龙芯中科
LOONGSON TECHNOLOGY

LOONGSON

龙芯架构参考手册

卷一：基础架构

V1.02

龙芯中科技术股份有限公司

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。

龙芯中科技术股份有限公司

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

本手册是龙芯架构参考手册的第一卷，用于介绍龙芯架构中基础架构的内容。

版本历史

文档更新记录	文档名	龙芯架构参考手册 卷一 基础架构
	版本号	1.02
	创建人	芯片研发部

更新历史

版本号	更新内容
0.80	内部评审版本。
0.90	内部评审版本。
0.91	内部评审版本。
1.00	正式发布版本。
1.01	手册内容修正： 1) 3.2.3 节中浮点数转整数操作不判断是否允许报浮点不精确例外，即总是执行 convertToIntegerExact…或 roundToIntegralExact 操作。 2) 4.2.1 和 7.2.3 节中对 CSR 指令访问未定义或未实现 CSR 的行为进一步明确。 3) 5.1 节，LA32 架构下最大物理地址范围调整为 36 位。 4) 修正部分 CSR 寄存器及其域的名称不统一的问题，修正若干书写错误。 手册内容完善： 1) 2.2.1.3、2.2.4、2.2.5.3、2.2.7.2、3.2.5 节中，对指令汇编表示中所用立即数与指令码中立即数的关系进行了说明。
1.02	指令内容调整： 1) 明确 FSCALEB.S/D、FLOGB.S/D 和 FRINT.S/D 这 6 条指令仅需在 LA64 架构下实现。 手册内容完善与修正： 2) 2.2.7.2 节中补充了 LL/SC 指令计算地址时偏移值的具体操作方式。 3) 2.1.4、2.1.5 和 5.2.1 节对应用程序访存触发地址错例外的判定规则做了补充说明。 4) 5.4.2 节最后一句中[$\log_2$PS-1:12]应为[PS-1:12]。 5) 5.4.4 节中报页特权等级不合规例外应是 SignalException(PPI)。 6) 7.5.3 节 CSR 寄存器 TLBELO0 和 TLBELO1 在 LA32 架构下 PPN 域的描述考虑 PALEN<36 的情况。 7) 7.5.4 节中 ASID 域描述中，不应作为 INVTLB 指令查询 TLB 的 ASID 键值信息。 8) 7.5.15 节 CSR 寄存器 TLBRELO0 和 TLBRELO1 在 LA32 架构下的 PPN 域的定义与 CSR 寄存器 TLBELO0 和 TLBELO1 保持一致。

手册信息反馈: service@loongson.cn

也可通过问题反馈网站 http://bugs.loongnix.org/ 向我司提交产品使用过程中的问题，并获取技术支持。

目　录

图 目 录

表 目 录

1 引言

龙芯架构 LoongArch 是一种精简指令集计算机(Reduced Instruction Set Computing, 简称 RISC)风格的指令系统架构。龙芯架构参考手册用于阐述龙芯架构规范，共包含三卷，本文档是第一卷，讲述龙芯架构中基础部分的内容。

1.1 龙芯架构概述

龙芯架构具有 RISC 指令架构的典型特征。它的指令长度固定且编码格式规整，绝大多数指令只有两个源操作数和一个目的操作数，采用 load/store 架构，即仅有 load/store 访存指令可以访问内存，其它指令的操作对象均是处理器核内部的寄存器或指令码中的立即数。

龙芯架构分为 32 位和 64 位两个版本，分别称为 LA32 架构和 LA64 架构。LA64 架构应用级向下二进制兼容 LA32 架构。所谓“应用级向下二进制兼容”一方面是指采用 LA32 架构的应用软件的二进制可以直接运行在兼容 LA64 架构的机器上并获得相同的运行结果，另一方面是指这种向下二进制兼容仅限于应用软件，架构规范并不保证在兼容 LA32 架构的机器上运行的系统软件（如操作系统内核）的二进制直接在兼容 LA64 架构的机器上运行时总是获得相同的运行结果。

龙芯架构采用基础部分（Loongson Base）加扩展部分的组织形式（如图 1-1 所示)。其中扩展部分包括:二进制翻译扩展(Loongson Binary Translation, 简称 LBT)、虚拟化扩展(Loongson Virtualization, 简称 LVZ)、向量扩展（Loongson SIMD Extension，简称 LSX）和高级向量扩展（Loongson Advanced SIMD Extension，简称 LASX)。

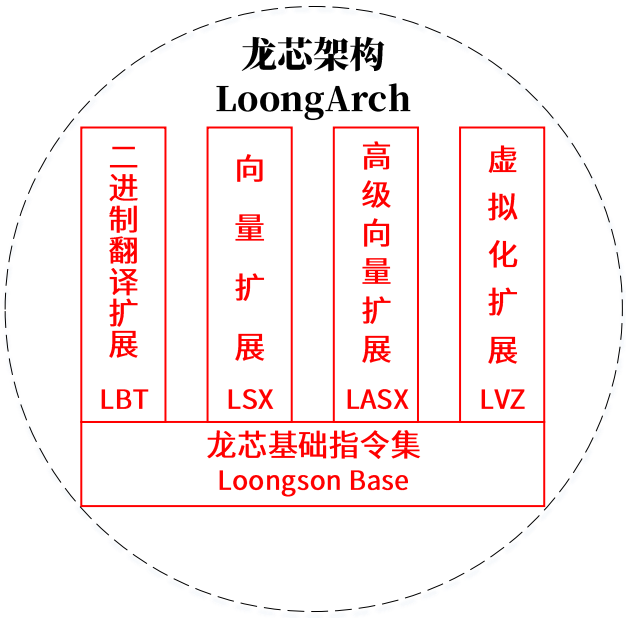


图 1-1 龙芯架构组成部分

龙芯架构的基础部分包含非特权指令集和特权指令集两个部分，其中非特权指令集部分定义了常用的整数和浮点数指令，能够充分支持现有各主流编译系统生成高效的目标代码。

龙芯架构的虚拟化扩展部分用于为操作系统虚拟化提供硬件加速以提升性能。这部分涉及的基本上都是特权资源，包括一些特权指令和控制状态寄存器、以及在例外和中断、存储管理等方面添加新的功能。

龙芯架构的二进制翻译扩展部分用于提升跨指令系统二进制翻译在龙芯架构平台上的执行效率。其在基础部分之上进行扩展，同样包含非特权指令集和特权指令集两个部分。

龙芯向量指令扩展和高级向量指令扩展两部分均是采用 SIMD 指令来加速计算密集型应用。两个扩展部分在指令功能上基本一致，区别在于向量指令扩展操作的向量位宽是 128 位而高级向量指令扩展操作的向量位宽是 256 位。

对于一个兼容龙芯架构的实现，架构中的基础部分必须实现，扩展部分可以选择实现。各扩展部分可以灵活选择，不过，当选择实现 LASX 时必须同时实现 LSX。除了将一整个扩展部分作为可选项，在基础部分和各扩展部分中还进一步包含了一些可选实现的功能子集。所有这些可选实现的扩展部分或是各部分中可选实现功能子集的具体实现情况，软件可以用 CPUCFG 指令读取的配置信息字来动态予以识别。

龙芯架构的后续演进采用细粒度增量式演进的方法。所谓“细粒度”是指，基础部分或是扩展部分中的各功能子集，都可以独立地演进。所谓“增量式”是指，对于任何一个可独立演进的部分，高版本总是向前二进制兼容低版本。

本手册从第 2 章开始将对龙芯架构的规范展开具体描述。其中第 2 章和第 3 章的内容涉及基础架构中的非特权指令集部分，包括基础整数指令和基础浮点数指令的功能定义及其应用级编程模型。第 4 章到第 7 章的内容用于讲述基础架构中特权资源，主要包括特权指令、控制状态寄存器（Control and Status Register，简称 CSR）的介绍，以及在运行模式、例外和中断、存储管理等方面的功能规范。本文档正文中描述指令功能定义时涉及的伪码描述集中于附录 A，所涉及的指令的具体编码定义统一列举在附录 B 中。

1.2 指令编码格式

龙芯架构中的所有指令均采用 32 位固定长度，且指令的地址都要求 4 字节边界对齐。当指令地址不对齐时将触发地址错例外。

指令编码的风格是所有寄存器操作数域都从第 0 比特开始从低到高依次摆放。操作码都是从第 31 比特开始从高到低依次摆放。如果指令中包含有立即数操作数，那么立即数域位于寄存器域和操作码域之间，根据不同指令类型有不同的长度。具体来说，包含 9 种典型的指令编码格式，即 3 种不含立即数的编码格式 2R、3R、4R，以及 6 种含立即数的编码格式 2RI8、2RI12、2RI14、2RI16、1RI21、I26。表 1-1 列举了这 9 种典型编码格式的具体定义。需要指出的是，存在少数指令，其指令编码域并不完全等同于这 9 种典型指令编码格式，而是在其基础上略有变化。不过这种指令的数目并不多，且变化的幅度也不大，不会对于编译系统的开发人员带来不便。

表 1-1 龙芯架构典型指令编码格式

	31–26	25–20	19–15	14–10	9–5	4–0
2R-type	opcode				rj	rd
3R-type	opcode			rk	rj	rd
4R-type	opcode		ra	rk	rj	rd
2RI8-type	opcode		I8		rj	rd
2RI12-type	opcode		I12		rj	rd
2RI14-type	opcode		I14		rj	rd
2RI16-type	opcode		I16		rj	rd
1RI21-type	opcode		I21[15:0]		rj	I21[20:16]
I26-type	opcode		I26[15:0]		I26[25:16]	

1.3 指令汇编助记格式

指令汇编助记格式主要包括指令名和操作数两部分。龙芯架构对指令名和操作数的前、后缀进行了统一考虑，以方便汇编编程人员和编译器开发人员的使用。

首先，通过指令名的前缀字母来区分非向量指令和向量指令、整数和浮点数指令。所有 128 位向量指令的指令名以字母“V”开头；所有 256 位向量指令的指令名以字母“XV”开头。所有非向量浮点数指令的指令名以字母“F”开头；所有 128 位向量浮点指令的指令名以“VF”开头；所有 256 位向量浮点指令的指令名以“XVF”开头。

其次，绝大多数指令通过指令名中“.XX”形式的后缀来指示指令的操作对象，且这种形式的后缀仅用来表征指令操作对象的类型。对于操作对象是整数类型的，指令名后缀为.B、.H、.W、.D、.BU、.HU、.WU、.DU 分别表示该指令操作的数据类型是有符号字节、有符号半字、有符号字、有符号双字、无符号字节、无符号半字、无符号字、无符号双字。不过这里有一种特殊情况，当操作数是有符号数还是无符号数不影响运算结果时，指令名中携带的后缀均不带 U，但此时并不限制操作对象只能是有符号数。对于操作对象是浮点数类型的，或者更具体来说是那些指令名以“F”、“VF”和“XVF”开头的指令，其指令名后缀为.H、.S、.D、.W、.L、.WU、.LU 分别表示该指令操作的数据类型是半精度浮点数、单精度浮点数、双精度浮点数、有符号字、有符号双字、无符号字、无符号双字。此外在涉及向量

操作的指令中，指令名后缀.V 表示该指令是将整个向量数据作为一个整体进行操作。需要指出的是，并不是所有指令都用“.XX”形式的后缀来指示指令的操作对象。当指令操作对象的数据位宽由所执行处理器是 32 位实现还是 64 位决定的，如 SLT 和 SLTU 指令，这种指令是不加后缀的。此外，操作 CSR、TLB 和 Cache 的特权态指令以及在不同寄存器文件之间移动数据的指令也是不加这种表征操作对象类型的后缀的。

当源操作数和目的操作数的数据位宽和有无符号情况一致时，指令名只有一个后缀。如果所有源操作数的数据位宽和有无符号情况一致，但是与目的操作数的不一致，那么指令名将有两个后缀，从左往右，第一个后缀表明目的操作数的情况，第二个后缀表明源操作数的情况。如果源操作和目的操作数的情况更复杂，那么指令名将从左往右依次列出目的操作数和每个源操作数的情况，其次序与指令助记符中后面操作数的顺序一致。例如，指令“MULW.D.WU rd, rj, rk”中.D 对应目的操作数 rd，.WU 对应源操作数 rj 和 rk，表明这个乘法是将两个无符号字相乘，得到的双字结果写入 rd 中。又例如，指令“CRC.W.B.W rd, rj, rk”中第一个.W 对应 rd，.B 对应 rj，第二个.W 对应 rk，表明这个 CRC 校验操作是将 rj 中的字节消息与 rk 中 32 位原校验值经生成新的 32 位校验值结果写入到 rd 中。

寄存器操作数通过不同的首字母表明其属于哪个寄存器文件。以“rN”来标记通用寄存器，以“fN”来标记浮点寄存器，以“vN”来标记 128 位向量寄存器，以“xN”来标记 256 位向量寄存器。其中 N 是数字，表示操作的是该寄存器文件中第 N 号寄存器。

1.4 本手册采用的一些书写规则

1.4.1 指令名缩写规则

龙芯架构所定义的指令中，常出现一些指令，它们的运算模式相同或相似，仅仅是操作对象存在一些差异。在本手册指令功能介绍过程中，常将这样的指令集中在一处进行介绍，以方便使用者学习和查阅。为了行文的简洁，本手册采用了一种指令名缩写规则。该规则中，{A/B/C}表示此处分别使用 A、B、C 来参与构成不同的指令名，A[B]表示此处分别使用 A 和 AB 来参与构成不同的指令名。例如，ADD.{W/D}表示的是 ADD.W 和 ADD.D 两个指令名，而 BLT[U]表示的是 BLT 和 BLTU 两个指令名，更复杂一点的，ADD[I].{W/D}表示的是 ADD.W、ADD.D、ADDI.W 和 ADDI.D 四个指令名。

需要注意的是，这种缩写规则仅仅是一种书写规则，它并不意味着被缩写在一起的若干条指令也一定具有极为相近的指令编码。

1.4.2 控制状态寄存器指称方式

龙芯架构下定义了一系列控制状态寄存器（Control and Status Register，简称 CSR），用于控制指令的执行行为，每个 CSR 通常包含若干个域。本手册在叙述过程中，将采用 CSR.%%%%.####的形式来指称名称缩写为%%%%的控制状态寄存器中名字为####的域。例如，CSR.CRMD.PLV 表示 CRMD 这个寄存器中的 PLV 域。

在实现虚拟化扩展的情况下，处理器中会存在两套 CSR，一套属于主机（Host）一套属于客户机（Guest）。当叙述过程中仅通过上下文内容无法区分这两套 CSR 时，以 CSR.XXXX 表示主机的 CSR，以 GCSR.XXXX 表示客户机的 CSR。

2 基础整数指令

龙芯架构基础部分的非特权指令集按照软件运行时上下文内容的差异可划分为基础整数指令和基础浮点数指令两个部分。本章将描述其中的整数指令部分。基础整数指令部分是非特权指令子集中最基础的一部分。

2.1 基础整数指令编程模型

本节所要描述的基础整数指令编程模型只涉及应用软件[1]开发人员所需关注的内容。这些内容主要属于架构中的非特权部分，不过由于应用软件所处的运行环境总不免与一些特权资源相关，因此在必要的地方将会引入有关特权资源的概念以确保叙述的完整性。有关特权资源的内容在此处虽有涉及但不会详细展开，需要全面深入了解的读者可以根据文中提示参看手册中的相关章节。

2.1.1 数据类型

基础整数指令操作的数据类型有 5 种，分别是：比特（bit，简记 b）、字节（Byte，简记 B，长度 8b）、半字（Halfword，简记 H，长度 16b）、字（Word，简记 W，长度 32b）、双字（Doubleword，简记 D，长度 64b）。在 LA32 架构下，没有操作双字的整数指令。

字节、半字、字和双字数据类型均采用二进制补码的编码方式。

2.1.2 寄存器

基础整数指令涉及的寄存器包括通用寄存器（General-purpose Register，简称 GR）和程序计数器（Program Couner，简称 PC），如图 2-1 所示。

[1] 应用软件指那些不能直接操作架构中特权资源的软件。在 Linux 操作系统中，指那些运行在 user mode 下的软件。

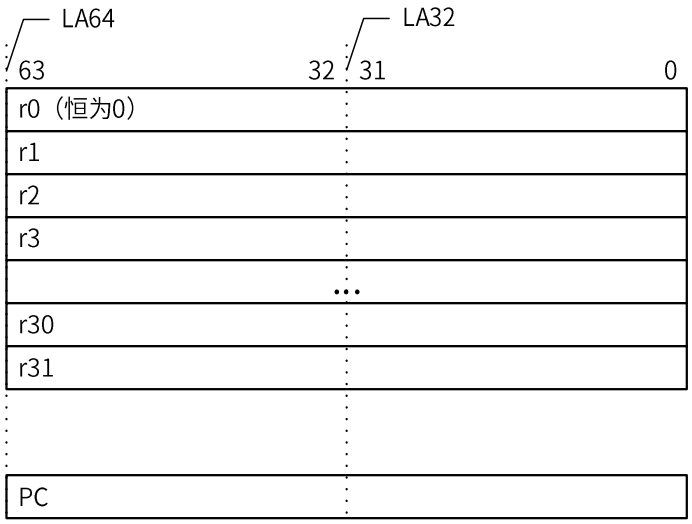


图 2-1 通用寄存器和 PC

2.1.2.1 通用寄存器

通用寄存器 GR 有 32 个，记为 r0~r31，其中第 0 号寄存器 r0 的值恒为 0。GR 的位宽记作 GRLEN。LA32 架构下 GR 的位宽是 32 比特，LA64 架构下 GR 的位宽是 64 比特。基础整数指令与通用寄存器存在正交关系。即从架构角度而言，这些指令中任一个寄存器操作数都可以采用 32 个 GR 中的任一个。唯一的例外是 BL 指令中隐含的目的寄存器一定是第 1 号寄存器 r1。在标准的龙芯架构应用程序二进制接口（Application Binary Interface，简称 ABI）中，r1 固定作为存放函数调用返回地址的寄存器。

2.1.2.2 PC

PC 只有 1 个，记录着当前指令的地址。PC 寄存器不能被指令直接修改，它只能被转移指令、例外陷入和例外返回指令间接修改。不过，PC 寄存器可以作为一些非转移类指令的源操作数而被直接读取。PC 的宽度总是与 GR 的宽度一致。

2.1.3 运行特权等级

龙芯架构定义了 4 个运行特权等级（Privilege LeVel，简称 PLV），分别是 PLV0~PLV3。应用软件应运行在 PLV1~PLV3 这三个非特权的等级上，从而与运行在 PLV0 级上的操作系统等系统软件隔离开。应用软件具体运行在哪个特权等级上是由系统软件在运行时决定的，应用软件对此无法确切感知。龙芯架构下，应用软件通常运行在 PLV3 级上。有关特权等级的更多信息请参看 4.1 节内容。

2.1.3.1 应用软件可访问的特权资源

通常而言特权资源不可以被运行在非特权等级的应用软件直接访问，但是当 CSR.MISC 中的 RPCNTL1/RPCNTL2/RPCNTL3 配置为 1 时，可以在 PLV1/PLV2/PLV3 特权等级下执行 CSRRD 指令读取性能监测计数器。有关性能监测计数器更多信息请参看 7.8 节内容。

2.1.3.2 部分非特权功能的禁用

部分上电复位后默认开启的非特权功能可以由系统软件在运行过程中予以禁用。通过将 CSR.MISC 中的 DRDTL1/DRDTL2/DRDTL3 位置 1，可以禁止在 PLV1/PLV2/PLV3 级下执行 RDTIME 类指令，违例将触发指令特权等级错例外（IPE）。

2.1.4 例外和中断

例外（Exception）和中断（Interrupt）会打断当前正在执行的应用程序，将程序执行流切换到例外/中断处理程序的入口处开始执行。其中例外由指令在执行过程中发生的异常情况引发，而中断则由外部事件（如中断输入信号）引发。在本架构参考手册中，我们将严格区分“产生例外/中断”和“触发例外/中断”两个概念，两者的区别在于前者未必引发执行流的改变而后者一定改变当前执行流转移到例外/中断处理程序入口处。

例外和中断的处理规范属于架构中特权资源处理部分的内容。这里主要对应用软件可以感知到的例外[1]进行一些简要的介绍。

- 系统调用例外：执行 SYSCALL 指令将确定地立刻触发系统调用例外（SYS）。
- 断点例外：执行 BREAK 指令将确定地立刻触发断点例外（BRK）。
- 指令不存在例外：所执行的指令编码在架构中未定义，或者架构规范定义在当前上下文中该指令视作不存在，那么将立刻触发指令不存在例外（INE）。
- 特权指令错例外：除了 2.1.3 节中所列的特殊情况之外，应用软件中执行一条特权指令将确定地立刻触发指令特权等级错例外（IPE）。
- 地址错例外：当程序有功能错误导致取指或访存指令的地址出现了非法的情况，如取指地址不是 4 字节边界对齐，访问了非法的地址空间等，此时将触发取指地址错例外（ADEF）或访存指令地址错例外（ADEM）。
- 浮点错例外：当浮点数指令执行过程中，数据出现异常情况需要特殊处理，可以产生或触发基础浮点错例外（FPE）。更多信息可参看 3.1.4 节中的内容。

2.1.5 内存地址空间

这里仅涉及应用软件可见的虚拟内存地址空间。虚拟内存地址到物理内存地址的翻译由运行时环境决定，这些内容涉及架构中特权资源的相关规范，将在本手册的后半部分介绍。

龙芯架构下内存地址空间是一个字节寻址的线性连续地址空间。

在 LA32 架构下，应用软件能够访问的内存地址空间范围是：$0\sim2^{31}-1$。

在 LA64 架构下，应用软件能够访问的内存地址空间范围是：$0\sim2^{VALEN-1}-1$。这里 VALEN 理论上是一个小于等于 64 的整数，由实现决定其具体的值。常见的 VALEN 在[40, 48]范围内。应用软件可以通过执行

[1] 在龙芯架构中，中断对于应用软件总是不可见的。

CPUCFG 指令读取 0x1 号配置字的 VALEN 域来确定 VALEN 的具体值。

当应用软件中取指或访存指令的虚地址超过了上述范围，且无法通过直接映射配置窗口完成合法映射（见 5.2.1 节），将触发取指地址错例外（ADEF）或访存指令地址错例外（ADEM）。

2.1.6 尾端

龙芯架构只采用小尾端的存储方式。

2.1.7 存储访问类型

龙芯架构下支持三种存储访问类型，分别是：一致可缓存（Coherent Cached，简称 CC）、强序非缓存（Strongly-ordered UnCached，简称 SUC）和弱序非缓存（Weakly-ordered UnCached，简称 WUC）。存储访问类型与访存虚拟地址绑定，通过页表项中的 MAT（Memory Access Type）域决定[1]。MAT 域的值域存储访问类型的对应关系是：0——强序非缓存，1——一致可缓存，2——弱序非缓存，3——保留。存储访问类型的设置过程对于应用软件是透明的。

采用一致可缓存访问类型访问时，所访问的对象既可以是最终存储对象也可以是处理器中维护有缓存一致性的缓存。通常采用这种访问类型访问内存以获得高性能。

采用强序非缓存或弱序非缓存类型访问时，只能直接访问最终存储对象。两者的区别在于：强序非缓存访问满足顺序一致性，即所有访问严格按照程序中的次序执行且当前访存操作彻底完成前不能开始执行下一个访存操作；而弱序非缓存的读访问允许推测执行，弱序非缓存的写数据可以在处理器核内部合并至更大的规模（如一个 Cache 行）后以突发（Burst）方式写出，合并过程中后面的写数据可以覆盖前面写数据。

龙芯架构下只要求强序非缓存类型的访存指令不能有副作用（Side Effect），即此类指令不可推测的执行。软件可以利用这一特性通过强序非缓存类型的访存指令来访问系统中的 I/O 设备。但是，龙芯架构允许强序非缓存类型的取指操作具有副作用。这是指，访问类型是强序非缓存类型的取指操作，即使它源自转移预测的结果，也允许执行。为避免此类推测执行所产生的核外访存操作误入非法的物理地址空间，需要在片上网络中过滤掉存在风险的访问。

弱序非缓存类型的访问通常用于加速非缓存的内存数据的访问，如显存数据。

2.1.7.1 指令 Cache 的缓存一致性维护

某个处理器核的指令 Cache 与其它处理器核内的 Cache 或缓存一致输入输出主设备（Cache Coherent I/O Master）之间的缓存一致性必须由硬件维护。

处理器核内部指令 Cache 与数据 Cache 之间的缓存一致性维护可以实现为硬件维护。这意味着对于自修改代码，软件不需要通过 Cache 维护指令来保证同一个核内部指令 Cache 与数据 Cache 之间的缓存

[1] 这里只是针对应用软件所涉及的范围。对于系统软件，其在直接地址翻译模式下，或是映射地址翻译模式下地址落在直接映射窗口所配置的地址范围内，其存储访问类型由指定的控制状态寄存器进行配置。

一致性。不过，由于流水线结构和推测取指行为的存在，软件仍需要使用 IBAR 指令来确保取指一定能够看到 store 指令的执行效果。

2.1.8 非对齐存储访问

所有取指操作的访存地址必须 4 字节边界对齐，否则将触发取指地址错例外（ADEF）。

除了原子访存指令、整数边界检查访存指令和浮点数边界检查访存指令外，其余的 load/store 访存指令可以实现为允许访存地址不对齐。不过，在一个允许访存地址不对齐的实现中，系统态软件可以通过配置 CSR.MISC 中的 ALCL0~ALCL3 控制位，在 PLV0~PLV3 特权等级下，对这些 load/store 访存指令也进行地址对齐检查。对于需要进行地址对齐检查的访存指令，如果其访问的地址不是自然对齐[1]的，将触发地址非对齐例外（ALE）。

2.1.9 存储一致性模型简述

龙芯架构的存储一致性模型采用弱一致性（Weakly Consistency，简称 WC）模型。本小节仅对架构所采用的弱一致性模型做一个简要描述。

在弱一致性模型中，同步操作和普通访存需要区分开来，程序员必须用架构所定义的同步操作把对于写共享单元的访问保护起来，以保证多个处理器核对于写共享单元的访问是互斥的。对访存事件发生次序做如下限制：

1. 同步操作的执行满足顺序一致性条件。即同步操作在所有处理器核中都严格按照其在程序中出现的次序执行，且在当前同步操作彻底完成之前不能开始执行下一个同步操作。
2. 在任一普通访存操作允许被执行之前，所有在同一处理器核中先于这一访存操作的同步操作都已经完成；
3. 在任一同步操作允许被执行之前，所有在同一处理机中先于这一同步操作的普通访存操作都已完成。

龙芯架构中能够产生同步操作的指令有 DBAR、IBAR、带有 DBAR 功能的 AM 原子访存指令以及 LL-SC 指令对。

2.2 基础整数指令概述

这一节将对 LA64 架构中应用级基础整数指令的功能进行描述。对于 LA32 架构，其只需要实现其中的一个子集，该子集所含指令列表见表 2-1。由于在 LA32 架构下 GR 的位宽只有 32 位，所以后续指令描述中“将 32 位结果符号扩展后写入到通用寄存器 rd 中”中的符号扩展操作并不需要。

[1]所谓自然对齐是指，访问半字对象时地址是 2 字节边界对齐，访问字对象时地址是 4 字节边界对齐，访问双字对象时地址是 8 字节边界对齐，访问 128 位向量对象时地址是 16 字节边界对齐，访问 256 位向量对象时地址是 32 字节边界对齐。。

表 2-1 LA32 应用级基础整数指令一览

算术运算类指令	ADD.W, SUB.W, ADDI.W, ALSL.W, LU12I.W, SLT, SLTU, SLTI, SLTUI, PCADDI, PCADDU12I, PCALAU12I, AND, OR, NOR, XOR, ANDN, ORN, ANDI, ORI, XORI, MUL.W, MULH.W, MULH.WU, DIV.W, MOD.W, DIV.WU, MOD.WU
移位运算类指令	SLL.W, SRL.W, SRA.W, ROTR.W, SLLI.W, SRLI.W, SRAI.W, ROTRI.W
位操作指令	EXT.W.B, EXT.W.H, CLO.W, CLZ.W, CTO.W, CTZ.W, BYTEPICK.W, REVB.2H, BITREV.4B, BITREV.W, BSTRINS.W, BSTRPICK.W, MASKEQZ, MASKNEZ
转移指令	BEQ, BNE, BLT, BGE, BLTU, BGEU, BEQZ, BNEZ, B, BL, JIRL
访存指令	LD.B, LD.H, LD.W, LD.BU, LD.HU, ST.B, ST.H, ST.W, PRELD
原子访存指令	LL.W, SC.W
栅障指令	DBAR, IBAR
其它杂项指令	SYSCALL, BREAK, RDTIMEL.W, RDTIMEH.W, CPUCFG

此外，对于那些操作对象的数据位宽是 GR 宽度的指令，其在 LA32 架构下操作位宽为 32 位，在 LA64 架构下操作位宽为 64 位。除非特殊情况，指令功能描述中将不再特别说明。

2.2.1 算术运算类指令

2.2.1.1 ADD.{W/D}, SUB.{W/D}

指令格式：

add.w	rd, rj, rk		add.d	rd, rj, rk
sub.w	rd, rj, rk		sub.d	rd, rj, rk

ADD.W 将通用寄存器 rj 中的[31:0]位数据加上通用寄存器 rk 中的[31:0]位数据，所得结果的[31:0]位符号扩展后写入通用寄存器 rd 中。

ADD.W:

```
tmp = GR[rj][31:0] + GR[rk][31:0]
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

SUB.W 将通用寄存器 rj 中的[31:0]位数据减去通用寄存器 rk 中的[31:0]位数据，所得结果的[31:0]位符号扩展后写入通用寄存器 rd 中。

SUB.W:

```
tmp = GR[rj][31:0] - GR[rk][31:0]
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

ADD.D 将通用寄存器 rj 中的[63:0]位数据加上通用寄存器 rk 中的[63:0]位数据，所得结果写入通用寄存器 rd 中。

ADD.D:

```
tmp = GR[rj][63:0] + GR[rk][63:0]
GR[rd] = tmp[63:0]
```

SUB.D 将通用寄存器 rj 中的[63:0]位数据减去通用寄存器 rk 中的[63:0]位数据，所得结果写入通用寄存器 rd 中。

SUB.D:

```
tmp = GR[rj][63:0] - GR[rk][63:0]
GR[rd] = tmp[63:0]
```

上述指令执行时不对溢出情况做任何特殊处理。

2.2.1.2 ADDI.{W/D}, ADDU16I.D

指令格式：

addi.w	rd, rj, si12	addi.d	rd, rj, si12
		addu16i.d	rd, rj, si16

ADDI.W 将通用寄存器 rj 中的[31:0]位数据加上 12 比特立即数 si12 符号扩展后的 32 位数据，所得结果的[31:0]位符号扩展后写入通用寄存器 rd 中。

ADDI.W:

```
tmp = GR[rj][31:0] + SignExtend(si12, 32)
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

ADDI.D 将通用寄存器 rj 中的[63:0]位数据加上 12 比特立即数 si12 符号扩展后的 64 位数据，所得结果写入通用寄存器 rd 中。

ADDI.D:

```
tmp = GR[rj][63:0] + SignExtend(si12, 64)
GR[rd] = tmp[63:0]
```

ADDU16I.D 将 16 比特立即数 si16 逻辑左移 16 位后再符号扩展，所得数据加上通用寄存器 rj 中的[63:0]位数据，相加结果写入通用寄存器 rd 中。ADDU16I.D 指令与 LDPTR.W/D、STPTR.W/D 指令配合使用，用于加速位置无关代码中基于 GOT 表的访问。

ADDU16I.D:

```
tmp = GR[rj][63:0] + SignExtend({si16, 16'b0}, 64)
GR[rd] = tmp[63:0]
```

上述指令执行时不对溢出情况做任何特殊处理。

2.2.1.3 ALSL.{W[U]/D}

指令格式：

alsl.w	rd, rj, rk, sa2	alsl.d	rd, rj, rk, sa2
alsl.wu	rd, rj, rk, sa2		

ALSL.W 将通用寄存器 rj 中的[31:0]位数据逻辑左移(sa2+1)位后加上通用寄存器 rk 中的[31:0]位数据，所得结果的[31:0]位符号扩展后写入通用寄存器 rd 中。

ALSL.W:

```
tmp = (GR[rj][31:0]<<(sa2+1)) + GR[rk][31:0]
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

ALSL.WU 将通用寄存器 rj 中的[31:0]位数据逻辑左移(sa2+1)位后加上通用寄存器 rk 中的[31:0]位数据，所得结果的[31:0]位零扩展后写入通用寄存器 rd 中。

ALSL.WU:

```
tmp = (GR[rj][31:0]<<(sa2+1)) + GR[rk][31:0]
GR[rd] = ZeroExtend(tmp[31:0], GRLEN)
```

ALSL.D 将通用寄存器 rj 中的[63:0]位数据逻辑左移(sa2+1)位后加上通用寄存器 rk 中的[63:0]位数据，所得结果写入通用寄存器 rd 中。

ALSL.D:

```
tmp = (GR[rj][63:0]<<(sa2+1)) + GR[rk][63:0]
GR[rd] = tmp[63:0]
```

上述指令执行时不对溢出情况做任何特殊处理。

需要注意的是，上述指令的汇编表示中的立即数应填入(sa2+1)，即实际的移位值而非指令码中立即数域的值。

2.2.1.4 LU12I.W, LU32I.D, LU52I.D

指令格式：

lu12i.w	rd, si20	lu32i.d	rd, si20
		lu52i.d	rd, rj, si12

LU12I.W 将 20 比特立即数 si20 最低位连接上 12 比特 0，然后符号扩展后写入通用寄存器 rd 中。

LU12I.W:

```
GR[rd] = SignExtend({si20, 12'b0}, GRLEN)
```

LU32I.D 将 20 比特立即数 si20 符号扩展后的数据最低位连接上通用寄存器 rd 中[31:0]位数据，结果写入到通用寄存器 rd 中。

LU32I.D:

```
GR[rd] = {SignExtend(si20, 32), GR[rd][31:0]}
```

LU52I.D 将 12 比特立即数 si12 符号扩展后的数据最低位连接上通用寄存器 rj 中[51:0]位数据，结果写入到通用寄存器 rd 中。

LU52I.D:

```
GR[rd] = {si12, GR[rj][51:0]}
```

上述指令与 ORI 指令一起，用于将超过 12 位的立即数装载到通用寄存器中。

2.2.1.5 SLT[U]

指令格式： slt rd, rj, rk

sltu rd, rj, rk

SLT 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据视作有符号整数进行大小比较，如果前者小于后者，则将通用寄存器 rd 的值置为 1，否则置为 0。

SLT:

```
GR[rd] = (signed(GR[rj]) < signed(GR[rk])) ? 1 : 0
```

SLTU 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据视作无符号整数进行大小比较，如果前者小于后者，则将通用寄存器 rd 的值置为 1，否则置为 0。

SLTU:

```
GR[rd] = (unsigned(GR[rj]) < unsigned(GR[rk])) ? 1 : 0
```

SLT 和 SLTU 比较的数据位宽与所执行机器的通用寄存器的位宽一致。

2.2.1.6 SLT[U]I

指令格式： slti rd, rj, si12

sltui rd, rj, si12

SLTI 将通用寄存器 rj 中数据与 12 比特立即数 si12 符号扩展后所得的数据视作有符号整数进行大小比较，如果前者小于后者，则将通用寄存器 rd 的值置为 1，否则置为 0。

SLTI:

```
tmp = SignExtend(si12, GRLEN)
GR[rd] = (signed(GR[rj]) < signed(tmp)) ? 1 : 0
```

SLTUI 将通用寄存器 rj 中数据与 12 比特立即数 si12 符号扩展后所得的数据视作无符号整数进行大小比较，如果前者小于后者，则将通用寄存器 rd 的值置为 1，否则置为 0。

SLTUI:

```
tmp = SignExtend(si12, GRLEN)
GR[rd] = (unsigned(GR[rj]) < unsigned(tmp)) ? 1 : 0
```

SLTI 和 SLTUI 比较的数据位宽与所执行机器的通用寄存器的位宽一致。

请注意，对于 SLTUI 指令，立即数仍是符号扩展。

2.2.1.7 PCADDI, PCADDU12I, PCADDU18I, PCALAU12I

指令格式： pcaddi rd, si20

pcaddu12i rd, si20

pcaddu18i rd, si20

pcalau12i rd, si20

PCADDI 将 20 比特立即数 si20 最低位连接上 2 比特 0 之后符号扩展，所得数据加上该指令的 PC，相加结果写入通用寄存器 rd 中。

PCADDI:

```
GR[rd] = PC + SignExtend({si20, 2'b0}, GRLEN)
```

PCADDU12I 将 20 比特立即数 si20 最低位连接上 12 比特 0 之后符号扩展，所得数据加上该指令的 PC，相加结果写入通用寄存器 rd 中。

PCADDU12I:

```
GR[rd] = PC + SignExtend({si20, 12'b0}, GRLEN)
```

PCADDU18I 将 20 比特立即数 si20 最低位连接上 18 比特 0 之后符号扩展，所得数据加上该指令的 PC，相加结果写入通用寄存器 rd 中。

PCADDU18I:

```
GR[rd] = PC + SignExtend({si20, 18'b0}, GRLEN)
```

PCALAU12I 将 20 比特立即数 si20 最低位连接上 12 比特 0 之后符号扩展，所得数据加上该指令的 PC，相加结果最低 12 位抹 0 后写入通用寄存器 rd 中。

PCALAU12I:

```
tmp = PC + SignExtend({si20, 12'b0}, GRLEN)
GR[rd] = {tmp[GRLEN-1:12], 12'b0}
```

上述指令操作的数据位宽与所执行机器的通用寄存器的位宽一致。

2.2.1.8 AND, OR, NOR, XOR, ANDN, ORN

指令格式：

and	rd, rj, rk
or	rd, rj, rk
nor	rd, rj, rk
xor	rd, rj, rk
andn	rd, rj, rk
orn	rd, rj, rk

AND 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据进行按位逻辑与运算，结果写入通用寄存器 rd 中。

AND:

```
GR[rd] = GR[rj] & GR[rk]
```

OR 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据进行按位逻辑或运算，结果写入通用寄存器 rd 中。

OR:

```
GR[rd] = GR[rj] | GR[rk]
```

NOR 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据进行按位逻辑或非运算，结果写入通用寄存器 rd 中。

NOR:

```
GR[rd] = ~(GR[rj] | GR[rk])
```

XOR 将通用寄存器 rj 中数据与通用寄存器 rk 中的数据进行按位逻辑异或运算，结果写入通用寄存器 rd 中。

XOR:

```
GR[rd] = GR[rj] ^ GR[rk]
```

ANDN 将通用寄存器 rk 中的数据按位取反后再与通用寄存器 rj 中数据进行按位逻辑与运算，结果写入通用寄存器 rd 中。

ANDN:

```
GR[rd] = GR[rj] & (~GR[rk])
```

ORN 将通用寄存器 rk 中的数据按位取反后再与通用寄存器 rj 中数据进行按位逻辑或运算，结果写入通用寄存器 rd 中。

ORN:

```
GR[rd] = GR[rj] | (~GR[rk])
```

上述指令操作的数据位宽与所执行机器的通用寄存器的位宽一致。

2.2.1.9 ANDI, ORI, XORI

指令格式：

andi	rd, rj, ui12
ori	rd, rj, ui12
xori	rd, rj, ui12

ANDI 将通用寄存器 rj 中数据与 12 比特立即数零扩展之后的数据进行按位逻辑与运算，结果写入通用寄存器 rd 中。

ANDI:

```
GR[rd] = GR[rj] & ZeroExtend(ui12, GRLEN)
```

ORI 将通用寄存器 rj 中数据与 12 比特立即数零扩展之后的数据进行按位逻辑或运算，结果写入通用寄存器 rd 中。

ORI:

```
GR[rd] = GR[rj] | ZeroExtend(ui12, GRLEN)
```

XORI 将通用寄存器 rj 中数据与 12 比特立即数零扩展之后的数据进行按位逻辑异或运算，结果写入通用寄存器 rd 中。

XORI:

```
GR[rd] = GR[rj] ^ ZeroExtend(ui12, GRLEN)
```

上述指令操作的数据位宽与所执行机器的通用寄存器的位宽一致。

2.2.1.10 NOP

NOP 指令是指令“andi r0, r0, 0”的别名。其功用仅为占据 4 字节的指令码位置并将 PC 加 4，除此之外不会改变其它任何软件可见的处理器状态。

2.2.1.11 MUL.{W/D}, MULH.{W[U]/D[U]}

指令格式：

mul.w	rd, rj, rk
mulh.w	rd, rj, rk

mulh.wu rd, rj, rk

mul.d rd, rj, rk

mulh.d rd, rj, rk

mulh.du rd, rj, rk

MUL.W 将通用寄存器 rj 中[31:0]位数据与通用寄存器 rk 中[31:0]位数据进行相乘，乘积结果的[31:0]位数据符号扩展后写入通用寄存器 rd 中。

MUL.W:

```
product = signed(GR[rj][31:0]) * signed(GR[rk][31:0])
GR[rd] = SignExtend(product[31:0], GRLEN)
```

MULH.W 将通用寄存器 rj 中[31:0]位数据与通用寄存器 rk 中[31:0]位数据视作有符号数进行相乘，乘积结果的[63:32]位数据符号扩展后写入通用寄存器 rd 中。

MULH.W:

```
product = signed(GR[rj][31:0]) * signed(GR[rk][31:0])
GR[rd] = SignExtend(product[63:32], GRLEN)
```

MULH.WU 将通用寄存器 rj 中[31:0]位数据与通用寄存器 rk 中[31:0]位数据视作无符号数进行相乘，乘积结果的[63:32]位数据符号扩展后写入通用寄存器 rd 中。

MULH.WU:

```
product = unsigned(GR[rj][31:0]) * unsigned(GR[rk][31:0])
GR[rd] = SignExtend(product[63:32], GRLEN)
```

MUL.D 将通用寄存器 rj 中[63:0]位数据与通用寄存器 rk 中[63:0]位数据进行相乘，乘积结果的[63:0]位数据写入通用寄存器 rd 中。

MUL.D:

```
product = signed(GR[rj][63:0]) * signed(GR[rk][63:0])
GR[rd] = product[63:0]
```

MULH.D 将通用寄存器 rj 中[63:0]位数据与通用寄存器 rk 中[63:0]位数据视作有符号数进行相乘，乘积结果的[127:64]位数据写入通用寄存器 rd 中。

MULH.D:

```
product = signed(GR[rj][63:0]) * signed(GR[rk][63:0])
GR[rd] = product[127:64]
```

MULH.DU 将通用寄存器 rj 中[63:0]位数据与通用寄存器 rk 中[63:0]位数据视作无符号数进行相乘，乘积结果的[127:64]位数据写入通用寄存器 rd 中。

MULH.DU:

```
product = unsigned(GR[rj][63:0]) * unsigned(GR[rk][63:0])
GR[rd] = product[127:64]
```

2.2.1.12 MULW.D.W[U]

指令格式： mulw.d.w rd, rj, rk

mulw.d.wu rd, rj, rk

MULW.D.W 将通用寄存器 rj 中[31:0]位数据与通用寄存器 rk 中[31:0]位数据视作有符号数进行相乘，64 位的乘积结果写入通用寄存器 rd 中。

MULW.D.W:

```
product = signed(GR[rj][31:0]) * signed(GR[rk][31:0])
GR[rd] = product[63:0]
```

MULW.D.WU 将通用寄存器 rj 中[31:0]位数据与通用寄存器 rk 中[31:0]位数据视作无符号数进行相乘，64 位的乘积结果写入通用寄存器 rd 中。

MULW.D.WU:

```
product = unsigned(GR[rj][31:0]) * unsigned(GR[rk][31:0])
GR[rd] = product[63:0]
```

2.2.1.13 DIV.{W[U]/D[U]}, MOD.{W[U]/D[U]}

指令格式：

	div.w	rd, rj, rk
	mod.w	rd, rj, rk
	div.wu	rd, rj, rk
	mod.wu	rd, rj, rk
	div.d	rd, rj, rk
	mod.d	rd, rj, rk
	div.du	rd, rj, rk
	mod.du	rd, rj, rk

DIV.W 和 DIV.WU 将通用寄存器 rj 中[31:0]位数据除以通用寄存器 rk 中[31:0]位数据，所得的商符号扩展后写入通用寄存器 rd 中。

DIV.W:

```
quotient = signed(GR[rj][31:0]) / signed(GR[rk][31:0])
GR[rd] = SignExtend(quotient[31:0], GRLEN)
```

DIV.WU:

```
quotient = unsigned(GR[rj][31:0]) / unsigned(GR[rk][31:0])
GR[rd] = SignExtend(quotient[31:0], GRLEN)
```

MOD.W 和 MOD.WU 将通用寄存器 rj 中[31:0]位数据除以通用寄存器 rk 中[31:0]位数据，所得的余数符号扩展后写入通用寄存器 rd 中。

MOD.W:

```
remainder = signed(GR[rj][31:0]) % signed(GR[rk][31:0])
GR[rd] = SignExtend(remainder[31:0], GRLEN)
```

MOD.WU:

```
remainder = unsigned(GR[rj][31:0]) % unsigned(GR[rk][31:0])
GR[rd] = SignExtend(remainder[31:0], GRLEN)
```

在 LoongArch64 位兼容的机器上，执行 DIV.W[U]和 MOD.W[U]指令时，如果通用寄存器 rj 或 rk 中的数值超过了 32 位有符号数的数值范围，则指令执行结果可以为无意义的任意值。

DIV.D 和 DIV.DU 将通用寄存器 rj 中[63:0]位数据除以通用寄存器 rk 中[63:0]位数据，所得的商写入通用寄存器 rd 中。

DIV.D:

```
GR[rd] = signed(GR[rj][63:0]) / signed(GR[rk][63:0])
```

DIV.DU:

```
GR[rd] = unsigned(GR[rj][63:0]) / unsigned(GR[rk][63:0])
```

MOD.D 和 MOD.DU 将通用寄存器 rj 中[63:0]位数据除以通用寄存器 rk 中[63:0]位数据，所得的余数写入通用寄存器 rd 中。

MOD.D:

```
GR[rd] = signed(GR[rj][63:0]) % signed(GR[rk][63:0])
```

MOD.DU:

```
GR[rd] = unsigned(GR[rj][63:0]) % unsigned(GR[rk][63:0])
```

DIV.W、MOD.W、DIV.D 和 MOD.D 进行除法操作时，操作数均视作有符号数。DIV.WU、MOD.WU、DIV.DU 和 MOD.DU 进行除法操作时，源操作数均视作无符号数。

每一对求商/余数的指令对 DIV.W/MOD.W，DIV.WU/MOD.WU，DIV.D/MOD.D，DIV.DU/MOD.DU 运算的结果满足，余数与被除数的符号一致且余数的绝对值小于除数的绝对值。

当除数是 0 时，结果可以为任意值，但不会因此触发任何例外。

2.2.2 移位运算类指令

2.2.2.1 SLL.W, SRL.W, SRA.W, ROTR.W

指令格式：

	sll.w	rd, rj, rk
	srl.w	rd, rj, rk
	sra.w	rd, rj, rk
	rotr.w	rd, rj, rk

SLL.W 将通用寄存器 rj 中[31:0]位数据逻辑左移，移位结果符号扩展写入通用寄存器 rd 中。

SLL.W:

```
tmp = SLL(GR[rj][31:0], GR[rk][4:0])
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

SRL.W 将通用寄存器 rj 中[31:0]位数据逻辑右移，移位结果符号扩展写入通用寄存器 rd 中。

SRL.W:

```
tmp = SRL(GR[rj][31:0], GR[rk][4:0])
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

SRA.W 将通用寄存器 rj 中[31:0]位数据算术右移，移位结果符号扩展写入通用寄存器 rd 中。

SRA.W:

```
tmp = SRA(GR[rj][31:0], GR[rk][4:0])
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

ROTR.W 将通用寄存器 rj 中[31:0]位数据循环右移，移位结果符号扩展写入通用寄存器 rd 中。

ROTR.W:

```
tmp = ROTR(GR[rj][31:0], GR[rk][4:0])
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

上述移位指令的移位量均是通用寄存器 rk 中[4:0]位数据，且视作无符号数。

2.2.2.2 SLLI.W, SRLI.W, SRAI.W, ROTRI.W

指令格式：

```
slli.w    rd, rj, ui5
srli.w    rd, rj, ui5
srai.w    rd, rj, ui5
rotri.w   rd, rj, ui5
```

SLLI.W 将通用寄存器 rj 中[31:0]位数据逻辑左移，移位结果符号扩展写入通用寄存器 rd 中。

SLLI.W:

```
tmp = SLL(GR[rj][31:0], ui5)
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

SRLI.W 将通用寄存器 rj 中[31:0]位数据逻辑右移，移位结果符号扩展写入通用寄存器 rd 中。

SRLI.W:

```
tmp = SRL(GR[rj][31:0], ui5)
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

SRAI.W 将通用寄存器 rj 中[31:0]位数据算术右移，移位结果符号扩展写入通用寄存器 rd 中。

SRAI.W:

```
tmp = SRA(GR[rj][31:0], ui5)
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

ROTRI.W 将通用寄存器 rj 中[31:0]位数据循环右移，移位结果符号扩展写入通用寄存器 rd 中。

ROTRI.W:

```
tmp = ROTR(GR[rj][31:0], ui5)
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

上述移位指令的移位量均是指令码中 5 比特无符号立即数 ui5。

2.2.2.3 SLL.D, SRL.D, SRA.D, ROTR.D

指令格式：

```
sll.d     rd, rj, rk
srl.d     rd, rj, rk
sra.d     rd, rj, rk
rotr.d    rd, rj, rk
```

SLL.D 将通用寄存器 rj 中[63:0]位数据逻辑左移，移位结果写入通用寄存器 rd 中。

SLL.D:

```
GR[rd] = SLL(GR[rj][63:0], GR[rk][5:0])
```

SRL.D 将通用寄存器 rj 中[63:0]位数据逻辑右移，移位结果写入通用寄存器 rd 中。

SRL.D:

```
GR[rd] =SRL(GR[rj][63:0], GR[rk][5:0])
```

SRA.D 将通用寄存器 rj 中[63:0]位数据算术右移，移位结果写入通用寄存器 rd 中。

SRA.D:

```
GR[rd] = SRA(GR[rj][63:0], GR[rk][5:0])
```

ROTR.D 将通用寄存器 rj 中[63:0]位数据循环右移，移位结果写入通用寄存器 rd 中。

ROTR.D:

```
GR[rd] = ROTR(GR[rj][63:0], GR[rk][5:0])
```

上述移位指令的移位量均是通用寄存器 rk 中[5:0]位数据，且视作无符号数。

2.2.2.4 SLLI.D, SRLI.D, SRAI.D, ROTRI.D

指令格式：

slli.d	rd, rj, ui6
srli.d	rd, rj, ui6
srai.d	rd, rj, ui6
rotri.d	rd, rj, ui6

SLLI.D 将通用寄存器 rj 中[63:0]位数据逻辑左移，移位结果写入通用寄存器 rd 中。

SLLI.D:

```
GR[rd] = SLL(GR[rj][63:0], ui6)
```

SRLI.D 将通用寄存器 rj 中[63:0]位数据逻辑右移，移位结果写入通用寄存器 rd 中。

SRLI.D:

```
GR[rd] =SRL(GR[rj][63:0], ui6)
```

SRAI.D 将通用寄存器 rj 中[63:0]位数据算术右移，移位结果写入通用寄存器 rd 中。

SRAI.D:

```
GR[rd] = SRA(GR[rj][63:0], ui6)
```

ROTRI.D 将通用寄存器 rj 中[63:0]位数据循环右移，移位结果写入通用寄存器 rd 中。

ROTRI.D:

```
GR[rd] = ROTR(GR[rj][63:0], ui6)
```

上述移位指令的移位量均是指令码中 6 比特无符号立即数 ui6。

2.2.3 位操作指令

2.2.3.1 EXT.W.{B/H}

指令格式：

ext.w.b	rd, rj
ext.w.h	rd, rj

EXT.W.B 将通用寄存器 rj 中[7:0]位数据符号扩展后写入通用寄存器 rd 中。

EXT.W.B:

```
GR[rd] = SignExtend(GR[rj][7:0], GRLEN)
```

EXT.W.H 将通用寄存器 rj 中[15:0]位数据符号扩展后写入通用寄存器 rd 中。

EXT.W.H:

```
GR[rd] = SignExtend(GR[rj][15:0], GRLEN)
```

2.2.3.2 CL{O/Z}.{W/D}, CT{O/Z}.{W/D}

指令格式：				
	clo.w	rd, rj	clo.d	rd, rj
	clz.w	rd, rj	clz.d	rd, rj
	cto.w	rd, rj	cto.d	rd, rj
	ctz.w	rd, rj	ctz.d	rd, rj

CLO.W 对于通用寄存器 rj 中[31:0]位数据，从第 31 位开始向第 0 位方向计量连续比特“1”的个数，结果写入通用寄存器 rd 中。

CLO.W:

```
GR[rd] = CLO(GR[rj][31:0])
```

CLZ.W 对于通用寄存器 rj 中[31:0]位数据，从第 31 位开始向第 0 位方向计量连续比特“0”的个数，结果写入通用寄存器 rd 中。

CLZ.W:

```
GR[rd] = CLZ(GR[rj][31:0])
```

CTO.W 对于通用寄存器 rj 中[31:0]位数据，从第 0 位开始向第 31 位方向计量连续比特“1”的个数，结果写入通用寄存器 rd 中。

CTO.W:

```
GR[rd] = CTO(GR[rj][31:0])
```

CTZ.W 对于通用寄存器 rj 中[31:0]位数据，从第 0 位开始向第 31 位方向计量连续比特“0”的个数，结果写入通用寄存器 rd 中。

CTZ.W:

```
GR[rd] = CTZ(GR[rj][31:0])
```

CLO.D 对于通用寄存器 rj 中[63:0]位数据，从第 63 位开始向第 0 位方向计量连续比特“1”的个数，结果写入通用寄存器 rd 中。

CLO.D:

```
GR[rd] = CLO(GR[rj][63:0])
```

CLZ.D 对于通用寄存器 rj 中[63:0]位数据，从第 63 位开始向第 0 位方向计量连续比特“0”的个数，结果写入通用寄存器 rd 中。

CLZ.D:

```
GR[rd] = CLZ(GR[rj][63:0])
```

CTO.D 对于通用寄存器 rj 中[63:0]位数据，从第 0 位开始向第 63 位方向计量连续比特“1”的个数，结果写入通用寄存器 rd 中。

CTO.D:

```
GR[rd] = CTO(GR[rj][63:0])
```

CTZ.D 对于通用寄存器 rj 中[63:0]位数据，从第 0 位开始向第 63 位方向计量连续比特“0”的个数，结果写入通用寄存器 rd 中。

CTZ.D:

```
GR[rd] = CTZ(GR[rj][63:0])
```

2.2.3.3 BYTEPICK.{W/D}

指令格式： bytepick.w rd, rj, rk, sa2 bytepick.d rd, rj, rk, sa3

BYTEPICK.W 将通用寄存器 rk 中[31:0]位与通用寄存器 rj 中[31:0]位左右连接成为一个 64 位（8 个字节）的比特串，从最左侧第 sa2 个字节开始截取连续 4 个字节，所得的 32 位比特串符号扩展后写入通用寄存器 rd 中。

BYTEPICK.W:

```
tmp = {GR[rk][8*(4-sa2):0], GR[rj][31:8*(4-sa2)]}
GR[rd] = SignExtend(tmp[31:0], GRLEN)
```

BYTEPICK.D 将通用寄存器 rk 中[63:0]位与通用寄存器 rj 中[63:0]位左右连接成为一个 128 位(16 个字节）的比特串，从最左侧第 sa3 个字节开始截取连续 8 个字节，所得的 64 位比特串写入通用寄存器 rd 中。

BYTEPICK.D:

```
GR[rd] = {GR[rk][8*(8-sa3):0], GR[rj][63:8*(8-sa3)]}
```

2.2.3.4 REVB.{2H/4H/2W/D}

指令格式： revb.2h rd, rj revb.4h rd, rj
revb.2w rd, rj
revb.d rd, rj

REVB.2H 将通用寄存器 rj 中[15:0]位中的 2 个字节逆序排列形成中间结果的[15:0]位，将通用寄存器 rj 中[31:16]中的 2 个字节逆序排列形成中间结果的[31:16]位，32 位的中间结果符号扩展写入通用寄存器 rd 中。

REVB.2H:

```
tmp0 = {GR[rj][7:0], GR[rj][15:8]}
tmp1 = {GR[rj][23:16], GR[rj][31:24]}
GR[rd] = SignExtend({tmp1, tmp0}, GRLEN)
```

REVB.4H 将通用寄存器 rj 中[15:0]位中的 2 个字节逆序排列写入通用寄存器 rd 的[15:0]位, 将通用寄存器 rj 中[31:16]位中的 2 个字节逆序排列写入通用寄存器 rd 的[31:16]位，将通用寄存器 rj 中[47:32]位中的 2 个字节逆序排列写入通用寄存器 rd 的[47:32]位，将通用寄存器 rj 中[63:48]位中的 2 个字节逆序排列写入通用寄存器 rd 的[63:48]位。

REVB.4H:

```
tmp0 = {GR[rj][7:0], GR[rj][15:8]}
tmp1 = {GR[rj][23:16], GR[rj][31:24]}
tmp2 = {GR[rj][39:32], GR[rj][47:40]}
tmp3 = {GR[rj][55:48], GR[rj][63:56]}
GR[rd] = {tmp3, tmp2, tmp1, tmp0}
```

REVB.2W 将通用寄存器 rj 中[31:0]位中的 4 个字节逆序排列写入通用寄存器 rd 的[31:0]位，将通用寄存器 rj 中[63:32]位中的 4 个字节逆序排列写入通用寄存器 rd 的[63:32]位。

REVB.2W:

```
tmp0 = {GR[rj][7:0], GR[rj][15:8], GR[rj][23:16], GR[rj][31:24]}
tmp1 = {GR[rj][39:32], GR[rj][47:40], GR[rj][55:48], GR[rj][63:56]}
GR[rd] = {tmp1, tmp0}
```

REVB.D 将通用寄存器 rj 中[63:0]位中的 8 个字节逆序排列写入通用寄存器 rd。

REVB.D:

```
GR[rd] = {GR[rj][7:0], GR[rj][15:8], GR[rj][23:16], GR[rj][31:24], GR[rj][3
9:32], GR[rj][47:40], GR[rj][55:48], GR[rj][63:56]}
```

2.2.3.5 REVH.{2W/D}

指令格式：revh.2w rd, rj

revh.d rd, rj

REVH.2W 将通用寄存器 rj 中[31:0]位中的 2 个半字逆序排列写入通用寄存器 rd 的[31:0]位，将通用寄存器 rj 中[63:32]位中的 2 个半字逆序排列写入通用寄存器 rd 的[63:32]位。

REVH.2W:

```
tmp0 = {GR[rj][15:0], GR[rj][31:16]}
tmp1 = {GR[rj][47:32], GR[rj][63:48]}
GR[rd] = {tmp1, tmp0}
```

REVH.D 将通用寄存器 rj 中[63:0]位中的 4 个半字逆序排列写入通用寄存器 rd。

REVH.D:

```
GR[rd] = { GR[rj][15:0], GR[rj][31:16], GR[rj][47:32], GR[rj][63:48]}
```

2.2.3.6 BITREV.{4B/8B}

指令格式：bitrev.4b rd, rj bitrev.8b rd, rj

BITREV.4B 将通用寄存器 rj 中[7:0]位中的 8 个比特逆序排列形成中间结果的[7:0]位，将通用寄存器 rj 中[15:8]位中的 8 个比特逆序排列形成中间结果的[15:8]位，将通用寄存器 rj 中[23:16]位中的 8 个比特逆序排列形成中间结果的[23:16]位，将通用寄存器 rj 中[31:24]位中的 8 个比特逆序排列形成中间结果的[31:24]位，32 位的中间结果符号扩展写入通用寄存器 rd 中。

BITREV.4B:

```
bstr32[31:24] = BITREV(GR[rj][31:24])
bstr32[23:16] = BITREV(GR[rj][23:16])
bstr32[15: 8] = BITREV(GR[rj][15: 8])
bstr32[ 7: 0] = BITREV(GR[rj][ 7: 0])
```

```
GR[rd] = SignExtend(bstr32, GRLEN)
```

BITREV.8B 将通用寄存器 rj 中[7:0]位中的 8 个比特逆序排列写入通用寄存器 rd 的[7:0]位，将通用寄存器 rj 中[15:8]位中的 8 个比特逆序排列写入通用寄存器 rd 的[15:8]位，将通用寄存器 rj 中[23:16]位中的 8 个比特逆序排列写入通用寄存器 rd 的[23:16]位，将通用寄存器 rj 中[31:24]位中的 8 个比特逆序排列写入通用寄存器 rd 的[31:24]位，将通用寄存器 rj 中[39:32]位中的 8 个比特逆序排列写入通用寄存器 rd 的[39:32]位，将通用寄存器 rj 中[47:40]位中的 8 个比特逆序排列写入通用寄存器 rd 的[47:40]位，将通用寄存器 rj 中[55:48]位中的 8 个比特逆序排列写入通用寄存器 rd 的[55:48]位，将通用寄存器 rj 中[63:56]位中的 8 个比特逆序排列写入通用寄存器 rd 的[63:56]位。

BITREV.8B:

```
GR[rd][63:56] = BITREV(GR[rj][63:56])
GR[rd][55:48] = BITREV(GR[rj][55:48])
GR[rd][47:40] = BITREV(GR[rj][47:40])
GR[rd][39:32] = BITREV(GR[rj][39:32])
GR[rd][31:24] = BITREV(GR[rj][31:24])
GR[rd][23:16] = BITREV(GR[rj][23:16])
GR[rd][15: 8] = BITREV(GR[rj][15: 8])
GR[rd][ 7: 0] = BITREV(GR[rj][ 7: 0])
```

2.2.3.7 BITREV.{W/D}

指令格式： bitrev.w rd, rj bitrev.d rd, rj

BITREV.W 将通用寄存器 rj 中[31:0]位中的 32 个比特逆序排列形成中间结果的[31:0]位，32 位的中间结果符号扩展写入通用寄存器 rd 中。

BITREV.W:

```
bstr32[31:0] = BITREV(GR[rj][31:0])
GR[rd] = SignExtend(bstr32, GRLEN)
```

BITREV.D 将通用寄存器 rj 中[63:0]位中的 64 个比特逆序排列后写入通用寄存器 rd 中。

BITREV.D:

```
GR[rd] = BITREV(GR[rj][63:0])
```

2.2.3.8 BSTRINS.{W/D}

指令格式： bstrins.w rd, rj, msbw, lsbw bstrins.d rd, rj, msbd, lsbd

BSTRINS.W 将通用寄存器 rd 最低 32 位中的[msbw:lsbw]位替换为通用寄存器 rj 中[msbw-lsbw:0]位，得到的 32 位结果符号扩展后写入通用寄存器 rd 中。

BSTRINS.W:

```
bstr32[31:msbw+1] = GR[rd][31:msbw+1]
bstr32[msbw:lsbw] = GR[rj][msbw-lsbw:0]
bstr32[lsbw-1:0]  = GR[rd][lsbw-1:0]
GR[rd] = SignExtend(bstr32[31:0], GRLEN)
```

BSTRINS.D 将通用寄存器 rd 中的[msbd:lsbd]位替换为通用寄存器 rj 中[msbd-lsbd:0]位，通用寄存器 rd 的其余位不变。

BSTRINS.D:

```
GR[rd][63:msbd+1] = GR[rd][63:msbd+1]
GR[rd][msbd:lsbd] = GR[rj][msbd-lsbd:0]
GR[rd][lsbd-1:0]  = GR[rd][lsbd-1:0]
```

2.2.3.9 BSTRPICK.{W/D}

指令格式： bstrpick.w rd, rj, msbw, lsbw　　bstrpick.d rd, rj, msbd, lsbd

BSTRPICK.W 提取通用寄存器 rj 中[msbw:lsbw]位零扩展至 32 位，所形成 32 位中间结果符号扩展后写入通用寄存器 rd 中。

BSTRPICK.W:

```
bstr32[31:0] = ZeroExtend(GR[rj][msbw:lsbw], 32)
GR[rd] = SignExtend(bstr32[31:0], GRLEN)
```

BSTRPICK.D 提取通用寄存器 rj 中[msbd:lsbd]位零扩展至 64 位写入通用寄存器 rd 中。

BSTRPICK.D:

```
GR[rd] = ZeroExtend(GR[rj][msbd:lsbd], 64)
```

2.2.3.10 MASKEQZ, MASKNEZ

指令格式： maskeqz　　rd, rj, rk

masknez　　rd, rj, rk

MASKEQZ 和 MASKNEZ 指令进行条件赋值操作。

MASKEQZ 执行时，如果通用寄存器 rk 的值等于 0 则将通用寄存器 rd 置为全 0，否则将其赋值为 rj 寄存器的值。

MASKEQZ:

```
GR[rd] = (GR[rk]==0) ? 0 : GR[rj]
```

MASKNEZ 执行时，如果通用寄存器 rk 的值不等于 0 则将通用寄存器 rd 置为全 0，否则将其赋值为 rj 寄存器的值。

MASKNEZ:

```
GR[rd] = (GR[rk]!=0) ? 0 : GR[rj]
```

2.2.4 转移指令

2.2.4.1 BEQ, BNE, BLT[U], BGE[U]

指令格式： beq　　rj, rd, offs16

bne　　rj, rd, offs16

blt　　rj, rd, offs16

bge　　rj, rd, offs16

bltu rj, rd, offs16

bgeu rj, rd, offs16

BEQ 将通用寄存器 rj 和通用寄存器 rd 的值进行比较，如果两者相等则跳转到目标地址，否则不跳转。

BEQ：

```
if GR[rj]==GR[rd] :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

BNE 将通用寄存器 rj 和通用寄存器 rd 的值进行比较，如果两者不等则跳转到目标地址，否则不跳转。

BNE：

```
if GR[rj]!=GR[rd] :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

BLT 将通用寄存器 rj 和通用寄存器 rd 的值视作有符号数进行比较，如果前者小于后者则跳转到目标地址，否则不跳转。

BLT：

```
if signed(GR[rj]) < signed(GR[rd]) :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

BGE 将通用寄存器 rj 和通用寄存器 rd 的值视作有符号数进行比较，如果前者大于等于后者则跳转到目标地址，否则不跳转。

BGE：

```
if signed(GR[rj]) >= signed(GR[rd]) :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

BLTU 将通用寄存器 rj 和通用寄存器 rd 的值视作无符号数进行比较，如果前者小于后者则跳转到目标地址，否则不跳转。

BLTU：

```
if unsigned(GR[rj]) < unsigned(GR[rd]) :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

BGEU 将通用寄存器 rj 和通用寄存器 rd 的值视作无符号数进行比较，如果前者大于等于后者则跳转到目标地址，否则不跳转。

BGEU：

```
if unsigned(GR[rj]) >= unsigned(GR[rd]) :
    PC = PC + SignExtend({offs16, 2'b0}, GRLEN)
```

上述六条分支指令的跳转目标地址计算方式是将指令码中的 16 比特立即数 offs16 逻辑左移 2 位后再符号扩展，所得的偏移值加上该分支指令的 PC。

不过需要注意的是，上述指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs16<<2。

2.2.4.2 BEQZ, BNEZ

指令格式： beqz rj, offs21

bnez rj, offs21

BEQZ 对通用寄存器 rj 的值进行判断，如果等于 0 则跳转到目标地址，否则不跳转。

BEQZ：

```
if GR[rj]==0 :
    PC = PC + SignExtend({offs21, 2'b0}, GRLEN)
```

BNEZ 对通用寄存器 rj 的值进行判断，如果不等于 0 则跳转到目标地址，否则不跳转。

BNEZ：

```
if GR[rj]!=0 :
    PC = PC + SignExtend({offs21, 2'b0}, GRLEN)
```

上述两条分支指令的跳转目标地址是将指令码中的21比特立即数offs21逻辑左移2位后再符号扩展，所得的偏移值加上该分支指令的 PC。

不过需要注意的是，上述指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs21<<2。

2.2.4.3 B

指令格式： b offs26

B 无条件跳转到目标地址处。其跳转目标地址是将指令码中的 26 比特立即数 offs26 逻辑左移 2 位后再符号扩展，所得的偏移值加上该分支指令的 PC。

B：

```
PC = PC + SignExtend({offs26, 2'b0}, GRLEN)
```

需要注意的是，该指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs26<<2。

2.2.4.4 BL

指令格式： bl offs26

BL 无条件跳转到目标地址处，同时将该指令的 PC 值加 4 的结果写入到 1 号通用寄存器 r1 中。

该指令的跳转目标地址是将指令码中的 26 比特立即数 offs26 逻辑左移 2 位后再符号扩展，所得的偏移值加上该分支指令的 PC。

BL：

```
GR[1] = PC + 4
PC = PC + SignExtend({offs26, 2'b0}, GRLEN)
```

在 LA ABI 中，1 号通用寄存器 r1 作为返回地址寄存器 ra。

需要注意的是，该指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs26<<2。

2.2.4.5 JIRL

指令格式： jirl rd, rj, offs16

JIRL 无条件跳转到目标地址处，同时将该指令的 PC 值加 4 的结果写入到通用寄存器 rd 中。

该指令的跳转目标地址是将指令码中的 16 比特立即数 offs16 逻辑左移 2 位后再符号扩展，所得的偏移值加上通用寄存器 rj 中的值。

JIRL：

```
GR[rd] = PC + 4
PC = GR[rj] + SignExtend({offs16, 2'b0}, GRLEN)
```

当 rd 等于 0 时，JIRL 的功能即是一条普通的非调用间接跳转指令。

rd 等于 0，rj 等于 1 且 offs16 等于 0 的 JIRL 常作为调用返回间接跳转使用。

需要注意的是，该指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs16<<2。

2.2.5 普通访存指令

2.2.5.1 LD.{B[U]/H[U]/W[U]/D}, ST.{B/H/W/D}

指令格式：

	ld.b	rd, rj, si12
	ld.h	rd, rj, si12
	ld.w	rd, rj, si12
	ld.d	rd, rj, si12
	ld.bu	rd, rj, si12
	ld.hu	rd, rj, si12
	ld.wu	rd, rj, si12
	st.b	rd, rj, si12
	st.h	rd, rj, si12
	st.w	rd, rj, si12
	st.d	rd, rj, si12

LD.{B/H/W}从内存取回一个字节/半字/字的数据符号扩展后写入通用寄存器 rd，LD.D 从内存取回一个双字的数据写入通用寄存器 rd。

LD.B：

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
byte = MemoryLoad(paddr, BYTE)
GR[rd] = SignExtend(byte, GRLEN)
```

LD.H：

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
halfword = MemoryLoad(paddr, HALFWORD)
GR[rd] = SignExtend(halfword, GRLEN)
```

LD.W:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
word = MemoryLoad(paddr, WORD)
GR[rd] = SignExtend(word, GRLEN)
```

LD.D:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
GR[rd] = MemoryLoad(paddr, DOUBLEWORD)
```

LD.{BU/HU/WU}从内存取回一个字节/半字/字的数据零扩展后写入通用寄存器 rd。

LD.BU:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
byte = MemoryLoad(paddr, BYTE)
GR[rd] = ZeroExtend(byte, GRLEN)
```

LD.HU:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
halfword = MemoryLoad(paddr, HALFWORD)
GR[rd] = ZeroExtend(halfword, GRLEN)
```

LD.WU:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
word = MemoryLoad(paddr, WORD)
GR[rd] = ZeroExtend(word, GRLEN)
```

ST.{B/H/W/D}将通用寄存器 rd 中的[7:0]/[15:0]/[31:0]/[63:0]位数据写入到内存中。

ST.B:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
MemoryStore(GR[rd][7:0], paddr, BYTE)
```

ST.H:

```
vaddr = GR[rj] + SignExtend(si12, GRLEN)
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
MemoryStore(GR[rd][15:0], paddr, HALFWORD)
```

```
ST.W:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][31:0], paddr, WORD)
ST.D:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][63:0], paddr, DOUBLEWORD)
```

上述指令的访存地址计算方式是将通用寄存器 rj 中的值与符号扩展后的 12 比特立即数 si12 相加求和。

对于 LD.{H[U]/W[U]/D}和 ST.{B/H/W/D}指令，无论在何种硬件实现及环境配置情况下，只要其访存地址是自然对齐的，都不会触发非对齐例外；当访存地址不是自然对齐时，如果硬件实现支持非对齐访存且当前运算环境配置为允许非对齐访存，那么不会触发非对齐例外，否则的话将触发非对齐例外。

2.2.5.2 LDX.{B[U]/H[U]/W[U]/D}, STX.{B/H/W/D}

指令格式：

	ldx.b	rd, rj, rk
	ldx.h	rd, rj, rk
	ldx.w	rd, rj, rk
	ldx.d	rd, rj, rk
	ldx.bu	rd, rj, rk
	ldx.hu	rd, rj, rk
	ldx.wu	rd, rj, rk
	stx.b	rd, rj, rk
	stx.h	rd, rj, rk
	stx.w	rd, rj, rk
	stx.d	rd, rj, rk

LDX.{B/H/W}从内存取回一个字节/半字/字的数据符号扩展后写入通用寄存器 rd，LDX.D 从内存取回一个双字的数据写入通用寄存器 rd。

```
LDX.B:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    byte = MemoryLoad(paddr, BYTE)
    GR[rd] = SignExtend(byte, GRLEN)
LDX.H:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
```

```
    paddr = AddressTranslation(vaddr)
    halfword = MemoryLoad(paddr, HALFWORD)
    GR[rd] = SignExtend(halfword, GRLEN)
LDX.W:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    word = MemoryLoad(paddr, WORD)
    GR[rd] = SignExtend(word, GRLEN)
LDX.D:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    GR[rd] = MemoryLoad(paddr, DOUBLEWORD)
```

LDX.{BU/HU/WU}从内存取回一个字节/半字/字的数据零扩展后写入通用寄存器 rd。

```
LDX.BU:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    byte = MemoryLoad(paddr, BYTE)
    GR[rd] = ZeroExtend(byte, GRLEN)
LDX.HU:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    halfword = MemoryLoad(paddr, HALFWORD)
    GR[rd] = ZeroExtend(halfword, GRLEN)
LDX.WU:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    word = MemoryLoad(paddr, WORD)
    GR[rd] = ZeroExtend(word, GRLEN)
```

STX.{B/H/W/D}将通用寄存器 rd 中的[7:0]/[15:0]/[31:0]/[63:0]位数据写入到内存中。

```
STX.B:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][7:0], paddr, BYTE)
STX.H:
```

```
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][15:0], paddr, HALFWORD)
STX.W:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][31:0], paddr, WORD)
STX.D:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][63:0], paddr, DOUBLEWORD)
```

上述指令的访存地址计算方式是将通用寄存器 rj 中的值与通用寄存器 rk 中的值相加求和。

对于 LDX.{H[U]/W[U]/D}和 STX.{B/H/W/D}指令，无论在何种硬件实现及环境配置情况下，只要其访存地址是自然对齐的，都不会触发非对齐例外；当访存地址不是自然对齐时，如果硬件实现支持非对齐访存且当前运算环境配置为允许非对齐访存，那么不会触发非对齐例外，否则的话将触发非对齐例外。

2.2.5.3 LDPTR.{W/D}, STPTR.{W/D}

指令格式：

ldptr.w	rd, rj, si14
ldptr.d	rd, rj, si14
stptr.w	rd, rj, si14
stptr.d	rd, rj, si14

LDPTR.W 从内存取回一个字的数据符号扩展后写入通用寄存器 rd，LDPTR.D 从内存取回一个双字的数据写入通用寄存器 rd。

```
LDPTR.W:
    vaddr = GR[rj] + SignExtend({si14, 2'b0}, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    word = MemoryLoad(paddr, WORD)
    GR[rd] = SignExtend(word, GRLEN)
LDPTR.D:
    vaddr = GR[rj] + SignExtend({si14, 2'b0}, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    GR[rd] = MemoryLoad(paddr, DOUBLEWORD)
```

STPTR.{W/D}将通用寄存器 rd 中[31:0]/[63:0]位的数据写入内存。

```
STPTR.W:
    vaddr = GR[rj] + SignExtend({si14, 2'b0}, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][31:0], paddr, WORD)
STPTR.D:
    vaddr = GR[rj] + SignExtend({si14, 2'b0}, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(GR[rd][63:0], paddr, DOUBLEWORD)
```

上述指令的访存地址计算方式是将 14 比特立即数 si14 逻辑左移 2 位后符号扩展，再与通用寄存器 rj 中的值相加求和。需要注意的是，上述指令的汇编表示中的立即数地址偏移值以字节为单位，即其值是指令码中 si14<<2。

对于 LDPTR.{W/D}和 STPTR.{W/D}指令，无论在何种硬件实现及环境配置情况下，只要其访存地址是自然对齐的，都不会触发非对齐例外；当访存地址不是自然对齐时，如果硬件实现支持非对齐访存且当前运算环境配置为允许非对齐访存，那么不会触发非对齐例外，否则的话将触发非对齐例外。

LDPTR.{W/D}、STPTR.{W/D}指令与 ADDU16I.D 指令配合使用，用于加速位置无关代码中基于 GOT 表的访问。

2.2.5.4 PRELD

指令格式： preld hint, rj, si12

PRELD 从内存中预取一个 Cache 行的数据进入 Cache 中。其访存地址的计算方式是将通用寄存器 rj 中的值与符号扩展后的 12 比特立即数 si12 相加求和。该访存地址落在待预取的 Cache 行内。

PRELD 指令中的 hint 提示处理器预取的类型以及取回的数据填入哪一级 Cache。hint 从 0~31 有 32 个可选值。目前 hint=0 定义为 load 预取至一级数据 Cache，hint=8 定义为 store 预取至一级数据 Cache。其余 hint 值的含义暂未定义，处理器执行时视同 NOP 指令处理。

如果 PRELD 指令的访存地址的 Cache 属性不是 cached，那么该指令不能产生访存动作，视同 NOP 指令处理。

PRELD 指令不会触发任何与 MMU 或是地址相关的例外。

2.2.5.5 PRELDX

指令格式： preldx hint, rj, rk

PRELDX 指令按照配置参数从内存中连续预取数据进入 Cache 中，所连续预取的数据是从指定基址（base）开始的数个（block_num）间距为 stride 的长度为 block_size 的数据块（block）。其中基址的计算方式是通用寄存器 rj 中[63:0]位与符号扩展后的通用寄存器 rk 中[15:0]位相加求和。通用寄存器 rk 中的[16]位为地址序列升降序标志位，0 表示地址升序，1 表示地址降序。通用寄存器 rk 中的[25:20]位的数值为 block_size-1，block_size 的基本单位为 16 字节，因此单个 block 的最大长度为 1KB。通用寄存器

rk 中的[39:32]位的数值为 block_num-1，因此单条指令最多可预取 256 个 block。通用寄存器 rk 中的[59:44]位的数值视作有符号数，定义了相邻 block 之间的 stride，stride 的基本单位是 1 字节。

PRELDX 指令中的 hint 提示处理器预取的类型以及取回的数据填入哪一级 Cache。hint 从 0~31 有 32 个可选值。目前 hint=0 定义为 load 预取至一级数据 Cache，hint=2 定义为 load 预取至三级 Cache，hint=8 定义为 store 预取至一级数据 Cache。其余 hint 值的含义暂未定义，处理器执行时视同 NOP 指令处理。

如果 PRELDX 指令的访存地址的 Cache 属性不是 cached，那么该指令不能产生访存动作，视同 NOP 指令处理。

PRELDX 指令不会触发任何与 MMU 或是地址相关的例外。

2.2.6 边界检查访存指令

2.2.6.1 LD{GT/LE}.{B/H/W/D}, ST{GT/LE}.{B/H/W/D}

指令格式：

	ldgt.b	rd, rj, rk
	ldgt.h	rd, rj, rk
	ldgt.w	rd, rj, rk
	ldgt.d	rd, rj, rk
	ldle.b	rd, rj, rk
	ldle.h	rd, rj, rk
	ldle.w	rd, rj, rk
	ldle.d	rd, rj, rk
	stgt.b	rd, rj, rk
	stgt.h	rd, rj, rk
	stgt.w	rd, rj, rk
	stgt.d	rd, rj, rk
	stle.b	rd, rj, rk
	stle.h	rd, rj, rk
	stle.w	rd, rj, rk
	stle.d	rd, rj, rk

LDGT/LDLE.{B/H/W}从内存取回一个字节/半字/字的数据符号扩展后写入通用寄存器 rd，LDGT/LDLE.D 从内存取回一个双字的数据写入通用寄存器 rd。

STGT/STLE.{B/H/W/D}将通用寄存器 rd 中的[7:0]/[15:0]/[31:0]/[63:0]位数据写入到内存中。

上述指令的访存地址直接来自于通用寄存器 rj 中的值。上述指令的访存地址均要求自然对齐，否则将触发非对齐例外。

LDGT.{B/H/W/D}和 STGT.{B/H/W/D}指令执行时，会检查通用寄存器 rj 中的值是否大于通用寄存器

rk 中的值，如果条件不满足则终止访存操作并触发边界检查例外。LDLE.{B/H/W/D}和 STLE.{B/H/W/D}指令执行时，会检查通用寄存器 rj 中的值是否小于等于通用寄存器 rk 中的值，如果条件不满足则终止访存操作并触发边界检查错误例外。

LDGT.B:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    byte = MemoryLoad(paddr, BYTE)
    GR[rd] = SignExtend(byte, GRLEN)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

LDGT.H:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    halfword = MemoryLoad(paddr, HALFWORD)
    GR[rd] = SignExtend(halfword, GRLEN)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

LDGT.W:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    word = MemoryLoad(paddr, WORD)
    GR[rd] = SignExtend(word, GRLEN)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

LDGT.D:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    GR[rd] = MemoryLoad(paddr, DOUBLEWORD)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

LDLE.B:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
```

```
    if GR[rj]<=GR[rk] :
        byte = MemoryLoad(paddr, BYTE)
        GR[rd] = SignExtend(byte, GRLEN)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

LDLE.H:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        halfword = MemoryLoad(paddr, HALFWORD)
        GR[rd] = SignExtend(halfword, GRLEN)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

LDLE.W:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        word = MemoryLoad(paddr, WORD)
        GR[rd] = SignExtend(word, GRLEN)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

LDLE.D:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        GR[rd] = MemoryLoad(paddr, DOUBLEWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STGT.B:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]>GR[rk] :
        MemoryStore(GR[rd][7:0], paddr, BYTE)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STGT.H:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
```

```
    paddr = AddressTranslation(vaddr)
    if GR[rj]>GR[rk] :
        MemoryStore(GR[rd][15:0], paddr, HALFWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STGT.W:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]>GR[rk] :
        MemoryStore(GR[rd][31:0], paddr, WORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STGT.D:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]>GR[rk] :
        MemoryStore(GR[rd][63:0], paddr, DOUBLEWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STLE.B:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(GR[rd][7:0], paddr, BYTE)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STLE.H:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(GR[rd][15:0], paddr, HALFWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

STLE.W:

```
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(GR[rd][31:0], paddr, WORD)
```

```
    else :
        RaiseException(BCE) #Bound Check Error Exception
STLE.D:
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(GR[rd][63:0], paddr, DOUBLEWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

2.2.7 原子访存指令

2.2.7.1 AM{SWAP/ADD/AND/OR/XOR/MAX/MIN}[_DB].{W/D}, AM{MAX/MIN}[_DB].{WU/DU}

指令格式：

amswap.w	rd, rk, rj	amswap_db.w	rd, rk, rj
amswap.d	rd, rk, rj	amswap_db.d	rd, rk, rj
amadd.w	rd, rk, rj	amadd_db.w	rd, rk, rj
amadd.d	rd, rk, rj	amadd_db.d	rd, rk, rj
amand.w	rd, rk, rj	amand_db.w	rd, rk, rj
amand.d	rd, rk, rj	amand_db.d	rd, rk, rj
amor.w	rd, rk, rj	amor_db.w	rd, rk, rj
amor.d	rd, rk, rj	amor_db.d	rd, rk, rj
amxor.w	rd, rk, rj	amxor_db.w	rd, rk, rj
amxor.d	rd, rk, rj	amxor_db.d	rd, rk, rj
ammax.w	rd, rk, rj	ammax_db.w	rd, rk, rj
ammax.d	rd, rk, rj	ammax_db.d	rd, rk, rj
ammin.w	rd, rk, rj	ammin_db.w	rd, rk, rj
ammin.d	rd, rk, rj	ammin_db.d	rd, rk, rj
ammax.wu	rd, rk, rj	ammax_db.wu	rd, rk, rj
ammax.du	rd, rk, rj	ammax_db.du	rd, rk, rj
ammin.wu	rd, rk, rj	ammin_db.wu	rd, rk, rj
ammin.du	rd, rk, rj	ammin_db.du	rd, rk, rj

AM*原子访存指令能够原子地完成对某个内存单元的“读-修改-写”操作序列。具体来说，其将内存指定地址处的旧值取回并写入通用寄存器 rd，同时将这个内存中的旧值与通用寄存器 rk 中的值进行一些简单的运算操作，然后将运算的结果写回到内存的指定地址处。整个“读-修改-写”的过程是原子的，意味着该指令执行时，从访存读操作数据返回到访存写操作执行效果全局可见这段时间内，执行该指令的处理器

既没有执行其它访存写操作也没有触发任何例外，且没有其它处理器核或 Cache 一致性模块对该指令访问对象所在 Cache 行的写操作的执行效果全局可见。

AM*原子访存指令的访存地址是通用寄存器 rj 的值。AM*原子访存指令的访存地址总是要求自然对齐，如果不满足这一条件的话将触发非对齐例外。

以.W 和.WU 结尾的原子访存指令读、写内存以及中间运算的数据宽度都是 32 位，以.D 和.DU 结尾的原子访存指令读、写内存以及中间运算的数据宽度都是 64 位。无论是以.W 还是.WU 结尾，原子访存指令从内存取回的一个字的数据都是符号扩展后写入到通用寄存器 rd。

AMSWAP[_DB].{W/D}指令写入内存的新值来自于通用寄存器 rk。AMADD[_DB].{W/D}指令写入内存的新值来自内存的旧值与通用寄存器 rk 中的值相加的结果。AMAND[_DB].{W/D}指令写入内存的新值来自内存的旧值与通用寄存器 rk 中的值按位逻辑与的结果。AMOR[_DB].{W/D}指令写入内存的新值来自内存的旧值与通用寄存器 rk 中的值按位逻辑或的结果。AMXOR[_DB].{W/D}指令写入内存的新值来自内存的旧值与通用寄存器 rk 中的值按位逻辑异或的结果。AMMAX[_DB].{W/D}指令写入内存的新值是将内存的旧值与通用寄存器 rk 中的值视作有符号数进行大小比较后所得的最大值。AMMIN[_DB].{W/D}指令写入内存的新值是将内存的旧值与通用寄存器 rk 中的值视作有符号数进行大小比较后所得的最小值。AMMAX[_DB].{WU/DU}指令写入内存的新值是将内存的旧值与通用寄存器 rk 中的值视作无符号数进行大小比较后所得的最大值。AMMIN[_DB].{WU/DU}指令写入内存的新值是将内存的旧值与通用寄存器 rk 中的值视作无符号数进行大小比较后所得的最小值。

AM*_DB.W[U]/D[U]指令除了完成上述原子化的操作序列外，还同时实现数据栅障功能。即当此类原子访存指令被允许执行之前，所有在同一处理器核中先于该原子访存指令的访存操作都已完成；同时只有等到此类原子访存指令执行完成后，所有在同一处理器核中后于该原子访存指令的访存操作才被允许执行。

AM*原子访存指令如果 rd 和 rj 的寄存器号相同，则触发指令不存在例外。

AM*原子访存指令如果 rd 和 rk 的寄存器号相同，则执行结果不确定，请软件避免出现这种情况。

2.2.7.2 LL.{W/D}, SC.{W/D}

指令格式：

	ll.w	rd, rj, si14
	ll.d	rd, rj, si14
	sc.w	rd, rj, si14
	sc.d	rd, rj, si14

LL.W 和 SC.W、LL.D 和 SC.D 这两对指令用于实现原子的“读-修改-写”访存操作序列。LL.{W/D}指令从内存指定地址取回一个字/双字的数据符号扩展后写入通用寄存器 rd，与之配对的 SC.{W/D}指令操作同样宽度的数据且访问相同的内存地址。访存操作序列原子性的维护机制是，LL.{W/D}执行时记录下访问地址并置上一个标记（LLbit 置为 1），SC.{W/D}指令执行时会查看 LLbit，仅当 LLbit 为 1 时才真正产生写动作，否则不写。当软件需要一定成功完成一个原子的“读-修改-写”访存操作序列时，需要构建一个循环来反复执行 LL-SC 指令对直至 SC 成功完成。为了构建这个循环，SC.{W/D}指令会将其执行成功与否的标志（也可以简单理解为 SC 指令执行时所看到的 LLbit 值）写入到通用寄存器 rd 中返回。

在配对的 LL-SC 执行期间，下列事件会让 LLbit 清 0：

- 执行了 ERTN 指令且执行时 CSR.LLBCTL 中的 KLO 位不等于 1；
- 其它处理器核或 Cache Coherent I/O master 对该 LLbit 对应的地址所在的 Cache 行执行完成了一个 store 操作。

如果 LL-SC 指令对访问地址的存储访问属性不是 Cached，那么执行结果不确定。

需要注意的是，上述指令在计算内存地址时，需要将指令码中的 si14 左移 2 位后再与基址相加，即

```
vaddr = GR[rj] + SignExtend({si14, 2'b0}, GRLEN)
```

但是这些指令的汇编表示中所呈现的立即数地址偏移值仍以字节为单位，即其值是指令码中 si14<<2。

2.2.8 栅障指令

2.2.8.1 DBAR

指令格式： dbar hint

DBAR 指令用于完成 load/store 访存操作之间的栅障功能。其携带的立即数 hint 用于指示该栅障的同步对象和同步程度。

hint 值为 0 是默认必须实现的，其指示一个完全功能的同步栅障。只有等到之前所有 load/store 访存操作彻底执行完毕后，“DBAR 0”指令才能开始执行；且只有“DBAR 0”执行完成执行后，其后所有 load/store 访存操作才能开始执行。

如果没有专门的功能实现，其它所有 hint 值都必须按照 hint=0 执行。

2.2.8.2 IBAR

指令格式： ibar hint

IBAR 指令使用完成单个处理器核内部 store 操作与取指操作之间的同步。其携带的立即数 hint 用于指示该栅障的同步对象和同步程度。

hint 值为 0 是默认必须实现的。它能够确保“IBAR 0”指令之后的取指一定能够观察到“IBAR 0”指令之前所有 store 操作的执行效果。

2.2.9 CRC 校验指令

2.2.9.1 CRC[C].W.{B/H/W/D}.W

指令格式：

crc.w.b.w	rd, rj, rk
crc.w.h.w	rd, rj, rk
crc.w.w.w	rd, rj, rk
crc.w.d.w	rd, rj, rk
crcc.w.b.w	rd, rj, rk
crcc.w.h.w	rd, rj, rk

crcc.w.w.w rd, rj, rk

crcc.w.d.w rd, rj, rk

CRC[C].W.{B/H/W/D}.W 用于进行 CRC-32 校验和计算，其将存于通用寄存器 rk 的 32 位累积的 CRC 校验和和存于通用寄存器 rj 中[7:0]/[15:0]/[31:0]/[63:0]位的消息，根据 CRC-32 校验和生成算法得到新的 32 位 CRC 校验和，符号扩展后写入通用寄存器 rd 中。其区别在于，CRC.W.{B/H/W/D}.W 使用 IEEE 802.3 多项式（多项式值为 0xEDB88320），CRCC.W.{B/H/W/D}.W 使用 Castagnoli 多项式（多项式值为 0x82F63B78）。本手册定义的 CRC 指令均只支持“LSB 优先”（小尾端）标准，即首先传输数据最低位（小尾端）的标准，数据的最低位映射到消息多项式的最高有效项的系数。

```
CRC.W.B.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][7:0], 8, 0xEDB88320)
    GR[rd] = SignExtend(chksum, GRLEN)
CRC.W.H.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][15:0], 16, 0xEDB88320)
    GR[rd] = SignExtend(chksum, GRLEN)
CRC.W.W.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][31:0], 32, 0xEDB88320)
    GR[rd] = SignExtend(chksum, GRLEN)
CRC.W.D.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][63:0], 64, 0xEDB88320)
    GR[rd] = SignExtend(chksum, GRLEN)

CRCC.W.B.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][7:0], 8, 0x82F63B78)
    GR[rd] = SignExtend(chksum, GRLEN)
CRCC.W.H.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][15:0], 16, 0x82F63B78)
    GR[rd] = SignExtend(chksum, GRLEN)
CRCC.W.W.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][31:0], 32, 0x82F63B78)
    GR[rd] = SignExtend(chksum, GRLEN)
CRCC.W.D.W:
    chksum = CRC32(GR[rk][31:0], GR[rj][63:0], 64, 0x82F63B78)
    GR[rd] = SignExtend(chksum, GRLEN)
```

2.2.10 其它杂项指令

2.2.10.1 SYSCALL

指令格式： syscall code

执行 SYSCALL 指令将立即无条件的触发系统调用例外。

指令码中 code 域携带的信息可供例外处理例程作为所传递的参数使用。

2.2.10.2 BREAK

指令格式： break code

执行 BREAK 指令将立即无条件的触发断点例外。

指令码中 code 域携带的信息可供例外处理例程作为所传递的参数使用。

2.2.10.3 ASRT{LE/GT}.D

指令格式： asrtle.d rj, rk

asrtgt.d rj, rk

将通用寄存器 rj 与通用寄存器 rk 中的值视作有符号数进行比较，如果比较条件不满足则触发地址边界检查例外。对于 ASRTLE.D 指令，如果通用寄存器 rj 中的值大于通用寄存器 rk 中的值则触发例外；对于 ASRTGT.D 指令，如果通用寄存器 rj 中的值小于等于通用寄存器 rk 中的值则触发例外。

2.2.10.4 RDTIME{L/H}.W, RDTIME.D

指令格式： rdtimel.w rd, rj rdtime.d rd, rj

rdtimeh.w rd, rj

龙芯指令系统定义了一个恒定频率计时器，其主体是一个 64 位的计数器，称为 Stable Counter。Stable Counter 在复位后置为 0，随后每个计数时钟周期自增 1，当计数至全 1 时自动绕回至 0 继续自增。同时每个计时器都有一个软件可配置的全局唯一编号，称为 Counter ID。恒定频率计时器的特点在于其计时频率在复位后保持不变，无论处理器核的时钟频率如何变化。

RDTIME{L/H}.W 和 RDTIME.D 指令用于读取恒定频率计时器信息，Stable Counter 值写入通用寄存器 rd 中，Counter ID 号信息写入通用寄存器 rj 中。三条指令的区别在于所读取的 Stable Counter 信息的不同，其中 RDTIMEL.W 读取 Counter 的[31:0]位，RDTIMEH.W 读取 Counter 的[63:32]位，而 RDTIME.D 读取整个 64 位的 Counter 值。在 64 位的处理器上，RDTIME{L/H}.W 指令读取的 32 位数值符号扩展后写入通用寄存器 rd。定义 RDTIME{L/H}.W 指令是为了在 32 位处理器上也可以访问 64 位的 Counter。

2.2.10.5 CPUCFG

指令格式： cpucfg rd, rj

CPUCFG 指令用于软件在执行过程中动态识别所运行的处理器中实现了龙芯架构中的哪些功能特性。这些指令系统功能特性的实现情况记录在一系列配置信息字中，CPUCFG 指令执行一次可以读取一个配置信息字。

使用 CPUCFG 指令时，源操作数寄存器 rj 中存放待访问的配置信息字的编号，指令执行后所读取的配置信息字信息写入通用寄存器 rd 中。每个配置信息字为 32 比特，其在 LA64 架构下将符号扩展后在写入结果寄存器。

配置信息字中包含一系列配置位（域），其记录形式为 CPUCFG.<配置字号>.<配置信息助记名称>[位下标]，其中单比特配置位的位下标记为 bitXX，表示配置字的第 XX 位；多比特的配置域的位下标记为 bitXX:YY，表示配置字的第 XX 位到第 YY 位的连续(XX-YY+1)位。例如，1 号配置字中的第 0 位用以表示是否实现 LA32 架构，将这个配置信息记录为 CPUCFG.1.LA32[bit0]，其中 1 表示配置信息字的字号是 1 号，LA32 表示这个配置信息域所起的助记名称叫做 LA32，bit0 表示 LA32 这个域位于配置字的第 0 位。1 号配置字中第 11 位到第 4 位的记录所支持物理地址位数的 PALEN 域则记为 CPUCFG.1.PALEN[bit11:4]。

龙芯架构中 CPUCFG 指令可访问的配置信息列举在表 2-2 中。CPUCFG 访问未定义的配置字将读回全 0 值。已定义的配置字中的未定义域，执行 CPUCFG 时可以读回任意值，软件对其应不做任何解读。

表 2-2 CPUCFG 访问配置信息列表

字号	位下标	助记名称	含义
0	31:0	PRID	处理器标识
1	1:0	ARCH	2'b00 表示实现 LA32 精简架构；2'b01 表示实现 LA32 架构；2'b10 表示实现 LA64 架构。2'b11 保留。
	2	PGMMU	为 1 表示 MMU 支持页映射模式
	3	IOCSR	为 1 表示支持 IOCSR 指令
	11:4	PALEN	所支持的物理地址位数 PALEN 的值减 1
	19:12	VALEN	所支持的虚拟地址位数 VALEN 的值减 1
	20	UAL	为 1 表示支持非对齐访存
	21	RI	为 1 表示支持“读禁止”页属性
	22	EP	为 1 表示支持“执行保护”页属性
	23	RPLV	为 1 表示支持 RPLV 页属性
	24	HP	为 1 表示支持 huge page 页属性
	25	IOCSR_BRD	为 1 表示 IOCSR 访问空间的 0 地址处记录了处理器产品信息的字符串。即“Loongson3A5000 @ 2.5GHz”这样的信息。
	26	MSG_INT	为 1 表示外部中断采用消息中断方式，否则为电平中断线方式
2	0	FP	为 1 表示支持基础浮点数指令
	1	FP_SP	为 1 表示支持单精度浮点数
	2	FP_DP	为 1 表示支持双精度浮点数
	5:3	FP_ver	浮点运算标准的版本号。1 为初始版本号，表示兼容 IEEE 754-2008 标准。
	6	LSX	为 1 表示支持 128 位向量扩展
	7	LASX	为 1 表示支持 256 位向量扩展
	8	COMPLEX	为 1 表示支持复数向量运算指令
	9	CRYPTO	为 1 表示支持加解密向量指令
	10	LVZ	为 1 表示支持虚拟化扩展
	13:11	LVZ_ver	虚拟化硬件加速规范的版本号。1 为初始版本号。
	14	LLFTP	为 1 表示支持恒定频率计时器和定时器

字号	位下标	助记名称	含义
	17:15	LLFTP_ver	恒定频率计时器和定时器的版本号。1 为初始版本。
	18	LBT_X86	为 1 表示支持 X86 二进制翻译扩展
	19	LBT_ARM	为 1 表示支持 ARM 二进制翻译扩展
	20	LBT_MIPS	为 1 表示支持 MIPS 二进制翻译扩展
	21	LSPW	为 1 表示支持软件页表遍历指令
	22	LAM	为 1 表示支持 AM*原子访存指令
3	0	CCDMA	为 1 表示支持硬件 Cache Coherent DMA
	1	SFB	为 1 表示支持 Store Fill Buffer（SFB）
	2	UCACC	为 1 表示支持 ucacc win
	3	LLEXC	为 1 表示支持 LL 指令取独占块功能
	4	SCDLY	为 1 表示支持 SC 后随机延迟功能
	5	LLDBAR	为 1 表示支持 LL 自动带 dbar 功能
	6	ITLBHMC	为 1 表示硬件维护 ITLB 与 TLB 之间的一致性
	7	ICHMC	为 1 表示硬件维护同一处理器核内 ICache 与 DCache 的数据一致性
	10:8	SPW_LVL	page walk 指令所支持的最大目录层数
	11	SPW_HP_HF	为 1 表示 page walk 指令在遇到大页时将折半填入 TLB
	12	RVA	为 1 表示支持软件配置缩短虚拟地址范围的功能
	16:13	RVAMAX-1	最大可以配置的虚拟地址缩短位数-1
4	31:0	CC_FREQ	恒定频率计时器和定时器所用时钟对应的晶振频率
5	15:0	CC_MUL	恒定频率计时器和定时器所用时钟对应的倍频因子
	31:16	CC_DIV	恒定频率计时器和定时器所用时钟对应的分频系数
6	0	PMP	为 1 表示支持性能计数器
	3:1	PMVER	性能监测器中，架构定义事件的版本号，1 为初始版本。
	7:4	PMNUM	性能监测器个数-1
	13:8	PMBITS	性能监测计数器位宽-1
	14	UPM	为 1 表示支持用户态读取性能计数器
10	0	L1 IU_Present	为 1 表示存在一级指令 Cache 或一级统一 Cache
	1	L1 IU Unify	为 1 表示 L1 IU_Present 所示的 Cache 是统一 Cache
	2	L1 D Present	为 1 表示存在一级数据 Cache
	3	L2 IU Present	为 1 表示存在二级指令 Cache 或二级统一 Cache
	4	L2 IU Unify	为 1 表示 L2 IU_Present 所示的 Cache 是统一 Cache
	5	L2 IU Private	为 1 表示 L2 IU_Present 所示的 Cache 是每个核私有的
	6	L2 IU Inclusive	为 1 表示 L2 IU_Present 所示的 Cache 对更低层次（L1）是包含关系
	7	L2 D Present	为 1 表示存在二级数据 Cache
	8	L2 D Private	为 1 表示二级数据 Cache 是每个核私有的
	9	L2 D Inclusive	为 1 表示二级数据 Cache 对更低层次（L1）是包含关系
	10	L3 IU Present	为 1 表示存在三级指令 Cache 或三级统一 Cache
	11	L3 IU Unify	为 1 表示 L3 IU_Present 所示的 Cache 是统一 Cache

字号	位下标	助记名称	含义
	12	L3 IU Private	为 1 表示 L3 IU_Present 所示的 Cache 是每个核私有的
	13	L3 IU Inclusive	为 1 表示 L3 IU_Present 所示的 Cache 对更低层次（L1 及 L2）是包含关系
	14	L3 D Present	为 1 表示存在三级数据 Cache
	15	L3 D Private	为 1 表示三级数据 Cache 是每个核私有的
	16	L3 D Inclusive	为 1 表示三级数据 Cache 对更低层次（L1 及 L2）是包含关系
11	15:0	Way-1	路数-1 （配置字 10 中 L1 IU_Present 对应的 Cache）
	23:16	Index-log2	$\log_2$(每一路 Cache 行数) （配置字 10 中 L1 IU_Present 对应的 Cache）
	30:24	Linesize-log2	$\log_2$(Cache 行字节数) （配置字 10 中 L1 IU_Present 对应的 Cache）
12	15:0	Way-1	路数-1 （配置字 10 中 L1 D Present 对应的 Cache）
	23:16	Index-log2	$\log_2$(每一路 Cache 行数) （配置字 10 中 L1 D Present 对应的 Cache）
	30:24	Linesize-log2	$\log_2$(Cache 行字节数) （配置字 10 中 L1 D Present 对应的 Cache）
13	15:0	Way-1	路数-1 （配置字 10 中 L2 IU Present 对应的 Cache）
	23:16	Index-log2	$\log_2$(每一路 Cache 行数) （配置字 10 中 L2 IU Present 对应的 Cache）
	30:24	Linesize-log2	$\log_2$(Cache 行字节数) （配置字 10 中 L2 IU Present 对应的 Cache）
14	15:0	Way-1	路数-1 （配置字 10 中 L3 IU Present 对应的 Cache）
	23:16	Index-log2	$\log_2$(每一路 Cache 行数) （配置字 10 中 L3 IU Present 对应的 Cache）
	30:24	Linesize-log2	$\log_2$(Cache 行字节数) （配置字 10 中 L3 IU Present 对应的 Cache）

3 基础浮点数指令

本章将介绍龙芯架构非特权子集基础部分中的浮点数指令。龙芯架构中的基础浮点数指令的功能定义遵循 IEEE 754-2008 标准。

基础浮点指令不能脱离基础整数指令而单独实现。通常来说，我们推荐同时实现基础整数指令和基础浮点数指令。但是，对于一些成本敏感且浮点数处理性能需求极低的嵌入式应用场合，架构规范也允许不实现基础浮点数指令，或是只实现基础浮点数指令中操作单精度浮点数和字整数的指令。实现基础浮点数指令时是否包含操作双精度浮点数和双字整数的指令与架构是 LA32 还是 LA64 无关。但是 FSCALEB.S/D、FLOGB.S/D 和 FRINT.S/D 这 6 条指令仅需在 LA64 架构下实现。

3.1 基础浮点数指令编程模型

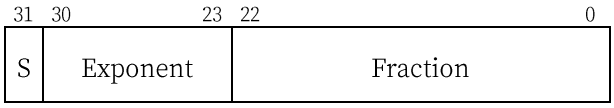
本节所要描述的基础浮点数指令编程模型只涉及应用软件开发人员所需关注的内容。软件人员在使用基础浮点数指令编程时，是在基础整数指令编程模型的基础之上，再进一步涉及本节所述的内容。

3.1.1 浮点数据类型

浮点数据类型包括单精度浮点数和双精度浮点数，两者均遵循 IEEE 754-2008 标准规范中的定义。

3.1.1.1 单精度浮点数

单精度浮点数的宽度为 32 比特，组织为如下格式：



根据 S、Exponent 和 Fraction 各个域数值的不同，所表示的浮点数数值如表 3-1 所示：

表 3-1 单精度浮点数数值计算方式

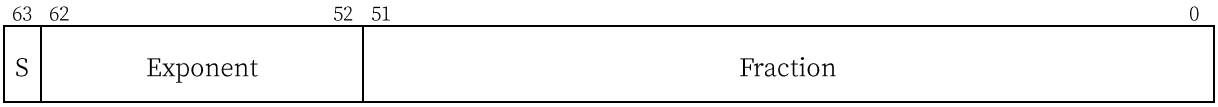
Exponent	Fraction	S	bit[22]	V
0	=0	0	0	+0
		1	0	-0
0	!=0	0	任意值	非规格化数，值为$+2^{-126}\times(0.Fraction)$
		1	任意值	非规格化数，值为$-2^{-126}\times(0.Fraction)$
[1, 0xFE]	任意值	0	任意值	规格化数，值为$+2^{(Exponent-127)}\times(1.Fraction)$
		1	任意值	规格化数，值为$-2^{(Exponent-127)}\times(1.Fraction)$
0xFF	=0	0	0	正无穷（+∞）
		1	0	负无穷（-∞）
0xFF	!=0	任意值	0	发信非数（Signaling Not a Number, SNaN）

Exponent	Fraction	S	bit[22]	V
		任意值	1	静默非数（Quiet Not a Number, QNaN）

有关±∞、SNaN 和 QNaN 的具体含义，请查看 IEEE 754-2008 标准规范。

3.1.1.2 双精度浮点数

双精度浮点数的宽度为 64 比特，组织为如下格式：



根据 S、Exponent 和 Fraction 各个域数值的不同，所表示的浮点数数值如表 3-2 所示：

表 3-2 双精度浮点数数值计算方式

Exponent	Fraction	S	bit[51]	V
0	=0	0	0	+0
		1	0	-0
0	!=0	0	任意值	非规格化数，值为$+2^{-1022}\times(0.Fraction)$
		1	任意值	非规格化数，值为$-2^{-1022}\times(0.Fraction)$
[1, 0x7FE]	任意值	0	任意值	规格化数，值为$+2^{(Exponent-1023)}\times(1.Fraction)$
		1	任意值	规格化数，值为$-2^{(Exponent-1023)}\times(1.Fraction)$
0x7FF	=0	0	0	正无穷（+∞）
		1	0	负无穷（-∞）
0x7FF	!=0	任意值	0	发信非数（Signaling Not a Number, SNaN）
		任意值	1	静默非数（Quiet Not a Number, QNaN）

有关±∞、SNaN 和 QNaN 的具体含义，请查看 IEEE 754-2008 标准规范。

3.1.1.3 指令产生的非数结果

浮点数指令产生的非数结果[1]或者来自于 NaN 传播，或者是直接生成的。其中需要进行 NaN 传播的情况有两种。

情况一、当指令由于含有 SNaN 的源操作数而生成 Invalid Operation 浮点例外，但是 Invalid Operation 浮点例外使能无效，此时会产生一个 QNaN 结果。这个 QNaN 的数值是选择源操作数中优先级最高的 SNaN，将其传播为对应的 NaN。

源操作数的优先级的判定规则是：如果有两个源操作数 fj 和 fk，那么 fj 的优先级高于 fk；如果有三个源操作数 fa、fj 和 fk，那么 fa 的优先级高于 fj，fj 的优先级高于 fk。

SNaN 传播为 QNaN 的数值生成规则如下：

- 如果结果与源操作数等宽，那么将被传播 SNaN 尾数的最高位置 1，其余位保持不变。
- 如果结果比源操作数窄，那么保留尾数的高位，丢弃超过范围的低位，最后将尾数的最高位置 1。
- 如果结果比源操作数宽，那么将尾数的最低位后面补齐 0，最后将尾数的最高位置 1。

[1] 此时的非数只能是 QNaN。

情况二、源操作数中没有 SNaN，但有 QNaN 存在时，选择优先级最高的 QNaN 作为这条指令的结果。

此时源操作数优先级的判断方式与上面情况一中的一致。

除了上面两种情况外，其它需要产生 QNaN 结果的情况都将直接置为缺省的 QNaN 值。规定缺省的单精度 QNaN 的值为 0x7FC00000，缺省的双精度 QNaN 的值为 0x7FF8000000000000。

3.1.2 定点数据类型

部分浮点指令（如浮点转换指令）也会操作定点数据，包括字（Word，简记 W，长度 32b）、长字（Longword，简记 L，长度 64b）。

字和长字数据类型均采用二进制补码的编码方式。

3.1.3 寄存器

浮点数指令编程涉及到寄存器有浮点寄存器（Floating-point Register，简称 FR）、条件标志寄存器（Condition Flag Register，简称 CFR）和浮点控制状态寄存器（Floating-point Control and Status Register，简称 FCSR）。

3.1.3.1 浮点寄存器

FR 共有 32 个，记为 f0~f31，每一个都可以读写。仅当只实现操作单精度浮点数和字整数的浮点指令时，FR 的位宽为 32 比特。通常情况下，FR 的位宽为 64 比特，无论是 LA32 还是 LA64 架构。基础浮点数指令与浮点寄存器存在正交关系，即从架构角度而言，这些指令中任一个浮点寄存器操作数都可以采用 32 个 FR 中的任一个。

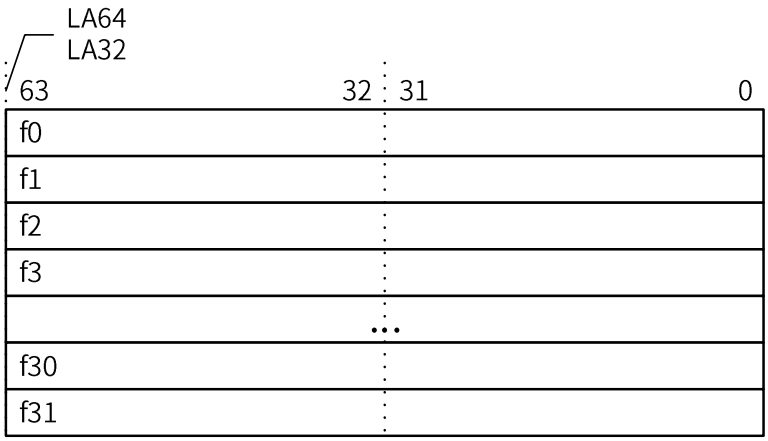


图 3-1 浮点寄存器

当浮点寄存器中记录的是一个单精度浮点数或字整数时，数据总是出现在浮点寄存器的[31:0]位上，此时浮点寄存器的[63:32]位可以是任意值。

3.1.3.2 条件标志寄存器

CFR 共有 8 个，记为 fcc0~fcc7，每一个都可以读写。CFR 的位宽为 1 比特。浮点比较的结果将写入到条件标志寄存器中，当比较结果为真则置 1，否则置 0。浮点分支指令的判断条件来自于条件标志寄存器。

3.1.3.3 浮点控制状态寄存器

FCSR 共有 4 个，记为 fcsr0~fcsr3，位宽均为 32 比特。其中 fcsr1~fcsr3 是 fcsr0 中部分域的别名，即访问 fcsr1~fcsr3 其实是访问 fcsr0 的某些域。当软件写 fcsr1~fcsr3 时，fcsr0 中对应的域被修改而其余比特保持不变。fcsr0 的各个域的定义如表 3-3 所示。

表 3-3 FCSR0 寄存器域定义

位	名字	读写	描述
4:0	Enables	RW	浮点操作 VZOUI 例外各自允许触发例外陷入的使能位。 比特 4 对应 V，比特 3 对应 Z，比特 2 对应 O，比特 1 对应 U，比特 0 对应 I。
7:5	0	R0	保留域。读返回 0，且软件不允许改变其值。
9:8	RM	RW	舍入模式控制。其包含 4 个合法值，各自含义如下： • 0：RNE，对应 IEEE 754-2008 中的 roundTiesToEven； • 1：RZ，对应 IEEE 754-2008 中的 roundTowardZero； • 2：RP，对应 IEEE 754-2008 中的 roundTowardsPositive； • 3：RM，对应 IEEE 754-2008 中的 roundTowardsNegative。
15:10	0	R0	保留域。读返回 0，且软件不允许改变其值。
20:16	Flags	RW	自上次 Flags 域被软件清空后，各类产生但未陷入的浮点操作 VZOUI 例外的累计情况。 比特 20 对应 V，比特 19 对应 Z，比特 18 对应 O，比特 17 对应 U，比特 16 对应 I。
23:21	0	R0	保留域。读返回 0，且软件不允许改变其值。
28:24	Cause	RW	最近一次浮点操作所产生的 VZOUI 例外的情况。 比特 28 对应 V，比特 27 对应 Z，比特 26 对应 O，比特 25 对应 U，比特 24 对应 I。
31:29	0	R0	保留域。读返回 0，且软件不允许改变其值。

FCSR1 是 FCSR0 中 Enables 域的别名。其位置与 FCSR0 中一致。

FCSR2 是 FCSR0 中 Cause 和 Flags 域的别名。各个域的位置与 FCSR0 中一致。

FCSR3 是 FCSR0 中 RM 域的别名。其位置与 FCSR0 中一致。

3.1.4 浮点例外

浮点例外是指，当浮点处理单元不能以常规的方式处理操作数或者浮点计算的结果时，浮点功能部件将产生相应的例外。

基础浮点指令支持五个 IEEE 754-2008 所定义的浮点例外：

- 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)

FCSR0 中 Cause 域的每一位对应上述的一个例外。每条浮点指令执行结束后会将其例外的产生情况更新至 FCSR0 的 Cause 域中。

FCSR0 中对于每一种浮点例外还包含一个使能位（Enables 域）。使能位决定了浮点处理单元产生的例外是将会触发一个例外陷入还是设置一个状态标志。当某个浮点例外产生时[1]，如果其对应的 Enable 位为 1，那么将触发一个浮点例外陷入；如果其对应的 Enable 位为 0，那么将不会触发浮点例外陷入，而是将 FCSR0 中 Flag 域的对应位置 1。

一条浮点指令在执行过程中，可以同时产生多个浮点例外。

当浮点指令执行过程中产生了浮点例外但并没有触发浮点例外陷入的时候，浮点处理单元将生成一个缺省的结果。不同的例外产生缺省结果的方式也不相同，表 3-4 列出了具体的生成规则。

表 3-4 浮点例外的缺省结果

域	描述	舍入模式	缺省结果
I	非精确	任何模式	舍入后的结果或上溢出后的结果
U	下溢	RNE	舍入后的结果，可能是 0, subnormal, 绝对值最小的 normal 数（单精度：$\pm 2^{-126}$，双精度：$\pm 2^{-1022}$）
		RZ	舍入后的结果，可能是 0, subnormal
		RP	舍入后的结果，可能是 0, subnormal, 最小的正 normal 数（单精度：$+2^{-126}$，双精度：$+2^{-1022}$）
		RM	舍入后的结果，可能是 0, subnormal, 最大的负 normal 数（单精度：-2^{-126}，双精度：-2^{-1022}）
O	上溢	RNE	根据中间结果的符号把结果置为+∞或-∞
		RZ	根据中间结果的符号把结果置为最大数
		RP	把负上溢修正为最小负数，把正上溢修正为+∞
		RM	把正上溢修正为最大正数，把负上溢修正为-∞
Z	被 0 除	任何模式	提供一个相应的带符号的无穷大数
V	非法操作	任何模式	提供一个 QNaN

3.1.4.1 非法操作例外（V）

当且仅当没有有效定义的结果时，才会发出无效操作例外通知信号。如果没有触发例外陷入，那么将生成一个 QNaN。有关非法操作例外的具体判定细节请参见 IEEE 754-2008 规范的 7.2 节。

如果例外允许陷入：结果寄存器不被修改，源寄存器保留。

如果例外禁止陷入：如果没有其他例外发生，QNaN 被写入目标寄存器中。

[1] 其实只有除下溢例外之外的其余四个例外严格符合这句话的描述。有关下溢例外的定义请看下文中的具体描述。

3.1.4.2 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。

如果例外允许陷入：结果寄存器不被修改，源寄存器保留。

如果例外禁止陷入：如果没有陷阱发生，结果是有符号的无穷值。

3.1.4.3 上溢例外（O）

把指数域看成无界的对中间结果进行舍入，当得到的结果的绝对值超过了目标格式的最大有限数时，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）

如果例外允许陷入：结果寄存器不被修改，源寄存器保留。

如果例外禁止陷入：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。

3.1.4.4 下溢例外（U）

当检测到结果是一个非零微小值时，会出现下溢例外的情况。检测非零微小值的方式是，在舍入后检测。

舍入后检测，即对于一个非 0 的结果，把指数域看成无界的情况下对中间结果进行舍入，如果舍入后的结果在(-2Emin, 2Emin)中，就认为这个结果是一个非零微小值。（单精度数 Emin = -126，双精度数 Emin = -1022。）

当 FCSR.Enable.U=0 时，若检测到结果是一个非零微小值：

（1）若该浮点操作最终的舍入后的结果是非精确的，就要将 FCSR.Cause 中的 U 和 I 都置为 1；

（2）若该浮点操作最终的舍入后的结果是精确的，那么 FCSR.Cause 中的 U 和 I 都不置 1。

当 FCSR.Enable.U=1 时，若检测到结果是一个非零微小值，不管该浮点操作最终的舍入后的结果是精确的还是非精确的，都会触发浮点例外陷入。

3.1.4.5 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，且上溢例外的使能位没有置位

如果例外允许陷入：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。

如果例外禁止陷入：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。

3.2 基础浮点数指令概述

本节所描述的指令，除了 FLDX.{S/D}、FSTX.{S/D}、FLD{GT/LE}.{S/D}和 FST{GT/LE}.{S/D}这 12 条浮点访存指令仅属于 LA64 架构，其余所有浮点数指令同时适用于 LA32 架构和 LA64 架构。

3.2.1 浮点运算类指令

3.2.1.1 F{ADD/SUB/MUL/DIV}.{S/D}

指令格式：				
	fadd.s	fd, fj, fk	fadd.d	fd, fj, fk
	fsub.s	fd, fj, fk	fsub.d	fd, fj, fk
	fmul.s	fd, fj, fk	fmul.d	fd, fj, fk
	fdiv.s	fd, fj, fk	fdiv.d	fd, fj, fk

FADD.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数加上浮点寄存器 fk 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。浮点加法运算遵循 IEEE 754-2008 标准中 addition(x,y)操作的规范。

FADD.S:

```
FR[fd][31:0] = FP32_addition(FR[fj][31:0], FR[fk][31:0])
```

FADD.D:

```
FR[fd] = FP64_addition(FR[fj], FR[fk])
```

FSUB.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数减去浮点寄存器 fk 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。浮点减法运算遵循 IEEE 754-2008 标准中 subtraction(x,y)操作的规范。

FSUB.S:

```
FR[fd][31:0] = FP32_subtraction(FR[fj][31:0], FR[fk][31:0])
```

FSUB.D:

```
FR[fd] = FP64_subtraction(FR[fj], FR[fk])
```

FMUL.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数乘以浮点寄存器 fk 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。浮点乘法运算遵循 IEEE 754-2008 标准中 multiplication(x,y)操作的规范。

FMUL.S:

```
FR[fd][31:0] = FP32_multiplication(FR[fj][31:0], FR[fk][31:0])
```

FMUL.D:

```
FR[fd] = FP64_multiplication(FR[fj], FR[fk])
```

FDIV.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数除以浮点寄存器 fk 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。浮点除法运算遵循 IEEE 754-2008 标准中 division(x,y)操作的规范。

FDIV.S:

```
FR[fd][31:0] = FP32_division(FR[fj][31:0], FR[fk][31:0])
```

FDIV.D:

```
FR[fd] = FP64_division(FR[fj], FR[fk])
```

当操作数是单精度浮点数时，结果浮点寄存器中的高 32 位可以是任意值。

3.2.1.2 F{MADD/MSUB/NMADD/NMSUB}.{S/D}

指令格式：				
	fmadd.s	fd, fj, fk, fa	fmadd.d	fd, fj, fk, fa
	fmsub.s	fd, fj, fk, fa	fmsub.d	fd, fj, fk, fa
	fnmadd.s	fd, fj, fk, fa	fnmadd.d	fd, fj, fk, fa
	fnmsub.s	fd, fj, fk, fa	fnmsub.d	fd, fj, fk, fa

FMADD.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数相乘，得到的结果加上浮点寄存器 fa 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。

FMADD.S:

```
FR[fd][31:0] = FP32_fusedMultiplyAdd(FR[fj][31:0], FR[fk][31:0], FR[fa][31:0])
```

FMADD.D:

```
FR[fd] = FP64_fusedMultiplyAdd(FR[fj], FR[fk], FR[fa])
```

FMSUB.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数相乘，得到的结果减去浮点寄存器 fa 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果写入到浮点寄存器 fd 中。

FMSUB.S:

```
FR[fd][31:0] = FP32_fusedMultiplyAdd(FR[fj][31:0], FR[fk][31:0], -FR[fa][31:0])
```

FMSUB.D:

```
FR[fd] = FP64_fusedMultiplyAdd(FR[fj], FR[fk], -FR[fa])
```

FNMADD.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数相乘，得到的结果加上浮点寄存器 fa 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果取负后写入到浮点寄存器 fd 中。

FNMADD.S:

```
FR[fd][31:0] = -FP32_fusedMultiplyAdd(FR[fj][31:0], FR[fk][31:0], FR[fa][31:0])
```

FNMADD.D:

```
FR[fd] = -FP64_fusedMultiplyAdd(FR[fj], FR[fk], FR[fa])
```

FNMSUB.{S/D}指令将浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数相乘，得到的结果减去浮点寄存器 fa 中的单精度/双精度浮点数，得到的单精度/双精度浮点数结果取负后写入到浮点寄存器 fd 中。

FNMSUB.S:

```
FR[fd][31:0] = -FP32_fusedMultiplyAdd(FR[fj][31:0], FR[fk][31:0], -FR[fa][31:0])
```

FNMSUB.D:

```
FR[fd] = -FP64_fusedMultiplyAdd(FR[fj], FR[fk], -FR[fa])
```

以上四个浮点融合乘加运算遵循 IEEE 754-2008 标准中 fusedMultiplyAdd(x,y,z)操作的规范。

3.2.1.3 F{MAX/MIN}.{S/D}

指令格式：	fmax.s	fd, fj, fk	fmax.d	fd, fj, fk
	fmin.s	fd, fj, fk	fmin.d	fd, fj, fk

FMAX.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数中的较大者写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 maxNum(x,y)操作的规范。

```
FMAX.S:
    FR[fd][31:0] = FP32_maxNum(FR[fj][31:0], FR[fk][31:0])
FMAX.D:
    FR[fd] = FP64_maxNum(FR[fj], FR[fk])
```

FMIN.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数中的较小者写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 minNum(x,y)操作的规范。

```
FMIN.S:
    FR[fd][31:0] = FP32_minNum(FR[fj][31:0], FR[fk][31:0])
FMIN.D:
    FR[fd] = FP64_minNum(FR[fj], FR[fk])
```

3.2.1.4 F{MAXA/MINA}.{S/D}

指令格式：	fmaxa.s	fd, fj, fk	fmaxa.d	fd, fj, fk
	fmina.s	fd, fj, fk	fmina.d	fd, fj, fk

FMAXA.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数中绝对值较大者写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 maxNumMag(x,y)操作的规范。

```
FMAXA.S:
    FR[fd][31:0] = FP32_maxNumMag(FR[fj][31:0], FR[fk][31:0])
FMAXA.D:
    FR[fd] = FP64_maxNumMag(FR[fj], FR[fk])
```

FMINA.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数与浮点寄存器 fk 中的单精度/双精度浮点数中绝对值较小者写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 minNumMag(x,y)操作的规范。

```
FMINA.S:
    FR[fd][31:0] = FP32_minNumMag(FR[fj][31:0], FR[fk][31:0])
FMINA.D:
    FR[fd] = FP64_minNumMag(FR[fj], FR[fk])
```

3.2.1.5 F{ABS/NEG}.{S/D}

指令格式：

fabs.s	fd, fj		fabs.d	fd, fj
fneg.s	fd, fj		fneg.d	fd, fj

FABS.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，取其绝对值（也即将符号位置为 0，其它部分不变），写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 abs(x)操作的规范。

FABS.S:

```
FR[fd][31:0] = FP32_abs(FR[fj][31:0])
```

FABS.D:

```
FR[fd] = FP64_abs(FR[fj])
```

FNEG.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，取其相反数（也即将符号位取反，其它部分不变），写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 negate(x)操作的规范。

FNEG.S:

```
FR[fd][31:0] = FP32_negate(FR[fj][31:0])
```

FNEG.D:

```
FR[fd] = FP64_negate(FR[fj])
```

3.2.1.6 F{SQRT/RECIP/RSQRT}.{S/D}

指令格式：

fsqrt.s	fd, fj		fsqrt.d	fd, fj
frecip.s	fd, fj		frecip.d	fd, fj
frsqrt.s	fd, fj		frsqrt.d	fd, fj

这些指令是与开方和倒数相关的操作。

FSQRT.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，将其开方后得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。浮点开方运算遵循 IEEE 754-2008 标准中 squareRoot(x)操作的规范。

FSQRT.S:

```
FR[fd][31:0] = FP32_squareRoot(FR[fj][31:0])
```

FSQRT.D:

```
FR[fd] = FP64_squareRoot(FR[fj])
```

FRECIP.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，用 1.0 除以这个浮点数后将得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。相当于 IEEE 754-2008 标准中 division(1.0,x)操作。

FRECIP.S:

```
FR[fd][31:0] = FP32_division(1.0, FR[fj][31:0])
```

FRECIP.D:

```
FR[fd] = FP64_division(1.0, FR[fj])
```

FRSQRT.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，将其开方后得到的单精度/双精度浮点数再用 1.0 除，得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。浮点开方求倒运算遵循 IEEE 754-2008 标准中 rSqrt(x)操作的规范。

FRSQRT.S:

```
FR[fd][31:0] = FP32_division(1.0, FP_squareRoot(FR[fj][31:0]))
```

FRSQRT.D:

```
FR[fd] = FP64_division(1.0, FP_squareRoot(FR[fj]))
```

3.2.1.7 F{SCALEB/LOGB/COPYSIGN}.{S/D}

指令格式：

fscaleb.s	fd, fj, fk	fscaleb.d	fd, fj, fk
flogb.s	fd, fj	flogb.d	fd, fj
fcopysign.s	fd, fj, fk	fcopysign.d	fd, fj, fk

FSCALEB.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数 a，再取浮点寄存器 fk 中的字/双字整数 N，计算 $a*2^N$，得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 scaleB(x, N)操作的规范。

FSCALEB.S:

```
FR[fd][31:0] = FP32_scaleB(FR[fj][31:0], FR[fk][31:0])
```

FSCALEB.D:

```
FR[fd] = FP64_scaleB(FR[fj], FR[fk])
```

FLOGB.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，以 2 为底求它的对数，得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。这两条指令的运算遵循 IEEE 754-2008 标准中 logB(x)操作的规范。

FLOGB.S:

```
FR[fd][31:0] = FP32_logB(FR[fj][31:0])
```

FLOGB.D:

```
FR[fd] = FP64_logB(FR[fj])
```

FCOPYSIGN.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数，将它的符号位改为浮点寄存器 fk 中的单精度/双精度浮点数的符号位，得到的新的单精度/双精度浮点数写入到浮点寄存器 fd 中。浮点复制符号运算遵循 IEEE 754-2008 标准中 copySign(x, y)操作的规范。

FCOPYSIGN.S:

```
FR[fd][31:0] = FP32_copySign(FR[fj][31:0], FR[fk][31:0])
```

FCOPYSIGN.D:

```
FR[fd] = FP64_copySign(FR[fj], FR[fk])
```

3.2.1.8 FCLASS.{S/D}

指令格式：

fclass.s	fd, fj	fclass.d	fd, fj

本指令对浮点寄存器 fj 中的浮点数进行类别的判断，所得的判断结果一共由 10 比特信息组成，每比特的含义如下表所示：

Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7	Bit8	Bit9
SNaN	QNaN	negative value				positive value			
		∞	normal	subnormal	0	∞	normal	subnormal	0

当被判断的数据符合某个比特对应的条件时，结果信息向量的对应比特就会被置为 1。该指令对应 IEEE-754-2008 标准中的 class(x)函数。

FCLASS.S:

```
    FR[fd][31:0] = FP32_class(FR[fj][31:0])
FCLASS.D:
    FR[fd] = FP64_class(FR[fj])
```

3.2.2 浮点比较指令

3.2.2.1 FCMP.cond.{S/D}

指令格式： fcmp.cond.s cc, fj, fk fcmp.cond.d cc, fj, fk

这是一条浮点比较指令，将比较的结果存入指定的状态码（CC）。这条指令的 cond 有 22 种，这些比较条件以及判断的标准在下表中列出来。

助记符	cond	含义	True Condition	QNaN 是否报例外	对应 IEEE 754-2008 函数
CAF	0x0	否	无	否	
CUN	0x8	无法比较	UN		compareQuietUnordered
CEQ	0x4	相等	EQ		compareQuietEqual
CUEQ	0xC	相等或无法比较	UN EQ		
CLT	0x2	小于	LT		compareQuietLess
CULT	0xA	小于或无法比较	UN LT		compareQuietLessUnordered
CLE	0x6	小于等于	LT EQ		compareQuietLessEqual
CULE	0xE	小于等于或无法比较	UN LT EQ		compareQuietNotGreater
CNE	0x10	不等	GT LT		
COR	0x14	有序	GT LT EQ		
CUNE	0x18	无法比较或不等	UN GT LT		compareSignalingNotEqual
SAF	0x1	否	无	是	
SUN	0x9	不是大于小于或等于	UN		
SEQ	0x5	相等	EQ		compareSignalingEqual
SUEQ	0xD	不是大于或小于	UN EQ		
SLT	0x3	小于	LT		compareSignalingLess
SULT	0xB	不是大于或等于	UN LT		compareSignalingLessUnordered
SLE	0x7	小于等于	LT EQ		compareSignalingLessEqual
SULE	0xF	不是大于	UN LT EQ		compareSignalingNotGreater
SNE	0x11	不等	GT LT		
SOR	0x15	有序	GT LT EQ		
SUNE	0x19	无法比较或不等	UN GT LT		

注：UN 表示无法比较、EQ 表示相等、LT 表示小于。当有两个操作数中有至少一个 NaN 时，这两个数就无法比较。

3.2.3 浮点转换指令

3.2.3.1 FCVT.S.D, FCVT.D.S

指令格式： fcvt.s.d fd, fj　　fcvt.d.s fd, fj

FCVT.S.D 指令选择浮点寄存器 fj 中的双精度浮点数转换为单精度浮点数，得到的单精度浮点数写入到浮点寄存器 fd 中。

FCVT.S.D:

```
FR[fd][31:0] = FP32_convertFormat(FR[fj], FP64)
```

FCVT.D.S 指令选择浮点寄存器 fj 中的单精度浮点数转换为双精度浮点数，得到的双精度浮点数写入到浮点寄存器 fd 中。

FCVT.D.S:

```
FR[fd] = FP64_convertFormat(FR[fj][31:0], FP32)
```

浮点格式转换运算遵循 IEEE 754-2008 标准中 convertFormat(x)操作的规范。

3.2.3.2 FFINT.{S/D}.{W/L}, FTINT.{W/L}.{S/D}

指令格式： ffint.s.w fd, fj　　ftint.w.s fd, fj

ffint.s.l fd, fj　　ftint.l.s fd, fj

ffint.d.w fd, fj　　ftint.w.d fd, fj

ffint.d.l fd, fj　　ftint.l.d fd, fj

FFINT.{S/D}.{W/L}指令选择浮点寄存器 fj 中的整数型/长整数型定点数转换为单精度/双精度浮点数，得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。此浮点格式转换运算遵循 IEEE 754-2008 标准中 convertFromInt(x)操作的规范。

FFINT.S.W:

```
FR[fd][31:0] = FP32_convertFromInt(FR[fj][31:0], SINT32)
```

FFINT.S.L:

```
FR[fd][31:0] = FP32_convertFromInt(FR[fj], SINT64)
```

FFINT.D.W:

```
FR[fd] = FP64_convertFromInt(FR[fj][31:0], SINT32)
```

FFINT.D.L:

```
FR[fd] = FP64_convertFromInt(FR[fj], SINT64)
```

FTINT.{W/L}.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数型/长整数型定点数，得到的整数型/长整数型定点数写入到浮点寄存器 fd 中。根据 FCSR 中不同的状态，此浮点格式转换运算遵循的 IEEE 754-2008 标准中的操作见下表。

舍入模式	IEEE 754-2008 标准中的操作
向最近的偶数舍入	convertToIntegerExactTiesToEven(x)
向零方向舍入	convertToIntegerExactTowardZero(x)
向正无穷方向舍入	convertToIntegerExactTowardPositive(x)
向负无穷方向舍入	convertToIntegerExactTowardNegative(x)

```
FTINT.W.S:
    FR[fd][31:0] = FP32convertToSint32(FR[fj][31:0], FCSR.RM)
FTINT.W.D:
    FR[fd] = FP64convertToSint32(FR[fj], FCSR.RM)
FTINT.L.S:
    FR[fd][31:0] = FP32convertToSint64(FR[fj][31:0], FCSR.RM)
FTINT.L.D:
    FR[fd] = FP64convertToSint64(FR[fj], FCSR.RM)
```

3.2.3.3 FTINT{RM/RP/RZ/RNE}.{W/L}.{S/D}

指令格式：				
	ftintrm.w.s	fd, fj	ftintrp.w.s	fd, fj
	ftintrm.w.d	fd, fj	ftintrp.w.d	fd, fj
	ftintrm.l.s	fd, fj	ftintrp.l.s	fd, fj
	ftintrm.l.d	fd, fj	ftintrp.l.d	fd, fj
	ftintrz.w.s	fd, fj	ftintrne.w.s	fd, fj
	ftintrz.w.d	fd, fj	ftintrne.w.d	fd, fj
	ftintrz.l.s	fd, fj	ftintrne.l.s	fd, fj
	ftintrz.l.d	fd, fj	ftintrne.l.d	fd, fj

这些指令用指定的舍入模式将浮点数转换成定点数。

FTINTRM.{W/L}.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数型/长整数型定点数，得到的整数型/长整数型定点数写入到浮点寄存器 fd 中，采用“向负无穷方向舍入”的方式。

```
FTINTRM.W.S:
    FR[fd][31:0] = FP32convertToSint32(FR[fj][31:0], 3)
FTINTRM.W.D:
    FR[fd] = FP64convertToSint32(FR[fj], 3)
FTINTRM.L.S:
    FR[fd][31:0] = FP32convertToSint64(FR[fj][31:0], 3)
FTINTRM.L.D:
    FR[fd] = FP64convertToSint64(FR[fj], 3)
```

FTINTRP.{W/L}.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数型/长整数型定点数，得到的整数型/长整数型定点数写入到浮点寄存器 fd 中,采用“向正无穷方向舍入”的方式。

```
FTINTRP.W.S:
    FR[fd][31:0] = FP32convertToSint32(FR[fj][31:0], 2)
FTINTRP.W.D:
    FR[fd] = FP64convertToSint32(FR[fj], 2)
FTINTRP.L.S:
    FR[fd][31:0] = FP32convertToSint64(FR[fj][31:0], 2)
FTINTRP.L.D:
    FR[fd] = FP64convertToSint64(FR[fj], 2)
```

FTINTRZ.{W/L}.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数型/长整数型定点

数，得到的整数型/长整数型定点数写入到浮点寄存器 fd 中，采用“向零方向舍入”的方式。

FTINTRZ.W.S:

```
FR[fd][31:0] = FP32convertToSint32(FR[fj][31:0], 1)
```

FTINTRZ.W.D:

```
FR[fd] = FP64convertToSint32(FR[fj], 1)
```

FTINTRZ.L.S:

```
FR[fd][31:0] = FP32convertToSint64(FR[fj][31:0], 1)
```

FTINTRZ.L.D:

```
FR[fd] = FP64convertToSint64(FR[fj], 1)
```

FTINTRNE.{W/L}.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数型/长整数型定点数，得到的整数型/长整数型定点数写入到浮点寄存器 fd 中，采用“向最近的偶数舍入”的方式。

FTINTRNE.W.S:

```
FR[fd][31:0] = FP32convertToSint32(FR[fj][31:0], 0)
```

FTINTRNE.W.D:

```
FR[fd] = FP64convertToSint32(FR[fj], 0)
```

FTINTRNE.L.S:

```
FR[fd][31:0] = FP32convertToSint64(FR[fj][31:0], 0)
```

FTINTRNE.L.D:

```
FR[fd] = FP64convertToSint64(FR[fj], 0)
```

上述四个浮点格式转换运算遵循的 IEEE 754-2008 标准中的操作见下表。

指令名称	IEEE 754-2008 标准中的操作
FTINTRNE.{W/L}.{S/D}	convertToIntegerExactTiesToEven(x)
FTINTRZ.{W/L}.{S/D}	convertToIntegerExactTowardZero(x)
FTINTRP.{W/L}.{S/D}	convertToIntegerExactTowardPositive(x)
FTINTRM.{W/L}.{S/D}	convertToIntegerExactTowardNegative(x)

3.2.3.4 FRINT.{S/D}

指令格式： frint.s fd, fj frint.d fd, fj

FRINT.{S/D}指令选择浮点寄存器 fj 中的单精度/双精度浮点数转换为整数数值的单精度/双精度浮点数，得到的单精度/双精度浮点数写入到浮点寄存器 fd 中。此浮点格式转换运算遵循的 IEEE 754-2008 标准中的操作见下表。

舍入模式	IEEE 754-2008 标准中的操作
向最近的偶数舍入	roundToIntegralExact(x)
向零方向舍入	
向正无穷方向舍入	
向负无穷方向舍入	

FRINT.S:

```
FR[fd][31:0] = FP32_roundToInteger(FR[fj])
```

FRINT.D:

```
FR[fd] = FP64_roundToInteger(FR[fj])
```

3.2.4 浮点搬运指令

3.2.4.1 FMOV.{S/D}

指令格式： fmov.s fd, fj fmov.d fd, fj

FMOV.{S/D}将浮点寄存器 fj 的值按单精度/双精度浮点数格式写入到浮点寄存器 fd 中，如果 fj 的值不是单精度/双精度浮点数格式，则结果不确定。

FMOV.S:

```
FR[fd][31:0] = FR[fj][31:0]
```

FMOV.d:

```
FR[fd] = FR[fj]
```

上述指令操作是非算术的，不会引发 IEEE 754 例外，也不修改浮点控制状态寄存器的 Cause 和 Flags 域。

3.2.4.2 FSEL

指令格式： fsel fd, fj, fk, ca

FSEL 指令进行条件赋值操作。

FSEL 执行时，如果条件标志寄存器 ca 的值等于 0 则将浮点寄存器 fj 的值写入到浮点寄存器 fd 中，否则将浮点寄存器 fk 的值写入到浮点寄存器 fd 中。

FSEL:

```
FR[fd] = CFR[ca] ? FR[fk] : FR[fj]
```

3.2.4.3 MOVGR2FR.{W/D}, MOVGR2FRH.W

指令格式： movgr2fr.w fd, rj movgr2fr.d fd, rj

movgr2frh.w fd, rj

MOVGR2FR.W 将通用寄存器 rj 的低 32 位值写入浮点寄存器 fd 的低 32 位中。若浮点寄存器位宽为 64 位，则 fd 的高 32 位值不确定。

MOVGR2FR.W:

```
FR[fd][31:0] = GR[rj][31:0]
```

MOVGR2FRH.W 将通用寄存器 rj 的低 32 位值写入浮点寄存器 fd 的高 32 位中，浮点寄存器 fd 的低 32 位值不变。

MOVGR2FRH.W:

```
FR[fd][63:32] = GR[rj][31:0]
FR[fd][31: 0] = FR[fd][31:0]
```

MOVGR2FR.D 将通用寄存器 rj 的 64 位值写入浮点寄存器 fd。

MOVGR2FR.D:

```
FR[fd] = GR[rj]
```

3.2.4.4 MOVFR2GR.{S/D}, MOVFRH2GR.S

指令格式：movfr2gr.s rd, fj　　movfr2gr.d rd, fj

movfrh2gr.s rd, fj

MOVFR2GR/MOVFRH2GR.S 将浮点寄存器 fj 的低 32 位/高 32 位值符号扩展后写入通用寄存器 rd。

MOVFR2GR.S:

```
GR[rd] = SignExtend(FR[fj][31:0], GRLEN)
```

MOVFRH2GR.S:

```
GR[rd] = SignExtend(FR[fj][63:32], GRLEN)
```

MOVFR2GR.D 将浮点寄存器 fj 的 64 位值写入通用寄存器 rd。

MOVFR2GR.D:

```
GR[rd] = FR[fj]
```

3.2.4.5 MOVGR2FCSR, MOVFCSR2GR

指令格式：movgr2fcsr fcsr, rj

movfcsr2gr rd, fcsr

MOVGR2FCSR 根据通用寄存器 rj 低 32 位的值修改 fcsr 指示的浮点控制状态寄存器对应的软件可写域的值。如果 MOVGR2FCSR 指令修改 FCSR0 使得其 Cause 域的位和对应的 Enables 的位同时为 1，或者修改 FCSR1 的 Enables 域、FCSR2 的 Cause 域，使得 Cause 的位和对应 Enables 位同时为 1，MOVGR2FCSR 指令自身不会触发浮点例外。

MOVGR2FCSR:

```
FCSR[fcsr] = GR[rj][31:0]
```

MOVFCSR2GR 将 fcsr 指示的浮点控制状态寄存器的 32 位值符号扩展后写入通用寄存器 rd。

MOVFCSR2GR:

```
GR[rd] = SignExtend(FCSR[fcsr], GRLEN)
```

如果上述指令中的 fcsr 指示的浮点控制状态寄存器不存在，则结果不确定。

3.2.4.6 MOVFR2CF, MOVCF2FR

指令格式：movfr2cf cd, fj

movcf2fr fd, cj

MOVFR2CF 将浮点寄存器 fj 的最低一比特的值写入条件标志寄存器 cd。

MOVFR2CF:

```
CFR[cd] = FR[fj][0]
```

MOVCF2FR 将条件标志寄存器 cj 的值写入浮点寄存器 fd 的最低一比特。

MOVCF2FR:

```
FR[fd][0] = CFR[cj]
```

3.2.4.7 MOVGR2CF, MOVCF2GR

指令格式：movgr2cf cd, rj

movcf2gr rd, cj

MOVGR2CF 将通用寄存器 rj 的最低一比特的值写入条件标志寄存器 cd。

MOVGR2CF:

```
CFR[cd] = GR[rj][0]
```

MOVCF2GR 将条件标志寄存器 cj 的值写入通用寄存器 rd 的最低一比特。

MOVCF2GR:

```
GR[rd][0] = CFR[cj]
```

3.2.5 浮点分支指令

3.2.5.1 BCEQZ, BCNEZ

指令格式：

bceqz	cj, offs21	
bcnez	cj, offs21	

BCEQZ 对条件标志寄存器 cj 的值进行判断，如果等于 0 则跳转到目标地址，否则不跳转。

BCNEZ 对条件标志寄存器 cj 的值进行判断，如果不等于 0 则跳转到目标地址，否则不跳转。

上述两条分支指令的跳转目标地址是将指令码中的 21 比特立即数 offs21 逻辑左移 2 位后再符号扩展，所得的偏移值加上该分支指令的 PC。

BCEQZ:

```
if CFR[cj]==0 :
    PC = PC + SignExtend({offs21, 2'b0}, GRLEN)
```

BCNEZ:

```
if CFR[cj]!=0 :
    PC = PC + SignExtend({offs21, 2'b0}, GRLEN)
```

不过需要注意的是，上述指令如果在写汇编时采用直接填入偏移值的方式，则汇编表示中的立即数应填入以字节为单位的偏移值，即指令码中 offs21<<2。

3.2.6 浮点普通访存指令

3.2.6.1 FLD.{S/D}, FST.{S/D}

指令格式：

fld.s	fd, rj, si12	fld.d	fd, rj, si12
fst.s	fd, rj, si12	fst.d	fd, rj, si12

FLD.S 从内存取回一个字的数据写入浮点寄存器 fd 的低 32 位。若浮点寄存器位宽为 64 位，则 fd 的高 32 位值不确定。

FLD.D 从内存取回一个双字的数据写入浮点寄存器 fd。

FST.S 将浮点寄存器 fd 中低 32 位字数据写入到内存中。

FST.D 将浮点寄存器 fd 中双字数据写入到内存中。

上述指令的访存地址计算方式是将通用寄存器 rj 中的值与符号扩展后的 12 比特立即数 si12 相加求和。

对于 FLD.{S/D}和 FST.{S/D}指令，无论在何种硬件实现及环境配置情况下，只要其访存地址是自然对

齐的，都不会触发非对齐例外；当访存地址不是自然对齐时，如果硬件实现支持非对齐访存且当前运算环境配置为允许非对齐访存，那么不会触发非对齐例外，否则的话将触发非对齐例外。

```
FLD.S:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    word = MemoryLoad(paddr, WORD)
    FR[fd][31:0] = word
FLD.D:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    doubleword = MemoryLoad(paddr, DOUBLEWORD)
    FR[fd] = doubleword
FST.S:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(FR[fd][31:0], paddr, WORD)
FST.D:
    vaddr = GR[rj] + SignExtend(si12, GRLEN)
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(FR[fd][63:0], paddr, DOUBLEWORD)
```

3.2.6.2 FLDX.{S/D}, FSTX.{S/D}

指令格式：	fldx.s	fd, rj, rk	fldx.d	fd, rj, rk
	fstx.s	fd, rj, rk	fstx.d	fd, rj, rk

FLDX.S 从内存取回一个字的数据写入浮点寄存器 fd 的低 32 位。若浮点寄存器位宽为 64 位，则 fd 的高 32 位值不确定。

FLDX.D 从内存取回一个双字的数据写入浮点寄存器 fd。

FSTX.S 将浮点寄存器 fd 中低 32 位字数据写入到内存中。

FSTX.D 将浮点寄存器 fd 中双字数据写入到内存中。

上述指令的访存地址计算方式是将通用寄存器 rj 中的值与通用寄存器 rk 中的值相加求和。

对于 FLDX.{S/D}和 FSTX.{S/D}指令，无论在何种硬件实现及环境配置情况下，只要其访存地址是自然对齐的，都不会触发非对齐例外；当访存地址不是自然对齐时，如果硬件实现支持非对齐访存且当前运算环境配置为允许非对齐访存，那么不会触发非对齐例外，否则的话将触发非对齐例外。

```
FLDX.S:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
```

```
    paddr = AddressTranslation(vaddr)
    word = MemoryLoad(paddr, WORD)
    FR[fd][31:0] = word
FLDX.D:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    doubleword = MemoryLoad(paddr, DOUBLEWORD)
    FR[fd] = doubleword
FSTX.S:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(FR[fd][31:0], paddr, WORD)
FSTX.D:
    vaddr = GR[rj] + GR[rk]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    MemoryStore(FR[fd][63:0], paddr, DOUBLEWORD)
```

3.2.7 浮点边界检查访存指令

3.2.7.1 FLD{GT/LE}.{S/D}, FST{GT/LE}.{S/D}

指令格式：				
	fldgt.s	fd, rj, rk	fldgt.d	fd, rj, rk
	fldle.s	fd, rj, rk	fldle.d	fd, rj, rk
	fstgt.s	fd, rj, rk	fstgt.d	fd, rj, rk
	fstle.s	fd, rj, rk	fstle.d	fd, rj, rk

FLD{GT/LE}.{S/D}判断有效地址是否越界，从内存取值写入浮点寄存器。

FLD{GT/LE}.S 检查通用寄存器 rj 中的值是否大于/小于等于通用寄存器 rk 中的值，如果条件满足则从内存取回一个字的数据写入浮点寄存器 fd 的低 32 位。若浮点寄存器位宽为 64 位，则 fd 的高 32 位值不确定。

FLD{GT/LE}.D 检查通用寄存器 rj 中的值是否大于/小于等于通用寄存器 rk 中的值，如果条件满足则从内存取回一个双字的数据写入浮点寄存器 fd。

FST{GT/LE}.{S/D}判断有效地址是否越界，将浮点寄存器的值写入内存。

FST{GT/LE}.S 检查通用寄存器 rj 中的值是否大于/小于等于通用寄存器 rk 中的值，如果条件满足则将浮点寄存器 fd 中低 32 位字数据写入到内存中。

FST{GT/LE}.D 检查通用寄存器 rj 中的值是否大于/小于等于通用寄存器 rk 中的值，如果条件满足则将浮点寄存器 fd 中双字数据写入到内存中。

上述指令的访存地址直接来自于通用寄存器 rj 中的值。上述指令的访存地址均要求自然对齐，否则将触发非对齐例外。上述指令如果检查条件不满足则终止访存操作并触发边界检查错误例外。

FLDGT.S:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    word = MemoryLoad(paddr, WORD)
    FR[fd][31:0] = word
else :
    RaiseException(BCE) #Bound Check Error Exception
```

FLDGT.D:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
    FR[fd] = MemoryLoad(paddr, DOUBLEWORD)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

FLDLE.S:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]<=GR[rk] :
    word = MemoryLoad(paddr, WORD)
    FR[fd][31:0] = word
else :
    RaiseException(BCE) #Bound Check Error Exception
```

FLDLE.D:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]<=GR[rk] :
    FR[fd] = MemoryLoad(paddr, DOUBLEWORD)
else :
    RaiseException(BCE) #Bound Check Error Exception
```

FSTGT.S:

```
vaddr = GR[rj]
AddressComplianceCheck(vaddr)
paddr = AddressTranslation(vaddr)
if GR[rj]>GR[rk] :
```

```
        MemoryStore(FR[fd][31:0], paddr, WORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
FSTGT.D:
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]>GR[rk] :
        MemoryStore(FR[fd][63:0], paddr, DOUBLEWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
FSTLE.S:
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(FR[fd][31:0], paddr, WORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
FSTLE.D:
    vaddr = GR[rj]
    AddressComplianceCheck(vaddr)
    paddr = AddressTranslation(vaddr)
    if GR[rj]<=GR[rk] :
        MemoryStore(FR[fd][63:0], paddr, DOUBLEWORD)
    else :
        RaiseException(BCE) #Bound Check Error Exception
```

4 特权资源架构概述

4.1 特权等级

龙芯架构中处理器核分为 4 个特权等级（Privilege LeVel，简称 PLV），分别是 PLV0~PLV3。处理器核当前处于哪个特权等级由 CSR.CRMD 中 PLV 域的值唯一确定。

所有特权等级中，PLV0 是具有最高权限的特权等级，也是唯一可以使用特权指令并访问所有特权资源的特权等级。PLV1~PLV3 这三个特权等级，都不能执行特权指令访问特权资源，不过三个特权等级在 MMU 采用映射地址翻译模式下具有不同的访问权限。

对于 Linux 系统来说，架构中仅 PLV0 级可对应核心态，同时建议以 PLV3 级对应用户态。

4.2 特权指令概述

所有特权指令仅在 PLV0 特权等级下才能访问。仅有一个例外情况，当 CSR.MISC 中的 RPCNTL1/RPCNTL2/RPCNTL3 配置为 1 时，可以在 PLV1/PLV2/PLV3 特权等级下执行 CSRRD 指令读取性能计数器。

4.2.1 CSR 访问指令

4.2.1.1 CSRRD, CSRWR, CSRXCHG

指令格式：

csrrd	rd, csr_num
csrwr	rd, csr_num
csrxchg	rd, rj, csr_num

CSRRD、CSRWR 和 CSRXCHG 指令用于软件访问 CSR。CSRRD 指令将指定 CSR 的值写入到通用寄存器 rd 中。CSRWR 指令将通用寄存器 rd 中的旧值写入到指定 CSR 中，同时将指定 CSR 的旧值更新到通用寄存器 rd 中。CSRXCHG 指令根据通用寄存器 rj 中存放的写掩码信息，将通用寄存器 rd 中的旧值写入到指定 CSR 中对应写掩码为 1 的那些比特，该 CSR 中的其余比特保持不变，同时将该 CSR 的旧值更新到通用寄存器 rd 中。

所有 CSR 寄存器采用独立的寻址空间。上述指令中 CSR 的寻址值来自于指令中的 14 比特立即数 csr_num。CSR 的寻址单位是一个 CSR 寄存器，即 0 号 CSR 的 csr_num 是 0，1 号 CSR 的 csr_num 是 1，以此类推。

所有 CSR 寄存器的位宽要么是 32 位，要么与架构中的 GR 等宽，因此 CSR 访问指令不区分位宽。在 LA32 架构下，所有 CSR 自然都是 32 位宽。在 LA64 架构下，定义中宽度固定为 32 位的 CSR 总是符号扩展后写入到通用寄存器 rd 中的。

当 CSR 访问指令访问一个架构中未定义或硬件未实现的 CSR 时，读动作可返回任意值（推荐返回全 0 值），写动作不修改处理器的任何软件可见状态。需要提请注意的是，CSRWR 和 CSRXCHG 指令不仅包含更新 CSR 的写动作，也包含读取 CSR 旧值的读动作。

4.2.2 IOCSR 访问指令

4.2.2.1 IOCSR{RD/WR}.{B/H/W/D}

指令格式：

```
iocsrrd.b    rd, rj
iocsrrd.h    rd, rj
iocsrrd.w    rd, rj
iocsrrd.d    rd, rj
iocsrwr.b    rd, rj
iocsrwr.h    rd, rj
iocsrwr.w    rd, rj
iocsrwr.d    rd, rj
```

IOCSR{RD/WR}.{B/H/W/D}指令用于访问 IOCSR。

所有 IOCSR 寄存器采用独立的寻址空间，寻址基本单位为字节。所有数据在 IOCSR 空间中采用小尾端存储格式。IOCSR空间采用直接地址映射方式，物理地址直接等于逻辑地址。IOCSR{RD/WR}.{B/H/W/D}指令的 IOCSR 地址来自于通用寄存器 rj。

IOCSRRD.{B/H/W}指令从 IOCSR 空间的指定地址处取回字节/半字/字长度的数据符号扩展后写入到通用寄存器 rd 中，IOCSRRD.D 指令从 IOCSR 空间的指定地址处取回双字长度的数据符号扩展后写入到通用寄存器 rd 中。

IOCSRWR.{B/H/W/D}指令将通用寄存器 rd 中的[7:0]/[15:0]/[31:0]/[63:0]位数据写入到 IOCSR 空间的指定地址开始处。

IOCSRRD.D 和 IOCSRWR.D 指令只出现在 LA64 架构中。

IOCSR 寄存器通常可以被多个处理器核同时访问。多个处理器核上 IOCSR 访问指令的执行满足顺序一致性条件。

4.2.3 Cache 维护指令

4.2.3.1 CACOP

指令格式：

```
cacop    code, rj, si12
```

CACOP 指令主要用于 Cache 的初始化以及 Cache 一致性维护。

通用寄存器 rj 的值加上符号扩展后的 12 位立即数 si12，将得到 CACOP 指令所用的虚拟地址 VA，其将用于指示被操作 Cache 行的位置。

CACOP 指令访问哪个 Cache 以及进行何种 Cache 操作由指令中 5 比特的 code 决定。code[2:0]指示操作的 Cache 对象，code[4:3]指示操作类型。

code[2:0]指示的 Cache 对象与 CPUCFG10 中所标识的 Cache 顺序一致。例如，当 CPUCFG10=0x02C3D 时，code[2:0]=0 表示操作一级私有指令 Cache，code[2:0]=1 表示操作一级私有数据 Cache，code[2:0]=2 表示操作二级私有混合 Cache，code[2:0]=3 表示操作三级共享混合 Cache。

code[4:3]=0 表示用于 Cache 初始化（Store Tag），将指定 Cache 行的 tag 置为全 0。假设被访问的 Cache 有（1<<Way）路，每一路有（1<<Index）个 Cache 行，每个 Cache 行大小为（1<<Offset）个字节，那么采用地址直接索引方式意味着，操作该 Cache 的第 VA[Way-1:0]路的第 VA[Index+Offset-1:Offset]个 Cache 行。

code[4:3]=1 表示采用地址直接索引方式维护 Cache 一致性（Index Invalidate / Invalidate and Writeback）。地址直接索引方式的定义请见上一段的描述。维护一致性的操作是对指定的 Cache 进行无效并写回的操作。如果被操作的是指令 Cache，那么仅需要进行无效操作，并不需要将 Cache 行中的数据写回。写回的数据进入到哪一级存储中由具体实现的 Cache 层次及各级间的包含或互斥关系决定。对于数据 Cache 或混合 Cache，由具体实现决定是否仅在 Cache 行数据为脏时才将其写回。

code[4:3]=2 表示采用查询索引方式维护 Cache 一致性（Hit Invalidate / Invalidate and Writeback）。这里维护 Cache 一致性的操作与上面一段所述一致。所谓查询索引方式，是将 CACOP 指令的 VA 视作一个普通 load 指令去访问待操作的 Cache，如果命中则对命中的 Cache 行进行操作，否则不做任何操作。由于这个查询过程可能涉及虚实地址转换，所以这种情况下 CACOP 指令可能触发 TLB 相关的例外。不过，由于 CACOP 指令操作的对象是 Cache 行，所以这种情况下并不需要考虑地址对齐与否。

code[4:3]=3 属于实现自定义的 Cache 操作，架构规范中不予明确的功能定义。

4.2.4 TLB 维护指令

4.2.4.1 TLBSRCH

指令格式： tlbsrch

这里给出未实现 LVZ 扩展的情况下，TLBSRCH 指令的功能定义。

使用 CSR.ASID 和 CSR.TLBEHI 的信息去查询 TLB。如果有命中项，那么将命中项的索引值写入到 CSR.TLBIDX 的 Index 域，同时将 CSR.TLBIDX 的 NE 位置为 0；如果没有命中项，那么将 CSR.TLBIDX 的 NE 位置为 1。

TLB 中各项的索引值计算规则是，从 0 开始依次递增编号，先 STLB 后 MTLB，STLB 中从第 0 路的第 0 行至最后一行，然后是第 1 路第 0 行至最后一行，直至最后一路的最后一行，MTLB 从第 0 行至最后一行。

4.2.4.2 TLBRD

指令格式： tlbrd

这里给出未实现 LVZ 扩展的情况下，TLBRD 指令的功能定义。

将 CSR.TLBIDX 的 Index 域的值作为索引值去读取 TLB 中的指定项。如果指定位置处是一个有效 TLB 项，那么将该 TLB 项的页表信息写入到 CSR.TLBEHI、CSR.TLBELO0、CSR.TLBELO1 和 CSR.TLBIDX.PS 中，且将 CSR.TLBIDX 的 NE 位置为 0；如果指定位置处是一个无效 TLB 项，需将 CSR.TLBIDX 的 NE 位置为 1，且建议对读出内容进行屏蔽保护，如 CSR.ASID.ASID、CSR.TLBEHI、CSR.TLBELO0、CSR.TLBELO1 和 CSR.TLBIDX.PS 都不更新或全置为 0。

需要注意的是，有效/无效 TLB 项和 TLB 中的页表项有效/无效是两个概念。

如果访问所用的 index 值超过了 TLB 的范围，则处理器的行为不确定。

4.2.4.3 TLBWR

指令格式： tlbwr

这里给出未实现 LVZ 扩展的情况下， TLBWR 指令的功能定义。

TLBWR 指令将 TLB 相关 CSR 中所存放的页表项信息写入到 TLB 中的指定项。被填入的页表项信息来自于 CSR.TLBEHI、CSR.TLBELO0、CSR.TLBELO1 和 CSR.TLBIDX.PS。如果 CSR.TLBIDX.NE=1，那么 TLB 中会被填入一个无效 TLB 项；仅当 CSR.TLBIDX.NE=0 时，TLB 中才会被填入一个有效 TLB 项。

页表项写入 TLB 的位置是由 CSR.TLBIDX 的 Index 域的值指定的。具体的对应规则请参看 TLBSRCH 指令中关于 TLB 中各项索引值的计算规则。如果页表项是要写入 STLB 的，但是 CSR.TLBIDX 的 Index 域的值和 CSR.TLBEHI 中的 VPPN、CSR.TLBIDX.PS 之间发生矛盾，则处理器的行为不确定。

4.2.4.4 TLBFILL

指令格式： tlbfill

这里给出未实现 LVZ 扩展的情况下，TLBFILL 指令的功能定义。

TLBFILL 指令将 TLB 相关 CSR 中所存放的页表项信息填入到 TLB 中。被填入的页表项信息来自于 CSR.TLBEHI、CSR.TLBELO0、CSR.TLBELO1 和 CSR.TLBIDX.PS。如果 CSR.TLBIDX.NE=1，那么 TLB 中会被填入一个无效 TLB 项；仅当 CSR.TLBIDX.NE=0 时，TLB 中才会被填入一个有效 TLB 项。

页表项填入时，首先根据被填入页表项的页大小来决定是写入 STLB 还是 MTLB。当被填入的页表项的页大小与 STLB 所配置的页大小（CSR.STLBPS）相等时将被填入 STLB，否则将被填入 MTLB。页表项被填入到 STLB 的哪一路，或者被填入到 MTLB 的哪一项，是由硬件随机选择的。

4.2.4.5 TLBCLR

指令格式： tlbclr

根据 TLB 相关 CSR 的信息无效 TLB 中的内容，以维持 TLB 与内存之间页表数据的一致性。这里给出未实现 LVZ 扩展的情况下，TLBCLR 指令的功能定义。

当 CSR.TLBIDX.Index 落在 MTLB 范围内(大于等于 STLB 项数)时，执行 TLBCLR，将 MTLB 中所有 G=0 且 ASID 等于 CSR.ASID.ASID 的页表项无效掉。

当 CSR.TLBIDX.Index 落在 STLB 范围内(小于 STLB 项数)时，执行一条 TLBCLR，将 STLB 中由 CSR.TLBIDX.Index 低位所指示的那一组中所有路中等于 G=0 且 ASID 等于 CSR.ASID.ASID 的页表项无效掉。

4.2.4.6 TLBFLUSH

指令格式： tlbflush

根据 TLB 相关 CSR 的信息无效 TLB 中的内容，以维持 TLB 与内存之间页表数据的一致性。这里给出未实现 LVZ 扩展的情况下，TLBFLUSH 指令的功能定义。

当 CSR.TLBIDX.Index 落在 MTLB 范围内(大于等于 STLB 项数)时，执行 TLBFLUSH，将 MTLB 中所有页表项无效掉。

当 CSR.TLBIDX.Index 落在 STLB 范围内(小于 STLB 项数)时，执行一条 TLBFLUSH，将 STLB 中由 CSR.TLBIDX.Index 低位所指示的那一组中所有路中的页表项无效掉。

4.2.4.7 INVTLB

指令格式： invtlb op, rj, rk

INVTLB 指令用于无效 TLB 中的内容，以维持 TLB 与内存之间页表数据的一致性。这里给出未实现 LVZ 扩展的情况下，INVTLB 指令的功能定义。

指令的三个源操作数中，op 是 5 比特立即数，用于指示操作类型。

通用寄存器 rj 的[9:0]位存放无效操作所需的 ASID 信息（称为“寄存器指定 ASID”），其余比特必须填 0。当 op 所指示的操作不需要 ASID 时，应将通用寄存器 rj 设置为 r0。

通用寄存器 rk 中用于存放无效操作所需的虚拟地址信息（称为“寄存器指定 VA”）。当 op 所指示的操作不需要虚拟地址信息时，应将通用寄存器 rk 设置为 r0。

各 op 对应的操作如下表所示，未在表中出现的 op 将触发保留指令例外。

op	操作
0x0	清除所有页表项。
0x1	清除所有页表项。此时操作效果与 op=0 完全一致。
0x2	清除所有 G=1 的页表项。
0x3	清除所有 G=0 的页表项。
0x4	清除所有 G=0，且 ASID 等于寄存器指定 ASID 的页表项。
0x5	清除 G=0，且 ASID 等于寄存器指定 ASID，且 VA 等于寄存器指定 VA 的页表项。
0x6	清除所有 G=1 或 ASID 等于寄存器指定 ASID，且 VA 等于寄存器指定 VA 的页表项。

4.2.5 软件页表遍历指令

4.2.5.1 LDDIR

指令格式： lddir rd, rj, level

LDDIR 指令用于在软件页表遍历过程中目录项的访问。

LDDIR 指令中的 8 比特立即数 level 用于指示当前访问的是哪一级页表。level=1 对应 CSR.PWCL 中的 PT, level=2 对应 CSR.PWCL 中的 Dir1, level=3 对应 CSR.PWCL 中的 Dir2, level=4 对应 CSR.PWCH 中的 Dir3。

如果通用寄存器 rj 的第[6]位是 0，表明此时 rj 中的内容是第 level 级页表的基址的物理地址。这种情况下执行 LDDIR 指令，会根据当前处理的 TLB 重填地址访问 level 级页表，取回其对应的 level+1 级页表的基址，将其写入到通用寄存器 rd 中。

如果通用寄存器 rj 的第[6]位是 1，表明此时 rj 中的内容是一个大页（Huge Page）的页表项。这种情况下，执行 LDDIR 指令后，通用寄存器 rj 中值将被直接写入到通用寄存器 rd 中。

4.2.5.2 LDPTE

指令格式： ldpte rj, req

LDPTE 指令用于在软件页表遍历过程中页表项的访问。

LDPTE 指令中的立即数 seq 用于指示访问的偶数页还是奇数页。访问偶数页时结果将被写入 CSR.TLBRELO0，访问奇数页时结果将被写入 CSR.TLBRELO1。

如果通用寄存器 rj 的第[6]位是 0，表明此时 rj 中的内容是 PTE 那一级页表的基址的物理地址。这种情况下执行 LDPTE 指令，会根据当前处理的 TLB 重填地址访问 PTE 级页表，取回页表项将其写入到对应的 CSR 中。

如果通用寄存器 rj 的第[6]位是 1，表明此时 rj 中的内容是一个大页（Huge Page）的页表项。这种情况下执行 LDPTE 指令，直接将通用寄存器 rj 中的值转换成最终的页表项格式后写入到对应的 CSR 中。

4.2.6 其它杂项指令

4.2.6.1 ERTN

指令格式： ertn

ERTN 指令用于从例外处理返回。

如果所处理的例外是 Debug 例外，将 CSR.DBG 中的 DS 位清 0，同时跳转到 CSR.DERA 所存放的地址处开始取指。

如果所处理的例外是 Debug 例外之外的其它例外，将例外对应的 PPLV、PIE、PWE 等信息更新至 CSR.CRMD 中，同时跳转到例外所对应的返回地址处开始取指。

如果所处理的例外是 Error 类例外，例外对应的 PPLV、PIE 和 PWE 信息来自于 CSR.MERRCTL，例外对应的返回地址来自于 CSR.MERRERA。除此之外，Error 类例外还要将 CSR.MERRCTL 中的 PDA、PPG、PDCAF、PDCAM 信息更新到 CSR.CRMD 中。

如果所处理的例外是 TLB 重填例外，例外对应的 PPLV、PIE 和 PWE 信息来自于 CSR.TLBRSAVE，例外对应的返回地址来自于 CSR.TLBRERA。除此之外，还要将 CSR.CRMD 中的 DA 位清 0、PG 位置 1。

如果所处理的例外不是 Debug 例外、Error 类例外和 TLB 重填例外，那么例外对应的 PPLV、PIE 和 PWE 信息来自于 CSR.PRMD，例外对应的返回地址来自于 CSR.ERA。

执行 ERTN 指令时，如果 CSR.LLBCTL 中的 KLO 位不等于 1，则将 LLbit 置 0，否则 LLbit 不修改。

4.2.6.2 DBCL

指令格式： dbcl code

执行 DBCL 指令将立即进入调试模式。

4.2.6.3 IDLE

指令格式： idle level

执行 IDLE 指令后，处理器核将停止取指进入等待状态，直至其被中断唤醒或被复位。从停止状态被中断唤醒后，处理器核执行的第一条指令是 IDLE 之后的那一条指令。

5 存储管理

5.1 物理地址空间

内存物理地址空间范围是：0~2^{PALEN}-1。

在 LA32 架构下，PALEN 理论上是一个不超过 36 的正整数，由实现决定其具体的值，通常建议为 32。

在 LA64 架构下，PALEN 理论上是一个不超过 60 的正整数，由实现决定其具体的值。

系统软件可以通过 CPUCFG 读取配置字 0x1 的 PALEN 域来确定 PALEN 的具体值。

5.2 虚拟地址空间与地址翻译模式

龙芯架构中虚拟地址空间是线性平整的。对于 PLV0 级来说，LA32 架构下虚拟地址空间大小为 2^{32} 字节，LA64 架构下虚拟地址空间大小为 2^{64} 字节。不过对于 LA64 架构来说，2^{64} 字节大小的虚拟地址空间并不都是合法的，可以认为存在一些虚拟地址的空洞。合法的虚拟地址空间与地址映射模式紧密相关，接下来将结合地址映射模式的定义进行介绍。

龙芯架构的 MMU 支持两种虚实地址翻译模式：直接地址翻译模式和映射地址翻译模式。

当 CSR.CRMD 的 DA=1 且 PG=0 时，处理器核的 MMU 处于直接地址翻译模式。在这种映射模式下，物理地址默认直接等于虚拟地址的[PALEN-1:0]位（不足补 0），除非具体实现中采用了其它优先级更高的虚实地址翻译规则。可以看到此时整个虚拟地址空间都是合法的。处理器复位结束后将进入直接地址翻译模式。

当 CSR.CRMD 的 DA=0 且 PG=1 时，处理器核的 MMU 处于映射地址翻译模式。具体又分为直接映射地址翻译模式（简称“直接映射模式”）和页表映射地址翻译模式（简称“页表映射模式”）两种。翻译地址时将优先看其能否按照直接映射模式进行翻译，无法进行后再按照页表映射模式进行翻译。有关直接映射模式的详细说明请看 5.2.1 节内容，有关页表映射模式的详细说明请看 5.4 节内容。这里重点讲述 LA64 架构下，采用页表映射模式时，虚拟地址空间合法性的判定规则：合法虚拟地址的[63:PALEN]位必须与[PALEN-1]位相同，即[PALEN-1]之上的所有位是其符号扩展，否则将触发地址错（ADE）例外。然而在直接映射模式下，就不需要进行这种地址非法性检查。

5.2.1 直接映射地址翻译模式

当处理器核的 MMU 处于映射地址模式时，还可以通过直接映射配置窗口机制完成虚实地址的直接映射。直接映射配置窗口共设置有四个，前两个窗口可同时用于取指和 load/store 操作，后两个窗口仅用于 load/store 操作。

系统软件通过配置 CSR.DMW0~CSR.DMW3 寄存器来分别设置四个直接映射配置窗口。每个窗口除

了地址范围信息外，还可以配置该窗口在哪些特权等级下可用，以及虚地址落在该窗口上的访存操作的存储访问类型。

在 LA64 架构下，每一个直接映射配置窗口可以配置一个 2^{PALEN} 字节固定大小的虚拟地址空间。当虚地址命中某个有效的直接映射配置窗口时，其物理地址直接等于虚地址的[PALEN-1:0]位。命中的判断方式是：虚地址最高 4 位（[63:60]位）与配置窗口寄存器中的 VSEG 域相等，且当前特权等级在该配置窗口中被允许。

举例来说，在 PALEN 等于 48 的情况下，通过将 DMW0 配置为 0x9000000000000011，那么在 PLV0 级下，0x9000000000000000 ~ 0x9000FFFFFFFFFFFF 这段原本在页映射模式下不合法的虚地址空间，将被映射到物理地址空间 0x0 ~ 0xFFFFFFFFFFFF 上，且存储访问类型是一致可缓存的。

在 LA32 架构下，每一个直接映射配置窗口可以配置一个 2^{29} 字节固定大小的虚拟地址空间。当虚地址命中某个有效的直接映射配置窗口时，其物理地址直接等于虚地址的[28:0]位拼接上该映射窗口所配置的物理地址高位。命中的判断方式是：虚地址的最高 3 位（[31:29]位）与配置窗口寄存器中的[31:29]相等，且当前特权等级在该配置窗口中被允许。

举例来说，通过将 DMW0 配置为 0x80000011，那么在 PLV0 级下，0x80000000 ~ 0x9FFFFFFF 这段地址将直接被映射到物理地址空间 0x0 ~ 0x1FFFFFFF 上，其存储访问类型是一致可缓存的。

此外需要说明的是，通过直接映射配置窗口理论上可以将 LA32 架构下落在 2^{31}~2^{32}-1 地址范围或是 LA64 架构下落在 2^{VALEN}~2^{64}-1 地址范围的一段地址空间配置为 PLV3 级下可访问，此时若应用软件中取指或访存指令的虚地址落在这样的窗口内，不会触发取指地址错例外（ADEF）或访存指令地址错例外（ADEM）。

5.2.2 LA64 架构下的 32 位地址模式

当 LA32 架构上的应用软件二进制在实现 LA64 架构的处理器上运行时，为了获得相同的运行结果，需要对指令中涉及地址的计算进行特殊处理，这就是 LA64 架构下所独有的 32 位地址模式控制。当 CSR.MISC 中的 VA32L1/VA32L2/VA32L3 被置为 1 时，运行在 PLV1/PLV2/PLV3 级下的软件就以 32 位地址模式运行。此时硬件将访存（含取指的访存）虚地址低 32 位零扩展至 64 位之后的值作为访存的虚地址，而且 BL、JIRL 和 PCADD 类指令也会将结果的低 32 位符号扩展至 64 位后再写回到结果寄存器中。

5.2.3 LA64 架构下的虚地址缩减模式

为了在某些应用场合减少页表级数，LA64 架构下还提供了一种虚地址缩减模式。当系统软件将 CSR.RVACFG 寄存器中的 RDVA 配置为 1~8 中的一个值时，映射地址翻译模式下虚拟地址的有效位将按照(VALEN-RDVA)这么多位来处理。例如，在一台 VALEN=48 的处理器上，当 RDVA 配置为 8 时，合法地址的[63:40]位需要是第[39]位的符号扩展。

5.3 存储访问类型

如前文 2.1.7 节所述，龙芯架构下支持三种存储访问类型，分别是：一致可缓存（Coherent Cached，简称 CC）、强序非缓存（Strongly-ordered UnCached，简称 SUC）和弱序非缓存（Weakly-ordered UnCached，简称 WUC）。

当处理器核 MMU 处于直接地址翻译模式时，所有取指的存储访问类型由 CSR.CRMD.DATF 决定，所有 load/store 操作的存储访问类型由 CSR.CRMD.DATM 域决定。

当处理器核 MMU 处于映射地址翻译模式时，存储访问类型的确定分为两种情况。如果取指或 load/store 操作的地址落在某个直接映射配置窗口上，那么该取指或 load/store 操作的存储访问类型由配置该窗口的 CSR 寄存器中的 MAT 域决定。如果取指或 load/store 只能通过页表完成映射，那么其存储访问类型由页表项中的 MAT 域决定。

无论在哪种情况下，存储访问类型控制值的定义是相同的，均是：0——强序非缓存，1——一致可缓存，2——弱序非缓存，3——保留。

5.4 页表映射存储管理

映射地址翻译模式下，除了落在直接映射配置窗口中的地址之外，其余所有合法地址都必须通过页表映射完成虚实地址转换。TLB 作为处理器中存放操作系统页表信息的一个临时缓存，用于加速映射地址翻译模式下的取指和 load/store 操作的虚实地址转换过程。

5.4.1 TLB 的组织结构

龙芯架构下 TLB 分为两个部分，一个是所有表项的页大小相同的单一页大小 TLB（Singular-Page-Size TLB，简称 STLB），另一个是支持不同表项的页大小可以不同的多重页大小 TLB（Multiple-Page-Size TLB，简称 MTLB）。

页大小与 STLB 所配置的页大小相同的页表项能否进入 MTLB，由实现决定，架构规范中不做限制。

在虚实地址转换过程中，STLB 和 MTLB 同时查找。相应地，软件需保证不会出现 MTLB 和 STLB 同时命中的情况，否则处理器行为将不可知。

MTLB 采用全相联查找表的组织形式，STLB 采用多路组相联的组织形式。对于 STLB，如果其有 2^{INDEX} 组，且配置的页大小为 2^{PS} 字节，那么硬件查询 STLB 的过程中，是将虚地址的[PS+INDEX:PS+1]位作为索引值来访问各路信息的。

5.4.2 TLB 的表项

STLB 和 MTLB 的表项格式基本一致，区别仅在于 MTLB 每个表项均包含页大小信息，而 STLB 因为是同一页大小所以 TLB 表项中不再需要重复存放页大小信息。对于 STLB 来说，其存放的页表项的页大小

是由系统软件配置在 CSR.STLBPS 寄存器的 PS 域。

每一个 TLB 表项的格式如图 5-1 所示，包含两个部分：比较部分和物理转换部分。

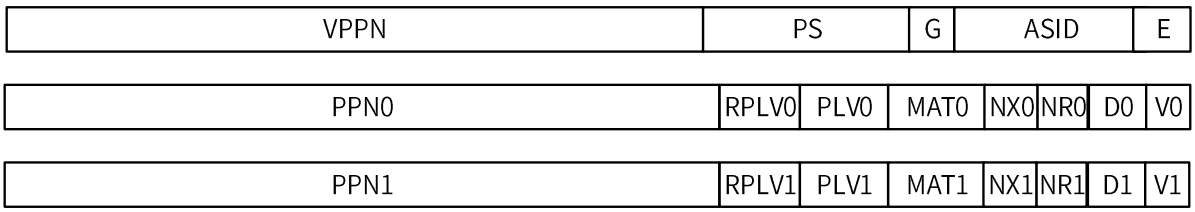


图 5-1 TLB 表项格式

TLB 表项的比较部分包括：

- 存在位(E)，1 比特。为 1 表示所在 TLB 表项非空，可以参与查找匹配。
- 地址空间标识(ASID)，10 比特。地址空间标识用于区分不同进程中的同样的虚地址，避免进程切换时清空整个 TLB 所带来的性能损失。操作系统为每个进程分配唯一的 ASID，TLB 在进行查找时除了比对地址信息一致外，还需要比对 ASID 信息。
- 全局标志位(G)，1 比特。当该位为 1 时，查找时不进行 ASID 是否一致性的检查。当操作系统需要在所有进程间共享同一虚拟地址时，可以设置 TLB 页表项中的 G 位置为 1。
- 页大小(PS)，6 比特。仅在 MTLB 中出现。用于指定该页表项中存放的页大小。数值是页大小的 2 的幂指数。即对于 16KB 大小的页，PS=14。
- 虚双页号(VPPN)，(VALEN-13)比特。在龙芯架构中，每一个页表项存放了相邻的一对奇偶相邻页表信息，所以 TLB 页表项中存放虚页号的是系统中虚页号/2 的内容，即虚页号的最低位不需要存放在 TLB 中。查找 TLB 时在根据被查找虚页号的最低位决定是选择奇数号页还是偶数号页的物理转换信息。

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

- 有效位(V)，1 比特。为 1 表明该页表项是有效的且被访问过的。
- 脏位(D)，1 比特。为 1 表示该页表项项所对应的地址范围内已有脏数据。
- 不可读位(NR)，1 比特。为 1 表示该页表项所在地址空间上不允许执行 load 操作。该控制位仅定义在 LA64 架构下。
- 不可执行位(NX)，1 比特。为 1 表示该页表项所在地址空间上不允许执行取指操作。该控制位仅定义在 LA64 架构下。
- 存储访问类型(MAT)，2 比特。控制落在该页表项所在地址空间上访存操作的存储访问类型。各数值具体含义见 5.3 节。
- 特权等级（PLV），2 比特。该页表项对应的特权等级。当 RPLV=0 时，该页表项可以被任何特权等级不低于 PLV 的程序访问；当 RPLV=1 时，该页表项仅可以被特权等级等于 PLV 的程序访问。
- 受限特权等级使能（RPLV），1 比特。页表项是否仅被对应特权等级的程序访问的控制位。请参看上面 PLV 中的内容。该控制位仅定义在 LA64 架构下。

- 物理页号(PPN), (PALEN-12)比特。当页大小大于 4KB 的时候, TLB 中所存放的 PPN 的[PS-1:12]位可以是任意值。

5.4.3 TLB 的软件管理

龙芯架构下 TLB 的管理涉及软件方面的工作。在本架构规范 1.00 版本中，TLB 重填以及 TLB 与内存页表之间的一致性维护仍全部由软件主导完成。

5.4.3.1 TLB 相关的例外

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

- TLB 重填例外：当访存操作的虚地址在 TLB 中查找没有匹配项时，触发该例外，通知系统软件进行 TLB 重填工作。该例外拥有独立的例外入口、独立的用于维护例外现场的 CSR 以及一套独立的 TLB 访问接口 CSR，意味着该例外允许在其它例外的处理过程中被触发。TLB 重填例外陷入的同时，硬件会自动将 CSR.CRMD 的 DA 置为 1，PG 置为 0，即自动进入直接地址翻译模式，从而避免 TLB 重填例外处理程序自身再次触发 TLB 重填例外，此时例外现场将无法保存与恢复。为了区分 TLB 重填例外陷入后所使用的 CSR 和其它例外可使用的 CSR，TLB 重填例外陷入的同时，硬件还会自动将 CSR.TLBRERA.ISTLBR 位置 1。
- load 操作页无效例外：load 操作的虚地址在 TLB 中找到了匹配项但是匹配页表项的 V=0，将触发该例外。
- store 操作页无效例外：store 操作的虚地址在 TLB 中找到了匹配项但是匹配页表项的 V=0，将触发该例外。
- 取指操作页无效例外：取指操作的虚地址在 TLB 中找到了匹配项但是匹配页表项的 V=0，将触发该例外。
- 页特权等级不合规例外：访存操作的虚地址在 TLB 中找到了匹配且 V=1 的项，但是访问的特权等级不合规，将触发该例外。特权等级不合规体现为，该页表项的 RPLV=0 且 CSR.CRMD.PLV 值大于页表项中的 PLV；或是该页表项的 RPLV=1 且 CSR.CRMD.PLV 不等于页表项中的 PLV。
- 页修改例外：store 操作的虚地址在 TLB 中找到了匹配，且 V=1，且特权等级合规的项，但是该页表项的 D 位为 0，将触发该例外。
- 页不可读例外：load 操作的虚地址在 TLB 中找到了匹配，且 V=1，且特权等级合规的项，但是该页表项的 NR 位为 1，将触发该例外。
- 页不可执行例外：取指操作的虚地址在 TLB 中找到了匹配，且 V=1，且特权等级合规的项，但是该页表项的 NX 位为 1，将触发该例外。

5.4.3.2 TLB 相关的指令

TLB 相关的指令主要涉及对 TLB 的查找、读、写、无效等操作，用于进行 TLB 的填充、更新与一致性维护。具体的指令定义请参看本手册 4.2.4 节和 4.2.5 节中的内容。

5.4.3.3 TLB 相关的 CSR

TLB 相关的 CSR 按照功能主要分为三类，第一类用于非 TLB 重填例外情况下 TLB 的交互接口，第二类用于软硬件页表遍历，第三类用于 TLB 重填例外。

第一类包括：

- BADV
- TLBEHI
- TLBELO0
- TLBELO1
- TLBIDX
- ASID
- STLBPS

第二类包括：

- PGDL
- PGDH
- PGD
- PWCL
- PWCH

第三类包括：

- TLBRENTRY
- TLBRERA
- TLBRBADV
- TLBREHI
- TLBRELO0
- TLBRELO1
- TLBRPRMD
- TLBRSAVE

上述各 CSR 寄存器与 TLB 交互的细节，请参考 7.4 节中各 CSR 的详细定义。

5.4.3.4 TLB 的初始化

龙芯架构允许不实现 TLB 的硬件初始化，让启动阶段的软件通过执行“INVTLB r0, r0”来完成这一功能。

5.4.4 基于 TLB 的虚实地址转换过程

这里对基于 TLB 所进行的虚实地址转换过程进行描述。下面以伪码形式介绍时的先查 STLB 后查 MTLB 的过程仅为描述方便，处理器硬件实现时可以同时对 STLB 和 MTLB 进行查找。

```
# va：待查找虚地址
# mem_type：访存操作类型，FETCH 是取指操作，LOAD 是 load 操作，STORE 是 store 操作
# plv：当前特权等级，即 CSR.CRMD.PLV 的值
# pa：转换后的物理地址
# mat：转换后得到的存储访问类型
# VALEN：虚地址的有效位数
# PALEN：物理地址的有效位数
# STLB[][]：STLB[N][M]表示 STLB 第 N 路第 M 项
# STLB_WAY：STLB 的路数
# STLB_INDEX：STLB 每一路组数的 2 的幂指数，即每一路有 2STLB_INDEX 组
# MTLB[]：MTLB[N]表示 MTLB 的第 N 项
# MTLB_ENTRIES：MTLB 的项数

# 查找 STLB
stlb_found = 0
stlb_ps = CSR.STLBPS.PS
stlb_idx = va[stlb_ps+STLB_INDEX-1:stlb_ps]
for way in range(STLB_WAY):
    if (STLB[way][stlb_idx].E==1) and
     ((STLB[way][stlb_idx].G==1) or (STLB[way][stlb_idx].ASID==CSR.ASID.ASID))
 and
     (STLB[way][stlb_idx].VPPN[VALEN-1:stlb_ps+1]==va[VALEN-1:stlb_ps+1]) :
        if (stlb_found==0) :
            stlb_found = 1
            if (va[stlb_ps]==0) :
                sfound_v = STLB[way][stlb_idx].V0
                sfound_d = STLB[way][stlb_idx].D0
                sfound_nr = STLB[way][stlb_idx].NR0
                sfound_nx = STLB[way][stlb_idx].NX0
                sfound_mat = STLB[way][stlb_idx].MAT0
                sfound_plv = STLB[way][stlb_idx].PLV0
                sfound_rplv = STLB[way][stlb_idx].RPLV0
                sfound_ppn = STLB[way][stlb_idx].PPN0
            else :
                sfound_v = STLB[way][stlb_idx].V1
                sfound_d = STLB[way][stlb_idx].D1
                sfound_nr = STLB[way][stlb_idx].NR1
                sfound_nx = STLB[way][stlb_idx].NX1
```

```
                sfound_mat = STLB[way][stlb_idx].MAT1
                sfound_plv = STLB[way][stlb_idx].PLV1
                sfound_rplv = STLB[way][stlb_idx].RPLV1
                sfound_ppn = STLB[way][stlb_idx].PPN1
        else :
            # 出现多项命中，处理器运行结果不确定

# 查找 MTLB
mtlb_found = 0
for i in range(MTLB_ENTRIES) :
    if (MTLB[i].E==1) and
     ((MTLB[i].G==1) or (MTLB[i].ASID==CSR.ASID.ASID)) and
     (MTLB[i].VPPN[VALEN-1:MTLB[i].PS+1]==va[VALEN-1: MTLB[i].PS+1]) :
        if (mtlb_found==0) :
            mtlb_found = 1
            mfound_ps = MTLB[i].PS
            if (va[mfound_ps]==0) :
                mfound_v = MTLB[i].V0
                mfound_d = MTLB[i].D0
                mfound_nr = MTLB[i].NR0
                mfound_nx = MTLB[i].NX0
                mfound_mat = MTLB[i].MAT0
                mfound_plv = MTLB[i].PLV0
                mfound_rplv = MTLB[i].RPLV0
                mfound_ppn = MTLB[i].PPN0
            else :
                mfound_v = MTLB[i].V1
                mfound_d = MTLB[i].D1
                mfound_nr = MTLB[i].NR1
                mfound_nx = MTLB[i].NX1
                mfound_mat = MTLB[i].MAT1
                mfound_plv = MTLB[i].PLV1
                mfound_rplv = MTLB[i].RPLV1
                mfound_ppn = MTLB[i].PPN1
        else:
            #出现多项命中，处理器运行结果不确定

if (stlb_found==1) and (mtlb_found==1) :
    #出现多项命中，处理器运行结果不确定
elif (stlb_found==1) :
    found_v = sfound_v
    found_d = sfound_d
    found_nr = sfound_nr
```

```
    found_nx = sfound_nx
    found_mat = sfound_mat
    found_plv = sfound_plv
    found_rplv = sfound_rplv
    found_ppn = sfound_ppn
    found_ps = stlb_ps
elif (mtlb_found==1) :
    found_v = mfound_v
    found_d = mfound_d
    found_nr = mfound_nr
    found_nx = mfound_nx
    found_mat = mfound_mat
    found_plv = mfound_plv
    found_rplv = mfound_rplv
    found_ppn = mfound_ppn
    found_ps = mfound_ps
else :
    SignalException(TLBR)        #报 TLB 重填例外

if (found_v==0) :
    case mem_type :
        FETCH : SignalException(PIF)      #报取指操作页无效例外
        LOAD  : SignalException(PIL)      #报 load 操作页无效例外
        STORE : SignalException(PIS)      #报 store 操作页无效例外
elif (mem_type==FETCH) and (found_nx==1) :
    SignalException(PNX)         #报页不可执行例外
elif ((found_rplv==0) and (plv > found_plv)) or
     ((found_rplv==1) and (plv!= found_plv)) :
    SignalException(PPI)         #报页特权等级不合规例外
elif (mem_type==LOAD) and (found_nr==1) :
    SignalException(PNR)         #报页不可读例外
elif (mem_type==STORE) and (found_d==0)
     and ((plv==3) or (CSR.MISC[16+plv]==0)) :  #禁止写允许检查功能未开启
    SignalException(PME)         #报页修改例外
else :
    pa = {found_ppn[PALEN-1:found_ps], va[found_ps-1:0]}
    mat = found_mat
■
```

5.4.5 页表遍历过程所支持的多级页表结构

无论是使用 LDDIR 和 LDPTE 指令实现的软件页表遍历还是硬件页表遍历，其所支持的多级页表结构是一样的，如图 5-2 所示。

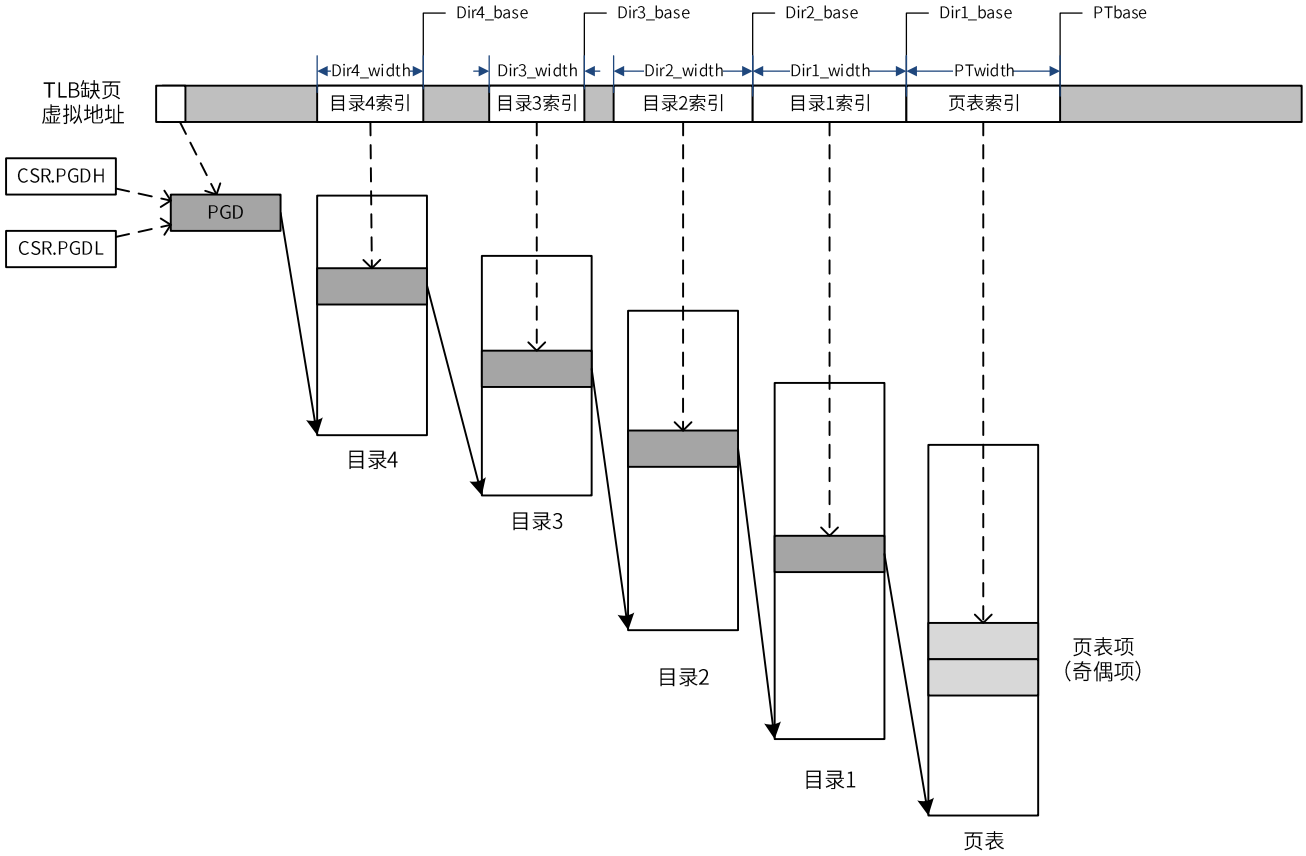


图 5-2 页表遍历过程所支持的多级页表结构

被遍历页表最顶层目录（全局目录，Global Directory）的基址 PGD 需要根据被查询虚地址的第(PALEN-1)位决定。当该位为 0 时，PGD 来自于 CSR.PGDL；当该位为 1 时，PGD 来自于 CSR.PGDH。当这意味着整个页表结构为(PALEN-1)比特。

各级目录项和页表项的规格由系统软件配置在 CSR.PWCL 和 CSR.PWCH 中。

无论是使用 LDDIR 和 LDPTE 指令进行页表遍历还是使用硬件进行页表遍历，系统软件需要按照如下格式定义页表项。

基本页页表项格式：

63	62	61	PALEN-1 … 12	8	7	6	5 4	3 2	1	0
RPLV	NX	NR	PA[PALEN-1:12]	W	P	G	MAT	PLV	D	V

大页页表项格式：

63	62	61	PALEN-1 … 24	12	8	7	6	5 4	3 2	1	0
RPLV	NX	NR	PA[PALEN-1:24]	G	W	P	H	MAT	PLV	D	V

上述页表项格式的定义中，大页的页表项和基本页的页表项在格式上的主要区别是：（1）目录项的第 6 位是大页表项标志位 H，为 1 表示此时的目录项实际上存放了一个大页的页表项信息；（2）基本页页表项的 G 位在第 6 位，而大页页表项的 G 位在第 12 位。

上述两种格式中未明确定义的比特位，将被 LDDIR、LDPTE 指令或硬件页表遍历逻辑自动忽略。

上述页表项格式中定义的“P”和“W”两个域分别代表物理页是否存在，以及该页是否可写。这些信息虽然不填入到 TLB 表项中，但用于页表遍历的处理过程。

由于 TLB 表项采用的双页的存储结构，对于大页的页表项（其只有一项），硬件页表重填逻辑或者软件的 LDPTE 指令会将根据大页的页表项信息自动拆分出两个尺寸折半的页表项后填入 TLB 中。例如，标准页大小为 16KB，则此时第一级大页的大小通常为 32MB，软件页表遍历过程执行完“LDPTE rj, 0”和“LDPTE rj, 1”指令后，TLB 中将被填入两个 16MB 页大小的页表项，软件无需特殊干预。

因为在 TLB 重填例外（TLBR）处理过程中地址映射是处于直接地址翻译模式，所以 PGD 以及内存中页表的目录项中所配置的地址必须为物理地址。

6 例外与中断

6.1 中断

6.1.1 中断类型

龙芯架构下的中断采用线中断的形式。每个处理器核内部可记录 13 个线中断，分别是：1 个核间中断（IPI），1 个定时器中断（TI），1 个性能监测计数溢出中断（PMI），8 个硬中断（HWI0~HWI7），2 个软中断（SWI0~SWI1）。所有的线中断都是电平中断，且都是高电平有效。

核间中断的中断输入来自于核外的中断控制器，其被处理器核采样记录在 CSR.ESTAT.IS[12]位。

定时器中断的中断源来自于核内的恒定频率定时器。当恒定频率定时器倒计时至全 0 值时，该中断被置起。置起后的定时器中断被处理器核采样记录在 CSR.ESTAT.IS[11]位。清除定时器中断需要通过软件向 CSR.TICLR 寄存器的 TI 位写 1 来完成。

性能计数器溢出中断的中断源来自于核内的性能计数器。当任一个中断使能开启的性能计数器的计数值的第[63]位为 1 时，该中断将被置起。置起后的性能计数器溢出中断被处理器核采样记录在 CSR.ESTAT.IS[10]位。清除性能计数器溢出中断需要将引起中断的那个性能计数器的第[63]位置为 0，或者关闭该性能计数器的中断使能。

硬中断的中断源来自于处理器核外部，其直接来源通常是核外的中断控制器。8 个硬中断 HWI[7:0]被处理器核采样记录在 CSR.ESTAT.IS[9:2]位。

软中断的中断源来自于处理器核内部，软件通过 CSR 指令对 CSR.ESTAT.IS[1:0]写 1 则置起软中断，写 0 则清除软中断。

中断在 CSR.ESTAT.IS 域中记录的位置的索引值也被称为中断号（Int Number）。SWI0 的中断号等于 0，SWI1 的中断号等于 1，……，IPI 的中断号等于 12。

6.1.2 中断优先级

同一时刻多个中断的响应采用固定优先级仲裁机制，中断号越大优先级越高。因此 IPI 的优先级最高，TI 次之，……，SWI0 的优先级最低。

6.1.3 中断入口

中断被处理器硬件标记到指令上以后就被当作一种例外进行处理，因此中断入口的计算遵循普通例外入口的计算规则。有关普通例外入口的计算规则请参看 6.2.1 节的介绍。需要补充说明的是，在计算入口地址时，中断对应的例外号是其自身的中断号加上 64。即 0 号中断 SWI0 对应的例外号是 64，1 号中断 SWI1

对应的例外号是 65，……，以此类推。

6.1.4 处理器硬件响应中断的处理过程

各中断源发来的中断信号被处理器采样至 CSR.ESTAT.IS 域中，这些信息与软件配置在 CSR.ECFG.LIE 域中的局部中断使能信息按位与，得到一个 13 位中断向量 int_vec。当 CSR.CRMD.IE=1 且 int_vec 不为全 0 时，处理器认为有需要响应的中断，于是从执行的指令流中挑选出一条指令，将其标记上一种特殊的例外——中断例外。

随后处理器硬件的处理过程与普通例外的处理过程一样，请参看 6.2.3 节中的介绍。

6.2 例外

6.2.1 例外入口

TLB 重填例外的入口来自于 CSR.TLBRENTRY。

机器错误例外的入口来自于 CSR.MERRENTRY。

除上述两种例外之外的例外称为普通例外，其入口地址采用“入口页号 | 页内偏移”的计算方式。这里“|”是按位或运算。

所有普通例外入口的入口页号相同，均来自于 CSR.EENTRY。

普通例外入口的偏移由中断偏移的模式和例外号（ecode）共同决定，其值等于 $2^{(CSR.ECFG.VS+2)} \times (ecode+64)$。除中断外，普通例外的 ecode 值请见表 7-8 中 Ecode 栏；中断的 ecode 是其中断号加上 64。

当 CSR.ECFG.VS=0 时，所有普通例外的入口相同，此时需要软件通过 CSR.ESTAT 中的 Ecode、IS 域的信息来判断具体的例外类型。当 CSR.ECFG.VS !=0 时，不同的中断源具有不同的例外入口，软件无需通过访问 CSR.ESTAT 来确认例外类型。

由于例外入口是基址“按位或”上偏移值，当 CSR.ECFG.VS !=0 时，软件在分配例外入口基址时需要确保所有可能的偏移值都不会超出入口基址低位所对应的边界对齐空间。

6.2.2 例外优先级

例外优先级遵循两个基本原则：其一，中断的优先级高于例外；其二，对于例外，取指阶段检测出的优先级最高，译码阶段检测出的优先级次之，执行阶段检测出的优先级再次之。

对于取指阶段检测出的例外：取指 Watch 例外优先级最高，取指地址错例外优先级次之，取指 TLB 相关例外优先级再次之，取指的机器错误例外优先级最低。

译码阶段可检测出的例外彼此互斥，故无需考虑其间的优先级。

执行阶段仅访存指令或同时触发多种例外，其优先级从高到低依次为：要求地址对齐的访存指令因地址

不对齐而产生的地址对齐错例外(ALE) > 地址错例外(ADE) >边界约束检查错例外[1](BCE) > TLB 相关的例外[2] > 允许地址非对齐的访存指令因地址跨越了不同存储访问类型的两个页时而产生的地址对齐错例外(ALE)。

6.2.3 普通例外硬件处理通用过程

不同的普通例外，处理器硬件在处理时可能存在一些细节上的差异，这里对所有普通例外共有的通用处理过程进行描述。

当触发普通例外时，处理器硬件会进行如下操作：

- 将 CSR.CRMD 的 PLV、IE 分别存到 CSR.PRMD 的 PPLV、PIE 中，然后将 CSR.CRMD 的 PLV 置为 0，IE 置为 0；
- 对于支持 Watch 功能的实现，还要将 CSR.CRMD 的 WE 存到 CSR.PRMD 的 PWE 中，然后将 CSR.CRMD 的 WE 置为 0；
- 将触发例外指令的 PC 值记录到 CSR.ERA 中；
- 跳转到例外入口处取指。

当软件执行 ERTN 指令从普通例外执行返回时，处理器硬件会完成如下操作：

- 将 CSR.PRMD 中的 PPLV、PIE 值恢复到 CSR.CRMD 的 PLV、IE 中；
- 对于支持 Watch 功能的实现，还要将 CSR.PRMD 中的 PWE 值恢复到 CSR.CRMD 的 WE 中；
- 跳转到 CSR.ERA 所记录的地址处取指。

针对上述硬件实现，软件在例外处理过程的中途如果需要开启中断，需要保存 CSR.PRMD 中的 PPLV、PIE 等信息，并在例外返回前，将所保存的信息恢复到 CSR.PRMD 中。

6.2.4 TLB 重填例外硬件处理过程

当触发 TLB 重填例外时，处理器硬件会进行如下操作：

- 将 CSR.CRMD 的 PLV、IE 分别存到 CSR.TLBRPRMD 的 PPLV、PIE 中，然后将 CSR.CRMD 的 PLV 置为 0，IE 置为 0，DA 置为 1，PG 置为 0；
- 对于支持 Watch 功能的实现，还要将 CSR.CRMD 的 WE 存到 CSR.TLBRPRMD 的 PWE 中，然后将 CSR.CRMD 的 WE 置为 0；
- 将触发例外指令的 PC 的[GRLEN-1:2]位记录到 CSR.TLBRERA 的 ERA 域中，将 CSR.TLBRERA 的 IsTLBR 置为 1；
- 将触发该例外的访存虚地址（如果是取指触发的则就是 PC）记录到 CSR.TLBRBADV 中，将虚地址的[PALEN-1:13]位记录到 CSR.TLBREHI 的 VPPN 域中；

[1] 仅当是 Bound 类访存指令时才会产生。

[2] 除 AM*类原子访存指令外，其余所有访存指令只会产生唯一一种 TLB 相关例外，但 AM*类原子访存指令可能同时检测出页不可读例外和页修改例外，此时页不可读例外优先于页修改例外触发。

- 跳转到 CSR.TLBRENTTRY 所配置的例外入口处取指。

当软件执行 ERTN 指令从 TLB 重填例外执行返回时，处理器硬件会完成如下操作：

- 将 CSR.TLBRPRMD 中的 PPLV、PIE 值恢复到 CSR.CRMD 的 PLV、IE 中；
- 对于支持 Watch 功能的实现，还要将 CSR.TLBRPRMD 中的 PWE 值恢复到 CSR.CRMD 的 WE 中；
- 将 CSR.CRMD 的 DA 置为 0，PG 置为 1；
- 将 CSR.TLBRERA 的 IsTLBR 置为 0；
- 跳转到 CSR.TLBRERA 所记录的地址处取指。

6.2.5 机器错误例外硬件处理过程

当触发机器错误例外时，处理器硬件会进行如下操作：

- 将 CSR.CRMD 的 PLV、IE、DA、PG、DATF、DATM 分别存到 CSR.MERRCTL 的 PPLV、PIE、PDA、PPG、PDATF、PDATM 中，然后将 CSR.CRMD 的 PLV 置为 0，IE 置为 0，DA 置为 1，PG 置为 0，DATF 置为 0，DATM 置为 0；
- 对于支持 Watch 功能的实现，还要将 CSR.CRMD 的 WE 存到 CSR.MERRCTL 的 PWE 中，然后将 CSR.CRMD 的 WE 置为 0；
- 将触发例外指令的 PC 记录到 CSR.MERRERA 中；
- 将 CSR.MERRCTL 的 IsMERR 位置为 1；
- 将校验的具体错误信息记录到 CSR.MERRINFO1 和 CSR.MERRINFO2 中；
- 跳转到 CSR.MERRENTRY 所配置的例外入口处取指。

当软件执行 ERTN 指令从机器错误例外执行返回时，处理器硬件会完成如下操作：

- 将 CSR.MERRCTL 中的 PPLV、PIE、PDA、PPG、PDATF、PDATM 值恢复到 CSR.CRMD 的 PLV、IE、DA、PG、DATF、DATM 中；
- 对于支持 Watch 功能的实现，还要将 CSR.MERRCTL 中的 PWE 值恢复到 CSR.CRMD 的 WE 中；
- 将 CSR.MERRCTL 的 IsMERR 位置为 0；
- 跳转到 CSR.MERRERA 所记录的地址处取指。

6.3 复位

复位将重新处理器核中的所有逻辑，将电路置于确定的状态。这里将给出复位后处理器的状态的定义。

复位后第一条指令的 PC 是 0x1C000000。由于复位撤销后 MMU 一定处于直接地址翻译模式，所以复位后所取的第一条指令的物理地址也是 0x1C000000。

复位撤销后，处于确定状态的寄存器内容有：

- CSR.CRMD 的 PLV=0，IE=0，DA=1，PG=0，DATF=0，DATM=0，WE=0；
- CSR.EUEN 的 FPUen、VPUen、XVPUen、BTUen 均为 0；
- CSR.MISC 中的所有可配置位均为 0；
- CSR.ECFG 中的 VS 和 LIE 均为 0；
- CSR.ESTAT 中 IS[1:0]均为 0；
- CSR.RVACFG 中的 RDVA=0；
- CSR.TCFG 的 En=0；
- CSR.LLBCTL 的 KLO=0；
- CSR.TLBRERA 的 IsTLBR=0；
- CSR.ERRCTL 的 IsMERR=0；
- 所有实现的 CSR.DMW 中的 PLV0~PLV3 均为 0；
- 所有实现的 CSR.PMCFG 中除 EvCode 之外的所有可配置位均为 0；
- 所有实现的数据断点控制 CSR 中的所有可配置位均为 0；
- 所有实现的指令断点控制 CSR 中的所有可配置位均为 0；
- CSR.DBG 中的 DS=0。

除了上述指定的内容外，复位撤销后，处理器中其它软件可见的寄存器的值都是不确定的，软件在使用前都要将其状态置于确定状态。

TLB 和 Cache 在复位期间是否进行硬件复位由实现决定，启动软件可以通过处理器提供的配置信息来决定是否需要进行软件复位。

7 控制状态寄存器

7.1 控制状态寄存器一览

表 7-1 控制状态寄存器一览表

地址	名称	
0x0	当前模式信息	CRMD
0x1	例外前模式信息	PRMD
0x2	扩展部件使能	EUEN
0x3	杂项控制	MISC
0x4	例外配置	ECFG
0x5	例外状态	ESTAT
0x6	例外返回地址	ERA
0x7	出错虚地址	BADV
0x8	出错指令	BADI
0xc	例外入口地址	EENTRY
0x10	TLB 索引	TLBIDX
0x11	TLB 表项高位	TLBEHI
0x12	TLB 表项低位 0	TLBELO0
0x13	TLB 表项低位 1	TLBELO1
0x18	地址空间标识符	ASID
0x19	低半地址空间全局目录基址	PGDL
0x1A	高半地址空间全局目录基址	PGDH
0x1B	全局目录基址	PGD
0x1C	页表遍历控制低半部分	PWCL
0x1D	页表遍历控制高半部分	PWCH
0x1E	STLB 页大小	STLBPS
0x1F	缩减虚地址配置	RVACFG
0x20	处理器编号	CPUID
0x21	特权资源配置信息 1	PRCFG1

地址	名称	
0x22	特权资源配置信息 2	PRCFG2
0x23	特权资源配置信息 3	PRCFG3
0x30+n (0≤n≤15)	数据保存	SAVEn
0x40	定时器编号	TID
0x41	定时器配置	TCFG
0x42	定时器值	TVAL
0x43	计时器补偿	CNTC
0x44	定时中断清除	TICLR
0x60	LLBit 控制	LLBCTL
0x80	实现相关控制 1	IMPCTL1
0x81	实现相关控制 2	IMPCTL2
0x88	TLB 重填例外入口地址	TLBRENTRY
0x89	TLB 重填例外出错虚地址	TLBRBADV
0x8A	TLB 重填例外返回地址	TLBRERA
0x8B	TLB 重填例外数据保存	TLBRSAVE
0x8C	TLB 重填例外表项低位 0	TLBRELO0
0x8D	TLB 重填例外表项低位 1	TLBRELO1
0x8E	TLB 重填例外表项高位	TLBREHI
0x8F	TLB 重填例外前模式信息	TLBRPRMD
0x90	机器错误控制	MERRCTL
0x91	机器错误信息 1	MERRINFO1
0x92	机器错误信息 2	MERRINFO2
0x93	机器错误例外入口地址	MERRENTRY
0x94	机器错误例外返回地址	MERRERA
0x95	机器错误例外数据保存	MERRSAVE
0x98	高速缓存标签	CTAG
0x180+n (0≤n≤3)	直接映射配置窗口 n	DMWn
0x200+2n (0≤n≤31)	性能监测配置 n	PMCFGn
0x201+2n (0≤n≤31)	性能监测计数器 n	PMCNTn
0x300	load/store 监视点整体控制	MWPC
0x301	load/store 监视点整体状态	MWPS
0x310+8n (0≤n≤7)	load/store 监视点 n 配置 1	MWPnCFG1
0x311+8n (0≤n≤7)	load/store 监视点 n 配置 2	MWPnCFG2

地址	名称	
0x312+8n (0≤n≤7)	load/store 监视点 n 配置 3	MWPnCFG3
0x313+8n (0≤n≤7)	load/store 监视点 n 配置 4	MWPnCFG4
0x380	取指监视点整体控制	FWPC
0x381	取指监视点整体状态	FWPS
0x390+8n (0≤n≤7)	取指监视点 n 配置 1	FWPnCFG1
0x391+8n (0≤n≤7)	取指监视点 n 配置 2	FWPnCFG2
0x392+8n (0≤n≤7)	取指监视点 n 配置 3	FWPnCFG3
0x393+8n (0≤n≤7)	取指监视点 n 配置 4	FWPnCFG4
0x500	调试寄存器	DBG
0x501	调试例外返回地址	DERA
0x502	调试数据保存	DSAVE

7.2 控制状态寄存器访问特性说明

7.2.1 读写属性

在本手册后续关于控制状态寄存器域定义说明中会有各个域“读写”属性的定义。该“读写”属性主要从软件访问的视角进行定义，具体分为四种类型：

- RW——软件可读、可写。除在定义中明确指出的会导致处理器执行结果不确定的非法值，软件可以写入任意值。通常情况下，软件对这些域进行先写后读的操作，读出的应该是写入的值。但是，当所访问的域可以被硬件更新时，或者执行读、写操作的两条指令之间有中断发生，则有可能出现读出值与写入值不一致的情况。
- R——软件只读。软件写这些域不会更新其内容，且不产生其它任何副作用。
- R0——软件读取这些域永远返回 0。但是同时软件必须保证，要么通过设置 CSR 写屏蔽位避免更新这些域，要么在更新这些域时必须要写入 0 值。这一要求是为了确保软件向后兼容。对于硬件实现来说，标记这种属性的域将禁止软件写入。
- W1——软件写 1 有效。软件对这些域写 0 不会将其清 0，且不产生其它任何副作用。同时，定义为该属性的域的读出值没有任何软件意义，软件应该无视这些读出值。

7.2.2 LA32 与 LA64 架构下控制状态寄存器位宽的异同

所有控制状态寄存器的位宽，或者固定为 32 位，或者与所实现的是 LA32 还是 LA64 相关。对于第一种类别的寄存器，其在 LA64 架构下被 CSR 指令访问时，读返回的是符号扩展至 64 位后的值，写的时候

高 32 位的值自动被硬件忽略。对于第二种类型，定义将明确指出 LA32 和 LA64 架构下的差异。

7.2.3 未定义及未实现的控制状态寄存器的访问效果

当软件使用 CSR 指令访问的 CSR 对象是架构规范中未定义的，或者是架构规范中定义的可实现项但是具体硬件未实现的，此时读动作返回的可以是任意值，但写动作不应改变软件可见的处理器状态。

尽管软件用 CSRWR 或 CSRXCHG 指令写这些未定义或未实现的控制状态寄存器除了将通用寄存器 rd 置为任一无意义值之外并不会改变其它软件可见的处理器状态，但如果想确保向后兼容，则软件不应主动写这些寄存器。

7.3 控制状态寄存器相关所引发的冲突

控制状态寄存器相关所引发的冲突由硬件负责维护，软件无需添加栅障类指令进行冲突规避。

7.4 基础控制状态寄存器

7.4.1 当前模式信息（CRMD）

该寄存器中的信息用于决定处理器核当前所处的特权等级、全局中断使能、监视点使能和地址翻译模式。

表 7-2 当前模式信息寄存器定义

位	名字	读写	描述
1:0	PLV	RW	当前特权等级。其合法的取值范围为 0~3。其中 0 表示最高特权等级，3 表示最低特权等级。 当触发例外时，硬件将该域的值置为 0，以确保陷入后处于最高特权等级。 当执行 ERTN 指令从例外处理程序返回时， 如果 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PPLV 域的值恢复到这里； 否则，如果 CSR.TLBRERA.IsTLBR=1，则硬件将 CSR.TLBRPRMD 的 PPLV 域的值恢复到这里； 否则，硬件将 CSR.PRMD 的 PPLV 域的值恢复到这里。

位	名字	读写	描述
2	IE	RW	当前全局中断使能，高有效。 当触发例外时，硬件将该域的值置为 0，以确保陷入后屏蔽中断。例外处理程序决定重新开启中断响应时，需显式地将该位置 1。 当执行 ERTN 指令从例外处理程序返回时， 如果 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PIE 域的值恢复到这里； 否则，如果 CSR.TLBRERA.IsTLBR=1，则硬件将 CSR.TLBRPRMD 的 PIE 域的值恢复到这里； 否则，硬件将 CSR.PRMD 的 PIE 域的值恢复到这里。
3	DA	RW	直接地址翻译模式的使能，高有效。 当触发 TLB 重填例外或是机器错误例外时，硬件将该域置为 1。 当执行 ERTN 指令从例外处理程序返回时， 如果 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PDA 域的值恢复到这里； 否则，如果 CSR.TLBRERA.IsTLBR=1，则硬件将该域置为 0。 DA 位和 PG 位的合法组合情况为 0、1 或 1、0，当软件配置成其它组合情况时结果不确定。
4	PG	RW	映射地址翻译模式的使能，高有效。 当触发 TLB 重填例外或是机器错误例外时，硬件将该域置为 0。 当执行 ERTN 指令从例外处理程序返回时， 如果 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PPG 域的值恢复到这里； 否则，如果 CSR.TLBRERA.IsTLBR=1，则硬件将该域置为 1。 PG 位和 DA 位的合法组合情况为 0、1 或 1、0，当软件配置成其它组合情况时结果不确定。
6:5	DATF	RW	直接地址翻译模式时，取指操作的存储访问类型。 当触发机器错误例外时，硬件将该域置为 0。 当执行 ERTN 指令从例外处理程序返回，且 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PDATF 域的值恢复到这里。 在采用软件处理 TLB 重填的情况下，当软件将 PG 置为 1 时，需同时将 DATF 域置为 0b01，即一致可缓存类型。
8:7	DATM	RW	直接地址翻译模式时，load 和 store 操作的存储访问类型。 当触发机器错误例外时，硬件将该域置为 0。 当执行 ERTN 指令从例外处理程序返回，且 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PDATM 域的值恢复到这里。 在采用软件处理 TLB 重填的情况下，当软件将 PG 置为 1 时，需同时将 DATM 置为 0b01，即一致可缓存类型。

位	名字	读写	描述
9	WE	RW	指令和数据监视点的使能位，高电平有效。 当触发例外时，硬件将该域的值置为 0。 当执行 ERTN 指令从例外处理程序返回时， 如果 CSR.ERRCTL.IsMERR=1，则硬件将 CSR.ERRCTL 的 PWE 域的值恢复到这里； 否则，如果 CSR.TLBRERA.IsTLBR=1，则硬件将 CSR.TLBRPRMD 的 PWE 域的值恢复到这里； 否则，硬件将 CSR.PRMD 的 PWE 域的值恢复到这里。
31:10	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.2 例外前模式信息（PRMD）

当触发例外时，如果例外类型不是 TLB 重填例外和机器错误例外，硬件会将此时处理器核的特权等级、全局中断使能和监视点使能位保存至例外前模式信息寄存器中，用于例外返回时恢复处理器核的现场。

表 7-3 例外前模式信息寄存器定义

位	名字	读写	描述
1:0	PPLV	RW	当触发例外时，如果例外类型不是 TLB 重填例外和机器错误例外，硬件会将 CSR.CRMD 中 PLV 域的旧值记录在这个域。 当所处理的例外既不是 TLB 重填例外（CSR.TLBRERA.IsTLBR=0）也不是机器错误例外（CSR.ERRCTL.IsMERR=0）时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 PLV 域。
2	PIE	RW	当触发例外时，如果例外类型不是 TLB 重填例外和机器错误例外，硬件会将 CSR.CRMD 中 IE 域的旧值记录在这个域。 当所处理的例外既不是 TLB 重填例外（CSR.TLBRERA.IsTLBR=0）也不是机器错误例外（CSR.ERRCTL.IsMERR=0）时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 IE 域。
3	PWE	RW	当触发例外时，如果例外类型不是 TLB 重填例外和机器错误例外，硬件会将 CSR.CRMD 中 WE 域的旧值记录在这个域。 当所处理的例外既不是 TLB 重填例外（CSR.TLBRERA.IsTLBR=0）也不是机器错误例外（CSR.ERRCTL.IsMERR=0）时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 WE 域。
31:4	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.3 扩展部件使能（EUEN）

除基础整数指令集和特权指令集外，基础浮点数指令集、二进制翻译扩展指令集、128 位向量扩展指令

集和 256 位向量扩展指令集各自均有软件可配置的使能控制位。当这些使能控制无效时，执行对应的指令将触发相应的指令不可用例外。软件利用这套机制可以决定保存上下文时的范围。硬件实现也可以利用此处的控制位实现电路功耗控制。

表 7-4 扩展指令使能寄存器定义

位	名字	读写	描述
0	FPE	RW	基础浮点指令使能控制位。当该位为 0 时，执行 3.2 节所述基础浮点数指令将会触发浮点指令未使能例外（FPD）。
1	SXE	RW	128 位向量扩展指令使能控制位。当该位为 0 时，执行卷二-a 所述的 128 位向量扩展指令将会触发 128 位向量扩展指令未使能例外（SXD）。
2	ASXE	RW	256 位向量扩展指令使能控制位。当该位为 0 时，执行卷二-a 所述的 256 位向量扩展指令将会触发 256 位向量扩展指令未使能例外（ASXD）。
3	BTE	RW	二进制翻译扩展指令使能控制位。当该位为 0 时，执行卷三所述的二进制翻译扩展指令将会触发二进制翻译扩展指令未使能例外（BTD）
31:4	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.4 杂项（MISC）

该寄存器包含了一些在不同特权等级下处理器核运行行为的控制位，包括是否开启 32 位地址模式，是否允许在非特权级下使用部分特权指令，以及地址非对齐检查使能和页表写允许检查的控制。

表 7-5 杂项寄存器定义

位	名字	读写	描述
0	0	R0	保留域。读返回 0，且软件不允许改变其值。
1	VA32L1	RW	PLV1 特权等级下，是否开启 32 位地址模式。0—关闭，1—开启。 该位仅在 LA64 架构下可读写，在 LA32 架构特权等级下，该位读写属性为 R0。
2	VA32L2	RW	PLV2 特权等级下，是否开启 32 位地址模式。0—关闭，1—开启。 该位仅在 LA64 架构下可读写，在 LA32 架构特权等级下，该位读写属性为 R0。
3	VA32L3	RW	PLV3 特权等级下，是否开启 32 位地址模式。0—关闭，1—开启。 该位仅在 LA64 架构下可读写，在 LA32 架构特权等级下，该位读写属性为 R0。
4	0	R0	保留域。读返回 0，且软件不允许改变其值。
5	DRDTL1	RW	PLV1 特权等级下，是否禁用 RDTIME 类指令。当该位为 1 时，PLV1 特权等级下执行 RDTIME 类指令将触发指令特权等级错例外（IPE）。
6	DRDTL2	RW	PLV2 特权等级下，是否禁用 RDTIME 类指令。当该位为 1 时，PLV2 特权等级下执行 RDTIME 类指令将触发指令特权等级错例外（IPE）。
7	DRDTL3	RW	PLV3 特权等级下，是否禁用 RDTIME 类指令。当该位为 1 时，PLV3 特权等级下执行 RDTIME 类指令将触发指令特权等级错例外（IPE）。

位	名字	读写	描述
8	0	R0	保留域。读返回 0，且软件不允许改变其值。
9	RPCNTL1	RW	PLV1 特权等级下，是否允许软件读取性能计数器。当该位为 1 时，PLV1 特权等级下使用 CSRRD 指令访问任一已实现的性能计数器 PCNT 不会触发指令特权等级错例外（IPE）。
10	RPCNTL2	RW	PLV2 特权等级下，是否允许软件读取读性能计数器。当该位为 1 时，PLV2 特权等级下使用 CSRRD 指令访问任一已实现的性能计数器 PCNT 不会触发指令特权等级错例外（IPE）。
11	RPCNTL3	RW	PLV3 特权等级下，是否允许软件读取读性能计数器。当该位为 1 时，PLV3 特权等级下使用 CSRRD 指令访问任一已实现的性能计数器 PCNT 不会触发指令特权等级错例外（IPE）。
12	ALCL0	RW	PLV0 特权等级下，是否对允许非对齐的非向量 load/store 指令[1]进行非对齐检查。为 1 表示进行检查，如果违例则触发地址对齐错例外。 该位仅当硬件实现支持这些非向量 load/store 指令地址非对齐时才是可读写的，否则该位只读恒为 1。
13	ALCL1	RW	PLV1 特权等级下，是否对允许非对齐的非向量 load/store 指令[1]进行非对齐检查。为 1 表示进行检查，如果违例则触发地址对齐错例外。 该位仅当硬件实现支持这些非向量 load/store 指令地址非对齐时才是可读写的，否则该位只读恒为 1。
14	ALCL2	RW	PLV2 特权等级下，是否对允许非对齐的非向量 load/store 指令[1]进行非对齐检查。为 1 表示进行检查，如果违例则触发地址对齐错例外。 该位仅当硬件实现支持这些非向量 load/store 指令地址非对齐时才是可读写的，否则该位只读恒为 1。
15	ALCL3	RW	PLV3 特权等级下，是否对允许非对齐的非向量 load/store 指令[1]进行非对齐检查。为 1 表示进行检查，如果违例则触发地址对齐错例外。 该位仅当硬件实现支持这些非向量 load/store 指令地址非对齐时才是可读写的，否则该位只读恒为 1。
16	DWPL0	RW	PLV0 特权等级下，是否禁止 TLB 虚实地址翻译过程中页表项写允许位的检查。当该位为 1 时，store 指令即使访问一个 D=0 的页表项也不会触发页修改例外。
17	DWPL1	RW	PLV1 特权等级下，是否禁止 TLB 虚实地址翻译过程中页表项写允许位的检查。当该位为 1 时，store 指令即使访问一个 D=0 的页表项也不会触发页修改例外。
18	DWPL2	RW	PLV2 特权等级下，是否禁止 TLB 虚实地址翻译过程中页表项写允许位的检查。当该位为 1 时，store 指令即使访问一个 D=0 的页表项也不会触发页修改例外。
31:19	0	R0	保留域。读返回 0，且软件不允许改变其值。

[1] 受到该控制位影响的指令有：LD[X].{H[U]/W[U]/D}、ST[X].{H/W/D}、LDPTR.{W/D}、STPTR.{W/D}、FLD[X].{S/D}、FST[X].{S/D}、LDPTE、LDDIR、IOCSRRD.{H/W/D}、IOCSRWR.{H/W/D}。

7.4.5 例外配置（ECFG）

该寄存器用于控制例外和中断的入口计算方式以及各中断的局部使能位。

表 7-6 例外配置寄存器定义

位	名字	读写	描述
12:0	LIE	RW	局部中断使能位，高有效。这些局部中断使能位与 CSR.ESTAT 中 IS 域记录的 13 个中断源一一对应，每一位控制一个中断源。
15:13	0	R0	保留域。读返回 0，且软件不允许改变其值。
18:16	VS	RW	配置例外和中断入口的间距。当 VS=0 时，所有例外和中断的入口地址是同一个。当 VS!=0 时，各例外和中断之间的入口地址间距是 2^{VS} 条指令。 因为 TLB 重填例外和机器错误例外其独立的入口基址，所以二者的例外入口不受 VS 域的影响。
31:19	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.6 例外状态（ESTAT）

该寄存器记录例外的状态信息，包括所触发例外的一二级编码，以及各中断的状态。

表 7-7 例外状态寄存器定义

位	名字	读写	描述
1:0	IS[1:0]	RW	两个软件中断的状态位。比特 0 和 1 分别对应 SWI0 和 SWI1。 软件中断的设置也是通过这两位完成，软件写 1 置中断写 0 清中断。
12:2	IS[12:2]	R	中断状态位。其值为 1 表示对应的中断置起。1 个核间中断（IPI），1 个定时器中断（TI），1 个性能计数器溢出中断（PMI），8 个硬中断（HWI0~HWI7） 在线中断模式下，硬件仅是逐拍采样各个中断源并将其状态记录与此。此时对于所有中断须为电平中断的要求，是由中断源负责保证，并不在此处维护。
15:13	0	R0	保留域。读返回 0，且软件不允许改变其值。
21:16	Ecode	R	例外类型一级编码。触发例外时： 如果是 TLB 重填例外或机器错误例外，该域保持不变； 否则，硬件会根据例外类型将表 7-8 中 Ecode 栏定义的数值写入该域。
30:22	EsubCode	R	例外类型二级编码。触发例外时： 如果是 TLB 重填例外或机器错误例外，该域保持不变； 否则，硬件会根据例外类型将表 7-8 中 EsubCode 栏定义的数值写入该域。
31	0	R0	保留域。读返回 0，且软件不允许改变其值。

表 7-8 例外编码表

Ecode	EsubCode	例外代号	例外类型
0x0		INT	仅当 CSR.ECFG.VS=0 时，表示是中断。
0x1		PIL	load 操作页无效例外
0x2		PIS	store 操作页无效例外
0x3		PIF	取指操作页无效例外
0x4		PME	页修改例外
0x5		PNR	页不可读例外
0x6		PNX	页不可执行例外
0x7		PPI	页特权等级不合规例外
0x8	0	ADEF	取指地址错例外
	1	ADEM	访存指令地址错例外
0x9		ALE	地址非对齐例外
0xA		BCE	边界检查错例外
0xB		SYS	系统调用例外
0xC		BRK	断点例外
0xD		INE	指令不存在例外
0xE		IPE	指令特权等级错例外
0xF		FPD	浮点指令未使能例外
0x10		SXD	128 位向量扩展指令未使能例外
0x11		ASXD	256 位向量扩展指令未使能例外
0x12	0	FPE	基础浮点指令例外
	1	VFPE	向量浮点指令例外
0x13	0	WPEF	取指监测点例外
	1	WPEM	load/store 操作监测点例外
0x14		BTD	二进制翻译扩展指令未使能例外
0x15		BTE	二进制翻译相关例外
0x16		GSPR	客户机敏感特权资源例外
0x17		HVC	虚拟机监控调用例外
0x18	0	GCSC	客户机 CSR 软件修改例外
	1	GCHC	客户机 CSR 硬件修改例外
0x1A-0x3E			保留编码

7.4.7 例外程序返回地址（ERA）

该寄存器记录普通例外处理完毕之后的返回地址。当触发例外时，如果例外类型既不是 TLB 重填例外也不是机器错误例外，则触发例外的指令的 PC 将被记录在该寄存器中。

表 7-9 例外程序计数器寄存器定义

位	名字	读写	描述
GRLEN-1:0	PC	RW	触发例外时： 如果是 TLB 重填例外或机器错误例外，该域保持不变； 否则，硬件会将触发例外的指令的 PC 记录到这里。对于 LA64 架构，在这种情况下，如果触发例外的特权等级处于 32 位地址模式，那么记录的 PC 值的高 32 位强制置为 0。

7.4.8 出错虚地址（BADV）

该寄存器用于触发地址错误相关例外时，记录出错的虚地址。此类例外包括：

- 取指地址错例外（ADEF），此时记录的是该指令的 PC。
- load/store 操作地址错例外（ADEM）
- 地址对齐错例外（ALE）
- 边界约束检查错例外（BCE）
- load 操作页无效例外（PIL）
- store 操作页无效例外（PIS）
- 取指操作页无效例外（PIF）
- 页修改例外（PME）
- 页不可读例外（PNR）
- 页不可执行例外（PNX）
- 页特权等级不合规例外（PPI）

表 7-10 出错虚地址寄存器定义

位	名字	读写	描述
GRLEN-1:0	VAddr	RW	当触发地址错误相关例外时，硬件将出错的虚地址记录于此。对于 LA64 架构，在这种情况下，如果触发例外的特权等级处于 32 位地址模式，那么记录的虚地址的高 32 位强制置为 0。

7.4.9 出错指令（BADI）

该寄存器用于记录触发同步类例外的指令的指令码。所谓同步类例外是指除了中断（INT）、客户机 CSR

硬件修改例外（GCHC）、机器错误例外（MERR）之外的所有例外。

表 7-11 出错指令寄存器定义

位	名字	读写	描述
31:0	Inst	R	触发同步类例外时，硬件将触发该例外的指令码记录到这里。

7.4.10 例外入口地址（EENTRY）

该寄存器用于配置普通例外和中断的入口地址。

表 7-12 例外入口页号寄存器定义

位	名字	读写	描述
11:0	0	R	只读恒为 0，写被忽略。
GRLEN-1:12	VPN	RW	普通例外和中断入口地址所在页的页号。

7.4.11 缩减虚地址配置（RVACFG）

该寄存器用于控制虚地址缩减模式下被缩减的地址位宽。

表 7-13 缩减虚地址寄存器定义

位	名字	读写	描述
3:0	RBits	RW	虚地址缩减模式下，被缩减的高位地址的位数。可以配置为 0~8 之间的值。 0 是一个特殊的配置值，意味着不启用虚地址缩减模式。 如果配置的值大于 8，则处理器行为不确定。
31:4	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.12 处理器编号（CPUID）

该寄存器中存有处理器核编号信息。

表 7-14 处理器编号寄存器定义

位	名字	读写	描述
8:0	CoreID	R	处理器核的编号。该信息用于软件在多核系统中区分各个处理器核。系统集成时，每个处理器核的处理器核号信息由硬件根据具体实现情况予以设置。建议系统中处理器核号从 0 开始递增编号。
31:9	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.13 特权资源配置信息 1（PRCFG1）

该寄存器包含一些特权资源的配置信息。

表 7-15 特权资源配置信息 1 寄存器定义

位	名字	读写	描述
3:0	SAVENum	R	SAVE 控制状态寄存器的个数。
11:4	TimerBits	R	定时器（Timer）的有效位数减 1。
14:12	VSMax	R	例外和中断向量入口间距（CSR.ECFG.VS 域）可以设置的最大值。
31:15	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.14 特权资源配置信息 2（PRCFG2）

该寄存器包含一些特权资源的配置信息。

表 7-16 特权资源配置信息 2 寄存器定义

位	名字	读写	描述
GRLEN-1:0	PSAVL	R	指示 TLB 能够支持的页大小（Page Size）。当第 i 位为 1，表明支持 2^i 字节大小的页。

7.4.15 特权资源配置信息 3（PRCFG3）

该寄存器包含一些特权资源的配置信息。

表 7-17 特权资源配置信息 3 寄存器定义

位	名字	读写	描述
3:0	TLBType	R	指示 TLB 组织方式： 0：没有 TLB； 1：一个全相联的多重页大小 TLB（MTLB） 2：一个全相联的多重页大小 TLB（MTLB）+一个组相联的单个页大小 TLB（STLB）； 其它值：保留。
11:4	MTLBEntries	R	当 TLBType=0 时，该域只读恒为 0； 当 TLBType=1 或 2 时，该域的值是全相联多重页大小 TLB 的项数减 1。
19:12	STLBWays	R	当 TLBType=0 或 1 时，该域只读恒为 0； 当 TLBType=2 时，该域的值是组相联单个页大小 TLB 的路数减 1。

位	名字	读写	描述
25:20	STLBSets	R	当 TLBType=0 或 1 时，该域只读恒为 0； 当 TLBType=2 时，该域的值是组相联单个页大小 TLB 的每一路项数的幂指数，即每一路有 $2^{STLBSets}$ 项。
31:26	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.16 数据保存（SAVE）

数据保存控制状态寄存器用于给系统软件暂存数据。每个数据保存寄存器可以存放一个通用寄存器的数据。

数据保存寄存器最少实现 1 个，最多实现 16 个。具体实现的个数软件可以从 CSR.PRCFG1.SAVENum 中获知。从 SAVE0 开始，各个 SAVE 寄存器的地址依次为 0x30、0x31、……、0x30+SAVENum-1。

所有数据保存控制状态寄存器的格式均相同，如表 7-18 所示。

表 7-18 数据保存寄存器定义

位	名字	读写	描述
GRLEN-1:0	Data	RW	仅供软件读写的数据。除执行 CSR 指令外，硬件不会修改该域的内容。

7.4.17 LLBit 控制（LLBCTL）

该寄存器用于对 LLBit 进行的访问控制操作。

表 7-19 LLBit 寄存器定义

位	名字	读写	描述
0	ROLLB	R	只读位，返回当前 LLBit 的值。
1	WCLLB	W1	软件对该位写 1 将 LLBit 清 0。软件对该位写 0 将被硬件忽略。
2	KLO	RW	用于控制 ERTN 指令执行时对 LLBit 的操作。 当该位等于 1 的时候，执行 ERTN 指令的时候不将 LLBit 清 0，但是该位会被硬件自动清 0。意味着，每次 KLO 置 1 后只能影响一次 ERTN 指令的执行。
31:3	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.4.18 实现相关控制 1（IMPCTL1）

该寄存器包含与具体实现时微结构特性相关的控制信息。其格式及各个域的具体含义由具体实现定义。

7.4.19 实现相关控制 2（IMPCTL2）

该寄存器包含与具体实现时微结构特性相关的控制信息。其格式及各个域的具体含义由具体实现定义。

7.4.20 高速缓存标签（CTAG）

该寄存器用于 CACOP 指令直接访问 Cache 时，存放从 Cache Tag 中读出的内容或是将要写入 Cache Tag 的内容。其格式及各个域的具体含义由具体实现定义。

7.5 映射地址翻译相关控制状态寄存器

7.5.1 TLB 索引（TLBIDX）

该寄存器包含 TLB 指令操作 TLB 时相关的索引值等信息。表 7-20 中 Index 域的位宽与实现相关，不过本架构所允许的 Index 位宽不超过 16 比特。

该寄存器还包含 TLB 指令操作时与 TLB 表项中 PS、E 域相关的信息。

表 7-20 TLB 索引寄存器定义

位	名字	读写	描述
n-1:0	Index	RW	执行 TLBRD 和 TLBWR 指令时，访问 TLB 表项的索引值来自于此。 执行 TLBSRCH 指令时，如果命中，则命中项的索引值记录到这里。 有关索引值与 TLB 表项间的对应关系，请参看 4.2.4.1 节中的相关内容。
15:n	0	R	只读恒为 0，写被忽略。
23:16	0	R0	保留域。读返回 0，且软件不允许改变其值。
29:24	PS	RW	执行 TLBRD 指令时，所读取 TLB 表项的 PS 域的值记录到这里。 在 CSR.TLBRERA.IsTLBR=0 时，执行 TLBWR 和 TLBFILL 指令，写入的 TLB 表项的 PS 域的值来自于此。
30	0	R0	保留域。读返回 0，且软件不允许改变其值。
31	NE	RW	该位为 1 表示该 TLB 表项为空（无效 TLB 表项），为 0 表示该 TLB 表项非空（有效 TLB 表项）。 执行 TLBSRCH 时，如果有命中项该位记为 0，否则该位记为 1。 执行 TLBRD 时，所读取 TLB 表项的 E 位信息取反后记录到这里。 执行 TLBWR 或 TLBFILL 指令时，若 CSR.TLBRERA.IsTLBR=0，将该位的值取反后写入到被写 TLB 项的 E 位；若此时 CSR.TLBRERA.IsTLBR=1，那么被写入的 TLB 项的 E 位总是置为 1，与该位的值无关。

7.5.2 TLB 表项高位（TLBEHI）

该寄存器包含 TLB 指令操作时与 TLB 表项高位部分虚页号相关的信息。因 TLB 表项高位所含的 VPPN 域的位宽与实现所支持的有效虚地址范围相关，故有关寄存器域的定义分开表述。

表 7-21 TLB 页表高位寄存器定义（LA64 架构）

位	名字	读写	描述
12:0	0	R	只读恒为 0，写被忽略。
VALEN-1:13	VPPN	RW	执行 TLBRD 指令时，所读取 TLB 表项的 VPPN 域的值记录到这里。 在 CSR.TLBRERA.IsTLBR=0 时，执行 TLBSRCH 指令时查询 TLB 所用 VPPN 值，以及执行 TLBWR 和 TLBFILL 指令时写入 TLB 表项的 VPPN 域的值来自于此。 当触发 load 操作页无效例外、store 操作页无效例外、取指操作页无效例外、页修改例外、页不可读例外、页不可执行例外和页特权等级不合规例外时，触发例外的虚地址的[VALEN-1:13]位被记录到这里。
63:VALEN	Sign_Ext	R	读返回值是 VPPN 域最高位的符号扩展，写这些位被忽略。

表 7-22 TLB 页表高位寄存器定义（LA32 架构）

位	名字	读写	描述
12:0	0	R	只读恒为 0，写被忽略。
31:13	VPPN	RW	执行 TLBRD 指令时，所读取 TLB 表项的 VPPN 域的值记录到这里。 在 CSR.TLBRERA.IsTLBR=0 时，执行 TLBSRCH 指令时查询 TLB 所用 VPPN 值，以及执行 TLBWR 和 TLBFILL 指令时写入 TLB 表项的 VPPN 域的值来自于此。 当触发 load 操作页无效例外、store 操作页无效例外、取指操作页无效例外、页修改例外、页不可读例外、页不可执行例外和页特权等级不合规例外时，触发例外的虚地址的[31:13]位被记录到这里。

7.5.3 TLB 表项低位（TLBELO0, TLBELO1）

TLBELO0 和 TLBELO1 两个寄存器包含了 TLB 指令操作时 TLB 表项低位部分物理页号等相关的信息。因龙芯架构下 TLB 采用双页结构，所以 TLB 表项的低位信息对应奇偶两个物理页表项，其中偶数页信息在 TLBELO0 中，奇数页信息在 TLBELO1 中。TLBELO0 和 TLBELO1 寄存器的格式定义完全相同，其各个域的定义在表 7-23 和表 7-24 中。

当 CSR.TLBRERA.IsTLBR=0 时，执行 TLBWR 和 TLBFILL 指令，写入 TLB 表项的 G、PPN0、V0、PLV0、MAT0、D0、NR0、NX0、RPLV0、PPN1、V1、PLV1、MAT1、D1、NR1、NX1、RPLV1 域的值分别来自于 TLBELO0 和 TLBELO1。

执行 TLBRD 指令时，从 TLB 表项中读出的上述信息逐个写入到 TLBELO0 和 TLBELO1 两寄存器中

的对应域中。

表 7-23 TLB 表项低位寄存器定义（LA64 架构）

位	名字	读写	描述
0	V	RW	页表项的有效位（V）。
1	D	RW	页表项的脏位（D）。
3:2	PLV	RW	页表项的特权等级（PLV）。
5:4	MAT	RW	页表项的存储访问类型（MAT）。
6	G	RW	页表项的全局标志位（G）。 执行 TLBFILL 和 TLBWR 指令时，仅当 TLBELO0 和 TLBELO1 中的 G 位均为 1 时，填入到 TLB 中的页表项的 G 位才为 1。 执行 TLBRD 指令时，当所读取的 TLB 表项的 G 位为 1，则 TLBELO0 和 TLBELO1 中的 G 位被同时置为 1。
11:7	0	R	只读恒为 0，写被忽略。
PALEN-1:12	PPN	RW	页表的物理页号（PPN）。
60:PALEN	0	R	只读恒为 0，写被忽略。
61	NR	RW	页表项的不可读位（NR）。
62	NX	RW	页表项的不可执行位（NX）。
63	RPLV	RW	页表的受限特权等级使能（RPLV）。当 RPLV=0 时，该页表项可以被任何特权等级不低于 PLV 的程序访问；当 RPLV=1 时，该页表项仅可以被特权等级等于 PLV 的程序访问。

表 7-24 TLB 表项低位寄存器定义（LA32 架构）

位	名字	读写	描述
0	V	RW	页表项的有效位（V）。
1	D	RW	页表项的脏位（D）。
3:2	PLV	RW	页表项的特权等级（PLV）。
5:4	MAT	RW	页表项的存储访问类型（MAT）。
6	G	RW	页表项的全局标志位（G）。 执行 TLBFILL 和 TLBWR 指令时，仅当 TLBELO0 和 TLBELO1 中的 G 位均为 1 时，填入到 TLB 中的页表项的 G 位才为 1。 执行 TLBRD 指令时，当所读取的 TLB 表项的 G 位为 1，则 TLBELO0 和 TLBELO1 中的 G 位被同时置为 1。
7	0	R	只读恒为 0，写被忽略。
PALEN-5:8	PPN	RW	页表的物理页号（PPN）。

位	名字	读写	描述
31:PALEN-4	0	R	只读恒为 0，写被忽略。当 PALEN=36 时，这个域不存在。

7.5.4 地址空间标识符（ASID）

该寄存器中包含了用于访存操作和 TLB 指令的地址空间标识符（ASID）信息。ASID 的位宽随着架构规范的演进可能进一步增加，为方便软件明确 ASID 的位宽，将直接给出这一信息。

表 7-25 地址空间标识符寄存器定义

位	名字	读写	描述
9:0	ASID	RW	当前执行的程序所对应的地址空间标识符。 在取指、执行 load/store 指令时，作为查询 TLB 的 ASID 键值信息。 执行 TLBSRCH 和 TLBCLR 指令时，作为查询 TLB 的 ASID 键值信息。 执行 TLBWR 或 TLBFILL 指令时，写入 TLB 表项 ASID 域的值来自于此。 执行 TLBRD 指令时，所读取的 TLB 表项的 ASID 域的内容记录到这里。
15:10	0	R	只读恒为 0，写被忽略。
23:16	ASIDBITS	R	ASID 域的位宽。其直接等于这个域的数值。
31:24	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.5.5 低半地址空间全局目录基址（PGDL）

该寄存器用于配置低半地址空间的全局目录的基址。要求全局目录的基址一定是 4KB 边界地址对齐的，所以该寄存器的最低 12 位软件不可配置，只读恒为 0。

表 7-26 低半地址空间全局目录基址寄存器定义

位	名字	读写	描述
11:0	0	R	只读恒为 0，写被忽略。
GRLEN-1:12	Base	RW	低半地址空间的全局目录的基址。 所谓低半地址空间是指虚地址的第[VALEN-1]位等于 0。

7.5.6 高半地址空间全局目录基址（PGDH）

该寄存器用于配置高半地址空间的全局目录的基址。要求全局目录的基址一定是 4KB 边界地址对齐的，所以该寄存器的最低 12 位软件不可配置，只读恒为 0。

表 7-27 高半地址空间全局目录基址寄存器定义

位	名字	读写	描述
11:0	0	R	只读恒为 0，写被忽略。
GRLEN-1:12	Base	RW	高半地址空间的全局目录的基址。 所谓高半地址空间是指虚地址的第[VALEN-1]位等于 1。

7.5.7 全局目录基址（PGD）

该寄存器是一个只读寄存器，其内容是当前上下文中出错虚地址所对应的全局目录基址信息。该寄存器的只读信息，不仅用于 CSR 类指令的读返回值，也用于 LDDIR 指令访问全局目录时所需的基址信息。

表 7-28 全局目录基址寄存器定义

位	名字	读写	描述
11:0	0	R	只读恒为 0，写被忽略。
GRLEN-1:12	Base	R	如果当前上下文中出错虚地址的最高位是 0，读返回值等于 CSR.PGDL 的 Base 域；否则，读返回值等于 CSR.PGDH 的 Base 域。 当 CSR.TLBRERA.IsTLBR=0 时，当前上下文中出错虚地址信息位于 CSR.BADV 中；否则，出错虚地址信息位于 CSR.TLBRBADV 中。

7.5.8 页表遍历控制低半部分（PWCL）

该寄存器和 CSR.PWCH 寄存器中的信息在一起定义了操作系统中所采用的页表结构。这些信息将用于指示软件或硬件进行页表遍历。页表结构及遍历过程示意请参看 5.4.5 节内容。

在 LA32 架构下仅实现 CSR.PWCL。为此 PWCL 寄存器需包含刻画页表结构的所有信息，由此导致末级页表和最低两级目录的起始地址位置均不超过 32 位，该限制在 LA64 架构下依然存在。

表 7-29 页表遍历控制低半部分寄存器定义

位	名字	读写	描述
4:0	PTbase	RW	末级页表的起始地址。
9:5	PTwidth	RW	末级页表的索引位数。
14:10	Dir1_base	RW	最低一级目录的起始地址。
19:15	Dir1_width	RW	最低一级目录的索引位数。0 表示没有这一级。
24:20	Dir2_base	RW	次低一级目录的起始地址。
29:25	Dir2_width	RW	次低一级目录的索引位数。0 表示没有这一级。
31:30	PTEWidth	RW	内存中每个页表项的位宽。 0 表示 64 比特，1 表示 128 比特，2 表示 192 比特，3 表示 256 比特。

7.5.9 页表遍历控制高半部分（PWCH）

该寄存器和 CSR.PWCL 寄存器中的信息在一起定义了操作系统中所采用的页表结构。这些信息将用于指示软件或硬件进行页表遍历。页表结构及遍历过程示意请参看 5.4.5 节内容。

该寄存器仅定义在 LA64 架构下。

表 7-30 页表遍历控制高半部分寄存器定义

位	名字	读写	描述
5:0	Dir3_base	RW	次高一级目录的起始地址。
11:6	Dir3_width	RW	次高一级目录的索引位数。0 表示没有这一级。
17:12	Dir4_base	RW	最高一级目录的起始地址。
23:18	Dir4_width	RW	最高一级目录的索引位数。0 表示没有这一级。
31:24	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.5.10 STLB 页大小（STLBPS）

该寄存器用于配置 STLB 中页的大小。

表 7-31 STLB 页大小寄存器定义

位	名字	读写	描述
5:0	PS	RW	STLB 的页大小的 2 的幂指数。例如，若页大小为 16KB，则 PS=0xE。
31:6	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.5.11 TLB 重填例外入口地址（TLBRENTRY）

该寄存器用于配置 TLB 重填例外的入口地址。由于触发 TLB 重填例外之后，处理器核将进入直接地址翻译模式，所以此处所填入口地址应当是物理地址。

表 7-32 TLB 重填例外入口地址寄存器定义（LA64 架构）

位	名字	读写	描述
11:0	0	R	TLB 重填例外入口地址[11:0]位。只读恒为 0，写被忽略。
PALEN-1:12	PPN	RW	TLB 重填例外入口地址[PALEN-1:12]位。此处填入的地址应为物理地址。
63:PALEN	0	R	只读恒为 0，写被忽略。

表 7-33 TLB 重填例外入口地址寄存器定义（LA32 架构）

位	名字	读写	描述
11:0	0	R	TLB 重填例外入口地址[11:0]位。只读恒为 0，写被忽略。
31:12	PPN	RW	TLB 重填例外入口地址[31:12]位。此处填入的地址应为物理地址。

7.5.12 TLB 重填例外出错虚地址（TLBRBADV）

该寄存器用于记录触发 TLB 重填例外的出错虚地址。

表 7-34 TLB 寄存器定义

位	名字	读写	描述
GRLEN-1:0	VAddr	RW	当触发 TLB 重填例外时，硬件将出错的虚地址记录与此。对于 LA64 架构，在这种情况下，如果触发例外的特权等级处于 32 位地址模式，那么记录的虚地址的高 32 位强制置为 0。

7.5.13 TLB 重填例外返回地址（TLBRERA）

该寄存器保存 TLB 重填例外处理完毕之后的返回地址。除此之外，该寄存器还包含用于标识当前例外是 TLB 重填例外的标志位。

表 7-35 CSR 0x8A 寄存器定义

位	名字	读写	描述
0	IsTLBR	RW	该位为 1 表示当前处于 TLB 重填例外处理的上下文中。 当触发 TLB 重填例外时，硬件将该位置 1。 当该位为 1 时，仅在 CSR.ERRCTL.IsMERR=0 的情况下，执行 ERTN 指令会将其清 0，否则保持不变。 因为架构中为 TLB 重填例外定义了一套独立的 CSR，所以当该位为 1 时， • ERTN 返回时，用于恢复 CSR.CRMD 的信息将来自于 CSR.TLBRPRMD； • ERTN 返回地址信息将来自于 CSR.TLBRERA； • TLBWR 和 TLBFILL 指令待写入的表项信息将来自于 CSR.TLBREHI、CSR.TLBELO0、CSR.TLBELO1； • TLBSRCH 指令查询的信息来自于 CSR.TLBREHI； • LDDIR、LDPTE 指令执行所需的出错虚地址信息将来自于 CSR.TLBRBADV。
1	0	R	只读恒为 0，写被忽略。
GRLEN-1:2	PC	RW	记录触发 TLB 重填例外的指令的 PC 的[GRLEN-1:2]位。当执行 ERTN 指令从 TLB 重填例外处理程序返回时（此时本寄存器 IsTLBR=1 且 CSR.ERRCTL.IsMERR=0），硬件自动将存放在此处的值最低位补上两比特 0 后作为最终的返回地址。

7.5.14 TLB 重填例外数据保存（TLBRSAVE）

该寄存器用于给系统软件暂存数据。每个数据保存寄存器可以存放一个通用寄存器的数据。

之所以额外设置一个供 TLB 重填例外处理程序使用的 SAVE 寄存器，是针对非 TLB 重填例外的处理过程中触发 TLB 重填例外这一情况。

表 7-36 TLB 重填例外数据保存寄存器定义

位	名字	读写	描述
GRLEN-1:0	Data	RW	仅供软件读写的数据。除执行 CSR 指令外，硬件不会修改该域的内容。

7.5.15 TLB 重填例外表项低位（TLBRELO0, TLBRELO1）

TLBRELO0/1 两寄存器是处于 TLB 重填例外上下文时（此时 CSR.TLBRERA.IsTLBR=1），存放 TLB 指令操作时 TLB 表项低位部分物理页号等相关的信息。TLBRELO0/1 两寄存器的格式及各个域的含义分别与 TLBELO0/1 两寄存器一样。

不过，TLBRELO0/1 两寄存器并不是 CSR.TLBRERA.IsTLBR=1 情况下 TLBELO0/1 两寄存器的完全翻版。这体现为两点：

- 无论 CSR.TLBRERA.IsTLBR 等于何值，执行 TLBRD 指令都只更新 TLBELO0/1 两寄存器。
- 无论 CSR.TLBRERA.IsTLBR 等于何值，执行 LDPTE 指令都只更新 TLBRELO0/1 两寄存器。

表 7-37 TLB 重填例外表项低位寄存器定义（LA64 架构）

位	名字	读写	描述
0	V	RW	页表项的有效位（V）。
1	D	RW	页表项的脏位（D）。
3:2	PLV	RW	页表项的特权等级（PLV）。
5:4	MAT	RW	页表项的存储访问类型（MAT）。
6	G	RW	页表项的全局标志位（G）。 执行 TLBFILL 和 TLBWR 指令时，仅当 TLBELO0 和 TLBELO1 中的 G 位均为 1 时，填入到 TLB 中的页表项的 G 位才为 1。
11:7	0	R	只读恒为 0，写被忽略。
PALEN-1:12	PPN	RW	页表的物理页号（PPN）。
60:PALEN	0	R	只读恒为 0，写被忽略。
61	NR	RW	页表项的不可读位（NR）。
62	NX	RW	页表项的不可执行位（NX）。

位	名字	读写	描述
63	RPLV	RW	页表的受限特权等级使能（RPLV）。当 RPLV=0 时，该页表项可以被任何特权等级不低于 PLV 的程序访问；当 RPLV=1 时，该页表项仅可以被特权等级等于 PLV 的程序访问。

表 7-38 TLB 重填例外表项低位寄存器定义（LA32 架构）

位	名字	读写	描述
0	V	RW	页表项的有效位（V）。
1	D	RW	页表项的脏位（D）。
3:2	PLV	RW	页表项的特权等级（PLV）。
5:4	MAT	RW	页表项的存储访问类型（MAT）。
6	G	RW	页表项的全局标志位（G）。 执行 TLBFILL 和 TLBWR 指令时，仅当 TLBELO0 和 TLBELO1 中的 G 位均为 1 时，填入到 TLB 中的页表项的 G 位才为 1。
7	0	R	只读恒为 0，写被忽略。
PALEN-5:8	PPN	RW	页表的物理页号（PPN）。
31:PALEN-4	0	R	只读恒为 0，写被忽略。当 PALEN=36 时，这个域不存在。

7.5.16 TLB 重填例外表项高位（TLBREHI）

TLBREHI 寄存器是处于 TLB 重填例外上下文时（此时 CSR.TLBRERA.IsTLBR=1），存放 TLB 指令操作时 TLB 表项低位部分物理页号等相关的信息。TLBREHI 寄存器的格式及各个域的含义分别与 TLBEHI 寄存器一样。

不过，TLBREHI 寄存器并不是 CSR.TLBRERA.IsTLBR=1 情况下 TLBEHI 寄存器的完全翻版。这体现在：

- 无论 CSR.TLBRERA.IsTLBR 等于何值，执行 TLBRD 指令都只更新 TLBEHI 寄存器。

表 7-39 TLB 重填例外页表高位寄存器定义（LA64 架构）

位	名字	读写	描述
5:0	PS	RW	TLB 重填例外专用的页大小值。即在 CSR.TLBRERA.IsTLBR=1 时，执行 TLBWR 和 TLBFILL 指令，写入的 TLB 表项的 PS 域的值来自于此。
12:6	0	R	只读恒为 0，写被忽略。

位	名字	读写	描述
VALEN-1:13	VPPN	RW	在 CSR.TLBRERA.IsTLBR=1 时，执行 TLBSRCH 指令时查询 TLB 所用 VPPN 值，以及执行 TLBWR 和 TLBFILL 指令时写入 TLB 表项的 VPPN 域的值来自于此。 当触发 TLB 重填例外时，触发例外的虚地址的[VALEN-1:13]位被记录到这里。
63:VALEN	Sign_Ext	R	读返回值是 VPPN 域最高位的符号扩展，写这些位被忽略。

表 7-40 TLB 重填例外页表高位寄存器定义（LA32 架构）

位	名字	读写	描述
5:0	PS	RW	TLB 重填例外专用的页大小值。即在 CSR.TLBRERA.IsTLBR=1 时，执行 TLBWR 和 TLBFILL 指令，写入的 TLB 表项的 PS 域的值来自于此。
12:6	0	R	只读恒为 0，写被忽略。
31:13	VPPN	RW	在 CSR.TLBRERA.IsTLBR=1 时，执行 TLBSRCH 指令时查询 TLB 所用 VPPN 值，以及执行 TLBWR 和 TLBFILL 指令时写入 TLB 表项的 VPPN 域的值来自于此。 当触发 TLB 重填例外时，触发例外的虚地址的[31:13]位被记录到这里。

7.5.17 TLB 重填例外前模式信息（TLBRPRMD）

当触发 TLB 重填例外时，硬件会将此时处理器核的特权等级、客户机模式、全局中断使能和监视点使能位保存至该寄存器中，用于例外返回时恢复处理器核的现场。

表 7-41 TLB 重填例外前模式信息寄存器定义

位	名字	读写	描述
1:0	PPLV	RW	当触发 TLB 重填例外时，硬件会将 CSR.CRMD 中 PLV 域的旧值记录在这个域。 当 CSR.TLBRERA.IsTLBR=1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 PLV 域。
2	PIE	RW	当触发 TLB 重填例外时，硬件会将 CSR.CRMD 中 IE 域的旧值记录在这个域。 当 CSR.TLBRERA.IsTLBR=1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 IE 域。
3	0	R	若未实现虚拟化扩展，则该位只读恒为 0，写被忽略。
4	PWE	RW	当触发 TLB 重填例外时，硬件会将 CSR.CRMD 中 WE 域的旧值记录在这个域。 当 CSR.TLBRERA.IsTLBR=1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 WE 域。
31:5	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.5.18 直接映射配置窗口（DMW0~DMW3）

这一组寄存器参与完成直接映射地址翻译模式。有关该地址翻译模式的内容请参看 5.2.1 节。

表 7-42 直接映射配置窗口寄存器定义（LA64 架构）

位	名字	读写	描述
0	PLV0	RW	为 1 表示在特权等级 PLV0 下可以使用该窗口的配置进行直接映射地址翻译。
1	PLV1	RW	为 1 表示在特权等级 PLV1 下可以使用该窗口的配置进行直接映射地址翻译。
2	PLV2	RW	为 1 表示在特权等级 PLV2 下可以使用该窗口的配置进行直接映射地址翻译。
3	PLV3	RW	为 1 表示在特权等级 PLV3 下可以使用该窗口的配置进行直接映射地址翻译。
5:4	MAT	RW	虚地址落在该映射窗口下访存操作的存储访问类型。
59:6	0	R0	保留域。读返回 0，且软件不允许改变其值。
63:60	VSEG	RW	直接映射窗口的虚地址的[63:60]位。

表 7-43 直接映射配置窗口寄存器定义（LA32 架构）

位	名字	读写	描述
0	PLV0	RW	为 1 表示在特权等级 PLV0 下可以使用该窗口的配置进行直接映射地址翻译。
1	PLV1	RW	为 1 表示在特权等级 PLV1 下可以使用该窗口的配置进行直接映射地址翻译。
2	PLV2	RW	为 1 表示在特权等级 PLV2 下可以使用该窗口的配置进行直接映射地址翻译。
3	PLV3	RW	为 1 表示在特权等级 PLV3 下可以使用该窗口的配置进行直接映射地址翻译。
5:4	MAT	RW	虚地址落在该映射窗口下访存操作的存储访问类型。
24:6	0	R0	保留域。读返回 0，且软件不允许改变其值。
27:25	PSEG	RW	直接映射窗口的物理地址的[31:29]位。
28	0	R0	保留域。读返回 0，且软件不允许改变其值。
31:29	VSEG	RW	直接映射窗口的虚地址的[31:29]位。

7.6 定时器相关控制状态寄存器

7.6.1 定时器编号（TID）

处理器中每个定时器都有一个唯一可识别的编号，由软件配置在该寄存器中。每个定时器也同时唯一对应着一个计时器，当软件使用 RDTIME 指令读取计时器数值时，一并返回的计时器 ID 号也就是与之对应的定时器编号。

表 7-44 CSR 0x40 寄存器定义

位	名字	读写	描述
31:0	TID	RW	定时器编号。软件可配置。处理器核复位期间，硬件可以将其复位成与 CSR.CPUID 中 CoreID 相同的值。

7.6.2 定时器配置（TCFG）

该寄存器是软件配置定时器的接口。定时器的有效位数由实现决定，因此该寄存器中 TimeVal 域的位宽也将随之变化。

表 7-45 定时器配置寄存器定义

位	名字	读写	描述
0	En	RW	定时器使能位。仅当该位为 1 时，定时器才会进行倒计时自减，并在减为 0 值时置起定时中断信号。
1	Periodic	RW	定时器循环模式控制位。若该位为 1，定时器在倒计时自减至 0 时，在置起定时中断信号的同时，还会自动定时器重新装载成 InitVal 域中配置的初始值，然后再下一个时钟周期继续自减。若该位为 0，定时器在倒计时自减至 0 时，将停止计数直至软件再次配置该定时器。
n-1:2	InitVal	RW	定时器倒计时自减计数的初始值。要求该初始值必须是 4 的整倍数。硬件将自动在该域数值的最低位补上两比特 0 后再使用。
GRLEN-1:n	0	R	只读恒为 0，写被忽略。

7.6.3 定时器数值（TVAL）

软件可通过读取该寄存器来获知定时器当前的计数值。定时器的有效位数由实现决定，因此该寄存器中 TimeVal 域的位宽也将随之变化。

表 7-46 定时器剩余寄存器定义

位	名字	读写	描述
n-1:0	TimeVal	R	当前定时器的计数值。
GRLEN-1:n	0	R	只读恒为 0，写被忽略。

7.6.4 计时器补偿（CNTC）

软件可以通过配置该寄存器来对计时器的读出值进行修正，最终的读出值为：计时器原始计数值+计时器补偿值。需知配置该寄存器不可直接改变计时器的计数值。

在 LA32 架构下，该寄存器为 32 位，其值将符号扩展至 64 位后于计数器原始计数值相加。

表 7-47 计时器补偿寄存器定义

位	名字	读写	描述
GRLEN-1:0	Compensation	RW	软件可配置的计时器补偿值。

7.6.5 定时中断清除（TICLR）

软件通过对该寄存器位 0 写 1 来清除定时器置起的定时中断信号。

表 7-48 定时中断清除寄存器定义

位	名字	读写	描述
0	CLR	W1	当对该 bit 写值 1 时，将清除时钟中断标记。该寄存器读出结果总为 0。
31:1	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.7 RAS 相关控制状态寄存器

7.7.1 机器错误控制（MERRCTL）

因机器错误例外的发生时机软件无法预判和控制，所以为了在触发机器错误例外时不破坏任何其它现场，特别为机器错误例外定义了一组独立的 CSR，供系统软件保存恢复其它现场之用。这一组独立的 CSR 除了 MERRERA 和 ERRSAVE 外，其余集中在 MERRCTL 寄存器中。

表 7-49 机器错误控制寄存器定义

位	名字	读写	描述
0	IsMERR	R	为 1 表示当前处于机器错误例外处理的上下文中。 当触发机器错误例外时，硬件将该位置 1。 当该位为 1 时，执行 ERTN 指令会将其清 0。 因为架构中为机器错误例外定义了一套独立的 CSR，所以当该位为 1 时， • ERTN 返回时，用于恢复 CSR.CRMD 的信息将来自于本寄存器中的 PPLV、PIE 等域； • ERTN 返回地址信息将来自于 CSR.ERRERA。
1	Repairable	R	为 1 表示出现的机器错误硬件可以自动修复，因此例外处理程序可以不做任何处理直接返回。
3:2	PPLV	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 PLV 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 PLV 域。

位	名字	读写	描述
4	PIE	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 IE 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 IE 域。
5	0	R	若未实现虚拟化扩展，则该位只读恒为 0，写被忽略。
6	PWE	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 WE 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 WE 域。
7	PDA	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 DA 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 DA 域。
8	PPG	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 PG 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 PG 域。
10:9	PDATF	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 DATF 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 DATF 域。
12:11	PDATM	RW	当触发机器错误例外时，硬件会将 CSR.CRMD 中 DATM 域的旧值记录在这个域。 当本寄存器的 IsMERR 为 1 时，执行 ERTN 指令从例外处理程序返回时，硬件会将这个域的值恢复到 CSR.CRMD 的 DATM 域。
15:13	0	R0	Reserved，读返回 0，必须写 0，或者通过 csr mask write 屏蔽掉。
23:16	Cause	R	机器错误类型编码。 目前只定义了 0x1 值表示 Cache 校验错误。其余编码值保留。
31:24	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.7.2 机器错误信息 1/2（MERRINFO1/2）

在触发机器错误例外时，硬件会将更多与该错误相关的信息存入到这两个寄存器中，供系统软件诊断之用。其格式及各个域的具体含义由具体实现定义。

7.7.3 机器错误例外入口地址（MERRENTRY）

该寄存器用于配置机器错误例外的入口地址。由于触发机器错误例外之后，处理器核将进入直接地址翻译模式，所以此处所填入口地址应当是物理地址。

表 7-50 机器错误例外入口基址寄存器定义（LA64 架构）

位	名字	读写	描述
11:0	0	R	机器错误例外入口地址[11:0]位。只读恒为 0，写被忽略。
PALEN-1:12	PPN	RW	机器错误例外入口地址[PALEN-1:12]位。此处填入的地址应为物理地址。
63:PALEN	0	R	只读恒为 0，写被忽略。

表 7-51 机器错误例外入口基址寄存器定义（LA32 架构）

位	名字	读写	描述
11:0	0	R	机器错误例外入口地址[11:0]位。只读恒为 0，写被忽略。
31:12	PPN	RW	机器错误例外入口地址[31:12]位。此处填入的地址应为物理地址。

7.7.4 机器错误例外返回地址（MERRERA）

该寄存器用于记录机器错误例外处理完毕之后的返回地址。

表 7-52 机器错误例外程序计数器寄存器定义

位	名字	读写	描述
GRLEN-1:0	PC	RW	记录触发机器错误例外的指令的 PC。当执行 ERTN 指令从机器错误例外处理程序返回时（此时 CSR.ERRCTL.IsMERR=1），将存放在此的值作为返回地址。

7.7.5 机器错误例外数据保存（MERRSAVE）

该寄存器用于给系统软件暂存数据。每个数据保存寄存器可以存放一个通用寄存器的数据。

之所以额外设置一个供机器错误例外处理程序使用的 SAVE 寄存器，是因为机器错误例外的发生时机软件无法预判和控制，它有可能发生在任何其它例外的处理过程中。

表 7-53 机器错误例外数据保存寄存器定义

位	名字	读写	描述
GRLEN-1:0	Data	RW	仅供软件读写的数据。除执行 CSR 指令外，硬件不会修改该域的内容。

7.8 性能监测相关控制状态寄存器

龙芯架构定义了硬件性能监测机制，支持软件进行性能分析。该机制的主体是一系列性能监测器。性能监测器至少实现 1 个，至多实现 32 个，具体数目由实现决定。软件可以通过读取 CPUCFG.6.PMNUM[bit7:4]来确定有多少性能监测器可以使用。

每个性能监测器包含两个 CSR：一个性能监测配置寄存器（PMCFG）和一个性能监测计数器（PMCNT）。

所有性能监测相关的 CSR 从 0x200 地址开始交替编址，第 n 个性能监测配置寄存器的地址是 0x200+n，第 n 个性能监测计数器的地址是 0x201+n。所有性能监测配置寄存器的格式一样，在 7.8.1 节中介绍；所有性能监测计数器的格式一样，在 7.8.2 节中介绍。

7.8.1 性能监测配置（PMCFG）

表 7-54 性能监测配置寄存器定义

位	名字	读写	描述
9:0	EvCode	RW	被监测的性能事件的事件号。事件号的定义分为两部分，一部分是架构规范中明确其含义的，所有兼容本架构的处理器都必须实现；余下的事件号的含义与具体实现相关的，由处理器的实现者自行定义。
15:10	0	R0	保留域。读返回 0，且软件不允许改变其值。
16	PLV0	RW	PLV0 特权等级下该性能监测器的计数使能。1——进行计数，0——停止计数。
17	PLV1	RW	PLV1 特权等级下该性能监测器的计数使能。1——进行计数，0——停止计数。
18	PLV2	RW	PLV2 特权等级下该性能监测器的计数使能。1——进行计数，0——停止计数。
19	PLV3	RW	PLV3 特权等级下该性能监测器的计数使能。1——进行计数，0——停止计数。
20	PMIEn	RW	该性能监测器的性能监测计数溢出中断使能位。1——使能，0——禁止。
22:21	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
31:23	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.8.2 性能监测计数器（PMCNT）

表 7-55 CSR 0x201 寄存器定义

位	名字	读写	描述
GRLEN-1:0	Count	RW	该性能监测器的所监测性能事件每发生一次，则该计数器自增 1。 如果该性能监测器使能了性能监测计数溢出中断，那么当 Count 的最高位为 1 时，将置起该中断。这也意味着，软件可以通过将 Count 的最高位清 0 来撤销该中断。

7.9 监视点相关控制状态寄存器

龙芯架构定义了针对取指和 load/store 操作的硬件监视点功能。软件在配置了取指和 load/store 的监视点后，处理器硬件将监视取指和 load/store 操作的访存地址，在其满足监视点设置条件时将触发监视点例外。

监视点相关控制状态寄存器用作软件配置取指和 load/store 操作监视点的接口。load/store 监视点和取指监视点各自的控制状态寄存器布局相似，都是一个关于所有监视点整体配置的寄存器、一个记录所有

监视点状态的寄存器以及每个监视点各自进行配置所需的四个寄存器。其中 load/store 监视点整体配置寄存器的地址为 0x300，load/store 监视点整体状态寄存器的地址为 0x301，第 n 个 load/store 监视点的 1~4 号四个配置寄存器的地址依次是 0x310+8n、0x311+8n、0x312+8n、0x313+8n；取指监视点整体配置寄存器的地址为 0x380，取指监视点整体状态寄存器的地址为 0x381，第 n 个取指监视点的 1~4 号四个配置寄存器的地址依次是 0x390+8n、0x391+8n、0x392+8n、0x393+8n。

load/store 监视点和取指监视点各自至多实现 14 个，其实际个数由实现具体决定。软件可以通过读取 CSR.MWPC.Num 和 CSR.FWPC.Num 的值来确定可以使用多少个硬件监视点。

7.9.1 load/store 监视点整体配置（MWPC）

该寄存器包含的配置信息用于告知软件 load/store 监视点确切的个数。

提醒关注的是，所有监视点的全局使能控制信号是在 CSR.CRMD 的 WE 位。

表 7-56 load/store 监视点整体配置寄存器定义

位	名字	读写	描述
5:0	Num	R	load/store 监视点的个数。
19:16	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
31:20	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.9.2 load/store 监视点整体状态（MWPS）

表 7-57 CSR 0x301 寄存器定义

位	名字	读写	描述
n-1:0	Status	RW1	load/store 监视点的命中情况。其与监视点一一对应，比特 i 对应第 i 号监视点。 当有 load/store 操作的地址命中某个监视点，则其对应的比特被置 1。除复位期间外，硬件不会将这个域的比特清 0。 软件只能通过对比特写 1 将其清 0，对比特写 0 将被忽略。
15:n	0	R	只读恒为 0，写被忽略。

位	名字	读写	描述
16	Skip	RW	软件通过将该位置 1，用以通知硬件忽略下一次的 load/store 监视点命中结果。所谓忽略是指既不将本寄存器 Stauts 域中的对应比特置 1 也不触发监视点例外。该功能可以在不取消监视点的情况下避免无休止的重复触发同一监视点，从而简化监视点例外的处理。 当 Skip 位为 1 时，如果硬件遇到了一次 load/store 监视点命中的情况，会在忽略该命中结果的同时将 Skip 位清为 0。这意味着每次软件将 Skip 位置 1 后，硬件至多忽略一次监视点命中。该特性还会造成软件对该位写 1 后读出的值未必是 1。 该 Skip 位对应所有的 load/store 监视点。如果软件修改了断点的配置，更换了断点，则不要设置该位，甚至为了安全起见，需要写 0 清除该位。
31:17	0	R	只读恒为 0，写被忽略。

7.9.3 load/store 监视点 n 配置 1~4（MWPnCFG1~4）

每个 load/store 监视点的配置 1~3 寄存器中包含的信息直接用于监视点检查的比较判断。假设待比较的操作的地址是 maddr，字节范围是 mbyten，每个监视点的命中的判断过程如下：

1. 如果 CSR.CRMD.WE=0，判断终止，否则转 2；
2. 如果当前不处于调试模式但 MWPCFG3 的 DSOnly 位等于 1，判断终止，否则转 3；
3. 如果 MWPCFG3 中 PLV0~PLV3 中对应当前特权等级的那一位等于 0，判断终止，否则转 4；
4. 如果该操作是 load 操作但是 MWPCFG3 中 LoadEn 位等于 0，或者该操作是 store 操作但是 MWPCFG3 中的 StoreEn 位等于 0，判断终止，否则转 5；
5. 如果 MWPCFG3 中 LCL 位等于 1，但是 CSR.ASID.ASID 不等于 MWPCFG4 中的 ASID，判断终止，否则转 6；
6. 如果(maddr & (~MWPCFG2.Mask)) != (MWPCFG1.VAaddr & (~MWPCFG2.Mask))，即地址比较不相等，判断终止，否则转 7；
7. 如果(~bytemask[7:0] & mbyten[7:0])等于全 0，判断终止，否则认为命中该监视点。

下面对上面判断过程描述中出现的 mbyten 和 bytemask 两个概念做进一步说明。

mbyten 表示操作涉及的字节，这是一个 8 比特的位向量，其值与 load/store 操作的类型以及地址低位值有关，具体的定义如表 7-58 所示：

表 7-58 load/store 监视点判断过程 mbyten 定义

指令名	maddr[2:0]							
	0	1	2	3	4	5	6	7
LD[X].B[U], ST[X].B, LD{GT/LE}.B, ST{GT/LE}.B	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80
LD[X].H[U], ST[X].H LD{GT/LE}.H, ST{GT/LE}.H	0x03		0x0C		0x30		0xC0	

指令名	maddr[2:0]							
	0	1	2	3	4	5	6	7
LD[X].W[U], ST[X].W, LD{GT/LE}.W, ST{GT/LE}.W, LDPTR.W, STPTR.W, LL.W, SC.W, AM{SWAP/ADD/AND/OR/XOR/MAX/MIN}[_DB].W, AM{MAX/MIN}[_DB].WU, FLD[X].S, FST[X].S, FLD{GT/LE}.S, FST{GT/LE}.S	0x0F				0xF0			
LD[X].D, ST[X].D, LD{GT/LE}.D, ST{GT/LE}.D, LDPTR.D, STPTR.D, LL.D, SC.D, AM{SWAP/ADD/AND/OR/XOR/MAX/MIN}[_DB].D, AM{MAX/MIN}[_DB].DU, FLD[X].D, FST[X].D, FLD{GT/LE}.D, FST{GT/LE}.D	0xFF							

bytemask 表示监视点比较时哪些字节不参与比较的掩码，这是一个 8 比特的位向量，其值与 MWPCFG1 中的 VAddr 的低位以及 MWPCFG3 中的 Size 有关，具体定义如所示。

表 7-59 load/store 监视点 bytemask 定义

MWPCFG3.Size	MWPCFG1.VAddr[2:0]							
	0	1	2	3	4	5	6	7
0b00	0x00							
0b01	0xF0				0x0F			
0b10	0xFC		0xF3		0xCF		0x3F	
0b11	0xFE	0xFD	0xFB	0xF7	0xEF	0xDF	0xBF	0x7F

表 7-60 load/store 监视点配置 1 寄存器定义

位	名字	读写	描述
GRLEN-1:0	VAddr	RW	该 load/store 监视点待比较的虚地址。

表 7-61 load/store 监视点配置 2 寄存器定义

位	名字	读写	描述
GRLEN-1:0	Mask	RW	该 load/store 监视点地址比较的屏蔽位。若第 i 位（0≤i<GRLEN）为 1，表示地址的第 i 位不参与比较。

表 7-62 load/store 监视点配置 3 寄存器定义

位	名字	读写	描述
0	DSOnly	RW	该位为 1 表示该 load/store 监视点仅在调试模式下可用。这里“可用”包含两方面含义：一是该监视点的配置寄存器可以在这个模式下被软件修改，二是该监视点检查命中后仅在这个模式下才会触发监视点例外并标记监视点状态。 该位仅在调试模式下（CSR.DBG.DS=1）才可被修改。这意味着运行在调试模式下的（上位机）软件具有监视点的优先使用权。
1	PLV0	RW	该监视点在 PLV0 特权等级下触发监视点例外的使能。1——使能，0——禁止。
2	PLV1	RW	该监视点在 PLV1 特权等级下触发监视点例外的使能。1——使能，0——禁止。
3	PLV2	RW	该监视点在 PLV2 特权等级下触发监视点例外的使能。1——使能，0——禁止。
4	PLV3	RW	该监视点在 PLV3 特权等级下触发监视点例外的使能。1——使能，0——禁止。
6:5	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
7	LCL	RW	为 1 表示这是一个局部监视点。
8	LoadEn	RW	为 1 表示对 load 操作进行监视点检查，否则不检查。
9	StoreEn	RW	为 1 表示对 store 操作进行监视点检查，否则不检查。
11:10	Size	RW	进行监视点检查时，哪些字节落在比较范围内。
31:12	0	R0	保留域。读返回 0，且软件不允许改变其值。

表 7-63 load/store 监视点配置 4 寄存器定义

位	名字	读写	描述
9:0	ASID	RW	被比较的 ASID
15:10	0	R	只读恒为 0，写被忽略。
23:16	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
31:24	0	R	只读恒为 0，写被忽略。

7.9.4 取指监视点整体配置（FWPC）

该寄存器包含的配置信息用于告知软件取指监视点确切的个数。

提醒关注的是，所有监视点的全局使能控制信号是在 CSR.CRMD 的 WE 位。

表 7-64 取指监视点整体配置寄存器定义

位	名字	读写	描述
5:0	Num	R	取指监视点的个数。
19:16	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
31:20	0	R0	保留域。读返回 0，且软件不允许改变其值。

7.9.5 取指监视点整体状态（FWPS）

表 7-65 CSR 0x301 寄存器定义

位	名字	读写	描述
n-1:0	Status	RW1	取指监视点的命中情况。其与监视点一一对应，比特 i 对应第 i 号监视点。 当有取指的 PC 命中某个监视点，则其对应的比特被置 1。除复位期间外，硬件不会将这个域的比特清 0。 软件只能通过对比特写 1 将其清 0，对比特写 0 将被忽略。
15:n	0	R	只读恒为 0，写被忽略。
16	Skip	RW	软件通过将该位置 1，用以通知硬件忽略下一次的取指监视点命中结果。所谓忽略是指既不将本寄存器 Stauts 域中的对应比特置 1 也不触发监视点例外。该功能可以在不取消监视点的情况下避免无休止的重复触发同一监视点，从而简化监视点例外的处理。 当 Skip 位为 1 时，如果硬件遇到了一次取指监视点命中的情况，会在忽略该命中结果的同时将 Skip 位清为 0。这意味着每次软件将 Skip 位置 1 后，硬件至多忽略一次监视点命中。该特性还会造成软件对该位写 1 后读出的值未必是 1。 该 Skip 位对应所有的取指监视点。如果软件修改了断点的配置，更换了断点，则不要设置该位，甚至为了安全起见，需要写 0 清除该位。
31:17	0	R	只读恒为 0，写被忽略。

7.9.6 取指监视点 n 配置 1~3（FWPnCFG1~3）

每个取指监视点的配置 1~3 寄存器中包含的信息直接用于监视点检查的比较判断。每个监视点的命中的判断过程如下：

1. 如果 CSR.CRMD.WE=0，判断终止，否则转 2；
2. 如果当前不处于调试模式但 FWPCFG3 的 DSOnly 位等于 1，判断终止，否则转 3；
3. 如果 FWPCFG3 中 PLV0~PLV3 中对应当前特权等级的那一位等于 0，判断终止，否则转 4；
4. 如果 FWPCFG3 中 LCL 位等于 1，但是 CSR.ASID.ASID 不等于 FWPCFG4 中的 ASID，判断终止，否则转 6；

5. 如果(pc & (~FWPCFG2.Mask)) != (FWPCFG1.VAddr & (~FWPCFG2.Mask))，即地址比较不相等，判断终止，否则认为命中该监视点。

表 7-66 取指监视点配置 1 寄存器定义

位	名字	读写	描述
GRLEN-1:0	VAddr	RW	该取指监视点待比较的虚地址。

表 7-67 取指监视点配置 2 寄存器定义

位	名字	读写	描述
GRLEN-1:0	Mask	RW	该取指监视点地址比较的屏蔽位。若第 i 位（0≤i<GRLEN）为 1，表示地址的第 i 位不参与比较。

表 7-68 取指监视点配置 3 寄存器定义

位	名字	读写	描述
0	DSOnly	RW	该位为 1 表示该取指监视点仅在调试模式下可用。这里“可用”包含两方面含义：一是该监视点的配置寄存器可以在这个模式下被软件修改，二是该监视点检查命中后仅在这个模式下才会触发监视点例外并标记监视点状态。 该位仅在调试模式下（CSR.DBG.DS=1）才可被修改。这意味着运行在调试模式下的（上位机）软件具有监视点的优先使用权。
1	PLV0	RW	该监视点在 PLV0 特权等级下触发监视点例外的使能。1——使能，0——禁止。
2	PLV1	RW	该监视点在 PLV1 特权等级下触发监视点例外的使能。1——使能，0——禁止。
3	PLV2	RW	该监视点在 PLV2 特权等级下触发监视点例外的使能。1——使能，0——禁止。
4	PLV3	RW	该监视点在 PLV3 特权等级下触发监视点例外的使能。1——使能，0——禁止。
6:5	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。
7	LCL	RW	为 1 表示这是一个局部监视点。
31:8	0	R0	保留域。读返回 0，且软件不允许改变其值。

表 7-69 取指监视点配置 4 寄存器定义

位	名字	读写	描述
9:0	ASID	RW	被比较的 ASID
15:10	0	R	只读恒为 0，写被忽略。
23:16	0	R	若未实现虚拟化扩展，则该域只读恒为 0，写被忽略。

位	名字	读写	描述
31:24	0	R	只读恒为 0，写被忽略。

7.10 调试相关控制状态寄存器

7.10.1 调试寄存器（DBG）

表 7-70 调试寄存器定义

位	名字	读写	描述
0	DS	R	为 1 表示当前处于调试模式。 在非调试模式下触发调式例外时，硬件将此位置 1。 当该位为 1 时，执行 ERTN 指令将该位清 0。
7:1	DRev	R	调试机制的版本号。1 为初始版本。
8	DEI	R	为 1 表示陷入调试模式的调试例外类型是调试外部中断例外（DEI）。
9	DCL	R	为 1 表示陷入调试模式的调试例外类型是调试调用例外（DCL）。
10	DFW	R	为 1 表示陷入调试模式的调试例外类型是调试取指监视点例外（DFW）。
11	DMW	R	为 1 表示陷入调试模式的调试例外类型是调试 load/store 监视点例外（DMW）。
15:12	0	R0	只读为 0
21:16	Ecode	R	调试模式下发生非调试例外时，将例外类型编码记录在这里。这里的编码含义与表 7-8 中的定义基本一致，仅有三点区别： • TLB 重填例外复用了 0x7 号例外码； • 调试调用例外复用了 0xC 号例外码； • 机器错误例外复用了 0xE 号例外码。
31:22	0	R0	只读为 0

7.10.2 调试例外返回地址（DERA）

表 7-71 调试例外返回地址寄存器定义

位	名字	读写	描述
63:0	PC	RW	在非调试模式下触发调式例外时，硬件将触发该例外的 PC 记录于此。 当 CSR.DBG.DS=1 时，执行 ERTN 指令时，从这里取回返回地址。

7.10.3 调试数据保存（DSAVE）

该寄存器用于给系统软件暂存数据。每个数据保存寄存器可以存放一个通用寄存器的数据。

之所以额外设置一个供调试例外处理程序使用的 SAVE 寄存器，是因为调试例外可以发生在任何场景下且调试例外的处理对于被调试主机上的软件要呈现透明的效果。

表 7-72 调试数据保存寄存器定义

位	名字	读写	描述
63:0	Data	RW	仅供软件读写的数据。除执行 CSR 指令外，硬件不会修改该域的内容。

8 附录 A 功能定义伪码描述

8.1 伪码中操作符释义

本节列举了伪码中涉及的语句关键字和各类操作符的含义，以及操作符优先级关系。

另外，在伪码中数值的常见不同进制表示方法约定如下：

- 没有前缀或采用“'d”或“##'d”前缀表示十进制数，其中“##'d”的前缀表示这个十进制数的位宽为##比特；
- 采用“'b”或“##'b”前缀表示二进制数，其中“##'b”的前缀表示这个二进制数的位宽为##比特；
- 采用“'h”或“##'h”前缀表示十六进制数，其中“##'h”的前缀表示这个十六进制数的位宽为##比特，十六进制的数值中 A~F 使用大写字母。

表 8-1 语句关键字释义

操作符	含义
返回类型 *函数名*（*变量*，…）： 　　*函数主体* return *返回值*	函数定义
if *判断条件 1*： 　　*执行语句 1* elif *判断条件 2*： 　　*执行语句 2* else： 　　*执行语句 3*	条件语句
case *判断变量* of： 　　*值 1*：*执行语句 1* 　　*值 2*：*执行语句 2* 　　default：*缺省执行语句*	case 条件语句
判断条件 ？*TRUE 执行语句* ：*FALSE 执行语句*	条件判断语句
for *循环变量* in *序列* ： 　　*执行语句*	for 循环语句
range(*N*)	0 到 N-1，步长为 1 的整数序列
range(*开始值*，*结束值*，*步长值*)	从开始值（含）到结束值（不含），指定步长值的序列
break	中止当前循环
signed(…)	有符号整数
unsigned(…)	无符号整数

操作符	含义
fp16(…)	半精度浮点数
fp32(…)	单精度浮点数
fp64(…)	双精度浮点数
boolean	布尔类型
bit	比特类型
integer	整数类型
bits(*N*)	N 比特类型
ZeroExtend(*变量*,*N*)	变量零扩展至 N 位
SignExtend(*变量*,*N*)	变量符号扩展至 N 位
isSNaN(*变量*)	变量是 signaling NaN 数则为 TRUE，否则为 FALSE
isQNaN(*变量*)	变量是 quiet NaN 数则为 TRUE，否则为 FALSE
SignalException(*例外*)	触发例外
#	单行注释
=	赋值

表 8-2 位串操作符释义

操作符	含义
[*M*:*N*]	位串的 N 到 M 位
{*N*{*M*}}	位串 M 复制 N 次拼接
{*N*, *M*, …}	位串 N，M，…依次拼接

表 8-3 算术运算符释义

操作符	含义
+	加
-	减
*	乘
/	除
%	取模
**	幂

表 8-4 比较运算符释义

操作符	含义
==	等于
!=	不等于

操作符	含义
>	大于
<	小于
>=	大于等于
<=	小于等于

表 8-5 位运算符释义

操作符	含义
&	按位与
|	按位或
^	按位异或
~	按位取反
<<	逻辑左移
>>	逻辑右移
>>>	算术右移

表 8-6 逻辑运算符释义

操作符	含义
and	逻辑与
or	逻辑或
not	逻辑非

伪码中运算符优先级由高到低列举如表 8-7 所示：

表 8-7 运算符优先级

运算符	含义
**	幂
~	按位取反
*, /, %	乘，除，取模
+, -	加，减
<<, >>, >>>	逻辑左移，逻辑右移，算术右移
&	按位与
^, |	按位异或，按位或
>, <, >=, <=	大于，小于，大于等于，小于等于
==, !=	等于，不等于
not	逻辑非
and, or	逻辑与，逻辑或

8.2 功能函数的伪码描述

本手册指令描述中涉及的伪代码定义如下。

8.2.1 逻辑左移

```
bits(N) SLL(bits(N) x, integer sa):
    if sa==0 :
        result = x
    else :
        result = {x[N-sa-1:0], {sa{1'b0}}}
    return result
```

8.2.2 逻辑右移

```
bits(N) SRL(bits(N) x, integer sa):
    if sa==0 :
        result = x
    else :
        result = {{sa{1'b0}}, x[N-1:sa]}
    return result
```

8.2.3 算术右移

```
bits(N) SRA(bits(N) x, integer sa):
    if sa==0 :
        result = x
    else :
        result = {{sa{x[N-1]}}, x[N-1:sa]}
    return result
```

8.2.4 循环右移

```
bits(N) ROTR(bits(N) x, integer sa):
    if sa==0 :
        result = x
    else :
```

```
        result = {x[sa-1:0], x[N-1:sa]}
    return result
```

8.2.5 统计高位起始连续 1 的个数

```
{bits(N) } CLO(bits(N) x):
    cnt = 0
    for i in range(N) :
        if x[N-1-i]==1'b0 :
            return cnt
        else :
            cnt = cnt + 1
```

8.2.6 统计高位起始连续 0 的个数

```
{bits(N) } CLZ(bits(N) x):
    cnt = 0
    for i in range(N) :
        if x[N-1-i]==1'b1 :
            return cnt
        else :
            cnt = cnt + 1
```

8.2.7 统计低位起始连续 1 的个数

```
{bits(N) } CTO(bits(N) x):
    cnt = 0
    for i in range(N) :
        if x[i]==1'b0 :
            return cnt
        else :
            cnt = cnt + 1
```

8.2.8 统计低位起始连续 0 的个数

```
{bits(N) } CTZ(bits(N) x):
    cnt = 0
    for i in range(N) :
```

```
        if x[i]==1'b1 :
            return cnt
        else :
            cnt = cnt + 1
```

8.2.9 比特串逆序

```
{bits(N) } BITREV(bits(N) x):
    for i in range(N) :
        res[i] = x[N-1-i]
    return res
```

8.2.10 CRC-32 校验和计算

```
bits(32) CRC32(old_chksum, msg, width, poly):
    new_chksum = (old_chksum & 0xFFFFFFFF) ^ {{(64-width){1'b0}}, msg}
    for i in range(width):
        if (new_chksum & 1'b1):
            new_chksum = (new_chksum >> 1) ^ poly
        else:
            new_chksum = ((new_chksum >> 1)
    return new_chksum
```

8.2.11 单精度浮点数转有符号字整数

```
{bits(32) } FP32convertToSint32(bits(32) x, bits(2) rm):
    case {rm} of:
        {2'd0}: return Sint32_convertToIntegerExactTiesToEven(x)
        {2'd1}: return Sint32_convertToIntegerExactTowardZero(x)
        {2'd2}: return Sint32_convertToIntegerExactTowardPositive(x)
        {2'd3}: return Sint32_convertToIntegerExactTowardNegative(x)
```

8.2.12 单精度浮点数转有符号双字整数

```
{bits(64) } FP32convertToSint64(bits(32) x, bits(2) rm):
    case {rm} of:
        {2'd0}: return Sint32_convertToIntegerExactTiesToEven(x)
        {2'd1}: return Sint32_convertToIntegerExactTowardZero(x)
```

```
        {2'd2}: return Sint32_convertToIntegerExactTowardPositive(x)
        {2'd3}: return Sint32_convertToIntegerExactTowardNegative(x)
```

8.2.13 双精度浮点数转有符号字整数

```
{bits(64) } FP64convertToSint32(bits(64) x, bits(2) rm):
    case {rm} of:
        {2'd0}: return Sint64_convertToIntegerExactTiesToEven(x)
        {2'd1}: return Sint64_convertToIntegerExactTowardZero(x)
        {2'd2}: return Sint64_convertToIntegerExactTowardPositive(x)
        {2'd3}: return Sint64_convertToIntegerExactTowardNegative(x)
```

8.2.14 双精度浮点数转有符号双字整数

```
{bits(64) } FP64convertToSint64(bits(64) x, bits(2) rm):
    case {rm} of:
        {2'd0}: return Sint64_convertToIntegerExactTiesToEven(x)
        {2'd1}: return Sint64_convertToIntegerExactTowardZero(x)
        {2'd2}: return Sint64_convertToIntegerExactTowardPositive(x)
        {2'd3}: return Sint64_convertToIntegerExactTowardNegative(x)
```

8.2.15 单精度浮点数取整

```
{bits(32) } FP32_roundToInteger(bits(N) x):
    return FP32_roundToIntegralExact(x)
```

8.2.16 双精度浮点数取整

```
{bits(64) } FP64_roundToInteger(bits(N) x):
    return FP64_roundToIntegralExact(x)
```

9 附录 B 指令码一览

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLO.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
CLZ.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
CTO.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
CTZ.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	rj					rd				
CLO.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
CLZ.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
CTO.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
CTZ.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
REVB.2H	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
REVB.4H	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	rj					rd				
REVB.2W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
REVB.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	rj					rd				
REVH.2W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
REVH.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
BITREV.4B	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
BITREV.8B	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
BITREV.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	rj					rd				
BITREV.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
EXT.W.H	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
EXT.W.B	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	rj					rd				
RDTIMEL.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
RDTIMEH.W	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
RDTIME.D	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
CPUCFG	rd, rj	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	rj					rd				
ASRTLE.D	rj, rk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	rk				rj			0	0	0	0	0
ASRTGT.D	rj, rk	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	rk				rj			0	0	0	0	0
ALSL.W	rd, rj, rk, sa2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	sa2			rk				rj			rd				
ALSL.WU	rd, rj, rk, sa2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	sa2			rk				rj			rd				
BYTEPICK.W	rd, rj, rk, sa2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	sa2			rk				rj			rd				
BYTEPICK.D	rd, rj, rk, sa3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	sa3				rk				rj			rd				
ADD.W	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	rk							rj			rd				
ADD.D	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	rk							rj			rd				
SUB.W	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	rk							rj			rd				
SUB.D	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	rk							rj			rd				
SLT	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	rk							rj			rd				
SLTU	rd, rj, rk	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	rk							rj			rd				

		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15	14 13 12 11 10	09 08 07 06 05	04 03 02 01 00
MASKEQZ	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0	rk	rj	rd
MASKNEZ	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1	rk	rj	rd
NOR	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0	rk	rj	rd
AND	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1	rk	rj	rd
OR	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0	rk	rj	rd
XOR	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1	rk	rj	rd
ORN	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0	rk	rj	rd
ANDN	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1	rk	rj	rd
SLL.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0	rk	rj	rd
SRL.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1	rk	rj	rd
SRA.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	rk	rj	rd
SLL.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1	rk	rj	rd
SRL.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0	rk	rj	rd
SRA.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1	rk	rj	rd
ROTR.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0	rk	rj	rd
ROTR.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1	rk	rj	rd
MUL.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0	rk	rj	rd
MULH.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1	rk	rj	rd
MULH.WU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0	rk	rj	rd
MUL.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1	rk	rj	rd
MULH.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0	rk	rj	rd
MULH.DU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1	rk	rj	rd
MULW.D.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0	rk	rj	rd
MULW.D.WU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1	rk	rj	rd
DIV.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	rk	rj	rd
MOD.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1	rk	rj	rd
DIV.WU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0	rk	rj	rd
MOD.WU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1	rk	rj	rd
DIV.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0	rk	rj	rd
MOD.D	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1	rk	rj	rd
DIV.DU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0	rk	rj	rd
MOD.DU	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1	rk	rj	rd
CRC.W.B.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0	rk	rj	rd
CRC.W.H.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1	rk	rj	rd
CRC.W.W.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0	rk	rj	rd
CRC.W.D.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1	rk	rj	rd
CRCC.W.B.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0	rk	rj	rd
CRCC.W.H.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1	rk	rj	rd
CRCC.W.W.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0	rk	rj	rd
CRCC.W.D.W	rd, rj, rk	0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1	rk	rj	rd
BREAK	code	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0	code		
DBCL	code	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1	code		

		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
SYSCALL	code	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 code
ALSL.D	rd, rj, rk, sa2	0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 sa2 rk rj rd
SLLI.W	rd, rj, ui5	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 ui5 rj rd
SLLI.D	rd, rj, ui6	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 ui6 rj rd
SRLI.W	rd, rj, ui5	0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 ui5 rj rd
SRLI.D	rd, rj, ui6	0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 ui6 rj rd
SRAI.W	rd, rj, ui5	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 ui5 rj rd
SRAI.D	rd, rj, ui6	0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 ui6 rj rd
ROTRI.W	rd, rj, ui5	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 ui5 rj rd
ROTRI.D	rd, rj, ui6	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 ui6 rj rd
BSTRINS.W	rd, rj, msbw, lsbw	0 0 0 0 0 0 0 0 0 1 1 msbw 0 lsbw rj rd
BSTRPICK.W	rd, rj, msbw, lsbw	0 0 0 0 0 0 0 0 0 1 1 msbw 1 lsbw rj rd
BSTRINS.D	rd, rj, msbd, lsbd	0 0 0 0 0 0 0 0 1 0 msbd lsbd rj rd
BSTRPICK.D	rd, rj, msbd, lsbd	0 0 0 0 0 0 0 0 1 1 msbd lsbd rj rd
FADD.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 fk fj fd
FADD.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 fk fj fd
FSUB.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 fk fj fd
FSUB.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 fk fj fd
FMUL.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 fk fj fd
FMUL.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 fk fj fd
FDIV.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 fk fj fd
FDIV.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 fk fj fd
FMAX.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 fk fj fd
FMAX.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 fk fj fd
FMIN.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 fk fj fd
FMIN.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 fk fj fd
FMAXA.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 fk fj fd
FMAXA.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 fk fj fd
FMINA.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 fk fj fd
FMINA.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 fk fj fd
FSCALEB.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 fk fj fd
FSCALEB.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 fk fj fd
FCOPYSIGN.S	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 fk fj fd
FCOPYSIGN.D	fd, fj, fk	0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 fk fj fd
FABS.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 fj fd
FABS.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 fj fd
FNEG.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 fj fd
FNEG.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 fj fd
FLOGB.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 fj fd
FLOGB.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 fj fd
FCLASS.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 fj fd
FCLASS.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 fj fd

		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10	09 08 07 06 05	04 03 02 01 00
FSQRT.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1	fj	fd
FSQRT.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0	fj	fd
FRECIP.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1	fj	fd
FRECIP.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0	fj	fd
FRSQRT.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 0 1	fj	fd
FRSQRT.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0	fj	fd
FMOV.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1	fj	fd
FMOV.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0	fj	fd
MOVGR2FR.W	fd, rj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1	rj	fd
MOVGR2FR.D	fd, rj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0	rj	fd
MOVGR2FRH.W	fd, rj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 1	rj	fd
MOVFR2GR.S	rd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1	fj	rd
MOVFR2GR.D	rd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0	fj	rd
MOVFRH2GR.S	rd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 1	fj	rd
MOVGR2FCSR	fcsr, rj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0	rj	fcsr
MOVFCSR2GR	rd, fcsr	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 1 0	fcsr	rd
MOVFR2CF	cd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0	fj	0 0 cd
MOVCF2FR	fd, cj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 1	0 0 cj	fd
MOVGR2CF	cd, rj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0	rj	0 0 cd
MOVCF2GR	rd, cj	0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1	0 0 cj	rd
FCVT.S.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0	fj	fd
FCVT.D.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1	fj	fd
FTINTRM.W.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1	fj	fd
FTINTRM.W.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0	fj	fd
FTINTRM.L.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 1	fj	fd
FTINTRM.L.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0	fj	fd
FTINTRP.W.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 1	fj	fd
FTINTRP.W.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 0 1 0	fj	fd
FTINTRP.L.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1	fj	fd
FTINTRP.L.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0	fj	fd
FTINTRZ.W.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1	fj	fd
FTINTRZ.W.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0	fj	fd
FTINTRZ.L.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1	fj	fd
FTINTRZ.L.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0	fj	fd
FTINTRNE.W.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0 0 0 1	fj	fd
FTINTRNE.W.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 0	fj	fd
FTINTRNE.L.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1	fj	fd
FTINTRNE.L.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 1 1 0 1 0	fj	fd
FTINT.W.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1	fj	fd
FTINT.W.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0	fj	fd
FTINT.L.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 1 0 0 1	fj	fd
FTINT.L.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0	fj	fd

		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10	09 08 07 06 05	04 03 02 01 00
FFINT.S.W	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0	fj	fd
FFINT.S.L	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0	fj	fd
FFINT.D.W	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0	fj	fd
FFINT.D.L	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0	fj	fd
FRINT.S	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1	fj	fd
FRINT.D	fd, fj	0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0	fj	fd
SLTI	rd, rj, si12	0 0 0 0 0 0 1 0 0 0	si12	rj	rd
SLTUI	rd, rj, si12	0 0 0 0 0 0 1 0 0 1	si12	rj	rd
ADDI.W	rd, rj, si12	0 0 0 0 0 0 1 0 1 0	si12	rj	rd
ADDI.D	rd, rj, si12	0 0 0 0 0 0 1 0 1 1	si12	rj	rd
LU52I.D	rd, rj, si12	0 0 0 0 0 0 1 1 0 0	si12	rj	rd
ANDI	rd, rj, ui12	0 0 0 0 0 0 1 1 0 1	ui12	rj	rd
ORI	rd, rj, ui12	0 0 0 0 0 0 1 1 1 0	ui12	rj	rd
XORI	rd, rj, ui12	0 0 0 0 0 0 1 1 1 1	ui12	rj	rd
CSRRD	rd, csr	0 0 0 0 0 1 0 0	csr	0 0 0 0 0	rd
CSRWR	rd, csr	0 0 0 0 0 1 0 0	csr	0 0 0 0 1	rd
CSRXCHG	rd, rj, csr	0 0 0 0 0 1 0 0	csr	rj!=0,1	rd
CACOP	code, rj, si12	0 0 0 0 0 1 1 0 0 0	si12	rj	code
LDDIR	rd, rj, level	0 0 0 0 0 1 1 0 0 1 0 0 0 0	level	rj	rd
LDPTE	rj, seq	0 0 0 0 0 1 1 0 0 1 0 0 0 1	seq	rj	0 0 0 0 0
IOCSRRD.B	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0	rj	rd
IOCSRRD.H	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1	rj	rd
IOCSRRD.W	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0	rj	rd
IOCSRRD.D	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1	rj	rd
IOCSRWR.B	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0	rj	rd
IOCSRWR.H	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1	rj	rd
IOCSRWR.W	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0	rj	rd
IOCSRWR.D	rd, rj	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1	rj	rd
TLBCLR		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0	0 0 0 0 0
TLBFLUSH		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1	0 0 0 0 0	0 0 0 0 0
TLBSRCH		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0	0 0 0 0 0	0 0 0 0 0
TLBRD		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1	0 0 0 0 0	0 0 0 0 0
TLBWR		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0	0 0 0 0 0	0 0 0 0 0
TLBFILL		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1	0 0 0 0 0	0 0 0 0 0
ERTN		0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0	0 0 0 0 0	0 0 0 0 0
IDLE	level	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1	level		
INVTLB	op, rj, rk	0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1	rk	rj	op
FMADD.S	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 0 0 1	fa	fk	fj	fd
FMADD.D	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 0 1 0	fa	fk	fj	fd
FMSUB.S	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 1 0 1	fa	fk	fj	fd
FMSUB.D	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 1 1 0	fa	fk	fj	fd
FNMADD.S	fd, fj, fk, fa	0 0 0 0 1 0 0 0 1 0 0 1	fa	fk	fj	fd

		31 30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15	14 13 12 11 10	09 08 07 06 05	04 03 02 01 00
FNMADD.D	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 1 0 1 0	fa	fk	fj	fd
FNMSUB.S	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 1 1 0 1	fa	fk	fj	fd
FNMSUB.D	fd, fj, fk, fa	0 0 0 0 1 0 0 0 0 1 1 1 0	fa	fk	fj	fd
FCMP.cond.S	cd, fj, fk	0 0 0 0 1 1 0 0 0 0 0 0 1	cond	fk	fj	0 0 cd
FCMP.cond.D	cd, fj, fk	0 0 0 0 1 1 0 0 0 0 0 1 0	cond	fk	fj	0 0 cd
FSEL	fd, fj, fk, ca	0 0 0 0 1 1 0 1 0 0 0 0 0	0 0 ca	fk	fj	fd
ADDU16I.D	rd, rj, si16	0 0 0 1 0 0	si16		rj	rd
LU12I.W	rd, si20	0 0 0 1 0 1 0	si20			rd
LU32I.D	rd, si20	0 0 0 1 0 1 1	si20			rd
PCADDI	rd, si20	0 0 0 1 1 0 0	si20			rd
PCALAU12I	rd, si20	0 0 0 1 1 0 1	si20			rd
PCADDU12I	rd, si20	0 0 0 1 1 1 0	si20			rd
PCADDU18I	rd, si20	0 0 0 1 1 1 1	si20			rd
LL.W	rd, rj, si14	0 0 1 0 0 0 0 0	si14		rj	rd
SC.W	rd, rj, si14	0 0 1 0 0 0 0 1	si14		rj	rd
LL.D	rd, rj, si14	0 0 1 0 0 0 1 0	si14		rj	rd
SC.D	rd, rj, si14	0 0 1 0 0 0 1 1	si14		rj	rd
LDPTR.W	rd, rj, si14	0 0 1 0 0 1 0 0	si14		rj	rd
STPTR.W	rd, rj, si14	0 0 1 0 0 1 0 1	si14		rj	rd
LDPTR.D	rd, rj, si14	0 0 1 0 0 1 1 0	si14		rj	rd
STPTR.D	rd, rj, si14	0 0 1 0 0 1 1 1	si14		rj	rd
LD.B	rd, rj, si12	0 0 1 0 1 0 0 0 0 0	si12		rj	rd
LD.H	rd, rj, si12	0 0 1 0 1 0 0 0 0 1	si12		rj	rd
LD.W	rd, rj, si12	0 0 1 0 1 0 0 0 1 0	si12		rj	rd
LD.D	rd, rj, si12	0 0 1 0 1 0 0 0 1 1	si12		rj	rd
ST.B	rd, rj, si12	0 0 1 0 1 0 0 1 0 0	si12		rj	rd
ST.H	rd, rj, si12	0 0 1 0 1 0 0 1 0 1	si12		rj	rd
ST.W	rd, rj, si12	0 0 1 0 1 0 0 1 1 0	si12		rj	rd
ST.D	rd, rj, si12	0 0 1 0 1 0 0 1 1 1	si12		rj	rd
LD.BU	rd, rj, si12	0 0 1 0 1 0 1 0 0 0	si12		rj	rd
LD.HU	rd, rj, si12	0 0 1 0 1 0 1 0 0 1	si12		rj	rd
LD.WU	rd, rj, si12	0 0 1 0 1 0 1 0 1 0	si12		rj	rd
PRELD	hint, rj, si12	0 0 1 0 1 0 1 0 1 1	si12		rj	hint
FLD.S	fd, rj, si12	0 0 1 0 1 0 1 1 0 0	si12		rj	fd
FST.S	fd, rj, si12	0 0 1 0 1 0 1 1 0 1	si12		rj	fd
FLD.D	fd, rj, si12	0 0 1 0 1 0 1 1 1 0	si12		rj	fd
FST.D	fd, rj, si12	0 0 1 0 1 0 1 1 1 1	si12		rj	fd
LDX.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 0	rk	rj	rd
LDX.H	rd, rj, rk	0 0 1 1 1 0 0 0 0 0 0 0 0 0 1	0 0 0	rk	rj	rd
LDX.W	rd, rj, rk	0 0 1 1 1 0 0 0 0 0 0 0 0 1 0	0 0 0	rk	rj	rd
LDX.D	rd, rj, rk	0 0 1 1 1 0 0 0 0 0 0 0 0 1 1	0 0 0	rk	rj	rd
STX.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 0 0 0 1 0 0	0 0 0	rk	rj	rd

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
STX.H	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	rk					rj					rd				
STX.W	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	rk					rj					rd				
STX.D	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	rk					rj					rd				
LDX.BU	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	rk					rj					rd				
LDX.HU	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	rk					rj					rd				
LDX.WU	rd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	rk					rj					rd				
PRELDX	hint, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	0	1	1	0	0	rk					rj					hint				
FLDX.S	fd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	rk					rj					fd				
FLDX.D	fd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	0	1	0	0	rk					rj					fd				
FSTX.S	fd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	rk					rj					fd				
FSTX.D	fd, rj, rk	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	rk					rj					fd				
AMSWAP.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	rk					rj					rd				
AMSWAP.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	rk					rj					rd				
AMADD.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	rk					rj					rd				
AMADD.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	rk					rj					rd				
AMAND.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	0	rk					rj					rd				
AMAND.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1	rk					rj					rd				
AMOR.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	0	rk					rj					rd				
AMOR.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1	rk					rj					rd				
AMXOR.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	0	rk					rj					rd				
AMXOR.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	rk					rj					rd				
AMMAX.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	0	rk					rj					rd				
AMMAX.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	rk					rj					rd				
AMMIN.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	1	0	rk					rj					rd				
AMMIN.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	1	1	rk					rj					rd				
AMMAX.WU	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	1	0	rk					rj					rd				
AMMAX.DU	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	1	1	rk					rj					rd				
AMMIN.WU	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	0	rk					rj					rd				
AMMIN.DU	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	rk					rj					rd				
AMSWAP_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	0	rk					rj					rd				
AMSWAP_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	rk					rj					rd				
AMADD_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	1	0	rk					rj					rd				
AMADD_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	1	1	rk					rj					rd				
AMAND_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	1	0	rk					rj					rd				
AMAND_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	1	1	rk					rj					rd				
AMOR_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	rk					rj					rd				
AMOR_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	1	rk					rj					rd				
AMXOR_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	rk					rj					rd				
AMXOR_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	1	rk					rj					rd				
AMMAX_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	1	0	rk					rj					rd				
AMMAX_DB.D	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	1	1	rk					rj					rd				
AMMIN_DB.W	rd, rk, rj	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	1	0	rk					rj					rd				

		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15	14 13 12 11 10	09 08 07 06 05	04 03 02 01 00
AMMIN_DB.D	rd, rk, rj	0 0 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1	rk	rj	rd
AMMAX_DB.WU	rd, rk, rj	0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0	rk	rj	rd
AMMAX_DB.DU	rd, rk, rj	0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1	rk	rj	rd
AMMIN_DB.WU	rd, rk, rj	0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0	rk	rj	rd
AMMIN_DB.DU	rd, rk, rj	0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1	rk	rj	rd
DBAR	hint	0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 0	hint		
IBAR	hint	0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1	hint		
FLDGT.S	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0	rk	rj	fd
FLDGT.D	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1	rk	rj	fd
FLDLE.S	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0	rk	rj	fd
FLDLE.D	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1	rk	rj	fd
FSTGT.S	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 0	rk	rj	fd
FSTGT.D	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1	rk	rj	fd
FSTLE.S	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 0	rk	rj	fd
FSTLE.D	fd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1	rk	rj	fd
LDGT.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0	rk	rj	rd
LDGT.H	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1	rk	rj	rd
LDGT.W	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0	rk	rj	rd
LDGT.D	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1	rk	rj	rd
LDLE.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 1 0 0	rk	rj	rd
LDLE.H	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 1 0 1	rk	rj	rd
LDLE.W	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 1 1 0	rk	rj	rd
LDLE.D	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1	rk	rj	rd
STGT.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0	rk	rj	rd
STGT.H	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1	rk	rj	rd
STGT.W	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 0	rk	rj	rd
STGT.D	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1	rk	rj	rd
STLE.B	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0	rk	rj	rd
STLE.H	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 1	rk	rj	rd
STLE.W	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0	rk	rj	rd
STLE.D	rd, rj, rk	0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1	rk	rj	rd
BEQZ	rj, offs	0 1 0 0 0 0	offs[15:0]	rj	offs[20:16]
BNEZ	rj, offs	0 1 0 0 0 1	offs[15:0]	rj	offs[20:16]
BCEQZ	cj, offs	0 1 0 0 1 0	offs[15:0]	0 0 cj	offs[20:16]
BCNEZ	cj, offs	0 1 0 0 1 0	offs[15:0]	0 1 cj	offs[20:16]
JIRL	rd, rj, offs	0 1 0 0 1 1	offs[15:0]	rj	rd
B	offs	0 1 0 1 0 0	offs[15:0]	offs[25:16]	
BL	offs	0 1 0 1 0 1	offs[15:0]	offs[25:16]	
BEQ	rj, rd, offs	0 1 0 1 1 0	offs[15:0]	rj	rd
BNE	rj, rd, offs	0 1 0 1 1 1	offs[15:0]	rj	rd
BLT	rj, rd, offs	0 1 1 0 0 0	offs[15:0]	rj	rd
BGE	rj, rd, offs	0 1 1 0 0 1	offs[15:0]	rj	rd

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BLTU	**rj, rd, offs**	0	1	1	0	1	0	offs[15:0]																rj					rd				
BGEU	**rj, rd, offs**	0	1	1	0	1	1	offs[15:0]																rj					rd				