, index_of_some_entry      # needed for tlbwi only
11. mtc0 r2, C0_INDEX               # needed for tlbwi only
    tlbwr                           # or tlbwi
```

一个完备的虚拟存储操作系统（如 UNIX），用 MMU 进行内存保护，并进行主存和大容量存储设备的换页。这个机制使程序可以访问更大的存储设备而不仅仅局限于系统物理分配的空间。这个依赖于请求调页的机制需要动态页面映射。动态映射通过一系列不同类型的 MMU 例外实施，TLB 重填是这种系统中最常见的例外。下面是一个可能的 TLB 重填例外控制。

```
12. refill_exception:
13.     mfc0 k0,C0_CONTEXT
```

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

这个例外控制处理非常简单，因为它的经常执行会影响系统性能，这就是 TLB 重填例外分配独立的例外向量的原因。这段代码假设需要的映射在主存储器的页表中已经建立起来了。如果没有建立起来，那么在 ERET 指令后将发生 TLB 失效例外。TLB 失效例外很少发生，这是有益的，因为它必须计算期望的映射，并可能需要从后援存储器中读取部分页表。TLB 修改例外用于实现只读页面和标记进程清除代码需要修改的页。为了保护不同的进程和用户不受相互的干扰，虚拟存储操作系统通常在用户模式执行用户程序。下面的例子表示如何从内核模式进入用户模式。

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

5.5 物理地址空间分布

龙芯 3 号的地址空间按照地址的高位均匀分布到各个结点。48 位地址的高 4 位[47:44]对应地址空间所在的结点编号，每个结点拥有固定的 44 位地址空间。而在结点内 44 位的地址空间又进一步划分为 8 个 41 位的地址空间，采用 41 位的空间主要是由于一个端口可能会接两个 HT 控制器，而每个 HT 需要 40 位的地址空间。

6 处理器例外

本章介绍龙芯 GS464 处理器核例外，内容包括：例外的产生及返回，例外向量位置和所支持的例外类型。其中对每一类支持的例外类型，介绍内容包括例外的原因，处理和服务。

6.1 例外的产生及返回

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

从例外返回也会将 EXL 位置为 0。

6.2 例外向量位置

冷重置、软重置和非屏蔽中断(NMI)例外的向量地址是专用的重置例外向量地址 0xFFFFFFFFBFC00000，这个地址既不通过 Cache 进行存取，也无需地址映射。此外，EJTAG 调试中断的入口根据其控制寄存器中的 ProbeTrap 位是 0 还是 1 分别选用 0xFFFFFFFFBFC00480 和 0xFFFFFFFFFF200200。所有其它例外向量地址的形式都是基地址加上向量偏移。当状态寄存器中的 BEV 位为 0 时用户可定义例外向量的基址，详见表 6-1 例外向量基址。

表 6-1 例外向量基址

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

表 6-2 列出了龙芯 GS464 处理器核中例外向量的偏移。

表 6-2 例外向量偏移

例外	例外向量偏移
TLB Refill, EXL=0	0x000
XTLB Refill, EXL=0	0x080
Cache error	0x100
其它共用例外	0x180
中断且 CauseIV=1	0x200
Reset, Soft Reset, NMI	无（使用基地址）

对于外部中断(包括时钟和性能计数器中断)，传统做法是使用共用例外入口，由软件负责分发到相应的服务。龙芯 GS464 处理器核支持向量中断模式(Vectored Interrupt)，该模式由 Cause 寄存器的 IV 位选择。在向量中断模式下，中断优先级从 IP7 到 IP0 依次降低并且有专门的例外入口。IntCtl 寄存器的 VS 域控制这些例外处理代码所占用的空间大小，每个中断对应的入口偏移可用下式计算(其中向量号从零开始)：

向量中断偏移 = 0x200 + 向量号 * IntCtlVS

6.3 例外优先级

这一章的剩余部分将按照表 6-3 中给出的优先顺序依次介绍各个例外(对某些特定例外，如 TLB 例外和指令/数据例外，为了方便而放在一起介绍)。当一条指令同时产生一个以上例外时，只向处理器报告其中优先级最高的例外。有些例外并不是由当时正在执行的指令产生的，而有些例外可能被推迟处理。更多细节请查看本章对各个例外的单独介绍。

表 6-3 例外优先顺序

例外优先顺序
冷重置(最高优先级)
不可屏蔽中断 (NMI)
地址错误 — 取指
TLB 重填 — 取指
TLB 无效 — 取指
Cache 错 — 取指

总线错误 —取指
整型溢出，陷阱，系统调用，断点，保留指令，协处理器不可用，浮点例外
EJTAG 中断
地址错误 — 数据存取
TLB 重填 — 数据存取
TLB 无效 — 数据存取
TLB 修改 — 写数据
EJTAG 数据断点
Cache 错 — 数据存取
总线错误 —数据存取
中断(最低优先级)

一般来说，下面各节中介绍的例外先由由硬件来处理，然后由软件来服务。

6.4 冷重置例外

原因

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。

处理

CPU 为这个例外提供了一个特殊的中断向量：

- 32 位模式下位于 0xBFC0 0000
- 64 位模式下位于 0xFFFF FFFF BFC0 0000

冷重置向量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。

当例外发生时，CPU 中所有寄存器内容是不确定的，但下列寄存器域除外：

- 状态（Status）寄存器的初值为 0x30c000e4，SR 位被清为 0，ERL 位和 BEV 位被置为 1。
- 配置（Config）寄存器的初值为 0x80034482。
- 随机（Random）寄存器初始化为它的最大值。
- Wired 寄存器初始化为 0。
- 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。

服务

冷重置例外服务包括：

- 初始化所有的处理器寄存器，协处理器，Cache 和存储系统。
- 执行诊断测试。
- 引导自举操作系统。

6.5 NMI 例外

原因

NMIn 为低产生 NMI 例外。该例外不可屏蔽。

处理

当发生 NMI 例外时，状态寄存器的 SR 位被置为 1 ，用以区分冷重置。

NMI 例外只能在指令的边界被提取。它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。Cause 寄存器内容保持不变，而系统则跳到 NMI 例外处理程序开始处。

NMI 例外保留除下列寄存器外的所有寄存器值：

- 包含 PC 值的 ErrorEPC 寄存器。
- 置为 1 的状态寄存器 ERL 位。
- 软重置或 NMI 置为 1，冷重置置为 0 的状态寄存器 SR 位。
- 置为 1 的状态寄存器 BEV 位。
- PC 寄存器重置为 0xFFFF FFFF BFC0 0000

服务

NMI 例外可以用于除“重置处理器，同时保持 Cache 和内存内容”之外的情形。例如，当检测到电源故障时，系统可以通过 NMI 例外立即、可控地关闭系统。

由于 NMI 例外在另外一个错误例外中发生，因此从例外返回后，通常不太可能继续执行程序。

6.6 地址错误例外

原因

当执行以下情况时，会发生地址错误例外：

- 引用非法地址空间。
- 在用户模式下引用超级用户地址空间。
- 在用户或超级用户模式下引用内核地址空间。
- 取(Load)或存(Store)一个双字，但双字不对齐于双字边界。
- 取(Load，Fetch)或存(Store)一个字，但字不对齐于字的边界。
- 取或存一个半字，但半字不对齐于半字的边界。

该例外不可屏蔽。

处理

共用例外向量用于地址错误例外。Cause 寄存器的 ExcCode 字段值被设为 ADEL 或 ADES 编码值，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

例外发生时，BadVAddr 寄存器保存了没有正确对齐的虚地址，或者受保护的地址空间的虚地址。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

此时，正在运行的导致例外发生的进程会收到 UNIX SIGSEGV(段违例)信号，这个错误对该进程来说通常是致命的。

6.7 TLB 例外

可能发生三种 TLB 例外：

- 当 TLB 中没有项与欲引用的映射地址空间的地址匹配时，会导致 TLB 重填例外。
- 当虚地址引用与 TLB 中某一项匹配，但该项被标示为无效时，TLB 无效例外发生。
- 当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”时

（表示该项不可写），TLB 修改例外发生。

下面三节将介绍这些 TLB 例外。

注：TLB 重填向量选择已经在本章前面作了介绍，具体章节见 6.8“TLB 重填例外”。

6.8 TLB 重填例外

原因

当 TLB 中没有项匹配映射地址空间的引用地址时，TLB 重填例外发生，该例外是不可屏蔽的。

处理

对于这个例外来说，MIPS 体系结构存在两个特殊的例外向量：一个用于 32 位地址空间，另一个用于 64 位地址空间。当引用地址在 32 位地址空间中时例外向量偏移为 0x000，当引用地址在 64 位地址空间中时例外向量偏移为 0x080。

当状态寄存器中的 EXL 位被设为 0 时，所有的地址引用使用这些例外向量。这个例外设置 Cause 寄存器中 ExcCode 字段的值为 TLBL 或 TLBS 编码。这个编码与 EPC 寄存器以及 Cause 寄存器的 BD 一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。Random 寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了导致例外的指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

为了服务这个例外，Contex 或 XContext 寄存器的内容被作为虚地址以取得某些内存位置，这些位置包含了一对 TLB 项的物理页地址和访问控制位。这一对 TLB 项被放入了 EntryLo0/EntryLo1 寄存器；EntryHi 和 EntryLo 寄存器被写入 TLB。

用于获得物理地址和访问控制信息的虚地址有可能位于一个没有驻留在 TLB 中的页面上。如果出现这种情况，则在 TLB 重填处理程序允许另外一个 TLB 重填例外来解决。由于 Status 寄存器的 EXL 位被置为 1，第二个 TLB 重填例外被传递的是共用例外向量。

6.9 TLB 无效例外

原因

当一个虚地址引用匹配到一项被标记为无效的 TLB 项(TLB 有效位被清掉)时，TLB 无效例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外。Cause 寄存器的 ExcCode 字段值被设为 TLBL 或 TLBS，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。Random 寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

当发生下面情况之一时，TLB 项被标记为无效：

- 虚地址不存在
- 虚地址存在，但是不在主存中(缺页)
- 引用这个页而引发一个陷阱(例如,维护引用位)

在服务完 TLB 无效例外的起因之后，通过 TLBP 指令来定位 TLB 项(探测 TLB 来找匹配的项)，然后用标记位有效的一项来替换该 TLB 项。

6.10 TLB 修改例外

原因

当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”，因此该项不可写时，TLB 修改例外发生。该例外不可屏蔽。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器中的 ExcCode 字段值被设置为

MOD。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

内核使用失败的虚地址或虚页号来识别相应的访问控制信息。被识别的页可能允许或者不允许写访问；如果写访问不允许，那么写保护违例发生。

如果写访问是允许的，那么内核在其自己的数据结构内将页标记为可写。TLBP 指令把必须改变的 TLB 项的索引放入到 Index 寄存器中。包含物理页和访问控制位(D 位被设置)的一个字被取出放入 EntryLo 寄存器中，然后，EntryHi 和 EntryLo 寄存器被写入 TLB 中。

6.11 Cache 错误例外

原因

当处理器取指或者访存而出现内部 Cache 校验错时，Cache 错误例外发生。该例外不可屏蔽。

处理

偏移量为 0x100 的 Cache 错例外入口用于处理 Cache 错误例外。此时例外入口基址被设在不经过 Cache 的地址段。Cause 寄存器的 ExcCode 字段值被设为 CacheErr，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用还是访存操作指令。CacheErr 寄存器记录出错类型和在组相联 Cache 中的位置。CacheErr1 寄存器记录出错的指令虚地址或者内存物理地址，详见第 **3.30** 节 CacheErr 和 CacheErr1 寄存器描述。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

龙芯 GS464 处理器核对 Cache 错实现了硬件自纠错功能，应用程序可以简单地直接从例外返回。

若为指令 Cache 出错，出错的 Cache 行将被无效；若为数据 Cache 出错且只有一位错时，有错的数据将被自动纠正；若数据 Cache 出错且有两位错时，操作系统应视出错数据块的位置决定处理方式。

6.12 总线错误例外

原因

当处理器进行数据块的读取、更新或双字/单字/半字的读请求时收到外部的 ERR 完成应答信号，总线错误例外发生。该例外不可屏蔽。

处理

共用中断向量用于处理总线错误例外。Cause 寄存器的 ExcCode 字段值被设为 IBE 或 DBE，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

发生错误的物理地址可以通过 CP0 寄存器中的信息计算出来。

如果 Cause 寄存器中的 ExcCode 字段值被设置为 IBE 编码(表示是取指引用)，那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

如果 Cause 寄存器中的 ExcCode 字段值被设置为 DBE 编码(表示是读取或存储引用)，那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

于是，读取和存储引用的虚地址就可以通过解释这条指令来获得。而物理地址可以通过

TLBP 指令以及读取 EntryLo 寄存器内容计算物理页号来获得。导致例外发生的正在运行的进程会收到 UNIX SIGBUS(总线错误)信号，对该进程来说这通常是致命的。

6.13 整型溢出例外

原因

当一条 ADD、ADDI、SUB、DADD、DADDI 或 DSUB 指令执行，导致结果的补码溢出时，整型溢出例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 OV 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP(浮点例外/整型溢出)信号。对该进程来说，这个错误通常是致命的。

6.14 陷阱例外

原因

当 TGE、TGUE、TLT、TLTU、TEQ、TNE、TGEI、TGEUI、TLTI、TLTUI、TEQI、TNEI 指令执行，条件结果为真时，陷阱例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 TR 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP（浮点

例外/整型溢出）信号。对该进程来说，这个错误通常是致命的。

6.15 系统调用例外

原因

当执行 SYSCALL 指令的时候，系统调用例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 SYS 编码值。

如果 SYSCALL 指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 SYSCALL 指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位被清 0。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的系统调用区分可以分析 SYSCALL 指令的 Code 字段（位 25：6），以及载入 EPC 寄存器中所存地址的指令的内容。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 SYSCALL 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 SYSCALL 指令处在分支延迟槽中，则需要更复杂的算法，这已经超出了本节所能描述的范围。

6.16 断点例外

原因

当执行一条 BREAK 指令时，发生断点例外。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 BP 编码值。

如果 BREAK 指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 BREAK 指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位清 0。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的区分可以分析 BREAK 指令的 Code 字段（位 25：6），以及载入 EPC 寄存器中所存地址的指令的内容。如果这条指令在分支延迟槽中，那么 EPC 寄存器中的内容必须加上 4 以定位到该指令。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 BREAK 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 BREAK 指令在分支延迟槽中，那么为了恢复进程的继续执行，需要解释这条分支指令。

6.17 保留指令例外

原因

当试图执行一条没有在 MIPS64 Release2 定义并且非龙芯自定义的指令时，保留指令例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 RI 编码值。

如果保留指令指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令地址。

服务

此时，没有指令被解释执行。正在执行的导致例外发生的进程会收到 UNIX SIGILL/ILL_RESOP_FAULT(非法指令/保留的操作错误)信号。对该进程来说，这个错误通常是致命的。

6.18 协处理器不可用例外

原因

试图执行以下任意一条协处理器指令，将会导致协处理器不可用例外发生：

- 相应的协处理器单元（CP1 或 CP2）没有被标记为可用。
- CP0 单元没有被标记为可用，并且进程执行在用户或者超级用户的模式下的 CP0 指令。

这个例外是不可屏蔽的。

龙芯自定义扩展指令的协处理器不可用例外触发条件如下：

- 自定义扩展访存指令（表 2-15）、自定义扩展 64 位多媒体加速指令（表 2-18）、自定义扩展浮点访存指令（表 7-5）当 CP2 没有标记为可用时触发协处理器不可用例外。
- 自定义扩展浮点访存指令（表 7-5）当 CP1 没有标记为可用时触发协处理器不可用例外。

需要注意的是，CVT.D.LD、CVT.LD.D、CVT.UD.D 这三条自定义扩展浮点格式转换指令即使在 CP1 没有标记为可用时也不会触发协处理器不可用例外。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 CPU 编码值。Cause 寄存器的 CE 域指示四个协处理器的哪个被引用。如果这条指令不是在分支延迟槽中，EPC 寄存器保存了不可使用协处理器指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址。

服务

有以下的几种情形：

如果进程被授权访问协处理器，协处理器被标记为可用，那么相应的用户状态被恢复以便协处理器执行。

如果进程被授权访问协处理器，但是协处理器不存在或者有故障，则需要解释/模拟这条协处理器指令。

如果在 Cause 寄存器中的 BD 位被设置了，分支指令必须被解释；然后协处理器指令被模拟。例外返回时跳过例外的协处理器指令继续执行。

如果进程没有被授权访问协处理器，这时执行的进程收到 UNIX SIGILL/ILL_PRIVIN_FAULT(非法指令/特权指令错误)信号。这个错误通常是致命的。

6.19 浮点例外

原因

浮点协处理器使用浮点例外。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 FPE 编码值。

浮点控制/状态寄存器的内容指示这个例外产生的原因。

服务

清除浮点/状态寄存器中的适当位可以清除这个例外。

6.20 EJTAG 例外

当与某些 EJTAG 相关的条件满足时，触发 EJTAG 例外。具体描述见第 **X 章**

6.21 中断例外

原因

当八个中断条件中的一个触发，中断例外发生。这些中断的重要性依赖于特定的系统实现。

通过清掉在状态寄存器中的 Interrupt-Mask (IM)域中的相应的位，八个中断中的任何一个都可以被屏蔽，并且，通过清掉状态寄存器的 IE 位，可以一次屏蔽所有的八个中断。

处理

Cause 寄存器的 ExcCode 字段被置为 INT 编码值。根据当前配置，处理器使用传统的共用例外向量处理或者使用向量例外模式选择最高优先级的中断号对应的入口进行处理。

Cause 寄存器中的 IP 域指明了当前的中断请求。不止一个的中断位可能同时被设置(如果中断触发并且在寄存器被读到之前被撤消，甚至没有位被设置)。

IP[7]中断有三个源，除中断线 5 外，在 Count 寄存器内容与 Compare 寄存器内容相等时或者 CP0 性能计数器溢出时产生中断。时钟中断和性能计数器溢出中断由 Cause 寄存器中的 TI 和 PCI 位指示。

如果未用向量中断模式，软件需要对每一个可能的中断源进行查询来判断中断产生的原因（一个中断同时可能有多个源）。

服务

如果中断是由两个软件产生例外之一导致的，则设置 Cause 寄存器中的相应位，IP[1：

0]，为 0 来清除中断条件。

软件中断是非精确的。一旦软件中断触发，在例外被处理之前，程序还可能继续执行几条指令。定时器中断的清除通过向 Compare 寄存器写入值来完成。性能计数器中断的清除则是向计数器的溢出位，即位 31，写入 0 来实现。

冷重置和软重置会清除所有未完成的外部中断请求，IP[2]至 IP[6]。

如果中断是硬件产生的，那么撤消引起触发的中断管脚的条件，就可清除中断条件。

7 浮点协处理器

本章描述了龙芯 3A1000 处理器浮点协处理器（Floting Point Unit，简称 FPU）的特性，包括编程模型、指令集和指令格式、指令流水线以及异常。龙芯 3A1000 浮点协处理器及其相关的系统软件完全符合 ANSI/IEEE 754－1985 二进制浮点运算标准。

7.1 概述

FPU 作为 CPU 的协处理器，被称为 CP1（Coprocessor 1），通过扩展 CPU 的指令集来完成浮点算术运算功能。

FPU 由以下两个功能单元组成：

- FALU1 单元
- FALU2 单元

FALU1 模块可以执行除浮点访存以及浮点和定点数据传送之外的所有浮点操作，包括浮点加（减）法、浮点乘法、浮点乘加（减），浮点除法，浮点开平方根，浮点求倒，浮点开根后求倒，浮点与定点转换,浮点精度转换，浮点比较，转移判断和其它简单逻辑等。此外，FALU1 模块通过指令编码中 FMT 域的扩展与复用来执行 SIMD 媒体操作。

FALU2 执行浮点乘加运算部件（可计算浮点乘、加和浮点乘加指令），以及媒体指令操作。同时，龙芯 3A1000 的 FPU 支持执行 MIPS64 指令集中的并行单精度（Paired-Single，简称 PS）浮点指令。图 7-1 对龙芯 3A1000 体系结构中功能单元的组织构成进行了图解说明。

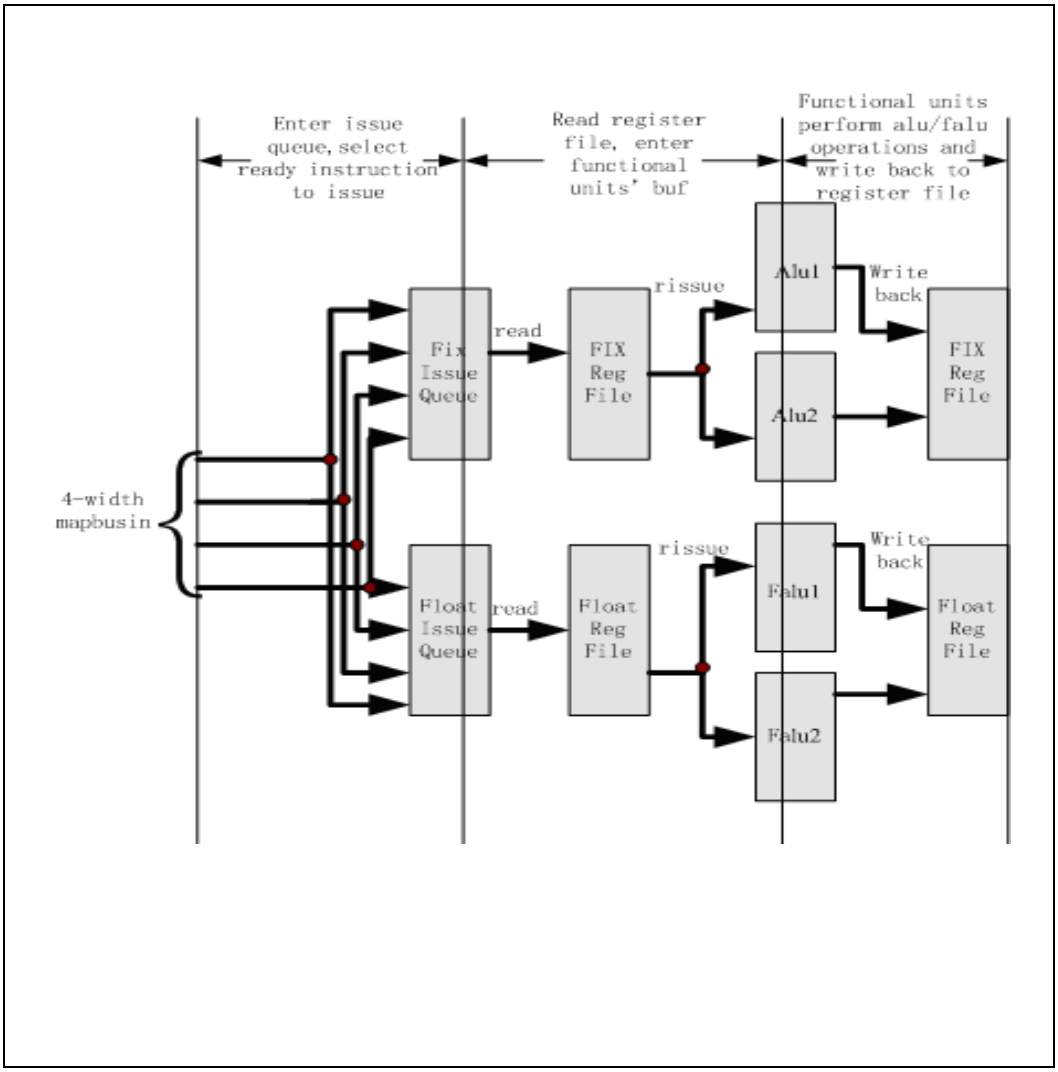


图 7-1 龙芯 3A1000 体系结构中功能单元的组织构成

浮点队列每个时钟周期可以分别发射 1 条指令到 FALU1 单元、1 条指令到 FALU2 单元。浮点寄存器文件为 FALU1 单元与 FALU2 单元各提供三个专用的读端口和一个专用的写端口。

7.2 FPU 寄存器

这部分描述 FPU 寄存器组和它们的数据组织结构。龙芯 3A1000 的 FPU 寄存器与 MIPS64 的 FPU 寄存器兼容。MIPS64 的 FPU 寄存器包括浮点寄存器和浮点控制寄存器。其中浮点控制寄存器包括 FIR（1 号）、FCSR（31 号）、FCCR（25 号）、FEXR（26 号）、FENR（28 号）等。

7.2.1 浮点寄存器

龙芯 3A1000 的浮点寄存器沿袭 R10000 用法，与 MIPS64 略有不同。在 Status 控制寄

存器的 FR 位为 1 时，均有 32 个 64 位的浮点寄存器，如下图所示；在 Status 控制寄存器的 FR 位为 0 时，R10000 只有 16 个 32 位或 64 位的浮点寄存器，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。

63 0	63 0
f1	f0
f3	f2
f5	f4
f7	f6
f9	f8
f11	f10
f13	f12
f15	f14
f17	f16
f19	f18
f21	f20
f23	f22
f25	f24
f27	f26
f29	f28
f31	f30

图 7-2 浮点寄存器格式

7.2.2 FIR 寄存器（CP1，0）

FIR 是 32 位只读寄存器，它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。龙芯 3A1000 里 FIR 的初始值为 0x00770501。

图 7-3 显示了 FIR 寄存器的格式，表 7-1 描述了该寄存器的域。

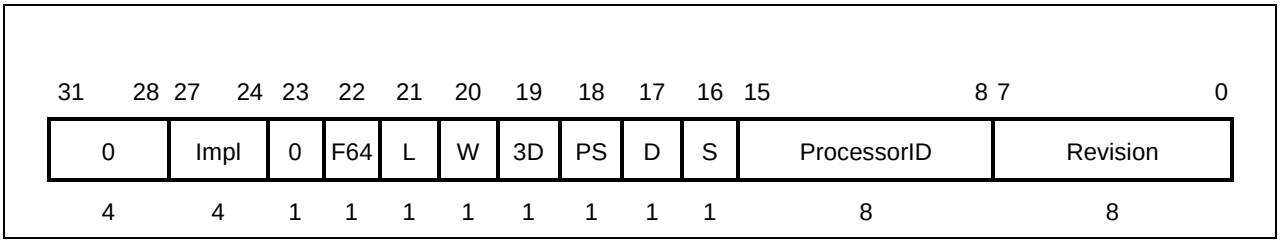


图 7-3 FIR 寄存器

表 7-1 FIR 寄存器域

域	描 述
0	保留。必须按 0 写入，读时返回 0。
Impl	实现相关
F64	浮点数据通路是否为 64 位 0－32 位 1－64 位
L	长字（64 位）定点数据类型是否实现 0－未实现 1－已实现
W	字（32 位）定点数据类型是否实现 0－未实现 1－已实现
3D	MIPS-3D ASE 是否实现 0－未实现 1－已实现
PS	浮点对数据类型是否实现 0－未实现 1－已实现
D	双精度浮点数据类型是否实现 0－未实现 1－已实现

域	描　述
S	单精度浮点数据类型是否实现 0－未实现 1－已实现
ProcessorID	浮点处理器标识
Revision	浮点单元的修订版本号

7.2.3 FCSR 寄存器（CP1，31）

FCSR 寄存器用于控制浮点单元的操作和表示一些状态。GS464 中 FCSR 的初始值为 0x00000F80。如图 7-4 所示是 FCSR 寄存器的格式，表 7-2 描述了 FCSR 寄存器的域。其中 E、V、Z、O、U、I 分别表示未实现操作、无效操作、除零、上溢、下溢、不精确。

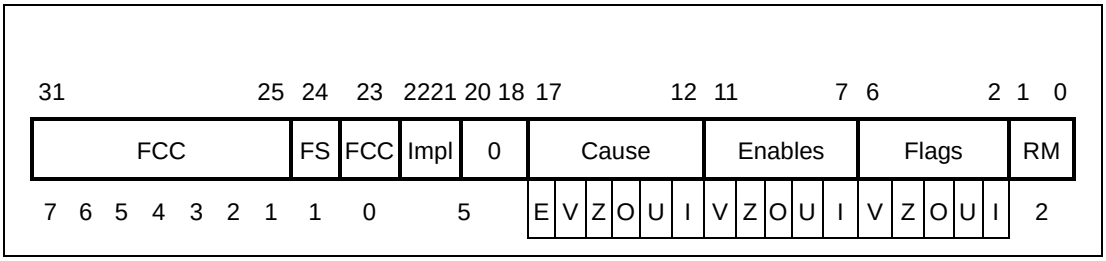


图 7-4 FCSR 寄存器

表 7-2 FCSR 寄存器域

域	描　述
0	保留。必须按 0 写入，读时返回 0。
FCC	浮点条件码。记录浮点比较结果，用于条件跳转或转移。
FS	冲刷到 0。这位设置时，非正常运算的结果将被设为 0，而不是产生一个例外。
Impl	实现相关，GS464 使用 FSCR[21]作为 top_mode，该位用于在译码时指示要不要使用 X86 的 TOP 寄存器对浮点寄存器号进行重命名。
Cause	当产生浮点运算的例外时，相应位被设置。
Enables	是否允许相应的条件产生例外。
Flags	是否有 IEEE 的浮点例外产生。（比如 Enables 里未打开相应位可查看本字段）
RM	双精度浮点数据类型是否实现 0－未实现 1－已实现

控制/状态寄存器条件（CC0）位

当一个浮点比较操作发生时，结果被保存在 CC0 位，即条件位。如果比较结果为真，则 CC0 位被置 1；反之则置 0。CC0 位仅能被浮点比较指令和 CTC1 指令所修改。

控制/状态寄存器导致(Causes)域

控制/状态寄存器的位 17：12 为导致（Causes）域，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。

控制/状态寄存器使能(Enables)域

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。

控制/状态寄存器标志(Flags)域

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。

当一个浮点例外发生时，Flag 位并不由硬件来设置；浮点例外的处理软件有责任在调用用户程序之前设置这些位。

控制/状态寄存器的舍入模式(RM)域

控制/状态寄存器中第 0 位和第 1 位组成了舍入模式（RM）域。如表 7-3 中所示，FPU 根据这些位所指定的舍入方式来对所有的浮点运算进行相应的舍入处理。

表 7-3 舍入模式位解码

舍入模式 RM(1:0)	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

7.2.4 FCCR 寄存器（CP1，25）

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 7-5 是 FCCR 寄存器的格式。

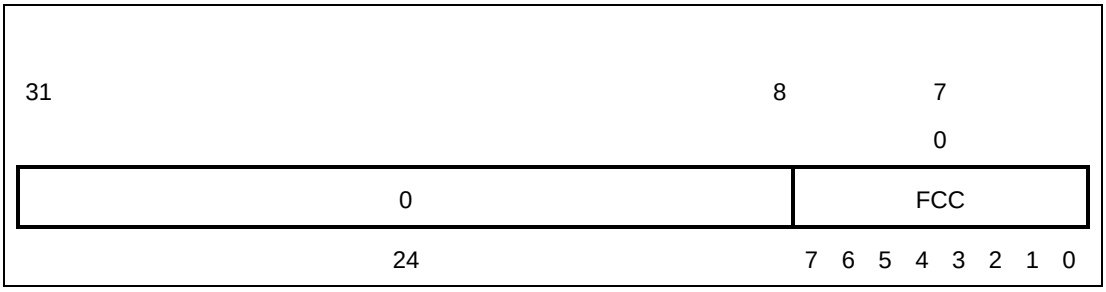


图 7-5 FCCR 寄存器

7.2.5 FEXR 寄存器（CP1，26）

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 7-6 显示了 FEXR 寄存器的格式。

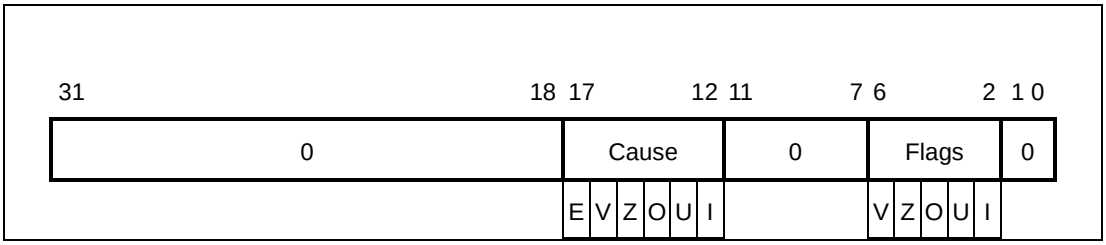


图 7-6 FEXR 寄存器

7.2.6 FENR 寄存器（CP1，28）

FENR 寄存器是访问 Enable, FS 和 RM 字段的另一种方式，其内容与 FCSR 里的相应字段完全相同。图 7-7 显示了 FENR 寄存器的格式。

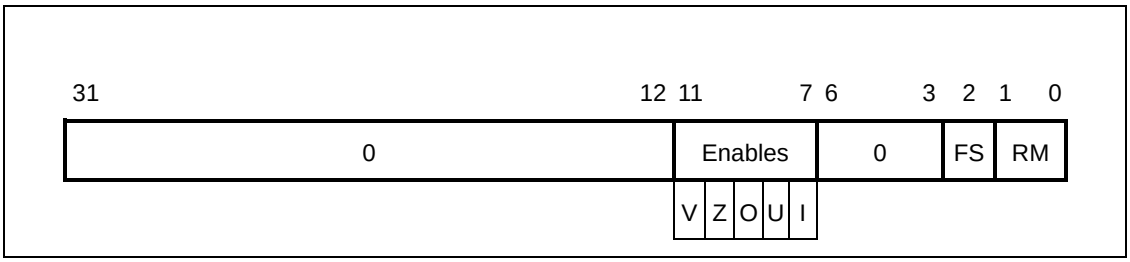


图 7-7 FENR 寄存器

7.3 浮点指令

7.3.1 MIPS64 兼容浮点指令列表

GS464 实现了 MIPS64 中 FPU 部分的所有数据类型，包括 S，D，W，L，和可选的 PS。表 7-4 列出了 GS464 中 MIPS64 部分的 FPU 指令。

表 7-4 MIPS64 的 FPU 指令集

OpCode	Description	MIPS ISA
算术指令		
ABS.fmt	绝对值	MIPS32
ADD.fmt	加法	MIPS32
DIV.fmt	除法	MIPS32
MADD.fmt	乘加	MIPS64

MSUB.fmt	乘减	MIPS64
MUL.fmt	乘法	MIPS32
NEG.fmt	求反	MIPS32
NMADD.fmt	乘加后求反	MIPS64
NMSUB.fmt	乘减后求反	MIPS64
RECIP.fmt	求倒数	MIPS64
RSQRT.fmt	平方根后求倒数	MIPS64
SQRT.fmt	平方根	MIPS32
SUB.fmt	减法	MIPS32
分支跳转指令		
BC1F	浮点假时跳转	MIPS32
BC1FL	浮点假时 Likely 跳转	MIPS32
BC1T	浮点真时跳转	MIPS32
BC1TL	浮点真时 Likely 跳转	MIPS32
比较指令		
C.cond.fmt	比较浮点值并置标志位	MIPS32
转换指令		
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.PL	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
访存指令		
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
MOVE 指令		
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
ALNV.PS	可变浮点对齐	MIPS64
MOV.fmt	复制 FPR	MIPS32
MOVF.fmt	浮点假时复制 FPR	MIPS32
MOVN.fmt	GPR 不为 0 时复制 FPR	MIPS32
MOVT.fmt	浮点真时复制 FPR	MIPS32
MOVZ.fmt	GPR 为 0 时复制 FPR	MIPS32
MTC1	从 GPR 复制低字到 FPR	MIPS32
MTHC1	从 GPR 复制高字到 FPR	MIPS32 R2

7.3.2 MIPS64 兼容浮点指令实现相关说明

GS464 与 MIPS64 Release 2 版本兼容，从功能上实现了 MIPS64 体系结构规定的所有 FPU 指令，但是有些指令在实现上有细微的并不影响兼容性但是比较重要的差别，以下两点值得编程人员注意。

（1）乘加、乘减指令。在执行 MADD.fmt，MSUB.fmt，NMADD.fmt，NMSUB.fmt 这四组指令时，GS464 的运算结果与 MIPS64 处理器略有不同，这是因为 GS464 在做乘加运算时只在最后结果处做精度舍入（即所谓 fused-mulitply-add），而 MIPS64 处理器在进行乘运算后和加运算后分别进行两次舍入，两种舍入处理方式的不同导致在某些情况下最终结果的最低位相差 1。

（2）单精度运算指令。在 Status 控制寄存器的 FR 位为 0 时，abs.s, add.s, ceil.w.d, ceil.w.s, div.s, floor.w.d, floor.w.s, mul.s, neg.s, round.w.d, round.w.s, sqrt.s, sub.s, trunc.w.d, trunc.w.s, mov.s, cvt.d.s, cvt.d.w, cvt.s.d, cvt.s.w, cvt.w.d, cvt.w.s, movf.s, movn.s, movt.s, movz.s 等 26 条指令不能使用奇数号寄存器，而 MIPS64 体系结构的处理器就可以，在这点上龙芯沿用了 MIPS R4000 与 MIPSR10000 的做法，与 MIPS64 的规定略有不同。（早期的 MIPS 处理器中 FR 位表示浮点寄存器是 16 个还是 32 个，MIPS64 中 FR 位表示浮点寄存器是 32 位还是 64 位）。

7.3.3 龙芯自定义扩展浮点指令

表 7-5 自定义扩展浮点访存指令

指令助记符	指令功能简述
GSSQC1	双源寄存器存定点四字
GSSWLEC1	带上越界检查的从浮点寄存器存字
GSLWXC1	带偏移的取浮点字
GSLQC1	双目标寄存器取浮点四字
GSLWLEC1	带上越界检查的取字到浮点寄存器
GSLWLC1	取字左部到浮点寄存器
GSLWRC1	取字右部到浮点寄存器
GSLDLC1	取双字左部到浮点寄存器
GSLDRC1	取双字右部到浮点寄存器

指令助记符	指令功能简述
GSLWGTC1	带下越界检查的取字到浮点寄存器
GSLDLEC1	带上越界检查的取双字到浮点寄存器
GSLDGTC1	带下越界检查的取双字到浮点寄存器
GSLDXC1	带偏移的取浮点双字
GSSWLC1	从浮点寄存器存字左部
GSSWRC1	从浮点寄存器存字右部
GSSDLC1	从浮点寄存器存双字左部
GSSDRC1	从浮点寄存器存双字右部
GSSWGTC1	带下越界检查的从浮点寄存器存字
GSSDLEC1	带上越界检查的从浮点寄存器存双字
GSSDGTC1	带下越界检查的从浮点寄存器存双字
GSSWXC1	带偏移的存浮点字
GSSDXC1	带偏移的存浮点双字

表 7-6 自定义扩展浮点格式转换指令

指令助记符	指令功能简述
CVT.D.LD	扩展双精度转化为双精度
CVT.LD.D	双精度转化为扩展双精度低位
CVT.UD.D	双精度转化为扩展双精度高位

7.4 浮点部件格式

7.4.1 浮点格式

FPU 既可以对 32 位（单精度）也可以对 64 位（双精度）符合 IEEE 标准的浮点数进行操作。32 位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（F＋S）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(F＋S)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。分别如图 7-8 所示。

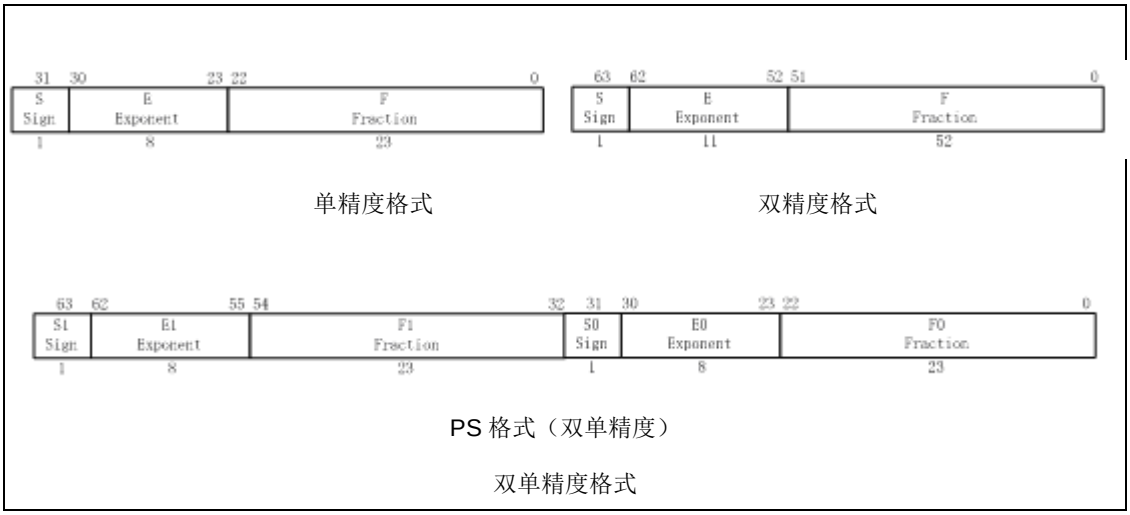


图 7-8 浮点格式

正如图 7-8 所示，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，$E = E_0 + Bias$，E_0 是不带偏移的指数
- 小数域，$F= .b_1b_2....b_{p-1}$

指数 E_0 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 和亚正常数）
- $E_{max} +1$ （用来编码∞和 NaN[Not a Number]）

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 由表 7-7 中的等式所决定。

表 7-7 计算单精度和双精度格式的浮点数的值的公式

NO.	公式
(1)	if E0 = Emax+1 and F≠0, then V = NaN, regardless of s
(2)	if E0 = Emax+1 and F = 0, then V= (–1)S∞□
(3)	if Emin≤E0≤Emax, then V = (–1)S2 E0 (1.F)
(4)	if E0 = Emin–1 and F≠0, then V= (–1)S2Emin(0.F)
(5)	if E0 = Emin–1 and F = 0, then V = (–1)S0

对于所有的浮点格式，如果 V 是一个 NaN，那么 F 的最高位决定了这个数是 Signaling

NaN 还是 Quiet NaN：如果 F 的最高位被设置，那么 V 是 Signaling NaN，否则 V 是 Quiet NaN。

表 7-8 定义了一些浮点格式的相关参数的值；浮点的最大值和最小值在表 7-9 中给出。

表 7-8 浮点格式参数值

参数	格式	
	单精度	双精度
Emax	+127	+1203
Emin	-126	-1022
指数偏移量	+127	+1023
指数位宽度	8	11
整数位	隐藏（Hidden）	隐藏（Hidden）
F (小数位宽度)	24	53
格式总宽度	32	64

表 7-9 最大数和最小数的浮点值

类型	值
单精度浮点最小数	1.40129846e–45
单精度浮点最小正规数	1.17549435e–38
单精度浮点最大数	3.40282347e+38
双精度浮点最小数	4.9406564584124654e–324
双精度浮点最小正规数	2.2250738585072014e–308
双精度浮点最大数	1.7976931348623157e+308

7.5 FPU 指令流水线概述

FPU 提供一个和 CPU 指令流水线并行的指令流水线。它和 CPU 共享基本的 9 级流水线体系结构，但根据浮点操作的不同，执行流水级又细分为 2~6 个流水级。每个 FPU 指令被两个浮点功能单元中的一个执行：FALU1 或者 FALU2。FALU1 可以执行所有的浮点运算操作及媒体操作。FALU2 仅执行浮点加减、乘法、乘加操作以及所有媒体操作。

每个 FALU 单元每个周期能够分别接收 1 条指令，并能向浮点寄存器文件分别送出一个结果。在每个 FALU 单元中，浮点加减、浮点乘法、浮点乘加运算需要 6 个执行周期；定点与浮点间的格式转换运算需要 4 个执行周期；浮点除法根据操作数的不同需要 4~16 个执行周期；浮点开平方根根据操作数的不同需要 4~31 个执行周期，其它浮点运算需要 2 个执行

周期。在每个 FALU 单元中，如果两条有着不同执行周期的指令在同一拍输出结果，在这种情况下，执行周期较短的指令优先向总线输出结果。其中除浮点除法和浮点开根之外的浮点操作和所有媒体操作都是全流水的。如果同时有两个浮点除法指令或者两个浮点开平方根指令在 FALU1 中，那么 FALU1 单元将向前一流水级发出一个停顿信号，并且 FALU1 单元在除法或开平方根指令写回前不能接收新的指令。

7.6 浮点例外处理

该节描述了浮点计算的例外。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。表 7-10 列出了 FPU 对于每个 IEEE 例外的默认处理。

表 7-10 例外的默认处理

域	描述	舍入模式	默认操作
I	非精确例外	Any	提供舍入后的结果
U	下溢例外	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	上溢例外	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	Any	提供一个相应的带符号的无穷大数
V	非法操作	Any	提供一个 Quiet Not a Number(QNaN)

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

不精确例外 (I)

FPU 在发生如下的情况时产生不精确例外：

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

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

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

非法操作例外 (V)

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

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

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

陷阱被使能的结果：源操作数的值不被发送。

陷阱不使能的结果：如果没有其他例外发生，QNaN被发送到目标寄存器中。

除零例外 (Z)

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

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

陷阱不使能的情况：如果没有陷阱发生，结果是有符号的无穷值。

上溢例外 (O)

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

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

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

下溢例外 (U)

两个相关的事件导致了下溢例外：

- 一个很小的在$\pm 2^{Emin}$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。
- 用亚正常数据（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$\pm 2^{Emin}$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$\pm 2^{Emin}$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：

- 亚正常数据的失真（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：

- 亚正常操作数（Denormalized Operand），比较指令除外
- Quite Not a Number 操作数（QNaN），比较指令除外
- 亚正常数据或者下溢，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：亚正常和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。

8 性能分析和优化

本章提供了龙芯 3A1000 体系结构中一些与软件性能优化相关的信息，包括指令延迟和指令循环的间隔、扩展指令、指令流和存储访问处理等，可供编译器和其他软件开发者参考。

8.1 用户指令的延迟和循环间隔

表 8-1 给出了在 ALU1/2, MEM, FALU1/2 功能单元中执行的所有用户指令的延迟和循环间隔，不包括内核指令和控制指令。这里的指令延迟是该指令发射到其结果能被下一条指令使用所需要的拍数（一个处理器周期为一拍）。例如，大部分的 ALU 指令延迟为 2,这表示 ALU 指令的结果要隔一拍后才能被后续指令使用。因此，形如 i = i + 1 的相关循环（下一个循环依赖上一个循环的结果）不能每拍出一个结果。而一个指令的循环间隔则是指功能部件接受这种指令的频度，1 表示每拍都能接受一个以上的同类指令，n 表示功能部件接受一个该指令后，需要等 n-1 拍后才能再接受同类指令。全流水功能部件的指令循环间隔为 1。

表 8-1 龙芯 3A1000 指令延迟

指令类型	执行部件	延迟	循环间隔
整型操作			
add/sub/logical/shift/lui/cmp	ALU1/2	2	1
trap/branch	ALU1	2	1
MF/MT HI/LO	ALU1/2	2	1
(D)MULT(U)	ALU2	5	2
(D)MULT(U)G	ALU2	5	1
(D)DIV(U)	ALU2	2-38	10-76
(D)DIV(U)G	ALU2	2-38	4-37
(D)MOD(U)G	ALU2	2-38	4-37
load	MEM	5	1
store	MEM	-	1
浮点操作			
(D)MTC1/(D)MFC1	MEM	5	1
abs/neg/C.cond/bc1t/bc1f/move/cvt*	FALU1	3	1
round/trunc/ceil/floor/cvt*	FALU1	5	1
add/sub/mul/madd/msub/nmadd/nmsub	FALU1/2	7	1
div.s	FALU2	5-11	4-10
div.d	FALU2	5-18	4-17
sqrt..s	FALU2	5-17	4-16
sqrt..d	FALU2	5-32	4-31
lwc1,ldc1	MEM	5	1

指令类型	执行部件	延迟	循环间隔
swc1,sdc1	MEM	-	1

对于表 1，还有以下几点说明：

这里的 load/store 操作的循环间隔并不包括 LL/SC。LL/SC 是等待发射操作，只有当它们位于 reorder 队列队首， 而且此时 cp0 队列为空时，才可以被发射。

对于 HI/LO 寄存器，没有特别的使用限制。它们和其它的通用寄存器一样使用。这个表中并不包含 CTC1/CFC1。它们和许多其它的控制指令一样被序列化。

这个表中也不包含多媒体指令。因为它们是通过扩展普通浮点指令的格式而完成的，它们的功能单元和延时与被扩展的指令相同。

8.2 指令扩充和使用注意事项

龙芯 3A1000 完成了以下几种指令扩充：

只写一个结果到通用寄存器的定点乘除。包括 12 条指令：

(D)MULTG, (D)MULTUG, (D)DIVG

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

在标准的 MIPS 指令集中，乘法和除法在一个操作中需要写两个特殊的结果寄存器(HI/LO)，它们在 RISC 流水线中很难实现。为了使用这些结果，将不得不使是用额外的指令把它从 HI/LO 中取出送入通用寄存器中。更麻烦的是，由于流水线的问题，很多 MIPS 处理器对这些指令的使用还有些限制。这些新指令执行速度更快，同时也更容易使用。

由于寄存器重命名实现原因，在龙芯 3A1000 处理器上，标准 MIPS 指令集中涉及到 HI/LO 寄存器操作的相关指令均要等到处理器指令队列中位于这些指令前的指令全部提交之后才能开始执行，会造成流水线的停顿。但使用上述扩充指令时，流水线不存在 HI/LO 寄存器重命名问题，这些指令不会受到阻塞。因此，在使用汇编指令编写程序时，应该尽量使用上述扩展指令。如果确实需要使用 HI/LO 运算结果，例如需要在 64 位乘法之后得到乘法结果的高 64 位，那么程序员应该清楚的知道使用标准 MIPS 乘法指令将会对流水线造成的影响。

定点操作使用浮点的数据通路：

在执行定点程序的过程中，浮点数据通路常常处于空闲状态，这些指令使得我们有机会利用它们，进一步增加指令并行的程度。

8.3 编译器使用提示

开源编译套件 GCC 目前已经支持龙芯 3A1000 处理器的体系结构调优选项。在 GCC4.6.0 以上版本中，使用-march=loongson3A，即可使用针对该处理器的流水线描述来进行代码调度，生成的代码也能够充分利用龙芯 3A1000 的指令扩充。在大部分情况下，使用该选项都能在龙芯 3A1000 处理器上得到性能提升。

此外，我们在对 SPEC CPU2000 基准测试集进行编译器调优空间探索过程中，总结出以下 GCC 的编译选项可能对龙芯 3A1000 处理器性能有提升作用。在对程序进行精细调优过程中，下列选项具有一定的参考意义。

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

8.4 指令流

龙芯 3A1000 是一个多发射高度并行的处理器，对本质上是串行的指令流的处理可能会对程序性能产生明显的影响，本节讨论关于指令对齐、转移指令、指令调度等问题。

8.4.1 指令对齐

在一个周期内，龙芯 3A1000 可以从一个高速缓存行中取出四条指令，但这四条指令不能跨越高速缓存行的边界。我们应该对那些经常性被执行的基本块进行合适的对齐，以避免跨越高速缓存行的边界。此外，如果在一次所取的四条指令中存在转移指令，也会影响取指令的效率。如果第一条是转移分支指令，而且转移预测是成功，那么最后两条指令将被抛弃。

如果最后一条是转移指令，即使转移成功，处理器也不得不再取下一个高速缓存行以得到它的延迟槽中的指令。龙芯 3A1000 一个周期只能给一条转移指令译码，如果在一束指令中存在两条转移指令的话，它将需要两个周期来完成译码，也就是说取指部件将被阻塞一个周期。

8.4.2 转移指令的处理

在龙芯 3A1000 的处理器中，指令流地址的一个意想不到的变化会浪费大约 10 条指令的时间。“意想不到”可以是由转移成功的指令导致，也可能会由转移预测错误导致。对于目前的龙芯 3A1000 而言，即使是一个正确预测且预测转移成功的转移指令也比顺序代码慢，它会浪费一个周期，因为对于普通条件转移指令，转移目标缓存器（BTB）不能给出下一个正确的程序计数器 PC 值。

编译器可以通过以下的方法来减少转移指令引起的开销：

龙芯 3A1000 的转移指令预测方法和其它的高性能处理器都不同，且不同的版本都有一些细微的差别。基于执行剖析（profile)，编译器可以根据实际的转移频率对代码位置进行重新安排，从而得到较好的预测结果。

尽可能使基本块变大。一种比较好的优化结果就是使得在两条转移成功的转移指令之间平均有 20 条指令。为了使它们之间至少含有 20 条指令，这就需要循环展开，还要把不到 20 条指令的子程序直接内联展开。龙芯 3A1000 实现了条件性移动指令，它可以用来减少分支指令数量。通过执行剖析来重新组织代码也有助于这个优化。

在龙芯 3A1000 上，不同的转移指令使用不同的方式进行预测：

静态预测

针对 likely 类转移指令和直接跳转指令。

G-share 预测器

一个 9 位的全局历史寄存器 GHR，和一个有 4K 项的模式历史表 PHT。用于条件转移指令。

BTB（转移目标缓存）

有 16 项的全相联的缓存。用于预测寄存器跳转指令的目标地址。

RAS（返回地址栈）

4 项，被用于预测函数返回的目标地址。

以下有几点关于软件需要注意的地方：

在龙芯 3A1000 处理器上需要特别小心地使用 likely 类转移指令。尽管 likely 类转移

指令也许对顺序标量处理器的简单的静态调度很有效，但是它对现代高性能处理器并不是同样有效。因为现代高性能处理器的转移预测硬件是比较复杂，它们通常有 90%以上的正确预测率。（比如说，龙芯 3A1000 能够正确预测 85%-100%，平均 95%的条件转移的转移方向）在这种情况下，编译器不应该使用预测率不太高的 likely 类转移指令。事实上，我们发现带有-mno-branch-likely 选项的 gcc 通常会工作得更好。

取指译码单元被划分成 3 个流水段，其中转移的目标地址在第三阶段被计算。转移成功的转移指令将会导致有两个周期的停顿，也就是说，如果在周期 0 取出一条转移指令，在周期 1 取出地址为 PC+16 的指令，周期 2 取出地址为 PC+32 的指令，在周期 3 时，才会取到转移指令的目标地址。所以减少转移成功的转移指令数将会比较有一些帮助。

龙芯 3A1000 中的 BTB 仅被用于寄存器跳转指令(没有 jr31 和 jalr 例外的 jr 指令)。

通过一个 4 项的 RAS 来预测 jr31 指令的目标地址。函数返回的预测有效性取决于那些使用 jr31 指令作为函数返回指令的软件。

8.4.3 指令流密度的提高

编译器应该尽量利用执行剖析，以确保调入指令 Cache 的那些字节均被执行。这就要求跳转指令的目标地址需对齐，并且把那些很少被执行的代码移出 Cache 行。

8.4.4 指令调度

龙芯 3A1000 内部有比较大的指令窗口会进行动态的指令调度，但是由于处理器内部各种资源有限无法做到最优的调度，编译器可以在一定程度上协助处理器进行更好的调度。

现代的编译器（如 gcc）有指令调度的支持，把龙芯 3A1000 内部的部件资源情况和指令的延迟情况（见表 1)告诉编译器，它就能够进行较好的调度。

8.5 存储器访问

Load-store 指令的执行对整个系统性能有很大的影响。如果一级数据高速缓存中包括所需的内容，那么这些指令可以很快被执行。如果数据只在二级高速缓存则稍微慢些，如果只在主存中则会有很大的延迟。不过，乱序执行和非阻塞 cache 可以减少由这些延时带来的性能损失。

龙芯 3A1000 包括 4 个片上二级 cache 模块，每个二级 cache 模块大小为 1MB，共 4MB。每个模块的组织为四路组相联。龙芯 3A1000 内置 DDR 内存控制器，最大限度地减少了

内存访问的延迟。龙芯 3A1000 系统中内存的频率与处理器工作频率均可单独配置，尽量提高内存频率，减小与处理频率的差距，非常有利于提升大部分应用程序的性能。有关内存控制器更多的信息可以参考 3A1000 处理器文档中相关部分。

龙芯 3A1000 提供了预取指令，可以通过 load 到 0 号定点寄存器的方式来将数据预取到一级数据 Cache。此外龙芯 3 号中的 DSP 引擎可以将内存或 IO 中的数据预取到二级 Cache 中，具体可参考手册相关部分。

编译器应该尽量减少不必要的存储访问。目前的龙芯 3A1000 处理器的存储指令延迟较大（即使是高速缓存命中，也需要 4 个周期），同时指令窗口也没有大到可以容忍几十周期的访问延迟。

软件还要特别注意数据对齐的问题。集合体（数组，一些纪录，子程序堆栈帧）应该被分配在对齐的高速缓存行边界上，这样就可以利用高速缓存行对齐数据通路，还可以降低高速缓存行被填满的数目。在那些强迫不对齐（例如 gcc 的 packed 属性）的集合体（纪录，普通块）中的项目中，应该产生一个编译时间的警告信息。在龙芯 3A1000 中正常的 load/store 指令有对齐要求，不满足要求的指令要通过内核模拟来实现。例如，从非四字节对齐的地址取一个字（四字节）会触发例外，由操作系统来处理；通常操作系统需要几千个处理器周期才能完成这个任务。因此用户需要知道这些警告信息代表代码的性能可能会很低。编译参数的代码都默认这些参数是对齐的。那些经常被使用的标量应该驻留在寄存器中。

8.6 其他提示

使用所有的浮点寄存器。尽管 O32 ABI 只开放了 16 个给用户使用，但是龙芯 3A1000 提供了 32 个 64 位的浮点寄存器。使用 N32 或者 N64 ABI 有助于发挥处理器的性能。

使用性能计数器。龙芯 3A1000 的性能计数器可以用来监控程序的实时性能参数。编译器和软件开发者可以通过分析这个结果来改进他们的代码。