玄铁 E803 用户手册

2024 年 05 月 27 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	2020.07.28
02	配置选项描述更新	2021.01.29
03	更新模板	2021.05.18
04	修复了“系统计时器”小节重复出现的问题；一些图片文字修正	2022.03.23
05	更新模板	2024.03.23

玄铁 E803 用户手册

第一章　概述

1.1　简介

E803 是面向控制领域的 32 位高能效嵌入式 CPU 核，具有低成本、低功耗、高代码密度等多种特点。E803 采用 16/32 位混合编码指令系统，设计了精简高效的 3 级流水线。

E803 提供多种可配置功能，包括片上高速缓存、可信防护技术、片上紧耦合 IP 等，用户可根据应用需要进行配置。此外，E803 提供多个总线接口，包括系统总线、指令总线和数据总线。E803 针对内存拷贝的应用做了特殊的优化，可以获得极致的内存拷贝性能。此外，E803 针对中断响应做了特殊的加速，中断响应延时仅需 13 个周期。

1.2　特点

E803 的体系结构和编程模型的主要特点为:

- 精简指令集处理器架构（RISC）;
- 32 位数据，16 位/32 位混合编码指令;
- 16 个 32 位通用寄存器;
- 3 级流水线;
- 按序发射、按序执行、按序退休;
- 多个总线接口;
- 可配置的高速缓存;
- 可配置的可信防护技术;
- 可配置的内存保护单元（0-8）;
- 可配置的硬件乘法器，支持 1 个周期快速产生乘法结果;
- 可配置的紧耦合 IP，包括矢量中断控制器与计时器;
- 支持多种处理器时钟与系统时钟比;
- 中断响应延时仅为 13 个处理器周期;

E803 的微体系结构的主要特点为

- 静态分支预测；
- 支持硬件除法；
- 支持连续内存访问；
- 支持 big endian 和 little endian。

1.3 可配置选项

E803 可配置选项如下表所示。

表 1.1: E803 可配置选项

可配置单元	配置选项	详细
硬件乘法器	无/有	若配置则 1 个周期产生乘法结果，否则要 1-32 周期完成。
内存保护单元	0 到 8 个表项	可以配置为 0-8 个表项，其中 0 表示不实现内存保护单元。
高速缓存器	无 /2K/4K/8K/16K	可以配置为 2KB、4KB、8KB、16KB。
可信防护技术	无/有	配置该技术，结合中天微公司的 SoC 平台技术/系统软件，将提供系统的安全防护功能。
指令总线	固定包含	仅支持 AHB-L ite，直接输出（Non-Flop-out）方式
数据总线	固定包含	仅支持 AHB-L ite，直接输出（Non-Flop-out）方式
系统总线	兼容 AHB/ 兼容 AHB Lite	可以配置为兼容 AHB 协议或者兼容 AHB-Lite 协议。 仅支持直接输出（Non-Flop-out）方式。
矢量中断控制器	无 INT16/INT32 /INT64/INT128	支持硬件中断的嵌套处理。支持 16/32/64/128 个中断源。
系统计时器	无/有	用于计时。

1.4 可测性设计

E803 支持扫描链测试（SCAN）和内建自测试 (BIST)。其中，扫描链测试用于测试处理器内部的组合和时序逻辑是否存在制造错误，内建自测试用于测试高速缓存是否存在制造错误。

E803 的扫描链数目可由客户指定。

1.5 可调试性设计

E803 使用 JTAG 标准（2 线）设计硬件调试接口。E803 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。具体请详见调试接口。

1.6 命名规则

1.6.1 符号

本文档用到的标准符号和操作符如下表所示：

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: 符号列表

1.6.2 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_cpu_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　微体系结构

2.1　结构框图

E803 结构框图如下所示：

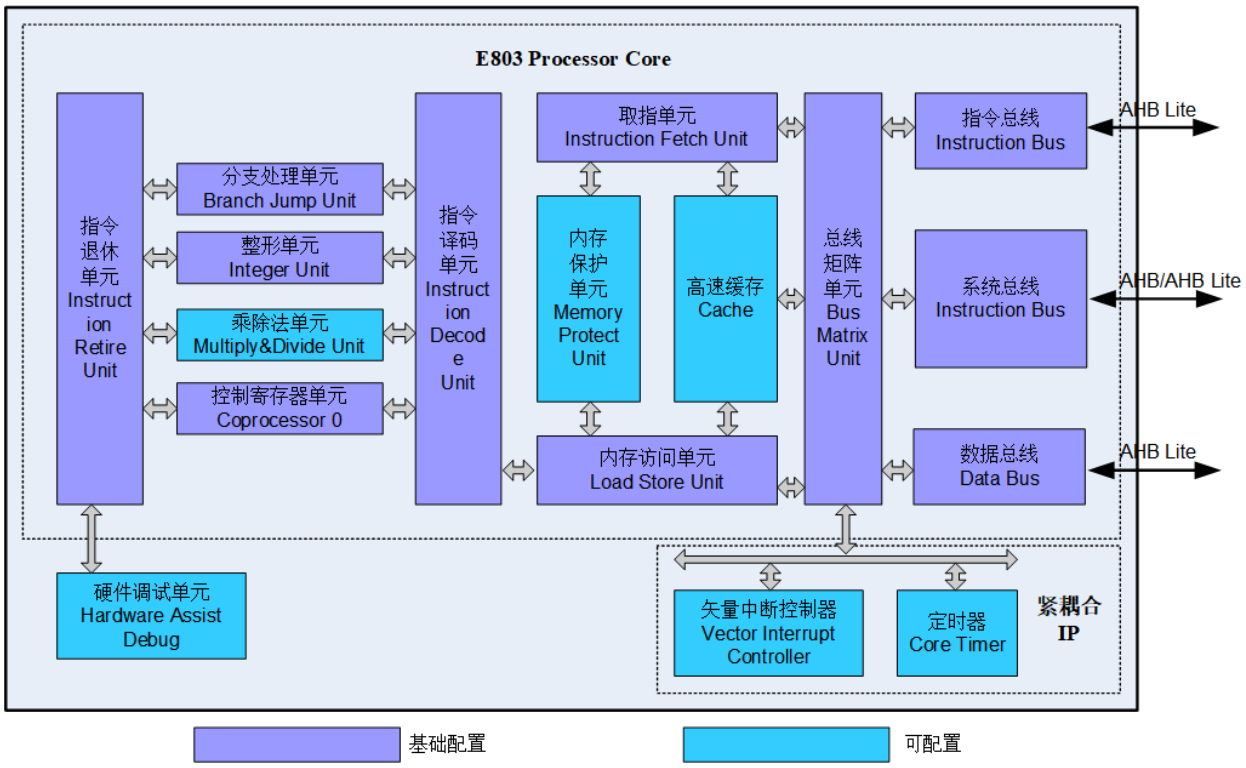


图 2.1: E803 结构框图

取指单元负责指令的访问与提取，每个时钟周期可取得 32 位数据，即每个周期可以提取 1 条 32 位指令或者 2 条 16 位指令。此外，取指单元对分支指令进行预测并对复杂指令进行分拆。

指令译码单元对指令进行译码，并访问通用寄存器以及完成指令的发射。

分支处理单元对分支预测指令进行检查并对寄存器跳转指令进行处理。整形单元负责 ALU 指令的执行。控制寄存器单元主对控制寄存器相关的指令（MTCR/MFCR）进行处理。分支指令、ALU 指令与控制寄存器指令的执行延时均为 1 个周期。

乘除法单元负责乘、乘累加以及除法指令的运算。乘法实现上，用户可配置慢速乘法与快速乘法两种方式。其中，慢速乘法的成本开销小，但是需要 1-34 个周期产生乘法结果；快速乘法支持一个周期产生乘法结果，但是成本开销大。除法需要 3-35 个周期产生运算结果。

内存访问单元负责加载（Load）/存储指令（Store）的顺序执行，支持有符号/无符号的字节/半字/字访问。加载存储指令可连续执行，实现性能的最优化。

退休单元收集指令的执行结果并进行处理，完成结果的回写以及中断与异常的处理。此外，退休单元也负责与硬件调试单元的交互。

E803 实现了可配置的内存保护单元（MPU），用于保护敏感数据的存储器访问，支持 1-8 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。

E803 实现了可配置的高速缓存单元，支持 2KB~32KB 可配置。高速缓存采用四路组相联的结构，支持写回、写通过两种工作模式。

E803 设计了多层次的总线接口，支持指令总线、数据总线和系统总线 3 种总线接口。指令/数据总线实现高性能的指令/数据访问，用户可通过指令/数据总线设计本地存储器子系统提高系统综合性能。指令/数据总线支持 AHB-Lite 接口，直接输出（Non-Flop-out，对接口时序有一定要求）方式。系统总线也仅支持直接输出，并且同时支持 AMBA2 AHB 协议和 AHB-Lite 协议可配置。CPU 时钟与系统时钟必须保持 1:1 的关系。

为了便于系统集成与开发，E803 设计了紧耦合 IP，包括矢量中断控制器和系统计时器，用户可根据应用需要进行配置。矢量中断控制器支持最多 128 个中断源，支持电平和脉冲两种中断方式，并且在硬件上支持中断嵌套。系统计时器提供了 1 个 24 位的循环递减计数器，可按照 CPU 时钟或者外部参考时钟计时，并且支持系统计时器产生中断。紧耦合 IP 的相关内容请参考*紧耦合 IP*。

E803 面向安全防护，设计了可配置的可信防护技术，配合本公司的 SoC 平台和系统软件，提供对系统的安全防护功能，具体请参考《E803 安全防护技术手册》。

2.2 流水线介绍

E803 设计了精简高效的 3 级流水线，分别是取指、译码与执行。对于绝大多数整型指令（包括分支指令、整型运算类指令、加载存储指令），均可在 3 级流水线内完成。

表 2.1: E803 流水线介绍

流水线名称	缩写	流水线作用
取指	IF	1. 发起取指令请求，处理返回的指令数据； 2. 指令预译码； 3. 静态分支预测； 4. 复杂指令拆分。
译码	ID	1. 指令译码； 2. 指令相关性分析； 3. 指令发射； 4. 执行分支跳转指令； 5. 计算加载存储指令的地址并发起访问请求。
执行	EX	1. 执行并完成整型类指令； 2. 完成加载存储指令； 3. 指令执行结果回写； 4. 指令退休； 5. 异常和中断处理。

2.3 可信防护技术

E803 面向安全领域，设计了可信防护技术，可用于系统的安全防护，主要特征包括：

- 基于同一物理处理器核，虚拟化出两个世界，分别为安全世界和非安全世界；
- 支持软件形式对存储器和 I/O 空间进行两个世界的空间划分；
- 支持可信中断；
- 支持可信调试；
- 支持可信引导；
- 更多信息请参考《E803 安全防护技术手册》

2.4 紧耦合 IP 架构

为了提高 E803 的系统集成度，方便用户集成与开发，E803 实现了一系列与 CPU 核关系密切的系统 IP，这些 IP 统称为紧耦合 IP（Titly Coupled IP，TCIP）。E803 的紧耦合 IP 包括系统计时器 CoreTim、矢量中断控制器 VIC。

矢量中断控制器的主要特征包括:

- 中断数量硬件可配置，支持 16/32/64/128 个中断源；
- 中断优先级软件可定义，可定义 4 个级别优先级；
- 支持硬件中断嵌套；
- 支持电平和脉冲两种中断源信号。

系统计时器的主要特征包括:

- 1 个 24 位的计数器；
- 支持输入时钟可选择，可以选择 CPU 时钟或外部输入时钟；
- 支持中断产生；
- 更多信息请参考紧耦合 *IP*。

第三章　编程模型

3.1　工作模式及寄存器视图

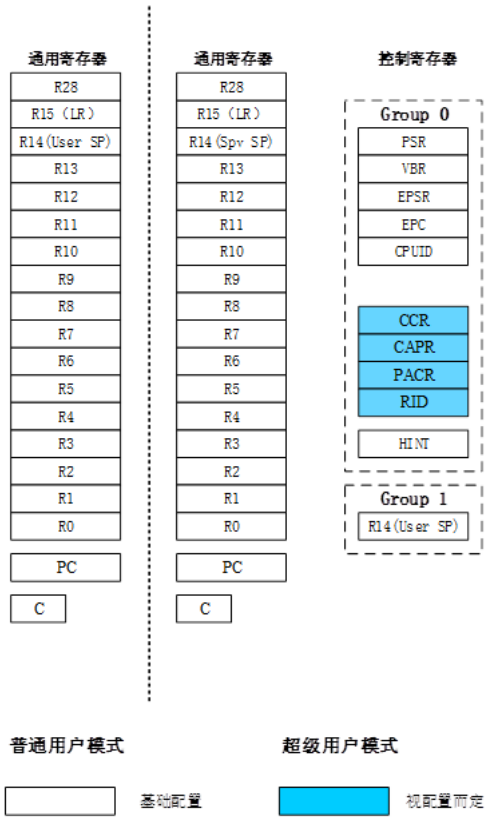


图 3.1: 编程模型

E803 定义了两种处理器工作模式：普通用户模式和超级用户模式。两种工作模式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；

2）特权指令的使用；

3）对内存空间的访问。

程序根据其权限来访问寄存器。普通用户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，并使用控制寄存器进行超级用户操作。通过对寄存器访问权限的管理，可以避免用户程序接触特权信息，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。

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。

E803 设计了内存保护单元。操作系统可通过对内存保护单元的设置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。

处理器的工作模式由处理器状态寄存器（PSR）的 S 位控制。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 0 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，处理器把模式从普通用户模式切换到超级用户模式，并把当前 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，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(从异常返回)，从异常发生的地方重新取指执行。作为系统调用指令，TRAP #n 指令为普通用户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。

普通用户模式可以操作的寄存器包括 16 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 位位于 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）。

3.2 通用寄存器

E803 包括 18 个 32 位的通用寄存器：R15~R0，R14’，R28。

表 3.1: 普通用户编程模式寄存器

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 SP)	堆栈指针（普通用户模式）
R14’(Spv SP)	堆栈指针（超级用户模式）
R15	链接寄存器
R28	不确定

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。

E803 为普通用户模式和超级用户模式分别设计了堆栈指针 R14。普通用户模式只能访问用户程序的 R14（User SP）；在超级用户模式下，系统软件不仅可以访问系统程序的 R14（Spv SP），还可以访问普通用户程序的 R14（User SP）。超级用户模式下，对通用寄存器 R14 的索引将会使用超级用户模式的寄存器 R14(Spv SP)。若用户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 条件码 / 进位标志位

条件码 / 进位标志位代表了一次操作后的结果。条件码 / 进位标志位能够作为比较操作指令的结果被设置，或者作为另一些高精度算术或逻辑指令的结果而被设置。

3.3 系统控制寄存器

3.3.1 系统控制寄存器

系统编程员用超级用户模式来设置系统操作功能，I/O 控制，以及其他受限的操作。

超级用户模式下可操作的系统控制寄存器如下所示：

- 处理器状态寄存器 (PSR)；
- 向量基址寄存器 (VBR)；
- 异常保留程序计数器 (EPC)；
- 异常保留处理器状态寄存器 (EPSR)；
- 产品序号寄存器 (CPUID)；
- 高速缓存配置寄存器 (CCR)；
- 可高缓和访问权限配置寄存器 (CAPR)*；
- 保护区控制寄存器 (PACR)*；
- 保护区选择寄存器 (PRSR)*；
- 隐式操作寄存器 (HINT)；

注意： /* 仅在特定配置时有效。

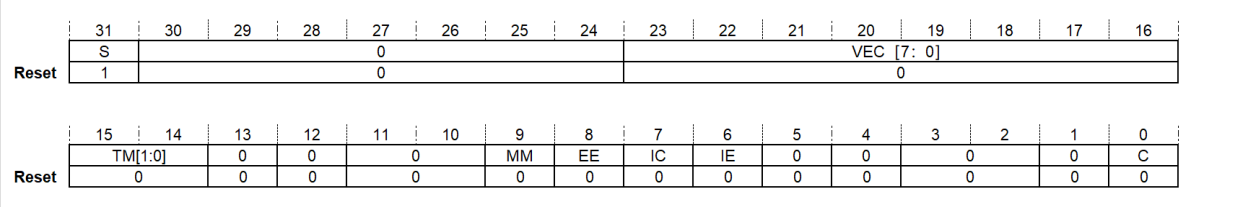
表 3.2: 超级用户编程模型控制寄存器

名称	类型	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
EPC	读/写	CR<4,0>	异常保留程序计数器
CPUID	读	CR<13,0>	产品序号寄存器
CCR	读/写	CR<18,0>	高速缓存配置寄存器
CAPR	读/写	CR<19,0>	可高缓和访问权限配置寄存器
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
CHR	读/写	CR<31,0>	隐式操作寄存器
R14(User SP)	读/写	CR<14,1>	堆栈指针寄存器

3.3.1.1 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、中断有效位和其他控制位。在超级用户模式下，软件可以访问处理器状态寄存器（PSR）。控制位为处理器指出了以下的状态：跟踪模式（TM [1:0]），超级用户模式或者普通用户模式（S 位）。它们同样也指定了中断申请是否有效。

S-超级用户模式设置位：



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	S	0							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	0	0		MM	EE	IC	IE	0	0	0		0	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

- 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 位在处理器重启和进入异常处理时由硬件置 1。

VEC[7:0]-异常向量值：

- 当异常出现时，VEC[7:0] 用来计算异常服务程序入口地址，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，E803 都将会进入跟踪异常服务程序；在跳转跟踪模式下，当碰到含有跳转（不管是跳转还是不跳转）的指令执行完，E803 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 3-5 所示为 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第六章异常处理。

MM-不对齐异常掩盖位：

- 当 MM 为 0 时，加载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，加载或存储的地址不对齐异常将会被掩盖，访问内存时加载或存储的地址低位都将会被默认为 0。

该位会被 reset 清零。

EE-异常有效控制位：

- 当 EE 为 0 时，异常无效，此时除了普通中断之外的任何异常一旦发生，都会被 E803 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和使用 EPSR 与 EPC。

IC-指令内中断控制位：

- 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 * 的指令执行时被响应；

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，中断有效；

该位会被 reset 清零，也在进入异常服务程序时被清零。

C-条件码 / 进位位：

该位用作条件判断位为一些指令服务。它在 reset 和在被拷贝到 EPSR 之后不确定。

* 部分多周期指令包括 LDM、STM、PUSH、POP、IPUSH、IPOP、LDQ32、STQ32，可以被中断而不等它们完成，从而缩短中断响应延时。多周期指令 NIE 不可响应中断，NIR 只在指令执行的末尾响应中断，不能被 PSR（IC）位打断。

3.3.1.2 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，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 是异常响应中的一部分，它将更新 PSR 中 S，TM，VEC，IE 以及 EE 位。

- RTE 指令更新 PSR：

更新 PSR 作为 RTE 指令执行的一部分，会对 PSR 中的所有位都改动。

- 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。

3.3.1.3 向量基址寄存器（VBR，CR<1,0>）

VBR 寄存器用来保存异常入口地址表的基址。VBR 的复位值为 0X00000000。

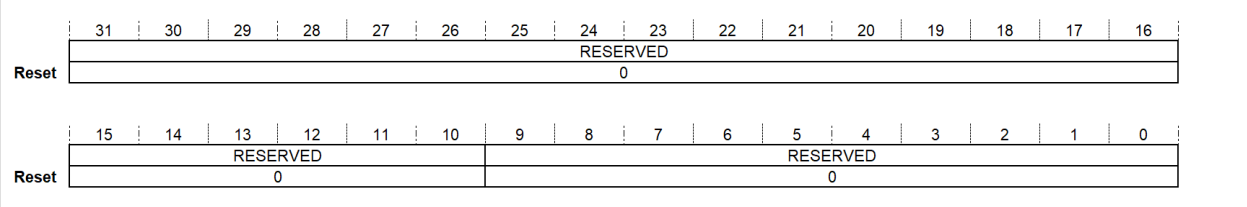


图 3.3: 基址向量寄存器

3.3.1.4 异常保留寄存器（CR<2,0>，CR<4,0>）

EPSR，EPC，寄存器在遇到异常情况时被用来保存当前处理器的状态。更详细的信息请参考第六章异常处理。

3.3.1.5 产品序号寄存器（CPUIDR，CR<13,0>）

该寄存器用于记录 CPU 的信息，包括 CPU 的型号、配置情况等。产品序号寄存器是只读的，其复位值由产品本身决定。

3.3.1.6 隐式操作寄存器（CHR，CR<31,0>）

处理器隐式操作寄存器是处理器一些隐式操作的集合，包括软复位操作以及处理器某些加速功能的使能。

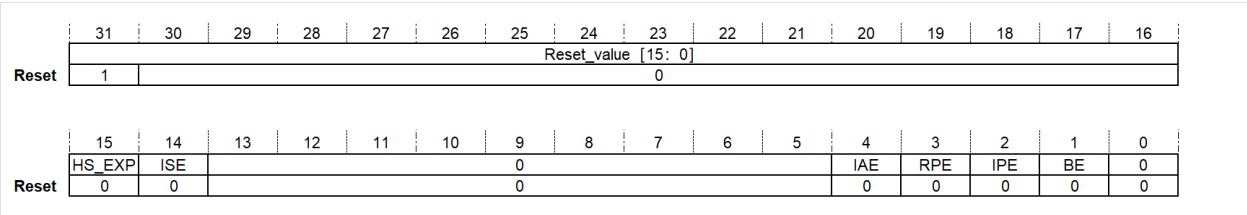


图 3.4: 隐式操作寄存器

Reset_value-软复位使能域：

- 当 Reset_Value 被写入 16’hABCD 时，将触发处理器复位操作，处理器自行复位，并通过处理器引脚 sysio_pad_srst 指示处理器发生软件复位，sysio_pad_srst 维持一个系统时钟周期；

HS_EXP-硬件压栈异常指示位：

- 当处理器支持可信执行技术时，指示可信世界的硬件压栈操作出现异常；

IA-中断响应加速控制位：

- 当 IAE 为 0 时，中断响应指令 NIE/IPUSH 将顺序执行；
- 当 IAE 为 1 时，中断响应指令 NIE/IPUSH 将投机执行，加快中断响应速度。

该位会被 reset 清零。

ISE-中断指针使能位：

- 当 ISE 位为 1 时，中断指针被使能，在处理器发生任意中断（不含 tspending），即异常向量号大于等于 32 时，均使用该指针。
- 当 ISE 位为 0 时，中断指针没有被使能，在各个状态下均使用原指针。
- 当 Int_SP 没有被配置时，中断指针使能位默认为 0。

RPE-函数返回加速控制位：

- 当 RPE 为 0 时，函数返回指令 RTS 将顺序执行；
- 当 RPE 为 1 时，函数返回指令 RTS 将投机执行，加快函数返回。

该位会被 reset 清零。

IP-指令预取加速控制位：

- 当 IP 为 0 时，指令预取加速功能不使能；
- 当 IP 为 1 时，指令预取加速功能使能，加快指令预取效率。

该位会被 reset 清零。

BE-总线 Burst 传输控制位：

- 当 BE 为 0 时，总线传输类型只支持 SINGLE；
- 当 BE 为 1 时，系统总线支持 Burst 传输类型。

该位会被 reset 清零。

3.3.1.7 其它控制寄存器

E803 的其它控制寄存器还包括：

- 高速缓存配置寄存器 (CCR)*；
- 可高缓和访问权限配置寄存器 (CAPR)*；
- 保护区控制寄存器 (PACR)*；
- 保护区选择寄存器 (PRSR)*；

其中，高速缓存配置寄存器 (CCR)、可高缓和访问权限配置寄存器 (CAPR)*、保护区控制寄存器 (PACR)*、保护区选择寄存器 (PRSR)* 是和内存保护单元设置相关的控制寄存器，在 CPU 配置内存保护单元时有效，具体的控制寄存器定义请参考第六部分-内存保护。

3.3.2 普通用户模式通用寄存器 14（R14(User SP)，CR<14,1>）

普通用户模式的通用寄存器 14 映射为控制寄存器 CR<14,1>。

3.3.3 中断指针寄存器（R14(Int_SP),CR<15,1>)

E803 可选择配置中断指针（Int_SP）用于在不同线程下共享中断堆栈空间，以节省堆栈总开销。如配置且 ISE 使能，则仅在中断（不含 tspending）中，即向量号大于等于 32 时使用该指针，硬件压栈及其余时刻均使用原指针。

在非中断的超级用户态下，该寄存器映射为控制寄存器 CR<15,1>, 即超级用户可通过访问 CR<15,1> 访问中断堆栈指针寄存器。

3.4 数据大小端

数据大小端是相对于存储器数据存储的格式提出的，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。

E803 支持小端格式和 V1 版本大端（玄铁 CPU 设计了 V1 和 V2 两个版本的大小端）格式，且不支持非对齐访问。在小端模式下，无论存取指令的访问大小为字节/半字/字，始终以最低位字节数据对应于地址的最低位；在 V1 版本大端模式下，则需要根据存取指令的访问大小（字节/半字/字）确定数据。

表 3.4: 小端/大端存储访问模式

访问大小	访问地址	小端	V1 版本大端
字	A	D3D2D1D0	D3D2D1D0
半字	A	D1D0	D3D2
半字	A+2	D3D2	D1D0
字节	A	D0	D3
字节	A+1	D1	D2
字节	A+2	D2	D1
字节	A+3	D3	D0

3.5 数据未对齐访问

数据不支持未对齐访问，当 CPU 检测到未对齐访问发生时，会进入未对齐访问异常，详情参考*未对齐访问异常（向量偏移 0X4）*。

E803 支持通过配置处理器状态寄存器 PSR 中的 MM 位，来屏蔽未对齐访问异常，详情参考*处理器状态寄存器（PSR，CR<0,0>）*。

3.6 系统地址映射

为了方便系统集成与开发，E803 对 4GB 的内存空间进行了地址划分与功能指定。总线矩阵单元对内存访问（指令或数据访问）地址进行仲裁并分发到不同的总线上。E803 推荐的系统地址划分及功能如下表所示。

为了让 SOC 设计更加灵活，芯片厂商也可以自己定义总线址空间的划分，通过配置 pad_bmu_iahbl_base，pad_bmu_iahbl_mask，pad_bmu_dahbl_base，pad_bmu_dahbl_mask 来达到目的。具体配置方法详见总线矩阵与总线接口。

表 3.5: 地址划分图

名称	内存地址空间	功能
指令总线	0x00000000-0x1FFFFFFF	存放指令
数据总线	0x20000000-0x3FFFFFFF	存放数据
系统总线	0x40000000-0xDFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP
紧耦合 IP 总线	0xE0000000-0XEFFFFFFF	紧耦合 IP 的访问地址空间
系统总线	0xF0000000-0XFFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP

E803 的总线矩阵单元只根据内存访问地址进行仲裁，而不关心内存访问类型（指令访问/数据访问/紧耦合 IP 访问）。无论取指请求还是取数据请求都可以访问任一总线及存储器空间。编程者必须保证内存访问地址的正确性，以确保成功访问目标存储器。譬如：如果指令访问地址落在了数据总线的范围内，总线矩阵单元将把指令访问请求分发到数据总线上，并从数据总线存储器中访问。

E803 具有一定的可配置性，用户可以根据应用选择是否实现紧耦合 IP 总线。总线的配置方式对内存访问的影响如图表 表 3.6 所示

表 3.6: 地址划分图-2

名称	可配置性	配置情况对内存访问的影响	
指令总线	固定包含	位于 [0x 00000000-0x1FFFF FFF] 的内存访问将分发到指令总线上	
数据总线	固定包含	位于 [0x 20000000-0x3FFFF FFF] 的内存访问将分发到数据总线上	
系统总线	固定包含	对系统总线空间的访问均被分发到系统总线上	
紧耦合 IP 总线	可配置	实现	位于 [0xE000 0000-0XEFFFFFFF] 的内存访问将分发到紧耦合 IP 总线上
		不实现	位于 [0x E0000000-0XEFFFF FFF] 的内存访问将分发到系统总线上
系统总线	固定包含	对系统总线空间的访问均被分发到系统总线上	

以上访问只会发生在不可高缓的区域或者高速缓存缺失的情况下。

3.7 内存访问顺序

E803 设计了多总线接口，系统的集成者可在总线上外接不同的存储器。由于不同总线上的存储器设备的访问延时不同，为了便于用户开发，E803 在硬件设计上保证了内存访问指令严格按照汇编指令的顺序依

次完成，避免了用户以软件的方式保证内存访问的顺序。

譬如，如下两条指令序列，Ins A 访问系统总线的存储器，Ins B 访问数据总线的存储器。假设系统总线存储器的访问延时远大于数据总线存储器，为了保证指令按照程序的顺序完成，硬件保证 Ins A 指令执行完毕，才允许 Ins B 指令执行。

Ins A: ld r4, (r7)

Ins B: ld r5, (r14)

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等。本章主要描述异常种类、异常优先级、异常向量表、异常返回和总线错误恢复等内容。

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、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、特权异常和指令跟踪，TRAP 和 BKPT 指令正常执行时也会产生异常。而且，非法指令、LD 和 ST 访问的地址没有对齐还有用户模式下执行特权指令都会产生异常。异常处理利用异常向量表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。异常在指令的边界上被处理，即 CPU 在指令退休时响应中断，并保存退出异常处理时下一条被执行的指令的地址。即使异常在取指或者译码阶段被识别，异常也要在相应的指令退休时才会被处理。E803 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。

异常按以下步骤被处理：

第一步，处理器保存 PSR 和 PC 到影子寄存器（EPSR 和 EPC）中。

第二步，将 PSR 中的超级用户模式设置位 S 位置 1（不管发生异常时处理器处于哪种运行模式），使处理器进入超级用户模式。

第三步，将 PSR 中的异常向量号 VEC 域更新为当前发生的异常向量号，标识异常类别以及支持共享异常服务的情况。

第四步，将 PSR 中的异常使能位 EE 位清零，禁止异常响应。在 EE 为零时发生的任何异常（除了普通中断），处理器都将其作为不可恢复错误异常处理。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第五步，将 PSR 中的中断使能位 IE 位清零，禁止响应中断。

以上 2-4 步，同时发生。

第六步，处理器首先根据 PSR 中的异常向量号计算得到异常入口地址，然后用该地址获得异常服务程序的第一条指令的地址。将异常向量乘以 4 后加上异常向量基准地址（存在向量基准地址寄存器 VBR 中，当 VBR 不存在时该值恒为零）即得到异常入口地址，以该异常入口地址从存储器中读取一个字，并将该字的 [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）。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。

最后一步，处理器从异常服务程序的第一条指令处开始执行并将 CPU 的控制权转交给异常服务程序，开始异常的处理。

所有的异常向量存放在超级用户允许访问的地址空间上。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。

E803 支持最大 1024 个字节的向量表，即支持 256 个异常向量（见）。开始的 31 个向量是用作在处理器内部识别的向量。第 32 个向量保留。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。（自动向量）
11 - 15	02C - 03C	保留。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3）。
20 - 21	050 - 054	保留。
22	058	TSPEND 中断
23-29	05C - 074	保留
30	078	保留。
31	07C	保留。
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 E803 内部产生的异常。E803 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；
- 陷阱指令异常；
- TSPEND 中断。

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）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服务程序入口地址，并把它装载到程序计数器（PC）。

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访问上。

4.2.3 访问错误异常（向量偏移 0X8）

如果总线访问导致了一个错误的回复，就意味着发生了访问错误异常。访问内存保护的区域出现访问错误时，也发生访问错误异常。EPC 指向发起错误访问的指令。

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、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，E803 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，TRAP，STOP，WAIT，DOZE 和 BKPT。如果 EPSR（TP）有效，跟踪异常作为 RTE 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（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 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）

E803 提供了断点指令 BKPT，退休时产生断点异常。断点异常发生时，EPC 指向该指令。

4.2.9 不可恢复错误异常（向量偏移 0X20）

当 PSR（EE）为零时，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）为零时默认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。

4.2.12 TSPEND 中断（向量偏移 0X58）

当配置矢量中断控制器时，通过软件设置 VIC_TSPEND 寄存器产生 TSPEND 中断。具体请参考紧耦合 *IP*。

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。

E803 提供了紧耦合的矢量中断控制器。用户可以选用 E803 CPU 外加矢量中断控制器的方式进行系统集成，此时只需要将外部 IP 中断源信号接入矢量中断控制器即可。用户也可以选择 CPU 核，自己在系统上集成中断控制器与其它 IP。E803 CPU 核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。

图表 图 4.1 显示了和中断相关的处理器核的接口信号。在不配置 VIC 时，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/intc_cpu_avec_b 使用自动中断向量号。如果 PSR（IE）或者 PSR（EE）被清零了，pad/intc_cpu_int_b 输入信号被屏蔽，处理器不响应中

断，当 PSR（IE）或者 PSR（EE）同时有效时，CPU 响应中断。此时，如果 pad/intc_cpu_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/intc_cpu_vec_b[7:0] 上提供的向量号。cpu_ intc/pad_int_ack 指示 CPU 已经响应了中断。pad/intc_cpu_int_b 和 pad/intc_cpu_avec_b 都是低电平有效，pad/intc_int_ack 高电平有效。在配置有 VIC 的情况下，由控制信号生成相应的中断信号，这些信号的处理及响应与上文描述一致。

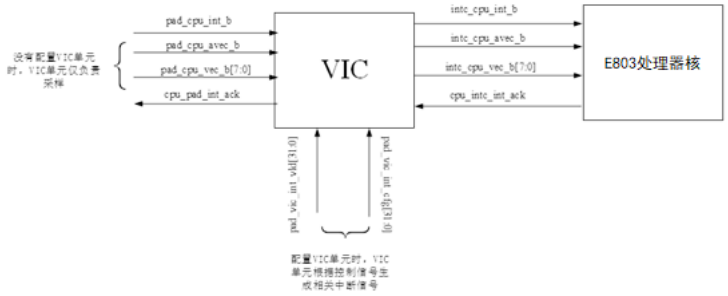


图 4.1: 中断接口信号

在上图中，当中断向量号已经准备好时，拉低 pad_cpu_int_b 中断信号线。如果没有配置 VIC 单元，则 sys_clk 采样 (T1=1cyc) 后拉低 intc_cpu_int_b 向 CPU 发送中断请求；如果配置有 VIC 则需要用 cpu_clk 经过两个周期进行采样和优先级的判断（T1=2cyc），之后才能向 CPU 发送中断请求。该信号经 CPU 内部时钟 cpu_clk 的上升沿采样后，CPU 内部收到中断，并根据向量信号取得中断向量。在外部系统均能在一个周期内响应的条件下，响应中断后，CPU 需要五个周期（T2=5cyc）才能发出中断服务程序第一条指令的取指令请求，进入中断服务程序。在中断服务程序中，应该由软件清除外部中断源，即拉高中断有效信号，此信号亦需 CPU 及外部两个时钟依次采样后才会退出中断。

E803 可配置中断加速功能。在 CPU 响应中断后，取异常入口地址的同时开始投机执行 NIE，IPUSH 指令。如中断服务程序也首先执行这两条指令，则在外部系统均能在一个周期内响应的条件下能够节省六个周期的时间。如果发生投机预测错误，访问异常或调试请求，则中止中断加速功能。

如配置相应的中断控制器，则支持多个中断来源，可分别设置其对应的中断优先级并实现中断嵌套功能。更详细的中断机制及接口信号说明可参考紧耦合 *IP*。

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，E803 把优先级分为 10 级。在图表 表 4.2 中，1 代表最高优先级，10 代表了最低优先级。值得注意的是，在第 9 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 E803 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，其余的异常可以依次重现。

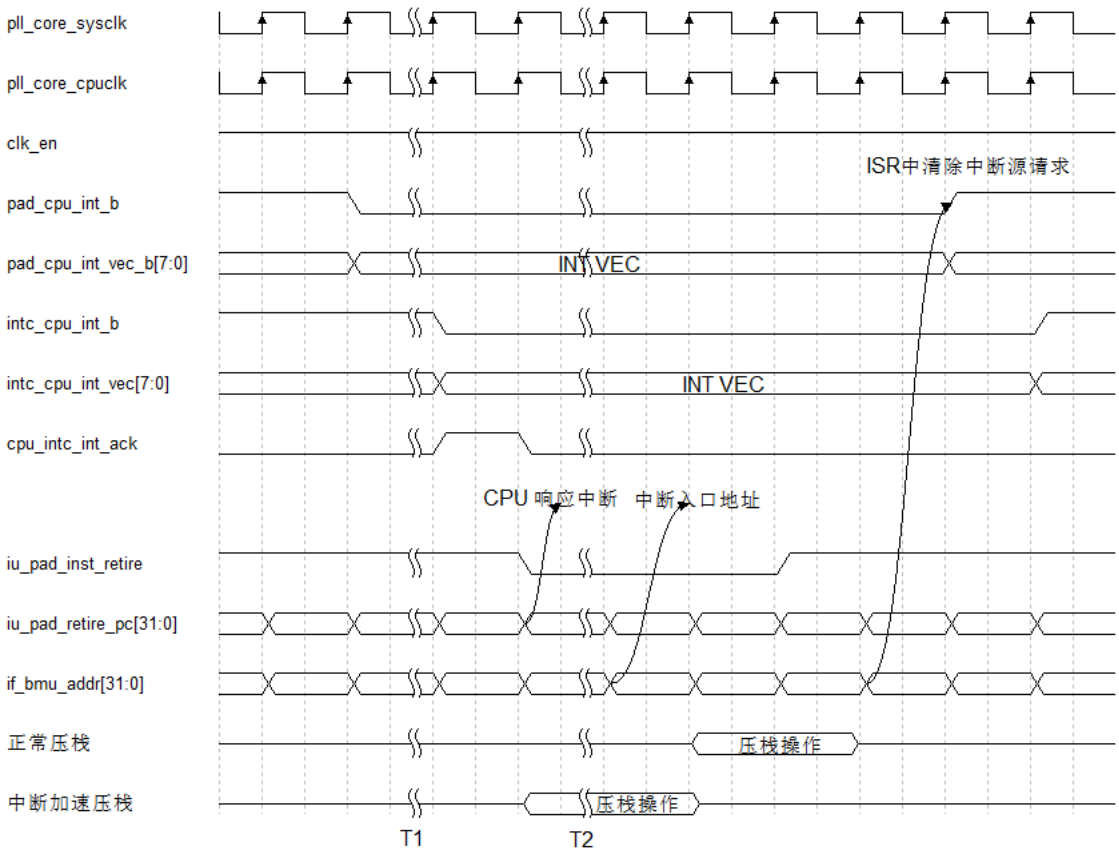


图 4.2: 处理器中断响应的时序图

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 EPSR 的 TP =1，在 RTE 指令退休后，处理器处理待处理的跟踪异常。
3	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。
4	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	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 7.1	不可恢复错误异常 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 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。

4.5 异常返回

处理器通过执行 RTE 指令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　紧耦合 IP

5.1　紧耦合 IP 简介

为了提高 E803 的系统集成度，方便用户集成与使用，E803 实现了一系列与处理器关系密切的系统关键 IP，这些 IP 统称为紧耦合 IP（Titly Coupled IP，TCIP）。E803 的紧耦合 IP 包括系统计时器 CoreTim、矢量中断控制器 VIC、片内高速缓存控制寄存器单元 CRU。这些紧耦合 IP 配合 E803，外加存储器等少量资源，便可以组成一款最小功能的 SoC 系统，提高了用户使用 E803 的便捷性，减少了 E803 的开发与应用成本。E803 的紧耦合 IP 主要功能如图表 表 5.1 ，系统结构图如图表 图 5.1 。

表 5.1: 紧耦合 IP 主要功能

IP 名	主要功能
系统计时器	完成系统的计时功能，可在低功耗时唤醒 CPU
矢量中断控制器	完成中断的收集、仲裁、硬件嵌套以及与处理器的交互
片内高速缓存控制寄存器单元	设置 E803 片内高速缓存，如开关 Cache，配置可高缓的区域等

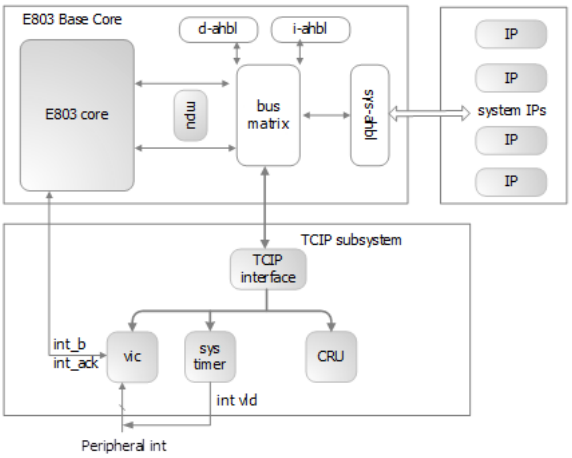


图 5.1: 紧耦合 IP 的系统结构图

与传统 IP 不同，紧耦合 IP 通过专用的紧耦合 IP 总线接口与处理器相连，无需通过系统总线访问。其中，紧耦合 IP 总线接口直接与 E803 的总线互联单元（Bus Matrix Unit，BMU）相连，支持单个 CPU 时

钟周期的紧耦合 IP 访问传输，不仅提高了紧耦合 IP 的访问效率，而且提高了系统集成效率。紧耦合 IP 与其它系统 IP 共享统一的内存地址空间，通过传输指令（Load）和存储指令（Store）进行寄存器访问和功能控制。紧耦合 IP 的内存地址分配如图表 表 5.2 所示。

表 5.2: 紧耦合 IP 的内存地址分配

IP 名	内存地址空间
系统计时器	0xE000E010-0xE000E0FF
矢量中断控制器	0xE000E100-0xE000ECFF
片内高速缓存控制寄存器单元	0xE000F000-0xE000FFFF

紧耦合 IP 除了通过专用总线接口与内核发生通信，紧耦合 IP 还以直接相连的方式与处理器进行功能交互。其中，矢量中断控器制将仲裁之后的中断信息传送给处理器并接受处理器的中断响应信号；片内高速缓存位于内核总线互联单元（BMU）和指令总线接口单元（I-AHBL）之间。

紧耦合 IP 的所有控制寄存器都是 32-bit 的寄存器，因此只能通过以 word 为单位进行访问传输，任何以 half-word 或者 byte 为单位的访问都将造成不可预期的错误，并且紧耦合 IP 寄存器只支持小端格式。所有对紧耦合 IP 寄存器的访问均需在超级用户模式下才可进行，在普通用户模式下访问会产生访问错误异常。

5.2 系统计时器

5.2.1 简介

系统计时器是 E803 的一个可选模块，它是一个 24 位的循环递减计数器。当计数器递减到 0 时，会向矢量中断控制器发起中断请求，申请获得处理器响应并处理系统计时器的事务。系统计时器的结构框图如图表 图 5.2 所示。

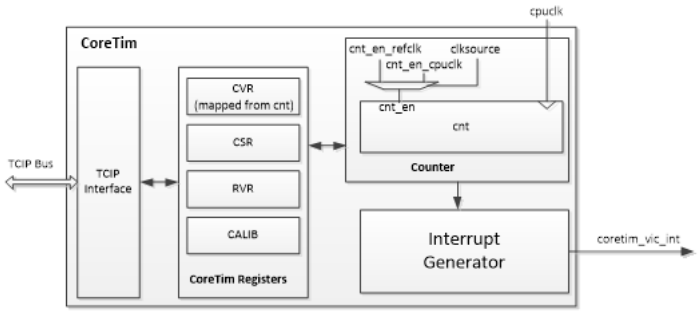


图 5.2: 系统计时器结构框图

5.2.2 寄存器定义

系统计时器每一个寄存器宽度是 32 位，寄存器地址空间为：

表 5.3: 系统计时器寄存器定义

地址	名称	类型	初始值	描述
0xE000E010	CORET_CSR	读/写	0x00000004	控制和状态寄存器
0xE000E014	CORET_RVR	读/写	-	回填值寄存器
0xE000E018	CORET_CVR	读/写	-	当前值寄存器
0xE000E01C	CORET_CALIB	只读	-	校准寄存器
0xE000E020-0xE000E0FF	-	-	-	保留

5.2.2.1 控制和状态寄存器（CORET_CSR）

CORET_CSR 是系统计时器的控制和状态寄存器，如图表 图 5.3 所示，CORET_CSR 寄存器的位说明如图表 表 5.4 所示。

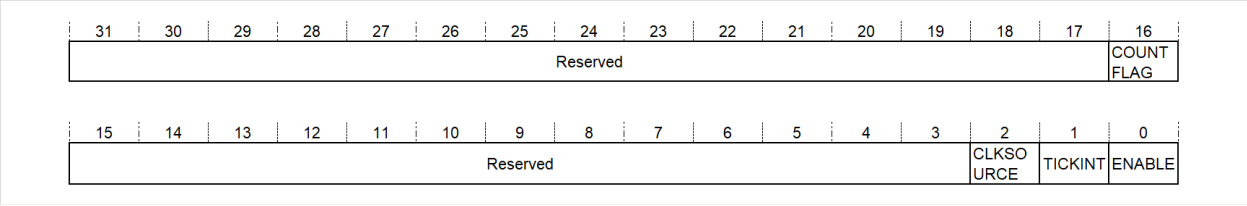


图 5.3: 系统计时器控制与状态寄存器

表 5.4: 系统计时器控制与状态寄存器域描述

位	类型	名称	描述
31:17	-	-	保留
16	只读	COUNTFLAG	中断标志位: 0 系统计数器未产生中断 1 系统计数器产生中断 计数到 0 时，COUNTFLAG 会被置 1。软件读和写 CVR 寄存器都会清零 COUNTFLAG。
15:3	-	-	保留
2	读/写	CLKSOURCE	时钟源标识位: 0 系统计数器时钟源为外部时钟源 1 系统计数器时钟源为内部时钟源 注意：写此位无作用。外部参考时钟频率必须小于或等于内部时钟频率的一半。
1	读/写	TICKINT	中断标志使能位: 0 中断不使能，计数到 0 时，COUNTFLAG 不置 1 1 中断使能，计数到 0 时，COUNTFLAG 置 1 软件写 CVR 寄存器会清零系统计数器，但不会改变系统计时器的中断标志位。
0	读/写	ENABLE	使能位: 0 系统计数器不使能 1 系统计数器使能

5.2.2.2 回填值寄存器（CORET_RVR）

CORET_RVR 寄存器用于在每一次计数循环开始时给 CORET_CVR 寄存器赋值，CORET_RVR 寄存器及其位说明如图表 图 5.4 ，图表 表 5.5 所示。

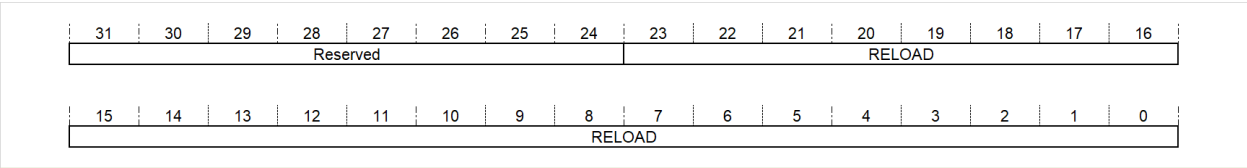


图 5.4: 系统计时器回填值寄存器

表 5.5: 系统计时器回填值寄存器域描述

位	名称	描述
31:24	-	保留
23:0	RELOAD	回填值位: 计数器使能后，当计数到 0 时，RELOAD 值会被赋给 CO RET_CVR 寄存器。向 CORET_RVR 寄存器写 0 会使计数器在下一次循环时停止工作，此后计数器的值将一直保持为 0。 注意：必须等到计数器正常计数开始后（即 CORET_CVR 变为非 0 值时），才可以将 CORET_RVR 置为 0 以让计数器在下一次循环时停止工作，否则计数器无法开始第一次计数。

如何计算 RELOAD 值

系统计时器的 RELOAD 值正常取值范围在 0x1-0x00FFFFFF。RELOAD 可以被赋值为 0，但它是没有任何作用的，因为中断标志位 COUNTFLAG 只有在计数值由 1 减为 0 时才有效。要产生一个计数周期为 N 的计时器，RELOAD 的值需要被赋为 N-1。比如要在每 100 计数时钟周期时产生一个 CoreTim 中断，需要给 RELOAD 赋值 99。

5.2.2.3 当前值寄存器（CORET_CVR）

CORET_CVR 包含了系统计时器的当前值，CORET_CVR 寄存器及其位说明如图表 图 5.5 ，图表 表 5.6 所示。

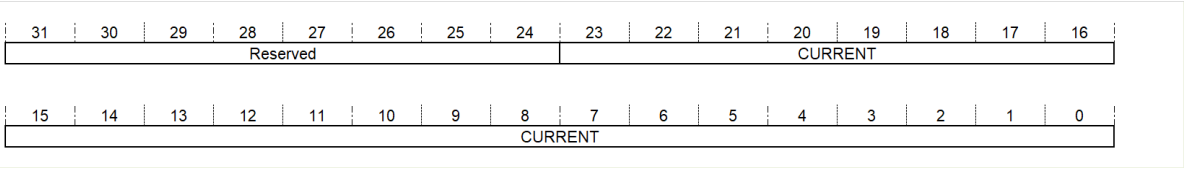


图 5.5: 系统计时器当前值寄存器

表 5.6: 系统计时器当前值寄存器域描述

位	名称	描述
31:24	-	保留
23:0	CURRENT	系统计数器当前值位。 写 CORET_CVR 寄存器会同时使此寄存器和 COUNTFLAG 状态位清零，它会导致下一个时钟周期开始时，系统计时器取出寄存器 CORET_RVR 里的值并赋给 CORET_CVR。注意写 C ORET_CVR 不会导致系统计时器的中断状态位发生改变。 读 CORET_CVR 会返回访问寄存器时计数器的值。

5.2.2.4 校准寄存器（CORET_CALIB）

CORET_CALIB 寄存器描述了系统计时器的校准功能。它的复位值跟具体实现相关：需要从设备提供商提供的文档里得到关于 CORET_CALIB 位信息的含义，以及 CORET_CALIB 寄存器中的校验值 TENMS。有了 TENMS 这个校准值，软件可以通过将这个值乘上一定的比例，从而得到其他不同的计数周期，当然这个计数周期必须在计数器的取值范围之内；CORET_CALIB 寄存器及其位说明如图表 图 5.6 ，图表 表 5.7 所示。

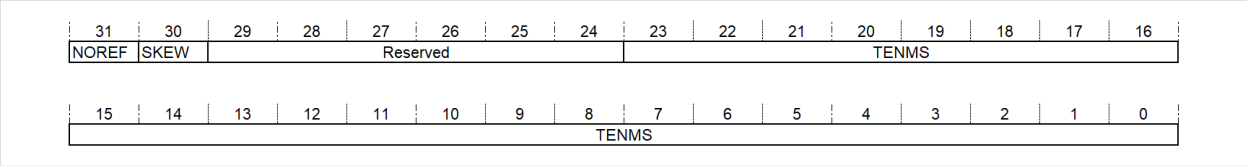


图 5.6: 系统计时器校准寄存器

表 5.7: 系统计时器校准寄存器域描述

位	名称	描述
31	NOREF	表示设备是否实现了外部参考时钟: 0 设备有外部参考时钟 1 设备没有外部参考时钟 当这位是 1 时，CORET_CSR 寄存器的 CLKSOURCE 位固定为 1，不能被改写。
30	SKEW	表示 10ms 校准值是否准确无误: 0 10ms 校准值准确无误 1 10ms 校准值由于时钟频率的问题而有误差
29:24	-	保留
23:0	TENMS	用以表示 10ms 时间对应的回填值。根据 SKEW 具体值的不同，它可能表示完全准确的 10ms 值或者最接近 10ms 的值。如果这个域的值是 0，表示校准值未知。这可能是因为参考时钟是一个未知的输入或者是动态变化的。

注意: 如果将 CORET_RVR 设置为 0，那么 CoreTim 计数器将在下一回合停止工作，而不管计数器的使能位状态。

SYST_CVR 寄存器的值在复位时是未知的。在使能 CoreTim 计数器之前，软件必须先将需要的计数值写入 CORET_RVR 寄存器，然后再向 CORET_CVR 写入任意值，后一操作会使 CORET_CVR 的值清零。这样在使能计数器之后，计数器就可以读取 CORET_RVR 回填寄存器里的值并从这个值开始向下计数，从而避免了从一个任意的值开始计数。

5.2.3 操作步骤

由于系统计时器中 CORET_RVR 和 CORET_CVR 两个寄存器没有复位值，在系统计时器工作之前，必须按照下列步骤进行操作:

1) 向 CORET_RVR 寄存器里写入需要的回填值;
2) 向 CORET_CVR 寄存器里写入任意值从而使它清零;
3) 操作 CORET_CSR 寄存器，使能系统计时器。

5.3 矢量中断控制器

5.3.1 简介

矢量中断控制器（VIC）是一个与 E803 紧耦合的 IP 单元，用于中断的高效处理。矢量中断控制器最大可支持 128 个中断源（IRQ[127:0]），每个中断源拥有独立软件可编程的中断优先级。矢量中断控制器收集来自不同中断源的中断请求，依据中断优先级对中断请求进行仲裁。最高优先级的中断将获得中断控制权并向处理器发出中断请求。当处理器响应了中断请求，处理器返回中断请求响应信号给 VIC；当处理器退出中断服务程序（ISR），处理器返回中断退出信号给 VIC。

矢量中断控制器支持中断嵌套。当处理器正在处理一个中断请求的同时来了一个更高优先级的中断请求，处理器将中断当前中断服务程序的处理，响应该更高优先级的中断请求。在更高优先级的中断请求处理结束时，CPU 返回被打断的中断服务程序继续执行。矢量中断控制器允许高优先级的中断请求抢占低优先级的中断请求，但不允许同级别或者低优先级的中断抢占，保证了中断响应的实时性。

矢量中断控制器的系统结构图如 图 5.7 所示。

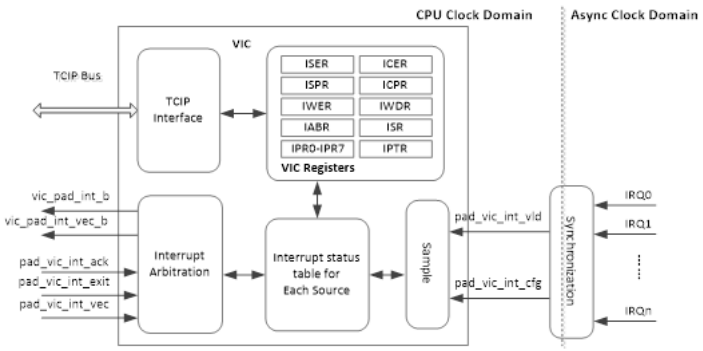


图 5.7: 矢量中断控制器系统结构图

矢量中断控制器支持以下功能：

- 中断数量硬件可配，支持 16、32、64、128 个；
- 软件通过 8 比特寄存器为每个中断配置优先级，在 E803 中若中断源数量小于或等于 32 个则仅使用 8 位中的最高 2 位有效位表征优先级高低，其余位始终保持为 0；若中断源数量为 64 个则可以使用 8 比特中的最高 3 位来表征 8 个优先级；若中断源数量为 128 个则可以使用 8 比特中的最高 4 位来表征 16 个优先级。优先级从低到高排列，0 级优先级为最高优先级，P（P=4，8，16）级优先级为最低优先级；
- 支持电平和脉冲两种中断源信号；
- 中断在处理的过程中支持优先级的动态调整，通过设置优先级阈值寄存器以较小的硬件代价实现中断优先级的反转；
- 支持中断嵌套，CPU 在执行中断服务程序过程中，允许更高优先级的中断抢占。

5.3.2 寄存器定义

VIC 提供一组 32-bit 的寄存器，各个寄存器的地址空间如 表 5.8 所示。

表 5.8: 矢量中断控制器寄存器定义

地址	名称	类型	初始值	描述
0xE000E100	VIC_ISER0	读/写	0x00000000	中断使能设置寄存器 (0-31 号中断)
0xE000E104	VIC_ISER1	读/写	0x00000000	中断使能设置寄存器 (32-63 号中断)
0xE000E108	VIC_ISER2	读/写	0x00000000	中断使能设置寄存器 (64-95 号中断)
0xE000E10C	VIC_ISER3	读/写	0x00000000	中断使能设置寄存器 (96-127 号中断)
0xE000E110- 0xE000E13F	-	-	-	保留
0xE000E140	VIC_IWER0	读/写	0x00000000	低功耗唤醒设置寄存器 (0-31 号中断)
0xE000E144	VIC_IWER1	读/写	0x00000000	低功耗唤醒设置寄存器 (32-63 号中断)
0xE000E148	VIC_IWER2	读/写	0x00000000	低功耗唤醒设置寄存器 (64-95 号中断)
0xE000E14C	VIC_IWER3	读/写	0x00000000	低功耗唤醒设置寄存器 (96-127 号中断)
0xE000E150- 0xE000E17F	-	-	-	保留
0xE000E180	VIC_ICER0	读/写	0x00000000	中断使能清除寄存器 (0-31 号中断)
0xE000E184	VIC_ICER1	读/写	0x00000000	中断使能清除寄存器 (32-63 号中断)
0xE000E188	VIC_ICER2	读/写	0x00000000	中断使能清除寄存器 (64-95 号中断)
0xE000E18C	VIC_ICER3	读/写	0x00000000	中断使能清除寄存器 (96-127 号中断)
0xE000E190- 0xE000E1BF	-	-	-	保留
0xE000E1C0	VIC_IWDR0	读/写	0x00000000	低功耗唤醒清除寄存器 (0-31 号中断)
0xE000E1C4	VIC_IWDR1	读/写	0x00000000	低功耗唤醒清除寄存器 (32-63 号中断)
0xE000E1C8	VIC_IWDR2	读/写	0x00000000	低功耗唤醒清除寄存器 (64-95 号中断)
0xE000E1CC	VIC_IWDR3	读/写	0x00000000	低功耗唤醒清除寄存器 (96-127 号中断)
0xE000E1D0- 0xE000E1FF	-	-	-	保留

下页继续

表 5.8 – 续上页

地址	名称	类型	初始值	描述
0xE000E200	VIC_ISPR0	读/写	0x00000000	中断等待设置寄存器（0-31 号中断）
0xE000E204	VIC_ISPR1	读/写	0x00000000	中断等待设置寄存器（32-63 号中断）
0xE000E208	VIC_ISPR2	读/写	0x00000000	中断等待设置寄存器（64-95 号中断）
0xE000E20C	VIC_ISPR3	读/写	0x00000000	中断等待设置寄存器（96-127 号中断）
0xE000E210- 0xE000E27F	-	-	-	保留
0xE000E280	VIC_ICPR0	读/写	0x00000000	中断等待清除寄存器（0-31 号中断）
0xE000E284	VIC_ICPR1	读/写	0x00000000	中断等待清除寄存器（32-63 号中断）
0xE000E288	VIC_ICPR2	读/写	0x00000000	中断等待清除寄存器（64-95 号中断）
0xE000E28C	VIC_ICPR3	读/写	0x00000000	中断等待清除寄存器（96-127 号中断）
0xE000E290- 0xE000E2FF	-	-	-	保留
0xE000E300	VIC_IABR0	读/写	0x00000000	中断响应状态寄存器（0-31 号中断）
0xE000E304	VIC_IABR1	读/写	0x00000000	中断响应状态寄存器（32-63 号中断）
0xE000E308	VIC_IABR2	读/写	0x00000000	中断响应状态寄存器（64-95 号中断）
0xE000E30C	VIC_IABR3	读/写	0x00000000	中断响应状态寄存器（96-127 号中断）
0xE000E310- 0xE000E3FF				
0xE000E400- 0xE000E47C	VIC_IPR0- VIC_IPR31	读/写	0x00000000	中断优先级设置寄存器
0xE000E480- 0xE000EBFF	-	-	-	保留
0xE000EC00	VIC_ISR	只读	0x00000000	中断状态寄存器
0xE000EC04	VIC_IPTR	读/写	0x00000000	中断优先级阈值寄存器
0xE000EC08	VIC_TSPEND	读/写	0x00000000	Tspending 使能设置寄存器
0xE000EC0C	VIC_TSABR	读/写	0x00000000	Tspending 响应状态寄存器
0xE000EC10	VIC_TSPR	读/写	0x00000000	Tspending 等待设置寄存器

下页继续

表 5.8 – 续上页

地址	名称	类型	初始值	描述
0xE000EC14- 0xE000ECFF	-	-	-	保留

5.3.2.1 中断使能设置寄存器（VIC_ISER）

VIC_ISER 用于使能各个中断，并且反馈各个中断的使能状态。图 5.8 描述了 VIC_ISER 的位分布，表 5.9 描述了 VIC_ISER 的位定义。

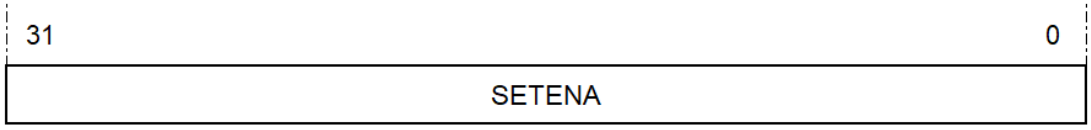


图 5.8: 中断使能设置寄存器

表 5.9: 中断使能设置寄存器域定义

位	名称	描述
31:0	SETENA	设置使用，读取一个或者多个中断的使能状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断未使能。 **1** 对应中断已使能。 **写操作 0** 无效。 **1** 使能对应中断。

如果一个处于等待状态的中断已使能，矢量中断控制器会根据其优先级激活该中断。如果一个中断未使能，该中断即使处于等待状态，矢量中断控制器也不会激活该中断。

5.3.2.2 中断低功耗唤醒设置寄存器（VIC_IWER）

VIC_IWER 用于使能各个中断的低功耗唤醒功能，并且反馈各个中断低功耗唤醒的使能状态。图 5.9 描述了 VIC_IWER 的位分布，表 5.10 描述了 VIC_IWER 的位定义。

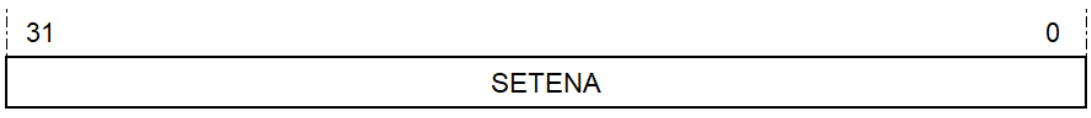


图 5.9: 中断低功耗唤醒设置寄存器

表 5.10: 低功耗唤醒使能设置寄存器域定义

位	名称	描述
31:0	SETENA	设置使用，读取一个或者多个中断低功耗唤醒的使能状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断的低功耗唤醒功能未使能。 **1** 对应中断的低功耗唤醒功能已使能。 **写操作 0** 无效。 **1** 使能对应中断的低功耗唤醒功能。

如果一个中断的低功耗唤醒功能已使能且该中断处于等待状态，VIC 产生低功耗唤醒请求。如果一个中断的低功耗唤醒功能未使能，即使该中断处于等待状态，VIC 也不产生低功耗唤醒请求。

注意： 中断使能和中断唤醒使能分别控制中断事务和中断唤醒功能。当两者都设置时，一个处于等待状态的中断既产生中断请求又产生低功耗唤醒请求；当只有其中一个使能时，只激活对应的功能；当两者都没有使能时，即使该中断处于等待状态，也不会产生中断请求或低功耗唤醒请求。

5.3.2.3 中断使能清除寄存器（VIC_ICER）

VIC_ICER 用于清除各个中断的使能，并且反馈各个中断的使能状态。图 5.10 描述了 VIC_ICER 的位分布，表 5.11 描述了 VIC_ICER 的位定义。

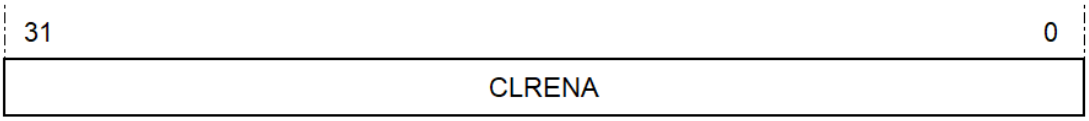


图 5.10: 中断使能清除寄存器

表 5.11: 中断使能清除寄存器域描述

位	名称	描述
31:0	CLRENA	清除使用，读取一个或者多个中断的使能状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断未使能。 **1** 对应中断已使能。 **写操作 0** 无效。 **1** 清除对应中断的使能。

5.3.2.4 中断低功耗唤醒清除寄存器（VIC_IWDR）

VIC_IWDR 用于清除各个中断的低功耗唤醒使能，并且反馈各个中断低功耗唤醒的使能状态。图 5.11 描述了 VIC_IWDR 的位分布，表 5.12 描述了 VIC_IWDR 的位定义。

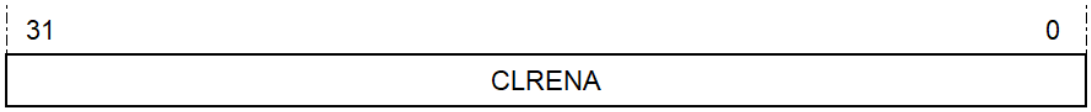


图 5.11: 中断低功耗唤醒清除寄存器

表 5.12: 中断低功耗唤醒清除寄存器域描述

位	名称	描述
31:0	CLRENA	清除使用，读取一个或者多个中断低功耗唤醒的使能状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断的低功耗唤醒功能未使能。 **1** 对应中断的低功耗唤醒功能已使能。 **写操作 0** 无效。 **1** 清除使能对应中断的低功耗唤醒功能。

5.3.2.5 中断等待设置寄存器（VIC_ISPR）

VIC_ISPR 表征设置各个中断到等待状态。图 5.12 描述了 VIC_ISPR 的位分布，表 5.13 描述了 VIC_ISPR 的位定义。

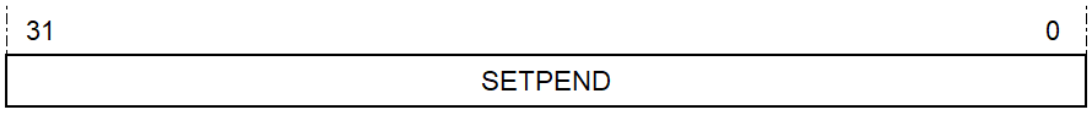


图 5.12: 中断等待设置寄存器

表 5.13: 中断等待设置寄存器域描述

位	名称	描述
31:0	SETPEND	更改一个或多个中断到等待状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断未处于等待状态。 **1** 对应中断处于等待状态。 **写操作 0** 无效。 **1** 改变对应中断到等待状态。

5.3.2.6 中断等待清除寄存器（VIC_ICPR）

VIC_ICPR 表征清除各个中断的等待状态。图表 图 5.13 描述了 VIC_ICPR 的位分布，表 5.14 描述了 VIC_ICPR 的位定义。

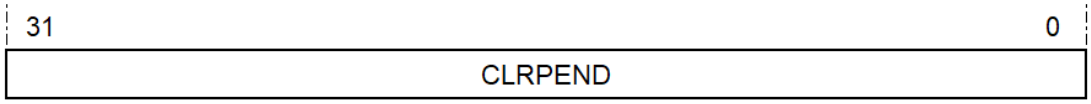


图 5.13: 中断等待清除寄存器

表 5.14: 中断等待清除寄存器

位	名称	描述
31:0	CLRPEND	清除一个或多个中断的等待状态。每一个位对应相同编号的中断源： **读操作 0** 对应中断处于未等待状态。 **1** 对应中断处于等待状态。 **写操作 0** 无效。 **1** 清除对应中断的等待状态。

5.3.2.7 中断响应状态寄存器（VIC_IABR）

VIC_IABR 用于指示各个中断当前的 Active 状态，是一个供软件查询的寄存器，另外，软件可在初始化 VIC 时，将所有中断的 Active 状态清 0。图 5.14 描述了 VIC_IABR 的位分布，表 5.15 描述了 VIC_IABR 的位定义。

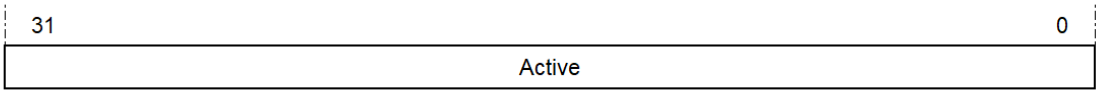


图 5.14: 中断响应状态寄存器

表 5.15: 中断响应状态寄存器域描述

位	名称	描述
31:0	Active	查询位，指示该中断源是否已经被 CPU 响应但还没处理完。每一位对应相同编号的中断源。 **读操作 0** 没有被 CPU 响应 **1** 已经被 CPU 响应但还没处理完 **写操作 0** 清除中断的 Active 状态（软件不可对该寄存器写 1，否则会导致不可预期的错误）

5.3.2.8 中断优先级设置寄存器（VIC_IPR0 - VIC_IPR31）

每个中断优先级设置寄存器提供 4 个中断源的优先级设置。根据应用的定义，每个中断源设置区域对应相应的中断源。对于硬件支持 128 个中断源的实现，寄存器从 IPR0 到 IPR31 如图表 3-17 所示，图 5.16 描述了描述了中断优先级设置寄存器。

矢量中断控制器根据优先级号选择中断优先级，优先级号越小，优先级越高。如果优先级号相同，根据中断源号决定优先级顺序，号码越小，优先级越高。

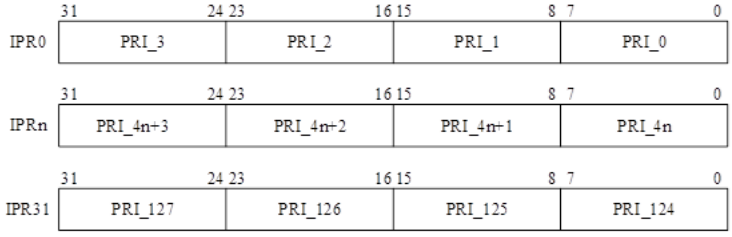


图 5.15: 中断优先级设置寄存器整体分布

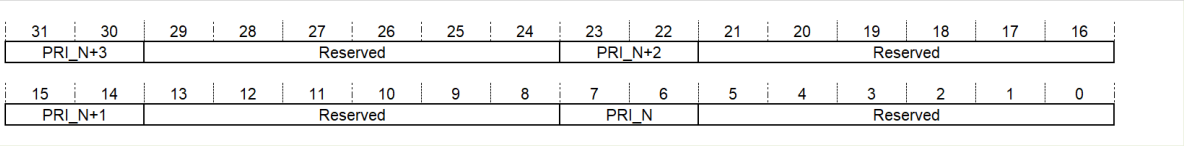


图 5.16: 中断优先级设置寄存器

表 5.16 描述了 VIC_IPRn 的位定义。在这张表中，N = 4n，n 为 VIC_IPRn 寄存器编号。如 VIC_IPR2，n 为 2 则 N 为 8。

表 5.16: 中断优先级设置寄存器域描述

优先级阶数	位	名称	描述
2 比特 4 个优先级（中断数小于等于 32 个）	31:30	PRI_N+3	中断号为 N+3 的优先级，值越小优先级越高。
	29:24	-	保留
	23:22	PRI_N+2	中断号为 N+2 的优先级，值越小优先级越高。
	21:16	-	保留
	15:14	PRI_N+1	中断号为 N+1 的优先级，值越小优先级越高。
	13:8	-	保留
	7:6	PRI_N	中断号为 N 的优先级，值越小优先级越高。
	5:0	-	保留
3 比特 8 个优先级（中断数量为 64 个）	31:29	PRI_N+3	中断号为 N+3 的优先级，值越小优先级越高。

下页继续

表 5.16 – 续上页

优先级阶数	位	名称	描述
	28:24	-	保留
	23:21	PRI_N+2	中断号为 N+2 的优先级，值越小优先级越高。
	20:16	-	保留
	15:13	PRI_N+1	中断号为 N+1 的优先级，值越小优先级越高。
	12:8	-	保留
	7:5	PRI_N	中断号为 N 的优先级，值越小优先级越高。
	4:0	-	保留
4 比特 16 个优先级（中断数量为 128 个）	31:28	PRI_N+3	中断号为 N+3 的优先级，值越小优先级越高。
	27:24	-	保留
	23:20	PRI_N+2	中断号为 N+2 的优先级，值越小优先级越高。
	19:16	-	保留
	15:12	PRI_N+1	中断号为 N+1 的优先级，值越小优先级越高。
	11:8	-	保留
	7:4	PRI_N	中断号为 N 的优先级，值越小优先级越高。
	3:0	-	保留

5.3.2.9 中断状态寄存器（VIC_ISR）

VIC_ISR 指示了当前 CPU 正在处理的中断向量号和处于等待的优先级最高的中断向量号，该寄存器是一个供软件查询的只读寄存器。图 5.17 描述了 VIC_ISR 的位分布，表 5.17 描述了 VIC_ISR 的位定义。

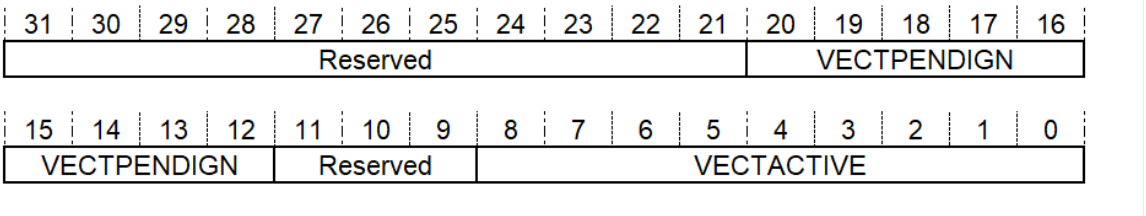


图 5.17: 中断状态寄存器

表 5.17: 中断状态寄存器域描述

位	名称	描述
31:21	Reserved	保留
20:12	VECTPENDING	指示当前处于等待状态的优先级最高的中断向量号;
11:9	Reserved	保留
8:0	VECTACTIVE	指示 CPU 当前正在处理的中断向量号

5.3.2.10 中断优先级阈值寄存器（VIC_IPTR）

VIC_IPTR 定义了当前处于等待状态的中断请求能够发起中断抢占的优先级临界值。处于等待状态的中断请求的优先级必须高于 VIC_IPTR 定义的优先级阈值，才能发起中断抢占请求。图 5.18 描述了 VIC_IPTR 的位分布，表 5.18 描述了 VIC_IPTR 的位定义。

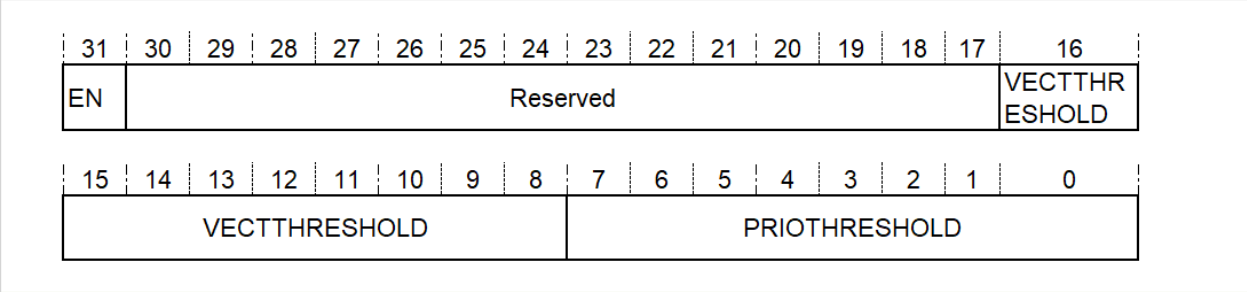


图 5.18: 中断优先级阈值寄存器

表 5.18: 中断优先级阈值寄存器域描述

位	名称	描述
31	EN	中断优先级阈值有效位: **0** 中断抢占不需要优先级高于阈值 **1** 中断抢占需要优先级高于阈值
30:17	Reserved	保留
16:8	VECTTHRESHOLD	指示优先级阈值对应的中断向量号。当 V IC 发现 CPU 从 VECTTHRESHOLD 对应的中断服务程序退出时，硬件自动清除中断优先级阈值有效位。
7:0	PRIOTHRESHOLD	指示中断抢占的优先级阈值。 注：E803 中根据配置的中断数量决定的优先级个数可以设 [7:6],[7:5] 或 [7:4] 位来表征优先级阈值。

5.3.2.11 Tspend 中断使能设置寄存器（VIC_TSPEND）

VIC_TSPEND 用于使能 tspend 中断，并且反馈 tspend 中断的使能状态。图 5.19 描述了 VIC_TSPEND 的位分布，表 5.19 描述了 VIC_TSPEND 的位定义。

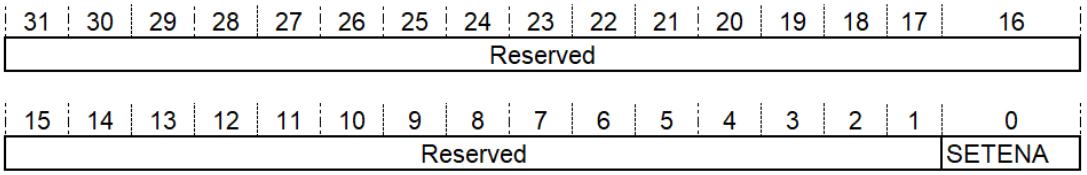


图 5.19: tspend 中断使能设置寄存器

表 5.19: tspend 中断使能设置寄存器域定义

位	名称	描述
0:0	SETENA	设置使能，读取 tspend 中断的使能状态： **读操作 0** 对应 tspend 中断未使能。 **1** 对应 tspend 中断已使能。 **写操作 0** 无效。 **1** 使能 tspend 中断。

5.3.2.12 Tspend 中断响应状态寄存器（VIC_TSABR）

VIC_TSABR 用于指示 tspend 中断当前的 Active 状态，是一个供软件查询的寄存器，另外，软件可在初始化 VIC 时，将所有 tspend 中断的 Active 状态清 0。图 5.20 描述了 VIC_TSABR 的位分布，表 5.20 描述了 VIC_IABR 的位定义。

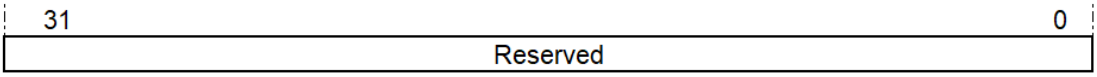


图 5.20: tspend 中断响应状态寄存器

表 5.20: tspend 中断响应状态寄存器域描述

位	名称	描述
0	Active	查询位，指示 tspend 中断是否已经被 CPU 响应但还没处理完。该寄存器只有最低位有效。 **读操作 0** 没有被 CPU 响应 **1** 已经被 CPU 响应但还没处理完 **写操作 0** 清除 tspend 中断的 Active 状态 （软件不可对该寄存器写 1，否则会导致不可预期的错误）

5.3.2.13 Tspend 中断优先级设置寄存器（VIC_TSPR）

VIC_TSPR 提供 tspend 中断的优先级设置，tspend 中断的优先级需要设置为最低。图 5.21 描述了 VIC_TSPR 的位分布，表 5.21 描述了 VIC_TSPR 的位定义。

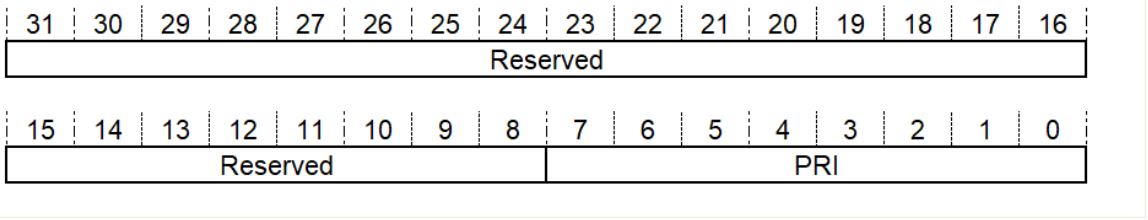


图 5.21: tspend 中断优先级设置寄存器

表 5.21: tspend 中断优先级设置寄存器域描述

位		名称	描述
2 比特 4 个优先级 （中断数小于等于 32 个）	31:8	-	保留
	7:6	PRI	建议设置成最低优先级 2’b11
	5:0	-	保留
3 比特 8 个优先级 （中断数量为 64 个）	31:8	-	保留
	7:5	PRI	建议设置成最低优先级 3’b111
	5:0	-	保留
4 比特 16 个优先级 （中断数量为 128 个）	31:8	-	保留
	7:4	PRI	建议设置成最低优先级 4’b1111
	5:0	-	保留

5.3.3 中断处理机制

矢量中断控制器支持电平中断和脉冲中断。

对于电平中断，矢量中断控制器采样到中断有效信号的高电平后设置对应中断进入等待状态，然后请求 CPU 响应。电平中断要求中断服务程序中清除外设的中断源有效信号，否则当中断退出时中断控制器会重新向 CPU 发起中断请求。外设可以根据这一特点，常置中断信号直到不再需要中断处理程序处理。

对于脉冲中断，又称为边沿中断，矢量中断控制器采样中断有效信号的上升沿，然后设置对应中断进入等待状态，随后向 CPU 发起中断请求。为了确保矢量中断控制器检测到脉冲中断，外围需要将中断信号至少保持一个 CPU 时钟周期。在 CPU 响应该脉冲中断请求前，若脉冲中断源向矢量中断控制器发起多次中断请求，矢量中断控制器只会记录一次中断请求；在 CPU 响应该脉冲中断请求后，若脉冲中断源再次向中断控制器发起请求，矢量中断控制器会再次触发对应中断进入等待状态，该处于等待状态的中断请求在上次中断退出后才能够再次被 CPU 响应。

另外，矢量中断控制器支持软件中断。软件可通过设置中断设置等待寄存器（VIC_ISPR）置高相应的中断等待状态位，触发该中断进入等待状态，向 CPU 发送中断请求。

当处理器响应中断请求后，矢量中断控制器会自动清除相应中断的等待状态位。也可以通过设置中断清除等待寄存器（VIC_ICPR）清除相应中断的等待状态位。对于电平中断，如果中断有效信号持续为高，则无法通过设置 VIC_ICPR 寄存器清除等待状态位。

5.3.3.1 中断状态位

VIC 为每个中断源提供 2 比特状态位，分别为：

- Pending：表征该中断处于等待状态，即该中断请求等待 CPU 进行响应。
  - 0：表征中断尚未处于等待状态；
  - 1：表征中断已经处于等待状态。
- Active：表征该中断请求已经被 CPU 响应，但尚未处理完成。
  - 0：表征该中断请求尚未被 CPU 响应；
  - 1：表征该中断请求已经被 CPU 响应，但尚未处理完成。

Pending 位的设置条件：

1) 电平中断源，中断源有效信号置高，且 Active 为低或者 CPU 正在退出该中断服务程序；
2) 脉冲中断源，上升沿有效；
3) 软件设置 ISPR。

Pending 位的清除条件：

1) CPU 响应该中断请求；
2) 软件清除 ICPR。

Active 位的设置条件：

1) CPU 响应该中断请求。

Active 位清除条件：

1) CPU 退出该中断服务程序。

注意： 对于电平中断源，由于需要中断服务程序中将电平中断源请求拉低，因此只需要在 Active 为低或者退出中断服务程序时采样中断源有效信号并设置 Pending 位；对于脉冲中断源，由于请求信号会自动拉低，因此需要实时采样脉冲中断信号的上升沿并设置 Pending 位。

5.3.3.2 中断优先级

VIC 通过中断优先级设置寄存器（IPR0~IPR31）为每个中断源提供优先级设置，中断个数小于等于 32 个时，硬件可以提供 4 个优先级；中断个数为 64 个时，硬件可以提供 8 个优先级；中断个数为 128 个时，硬件可以提供 16 个优先级。优先级号越低，优先级越高，具体参考 3.2.8 节。当软件没有设置优先级寄存器时，所有中断源优先级默认为最高优先级——0.

当多个中断处于 Pending 状态时，VIC 根据各个中断的优先级仲裁出优先级最高的中断请求提交给 CPU 处理。例如，两个中断请求 IRQ0 和 IRQ1 同时处于 Pending 状态，如果 IRQ1 的优先级号小于 IRQ0，即 IRQ1 的优先级高于 IRQ0，因此 IRQ0 先提交给 CPU 处理。

当多个 Pending 的中断拥有相同的优先级号时，根据中断号决定中断提交的顺序，中断号小的优先提交给 CPU 处理。例如，两个中断请求 IRQ0 和 IRQ1 的中断优先级相同，IRQ0 的中断源号小于 IRQ1，因此 IRQ0 优先提交给 CPU 处理。

5.3.3.3 中断向量号

中断向量号，是中断请求在异常向量表的位置编号。下图给出了 E803 的异常向量表，开始的 0~30 号向量是用作处理器内部识别的向量；31 号向量保留；从 32 开始的向量号预留给外部中断请求，且每个中断源对应一个中断向量号。

表 5.22: 中断向量号描述

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 0 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。(自动向量)
11-15	02C - 03C	保留。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3)。
20-21	050-054	保留。
22	058	Tspend 中断
23-29	05C-074	保留。
30	078	保留。
31	07C	保留
32	080	IRQ0。
33	084	IRQ1.
……	……	……
32+n	0x80+4n	IRQn

5.3.3.4 中断处理过程

中断处理过程可分以下几个步骤进行:

- 中断源请求的同步: 外部设备产生中断源请求，系统完成异步中断源请求到 CPUCLK 时钟域的同步操作，置高 pad_vic_int_vld;
- 中断源请求的采样: VIC 根据中断源的类型对中断源请求进行采样; 当采样到有效的中断请求时，设置 Pending 状态位，触发对应中断进入等待状态;
- 中断请求发起: 在所有处于等待状态的中断中，经过优先级仲裁向 CPU 发起中断请求;
- 中断响应: CPU 在指令退休时响应中断，返回中断响应信号给 VIC，同时将 PSR 和 PC 更新到 EPSR 和 EPC，并将被响应中断的中断向量号更新 PSR.VEC，清除 PSR.EE，最后取异常入口地址; VIC 根据中断响应信号清除相应中断的 Pending 状态位，并设置其 Active 状态位;
- 中断现场保存: 首先保存中断控制寄存器现场 {EPSR，EPC}，打开 PSR.EE 和 PSR.IE，以使能中断嵌套; 随后保存通用寄存器现场;

- 中断事务：CPU 开始处理中断事务，对于电平中断，需要将中断源信号清除，否则在中断退出时会重入该中断；
- 中断现场恢复和中断退出：首先恢复通用寄存器现场；随后恢复中断控制寄存器现场 {EPSR，EPC}，将 EPC 和 EPSR 恢复到 PC 和 PSR，退出中断服务程序；VIC 接收中断退出信号，清除 Active 状态位。

中断现场的保存可通过在中断服务程序的起始处执行 NIE 和 IPUSH 指令完成，中断现场的恢复和退出可通过在中断服务程序的结尾处执行 IPOP 和 NIR 指令完成。

中断源请求的同步由系统完成，VIC 内部不实现。下图给出了 pad_vic_int_vld 信号同步的一个示例，中断源请求信号 irq_n 经过两级 CPUCLK 的寄存器同步到 CPU 时钟域上。另外，中断源的类型配置信号 pad_vic_int_cfg 对于给定的中断源是一个固定值，0 表示电平中断源，1 表示脉冲中断源，因此不需要同步。

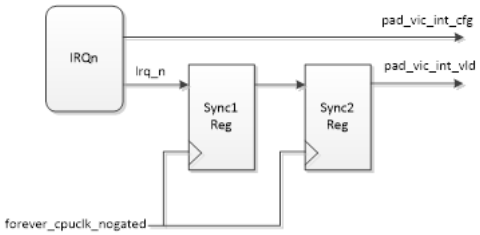


图 5.22: 中断源请求同步电路示例

5.3.3.5 中断嵌套

VIC 支持中断嵌套功能，在中断处理的过程中允许更高优先级的中断抢占，从而提高中断响应实时性。当存在多级中断嵌套时，特定场景下可能需要改变被抢占中断的优先级，例如低优先级中断响应时间达到最大限制。此时软件可通过设置中断优先级阈值寄存器，提高中断抢占的优先级条件，使被抢占的低优先级中断能得到及时响应。

5.3.3.6 中断嵌套优先级条件

中断抢占的优先级条件可分为两种，如下所示：

- 当中断优先级阈值未使能时，中断抢占的优先级必须高于当前 CPU 正在处理的中断的优先级；同级优先级不能进行抢占；
- 当中断优先级阈值使能时，中断抢占的优先级不仅要高于当前 CPU 正在处理的中断优先级，而且要高于中断优先级阈值寄存器设置的阈值。

VIC 支持中断优先级的动态调整，当被抢占中断的优先级需要调高或者调低时，在设置中断优先级设置寄存器的同时设置中断优先级阈值寄存器。

下图给出了中断抢占的示例。中断优先级设置为：IRQ0<IRQ1<IRQ2<IRQ3；中断源请求产生的顺序为：IRQ0>IRQ1>IRQ2>IRQ3。CPU 首先响应了 IRQ0，在 IRQ0 中断服务程序执行的过程中，来了更高

优先级的 IRQ1，因此 IRQ0 被抢占，CPU 开始执行 IRQ1 的中断服务程序。同样，IRQ2 对 IRQ1 进行了抢占，并设置了中断优先级阈值寄存器（IPTR.VECTTHRESHOLD=IRQ0，IPTR.PRIOTHRESHOLD=0，IPTR.EN=1）。当 IRQ3 到来时，尽管优先级高于 IRQ2，但 IRQ3 的优先级没有高于 IPTR 中设置的优先级阈值所表征的中断，因此 IRQ3 无法抢占 IRQ2。IRQ3 等到 IRQ0 的中断服务程序执行结束硬件自动清除 IPTR.EN 后，才得到 CPU 响应。

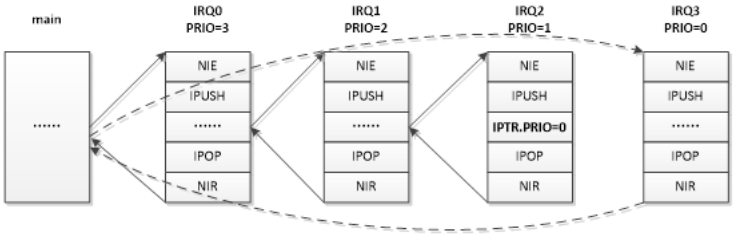


图 5.23: 中断嵌套优先级示例

5.3.3.7 中断嵌套时机条件

中断抢占在满足优先级的条件下，还需要判断当前中断响应的阶段。中断响应的过程如 3.3.4 所示，和嵌套相关的主要分为以下几个阶段：1）EPSR、EPC、PSR 的更新和异常入口地址的读取、2）NIE 指令、3）IPUSH 指令、4）中断事务、5）IPOP 指令、6）NIR 指令。

为了保证中断嵌套现场的保存和恢复，CPU 在以下阶段不能被中断打断：

- 中断响应之后更新 EPSR、EPC、PSR 和获取异常入口地址的过程中；
- NIE 指令执行过程中，且包括指令退休；
- PSR.IC 位关闭，IPUSH 和 IPOP 指令执行过程中，不包括指令退休；
- NIR 指令执行过程中，不包括指令退休。

CPU 在以下阶段可安全可靠地响应新的中断：

- 正常程序的执行过程中，在中断响应之前；
- IPUSH、IPOP 指令退休时；
- PSR.IC 位打开，IPUSH、IPOP 指令执行过程中；
- NIR 指令退休时；
- 中断事务处理的过程中。

在 PSR.IC 位打开时，IPUSH 和 IPOP 指令在执行过程中可响应中断，因此中断返回时需要重新执行该指令。对于 IPUSH 或 IPOP 指令在退休时响应中断的情况，在中断退出后直接执行 IPUSH/IPOP 的下一条指令，不需要重新执行该 IPUSH/IPOP 指令。NIR 指令执行过程中不可被中断打断，但在退休时可响应中断，如果中断打在 NIR 指令上，CPU 在 NIR 指令退休时，直接将 NIR 的返回地址压入堆栈，在中断退出时返回 NIR 的目标地址。

下图给出了 IRQ0/IRQ1/IRQ2/IRQ3 中断嵌套的过程。在 IRQ0 被 CPU 响应后，产生了更高优先级的 IRQ1，当 PSR.IC 打开时，在执行到 IPUSH 指令时响应 IRQ1。IRQ2 在 IRQ1 处理中断事务时产生，因此可立即被 CPU 响应。IRQ3 在 IRQ2 执行 NIR 指令时产生，在 NIR 指令退休时响应 IRQ3。当 IRQ3 处理完退出中断服务程序时，直接返回到 IRQ1 被 IRQ2 打断的点。当 IRQ1 返回 IRQ0 时，需要重新执行 IPUSH 指令。

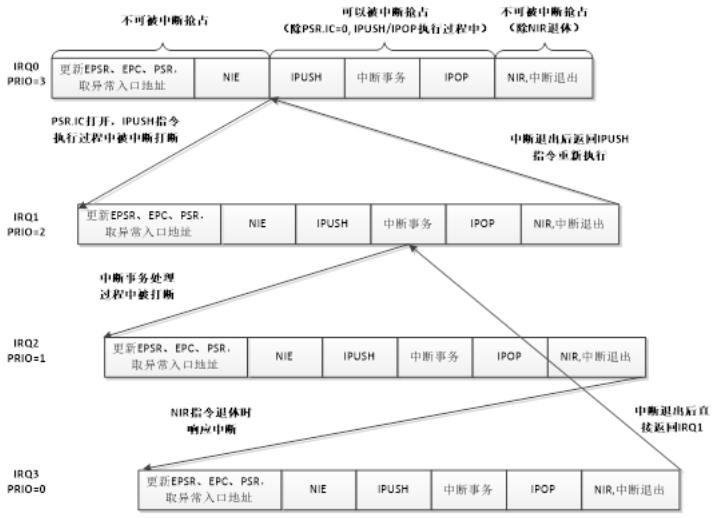


图 5.24: 中断嵌套时机条件示例

5.3.4 Tspend 中断

Tspend 中断控制器提供的，一个不占用外部中断向量号（32~159）的中断，Tspend 中断向量号为 22，tspend 中断功能和设置同其他中断基本相同，tspend 中断和其他中断的区别在于：

1. Tspend 中断，不需要外部设别产生中断源请求，无 pending 状态位，软件设置 VIC_TSPEND 使能时进入等待状态。
2. Tspend 中断和其他中断优先级相同时，不以中断号大小判断中断优先级，相同中断优先级时 Tspend 中断优先级为最低；等到没有其他同级别中断处于 pending 状态时，处理 tspend 中断。

5.3.5 操作步骤

为保证 VIC 能够生成预期优先级的中断请求，用户需要提前设置好中断优先级并使能响应中断，步骤如下：

- 首先设置 VIC_IPR0~31，给每个中断配置合适的优先级；
- 然后设置 VIC_ISER，使能相应的中断；

另外，VIC 支持软件设置 VIC_ISPR 产生中断请求，同样在 VIC_ISPR 设置之前必须按上述步骤配置中断优先级和中断使能位。

注意： CPU 响应 VIC 触发的中断请求之前需要使能 PSR.IE 和 PSR.EE，以开启 CPU 的中断响应使能，否则 CPU 无法响应中断。

VIC 采样外部中断源产生中断低功耗唤醒之前，必须先设置 VIC_IWER，置高相应的低功耗唤醒使能位，否则无法产生低功耗唤醒请求。

注意： 对于某个外部中断源请求，只需要相应中断的低功耗唤醒功能被使能（对应于 VIC_IWER），就能产生低功耗唤醒请求，而不依赖于该中断自身的中断使能（对应于 VIC_ISER）。

5.3.6 接口信号

矢量中断控制器的接口信号可分为三组，分别为：

矢量中断控制器与中断源的接口：VIC 接收并采样中断源的中断请求信号；

矢量中断控制器与 TCIP 的接口：TCIP 提供一组读写 VIC 寄存器的接口，完成 VIC 相关控制寄存器的读写，如使能中断、设置中断优先级、查询当前正在处理中断等；

矢量中断控制器与处理器核 E803 的接口：VIC 向 E803 发起中断请求和相应的中断向量号，E803 返回 VIC 中断响应和中断返回指示信号以及目前正在处理的中断向量号。

表 5.23: 矢量中断控制器接口信号

信号名	I/O	Reset	定义
紧耦合 IP 总线接口信号：			
tcipif_vic_sel	I	0	选中信号： 指示选中矢量中断控制器并进行数据传输。 1：选中 VIC 0：未选中 VIC
tcipif_vic_addr[15:0]	I	-	地址总线： 16 位地址总线（截取 32 位地址总线的低 16 位），指示访问地址。
tcipif_vic_write	I	0	读写表示信号： 指示当前 TCIP 访问是读取数据还是写数据： 1：写访问； 0：读访问。
tcipif_vic_wdata[31:0]	I	-	写数据总线： 32 位写数据总线。

下页继续

表 5.23 – 续上页

信号名	I/O	Reset	定义
vic_tcipif_rdata[31:0]	O	-	读数据总线: 32 位读数据总线。
vic_tcipif_cmplt	O	0	传输完成指示信号: 有效时指示当前传输已完成。
处理器中断握手信号:			
vic_pad_int_b	O	1	中断请求信号: 低电平时表示进行普通中断申请。
vic_pad_intraw_b	O	1	中断唤醒请求信号: 低电平时表示进行低功耗唤醒申请
vic_pad_int_vec_b[7:0]	O	-	中断矢量序号信号: 提供 core 进行中断处理的向量号。
pad_vic_int_ack	I	0	中断响应信号: 指示 CPU 响应了当前的中断请求
pad_vic_int_exit	I	0	中断服务程序退出信号: 指示 CPU 退出中断服务程序
pad_vic_int_vec[7:0]	I	-	指示 CPU 正在处理的中断向量号
pad_vic_ack_vec[7:0]	I	-	指示 CPU 响应的中断向量号
中断源信号:			
pad_vic_int_cfg[127:0]	I	-	中断源类型配置信号: 0：电平中断源 1：脉冲中断源
pad_vic_int_vld[127:0]	I	0	中断有效信号: 高电平有效。
时钟信号:			
forever_cpuclk	I	-	VIC 的工作时钟: 矢量中断控制器中和低功耗唤醒无关的寄存器均工作于该时钟,低功耗模式下可以关闭该时钟信号。
forever_cpuclk_nogated	I	-	VIC 中断采样电路的时钟: 矢量中断控制器中和低功耗唤醒相关的寄存器均工作于该时钟。在低功耗模式下不可以关闭该时钟信号。

下页继续

表 5.23 – 续上页

信号名	I/O	Reset	定义
复位信号：			
cpurst_b	I	-	矢量中断控制器复位信号： 低电平时，初始化矢量中断控制器内部。矢量中断控制器采用异步复位方式。
其它信号：			
pad_yy_gate_clk_en_b	I	-	门控时钟使能信号： 只有当这个信号有效时，VIC 的内部模块的门控时钟才能有效。不用该信号时，需要接 1。
pad_yy_test_mode	I	-	进入测试模式： 使 VIC 进入测试模式，此时 VIC 时钟为测试时钟（pad_had_jtg_tclk）。只有 VIC 进入测试模式，并且处理器输入信号 pad_ yy_scan_enable 有效时，才可以通过扫描链进行测试。不用该信号时，需要接 0。

当 CPU 内部没有集成矢量中断控制器时，由外部中断控制器对中断源进行仲裁，产生中断请求发送给 CPU。CPU 对外部输入的中断请求信号进行同步并传递给核内处理。另外，CPU 也需要对外部 IP 产生低功耗唤醒信号进行同步。

表 5.24: 没有配置矢量中断控制器的接口信号

信号名	I/O	Reset	定义
紧耦合 IP 总线接口信号：			
pad_cpu_int_b	I	1	中断请求信号： 低电平时表示进行普通中断申请。
pad_cpu_intraw_b	I	1	低功耗唤醒请求信号： 低电平时表示进行低功耗唤醒申请。
pad_cpu_int_vec_b[7:0]	I	-	中断向量号： 指示当前中断请求对应中断向量号.
时钟信号：			
forever_cpuclk_nogated	I	-	VIC 同步电路时钟： 用于中断请求和低功耗唤醒信号的同步。

5.3.7 中断设置示例

```
//设置 psr 中的 ee 和 ie 位，cpu 响应中断
psrset ee，ie
//设置中断号为 32~35 的中断使能位
lrw r1，0x0
bseti r1, 0x0 //设置 32 号中断的使能位
bseti r1, 0x1 //设置 33 号中断的使能位
bseti r1, 0x2 //设置 34 号中断的使能位
bseti r1, 0x3 //设置 35 号中断的使能位
lrw r2, 0xe000e100 //中断使能寄存器 ISER 对应的地址
st.w r1，（r2, 0x0）//使能中断号为 32~35 的 4 个中断
//其他中断的使能同上，通过设置 ISER 寄存器的不同位使能相应的中断
//设置中断优先级寄存器 IPR0，设置 32~35 号中断的优先级
lrw r1，0x0
//设置 IPR0[7:6] 设置 32 号中断的优先级为 2’b11，最低优先级
//设置 IPR0[15:14] 设置 33 号中断的优先级为 2’b00，最高优先级
//设置 IPR0[23:22] 设置 34 号中断的优先级为 2’b10
//设置 IPR0[31:30] 设置 35 号中断的优先级为 2’b10
bseti r1, 0x6 //通过低 [7:6] 两位设置 32 号中断优先级
bseti r1, 0x7 //设置 32 号中断的优先级为最低，IPR0[7:6]=2’b11
bclri r1, 0x14 //通过 [15:14] 两位设置 33 号中断优先级
bclri r1, 0x15 //设置 33 号中断的优先级为最高，IPR0[15:14]=2’b00
bclri r1, 0x22 //通过 [23:22] 两位设置 34 号中断优先级
bseti r1, 0x23 //设置 34 号中断的优先级为 2，IPR0[23:22]=2’b10
bclri r1, 0x30 //通过 [31:30] 两位设置 35 号中断优先级
bseti r1, 0x31 //设置 35 号中断的优先级为 2，IPR0[31:30]=2’b10
//将设置的优先级写入 IPR0
lrw r2, 0xe000e400 //中断使能寄存器 IPR0 对应的地址
st.w r1，（r2, 0x0）//完成 32~35 号的中断优先级设置
//其他中断的优先级设置同上，通过设置对应的中断优先级寄存器 IPR1~IPR7 即可。
```



```
//设置中断优先级阈值寄存器 VIC_IPTR
lrw r1，0x2200 //设置中断优先级阈值寄存器的中断号, IPTR[16:8]=34
bseti r1, 0x0 //使能中断优先级阈值寄存器
//通过 IPTR[7:6] 两位设置能够抢占 34 号中断的中断优先级,
//设置能够抢占 34 号中断的先级必须优先级高于 2’b01,
//即只有优先级为 2’b00（最高）的中断可以抢占 34 号中断
bseti r1, 0x6
bclri r1, 0x7
//将设置的 34 号中断号对应的优先级阈值, 使能和中断号写入 IPTR
lrw r2, 0xe000ec04 //中断优先级阈值寄存器地址
st.w r1, （r2, 0x0）//完成 34 号中断的优先级阈值设置
```

5.4 高速缓存控制寄存器单元

5.4.1 简介

E803CPU 提供硬件可配置的高速缓存器（Cache）。Cache 控制寄存器单元（CRU）为 Cache 提供了一组设置寄存器，包括 Cache 的使能、缓存行的无效和高速缓存区的配置。

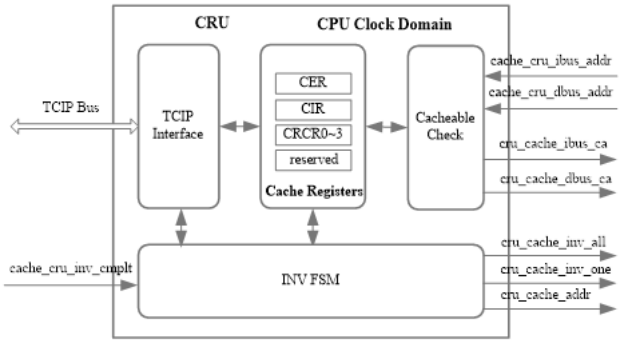


图 5.25: Cache 控制寄存器单元结构图

E803 提供的硬件可配置高速缓存，其主要特征如下：

- 高速缓存大小硬件可配置，支持 2K/4K/8KB/16KB/32KB/64KB。
- 4 路组相联，缓存行大小为 16 个字节；
- 每次访问的最大宽度为 4 字节，支持字节/半字/字访问；
- 物理地址索引，物理地址标记；
- 写策略同时支持写直模式和写回模式；

- 高速缓存采用先进先出（FIFO）的替换策略；
- 支持对整个高速缓存的无效和清除操作，支持对单条缓存行的无效和清除操作。

当内存访问不可高缓区域时，直接旁路片上高速缓存，通过总线访问片外存储器，加快不可高缓存区域的访问速度。当内存访问可高缓区域时，首先访问片上高速缓存，在高速缓存缺失的情况下再访问片外存储器（写直模式下的写访问在高速缓存命中时，也需要回写片外存储器）。

5.4.2 寄存器定义

CRU 提供一组 32-bit 的寄存器，各个寄存器地址空间如图表 表 5.25 所示。

表 5.25: Cache 控制寄存器定义

地址	名称	类型	初始值	描述
0xE000F000	CER	读/写	0	高速缓存使能寄存器
0xE000F004	CIR	读/写	0	高速缓存无效寄存器
0xE000F008	CRCR0	读/写	0	0 号可高缓区配置寄存器
0xE000F00C	CRCR1	读/写	0	1 号可高缓区配置寄存器
0xE000F010	CRCR2	读/写	0	2 号可高缓区配置寄存器
0xE000F014	CRCR3	读/写	0	3 号可高缓区配置寄存器
0xE000F018~0xE000FFF3	-	-	-	保留
0xE000FFF4	CPFCR	读/写	0	高速缓存性能分析控制寄存器
0xE000FFF8	CPFATR	读/写	0	高速缓存访问次数寄存器
0xE000FFFC	CPFMTR	读/写	0	高速缓存缺失次数寄存器

5.4.2.1 高速缓存使能寄存器（CER）

高速缓存使能寄存器用于配置高速缓存（Cache）使能以及高速缓存的工作模式。

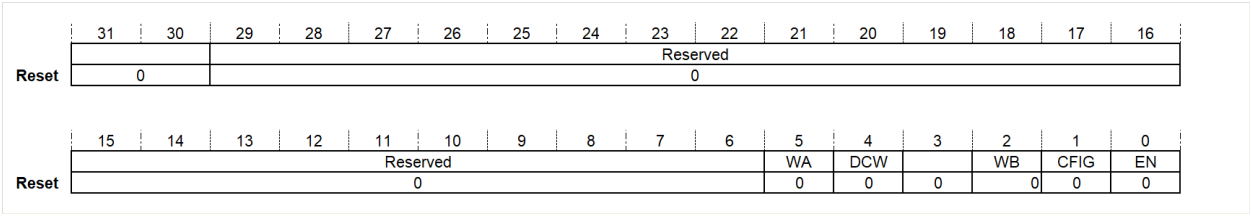


图 5.26: 高速缓存使能寄存器

WA 高速缓存写分配有效设置位：

- 0：高速缓存为 write non-allocate 模式；
- 1：高速缓存为 write allocate 模式。

- write allocate 模式只能在写回模式下使用，否则结果不可预知

DCW-Cache 可写属性配置位：

- 0：高速缓存不可写；
- 1：高速缓存可写。

WB-Cache 写回模式配置位：

- 0：高速缓存为写直模式；
- 1：高速缓存为写回模式。

CFIG-Cache 属性配置位：

- 0：指令与数据高速缓存；
- 1：指令高速缓存。

EN-Cache 使能位：

- 当 EN 为 0 时，高速缓存处于关闭状态。
- 当 EN 为 1 时，高速缓存处于工作状态。

5.4.2.2 高速缓存无效寄存器 (CIR)

高速缓存无效寄存器用于控制高速缓存的清除和无效，可支持对特定高速缓存行操作和对整个高速缓存操作两种方式。

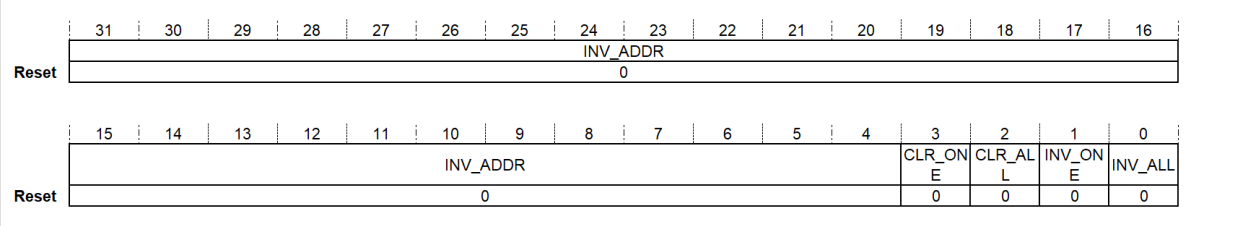


图 5.27: 高速缓存无效寄存器

INV_ADDR-缓存行地址：

- 表征需要被无效的缓存行地址。

CLR_ONE-单条缓存行清除设置位：

- 当 CLR_ONE 为 1 时，选中的缓存行若标记为脏，将被回写到片外存储器中。

CLR_ALL-整个高速缓存清除设置位：

- 当 CLR_ALL 为 1 时，标记为脏的所有缓存行，将被回写到片外存储器中。

INV_ONE-单条缓存行无效设置位：

- 当 INV_ONE 为 1 时，无效选中的缓存行。

INV_ALL-整个高速缓存无效设置位：

- 当 INV_ALL 为 1 时，无效高速缓存中所有缓存行。

> **注意：** 任何按行操作不可与任何全 cache 操作同时进行。

举例说明：往 CIR 中写 32’b1111，INV_ONE 和 CLR_ONE 操作将被屏蔽。

5.4.2.3 可高缓区配置寄存器 0~3（CRCR）

可高缓区配置寄存器用于配置可高缓（Cacheable）的地址空间范围，提供四个可高缓区配置寄存器，用来配置四块不同的可高缓区域。

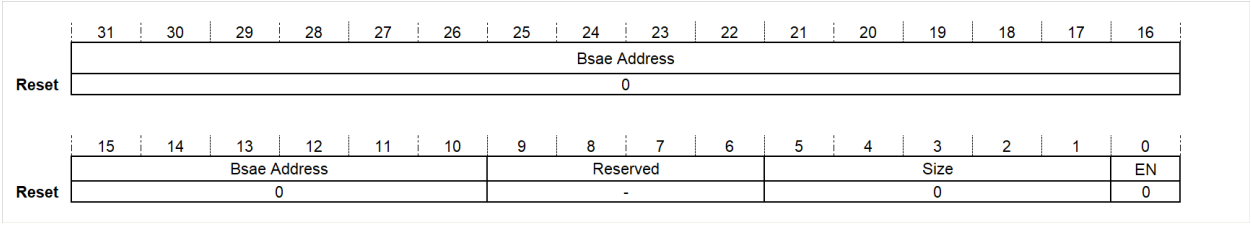


图 5.28: 可高缓区配置寄存器

Base Address-可高缓区基地址：

- 可高缓区大小 4KB 到 4GB 可配置，该域指出了可高缓区地址的高位，例如设置页面大小为 8MB，CRCR [22:12] 必须为 0，可高速缓存区大小对基地址的要求见图表 5-29。

Size-可高缓区大小：

- 可高缓区大小可以通过公式：可高缓区大小= 2^（Size+1）计算得到。因此 Size 便从 01001 到 11111，其它一些值都会造成不可预测的结果。

表 5.26: 可高缓区大小配置和其对基址要求

Size	高速缓存区大小	对基地址的要求
00000—01010	保留	-
01011	4KB	没有要求
01100	8KB	CRCR.bit[12]=0
01101	16KB	CRCR.bit[13:12] =0
01110	32KB	CRCR.bit[14:12] =0
01111	64KB	CRCR.bit[15:12] =0
10000	128KB	CRCR.bit[16:12] =0
10001	256KB	CRCR.bit[17:12] =0
10010	512KB	CRCR.bit[18:12]=0
10011	1MB	CRCR.bit[19:12]=0
10100	2MB	CRCR.bit[20:12]=0
10101	4MB	CRCR.bit[21:12]=0
10110	8MB	CRCR.bit[22:12]=0
10111	16MB	CRCR.bit[23:12]=0
11000	32MB	CRCR.bit[24:12]=0
11001	64MB	CRCR.bit[25:12]=0
11010	128MB	CRCR.bit[26:12]=0
11011	256MB	CRCR.bit[27:12]=0
11100	512MB	CRCR.bit[28:12]=0
11101	1GB	CRCR.bit[29:12]=0
11110	2GB	CRCR.bit[30:12]=0
11111	4GB	CRCR.bit[31:12]=0

EN-可高缓区有效位：

- 当 EN 为 0 时，可高缓区无效；
- 当 EN 为 1 时，可高缓区有效。

5.4.2.4 高速缓存性能分析控制寄存器 (CPFCR)

高速缓存性能分析控制寄存器用于使能高速缓存性能分析功能（Cache Profiling）和重置性能分析相关计数器。

PFRST-Cache profiling 复位位：

- 当 PERST 为 1 时，将 CPFATR/CPFMTR 复位为 0。

PFEN-Cache profiling 使能位：

- 当 PFEN 为 0 时，高速缓存性能分析功能处于关闭状态。

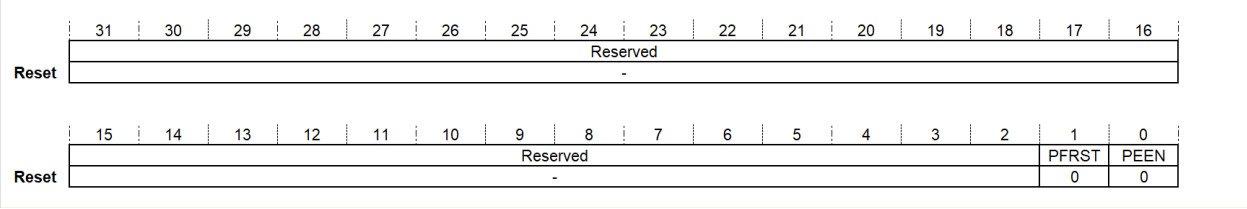


图 5.29: 高速缓存性能分析控制寄存器

- 当 PFEN 为 1 时，高速缓存性能分析功能处于工作状态。

5.4.2.5 高速缓存访问次数寄存器 (CPFATR)

高速缓存访问次数寄存器用于记录访问高速缓存的次数。

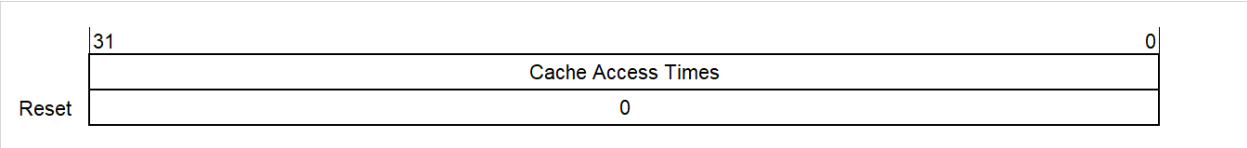


图 5.30: 高速缓存访问次数寄存器

Cache Access Times –高速缓存访问次数:

- 记录高速缓存被访问次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存访问将使寄存器数值加 1。

5.4.2.6 高速缓存缺失次数寄存器 (CPFMTR)

高速缓存缺失次数寄存器用于记录访问高速缓存中缺失的次数。

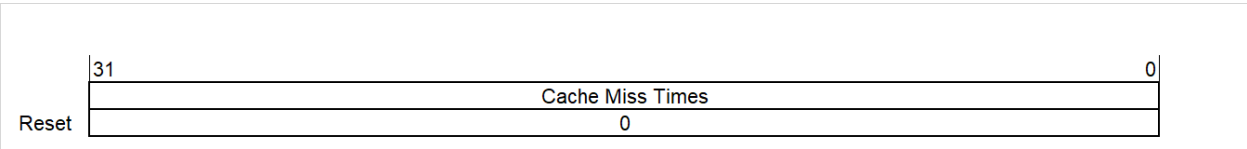


图 5.31: 高速缓存缺失次数寄存器

Cache Miss Times –访问高速缓存缺失次数:

- 记录访问高速缓存中的缺失次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存缺失将使寄存器数值加 1。

注意事项:

- 建议所有高速缓存配置操作都在其关闭情况下进行。

5.4.2.7 寄存器使用说明

CER 使用说明 CER 寄存器提供给用户以下功能：

- 配置写分配 write allocate

  通过对 CER[5] 置 1 使能 write allocate 属性，否则不支持 write allocate。

- 配置 cache 写属性

  通过对 CER[4] 置 1 使能 cahe 写属性，否则 cache 中的数据不能被写操作修改。

- 配置 cache write back

  通过对 CER[2] 置 1 使能 cache write back 属性，否则 cache 工作在 write through 模式。

- 配置 Cache 属性

  通过设置 CER[1] 可以配置 Cache 的属性，CER[1] 置 1 为纯指令 Cache，CER[1] 置 0 为指令和数据 Cache。

- 使能 Cache

  通过对 CER[0] 置 1 使能 Cache，置 0 关闭 Cache。

  需要说明的是，E803 中，cache write allocate 属性针对数据 store 操作，使能 write allocate 属性需要 cache 配置 write back 属性且 cache 需要设置成指令和数据 cache 且 cache 的写属性为开，否则结果不可预知。

Cache 只映射指令总线对应的地址空间，若 Cache 配置为指令 Cache，只有取指部件（IFU）发出的取指请求才通过 CRCR 判断是否可高缓；若 Cache 配置为数据和指令 Cache，则取指部件（IFU）发出的取指请求以及 LSU 部件发出的读写数据请求都需要通过 CRCR 判定是否可高缓。

CIR 使用说明 CIR 寄存器提供给用户以下功能：

- 清除单条缓存行，clr_one
- 清除整个 Cache，clr_all
- 无效单条缓存，inv_one
- 无效整个 Cache，inv_all

需要说明的是，在设置无效单条缓存行 CIR[1] 或者清除单条缓存行 CIR[3] 的同时需要指定该缓存行的地址 CIR[31:4]，否则将无效或者清除 CIR[31:4] 原先指向的缓存行。

- 当设置 clr_all/inv_all 整个 Cache 时，设置 clr_one/inv_one 单个缓存行无效。
- 当同时设置 clr_one 和 inv_one 时，cache 先执行 clr_one 操作再执行 inv_one 操作。
- 当同时设置 clr_all 和 inv_all 时，cache 优先先执行 clr_all 操作再执行 clr_all 操作。

当清除和无效 cache 操作完成后，硬件自动将 CIR[3:0] 清除，通过软件查询这四位，读到的值恒为 0。使能 Cache 的同时或者之前需要无效整个 Cache，否则可能导致不可预测的错误。

CRCR 使用说明 CRCR 寄存器定义了可高缓区基地址、可高缓区大小和可高缓区有效位供用户配置内存地址的可高缓属性。CRCR 寄存器的个数与用户硬件配置的可高缓区数目一致，最多可配置 4 个可高缓区。当硬件配置了多个可高缓区时，各个高缓区地址不可重叠，因此 CRCR 寄存器的配置不存在优先级，用户可根据需求灵活配置。一旦可高缓区地址配置重叠将导致不可预测的错误。用户在配置 CRCR 寄存器后，需要打开 Cache 使能位 CER[0]，否则尽管 CRCR 寄存器指定了可高缓的内存区，它的访问属性也是不可高缓的。

5.4.3 操作步骤

Cache 在 CPU reset 后默认是关闭的，因此在使用 Cache 前需要的准备工作如下：

- 首先设置 CRCR0~3，配置可高缓区；
- 然后设置 CIR[0] ，无效整个 Cache；
- 最后设置 CER[0]，使能 Cache，同时设置 CER[5:1]，配置 Cache 各种属性。

第六章　指令集

6.1　概述

E803 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。E803 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。

6.2　32 位指令

本章主要介绍 E803 实现的 XuanTie V2 的 32 位指令集。

6.2.1　32 位指令功能分类

E803 的 32 位指令集按照指令实现的功能划分，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

6.2.1.1　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：

表 6.1: 32 位加减法指令列表

指令名称	指令描述
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

逻辑操作指令:

表 6.2: 32 位逻辑操作指令列表

指令名称	指令描述
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

移位指令:

表 6.3: 32 位移位指令列表

指令名称	指令描述
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

比较指令:

表 6.4: 32 位比较指令列表

指令名称	指令描述
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

数据传输指令:

表 6.5: 32 位数据传输指令列表

指令名称	指令描述
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
MVCV32	C 位取反传送指令
MVC32	C 位传送指令
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

比特操作指令：

表 6.6: 32 位比特操作指令列表

指令名称	指令描述
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

提取插入指令：

表 6.7: 32 位提取插入指令列表

指令名称	指令描述
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

乘除法指令:

表 6.8: 32 位乘除法指令列表

指令名称	指令描述
MULT32	乘法指令
MULLL.S16 (MULSH32)*	16 位有符号乘法指令
MUL.(U/S)32	32 位 (无/有) 符号乘法指令
MULA.32.L	32 位有符号乘累加取低 32 位指令
MULA.(U/S)32	32 位 (无/有) 符号乘累加指令
MULALL.S16.S	带饱和操作的 16 位有符号低半字乘累加指令
DIVU32	无符号除法指令
DIVS32	有符号除法指令

* 历史版本中的指令 MULSH32 将由 MULLL.S16 替代，功能不变

杂类运算指令:

表 6.9: 32 位杂类运算指令列表

指令名称	指令描述
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

6.2.1.2 分支跳转指令

分支跳转指令可以进一步分为:

分支指令:

表 6.10: 32 位分支指令列表

指令名称	指令描述
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
BNEZAD32	寄存器自减大于零分支指令
BHZ32	寄存器大于零分支指令
BHSZ32	寄存器大于等于零分支指令
BLZ32	寄存器小于零分支指令
BLSZ32	寄存器小于等于零分支指令

跳转指令:

表 6.11: 32 位跳转指令列表

指令名称	指令描述
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

6.2.1.3 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：

表 6.12: 32 位立即数偏移存取指令列表

指令名称	指令描述
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
ST32.B	字节存储指令
ST32.H	半字存储指令
ST32.W	字存储指令

向量寄存器偏移存取指令：

表 6.13: 32 位向量寄存器偏移存取指令列表

指令名称	指令描述
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

多寄存器存取指令：

表 6.14: 32 位多寄存器存取指令列表

指令名称	指令描述
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

符号存取指令：

表 6.15: 32 位符号存取指令列表

指令名称	指令描述
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

6.2.1.4 特权指令

特权指令可以进一步分为：

控制寄存器操作指令：

表 6.16: 32 位控制寄存器操作指令列表

指令名称	指令描述
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

低功耗指令：

表 6.17: 32 位低功耗指令列表

指令名称	指令描述
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

异常返回指令：

表 6.18: 32 位异常返回指令列表

指令名称	指令描述
RTE32	异常和普通中断返回指令

6.2.1.5 特殊功能指令

特殊功能指令具体包括:

表 6.19: 32 位特殊功能指令列表

指令名称	指令描述
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

6.3 16 位指令

本章主要介绍 E803 的 16 位指令集。

6.3.1 16 位指令功能分类

E803 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

6.3.1.1 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:

表 6.20: 16 位加减法指令列表

指令名称	指令描述
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

逻辑操作指令:

表 6.21: 16 位逻辑操作指令列表

指令名称	指令描述
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

移位指令:

表 6.22: 16 位移位指令列表

指令名称	指令描述
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

比较指令:

表 6.23: 16 位比较指令列表

指令名称	指令描述
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

数据传输指令:

表 6.24: 16 位数据传输指令列表

指令名称	指令描述
MOV16	数据传送指令
MOVI16	立即数数据传送指令
MVCV16	C 位传送指令
LRW16	存储器读入指令

比特操作指令：

表 6.25: 16 位比特操作指令列表

指令名称	指令描述
BCLRI16	立即数位清零指令
BSETI16	立即数位置位指令
BTSTI16	立即数位测试指令

提取插入指令：

表 6.26: 16 位提取插入指令列表

指令名称	指令描述
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

乘除法指令：

表 6.27: 16 位乘法指令列表

指令名称	指令描述
MULT16	乘法指令

6.3.1.2 分支跳转指令

分支跳转指令可以进一步分为：

分支指令：

表 6.28: 16 位分支指令列表

指令名称	指令描述
BT16	C 为 1 分支指令
BF16	C 为 0 分支指令

跳转指令:

表 6.29: 16 位跳转指令列表

指令名称	指令描述
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

6.3.1.3 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:

表 6.30: 16 位立即数偏移存取指令列表

指令名称	指令描述
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

多寄存器存取指令

表 6.31: 16 位多寄存器存取指令列表

指令名称	指令描述
POP16	出栈指令
PUSH16	压栈指令
IPOP16	中断出栈指令
IPUSH16	中断压栈指令
NIE16	中断嵌套使能指令
NIR16	中断嵌套返回指令

注解： NIE16 和 NIR16 需要在超级用户模式下执行。

6.4 指令集列表

E803 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。

列出了 E803 指令集中所有 16 位和 32 位指令。

表 6.32: E803 的指令集

汇编指令	32 位	16 位	汇编格式	指令描述
ABS	√	×	ABS32 RZ, RX	绝对值指令
ADDC	√	√	ADDC32 RZ, RX, RY ADDC16 RZ, RX	无符号带进位加法指令
ADDI	√	√	ADDI32 RZ, RX, OIMM12 ADDI16 RZ, OIMM8	无符号立即数加法指令
ADDU	√	√	ADDU32 RZ, RX, RY ADDU16 RZ, RX	无符号加法指令
AND	√	√	AND32 RZ, RX, RY AND16 RZ, RX	按位与指令
ANDI	√	×	ANDI32 RZ, RX, IMM12	立即数按位与指令
ANDN	√	√	ANDN32 RZ, RZ, RX ANDN16 RZ, RX	按位非与指令
ANDNI	√	×	ANDNI32 RZ, RX, IMM12	立即数按位非与指令
ASR	√	√	ASR32 RZ, RX, RY ASR16 RZ, RX	算术右移指令
ASRC	√	×	ASRC32 RZ, RX, OIMM5	立即数算术右移至 C 位指令
ASRI	√	√	ASRI32 RZ, RX, IMM5 ASRI16 RZ, RX, IMM5	立即数算术右移指令
BCLRI	√	√	BCLRI32 RZ, RX, IMM5 BCLRI16 RZ, IMM5	立即数位清零指令
BEZ	√	×	BEZ32 RX, LABEL	寄存器等于零分支指令
BF	√	√	BF32 LABEL BF16 LABEL	C 为 0 分支指令
BGENI	√	×	BGENI32 RZ, IMM5	立即数位产生指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
BGENR	√	×	BGENR32 RZ, RX	寄存器位产生指令
BHSZ	√	×	BHSZ32 RX, LABEL	寄存器大于等于零分支指令
BHZ	√	×	BHZ32 RX, LABEL	寄存器大于零分支指令
BKPT	×	√	BKPT16	断点指令
BLSZ	√	×	BLSZ32 RX, LABEL	寄存器小于等于零分支指令
BLZ	√	×	BLZ32 RX, LABEL	寄存器小于零分支指令
BMASKI	√	×	BMASKI32 RZ, OIMM5	立即数位屏蔽产生指令
BNEZ	√	×	BNEZ32 RX, LABEL	寄存器不等于零分支指令
BR	√	√	BR32 LABEL BR16 LABEL	无条件跳转指令
BREV	√	×	BREV32 RZ, RX	位倒序指令
BSETI	√	√	BSETI32 RZ, RX, IMM5; BSETI16 RZ, IMM5;	立即数位置位指令
BSR	√	×	BSR32 LABEL	跳转到子程序指令
BT	√	√	BT32 LABEL BT16 LABEL	C 为 1 分支指令
BTSTI	√	√	BTSTI32 RX, IMM5; BTSTI16 RX, IMM5;	立即数位测试指令
CLRF	√	×	CLRF32 RZ	C 为 0 清零指令
CLRT	√	×	CLRT32 RZ	C 为 1 清零指令
CMPHS	√	√	CMPHS32 RX, RY CMPHS16 RX, RY	无符号大于等于比较指令
CMPHSI	√	√	CMPHSI32 RX, OIMM16 CMPHSI16 RX, IMM5	立即数无符号大于等于比较指令
CMPLT	√	√	CMPLT32 RX, RY CMPLT16 RX, RY	有符号小于比较指令
CMPLTI	√	√	CMPLTI32 RX, OIMM16 CMPLTI16 RX, OIMM5	立即数有符号小于比较指令
CMPNE	√	√	CMPNE32 RX, RY CMPNE16 RX, RY	不等比较指令
CMPNEI	√	√	CMPNEI32 RX, IMM16 CMPNEI16 RX, IMM5	立即数不等比较指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
DECF	√	×	DECF32 RZ, RX, IMM5	C 为 0 立即数减法指令
DECGT	√	×	DECGT32 RZ, RX, IMM5	减法大于零置 C 位指令
DECLT	√	×	DECLT32 RZ, RX, IMM5	减法小于零置 C 位指令
DECNE	√	×	DECNE32 RZ, RX, IMM5	减法不等于零置 C 位指令
DECT	√	×	DECT32 RZ, RX, IMM5	C 为 1 立即数减法指令
DIVS	√	×	DIVS32 RZ, RX, RY	有符号除法指令
DIVU	√	×	DIVU32 RZ, RX, RY	无符号除法指令
DOZE	√	×	DOZE32	进入低功耗睡眠模式指令
FF0	√	×	FF0.32 RZ, RX	快速找 0 指令
FF1	√	×	FF1.32 RZ, RX	快速找 1 指令
GRS	√	×	GRS32 RZ, LABEL GRS32 RZ, IMM32	符号产生指令
IDLY	√	×	IDLY32 N	中断识别禁止指令
INCF	√	×	INCF32 RZ, RX, IMM5	C 为 0 立即数加法指令
INCT	√	×	INCT32 RZ, RX, IMM5	C 为 1 立即数加法指令
INS	√	×	INS32 RZ, RX, MSB, LSB	位插入指令
IPOP	×	√	IPOP16	中断出栈指令
IPUSH	×	√	IPUSH16	中断压栈指令
IXD	√	×	IXD32 RZ, RX, RY	索引双字指令
IXH	√	×	IXH32 RZ, RX, RY	索引半字指令
IXW	√	×	IXW32 RZ, RX, RY	索引字指令
JMP	√	√	JMP32 RX JMP16 RX	寄存器跳转指令
JMPI	√	×	JMPI32 LABEL	间接跳转指令
JSR	√	√	JSR32 RX JSR16 RX	寄存器跳转到子程序指令
JSRI	√	×	JSRI32 LABEL	间接跳转到子程序指令
LD.B	√	√	LD32.B RZ, (RX, DISP) LD16.B RZ, (RX, DISP)	无符号扩展字节加载指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
LD.BS	√	×	LD32.BS RZ, (RX, DISP)	有符号扩展字节加载指令
LD.H	√	√	LD32.H RZ, (RX, DISP) LD16.H RZ, (RX, DISP)	无符号扩展半字加载指令
LD.HS	√	×	LD32.HS RZ, (RX, DISP)	有符号扩展半字加载指令
LD.W	√	√	LD32.W RZ, (RX, DISP) LD16.W RZ, (RX, DISP)	字加载指令
LDM	√	×	LDM32 RY-RZ, (RX)	连续多字加载指令
LDQ	√	×	LDQ32 R4-R7, (RX)	连续四字加载指令
LDR.B	√	×	LDR32.B RZ, (RX, RY << 0) LDR32.B RZ, (RX, RY << 1) LDR32.B RZ, (RX, RY << 2) LDR32.B RZ, (RX, RY << 3)	寄存器移位寻址无符号扩展字节加载指令
LDR.BS	√	×	LDR32.BS RZ, (RX, RY << 0) LDR32.BS RZ, (RX, RY << 1) LDR32.BS RZ, (RX, RY << 2) LDR32.BS RZ, (RX, RY << 3)	寄存器移位寻址有符号扩展字节加载指令
LDR.H	√	×	LDR32.H RZ, (RX, RY << 0) LDR32.H RZ, (RX, RY << 1) LDR32.H RZ, (RX, RY << 2) LDR32.H RZ, (RX, RY << 3)	寄存器移位寻址无符号扩展半字加载指令
LDR.HS	√	×	LDR32.HS RZ, (RX, RY << 0) LDR32.HS RZ, (RX, RY << 1) LDR32.HS RZ, (RX, RY << 2) LDR32.HS RZ, (RX, RY << 3)	寄存器移位寻址有符号扩展半字加载指令
LDR.W	√	×	LDR32.W RZ, (RX, RY << 0) LDR32.W RZ, (RX, RY << 1) LDR32.W RZ, (RX, RY << 2) LDR32.W RZ, (RX, RY << 3)	寄存器移位寻址字加载指令
LRS.B	√	×	LRS32.B RZ, [LABEL]	字节符号加载指令
LRS.H	√	×	LRS32.H RZ, [LABEL]	半字符号加载指令
LRS.W	√	×	LRS32.W RZ, [LABEL]	字符号加载指令
LRW	√	√	LRW16 LABEL LRW16 IMM32 LRW32 LABEL LRW32 IMM32	存储器读入指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
LSL	√	√	LSL32 RZ, RX, RY LSL16 RZ, RY	逻辑左移指令
LSLC	√	×	LSLC32 RZ, RX, OIMM5	立即数逻辑左移至 C 位指令
LSLI	√	√	LSLI32 RZ, RX, IMM5 LSLI16 RZ, RX, IMM5	立即数逻辑左移指令
LSR	√	√	LSR32 RZ, RX, RY LSR16 RZ, RY	逻辑右移指令
LSRC	√	×	LSRC32 RZ, RX, OIMM5	立即数逻辑右移至 C 位指令
LSRI	√	√	LSRI32 RZ, RX, IMM5 LSRI16 RZ, RX, IMM5	立即数逻辑右移指令
MFCR	√	×	MFCR32 RZ, CR<X, SEL>	控制寄存器读传送指令
MOV	√	√	MOV16 RZ, RX	数据传送指令
MOVF	√	×	MOVF32 RZ, RX	C 为 0 数据传送指令
MOVI	√	√	MOVI32 RZ, IMM16 MOVI16 RZ, IMM8	立即数数据传送指令
MOVIH	√	×	MOVIH32 RZ, IMM16	立即数高位数据传送指令
MOVT	√	×	MOVT32 RZ, RX	C 为 1 数据传送指令
MTCR	√	×	MTCR32 RX, CR<Z, SEL>	控制寄存器写传送指令
MULA.32.L	√	×	MULA.32.L RZ，RX，RY	32 位有符号乘累加取低 32 位指令
MULALL.S16.S	√	×	MULALL.S16.S RZ, RX, RY	带饱和操作的 1 6 位有符号低半字乘累加指令
MULLL.S16 (MULSH)	√	√	MULLL.S16 RZ, RX, RY	16 位有符号乘法指令
MULT	√	√	MULT32 RZ, RX, RY MULT16 RZ, RX	乘法指令
MUL.(U/S)32	√	×	MUL.S32 RZ, RX, RY	32 位 (无/有) 符号乘法指令
MVC	√	×	MVC32 RZ	C 位传送指令
MVCV	√	√	MVCV32 RZ MVCV16 RZ	C 位取反传送指令
NIE	×	√	NIE	中断嵌套使能指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
NIR	×	√	NIR	中断嵌套返回指令
NOR	√	√	NOR32 RZ, RX, RY NOR16 RZ, RX	按位或非指令
NOT	√	√	NOT32 RZ, RX NOT16 RZ	按位非指令
OR	√	√	OR32 RZ, RX, RY OR16 RZ, RX	按位或指令
ORI	√	×	ORI32 RZ, RX, IMM16	立即数按位或指令
POP	√	√	POP32 REGLIST POP16 REGLIST	出栈指令
PSRCLR	√	×	PSRCLR32 EE, IE, FE, AF	PSR 位清零指令
PSRSET	√	×	PSRSET32 EE, IE, FE, AF	PSR 位置位指令
PUSH	√	√	PUSH32 REGLIST PUSH16 REGLIST	压栈指令
REVB	√	√	REVB32 RZ, RX REVB16 RZ, RX	字节倒序指令
REVH	√	√	REVH32 RZ, RX REVH16 RZ, RX	半字内字节倒序指令
ROTL	√	√	ROTL32 RZ, RX, RY ROTL16 RZ, RY	循环左移指令
ROTLI	√	×	ROTLI32 RZ, RX, IMM5	立即数循环左移指令
RSUB	√	×	RSUB32 RZ, RX, RY	反向减法指令
RTE	√	×	RTE32	异常和普通中断返回指令
RTS	√	√	RTS16	链接寄存器跳转指令
SCE	√	×	SCE32 COND	条件执行设置指令
SEXT	√	×	SEXT32 RZ, RX, MSB, LSB	位提取并有符号扩展指令
SEXTB	√	√	SEXTB32 RZ, RX SEXTB16 RZ, RX	字节提取并有符号扩展指令
SEXTH	√	√	SEXTH32 RZ, RX SEXTH16 RZ, RX	半字提取并有符号扩展指令
SRS.B	√	×	SRS32.B RZ, [LABEL]	字节符号存储指令
SRS.H	√	×	SRS32.H RZ, [LABEL]	半字符号存储指令
SRS.W	√	×	SRS32.W RZ, [LABEL]	字符号存储指令
ST.B	√	√	ST32.B RZ, (RX, DISP) ST16.B RZ, (RX, DISP)	字节存储指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
ST.H	√	√	ST32.H RZ, (RX, DISP) ST16.H RZ, (RX, DISP)	半字存储指令
ST.W	√	√	ST32.W RZ, (RX, DISP) ST16.W RZ, (RX, DISP)	字存储指令
STM	√	×	STM32 RY-RZ, (RX)	连续多字存储指令
STOP	√	×	STOP32	进入低功耗暂停模式指令
STQ	√	×	STQ32 R4-R7, (RX)	连续四字存储指令
STR.B	√	×	STR32.B RZ, (RX, RY << 0) STR32.B RZ, (RX, RY << 1) STR32.B RZ, (RX, RY << 2) STR32.B RZ, (RX, RY << 3)	寄存器移位寻址字节存储指令
STR.H	√	×	STR32.H RZ, (RX, RY << 0) STR32.H RZ, (RX, RY << 1) STR32.H RZ, (RX, RY << 2) STR32.H RZ, (RX, RY << 3)	寄存器移位寻址半字存储指令
STR.W	√	×	STR32.W RZ, (RX, RY << 0) STR32.W RZ, (RX, RY << 1) STR32.W RZ, (RX, RY << 2) STR32.W RZ, (RX, RY << 3)	寄存器移位寻址字存储指令
SUBC	√	√	SUBC32 RZ, RX, RY SUBC16 RZ, RY	无符号带借位减法指令
SUBI	√	√	SUBI32 RZ, RX, OIMM12 SUBI16 RZ, OIMM8	无符号立即数减法指令
SUBU	√	√	SUBU32 RZ, RX, RY SUBU16 RZ, RY	无符号减法指令
SYNC	√	×	SYNC32 IMM5	CPU 同步指令
TRAP	√	×	TRAP32 0 TRAP32 1 TRAP32 2 TRAP32 3	无条件操作系统陷阱指令
TST	√	√	TST32 RX, RY TST16 RX, RY	零测试指令
TSTNBZ	√	√	TSTNBZ32 RX TSTNBZ16 RX	无字节等于零寄存器测试指令
WAIT	√	×	WAIT32	进入低功耗等待模式指令

下页继续

表 6.32 – 续上页

汇编指令	32 位	16 位	汇编格式	指令描述
XOR	√	√	XOR32 RZ, RX, RY XOR16 RZ, RX	按位异或指令
XORI	√	×	XORI32 RZ, RX, IMM12	立即数按位异或指令
XSR	√	×	XSR32 RZ, RX, OIMM5	扩展右移指令
XTRB0	√	×	XTRB0.32 RZ, RX	提取字节 0 并无符号扩展指令
XTRB1	√	×	XTRB1.32 RZ, RX	提取字节 1 并无符号扩展指令
XTRB2	√	×	XTRB2.32 RZ, RX	提取字节 2 并无符号扩展指令
XTRB3	√	×	XTRB3.32 RZ, RX	提取字节 3 并无符号扩展指令
ZEXT	√	×	ZEXT32 RZ, RX, MSB, LSB	位提取并无符号扩展指令
ZEXTB	√	√	ZEXTB32 RZ, RX ZEXTB16 RZ, RX	字节提取并无符号扩展指令
ZEXTH	√	√	ZEXTH32 RZ, RX ZEXTH16 RZ, RX	半字提取并无符号扩展指令

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。

6.5 指令执行延迟

表 6.33: 指令执行延时表

指令	执行周期	备注
加减法指令		-
ADDU32/16	1	-
ADDC32/16	1	-
ADDI32/16	1	-
SUBU32/16	1	-
SUBC32/16	1	-
SUBI32/16	1	-
RSUB32	1	-
IXH32	1	-

下页继续

表 6.33 – 续上页

指令	执行周期	备注
IXW32	1	-
IXD32	1	-
INCF32	1	-
INCT32	1	-
DECF32	1	-
DECT32	1	-
DECGT32	1	-
DECLT32	1	-
DECNE32	1	-
逻辑操作指令		-
AND32/16	1	-
ANDI32/16	1	-
ANDN32	1	-
ANDNI32	1	-
OR32/16	1	-
ORI32	1	-
XOR32	1	-
XORI32	1	-
NOR32/16	1	-
NOT32/16	1	-
移位指令		-
LSL32/16	1	-
LSLI32/16	1	-
LSLC32	1	-
LSR32/16	1	-
LSRI32/16	1	-
LSRC32	1	-
ASR32/16	1	-
ASRI32/16	1	-
ASRC32	1	-
ROTL32/16	1	-
ROTLI32	1	-
XSR32	1	-
比较指令		-
CMPNE32/16	1	-
CMPNEI32/16	1	-
CMPHS32/16	1	-

下页继续

表 6.33 – 续上页

指令	执行周期	备注
CMPHSI32/16	1	-
CMPLT32/16	1	-
CMPLTI32/16	1	-
TST32/16	1	-
TSTNBZ32/16	1	-
数据传输指令		-
MOV32/16	1	-
MOVF32	1	-
MOVT32	1	-
MOVI32/16	1	-
MOVIH32	1	
MVCV32/16	1	-
MVC32	1	-
CLRF32	1	-
CLRT32	1	-
LRW32/16*	1	-
GRS32	1	-
比特操作指令		-
BCLRI32/16	1	-
BSETI32/16	1	-
BTSTI32/16	1	-
提取插入指令		-
ZEXT32/16	1	-
SEXT32/16	1	-
INS32	1	-
ZEXTB32	1	-
ZEXTH32	1	-
SEXTB32/16	1	-
SEXTH32/16	1	-
XTRB0.32	1	-
XTRB1.32	1	-
XTRB2.32	1	-
XTRB3.32	1	-
BREV32	1	-
REVB32/16	1	-
REVH32/16	1	-
乘除法指令		-

下页继续

表 6.33 – 续上页

指令	执行周期	备注
MULT32/16	1/3	基础配置下为 3 个周期
MULLLL.S16(MULSH)	1	-
MUL.(U/S)32	1/2/5	基础配置下为 5 个周期
MULA.32.L	1/2/4	基础配置下为 4 个周期
MULA.(U/S)32	1/2/5	基础配置下为 4 个周期
MULALL.S16.S	1-4	基础配置下为 2 个周期
DIVU32	5-36	-
DIVS32	5-36	-
杂类运算指令		-
ABS32	1	-
FF0. 32	1	-
FF1. 32	1	-
BMASKI32	1	-
BGENR32	1	-
BGENI32	1	-
分支指令		-
BT32/16	1	-
BF32/16	1	-
BEZ32	1	-
BNEZ32	1	-
BHZ32	1	-
BLSZ32	1	-
BLZ32	1	-
BHSZ32	1	-
跳转指令		-
BR32/16	1	-
BSR32	1	-
JMPI32*	3	拆分为 LRW 及 JMP 指令
JSRI32*	3	拆分为 LRW 及 JSR 指令
JMP32/16	1	-
JSR32/16	1	-
RTS32/16	1	-
立即数偏移存取指令		-
LD32/16.B*	1	-
LD32.BS*	1	-
LD32/16.H*	1	-
LD32.HS*	1	-

下页继续

表 6.33 – 续上页

指令	执行周期	备注
LD32/16.W*	1	-
ST32/16.B*	1	-
ST32/16.H*	1	-
ST32/16.W*	1	-
寄存器偏移存取指令		-
LDR32.B*	3	-
LDR32.BS*	3	-
LDR32.H*	3	-
LDR32.HS*	3	-
LDR32.W*	3	-
STR32.B*	3	-
STR32.H*	3	-
STR32.W*	3	-
多寄存器存取指令		-
LDQ32*	4	拆分为 4 条 LD.W 指令。
LDM32*	N	拆分为 N 条 LD.W 指令。
STQ32*	4	拆分为 4 条 ST.W 指令。
STM32*	N	拆分为 N 条 ST.W 指令。
PUSH32/16*	1+N	拆分为 SUB 和 N 条 ST.W 指令。
POP32/16*	2+N	拆分为 N 条 LD.W、ADD 和 RTS 指令。
IPOP16*	7	拆分为 7 条原子指令
IPUSH16*	7	拆分为 7 条原子指令
NIE16*	5	拆分为 5 条原子指令
NIR16*	5	拆分为 5 条原子指令
符号存取指令		
LRS32.B*	1	-
LRS32.H*	1	-
LRS32.W*	1	-
SRS32.B*	1	-
SRS32.H*	1	-
SRS32.W*	1	-
控制寄存器操作指令		
MFCR32	1	-
MTCR32	1	-
PSRSET32	1	-
PSRCLR32	1	-

下页继续

表 6.33 – 续上页

指令	执行周期	备注
低功耗指令		
WAIT32	1	-
DOZE32	1	-
STOP32	1	-
异常返回指令		-
RTE32	1	-
特殊功能指令		-
SYNC32	1	-
BKPT32	1	-
SCE32	1	-
IDLY32	1	-
TRAP32	1	-

注解： 表示内存存取相关指令，指令完成周期取决于总线延时或者高速缓存命中情况，表中数值所列为最快情况。

第七章　内存保护

7.1　内存保护单元简介

在受保护的系统中，主要有两类资源的访问需要被监视：存储器系统和外围设备。内存保护单元负责对存储器系统（包括外围设备）的访问合法性进行检查，其主要功能包括：

1）判定当前工作模式下 CPU 是否具备对内存地址的读/写访问权限。

2）获取该访问地址的附加属性，包括安全属性等。

内存保护单元支持 N 个表项（N 为 1-8 硬件可配置），可对 N 个区域的访问权限和属性进行设置。每个表项通过 0-7 的号码来标识和索引。内存保护单元的表项内容如下：

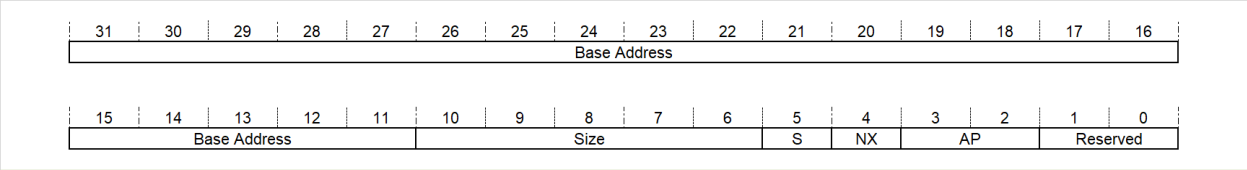


图 7.1: 内存保护单元表项

其中：

EN：表示该区域是否生效；

Base Address：表示该区域的起始地址；

Size：表示该区域的大小；

S：表示该区域的安全属性；

NX：表示区域取指的可执行性；

AP：表示该区域的访问权限；

具体的每个字段的属性参考 6.2 的系统控制寄存器部分。

7.2　相关系统控制寄存器

7.2.1 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保护使能，大小端模式，以及系统和处理器的时钟比。

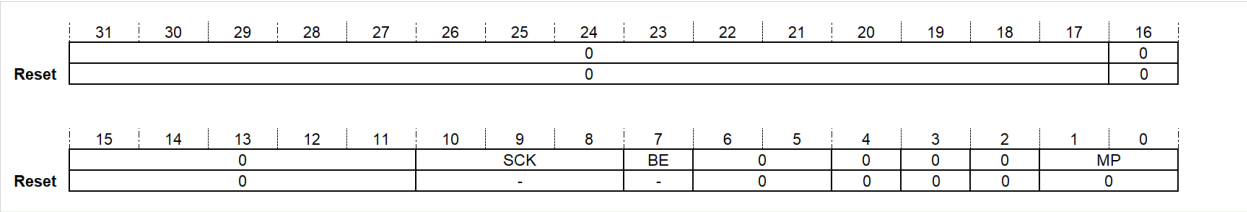


图 7.2: 高速缓存配置寄存器

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 power on reset 时被配置且不能在之后改变。

该域目前没有任何功能，只供软件查询。

BE-Endian 模式：

- 当 BE 为 0 时，小端；
- 当 BE 为 1 时，大端；

BE 在 power on reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

MP-内存保护设置位：

- MP 用来设置 MPU 是否有效，如下表：

表 7.1: E803 内存保护设置

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

7.2.2 可高缓和访问权限配置寄存器（CAPR，CR<19,0>）

CAPR 的各位如下图所示：

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。

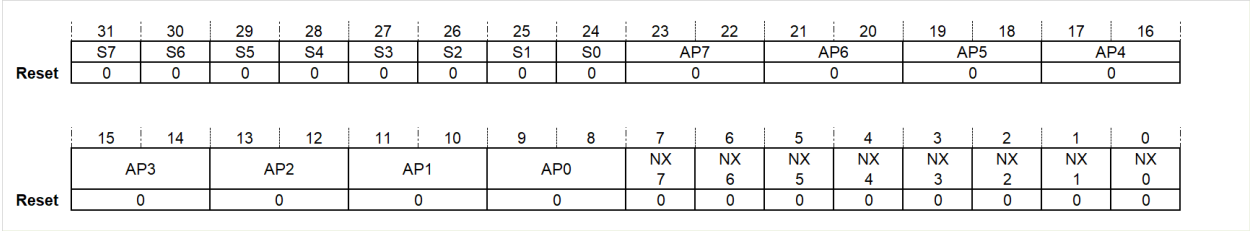


图 7.3: 可高缓和访问权限配置寄存器

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：

表 7.2: 访问权限设置

AP	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

7.2.3 保护区控制寄存器（PACR，CR<20,0>）

PACR 的各位如下图所示：

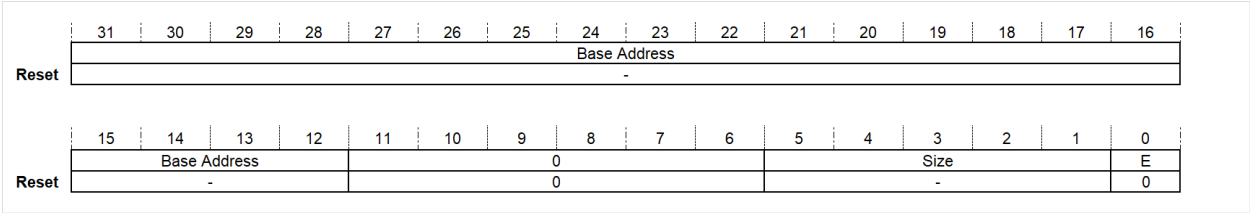


图 7.4: 保护区控制寄存器

Base Address-保护区地址的基地址：

- 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下表。

Size-保护区大小：

- 保护区大小从 4KB 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 01011 到 11111，其它一些值都会造成不可预测的结果。

表 7.3: 保护区大小配置和其对基址要求

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

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。

7.2.4 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：

- RID 表示所选择的对应的保护区，如 000 表示第 0 保护区。

7.3 内存访问处理

内存保护单元对 CPU 内存访问的处理流程及结果如图表 表 7.4 所示。

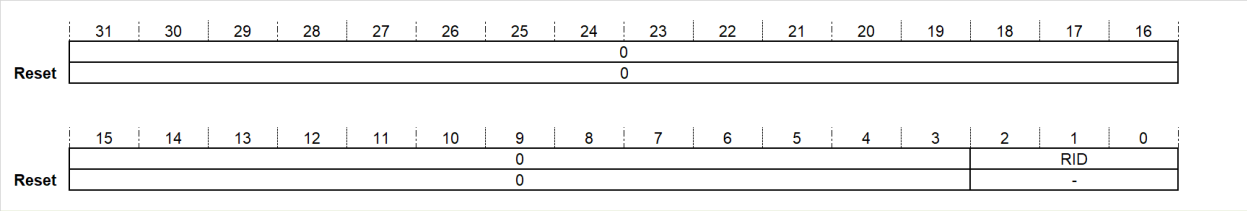


图 7.5: 保护区选择寄存器

表 7.4: 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; 访问属性为不可缓冲;
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。

第八章　片上高速缓存

8.1　高速缓存简介

E803 提供了硬件可配置的高速缓存，其主要特征如下：

- 高速缓存大小硬件可配置，支持 2K/4K/8KB/16KB；
- 4 路组相联，缓存行大小为 16 个字节；
- 每次访问的最大宽度为 4 字节，支持字节/半字/字访问；
- 物理地址索引，物理地址标记；
- 写策略同时支持写直模式和写回模式；
- 高速缓存采用先进先出（FIFO）的替换策略；
- 支持对整个高速缓存的无效和清除操作，支持对单条缓存行的无效和清除操作。

当内存访问不可高缓区域时，直接旁路片上高速缓存，通过总线访问片外存储器，加快不可高缓存区域的访问速度。当内存访问可高缓区域时，首先访问片上高速缓存；在高速缓存缺失的情况下再访问片外存储器（写直模式下的写访问在高速缓存命中时，也需要回写片外存储器）。

8.2　相关系统控制寄存器

Cache 提供一组 32-bit 的寄存器，包括 Cache 的使能、缓存行的无效和高速缓存区的配置等。各个寄存器地址空间如图表 表 8.1 所示。

表 8.1: Cache 控制寄存器定义

地址	名称	类型	初始值	描述
0xE000F000	CER	读/写	0	高速缓存使能寄存器
0xE000F004	CIR	读/写	0	高速缓存无效寄存器
0xE000F008	CRCR0	读/写	0	0 号可高缓区配置寄存器
0xE000F00C	CRCR1	读/写	0	1 号可高缓区配置寄存器
0xE000F010	CRCR2	读/写	0	2 号可高缓区配置寄存器
0xE000F014	CRCR3	读/写	0	3 号可高缓区配置寄存器
0xE000F018 ~0xE000FFF3	-	-	-	保留
0xE000FFF4	CPFCR	读/写	0	高速缓存性能分析控制寄存器
0xE000FFF8	CPFATR	读/写	0	高速缓存访问次数寄存器
0xE000FFFC	CPFMTR	读/写	0	高速缓存缺失次数寄存器

8.2.1 高速缓存使能寄存器 (CER)

高速缓存使能寄存器用于配置高速缓存（Cache）使能以及高速缓存的工作模式。

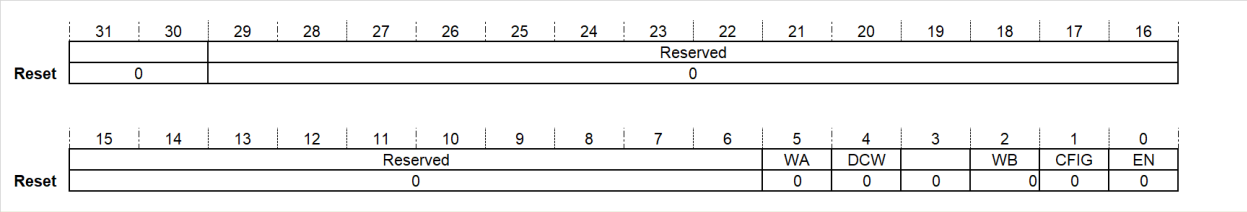


图 8.1: 高速缓存使能寄存器

WA 高速缓存写分配有效设置位：

- 0：高速缓存为 write non-allocate 模式；
- 1：高速缓存为 write allocate 模式。
- write allocate 模式只能在写回模式下使用，否则结果不可预知

DCW-Cache 可写属性配置位：

- 0：高速缓存不可写；
- 1：高速缓存可写。

WB-Cache 写回模式配置位：

- 0：高速缓存为写直模式；
- 1：高速缓存为写回模式。

CFIG-Cache 属性配置位：

- 0：指令与数据高速缓存；
- 1：指令高速缓存。

EN-Cache 使能位：

- 当 EN 为 0 时，高速缓存处于关闭状态。
- 当 EN 为 1 时，高速缓存处于工作状态。

8.2.2 高速缓存无效寄存器（CIR）

高速缓存无效寄存器用于控制高速缓存的清除和无效，可支持对特定高速缓存行操作和对整个高速缓存操作两种方式。

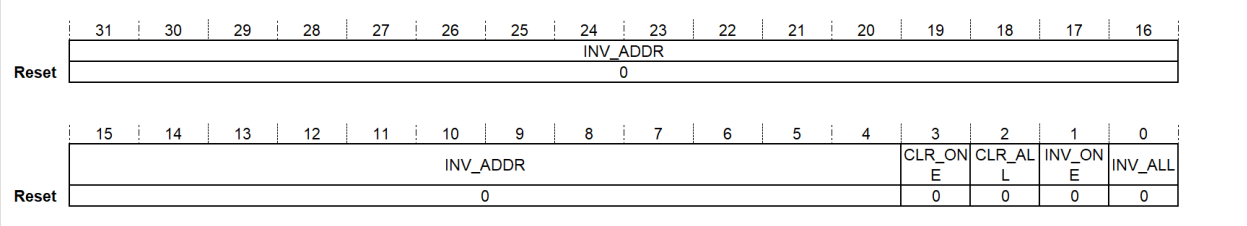


图 8.2: 高速缓存无效寄存器

INV_ADDR-缓存行地址：

- 表征需要被无效的缓存行地址。

CLR_ONE-单条缓存行清除设置位：

- 当 CLR_ONE 为 1 时，选中的缓存行若标记为脏，将被回写到片外存储器中。

CLR_ALL-整个高速缓存清除设置位：

- 当 CLR_ALL 为 1 时，标记为脏的所有缓存行，将被回写到片外存储器中。

INV_ONE-单条缓存行无效设置位：

- 当 INV_ONE 为 1 时，无效选中的缓存行。

INV_ALL-整个高速缓存无效设置位：

- 当 INV_ALL 为 1 时，无效高速缓存中所有缓存行。

注意：任何按行操作不可与任何全 cache 操作同时进行。

举例说明：往 CIR 中写 32’b1111，INV_ONE 和 CLR_ONE 操作将被屏蔽。

8.2.3 可高缓区配置寄存器 0~3（CRCR）

可高缓区配置寄存器用于配置可高缓（Cacheable）的地址空间范围，提供四个可高缓区配置寄存器，用来配置四块不同的可高缓区域。

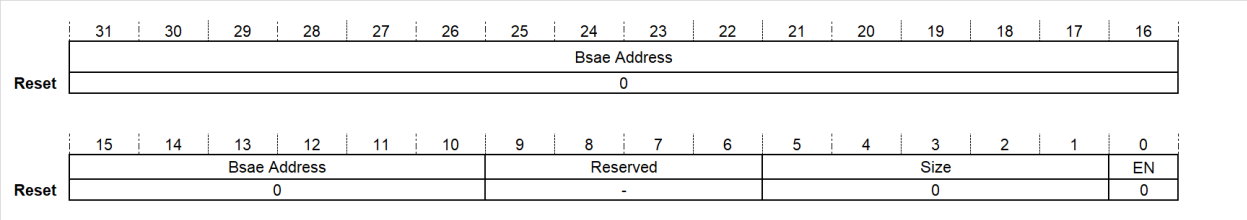


图 8.3: 可高缓区配置寄存器

Base Address-可高缓区基地址：

- 可高缓区大小 4KB 到 4GB 可配置，该域指出了可高缓区地址的高位，例如设置页面大小为 8M，CRCR [22:12] 必须为 0，不同大小的区各页面的具体要求见下表。

Size-可高缓区大小：

- 可高缓区大小可以通过公式：可高缓区大小= 2^（Size+1）计算得到。因此 Size 可设置的范围为 01011 到 11111，其它一些值都会造成不可预测的结果。

表 8.2: 可高缓区大小配置和其对基址要求

Size	高速缓存区大小	对基地址的要求
00000—01010	保留	-
01011	4KB	没有要求
01100	8KB	CRCR.bit[12]=0
01101	16KB	CRCR.bit[13:12] =0
01110	32KB	CRCR.bit[14:12] =0
01111	64KB	CRCR.bit[15:12] =0
10000	128KB	CRCR.bit[16:12] =0
10001	256KB	CRCR.bit[17:12] =0
10010	512KB	CRCR.bit[18:12]=0
10011	1MB	CRCR.bit[19:12]=0
10100	2MB	CRCR.bit[20:12]=0
10101	4MB	CRCR.bit[21:12]=0
10110	8MB	CRCR.bit[22:12]=0
10111	16MB	CRCR.bit[23:12]=0
11000	32MB	CRCR.bit[24:12]=0
11001	64MB	CRCR.bit[25:12]=0
11010	128MB	CRCR.bit[26:12]=0
11011	256MB	CRCR.bit[27:12]=0
11100	512MB	CRCR.bit[28:12]=0
11101	1GB	CRCR.bit[29:12]=0
11110	2GB	CRCR.bit[30:12]=0
11111	4GB	CRCR.bit[31:12]=0

EN-可高缓区有效位：

- 当 EN 为 0 时，可高缓区无效；
- 当 EN 为 1 时，可高缓区有效。

8.2.4 高速缓存性能分析控制寄存器（CPFCR）

高速缓存性能分析控制寄存器用于使能高速缓存性能分析功能（Cache Profiling）和重置性能分析相关计数器。

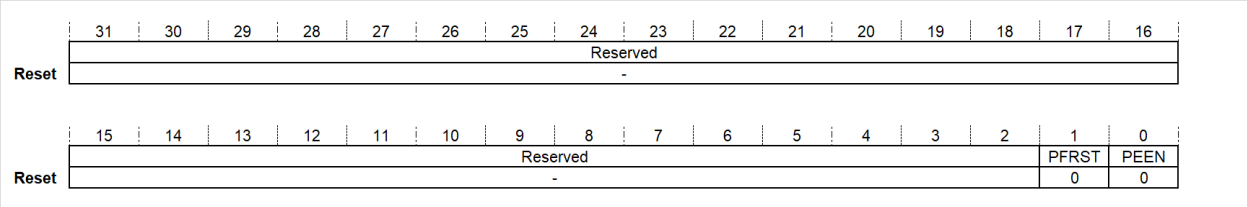


图 8.4: 高速缓存性能分析控制寄存器

PFRST-Cache profiling 复位位：

- 当 PERST 为 1 时，将 CPFATR/CPFMTR 复位为 0。

PFEN-Cache profiling 使能位：

- 当 PFEN 为 0 时，高速缓存性能分析功能处于关闭状态。
- 当 PFEN 为 1 时，高速缓存性能分析功能处于工作状态。

8.2.5 高速缓存访问次数寄存器（CPFATR）

高速缓存访问次数寄存器用于记录访问高速缓存的次数。

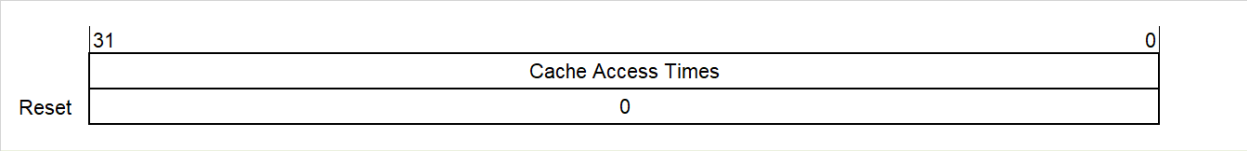


图 8.5: 高速缓存访问次数寄存器

Cache Access Times –高速缓存访问次数：

- 记录高速缓存被访问次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存访问将使寄存器数值加 1。

8.2.6 高速缓存缺失次数寄存器（CPFMTR）

高速缓存缺失次数寄存器用于记录访问高速缓存中缺失的次数。

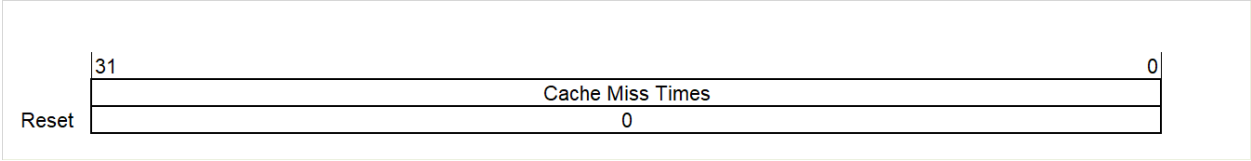


图 8.6: 高速缓存缺失次数寄存器

Cache Miss Times –访问高速缓存缺失次数:

- 记录访问高速缓存中的缺失次数，在高速缓存性能分析使能情况下，每 (256) 次高速缓存缺失将使寄存器数值加 1。

注意事项:

- 建议所有高速缓存配置操作都在其关闭情况下进行。

第九章　总线矩阵与总线接口

9.1　简介

E803 实现了多总线接口，分别包括系统总线、指令总线和数据总线。

总线矩阵为处理器内部请求访问外部总线接口提供了互联功能。总线矩阵与 CPU 内部请求及总线接口的连接关系如 图 9.1 所示。总线矩阵根据内存访问的地址仲裁总线接口类型，将处理器内部访问分发到系统总线、指令总线或数据总线上。

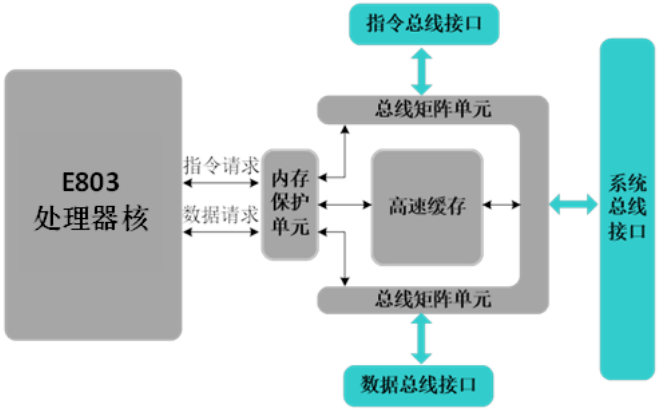


图 9.1: E803 总线矩阵

处理器内部的取指访问和数据访问拥有相同的总线访问权限，可以访问所有总线接口。为了解决同一时钟周期取指访问和数据访问竞争同一总线接口的问题，总线矩阵也负责请求的优先级判断。当取指请求和数据请求竞争同一总线接口时，数据请求拥有更高的优先级。

E803 多总线接口的基本信息和可配置性如图表 表 9.1 所示。

表 9.1: 多总线接口的基本信息和可配置性

总线接口	可配置性	总线协议	时序方式
系统总线	固定包含，协议可配	AHB	直接输出
系统总线	固定包含，协议可配	AHB-Lite	直接输出
指令总线	固定包含	AHB-Lite	直接输出
数据总线	固定包含	AHB-Lite	直接输出

另外，E803 通过提供一组接口信号（pad_bmu_xahbl_base 和 pad_bmu_xahbl_mask），支持指令总线和数据总线的基地址和空间大小硬件可配置。其中，pad_bmu_xahbl_base 指定了指令总线/数据总线的基地址。pad_bmu_xahbl_base 为位宽为 20 的信号，其表示的真实基地址需要将低位 0 扩展，即 {pad_bmu_xahbl_base[19:0],12' b0}。pad_bmu_xahbl_mask 指定了不同地址空间下对地址对齐的需求。指令总线或者数据总线的地址空间 4KB 到 4GB 可配置，例如设置地址空间大小为 8M，pad_bmu_xahbl_base[10:0] 必须为 11' b0，pad_bmu_xahbl_mask[19:0] 必须为 20' b1111 1111 1000 0000 0000，不同大小的地址空间具体要求见下表。

表 9.2: 指令总线和数据总线对基地址和地址对齐的要求

地址空间大小	对 pad_ bmu_xahbl_base 的要求	对 pad_ bmu_xahbl_mask 的要求
4KB	没有要求	bit[19:0]=20' b1111 1111 1111 1111 1111
8KB	bit[0] =1' b0	bit[19:0]=20' b1111 1111 1111 1111 1110
16KB	bit[1:0] =2' b0	bit[19:0]=20' b1111 1111 1111 1111 1100
32KB	bit[2:0] =3' b0	bit[19:0]=20' b1111 1111 1111 1111 1000
64KB	bit[3:0] =4' b0	bit[19:0]=20' b1111 1111 1111 1111 0000
128KB	bit[4:0] =5' b0	bit[19:0]=20' b1111 1111 1111 1110 0000
256KB	bit[5:0] =6' b0	bit[19:0]=20' b1111 1111 1111 1100 0000
512KB	bit[6:0] =7' b0	bit[19:0]=20' b1111 1111 1111 1000 0000
1MB	bit[7:0] =8' b0	bit[19:0]=20' b1111 1111 1111 0000 0000
2MB	bit[8:0] =9' b0	bit[19:0]=20' b1111 1111 1110 0000 0000
4MB	bit[9:0] =10' b0	bit[19:0]=20' b1111 1111 1100 0000 0000
8MB	bit[10:0] =11' b0	bit[19:0]=20' b1111 1111 1000 0000 0000
16MB	bit[11:0] =12' b0	bit[19:0]=20' b1111 1111 0000 0000 0000
32MB	bit[12:0] =13' b0	bit[19:0]=20' b1111 1110 0000 0000 0000
64MB	bit[13:0] =14' b0	bit[19:0]=20' b1111 1100 0000 0000 0000
128MB	bit[14:0] =15' b0	bit[19:0]=20' b1111 1000 0000 0000 0000
256MB	bit[15:0] =16' b0	bit[19:0]=20' b1111 0000 0000 0000 0000
512MB	bit[16:0] =17' b0	bit[19:0]=20' b1110 0000 0000 0000 0000
1GB	bit[17:0] =18' b0	bit[19:0]=20' b1100 0000 0000 0000 0000
2GB	bit[18:0] =19' b0	bit[19:0]=20' b1000 0000 0000 0000 0000
4GB	bit[19:0] =20' b0	bit[19:0]=20' b0000 0000 0000 0000 0000

9.2 系统总线接口

9.2.1 特点

E803 的系统总线接口可以配置为支持 AMBA2.0 AHB 协议（此时请参考 AMBA 2.0 规格说明—AMBA™ Specification Rev 2.0），也可以配置为支持 AMBA3.0 AHB-Lite 协议（此时请参考 AMBA 3.0 规格说明—AMBA3 AHB-Lite Protocol Specification Rev 1.0）。

系统总线接口的基本特点包括：

- 支持 AMBA2.0 AHB 或 AMBA3.0 AHB-Lite 总线协议，可由用户配置；
- 仅支持直接输出（non-Flop-out）；
- 处理器和系统时钟频率比必须为 1：1；

9.2.2 协议内容

9.2.2.1 支持传输类型

考虑到 E803 的应用领域及成本，系统总线接口只实现了 AHB/AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 和 WRAP4 传输，其它突发类型均不支持；
- HTRANS 支持 IDLE 和 NONSEQ 和 SEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

在 AHB 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 和 WRAP4 传输，其它突发类型均不支持；
- HTRANS 支持 IDLE、NONSEQ 和 SEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

9.2.2.2 支持响应类型

在 AHB 及 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

9.2.3 不同总线响应下的行为

图表 表 9.3 列出了总线上出现不同总线响应时 CPU 的行为。

表 9.3: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

9.2.4 AHB 协议的接口信号

表 9.4: AHB 协议接口信号

信号名	I/O	Reset	定义
biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 E803 支持 SINGLE 和 WRAP4 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。 E803 支持 IDLE、NONSEQ 和 SEQ 传输类型。
biu_pad_hwrite	O	0	读写表示信号： 1 ： 指示是写总线传输； 0 ： 指示是读总线传输。

下页继续

表 9.4 – 续上页

信号名	I/O	Reset	定义
biu_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
biu_pad_hbusreq	O	0	总线请求信号： 指示 CPU 请求总线的使用权。
biu_pad_hlock	O	0	锁定总线信号： 当 lock 申明时，CPU 独占有总线控制权，没有其他的 bus master 可以占有总线。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_biu_hgrant	I	-	总线占用指示信号： 有效时指示 CPU 当前已占用总线。
pad_biu_hresp[1:0]	I	-	传输应答信号： 00：OKAY； 01：ERROR； 10：RETRY； 11：SPLIT。 E803 仅支持 OKAY 和 ERROR 响应类型。

9.2.5 AHB-Lite 协议的接口信号

表 9.5: AHB-Lite 协议接口信号

信号名	I/O	Reset	定义
biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。

下页继续

表 9.5 – 续上页

信号名	I/O	Reset	定义
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 E803 支持 SINGLE 和 WRAP4 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。 E803 支持 IDLE、NONSEQ 和 SEQ 传输类型。
biu_pad_hwrite	O	0	读写表示信号： 1 ： 指示是写总线传输； 0 ： 指示是读总线传输。
biu_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。

下页继续

表 9.5 – 续上页

信号名	I/O	Reset	定义
pad_biu_hresp[1:0]	I	-	传输应答信号： 00：OKAY； 01：ERROR； 10：RETRY； 11：SPLIT。 E803 仅支持 OKAY 和 ERROR 响应类型。

9.3 指令总线接口

9.3.1 特点

E803 的指令总线只支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocol Specification Rev 1.0。

指令总线接口的基本特点有：

- 与 AMBA3.0 AHB-Lite 总线协议兼容；
- 仅支持直接输出（Non-Flop-out）方式；

9.3.2 协议内容

9.3.2.1 支持传输类型

考虑到 E803 的应用领域及成本，指令总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 只支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

9.3.2.2 支持响应类型

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

9.3.3 不同总线响应下的行为

图表 表 9.6 列出了总线上出现不同总线响应时 CPU 的行为。

表 9.6: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

9.3.4 指令总线接口信号

表 9.7: 指令总线接口信号

信号名	I/O	Reset	定义
iahbl_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
iahbl_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 E803 仅支持 SINGLE 突发传输。
iahbl_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
iahbl_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。

下页继续

表 9.7 – 续上页

信号名	I/O	Reset	定义
iahbl_pad_htrans[1:0]	O	00	传输类型表示信号: 00：IDLE; 01：BUSY; 10：NONSEQ; 11：SEQ。 E803 仅支持 IDLE 和 NONSEQ 传输类型。
iahbl_pad_hwdata[31:0]	O	-	写数据总线: 32 位写数据总线。
iahbl_pad_hwrite	O	0	读写表示信号: 1 ： 指示是写总线传输; 0 ： 指示是读总线传输。
pad_iahbl_hrdata[31:0]	I	-	读数据总线: 32 位读数据总线。
pad_iahbl_hready	I	-	传输完成指示信号: 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_iahbl_hresp	I	-	传输应答信号: 0：OKAY; 1：ERROR。
pad_bmu_iahbl_base[19:0]	I	-	IAHB-Lite 基址控制信号, 上电复位之后需固定
pad_bmu_iahbl_mask[19:0]	I	-	IAHB-Lite 地址对齐控制信号, 上电复位之后需固定

9.4 数据总线接口

9.4.1 特点

E803 的数据总线只支持 AMBA3.0 AHB-Lite 协议，可参考 AMBA 3.0 规格说明- AMBA3 AHB-Lite Protocol Specification Rev 1.0。

数据总线接口的基本特点有:

- 与 AMBA3.0 AHB-Lite 总线协议兼容;
- 仅支持直接输出（Non-Flop-out）方式。

9.4.2 协议内容

9.4.2.1 支持传输类型

考虑到 E803 的应用领域及成本，数据总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作；

9.4.2.2 支持响应类型

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持；

9.4.3 不同总线响应下的行为

图表 9-9 列出了总线上出现不同总线响应时 CPU 的行为：

表 9.8: 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

9.4.4 数据总线接口信号

表 9.9: 数据总线接口信号

信号名	I/O	Reset	定义
dahbl_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
dahbl_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 E803 仅支持 SINGLE 突发传输。

下页继续

表 9.9 – 续上页

信号名	I/O	Reset	定义
dahbl_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
dahbl_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。
dahbl_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。 E803 仅支持 IDLE 和 NONSEQ 传输类型。
dahbl_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
dahbl_pad_hwrite	O	0	读写表示信号： 1 ： 指示是写总线传输； 0 ： 指示是读总线传输。
pad_dahbl_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_dahbl_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_dahbl_hresp	I	-	传输应答信号： 0：OKAY； 1：ERROR。
pad_bmu_dahbl_base[19:0]	I	-	DAHB-Lite 基址控制信号，上电复位之后需固定
pad_bmu_dahbl_mask[19:0]	I	-	DAHB-Lite 地址对齐控制信号，上电复位之后需固定

9.5 指令与数据的访问顺序

一旦 CPU 配置了高速缓存，则所有不可高缓的数据都会绕过 cache 直接从总线矩阵访问对应的总线，如图中的通道 1 所示；可高缓的数据会首先访问 cache，如果命中则直接从 cache 中取回数据，如果不命中，则会由 cache 通过总线矩阵向总线发起请求，具体如图中的通道 2 所示。

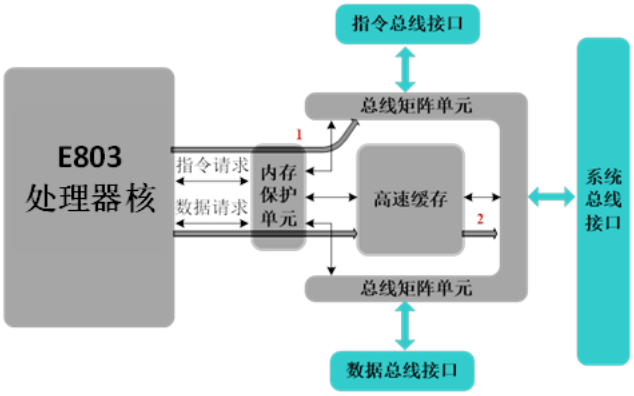


图 9.2: 访问外总线顺序

第十章　调试接口

10.1　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。

为了满足低成本的应用需求，节省 CPU 外部引脚，XuanTie CPU 定义了一套调试接口，JTAG2 接口。JTAG2 调试接口包括 JTAG2 通信协议，JTAG2 接口控制器。E803 支持 2 线制 JTAG 协议，调试接口使用 JTAG2 协议与外部的调试器通信。

调试接口的主要特性如下：

- 支持 2 线制 JTAG 接口；
- 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；
- 可使用 TCIP 接口访问调试寄存器资源，详见调试手册；

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 10.1 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调试器通过 USB 连接，调试器与 CPU 的调试接口以 JTAG 模式通信。

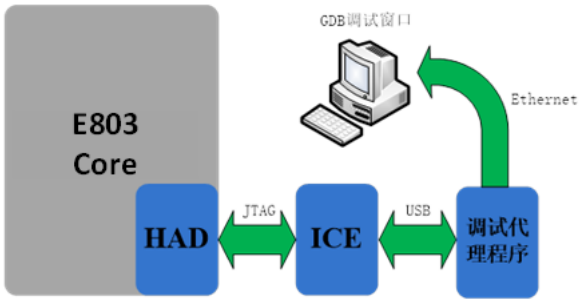


图 10.1: 调试接口在整个 CPU 调试环境中的位置

10.2 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 10.1 列出了与调试相关的接口信号。

表 10.1: 调试模块与外部的接口信号

信号名	方向
(sysio_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
had_pad_jtg_tms_o	输出
had_pad_htg_tms_oe	输出

1. **biu_pad_dbg_b (sysio_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 10.2 所示。

表 10.2: 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. **JTAG_2 相关信号**

pad_had_jtg_tms_i 信号为 2 线制 JTAG 串行数据输入信号，调试接口在 JTAG 时钟信号 TCLK 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

该信号在空闲时必须保持为高电平，同时空闲时时钟信号最好停止。用户可以利用该信号同步复位 HAD 逻辑：在实现 2 线制 JTAG 接口的调试模块中，如果时钟信号一直有效，用户只需保持该信号为高电平状态并维持 80 个时钟周期即可同步复位调试模块。复位调试模块后，调试模块的 TAP 状态机回到 RESET 态，同时调试模块寄存器 HACR 复位到 0x82（指向 ID 寄存器）。

had_pad_jtg_tms_o 信号为 2 线制 JTAG 串行数据输出信号，调试接口在 JTAG 时钟信号 TCLK 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

had_pad_jtg_tms_oe 信号为 had_pad_jtg_tms_o 信号有效指示信号。CPU 外部应利用该信号将 pad_had_jtg_tms_i 和 had_pad_jtg_tms_o 信号合为一个双向端口信号。

第十一章　工作模式与转换

E803 总共有三类工作模式：正常运行模式、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，这三种低功耗模式的不同是由 SOC 设计者定义的。本章将详细介绍 CPU 的工作模式以及模式之间的转换。

如图表 图 11.1 所示，E803 的工作模式有三类，即正常工作模式、低功耗工作模式和调试模式，CPU 处于哪种工作模式可以通过查询 sysio_pad_jdb_pm[1:0] 信号得到。

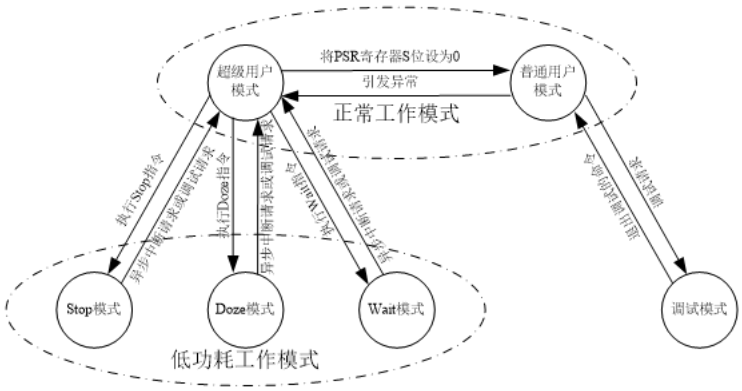


图 11.1: CPU 的各种工作状态示意图

11.1　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；当 S 位为 0 时，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。

11.2　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拉起 sysio_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。

只有异步中断请求（pad_sysio_intraw_b 和 pad_sysio_fintraw_b 信号）或者调试请求才能使 CPU 退出低功耗模式，然后 CPU 从进入低功耗模式的低功耗指令处继续执行后续指令。当前的 CPU 工作于哪种低功耗模式可以通过查询信号线 sysio_pad_lpmd_b[1:0] 来得到，每种模式具体对应的低功耗场景由 SOC 设计者决定。

11.2.1 调试模式

11.2.1.1 调试模式

CPU 进入调试模式后将停止取指和执行指令，处于一种等待状态，该状态下 CPU 只执行 HAD 输入的指令，通过 HAD 输入指令可以查询和修改 CPU 的状态，进而进行调试。

11.2.1.2 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 7 种：

- HAD 的外部输入信号（pad_had_jdb_req_b）可使处理器异步进入调试模式。
- 当 E803 HCR 的 IDRE 位有效时，HAD 的外部输入信号 biu_had_sdb_req_b(pad_biu_dbgreq_b) 被 CPU 同步后请求处理器同步进入调试模式。
- 当 E803 HCR 的 ADR 位有效时，处理器直接进入调试模式。
- 当 E803 HCR 的 DR 位有效时，处理器在完成当前指令后进入调试模式。
- 当 E803 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式。
- 当 E803 HCR 的 TME 位有效时，处理器在跟踪计数器的值减到 0 后进入调试模式。
- 在 E803 存储硬件断点进入调试模式。

处理器执行完当前的指令，保存流水线的信息，然后进入调试模式。当处理器处于低功耗模式时，通过设置 HCR 中的 ADR 以及 DR 的调试请求，可以使得处理器退出低功耗并进入调试模式。进入调试模式后，CPU 停止执行当前指令，等待用户通过调试接口输入有效的指令用于执行或退出调试模式。

11.2.1.3 退出调试模式

如果 E803 HACR 的 GO、EX 位被置为 1，同时 R/W 为 0（写操作），RS 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时，上述寄存器中的值必须是刚进入调试模式时保存过的值。

第十二章　初始化参考代码

12.1　MPU 设置示例

/* 设置区域 0 属性，可执行，非安全，超级用户和普通用户可读写

mfcr r10,cr<19,0>

bclri r10,0 //设置控制寄存器第 0 位为 0，区域 0 可执行

bseti r10,8 //设置控制寄存器 8 和 9 位为 1，区域 0 超级用户和普通用户都可读写

bseti r10,9

bclri r10,24 //设置控制寄存器第 24 位为 0，区域 0 为非安全区域

mtcr r10,cr<19,0>

/* 设置区域 1 属性，不可执行，安全，超级用户和普通用户可读写

mfcr r10,cr<19,0>

bseti r10,1 //设置控制寄存器第 1 位为 1，区域 1 不可执行

bseti r10,10 //设置控制寄存器 10 和 11 位为 1，区域 1 超级用户和普通用户都可读写

bseti r10,11

bseti r10,25 //设置控制寄存器第 25 位为 1，区域 1 为安全区域

mtcr r10,cr<19,0>

/* 设置区域 2 属性，不可执行，非安全，超级用户可读写，普通用户只读

mfcr r10,cr<19,0>

bseti r10,2 //设置控制寄存器第 2 位为 1，区域 2 不可执行

bclri r10,12 //设置 12 位为 0，13 位为 1，超级用户可读写，普通用户只读

bseti r10,13

bclri r10,26 //设置控制寄存器第 26 位为 0，区域 2 非安全区域

mtcr r10,cr<19,0>

```
/* 设置区域 3 属性，可执行，安全，超级用户和普通用户都可读写
mfcr r10,cr<19,0>
bclri r10,3 //设置控制寄存器第 3 位为 0，区域 3 可执行
bseti r10,14 //设置 14，15 位为 1，区域 3 超级用户和普通用户都可读写
bclri r10,15
bseti r10,27 //设置控制寄存器第 27 位为 1，区域 3 为安全区域
mtcr r10,cr<19,0>
/* 区域 4~7 属性设置和区域 0、1、2、3 的属性设置相同
//区域 0~7 的可执行属性，设置控制寄存器 CR<19,0> 的 [7:0] 位
//区域 0~7 的访问权限，设置控制寄存器 CR<19,0> 的 [23:8] 位
//区域 0~7 的安全属性，通过设置控制寄存器 CR<19,0> 的 [31:24] 位
/* 设置保护区域 0，基地址和保护区大小
movi r10,0
mtcr r10,cr<21,0> //选择保护区域 0
movi r10,0x0 //设置保护区基地址 0x00000000
ori r10,r10,0x3f //设置保护区大小 4G
mtcr r10,cr<20,0> //设置保护区基地址为 0，保护区大小为 4G
/* 设置保护区域 1，基地址和保护区大小
movi r10,1
mtcr r10,cr<21,0> //选择保护区域 1
movih r10,0x2800 //设置保护区基地址 0x28000000
ori r10,r10,0x2f //设置保护区大小 16M
mtcr r10,cr<20,0> //设置保护区地址区间为 0x28000000 ~ 0x29000000
/* 设置保护区域 2，基地址和保护区大小
movi r10,2
mtcr r10,cr<21,0> //选择保护区域 2
movih r10,0x2800 //设置保护区基地址 0x28000000
ori r10,r10,0x27 //设置保护区大小 1M
mtcr r10,cr<20,0> //设置保护区地址区间为 0x28000000 ~0x28100000
/* 设置保护区域 3，基地址和保护区大小
```


movi r10,3

mtcr r10,cr<21,0> //选择保护区域 2

movih r10,0x28f0 //设置保护区基地址 0x28f00000

ori r10,r10,0x27 //设置保护区大小为 1M

mtcr r10,cr<20,0> //设置保护区地址区间为 0x28f00000 ~0x29000000

//保护区域 3~7 的基地址和保护区大小设置和区域 0、1、2 的设置相同

//设置控制寄存器 CR<21,0> 选择保护区域

//将保护区基地址和保护区大小，写入控制寄存器 CR<20,0>,

/* 使能 MPU

mfcr r7, cr<18,0> //选择 MUP 使能控制寄存器

bseti r7, 0 //设置控制寄存器 CR<18,0> 最低两位为 2’b01, 使能 MPU

bclri r7, 1

mtcr r7, cr<18,0> //将预置值写入 MPU 使能控制寄存器，开启 MPU

12.2 高速缓存设置示例

开启高速缓存之前需要对整个高速缓存无效化，然后再根据实际应用需求配置 CER 并使能 CACHE。

// 高速缓存无效化

lrw r1，1 // 预设第 0 位 INV_ALL 为 1

lrw r2，0xe000f004 // 预设 CIR 所在地址

st.w r1，(r2) // 将预设值写入 CIR，无效化操作开始

// 设置可高速缓存区域 crcr0

lrw r1，0x00000039 // 预设起始地址为 0x0 的 512M 地址空间可高速缓存,

lrw r2，0xe000f008 // 预设 CRCR0 所在地址

st.w r1，(r2) // 将预设值写入 CIR，无效化操作开始

//可缓冲区域 crcr1~3 设置同 crcr0 ，将可高速缓冲的起始地址和 size 及使能位分别写到

//crcr1 地址 0xe000f00c,

//crcr2 地址 0xe000f010

//crcr3 地址 0xe000f014

// 开启高速缓存

lrw r1，0x0 // 预设值为 0x0000000

bseti r1，0 //设置 Cache enable 打开

bclri r1，1 //设置，指令和数据都可高速缓冲，

bseti r1，2 //设置 Cache 为 write back 模式

bseti r1，4 //设置 Cache 可写

bclri r1，5 //设置 Cache 关闭 write allocate

lrw r2，0xe000f000 // 预设 CER 所在地址

st.w r1，(r2) //将预设值写入 CER，Cache 使能

//此时 cache 可写并为写回模式，四路组相连，指令和数据 Cache，关闭 write allocate 功能

12.3 中断使能初始化

在配置好中断控制器和中断向量表之后（具体参考紧耦合 *IP*），需要将中断使能位打开，具体设置如下：

psrset ee，ie

12.4 通用寄存器初始化示例

//初始化通用寄存器 r0~r15 和 r28 为 0。

movi r0，0 //初始化通用寄存器 0 为 0

movi r1，0

movi r2，0

movi r3，0

movi r4，0

movi r5，0

movi r6，0

movi r7，0

movi r8，0

movi r9，0

movi r10，0

movi r11，0

movi r12，0

movi r13，0

movi r14，0

movi r15，0

movi r28，0

12.5 堆栈指针初始化示例

堆栈指针的设置和程序当前处在状态有关系如下所示，超级用户态下堆栈指针初始化为示例 1，用户态下堆栈指针初始化为示例 2。

示例 1：

//超级用户态下，设置超级用户态和用户态堆栈指针

//超级用户态下 r14 映射为超级用户态的堆栈指针寄存器

//超级用户态下，用户态堆栈指针寄存器为 CR<14,1>

lrw r14，0x01000000 //设置超级用户态指针

lrw r0，0x02000000 //

mtcr r0，cr<14,1> //设置用户态堆栈指针

示例 2：

//用户态下，只能设置用户态指针：

//用户态下 r14 映射为用户态的堆栈指针寄存器

lrw r14, 0x02000000 //设置用户态指针

12.6 异常和中断服务程序入口地址设置示例

异常和中断服务程序的入口地址设置分两个步骤：

Step1：设置异常向量表地址寄存器 VBR，根据向量号，各个异常向量号对应异常向量表地址为，VBR + （向量号 << 2）

Step2：将异常服务程序的入口地址写到 Step1 中异常向量号对应的异常向量表地址中。

异常向量号为 2 的访问错误异常示例如下：

//设置异常向量表入口地址，VBR 对应控制寄存器 cr<1,0>

Lrw r2，0

mtcr r2, cr<1,0> //设置异常向量表的地址为 0

//异常/中断服务程序入口地址设置

lrw r1, ACCERR_ERROR_BEGIN //设置异常服务程序入口地址


```
lrw r2, 0 //VBR 地址
movi r3, 0x2 //异常向量号
lsli r3，r3，2
addu r2，r2，r3 //计算异常向量号对应异常向量表地址
st.w r1,(r2，0x0) //将异常服务程序入口地址，存入相应的异常向量表地址中
br START
//用户设置的访问错误异常服务程序
label ACCERR_ERROR_BEGIN
/用户的异常服务程序/
label ACCERR_ERROR_END
label START
```


第十三章　附录 A 指令术语表

以下是每条 E803 实现的 XuanTie 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 XuanTie CPU V2 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。

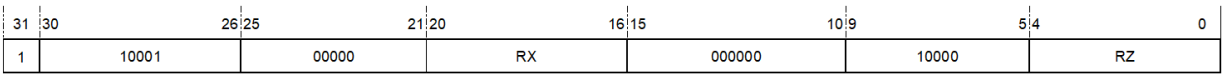
其中，指令中文名称中带 # 的为伪指令。

13.1　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:



13.2 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：

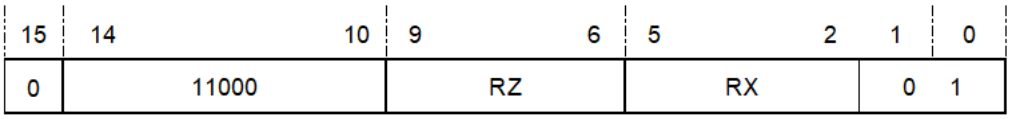


图 13.1: ADDC-1

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：

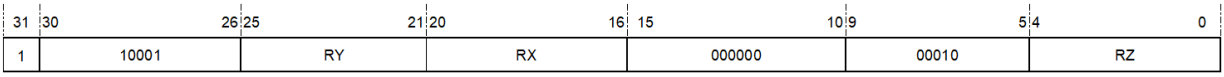


图 13.2: ADDC-2

13.3 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：

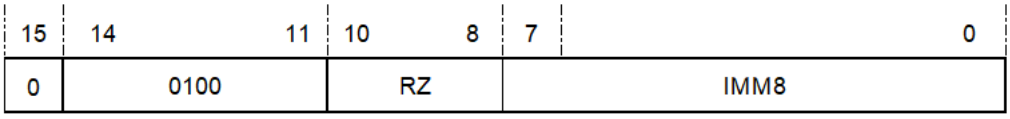


图 13.3: ADDI-1

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：

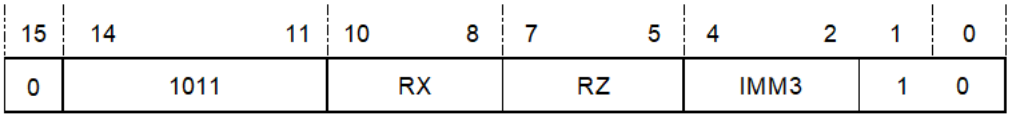


图 13.4: ADDI-2

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：

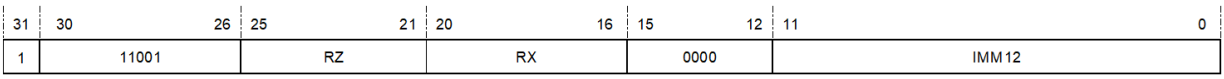


图 13.5: ADDI-3

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：

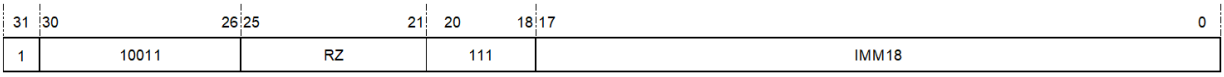


图 13.6: ADDI-4

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

13.4 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：

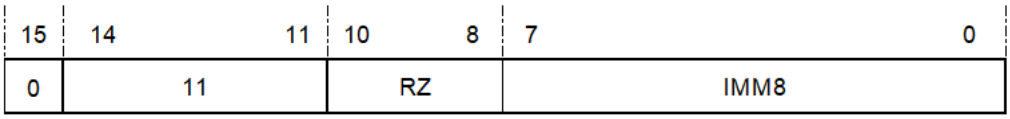


图 13.7: ADDI(SP)-1

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：

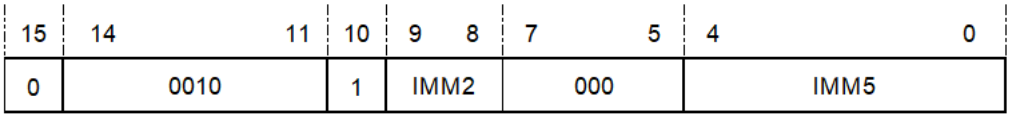


图 13.8: ADDI(SP)-2

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

13.5 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：

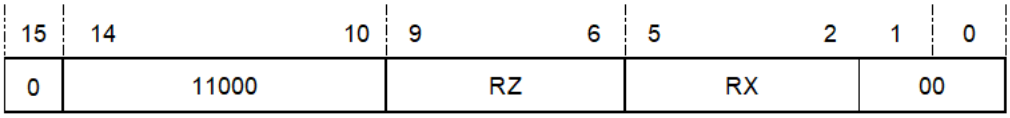


图 13.9: ADDU-1

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：

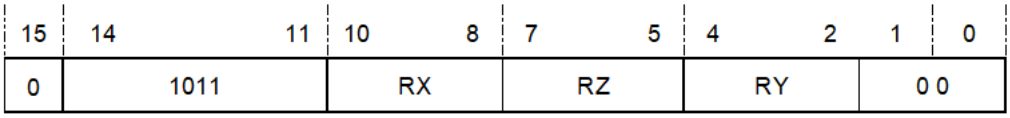


图 13.10: ADDU-2

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：

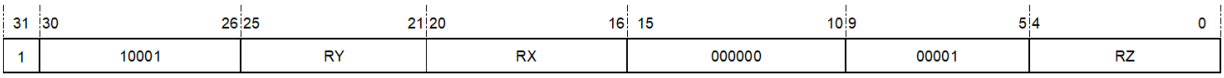


图 13.11: ADDU-3

13.6 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：

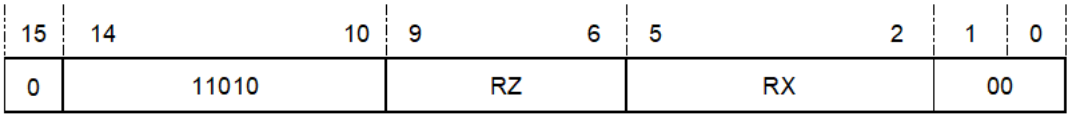


图 13.12: AND-1

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：

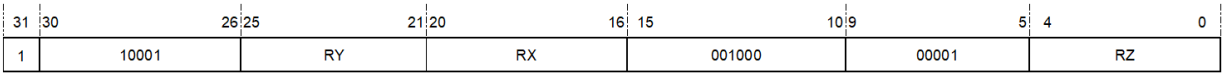


图 13.13: AND-2

13.7 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：

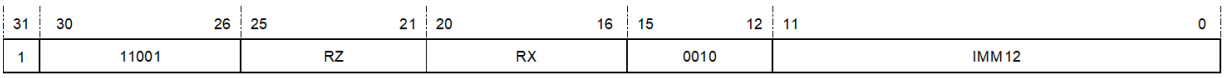


图 13.14: ANDI

13.8 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：

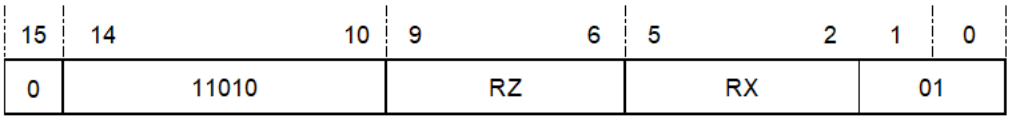


图 13.15: ANDN-1

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：

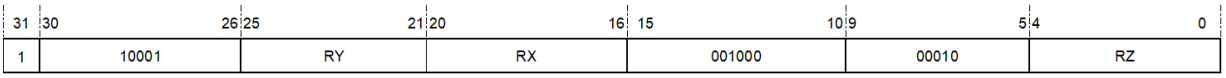


图 13.16: ANDN-2

13.9 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：

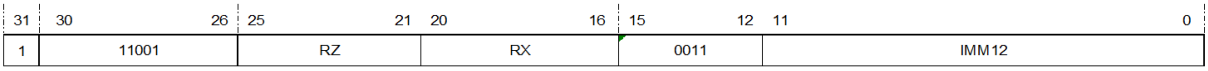


图 13.17: ANDNI

13.10 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：

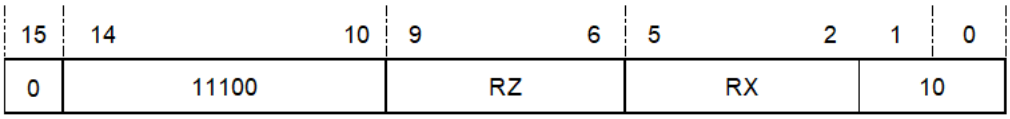


图 13.18: ASR-1

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：

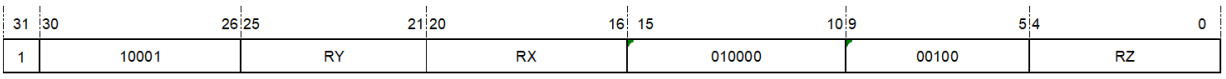


图 13.19: ASR-2

13.11 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：

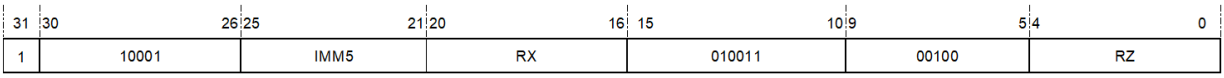


图 13.20: ASRC

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

13.12 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：

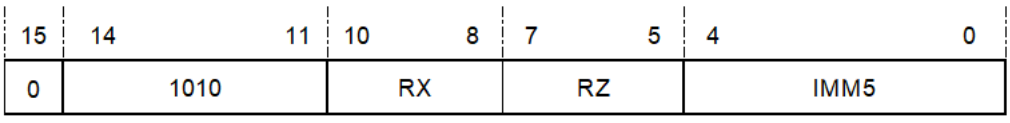


图 13.21: ASRI-1

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：

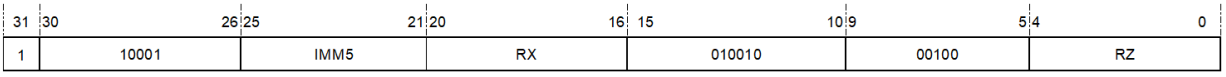


图 13.22: ASRI-2

13.13 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：

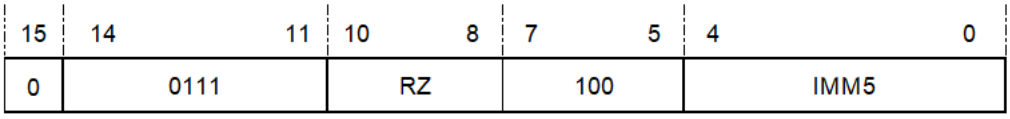


图 13.23: BCLRI-1

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：

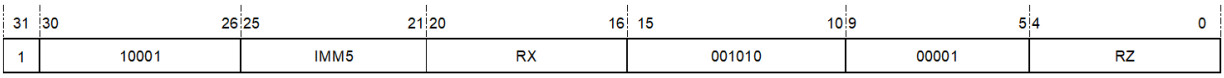


图 13.24: BCLRI-2

13.14 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：

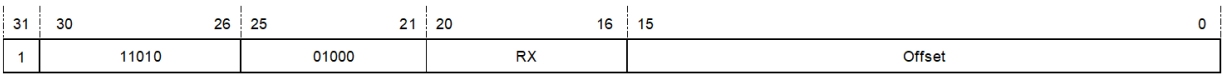


图 13.25: BEZ

13.15 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：

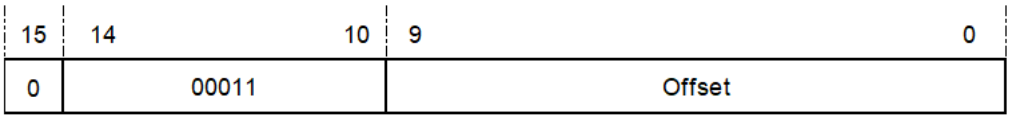


图 13.26: BF-1

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：

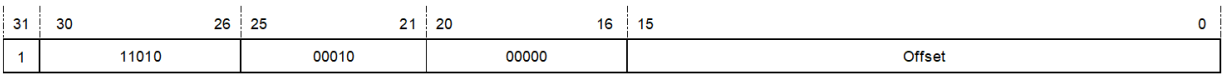


图 13.27: BF-2

13.16 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：

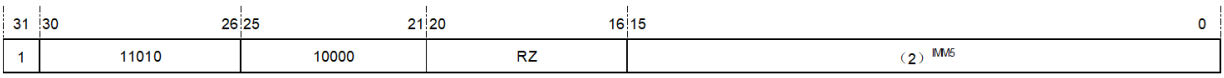


图 13.28: BGENI-1

如果 IMM5 大于等于 16：

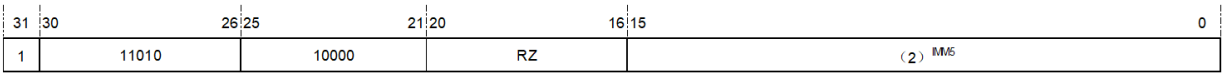


图 13.29: BGENI-2

13.17 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：

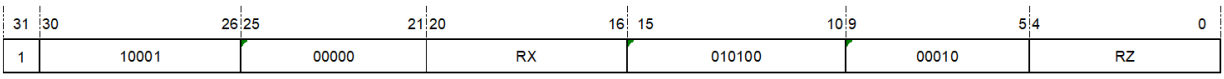


图 13.30: BGENR

13.18 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：

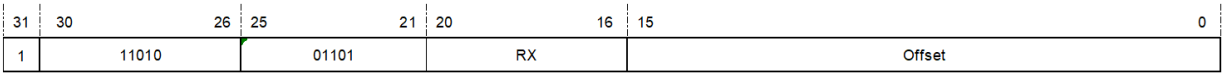


图 13.31: BHSZ

13.19 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：

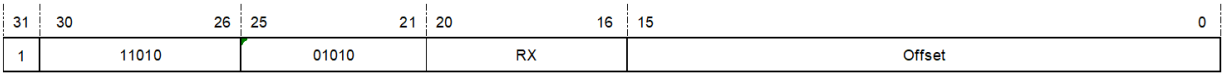


图 13.32: BHZ

13.20 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：

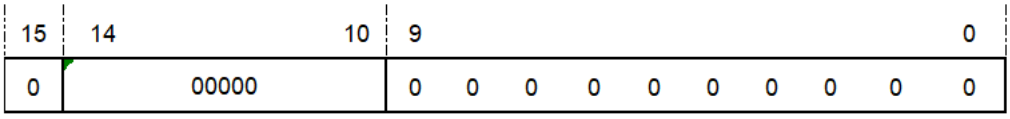


图 13.33: BKPT

13.21 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：

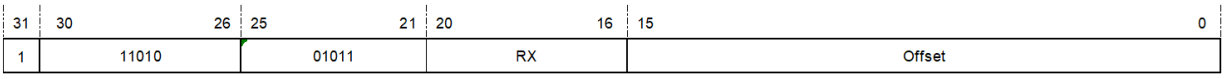


图 13.34: BLSZ

13.22 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：

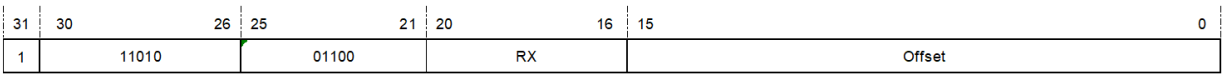


图 13.35: BLZ

13.23 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：

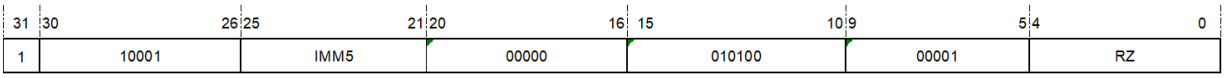


图 13.36: BMASKI

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

13.24 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：

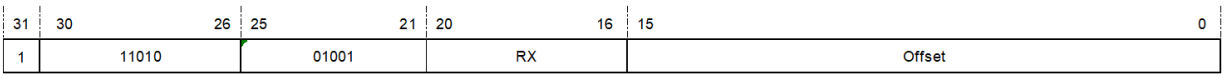


图 13.37: BNEZ

13.25 BNEZAD——寄存器自减大于零分支指令

统一化指令	
语法	bnezad rx, label

操作	寄存器自减 1 后判断，大于零则程序转移，自减结果无条件更新寄存器。
	RX ← RX-1
	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:	寄存器 RX 自减 1 后判断，若结果大于零，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。自减结果无条件更新寄存器 RX。 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。BNEZAD 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
	RX ← RX-1
	if(RX> 0)
	PC ← PC + sign_extend(offset << 1)
	else
	PC ← PC + 4

语法:	bnezad32 rx, label
说明:	寄存器 RX 自减 1 后判断，如果结果大于零，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。自减结果无条件更新寄存器 RX。 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。BNEZAD 指令的转移范围是 ±64KB 地址空间。
影响标志位:	无影响
异常:	无

指令格式:

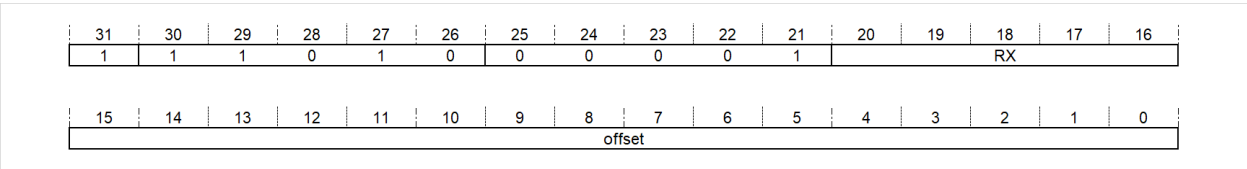


图 13.38: BNEZAD

13.26 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：

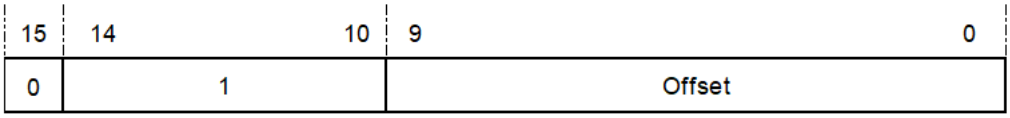


图 13.39: BR-1

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：

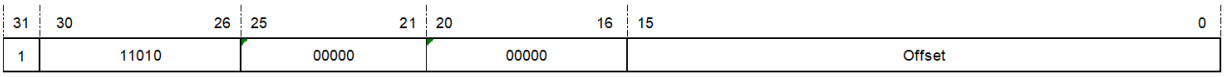


图 13.40: BR-2

13.27 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：

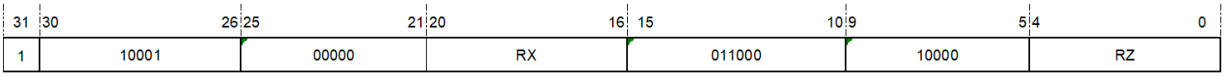


图 13.41: BREV

13.28 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：

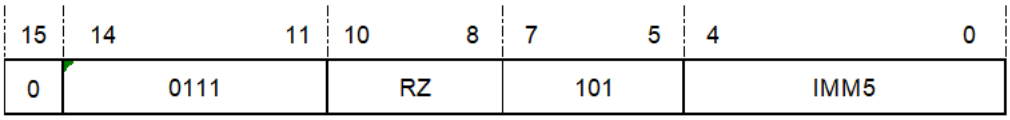


图 13.42: BSETI-1

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：

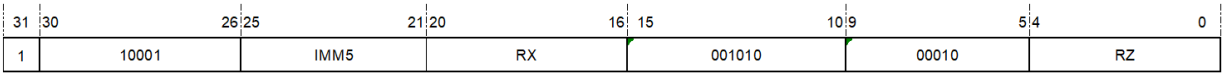


图 13.43: BSETI-2

13.29 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:

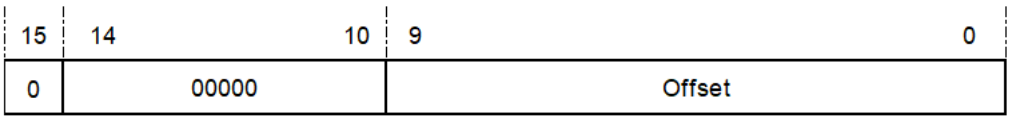


图 13.44: BSR-1

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:

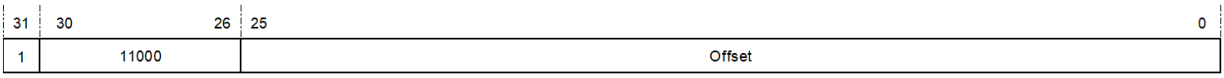


图 13.45: BSR-2

13.30 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：

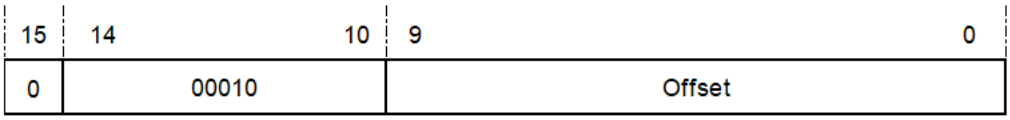


图 13.46: BT-1

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：

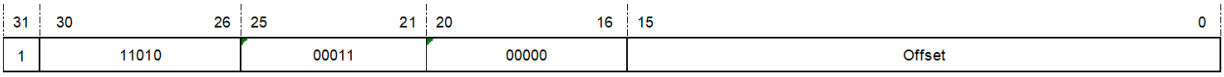


图 13.47: BT-2

13.31 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：

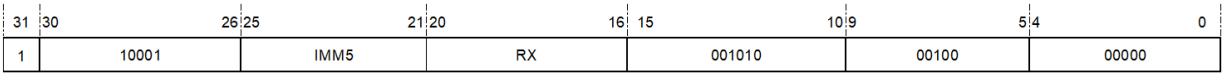


图 13.48: BTSTI

13.32 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：

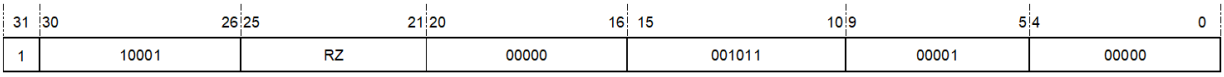


图 13.49: CLRF

13.33 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：

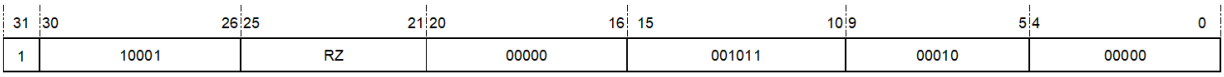


图 13.50: CLRT

13.34 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：

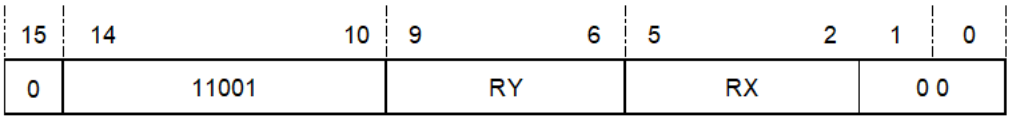


图 13.51: CMPHS-1

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：

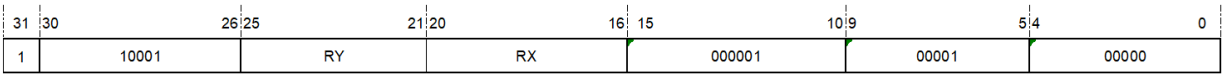


图 13.52: CMPHS-2

13.35 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：

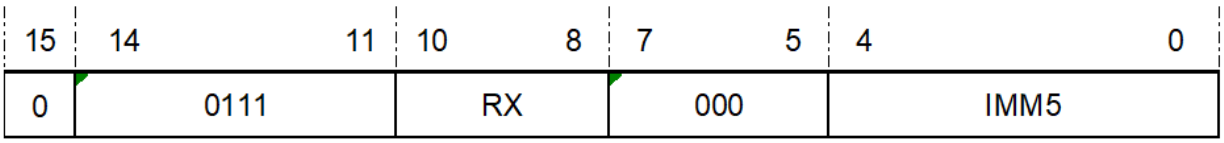


图 13.53: CMPHSI-1

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：

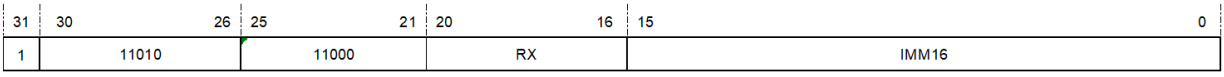


图 13.54: CMPHSI-2

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

13.36 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：

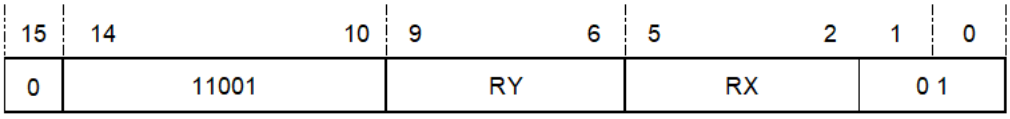


图 13.55: CMPLT-1

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：

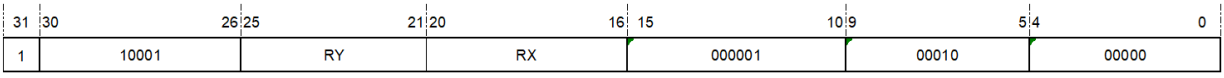


图 13.56: CMPLT-2

13.37 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：

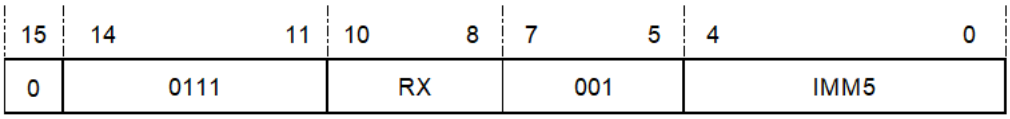


图 13.57: CMPLT-1

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：

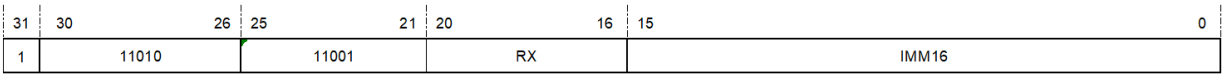


图 13.58: CMPLT-2

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

13.38 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：

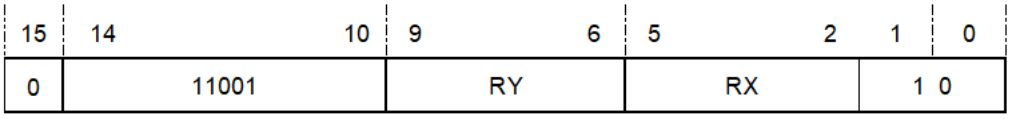


图 13.59: CMPNE-1

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：

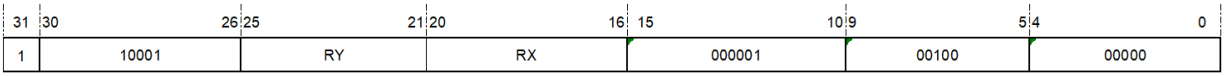


图 13.60: CMPNE-2

13.39 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：

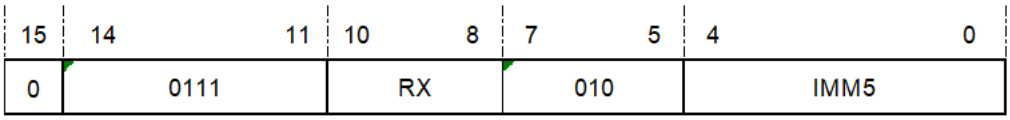


图 13.61: CMPNEI-1

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：

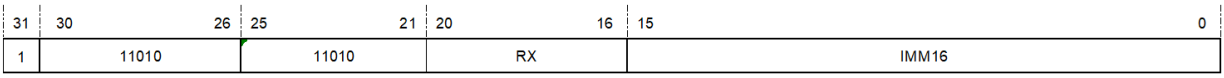


图 13.62: CMPNEI-2

13.40 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：

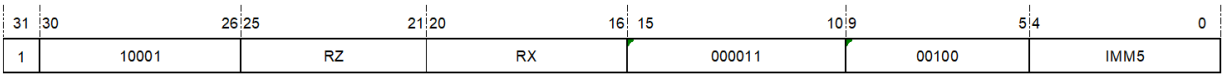


图 13.63: DECF

13.41 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：

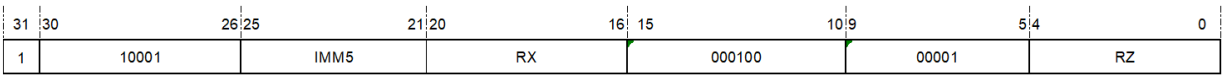


图 13.64: DECGT

13.42 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：

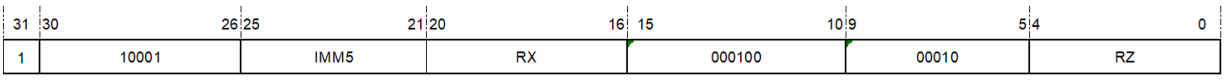


图 13.65: DECLT

13.43 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：

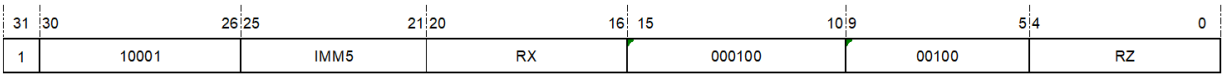


图 13.66: DECNE

13.44 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：

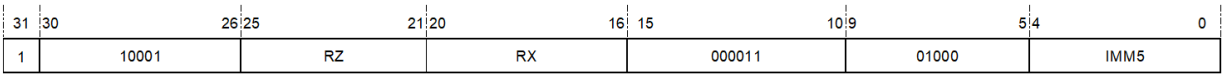


图 13.67: DECT

13.45 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：

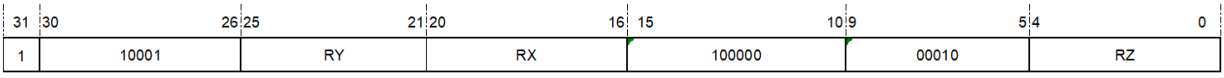


图 13.68: DIVS

13.46 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：

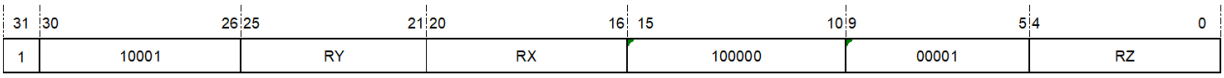


图 13.69: DIVU

13.47 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：

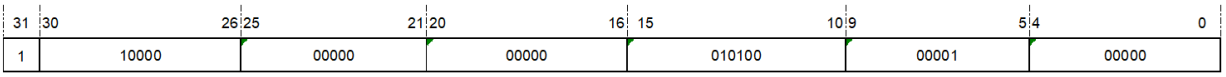


图 13.70: DOZE

13.48 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：

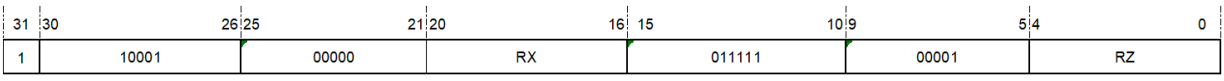


图 13.71: FF0

13.49 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：

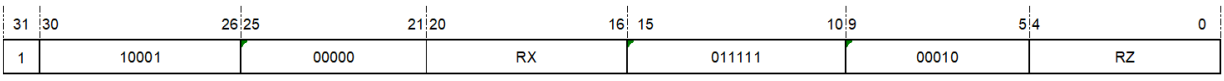


图 13.72: FF1

13.50 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 13.73: GRS

13.51 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：

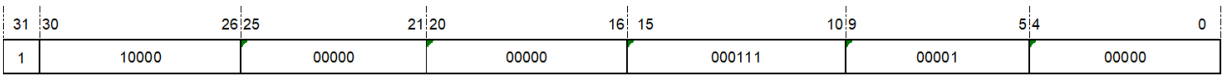


图 13.74: IDLY

13.52 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：

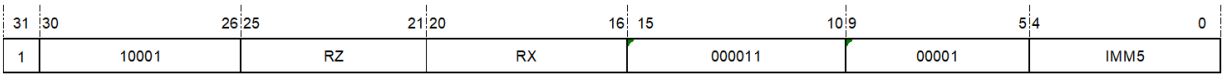


图 13.75: INCF

13.53 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：

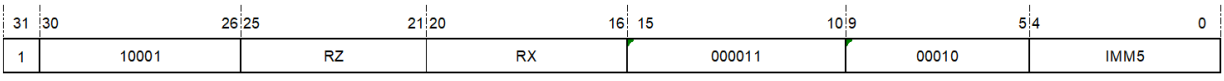


图 13.76: INCT

13.54 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：

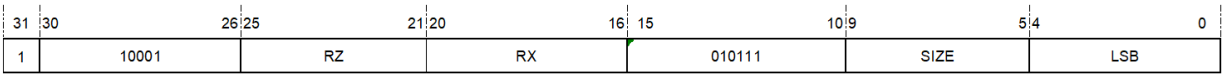


图 13.77: INS

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

13.55 IPOP——中断出栈指令

统一化指令	
语法	Ipop
操作	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端； {R0~R3,R12,R13} ← MEM[SP] ~ MEM[SP+20]; SP←SP+24;
编译结果	仅存在 16 位指令 ipop;

说明：	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
影响标志位：	无影响
异常：	访问错误异常、未对齐异常

16 位指令	
操作:	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端; {R0~R3,R12,R13} ← MEM[SP]~MEM[SP+20]; SP←SP+24;
语法:	ipop16
说明:	从堆栈指针寄存器中载入中断的通用寄存器现场 {R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常:	访问错误异常、未对齐异常

指令格式:

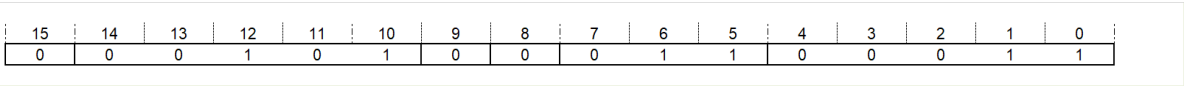


图 13.78: IPOP

13.56 IPUSH——中断压栈指令

统一化指令	
语法	Ipush
操作	将中断的通用寄存器现场 {R0~R3, R12, R13 } 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端; MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;
编译结果	仅存在 16 位指令 ipush;

说明:	将中断的通用寄存器现场 { R0~R3, R12, R13 } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常:	访问错误异常、未对齐异常

16 位指令	
操作:	将中断的通用寄存器现场 {R0~R3, R12, R13 } 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端; MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;
语法:	ipush16
说明:	将中断的通用寄存器现场 { R0~R3, R12, R13 } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常:	访问错误异常、未对齐异常

指令格式:

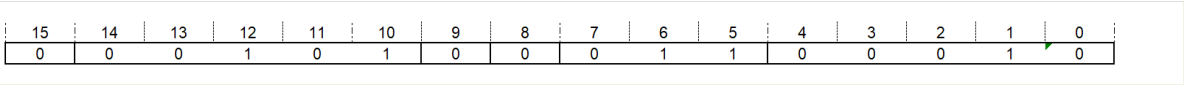


图 13.79: IPUSH

13.57 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：

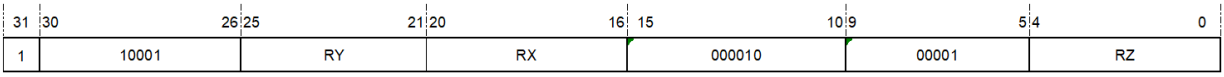


图 13.80: IXH

13.58 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：

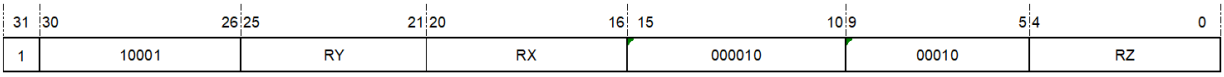


图 13.81: IXW

13.59 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：

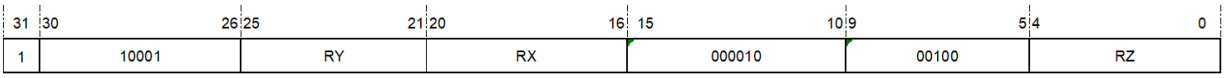


图 13.82: IXD

13.60 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：

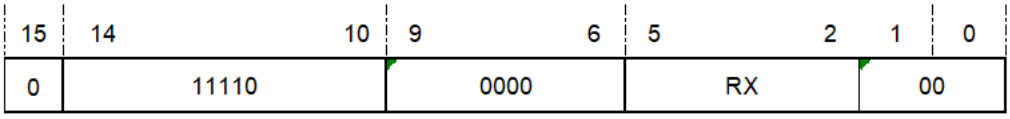


图 13.83: JMP-1

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：

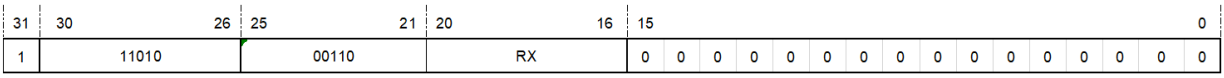


图 13.84: JMP-2

13.61 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：

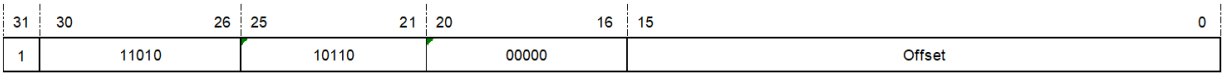


图 13.85: JMPI

13.62 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：

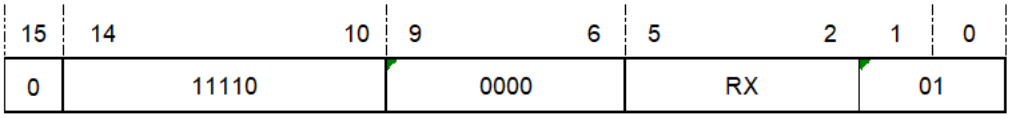


图 13.86: JSR-1

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：

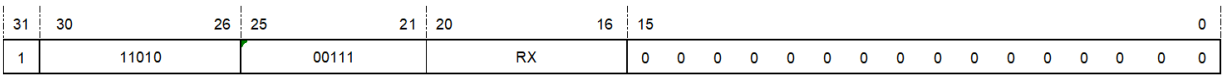


图 13.87: JSR-2

13.63 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：

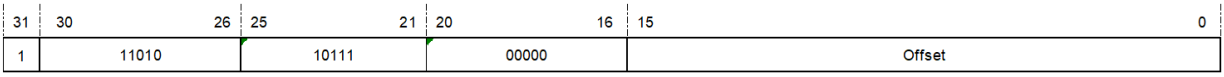


图 13.88: JSRI

13.64 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：

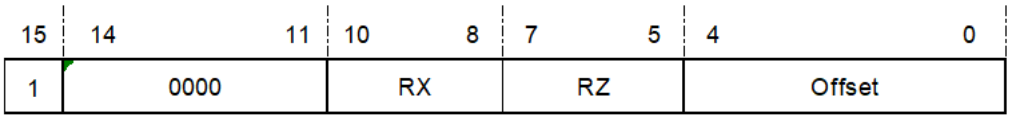


图 13.89: LD.B-1

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：

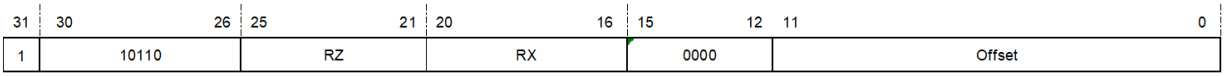


图 13.90: LD.B-2

13.65 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：

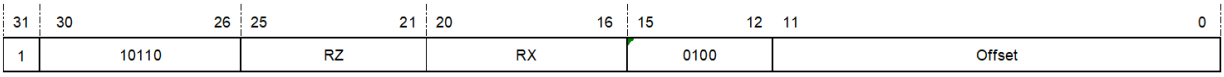


图 13.91: LD.BS

13.66 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：

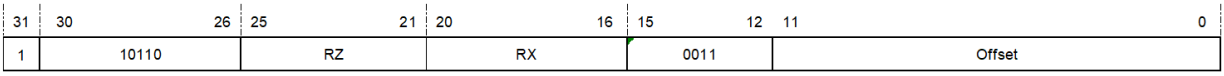


图 13.92: LD.D

13.67 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：

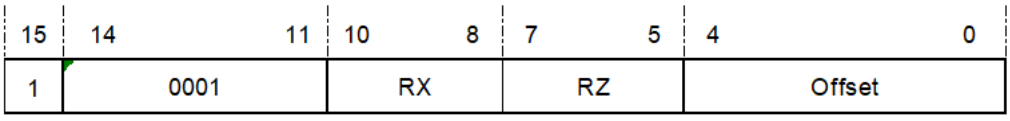


图 13.93: LD.H-1

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：

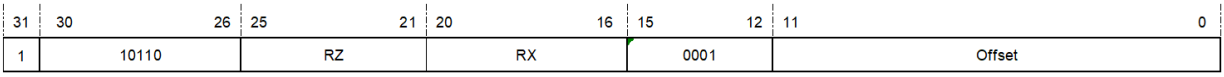


图 13.94: LD.H-2

13.68 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：

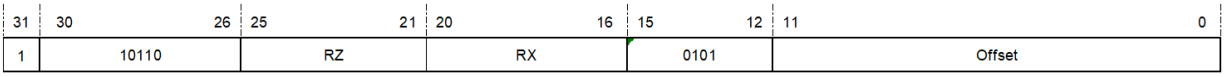


图 13.95: LD.HS

13.69 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)

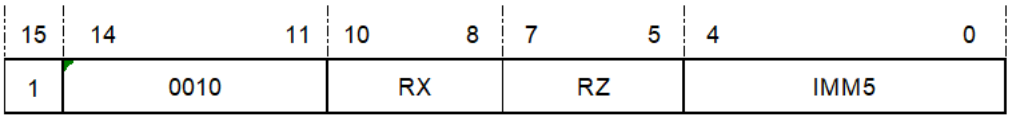


图 13.96: LD.W-1

ld16.w rz, (sp, disp)

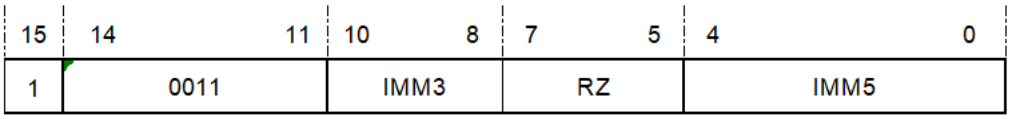


图 13.97: LD.W-2

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：

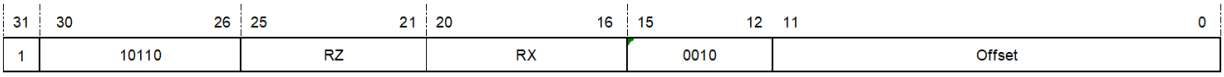


图 13.98: LD.W-3

13.70 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：

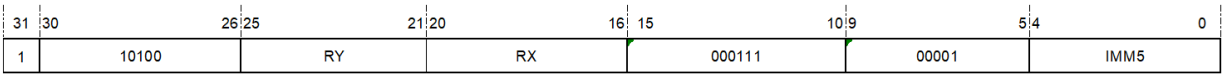


图 13.99: LDM

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

13.71 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：

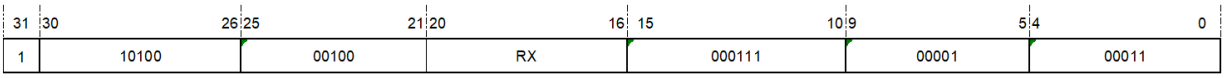


图 13.100: LDQ

13.72 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)

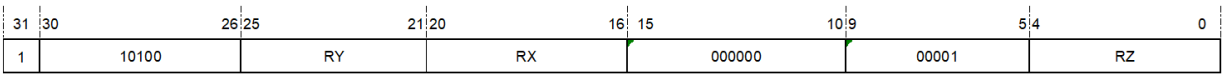


图 13.101: LDR.B-1

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

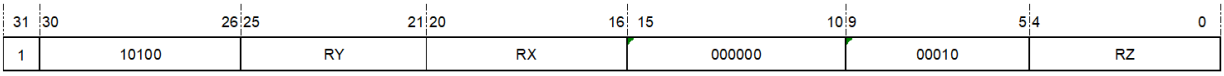


图 13.102: LDR.B-2

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

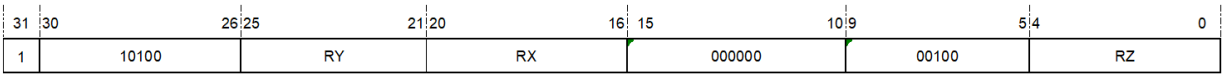


图 13.103: LDR.B-3

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

13.73 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

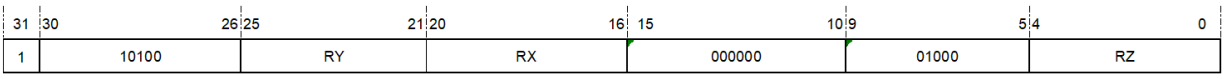


图 13.104: LDR.B-4

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)

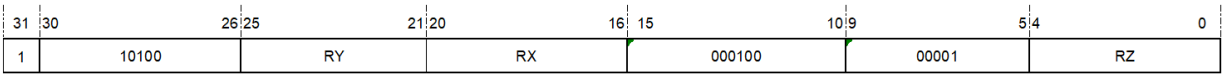


图 13.105: LDR.BS-1

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

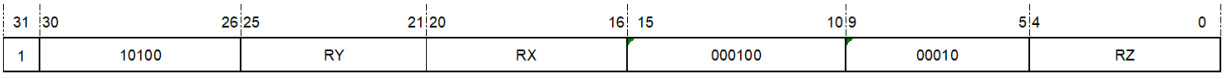


图 13.106: LDR.BS-2

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

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

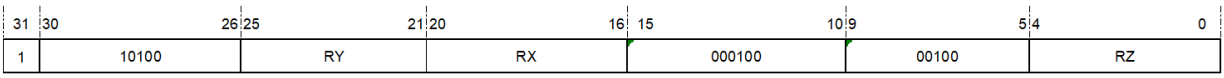


图 13.107: LDR.BS-3

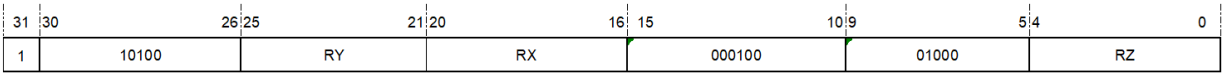


图 13.108: LDR.BS-4

13.74 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)

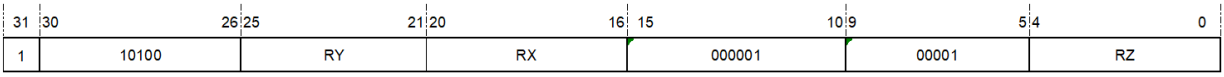


图 13.109: LDR.H-1

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

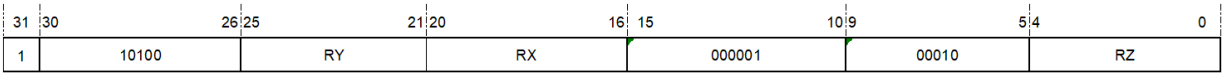


图 13.110: LDR.H-2

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

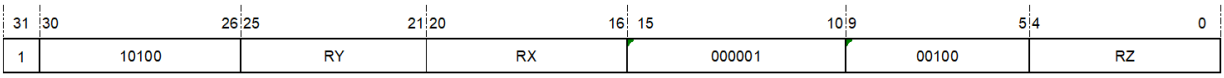


图 13.111: LDR.H-3

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

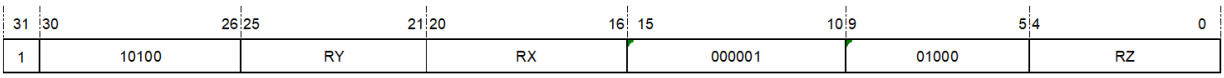


图 13.112: LDR.H-4

13.75 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)

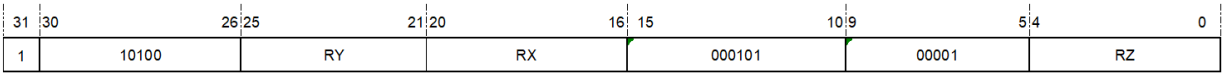


图 13.113: LDR.HS-1

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

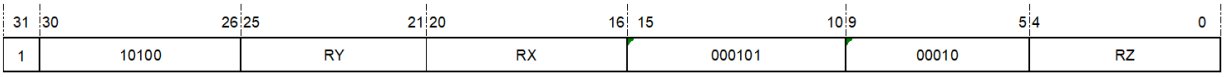


图 13.114: LDR.HS-2

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

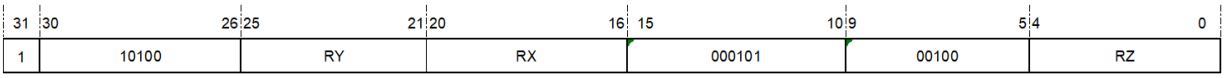


图 13.115: LDR.HS-3

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

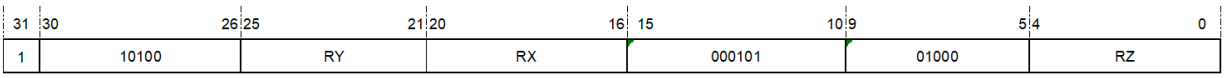


图 13.116: LDR.HS-4

13.76 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)

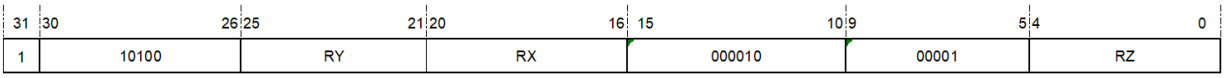


图 13.117: LDR.W-1

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

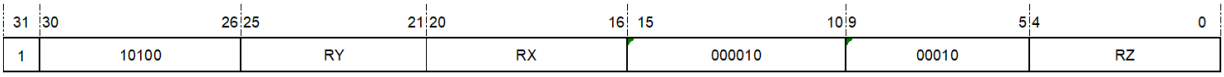


图 13.118: LDR.W-2

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

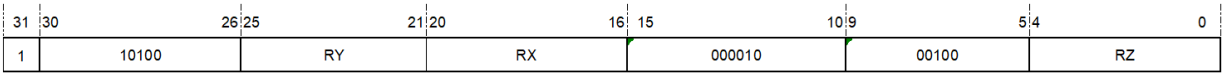


图 13.119: LDR.W-3

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

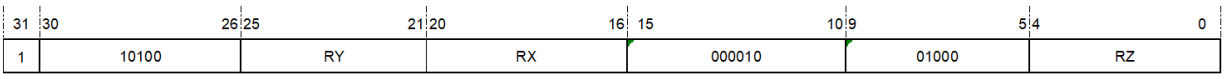


图 13.120: LDR.W-4

13.77 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：

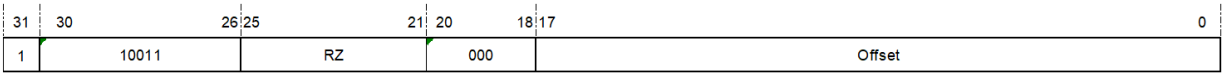


图 13.121: LRS.B

13.78 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：

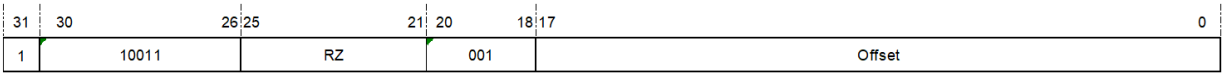


图 13.122: LRS.H

13.79 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：

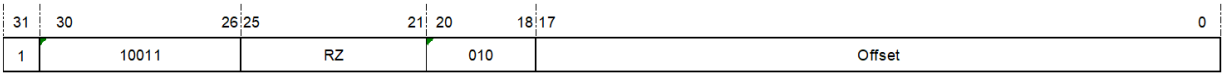


图 13.123: LRS.W

13.80 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：

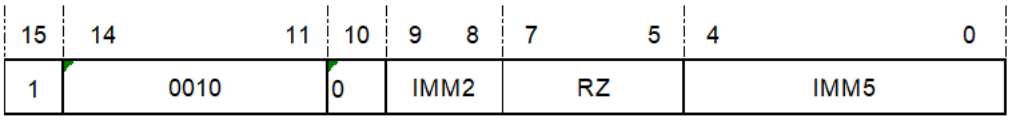


图 13.124: LRW-1

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：

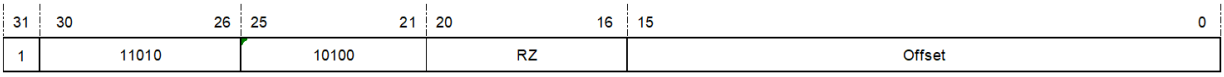


图 13.125: LRW-2

13.81 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：

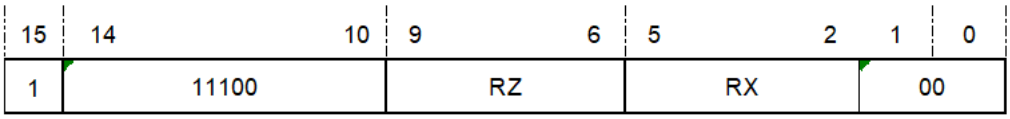


图 13.126: LSL-1

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：

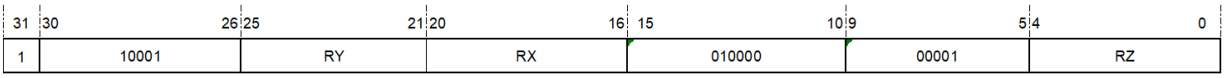


图 13.127: LSL-2

13.82 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：

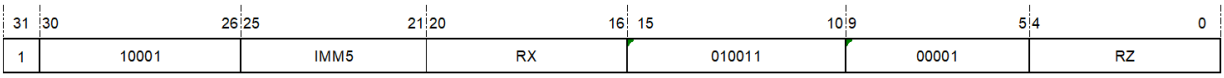


图 13.128: LSLC

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

13.83 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：

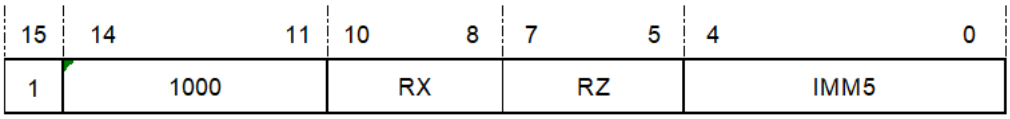


图 13.129: LSLI-1

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：

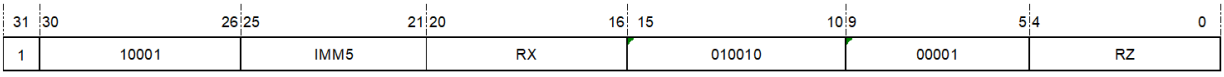


图 13.130: LSLI-2

13.84 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：

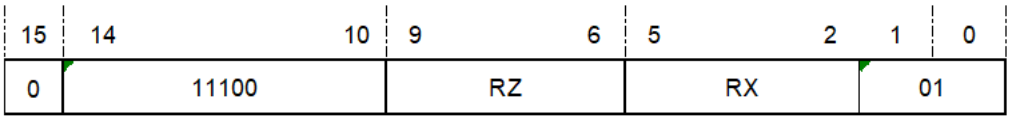


图 13.131: LSR-1

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：

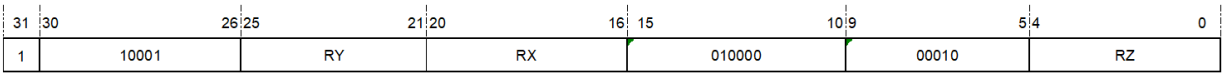


图 13.132: LSR-2

13.85 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：

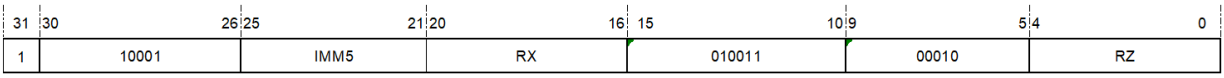


图 13.133: LSRC

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

13.86 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：

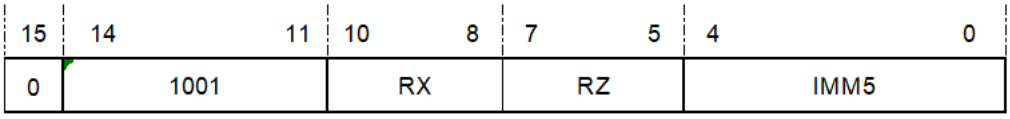


图 13.134: LSRI-1

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：

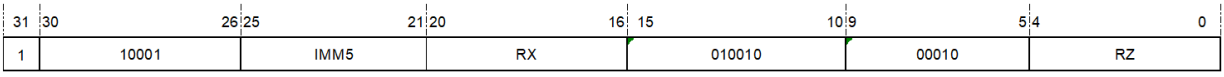


图 13.135: LSRI-2

13.87 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：

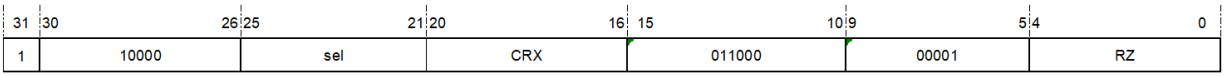


图 13.136: MFCR

13.88 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：

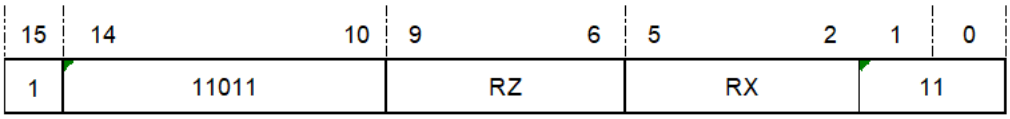


图 13.137: MOV-1

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：

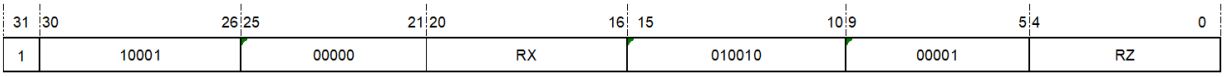


图 13.138: MOV-2

13.89 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：

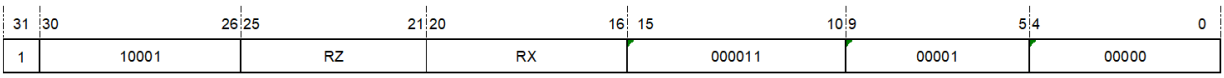


图 13.139: MOVF

13.90 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：

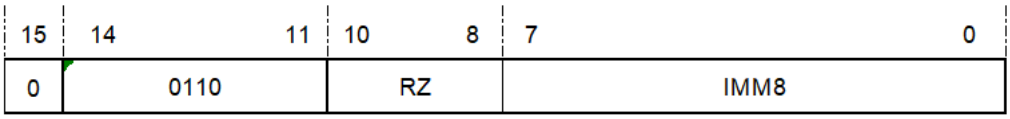


图 13.140: MOVI-1

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：

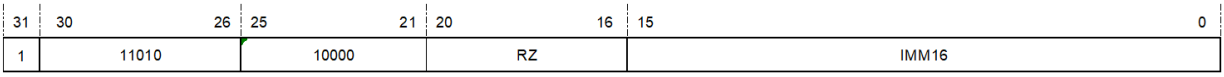


图 13.141: MOVI-2

13.91 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：

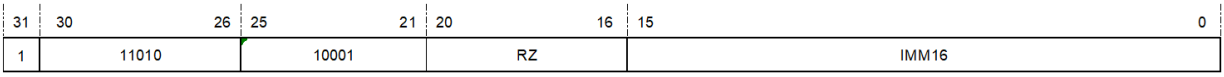


图 13.142: MOVIH

13.92 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：

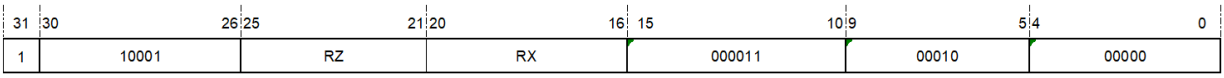


图 13.143: MOVT

13.93 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：

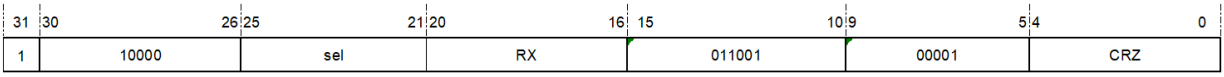


图 13.144: MTCR

13.94 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：

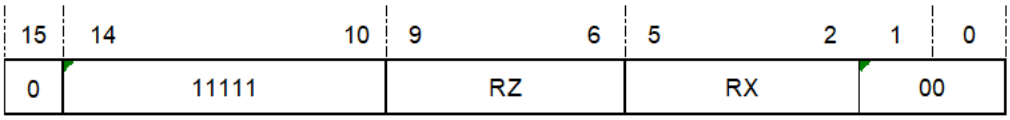


图 13.145: MULT-1

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：

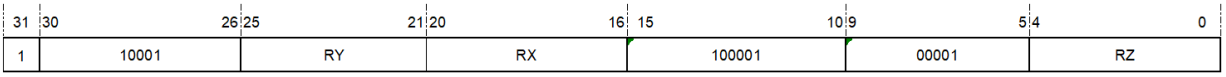


图 13.146: MULT-2

13.95 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：

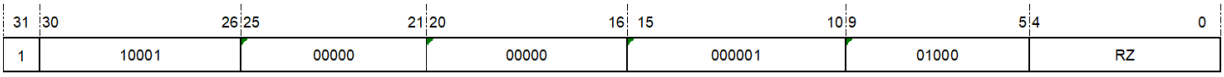


图 13.147: MVC

13.96 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：

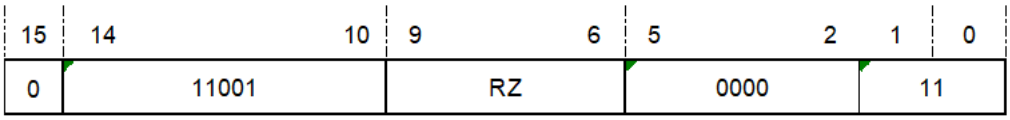


图 13.148: MVCV-1

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：

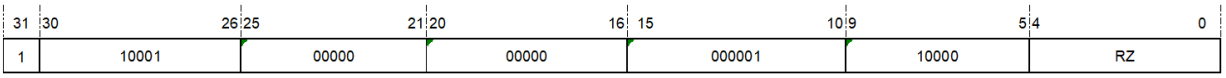


图 13.149: MVCV-2

13.97 NIE——中断嵌套使能指令

统一化指令	
语法	nie
操作	将中断的控制寄存器现场 {EPSR, EPC} 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE; MEM[SP-4] ←EPC; MEM[SP-8] ←EPSR; SP←SP-8; PSR({EE,IE}) ← 1;
编译结果	仅存在 16 位指令 nie;

属性：	特权指令
说明:	将中断的控制寄存器现场 {EPSR, EPC } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PS R.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常：	访问错误异常、未对齐异常、特权违反异常

16 位指令	
操作:	将中断的控制寄存器现场 {EPSR, EPC} 存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE; MEM[SP-4] ←EPC; MEM[SP-8] ←EPSR; SP←SP-8; PSR({EE,IE}) ← 1;
语法:	nie16
属性：	特权指令
说明:	将中断的控制寄存器现场 {EPSR, EPC } 保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PS R.IE 和 PSR.EE。采用堆栈指针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、未对齐访问异常、访问错误异常

指令格式:

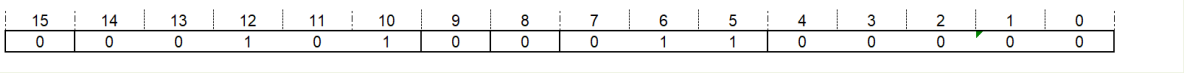


图 13.150: NIE

13.98 NIR——中断嵌套返回指令

统一化指令	
语法	Nir
操作	从堆栈存储器中载入中断的控制寄存器现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回 EPSR←MEM[SP]; EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC;
编译结果	仅存在 16 位指令 nir;

属性:	特权指令
说明:	从堆栈存储器中载入中断的现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。
影响标志位:	无影响
异常:	访问错误异常、未对齐异常、特权违反异常

16 位指令	
操作:	从堆栈存储器中载入中断的控制寄存器现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回 EPSR←MEM[SP]; EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC;
语法:	nir16
属性:	特权指令
说明:	从堆栈存储器中载入中断的现场到 {EPSR, EPC } 中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针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、未对齐访问异常、访问错误异常

指令格式:

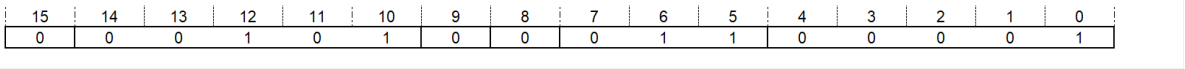


图 13.151: NIR

13.99 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：

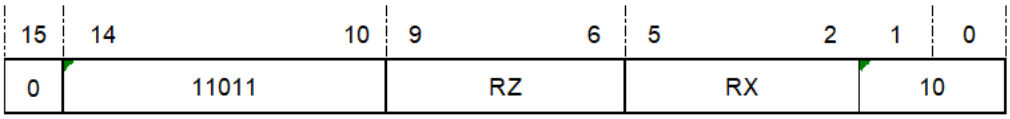


图 13.152: NOR-1

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：

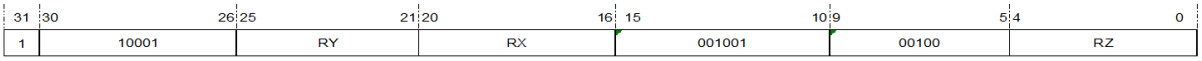


图 13.153: NOR-2

13.100 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：

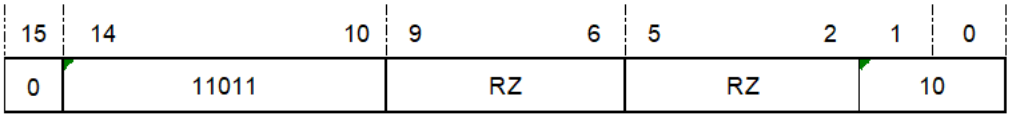


图 13.154: NOT-1

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：

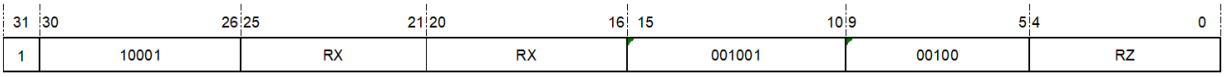


图 13.155: NOT-2

13.101 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：

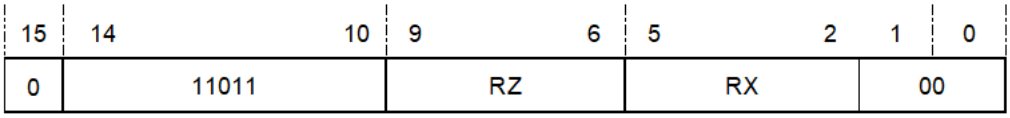


图 13.156: OR-1

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：

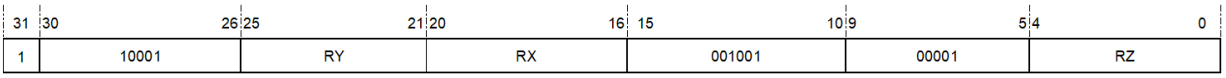


图 13.157: OR-2

13.102 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：

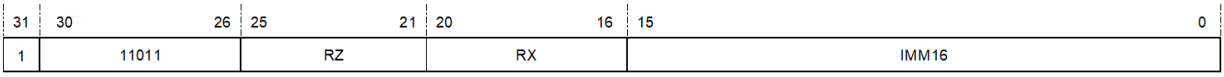


图 13.158: ORI

13.103 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：

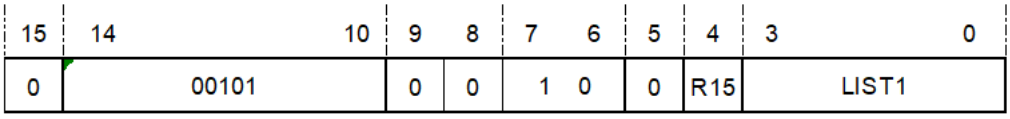


图 13.159: POP-1

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：

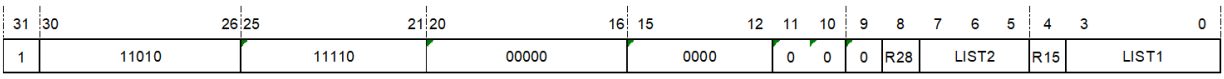


图 13.160: POP-2

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

13.104 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：

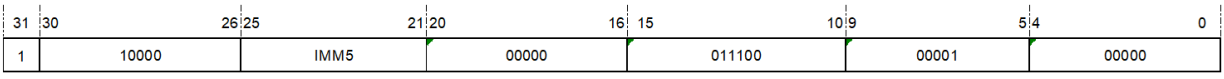


图 13.161: PSRCLR

13.105 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：

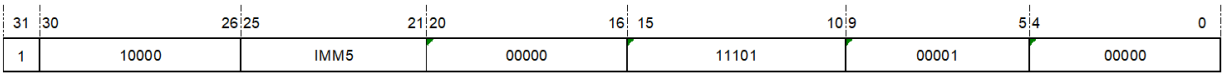


图 13.162: PSRSET

13.106 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：

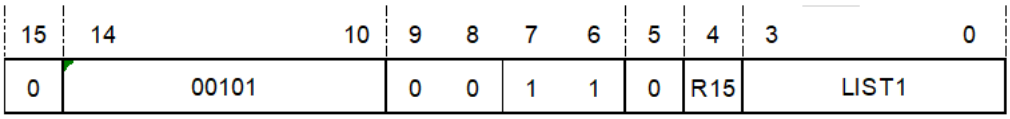


图 13.163: PUSH-1

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：

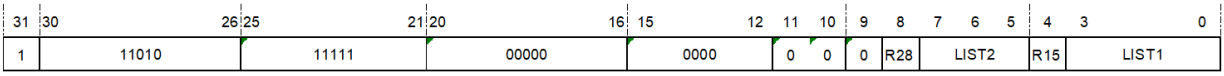


图 13.164: PUSH-2

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

13.107 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：

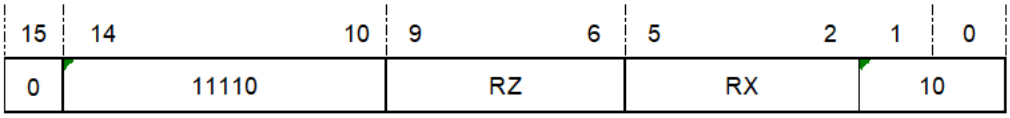


图 13.165: REVB-1

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：

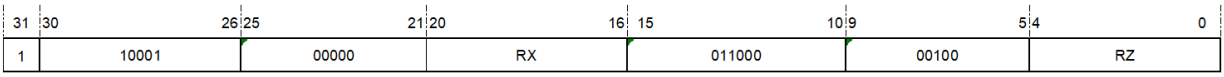


图 13.166: REVB-2

13.108 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：

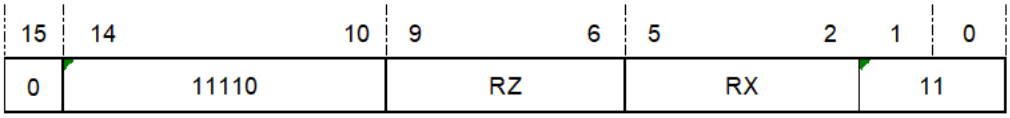


图 13.167: REVH-1

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：

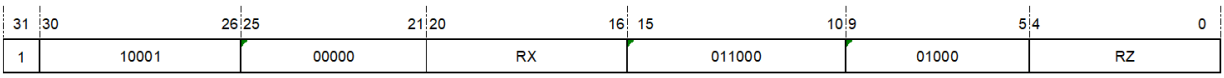


图 13.168: REVH-2

13.109 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：

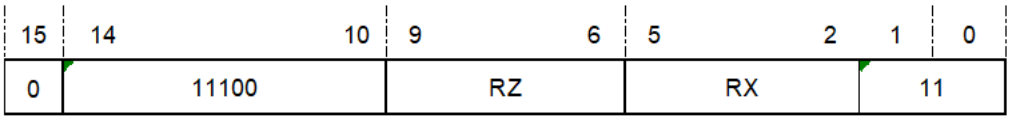


图 13.169: ROTL-1

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：

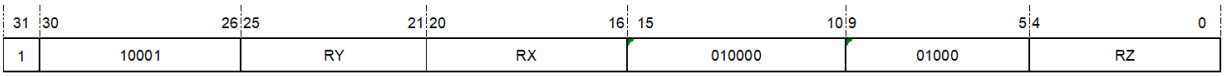


图 13.170: ROTL-2

13.110 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：

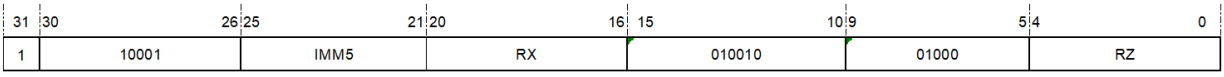


图 13.171: ROTLI

13.111 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：

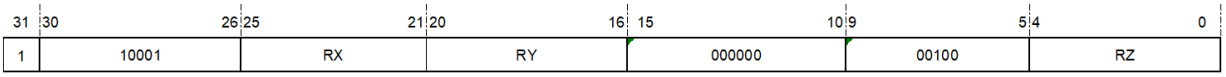


图 13.172: RSUB

13.112 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：

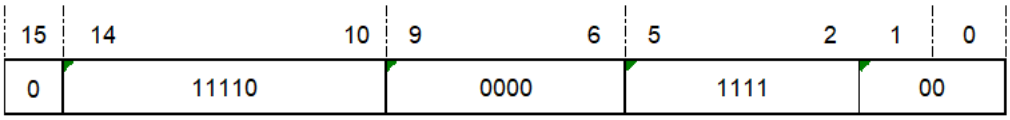


图 13.173: RTS-1

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：

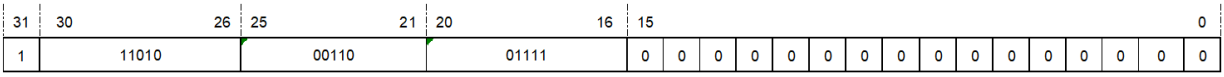


图 13.174: RTS-2

13.113 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：

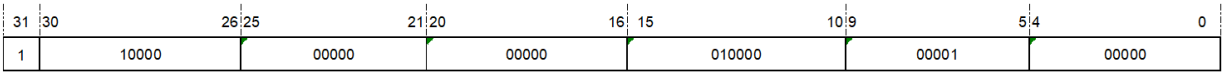


图 13.175: RTE

13.114 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:

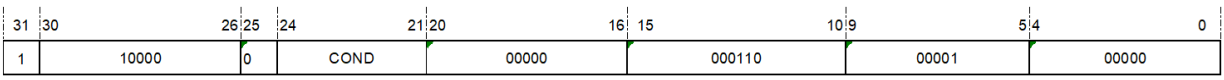


图 13.176: SCE

13.115 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：

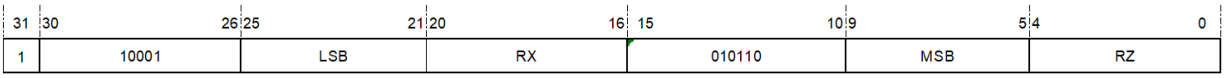


图 13.177: SEXT

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

13.116 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：

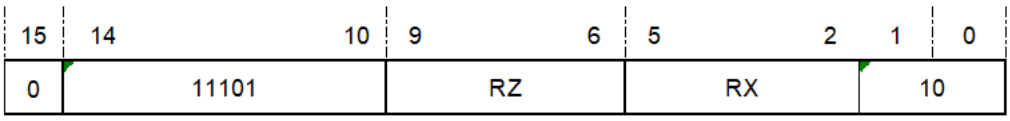


图 13.178: SEXTB-1

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：

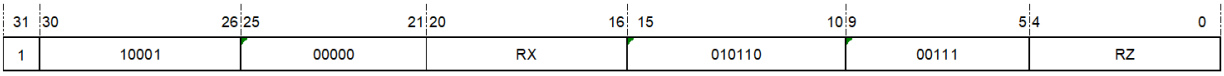


图 13.179: SEXTB-2

13.117 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：

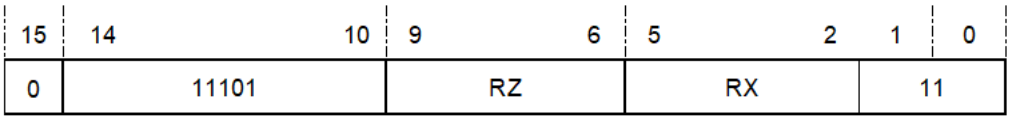


图 13.180: SEXTH-1

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：

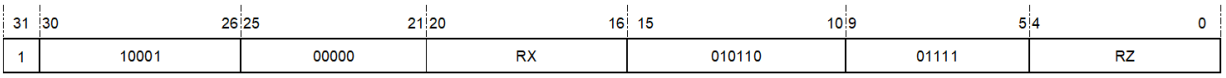


图 13.181: SEXTH-2

13.118 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：

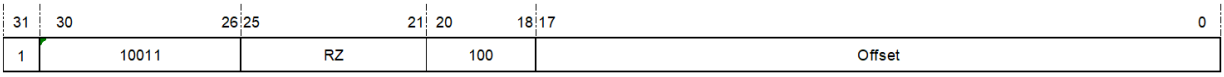


图 13.182: SRS.B

13.119 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：

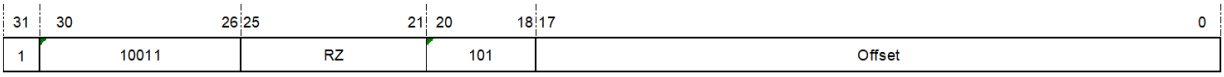


图 13.183: SRS.H

13.120 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：

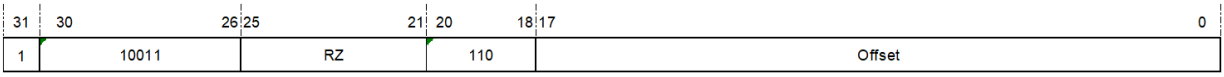


图 13.184: SRS.W

13.121 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：

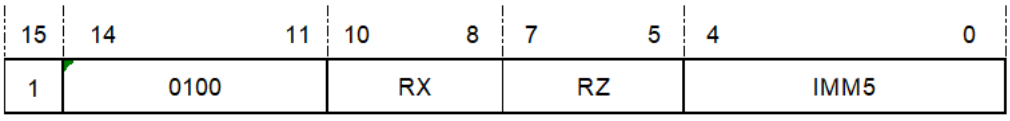


图 13.185: ST.B-1

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：

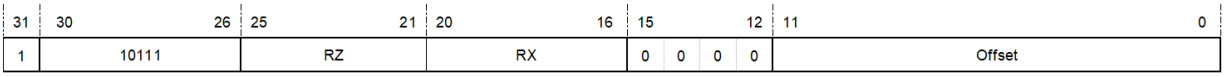


图 13.186: ST.B-2

13.122 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：

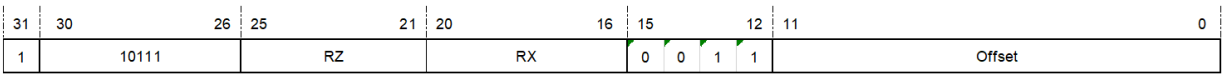


图 13.187: ST.D

13.123 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：

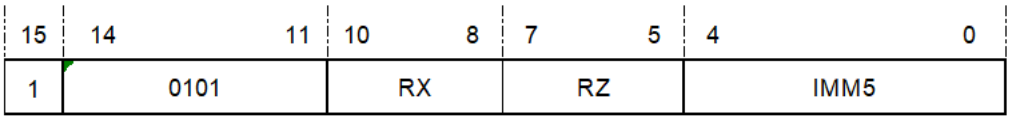


图 13.188: ST.H-1

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：

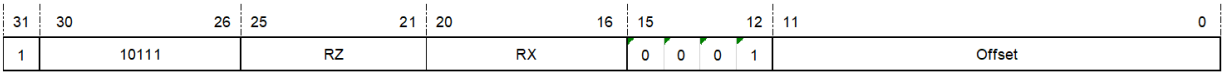


图 13.189: ST.H-2

13.124 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)

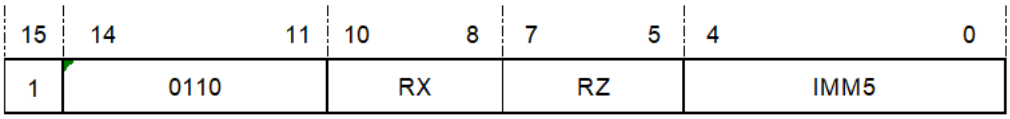


图 13.190: ST.W-1

st16.w rz, (sp, disp)

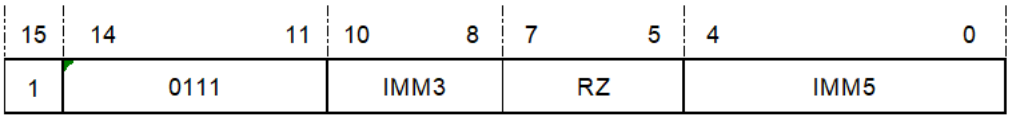


图 13.191: ST.W-2

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：

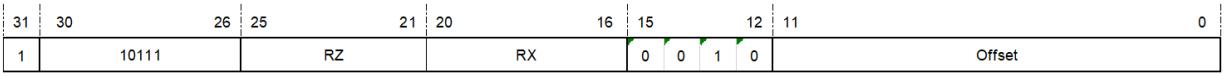


图 13.192: ST.W-3

13.125 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：

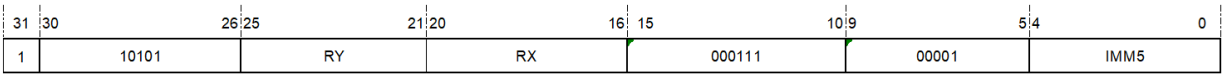


图 13.193: STM

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

13.126 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：

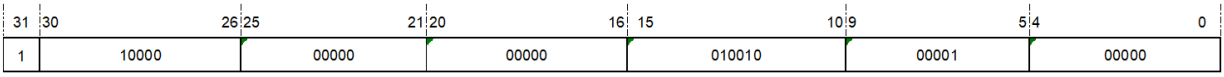


图 13.194: STOP

13.127 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：

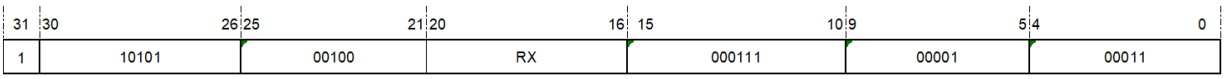


图 13.195: STQ

13.128 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)

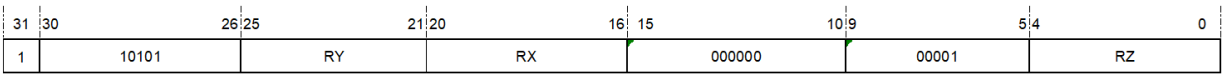


图 13.196: STR.B-1

str32.b rz, (rx, ry << 1)

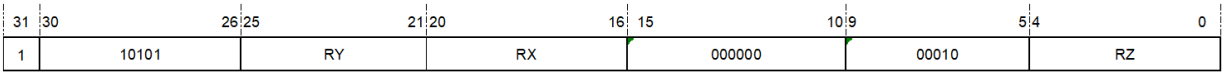


图 13.197: STR.B-2

str32.b rz, (rx, ry << 2)

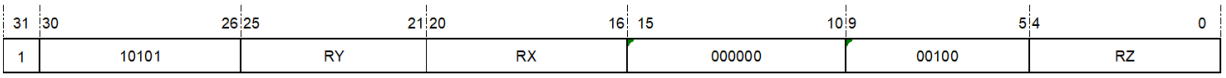


图 13.198: STR.B-3

str32.b rz, (rx, ry << 3)

13.129 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

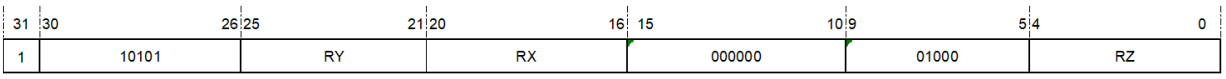


图 13.199: STR.B-4

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)

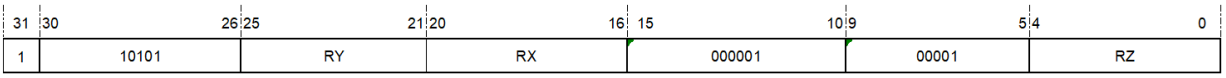


图 13.200: STR.H-1

str32.h rz, (rx, ry << 1)

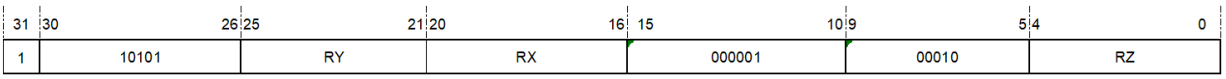


图 13.201: STR.H-2

str32.h rz, (rx, ry << 2)

str32.h rz, (rx, ry << 3)

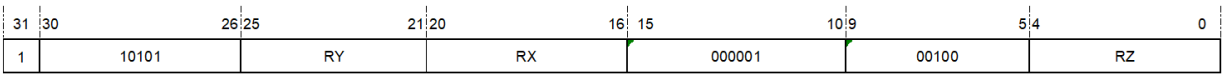


图 13.202: STR.H-3

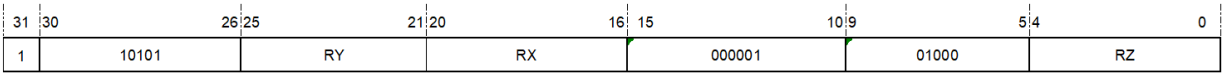


图 13.203: STR.H-4

13.130 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)

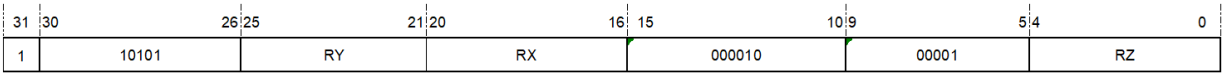


图 13.204: STR.W-1

str32.w rz, (rx, ry << 1)

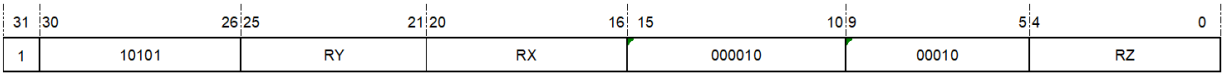


图 13.205: STR.W-2

str32.w rz, (rx, ry << 2)

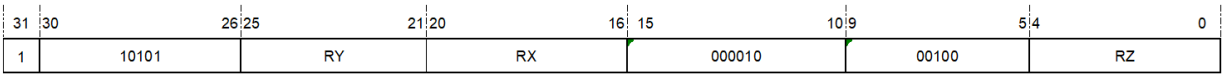


图 13.206: STR.W-3

str32.w rz, (rx, ry << 3)

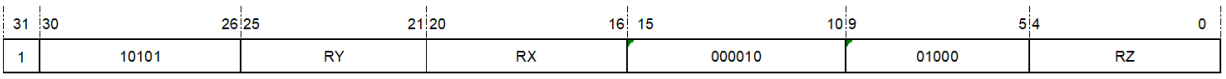


图 13.207: STR.W-4

13.131 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：

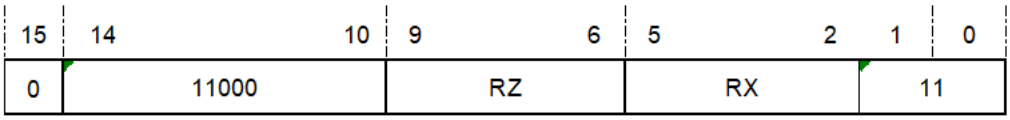


图 13.208: SUBC-1

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：

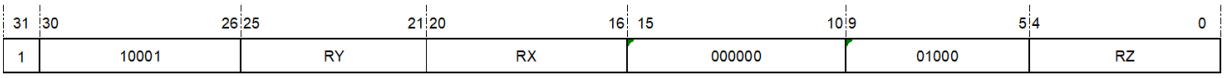


图 13.209: SUBC-2

13.132 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：

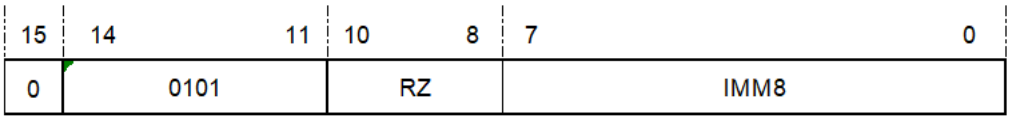


图 13.210: SUBI-1

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：

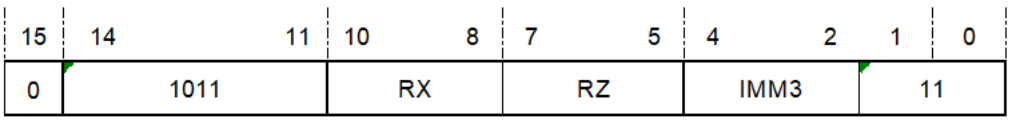


图 13.211: SUBI-2

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

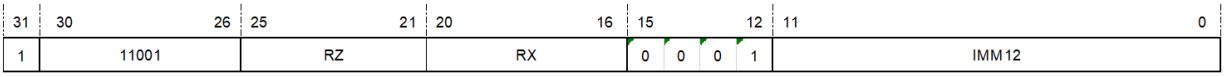
……

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

13.133 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：

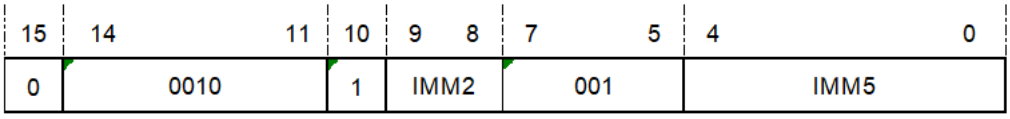


图 13.212: SUBI(SP)

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

13.134 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：

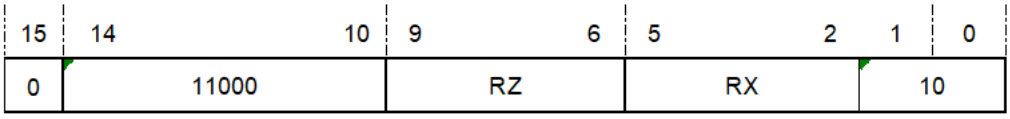


图 13.213: SUBU-1

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：

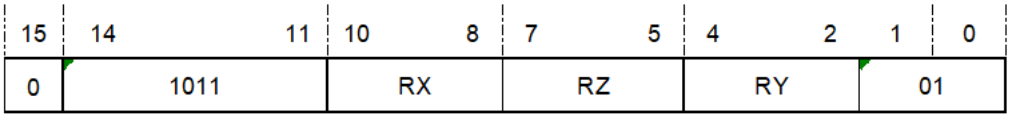


图 13.214: SUBU-2

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：

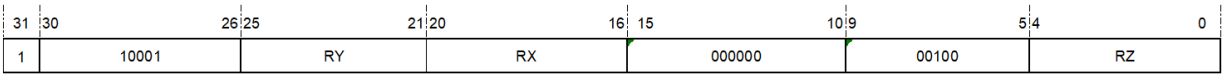


图 13.215: SUBU-3

13.135 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：

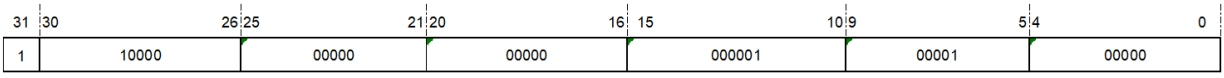


图 13.216: SYNC

13.136 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

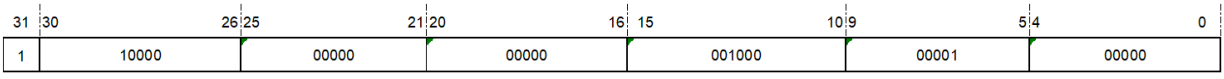


图 13.217: TRAP-1

trap32 1

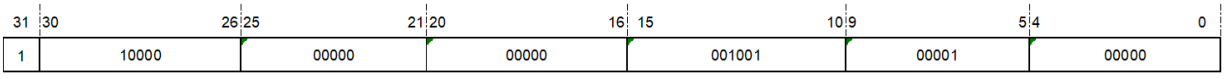


图 13.218: TRAP-2

trap32 2

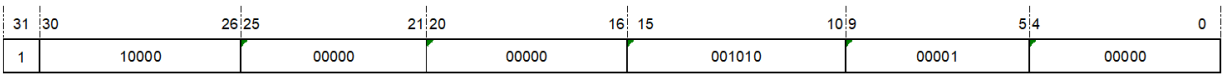


图 13.219: TRAP-3

trap32 3

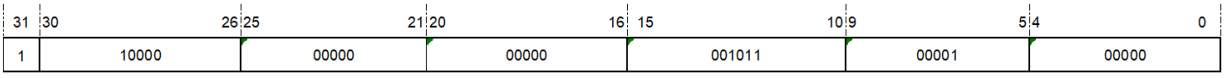


图 13.220: TRAP-4

13.137 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：

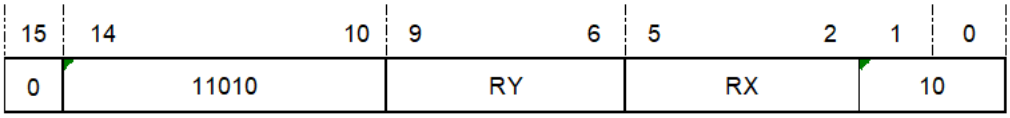


图 13.221: TST-1

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：

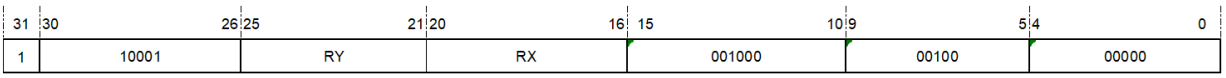


图 13.222: TST-2

13.138 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：

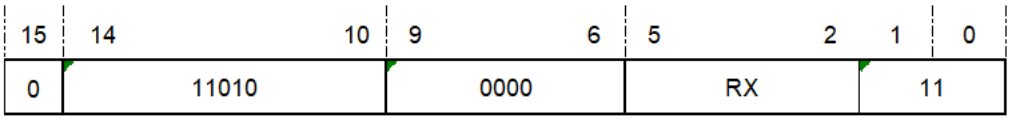


图 13.223: TSTNBZ-1

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：

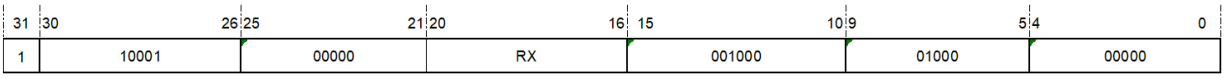


图 13.224: TSTNBZ-2

13.139 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：

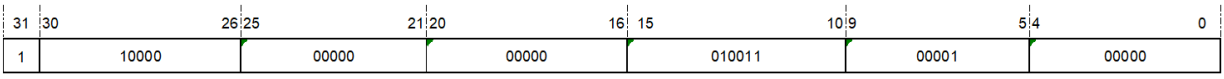


图 13.225: WAIT

13.140 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：

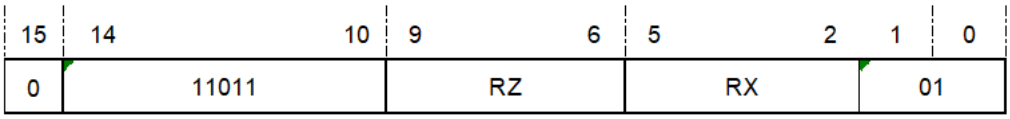


图 13.226: XOR-1

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：

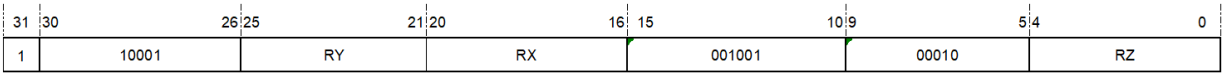


图 13.227: XOR-2

13.141 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：

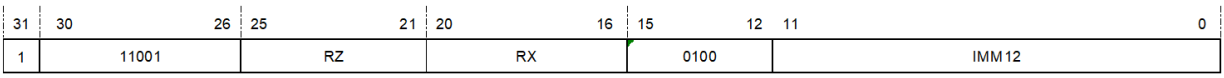


图 13.228: XORI

13.142 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：

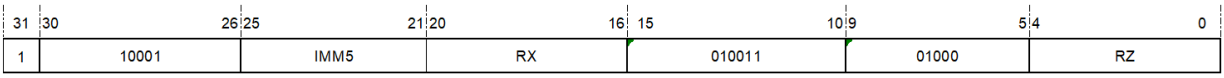


图 13.229: XSR

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

13.143 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：

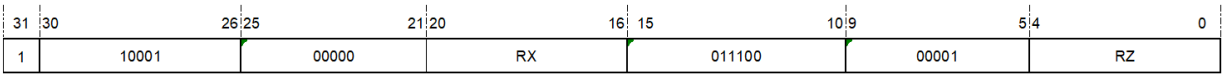


图 13.230: XTRB0

13.144 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：

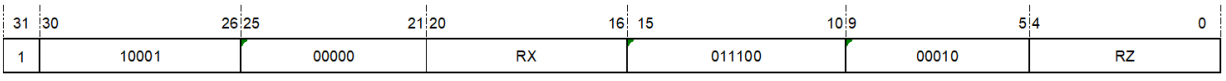


图 13.231: XTRB1

13.145 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：

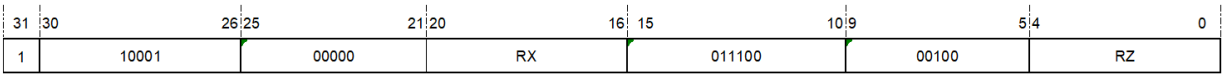


图 13.232: XTRB2

13.146 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：

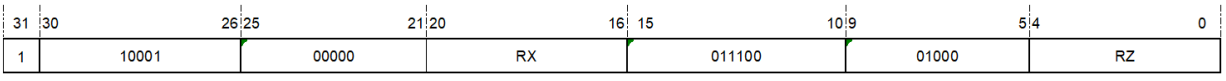


图 13.233: XTRB3

13.147 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：

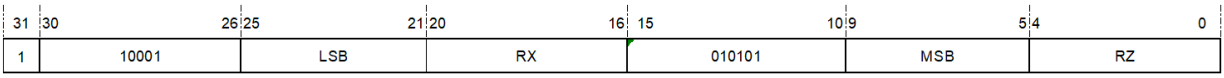


图 13.234: ZEXT

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

13.148 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：

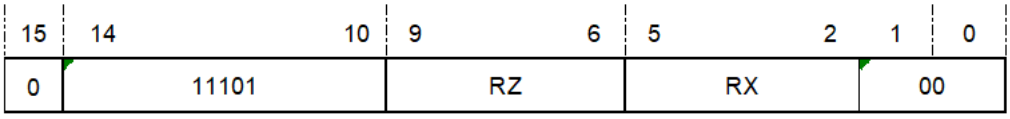


图 13.235: ZEXTB-1

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：

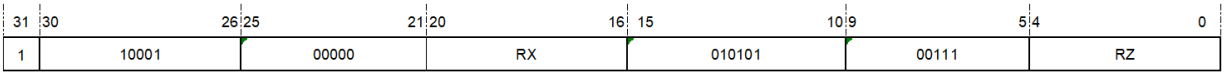


图 13.236: ZEXTB-2

13.149 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：

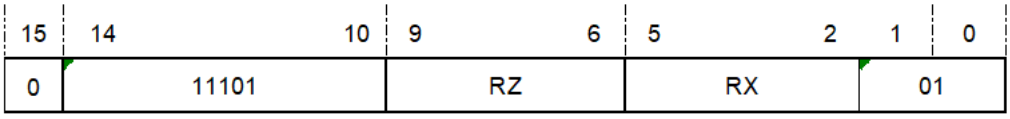


图 13.237: ZEXTH-1

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：

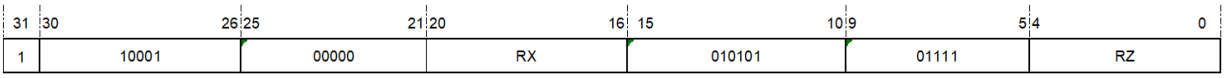


图 13.238: ZEXTH-2

