玄铁 R807 用户手册

2024 年 04 月 24 日

杭州中天微系统有限公司 Hangzhou C-SKY MicroSystems Co., LTD

Address: Room 201, 2/F, Building 5, No.699 Wangshang Road , Hangzhou, Zhejiang, China

Website: www.xrvm.cn

杭州中天微系统有限公司 Hangzhou C-SKY MicroSystems Co., LTD

地址：中国浙江省杭州市网商路 699 号 5 号楼 2 楼 201 室

网址：www.xrvm.cn

版本历史

版本	描述	日期
01	第一次正式发布。	2020.08.03
02	更新 VBR 复位值；增加 MPID。	2021.08.24
03	修正了时钟使能信号的时序描述。	2021.12.02
04	修正了 MPU 区域优先级的描述。	2021.12.27
05	几处文字描述修正：C 位，异常处理概述，总线特性	2022.04.01

玄铁 R807 用户手册

第一章　概述

1.1　简介

R807 是玄铁面向 SSD/HDD、工业控制以及网络/路由等实时控制领域研发设计的 32 位高能效嵌入式 CPU 核，具有出色的实时性和性能表现。基于 16/32 位混合编码的玄铁 CPU V2 RISC 指令集，R807 采用了 8 级乱序双发射流水线，并具备丰富的实时性功能。其中，可灵活配置的紧耦合内存（TCM）系统能够提供低访问延时且相比高速缓存更加确定的内存访问延时。同时，TCM 系统配合着 64-bit AXI 的 TCM 从接口，有效提高了处理器核内和 DMA 之间数据交互的吞吐率。R807 还配置了专用的低延时外设访问接口，支持 32-bit 的 AXI 以及 AHB 外设接口，将关键的外设与 CPU 内核更加紧密的耦合，从而提高 CPU 访问特定私有外设的速度和效率。

此外，R807 重点针对功耗进行了优化，支持短循环低功耗执行、数据高速缓存过滤访问等低功耗技术。系统管理方面，R807 支持通过静态设计、动态电源管理和低电压供电来减少功耗，也支持进入省电模式来节省功耗。同时，R807 还支持实时检测并关断内部空闲功能模块，进一步降低处理器动态功耗。

1.2　特点

R807 处理器体系结构的主要特点如下：

- 精简指令集计算机结构（RISC）；
- 32 位数据长度，32 位或 16 位可变的指令长度；
- 双发射超标量 8 级流水线，对软件完全透明；
- 按序发射，乱序完成和按序退休；
- 内存保护单元（MPU），实现内存管理和保护；
- 可配置的指令高缓 I-Cache 和数据高缓 D-Cache；
- 最大 1MB 可配的紧耦合内存 I-TCM 和 D-TCM，其中 D-TCM 具备两组独立的访问接口；
- 基于 AXI 协议的 64-bit TCM SLAVE 接口；
- 1 条 AHB 外设高速接口，支持 32-bit 总线宽度；
- 1 条 AXI 外设高速接口，支持 32-bit 总线宽度；

- 1 条指令和数据共用的 AXI 通用总线接口，支持 64-bit 总线宽度；
- ECC 校验技术，可修正 1-bit 错误，检测 2-bit 错误；
- 支持用于测试的 ECC 注入功能；
- Lock Step 技术，可检测核内逻辑错误，配合 ECC 使用；
- 支持大端模式和小端模式；
- 混合分支预测技术；
- 存储器拷贝加速技术；
  - 支持通过控制寄存器读取高缓以及 TCM，用于调试
  - 内部硬件调试模块支持片上硬件调试；
- 支持向量中断和自动向量中断，支持中断快速响应；
- 可选的浮点执行单元，支持单精度和双精度。

1.3 可配置选项

R807 可配置选项如下表所示。

表 1.1: R807 可配置选项

可配置单元	定义	可选项
I-Cache	指令高速缓存	Size： 8K/16K/32K/64K
D-Cache	数据高速缓存	Size： 8K/16K/32K/64K
I-TCM	指令片上紧耦合内存	Size： None/4K/8K/1 6K/32K/64K/128K/256K/512K/1M Access latency： 单/双周期访问。
D-TCM	数据片上紧耦合内存	Size： None/4K/8K/1 6K/32K/64K/128K/256K/512K/1M Access latency： 单/双周期访问。 Interleave type： 按高位地址/按低 64-bit interleave 的方式划分 bank0/bank1
PHP_AXI	32-bit 外设 AXI 总线	有/无。
PHP_AHB	32-bit 外设 AHB 总线	有/无。
MPU	内存保护单元	Entry number： 12/16。
FPU	浮点运算单元	有/无。
ECC	ECC 校验	有/无。
Lock Step	Lock Step 功能	有/无（若配置的话，需同时配置 ECC）。

1.4 可调试性设计

R807 使用 JTAG 标准设计硬件调试接口。R807 支持所有常见的调试功能，包括软断点、内存断点，改寄存器检查和存储器检查和修改，指令单步跟踪与多步跟踪、程序流跟踪等。

1.5 命名规则

1.5.1 符号

本文档用到的标准符号和操作符如图表 图 1.1 所示。

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
≠	不等于
.	与
+	或
⊕	异或
NOT	取反
:	连接
⇒	传输
⇔	交换
±	误差
0b0011	二进制数
0x0F	十六进制数

图 1.1: 符号术语表

1.5.2 术语

- 逻辑 1 是指对应于布尔逻辑真的电平值。
- 逻辑 0 是指对应于布尔逻辑伪的电平值。
- 置位是指使得某个或某几个位达到逻辑 1 对应的电平值。
- 清除是指使得某个或某几个位达到逻辑 0 对应的电平值。
- 保留位是为功能的扩展而预留的，没有特殊说明时其值为 0。
- 信号是指通过它的状态或状态间的转换来传递信息的电气值。
- 引脚是表示一种外部电气物理连接，同一个引脚可以连接多个信号。
- 使能是指使某个离散信号处在有效的状态：
  低电平有效信号从高电平切换到低电平；
  高电平有效信号从低电平切换到高电平。
- 禁止是指使某个处在使能状态的信号状态改变：
  低电平有效信号从低电平切换到高电平；
  高电平有效信号从高电平切换到低电平。
- LSB 代表最低有效位,MSB 代表最高有效位。
  存储单元和寄存器当“pad_biu_bigend_b=0”时采用高位收尾模式，其字节次序是高字节在最低位的排列。一个字中的所有位是从最高有效位 (第 31 位) 开始往下排列。
- 当“pad_biu_bigend_b=1”时，采用低位收尾模式。
- 信号，位域，控制位的表示都使用一种通用的规则。
- 标识符后来跟着表示范围的数字，从高位到低位表示一组信号，比如 addr[4:0] 就表示一组地址总线，最高位是 addr[4]，最低位是 addr[0]。
- 单个的标识符就表示单个信号，例如 pad_biu_reset_b 就表示单独的一个信号。有时候会在标识符后加上数字表示一定的意义，比如 addr15 就表示一组总线中的第 16 位。

第二章　功能

2.1　处理器微架构

R807 结构框图如下所示：

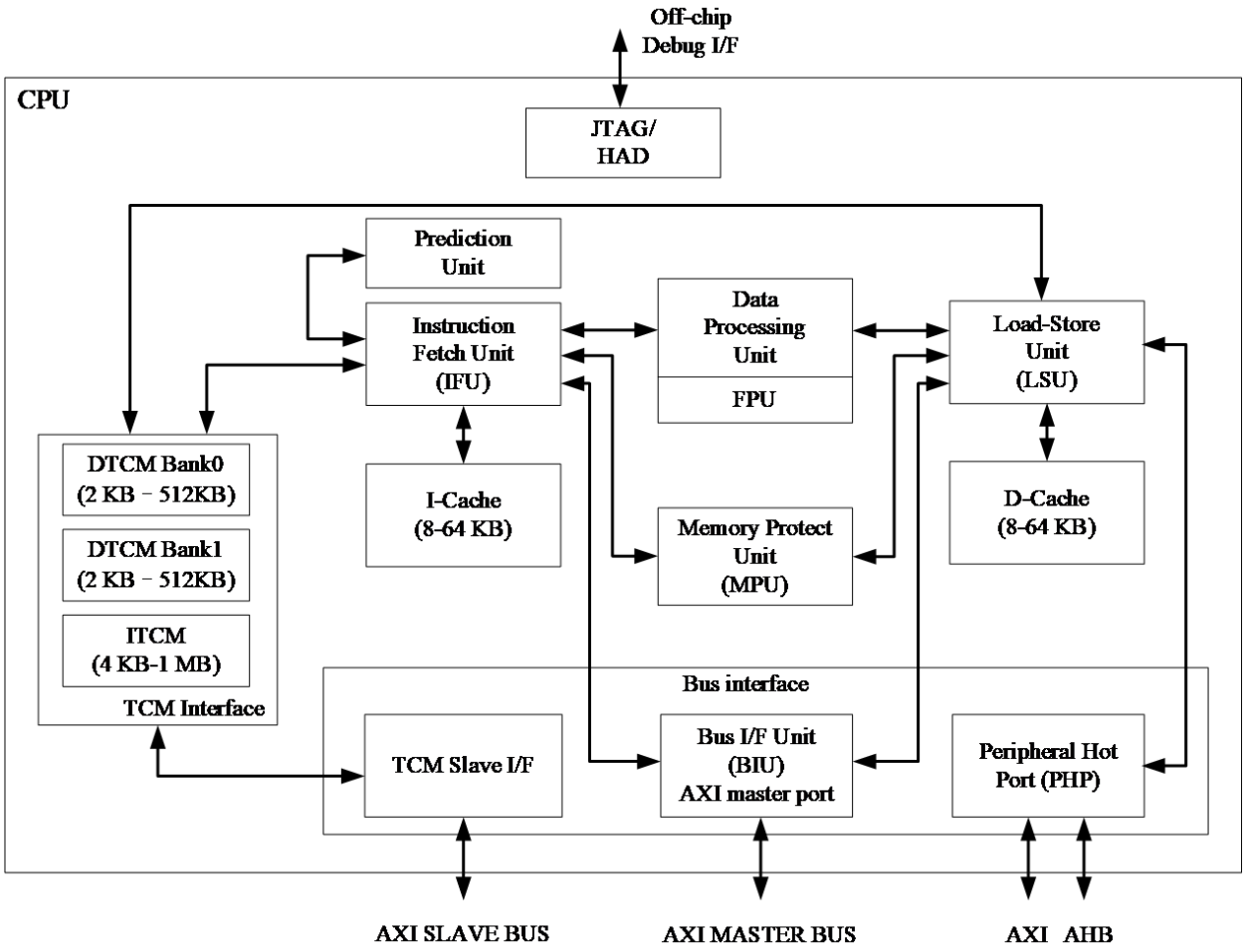


图 2.1: R807 结构图

2.1.1 流水线介绍

R807 处理器使用了 8 级流水线结构，具体如下表所示。

表 2.1: R807 流水线

流水线名称	缩写	流水线作用
指令访问	IF	1、访问指令高速缓存或者 TCM，获取 64-bit 指令数据。
指令预译码	IP	1. 指令预译码； 2. 将输出的指令码封装成有效指令； 3. 处理一部分分支指令。
指令缓存	IB	1. 提取并封装出最多 4 条有效指令； 2. 缓存预译码指令； 3. 间接跳转目标地址和返回地址预测； 4. 相关性检测。
指令译码	ID	1. 指令译码； 2. 指令发射队列创建； 3. 指令乱序调度。
指令执行 1	EX1	1. 指令执行（ALU 指令和分支指令产生结果）；
（地址产生）	(AG)	2. 存储加载指令产生访问地址。
指令执行 2	EX2	1. 乘除法指令执行；
（数据访问）	(DC)	2. 访问数据高速缓存。
指令执行 3	EX3	1. 乘除法指令产生结果；
（地址比较）	(DA)	2. 存储加载指令产生结果。
回写	WB	1、指令退休； 2、将指令执行结果回写到寄存器堆。

指令提取单元 (IFU) 一次可最多提取四条指令并对其并行处理；配备高速缓存，在高速缓存缺失时采用关键指令先取和发射，以及后续指令旁路技术；配备指令暂存器缓存预取指令；采用先进二级指令分支跳转

预测，可最多同时预测四条分支指令，预测精度高。整个指令提取单元拥有低功耗，高发射效率的特点。

指令译码单元 (IDU) 可以同时对两条指令进行译码，并检测出指令间的数据相关性。指令译码单元根据后继流水线执行情况，及时更新指令的数据相关性信息，并将指令乱序发送至下级流水线执行。指令译码单元支持多达 5 条指令的乱序执行调度。除此之外，指令译码单元还能够分解 LDM/STM 等复杂指令，简化执行逻辑。

存储载入单元 (LSU) 支持存储/加载指令的按序执行，支持高速缓存的非阻塞访问。具有内部前馈机制，消除存储指令回写数据的相关性。支持字节、半字、字和双字的存储/载入指令，并支持字节和半字的载入指令的符号位和 0 扩展。支持非对齐访问。存储/加载指令可以流水执行，使得数据吞吐量达到一个周期存取一个数据。

执行单元 (IU) 包含 2 条整型流水线和 1 条存储流水线。整型流水线包含 2 个算术逻辑单元（ALU），1 个乘除法单元 (MAD) 和 1 个分支跳转单元 (BJU)。

ALU 执行标准的 32 位整数操作，单周期产生运算结果。ALU 支持快速查找 1/0 算法（FF1/FF0），支持移位加操作（IXH，IXW，IXD）等。

一个条件/进位位（C）用于条件测试和多于 32 位的算术逻辑运算。通常，C 位只在显式的测试/比较运算中被改变，但是在某些特定的把条件设置和运算结合的指令中可能例外。

ALU 通过操作数前馈减少数据真相关，单周期 ALU 指令不存在数据真相关停顿延时。

MAD 支持 16*16,16*32,32*32 整数乘法，支持乘累加、乘累减操作。除法器的设计采用了快速算法，执行周期 4~36 不等。

BJU 支持快速分支预测错误处理，加速处理器性能。

指令退休单元 (RTU) 包括一个 8 表项的重排序缓冲器，最多支持 8 条指令的并行乱序执行。重排序缓冲器负责指令的乱序回收与按序退休，并实现结果的按序回写。通过支持指令并行回收与快速退休提高指令退休效率和指令执行带宽。指令退休单元每个时钟周期并行退休与写回两条指令，负责实现精确异常，支持普通中断和快速中断。

总线接口单元（BIU）支持 AXI 协议，支持关键字优先的地址访问，可以在不同的系统时钟与 CPU 时钟比例（1:1，1:2，1:3，1:4，1:5，1:6，1:7，1:8）下工作。

硬件辅助调试单元（HAD）支持各种调试方式，包括软件设置断点方式、硬件设置断点方式、单步和多步指令跟踪、跳转指令跟踪等 6 种方式，可以访问各种核内资源以及内存。

2.2 功能部件介绍

2.2.1 中断快速响应

R807 支持矢量和自动矢量中断，并且针对应用中的实时性需求，R807 为中断快速响应设计了加速机制。

处理器可以被配置为中断普通响应模式以及中断快速响应模式，通过软件配置。在中断普通响应模式下，中断响应耦合与指令退休时机，一旦遇上多周期指令或者执行延时不可预期指令时，中断响应速度会受到极大影响，大幅降低系统的实时性。通过置位 CR<31,0>[8]，可软件开启中断快速响应模式，当中断快速响应

模式被使能时，处理器无需等待指令退休，即可直接响应中断。这一机制可以打断较长执行延时的指令，从而提高中断响应的平均速度。

支持无退休响应中断的指令包含：

- 整型算术运算指令；
- 乘除法指令（回写 HILO 的指令需要等待退休回写）；
- 浮点指令；
- 对内存进行清除无效操作的指令；
- weak-order 的加载存储指令（包括尚未被 LSU 标记为 strong-order 的指令）；
- 分支跳转指令；

有部分指令无法在已经被发送到执行单元的情况下被打断，因此当这些指令为流水线上最老指令时，处理器无法在未退休的情况下响应中断。不支持无退休响应中断的指令包括：

- 已经被 LSU 标记为 Strong-order 的加载存储指令；
- 原子操作 STEX 指令；
- 特殊指令，包含 CP0 等特权态指令。

2.2.2 内存保护单元

物理内存保护单元（MPU）负责对存储器系统（包括外围设备）的访问合法性进行检查，其主要功能是判定当前工作模式下 CPU 是否具备对内存地址的读/写/执行访问权限。支持 12/16 个表项可配置，用户通过对内存保护单元的设置可定义每个保护区的访问属性。

内存保护单元对 CPU 内存访问的处理流程及结果如图表 表 2.2 所示。

表 2.2: 内存访问处理流程及结果

内存保护单元 使能情况	内存保护单元 命中情况	处理结果
不使能	不关心	所有对内存的访问都视为: 超级用户模式/普通用户模式均可读可写; 访问属性为 un-bufferable，un-cacheable，not-secure，strong order。
使能	不命中	处理器将产生访问错误异常。
	命中	如果访问命中一个保护区，将以该保护区的访问权限与属性为准; 如果访问地址命中多个保护区，则以索引号最高的保护区的访问权限与属性为准。 该地址的访问权限结合当前处理器的工作模式，判定该访问是否合法。如果访问符合该区域的访问权限，则内存保护单元允许本次内存操作，并提供该访问地址的访问属性；如果访问请求不被允许，则内存保护单元将产生一个访问错误异常并中断处理器的执行。

注意: 访问地址可能命中多个保护区，此时认为命中索引号最高的保护区。

2.2.3 TCM 系统介绍

R807 支持硬件可配片上核内紧耦合内存 I-TCM/D-TCM，用于存放对实时性有严格要求的指令和数据。CPU 的取指请求和数据请求均可访问 I-TCM/D-TCM。TCM 系统还支持硬件一组 AXI 64 位的 slave 接口，用于 DMA 等其他系统主机访问 TCM 进行数据交互。当出现 CPU 取指访问、CPU 数据访问以及 TCM slave 访问冲突时，TCM 的访问优先级为 CPU 数据访问 >CPU 取指访问 >TCM slave 访问。此外，D-TCM 可分为两组可同时访问的 bank0 和 bank1，能够根据需要按照地址次高位或者低位 interleave 的方式划分 bank0 和 bank1。这种灵活的分组方式，能够支持 CPU 和 DMA 分别访问其中一块 bank 而不产生冲突，从而提高 TCM 的使用效率。

软件可通过读写 TCM 相关控制寄存器 CR<22,1> 和 CR<23,1> 来使能 I-TCM/D-TCM 以及配置其地址空间，查询 TCM 的尺寸和访问延时。此外，控制寄存器 CR<22,1> 和 CR<23,1> 还分别包含一位

TCM slave 访问使能位 SIF 来允许或者禁止 TCM slave 向相应的 I-TCM/D-TCM 发起访问。若相应的使能位关闭，则任何来自 TCM slave 的访问请求都将返回访问 error。

2.2.4 外设接口介绍

R807 支持硬件可配的 PHP 外设总线接口，将特定的外设和 CPU 更紧密的耦合，用于支持外设的高速访问，减少访问延时。PHP 外设接口分别支持一组 AHB 协议和一组 AXI 协议的外设总线，也可选配其中一种。外设总线上的数据固定为不可缓存。

2.2.5 ECC 校验功能

此外，高速缓存和核内紧耦合内存（TCM）均支持硬件可配的 ECC 校验，适用于对可靠性有需求的场景。ECC 校验支持 SECDED（一比特纠错，两比特检错），并提供 ECC 错误注入功能。另外，为了方便调试，提供了额外的高速缓存和 TCM 内容读取接口，可用于调试等场景。读取功能通过控制寄存器实现。

软件可通过写控制寄存器 CR<31,0>[7:6] 来使能 ECC 校验。需要注意的是，未开启 ECC 时，CPU 不保证 ECC 编码的正确性。这就要求 ECC 及 CACHE 或者 TCM 推荐同时开关，如果采用先开启 CACHE 再开启 ECC 的操作，有几率产生不可预知的 ECC 错误。

任何小于校验粒度的写 RAM 行为都将直接写入 RAM，而对应的 ECC 编码并未更新，导致编码出错。而在开启 ECC 后，对于这种小于检验粒度的写 RAM 行为，CPU 将会采取 read-merge-write 的方式，即将整个粒度大小的值全部读出，与部分写的数据进行融合拼接，再重新编码，写入 RAM，保证 ECC 编码的正确性。这种 read-merge-write 的操作不可避免地会降低 CPU 的性能，因此在 ECC 开启时，若需要保证性能，则不推荐频繁使用部分写操作。(由于 I-TCM 的校验粒度是 64-bit，LSU 发起的 word 写请求也会被视为部分写)。

表 2.3: R807 ECC 校验粒度

RAM	校验方式	校验粒度
I-cache tag	6 比特 ECC 校验	20 位 tag+1 位有效位
I-cache data	8 比特 ECC 校验	64 位数据
D-cache tag	6 比特 ECC 校验	21 位 tag+1 位有效位
D-cache data	7 比特 ECC 校验	32 位数据
D-Cache dirty	1 比特 parity 校验	1 位脏位
I-TCM	8 比特 ECC 校验	64 位数据
D-TCM	7 比特 ECC 校验	32 位数据

2.2.6 Lock Step 功能

R807 Lock Step 是面向汽车电子等对可靠性有较高需求的应用所增加的可配功能。当硬件配置了该功能时，CPU 除包含正常的功能核外，还会包含一个与功能核逻辑完全一致的冗余核，两者共用相同的 Cache 和 TCM RAM。在 Lock Step 位使能时，冗余核将会接收和功能核相同的输入激励从而执行相同的行为，CPU

将会对两核的输出进行比较，确认两核的输出是否一致。配合已有的 ECC 校验功能，R807 能够检测到 CPU 内核因为物理原因所产生的大部分错误。

R807 Lock Step 功能的主要特点如下：

- 冗余核与功能核的内核逻辑完全一致；
- 冗余核与功能核共用 Cache 及 TCM 等 RAM 模块（RAM 通过 ECC 保护）；
- Lock Step 比较两核的输出信号结果是否完全一致（调试信号不进行比较）；
- 支持硬件错误注入。

Lock-Step 的主要功能如下图所示。输入信号 pad_core_lock_step_cmp_enable 用于使能 Lock-Step 功能，该信号需要在 CPU reset 完成前置起，确保两核在相同的初始化状态进行比较。当该信号开启时，冗余核将和功能核同步工作，比较逻辑将对两核的信号进行比较。当发现两核输出信号不一致时，输出信号 core_pad_lock_step_cmp_fault 将被置高，指示 Lock-Step 比较出错。

此外，为了方便集成时的 Lock-Step 调试，R807 还支持了 Lock-Step 错误注入功能，该功能通过输入信号 pad_core_lock_step_fault_inject 实现。当该信号置起时，将会干扰功能核的正常信号，从而导致输出出错，造成比较失败。

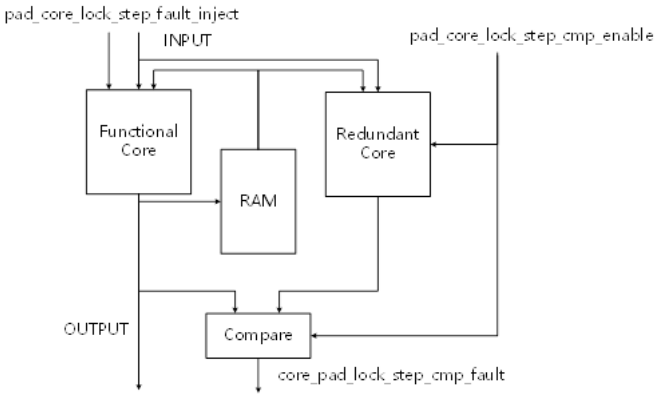


图 2.2: Lock Step 功能

注解： 硬件上若配置 Lock Step 功能，则必须同时配置 ECC 功能；

注解： 冗余核的输出仅做比较使用，不会影响 CPU 的任何行为；

2.3 处理器接口

R807 系列处理器支持多种总线接口，包括 AHB 和 AXI 总线接口协议。为了降低 R807 集成用户的硬件集成复杂度，R807 除了总线接口部分的信号外，其他信号在 R807 的多款系列处理器中，均保持了稳定

的延续性，以方便 R807 的升级换代。

图表 图 2.3 给出了 R807 系列处理器的接口信号分类图。根据 R807 顶层端口信号的特点，划分为总线接口信号，全局状态和控制信号，复位控制信号，JTAG 支持信号，中断和低功耗握手信号，调制支持信号，R807 状态信号和时钟信号，测试支持信号以及内建自测试信号等九类信号。

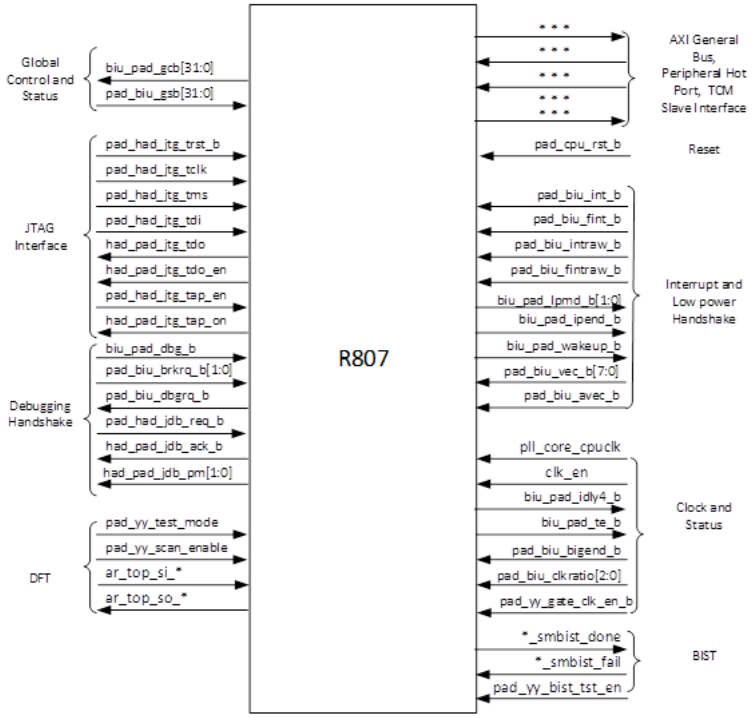


图 2.3: R807 系列 CPU 系统接口总体描述

2.3.1 接口信号详细功能描述

时钟信号：

信号名	方向	复位	时钟	功能描述
pll_core_cpuclk	I	•	•	内核工作时钟信号: 提供 CPU 内核工作的时钟。
clk_en	I	•	•	CP U 同步时钟信号: 当为 1’b1 时, pll_core_ cpuclk 可以采样 s ystem 域的信号。
pad_biu_clkratio[2:0]	I	•	CPU	cpu 时钟与总线时钟比例信号: CPU 时钟频率/总线时钟频率 = p ad_biu_clkratio + 1: 3’b000: 1 比 1; 3’b001: 2 比 1; 3’b010: 3 比 1; 3’b011: 4 比 1; 3’b100: 5 比 1; 3’b101: 6 比 1; 3’b110: 7 比 1; 3’b111: 8 比 1。

复位控制信号:

信号名	方向	复位	时钟	功能描述
pad_cpu_rst_b	I	•	SYS	处理器复位信号: 低电平时, 初始化 R807 内部及调试模块, R807 采用异步复位方式。

全局状态和控制信号:

表 2.4: 全局状态和控制信号

信号名	方向	复位	时钟	功能描述
bi u_pad_gcb[31:0]	O	32’ b0	SYS	全局控制信号: 全局控制总线输出，用于控制外围设备和事件，它对应于 GCR 寄存器的 32 位，因此可以通过 MTCR 修改 GCR 寄存器的值，达到控制外围设备的目的。当执行指令 MTCR 对全局控制寄存器进行更新时 biu_pad_gcb[31 :0] 信号会改变。
pa d_biu_gsb[31:0]	I		SYS	全局状态信号: 全局状态总线输入，用于检测外围设备和事件，它对应于 GSR 寄存器的 32 位，但它只读。当执行指令 MFCR 将全局状态寄存器的值采样到一个通用寄存器中时，处理器对信号 pad_biu_gsb[31:0] 进行采样。

JTAG 支持信号:

表 2.5: JTAG 支持信号

信号名	方向	复位	时钟	功能描述
pad _had_jtg_trst_b	I	•	JTAG	JTA G 测试复位信号: JTA G 复位信号，下跳变可以初始化和 JTAG 接口相关的触发器，正常工作情况下该信号需置成高电平。
p ad_had_jtg_tclk	I	•	•	JTA G 测试时钟信号: JTAG 和 HA D 内部相关触发器的时钟信号,要求和 cpuclk 的频率比在 1:8 以上。
pad_had_jtg_tms	I	•	JTAG	JTAG 测试模式选择信号: JTAG TAP 有限状态机状态跳转控制信号，该位可以控制命令的输入，数据的输入输出。
pad_had_jtg_tdi	I	•	JTAG	JTA G 数据输入信号: JTA G 串行输入端口，包括控制命令，数据，输入顺序由低位到高位。

下页继续

表 2.5 – 续上页

信号名	方向	复位	时钟	功能描述
had_pad_jtg_tdo	O	•	JTAG	JTA G 数据输出信号: J TAG 串行数据输出端口，输出顺序由低位到高位。
had _pad_jtg_tdo_en	O	1’ b0	JTAG	TD O 输出使能信号: 用于支持多个并行的 JTAG 控制器，当 tap 控制器处在 shift-dr 或 shift-ir 状态，并且 had_pad_jtg _tap_on 有效时，这个信号有效。
pad _had_jtg_tap_en	I	•	JTAG	JTAG tap 控制器使能信号: 用于使能 JTAG tap 控制器，另外信号 pad_had_jdb_re q_b 也可以达到同样的使能效果。
had _pad_jtg_tap_on	O	•	JTAG	表明 tap 控制器状态指示信号: 用于指示 ta p 控制器是否工作，高电平有效。

中断和低功耗握手信号:

表 2.6: 中断和低功耗握手信号

信号名	方向	复位	时钟	功能描述
pad_biu_int_b	I	•	SYS	中断请求信号: 低电平时表示进行普通中断申请。
pad_biu_fint_b	I	•	SYS	快速中断请求信号: 低电平时表示进行快速中断申请。
p ad_biu_intraw_b	I	•	SYS	异步唤醒信号: 不请求进入中断，用于将 CPU 从低功耗模式中唤醒。这个信号会使 b iu_pad_wakeup_b 和 biu_pa d_ipend_b 有效。
pa d_biu_fintraw_b	I	•	SYS	快速异步唤醒信号: 不请求进入中断，用于将 CPU 从低功耗模式中唤醒。这个信号会使 b iu_pad_wakeup_b 和 biu_pa d_ipend_b 有效。

下页继续

表 2.6 – 续上页

信号名	方向	复位	时钟	功能描述
biu_pad_ipend_b	O	•	SYS	异步唤醒确认信号: 指示异步唤醒请求，已经被处理器确认。
pad _biu_vec_b[7:0]	I	•	SYS	中断矢量序号信号: 提供 cpu 进行中断处理的向量号，如果 pad_biu_ave c_b 为低电平，这个向量被忽略。
pad_biu_avec_b	I	•	SYS	中断向量有效控制信号: 如果为低电平，则普通中断和快速中断的向量入口取默认值；否则，普通中断和快速中断的向量入口取外部的输入（pad_biu _vec_b[7:0]）。
biu_ pad_lpmd_b[1:0]	O	2’ b11	SYS	低功耗模式状态信号: 当处理器执行 doze, stop 或 wait 指令时，bi u_pad_lpmd_b[1: 0] 被相应的改变: 2’ b00：STOP； 2’ b01：WAIT； 2’ b10：DOZE； 2’ b11：normal。
b iu_pad_wakeup_b	O	•	SYS	处理器唤醒指示信号: 低电平表示 CPU 被唤醒且进入正常工作模式，这个信号将会在 pad_biu _intraw_b, pad_ biu_fintraw_b, 或 pad_ had_jdb_req_b 设置之后被设置。

状态信号:

信号名	方向	复位	时钟	功能描述
biu_pad_idly4_b	O	1’b1	SYS	idly 4 信号指示信号: 指示一个 idly4 指令序列。当处理器执行了 idly4 这条指令之后，这个信号被清零，直到 IDLY4 指令序列被执行完才释放。这个信号与系统时钟同步，这就意味着当 CLK RATIO 不等于 1(时钟比为 1:1) 时，这个信号就不一一对应于 CPU 指令执行的情况，用户应根据具体情况深刻理解这信号与内部指令执行的对应关系。
p ad_biu_bigend_b	I	•	CPU	指令解析模式指示信号: 低电平时表示取回来的数据或指令是 Big Endian 的字节顺序；这个信号为高电平时，表示取回来的数据或指令是 Little Endian 的字节顺序。该信号只有在 CPU 复位的时候才可以设置，一旦 CPU 退出复位状态，这个信号就保持静态输入，不能发生变化。
biu_pad_te_b	O	1’b1	SYS	外部传输控制信号: 指示所要读取的地址应该经过一个外部的 MPU 转译。其对应于 PSR(处理器状态寄存器) 的 TE 位，可以通过对 PSR 的修改

调试支持信号：

表 2.7: 调试支持信号

信号名	方向	复位	时钟	功能描述
pad_biu_dbgrq_b	I	•	SYS	外部调试请求信号： 该信号是外部调试请求信号，低电平有效，同步于 sys clk，使能处理器进入调试模式。 不用该信号时，需要接 1。
biu_pad_dbg_b	O	•	SYS	表明处理器进入调试模式： 该位信号指示处理器是否处于调试模式，该信号低电平有效，同步于 sysclk。
pa d_had_jdb_req_b	I	•	•	外部调试请求信号： 该信号是外部调试请求信号，异步于 tclk，低电平有效。所以在 CPU 处于低功耗状态或者复位状态时，该信号有效可以快速请求 CP U 进入调试模式。 不用该信号时，需要接 1。
ha d_pad_jdb_ack_b	O	1’b1	JTAG	处理器响应进入调试模式： 处理器进入调试模式后，该响应信号保持两个 tclk 的低电平。
had_ pad_jdb_pm[1:0]	O	2’b0	CPU	处理器工作模式指示信号： 这些输出信号表明处理器的工作模式，异步于 pad _had_jtg_tclk。 00: normal 模式； 01: STOP/DOZE/WAIT 模式； 10：调试模式； 11：保留。

门控时钟信号：

信号名	方向	复位	时钟	功能描述
pad_yy_gate_clk_en_b	I	•	CPU	门控时钟使能信号： 0：门控时钟有效； 1：门控时钟无效。

2.4 时钟和复位

2.4.1 时钟信号

处理器使用的时钟信号：

信号名	方向	复位	时钟	功能描述
pll_core_cpuclk	I	•	•	内核工作时钟信号： 提供给处理器的工作时钟。
p ad_had_jtg_tclk	I	•	•	JTAG、调试时钟信号： JTAG 和 HAD 内部相关触发器的时钟信号，要求和 cpuclk 的频率比在 1：2 以上。

R807 内部不产生时钟信号，内核工作所需的时钟由 SoC 的时钟产生模块产生，JTAG 时钟一般由调试器提供。如下图所示，pll_core_cpuclk 和 pad_had_jtg_tclk 信号作为 CPU 输入信号直接连接到处理器顶层；sysclk 为系统时钟，处理器不会直接使用此时钟信号；clk_en 信号为处理系统时钟与处理器时钟在非 1:1 情况下的同步逻辑所使用，具体功能与行为详见后面章节。

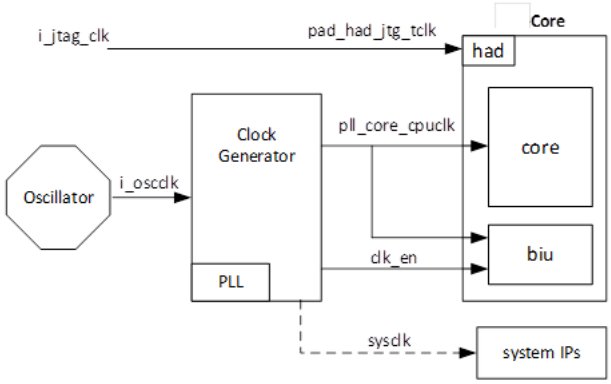


图 2.4: R807 的时钟管理方式

图表 图 2.4 的时钟管理示例，全局的系统时钟信号（i_oscclk）是从板上输入的系统时钟，为 SoC 提供时钟源，时钟产生逻辑为 CPU 产生 cpuclk、clk_en 与 sysclk，sysclk 提供给系统总线、外围的 IP 使用。jtgclk 是调试时钟信号，与 jtag 的其他输入信号（TDI，TMS，TRST）同步，该信号是 SoC 系统的顶层信号，需要从芯片引脚接入。CPU 核内（除调试相关逻辑）仅有 cpuclk 进行驱动。总线接口（BIU）模块内与 CPU 内核接口部分由 cpuclk 驱动。BIU 接口部分逻辑也由 cpuclk 驱动，但在 BIU 内部设计有同步控制信号，保证总线接口上的输入输出都在 sysclk 的上升沿发生变化和采样，由 cpuclk_en 信号控制。

为了降低功耗 R807 中广泛的使用了 clock gating 技术，顶层模块和底层子模块均采用了门控。局部的 clock gating 通过 clock gating 单元实现，全局的 clock gating 在 cp0 单元的控制之下。外部晶振产生的时钟

由 PLL(锁相环) 锁相倍频后，经过选择输出 pll_core_cpuclk 这个时钟，这个时钟经过门控单元后得到 cpu 内部各个模块使用的时钟信号。

2.4.1.1 CPU 内核时钟域与 JTAG 时钟域

在 R807 的设计中，CPU 内核时钟 cpuclk 与 HAD 内部的 jtagclk 之间有信号交互。两个信号属于异步信号，用户无需关系他们之间的相位。CPU 内部已经通过同步逻辑单元将两个之间的信号进行了有效同步。其逻辑框图如图表 图 2.5 所示。

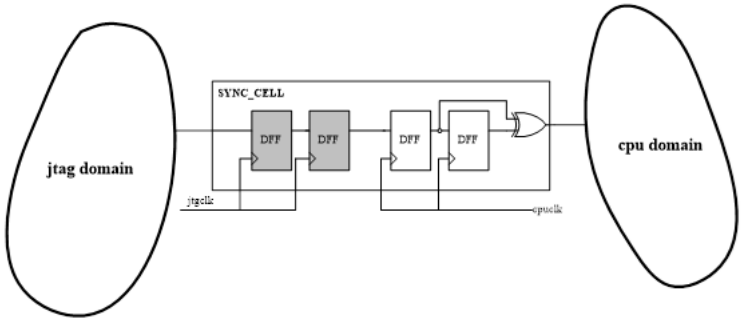


图 2.5: CPU 与 JTAG 时钟域之间的同步逻辑

2.4.1.2 CPU 内核时钟域与系统总线时钟域

在 R807 的设计中，CPU 内核时钟域与系统总线时钟域在总线接口部分逻辑中有信号交互。这两个时钟要求在物理设计中，需要保证相位对齐。由于这两个信号只有频率整数倍关系，所以无需进行专用的逻辑进行同步。

在 R807 的设计中，CPU 顶层提供了一个时钟使能信号（clk_en），该信号在 CPU 内部做了打拍处理，使用打拍后的信号保证 CPU 内部到总线上的所有信号都按照系统时钟的沿对齐。因此从 CPU 总线到系统的信号都具有 sysclk 的全周期时序延时。在集成时，要求顶层 clk_en 提前一个 CPU clock 提供。图表 图 2.6 给出了 CPU 与系统时钟比例在 1:1 到 8:1 的情况下，顶层 clk_en 输入信号的时序图。

2.4.2 复位信号

处理器包含的复位信号：

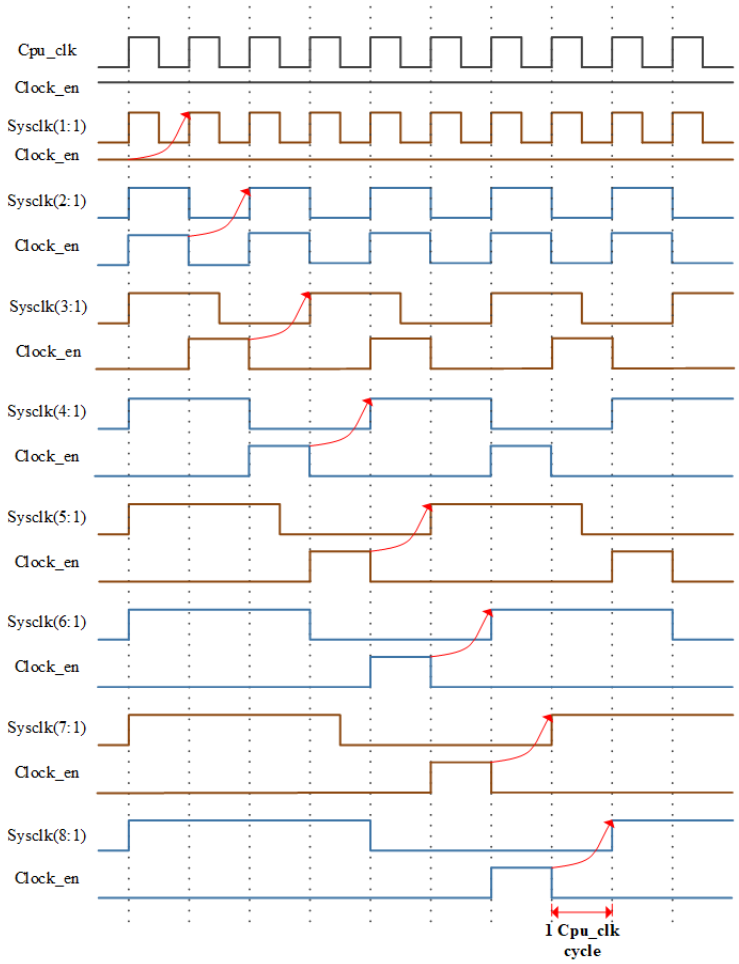


图 2.6: CPU 与系统时钟的接口信号（1:1 到 8:1）

信号名	方向	复位	时钟	功能描述
pad_cpu_rst_b	I	•	SYS	处理器复位信号： 低电平时，初始化 R807 内部及调试模块，R807 采用异步复位方式。
pad _had_jtg_trst_b	I	•	JTAG	调试、测试复位信号： 低电平可复位调试相关寄存器，正常工作情况下该信号需置成高电平。 测试模式下，此信号作为全局复位信号。

R807 采用异步复位，同步释放的方式进行系统的复位。这种复位方式可以有效的避免复位过程中亚稳态的出现。图表 2-10 显示了 R807 的复位信号产生机制。

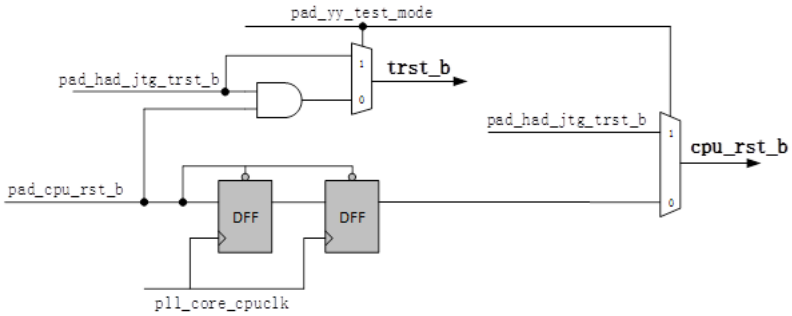


图 2.7: R807 复位信号产生机制

如图表 图 2.7 所示，R807 的内核复位信号采用两级复位系统。首先，在 CPU 时钟 cpuclk 的驱动下同步异步复位信号，可在复位信号释放时消除亚稳态。在测试模式下（pad_yy_test_mode=1），用于同步的寄存器都被旁路，复位信号选择 pad_had_jtg_trst_b。

对于调试辅助单元（HAD）部分的复位信号，在正常功能下，pad_had_jtg_trst_b 和 pad_cpu_rst_b 两个复位信号都会将调试单元（HAD）内部的逻辑复位。在测试模式下，pad_had_jtg_trst_b 为有效复位信号。

2.5 初始化设计

当处理器开始运行应用程序之前，有一个初始化的过程，包括功能部件的配置。这个章节将简要介绍处理器复位后的初始化步骤。

大多数处理器的架构寄存器，包括通用寄存器、部分控制寄存器以及浮点寄存器，都由不带复位端的寄存器组成，这意味着这些寄存器在上电后为一个随机值。因此，必须在使用前初始化这些寄存器，使用 MOV、MTCR 等指令。

进一步地，在运行应用程序之前，可能的操作还有：

- 特殊寄存器的初始化，比如：SP；
- 使能特定的功能，比如：ECC；
- 初始化特定的内存，比如：TCM。

1. MPU 初始化

在使能 MPU 前的初始化步骤请遵照：

- 配置区域范围。
- 配置区域属性。
- 使能 MPU。

具体操作请参考文档后续 MPU 相关章节。

特别需要注意的是，MPU 在使能之前，至少要保证至少一个区域配置完成并且是可读属性，并且保证当前程序段位于以上所述的区域之内。

2. Cache 初始化

开启 CACHE 前的初始化步骤请遵照：

- 配置区域属性并开启 MPU（参考 MPU 相关章节）；
- Invalid all cache；
- 使能 cache；

具体操作请参考文档后续 cache 相关章节。

特别需要注意的是，在开启 cache 前，如果跳过 cache invalid 步骤，将可能会发生不可预知的错误，不管是在仿真还是运行都会遇到问题。

第三章　编程模型

3.1　工作模式及寄存器视图

R807 定义了两种工作模式：超级用户模式和普通用户模式。两种工作模式由处理器状态寄存器（PSR）的 S 位（PSR[31]）控制。当 PSR 的 S 位被置位时，处理器工作在超级用户模式，S 位被清零时，处理器工作在普通用户模式。在普通用户模式下，处理器使用普通用户编程模型。在异常处理时，处理器把工作模式切换到超级用户模式，具体为将当前 PSR 的值存放在异常保留处理器状态寄存器（EPSR）中，然后在 PSR 中设置 S 位为 1，强制处理器进入超级用户模式。在异常处理完毕后，为返回到以前的工作模式，系统函数执行 RTE 指令 (从异常返回) 或 RFI 指令（从快速中断返回），从异常或中断发生的地方重新取指令执行。作为系统调用指令，TRAP #n 指令为工作在普通用户模式下的应用程序提供了访问操作系统服务程序的可控制接口。普通用户程序可通过 TRAP #n 产生系统调用异常，并强制处理器进入超级用户模式。

两种工作模式分别对应不同的操作权限，主要体现在以下几个方面：

1. **对寄存器的访问**

普通用户模式下的应用程序只允许访问那些指定给普通用户模式的寄存器；工作在超级用户模式下的系统软件则可以访问所有的寄存器，使用控制寄存器来进行超级用户操作。通过对寄存器访问权限的管理，可以避免普通用户程序接触特权信息。工作在超级用户模式下的操作系统则通过协调与普通用户程序的行为来为普通用户程序提供管理和服务。

图 3.1 所示为 R807 在两种工作模式下的可操作寄存器视图：

普通用户模式可以操作的寄存器包括 32 个 32 位的通用寄存器、32 位程序计数器（PC）、条件/进位位（C）以及其它寄存器（由配置决定）。其中，C 位处于 PSR 的最低位，是 PSR 中唯一能被在普通用户模式下被访问的数据位。除了普通用户模式可以访问的寄存器外，超级用户模式还包括含有处理器控制和状态信息的 PSR 寄存器，一套用来在异常发生时保存 PSR、PC 的异常影子寄存器 EPSR、EPC，一套用来节省在快速中断中上下文切换时间的快速中断影子寄存器 FPSR、FPC，一个保存中断向量表的基地址的寄存器 VBR 以及其他相关控制寄存器（由配置决定）。具体的控制寄存器信息可参考*系统控制寄存器*。

R807 额外配置有一组可选择通用寄存器 AGPR：R15’ ~R0’。通过 PSR 中的 AF 位来选择访问 GPR 或者 AGPR。当 PSR 中的 AF 位被置为 1 时，处理器从可选择通用寄存器组中取数进行运算；当 AF 位被清零时，处理器从通用寄存器组中取数据进行运算。可选择通用寄存器组可用来减少在实时事件处理时的上下文切换时间，该组寄存器仅在超级用户模式下被访问到。

2. **特权指令的使用**

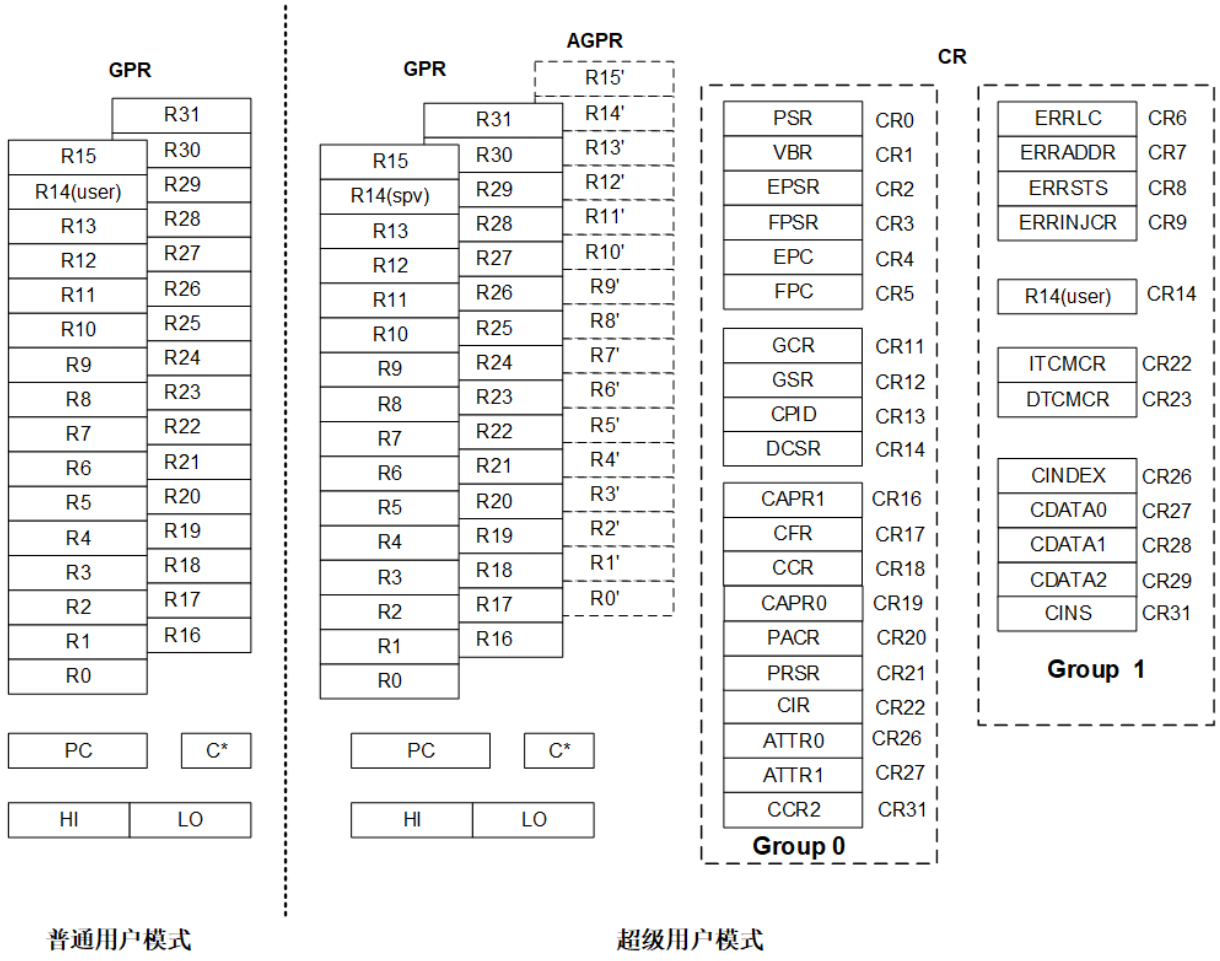


图 3.1: 寄存器视图

大多数指令在两种模式下都能执行，但是一些对系统产生重大影响的特权指令只能工作在超级用户模式下。特权指令包括 STOP，DOZE，WAIT，MFCR，MTCR，PSRSET，PSRCLR，RTE，RFI。

3. **对内存空间的访问**

R807 设计了内存保护单元（MPU），操作系统可通过对内存保护单元的设置实现对内存的访问进行管理与控制。程序根据其权限来访问内存空间，普通用户程序只允许访问那些对普通用户模式开放的内存空间。

3.2 通用寄存器

R807 实现了 32 个 32 位的通用寄存器，R31~R0，如图表 表 3.1 所示。

表 3.1: 通用寄存器

名称	功能
R0	不确定，函数调用时第一个参数
R1	不确定，函数调用时第二个参数
R2	不确定，函数调用时第三个参数
R3	不确定，函数调用时第四个参数
R4	不确定
R5	不确定
R6	不确定
R7	不确定
R8	不确定
R9	不确定
R10	不确定
R11	不确定
R12	不确定
R13	不确定
R14(user)	堆栈指针（普通用户编程模式）
R15	链接寄存器
R16	不确定
R17	不确定
R18	不确定
R19	不确定
R20	不确定
R21	不确定
R22	不确定
R23	不确定
R24	不确定
R25	不确定

下页继续

表 3.1 – 续上页

名称	功能
R26	不确定
R27	不确定
R28	不确定
R29	不确定
R30	不确定
R31	线程本地储存寄存器

通用寄存器用于保存指令操作数和指令执行结果，以及地址信息。软件可将这组通用寄存器用于子程序的链接调用，函数调用参数传递，临时变量保存以及堆栈指针。

R807 为普通用户模式和超级用户模式分别设计了堆栈指针 R14。普通用户模式下的应用程序只能访问普通用户模式的 R14（User SP）；在超级用户模式下，系统软件不仅可以访问超级用户模式的 R14（Spv SP），还可以访问普通用户模式的 R14（User SP）。超级用户模式下，对通用寄存器 R14 的索引将会指向超级用户模式的寄存器 R14(Spv SP)。若系统程序要在超级用户模式下访问 R14(User SP)，可通过 MFCR/MTCR 访问 CR<14,1> 完成。

3.2.1 可选择寄存器

R807 在超级用户模式下设计实现了 16 个可选择寄存器。在转向时间关键任务时可选择寄存器可用来减少因当前内容转换和保护现场引起的响应延迟时间。当 PSR（AF）为 1，则可选择寄存器被选中，所有的指令对通用寄存器 R15~R0 的引用都将使用可选择寄存器 R15’ ~R0’。反之，若 PSR（AF）为 0，则可选择寄存器不被选中。可选择寄存器可用于保存重要的参数和指针值，当处理器执行优先级较高的任务时切换选中可选择寄存器即可快速得到这些信息，省去上下文切换时间。

在异常出现并执行异常服务程序时，异常服务程序矢量入口值的低位将被拷贝到 AF 位中以用来选择在异常服务程序使用通用寄存器 R15~R0 抑或可选择寄存器 R15’ ~R0’。

通用寄存器 R31~R16 没有对应的可选择寄存器。

3.2.2 程序计数器

程序计数器（PC）包含了当前执行指令的地址。在指令执行和异常处理期间，处理器会根据程序运行的情况自动的调整程序计数器的值或放置一新值到程序计数器中。对一些指令来说，程序计数器能被用来作为相对地址计算。此外，程序计数器中的最低位一直为零。

3.2.3 条件码 / 进位标志位

条件码 / 进位标志位（C 位）代表了一次指令执行后的结果。条件码 / 进位标志位能够作为比较操作指令的结果清楚地被设置，或者作为另一些高精度算术或逻辑指令的结果而不确定地被设置。另外，特殊的指令如 DEC[GT,LT,NE]，以及 XTRB[0-3] 等将会影响条件码 / 进位标志位的值。条件码/进位标志位位于 PSR[0]，允许被普通用户模式和超级用户模式下的程序访问使用。

3.3 系统控制寄存器

系统编程人员采用超级用户模式的编程模型来设置系统操作功能，I/O 控制，以及其他受限的操作。超级用户模式下可访问处理器的所有寄存器，其中系统控制寄存器如图表 表 3.2 所示。

表 3.2: 系统控制寄存器

名称	属性	控制寄存器编号/地址	描述
PSR	读/写	CR<0,0>	处理器状态寄存器
VBR	读/写	CR<1,0>	向量基址寄存器
EPSR	读/写	CR<2,0>	异常保留处理器状态寄存器
FPSR	读/写	CR<3,0>	快速中断保留处理器状态寄存器
EPC	读/写	CR<4,0>	异常保留程序计数器
FPC	读/写	CR<5,0>	快速中断保留程序计数器
GCR	读/写	CR<11,0>	全局控制寄存器
GSR	只读	CR<12,0>	全局状态寄存器
CPIDR	只读	CR<13,0>	产品序号寄存器
DCSR	读/写	CR<14,0>	计算模式状态寄存器
CFR***	读/写	CR<17,0>	高速缓存功能设置寄存器
CCR***	读/写	CR<18,0>	高速缓存配置寄存器
CAPR0*	读/写	CR<19,0>	MPU 访问权限配置寄存器 0
CAPR1*	读/写	CR<16,0>	MPU 访问权限配置寄存器 1
PACR*	读/写	CR<20,0>	保护区控制寄存器
PRSR*	读/写	CR<21,0>	保护区选择寄存器
CIR*	读/写	CR<22,0>	高速缓存索引寄存器
ATTR0*	读/写	CR<26,0>	物理内存属性配置寄存器 0
ATTR1*	读/写	CR<27,0>	物理内存属性配置寄存器 1
CCR2	读/写	CR<31,0>	隐式操作寄存器
ERRLC**	读/写	CR<6,1>	RAM 错误信息寄存器
ERRADDR**	读/写	CR<7,1>	RAM 错误地址寄存器
ERRSTS**	读/写	CR<8,1>	RAM 异常状态寄存器
ERRINJCR**	读/写	CR<9,1>	ECC 注入功能寄存器
I-TCMCR**	读/写	CR<22,1>	I-TCM 配置寄存器
D-TCMCR**	读/写	CR<23,1>	D-TCM 配置寄存器
CINDEX***	只读	CR<26,1>	cache 访问索引寄存器
CDATA0***	只读	CR<27,1>	cache 访问数据寄存器 0
CDATA1***	只读	CR<28,1>	cache 访问数据寄存器 1
CDATA2***	只读	CR<29,1>	cache 访问数据寄存器 2

下页继续

表 3.2 – 续上页

名称	属性	控制寄存器编号/地址	描述
CINS***	只写	CR<31,1>	cache 访问指令寄存器

标记 * 的控制寄存器描述具体请参考控制寄存器，标记 ** 的控制寄存器描述具体请参考 *Cache 系统*，标记 *** 的控制寄存器描述具体请参考 *TCM 系统*。

3.3.1 处理器状态寄存器（PSR，CR<0,0>）

处理器状态寄存器（PSR）存储了当前处理器的状态和控制信息，包括 C 位，中断使能位和其他控制位。相关控制位为处理器指示出以下状态：跟踪模式（TM [1:0]），超级用户模式或者普通用户模式（S 位），通用寄存器或者可选择寄存器（AF 位），以及中断申请是否有效等。在超级用户编程模式下，软件可以访问处理器状态寄存器（PSR）。

	31	30 24	23 16
	S	Reserved	VEC[7：0]
Reset	1	0	0

	15 14	13	12	11 10	9	8	7	6	5	4	3 2	1	0
	TM[1:0]	0	TE	0	MM	EE	IC	IE	0	FE	0	AF*	C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0

图 3.2: 处理器状态寄存器

S-超级用户模式设置位：

当 S 为 0 时，处理器工作在普通用户模式；

当 S 为 1 时，处理器工作在超级用户模式；

S 位在被 reset 和进入异常处理时由硬件置 1。

VEC[7:0]-异常事件向量值：

当异常出现时，这些位被用来计算异常服务程序向量入口地址，且会在被 reset 时清零。

TM[1:0]-跟踪模式位：

在指令跟踪模式下，每一条指令执行完后，R807 都将会进入跟踪异常服务程序；在跳转跟踪模式下，当碰到内含有跳转（不管是跳转还是不跳转）的指令执行完，R807 都将会进入跟踪异常服务程序。这些位在被 reset 清零和进入异常服务程序时由硬件清零。图表 表 3.2 示出 TM[1:0] 编码与相对应的工作模式。

表 3.3: TM[1:0] 编码与相对应的工作模式

值	描述
00	正常执行模式
01	指令跟踪模式
10	未定义
11	跳转跟踪模式

跟踪模式具体的操作请参考异常处理。

TE-传输控制位：

该位在芯片的管脚上有对应的管脚信号 biu_pad_te_b，可以通过设置该位来控制传输，它会被 reset 清零，也称为 TC，可由用户自行定义。

MM-不对齐异常掩盖位：

当 MM 为 0 时，读取或存储的地址不对齐异常将正常发生，不会被掩盖即异常会被响应；

当 MM 为 1 时，读取或存储的地址不对齐异常将会被掩盖，加载存储指令将以非对齐方式访问存储器。

该位不能掩盖 jmpi 或 jsri 指令的不对齐异常的发生，且不受其它异常影响，但会被 reset 清零。

EE-异常有效控制位：

当 EE 为 0 时，异常无效，此时除了普通中断和快速中断之外的任何异常一旦发生，都会被 R807 认为是不可恢复的异常；

当 EE 为 1 时，异常有效，所有的异常都会正常的响应并使用 EPSR 与 EPC 寄存器保存处理器现场状态。

IC-中断控制位：

当 IC 为 0 时，中断只能在指令之间被响应；

当 IC 为 1 时，表明中断可在长时间、多周期的指令执行完之前被响应；

会被 reset 清零，不受其它异常影响。

IE-中断有效控制位：

当 IE 为 0 时，中断无效；

当 IE 为 1 时，中断使能，当 EE 位也有效时，普通中断可以得到响应；

该位会被 reset 清零，也在进入异常服务程序时被清零。

FE-快速中断有效控制位：

当 FE 为 0 时，快速中断无效；

当 FE 为 1 时，快速中断使能，不必考虑 EE 位。采用 FPSR 和 FPC 寄存器保存处理器现场状态；

该位会被 reset 清零，也在进入快速中断服务程序时被清零，但不受其它异常的影响。

AF-可选择寄存器有效控制位：

当 AF 为 0 时，通用寄存器被选中，可选择寄存器不被选中；

当 AF 为 1 时，通用寄存器不被选中，可选择寄存器被选中；

当异常发生时，异常入口地址的最低位被拷贝到该位用来选择哪一组寄存器以便在异常服务程序中使用。此位被 reset 清零，但同时它也被重启异常向量入口地址的低位所设置。

C-条件码 / 进位标志位

该位用作条件判断位为一些指令服务。

3.3.1.1 更新 PSR

PSR 可以通过几种不同的方式被更新，对 PSR 中控制位的更改所产生的影响也多种多样。PSR 通常可以通过异常响应，异常处理和执行 PSRSET，PSRCLR，RTE，RFI，MTCR 指令被修改，这些修改的实现有四个方面。

- 异常响应和异常处理更新 PSR：

更新 PSR 是异常响应和异常服务程序入口地址计算中的一部分，它将更新 PSR 中 S，TM，VEC，IE，FE，EE 以及 AF 位。对 S，TM，VEC，IE，FE 以及 EE 位的改动优先于异常服务程序向量入口地址的读取。对 AF 位的改动优先于异常服务程序中的第一条指令的执行。

- RTE 和 RFI 指令更新 PSR：

更新 PSR 作为 RTE 和 RFI 指令执行的一部分，会对 PSR 中的所有位都改动。其中对 S，TM，TP，IE，FE 和 EE 的改动优先于对返回 PC 的获取，对 VEC,MM, IC, AF 和 C 位的改动优先于程序返回后第一条指令的执行。

- MTCR 指令更新 PSR：

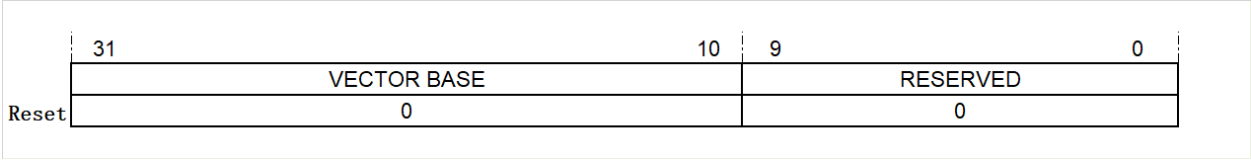
若目标寄存器是 CR<0,0> 的话，更新 PSR 将会作为 MTCR 指令执行的一部分。这种更新将可能会改变 PSR 中所有位的值，紧接着的指令、异常事件和中断响应将会采用新的 PSR 值。

- PSRCLR、PSRSET 指令更新 PSR：

更新 PSR 作为 PSRCLR 和 PSRSET 指令执行的一部分，紧接着的指令、异常事件和中断响应将会采用新的 PSR 值。

3.3.2 向量基址寄存器（VBR，CR<1,0>）

VBR 寄存器用来保存异常向量的基址。该寄存器包含 22 个高位有效位，10 个保留位（其值为 0）。VBR 的复位为输入信号 pad_cpu_rvba[31:0] 数值。



	31 — 10	9 — 0
	VECTOR BASE	RESERVED
Reset	0	0

图 3.3: 向量基址寄存器

3.3.3 异常保留寄存器（CR<2,0> ~ CR<5,0>）

EPSR, EPC,FPSR 和 FPC 这些寄存器在遇到异常情况时被用来保存当前处理器执行的内容。更详细的信息请参考异常处理。

3.3.4 全局控制寄存器（GCR，CR<11,0>）

12 位的全局控制寄存器用于控制外部设备和事件。它通过芯片引脚上提供的 12 位平行输出接口实现指定控制。一般来说，可以通过简单设置 GCR 来管理功耗，设备控制，事件安排处理以及其它基本的功能。至于 GCR 中每一位对应的控制功能，用户可以根据情况自行定义。全局控制寄存器是可读可写的。

3.3.5 全局状态寄存器（GSR，CR<12,0>）

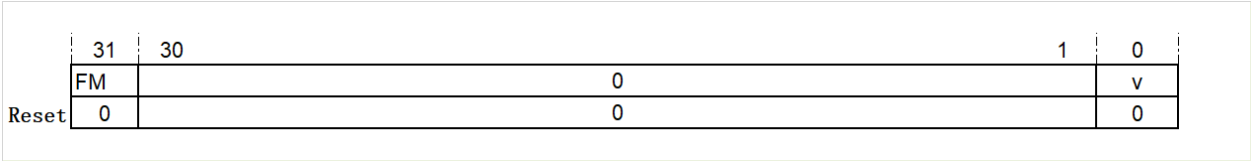
12 位的全局状态寄存器是用来标记外围设备和事件的。它通过芯片口上提供的 12 位平行输入接口将外部状态送入到 R807 内部，从而实现监测。一般来说，可以通过查看 GSR 来检测外围设备状态和事件。全局状态寄存器是只读的。

3.3.6 产品序号寄存器（CPIDR，CR<13,0>）

该寄存器用于存放杭州中天微系统有限公司产品的内部编号。产品序号寄存器是只读的，其值由产品本身决定。

3.3.7 计算模式状态寄存器（DCSR，CR<14,0>）

R807 提供一个额外的控制状态寄存器来控制 R807 所支持的 DSP 指令的执行，同时用来表征这些 DSP 指令执行结果的是否出现溢出。



	31	30 — 1	0
	FM	0	v
Reset	0	0	0

图 3.4: DSP 控制状态寄存器

FM-小数运算模式：

当 FM 位被设置成 1 时，所有的乘法运算被认为是小数运算。

V-溢出状态位：

用来表征最近的一次乘加或者乘减操作是否溢出。这一位为只读位。通过使用 MVTC 指令可以将这一比特位拷贝到 PSR 的状态位（C 位）。

3.3.8 多核编号寄存器（MPID，CR<30,0>）

CORE_ID- CORE 编号：指示当前 CORE 的编号。当 CORE_ID 为 0 时，表示 CORE0；当 CORE_ID 为 1 时，表示 CORE1；当 CORE_ID 为 2 时，表示 CORE2；当 CORE_ID 为 3 时，表示 CORE3; 以此类推。其数值表示为输入信号 pad_cpu_coreid[7:0] 的数值。

3.3.9 Sync-CPU HINT 寄存器（CCR2，CR<31,0>）

	31	16
	Reserved	
Reset	0	

	15 9	8	7	6	5	4	3	2	1	0
	Reserved	IFR	CRECC	ECC	0	PREFETCH	AMR	LP	1'b1	RWO
Reset	0	0	0	0	0	0	0	0	0	0

图 3.5: L2 高速缓存配置寄存器

RWO-读请求 weakorder 有效设置：

该位有效时，属性为 cacheable 的读请求可以乱序投机的从内存中获取数据。

LP-短循环预测使能位：

该位有效时，处理器可对短循环程序进行监视。使能该位可实现指令访问的低功耗。

AMR-写分配模式自动调节使能位：

该位有效时，处理器可根据写操作的缓存访问情况，自动调节写操作为写分配模式，从而提升处理器的运行效率。

PREFETCH-数据预取使能位：

该位有效时，处理器可根据数据访问规律，自动预取数据，减少数据的访问延时，提升处理器的运行效率。

ECC-ECC 校验使能位：

该位有效时，RAM 进行 ECC 校验。

CRECC-可修复 ECC 错误异常使能位：

ECC 校验使能且该位有效时，可修复的 ECC 错误也会上报异常。

IFR 中断快速响应使能位：

该位有效时，CPU 支持中断快速响应，可直接打断指令执行，响应中断。

3.4 内存视图

R807 支持字节（8 位），半字（16 位），字（32 位）为单位的数据存取操作。为了更好的性能考虑，数据存取地址应以存取数据位宽做对齐操作，如下所示：

- 字的数据存取应以 4 字节为边界做地址对齐；
- 半字的数据存取应以 2 字节为边界做地址对齐；
- 字节的数据存取可以任意字节为边界做地址对齐；

同时 R807 非对齐的数据存取访问以及大小端适配。

3.4.1 数据大小端

大小端基于存储器数据存储的格式提出，高地址字节存放至物理内存的低位被定义大端；高地址字节存放至物理内存的高位被定义为小端，如图表 图 3.6 所示。R807 CPU 在字节和半字访问时有大小端之分，但字访问时无大小端之分（格式相同）。

图 3.6: 内存中的数据组织形式

R807 支持标准补码的 2 进制整数。每个指令操作数的长度可以明确地编码在程序中（load/store 指令），也可以隐含在指令操作中（index operation, byte extraction）。通常，指令接收 32 位操作数，产生 32 位结果。

R807 的存储器可以配置成大端模式或小端模式。在大端模式（缺省模式）下，字 0 的最高位字节放在地址 0 上。而在小端模式下，字 0 的最高位字节放在地址 3 上。

3.4.2 非对齐数据访问

R807 支持地址非对齐的内存数据访问，此时需要将 PSR 的 MM 位开启以使能该特性。当 MM 位关闭时，地址非对其的内存访问将会产生异常。具体请参考*未对齐访问异常（向量偏移 0X4）*。

第四章　异常处理

异常处理 (包括指令异常和外部中断) 是处理器的一项重要技术。在某些异常事件产生时，处理器会停止当前指令的执行，并转入对异常事件的处理。这些事件包括外部中断，指令执行错误和系统调用请求等。本章主要描述异常种类、异常优先级、异常向量表和异常返回等内容。

4.1　异常处理概述

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序。引起异常的外部事件包括：外部设备的中断请求、硬断点请求、读写访问错误和硬件重启；引起异常的内部事件包括：非法指令、不对齐错误、ECC 错误、特权异常和指令跟踪；此外，TRAP 和 BKPT 指令正常执行时也会产生异常。在发生异常时，异常处理利用异常向量表和一组影子寄存器跳转到异常服务程序的入口。

异常处理的关键就是在异常发生时，保存 CPU 当前指令运行的状态，在退出异常处理时恢复异常处理前的状态。异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。即使异常在取指和译码阶段被识别，异常也要在相应的指令退休时才会被处理。为了异常处理不影响 CPU 的性能，CPU 在异常处理结束后要避免重复执行以前的指令。R807 根据异常识别时的指令是否完成决定异常地址寄存器存储哪一条指令的地址。例如，如果异常事件是外部中断服务请求，被中断的指令将正常退休并改变 CPU 的状态，它的下一条指令的地址将被保存在异常地址寄存器中作为中断返回时指令的入口；如果异常事件是由除以零的除法指令产生的，因为这条指令不能完成，它将异常退休但不改变 CPU 的状态 (即不改变寄存器的值)，这条除法指令的地址将被保存在异常地址寄存器中，CPU 从中断服务程序返回时继续执行这条除法指令。

此外，R807 为了满足更高的实时性要求，支持软件开启中断快速响应模式。该模式下，CPU 能够直接响应中断异常，无需等待乘除法以及部分加载存储等长执行延时指令退休，从而提高中断响应的实时性。

异常按以下步骤被处理：

第一步，处理器更新 PSR 中的 VEC，然后保存 PSR 和 PC 到影子寄存器中。对于快速中断，PSR 和 PC 被保存到 FPSR 和 FPC 中；对于其它的异常，它们被保存到 EPSR 和 EPC 中。如果 PSR 的 EE 位被清零了，异常事件（除了普通中断和快速中断）将导致不可恢复的错误异常。PSR 和 PC 被保存后，PSR 的 EE 位被清零，处理器就只能处理不可恢复的错误异常。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第二步，处理器设置 PSR 的 S 位进入超级用户模式，并且把 PSR 的 TM 位清零防止异常服务程序被跟踪。中断使能位，即 PSR 的 IE 位也被清零禁止响应中断。如果异常是快速中断或重启异常，则快速中断

使能位，即 PSR 的 FE 位会被清零，但是其它的异常不影响 PSR 的 FE 位。

传输控制位，即 PSR 的 TE 位被清零，防止外部存储器管理单元对在异常入口地址计算时进行地址转译，使下面的访问以非转译的方式进行。

处理器还要决定对应的异常向量。对于向量中断，异常向量由外部的中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。异常向量用来计算异常服务程序的入口地址，并被保存在 PSR 的 VEC 中以支持共享异常服务的情况。

最后一步，处理器计算异常服务程序的第一条指令的地址并将 CPU 的控制权转交给异常服务程序。处理器把异常向量乘以 4 再加上异常向量基准地址（存在向量基准地址寄存器中）就得到了异常向量表中对应的异常入口地址。处理器用这个入口地址从存储器中读取一个字，这个异常入口地址的 [31:1] 转载到程序计数器中作为异常服务程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值的最低位无关）。

所有的异常向量存放在超级用户地址空间，并以指令空间索引（TC1 = 1）访问。在处理器地址映射中，只有重启向量是固定的。一旦处理器完成初始化，VBR 允许异常向量表的基准地址被重载。

R807 支持 1024 个字节的向量表包含 256 个异常向量（见图表 4-1）。开始的 31 个向量是用作在处理器内部识别的向量。第 32 个向量是留给软件的，用作指向系统描述符的指针。其余的 224 个向量是留给外部设备的。外部设备通过 8 位的中断向量和中断请求使处理器响应中断服务。处理器响应中断请求时锁存这个中断向量。对那些不能提供中断向量的设备，处理器为一般中断和快速中断提供了自动向量。

表 4.1: 异常向量分配

向量号	向量偏移（十六进制）	向量分配
0	000	重启异常。
1	004	未对齐访问异常。
2	008	访问错误异常。
3	00C	除以零异常。
4	010	非法指令异常。
5	014	特权违反异常。
6	018	跟踪异常。
7	01C	断点异常。
8	020	不可恢复错误异常
9	024	Idly 异常。
10	028	普通中断。
11	02C	快速中断。
12	030	保留（HAI）。
13	034	保留（FP）。
14	038	保留。
15	03C	保留。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3）。
20	050	保留。
21	054	保留。
20 - 29	058 - 074	保留。
30	078	保留
31	07C	系统描述符指针。
32 - 255	080 - 3FC	保留给向量中断控制器使用。

4.2 异常类型

本节描述外部中断异常和在 R807 内部产生的异常。R807 处理的异常有以下几类：

- 重启异常；
- 未对齐访问异常；
- 访问错误异常；
- 除以零异常；
- 非法指令异常；
- 特权违反异常；

- 跟踪异常；
- 断点异常；
- 不可恢复错误异常；
- Idly 异常；
- 普通中断；
- 快速中断；
- 陷阱指令异常；
- 浮点异常。

4.2.1 重启异常（向量偏移 0X0）

重启异常是所有异常中优先级最高的，它是用于系统初始化和发生重大故障后恢复系统。重启会中止处理器的所有操作，被中止的操作是不可恢复的。重启也在测试时用于初始化扫描链和时钟控制逻辑中锁存器的值，它也同时对处理器进行上电初始化。

重启异常设置 PSR（S）为高电平，使处理器工作在超级用户模式，并且把 PSR（TM）清零禁止跟踪异常。重启异常也会把 PSR（IE）和 PSR（FE）清零以禁止中断响应。同时，VBR（向量基准寄存器）也被清零，异常向量表的基准地址就是 0X00000000，CPU 从异常向量表中以偏移地址 0X0 为偏移地址读取异常向量，并把它装载到程序计数器（PC）。异常处理器把控制权转移到 PC 指向的地址。

4.2.2 未对齐访问异常（向量偏移 0X4）

处理器试图在与访问大小不一致的地址边界上执行访问操作，就会发生地址未对齐访问异常。通过设置 PSR（MM），这个异常可以被屏蔽，处理器会忽略对数据的对齐检查，而访问小于这个未对齐地址又最接近它的地址边界上。EPC 指向试图进行未对齐访问的指令。

未对齐访问异常只发生在数据上。

4.2.3 访问错误异常（向量偏移 0X8）

访问错误异常有三种触发的可能性：总线访问错误异常、内存保护区域访问异常）和 ECC 检测错误异常。

如果总线访问导致了一个错误的回复（即 pad_biu_hresp[1:0]=1），就意味着发生了访问错误异常。当访问 MPU 保护的区域出现访问错误时，也发生访问错误异常。

此外，如果处理器在 ECC 使能时：（1）检测出不可修复的 ECC 错误（D-Cache/I-TCM/D-TCM 出现 2-bit 错误），或者（2）检测出可修复的 ECC 错误（I-cache 出现 1/2-bit 错误，或者 D-Cache/I-TCM/D-TCM 出现 1-bit 错误）且 CRECC 位使能，也会发生访问错误异常。在异常处理程序中，可以查看控制寄存器 CR<6,1>，CR<7,1>，CR<8,1> 来分别查看发生 ECC 检测错误的信息、发生错误的地址和错误状态。

4.2.4 除以零异常（向量偏移 0X0C）

当处理器发现除法指令的除数是零时，处理器进行异常处理而不执行该除法指令。EPC 指向该除法指令。

4.2.5 非法指令异常（向量偏移 0X10）

处理器译码时如果发现了非法指令或没有实现的指令，该指令不会被执行而进行异常处理。EPC 指向该非法指令。

4.2.6 特权违反异常（向量偏移 0X14）

为了保护系统安全，一些指令被授予了特权，它们只能在超级用户模式下被执行。试图在用户模式下执行下面的特权指令都会产生特权违反异常：MFCR、MTCR、PSRSET、PSRCLR、RFI、RTE、STOP、WAIT、DOZE。

处理器如果发现了特权违反异常，在执行该指令前进行异常处理。EPC 指向该特权指令。

4.2.7 跟踪异常（向量偏移 0X18）

为了便于程序开发调试，R807 对每条指令或对改变控制流指令的进行跟踪。在指令跟踪模式下，每条指令在执行完后都会产生一个跟踪异常，以便于调试程序监测程序的执行。在跳转跟踪模式下，每条改变控制流的指令（包括 POP）都会产生一个跟踪异常。对于条件跳转指令，不管程序有没有跳转，跟踪异常都会发生。

PSR 的 TM 位控制跟踪模式。TM 的状态决定了指令退休时是否产生跟踪异常。请参考第三章 PSR 的 TM 位的定义。

跟踪异常处理起始于被跟踪的指令退休之后且在下一条指令退休之前。EPC 指向下一条指令。

以下控制相关的指令不能被跟踪：RTE，RFI，TRAP，STOP，WAIT，DOZE 和 BKPT。如果 EPSR（TP）有效，跟踪异常作为 RTE 或 RFI 正常执行的一部分被处理，而不管 PSR 或 EPSR 的 TM 位的状态。

如果被跟踪的指令由于发生了其它的异常而没有完成，跟踪异常不会被处理。如果在被跟踪的指令退休时处理器发现有中断等待处理，中断的优先级比跟踪异常高，PSR 的影子寄存器的 TP 位（等待处理的跟踪异常）将有效，处理器响应中断。中断服务程序的 RTE 或 RFI 退休时，等待处理的跟踪异常会被处理。等待处理的跟踪异常是用来跟踪前面的指令的，响应中断时这条指令已经退休，为了避免中断处理完之后跟踪异常丢失这种情况，中断处理完之后需要立即处理跟踪异常，这种情况下跟踪异常优先级是最高的仅次于重启。

4.2.8 断点异常（向量偏移 0X1C）

R807 提供了断点指令 BKPT 和硬断点请求管脚（pad_biu_brkrq_b[1:0]），它们被分配同一个断点异常向量。请参考第 11 章接口操作中 pad_biu_brkrq_b[1:0] 管脚的操作。为了尽量避免硬断点异常被丢

掉，硬断点异常被赋予了比中断更高的优先级。如果 pad_biu_brkrq_b[1:0] 设置在指令访问上，相应的指令不会被执行，当它退休时，处理器响应硬断点异常。如果 pad_biu_brkrq_b[1:0] 设置在数据访问上，相应的指令被允许完成，当它退休时，硬断点异常被响应。如果断点异常是由于 BKPT 指令或指令访问时 pad_biu_brkrq_b[1:0] 设置，EPC 指向该指令。如果数据访问时 pad_biu_brkrq_b[1:0] 设置，EPC 指向它的下一条指令。

4.2.9 不可恢复错误异常（向量偏移 0X20）

当 PSR（EE）为零时，除了快速中断外的异常会产生不可恢复的异常，因为这时用于异常恢复的信息（存于 EPC 和 EPSR）由于不可恢复的错误而被重写了。

由于所写的软件在 PSR（EE）为零时应该排除了异常事件发生的可能，在这种情况下如果 CPU 发生异常，这种错误一般意味着有系统错误。在不可恢复错误异常的服务程序中，引起不可恢复错误异常的异常类型是不确定的。

4.2.10 IDLY 异常（异常偏移 0X24）

IDLY 异常用来指示在 IDLY 指令序列中发生了传输错误。在该异常服务程序中，EPC 指向引起传输错误的指令。异常服务程序应该分析发生传输错误的原因，并备份 EPC 的值以便在必要时重新执行 IDLY 指令序列。

4.2.11 陷阱指令异常（向量偏移 0X40 - 0X4C）

一些指令可以用来显示地产生陷阱异常。TRAP # N 指令可以强制产生异常，它可以用于用户程序的系统调用。在异常服务程序中，EPC 指向 TRAP 指令。

4.3 中断异常

当外部设备需要向处理器请求服务或发送处理器需要的信息时，它可以用中断请求信号和相应的中断向量信号向处理器请求中断异常。

尽管处理器可以通过允许多周期指令被中断的方式来缩短中断响应的延时，一般中断在指令的边界上被确认。如果 PSR（IC）位使能，DIVS，DIVU，LDM，STM，LDQ 和 STQ 指令可以被中断而不等它们完成。

R807 提供了两个中断请求信号，支持自动提供中断向量号和由外设显式地提供中断向量号的方式。

图表 图 4.1 显示了和中断相关的到处理器核的接口信号。为了支持向量化的中断，外设在中断请求时，用 8 位的中断向量信号提供中断向量号，或设置 pad_biu_avec_b 使用预先定义好的中断向量号。如果 pad_biu_avec_b 有效，处理器使用自动向量号，它的向量偏移是 0X28，否则处理器使用 pad_biu_vec_b[7:0] 上提供的向量号。biu_pad_int_ack 指示 CPU 已经响应了中断。pad_biu_int_b，pad_biu_fint_b 和 pad_biu_avec_b 都是低电平有效。

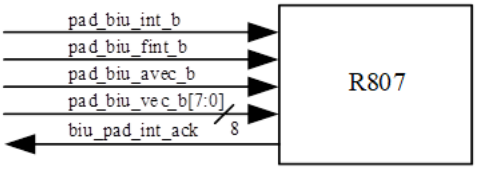


图 4.1: 中断接口信号

4.3.1 普通中断（INT）

pad_biu_int_b 是普通中断请求引脚。它是中断中优先级最低的。如果 PSR（IE）被清零了，pad_biu_int_b 输入信号被屏蔽，处理器不响应异常。普通中断使用 EPSR 和 EPC 这一组异常影子寄存器，它也可以被 PSR（EE）屏蔽。当 pad_biu_int_b 有效时，如果 pad_biu_avec_b 也有效，处理器使用自动向量号，它的向量偏移是 0X28，否则，处理器使用 pad_biu_vec_b[7:0] 上提供的向量号，它可以是 32 - 255 中的任意一个（硬件上不排除使用 0 - 31）。

4.3.2 快速中断（FINT）

pad_biu_fint_b 是快速中断请求输入引脚。如果快速中断被使能（PSR（FE）有效），它的优先级比普通中断请求高。快速中断使用 FPSR 和 FPC 这一组异常影子寄存器，它不会被 PSR（EE）屏蔽。如果 PSR（FE）清零了，快速中断就被屏蔽了。当 pad_biu_fint_b 有效时，如果 pad_biu_avec_b 也有效，处理器使用自动向量号，它的向量偏移是 0X2C，否则，处理器使用 pad_biu_vec_b[7:0] 上提供的向量号，它可以是 32 - 255 中的任意一个（硬件上不排除使用 0 - 31）。

4.3.3 中断处理过程

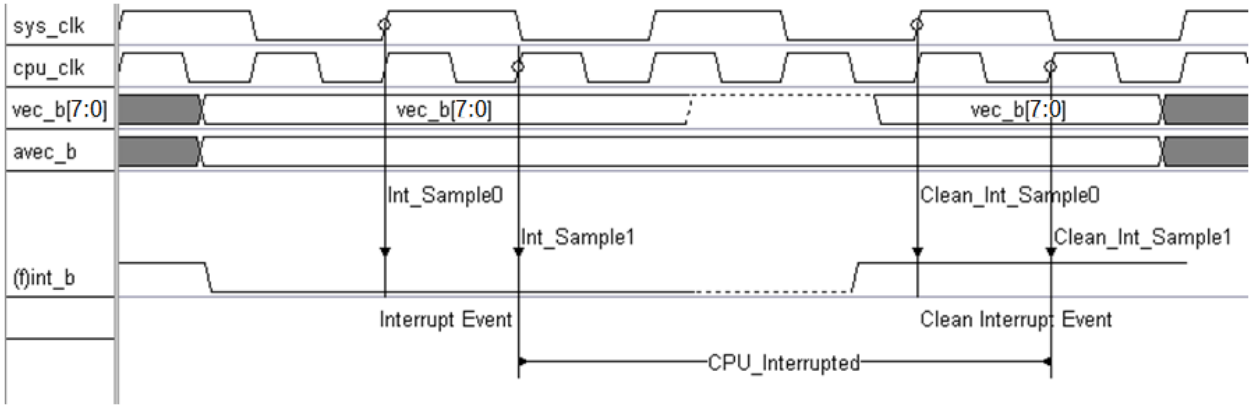


图 4.2: 中断处理过程

在图表 图 4.2 中，当中断向量号 pad_biu_vec_b[7:0] 和自动中断向量 pad_biu_avec_b 都已经准备好时，拉低普通中断信号线 pad_biu_int_b，该信号经系统时钟 sys_clk 和 CPU 内部时钟 cpu_clk 的上升沿依次采样后，CPU 内部收到中断，并根据 pad_biu_avec_b 的设置取得中断向量，进入中断服务程序。在

中断服务程序中，应该清除外部中断，即拉高 pad_biu_int_b，此信号亦需经此两个时钟依次采样后才会退出中断。快速中断 (fint) 的操作亦是如此。

4.4 异常优先级

如图表 表 4.2 所示，根据异常的特性和被处理的先后关系，R807 把优先级分为 9 级。在图表 表 4.2 中，1 代表最高优先级，9 代表了最低优先级。值得注意的是，在第 8 组中，几个异常共享一个优先级，因为它们之间相互有排斥性。

在 R807 里，多个异常可以同时发生。重启异常是很特别的，它有最高的优先级。所有其它的异常按图表 表 4.2 中的优先级关系进行处理。

如果 PSR（EE）被清零了，当异常发生时，处理器处理的是不可恢复异常。如果多个异常同时发生，拥有最高优先级的异常最先被处理。处理器在异常返回后，重新执行产生异常的指令时，其余的异常可以重现。

表 4.2: 异常优先级

优先级	异常与它相关的优先级	特征
1	重启异常	处理器中止所有程序运行，初始化系统。
2	待处理的跟踪异常	如果 TP=1，在 RTE 或 RFI 指令退休后，处理器处理待处理的跟踪异常。
3	硬断点请求	在相关的指令退休后，硬断点请求被处理。
4	IDLY 错误	在相关的指令退休后，处理器保存上下文并处理异常。
5	不对齐错误	在相关的指令退休后，处理器保存上下文并处理异常。
6.0 6.1	快速中断 普通中断	如果 IC=0，中断在指令退休后被响应；如果 IC=1，处理器允许中断在指令完成之前就被响应。
7.0	不可恢复异常 访问错误	在相关的指令退休后，处理器保存上下文并处理异常。
8	非法指令 特权异常 禁止硬件加速器 除以零 陷阱指令 断点指令	在相关的指令退休后，处理器保存上下文并处理异常。
9	跟踪异常	在相关的指令退休后，处理器保存上下文并处理异常。

4.4.1 发生待处理的异常时调试请求

处理器如果在异常发生的同时接收到了调试请求信号，先进入调试模式。异常延后处理，直到处理器退出调试模式和产生异常的指令重新被执行。

4.5 异常返回

异常返回根据正在处理的异常类型，处理器通过执行 RFI 或 RTE 指令从异常服务程序中返回。RFI 指令利用保存在 FPSR 和 FPC 影子寄存器中的上下文从异常服务程序中返回；RTE 指令利用保存在 EPSR 和 EPC 影子寄存器中的上下文从异常服务程序中返回。

第五章　指令集

5.1　概述

R807 的指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。R807 指令有两种宽度：16 位指令和 32 位指令，指令代码由两种指令混编而成，两种指令之间的切换没有额外的开销。

5.2　32 位指令

本节主要介绍 R807 实现的玄铁 CPU V2 的 32 位指令集。

5.2.1　32 位指令功能分类

R807 实现的玄铁 CPU V2 的 32 位指令集按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 分支跳转指令
- 内存存取指令
- 特权指令
- 特殊功能指令

5.2.1.1　数据运算指令

数据运算类指令可以进一步分为：

加减法指令：

ADDU32	**无符号加法指令**
ADDC32	无符号带进位加法指令
ADDI32	无符号立即数加法指令
SUBU32	无符号减法指令
SUBC32	无符号带借位减法指令
SUBI32	无符号立即数减法指令
RSUB32	反向减法指令
IXH32	索引半字指令
IXW32	索引字指令
IXD32	索引双字指令
INCF32	C 为 0 立即数加法指令
INCT32	C 为 1 立即数加法指令
DECF32	C 为 0 立即数减法指令
DECT32	C 为 1 立即数减法指令
DECGT32	减法大于零置 C 位指令
DECLT32	减法小于零置 C 位指令
DECNE32	减法不等于零置 C 位指令

图表 5-1 32 位加减法指令列表

逻辑操作指令:

AND32	**按位与指令**
ANDI32	立即数按位与指令
ANDN32	按位非与指令
ANDNI32	立即数按位非与指令
OR32	按位或指令
ORI32	立即数按位或指令
XOR32	按位异或指令
XORI32	立即数按位异或指令
NOR32	按位或非指令
NOT32	按位非指令

图表 5-2 32 位逻辑操作指令列表

移位指令:

LSL32	**逻辑左移指令**
LSLI32	立即数逻辑左移指令
LSLC32	立即数逻辑左移至 C 位指令
LSR32	逻辑右移指令
LSRI32	立即数逻辑右移指令
LSRC32	立即数逻辑右移至 C 位指令
ASR32	算术右移指令
ASRI32	立即数算术右移指令
ASRC32	立即数算术右移至 C 位指令
ROTL32	循环左移指令
ROTLI32	立即数循环左移指令
XSR32	扩展右移指令

图表 5-3 32 位移位指令列表

比较指令：

CMPNE32	**不等比较指令**
CMPNEI32	立即数不等比较指令
CMPHS32	无符号大于等于比较指令
CMPHSI32	立即数无符号大于等于比较指令
CMPLT32	有符号小于比较指令
CMPLTI32	立即数有符号小于比较指令
TST32	零测试指令
TSTNBZ32	无字节等于零寄存器测试指令

图表 5-4 32 位比较指令列表

数据传输指令：

MOV32	**数据传送指令**
MOVF32	C 为 0 数据传送指令
MOVT32	C 为 1 数据传送指令
MOVI32	立即数数据传送指令
MOVIH32	立即数高位数据传送指令
MTHI32	累加器高位写传送指令
MTLO32	累加器低位写传送指令
MFHI32	累加器高位读传送指令
MFHIS32	累加器高位饱和读传送指令
MFLO32	累加器低位读传送指令
MFLOS32	累加器低位饱和读传送指令
MVCV32	C 位取反传送指令
MVC32	C 位传送指令
MVTC32	溢出位复制到 C 位指令
CLRF32	C 为 0 清零指令
CLRT32	C 为 1 清零指令
LRW32	存储器读入指令
GRS32	符号产生指令

图表 5-5 32 位数据传输指令列表

比特操作指令：

BCLRI32	**立即数位清零指令**
BSETI32	立即数位置位指令
BTSTI32	立即数位测试指令

图表 5-6 32 位比特操作指令列表

提取插入指令：

ZEXT32	**位提取并无符号扩展指令**
SEXT32	位提取并有符号扩展指令
INS32	位插入指令
ZEXTB32	字节提取并无符号扩展指令
ZEXTH32	半字提取并无符号扩展指令
SEXTB32	字节提取并有符号扩展指令
SEXTH32	半字提取并有符号扩展指令
XTRB0.32	提取字节 0 并无符号展指令
XTRB1.32	提取字节 1 并无符号扩展指令
XTRB2.32	提取字节 2 并无符号扩展指令
XTRB3. 32	提取字节 3 并无符号扩展指令
BREV32	位倒序指令
REVB32	字节倒序指令
REVH32	半字内字节倒序指令

图表 5-7 32 位提取插入指令列表

乘除法指令:

MULU32	**无符号数乘法指令**
MULS32	有符号数乘法指令
MULUA32	无符号数乘累加指令
MULSA32	有符号数乘累加指令
MULUS32	无符号数乘累减指令
MULSS32	有符号数乘累减指令
MULT32	乘法指令
MULSH32	16 位有符号乘法指令
MULSHA32	16 位有符号乘累加指令
MULSHS32	16 位有符号乘累减指令
MULSW32	有符号数 16x32 乘法指令
MULSWA32	有符号数 16x32 乘累加指令
MULSWS32	有符号数 16x32 乘累减指令
DIVU32	无符号除法指令
DIVS32	有符号除法指令

图表 5-8 32 位乘除法指令列表

杂类运算指令:

ABS32	**绝对值指令**
FF0. 32	快速找 0 指令
FF1. 32	快速找 1 指令
BMASKI32	立即数位屏蔽产生指令
BGENR32	寄存器位产生指令
BGENI32	立即数位产生指令

图表 5-9 32 位杂类运算指令列表

5.2.1.2 分支跳转指令

分支跳转指令可以进一步分为:

分支指令:

BT32	**C 为 1 分支指令**
BF32	C 为 0 分支指令
BEZ32	寄存器等于零分支指令
BNEZ32	寄存器不等于零分支指令
BHZ32	寄存器大于零分支指令
BLSZ32	寄存器小于等于零分支指令
BLZ32	寄存器小于零分支指令
BHSZ32	寄存器大于等于零分支指令

图表 5-10 32 位分支指令列表

跳转指令:

BR32	**无条件跳转指令**
BSR32	跳转到子程序指令
JMPI32	间接跳转指令
JSRI32	间接跳转到子程序指令
JMP32	寄存器跳转指令
JSR32	寄存器跳转到子程序指令
RTS32	链接寄存器跳转指令

图表 5-11 32 位跳转指令列表

5.2.1.3 内存存取指令

内存存取指令可以进一步分为:

立即数偏移存取指令：

LD32.B	**无符号扩展字节加载指令**
LD32.BS	有符号扩展字节加载指令
LD32.H	无符号扩展半字加载指令
LD32.HS	有符号扩展半字加载指令
LD32.W	字加载指令
LD32.D	双字加载指令
ST32.B	字节存储指令
ST32.H	半字存储指令
ST32.W	字存储指令
ST32.D	双字存储指令

图表 5-12 32 位立即数偏移存取指令列表

向量寄存器偏移存取指令：

LDR32.B	**寄存器移位寻址无符号扩展字节加载指令**
LDR32.BS	寄存器移位寻址有符号扩展字节加载指令
LDR32.H	寄存器移位寻址无符号扩展半字加载指令
LDR32.HS	寄存器移位寻址有符号扩展半字加载指令
LDR32.W	寄存器移位寻址字加载指令
STR32.B	寄存器移位寻址字节存储指令
STR32.H	寄存器移位寻址半字存储指令
STR32.W	寄存器移位寻址字存储指令

图表 5-13 32 位向量寄存器偏移存取指令列表

多寄存器存取指令：

LDQ32	**连续四字加载指令**
LDM32	连续多字加载指令
STQ32	连续四字存储指令
STM32	连续多字存储指令
PUSH32	压栈指令
POP32	出栈指令

图表 5-14 32 位多寄存器存取指令列表

符号存取指令：

LRS32.B	**字节符号加载指令**
LRS32.H	半字符号加载指令
LRS32.W	字符号加载指令
SRS32.B	字节符号存储指令
SRS32.H	半字符号存储指令
SRS32.W	字符号存储指令

图表 5-15 32 位符号存取指令列表

独占存取指令：

LDEX32.W	**独占式字加载指令**
STEX32.W	独占式字存储指令

图表 5-16 32 位独占存取指令列表

5.2.1.4 特权指令

特权指令可以进一步分为：

控制寄存器操作指令：

MFCR32	**控制寄存器读传送指令**
MTCR32	控制寄存器写传送指令
PSRSET32	PSR 位置位指令
PSRCLR32	PSR 位清零指令

图表 5-17 32 位控制寄存器操作指令列表

低功耗指令：

WAIT32	**进入低功耗等待模式指令**
DOZE32	进入低功耗睡眠模式指令
STOP32	进入低功耗暂停模式指令

图表 5-18 32 位低功耗指令列表

异常返回指令：

RTE32	**异常和普通中断返回指令**
RFI32	快速中断返回指令

图表 5-19 32 位异常返回指令列表

5.2.1.5 特殊功能指令

特殊功能指令具体包括:

SYNC32	CPU 同步指令
BKPT32	断点指令
SCE32	条件执行设置指令
IDLY32	中断识别禁止指令
TRAP32	无条件操作系统陷阱指令
PLDR32	读数据预取指令
PLDW32	写数据预取指令

图表 5-20 32 位特殊功能指令列表

5.3 16 位指令

本节主要介绍 R807 实现的玄铁 CPU V2 的 16 位指令集。

5.3.1 16 位指令功能分类

玄铁 CPU V2 的 16 位指令集按照指令实现的功能来划分，可以分为:

- 数据运算指令
- 分支跳转指令
- 内存存取指令

5.3.1.1 数据运算指令

数据运算类指令可以进一步分为:

加减法指令:

ADDU16	无符号加法指令
ADDC16	无符号带进位加法指令
ADDI16	无符号立即数加法指令
SUBU16	无符号减法指令
SUBC16	无符号带借位减法指令
SUBI16	无符号立即数减法指令

图表 5-21 16 位加减法指令列表

逻辑操作指令：

AND16	**按位与指令**
ANDN16	按位非与指令
OR16	按位或指令
XOR16	按位异或指令
NOR16	按位或非指令
NOT16	按位非指令

图表 5-22 16 位逻辑操作指令列表

移位指令：

LSL16	**逻辑左移指令**
LSLI16	立即数逻辑左移指令
LSR16	逻辑右移指令
LSRI16	立即数逻辑右移指令
ASR16	算术右移指令
ASRI16	立即数算术右移指令
ROTL16	循环左移指令

图表 5-23 16 位移位指令列表

比较指令：

CMPNE16	**不等比较指令**
CMPNEI16	立即数不等比较指令
CMPHS16	无符号大于等于比较指令
CMPHSI16	立即数无符号大于等于比较指令
CMPLT16	有符号小于比较指令
CMPLTI16	立即数有符号小于比较指令
TST16	零测试指令
TSTNBZ16	无字节等于零寄存器测试指令

图表 5-24 16 位比较指令列表

数据传输指令：

MOV16	**数据传送指令**
MOVI16	立即数数据传送指令
MVCV16	C 位取反传送指令
LRW16	存储器读入指令

图表 5-25 16 位数据传输指令列表

比特操作指令:

BCLRI16	**立即数位清零指令**
BSETI16	立即数位置位指令

图表 5-26 16 位比特操作指令列表

提取插入指令:

ZEXTB16	**字节提取并无符号扩展指令**
ZEXTH16	半字提取并无符号扩展指令
SEXTB16	字节提取并有符号扩展指令
SEXTH16	半字提取并有符号扩展指令
REVB16	字节倒序指令
REVH16	半字内字节倒序指令

图表 5-27 16 位提取插入指令列表

乘除法指令:

MULT16	**乘法指令**
MULSH16	16 位有符号乘法指令

图表 5-28 16 位乘法指令列表

5.3.1.2 分支跳转指令

分支跳转指令可以进一步分为:

分支指令:

BT16	**C 为 1 分支指令**
BF16	C 为 0 分支指令

图表 5-29 16 位分支指令列表

跳转指令:

BR16	**无条件跳转指令**
JMP16	寄存器跳转指令
JSR16	寄存器跳转到子程序指令
RTS16	链接寄存器跳转指令

图表 5-30 16 位跳转指令列表

5.3.1.3 内存存取指令

内存存取指令可以进一步分为:

立即数偏移存取指令:

LD16.B	**无符号扩展字节加载指令**
LD16.H	无符号扩展半字加载指令
LD16.W	字加载指令
ST16.B	字节存储指令
ST16.H	半字存储指令
ST16.W	字存储指令

图表 5-31 16 位立即数偏移存取指令列表

多寄存器存取指令:

POP16	**出栈指令**
PUSH16	压栈指令

图表 5-32 16 位多寄存器存取指令列表

5.4 指令集列表

R807 的指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。

图表 5-33 列出了 R807 指令集中所有 16 位和 32 位指令。

图表 5-33 R807 指令集

汇编指令	32 位	16 位	指令描述
ADDU	√	√	无符号加法指令
ADDC	√	√	无符号带进位加法指令
ADDI	√	√	无符号立即数加法指令
SUBU	√	√	无符号减法指令
SUBC	√	√	无符号带借位减法指令
SUBI	√	√	无符号立即数减法指令

下页继续

表 5.1 – 续上页

汇编指令	32 位	16 位	指令描述
RSUB	√	×	反向减法指令
IXH	√	×	索引半字指令
IXW	√	×	索引字指令
IXD	√	×	索引双字指令
INCF	√	×	C 为 0 立即数加法指令
INCT	√	×	C 为 1 立即数加法指令
DECF	√	×	C 为 0 立即数减法指令
DECT	√	×	C 为 1 立即数减法指令
DECGT	√	×	减法大于零置 C 位指令
DECLT	√	×	减法小于零置 C 位指令
DECNE	√	×	减法不等于零置 C 位指令
AND	√	√	按位与指令
ANDI	√	×	立即数按位与指令
ANDN	√	√	按位非与指令
ANDNI	√	×	立即数按位非与指令
OR	√	√	按位或指令
ORI	√	×	立即数按位或指令
XOR	√	√	按位异或指令
XORI	√	×	立即数按位异或指令
NOR	√	√	按位或非指令
NOT	√	√	按位非指令
LSL	√	√	逻辑左移指令
LSLI	√	√	立即数逻辑左移指令
LSLC	√	×	立即数逻辑左移至 C 位指令
LSR	√	√	逻辑右移指令
LSRI	√	√	立即数逻辑右移指令
LSRC	√	×	立即数逻辑右移至 C 位指令
ASR	√	√	算术右移指令
ASRI	√	√	立即数算术右移指令
ASRC	√	×	立即数算术右移至 C 位指令
ROTL	√	√	循环左移指令
ROTLI	√	×	立即数循环左移指令
XSR	√	×	扩展右移指令
CMPNE	√	√	不等比较指令
CMPNEI	√	√	立即数不等比较指令
CMPHS	√	√	无符号大于等于比较指令
CMPHSI	√	√	立即数无符号大于等于比较指令

下页继续

表 5.1 – 续上页

汇编指令	32 位	16 位	指令描述
CMPLT	√	√	有符号小于比较指令
CMPLTI	√	√	立即数有符号小于比较指令
TST	√	√	零测试指令
TSTNBZ	√	√	无字节等于零寄存器测试指令
MOV	√	√	数据传送指令
MOVF	√	×	C 为 0 数据传送指令
MOVT	√	×	C 为 1 数据传送指令
MOVI	√	√	立即数数据传送指令
MOVIH	√	×	立即数高位数据传送指令
LRW	√	√	存储器读入指令
GRS	√	×	符号产生指令
MVCV	√	√	C 位取反传送指令
MVC	√	×	C 位传送指令
CLRF	√	×	C 为 0 清零指令
CLRT	√	×	C 为 1 清零指令
BCLRI	√	√	立即数位清零指令
BSETI	√	√	立即数位置位指令
BTSTI	√	×	立即数位测试指令
ZEXT	√	×	位提取并无符号扩展指令
SEXT	√	×	位提取并有符号扩展指令
INS	√	×	位插入指令
ZEXTB	√	√	字节提取并无符号扩展指令
ZEXTH	√	√	半字提取并无符号扩展指令
SEXTB	√	√	字节提取并有符号扩展指令
SEXTH	√	√	半字提取并有符号扩展指令
XTRB0	√	×	提取字节 0 并无符号扩展指令
XTRB1	√	×	提取字节 1 并无符号扩展指令
XTRB2	√	×	提取字节 2 并无符号扩展指令
XTRB3	√	×	提取字节 3 并无符号扩展指令
BREV	√	×	位倒序指令
REVB	√	√	字节倒序指令
REVH	√	√	半字内字节倒序指令
MULT	√	√	乘法指令
MULSH	√	√	16 位有符号乘法指令
DIVU	√	×	无符号除法指令
DIVS	√	×	有符号除法指令
ABS	√	×	绝对值指令

下页继续

表 5.1 – 续上页

汇编指令	32 位	16 位	指令描述
FF0	√	×	快速找 0 指令
FF1	√	×	快速找 1 指令
BMASKI	√	×	立即数位屏蔽产生指令
BGENR	√	×	寄存器位产生指令
BGENI	√	×	立即数位产生指令
BT	√	√	C 为 1 分支指令
BF	√	√	C 为 0 分支指令
BEZ	√	×	寄存器等于零分支指令
BNEZ	√	×	寄存器不等于零分支指令
BHZ	√	×	寄存器大于零分支指令
BLSZ	√	×	寄存器小于等于零分支指令
BLZ	√	×	寄存器小于零分支指令
BHSZ	√	×	寄存器大于等于零分支指令
BR	√	√	无条件跳转指令
BSR	√	×	跳转到子程序指令
JMPI	√	×	间接跳转指令
JSRI	√	×	间接跳转到子程序指令
JMP	√	√	寄存器跳转指令
JSR	√	√	寄存器跳转到子程序指令
RTS	√	√	链接寄存器跳转指令
LD.B	√	√	无符号扩展字节加载指令
LD.BS	√	×	有符号扩展字节加载指令
LD.H	√	√	无符号扩展半字加载指令
LD.HS	√	×	有符号扩展半字加载指令
LD.W	√	√	字加载指令
LD.D	√	×	双字加载指令
LDEX.W	√	×	独占式字加载指令
ST.B	√	√	字节存储指令
ST.H	√	√	半字存储指令
ST.W	√	√	字存储指令
ST.D	√	×	双字存储指令
STEX.W	√	×	独占式字存储指令
LDR.B	√	×	寄存器移位寻址无符号扩展字节加载指令
LDR.BS	√	×	寄存器移位寻址有符号扩展字节加载指令
LDR.H	√	×	寄存器移位寻址无符号扩展半字加载指令
LDR.HS	√	×	寄存器移位寻址有符号扩展半字加载指令
LDR.W	√	×	寄存器移位寻址字加载指令

下页继续

表 5.1 – 续上页

汇编指令	32 位	16 位	指令描述
STR.B	√	×	寄存器移位寻址字节存储指令
STR.H	√	×	寄存器移位寻址半字存储指令
STR.W	√	×	寄存器移位寻址字存储指令
LRS.B	√	×	字节符号加载指令
LRS.H	√	×	半字符号加载指令
LRS.W	√	×	字符号加载指令
SRS.B	√	×	字节符号存储指令
SRS.H	√	×	半字符号存储指令
SRS.W	√	×	字符号存储指令
LDQ	√	×	连续四字加载指令
LDM	√	×	连续多字加载指令
STQ	√	×	连续四字存储指令
STM	√	×	连续多字存储指令
POP	√	√	出栈指令
PUSH	√	√	压栈指令
MFCR	√	×	控制寄存器读传送指令
MTCR	√	×	控制寄存器写传送指令
PSRSET	√	×	PSR 位置位指令
PSRCLR	√	×	PSR 位清零指令
WAIT	√	×	进入低功耗等待模式指令
DOZE	√	×	进入低功耗睡眠模式指令
STOP	√	×	进入低功耗暂停模式指令
RTE	√	×	异常和普通中断返回指令
RFI	√	×	快速中断返回指令
SYNC	√	×	CPU 同步指令
BKPT	×	√	断点指令
SCE	√	×	条件执行设置指令
IDLY	√	×	中断识别禁止指令
TRAP	√	×	无条件操作系统陷阱指令
PLDR	√	×	读数据预取指令
PLDW	√	×	写数据预取指令
LDEX.W	√	×	独占式字加载指令
STEX.W	√	×	独占式字存储指令
*MTHI	√	×	累加器高位写传送指令
*MTLO	√	×	累加器低位写传送指令
*MFHI	√	×	累加器高位读传送指令
*MFHIS	√	×	累加器高位饱和读传送指令

下页继续

表 5.1 – 续上页

汇编指令	32 位	16 位	指令描述
*MFLO	√	×	累加器低位读传送指令
*MFLOS	√	×	累加器低位饱和读传送指令
*MVTC	√	×	溢出位复制到 C 位指令
*MULU	√	×	无符号数乘法指令
*MULS	√	×	有符号数乘法指令
*MULSHA	√	×	16 位有符号数乘累加指令
*MULSHS	√	×	16 位有符号数乘累减指令
*MULUA	√	×	无符号数乘累加指令
*MULSA	√	×	有符号数乘累加指令
*MULUS	√	×	无符号数乘累减指令
*MULSS	√	×	有符号数乘累减指令
*MULSW	√	×	有符号数 16x32 乘法指令
*MULSWA	√	×	有符号数 16x32 乘累加指令
*MULSWS	√	×	有符号数 16x32 乘累减指令

注解: √ 表示相应指令集中存在该指令，× 表示相应指令集中不存在该指令。

5.5 指令执行延时

图表 5-34 指令延时列表

指令类型	指令	执行周期	备注
加减法指令	ADDU32/16	1	•
加减法指令	ADDC32/16	1	•
加减法指令	ADDI32/16	1	•
加减法指令	SUBU32/16	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
加减法指令	SUBC32/16	1	•
加减法指令	SUBI32/16	1	•
加减法指令	RSUB32	1	•
加减法指令	IXH32	1	•
加减法指令	IXW32	1	•
加减法指令	IXD32	1	•
加减法指令	INCF32	1	•
加减法指令	INCT32	1	•
加减法指令	DECF32	1	•
加减法指令	DECT32	1	•
加减法指令	DECGT32	1	•
加减法指令	DECLT32	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
加减法指令	DECNE32	1	•
逻辑操作指令	AND32/16	1	•
逻辑操作指令	ANDI32/16	1	•
逻辑操作指令	ANDN32	1	•
逻辑操作指令	ANDNI32	1	•
逻辑操作指令	OR32/16	1	•
逻辑操作指令	ORI32	1	•
逻辑操作指令	XOR32/16	1	•
逻辑操作指令	XORI32	1	•
逻辑操作指令	NOR32/16	1	•
逻辑操作指令	NOT32/16	1	•
移位指令	LSL32/16	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
移位指令	LSLI32/16	1	•
移位指令	LSLC32	1	•
移位指令	LSR32/16	1	•
移位指令	LSRI32/16	1	•
移位指令	LSRC32	1	•
移位指令	ASR32/16	1	•
移位指令	ASRI32/16	1	•
移位指令	ASRC32	1	•
移位指令	ROTL32/16	1	•
移位指令	ROTLI32	1	•
移位指令	XSR32	1	•
比较指令	CMPNE32/16	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
比较指令	CMPNEI32/16	1	•
比较指令	CMPHS32/16	1	•
比较指令	CMPHSI32/16	1	•
比较指令	CMPLT32/16	1	•
比较指令	CMPLTI32/16	1	•
比较指令	TST32/16	1	•
比较指令	TSTNBZ32/16	1	•
数据传输指令	MOV32/16	1	•
数据传输指令	MOVF32	1	•
数据传输指令	MOVT32	1	•
数据传输指令	MOVI32/16	1	•
数据传输指令	MOVIH32	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
数据传输指令	MVCV32/16	1	•
数据传输指令	MVC32	1	•
数据传输指令	MVTC32	1	•
数据传输指令	CLRF32	1	•
数据传输指令	CLRT32	1	•
数据传输指令	LRW32/16*	>=3	1 周期 TCM 命中：3 个周期； 2 周期 TCM 命中：4 个周期。 D-Cache 命中：3 个周期。 D-Cache 缺失：片外访问延迟。
数据传输指令	GRS32	4	•
比特操作指令	BCLRI32/16	1	•
比特操作指令	BSETI32/16	1	•
比特操作指令	BTSTI32/16	1	•
提取插入指令	ZEXT32/16	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
提取插入指令	SEXT32/16	1	•
提取插入指令	INS32	1	•
提取插入指令	ZEXTB32	1	•
提取插入指令	ZEXTH32	1	•
提取插入指令	SEXTB32/16	1	•
提取插入指令	SEXTH32/16	1	•
提取插入指令	XTRB0.32	1	•
提取插入指令	XTRB1.32	1	•
提取插入指令	XTRB2.32	1	•
提取插入指令	XTRB3.32	1	•
提取插入指令	BREV32	1	•
提取插入指令	REVB32/16	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
提取插入指令	REVH32/16	1	•
乘除法指令	MULT32/16	3	•
乘除法指令	MULSH32/16	3	•
乘除法指令	*MTHI32	1	•
乘除法指令	*MTLO32	1	•
乘除法指令	*MFHI32	1	•
乘除法指令	*MFHIS32	1	•
乘除法指令	*MFLO32	1	•
乘除法指令	*MFLOS32	1	•
乘除法指令	*MULU32	3	•
乘除法指令	*MULS32	3	•
乘除法指令	*MULSW32	3	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
乘除法指令	*MULSWA32	4	•
乘除法指令	*MULSWS32	4	•
乘除法指令	*MULSHA32	4	•
乘除法指令	*MULSHS32	4	•
乘除法指令	*MULUA32	4	•
乘除法指令	*MULSA32	4	•
乘除法指令	*MULUS32	4	•
乘除法指令	*MULSS32	4	•
乘除法指令	DIVU32	4~36	•
乘除法指令	DIVS32	4~36	•
杂类运算指令	ABS32	1	•
杂类运算指令	FF0. 32	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
杂类运算指令	FF1. 32	1	•
杂类运算指令	BMASKI32	1	•
杂类运算指令	BGENR32	1	•
杂类运算指令	BGENI32	1	•
分支指令	BT32/16	1	•
分支指令	BF32/16	1	•
分支指令	BEZ32	1	•
分支指令	BNEZ32	1	•
分支指令	BHZ32	1	•
分支指令	BLSZ32	1	•
分支指令	BLZ32	1	•
分支指令	BHSZ32	1	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
跳转指令	BR32/16	1	•
跳转指令	BSR32	1	•
跳转指令	JMPI32	>=3+1	拆分为 LRW 及 JMP 指令。
跳转指令	JSRI32	>=3+1	拆分为 LRW 及 JSR 指令。
跳转指令	JMP32/16	1	•
跳转指令	JSR32/16	1	•
跳转指令	RTS32/16	1	•
立即数偏移存取指令	LD32/16.B	>=3	1 周期 TCM 命中：3 个周期； 2 周期 TCM 命中：4 个周期。 D-Cache 命中：3 个周期。 D-Cache 缺失：片外访问延迟。
立即数偏移存取指令	LD32.BS	>=3	
立即数偏移存取指令	LD32/16.H	>=3	
立即数偏移存取指令	LD32.HS	>=3	
立即数偏移存取指令	LD32/16.W	>=3	
立即数偏移存取指令	LD32.D	>=3+1	拆分为 2 条 LD.W 指令。
立即数偏移存取指令	ST32/16.B	2	•
立即数偏移存取指令	ST32/16.H	2	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
立即数偏移存取指令	ST32/16.W	2	•
立即数偏移存取指令	ST32.D	2+1	拆分为 2 条 ST.W 指令。
寄存器偏移存取指令	LDR32.B	>=3	1 周期 TCM 命中：3 个周期； 2 周期 TCM 命中：4 个周期。 D-Cache 命中：3 个周期。 D-Cache 缺失：片外访问延迟。
寄存器偏移存取指令	LDR32.BS	>=3	
寄存器偏移存取指令	LDR32.H	>=3	
寄存器偏移存取指令	LDR32.HS	>=3	
寄存器偏移存取指令	LDR32.W	>=3	
寄存器偏移存取指令	STR32.B	2+1	拆分为 1 条 IXW 和 1 条 ST.B/H/W 指令。
寄存器偏移存取指令	STR32.H	2+1	
寄存器偏移存取指令	STR32.W	2+1	
多寄存器存取指令	LDQ32	>=3+3	拆分为 4 条 LD.W 指令。
多寄存器存取指令	LDM32	>=3+N-1	拆分为 N 条 LD.W 指令。
多寄存器存取指令	STQ32	3+2	拆分为 4 条 ST.W 指令。
多寄存器存取指令	STM32	2+N-1	拆分为 N 条 ST.W 指令。
多寄存器存取指令	PUSH32/16	2+N	拆分为 SUB 和 N 条 ST.W 指令。
多寄存器存取指令	POP32/16	>=3+N+1	拆分为 N 条 LD.W、ADD 和 RTS 指令。
独占存取指令	LDEX32.W	>=3	D-Cache 命中：3 个周期 D-Cache 缺失：片外访问延迟。
独占存取指令	STEX32.W	>=2	•

下页继续

表 5.2 – 续上页

指令类型	指令	执行周期	备注
符号存取指令	LRS32.B	>=3	D-Cache 命中: 3 个周期 D-Cache 缺失:片外访问延迟。
符号存取指令	LRS32.H	>=3	
符号存取指令	LRS32.W	>=3	
符号存取指令	SRS32.B	2	•
符号存取指令	SRS32.H	2	•
符号存取指令	SRS32.W	2	•
控制寄存器操作指令	MFCR32	*	阻塞执行
控制寄存器操作指令	MTCR32	*	阻塞执行
控制寄存器操作指令	PSRSET32	*	阻塞执行
控制寄存器操作指令	PSRCLR32	*	阻塞执行
低功耗指令	WAIT32	*	阻塞执行
低功耗指令	DOZE32	*	阻塞执行
低功耗指令	STOP32	*	阻塞执行
异常返回指令	RTE32	*	阻塞执行
异常返回指令	RFI32	*	阻塞执行
异常返回指令	PLDW32	>=3	•

第六章　浮点单元

R807 支持可配的紧耦合浮点单元，用以实现对浮点运算的硬件支持，具有低成本和高效率的特点。在系统软件支持下，R807 浮点单元完全兼容 ANSI/IEEE Std 754 浮点标准。

6.1　特点

浮点单元的体系结构与编程模型的主要特点如下：

- 完全兼容 ANSI/IEEE Std 754（系统软件支持下）；
- 支持单精度和双精度浮点运算；
- 支持单精度浮点运算的单指令多数据（SIMD）处理；
- 支持向零舍入、向正无穷舍入、向负无穷舍入和就近舍入四种舍入方式；
- 支持浮点异常的陷入与非陷入两种处理模式；
- 支持浮点异常的精确处理；
- 支持浮点硬件除法与开方；

浮点单元的微体系结构的主要特点如下：

- 16 个独立的 64 位浮点寄存器；
- 单发射架构，每个周期处理一条浮点算术指令；
- 支持浮点算术指令的按序发射、按序执行、按序回写；
- 支持浮点算术指令与整形指令的并行发射与执行，支持浮点算术指令与加载存储指令的并行发射与执行；
- 包含三条独立的执行流水线，分别是浮点 ALU、浮点乘法与浮点除法开方；
- 优化的单精度执行流水线；
- 基于运算部件复用的成本优化技术；
- 基于门控时钟的功耗优化技术。

6.2 浮点指令功能分类

R807 实现的玄铁 CPU V2 的浮点指令集按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 传输类指令
- 内存存取指令

6.2.1 数据运算指令

图表 6-1 单精度数据运算指令

FSTOSI	**单精度浮点数转换为有符号整数**
FSTOUI	单精度浮点数转换为无符号整数
FSITOS	有符号整数转换为单精度浮点数
FUITOS	无符号整数转换为单精度浮点数
FCMPZHSS	单精度浮点大于等于 0
FCMPZLSS	单精度浮点小于等于 0
FCMPZNES	单精度浮点不等于 0
FCMPZUOS	单精度浮点单操作数是否非数
FCMPHSS	单精度浮点大于等于
FCMPLTS	单精度浮点小于
FCMPNES	单精度浮点不等于
FCMPUOS	单精度浮点双操作数是否非数
FMOVS	单精度浮点数在浮点寄存器间搬运
FABSS	单精度浮点数取绝对值
FNEGS	单精度浮点数取反
FADDS	单精度浮点数加法
FSUBS	单精度浮点数减法
FMULS	单精度浮点数乘法
FNMULS	单精度浮点数乘取反
FMACS	单精度浮点数乘累加
FMSCS	单精度浮点数乘累减
FNMACS	单精度浮点数乘取反累加
FNMSCS	单精度浮点数乘取反累减
FDIVS	单精度浮点数除法
FSQRTS	单精度浮点数取平方根
FRECIPS	单精度浮点数取倒数

图表 6-2 双精度数据运算指令

FDTOSI	**双精度浮点数转换为有符号整数**
FDTOUI	双精度浮点数转换为无符号整数
FSITOD	有符号整数转换为双精度浮点数
FUITOD	无符号整数转换为双精度浮点数
FDTOS	双精度浮点数转换为单精度浮点数
FSTOD	单精度浮点数转换为双精度浮点数
FCMPZHSD	双精度浮点大于等于 0
FCMPZLSD	双精度浮点小于等于 0
FCMPZNED	双精度浮点不等于 0
FCMPZUOD	双精度浮点单操作数是否非数
FCMPHSD	双精度浮点大于等于
FCMPLTD	双精度浮点小于
FCMPNED	双精度浮点不等于
FCMPUOD	双精度浮点双操作数是否非数
FMOVD	双精度浮点数在浮点寄存器间搬运
FABSD	双精度浮点数取绝对值
FNEGD	双精度浮点数取反
FADDD	双精度浮点数加法
FSUBD	双精度浮点数减法
FMULD	双精度浮点数乘法
FNMULD	双精度浮点数乘取反
FMACD	双精度浮点数乘累加
FMSCD	双精度浮点数乘累减
FNMACD	双精度浮点数乘取反累加
FNMSCD	双精度浮点数乘取反累减
FDIVD	双精度浮点数除法
FSQRTD	双精度浮点数取平方根
FRECIPD	双精度浮点数取倒数

图表 6-3 SIMD 数据运算指令

FMOVM	**单精度浮点数在浮点寄存器间搬运，SIMD 指令**
FABSM	单精度浮点数取绝对值，SIMD 指令
FNEGM	单精度浮点数取反，SIMD 指令
FADDM	单精度浮点数加法，SIMD 指令
FSUBM	单精度浮点数减法，SIMD 指令
FMULM	单精度浮点数乘法，SIMD 指令
FNMULM	单精度浮点数乘取反，SIMD 指令
FMACM	单精度浮点数乘累加，SIMD 指令
FMSCM	单精度浮点数乘累减，SIMD 指令
FNMACM	单精度浮点数乘取反累加，SIMD 指令
FNMSCM	单精度浮点数乘取反累减，SIMD 指令

6.2.2 传输类指令

图表 6-4 数据传输指令

FMTVRL	**浮点寄存器低字写传送指令**
FMTVRH	浮点寄存器高字写传送指令
FMFVRL	浮点寄存器低字读传送指令
FMFVRH	浮点寄存器高字读传送指令

6.2.3 内存存取指令

图表 6-5 内存存取指令

FLDS	单精度浮点加载指令
FLDD	双精度浮点加载指令
FLDM	矢量浮点加载指令
FLDRS	寄存器移位寻址单精度浮点加载指令
FLDRD	寄存器移位寻址双精度浮点加载指令
FLDRM	寄存器移位寻址矢量浮点加载指令
FLDMS	连续单精度浮点加载指令
FLDMD	连续双精度浮点加载指令
FLDMM	连续矢量浮点加载指令
FLRWS	单精度浮点立即数存储器读入指令
FLRWD	双精度浮点立即数存储器读入指令
FSTS	单精度浮点存储指令
FSTD	双精度浮点存储指令
FSTM	矢量浮点存储指令
FSTRS	寄存器移位寻址单精度浮点存储指令
FSTRD	寄存器移位寻址双精度浮点存储指令
FSTRM	寄存器移位寻址矢量浮点存储指令
FSTMS	连续单精度浮点存储指令
FSTMD	连续双精度浮点存储指令
FSTMM	连续矢量浮点存储指令

6.3 浮点指令执行延时

R807 浮点单元对指令的执行延时进行了严格的设计，在满足高性能和高频率的设计需求同时，减少浮点指令执行延时。软件开发人员可以利用指令延时信息对软件进行调整优化，提高浮点程序执行效率。

图表 6-6 浮点指令执行延时

浮点指令	指令含义	执行延时	备注
FSTOSI	单精度浮点数转换为有符号整数	2	•
FSTOUI	单精度浮点数转换为无符号整数	2	•
FSITOS	有符号整数转换为单精度浮点数	2	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FUITOS	无符号整数转换为单精度浮点数	2	•
FCMPZHSS	单精度浮点大于等于 0	2	•
FCMPZLSS	单精度浮点小于等于 0	2	•
FCMPZNES	单精度浮点不等于 0	2	•
FCMPZUOS	单精度浮点单操作数是否非数	2	•
FCMPHSS	单精度浮点大于等于	2	•
FCMPLTS	单精度浮点小于	2	•
FCMPNES	单精度浮点不等于	2	•
FCMPUOS	单精度浮点双操作数是否非数	2	•
FMOVS	单精度浮点数在浮点寄存器间搬运	1	•
FABSS	单精度浮点数取绝对值	1	•
FNEGS	单精度浮点数取反	1	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FADDS	单精度浮点数加法	2	•
FSUBS	单精度浮点数减法	2	•
FMULS	单精度浮点数乘法	3	•
FNMULS	单精度浮点数乘取反	3	•
FMACS	单精度浮点数乘累加	5	•
FMSCS	单精度浮点数乘累减	5	•
FNMACS	单精度浮点数乘取反累加	5	•
FNMSCS	单精度浮点数乘取反累减	5	•
FDIVS	单精度浮点数除法	4-14	•
FSQRTS	单精度浮点数取平方根	4-14	•
FRECIPS	单精度浮点数取倒数	4-14	•
FDTOSI	双精度浮点数转换为有符号整数	2	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FDTOUI	双精度浮点数转换为无符号整数	2	•
FSITOD	有符号整数转换为双精度浮点数	2	•
FUITOD	无符号整数转换为双精度浮点数	2	•
FDTOS	双精度浮点数转换为单精度浮点数	2	•
FSTOD	单精度浮点数转换为双精度浮点数	2	•
FCMPZHSD	双精度浮点大于等于 0	2	•
FCMPZLSD	双精度浮点小于等于 0	2	•
FCMPZNED	双精度浮点不等于 0	2	•
FCMPZUOD	双精度浮点单操作数是否非数	2	•
FCMPHSD	双精度浮点大于等于	2	•
FCMPLTD	双精度浮点小于	2	•
FCMPNED	双精度浮点不等于	2	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FCMPUOD	双精度浮点双操作数是否非数	2	•
FMOVD	双精度浮点数在浮点寄存器间搬运	1	•
FABSD	双精度浮点数取绝对值	1	•
FNEGD	双精度浮点数取反	1	•
FADDD	双精度浮点数加法	3	•
FSUBD	双精度浮点数减法	3	•
FMULD	双精度浮点数乘法	4	•
FNMULD	双精度浮点数乘取反	4	•
FMACD	双精度浮点数乘累加	7	•
FMSCD	双精度浮点数乘累减	7	•
FNMACD	双精度浮点数乘取反累加	7	•
FNMSCD	双精度浮点数乘取反累减	7	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FDIVD	双精度浮点数除法	4-29	•
FSQRTD	双精度浮点数取平方根	4-29	•
FRECIPD	双精度浮点数取倒数	4-29	•
FMOVM	单精度浮点数在浮点寄存器间搬运, SIMD 指令	1	•
FABSM	单精度浮点数取绝对值, SIMD 指令	1	•
FNEGM	单精度浮点数取反, SIMD 指令	1	•
FADDM	单精度浮点数加法, SIMD 指令	2	•
FSUBM	单精度浮点数减法, SIMD 指令	2	•
FMULM	单精度浮点数乘法, SIMD 指令	4	•
FNMULM	单精度浮点数乘取反, SIMD 指令	4	•
FMACM	单精度浮点数乘累加, SIMD 指令	6	•
FMSCM	单精度浮点数乘累减, SIMD 指令	6	•

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FNMACM	单精度浮点数乘取反累加，SIMD 指令	6	•
FNMSCM	单精度浮点数乘取反累减，SIMD 指令	6	•
FMTVRL	浮点寄存器低字写传送指令	1	•
FMTVRH	浮点寄存器高字写传送指令	1	•
FMFVRL	浮点寄存器低字读传送指令	1	•
FMFVRH	浮点寄存器高字读传送指令	1	•
FLDS	单精度浮点加载指令	>=3	1 周期 TCM 命中：3-个周期； 2 周期 TC M 命中：4 个周期。 D-Cach e 命中：3 个周期。 D-Cache 缺失：片外访问延迟。
FLDD	双精度浮点加载指令	>=4	拆分为 2 条 FLDS 指令。
FLDM	矢量浮点加载指令	>=4	拆分为 2 条 FLDS 指令。
FLDRS	寄存器移位寻址单精度浮点加载指令	>=4	拆分为 1 条 IX W 和 1 条 FLDS 指令。
FLDRD	寄存器移位寻址双精度浮点加载指令	>=5	拆分为 2 条 IX W 和 2 条 FLDS 指令。
FLDRM	寄存器移位寻址矢量浮点加载指令	>=5	拆分为 2 条 IX W 和 2 条 FLDS 指令。
FLDMS	连续单精度浮点加载指令	>=3+N-1	拆分为 N 条 FLDS 指令。
FLDMD	连续双精度浮点加载指令	>=3+2*N-1	拆分为 2*N 条 FLDS 指令。

下页继续

表 6.1 – 续上页

浮点指令	指令含义	执行延时	备注
FLDMM	连续矢量浮点加载指令	>=3+2*N-1	拆分为 2*N 条 FLDS 指令。
FLRWS	单精度浮点立即数存储器读入指令	>=3	1 周期 TC M 命中: 3 个周期; 2 周期 TC M 命中: 4 个周期。 D-Cach e 命中: 3 个周期。 D-Cache 缺失:片外访问延迟。
FLRWD	双精度浮点立即数存储器读入指令	>=4	拆分为 2 条 FLRWS 指令。
FSTS	单精度浮点存储指令	>=2	•
FSTD	双精度浮点存储指令	>=3	拆分为 2 条 FSTS 指令。
FSTM	矢量浮点存储指令	>=3	拆分为 2 条 FSTS 指令。
FSTRS	寄存器移位寻址单精度浮点存储指令	>=3	拆分为 1 条 IX W 和 1 条 FSTS 指令。
FSTRD	寄存器移位寻址双精度浮点存储指令	>=4	拆分为 2 条 IX W 和 2 条 FSTS 指令。
FSTRM	寄存器移位寻址矢量浮点存储指令	>=4	拆分为 2 条 IX W 和 2 条 FSTS 指令。
FSTMS	连续单精度浮点存储指令	>=2+N-1	拆分为 N 条 FSTS 指令。
FSTMD	连续双精度浮点存储指令	>=2+2*N-1	拆分为 2*N 条 FSTS 指令。
FSTMM	连续矢量浮点存储指令	>=2+2*N-1	拆分为 2*N 条 FSTS 指令。

第七章　内存保护单元

此章节描述了内存保护单元，它包括了如下几个部分：

- 简介
- 内存属性
- 控制寄存器
- 内存访问处理
- MPU 访问异常
- 编程示例

7.1　简介

内存保护单元（MPU）能对 L1 内存系统及片外系统进行访问控制，同时能够对除 TCM 模块外内存系统赋予其访问属性。

该内存保护单元能将内存切分成多片区域，并为每一片区域赋予各自的属性。在 R807 上，可最大支持 12 或 16 片区域。

每一片区域需被设置一个基地址和一个区域大小，通过保护区控制寄存器（PACR，CR<20,0>）和保护区选择寄存器 (PRSR，CR<21,0>) 可对每一块区域基地址和大小进行设定。每片区域的基地址和区域大小所组成的地址空间可以相互交叠。当某地址在多个区域交叠时，按照区域号最大的区域的属性分配。即区域号愈大，优先级愈高。

内存保护单元通过第 0 号控制寄存器组中的第 16,19,20,21,26,27 号寄存器进行控制。详见控制寄存器。

初始化时，内存保护单元默认为未开启状态，需通过 CCR 控制寄存器 (第 0 组第 18 号) 将其打开。否则，所有地址的属性默认为不可缓存 (non-bufferable)，不可高缓 (non-cacheable)，非安全 (non-secure)，强按序 (strong order)。

7.2　内存属性

R807 内存属性分为两大类别。

- 读写访问权限相关属性；
- 内存地址类型相关属性。

7.2.1 读写访问权限属性

每一片区间可配有访问权限和不可执行（NX）这两种访问权限属性，并按照区域号不同分别位于访问权限配置寄存器 0（CAPR，CR<19,0>）和访问权限配置寄存器 1（CAPR1，CR<16,0>）上。

对于内存读写访问请求来说，如普通存储（ST）和读取（LD）指令，只需根据访问权限属性（CAPR 上的 AP 位）以及当前 CPU 所出权限态即可确认。访问权限类别如图表 7-1。

AP[1:0]	超级用户权限	普通用户权限
00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

图表 7-1 访问权限设置

对于取指来说，需在上述 AP 位满足读权限的情况下，还需满足可执行的权限。即对应区域的 NX 位为 0 时为允许取指访问，为 1 时阻止取指访问。

7.2.2 内存地址类型属性

R807 控制两类地址，一类为 L1 内存，可通过读取存储模块直接在核内进行访问；一类为核外内存，需通过特定总线协议与核外系统交互获得数据。

在地址类别上共有可缓存（bufferable），可高缓（cacheable），强按序（strong order）以及安全（secure）四大类。细分地址类别有以下 12 种：

- Bufferable, cacheable, non-strong order, secure
- Bufferable, cacheable, non-strong order, non secure
- Bufferable, non-cacheable, non-strong order, secure
- Bufferable, non-cacheable, non-strong order, non secure
- Non-bufferable, cacheable, non-strong order, secure
- Non-bufferable, cacheable, non-strong order, non secure
- Non-bufferable, non-cacheable, non-strong order, secure
- Non-bufferable, non-cacheable, non-strong order, non secure
- Bufferable, non-cacheable, strong order, secure

- Bufferable, non-cacheable, strong order, non secure
- Non-bufferable, non-cacheable, strong order, secure
- Non-bufferable, non-cacheable, strong order, non secure

注解: 注：如果用户同时在软件层面通过 MTCR 指令对一片内存访问区域写入可高缓和强按序，则按照不可高缓，强按序内存类型来处理。

可缓存属性，按照 B 位（Bufferable）进行标记，对于每个内存访问区域，该信息存于物理内存属性配置寄存器 0/1(ATTR0~1, CR<26,0>/CR<27,0>) 中，位于每一片属性块的 ATTR[2] 位置。

该属性应用于核外内存访问上。若标记某地址区域为可缓存，则在片外访问该区域地址时可缓存该请求，亦可从总线中间节点返回结果（请求无需到达该地址的最终目标 IP）。若设置为不可缓存（non-bufferable），所有相关总线请求必须从目标 IP 获取结果。

强按序属性，按照 SO 位（Strong Order）进行标记。对于每个内存访问区域，该信息存于物理内存属性配置寄存器 0/1(ATTR0~1,CR<26,0>/CR<27,0>) 中，位于每一片属性块的 ATTR[1] 位置。若某片区域被标记了 SO 位，则其一定为非高缓。即每一次访问必须经过核外总线请求才能获取结果。

同时，根据此标记，R807 将核外地址再分为两大类：

- 强按序地址
- 普通内存地址

对于强按序的地址区域，所有的读写请求必须按序发送，按序接收。因而若在 AXI 总线上，该类访存请求的读写 ID 各只有一个。

对于普通内存地址区域，所有的读写请求可乱序发送，乱序接收。在传输效率上更高，但在总线上的先后顺序不受控制。

强按序地址请求和普通内存地址请求相互不影响。

可高缓属性，按照 Cacheable 位进行标记。对于每个内存访问区域，该信息存于物理内存属性配置寄存器 0/1(ATTR0~1, CR<26,0>/CR<27,0>) 中，位于每一片属性块的 ATTR[0] 位置。

该属性用于确认当访问该片区域的地址时，是否先访问 L1 系统内存，如 L1 Cache，再去访问核外内存。

安全属性，按照 S 位（Secure）来进行标记。对于每个内存访问区域，该信息存于访问权限配置寄存器 0（CAPR，CR<19,0>）和访问权限配置寄存器 1（CAPR1，CR<16,0>）中。用来判断该片区域是否具备安全属性。

7.3 控制寄存器

R807 支持可配的内存保护单元 MPU，可以配置 12 或 16 个内存保护区域。在硬件没有配置 MPU 或者 MPU 尚未使能时，所有地址都是可任意访问的，地址的属性默认为 non-bufferable，non-cacheable，

non-secure，strong order。在 MPU 使能后，所有取指、数据访问都需要经过 MPU 获取权限信息和属性信息。

注解： 注：命中 TCM 区域的访问仅看 MPU 返回的访问权限信息（AP 和 NX）来决定是否可访。其属性位将被忽视。命中快速外设地址区域则始终认为非缓存（non-cacheable）

为支持 12/16 个区域的 MPU 配置，R807 新增 CPAR1 控制寄存器。此外，PRSR（CR<21,0>）的保护区索引 RID 也扩展至 4 位。

7.3.1 访问权限配置寄存器 0（CAPR，CR<19,0>）

CAPR 的各位如下图所示：

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	S7	S6	S5	S4	S3	S2	S1	S0	AP7		AP6		AP5		AP4	
Reset	0	0	0	0	0	0	0	0	0		0		0		0	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AP3		AP2		AP1		AP0		NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
Reset	0		0		0		0		0	0	0	0	0	0	0	0

图 7.1: 访问权限配置寄存器 0

NX0~NX7-不可执行属性设置位：

当 NX 为 0 时，该区为可执行区；

当 NX 为 1 时，该区为不可执行区。

注：当处理器取指访问到不可执行的区域时，会出现访问错误异常。

S0~S7-安全属性设置位：

- 当 S 为 0 时，该区为非安全区；
- 当 S 为 1 时，该区为安全区。

AP0~AP7-访问权限设置位：（指令和数据访问均需参考此项）

AP[1:0]	超级用户权限	普通用户权限
00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

图表 7-3 访问权限设置

7.3.2 访问权限配置寄存器 1（CAPR1，CR<16,0>）

CAPR1 的各位如下图所示。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	S15	S14	S13	S12	S11	S10	S9	S8	AP15		AP14		AP13		AP12	
Reset	0	0	0	0	0	0	0	0	0		0		0		0	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AP11		AP10		AP9		AP8		NX15	NX14	NX13	NX12	NX11	NX10	NX9	NX8
Reset	0		0		0		0		0	0	0	0	0	0	0	0

图 7.2: 访问权限配置寄存器 1

NX8~NX15-不可执行属性设置位：

当 NX 为 0 时，该区为可执行区；

当 NX 为 1 时，该区为不可执行区。

注：当处理器取指访问到不可执行的区域时，会出现访问错误异常。

S8~S15-安全属性设置位：

- 当 S 为 0 时，该区为非安全区；
- 当 S 为 1 时，该区为安全区。

AP8~AP15-访问权限设置位：

AP[1:0]	超级用户权限	普通用户权限
00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

图表 7-5 访问权限设置

7.3.3 保护区控制寄存器（PACR，CR<20,0>）

PACR 的各位如下图所示。

Base Address-保护区地址的基地址：

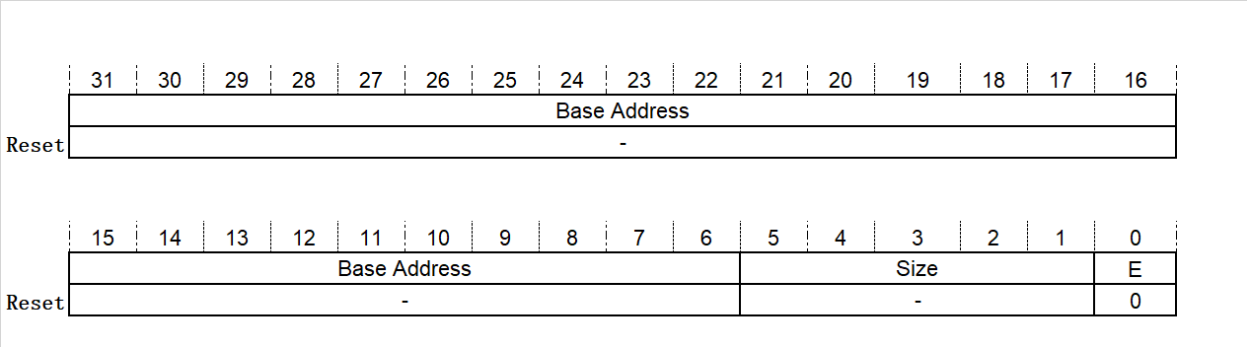


图 7.3: 保护区控制寄存器图

该寄存器指出了保护区地址的基地址，但写入的基地址必须与设置的页面大小对齐，例如设置页面大小为 8M，CR<20,0>[22:12] 必须为 0，各页面的具体要求见图表 7-7。

Size-保护区大小：

保护区大小从 128byte 到 4GB，它可以通过公式：保护区大小= 2^（Size+1）计算得到。Size 可设置的范围为 5’b00110 到 5’b11111，除此以外的数值都会造成不可预测的结果。

Size	保护区大小	对基地址的要求
00000—01010	保留	-
00110	128B	CR<20,0>.bit[6]=0
00111	256B	CR<20,0>.bit[7:6]=0
01000	512B	CR<20,0>.bit[8:6]=0
01001	1KB	CR<20,0>.bit[9:6]=0
01010	2KB	CR<20,0>.bit[10:6]=0
01011	4KB	CR<20,0>.bit[11:6]=0
01100	8KB	CR<20,0>.bit[12:6]=0
01101	16KB	CR<20,0>.bit[13:6] =0
01110	32KB	CR<20,0>.bit[14:6] =0
01111	64KB	CR<20,0>.bit[15:6] =0
10000	128KB	CR<20,0>.bit[16:6] =0
10001	256KB	CR<20,0>.bit[17:6] =0
10010	512KB	CR<20,0>.bit[18:6] =0
10011	1MB	CR<20,0>.bit[19:6] =0
10100	2MB	CR<20,0>.bit[20:6] =0
10101	4MB	CR<20,0>.bit[21:6] =0
10110	8MB	CR<20,0>.bit[22:6] =0
10111	16MB	CR<20,0>.bit[23:6] =0
11000	32MB	CR<20,0>.bit[24:6] =0
11001	64MB	CR<20,0>.bit[25:6] =0
11010	128MB	CR<20,0>.bit[26:6] =0
11011	256MB	CR<20,0>.bit[27:6] =0
11100	512MB	CR<20,0>.bit[28:6] =0
11101	1GB	CR<20,0>.bit[29:6] =0
11110	2GB	CR<20,0>.bit[30:6] =0
11111	4GB	CR<20,0>.bit[31:6] =0

图表 7-7 保护区大小配置和其对基址要求

E-保护区有效设置：

- 当 E 为 0 时，保护区无效；
- 当 E 为 1 时，保护区有效。

7.3.4 保护区选择寄存器 (PRSR，CR<21,0>)

PRSR 用来选择当前操作的保护区，其各位如下图所示。

RID-保护区索引值：

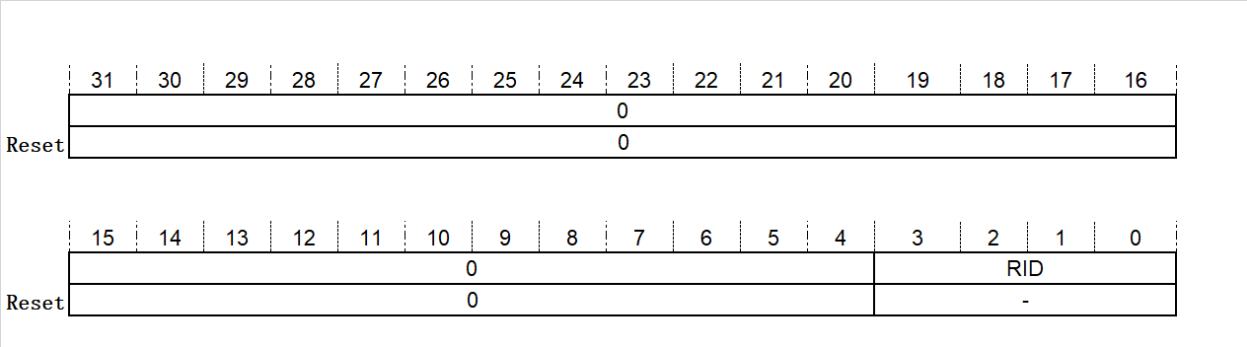


图 7.4: 保护区选择寄存器

RID 表示所选择的对应的保护区，4’b0000~4’b1111 表示第 0~ 第 15 保护区。

该寄存器用来选择对应区域 ID 的 PACR 寄存器。当设定一个 RID 后（通过 MTCR 指令），读写 PACR 寄存器所获得/修改的区域均为 ID 为 RID 的区域。

7.3.5 物理内存属性配置寄存器 0/1 (ATTR0~1, CR<26,0>/CR<27,0>)

MPU 定义了 2 个物理内存属性配置寄存器，用于存储最多 16 个表项的属性信息。因配置不同造成的不存在的表项将无法写入。如 MPU 配置为 12 个表项，则 Attr12~Attr15 将无法被写入。

每个表项占据 4 位信息，Attr[3:0]={reserved, B, SO, C}; 即 {reserved,B(ufferable), S(trong)O(rder), C(acheable)}。每一位属性的性质请看内存地址类型属性。

物理内存属性配置寄存器表示方式如图。

CR<26,0>

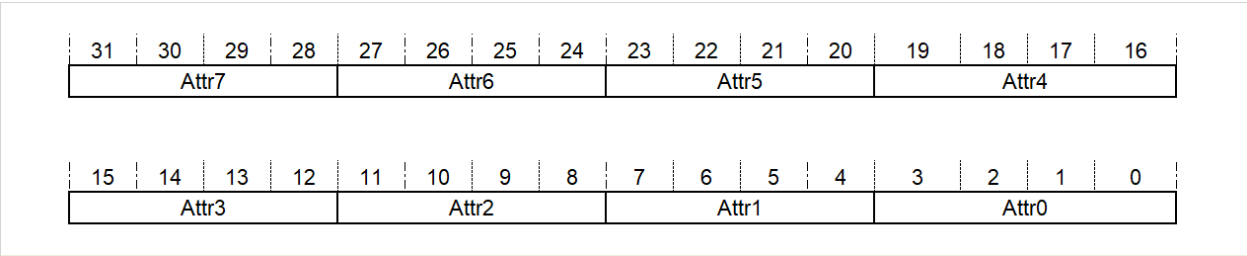


图 7.5: 物理内存属性配置寄存器

CR<27,0>

7.4 内存访问处理

MPU 模块通过 MTCR 及 MFCR 指令来进行设置。出于安全和可靠性考虑，MPU 开启后需要更改区域属性时，一般的建议设置顺序为：

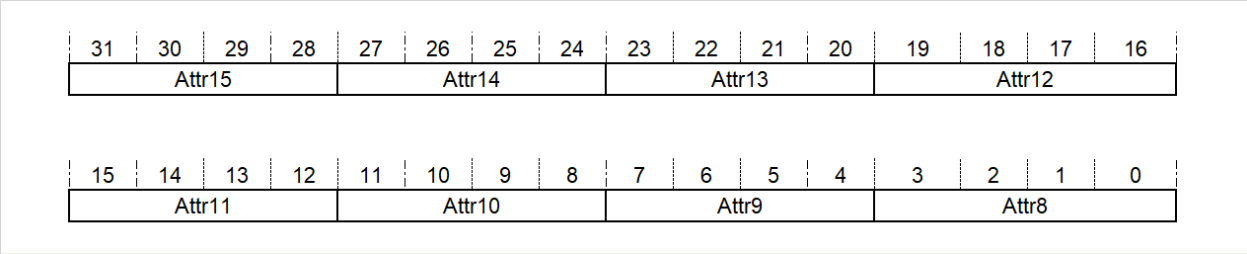


图 7.6: 物理内存属性配置寄存器 0/1

1. 关闭 MPU 模块；
2. 通过 MFCR 指令读取相应控制寄存器并进行值修改操作；
3. 通过 MTCR 指令将该修改值写回原控制寄存器；
4. 打开 MPU 模块；
5. 程序继续运行。

示例可参见编程示例。

此外，考虑到早先的区域设置可能发生改变，包含高缓属性等，意味着 cache 中的状态可能已经不可能符合当前的属性设置。因此，在配置 MPU 前，推荐操作：

1. 清除和无效化 D-Cache；
2. 关闭 I-Cache 和 D-Cache；
3. 无效化 I-Cache。

7.5 MPU 访问异常

若内存访问单元没有开启，则永远不会出 MPU 访问异常。

若内存访问单元已开启，则当 CPU 读写访问某地址时，若未命中任何一片地址区域或命中的最高优先级的地址区域且不满足对应区域的访问权限设置（指令抓取，读操作，写操作），则会产生 MPU 访问异常。

该访问异常对应向量号为 2 号异常，且为精确异常，在该指令退休时将进入对应的 2 号异常服务程序。若配有 ECC 功能，则在 ECC 异常状态寄存器 CR<8,1> 的第 0 位进行置 “1” 操作。

7.6 编程示例

```
/*************************************************************************
* Function: An example of set R807 MPU.
* Enable/disable R807 memory space region.
* Id | Memory Space | Write | Read | Cache | Buffer | So | Nx | Sec |
```



```
* 0 | 0x00000000 ~ 0xFFFFFFFF | Yes | Yes | NO | NO | Yes | Yes | Yes |
* 1 | 0x28000000 ~ 0x29000000 | Yes | Yes | Yes | Yes | NO | NO | NO |
* 2 | 0x28000000 ~ 0x28100000 | No | Yes | Yes | NO | NO | Yes | Yes |
* 3 | 0x28F00000 ~ 0x29000000 | Yes | Yes | No | Yes | Yes | NO | NO |
*
******************************************************************************/
/* Enable/disable every memory area. */
/* Set the access authorization. */
/* Set the cachable or not. */
/* Set the bufferable or not. */
/* Set the strong order or not. */
/* Set the security or not. */
/* Set the executable or not. */
mfcr r10,cr<19,0>
bseti r10,0 //region 0 exe
bclri r10,1 //region 1 exe
bseti r10,2 //region 2 exe
bclri r10,3 //region 3 exe
bseti r10,8 //region 0 AP
bseti r10,9
bseti r10,10 //region 1 AP
bseti r10,11
bclri r10,12 //region 2 AP
bseti r10,13
bseti r10,14 //region 3 AP
bseti r10,15
bseti r10,24 //region 0 AP
bclri r10,25 //region 1 sec
bseti r10,26 //region 2 sec
bclri r10,27 //region 3 sec
```



```
mtcr r10,cr<19,0>
/* Set the cachable or not. */
/* Set the bufferable or not. */
/* Set the strong order or not. */
mfcr r10,cr<26,0>
bclri r10,0 //region 0 ca
bseti r10,1 //region 0 so
bseti r10,2 //region 0 buf
bseti r10,3 //region 1 ca
bclri r10,8 //region 1 so
bclri r10,9 //region 1 buf
bclri r10,10 //region 2 ca
bseti r10,11 //region 2 so
bclri r10,12 //region 2 buf
bseti r10,13 //region 3 ca
bclri r10,14 //region 3 so
bseti r10,24 //region 3 buf
mtcr r10,cr<26,0>
/* The first area (0x00000000 ~ 0xFFFFFFFF) */
movi r10,0x0 //region 0 rid
mtcr r10,cr<21,0>
movi r10,0x3f /* 4G space, Base address: 0x00000000 */
mtcr r10,cr<20,0>
/* The second area (0x28000000 ~ 0x29000000) */
movi r10,1
mtcr r10,cr<21,0>
movih r10,0x2800
ori r10,r10,0x2f /* 16M Space, Base address: 0x28000000 */
mtcr r10,cr<20,0>
/* The third area (0x28000000 ~ 0x28100000) */
```



```
movi r10,2
mtcr r10,cr<21,0>
movih r10,0x2800
ori r10,r10,0x27 /* 1M Space, Base address: 0x28000000 */
mtcr r10,cr<20,0>
/* The fourth area (0x28F00000 ~ 0x29000000) */
movi r10,3
mtcr r10,cr21
movih r10,0x2800
ori r10,r10,0x27 /* 1M Space, Base address: 0x28F00000 */
mtcr r10,cr<20,0>
/* Enable MPU */
mfcr r7, cr<18,0>
bseti r7, 0
bclri r7, 1
mtcr r7, cr<18,0>
```


第八章　存储系统

此章节描述了存储系统，它包括了如下几个部分：

- 简介
- Cache 系统
- TCM 系统
- 内存检错和纠错
- 内存访问异常
- 存储系统访问处理

8.1　简介

储存系统包含以下几个部分：

- 分离的指令高速缓存和数据高速缓存（CACHE）
- 片上紧耦合内存（TCM）
- 内存保护单元

指令和数据均拥有自己的高速缓存。高速缓存的结构为哈佛结构，即指令和数据只能访问各自对应的高速缓存，互不影响。同时，除了高速缓存之外，有两块专为指令和数据访问设计的紧耦合内存，即 TCM。TCM 最多可配置两块，I-TCM 和 D-TCM，也可以选配其一，两块 TCM 均支持指令和数据的访问。

所有的内存访问操作，包括取指和数据访问，访问顺序如下：

- 如果命中 TCM 所配置的区域（对于区域开启使能），则直接访问响应的 TCM 来获取数据。
- 如果命中外设总线（仅适用数据访问），则访问外设总线。
- 如果属性被配置为可高缓，则访问对应 cache，如果 cache 不命中将通过 AXI 总线访问外部存储器。
- 如果属性被配置为不可高缓，则直接通过 AXI 总线访问外部存储器。

TCM 和 CACHE 均能可选的配置 ECC 校验功能来增强 RAM 可靠性。

MPU 相关内容详见上一章节。

8.2 Cache 系统

高速缓存作为支持内存高速访问的主要核心，指令和数据高速缓存可以分别硬件配置成不同的大小。高速缓存的软件配置通过控制寄存器实现，具体见高速缓存控制寄存器。

任何未命中 TCM 及外设总线地址的请求，如果其地址为可高缓属性且 CACHE 开启，则该请求会首先去 CACHE 中查找内容，如果命中，则从 CACHE 中获取或写入数据，如果属性为不可高缓或 CACHE 关闭，则通过 AXIMASTER 接口获取或写入数据。

不管对于数据缓存还是指令缓存，上述所有会访问 CACHE 的读操作总是从 CACHE 中获取数据，如果未命中 CACHE，则都会在 CACHE 中分配一个 cacheline，通过 AXI 总线将所需要的 cacheline 回填到分配的位置，即 RA（read allocate)。对于上述 CACHE 回填，也称为 linefill，总是以 critical word first 为原则，即总线总是先返回包含所需 word 的数据，减少指令获取对应数据的延迟。同理，当总线发生 error 时，只有包含 critical word 的当拍数据出现 error，才会被指令或处理器感知，并触发异常。

而对于数据缓存，由于存在写内存的需要，当发生未命中 CACHE 的情况时，也可以像上述 RA 一样，先在 CACHE 中分配 cacheline，通过 AXI 总线回填，然后再将需要写入的数据写入 CACHE 中。类比 RA，这种处理方式称为 WA（write allocate), 这是一种可选配置，需要通过控制寄存器开启。如果不开启 WA，当 CACHE 缺失时，不再进行 CACHE 回填，而是直接通过 AXI MASTER 总线向片外写出数据。

众所周知，对于读操作来讲，命中 CACHE 只需要从 CACHE 中获取数据。但对与数据缓存的写操作，当命中 CACHE 时，除了将对应数据写入 CACHE 以外，还可以选择同时将数据写到总线。如果只写到 CACHE 中，则称为 WB(writeback)，同 WA 一样，需要在控制寄存器中开启（推荐开启)。如果不开启，则称为 WT（writethrough), 即除了写 CACHE 以外，同样会将数据写到总线，这种配置下 CACHE 中不会存在脏数据。

不管是指令缓存还是数据缓存，都可以选配 ECC 校验来增强可靠性。任何对 CACHE 的操作都会进行 ECC 校验检测，不管是 tag 还是 data，具体见内存检错和纠错章节。

8.2.1 Prefetch 和 AMR

对于 MEM_COPY 等应用场景，如果不想写操作污染 CACHE 中的数据，可以通过开启控制寄存器中的 AMR 开关，达到动态关闭 write allocate 的效果，即在 mem_copy 期间屏蔽 write allocate 使能，mem_copy 结束后又解除屏蔽，重新启用 write allocate。

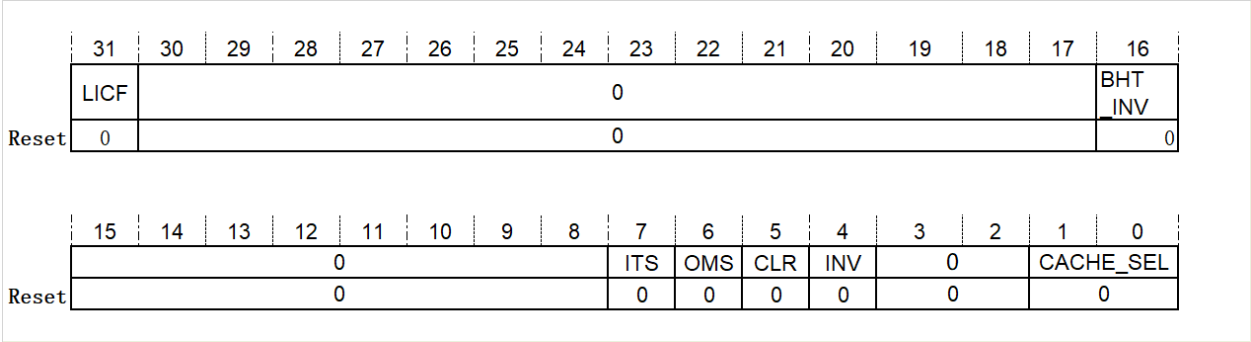
另一方面，对于 mem_copy 等连续地址 cache miss 的场景下，可以通过控制寄存器启用数据缓存的 prefetch 功能，达到提前预取总线数据，减小内存访问带来的延时。

8.2.2 高速缓存控制寄存器

8.2.2.1 高速缓存功能设置寄存器（CFR，CR<17,0>）

高速缓存功能设置寄存器用来控制高速缓存，让相应高速缓存内的数据全部无效。

LICF –Cache Line INV/CLR 失败状态位：



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LICF	0														BHT_INV
Reset	0	0														0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0								ITS	OMS	CLR	INV	0		CACHE_SEL	
Reset	0								0	0	0	0	0		0	

图 8.1: 高速缓存功能设置寄存器

当一个使用虚拟地址作为索引的 Cache line INV/CLR 操作，在虚拟地址转换的过程中遇到 TLB MISS/TLB FATAL/TLB READ INV/ACCESS ERR 等异常时，此位将被置起。

OMS–Cache INV/CLR 模式位：

当 OMS 为 0 时，cache 的 CLR 和 INV 操作将对 cache 的所有 line 起作用。

当 OMS 为 1 时，cache 的 CLR 和 INV 操作将对 cache 的一个 line 起作用。并且使用 CIR 作为 cache INV/CLR 操作的索引。

ITS–Cache INV/CLR 模式位：

当 ITS 为 0 时，CIR 将被用于虚拟地址索引。

当 ITS 为 1 时，CIR 将被用于 SET/WAY/LEVEL 索引。

BTB_INV-BTB 无效设置位：

当 BTB_INV 为 1 时，分支目标缓冲器内的数据将失效。

BHT_INV-BHT 无效设置位：

当 BHT_INV 为 1 时，分支历史表内的数据将失效。

CLR-脏表项清除设置位：

当 CLR 为 1 时，高速缓存内的被标记为脏的表项中的数据将被写出到片外内存中。

INV-无效设置位：

当 INV 为 1 时，高速缓存内的数据将失效。

CACHE_SEL-高速缓存选择位：

当 SEL 为 01 时，选中指令高速缓存；

当 SEL 为 10 时，选中数据高速缓存；

当 SEL 为 11 时，选中数据和指令高速缓存

8.2.2.2 高速缓存配置寄存器（CCR，CR<18,0>）

高速缓存配置寄存器用来配置内存，高速缓存有效 / 无效，内存保护区，Endian 模式，以及系统和处理器的时钟比。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	IPE															
Reset	0															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	IPE	BSTE	E_V2	WA	0	SCK			BE	Z	RS	WB	DE	IE	MP	
Reset	0	0	0	0	0	0			0	0	0	0	0	0	0	

图 8.2: 高速缓存配置寄存器

IPE-间接分支跳转预测使能位：

当 IPE 为 0 时，间接分支跳转预测关闭；

当 IPE 为 1 时，间接分支跳转预测开启。

BSTE-写突发传输使能位：

当 BSTE 为 0 时，不支持写突发传输；

当 BSTE 为 1 时，支持写突发传输。

E_V2-大小端版本选择位：

当 E_V2 为 0 时，支持 V1 版本大小端模式；

当 E_V2 为 1 时，支持 V2 版本大小端模式。

WA-写分配使能位：

该位有效时，处理器执行 ST 指令且数据缓冲器缺失时，在缓冲器中分配表项。

SCK-系统和处理器的时钟比：

该位用来表示系统和处理器的时钟比，其计算公式为：时钟比＝ SCK ＋ 1，CPU 上有对应引脚引出。SCK 在 reset 时被配置且不能在之后改变。

BE-Endian 模式：

当 BE 为 0 时，Little endian；

当 BE 为 1 时，Big endian；

BE 在 reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

Z-允许预测跳转设置位：

当 Z 为 0 时，预测跳转关闭；

当 Z 为 1 时，预测跳转开启。

RS-地址返回栈设置位：

当 RS 为 0 时，返回栈关闭；

当 RS 为 1 时，返回栈开启。

WB-高速缓存写回设置位：

当 WB 为 0 时，数据高速缓存为写直模式；

当 WB 为 1 时，数据高速缓存为写回模式。

DE-数据高速缓存设置位：

当 DE 为 0 时，数据高速缓存关闭；

当 DE 为 1 时，数据高速缓存开启。

IE-指令高速缓存设置位：

当 IE 为 0 时，指令高速缓存关闭；

当 IE 为 1 时，指令高速缓存开启。

MP-内存保护设置位：

MP 用来设置 MPU 是否有效，如下表：

MP	功能
00	MPU 无效
01	MPU 有效
10	MPU 无效
11	MPU 有效

图表 8-3 R807 内存保护设置

8.2.2.3 高速缓存索引寄存器（CIR，CR<22,0>）

当 CFR 的 ITS 被配置为 0 时，CIR 的各个位如下

当 CFR 的 ITS 被配置为 1 时，CIR 的各个位如下

其中，A/B/L 由不同的高速缓存自身属性决定。其中：

A = Log2(ASSOCIATIVITY)

B = (L + S)

L = Log2(LINELEN)

S = Log2(SETS)

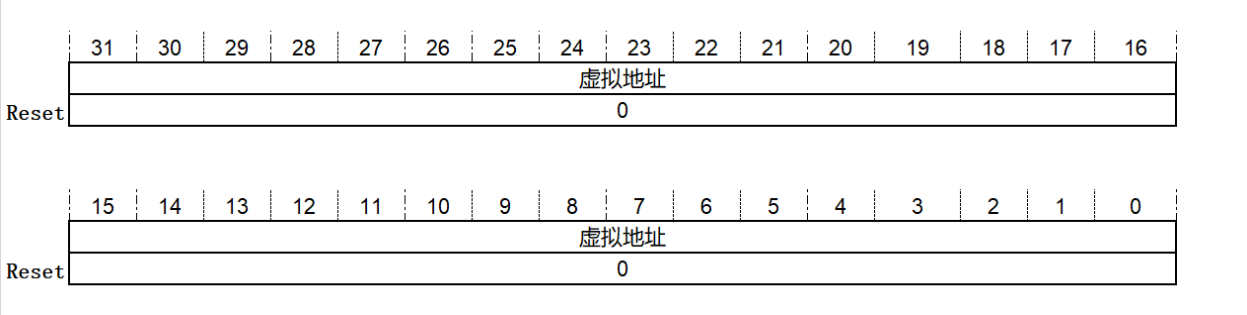


图 8.3: 高速缓存索引寄存器-1

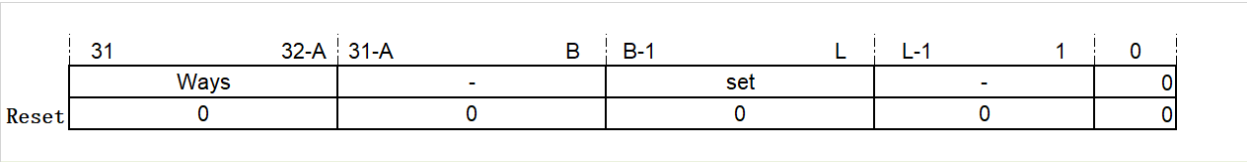


图 8.4: 高速缓存索引寄存器-2

Set：指示操作的块位于高速缓存第几组

Way：指示操作的块位于高速缓存第几路

LINELEN：块的长度

ASSOCIATIVITY：关联度

8.3 TCM 系统

R807 支持硬件可配片上核内紧耦合内存 I-TCM/D-TCM，用于存放对实时性有严格要求的指令和数据。CPU 的取指请求和数据请求均可访问 I-TCM/D-TCM。TCM 系统还支持硬件一组 AXI 64 位的 slave 接口，用于 DMA 等其他系统 master 访问 TCM 进行数据交互。当出现 CPU 取指访问、CPU 数据访问以及 TCM slave 访问冲突时，TCM 的访问优先级为 CPU 数据访问 >CPU 取指访问 >TCM slave 访问。此外，D-TCM 可分为两片可同时访问的 bank0 和 bank1，能够根据需要按照地址次高位或者低位 interleave 的方式划分 bank0 和 bank1。这种灵活的分 bank 方式，能够支持 CPU 和 DMA 分别访问其中一块 bank 而不产生冲突，从而提高 TCM 的使用效率。

软件可通过读写 TCM 相关控制寄存器 CR<22,1> 和 CR<23,1> 来使能 I-TCM/D-TCM 以及配置其地址空间，查询 TCM 的尺寸和访问延时。此外，控制寄存器 CR<22,1> 和 CR<23,1> 还分别包含一位 TCM slave 访问使能位 SIF 来允许或者禁止 TCM slave 向相应的 I-TCM/D-TCM 发起访问。若相应的使能位关闭，则任何来自 TCM slave 的访问请求都将返回访问 error。

8.3.1 TCM 属性和权限

核内所有针对 TCM 的访问，包括 IFU 和 LSU，都需要经过 MPU 的权限检查，如果发生 MPU 访问错误，则无法访问 TCM, 触发异常。

TCM 永远表征为不可高缓，不可共享的普通内存属性（非 strong order），无视 MPU 上配置的属性。即 MPU 地址区域配置不会对 TCM 的属性造成影响，仅影响权限检查。

8.3.2 TCM 可配选项

TCM 支持多种可配选项，可选配其一或全部配置：

- I-TCM
- D-TCM
- TCM AXI SLAVE

其中 D-TCM 分为两个 bank，可根据需要按照 interleave（bit-3）或者 D-TCM 基地址的次高位 (按照不同 size) 区分两个 bank。

I-TCM/D-TCM 可以根据需要配置成不同的 size 大小，且根据时序要求配置 1 cycle 或者 2 cycle 访问（软件可见但无法改写）。

8.3.3 TCM AXI slave

R807 推荐配置 64 位 AXI TCM slave 用于支持 AXI 总线对于 TCM 的访问。如前述所示，通过 AXI slave 访问 TCM 受到控制寄存器 SIF 影响，而不会收到 MPU 影响，slave 访问 TCM 的优先级低于核内 LSU 和 IFU。

对 TCM slave 而言，tcm 地址可根据 SOC 需要配置为和核内 TCM 不同的地址，由额外的外部信号线 sel 决定，便于 soc 更灵活地配置 TCM。TCM slave 不支持所有类型的 AXI 总线传输，具体支持类型见 AXI 总线从接口章节。

8.3.4 TCM 控制寄存器

8.3.4.1 I-TCM 配置寄存器（I-TCMCR，CR<22,1>）

此寄存器用以描述 I-TCM 的相关控制信息以及物理地址基址（base address）。

对于 CPU 的访问（LSU/IFU/CP0）以及 TCM slave 的访问，I-TCM 有分别的控制使能位进行控制，分别为 CR<22,1>[0] 以及 CR<22,1>[2]。

CR<22,1>[31:12] 表示 CPU 访问 I-TCM 的基址信息，只有 I-TCM 对应的使能位开启且 CPU 的访问地址命中该基址，CPU 才能向 I-TCM 发起访问请求。

复位值：0x0000_00X0

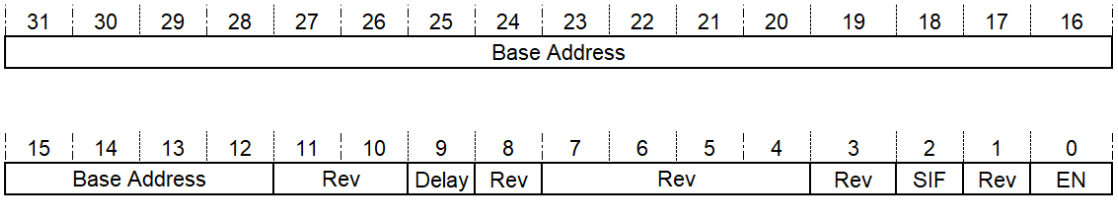


图 8.5: I-TCM 配置寄存器

X 表示 I-TCM 的物理大小

表 8.1: I-TCM 配置寄存器对应表

[31:12]	Base Address	**读写**	**描述 I-TCM 的基址（base address）。此地址为物理地址。用户在配置基址时，需要按照 I-TCM 的起始地址对齐此基址。例如：配置 8KB 大小的 I-TCM，基址必须设置成** 20'b??????????????????0。 **复位值**：0。
[11:10]	Reserved	只读	全零。
[9]	Delay	只读	表示 I-TCM 的访问是 1 个周期还是 2 个周期返回数据。 1'b0：1 周期； 1'b1：2 周期。 复位值：0。
[8]	Reserved	只读	0
[7:4]	Size	只读	表示 I-TCM 大小： 4'b0000~4'b0001 保留 4'b0010 4K (最小) 4'b0011 8K 4'b0100 16K 4'b0101 32K 4'b0110 64K 4'b0111 128K 4'b1000 256K 4'b1001 512K 4'b1010 1MB 4'b1011~4'b1111 保留
[3]	Reserved	只读	0
[2]	SIF	读写	表示 slave 接口请求是否能访问 I-TCM。 1'b0：不可访问； 1'b1：可访问。 复位值：0
[1]	Reserved	只读	0
[0]	EN	读写	I-TCM 使能位，表示 CPU 请求是否能访问 I-TCM： 1'b0：不可访问； 1'b1：可访问。 复位值：0

8.3.4.2 D-TCM 配置寄存器（D-TCMCR，CR<23,1>）

此寄存器用以描述 D-TCM 的相关控制信息以及物理地址基址（base address）。

对于 CPU 的访问（LSU/IFU/CP0）以及 TCM slave 的访问，D-TCM 有分别的控制使能位进行控制，分别为 CR<23,1>[0] 以及 CR<23,1>[2]。

CR<23,1>[31:12] 表示 CPU 访问 D-TCM 的基址信息，只有 D-TCM 对应的使能位开启且 CPU 的访问地址命中该基址，CPU 才能向 D-TCM 发起访问请求。

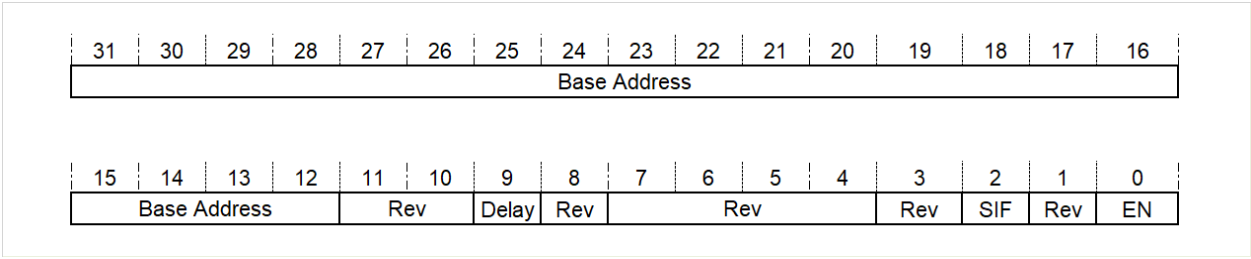


图 8.6: D-TCM 配置寄存器

复位值：0x0000_00X0

X 表示 D-TCM 的物理大小

表 8.2: D-TCM 配置寄存器对应表

[31:12]	Base Address	**读写**	**描述 D-TCM 的基址（base address）。此地址为物理地址。用户在配置基址时，需要按照 D-TCM 的起始地址对齐此基址。例如：配置 8KB 大小的 D-TCM，基址必须设置成** 20’b????????????????????0。 **复位值**：0
[11:10]	Reserved	只读	全零。
[9]	Delay	只读	表示 D-TCM 的访问是 1 个周期还是 2 个周期返回数据。 1’b0：1 周期； 1’b1：2 周期。 复位值：0
[8]	Reserved	只读	0
[7:4]	Size	只读	表示 D-TCM 大小： 4’b0000~4’b0001 保留 4’b0010 4K (最小) 4’b0011 8K 4’b0100 16K 4’b0101 32K 4’b0110 64K 4’b0111 128K 4’b1000 256K 4’b1001 512K 4’b1010 1MB 4’b1011~4’b1111 保留
[3]	Reserved	只读	0
[2]	SIF	读写	表示 slave 接口请求是否能访问 D-TCM。 1’b0：不可访问； 1’b1：可访问。 复位值：0
[1]	Reserved	只读	0
[0]	EN	读写	D-TCM 使能位，表示 CPU 请求是否能访问 D-TCM： 1’b0：不可访问； 1’b1：可访问。 复位值：0

8.4 内存检错和纠错

对于 RAM 而言，一些意外的放射性环境或者其他影响可能造成 RAM 中的数据被破坏。R807 支持针对 TCM 和 CACHE RAM 的错误检测和纠正，这些校验位和数据一起存储在 RAM 中。当处理器从 RAM 中读取数据时，会同时检测这些校验位，根据校验结果选择纠正或者将错误上报。

R807 采用硬件可配的 ECC 校验策略，用于对可靠性有需求的场景。ECC 校验支持 SECDED（一比特纠错，两比特检错），并提供 ECC 错误注入功能。

8.4.1 ECC 校验粒度

表 8.3: R807 ECC 校验粒度

RAM	校验方式	校验粒度
I-cache tag	6 比特 ECC 校验	20 位 tag+1 位有效位
I-cache data	8 比特 ECC 校验	64 位数据
D-cache tag	6 比特 ECC 校验	21 位 tag+1 位有效位
D-cache data	7 比特 ECC 校验	32 位数据
D-Cache dirty	1 比特 parity 校验	1 位脏位
I-TCM	8 比特 ECC 校验	64 位数据
D-TCM	7 比特 ECC 校验	32 位数据

8.4.2 部分写操作

开启 ECC 后，对于小于上述 RAM 粒度的写操作，CPU 将会采取 read-merge-write 的方式，即将整个粒度大小的值全部读出，与部分写的数据进行融合拼接，再重新编码，写入 RAM，保证 ECC 编码的正确性。

这种 read-merge-write 的操作不可避免地会降低 CPU 的性能，因此在 ECC 开启时，若需要保证性能吗，则不推荐频繁使用部分写操作。(由于 I-TCM 的校验粒度是 64-bit，LSU 发起的 word 写请求也会被视为部分写)。

8.4.3 ECC 使用和测试

软件可通过写控制寄存器 CR<31,0>[7:6] 来使能 ECC 校验。需要注意的是，未开启 ECC 时，CPU 不保证 ECC 编码的正确性。这是由于任何小于校验粒度的写 RAM 行为都将直接写入 RAM，而对应的 ECC 编码并未更新，导致编码出错。因此如要使用 ECC 功能，推荐在对应 RAM 使能前，先开启 ECC 使能位。

前述已经提到，为了软件测试，R807 提供了 ECC 注入功能，模拟 ECC 出错的场景。需要注意的是，ECC 注入会在下一次 RAM 访问时触发，因此根据实际情况可能会有一定的响应延时。

8.4.4 ECC 信息寄存器

为了提升 R807 的可靠性，R807 的所有片上存储均支持 1-bit 纠错和 2-bit 检错的 ECC 检查（除 dirty 位采用奇偶校验位）。并且，R807 提供丰富的 ECC 信息寄存器，方便软件查询出现 ECC 错误的相关地址和 RAM 信息。

8.4.4.1 ECC 错误信息寄存器（ERRLC，CR<6,1>）

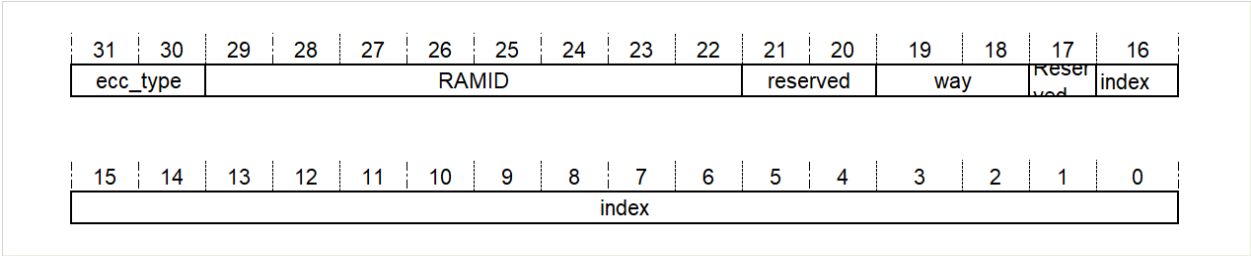


图 8.7: ECC 错误信息寄存器

复位值：0x0

表 8.4: ECC 错误信息寄存器对应表

[16:0]	index	**读写**	RAM index **索引位**: ICACHE DATA: **从** addr[3] **开始**; DACHE DATA: **从** addr[2] **开始**; ICACHE TAG/DCACHE TAG: **从** addr[5] **开始**; TCM: **从** addr[3] **开始**。
[17]	Reserved	只读	0
[19:18]	Way	读写	ICACHE/DCACHE WAY 索引位: 2'b00: way0; 2'b01: way1; 2'b10: way2; 2'b11: way3。
[21:20]	Reserved	只读	0
[29:22]	RAMID	读写	ECC RAM 索引: [29]: ICACHE TAG RAM; [28]: ICACHE DATA RAM; [27]: DCACHE TAG RAM; [26]: DCACHE DATA RAM; [25]: reserved, always 0; [24]: reserved, always 0; [23]: I-TCM RAM; [22]: D-TCM RAM。
[31:30]	ECC_TYPE	读写	ECC 类型: [31]: ECC fatal err; [30]: ECC correctable err ; 2'b00: entry 无效。

8.4.4.2 ECC 错误地址寄存器（ERRADDR，CR<7,1>）

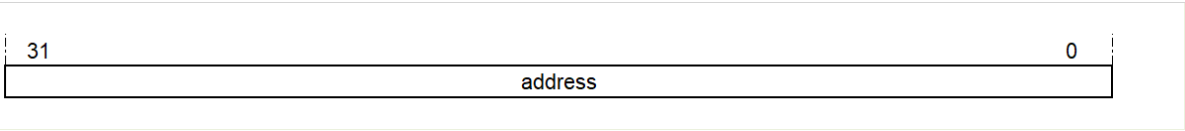


图 8.8: ram 错误地址寄存器

用于同步异常，供软件查询错误地址，地址为 VA。

8.4.4.3 异常状态寄存器（ERRSTS，CR<8,1>）

复位值: 0x0

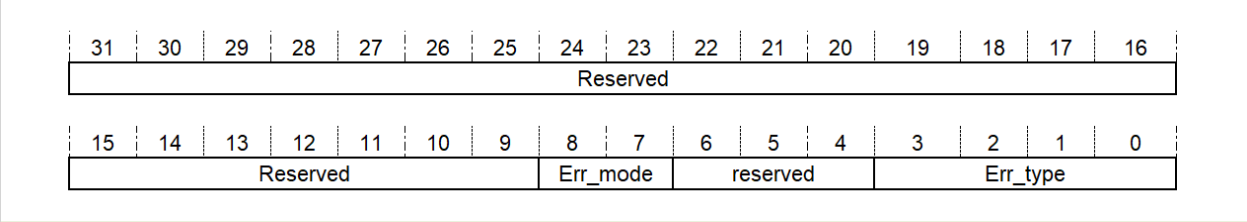


图 8.9: 异常状态寄存器

表 8.5: 异常状态寄存器对应表

[2:0]	Err_type	**读写**	ERR **异常响应类型**: [0]: MPU **访问异常**; [1]: **总线访问错误异常**; [2]: ECC **校验错误异常**。 3'b000: **无异常**。
[7]	Err_mode	读写	ERR 响应方式: 1'b0: 精确响应，所有 IFU 取指读访问和 LSU 读指令访问; 1'b1: 非精确响应，LSU 写指令访问。

硬件自动置位，软件可通过写入 0 的方式清除 ERR_TYPE 为 3'b000

8.4.5 ECC 注入功能寄存器（ERRINJCR，CR<9,1>）

ECC 注入功能用于软件控制硬件自动注入假的 ECC 错误，方便软件进行 ECC 功能调试。软件通过控制寄存器来控制对特定的 SRAM 进行 1-bit 或者 2-bit 的错误注入。

在控制寄存器使能后，一旦 CPU 检测到有读请求（LSU/IFU 发起的访问请求）访问到选定的 SRAM，则对读到的数据进行 1-bit/2-bit 的错误注入。

若是 1-bit 错误或者是可修复的 2-bit 错误（例如 ICACHE），CPU 则按照正常的 ECC 错误处理方式来修复错误并更新 ECC 信息寄存器，软件可通过查询 ECC 信息寄存器来查询错误是否注入成功，也可将控制寄存器 CR<31,0>[7] 开启，则任何检测到的 ECC 错误都会响应 2 号异常。

若是不可修复的 2-bit 错误，CPU 除了按照正常方式报异常外，还会将 2-bit 错误注入 SRAM 中，模拟实际不可修复错误的情况，方便软件测试。

复位值：0x0

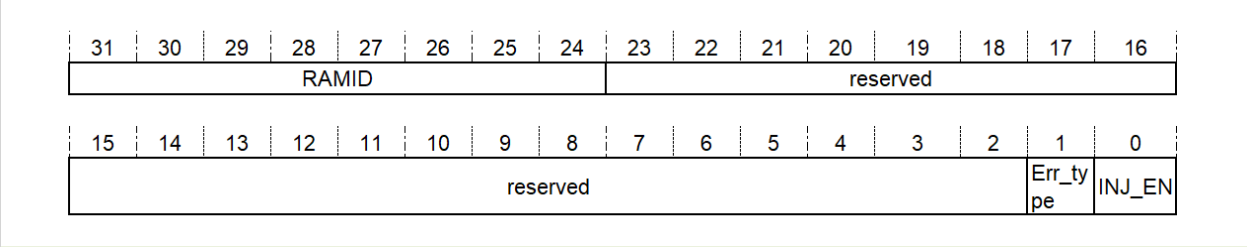


图 8.10: ECC 注入功能寄存器

表 8.6: ECC 注入功能寄存器对应表

[0]	INJ_EN	**读写**	ECC **注入使能位**: 0: ECC **注入关闭**; 1: ECC **注入使能**。
[1]	Err_type	读写	注入 ERR 类型: 0: correctable (1-bit error); 1: fatal(2-bit error)。
[31:24]	RAMID	读写	ECC RAM 索引: [31]: ICACHE TAG RAM; [30]: ICACHE DATA RAM; [29]: DCACHE TAG RAM (包含 DIRTY RAM); [28]: DCACHE DATA RAM; [27]: reserved, always 0; [26]: reserved, always 0; [25]: I-TCM RAM; [24]: D-TCM RAM。

8.5 内存访问异常

内存访问异常分为以下几种情况:

- MPU 访问异常
- 外部总线错误异常
- ECC 检验错误异常

当发生内存访问异常时，可通过查看异常状态寄存器 CR<8,1> 确认属于哪种访问异常。

8.5.1 MPU 访问异常

当内存请求的权限不符合其所访问区域的要求时，会产生 MPU 访问异常，且其优先级判断高于其他两类内存访问异常。

8.5.2 外部总线错误异常

当 CPU 访问外部内存时，由外部总线返回的错误即外部总线错误。对于 R807，所有的取指和加载操作，均会同步产生外部总线访问错误异常。对于所有的存储操作，则会丢弃该总线错误，处理器不会响应异常。

8.5.3 ECC 校验错误异常

ECC 校验错误分为可修复错误（correctable error）和不可修复错误（fatal error）。可修复错误只有在相应的控制寄存器 CRECC 位使能时，才会上报异常，否则处理器会进行自动修复，不会产生异常。而对于不可修复错误，则都需要上报异常。所有的 1-bit 错误以及 I-Cache 上的 2-bit 错误都是可修复错误。

对于 I-Cache 而言，所有的 ECC 校验错误都是可修复的，这是由于所有指令缓存内的缓存行都是可以从外部内存重新获取的，不会被改写，因此所有的校验错误都可以通过 invalid && reload 修复。

对于 D-Cache 而言，所有的 tag 2-bit 错误都是不可修复错误，需要上报异常。但对于 data 而言，出现 2-bit 错误并且该缓存块为脏时才是不可修复错误。如果该缓存块非脏，即未被改写过，则和 I-Cache 一样，都可以通过 invalid && reload 修复。

对于 TCM 而言，由于不存在 tag，所有的 2-bit 错误都是不可修复错误，需要上报异常。

如前文 ECC 的描述所述，只有读取 RAM 内容时才会进行 ECC 检测，大于等于 ECC 粒度的写操作不会进行 ECC 检测，即也不会产生 ECC 校验错误。

ECC 校验错误可能是同步的，也可能是异步的，可以通过错误信息寄存器 CR<6,1> 获取该信息。只有加载指令访问 RAM 时产生的 ECC 校验错误是同步的，其余情况异常都是异步响应的。

8.6 存储系统访问处理

存储系统会根据不同的内存属性进行相应的访问处理：

- 只有配置成可高缓属性的，weak order 的内存属性，当 CACHE 打开时，才能被缓存到 CACHE 中；
- 存储系统只会 merge 非 strong order 属性的写请求；
- TCM 和外设总线区域都会默认为不可高缓属性，但 tcm 默认 weak order, 而外设总线区域则可以根据 MPU 配置成 strong order 或 weak order。
- 对于不可共享的区域, 原子指令只会检查 local monitor，而对于可共享的区域, 原子指令除了检查 local monitor 以外，外部的 external monitor 也会起到检测作用，具体见*内存 Debug 访问*。

8.7 内存 Debug 访问

为了向软件提供更方便的内存访问及调试功能，R807 支持了软件访问内存的功能。通过配置相应的控制寄存器，软件可直接访问 CACHE 以及 TCM，获取特定内存索引的数据和 ECC 信息。具体的访问通过控制寄存器 CR<26,1> 和 CR<31,1> 设置，读取的数据将存放在 CR<27,1>~CR<29,1> 中。

8.7.1 内存访问索引寄存器（CINDEX，CR<26,1>）

内存访问索引寄存器用来提供内存访问的 RAM 信息，可以指定访问 ICACHE/DCACHE 和 I-TCM/D-TCM 等内存，以及指定 RAM 具体的 set/way 信息。在硬件配置 ECC 时，还可将对应 RAM 的 ECC 编码读出。

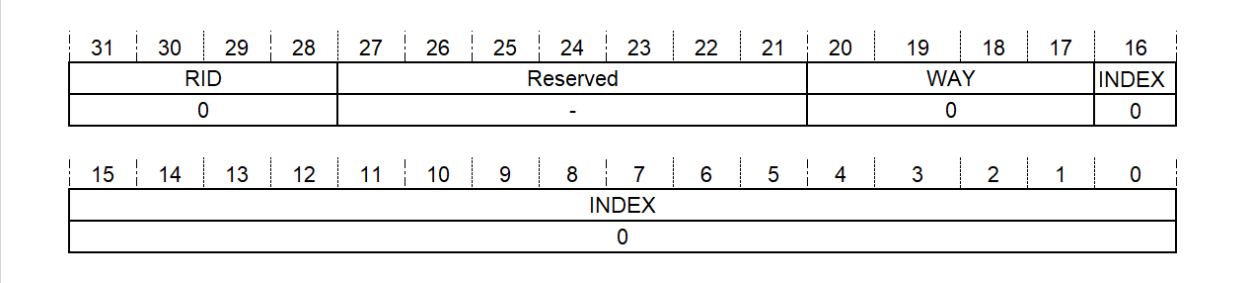


图 8.11: 内存访问索引寄存器

RID-RAM 标识位：

指示访问的 RAM 信息：

当 RID[3:0] 为 0 时，表示访问的是 ICACHE TAG RAM，读取 21-bit TAG 信息以及额外 ECC 编码信息（若配置 ECC）；

当 RID[3:0] 为 1 时，表示访问的是 ICACHE DATA RAM，读取 64-bit DATA 信息以及额外 ECC 编码信息（若配置 ECC）；

当 RID[3:0] 为 2 时，表示访问的是 DCACHE TAG RAM，读取 23-bit TAG 及 DIRTY 信息以及额外 ECC 编码信息（若配置 ECC）；

当 RID[3:0] 为 3 时，表示访问的是 DCACHE DATA RAM，读取 32-bit DATA 信息以及额外 ECC 编码信息（若配置 ECC）；

当 RID[3:0] 为 7 时，表示访问的是 I-TCM RAM，读取 64-bit DATA 信息以及额外 ECC 编码信息（若配置 ECC）；

当 RID[3:0] 为 8 时，表示访问的是 D-TCM RAM，读取 64-bit DATA 信息以及额外 ECC 编码信息（若配置 ECC）。

WAY-Cache 路信息：

指示 RAM 访问的路位置信息：

ICACHE TAG RAM：有两路 RAM，0001 指示的是 WAY0，0010 指示的是 WAY1；

ICACHE DATA RAM：有两路 RAM，0001 指示的是 WAY0，0010 指示的是 WAY1；

DCACHE TAG RAM：有四路 RAM，0001 指示的是 WAY0，0010 指示的是 WAY1，0100 指示的是 WAY2，1000 指示的是 WAY3；

DCACHE DATA RAM：有四路 RAM，0001 指示的是 WAY0，0010 指示的是 WAY1，0100 指示的是 WAY2，1000 指示的是 WAY3；

I-TCM RAM：无 way 信息；

D-TCM RAM：无 way 信息。

INDEX-CACHE 索引：

指示 RAM 访问的索引位置信息。

8.7.2 内存访问指令寄存器（CINS，CR<31,1>）

内存访问指令寄存器 CR<31,1> 用于使能内存 debug 访问操作，配合内存访问索引寄存器 CR<26,1> 使用，是一个只写寄存器。

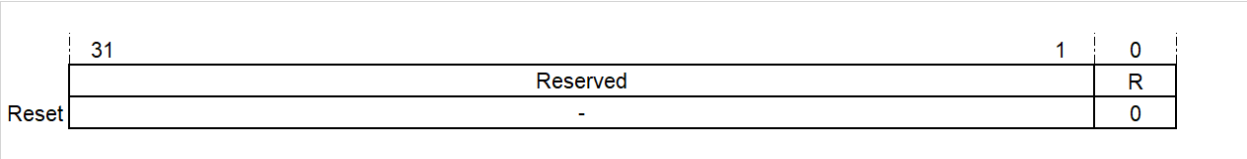


图 8.12: 内存访问指令寄存器

R-CACHE 读访问：

该位用于控制从 CR<26,1> 所指定的内存及索引从对应的内存中读出一个数据，得到的数据内容被装载到 CR<27,1>~CR<29,1> 寄存器。如果 CR<26,1> 寄存器的值无效时，处理器的操作将不可预知。软件将 R 位置 1 后，CPU 启动一次内存 debug 访问操作，访问完成后 R 位将由硬件清 0。

8.7.3 内存访问数据寄存器 0~2（CDATA0 ~ CDATA2，CR<27,1> ~ CR<29,1>）

内存访问数据寄存器用于存储从 CACHE 中读取的内容，不同 RAM 存储的数据含义不同，CDATA0~CDATA2 为只读寄存器。

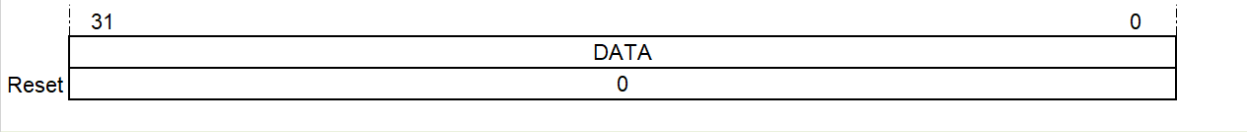


图 8.13: 内存访问数据寄存器

其中，CDATA0~CDATA1 存放的是 RAM 中的数据信息，CDATA2 存放的是数据对应的 ECC 编码信息。若硬件未配置 ECC，则 CDATA2 恒为 0。CDATA0~CDATA2 内容具体如下表所示：

表 8.7: 内存访问数据寄存器内容

RAM 类型	CDATA 内容
ICACHE TAG	CDATA0[20:1]: TAG CDATA0[0]: VALID CDATA2[6:0]: TAG ECC
ICACHE DATA	CDATA0[31:0]: DATA[31:0] CDATA1[31:0]: DATA[63:32] CDATA2[7:0]: 8-bit ECC for DATA
DCACHE TAG	CDATA0[22]: DIRTY CDATA0[21:1]: TAG CDATA0[0]: VALID CDATA2[7]: DIRTY ECC CDATA2[6:0]: TAG ECC
DCACHE DATA	CDATA0[31:0]: 32-bit DATA CDATA2[6:0]: 7-bit ECC for DATA
I-TCM DATA	CDATA0[31:0]: DATA[31:0] CDATA1[31:0]: DATA[63:32] CDATA2[7:0]: 8-bit ECC for DATA
D-TCM DATA	CDATA0[31:0]: DATA[31:0] CDATA1[31:0]: DATA[63:32] CDATA2[6:0]: 7-bit ECC for DATA[31:0] CDATA2[13:7]: 7-bit ECC for DATA[63:32]

第九章　总线接口

此章节描述了各总线接口，它包括了如下几个部分：

- 简介
- AXI 总线主接口
- 外设快速访问接口
- AXI 总线从接口

9.1　简介

本章主要讲解了各核外内存访问的总线接口，包括了：

- 一条基于 AXI3.0 的总线主接口，一般用于普通内存的读写访问；
- 两条外设快速访问总线，分别基于 AHB2.0 和 AXI3.0 总线协议；
- 一条基于 AXI3.0 的总线从接口，用于 TCM 的快速读写搬移操作。

R807 用于大规模的基于 AXI 和 AHB 协议的片上系统中，利用其两条选配的外设快速访问总线，可在多外设乃至超多外设的环境下可充分发挥其自有实时性能力。系统需为该两条外设总线分别配置相应的访问基址，之后该段地址将始终被认为是非高缓的属性（non-cacheable）。

当 R807 配有 TCM 时，将同时会配置一条基于 AXI3.0 的总线从接口，可使非核内的主接口往 TCM 进行读写操作，常用于 TCM 的初始化作业。

9.2　AXI 总线主接口

R807 核有一个 AXI 总线主接口，主要用于：

- I-Cache 的缓存行填充；
- D-Cache 的缓存行填充及缓存行读取；
- 非高缓的普通内存类型的指令抓取；
- 非高缓的普通内存类型的数据访问；

- 不支持强按序访问（强按序访问必须位于外设地址空间）；

该总线为 64 位数据位宽，并遵循 AXI3.0 协议，但其并不产生协议所允许的全部请求类型。本节将会描述 R807 能够产生的请求类型，若你设计的总线中仅有 R807，可利用该限制来优化从接口。同时，该总线可在与 CPU 相同频率下运作，也可在更低的 2 至 8 分频下运作。

其特性如下：

- 只支持以下 2 种传输方式：INCR（长度为 1）和 WRAP4，其中 WRAP4 只支持 64 位宽数据操作；
- 支持乱序完成，支持最多 5 个未完成读传输（5 outstanding transactions：1 个取指令传输，4 个数据预取传输），8 个未完成写传输；
- 支持关键字访问优先；
- 支持 cacheable 属性地址空间写 merge；
- 支持 CPU 到总线（cpu-to-bus）的整数倍分频（1:1, 2:1, 3:1, 4:1, 5:1,6:1, 7:1, 8:1）；
- 支持各种总线响应：OKAY、EXOKAY、SLVERR 和 DECERR；
- 总线控制和数据信号寄存器输出；
- 支持取指令请求与数据访问请求之间的乱序访问；

R807 AXI 已知的**限制**包括：

- 支持独占（Exclusive）原子操作但不支持锁（Lock）原子操作；
- 不支持写操作的交叉（interleaving），写数据按序发出；
- 不支持 4K 边界的检查（不会有 4K 边界问题）；
- 取指令访问与数据访问请求之间相关性不做检查。

9.2.1 读请求 ID

AXI 主接口可支持最大 5 个未完成乱序传输（outstanding transactions），意味着对于非高缓的请求，只要其 ID 不一致，即可实现乱序传输。而对于同一个突发请求，其 ID 号将保持一致，符合总线协议要求。

其 ID 分配如下表：

请求来源	分配的 ID 号
指令抓取	4’b0010
读取访问	4’b1000, 4’b1001
预取请求	4’b0001, 4’b0011, 4’b0101, 4’b0111

9.2.2 写请求 ID

写请求均于存储子模块发出，可分配 ID 为 4’0000~4’b0111。可实现最多 8 个乱序写操作。对于同一个突发请求，则必须使用同一个 ID，以符合总线协议要求。

9.2.3 突发类型

R807 只支持 64 位数据位宽的 BURST4 请求，其余均为 AxLen 为 0 的 Incr 类型请求。

其中，写突发请求的来源为：

- D-Cache 的缓存行读取操作；
- D-Cache 的清除操作。

读突发请求的来源为：

- D-Cache 的缓存行注入操作；
- Cache 的缓存行注入操作；
- 预读取操作。

9.3 外设快速访问接口

在大规模片上系统中，为提升外设响应的实时性，R807 可选配两条外设快速访问接口。其中一条总线为 32 位数据位宽，并遵循 AXI3.0 协议，但其并不产生协议所允许的全部请求类型。另一条总线也为 32 位数据位宽，但遵循 AHB2.0 协议，同样也不产生协议所允许的全部请求类型。

用户可根据外设所搭载的总线类型，自定义选配其中一条或多条总线接口。

若某读写访问请求命中该片外设访问接口所在地址区域时，将始终认为其为非高缓类请求。若指令抓取请求命中该片区域，则触发总线内存访问异常，效果相当于总线返回 ERR 信号。

9.3.1 基于 AHB2.0 的外设接口

该接口旨在为基于 AHB2.0 的外设提供 CPU 高速访问的接口。因其面向外设，该主接口不具备发送突发请求的能力，所有请求均为 single 类型的请求。因而其 htrans[1:0] 只有以下两种类型：

- NON-SEQ
- IDLE

该总线主接口所处的地址空间由片外信号决定。

pad_biu1_ahb_base[19:0] - 外设 AHB 类总线目标地址的高 20 位基地址；

pad_biu1_ahb_base_mask[19:0] - 外设 AHB 类总线目标地址的高 20 位基地址掩膜。

该片地址空间的最小大小为 4KB。

该主接口的特点有：

- 当执行 IDLY4 指令时通过 HLOCK 信号锁住总线；
- 所有的信号都 flopped in 或 flopped out 以获得较好的时序；

- 具备 AHB 协议背靠背发送请求的能力；
- 可发送任何不超过 32bit 数据位宽的 NON-SEQ 请求；
- 支持 RETRY 和 SPLIT 响应；
- 不支持写 merge；

9.3.1.1 异常处理

因仅支持 SINGLE 类别的请求访问，主接口对于 RETRY 和 SPLIT 类别的请求结果处理方式一致。对总线请求结果的各处理如下：

图表 9-1 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误-结束传输并处理访问错误。
High	OKEY	正常循环结束并继续。
Low	OKEY	插入等待状态。
不关心	RETRY/SPLIT	等待进入 RETRY 操作。

且对于写请求而言，访问错误（HRESP 为 ERROR）时，不告知 CPU 核该请求失败，而是将该错误信息丢弃。

9.3.2 基于 AXI3.0 的外设接口

该接口旨在为基于 AXI3.0 的外设提供 CPU 高速访问的接口。因其面向外设，该主接口不具备发送突发请求的能力，所有请求均为 single 类型的请求。因而其 AxLen[3:0] 恒为 0。同时，其 AxBurst 类型仅为 FIXED（2’b00）类型。

该总线主接口所处的地址空间由片外信号决定。

pad_biu1_axi_base[19:0] - 外设 AXI 类总线目标地址的高 20 位基地址；

pad_biu1_axi_base_mask[19:0] - 外设 AXI 类总线目标地址的高 20 位基地址掩膜。

该片地址空间的最小大小为 4KB。

该总线的主要**特点**有：

- 不支持乱序完成，读通道和写通道 ID 均为 0，支持强按序（Strong Order）和非强按序读写操作同时发生；
- 支持最多 8 个未完成传输写及 2 个未完成传输读（outstanding transactions）；
- 支持 CPU 到总线（cpu-to-bus）的整数倍分频（1:1, 2:1, 3:1, 4:1, 5:1,6:1, 7:1, 8:1）；
- 支持全部总线响应：OKAY、EXOKAY、SLVERR 和 DECERR；
- 总线控制和数据信号寄存器输出（flop-out），寄存器输入（flop-in）；

- 支持独占（Exclusive）原子操作但不支持锁（Lock）原子操作；
- 不支持写操作的交叉（interleaving）；
- 不支持 4K 边界的检查（不会有 4K 边界问题）；
- 不支持写 merge；

9.3.2.1 异常处理

对于非独占类的写操作，CPU 核将无视该请求是否成功而继续执行之后的指令；对于独占类的写操作及所有类型的读操作，CPU 核将接收请求的结果并产生对应的指令流。

对独占类请求：响应返回 Okay 信号后，认为独占请求失败；返回 EXOkay，则认为请求成功。

对普通类请求：响应返回 SLVERR 信号后，认为请求失败，若该请求为读请求则产生总线访问 2 号异常，否则丢弃；返回 Okay，认为请求响应成功。

9.4 AXI 总线从接口

若选配 TCM 模块，R807 将有一条基于 AXI3.0 的 64 位数据位宽的从接口。除此之外，该接口还额外增加了 2 个信号：

AW 通道：**pad_tcm_slave_awsel[1:0]** -用来选择当前接收写请求将去往哪块 TCM。

- 2’b01: 该请求将去往 I-TCM；
- 2’b10: 该请求将去往 D-TCM。

AR 通道：**pad_tcm_slave_arsel[1:0]** -用来选择当前接收读请求将去往哪块 TCM。

- 2’b01: 该请求将去往 I-TCM；
- 2’b10: 该请求将去往 D-TCM。

与本章其他总线相似，该从接口无法处理全部类型的请求，对于不支持的请求将返回 SLVERR 以告知对应的主接口请求操作失败。该接口主要用于核外系统对 TCM 进行访问操作，一般用于数据初始化，数据搬移等情景。

9.4.1 总线从接口特点

其**特点**如下：

- 读写通道的 ID 号均为 8 位；
- 支持以下 4 种传输方式：
  - 全部数据位宽的 AxLen 为 4’b0 的传输请求；
  - Double Word 数据位宽的全部 WRAP 类型的突发请求；

  - Double Word 数据位宽的全部 INCR 类型的突发请求；
  - Double Word 数据位宽的全部 FIXED 类型的突发请求；
  - 对其余类型请求返回 SLVERR，**请特别注意非 64 位的数据不支持 burst 类型**。
- 支持最多 2 个未完成的写传输，5 个未完成的读传输（outstanding transactions）；
- 支持 CPU 到总线（cpu-to-bus）的整数倍分频（1:1, 2:1, 3:1, 4:1, 5:1,6:1, 7:1, 8:1）；
- 产生 2 种总线响应：OKAY 和 SLVERR，对独占或锁类型的请求直接返回 OKAY，不进行任何操作；
- 总线控制和数据信号寄存器输出，寄存器输入；
- 支持 INCR 类型请求跨 4K 边界的检查；
- 读数据操作乱序深度为 1；
- 不支持写操作的交叉（interleaving），即写交织深度为 1；
- 不支持锁（Lock）和独占（Exclusive）原子操作（AWLOCK 和 ARLOCK）；
- 不支持 ARCACHE 和 AWCACHE 信号；
- 不支持 ARPROT 和 AWPROT 信号。

9.4.2 ECC 支持

在配有 ECC 的 R807 核中，该 AXI 从接口具备 ECC 校验的功能。

该从接口所有接收的读访问请求，在转发给 TCM 并读取数据后，会自行校验该数据的正确性。若不正确，将告知从接口产生了 ECC 校验错误。该错误只对总线有影响，不影响核内 CPU 正常执行。

该从接口所有接受的写访问请求，在转发给 TCM 并尝试写入数据时，只有“部分写”请求（指数据位宽小于 64bit 或数据 strobe 不为全“1”的写请求）将进行读取-修改-写入的操作，在此之间若检测 ECC 错误，则返回 ECC 校验错误给该从接口，并停止写入操作。同样，该错误只对总线有影响，不影响核内 CPU 正常执行。

从接口在接收 ECC 校验错误后，将该信号转换为 SLVERR 并放入对应的读写响应通道中，当系统中的主接口接收到该 SLVERR 后，确认该次请求失败。

9.4.3 总线异常

总线**异常情况**如下几种：

- 对于独占类请求（AxLock[1:0] != 2’ b0），将不进行对应的读写操作，返回 Okay 信号以告知主接口访问失败；
- 对于读请求，突发请求在地址跨 4K 后将返回 SLVERR；
- 对于写请求，突发请求在地址跨 4K 后将返回 SLVERR，该突发请求未跨 4K 部分数据被写入；
- 该从接口所接收的请求类型无法被执行（不支持），将返回 SLVERR；

- 该从接口所接收请求对应的 TCM 模块没有打开该从接口的使能位，无法进行相应读写操作，返回 SLVERR；
- 对于配有 ECC 校验的 R807，读请求获取校验失败的数据，返回 SLVERR；
- 对于配有 ECC 校验的 R807，读突发请求获取校验失败的数据，该拍请求返回 SLVERR；
- 对于配有 ECC 校验的 R807，“部分写”请求（小于 64 位宽数据的真实写入）因校验失败无法写入，返回 SLVERR；
- 对于配有 ECC 校验的 R807，“部分写”突发请求中某拍因校验失败无法写入，将返回 SLVERR，无法确认内存真实写入情况。

注意： 该从接口返回信号无法区别各类 SLVERR 的来源，因而对于配有 ECC 的 R807 核，对从接口发送的突发写请求一旦发生错误，其内存写情况应认为不可知。

9.4.4 访问控制

I-TCM_CFG 寄存器的第 2 位（CR<22,1>[2]）代表着 I-TCM 是否能接收来自 AXI 总线从接口的访问请求。初始化该位为关闭状态，只有在该位置“1”的情况下，从接口才能访问 I-TCM 片区域。

D-TCM_CFG 寄存器的第 3 位（CR<23,1>[2]）代表着 D-TCM 是否能接收来自 AXI 总线从接口的访问请求。初始化该位为关闭状态，只有在该位置“1”的情况下，从接口才能访问 D-TCM 片区域。

第十章 Lock Step 接口

R807 Lock Step 功能通过以下三根信号线与 SoC 进行交互：

10.1 Lock Step 功能使能信号

信号名	方向	复位	时钟	功能描述
pad_core_lock_step_cmp_enable	I	•	SYS	Lock Ste p 的功能使能位： 1：开启 lock step 功能； 0：关闭 lock step 功能。 若要开启该功能，需要将 C PU 核进行 reset 复位，并且在复位结束时，该信号应该置 1。若关闭该功能，则可直接关闭，但若要重新开启，则需要再次对 CPU 核进行初始化复位。

在 CPU 复位之后，CPU 将默认开启 lock step 功能，确保在 CPU 同步采样到输入信号 pad_core_lock_step_cmp_enable 的值之前，冗余核与主核能够一直保持相同的状态。若采样到 pad_core_lock_step_cmp_enable 为 0，则该功能关闭，CPU 将关闭冗余核来节省功耗，两核状态将不再一致。因此，若再次开启该功能，需要重新复位 CPU。

10.2 Lock Step Fault 指示信号

信号名	方向	复位	时钟	功能描述
core_pad_lock _step_cmp_fault	O	•	SYS	Lock Step Fault 指示位: 1: Lock Step 信号比较检测到错误; 0: Lock Step 信号比较检未测到错误。

在 Lock Step 功能使能时，一旦检测到错误，该信号将置 1，直到 RESET 信号将该位清 0 或者关闭 Lock Step 功能。一旦发生错误，CPU 内部的状态将不可知。此时，Lock Step 会对关键的总线输出信号进行控制，确保 CPU 将不再向总线发起任何新的请求，也不再接收总线的任何请求。

10.3 Lock Step 错误注入使能信号

信号名	方向	复位	时钟	功能描述
pad_biu_lock_st ep_fault_inject	I	•	SYS	Lock Step 错误注入使能位: 1: Lock Ste p 错误注入开启; 0: Lock Ste p 错误注入关闭。 当该位置 1 时，将对 CPU 进行干扰并导致出错，出错后 core_pad_ lock_step_cmp_f ault 信号将置 1。

为了方便测试，R807 支持通过该信号先来实现硬件注入错误，干扰主核的输出信号，致使 Lock Step 比较结果出错。

第十一章　多核系统与集成

此章节描述了多核系统与集成，它包括了如下几个部分：

- 多核集成
- 数据一致性

11.1　多核集成

R807 可以进行多核集成，多核之间地址属性是否可共享的情况如下：

- 所有可高缓区域不可共享
- 所有命中 TCM 的区域不可共享
- 其余区域都为可共享

以上共享属性为默认属性，无法通过 MPU 设置。

11.2　数据一致性

对于多核系统而言，当不同核对可共享的同一地址进行操作时，需要通过数据一致性来保证正确性。数据一致性可通过硬件或者软件来实现。由于 R807 不支持 CACHE 一致性，因此不支持多核之间的硬件同步一致性，在前一节也提到可高缓的区域默认不可共享。

因此，针对多核的可共享区域（不可高缓，不命中 TCM），R807 的数据一致性需要软件来维护，R807 提供了原子指令来支持软件的一致性同步操作，原子指令既可用于多核之间的可共享区域同步，也可用于不同进程在同一个核不可共享区域的同步操作。

11.2.1　原子指令操作

R807 支持独占式的内存访问指令 LDEX32.W 和 STEX32.W。用户可以使用这两条指令构成原子锁等同步原语，来实现同一个核不同进程之间或者不同核之间的同步。LDEX 指令用于标记需要独占访问的地址，STEX 指令判断被标记的地址是否被其他进程抢占。

R807 核内设置了一个位于 L1 内存系统（包括 CACHE 和 TCM）的局部监测器，由一个状态机和一个地址缓存器组成。其中，状态机包含两个状态：IDLE 和 EXCLUSIVE。

对于属性设置为可高缓的页面（即使数据高速缓存未使能）或者命中 TCM 的区域，通过局部监测器就能够实现独占式访问。局部监测器只支持监测一组地址，因此只支持同时一组 ldex/stex 进行独占式访问。LDEX 指令在执行过程中设置局部监测器的状态机为 EXCLUSIVE 态，并将访问的地址保存到地址缓存器中；STEX 指令在执行过程中读取局部监测器的状态和地址，如果状态为 EXCLUSIVE 并且地址匹配，那么执行该写操作，返回写成功，并清除状态机回到 IDLE 态；否则如果状态或者地址有一项不满足条件，不执行该写操作，返回写失败，并清除状态机回到 IDLE 态。此外，在进程切换时需要清除局部监测器。

对于属性设置为不可高缓且未命中 TCM（命中 TCM 则无视 cacheble 属性）的页面，需要局部监测器和全局监测器共同作用实现独占式访问。LDEX 在执行过程中不仅要设置局部监测器的状态，还要通过总线访问（AXI 系统总线或者 AXI 外设总线）去设置全局监测器；STEX 在局部监测器检查通过后，也需要通过总线访问进一步检查全局监测器。只有当全局监测器也通过检查，才执行写操作、返回写成功，清除状态机；否则不执行写操作，返回写失败，清除状态机。

第十二章　调试

本章描述 R807 的调试系统，主要包含调试架构，调试接口以及调试方法等。本章由以下小节组成：

- 概述
- 外部接口

12.1　概述

调试接口是软件与处理器交互的通道。用户可以通过调试接口获取 CPU 的寄存器以及存储器内容等信息，包括其他的片上设备信息。此外，程序下载等操作也可以通过调试接口完成的。

调试接口的主要特性如下：

- 非侵入式获取 CPU 状态；
- 支持同步调试和异步调试，保证在极端恶劣情况下使处理器进入调试模式；
- 支持软断点；
- 可以设置多个内存断点；
- 检查和设置 CPU 寄存器的值；
- 检查和改变内存值；
- 可进行指令单步执行或多步执行；
- 快速下载程序；
- 可在 CPU 复位之后或在普通用户模式下进入调试模式；

玄铁 CPU 的调试工作是调试软件，调试代理服务程序，调试器和调试接口一起配合完成的，调试接口在整个 CPU 调试环境中的位置如图表 图 3.1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与在线仿真器（ICE）通过 USB 连接，ICE 与 CPU 的调试接口以 JTAG 模式通信，R807 设计实现的 JTAG 接口通信兼容 IEEE1149.1 标准。

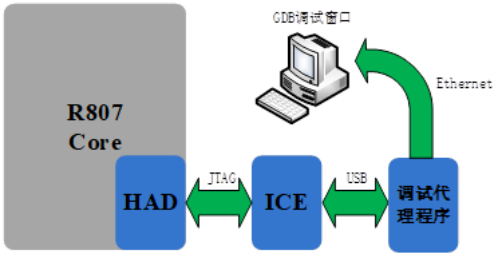


图 12.1: 调试接口在整个 CPU 调试环境中的位置

12.2 外部接口

调试模块与外部的接口主要是与 JTAG 相关的接口信号和调试相关的接口信号。图表 表 12.1 列出了与调试相关的接口信号。

表 12.1: 调试模块与外部的接口信号

信号名	方向
biu_pad_dbg_b	输出
pad_had_jdb_req_b	输入
had_pad_jdb_ack_b	输出
pad_had_jtg_tap_en	输入
had_pad_jtg_tap_on	输出
had_pad_jdb_pm[1:0]	输出
pad_had_jtg_tclk	输入
pad_had_jtg_trst_b	输入
pad_had_jtg_tms_i	输入
pad_had_jtg_tdi	输入
had_pad_jtg_tdo	输出
had_pad_htg_tdo_en	输出

1. **biu_pad_dbg_b**

   低电平表示 CPU 处于调试模式中。

2. **pad_had_jdb_req_b 与 had_pad_jdb_ack_b**

pad_had_jdb_req_b 信号是让 CPU 异步进入调试状态的请求信号，该信号至少要维持低电平两个 JTAG 时钟周期才能保证 CPU 进入调试状态并且能够调试程序。如果该信号维持低电平不足两个 JTAG 时钟周期，那么可能会出现 CPU 已进入调试状态但不能调试程序的情况，因为该信号还会用于使能调试接口中的 TAP 状态机。

在 CPU 进入调试状态之后，had_pad_jdb_ack_b 信号会维持两个 JTAG 时钟周期的低电平以作为应答。

3. **pad_had_jtg_tap_en 与 had_pad_jtg_tap_on**

pad_had_jtg_tap_en 信号为调试接口中 TAP 状态机的使能信号，维持该信号为高电平至少一个 JTAG 时钟周期可以使能调试接口中的 TAP 状态机。如果该信号在 CPU 上电之后一直无效，那么使用同步方式（如设置调试接口寄存器 HCR 中的 DR 位，断点等）使 CPU 进入调试状态时可能无法调试程序。

在 TAP 状态机启动之后，had_pad_jtg_tap_on 信号将拉高。

4. **had_pad_jdb_pm[1:0]**

had_pad_jdb_pm[1:0] 信号指示 CPU 当前工作模式，可以通过该信号确定 CPU 是否已进入调试模式。具体如图表 表 12.2 所示。

表 12.2: had_pad_jdb_pm 指示当前 CPU 状态

had_pad_jdb_pm[1:0]	说明
00	普通模式
01	低功耗模式（STOP，DOZE，WAIT）
10	调试模式
11	保留

5. **pad_had_jtg_tclk**

JTAG 时钟信号。该信号为外部输入信号，一般为调试器产生的时钟信号，要保证该时钟信号的频率低于 CPU 时钟信号的频率 1/2 才能保证调试模块与 CPU 核之间的正常工作。

6. **pad_had_jtg_trst_b**

pad_had_jtg_trst_b 信号为 JTAG 复位信号，可以复位 TAP 状态机以及其他相关控制信号。

7. **pad_had_jtg_tms**

pad_had_jtg_tms 是模式选择信号，用于驱动 TAP 状态机的跳转。

8. **pad_had_jtg_tdi**

pad_had_jtg_tdi 是串行数据输入信号，高位在先。在 tclk 时钟下降沿由 ICE 设置并传入 R807 内部的调试单元，在 tclk 时钟上升沿 R807 内部的调试单元对其进行采样。

9. **had_pad_jtg_tdo 和 had_pad_jtg_tdo_en**

had_pad_jtg_tdo 是串行数据输出信号，高位在先。在 tclk 时钟下降沿由 R807 内部的调试单元对其进行设置并输出，在 tclk 时钟上升沿 ICE 对其进行采样。只有在 had_pad_jtg_tdo_en 为高时 had_pad_jtg_tdo 有效。

第十三章　工作模式与转换

R807 支持三类工作模式：正常工作模式、低功耗工作模式和调试模式，其中低功耗工作模式分为三种：STOP 模式、DOZE 模式、WAIT 模式。本章将详细解析 CPU 的工作模式以及模式之间的转换。

13.1　工作模式

13.1.1　正常工作模式

CPU 的正常工作模式可以分为两种，超级用户模式和普通用户模式，CPU 处于哪种正常工作模式可以通过查询 PSR 寄存器中的 S 位得到，当 S 位为 1 时，CPU 工作于超级用户模式；反之，CPU 工作于普通用户模式。当 CPU 工作在超级用户模式时，可以通过将 S 位设置为 0 进入普通用户模式；当 CPU 工作于普通用户模式时，通过引发异常进入超级用户模式。更多介绍请参考工作模式及寄存器视图。

13.1.2　低功耗模式

当 CPU 执行完低功耗指令 (STOP、DOZE、WAIT) 之后，CPU 将进入相应的低功耗模式。三条指令执行的过程是，CPU 执行到低功耗指令后将等待前面的所有指令执行完，然后完成低功耗指令，同时根据低功耗指令类型驱动 biu_pad_lpmd_b[1:0] 信号，停止执行指令并冻结流水线，关掉内部时钟。

对于 R807 而言三条低功耗指令执行完成后 CPU 内部所实现的低功耗策略相同，即关闭 CPU 内部所有跟唤醒操作无关的寄存器驱动时钟。SoC 设计人员可依据 R807 顶层的输出信号 biu_pad_lpmd_b[1:0] 来区分所执行的低功耗指令进而在系统层面设计实现不同层次的系统低功耗策略。

当 CPU 处于低功耗模式时，唤醒中断请求（pad_biu_intraw_b 和 pad_biu_fintraw_b 信号）或者调试请求可以将 CPU 唤醒。若采用中断唤醒之后 CPU 将从进入低功耗模式的低功耗指令处继续执行后续指令。

13.1.3　调试模式

CPU 进入调试模式后将停止主动的取指执行操作，只会被动的等待执行由调试软件藉由 JTAG 接口传递进来的指令。通过这种方式设计人员可以查询和修改 CPU 的状态，进而进行调试。

13.1.3.1 进入调试模式

当 CPU 接到调试请求之后，进入调试模式，其中的调试请求源可以有以下 6 种:

- 在正常工作模式下，通过驱动 R807 顶层输入信号 pad_had_jdb_req_b 可以使 CPU 异步进入调试模式。
- 在正常工作模式下，当 HAD 模块 HCR 寄存器的 IDRE 位有效时，通过驱动 R807 顶层输入信号 pad_biu_dbgrq_b 可以使 CPU 同步进入调试模式。
- 当 R807 HAD 模块 CSR 寄存器的 FDB 位有效时，CPU 在执行 BKPT 指令后进入调试模式。
- 当 R807 HAD 模块 HCR 寄存器的 DR 位有效时，CPU 在执行完当前指令后进入调试模式。
- 当 R807 HAD 模块 HCR 寄存器的 TME 位有效时，CPU 在跟踪计数器的值减到 0 后进入调试模式。
- 在 R807 存储器断点调试模式下，当 MBCA 的值为 0 时，如果当前执行的指令符合断点要求，处理器进入调试模式。

13.1.3.2 退出调试模式

如果 R807 HAD 模块中 HACR 寄存器的 GO、EX 位被置为 1 时，同时 R/W 位为 0（写操作），RS 位选择的是 WBBR、PSR、PC、IR、CSR 或者 Bypass 寄存器，则执行指令时 CPU 退出调试模式，进入正常工作模式。

> **注意:** 由于在调试模式下 PC，CSR，PSR 是可变的，因此在退出调试模式前，上述寄存器中的值必须是刚进入调试模式时保存过的值。

13.2 模式转换

如图表 图 13.1 所示，R807 可以在正常工作模式、低功耗工作模式和调试模式间进行转换。CPU 当前的工作模式可以通过查询 had_pad_jdb_pm[1:0] 信号得到。

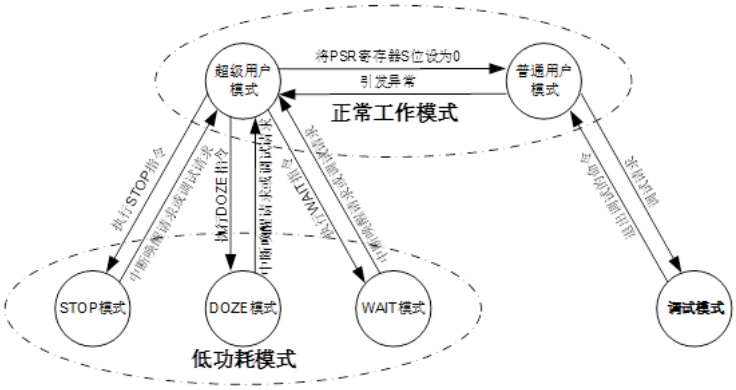


图 13.1: CPU 工作模式转换示意图

第十四章　附录指令术语表

以下是每条 C810 实现的玄铁 CPU V2 指令的具体描述，下面根据指令英文字母顺序对每条指令进行详细说明。

每条指令助记符结尾以数字“32”或“16”表示指令位宽。例如，“addc32”表示该指令为 32 位无符号带进位加法指令，“addc16”表示该指令为 16 位无符号带进位加法指令。

如果省略助记符中的指令位宽（如“addc”），系统会自动将其汇编为最优化的指令。

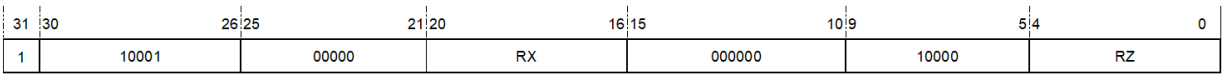
其中，指令中文名称中带 # 的为伪指令。

14.1　ABS——绝对值指令

统一化指令	
语法	abs rz, rx
操作	RZ ← | RX |
编译结果	仅存在 32 位指令。 abs32 rz, rx
说明	取 RX 值的绝对值，并把结果存在 RZ。 注意，操作数 0x80000000 的结果为 80000000。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← |RX|
语法	abs32 rz, rx
说明	取 RX 值的绝对值，并把结果存在 RZ。 注意，操作数 0x80000000 的结果为 80000000。
影响标志位	无影响
异常	无

指令格式:



14.2 ADDC——无符号带进位加法指令

统一化指令		
语法	addc rz, rx	addc rz, rx, ry
操作	RZ ← RZ + RX + C, C← 进位	RZ ← RX + RY + C, C← 进位

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	addc16 rz, rx;
	else
	addc32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		addc16 rz, rx;
		else
		addc32 rz, rx, ry;

说明	将 RZ/RY、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。	
影响标志位	C ← 进位	
异常	无	

16 位指令	
操作	RZ ← RZ + RX + C，C← 进位
语法	addc16 rz, rx
说明	将 RZ、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。
影响标志位	C ← 进位
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

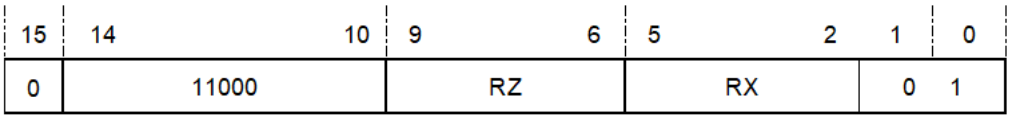


图 14.1: ADDC-1

32 位指令	
操作	RZ ← RX + RY + C，C← 进位
语法	addc32 rz, rx, ry
说明	将 RX、RY 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。
影响标志位	C ← 进位
异常	无

32 位指令格式：

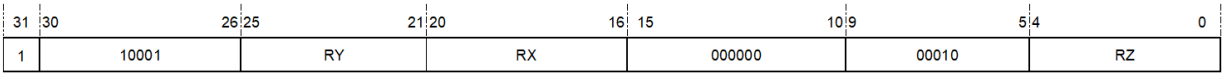


图 14.2: ADDC-2

14.3 ADDI——无符号立即数加法指令

统一化指令			
语法	addi rz, oimm12	addi rz, rx, oimm12	addi rz, r28, oimm18
操作	RZ ← RZ + zero_extend(OIMM12)	RZ ← RX + zero_extend(OIMM12)	RZ ← R28 + zero_extend(OIMM18)

编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<8) and (oimm12<257),
	addi16 rz, oimm8;
	else
	addi32 rz, rz, oimm12;
		根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (oimm12<257),
		addi16 rz, oimm8;
		else
		addi32 rz, rz, oimm12;
			根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
			if (oimm12<9) and (z<8) and (x<8),
			addi16 rz, rx, oimm3;
			elsif (oimm12<257) and (x==z) and (z<8),
			addi16 rz, oimm8;
			else
			addi32 rz, rx, oimm12;

说明	将带偏置 1 的立即数零扩展至 32 位，然后与 RX/RZ 的值相加，把结果存入 RZ。		
影响标志位	无影响		
限制	若源寄存器是 R28，立即数的范围为 0x1-0x40000。 若源寄存器不是 R28，立即数的范围为 0x1-0x1000。		

16 位指令 1	
操作	RZ ← RZ + zero_extend(OIMM8)
语法	addi16 rz, oimm8
说明	将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后与 RZ 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM8 等于 OIMM8 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-256。
异常	无

16 位指令格式 1：

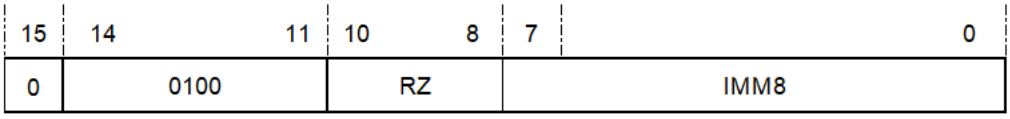


图 14.3: ADDI-1

IMM8 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000：

加 1

00000001：

加 2

……

11111111：

加 256

16 位指令 2	
操作	RZ ← RX + zero_extend(OIMM3)
语法	addi16 rz, rx, oimm3
说明	将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM3 等于 OIMM3 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-8。
异常	无

16 位指令格式 2：

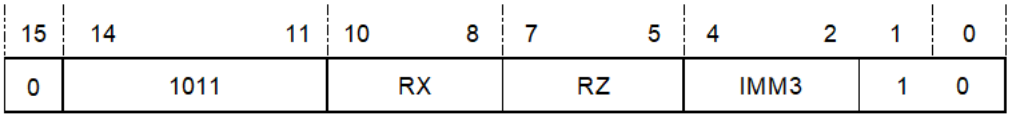


图 14.4: ADDI-2

IMM3 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000：

加 1

001：

加 2

……

111：

加 8

32 位指令 1	
操作	RZ ← RX + zero_extend(OIMM12)
语法	addi32 rz, rx, oimm12
说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM12 等于 OIMM12 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x1000。
异常	无

32 位指令格式 1：

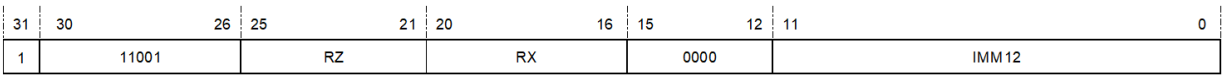


图 14.5: ADDI-3

IMM12 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

000000000000：

加 0x1

000000000001：

加 0x2

……

111111111111：

加 0x1000

32 位指令 2	
操作	RZ ← R28 + zero_extend(OIMM18)
语法	addi32 rz, r28, oimm18
说明	将带偏置 1 的 18 位立即数（OIMM18）零扩展至 32 位，然后与 R28 的值相加，把结果存入 RZ。 注意：二进制操作数 IMM18 等于 OIMM18 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x40000。
异常	无

32 位指令格式 2：

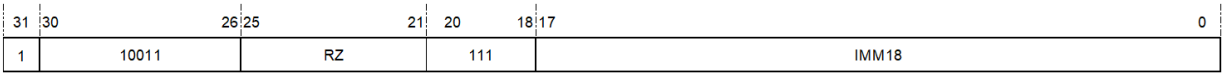


图 14.6: ADDI-4

IMM18 域：

指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM18 比起二进制操作数 IMM18 需偏置 1。

00000000000000：

加 0x1

00000000000001：

加 0x2

……

11111111111111：

加 0x40000

14.4 ADDI(SP)——无符号（堆栈指针）立即数加法指令

统一化指令		
语法	addi rz, sp, imm	addi sp, sp, imm
操作	RZ ← SP + zero_extend(IMM)	SP ← SP+ zero_extend(IMM)
编译结果	仅存在 16 位指令。 addi rz, sp, imm	仅存在 16 位指令。 addi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ 或者 SP。	
影响标志位	无影响	
限制	寄存器的范围为 r0-r7；立即数的范围为 0x0-0x3fc。	
异常	无	

16 位指令 1	
操作	RZ ← SP + zero_extend(IMM)
语法	addi16 rz, sp, imm8
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。 注意：立即数（IMM）等于二进制操作数 IMM8 << 2。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 (0x0-0xff) << 2。
异常	无

16 位指令格式 1：

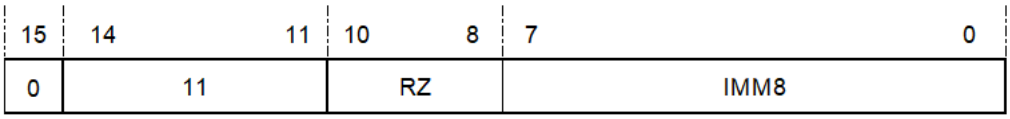


图 14.7: ADDI(SP)-1

IMM8 域：

指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 IMM8 需左移 2 位。

00000000：

加 0x0

00000001：

加 0x4

……

11111111：

加 0x3fc

16 位指令 2	
操作	SP ← SP + zero_extend(IMM)
语法	addi16 sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。 注意：立即数（IMM）等于二进制操作数 {IMM2, IMM5} << 2。
影响标志位	无影响
限制	源与目的寄存器均为堆栈指针寄存器（R14）；立即数的范围为 (0x0-0x7f) << 2。
异常	无

16 位指令格式 2：

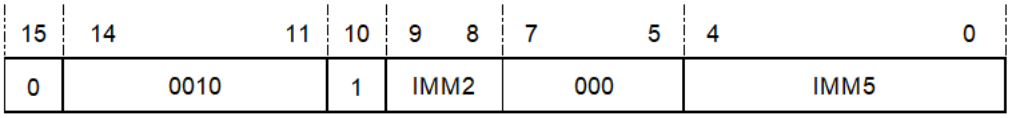


图 14.8: ADDI(SP)-2

IMM 域:

指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 {IMM2, IMM5} 需左移 2 位。

{00, 00000}

加 0x0

{00, 00001}

加 0x4

……

{11, 11111}

加 0x1fc

14.5 ADDU——无符号加法指令

统一化指令		
语法	addu rz, rx	addu rz, rx, ry
操作	RZ ← RZ + RX	RZ ← RX + RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<16) and (x<16), then
	addu16 rz, rx;
	else
	addu32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (x<8) and (y<8), then
		addu16 rz, rx, ry;
		elsif (y==z) and (x<16) and (z<16), then
		addu16 rz, rx;
		else
		addu32 rz, rx, ry;

说明	将 RZ/RY 与 RX 的值相加，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令 1	
操作:	RZ ← RZ + RX
语法	addu16 rz, rx
说明:	将 RZ 与 RX 的值相加，并把结果存在 RZ。
影响标志位:	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式 1：

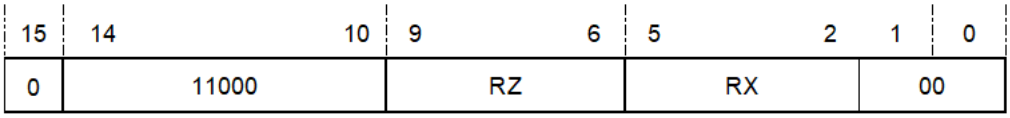


图 14.9: ADDU-1

16 位指令 2	
操作	RZ ← RX + RY
语法	addu16 rz, rx, ry
说明	将 RX 与 RY 的值相加，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	无

16 位指令格式 2：

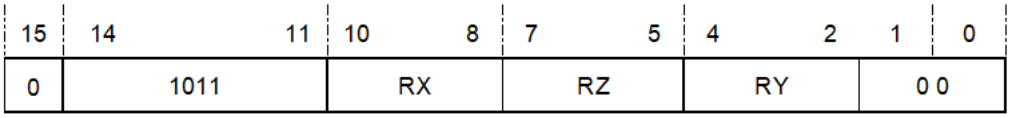


图 14.10: ADDU-2

32 位指令	
操作	RZ ← RX + RY
语法	addu32 rz, rx, ry
说明	将 RX 与 RY 的值相加，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

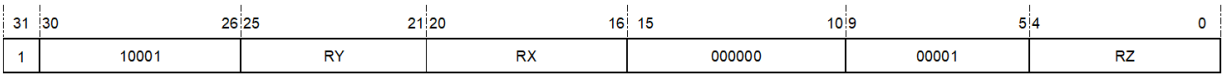


图 14.11: ADDU-3

14.6 AND——按位与指令

统一化指令		
语法	and rz, rx	and rz, rx, ry
操作	RZ ← RZ and RX	RZ ← RX and RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	and16 rz, rx;
	else
	and32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		and16 rz, rx;
		else
		and32 rz, rx, ry;

说明	将 RZ/RY 与 RX 的值按位与，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ and RX
语法	and16 rz, rx
说明	将 RZ 与 RX 的值按位与，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

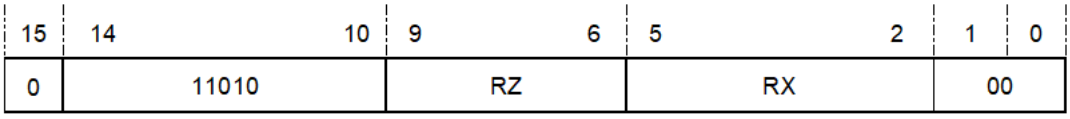


图 14.12: AND-1

32 位指令	
操作	RZ ← RX and RY
语法	and32 rz, rx, ry
说明	将 RX 与 RY 的值按位与，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

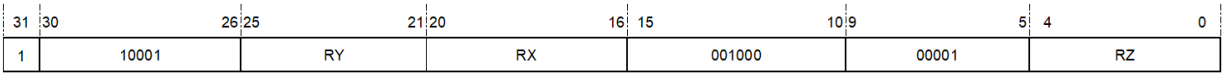


图 14.13: AND-2

14.7 ANDI——立即数按位与指令

统一化指令	
语法	andi rz, rx, imm16
操作	RZ ← RX and zero_extend(IMM12)
编译结果	仅存在 32 位指令 andi32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令	
操作	RZ ← RX and zero_extend(IMM12)
语法	andi32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令格式：

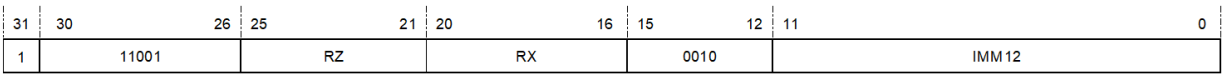


图 14.14: ANDI

14.8 ANDN——按位非与指令

统一化指令		
语法	andn rz, rx	andn rz, rx, ry
操作	RZ ← RZ and (!RX)	RZ ← RX and (!RY)

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	andn16 rz, rx;
	else
	andn32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		andn16 rz, ry;
		else
		andn32 rz, rz, rx;

说明	对于 andn rz, rx，将 RZ 的值与 RX 的非值按位与，并把结果存在 RZ；对于 andn rz, rx, ry，将 RX 的值与 RY 的非值按位与，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ and (!RX)
语法	andn16 rz, rx
说明	将 RZ 的值与 RX 的非值按位与，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

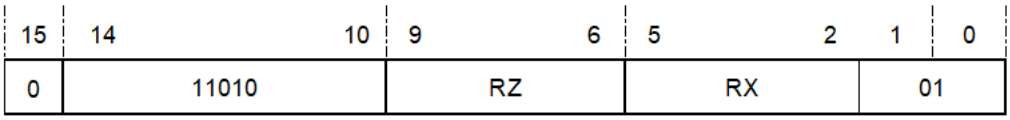


图 14.15: ANDN-1

32 位指令	
操作	RZ ← RX and (!RY)
语法	andn32 rz, rx, ry
说明	将 RX 的值与 RY 的非值按位与，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

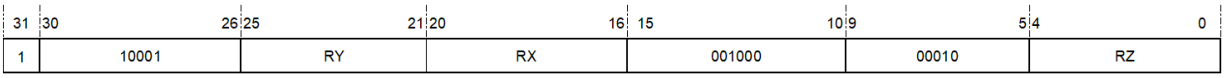


图 14.16: ANDN-2

14.9 ANDNI——立即数按位非与指令

统一化指令	
语法	andni rz, rx, imm16
操作	RZ ← RX and !(zero_extend(IMM12))
编译结果	仅存在 32 位指令 andni32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令	
操作	RZ ← RX and !(zero_extend(IMM12))
语法	andni32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令格式：

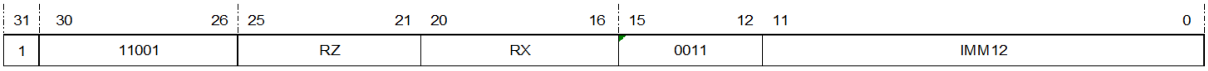


图 14.17: ANDNI

14.10 ASR——算术右移指令

统一化指令		
语法	asr rz, rx	asr rz, rx, ry
操作	RZ ← RZ >>> RX[5:0]	RZ ← RX >>> RY[5:0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	asr16 rz, rx;
	else
	asr32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		asr16 rz, ry;
		else
		asr32 rz, rx, ry;

说明	对于 asr rz, rx，将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定； 对于 asr rz, rx, ry，将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ >>> RX[5:0]
语法	asr16 rz, rx
说明	将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

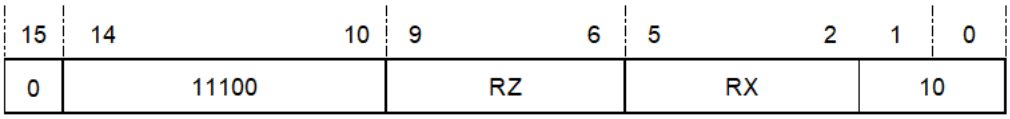


图 14.18: ASR-1

32 位指令	
操作	RZ ← RX >>> RY[5:0]
语法	asr32 rz, rx, ry
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。
影响标志位	无影响
异常	无

32 位指令格式：

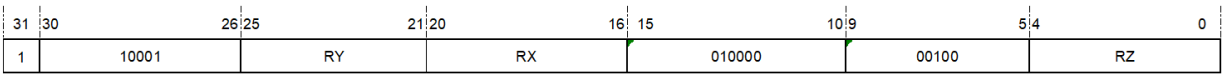


图 14.19: ASR-2

14.11 ASRC——立即数算术右移至 C 位指令

统一化指令	
语法	asrc rz, rx, oimm5
操作	RZ ← RX >>> OIMM5, C ← RX[OIMM5 - 1]
编译结果	仅存在 32 位指令 asrc32 rz, rx, oimm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 的符号位决定。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	RZ ← RX >>> OIMM5, C ← RX[OIMM5 - 1]
语法	asrc32 rz, rx, oimm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 符号位决定。注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

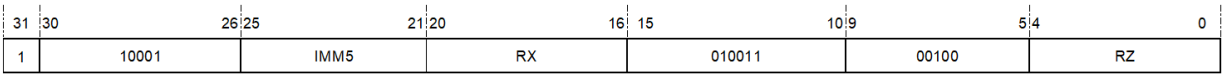


图 14.20: ASRC

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

……

11111：

移 32 位

14.12 ASRI——立即数算术右移指令

统一化指令	
语法	asri rz, rx, imm5
操作	RZ ← RX >>> IMM5

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<8) and (z<8), then
	asri16 rz, rx, imm5;
	else
	asri32 rz, rx, imm5;

说明	对 asri rz, rx, imm5 而言，将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← RX >>> IMM5
语法	asri16 rz, rx, imm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

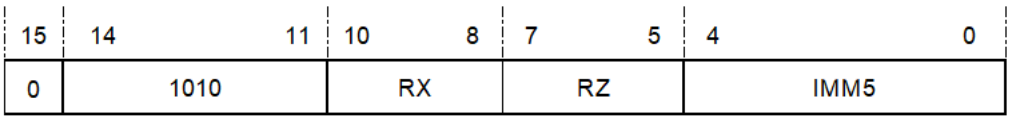


图 14.21: ASRI-1

32 位指令	
操作	RZ ← RX >>> IMM5
语法	asri32 rz, rx, imm5
说明	将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

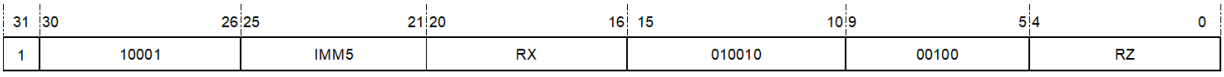


图 14.22: ASRI-2

14.13 BCLRI——立即数位清零指令

统一化指令		
语法	bclri rz, imm5	bclri rz, rx, imm5
操作	RZ ← RZ[IMM5] 清零	RZ ← RX[IMM5]

编译结果	清零根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<8)，then
	bclri16 rz, imm5;
	else
	bclri32 rz, rz, imm5;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (z<8), then
		bclri16 rz, imm5;
		else
		bclri32 rz, rx, imm5;

说明	将 RZ/RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。	
影响标志位	无影响	
限制	立即数的范围为 0-31。	

16 位指令	
操作	RZ ← RZ[IMM5] 清零
语法	bclri16 rz, imm5
说明	将 RZ 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7； 立即数的范围为 0-31。
异常	无

16 位指令格式：

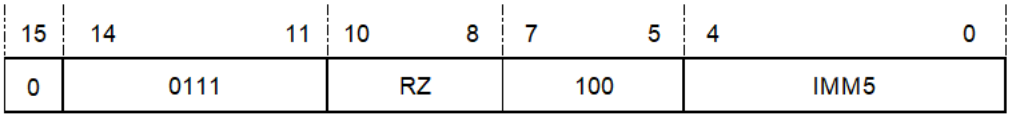


图 14.23: BCLRI-1

32 位指令	
操作	RZ ← RX[IMM5] 清零
语法	bclri32 rz, rx, imm5
说明	将 RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

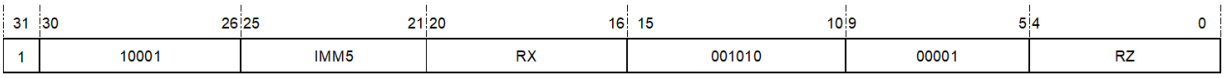


图 14.24: BCLRI-2

14.14 BEZ——寄存器等于零分支指令

统一化指令	
语法	bez rx, label

操作	寄存器等于零则程序转移
	if (RX == 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

编译结果	仅存在 32 位指令 bez32 rx, label
说明	如果寄存器 RX 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器等于零则程序转移
	if(RX == 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bez32 rx, label
说明	如果寄存器 RX 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

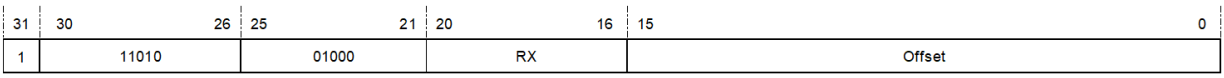


图 14.25: BEZ

14.15 BF——C 为 0 分支指令

统一化指令	
语法	bf label

操作	C 等于零则程序转移。
	if(C==0)
	PC ← PC + sign_extend(offset << 1);
	else
	PC ← next PC;

编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令。
	if (offset<1KB), then
	bf16 label;
	else
	bf32 label;

说明	如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

16 位指令	

操作	C 等于零则程序转移。
	if(C==0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 2

语法	bf16 label
说明	如果条件标志位 C 等于 0，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。 Label 由当前程序 P C 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BF16 指令的转移范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

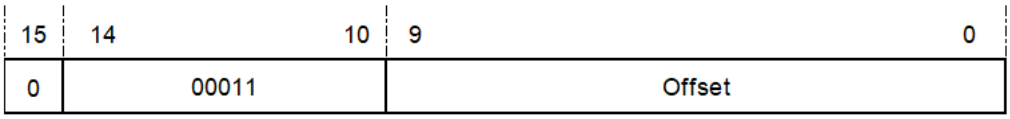


图 14.26: BF-1

32 位指令	

操作	C 等于零则程序转移
	if(C == 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bf32 label
说明	如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

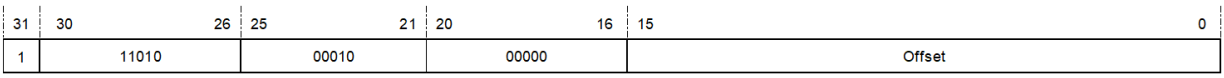


图 14.27: BF-2

14.16 BGENI——立即数位产生指令

统一化指令	
语法	bgeni rz, imm5
操作	RZ ← (2) ^{IMM5}
编译结果	仅存在 32 位指令。 bgeni32 rz, imm5
说明	对由 5 位立即数确定的 RZ 的位（RZ[IMM5]）置位，并清除 RZ 的其它位。 注意，如果 IMM5 小于 16，该指令是 movi rz, (2)^{IMM5} 的伪指令；如果 IMM5 大于等于 16，该指令是 movih rz, (2)^{IMM5} 的伪指令。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	RZ ← (2) ^{IMM5};
语法	bgeni32 rz, imm5
说明	对由 5 位立即数确定的 RZ 的位（RZ[IMM5]）置位，并清除 RZ 的其它位。 注意，如果 IMM5 小于 16，该指令是 movi32 rz, (2)^{IMM5} 的伪指令；如果 IMM5 大于等于 16，该指令是 movih32 rz, (2)^{IMM5} 的伪指令。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

如果 IMM5 小于 16：

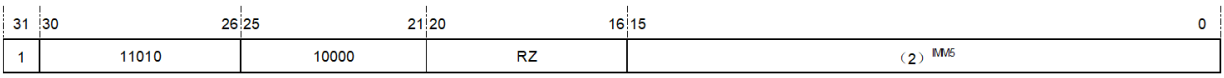


图 14.28: BGENI-1

如果 IMM5 大于等于 16：

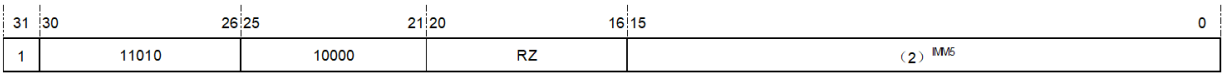


图 14.29: BGENI-2

14.17 BGENR——寄存器位产生指令

统一化指令	
语法	bgenr rz, rx

操作	If (RX[5] == 0) ， then
	RZ ← $2^{RX[4:0]}$;
	else
	RZ ← 0;

编译结果	仅存在 32 位指令 bgenr32 rz, rx
说明	如果 RX[5] 为 0，那么置 RZ 由 RX 低五位（RX[4:0]）确定的寄存器位，并清除 RZ 所有其它的位；否则，对 RZ 清零。
影响标志位	无影响
异常	无

32 位指令	

操作	If (RX[5] == 0) ， then
	RZ ← $2^{RX[4:0]}$;
	else
	RZ ← 0;

语法	bgenr32 rz, rx
说明	如果 R X[5] 为 0，那么置 RZ 由 RX 低五位（RX[4:0]）确定的寄存器位，并清除 RZ 所有其它的位；否则，对 RZ 清零。
影响标志位	无影响
异常	无

32 位指令格式：

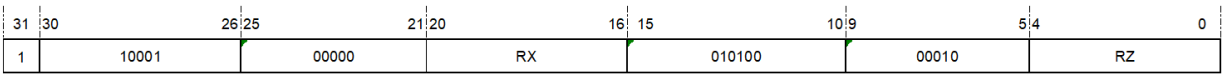


图 14.30: BGENR

14.18 BHSZ——寄存器大于等于零分支指令

统一化指令	
语法	bhsz rx, label

操作	寄存器大于等于零则程序转移
	if(RX >= 0)
	PC ← PC + sign_extend(offset << 1);
	else
	PC ← PC + 4;

编译结果	仅存在 32 位指令 bhsz32 rx, label
说明	如果寄存器 RX 大于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器大于等于零则程序转移
	if(RX >= 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bhsz32 rx, label
说明	如果寄存器 RX 大于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

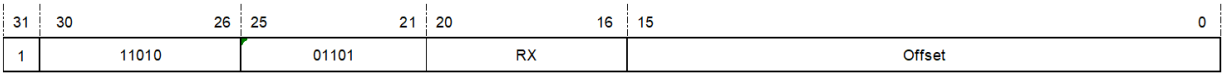


图 14.31: BHSZ

14.19 BHZ——寄存器大于零分支指令

统一化指令	
语法	bhz rx, label

操作	寄存器大于零则程序转移
	if(RX > 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

编译结果	仅存在 32 位指令 bhz32 rx, label
说明	如果寄存器 RX 大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器大于零则程序转移
	if(RX > 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bhz32 rx, label
说明	如果寄存器 RX 大于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BHZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

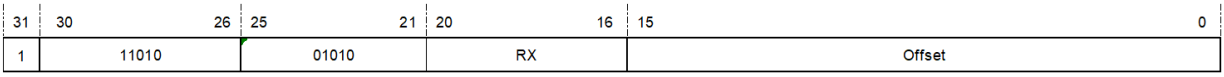


图 14.32: BHZ

14.20 BKPT——断点指令

统一化指令	
操作	引起一个断点异常或者进入调试模式
编译结果	总是编译为 16 位指令。 bkpt16
说明	断点指令
影响标志位	无影响
异常	断点异常

16 位指令	
操作	引起一个断点异常或者进入调试模式
语法	bkpt16
说明	断点指令
影响标志位	无影响
异常	断点异常

16 位指令格式：

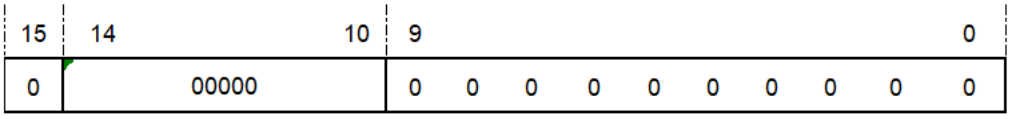


图 14.33: BKPT

14.21 BLSZ——寄存器小于等于零分支指令

统一化指令	
语法	blsz rx, label

操作	寄存器小于等于零则程序转移
	if(RX <= 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

编译结果	仅存在 32 位指令 blsz32 rx, label
说明	如果寄存器 RX 小于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器小于等于零则程序转移
	if(RX <= 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	blsz32 rx, label
说明	如果寄存器 RX 小于或等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLSZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

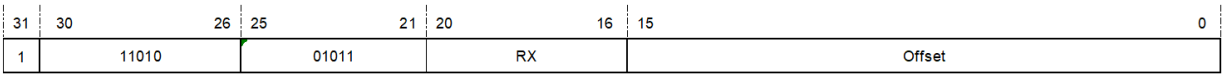


图 14.34: BLSZ

14.22 BLZ——寄存器小于零分支指令

统一化指令	
语法	blz rx, label

操作	寄存器小于零则程序转移
	if(RX < 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

编译结果	仅存在 32 位指令 blz32 rx, label
说明	如果寄存器 RX 小于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器小于零则程序转移
	if(RX < 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	blz32 rx, label
说明	如果寄存器 RX 小于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BLZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

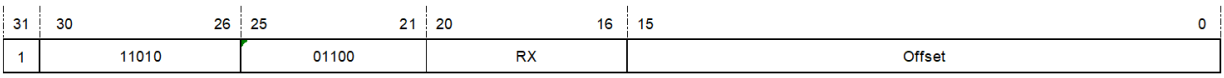


图 14.35: BLZ

14.23 BMASKI——立即数位屏蔽产生指令

统一化指令	
语法	bmaski rz, oimm5
操作	RZ ← $(2)^{\text{OIMM5}}$ - 1
编译结果	仅存在 32 位指令 bmaski32 rz, oimm5
说明	产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。 注意，OIMM5 为 1-16 时由 movi 指令执行。
影响标志位	无影响
限制	立即数的范围为 0，17-32；
异常	无

32 位指令	
操作	RZ ← $(2)^{\text{OIMM5}}$ - 1
语法	bmaski32 rz, oimm5
说明	产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。 注意，OIMM5 为 1 -16 时由 movi 指令执行；二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	无影响
限制	立即数的范围为 0，17-32；
异常	无

32 位指令格式：

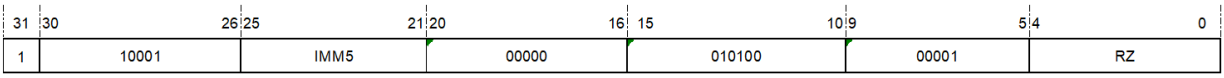


图 14.36: BMASKI

IMM5 域：

指定被置 1 的连续低位的最高位。

注意：立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

10000：

0-16 位置位

10001：

0-17 位置位

……

11111：

0-31 位置位

14.24 BNEZ——寄存器不等于零分支指令

统一化指令	
语法	bnez rx, label

操作	寄存器不等于零则程序转移
	if(RX != 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

编译结果	仅存在 32 位指令。 bnez32 rx, label
说明	如果寄存器 RX 不等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	

操作	寄存器不等于零则程序转移
	if(RX != 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bnez32 rx, label
说明	如果寄存器 RX 不等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BNEZ 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

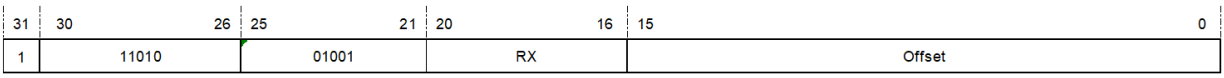


图 14.37: BNEZ

14.25 BR——无条件跳转指令

统一化指令	
语法	br label
操作	PC ← PC + sign_extend(offset << 1)

编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令
	if(offset<1KB), then
	br16 label;
	else
	br32 label;

说明	程序无条件跳转到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。
影响标志位	无影响
异常	无

16 位指令	
操作	PC ← PC + sign_extend(offset << 1)
语法	br16 label
说明	程序无条件跳转到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BR16 指令的跳转范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

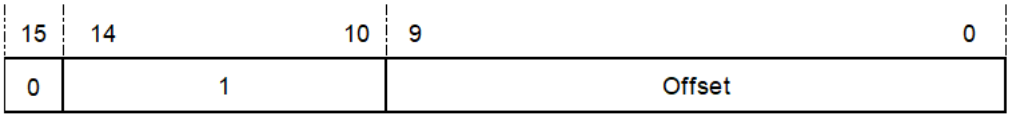


图 14.38: BR-1

32 位指令	
操作	PC ← PC + sign_extend(offset << 1)
语法	br32 label
说明	程序无条件跳转到 label 处执行。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BR 指令的跳转范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

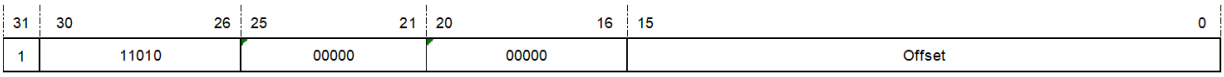


图 14.39: BR-2

14.26 BREV——位倒序指令

统一化指令	
语法	brev rz, rx
操作	for i=0 to 31 RZ[i] ← RX[31-i];
编译结果	仅存在 32 位指令。 brev32 rz, rx
说明	把 RX 的值按位取倒序，结果存入 RZ。 如果 RX 的值为”abc defghijklmnopqrstuvwxyz012345”，按位取倒序后，RZ 的值变为”543210zyxwvutsrqponmlkjihgfedcba”。
影响标志位	无影响
异常	无

32 位指令	
操作	for i=0 to 31 RZ[i] ← RX[31-i];
语法	brev32 rz, rx
说明	把 RX 的值按位取倒序，结果存入 RZ。 如果 RX 的值为”abc defghijklmnopqrstuvwxyz012345”，按位取倒序后，RZ 的值变为”543210zyxwvutsrqponmlkjihgfedcba”。
影响标志位	无影响
异常	无

指令格式：

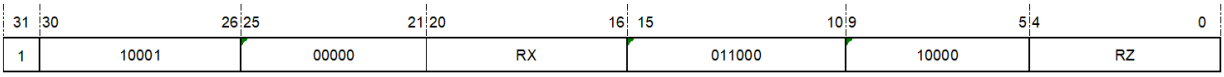


图 14.40: BREV

14.27 BSETI——立即数位置位指令

统一化指令		
语法	bseti rz, imm5	bseti rz, rx, imm5
操作	RZ ← RZ[IMM5] 置位	RZ ← RX[IMM5] 置位

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<8), then
	bseti16 rz, imm5;
	else
	bseti32 rz, rz, imm5;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if((x==z) and (z<8), then
		bseti16 rz, imm5;
		else
		bseti32 rz, rx, imm5;

说明	将 RZ/RX 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。	
影响标志位	无影响	
限制	立即数的范围为 0-31。	
异常	无	

16 位指令	
操作	RZ ← RZ[IMM5] 置位
语法	bseti16 rz, imm5
说明	将 RZ 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

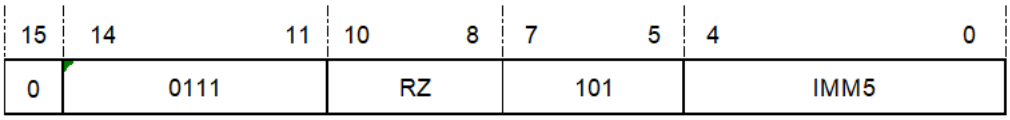


图 14.41: BSETI-1

32 位指令	
操作	RZ ← RX[IMM5] 置位
语法	bseti32 rz, rx, imm5
说明	将 RX 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

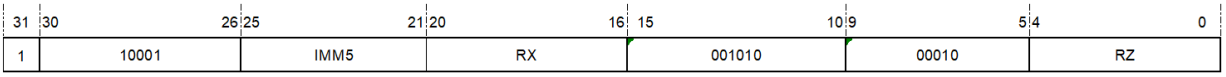


图 14.42: BSETI-2

14.28 BSR——跳转到子程序指令

统一化指令	
语法	bsr label
操作	链接并跳转到子程序： R15 ← next PC PC ← PC + sign_extend(offset << 1)

编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令
	if(0<offset<1KB), then
	bsr16 label;
	else
	bsr32 label;

说明:	**子程序跳转，将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 label 处执行。** **Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。**
影响标志位:	无影响
异常:	无

16 位指令	
操作:	链接并跳转到子程序: R15 ← PC + 2 PC ← PC + sign_extend(offset << 1)
语法:	bsr16 label
说明:	子程序跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC + 2）保存在链接寄存器 R15 中，程序转移到 label 处执行。 Label 由当前程序 P C 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BSR16 指令的跳转范围是 ±1KB 地址空间。
影响标志位:	无影响
限制:	BSR16 指令的跳转目标不能是 BSR16 指令本身。
异常:	无

指令格式:

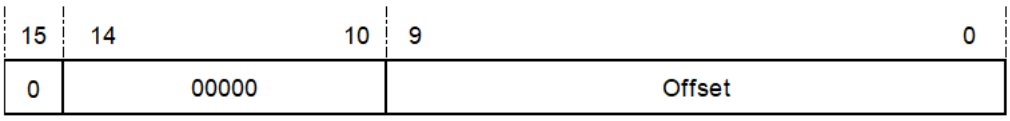


图 14.43: BSR-1

Offset 域——指定跳转的相对偏移量。

注意：跳转的相对偏移量（Offset）不能为 0x0。

32位指令	
操作:	链接并跳转到子程序: R15 ← PC+4 PC ← PC + sign_extend(offset << 1)
语法:	bsr32 label
说明:	子程序跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序转移到 label 处执行。 Label 由当前程序 PC 加上左移 1 位的 26 位相对偏移量有符号扩展到 32 位后的值得到。BSR 指令的跳转范围是 ±64MB 地址空间。
影响标志位:	无影响
异常:	无

指令格式:

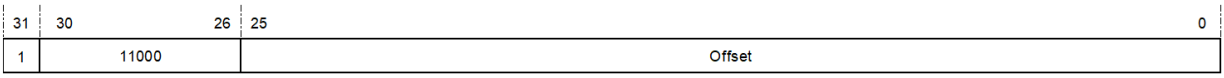


图 14.44: BSR-2

14.29 BT——C 为 1 分支指令

统一化指令	
语法	bt label

操作	if(C == 1)
	PC ← PC + sign_extend(offset << 1);
	else
	PC ← next PC;

编译结果	根据跳转的范围编译为对应的 16 位或 32 位指令。
	if (offset<1KB)，then
	bt16 label;
	else
	bt32 label;

说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令。 Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

16 位指令	

操作	C 等于一则程序转移
	if(C == 1)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 2

语法	bt16 label
说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。 Label 由当前程序 P C 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BT16 指令的转移范围是 ±1KB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

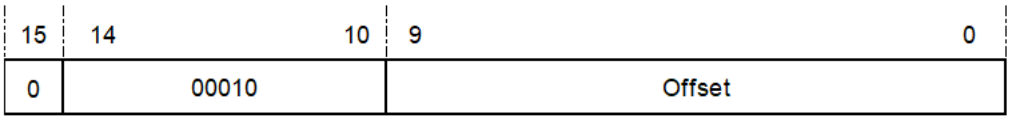


图 14.45: BT-1

32 位指令	

操作	C 等于一则程序转移
	if(C == 1)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法	bt32 label
说明	如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。 Label 由当前程序 P C 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是 ±64KB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

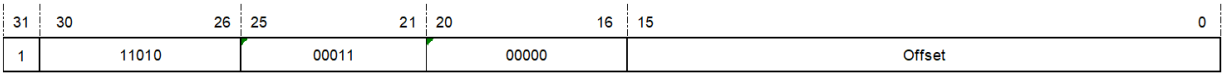


图 14.46: BT-2

14.30 BTSTI——立即数位测试指令

统一化指令	
语法	btsti rx, imm5
操作	C ← RX[IMM5]
编译结果	仅存在 32 位指令 btsti32 rx, imm5
说明	对由 IMM5 决定的 RX 的位（RX[IMM5]）进行测试，并使条件位 C 的值等于该位的值。
影响标志位	C ← RX[IMM5]
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	C ← RX[IMM5]
语法	btsti32 rx, imm5
说明	对由 IMM5 决定的 RX 的位（RX[IMM5]）进行测试，并使条件位 C 的值等于该位的值。
影响标志位	C ← RX[IMM5]
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

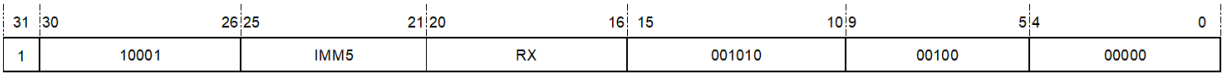


图 14.47: BTSTI

14.31 CLRF——C 为 0 清零指令

统一化指令	
语法	clrf rz

操作	if C==0, then
	RZ ← 0;
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 clrf32 rz
说明	如果 C 为 0，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令	

操作	if C==0, then
	RZ ← 0;
	else
	RZ ← RZ;

语法	clrf32 rz
说明	如果 C 为 0，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令格式：

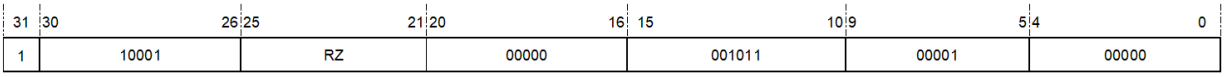


图 14.48: CLRF

14.32 CLRT——C 为 1 清零指令

统一化指令	
语法	clrt rz

操作	if C==1, then
	RZ ← 0;
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 clrt32 rz
说明	如果 C 为 1，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令	

操作	if C==1, then
	RZ ← 0;
	else
	RZ ← RZ;

语法	clrt32 rz
说明	如果 C 为 1，寄存器 RZ 被清零；否则，寄存器 RZ 保持不变。
影响标志位	无影响
异常	无

32 位指令格式：

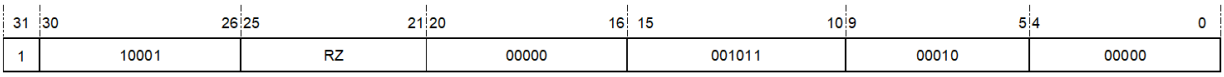


图 14.49: CLRT

14.33 CMPHS——无符号大于等于比较指令

统一化指令	
语法	cmphs rx, ry

操作	RX 与 RY 作无符号比较。
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	cmphs16 rx, ry;
	else
	cmphs32 rx, ry;

说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmphs 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	

操作	RX 与 RY 作无符号比较。
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;

语法	cmphs16 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmphs16 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

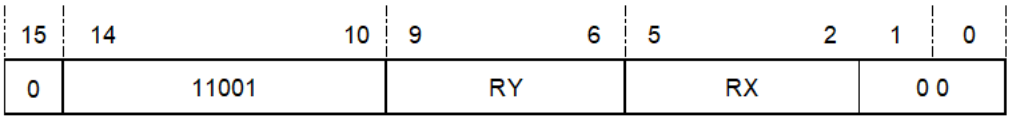


图 14.50: CMPHS-1

32 位指令	

操作	RX 与 RY 作无符号比较。
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;

语法	cmphs32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmphs32 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

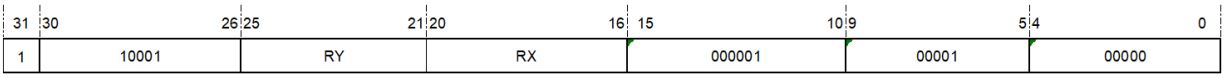


图 14.51: CMPHS-2

14.34 CMPHSI——立即数无符号大于等于比较指令

统一化指令	
语法	cmphsi rx, oimm16
操作	RX 与立即数作无符号比较。 If RX >= zero_extend(OIMM16), C ← 1; else C ← 0;

编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (oimm16<33) and (x<8),then
	cmphsi16 rx, oimm5;
	else
	cmphsi32 rx, oimm16;

说明	将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

16 位指令	

操作	RX 与立即数作无符号比较。
	If RX >= zero_extend(OIMM5), then
	C ← 1;
	else
	C ← 0;

语法	cmphsi16 rx, oimm5
说明	将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi16 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM5，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7；立即数的范围为 1-32。
异常	无

16 位指令格式：

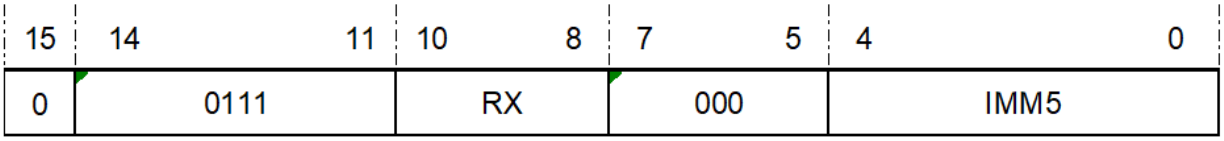


图 14.52: CMPHSI-1

IMM5 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

与 1 比较

00001：

与 2 比较

……

11111：

与 32 比较

32 位指令	

操作	RX 与立即数作无符号比较。
	If RX >= zero_extend(OIMM16), then
	C ← 1;
	else
	C ← 0;

语法	cmphsi32 rx, oimm16
说明	将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi32 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。注意：二进制操作数 IMM16 等于 OIMM16 - 1。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

32 位指令格式：

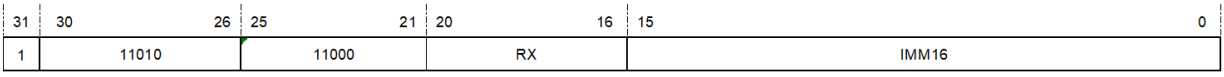


图 14.53: CMPHSI-2

IMM16 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

0000000000000000：

与 0x1 比较

0000000000000001：

与 0x2 比较

……

1111111111111111：

与 0x10000 比较

14.35 CMPLT——有符号小于比较指令

统一化指令	
语法	cmplt rx, ry

操作	RX 与 RY 作有符号比较。
	If RX < RY, then
	C ← 1;
	else
	C ← 0;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	cmplt16 rx, ry;
	else
	cmplt32 rx, ry;

说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	

操作	RX 与 RY 作有符号比较。
	If RX < RY, then
	C ← 1;
	else
	C ← 0;

语法	cmplt16 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt16 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

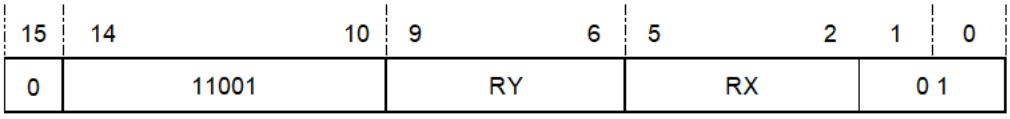


图 14.54: CMPLT-1

32 位指令	

操作	RX 与 RY 作有符号比较。
	If RX < RY, then
	C ← 1;
	else
	C ← 0;

语法	cmplt32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt32 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

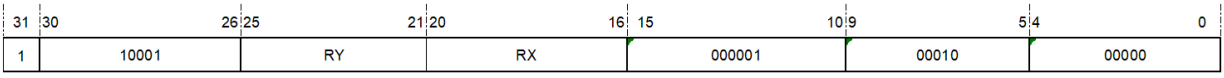


图 14.55: CMPLT-2

14.36 CMPLTI——立即数有符号小于比较指令

统一化指令	
语法	cmplti rx, oimm16
操作	RX 与立即数作有符号比较。 If RX < zero_extend(OIMM16), C ← 1; else C ← 0;

编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<8) and (oimm16<33), then
	cmplti16 rx, oimm5;
	else
	cmplti32 rx, oimm16;

说明	将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmplti 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

16 位指令	

操作	RX 与立即数作有符号比较。
	If RX < zero_extend(OIMM5), then
	C ← 1;
	else
	C ← 0;

语法	cmplti16 rx, oimm5
说明	将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cm plti16 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM5，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7；立即数的范围为 1-32。
异常	无

16 位指令格式：

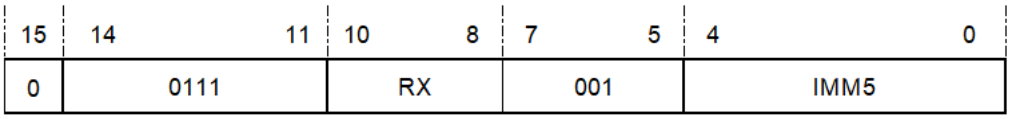


图 14.56: CMPLT-1

IMM5 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

与 1 比较

00001：

与 2 比较

……

11111：

与 32 比较

32 位指令	

操作	RX 与立即数作有符号比较。
	If RX < zero_extend(OIMM16), then
	C ← 1;
	else
	C ← 0;

语法	cmplti32 rx, oimm16
说明	将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cm plti32 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。 注意：二进制操作数 IMM16 等于 OIMM16 - 1。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x1-0x10000。
异常	无

32 位指令格式：

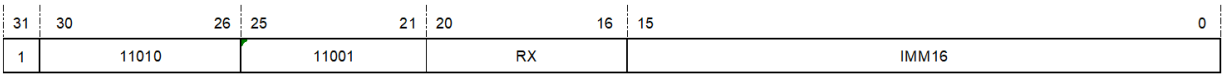


图 14.57: CMPLT-2

IMM16 域：

指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

0000000000000000：

与 0x1 比较

0000000000000001：

与 0x2 比较

……

1111111111111111：

与 0x10000 比较

14.37 CMPNE——不等比较指令

统一化指令	
语法	cmpne rx, ry

操作	RX 与 RY 作比较。
	If RX != RY, then
	C ← 1;
	else
	C ← 0;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	cmpne16 rx, ry;
	else
	cmpne32 rx, ry;

说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

16 位指令	

操作	RX 与 RY 作比较。
	If RX != RY, then
	C ← 1;
	else
	C ← 0;

语法	cmpne16 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

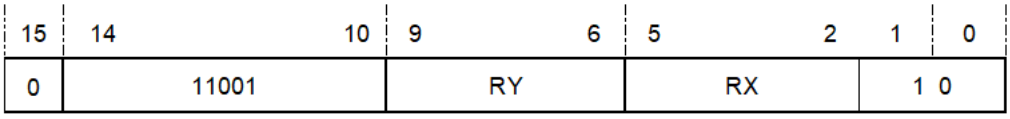


图 14.58: CMPNE-1

32 位指令	

操作	RX 与 RY 作比较。
	If RX != RY, then
	C ← 1;
	else
	C ← 0;

语法	cmpne32 rx, ry
说明	将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
异常	无

32 位指令格式：

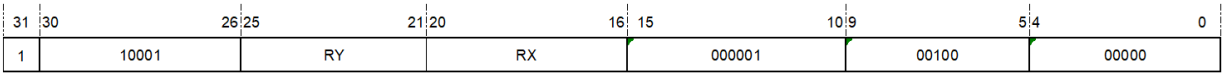


图 14.59: CMPNE-2

14.38 CMPNEI——立即数不等比较指令

统一化指令	
语法	cmpnei rx, imm16
操作	RX 与立即数作比较。 If RX != zero_extend(imm16), C ← 1; else C ← 0;

编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<7) and (imm16<33), then
	cmpnei16 rx, imm5;
	else
	cmpnei32 rx, imm16;

说明	将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x0-0xFFFF。
异常	无

16 位指令	

操作	RX 与立即数作比较。
	If RX != zero_extend(IMM5), then
	C ← 1;
	else
	C ← 0;

语法	cmpnei16 rx, imm5
说明	将 RX 的值减去零扩展至 32 位的 5 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM5，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	寄存器的范围为 r0-r7; 立即数的范围为 0-31。
异常	无

16 位指令格式：

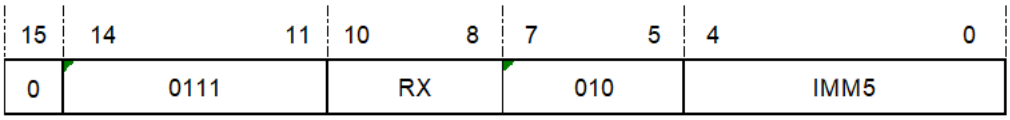


图 14.60: CMPNEI-1

32 位指令	

操作	RX 与立即数作比较。
	If RX != zero_extend(imm16), then
	C ← 1;
	else
	C ← 0;

语法	cmpnei rx, imm16
说明	将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据比较结果设置条件位 C
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

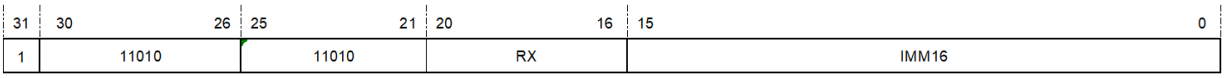


图 14.61: CMPNEI-2

14.39 DECF——C 为 0 立即数减法指令

统一化指令	
语法	decf rz, rx, imm5

操作	if C==0, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 decf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	if C==0, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;

语法	decf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

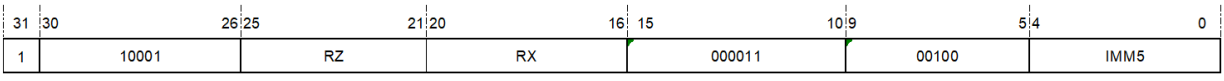


图 14.62: DECF

14.40 DECGT——减法大于零置 C 位指令

统一化指令	
语法	decgt rz, rx, imm5

操作	RZ ← RX - zero_extend(IMM5);
	If RZ > 0, then
	C ← 1;
	else
	C ← 0;

编译结果	仅存在 32 位指令 decgt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果大于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果大于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	RZ ← RX - zero_extend(IMM5);
	If RZ > 0, then
	C ← 1;
	else
	C ← 0;

语法	decgt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果大于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果大于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

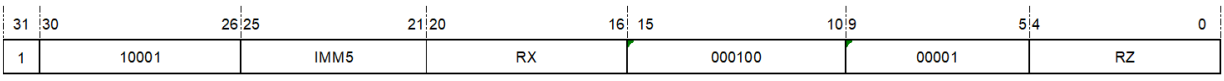


图 14.63: DECGT

14.41 DECLT——减法小于零置 C 位指令

统一化指令	
语法	declt rz, rx, imm5

操作	RZ ← RX - zero_extend(IMM5);
	If RZ < 0, then
	C ← 1;
	else
	C ← 0;

编译结果	仅存在 32 位指令 declt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果小于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果小于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	RZ ← RX - zero_extend(IMM5);
	If RZ < 0, then
	C ← 1;
	else
	C ← 0;

语法	declt32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。减法结果被认为是补码形式的有符号数。如果结果小于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果小于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

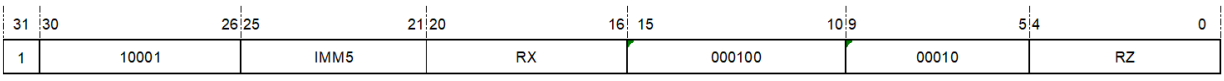


图 14.64: DECLT

14.42 DECNE——减法不等于零置 C 位指令

统一化指令	
语法	decne rz, rx, imm5

操作	RZ ← RX - zero_extend(IMM5);
	If RZ != 0, then
	C ← 1;
	else
	C ← 0;

编译结果	仅存在 32 位指令 decne32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。如果结果不等于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果不等于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	RZ ← RX - zero_extend(IMM5);
	If RZ != 0, then
	C ← 1;
	else
	C ← 0;

语法	decne32 rz, rx, imm5
说明	将 5 位立即数零扩展至 32 位，把 RX 减去该 32 位值的结果存在 RZ。如果结果不等于 0，设置条件位 C；否则清除条件位 C。
影响标志位	如果减法结果不等于 0，设置条件位 C；否则清除条件位 C。
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

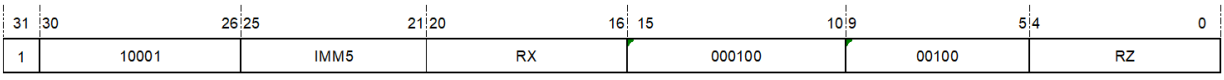


图 14.65: DECNE

14.43 DECT——C 为 1 立即数减法指令

统一化指令	
语法	dect rz, rx, imm5

操作	if C==1, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 dect32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	if C==1, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;

语法	dect32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

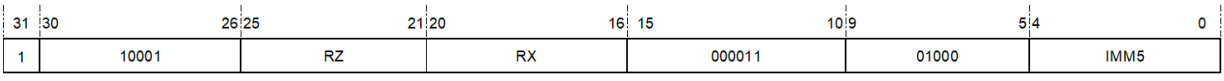


图 14.66: DECT

14.44 DIVS——有符号除法指令

统一化指令	
语法	divs rz, rx, ry
操作	有符号除法 RZ = RX / RY
编译结果	仅存在 32 位指令 divs32 rz, rx, ry
说明	有符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位有符号数。 注意，对于 0x80 000000 除以 0xffffffff 这种情况，结果没有定义。
影响标志位	不影响
异常	除零异常

32 位指令	
操作	有符号除法 RZ = RX / RY
语法	divs32 rz, rx, ry
说明	有符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位有符号数。 注意，对于 0x80000000 除以 0xffffffff 这种情况，结果没有定义。
影响标志位	不影响
异常	除零异常

32 位指令格式：

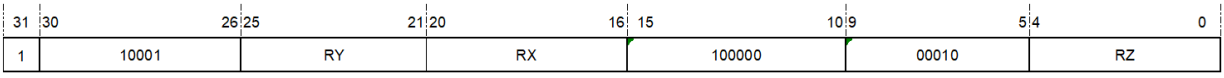


图 14.67: DIVS

14.45 DIVU——无符号除法指令

统一化指令	
语法	divu rz, rx, ry
操作	无符号除法 RZ = RX / RY
说明	仅存在 32 位指令 divu32 rz, rx, ry
说明	无符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位无符号数。
影响标志位	不影响
异常	除零异常

32 位指令	
操作	无符号除法 RZ = RX / RY
语法	divu32 rz, rx, ry
说明	无符号寄存器除法指令将寄存器 RX 的值除以寄存器 RY 的值，得到的商存放于 RZ 中。RX、RY 和 RZ 的值均被认为是 32 位无符号数。
影响标志位	不影响
异常	除零异常

32 位指令格式：

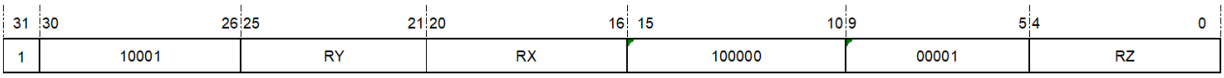


图 14.68: DIVU

14.46 DOZE——进入低功耗睡眠模式指令

统一化指令	
语法	doze
操作	进入低功耗睡眠模式
编译结果	仅存在 32 位指令 doze32
说明	此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。
影响标志位	不影响
异常	特权违反异常

32 位指令	
操作	进入低功耗睡眠模式
语法	doze32
属性：	特权指令
说明	此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。
影响标志位	不影响
异常	特权违反异常

32 位指令格式：

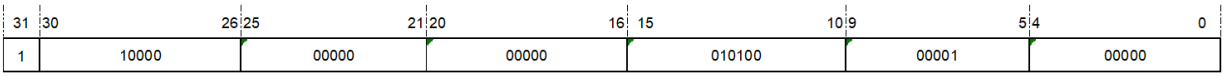


图 14.69: DOZE

14.47 FF0——快速找 0 指令

统一化指令	
语法	ff0 rz, rx
操作	RZ ← find_first_0(RX);
编译结果	仅存在 32 位指令 ff0.32 rz, rx
说明	查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← find_first_0(RX);
语法	ff0.32 rz, rx
说明	查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

指令格式：

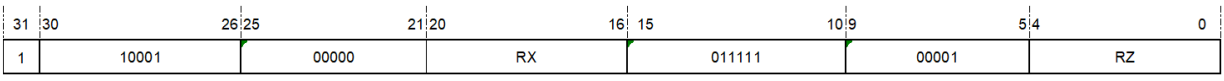


图 14.70: FF0

14.48 FF1——快速找 1 指令

统一化指令	
语法	ff1 rz, rx
操作	RZ ← find_first_1(RX);
编译结果	仅存在 32 位指令 ff1.32 rz, rx
说明	查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← find_first_1(RX);
语法	ff1.32 rz, rx
说明	查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。
影响标志位	无影响
异常	无

指令格式：

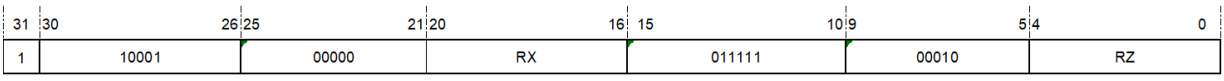


图 14.71: FF1

14.49 GRS——符号产生指令

统一化指令	
语法	grs rz, label grs rz, imm32
操作	RZ ← PC + sign_extend(offset << 1);
编译结果	仅存在 32 位指令。 grs32 rz, label grs32 rz, imm32
说明	产生符号的值，该值由 label 所在位置，或 32 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到 32 位后的值得到。符号的值的有效范围是 ±256KB 地址空间。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← PC + sign_extend(offset << 1);
语法	grs rz, label grs rz, imm32
说明	产生符号的值，该值由 label 所在位置，或 3 2 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到 32 位后的值得到。符号的值的有效范围是 ±256KB 地址空间。
影响标志位	无影响
异常	无

指令格式：

图 14.72: GRS

14.50 IDLY——中断识别禁止指令

统一化指令	
语法	idly
操作	禁止中断识别 4 个指令
编译结果	仅存在 32 位指令 idly32
说明	idly 指令后 4 条指令禁止中断识别，这样就允许一个不可中断指令序列在多任务环境中被执行。
影响标志位	标志位 C 在 idly 指令执行后被清零。如果在 idl y 指令执行以后的 4 个指令中发生异常（包括跟踪或断点异常），位 C 被置 1，中断指令序列就能被观测到。
限制	idly 指令后面的指令只能是单时钟周期的算术、逻辑指令，ld, st, 或分支指令。为了使一些潜在中断的影响达到最小，如果是其他的指令，就不能保证不会被中断。如果在 idly 后面的指令序列中有另一个 idly 指令，那么它也将被忽略。但是如果是 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idly 指令序列的中止。 idly 指令不允许在一个小于 8 条指令的循环中出现。
异常	无
备注:	如果 idly 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idly 指令序列中发生一个断点异常或一个跟踪异常，那么位 C 会被置 1，这样序列将会出现操作失败。在异常处理过程中，使中断屏蔽无效，从而让计数器清零。 idly 的计数器在使用 H AD 调试端口进行调试过程中不会变化，一旦处理器从调试模式中释放到正常的操作，那么计数就会继续。

32 位指令	
操作	禁止中断识别 4 个指令 disable_int_in_following(4);
语法	idly32
说明	idly 指令后 4 条指令禁止中断识别，这样就允许一个不可中断指令序列在多任务环境中被执行。
影响标志位	标志位 C 在 idly 指令执行后被清零。如果在 idly 指令执行以后的 4 个指令中发生异常（包括跟踪或断点异常），位 C 被置 1，中断指令序列就能被观测到。
限制	idly 指令后面的指令只能是单时钟周期的算术、逻辑指令，ld, st, 或分支指令。为了使一些潜在中断的影响达到最小，如果是其他的指令，就不能保证不会被中断。如果在 idly 后面的指令序列中有另一个 idly 指令，那么它也将被忽略。但是如果是 rte, rfi, doze, wait, stop 等指令，那么他们就会引起 idly 指令序列的中止。 idly 指令不允许在一个小于 8 条指令的循环中出现。
异常	无
备注：	如果 idly 计数器停在不为零的状态时，中断就会被屏蔽。如果在 idly 指令序列中发生一个断点异常或一个跟踪异常，那么位 C 会被置 1，这样序列将会出现操作失败。在异常处理过程中，使中断屏蔽无效，从而让计数器清零。 idly 的计数器在使用 HAD 调试端口进行调试过程中不会变化，一旦处理器从调试模式中释放到正常的操作，那么计数就会继续。

32 位指令格式：

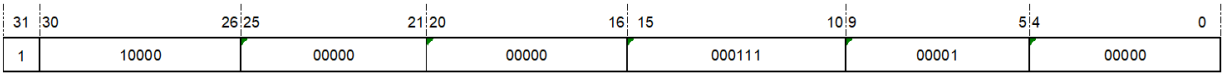


图 14.73: IDLY

14.51 INCF——C 为 0 立即数加法指令

统一化指令	
语法	incf rz, rx, imm5

操作	if C==0, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 incf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	if C==0, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;

语法	incf32 rz, rx, imm5
说明	如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

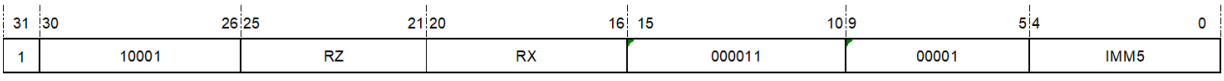


图 14.74: INCF

14.52 INCT——C 为 1 立即数加法指令

统一化指令	
语法	inct rz, rx, imm5

操作	if C==1, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令 inct32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	

操作	if C==1, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;

语法	inct32 rz, rx, imm5
说明	如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

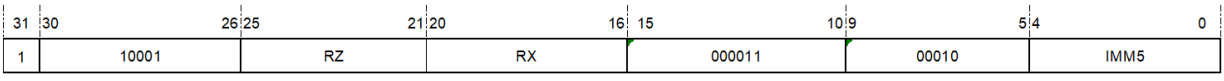


图 14.75: INCT

14.53 INS——位插入指令

统一化指令	
语法	ins rz, rx, msb, lsb
操作	RZ[MSB:LSB] ← RX[MSB-LSB:0]
编译结果	仅存在 32 位指令 ins32 rz, rx, msb, lsb
说明	将 RX 的一段连续低位，插入到由 2 个 5 位立即数（MSB,LSB）指定的 RZ 的一段连续位（RZ[MSB:LSB]）上，RZ 的其它位保持不变，RX 连续低位的宽度由 MSB 和 LSB 指定（即 RX[MSB-LSB:0]）。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的 MSB（即 LSB）位为 RX 的最低位，其它位保持不变。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令	
操作	RZ[MSB:LSB] ← RX[MSB-LSB:0]
语法	ins32 rz, rx, msb, lsb
说明	将 RX 的一段连续低位，插入到由 2 个 5 位立即数（MSB,LSB）指定的 RZ 的一段连续位（RZ[MSB:LSB]）上，RZ 的其它位保持不变，RX 连续低位的宽度由 MSB 和 LSB 指定（即 RX[MSB-LSB:0]）。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的 MSB（即 LSB）位为 RX 的最低位，其它位保持不变。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式：

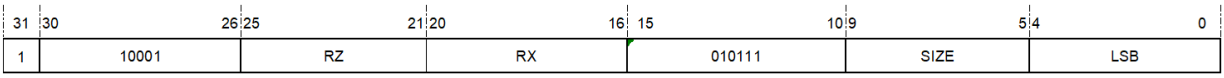


图 14.76: INS

SIZE 域：

指定插入位的宽度。

注意：二进制操作数 SIZE 等于 MSB-LSB。

00000：

1

00001：

2

……

11111：

32

LSB 域：

指定插入结束的位。

00000：

0 位

00001：

1 位

……

11111：

31 位

14.54 IXH——索引半字指令

统一化指令	
语法	ixh rz, rx, ry
操作	RZ ← RX + (RY << 1)
编译结果	仅存在 32 位指令 ixh32 rz, rx, ry
说明	将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← RX + (RY << 1)
语法	ixh32 rz, rx, ry
说明	将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

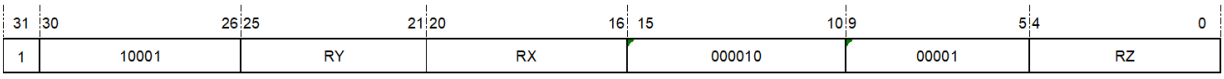


图 14.77: IXH

14.55 IXW——索引字指令

统一化指令	
语法	ixw rz, rx, ry
操作	RZ ← RX + (RY << 2)
编译结果	仅存在 32 位指令 ixw32 rz, rx, ry
说明	将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← RX + (RY << 2)
语法	ixw32 rz, rx, ry
说明	将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

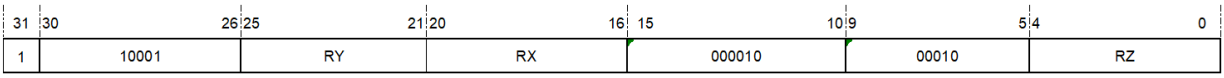


图 14.78: IXW

14.56 IXD——索引双字指令

统一化指令	
语法	ixd rz, rx, ry
操作	RZ ← RX + (RY << 3)
编译结果	仅存在 32 位指令 ixd32 rz, rx, ry
说明	将 RY 的值左移三位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← RX + (RY << 3)
语法	ixd32 rz, rx, ry
说明	将 RY 的值左移三位后加上 RX 的值，并把结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

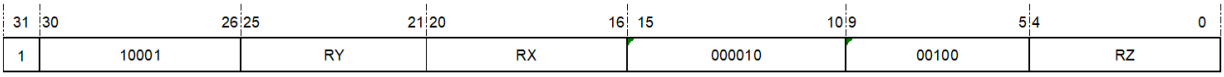


图 14.79: IXD

14.57 JMP——寄存器跳转指令

统一化指令	
语法	jmp rx
操作	跳转到寄存器指定的位置 PC ← RX & 0xfffffffe

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16), then
	jmp16 rx;
	else
	jmp32 rx;

说明	程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令	
操作	跳转到寄存器指定的位置 PC ← RX & 0xfffffffe
语法	jmp16 rx
说明	程序跳转到寄存器 RX 指定的位置, RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

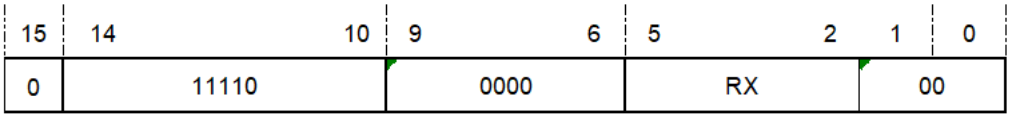


图 14.80: JMP-1

32 位指令	
操作	跳转到寄存器指定的位置 PC ← RX & 0xfffffffe
语法	jmp32 rx
说明	程序跳转到寄存器 RX 指定的位置, RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

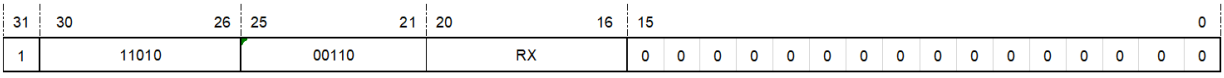


图 14.81: JMP-2

14.58 JMPI——间接跳转指令

统一化指令	
语法	jmpi label
操作	程序跳转到存储器指定的位置 PC ← MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc]
编译结果	仅存在 32 位指令。 jmpi32 label
说明	程序跳转到 label 所在的位置，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JMPI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	程序跳转到存储器指定的位置 PC ← MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc]
语法	jmpi32 label
说明	程序跳转到 label 所在的位置，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JMPI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

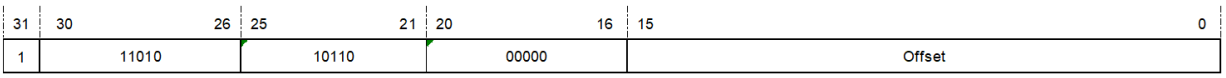


图 14.82: JMPI

14.59 JSR——寄存器跳转到子程序指令

统一化指令	
语法	jsr rx
操作	链接并跳转到寄存器指定的子程序位置 R15 ← PC + 4, PC ← RX & 0xfffffffe

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16), then
	jsr16 rx;
	else
	jsr32 rx;

说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令	
操作	链接并跳转到寄存器指定的子程序位置 R15 ← PC + 4，PC ← RX & 0xfffffffe
语法	jsr16 rx
说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

16 位指令格式：

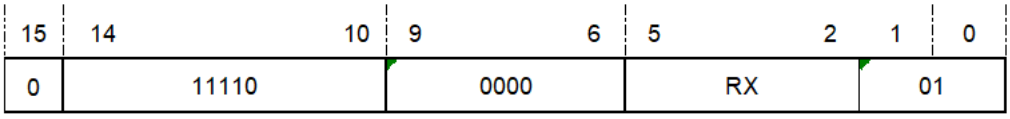


图 14.83: JSR-1

32 位指令	
操作	链接并跳转到寄存器指定的子程序位置 R15 ← PC + 4，PC ← RX & 0xfffffffe
语法	jsr32 rx
说明	子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	无

32 位指令格式：

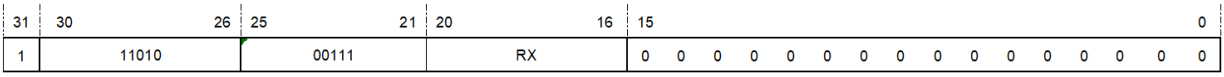


图 14.84: JSR-2

14.60 JSRI——间接跳转到子程序指令

统一化指令	
语法	jsri label
操作	程序跳转到存储器指定的子程序位置 R15 ← next PC, PC ← MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc]
编译结果	仅存在 32 位指令。 jsri32 label;
说明	子程序间接跳转，将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序跳转到 label 所在的位置处执行，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JSRI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	程序跳转到存储器指定的子程序位置 R15 ← PC + 4，PC ← MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc]
语法	jsri32 label
说明	子程序间接跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到 label 所在的位置处执行，label 由存储器加载得到。存储器地址根据 PC 加上左移两位的 16 位相对偏移量无符号扩展到 32 位后，再经最低两位强制清零得到。JSRI 指令的跳转范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

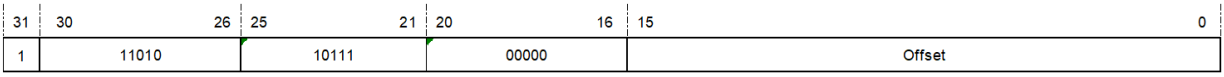


图 14.85: JSRI

14.61 LD.B——无符号扩展字节加载指令

统一化指令	
语法	ld.b rz,(rx, disp)
操作	RZ ← zero_extend(MEM[RX + zero_extend(offset)])

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (disp<32)and(x<7) and (z<7), then
	ld16.b rz, (rx, disp);
	else
	ld32.b rz, (rx, disp);

说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。 LD.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + zero_extend(offset)])
语法	ld16.b rz, (rx, disp)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.B 指令可以寻址 +32B 的地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

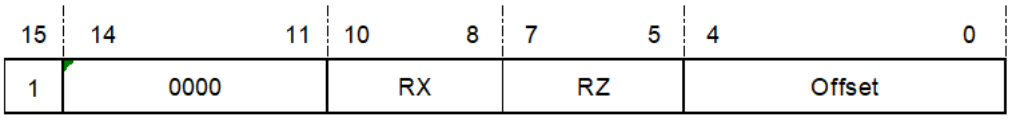


图 14.86: LD.B-1

32 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + zero_extend(offset)])
语法	ld32.b rz, (rx, disp)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

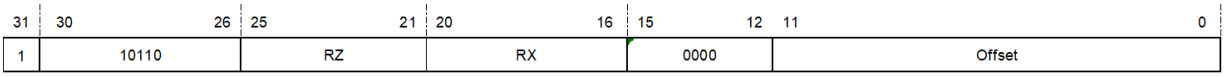


图 14.87: LD.B-2

14.62 LD.BS——有符号扩展字节加载指令

统一化指令	
语法	ld.bs rz, (rx, disp)
操作	RZ ← sign_extend(MEM[RX + zero_extend(offset)])
编译结果	仅存在 32 位指令。 ld32.bs rz, (rx, disp)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.BS 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + zero_extend(offset)])
语法	ld32.bs rz, (rx, disp)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.BS 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

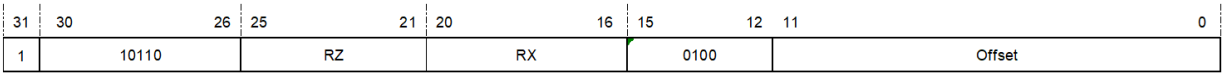


图 14.88: LD.BS

14.63 LD.D——双字加载指令

统一化指令	
语法	ld.d rz, (rx, disp)
操作	RZ ← MEM[RX + zero_extend(offset << 2)] RZ + 1 ← MEM[RX + zero_extend(offset << 2) + 0x4]
编译结果	仅存在 32 位指令。 ld32.d rz, (rx, disp);
说明	从存储器加载双字到寄存器 RZ、RZ + 1 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.D 指令可以寻址 +16KB 地址空间。 注意：1. 偏移量 DISP 是二进制操作数 Offset 左移两位得到的。 2.rz 和 rx 不可以相同，否则结果将不可预期。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载双字到寄存器 RZ ← MEM[RX + zero_extend(offset << 2)] RZ + 1 ← MEM[RX + zero_extend(offset << 2) + 0x4]
语法	ld32.d rz, (rx, disp)
说明	从存储器加载字到寄存器 RZ、RZ + 1 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.D 指令可以寻址 +16KB 地址空间。 注意： 1. 偏移量 DISP 是二进制操作数 Offset 左移两位得到的。 2.rz 和 rx 不可以相同，否则结果将不可预期。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

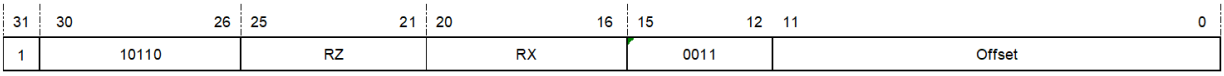


图 14.89: LD.D

14.64 LD.H——无符号扩展半字加载指令

统一化指令	
语法	ld.h rz, (rx, disp)
操作	RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (disp<64)and(x<7) and (z<7), then
	ld16.h rz, (rx, disp);
	else
	ld32.h rz, (rx, disp);

说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.H 指令可以寻址 +8KB 地址空间。 注意：偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])
语法	ld16.h rz, (rx, disp)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.H 指令可以寻址 +64B 的地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

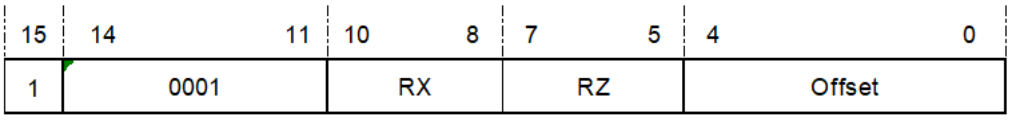


图 14.90: LD.H-1

32 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])
语法	ld32.h rz, (rx, disp)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.H 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

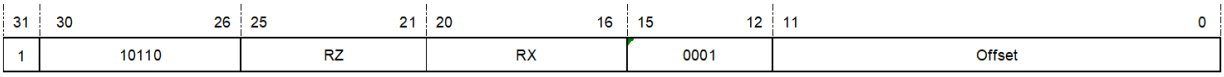


图 14.91: LD.H-2

14.65 LD.HS——有符号扩展半字加载指令

统一化指令	
语法	ld.hs rz, (rx, disp)
操作	RZ ← sign_extend(MEM[RX + zero_extend(offset << 1)])
编译结果	仅存在 32 位指令。 ld32.hs rz, (rx, disp)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.HS 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + zero_extend(offset << 1)])
语法	ld32.hs rz, (rx, disp)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.HS 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

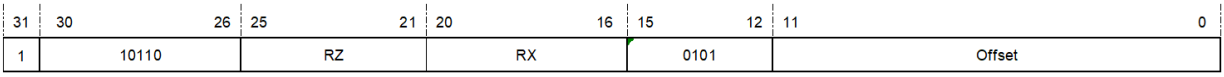


图 14.92: LD.HS

14.66 LD.W——字加载指令

统一化指令	
语法	ld.w rz, (rx, disp)
操作	RZ ← MEM[RX + zero_extend(offset << 2)]

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x=sp) and (z<7) and (disp < 1024),
	ld16.w rz, (sp, disp);
	else if (disp<128) and (x<7) and (z<7),
	ld16.w rz, (rx, disp);
	else
	ld32.w rz, (rx, disp);

说明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字到寄存器 RZ ← MEM[RX + sign_extend(offset << 2)]
语法	ld16.w rz, (rx, disp) ld16.w rz, (sp, disp)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。当 RX 为 SP 时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.W 指令可以寻址 +1KB 的地址空间。 注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数 {IMM3, IMM5} 左移两位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常。

16 位指令格式：

ld16.w rz, (rx, disp)

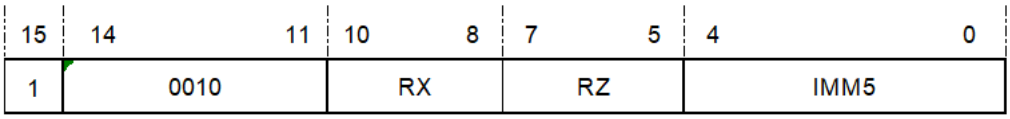


图 14.93: LD.W-1

ld16.w rz, (sp, disp)

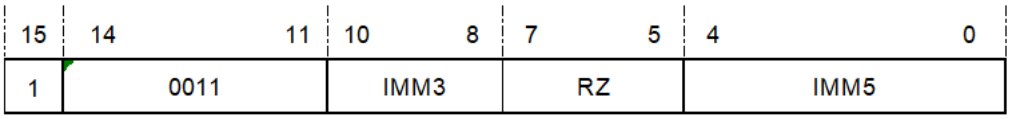


图 14.94: LD.W-2

32 位指令	
操作	从存储器加载字到寄存器 RZ ← MEM[RX + zero_extend(offset << 2)]
语法	ld32.w rz, (rx, disp)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

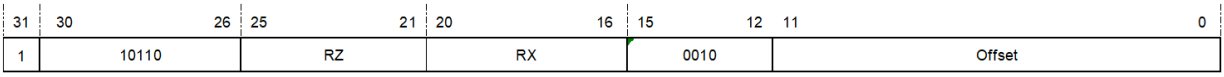


图 14.95: LD.W-3

14.67 （手动加）LDEX.W——独占式字加载指令

统一化指令	
语法	ldex.w rz, (rx, disp)
操作	RZ MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 位指令。 ldex32.w rz, (rx, disp)

说明:	从存储器加载字到通用寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LDEX.W 指令可以寻址 +16KB 地址空间。 该指令与 STEX.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位:	无影响
异常:	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令

操作:	从存储器加载字到通用寄存器 RZ MEM[RX + zero_extend(offset << 2)]
语法:	ldex32.w rz, (rx, disp)
说明:	从存储器加载字到通用寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LDEX32.W 指令可以寻址 +16KB 地址空间。 该指令与 STEX32.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位:	无影响
异常:	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式:

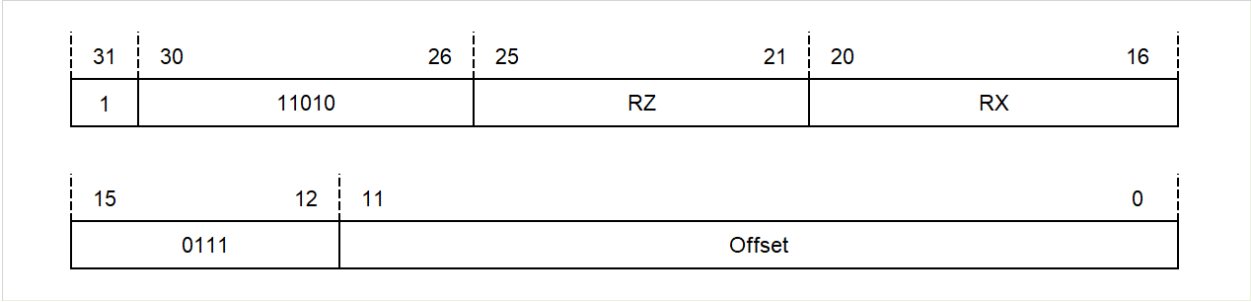


图 14.96: LDEX.W

14.68 LDM——连续多字加载指令

统一化指令	
语法	ldm ry-rz, (rx)

操作	从存储器加载连续的多个字到一片连续的寄存器堆中
	dst ← Y; addr ← RX;
	for (n = 0; n <= (Z-Y); n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}

编译结果	仅存在 32 位指令。 ldm32 ry-rz, (rx);
说明	从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。 RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	

操作	从存储器加载连续的多个字到一片连续的寄存器堆中
	dst ← Y; addr ← RX;
	for (n = 0; n <= IMM5; n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}

语法	ldm32 ry-rz, (rx)
说明	从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。 RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

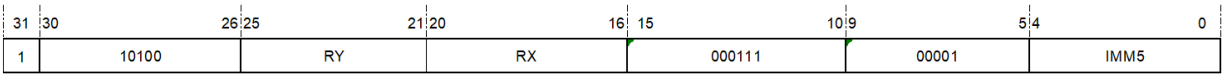


图 14.97: LDM

IMM5 域：

指定目标寄存器的个数，IMM5 = Z - Y。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

……

11111：

32 个目的寄存器

14.69 LDQ——连续四字加载指令

统一化指令	
语法	ldq r4-r7, (rx)

操作	从存储器加载连续的四个字到寄存器 R4—R7 中
	dst ← 4; addr ← RX;
	for (n = 0; n <= 3; n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}

编译结果	仅存在 32 位指令。 ldq32 r4-r7, (rx);
说明	从存储器依次加载连续的 4 个字到寄存器堆 [R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 ldm r4-r7, (rx) 的伪指令。
影响标志位	无影响
限制	R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	

操作	从存储器加载连续的四个字到寄存器 R4—R7 中
	dst ← 4; addr ← RX;
	for (n = 0; n <= 3; n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}

语法	ldq32 r4-r7, (rx)
说明	从存储器依次加载连续的 4 个字到寄存器堆 [R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。 注意：该指令是 ldm32 r4-r7, (rx) 的伪指令。
影响标志位	无影响
限制	R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

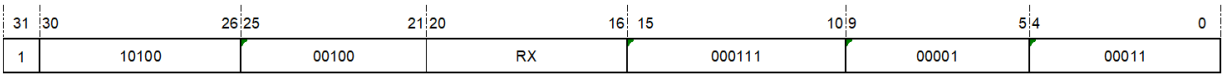


图 14.98: LDQ

14.70 LDR.B——寄存器移位寻址无符号扩展字节加载指令

统一化指令	
语法	ldr.b rz, (rx, ry << 0) ldr.b rz, (rx, ry << 1) ldr.b rz, (rx, ry << 2) ldr.b rz, (rx, ry << 3)
操作	从存储器加载字节到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + RY << IMM2])
编译结果	仅存在 32 位指令。 ldr32.b rz, (rx, ry << 0) ldr32.b rz, (rx, ry << 1) ldr32.b rz, (rx, ry << 2) ldr32.b rz, (rx, ry << 3)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + RY << IMM2])
语法	ldr32.b rz, (rx, ry << 0) ldr32.b rz, (rx, ry << 1) ldr32.b rz, (rx, ry << 2) ldr32.b rz, (rx, ry << 3)
说明	从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.b rz, (rx, ry << 0)

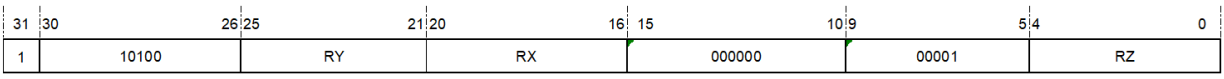


图 14.99: LDR.B-1

ldr32.b rz, (rx, ry << 1)

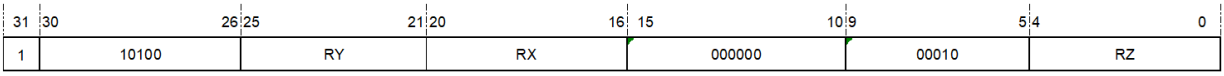


图 14.100: LDR.B-2

ldr32.b rz, (rx, ry << 2)

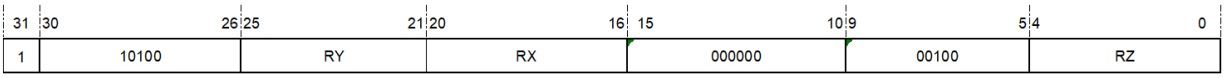


图 14.101: LDR.B-3

ldr32.b rz, (rx, ry << 3)

14.71 LDR.BS——寄存器移位寻址有符号扩展字节加载指令

统一化指令	
语法	ldr.bs rz, (rx, ry << 0) ldr.bs rz, (rx, ry << 1) ldr.bs rz, (rx, ry << 2) ldr.bs rz, (rx, ry << 3)
操作	从存储器加载字节到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + RY << IMM2])
编译结果	仅存在 32 位指令。 ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry << 2) ldr32.bs rz, (rx, ry << 3)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

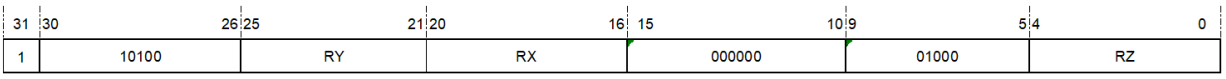


图 14.102: LDR.B-4

32 位指令	
操作	从存储器加载字节到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + RY << IMM2])
语法	ldr32.bs rz, (rx, ry << 0) ldr32.bs rz, (rx, ry << 1) ldr32.bs rz, (rx, ry << 2) ldr32.bs rz, (rx, ry << 3)
说明	从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.bs rz, (rx, ry<< 0)

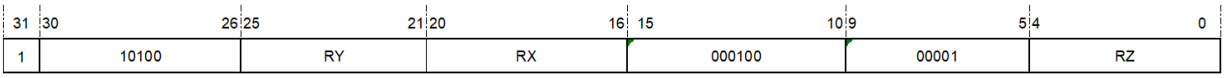


图 14.103: LDR.BS-1

ldr32.bs rz, (rx, ry<< 1)

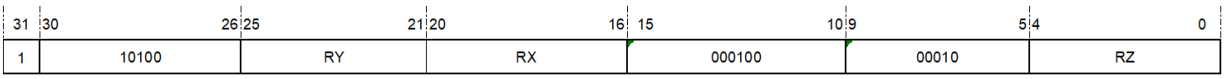


图 14.104: LDR.BS-2

ldr32.bs rz, (rx, ry<< 2)

ldr32.bs rz, (rx, ry<< 3)

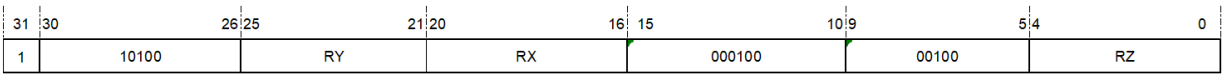


图 14.105: LDR.BS-3

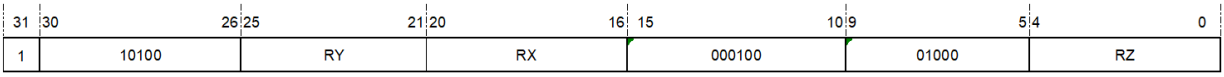


图 14.106: LDR.BS-4

14.72 LDR.H——寄存器移位寻址无符号扩展半字加载指令

统一化指令	
语法	ldr.h rz, (rx, ry << 0) ldr.h rz, (rx, ry << 1) ldr.h rz, (rx, ry << 2) ldr.h rz, (rx, ry << 3)
操作	从存储器加载半字到寄存器，无符号扩展。 RZ ← zero_extend(MEM[RX + RY << IMM2])
编译结果	仅存在 32 位指令。 ldr32.h rz, (rx, ry << 0) ldr32.h rz, (rx, ry << 1) ldr32.h rz, (rx, ry << 2) ldr32.h rz, (rx, ry << 3)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，无符号扩展 RZ ← zero_extend(MEM[RX + RY << IMM2])
语法	ldr32.h rz, (rx, ry << 0) ldr32.h rz, (rx, ry << 1) ldr32.h rz, (rx, ry << 2) ldr32.h rz, (rx, ry << 3)
说明	从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.h rz,(rx, ry << 0)

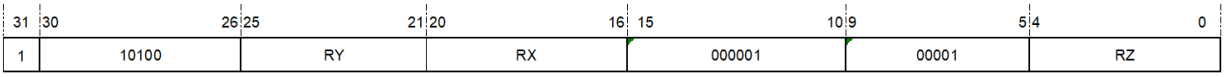


图 14.107: LDR.H-1

ldr32.h rz,(rx, ry << 1)

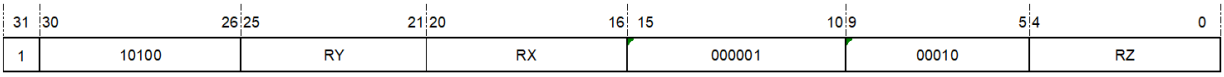


图 14.108: LDR.H-2

ldr32.h rz,(rx, ry << 2)

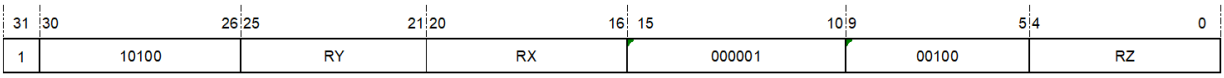


图 14.109: LDR.H-3

ldr32.h rz,(rx, ry << 3)

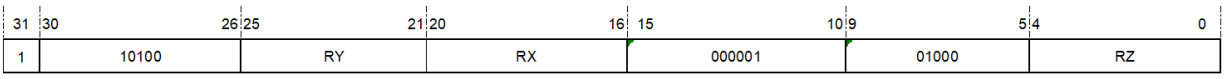


图 14.110: LDR.H-4

14.73 LDR.HS——寄存器移位寻址有符号扩展半字加载指令

统一化指令	
语法	ldr.hs rz, (rx, ry << 0) ldr.hs rz, (rx, ry << 1) ldr.hs rz, (rx, ry << 2) ldr.hs rz, (rx, ry << 3)
操作	从存储器加载半字到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + RY << IMM2])
编译结果	仅存在 32 位指令。 ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry << 2) ldr32.hs rz, (rx, ry << 3)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字到寄存器，有符号扩展 RZ ← sign_extend(MEM[RX + RY << IMM2])
语法	ldr32.hs rz, (rx, ry << 0) ldr32.hs rz, (rx, ry << 1) ldr32.hs rz, (rx, ry << 2) ldr32.hs rz, (rx, ry << 3)
说明	从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.hs rz, (rx, ry << 0)

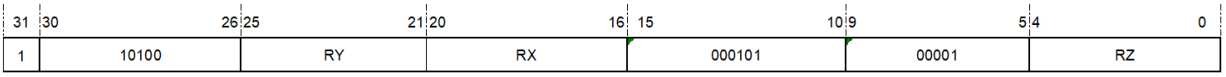


图 14.111: LDR.HS-1

ldr32.hs rz, (rx, ry << 1)

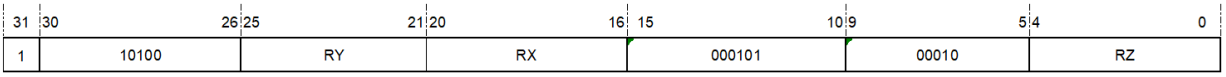


图 14.112: LDR.HS-2

ldr32.hs rz, (rx, ry << 2)

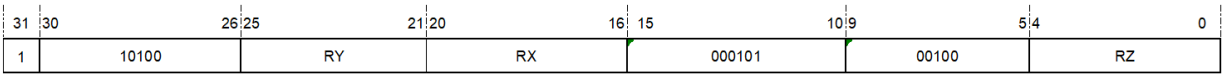


图 14.113: LDR.HS-3

ldr32.hs rz, (rx, ry << 3)

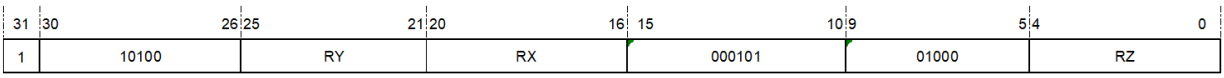


图 14.114: LDR.HS-4

14.74 LDR.W——寄存器移位寻址字加载指令

统一化指令	
语法	ldr.w rz, (rx, ry << 0) ldr.w rz, (rx, ry << 1) ldr.w rz, (rx, ry << 2) ldr.w rz, (rx, ry << 3)
操作	从存储器加载字到寄存器 RZ ← MEM[RX + RY << IMM2]
编译结果	仅存在 32 位指令。 ldr32.w rz, (rx, ry << 0) ldr32.w rz, (rx, ry << 1) ldr32.w rz, (rx, ry << 2) ldr32.w rz, (rx, ry << 3)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字到寄存器 RZ ← MEM[RX + RY << IMM2]
语法	ldr32.w rz, (rx, ry << 0) ldr32.w rz, (rx, ry << 1) ldr32.w rz, (rx, ry << 2) ldr32.w rz, (rx, ry << 3)
说明	从存储器加载字到寄存器 RZ 中。采用寄存器加寄存器移位的寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

ldr32.w rz, (rx, ry << 0)

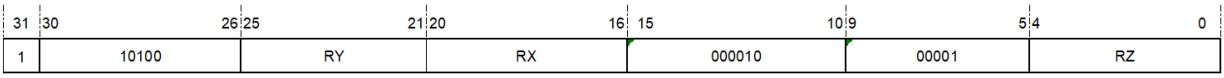


图 14.115: LDR.W-1

ldr32.w rz, (rx, ry << 1)

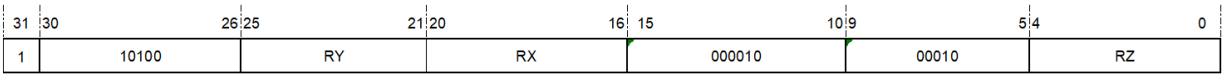


图 14.116: LDR.W-2

ldr32.w rz, (rx, ry << 2)

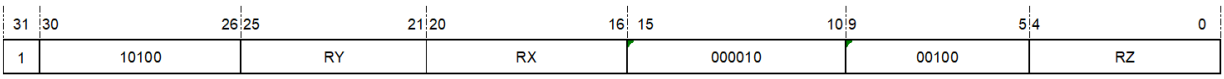


图 14.117: LDR.W-3

ldr32.w rz, (rx, ry << 3)

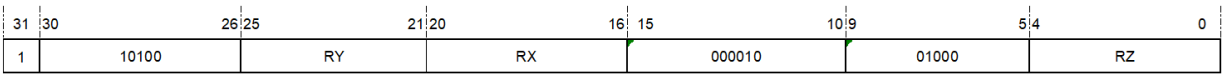


图 14.118: LDR.W-4

14.75 LRS.B——字节符号加载指令

统一化指令	
语法	lrs.b rz, [label]
操作	从存储器加载字节到寄存器 RZ ← zero_extend(MEM[R28 + zero_extend(offset)])
编译结果	仅存在 32 位指令。 lrs32.b rz, [label]
说明	加载 label 所在位置的字节符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R28 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.B 指令可以寻址 +256KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字节符号到寄存器，无符号扩展 RZ ← zero_extend(MEM[R28 + zero_extend(offset)])
语法	lrs32.b rz, [label]
说明	加载 label 所在位置的字节符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R28 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.B 指令可以寻址 +256KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

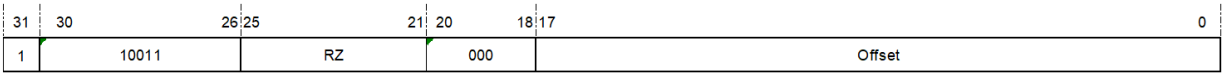


图 14.119: LRS.B

14.76 LRS.H——半字符号加载指令

统一化指令	
语法	lrs.h rz, [label]
操作	RZ ← zero_extend(MEM[R28 + zero_extend(offset << 1)])
编译结果	仅存在 32 位指令。 lrs32.h rz, [label]
说明	加载 label 所在位置的半字符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载半字符号到寄存器，无符号扩展 RZ ← zero_extend(MEM[R28 + zero_extend(offset << 1)])
语法	lrs32.h rz, [label]
说明	加载 label 所在位置的半字符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

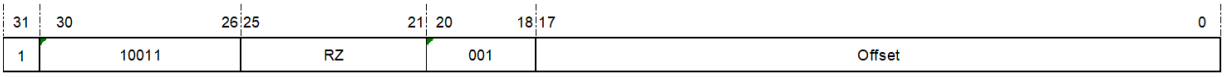


图 14.120: LRS.H

14.77 LRS.W——字符号加载指令

统一化指令	
语法	lrs.w rz, [label]
操作	RZ ← zero_extend(MEM[R28 + zero_extend(offset << 2)])
编译结果	仅存在 32 位指令。 lrs32.w rz, [label]
说明	加载 label 所在位置的字符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	从存储器加载字符号到寄存器，无符号扩展 RZ ← zero_extend(MEM[R28 + zero_extend(offset << 2)])
语法	lrs32.w rz, [label]
说明	加载 label 所在位置的字符号，经零扩展到 32 位后存放于目的寄存器 RZ。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R2 8 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。LRS.W 指令可以寻址 +1024KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

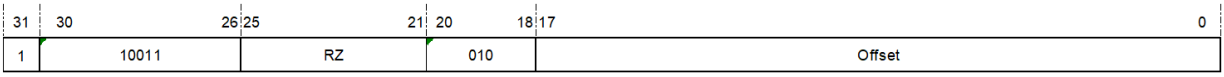


图 14.121: LRS.W

14.78 LRW——存储器读入指令

统一化指令	
语法	lrw rz, label lrw rz, imm32
操作	从存储器加载字到寄存器 RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

编译结果	根据加载的范围编译为对应的 16 位或 32 位指令
	if(offset<512B), then
	lrw16 label;
	lrw16 imm32;
	else
	lrw32 label;
	lrw32 imm32;

说明	加载 label 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ。存储器地址根据 PC 加左移两位的相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令	
操作	从存储器加载字到寄存器 RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])
语法	lrw16 rz, label lrw16 rz, imm32
说明	加载 labe l 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ 。存储器地址根据 PC 加左移两位的 10 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。 注意，相对偏移量 Offset 等于二进制编码 {IMM2, IMM5}。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

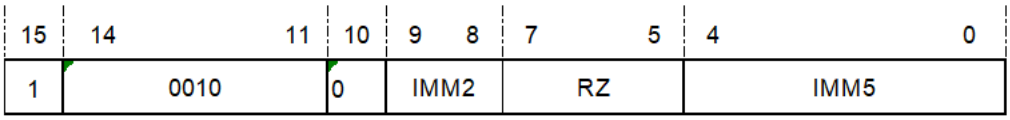


图 14.122: LRW-1

32 位指令	
操作	从存储器加载字到寄存器 RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])
语法	lrw32 rz, label lrw32 rz, imm32
说明	加载 labe l 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ 。存储器地址根据 PC 加左移两位的 16 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

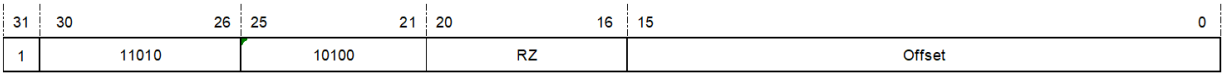


图 14.123: LRW-2

14.79 LSL——逻辑左移指令

统一化指令		
语法	lsl rz, rx	lsl rz, rx, ry
操作	RZ ← RZ << RX[5:0]	RZ ← RX << RY[5:0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	lsl16 rz, rx;
	else
	lsl32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		lsl16 rz, ry
		else
		lsl32 rz, rx, ry

说明	对于 lsl rz, rx，将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零； 对于 lsl rz, rx, ry，将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ << RX[5:0]
语法	lsl16 rz, rx
说明	将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

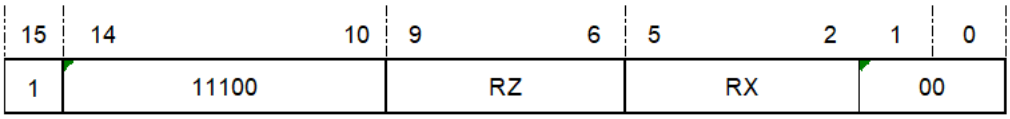


图 14.124: LSL-1

32 位指令	
操作	RZ ← RX << RY[5:0]
语法	lsl32 rz, rx, ry
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
异常	无

32 位指令格式：

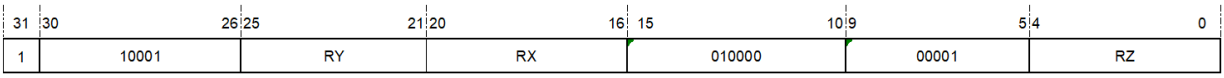


图 14.125: LSL-2

14.80 LSLC——立即数逻辑左移至 C 位指令

统一化指令	
语法	lslc rz, rx, oimm5
操作	RZ ← RX << OIMM5, C ← RX[32 – OIMM5]
编译结果	仅存在 32 位指令。 lslc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。
影响标志位	C ← RX[32 – OIMM5]
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	RZ ← RX << OIMM5, C ← RX[32 – OIMM5]
语法	lslc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	C ← RX[32 – OIMM5]
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

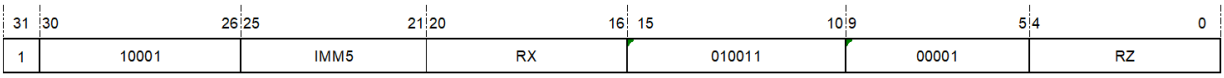


图 14.126: LSLC

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

……

11111：

移 32 位

14.81 LSLI——立即数逻辑左移指令

统一化指令	
语法	lsli rz, rx, imm5
操作	RZ ← RX << IMM5

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<8) and (z<8), then
	lsli16 rz, rx, imm5
	else
	lsli32 rz, rx, imm5

说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

16 位指令	
操作	RZ ← RX << IMM5
语法	lsli16 rz, rx, imm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	寄存器的范围为 r0-r7； 立即数的范围为 0-31。
异常	无

16 位指令格式：

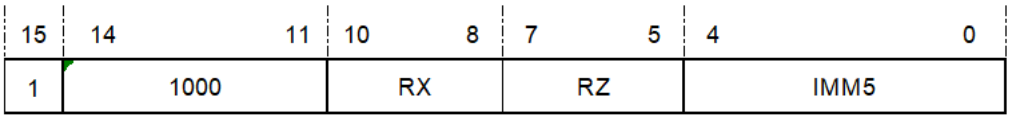


图 14.127: LSLI-1

32 位指令	
操作	RZ ← RX << IMM5
语法	lsli32 rz, rx, imm5
说明	将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

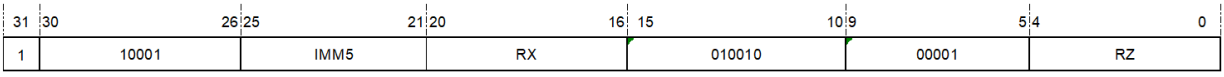


图 14.128: LSLI-2

14.82 LSR——逻辑右移指令

统一化指令		
语法	lsr rz, rx	lsr rz, rx, ry
操作	RZ ← RZ >> RX[5:0]	RZ ← RX >> RY[5:0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<16) and (x<16), then
	lsr16 rz, rx ;
	else
	lsr32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (z<16) and (y<16), then
		lsr16 rz, ry ;
		else
		lsr32 rz, rx, ry;

说明	对于 lsr rz, rx，将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。 对于 lsr rz, rx, ry，将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ >> RX[5:0]
语法	lsr16 rz, rx
说明	将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

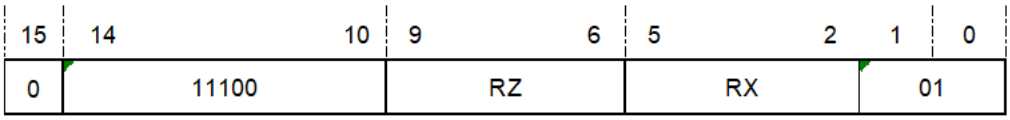


图 14.129: LSR-1

32 位指令	
操作	RZ ← RX >> RY[5:0]
语法	lsr32 rz, rx, ry
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
异常	无

32 位指令格式：

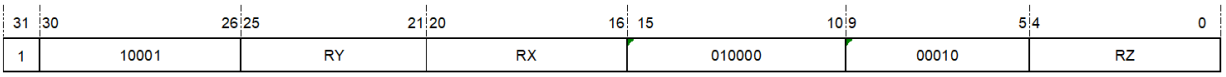


图 14.130: LSR-2

14.83 LSRC——立即数逻辑右移至 C 位指令

统一化指令	
语法	lsrc rz, rx, oimm5
操作	RZ ← RX >> OIMM5, C ← RX[OIMM5 - 1]
编译结果	仅存在 32 位指令。 lsrc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	RZ ← RX >> OIMM5, C ← RX[OIMM5 - 1]
语法	lsrc32 rz, rx, oimm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

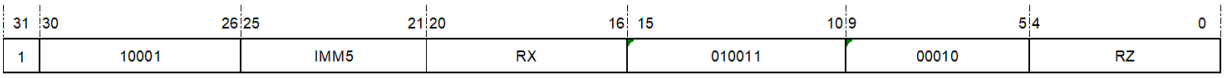


图 14.131: LSRC

IMM5 域：

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000：

移 1 位

00001：

移 2 位

……

11111：

移 32 位

14.84 LSRI——立即数逻辑右移指令

统一化指令	
语法	lsri rz, rx, imm5
操作	RZ ← RX >> IMM5

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<8) and (z<8), then
	lsri16 rz, rx, imm5
	else
	lsri32 rz, rx, imm5

说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值不变或者将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

16 位指令	
操作	RZ ← RX >> IMM5
语法	lsri16 rz, rx, imm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-31。
异常	无

16 位指令格式：

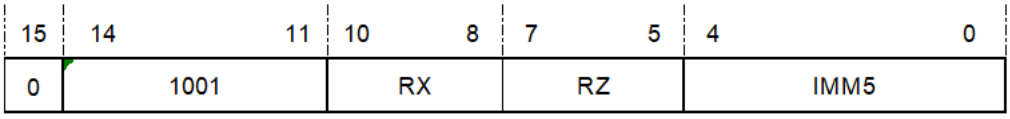


图 14.132: LSRI-1

32 位指令	
操作	RZ ← RX >> IMM5
语法	lsri32 rz, rx, imm5
说明	将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

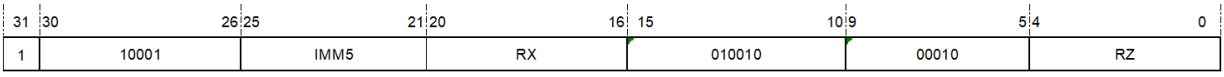


图 14.133: LSRI-2

14.85 MFCR——控制寄存器读传送指令

统一化指令	
语法	mfcr rz, cr<x, sel>
操作	将控制寄存器的内容传送到通用寄存器中 RZ ← CR<X, sel>
编译结果	仅存在 32 位指令。 mfcr32 rz, cr<x, sel>
属性：	特权指令
说明	将控制寄存器 CR<x, sel> 的内容传送到通用寄存器 RZ 中。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	将控制寄存器的内容传送到通用寄存器中 RZ ← CR<X, sel>
语法	mfcr32 rz, cr<x, sel>
属性：	特权指令
说明	将控制寄存器 CR<x, sel> 的内容传送到通用寄存器 RZ 中。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

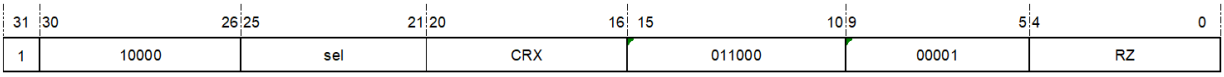


图 14.134: MFCR

14.86 MFHI——累加器高位读传送指令

统一化指令	
语法	mfhi rz
操作	将累加器高位寄存器的内容传送到通用寄存器中 RZ ← HI
编译结果	仅存在 32 位指令。 mfhi32 rz

说明:	**将 64 位累加器的高 32 位寄存器 HI 的内容传送到通用寄存器 RZ 中。**
影响标志位:	无影响
异常：	无

32 位指令	
操作:	将累加器高位寄存器的内容传送到通用寄存器中 RZ ← HI
语法:	mfhi32 rz
说明:	将 64 位累加器的高 32 位寄存器 HI 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常：	无

指令格式:

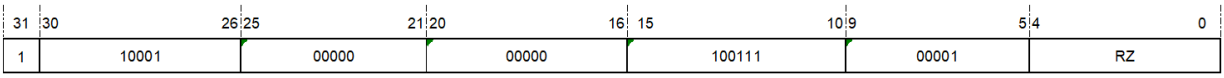


图 14.135: MFHI

14.87 MFHIS——累加器高位饱和读传送指令

统一化指令	
语法	mfhis rz
操作	将累加器高位寄存器的内容取饱和值后传送到通用寄存器中 RZ ← saturate(HI)
编译结果	仅存在 32 位指令。 mfhis32 rz

说明:	将 64 位累加器的高 32 位寄存器 HI 的内容取饱和值后传送到通用寄存器 RZ 中。饱和操作具体请参考处理器手册有关保护位的描述。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将累加器高位寄存器的内容取饱和值后传送到通用寄存器中 RZ ← saturate(HI)
语法:	mfhis32 rz
说明:	将 64 位累加器的高 32 位寄存器 HI 的内容取饱和值后传送到通用寄存器 RZ 中。饱和操作具体请参考处理器手册有关保护位的描述。
影响标志位:	无影响
异常:	无

指令格式:

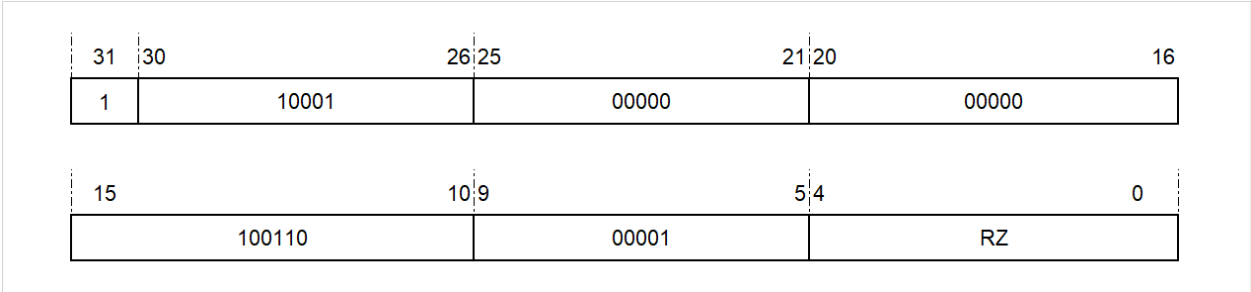


图 14.136: MFHIS

14.88 MFLO——累加器低位读传送指令

统一化指令	
语法	mflo rz
操作	将累加器低位寄存器的内容传送到通用寄存器中。 RZ ← LO
编译结果	仅存在 32 位指令。 mflo32 rz

说明:	将 64 位累加器的低 32 位寄存器 LO 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将累加器低位寄存器的内容传送到通用寄存器中 RZ ← LO
语法:	mflo32 rz
说明:	将 64 位累加器的低 32 位寄存器 LO 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常:	无

指令格式:

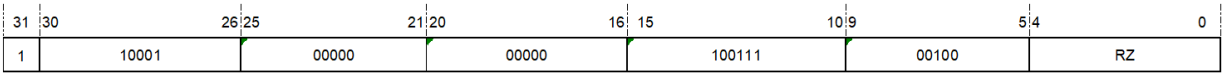


图 14.137: MFLO

14.89 MFLOS——累加器低位饱和读传送指令

统一化指令	
语法	mflos rz
操作	将累加器低位寄存器的内容取饱和后传送到通用寄存器中。 RZ ← saturate(LO)
编译结果	仅存在 32 位指令。 mflos32 rz

说明:	**将 64 位累加器的低 32 位寄存器 LO 的内容取饱和后传送到通用寄存器 RZ 中。饱和操作具体请参考处理器手册有关保护位的描述。**
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将累加器低位寄存器的内容取饱和后传送到通用寄存器中 RZ ← saturate(LO)
语法:	mflos32 rz
说明:	将 64 位累加器的低 32 位寄存器 LO 的内容取饱和后传送到通用寄存器 RZ 中。饱和操作具体请参考处理器手册有关保护位的描述。
影响标志位:	无影响
异常:	无

指令格式:

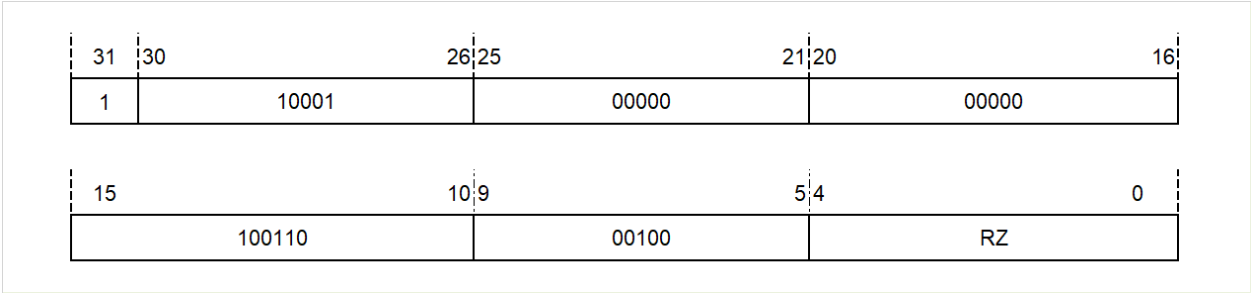


图 14.138: MFLOS

14.90 MOV——数据传送指令

统一化指令	
语法	mov rz, rx
操作	RZ ← RX
编译结果	总是编译为 16 位指令。 mov16 rz, rx
说明	把 RX 中的值复制到目的寄存器 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← RX
语法	mov16 rz, rx
说明	把 RX 中的值复制到目的寄存器 RZ 中。 注意，该指令寄存器索引范围为 r0-r31。
影响标志位	无影响
异常	无

16 蚊指令格式：

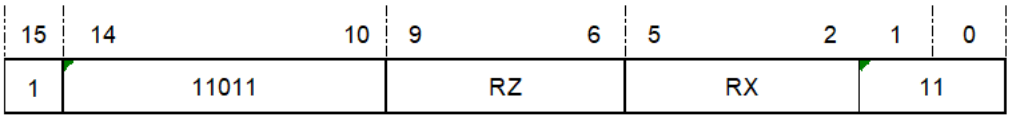


图 14.139: MOV-1

32 位指令	
操作	RZ ← RX
语法	mov32 rz, rx
说明	把 RX 中的值复制到目的寄存器 RZ 中。 注意，该指令是 lsli32 rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

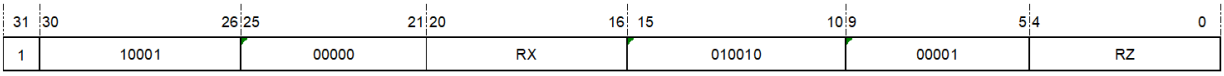


图 14.140: MOV-2

14.91 MOVF——C 为 0 数据传送指令

统一化指令	
语法	movf rz, rx

操作	if C==0, then
	RZ ← RX;
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令。 movf32 rz, rx
说明	如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 incf rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令	

操作	if C==0, then
	RZ ← RX;
	else
	RZ ← RZ;

语法	movf32 rz, rx
说明	如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 incf32 rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

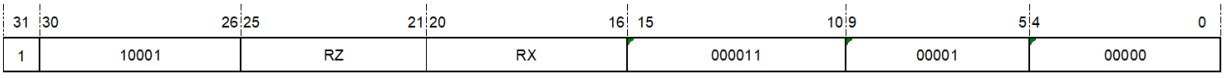


图 14.141: MOVF

14.92 MOVI——立即数数据传送指令

统一化指令	
语法	movi rz, imm
操作	RZ ← zero_extend(IMM);

编译结果	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (imm16<256) and (z<7), then
	movi16 rz, imm8;
	else
	movi32 rz, imm16;

说明	将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

16 位指令	
操作	RZ ← zero_extend(IMM8);
语法	movi16 rz, imm8
说明	将 8 位立即数零扩展至 32 位，然后传送至目的 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 0-255。
异常	无

16 位指令格式：

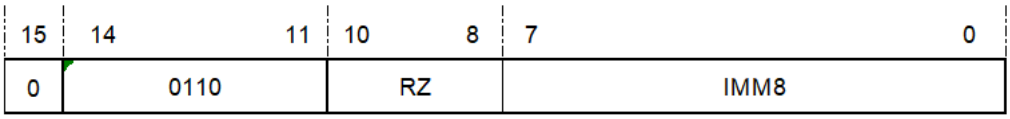


图 14.142: MOVI-1

32 位指令	
操作	RZ ← zero_extend(IMM16);
语法	movi32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

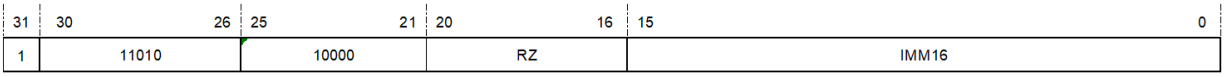


图 14.143: MOVI-2

14.93 MOVIH——立即数高位数据传送指令

统一化指令	
语法	movih rz, imm16
操作	RZ ← zero_extend(IMM16) << 16
编译结果	仅存在 32 位指令。 movih32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。 该指令可配合 ori rz, rz, imm16 指令产生任意 32 位立即数。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令	
操作	RZ ← zero_extend(IMM16) << 16
语法	movih32 rz, imm16
说明	将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。 该指令可配合 ori32 rz, rz, imm16 指令产生任意 32 位立即数。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令格式：

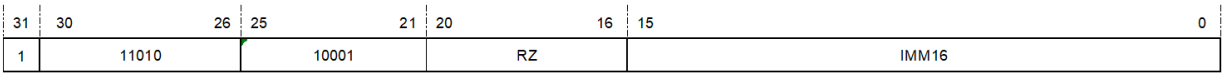


图 14.144: MOVIH

14.94 MOVT——C 为 1 数据传送指令

统一化指令	
语法	movt rz, rx

操作	if C==1, then
	RZ ← RX;
	else
	RZ ← RZ;

编译结果	仅存在 32 位指令。 movt32 rz, rx
影响标志位	无影响
异常	无

32 位指令	

操作	if C==1, then
	RZ ← RX;
	else
	RZ ← RZ;

语法	movt32 rz, rx
说明	如果 C 为 1，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。 注意，该指令是 inct32 rz, rx, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

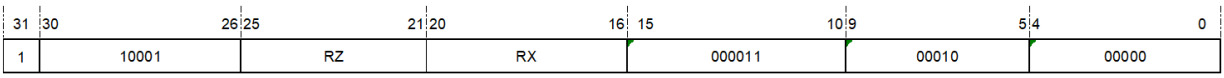


图 14.145: MOVT

14.95 MTCR——控制寄存器写传送指令

统一化指令	
语法	mtcr rx, cr<z, sel>
操作	将通用寄存器的内容传送到控制寄存器中 CR<Z, sel> ← RX
编译结果	仅存在 32 位指令。 mtcr32 rx, cr<z, sel>
属性：	特权指令
说明	将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel> 中。
影响标志位	如果目标控制寄存器不是 PSR，则该指令不会影响标志位。
异常	特权违反异常

32 位指令	
操作	将通用寄存器的内容传送到控制寄存器中 CR<Z, sel> ← RX
语法	mtcr32 rx, cr<z, sel>
属性：	特权指令
说明	将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel> 中。
影响标志位	如果目标控制寄存器不是 PSR，则该指令不会影响标志位。
异常	特权违反异常

32 位指令格式：

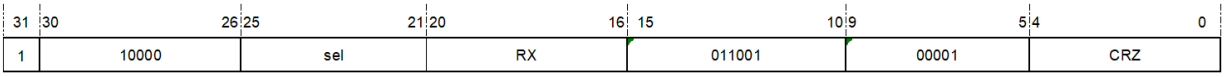


图 14.146: MTCR

14.96 MTHI——累加器高位写传送指令

统一化指令	
语法	mthi rx
操作	将通用寄存器的内容传送到累加器高位寄存器中. HI ← RX
编译结果	仅存在 32 位指令。 mthi32 rx

说明:	将通用寄存器 RX 的内容传送到 64 位累加器的高 32 位寄存器 HI 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将通用寄存器的内容传送到累加器高位寄存器中 HI ← RX
语法:	mthi32 rx
说明:	将通用寄存器 RX 的内容传送到 64 位累加器的高 32 位寄存器 HI 中。
影响标志位:	无影响
异常:	无

指令格式:

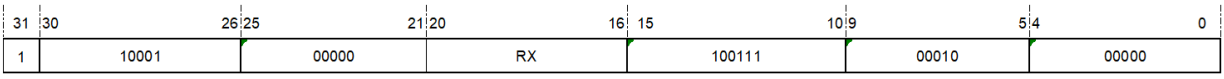


图 14.147: MTHI

14.97 MTLO——累加器低位写传送指令

统一化指令	
语法	mtlo rx
操作	将通用寄存器的内容传送到累加器低位寄存器中 LO ← RX
编译结果	仅存在 32 位指令。 mtlo32 rx

说明:	将通用寄存器 RX 的内容传送到 64 位累加器的低 32 位寄存器 LO 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将通用寄存器的内容传送到累加器低位寄存器中 LO ← RX
语法:	mtlo32 rx
说明:	将通用寄存器 RX 的内容传送到 64 位累加器的低 32 位寄存器 LO 中。
影响标志位:	无影响
异常:	无

指令格式:

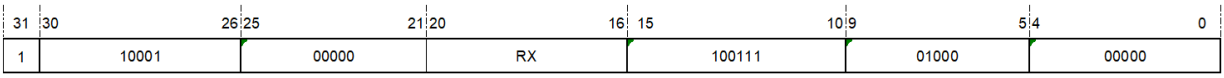


图 14.148: MTLO

14.98 MULS——有符号数乘法指令

统一化指令	
语法	muls rx, ry
操作	两个有符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
编译结果	仅存在 32 位指令。 muls32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个有符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
语法:	muls32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

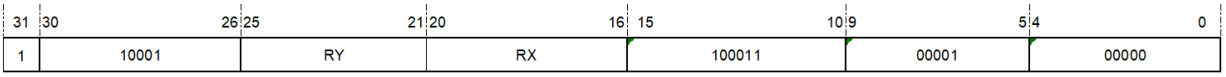


图 14.149: MULS

14.99 MULSA——有符号数乘累加指令

统一化指令	
语法	mulsa rx, ry
操作	两个有符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
编译结果	仅存在 32 位指令。 mulsa32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

32位指令	
操作:	两个有符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
语法:	mulsa32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

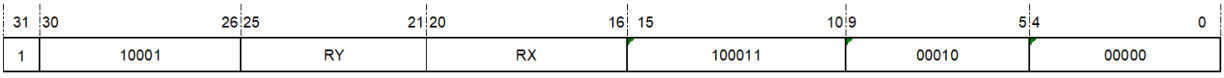


图 14.150: MULSA

14.100 MULSHA——16 位有符号数乘累加指令

统一化指令	
语法	mulsha rx, ry
操作	两个 16 位有符号数相乘再加上累加器低位的值，结果放回累加器低位中 LO ← LO + RX[15:0] × RY[15:0];
编译结果	仅存在 32 位指令。 mulsha32 rx, ry

说明:	将通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后得到的 32 位结果，再加上 64 位累加器的低 32 位寄存器 LO，结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 和 RY 的源操作数的符号位是寄存器的第 15 位，累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无
32 位指令	
操作:	两个 16 位有符号数相乘再加上累加器低位的值，结果放回累加器低位中 LO ← LO + RX[15:0] × RY[15:0];
语法:	mulsha32 rx, ry
说明:	将通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后得到的 32 位结果，再加上 64 位累加器的低 32 位寄存器 LO，结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 和 RY 的源操作数的符号位是寄存器的第 15 位，累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无

指令格式:

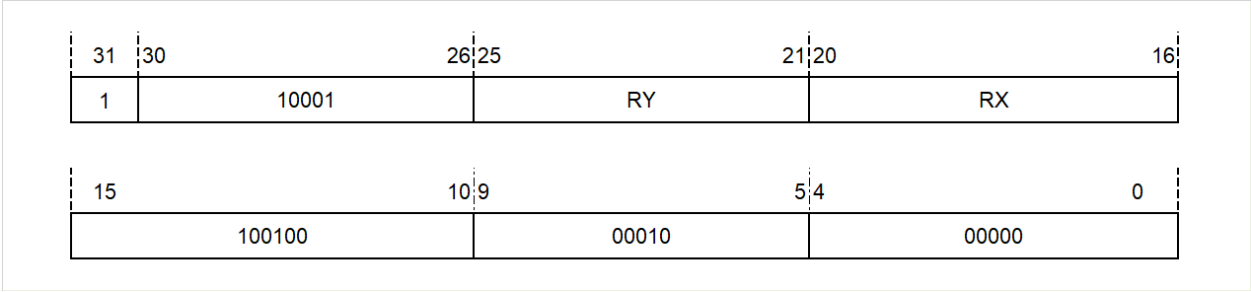


图 14.151: MULSHA

14.101 MULSHS——16 位有符号数乘累减指令

统一化指令	
语法	mulshs rx, ry
操作	累加器低位的值减去两个 16 位有符号数相乘后的值，结果放回累加器低位中 LO ← LO - RX[15:0] × RY[15:0];
编译结果	仅存在 32 位指令。 mulshs32 rx, ry

说明:	将 64 位累加器的低 32 位寄存器 LO 中的值减去通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后的值，结果放回到累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 和 RY 的源操作数的符号位是寄存器的第 15 位，累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无

32位指令	
操作:	累加器低位的值减去两个 16 位有符号数相乘后的值，结果放回累加器低位中 LO ← LO - RX[15:0] × RY[15:0];
语法:	mulshs32 rx, ry
说明:	将 64 位累加器的低 32 位寄存器 LO 中的值减去通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后的值，结果放回到累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 和 RY 的源操作数的符号位是寄存器的第 15 位，累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无

指令格式:

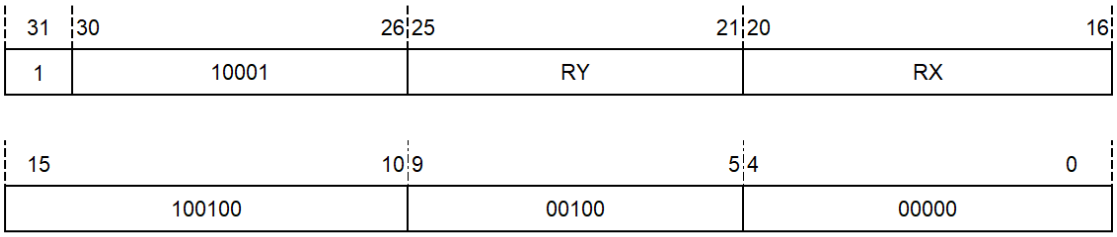


图 14.152: MULSHS

14.102 MULSS——有符号数乘累减指令

统一化指令	
语法	mulss rx, ry
操作	累加器中的值减去两个有符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
编译结果	仅存在 32 位指令。 mulss32 rx, ry

说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	累加器中的值减去两个有符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
语法:	mulss32 rx, ry
说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

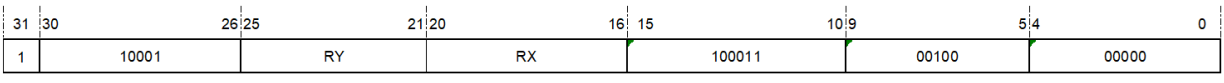


图 14.153: MULSS

14.103 MULSW——有符号数 16x32 乘法指令

统一化指令	
语法	mulsw rx, ry
操作	16 位有符号数和 32 位有符号数相乘，结果的高 32 位放回累加器低位中 LO ← (RX[15:0] × RY[31:0])[47:16];
编译结果	仅存在 32 位指令。 mulsw32 rx, ry

说明:	将通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 的源操作数的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	无影响
异常:	无

32位指令	
操作:	16 位有符号数和 32 位有符号数相乘结果的高 32 位放回累加器低位中 LO ← (RX[15:0] × RY[31:0])[47:16];
语法:	mulsw32 rx, ry
说明:	将通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 的源操作数的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	无影响
异常:	无

指令格式:

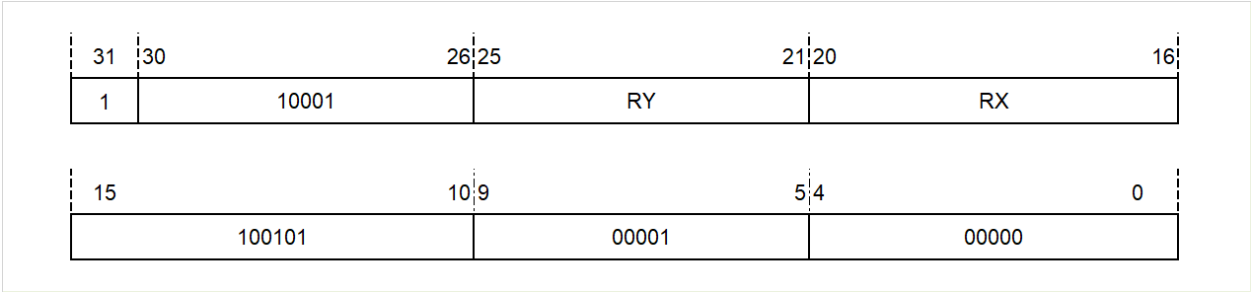


图 14.154: MULSW

14.104 MULSWA——有符号数 16x32 乘累加指令

统一化指令	
语法	mulswa rx, ry
操作	16 位有符号数和 32 位有符号数相乘，结果的高 32 位再加上累加器低位的值，放回累加器低位中 LO ← LO + (RX[15:0] × RY[31:0])[47:16];
编译结果	仅存在 32 位指令。 mulswa32 rx, ry

说明:	将通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位，再加上 64 位累加器的低 32 位寄存器 LO，结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 和累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无

32位指令	
操作:	16 位有符号数和 32 位有符号数相乘，结果的高 32 位再加上累加器低位的值，放回累加器低位中 LO ← LO + (RX[15:0] × RY[31:0])[47:16];
语法:	mulswa32 rx, ry
说明:	将通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位，再加上 64 位累加器的低 32 位寄存器 LO，结果放回累加器低位寄存器 LO 中。寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 和累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。 该指令支持 8 位保护位，具体请参考处理器手册有关保护位的描述。
影响标志位:	溢出会使溢出位置 1
异常:	无

指令格式:

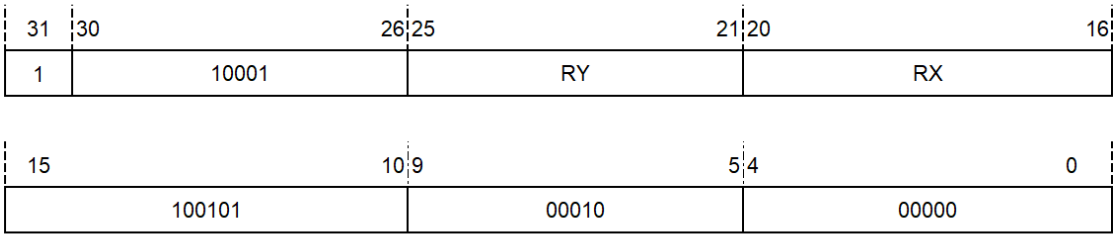


图 14.155: MULSWA

14.105 MULSWS——有符号数 16x32 乘累减指令

统一化指令	
语法	mulsws rx, ry
操作	累加器低位的值减去 16 位有符号数和 32 位有符号数相乘结果的高 32 位，放回累加器低位中 LO ← LO - (RX[15:0] × RY[31:0])[47:16];
编译结果	仅存在 32 位指令。 mulsws32 rx, ry

说明:	将 64 位累加器的低 32 位寄存器 LO 中的值减去通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位，放回到累加器低位寄存器 LO 中。 寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 和累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。
影响标志位:	溢出会使溢出位置 1
异常:	无

32位指令	
操作:	累加器低位的值减去 16 位有符号数和 32 位有符号数相乘结果的高 32 位，放回累加器低位中 LO ← LO - (RX[15:0] × RY[31:0])[47:16];
语法:	mulsws32 rx, ry
说明:	将 64 位累加器的低 32 位寄存器 LO 中的值减去通用寄存器 RX 的低 16 位和 RY 相乘后结果的高 32 位，放回到累加器低位寄存器 LO 中。 寄存器 RX 和 RY 以及的累加器低位寄存器 LO 中的内容均被认为是有符号数，其中寄存器 RX 的源操作数的符号位是寄存器的第 15 位，寄存器 RY 和累加器低位寄存器 LO 中源操作数和结果的符号位是寄存器的第 31 位。
影响标志位:	溢出会使溢出位置 1
异常:	无

指令格式:

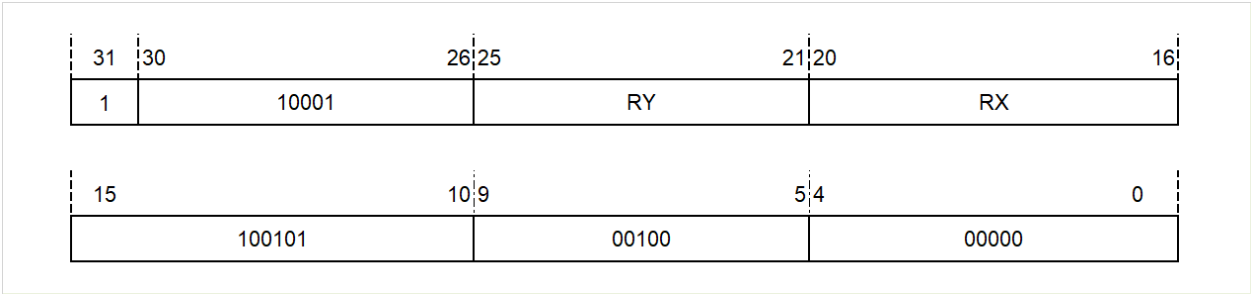


图 14.156: MULSWS

14.106 MULU——无符号数乘法指令

统一化指令	
语法	mulu rx, ry
操作	两个无符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
编译结果	仅存在 32 位指令。 mulu32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个无符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
语法:	mulu32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

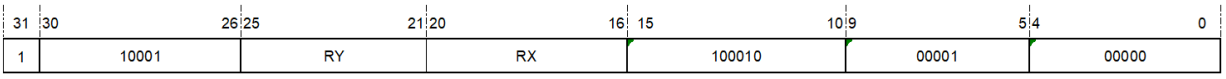


图 14.157: MULU

14.107 MULUA——无符号数乘累加指令

统一化指令	
语法	mulua rx, ry
操作	两个无符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
编译结果	仅存在 32 位指令。 mulua32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个无符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
语法:	mulua32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

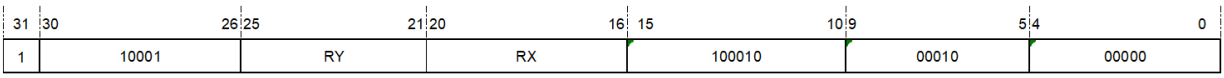


图 14.158: MULUA

14.108 MULUS——无符号数乘累减指令

统一化指令	
语法	mulus rx, ry
操作	累加器中的值减去两个无符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
编译结果	仅存在 32 位指令。 mulus32 rx, ry

说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	累加器中的值减去两个无符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
语法:	mulus32 rx, ry
说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

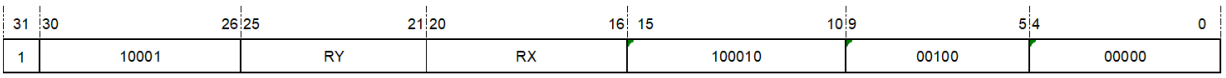


图 14.159: MULUS

14.109 MULSH——16 位有符号乘法指令

统一化指令	
语法	mulsh rz, rx
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RZ[15..0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	mulsh16 rz, rx;
	else
	mulsh32 rz, rz, rx;

说明	将通用寄存器 RX 的低 16 位和 RZ/ RY 的低 16 位相乘后的结果存放到通用寄存器 RZ 中。寄存器的内容均被认为是有符号数，其中源寄存器的符号位是第 15 位，目标寄存器的符号位是第 31 位。
影响标志位	无影响
异常	无

16 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RZ[15..0]
语法	mulsh16 rz, rx
说明	将通用寄存器 RX 的低 16 位和 RZ 的低 16 位相乘后得到的 32 位结果存放到通用寄存器 RZ 中。寄存器 RX 和 RZ 的内容均被认为是有符号数，其中源操作数的符号位是寄存器的第 15 位，结果的符号位是寄存器的第 31 位。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

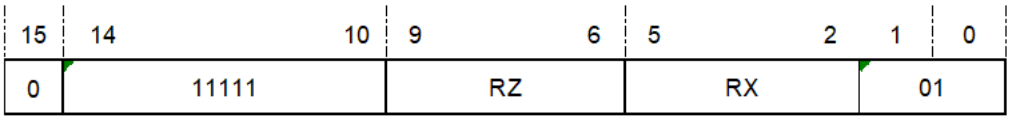


图 14.160: MULSH-1

32 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RY[15..0]
语法	mulsh32 rz, rx, ry
说明	将通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后的结果存放到通用寄存器 RZ 中。寄存器 RX、RY 和 RZ 的内容均被认为是有符号数，其中源寄存器 RX、RY 的符号位是第 15 位，目标寄存器 RZ 的符号位是第 31 位。
影响标志位	无影响
异常	无

32 位指令格式：

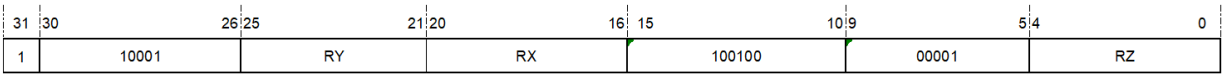


图 14.161: MULSH-2

14.110 MULT——乘法指令

统一化指令		
语法	mult rz, rx	mult rz, rx, ry
操作	两个数相乘，结果的低 3 2 位放入通用寄存器中 RZ ← RX × RZ	两个数相乘，结果的低 3 2 位放入通用寄存器中 RZ ← RX × RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	mult16 rz, rx;
	else
	mult32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16)and (z<16), then
		mult16 rz, rx;
		else
		mult32 rz, rx, ry;

说明	将两个源寄存器的内容相乘后结果的低 32 位存放到目的寄存器中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	两个数相乘，结果的低 32 位放入通用寄存器中 RZ ← RX × RZ
语法	mult16 rz, rx
说明	将通用寄存器 RX 和 RZ 的内容相乘后得到结果的低 32 位存放到通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

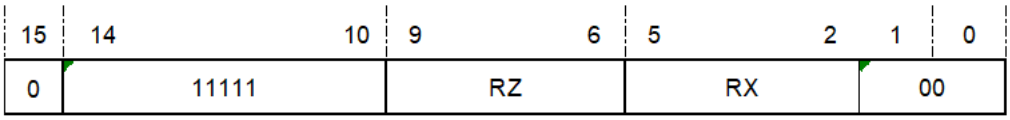


图 14.162: MULT-1

32 位指令	
操作	两个数相乘，结果的低 32 位放入通用寄存器中 RZ ← RX × RY
语法	mult32 rz, rx, ry
说明	将通用寄存器 RX 和 RY 的内容相乘后结果的低 32 位存放到通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。
影响标志位	无影响
异常	无

32 位指令格式：

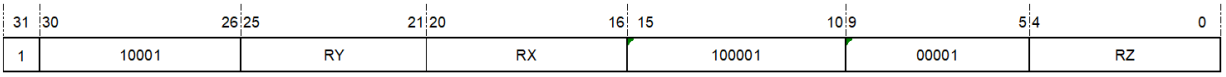


图 14.163: MULT-2

14.111 MVC——C 位传送指令

统一化指令	
语法	mvc rz
操作	RZ ← C
编译结果	仅存在 32 位指令。 mvc32 rz;
说明	把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← C
语法	mvc32 rz
说明	把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令格式：

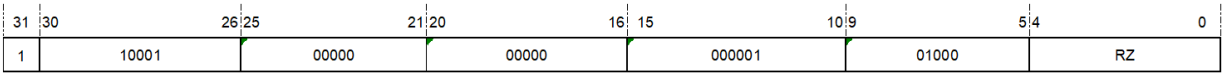


图 14.164: MVC

14.112 MVCV——C 位取反传送

统一化指令	
语法	mvcv rz
操作	RZ ← (!C)

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<16), then
	mvcv16 rz;
	else
	mvcv32 rz;

说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← (!C)
语法	mvcv16 rz
说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

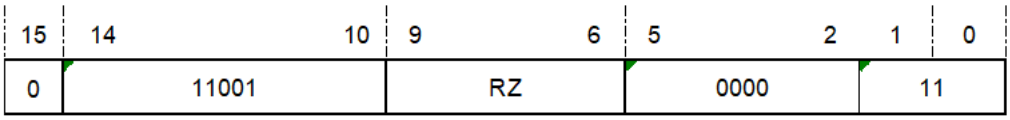


图 14.165: MVCV-1

32 位指令	
操作	RZ ← (!C)
语法	mvcv32 rz
说明	把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。
影响标志位	无影响
异常	无

32 位指令格式：

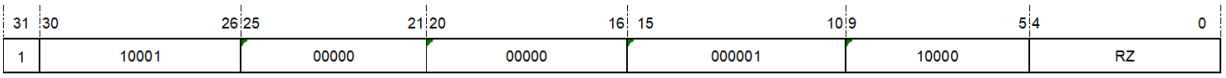


图 14.166: MVCV-2

14.113 MVTC——溢出位复制到 C 位指令

统一化指令	
语法	mvtc
操作	C ← V
编译结果	仅存在 32 位指令。 mvtc32;

说明:	将 DCSR(CR <14,0>) 的溢出位复制到 C 位。
影响标志位:	根据溢出位设置 C 位
异常:	无

32 位指令	
操作:	C ← V
语法:	mvtc32
说明:	将 DCSR(CR <14,0>) 的溢出位复制到 C 位。
影响标志位:	根据溢出位设置 C 位
异常:	无

指令格式：

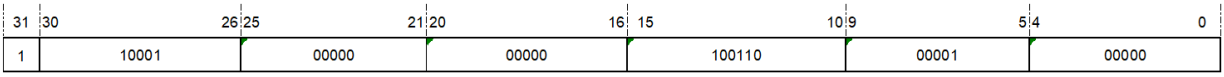


图 14.167: MVTC

14.114 NOR——按位或非指令

统一化指令		
语法	nor rz, rx	or rz, rx, ry
操作	RZ ← !(RZ | RX)	RZ ← !(RX | RY)

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	nor16 rz, rx;
	else
	nor32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		nor16 rz, rx
		else
		nor32 rz, rx, ry

说明	将 RX 与 RY/RZ 的值按位或，然后按位取非，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← !(RZ | RX)
语法	nor16 rz, rx
说明	将 RZ 与 RX 的值按位或，然后按位取非，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

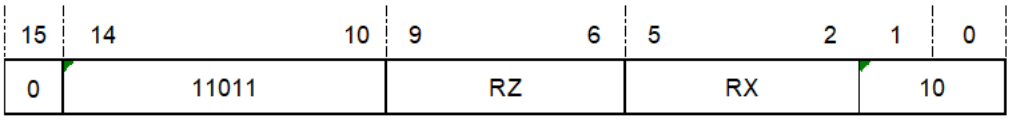


图 14.168: NOR-1

32 位指令	
操作	RZ ← !(RX | RY)
语法	nor32 rz, rx, ry
说明	将 RX 与 RY 的值按位或，然后按位取非，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

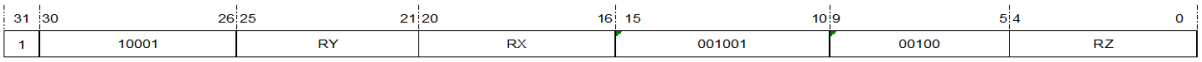


图 14.169: NOR-2

14.115 NOT——按位非指令

统一化指令		
语法	not rz	not rz, rx
操作	RZ ← !(RZ)	RZ ← !(RX)

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<16), then
	not16 rz;
	else
	not32 rz, rz;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (z<16), then
		not16 rz;
		else
		not32 rz, rx;

说明	将 RZ/RX 的值按位取反，把结果存在 RZ。 注意，该指令是 nor rz, rz 和 nor rz, rx, rx 的伪指令。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← !(RZ)
语法	not16 rz
说明	将 RZ 的值按位取反，把结果存在 RZ。 注意，该指令是 nor16 rz, rz 的伪指令。
影响标志位	无影响
异常	无

16 位指令格式：

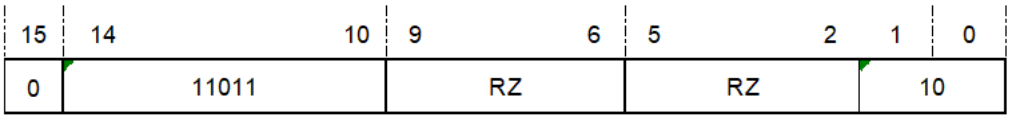


图 14.170: NOT-1

32 位指令	
操作	RZ ← !(RX)
语法	not32 rz, rx
说明	将 RX 的值按位取反，把结果存在 RZ。 注意，该指令是 nor32 rz, rx, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

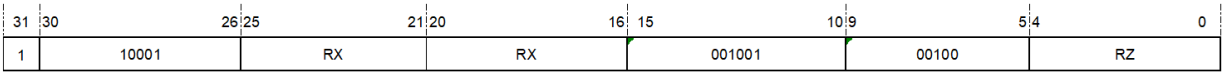


图 14.171: NOT-2

14.116 OR——按位或指令

统一化指令		
语法	or rz, rx	or rz, rx, ry
操作	RZ ← RZ | RX	RZ ← RX | RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	or16 rz, rx ;
	else
	or32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		or16 rz, rx
		else
		or32 rz, rx, ry

说明	将 RX 与 RY/RZ 的值按位或，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ | RX
语法	or16 rz, rx
说明	将 RZ 与 RX 的值按位或，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

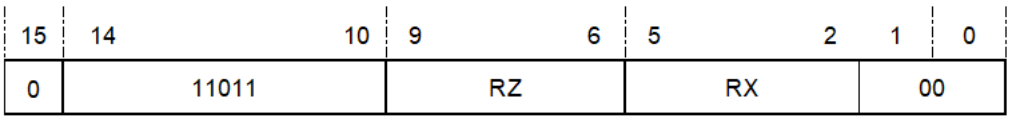


图 14.172: OR-1

32 位指令	
操作	RZ ← RX | RY
语法	or rz, rx, ry
说明	将 RX 与 RY 的值按位或，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

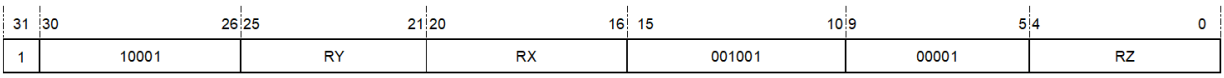


图 14.173: OR-2

14.117 ORI——立即数按位或指令

统一化指令	
语法	ori rz, rx, imm16
操作	RZ ← RX | zero_extend(IMM16)
编译结果	仅存在 32 位指令。 ori32 rz, rx, imm16
说明	将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
异常	无

32 位指令	
操作	RZ ← RX | zero_extend(IMM16)
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFFF。
语法	ori32 rz, rx, imm16
说明	将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。
异常	无

32 位指令格式：

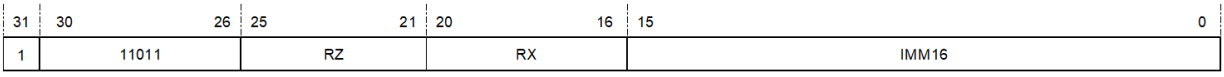


图 14.174: ORI

14.118 PLDR——读数据预取指令

统一化指令	
语法	pldr (rx, disp)
操作	读数据预取 MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 位指令。 pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令	
操作	读数据预取
语法	pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

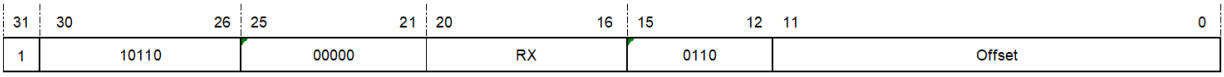


图 14.175: PLDR

14.119 PLDW——写数据预取指令

统一化指令	
语法	pldw (rx, disp)
操作	写数据预取 MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 指令。 pldw32 (r 位 x, disp)
说明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令	
操作	写数据预取
语法	pldw32 (rx, disp)
说明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

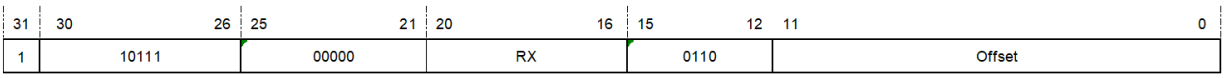


图 14.176: PLDW

14.120 POP——出栈指令

统一化指令	
语法	pop reglist

操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回；
	dst ← {reglist}; addr ← SP;
	foreach (reglist){
	Rdst ← MEM[addr];
	dst ← next {reglist};
	addr ← addr + 4;
	}
	sp ← addr;
	PC ← R15 & 0xfffffffe;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令
	if ({reglist}<16), then
	pop16 reglist;
	else
	pop32 reglist;

说明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈寄存器直接寻址方式。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	

操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回。
	dst ← {reglist}; addr ← SP;
	foreach (reglist){
	Rdst ← MEM[addr];
	dst ← next {reglist};
	addr ← addr + 4;
	}
	sp ← addr;
	PC ← R15 & 0xfffffffe;

语法	pop16 reglist
说明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令格式：

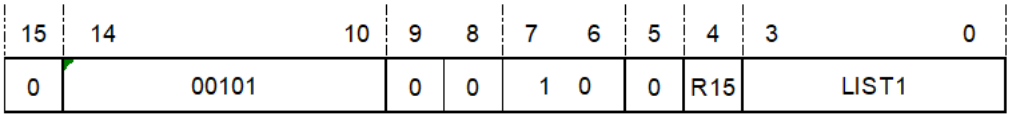


图 14.177: POP-1

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011：

r4-r6 在寄存器列表中

……

1000：

r4-r11 在寄存器列表中

R15 域：

指定寄存器 r15 是否在寄存器列表中。

0：

r15 不在寄存器列表中

1：

r15 在寄存器列表中

32 位指令	

操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中
	dst ← {reglist}; addr ← SP;
	foreach (reglist){
	Rdst ← MEM[addr];
	dst ← next {reglist};
	addr ← addr + 4;
	}
	sp ← addr;
	PC ← R15 & 0xfffffffe;

语法	pop32 reglist
说明	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15, r16 - r17, r29。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

LIST1 域：

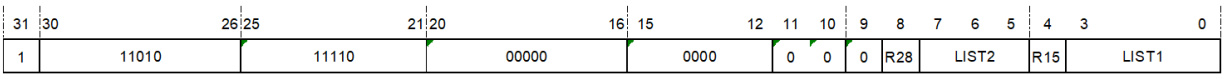


图 14.178: POP-2

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011：

r4-r6 在寄存器列表中

……

1000：

r4-r11 在寄存器列表中

R15 域：

指定寄存器 r15 是否在寄存器列表中。

0：

r15 不在寄存器列表中

1：

r15 在寄存器列表中

LIST2 域：

指定寄存器 r16-r17 是否在寄存器列表中。

000：

r16-r19 不在寄存器列表中

001：

r16 在寄存器列表中

010：

r16-r17 在寄存器列表中

R28 域：

指定寄存器 r28 是否在寄存器列表中。

0：

r28 不在寄存器列表中

1：

r28 在寄存器列表中

14.121 PSRCLR——PSR 位清零指令

统一化指令	
语法	psrclr ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
操作	清除状态寄存器的某一位或几位。 PSR({EE，IE，FE，AF}) ← 0
编译结果	仅存在 32 位指令。 psrclr32 ee, ie, fe, af
属性：	特权指令
说明	选中的 PSR 位被清零（1 表示选中）。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	清除状态寄存器的某一位或几位 PSR({EE，IE，FE，AF}) ← 0
语法	psrclr32 ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
属性：	特权指令
说明	选中的 PSR 位被清零（1 表示选中）。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

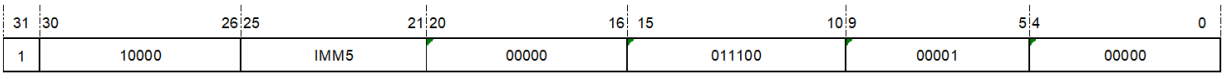


图 14.179: PSRCLR

14.122 PSRSET——PSR 位置位指令

统一化指令	
语法	psrset ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
操作	设置状态寄存器的某几位 PSR({EE，IE，FE，AF}) ← 1
编译结果	仅存在 32 位指令。 psrset32 ee, ie, fe, af
属性:	特权指令
说明	选中的 PSR 位被置位（1 表示选中）。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下: 立即数 IMM5 各位：对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	设置状态寄存器的某几位 PSR({EE，IE，FE，AF}) ← 1
语法	psrset32 ee, ie, fe, af 或者操作数也可以为 ee、ie、fe、af 的任意组合。
属性：	特权指令
说明	选中的 PSR 位被置位（1 表示选中）。五位立即数 IMM5 用于编码要清除的控制位，对应关系如下： 立即数 IMM5 各位对应的 PSR 控制位 Imm5[0]：AF Imm5[1]：FE Imm5[2]：IE Imm5[3]：EE Imm5[4]：保留
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

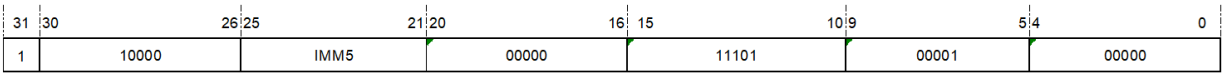


图 14.180: PSRSET

14.123 PUSH——压栈指令

统一化指令	
语法	push reglist

操作	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端;
	src ← {reglist}; addr ← SP;
	foreach (reglist){
	MEM[addr] ← Rsrc;
	src ← next {reglist};
	addr ← addr - 4;
	}
	sp ← addr;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令
	if ({reglist}<16), then
	push16 reglist;
	else
	push32 reglist;

说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	

操作	将寄存器列表中的字存储到堆栈存储器中
	src ← {reglist}; addr ← SP;
	foreach (reglist){
	MEM[addr] ← Rsrc;
	src ← next {reglist};
	addr ← addr - 4;
	}
	sp ← addr

语法	push16 reglist
说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令格式：

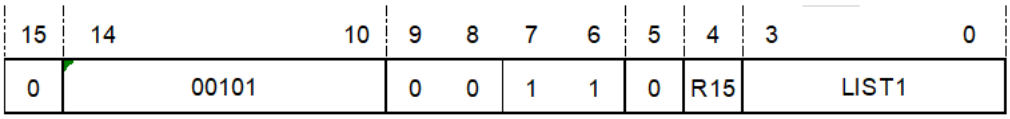


图 14.181: PUSH-1

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011：

r4-r6 在寄存器列表中

……

1000：

r4-r11 在寄存器列表中

R15 域：

指定寄存器 r15 是否在寄存器列表中。

1：

r15 在寄存器列表中

32 位指令	

操作	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中
	src ← {reglist}; addr ← SP;
	foreach (reglist){
	MEM[addr] ← Rsrc;
	src ← next {reglist};
	addr ← addr - 4;
	}
	sp ← addr

语法	push32 reglist
说明	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。
影响标志位	无影响
限制	寄存器的范围为 r4 – r11, r15, r16 - r17, r29。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

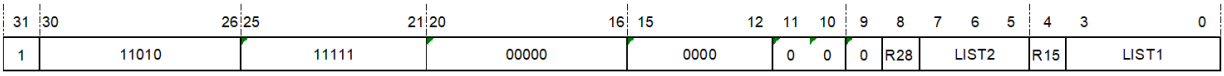


图 14.182: PUSH-2

LIST1 域：

指定寄存器 r4-r11 是否在寄存器列表中。

0000：

r4-r11 不在寄存器列表中

0001：

r4 在寄存器列表中

0010：

r4-r5 在寄存器列表中

0011：

r4-r6 在寄存器列表中

……

1000：

r4-r11 在寄存器列表中

R15 域：

指定寄存器 r15 是否在寄存器列表中。

0：

r15 不在寄存器列表中

1：

r15 在寄存器列表中

LIST2 域：

指定寄存器 r16-r17 是否在寄存器列表中。

000：

r16-r19 不在寄存器列表中

001：

r16 在寄存器列表中

010：

r16-r17 在寄存器列表中

R29 域：

指定寄存器 r29 是否在寄存器列表中。

0：

r29 不在寄存器列表中

1：

r29 在寄存器列表中

14.124 REVB——字节倒序指令

统一化指令	
语法	revb rz, rx
操作	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	revb16 rz, rx;
	else
	revb32 rz, rx;

说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];
语法	revb16 rz, rx
说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

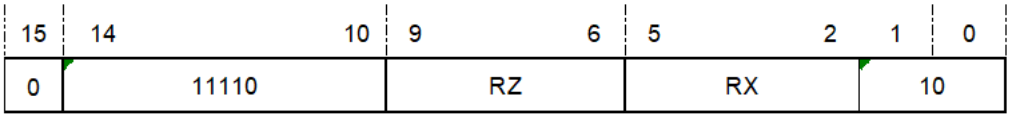


图 14.183: REVB-1

32 位指令	
操作	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];
语法	revb32 rz, rx
说明	把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

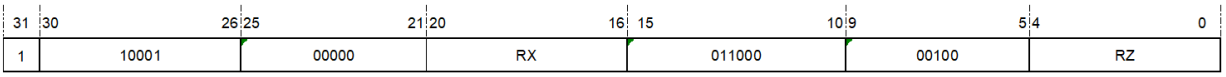


图 14.184: REVB-2

14.125 REVH——半字字节倒序指令

统一化指令	
语法	revh rz, rx
操作	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	revh16 rz, rx;
	else
	revh32 rz, rx;

说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];
语法	revh16 rz, rx
说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

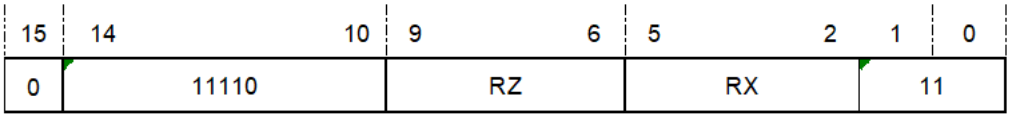


图 14.185: REVH-1

32 位指令	
操作	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];
语法	revh32 rz, rx
说明	把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

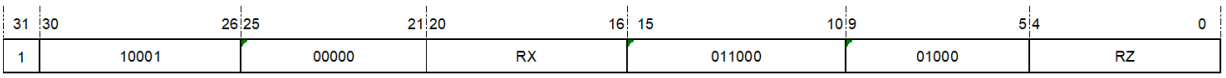


图 14.186: REVH-2

14.126 RFI——快速中断返回指令

统一化指令	
语法	rfi
操作	快速中断返回 PC ← FPC，PSR ← FPSR
编译结果	仅存在 32 位指令。 rfi 32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	快速中断返回 PC ← FPC，PSR ← FPSR
语法	rfi32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

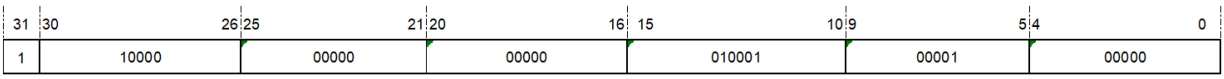


图 14.187: RFI

14.127 ROTL——循环左移指令

统一化指令		
语法	rotl rz, rx	rotl rz, rx, ry
操作	RZ ← RZ <<<< RX[5:0]	RZ ← RX <<<< RY[5:0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	rotl16 rz, rx ;
	else
	rotl32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		rotl16 rz, ry
		else
		rotl32 rz, rx, ry

说明	对于 rotl rz, rx，将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。 对于 rotl rz, rx, ry，将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ <<<< RX[5:0]
语法	rotl16 rz, rx
说明	将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

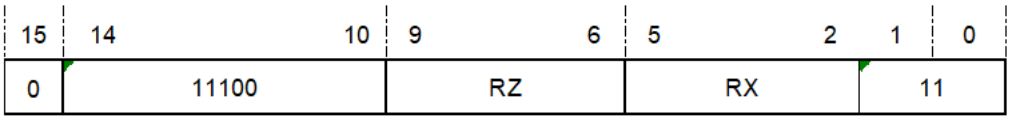


图 14.188: ROTL-1

32 位指令	
操作	RZ ← RX <<<< RY[5:0]
语法	rotl32 rz, rx, ry
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。
影响标志位	无影响
异常	无

32 位指令格式：

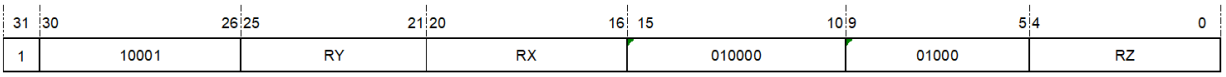


图 14.189: ROTL-2

14.128 ROTLI——立即数循环左移指令

统一化指令	
语法	rotli rz, rx, imm5
操作	RZ ← RX <<<< IMM5
编译结果	rotli32 rz, rx, imm5;
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令	
操作	RZ ← RX <<<< IMM5
语法	rotli32 rz, rx, imm5
说明	将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。
影响标志位	无影响
限制	立即数的范围为 0-31。
异常	无

32 位指令格式：

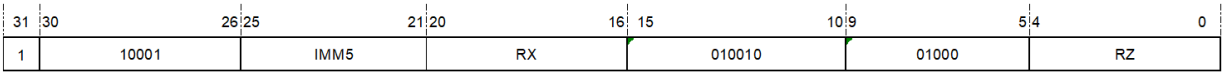


图 14.190: ROTLI

14.129 RSUB——反向减法指令

统一化指令	
语法	rsub rz, rx, ry
操作	RZ ← RY - RX
编译结果	仅存在 32 位指令。 rsub32 rz, rx, ry
说明	将 RY 的值减去 RX 值，并把结果存在 RZ 中。 注意，该指令是 subu rz, ry, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令	
操作	RZ ← RY - RX
语法	rsub32 rz, rx, ry
说明	将 RY 的值减去 RX 值，并把结果存在 RZ 中。 注意，该指令是 subu32 rz, ry, rx 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

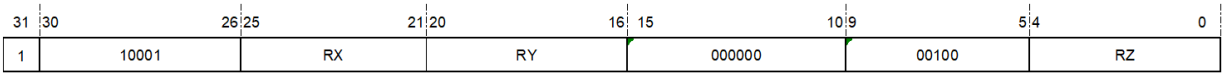


图 14.191: RSUB

14.130 RTS——子程序返回指令

统一化指令	
语法	rts
操作	程序跳转到链接寄存器指定的位置 PC ← R15 & 0xfffffffe
编译结果	总是编译为 16 位指令。 rts16
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp r15 的伪指令。
影响标志位	无影响
异常	无

16 位指令	
操作	程序跳转到链接寄存器指定的位置 PC ← R15 & 0xfffffffe
语法	rts16
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp16 r15 的伪指令。
影响标志位	无影响
异常	无

16 位指令格式：

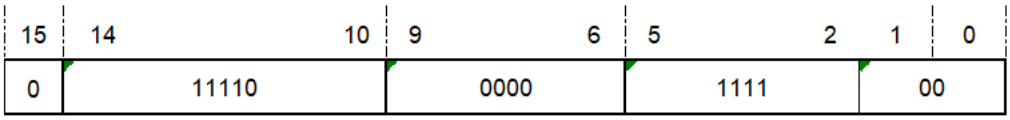


图 14.192: RTS-1

32 位指令	
操作	程序跳转到链接寄存器指定的位置 PC ← R15 & 0xfffffffe
语法	rts32
说明	程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS 指令的跳转范围是全部 4GB 地址空间。 该指令用于实现子程序返回功能。 注意，该指令是 jmp32 r15 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

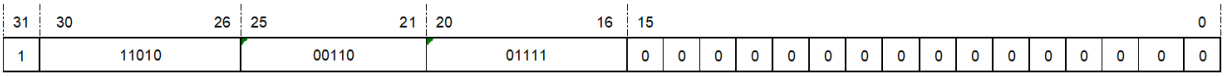


图 14.193: RTS-2

14.131 RTE——异常和普通中断返回指令

统一化指令	
语法	rte
操作	异常和普通中断返回 PC ← EPC，PSR ← EPSR
编译结果	仅存在 32 位指令。 rte32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 EPC 中的值，PSR 值恢复为保存在 EPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	异常和普通中断返回 PC ← EPC，PSR ← EPSR
语法	rte32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 EPC 中的值，PSR 值恢复为保存在 EPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

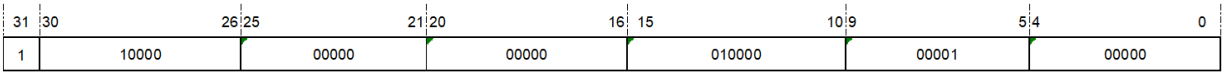


图 14.194: RTE

14.132 SCE——条件执行设置指令

统一化指令	
语法	sce cond
操作	设置其后 4 条指令的条件执行位
编译结果	仅存在 32 位指令 sce32 cond

说明:	sce 指令用于设置其后 4 条指令的条件执行位。操作数 COND 为 4 位二进制立即数，最低位表示 sce 指令后第一条指令的条件位，次低位表示 s ce 指令之后第二条指令的条件位……以此类推。条件位为 1 表示 C 为 1 正常执行，条件位为 0 表示 C 为 0 正常执行。如果 C 位不满足条件位，条件执行指令不产生任何影响。用于判定的 C 位的值以执行 sce 指令时的为准。
影响标志位:	如果后续 4 条指令产生异常或中断，条件执行位会被保存在 EPSR 或 FPSR 中。
限制:	sce 指令后面的指令只能是算术运算指令、乘除法指令、立即数寻址方式的字节、半字、字装载存储指令，且这些指令不能影响条件位 C。 操作数为 4 位二进制立即数。
异常:	无

备注:	例如指令序列为:
	sce 0101
	mov r1, r0
	mov r3, r2
	mov r5, r4
	mov r7, r6
	sce 指令的条件位为 0101。如果执行 sce 指令时，C 位为 0，则第二条和第四条 mov 指令满足执行条件，将结果写入寄存器 3 和寄存器 7；第一条和第三条 m ov 指令不满足执行条件，不将结果写入目的寄存器。

32位指令	
操作:	设置其后 4 条指令的条件执行位 set_condition_execution(COND);
语法:	sce32 cond
说明:	sce 指令用于设置其后 4 条指令的条件执行位。操作数 COND 为 4 位二进制立即数，最低位表示 sce 指令后第一条指令的条件位，次低位表示 s ce 指令之后第二条指令的条件位……以此类推。条件位为 1 表示 C 为 1 正常执行，条件位为 0 表示 C 为 0 正常执行。如果 C 位不满足条件位，条件执行指令不产生任何影响。用于判定的 C 位的值以执行 sce 指令时的为准。
影响标志位:	如果后续 4 条指令产生异常或中断，条件执行位会被保存在 EPSR 或 FPSR 中。
限制:	sce 指令后面的指令只能是算术运算指令、乘除法指令、立即数寻址方式的字节、半字、字装载存储指令，且这些指令不能影响条件位 C。 操作数为 4 位二进制立即数。
异常:	无
备注:	例如指令序列为: sce32 0101 mov32 r1, r0 mov32 r3, r2 mov32 r5, r4 mov32 r7, r6 sce 指令的条件位为 0101。如果执行 sce 指令时，C 位为 0，则第二条和第四条 mov 指令满足执行条件，将结果写入寄存器 3 和寄存器 7；第一条和第三条 m ov 指令不满足执行条件，不将结果写入目的寄存器。

指令格式:

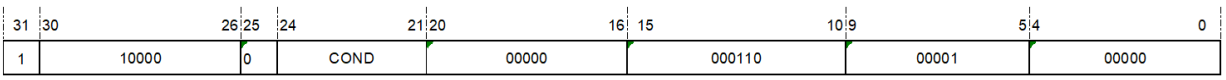


图 14.195: SCE

14.133 SEXT——位提取并有符号扩展指令

统一化指令	
语法	sext rz, rx, msb, lsb
操作	RZ ← sign_extend(RX[MSB:LSB])
编译结果	仅存在 32 位指令。 sext32 rz, rx, msb, lsb
说明	提取由 2 个 5 位立即数（MSB,LSB）所指定的 RX 的一段连续位（RX[MSB:LSB]），符号扩展至 32 位，并把结果存入 RZ。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的值为 RX[MSB]（即 RX[LSB]）一位符号扩展的结果。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令	
操作	RZ ← sign_extend(RX[MSB:LSB])
语法	sext32 rz, rx, msb, lsb
说明	提取由 2 个 5 位立即数（MSB,LSB）所指定的 RX 的一段连续位（RX[MSB:LSB]），符号扩展至 32 位，并把结果存入 RZ。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的值为 RX[MSB]（即 RX[LSB]）一位符号扩展的结果。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式：

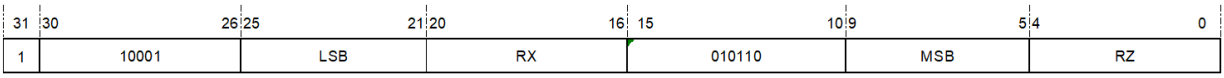


图 14.196: SEXT

MSB 域

指定被提取开始的位。

LSB 域

指定被提取结束的位。

00000

0

00000

0 位

00001

1

00001

1 位

……

11111

31

11111

31 位

14.134 SEXTB——字节提取并有符号扩展指令

统一化指令	
语法	sextb rz, rx
操作	RZ ← sign_extend(RX[7:0]);
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then sextb16 rz, rx; else sextb32 rz, rx;
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← sign_extend(RX[7:0]);
语法	sextb16 rz, rx
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

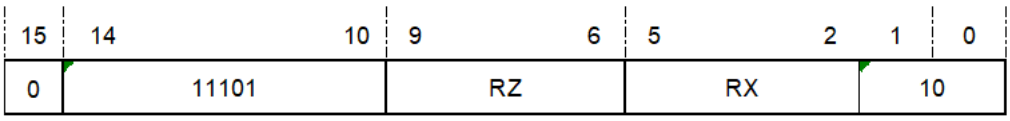


图 14.197: SEXTB-1

32 位指令	
操作	RZ ← sign_extend(RX[7:0]);
语法	sextb32 rz, rx
说明	将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。 注意，该指令是 sext32 rz, rx, 0x7, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

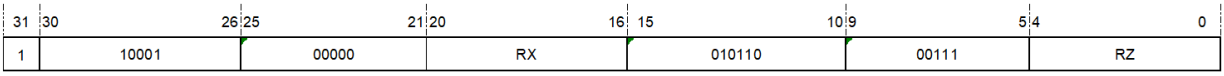


图 14.198: SEXTB-2

14.135 SEXTH——半字提取并有符号扩展指令

统一化指令	
语法	sexth rz, rx
操作	RZ ← sign_extend(RX[15:0]);
编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。 if (z<16) and (x<16), then sexth16 rz, rx; else sexth32 rz, rx;
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← sign_extend(RX[15:0]);
语法	sexth16 rz, rx
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

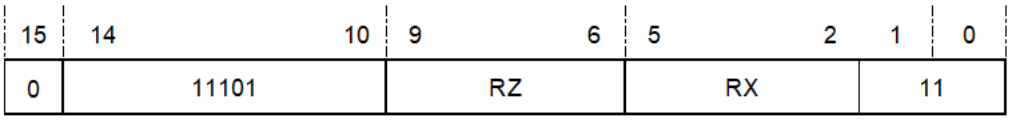


图 14.199: SEXTH-1

32 位指令	
操作	RZ ← sign_extend(RX[15:0]);
语法	sexth32 rz, rx
说明	将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。 注意，该指令是 sext32 rz, rx, 0x15, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

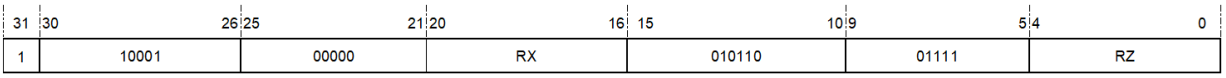


图 14.200: SEXTH-2

14.136 SRS.B——字节符号存储指令

统一化指令	
语法	srs.b rz, [label]
操作	将寄存器中的最低字节符号存储到存储器中 MEM[R28 + zero_extend(offset)] ← RZ[7:0]
编译结果	仅存在 32 位指令。 srs32.b rz, [label]
说明	将寄存器 RZ 中的最低字节符号存储到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.B 指令可以寻址 +256KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	将寄存器中的最低字节符号存储到存储器中 MEM[R28 + zero_extend(offset)] ← RZ[7:0]
语法	srs32.b rz, [label]
说明	将寄存器 RZ 中的最低字节符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.B 指令可以寻址 +256KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

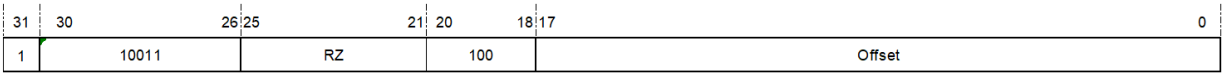


图 14.201: SRS.B

14.137 SRS.H——半字符号存储指令

统一化指令	
语法	srs.h rz, [label]
操作	将寄存器中的最低半字符号存储到存储器中 MEM[R28 + zero_extend(offset << 1)] ← RZ[7:0]
编译结果	仅存在 32 位指令。 srs32.h rz, [label]
说明	将寄存器 RZ 中的最低半字符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.H 指令可以寻址 +512KB 地址空间。 注意：偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	将寄存器中的最低半字符号存储到存储器中 MEM[R28 + zero_extend(offset << 1)] ← RZ[7:0]
语法	srs32.h rz, [label]
说明	将寄存器 RZ 中的最低半字符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 1 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.H 指令可以寻址 +512KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

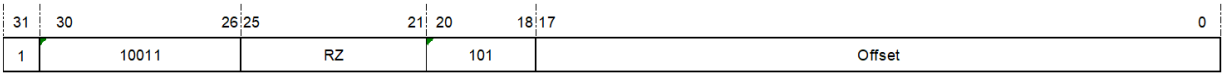


图 14.202: SRS.H

14.138 SRS.W——字符号存储指令

统一化指令	
语法	srs.w rz, [label]
操作	将寄存器中的最低字符号存储到存储器中 MEM[R28 + zero_extend(offset << 2)] ← RZ[7:0]
编译结果	仅存在 32 位指令。 srs32.w rz, [label]
说明	将寄存器 RZ 中的最低字符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令	
操作	将寄存器中的最低字符号存储到存储器中 MEM[R28 + zero_extend(offset << 2)] ← RZ[7:0]
语法	srs32.w rz, [label]
说明	将寄存器 RZ 中的最低字符号存储到到 label 所在位置。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 R X 加上左移 2 位的 18 位相对偏移量无符号扩展到 32 位后的值得到。SRS.W 指令可以寻址 +1024KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32 位指令格式：

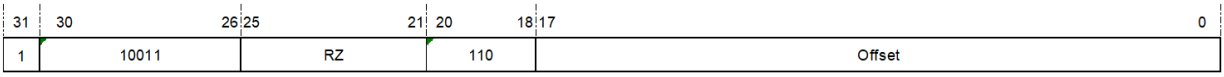


图 14.203: SRS.W

14.139 ST.B——字节存储指令

统一化指令	
语法	st.b rz, (rx, disp)
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + zero_extend(offset)] ← RZ[7:0]

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (disp<32) and (x<7) and (z<7), then
	st16.b rz, (rx, disp) ;
	else
	st32.b rz, (rx, disp) ;

说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + zero_extend(offset)] ← RZ[7:0]
语法	st16.b rz, (rx, disp)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.B 指令可以寻址 +32B 的空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

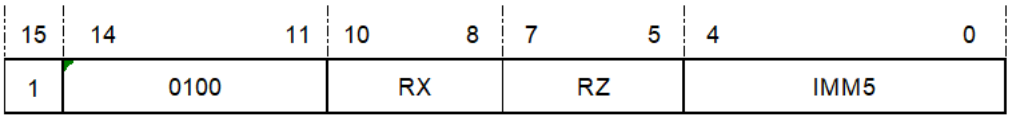


图 14.204: ST.B-1

32 位指令	
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + zero_extend(offset)] ← RZ[7:0]
语法	st32.b rz, (rx, disp)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.B 指令可以寻址 +4KB 地址空间。 注意，偏移量 DISP 即二进制操作数 Offset。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

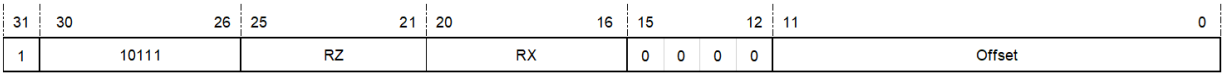


图 14.205: ST.B-2

14.140 ST.D——双字存储指令

统一化指令	
语法	st.d rz, (rx, disp)
操作	将寄存器中的双字存储到存储器中 MEM[RX + zero_extend(offset<< 2)] ← RZ[31:0] MEM[RX + zero_extend(offset<< 2) + 0x4] ← RZ + 1[31:0]
编译结果	仅存在 32 位指令 st32.d rz, (rx, disp) ;
说明	将寄存器 RZ、RZ + 1 中的双字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.D 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的双字存储到存储器中 MEM[RX + zero_extend(offset << 2)] ← RZ[31:0] MEM[RX + zero_extend(offset << 2) + 0x4] ← RZ + 1[31:0]
语法	st32.d rz, (rx, disp)
说明	将寄存器 RZ、RZ + 1 中的双字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.D 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

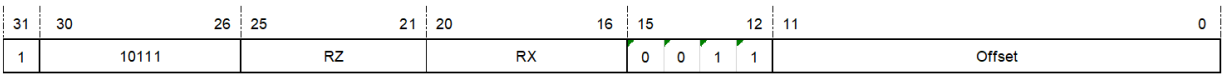


图 14.206: ST.D

14.141 ST.H——半字存储指令

统一化指令	
语法	st.h rz, (rx, disp)
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + zero_extend(offset<< 1)] ← RZ[15:0]

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (disp<64)and(x<7)and(z<7), then
	st16.h rz, (rx, disp) ;
	else
	st32.h rz, (rx, disp) ;

说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.H 指令可以寻址 +8KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的低半字存储到存储器中 MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]
语法	st16.h rz, (rx, disp)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.H 指令可以寻址 +64B 的空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

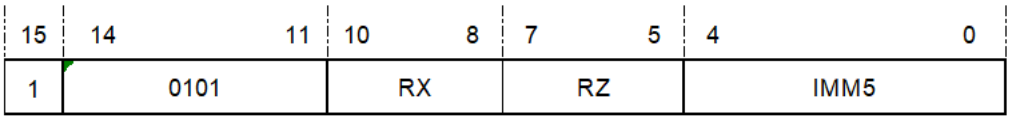


图 14.207: ST.H-1

32 位指令	
操作	将寄存器中的低半字存储到存储器中 MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]
语法	st32.h rz, (rx, disp)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.H 指令可以寻址 +8KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

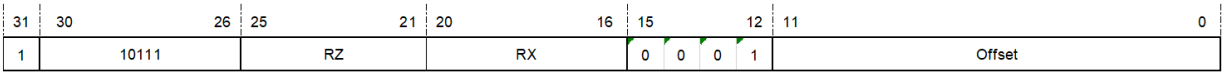


图 14.208: ST.H-2

14.142 ST.W——字存储指令

统一化指令	
语法	st.w rz, (rx, disp)
操作	将寄存器中的字存储到存储器中 MEM[RX + zero_extend(offset<< 2)] ← RZ[31:0]

编译结果	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x=sp) and (z<7) and (disp < 1024),
	st16.w rz, (sp, disp) ;
	else if (disp<128) and (x<7) and (z<7),
	st16.w rz, (rx, disp) ;
	else
	st32.w rz, (rx, disp) ;

说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.W 指令可以寻址 +16KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16 位指令	
操作	将寄存器中的字存储到存储器中 MEM[RX + zero_extend(offset << 2)] ← RZ[31:0]
语法	st16.w rz, (rx, disp) st16.w rz, (sp, disp)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。当 rx= sp 时，存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移两位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.W 指令可以寻址 +1KB 的空间。 注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数 {IMM3, IMM5} 左移两位得到的。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16 位指令格式：

st16.w rz, (rx, disp)

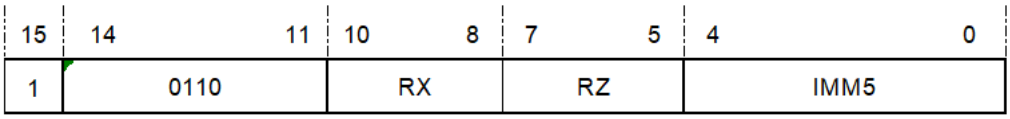


图 14.209: ST.W-1

st16.w rz, (sp, disp)

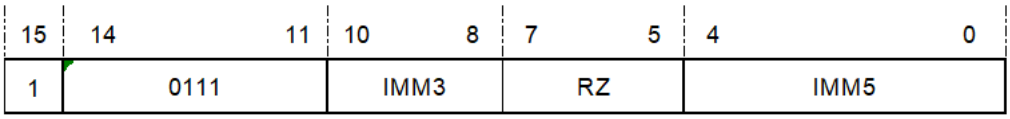


图 14.210: ST.W-2

32 位指令	
操作	将寄存器中的字存储到存储器中 MEM[RX + zero_extend(offset << 2)] ← RZ[31:0]
语法	st32.w rz, (rx, disp)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.W 指令可以寻址 +16KB 地址空间。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

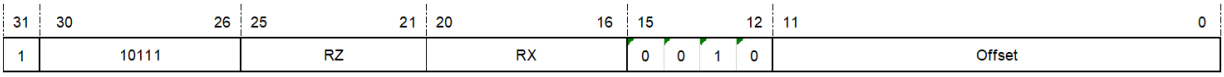


图 14.211: ST.W-3

14.143 STCPR——协处理器字存储指令

统一化指令	
语法	stcpr <cpid, cprz>, (rx, offset)
操作	将协处理器通用寄存器中的字存储到存储器中 MEM[RX + sign_extend(offset << 2)] ← CPRZ
编译结果	仅存在 32 位指令。 stcpr32 <cpid, cprz>, (rx, offset)
说明	将协处理器通用寄存器 CPRZ 中的字存储到存储器中。采用寄存器加立即数偏移量的寻址方式。指令低 12 位编码空间中 8~11 位约定为协处理器号，用于指定预操作的协处理器，其余 8 位为相对偏移量。存储器的有效地址由主流水线基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。STCPR 指令可以寻址 +1KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

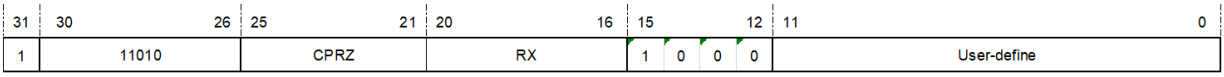


图 14.212: STCPR

14.144 STEX.W——独占式字存储指令

统一化指令	
语法	stex.w rz, (rx, disp)

操作	if 独占式字存储成功, then
	MEM[RX +
	sign_extend(offset << 2)] ← RZ;
	RZ ← 1;
	else
	RZ ← 0;

编译结果	仅存在 32 位指令。 stex32.w rz, (rx, disp)
说明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX.W 指令可以寻址 +16KB 空间。 该指令与 LDEX.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	

操作	if 独占式字存储成功, then
	MEM[RX + sign_extend(offset << 2)] ← RZ;
	RZ ← 1;
	else
	RZ ← 0;

语法	stex32.w rz, (rx, disp)
说明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX32.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX32.W 指令可以寻址 +16KB 空间。 该指令与 LDEX32.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

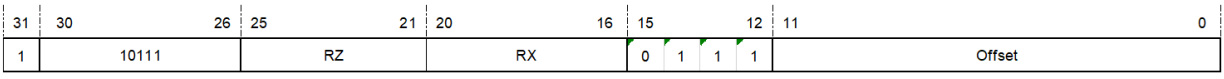


图 14.213: STEX.W

14.145 STM——连续多字存储指令

统一化指令	
语法	stm ry-rz, (rx)

操作	将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。src ← Y; addr ← RX;
	for (n = 0; n <=(Z-Y); n++){
	MEM[addr] ← Rsrc;
	src ← src + 1;
	addr ← addr + 4;
	}

编译结果	仅存在 32 位指令。 stm32 ry-rz, (rx)
说明	将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	

操作	将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。
	src ← Y; addr ← RX;
	for (n = 0; n <= IMM5; n++){
	MEM[addr] ← Rsrc;
	src ← src + 1;
	addr ← addr + 4;
	}

语法	stm32 ry-rz, (rx)
说明	将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。
影响标志位	无影响
限制	RZ 应当大于等于 RY。
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

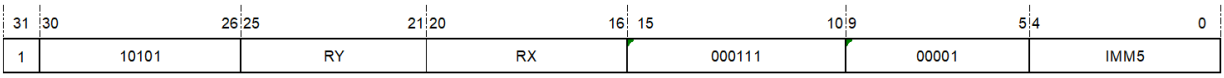


图 14.214: STM

IMM5 域：

指定目标寄存器的个数，IMM5 = Z - Y。

00000：

1 个目的寄存器

00001：

2 个目的寄存器

……

11111：

32 个目的寄存器

14.146 STOP——进入低功耗暂停模式指令

统一化指令	
语法	stop
操作	进入低功耗暂停模式
编译结果	仅存在 32 位指令。 stop32
说明	此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	进入低功耗暂停模式
语法	stop32
属性：	特权指令
说明	此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

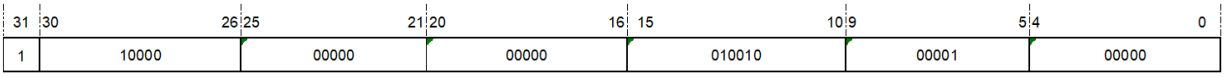


图 14.215: STOP

14.147 STQ——连续四字存储指令

统一化指令	
语法	stq r4-r7, (rx)
操作	将寄存器 R 4—R7 中的字依次存储到一片连续的存储器地址上。 src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }
编译结果	仅存在 32 位指令。 stq32 r4-r7, (rx);
说明	将寄存器堆 [R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 stm r4-r7, (rx) 的伪指令。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器 R 4—R7 中的字依次存储到一片连续的存储器地址上。 src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }
语法	stq32 r4-r7, (rx)
说明	将寄存器堆 [R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。 注意，该指令是 stm r4-r7, (rx) 的伪指令。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

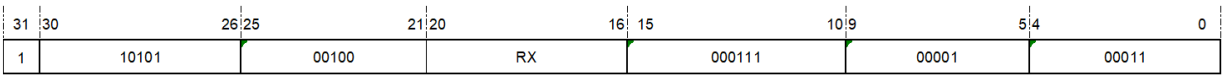


图 14.216: STQ

14.148 STR.B——寄存器移位寻址字节存储指令

统一化指令	
语法	str.b rz, (rx, ry << 0) str.b rz, (rx, ry << 1) str.b rz, (rx, ry << 2) str.b rz, (rx, ry << 3)
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + RY << IMM2] ← RZ[7:0]
编译结果	仅存在 32 位指令。 str32.b rz, (rx, ry << 0) str32.b rz, (rx, ry << 1) str32.b rz, (rx, ry << 2) str32.b rz, (rx, ry << 3)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的最低字节存储到存储器中 MEM[RX + RY << IMM2] ← RZ[7:0]
语法	str32.b rz, (rx, ry << 0) str32.b rz, (rx, ry << 1) str32.b rz, (rx, ry << 2) str32.b rz, (rx, ry << 3)
说明	将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

str32.b rz, (rx, ry << 0)

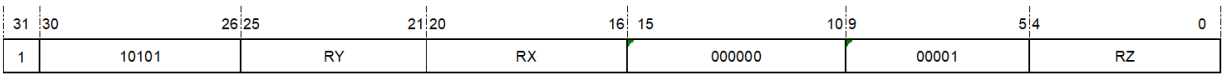


图 14.217: STR.B-1

str32.b rz, (rx, ry << 1)

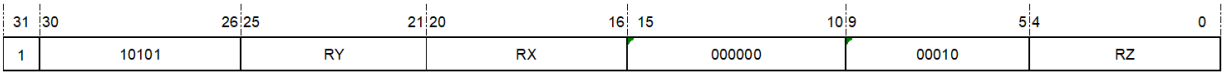


图 14.218: STR.B-2

str32.b rz, (rx, ry << 2)

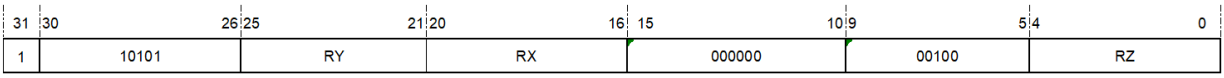


图 14.219: STR.B-3

str32.b rz, (rx, ry << 3)

14.149 STR.H——寄存器移位寻址半字存储指令

统一化指令	
语法	str.h rz, (rx, ry << 0) str.h rz, (rx, ry << 1) str.h rz, (rx, ry << 2) str.h rz, (rx, ry << 3)
操作	将寄存器中的低半字存储到存储器中 MEM[RX + RY << IMM2] ← RZ[15:0]
编译结果	仅存在 32 位指令。 str32.h rz, (rx, ry << 0) str32.h rz, (rx, ry << 1) str32.h rz, (rx, ry << 2) str32.h rz, (rx, ry << 3)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

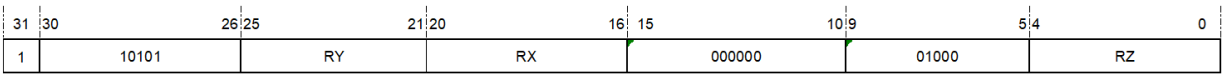


图 14.220: STR.B-4

32 位指令	
操作	将寄存器中的低半字存储到存储器中 MEM[RX + RY << IMM2] ← RZ[15:0]
语法	str32.h rz, (rx, ry << 0) str32.h rz, (rx, ry << 1) str32.h rz, (rx, ry << 2) str32.h rz, (rx, ry << 3)
说明	将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

str32.h rz, (rx, ry << 0)

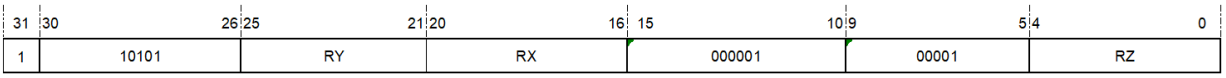


图 14.221: STR.H-1

str32.h rz, (rx, ry << 1)

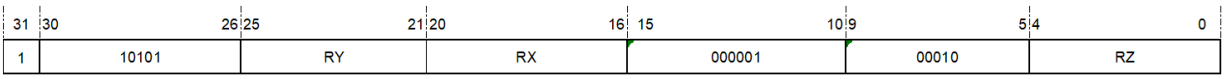


图 14.222: STR.H-2

str32.h rz, (rx, ry << 2)

str32.h rz, (rx, ry << 3)

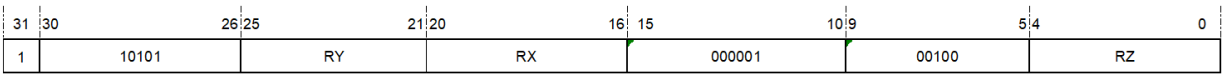


图 14.223: STR.H-3

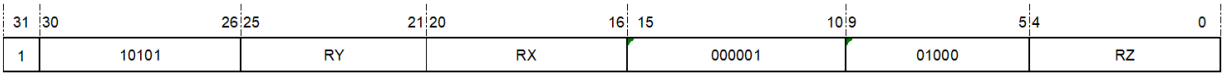


图 14.224: STR.H-4

14.150 STR.W——寄存器移位寻址字存储指令

统一化指令	
语法	str.w rz, (rx, ry << 0) str.w rz, (rx, ry << 1) str.w rz, (rx, ry << 2) str.w rz, (rx, ry << 3)
操作	将寄存器中的字存储到存储器中 MEM[RX + RY << IMM2] ← RZ[31:0]
编译结果	仅存在 32 位指令。 str32.w rz, (rx, ry << 0) str32.w rz, (rx, ry << 1) str32.w rz, (rx, ry << 2) str32.w rz, (rx, ry << 3)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	
操作	将寄存器中的字存储到存储器中 MEM[RX + RY << IMM2] ← RZ[31:0]
语法	str32.w rz, (rx, ry << 0) str32.w rz, (rx, ry << 1) str32.w rz, (rx, ry << 2) str32.w rz, (rx, ry << 3)
说明	将寄存器 RZ 中的字存储到存储器中。采用寄存器加寄存器移位寻址方式。存储器的有效地址由基址寄存器 RX 加上偏移量寄存器 RY 左移 2 位立即数 IMM2 后的值得到。IMM2 的缺省值为 0。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

str32.w rz, (rx, ry << 0)

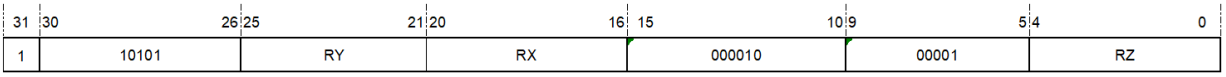


图 14.225: STR.W-1

str32.w rz, (rx, ry << 1)

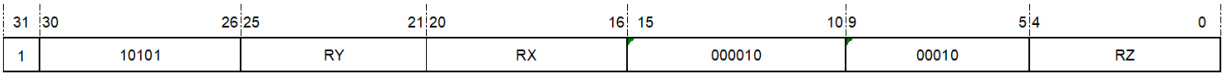


图 14.226: STR.W-2

str32.w rz, (rx, ry << 2)

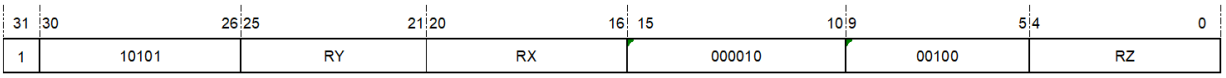


图 14.227: STR.W-3

str32.w rz, (rx, ry << 3)

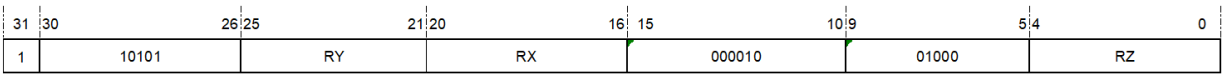


图 14.228: STR.W-4

14.151 SUBC——无符号带借位减法指令

统一化指令		
语法	subc rz, rx	subc rz, rx, ry
操作	RZ ← RZ - RX - (!C), C ← 借位	RZ ← RX - RY - (!C), C ← 借位

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	subc16 rz, rx;
	else
	subc32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		subc16 rz, ry;
		else
		subc32 rz, rx, ry;

说明	对于 subc rz, rx，将 RZ 的值减去寄存器 RX 的值和 C 位的非值；对于 subcrz, rx, ry，将 RX 的值减去寄存器 RY 的值和 C 位的非值。把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。	
影响标志位	C ← 借位	
异常	无	

16 位指令	
操作	RZ ← RZ - RX - (!C)，C ← 借位
语法	subc16 rz, rx
说明	将 RZ 的值减去寄存器 RX 的值和 C 位的非值，并把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。
影响标志位	C ← 借位
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

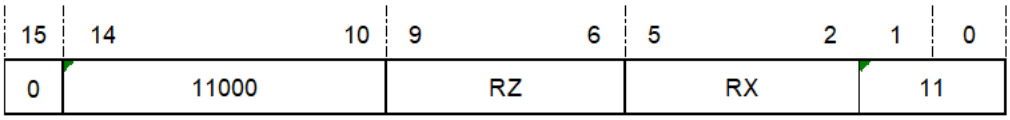


图 14.229: SUBC-1

32 位指令	
操作	RZ ← RX - RY - (!C)，C ← 借位
语法	subc32 rz, rx, ry
说明	将 RX 的值减去寄存器 RY 的值和 C 位的非值，并把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。
影响标志位	C ← 借位
异常	无

32 位指令格式：

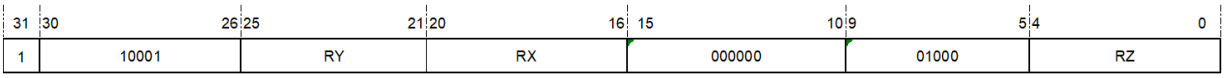


图 14.230: SUBC-2

14.152 SUBI——无符号立即数减法指令

统一化指令		
语法	ubi rz, oimm12	subi rz, rx, oimm12
S操作	RZ ← RZ - zero_extend(OIMM12)	RZ ← RX - zero_extend(OIMM12)

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (oimm12<257) and (z<8), then
	subi16 rz, oimm8;
	else
	subi32 rz, rz, oimm12;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (oimm12<8) and (z<8) and (x<8), then
		subi16 rz, rx, oimm3;
		elsif (x==z) and (z<8) and (oimm12<257), then
		subi16 rz, oimm8;
		else
		subi32 rz, rx, oimm12;

说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RZ/RX 的值减去该 32 位数，把结果存入 RZ。	
影响标志位	无影响	
限制	立即数的范围为 0x1-0x1000。	
异常	无	

16 位指令 1	
操作	RZ ← RZ - zero_extend(OIMM8)
语法	subi16 rz, oimm8
说明	将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后用 RZ 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM8 等于 OIMM8 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-256。
异常	无

16 位指令格式 1：

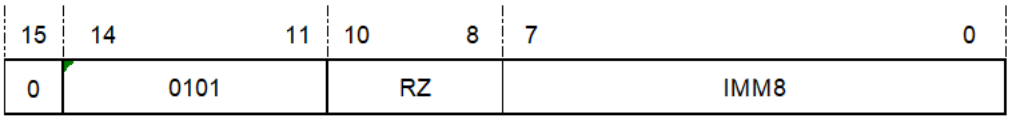


图 14.231: SUBI-1

IMM8 域：

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000：

减 1

00000001：

减 2

……

11111111：

减 256

16 位指令 2	
操作	RZ ← RX - zero_extend(OIMM3)
语法	subi16 rz, rx, oimm3
说明	将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM3 等于 OIMM3 - 1。
影响标志位	无影响
限制	寄存器的范围为 r0-r7；立即数的范围为 1-8。
异常	无

16 位指令格式 2：

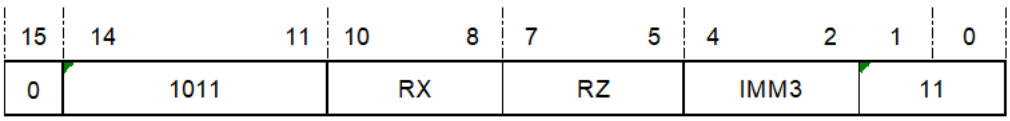


图 14.232: SUBI-2

IMM3 域

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000

减 1

001

减 2

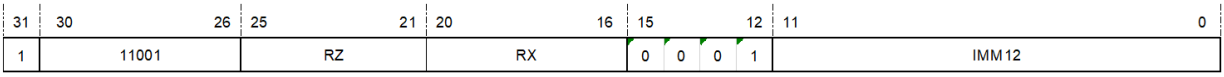
……

111

减 8

32 位指令	
操作	RZ ← RX - zero_extend(OIMM12)
语法	subi32 rz, rx, oimm12
说明	将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。 注意：二进制操作数 IMM12 等于 OIMM12 - 1。
影响标志位	无影响
限制	立即数的范围为 0x1-0x1000。
异常	无

32 位指令格式：



IMM12 域

指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

000000000000

减 0x1

000000000001

减 0x2

……

111111111111

减 0x1000

14.153 SUBI(SP)——无符号（堆栈指针）立即数减法指令

统一化指令	
语法	subi sp, sp, imm
操作	SP ← SP- zero_extend(IMM)
编译结果	仅存在 16 位指令。 subi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入 SP。
影响标志位	无影响
限制	立即数的范围为 0x0-0x1fc。
异常	无

16 位指令	
操作	SP ← SP - zero_extend(IMM)
语法	subi sp, sp, imm
说明	将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入堆栈指针。 注意：立即数（IMM）等于二进制操作数 {IMM2, IMM5} << 2。
影响标志位	无影响
限制	源与目的寄存器均为堆栈指令寄存器（R14）；立即数的范围为 (0x0-0x7f) << 2。
异常	无

16 位指令格式：

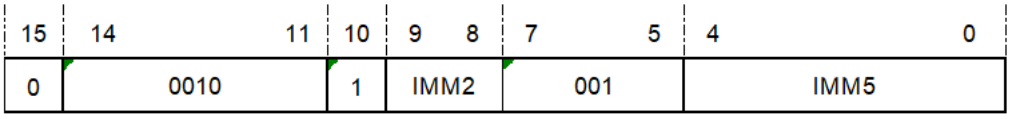


图 14.233: SUBI(SP)

IMM 域：

指定不带移位的立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 {IMM2, IMM5} 需左移 2 位。

{00, 00000}

减 0x0

{00, 00001}

减 0x4

……

{11, 11111}

减 0x1fc

14.154 SUBU——无符号减法指令

统一化指令		
语法	subu rz, rx sub rz, rx	subu rz, rx, ry
操作	RZ ← RZ - RX	RZ ← RX - RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (z<16) and (x<16), then
	subu16 rz, rx;
	else
	subu32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (x<8) and (y<8), then
		subu16 rz, rx, ry;
		elsif (x==z) and (z<16) and (y<16), then
		subu16 rz, ry;
		else
		subu32 rz, rx, ry;

说明	对于 subu rz, rx，将 RZ 的值减去 RX 值，并把结果存在 RZ 中。 对于 subu rz, rx, ry，将 RX 的值减去 RY 值，并把结果存在 RZ 中。	
影响标志位	无影响	
异常	无	

16 位指令 1	
操作	RZ ← RZ - RX
语法	subu16 rz, rx sub16 rz, rx
说明	将 RZ 的值减去 RX 值，并把结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式 1：

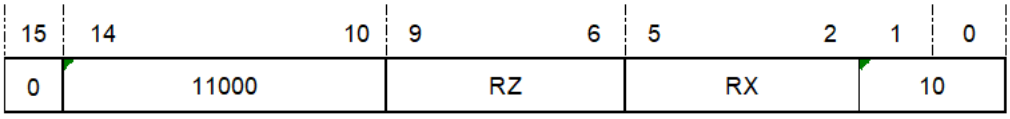


图 14.234: SUBU-1

16 位指令 2	
操作	RZ ← RX- RY
语法	subu16 rz, rx, ry sub16 rz, rx, ry
说明	将 RX 的值减去 RY 值，并把结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r7。
异常	无

16 位指令格式 2：

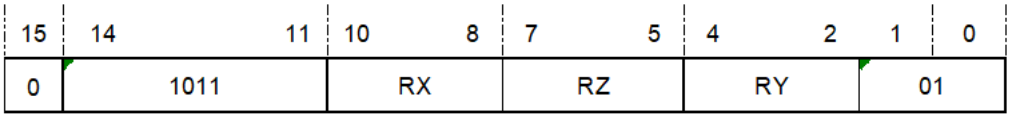


图 14.235: SUBU-2

32 位指令	
操作	RZ ← RX - RY
语法	subu32 rz, rx, ry
说明	将 RX 的值减去 RY 值，并把结果存在 RZ 中。
影响标志位	无影响
异常	无

32 位指令格式：

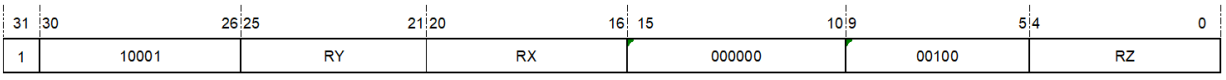


图 14.236: SUBU-3

14.155 SYNC——CPU 同步指令

统一化指令	
语法	sync.is sync.i sync.s sync
操作	该指令保证前序所有指令比该指令早退休，后续所有指令比该指令晚退休。
编译结果	仅存在 32 位指令。 sync32.is sync32.i sync32.s sync32
说明	S 表征是否广播到 CPU 核外，“1”为有效。 I 表征指令退休时是否同步取值，执行流水线清空操作，“1”为有效。
影响标志位	无影响
异常	无

32 位指令	
操作	使 CPU 同步
语法	sync32
说明	S 表征是否广播到 CPU 核外，“1”为有效。 I 表征指令退休时是否同步取值，执行流水线清空操作，“1”为有效。
影响标志位	无影响
异常	无

32 位指令格式：

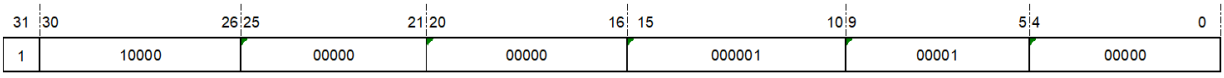


图 14.237: SYNC

14.156 TRAP——操作系统陷阱指令

统一化指令	
语法	trap 0, trap 1 trap 2, trap 3
操作	引起陷阱异常发生
编译结果 **说明**	仅存在 32 位指令。 trap32 0, trap32 1 trap32 2, trap32 3 当处理器碰到 trap 指令时，发生陷阱异常操作。
影响标志位	无影响
异常	陷阱异常

32 位指令	
操作	引起陷阱异常发生
语法	trap32 0, trap32 1, trap32 2, trap32 3
说明	当处理器碰到 trap 指令时，发生陷阱异常操作。
影响标志位	无影响
异常	陷阱异常

32 位指令格式：

trap32 0

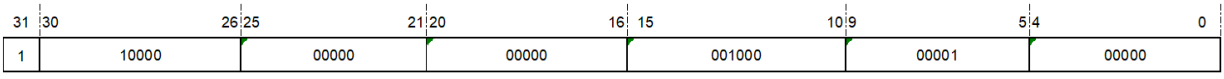


图 14.238: TRAP-1

trap32 1

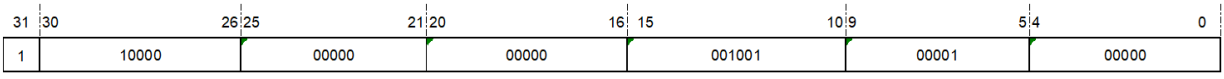


图 14.239: TRAP-2

trap32 2

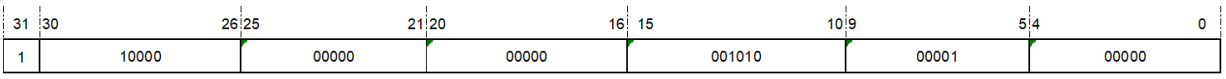


图 14.240: TRAP-3

trap32 3

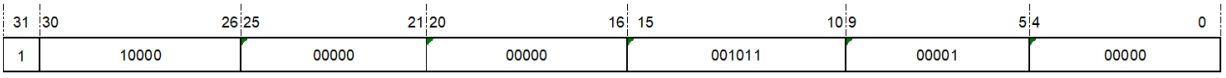


图 14.241: TRAP-4

14.157 TST——零测试指令

统一化指令	
语法	tst rx, ry

操作	If (RX & RY) != 0, then
	C ← 1;
	else
	C ← 0;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	tst16 rx, ry;
	else
	tst32 rx, ry;

说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

16 位指令	

操作	If (RX & RY) != 0, then
	C ← 1;
	else
	C ← 0;

语法	tst16 rx, ry
说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

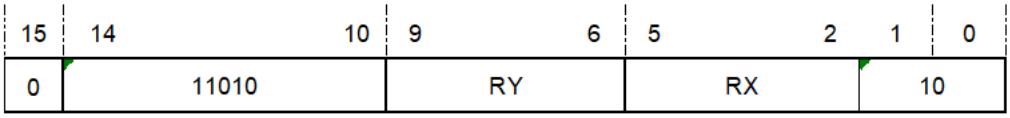


图 14.242: TST-1

32 位指令	

操作	If (RX & RY) != 0, then
	C ← 1;
	else
	C ← 0;

语法	tst32 rx, ry
说明	测试 RX 和 RY 的值按位与的结果。 如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

32 位指令格式：

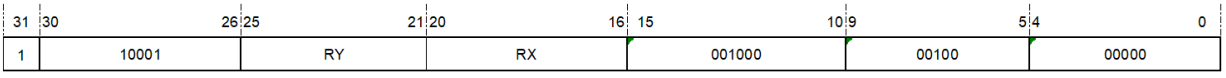


图 14.243: TST-2

14.158 TSTNBZ——无字节等于零寄存器测试指令

统一化指令	
语法	tstnbz16 rx

操作	If ((RX[31:24] != 0)
	&(RX[23:16] != 0)
	&(RX[15: 8] != 0)
	&(RX[7 : 0] != 0)), then
	C ← 1;
	else
	C ← 0;

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16), then
	tstnbz16 rx;
	else
	tstnbz32 rx;

说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

16 位指令	

操作	If ((RX[31:24] != 0)
	&(RX[23:16] != 0)
	&(RX[15: 8] != 0)
	&(RX[7 : 0] != 0)), then
	C ← 1;
	else
	C ← 0;

语法	tstnbz16 rx
说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

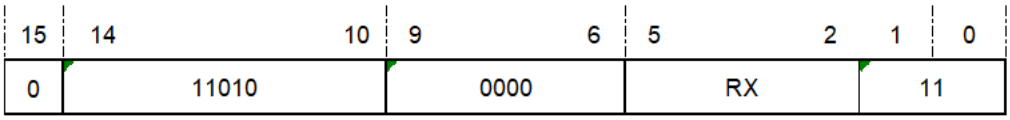


图 14.244: TSTNBZ-1

32 位指令	

操作	If ((RX[31:24] != 0)
	&(RX[23:16] != 0)
	&(RX[15: 8] != 0)
	&(RX[7 : 0] != 0)), then
	C ← 1;
	else
	C ← 0;

语法	tstnbz32 rx
说明	测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。
影响标志位	根据按位与结果设置条件位 C
异常	无

32 位指令格式：

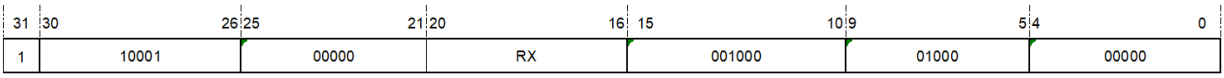


图 14.245: TSTNBZ-2

14.159 WAIT——进入低功耗等待模式指令

统一化指令	
语法	wait
操作	进入低功耗等待模式
编译结果	仅存在 32 位指令。 wait32
属性：	特权指令
说明	此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。
影响标志位	无影响
异常	特权违反指令

32 位指令	
操作	进入低功耗等待模式
语法	wait32
属性：	特权指令
说明	此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。
影响标志位	无影响
异常	特权违反指令

32 位指令格式：

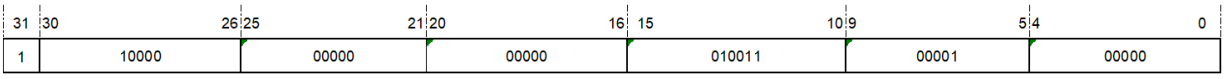


图 14.246: WAIT

14.160 XOR——按位异或指令

统一化指令		
语法	xor rz, rx	xor rz, rx, ry
操作	RZ ← RZ ^ RX	RZ ← RX ^ RY

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	xor16 rz, rx;
	else
	xor32 rz, rz, rx;
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (z<16) and (x<16), then
		xor16 rz, rx;
		else
		xor32 rz, rx, ry;

说明	将 RX 与 RZ/RY 的值按位异或，并把结果存在 RZ。	
影响标志位	无影响	
异常	无	

16 位指令	
操作	RZ ← RZ ^ RX
语法	xor16 rz, rx
说明	将 RZ 与 RX 的值按位异或，并把结果存在 RZ。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

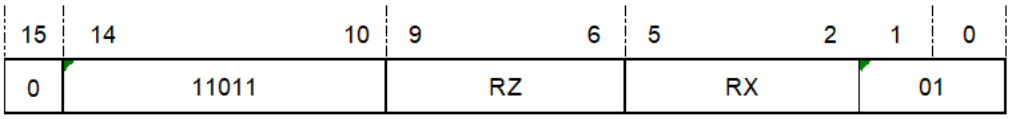


图 14.247: XOR-1

32 位指令	
操作	RZ ← RX ^ RY
语法	xor32 rz, rx, ry
说明	将 RX 与 RY 的值按位异或，并把结果存在 RZ。
影响标志位	无影响
异常	无

32 位指令格式：

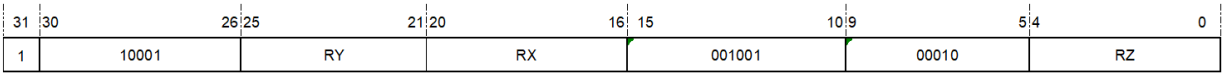


图 14.248: XOR-2

14.161 XORI——立即数按位异或指令

统一化指令	
语法	xori rz, rx, imm16
操作	RZ ← RX ^ zero_extend(IMM12)
编译结果	仅存在 32 位指令。 xori32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令	
操作	RZ ← RX ^ zero_extend(IMM12)
语法	xori32 rz, rx, imm12
说明	将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。
影响标志位	无影响
限制	立即数的范围为 0x0-0xFFF。
异常	无

32 位指令格式：

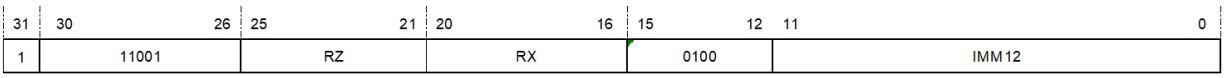


图 14.249: XORI

14.162 XSR——扩展右移指令

统一化指令	
语法	xsr rz, rx, oimm5
操作	{RZ,C} ←{RX,C} >>>> OIMM5
编译结果	仅存在 32 位指令。 xsr32 rz, rx, oimm5
说明	将 RX 带条件位 C 的值（{RX,C}）进行循环右移（原值右移，左侧移入右侧移出的位），把移位结果的最低位（[0]）存入条件位 C，高位（[32:1]）存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令	
操作	{RZ,C} ←{RX,C} >>>> OIMM5
语法	xsr32 rz, rx, oimm5
说明	将 RX 带条件位 C 的值（{RX,C}）进行循环右移（原值右移，左侧移入右侧移出的位），把移位结果的最低位（[0]）存入条件位 C，高位（[32:1]）存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位。 注意：二进制操作数 IMM5 等于 OIMM5 - 1。
影响标志位	C ← RX[OIMM5 - 1]
限制	立即数的范围为 1-32。
异常	无

32 位指令格式：

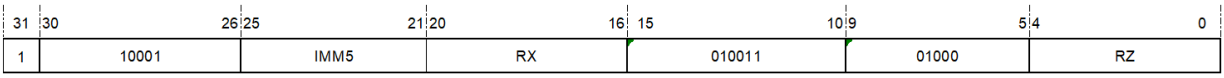


图 14.250: XSR

IMM5 域

指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000

移 1 位

00001

移 2 位

……

11111

移 32 位

14.163 XTRB0——提取字节 0 并无符号扩展指令

统一化指令	
语法	xtrb0 rz, rx

操作	RZ ← zero_extend(RX[31:24]);
	if (RX[31:24] == 0), then
	C ← 0;
	else
	C ← 1;

编译结果	仅存在 32 位指令。 xtrb0.32 rz, rx
说明	提取 RX 的字节 0（RX[31:24]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令	

操作	RZ ← zero_extend(RX[31:24]);
	if (RX[31:24] == 0), then
	C ← 0;
	else
	C ← 1;

语法	xtrb0.32 rz, rx
说明	提取 RX 的字节 0（RX[31:24]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令格式：

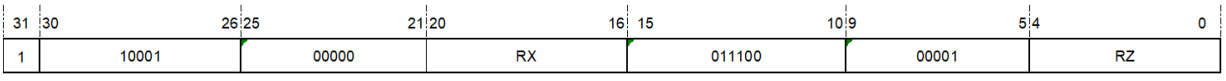


图 14.251: XTRB0

14.164 XTRB1——提取字节 1 并无符号扩展指令

统一化指令	
语法	xtrb1 rz, rx

操作	RZ ← zero_extend(RX[23:16]);
	if (RX[23:16] == 0), then
	C ← 0;
	else
	C ← 1;

编译结果	仅存在 32 位指令。 xtrb1.32 rz, rx
说明	提取 RX 的字节 1（RX[23:16]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令	

操作	RZ ← zero_extend(RX[23:16]);
	if (RX[23:16] == 0), then
	C ← 0;
	else
	C ← 1;

语法	xtrb1.32 rz, rx
说明	提取 RX 的字节 1（RX[23:16]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令格式：

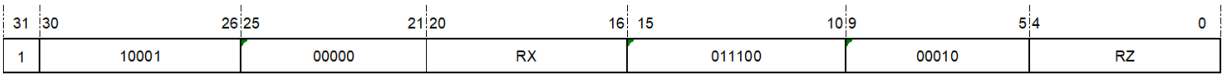


图 14.252: XTRB1

14.165 XTRB2——提取字节 2 并无符号扩展指令

统一化指令	
语法	xtrb2 rz, rx

操作	RZ ← zero_extend(RX[15:8]);
	if (RX[15:8] == 0), then
	C ← 0;
	else
	C ← 1;

编译结果	仅存在 32 位指令。 xtrb2.32 rz, rx
说明	提取 RX 的字节 2（RX[15:8]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令	

操作	RZ ← zero_extend(RX[15:8]);
	if (RX[15:8] == 0), then
	C ← 0;
	else
	C ← 1;

语法	xtrb2.32 rz, rx
说明	提取 RX 的字节 2（RX[15:8]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令格式：

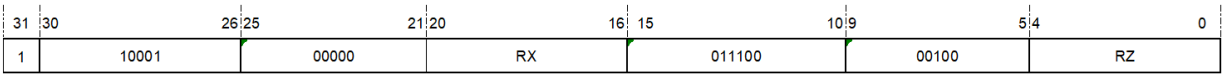


图 14.253: XTRB2

14.166 XTRB3——提取字节 3 并无符号扩展指令

统一化指令	
语法	xtrb3 rz, rx

操作	RZ ← zero_extend(RX[7:0]);
	if (RX[7:0] == 0), then
	C ← 0;
	else
	C ← 1;

编译结果	仅存在 32 位指令。 xtrb3.32 rz, rx
说明	提取 RX 的字节 3（RX[7:0]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令	

操作	RZ ← zero_extend(RX[7:0]);
	if (RX[7:0] == 0), then
	C ← 0;
	else
	C ← 1;

语法	xtrb3.32 rz, rx
说明	提取 RX 的字节 3（RX[7:0]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。
影响标志位	如果结果等于 0，则清除 C 位，反之设置 C 位。
异常	无

32 位指令格式：

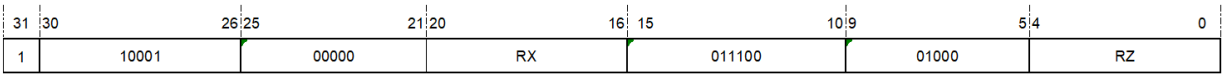


图 14.254: XTRB3

14.167 ZEXT——位提取并无符号扩展指令

统一化指令	
语法	zext rz, rx, msb,
操作 lsb	RZ ← zero_extend(RX[MSB:LSB])
编译结果	仅存在 32 位指令。 zext32 rz, rx, msb, lsb
说明	提取由 2 个 5 位立即数（MSB,LSB）所指定的 RX 的一段连续位（RX[MSB:LSB]），零扩展至 32 位，并把结果存入 RZ。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的值为 RX[MSB]（即 RX[LSB]）一位零扩展的结果。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令	
操作	RZ ← zero_extend(RX[MSB:LSB])
语法	zext32 rz, rx, msb, lsb
说明	提取由 2 个 5 位立即数（MSB,LSB）所指定的 RX 的一段连续位（RX[MSB:LSB]），零扩展至 32 位，并把结果存入 RZ。如果 MSB 等于 31，且 LSB 等于 0，则 RZ 的值与 RX 相同。如果 MSB 等于 LSB，则 RZ 的值为 RX[MSB]（即 RX[LSB]）一位零扩展的结果。如果 MSB 小于 LSB，该指令的行为不可预测。
影响标志位	无影响
限制	MSB 的范围为 0-31，LSB 的范围为 0-31，且 MSB 应当大于等于 LSB。
异常	无

32 位指令格式：

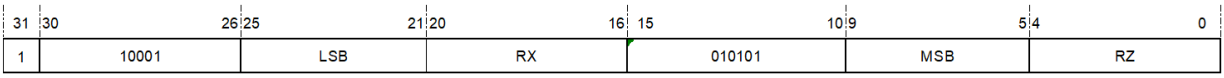


图 14.255: ZEXT

MSB 域

指定被提取开始的位。

LSB 域

指定被提取结束的位。

00000

0

00000

0 位

00001

1

00001

1 位

……

11111

31

11111

31 位

14.168 ZEXTB——字节提取并无符号扩展指令

统一化指令	
语法	zextb rz, rx
操作	RZ ← zero_extend(RX[7:0]);

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	zextb16 rz, rx;
	else
	zextb32 rz, rx

说明	将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← zero_extend(RX[7:0]);
语法	zextb16 rz, rx
说明	将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

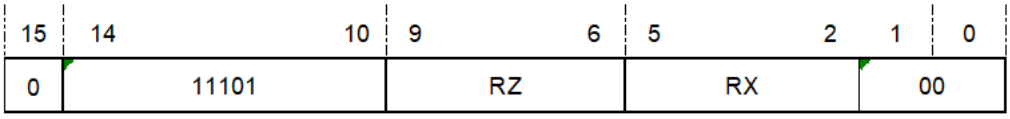


图 14.256: ZEXTB-1

32 位指令	
操作	RZ ← zero_extend(RX[7:0]);
语法	zextb32 rz, rx
说明	将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。 注意，该指令是 zext32 rz, rx, 0x7, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

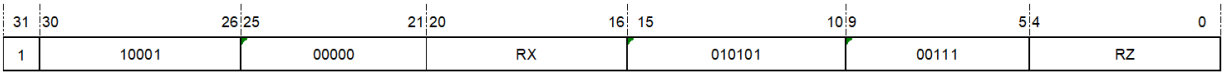


图 14.257: ZEXTB-2

14.169 ZEXTH——半字提取并无符号扩展指令

统一化指令	
语法	zexth rz, rx
操作	RZ ← zero_extend(RX[15:0]);

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (z<16), then
	zexth16 rz, rx;
	else
	zexth32 rz, rx

说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
异常	无

16 位指令	
操作	RZ ← zero_extend(RX[15:0]);
语法	zexth16 rz, rx
说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

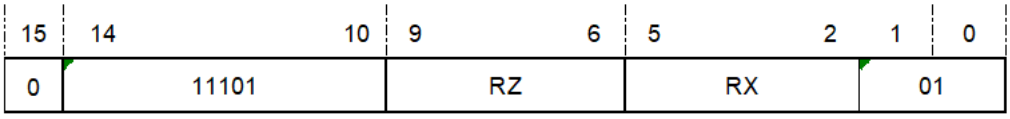


图 14.258: ZEXTH-1

32 位指令	
操作	RZ ← zero_extend(RX[15:0]);
语法	zexth32 rz, rx
说明	将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。 注意，该指令是 zext32 rz, rx, 0x15, 0x0 的伪指令。
影响标志位	无影响
异常	无

32 位指令格式：

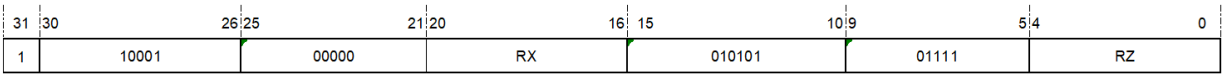


图 14.259: ZEXTH-2

