玄铁 C810 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	2020.08.04
02	更新模板	2021.05.14
03	增加了 cache line 大小的描述	2022.05.25
04	更新模板	2024.03.20

玄铁 C810 用户手册

第一章　简介

C810 是玄铁自主研发的面向高端嵌入式应用的嵌入式 CPU 核产品，它以先进的指令架构与流水线技术在性能和频率等方面达到业界领先水平。C810 基于玄铁 CPU V2 自主指令架构与 16/32 位混合指令编码系统，同时设计有高性能矢量 DSP 运算指令与单精度、双精度浮点指令。硬件上采用先进的 10 级流水线技术与乱序猜测执行框架，具有高主频、高单位性能、高代码密度、高功耗效率等优点。C810 系列嵌入式 CPU 包括针对浮点加强的 C810F 与针对 DSP 运算加强的 C810E。C810 可应用于新一代移动通信设备、下一代高清数字电视机顶盒、移动智能信息终端、汽车电子等高性能嵌入式应用领域。

C810 实现了玄铁 CPU V2 指令架构中除了多核、协处理器扩展等指令之外的所有基本指令。其中 16 位指令集的优势是成本低、代码密度高，但索引、立即数范围较小；32 位指令集的优势是立即数和相对跳转偏移量宽、操作数多、性能强。在实际使用中，编译器会根据编译优化的实际需求，有选择的选用 16 位和 32 位指令混合。用户在使用汇编时，仅需要按照需求书写统一格式的汇编指令，汇编器会根据实际情况选择 16 位或者 32 位指令，指令宽度对用户透明。

1.1　特点

C810 处理器体系结构的主要特点如下：

- RISC 指令架构；
- 32 位数据，32/16 位可变长的指令；
- 双发射超标量 10 级流水线；
- 乱序发射，乱序完成和按序退休；
- 非阻塞发射队列，投机猜测执行；
- 内含 MMU 单元，实现内存保护及管理；
- 哈佛结构，独立的指令和数据总线；
- 2 级多路并行分支预测技术；
- 8 入口硬件返回地址堆栈；
- 512 入口分支目标缓存器；
- 乱序存储载入技术；

- 存储器拷贝加速技术；
- L0 指令缓存降低指令预取功耗；
- 并发操作数前馈机制解决操作数的相关性问题；
- 支持 AMBA AHB/ AXI 总线协议；
- 支持大端模式和小端模式；
- 支持浮点运算与 DSP 矢量运算；
- 内部硬件调试模块支持片上硬件调试。
- 支持快速中断，支持向量中断和自动向量中断；
- 支持写直和写回操作的数据高速缓存；
- 指令高缓和数据高缓大小可变，cache line 大小为 32 bytes；

C810 主要的工艺和性能参数如下：

- 65nm 工艺下，工作频率高于 800MHz（最恶劣情况）；
- 性能大于 2.5 DMIPS/MHz。

1.2 微体系结构

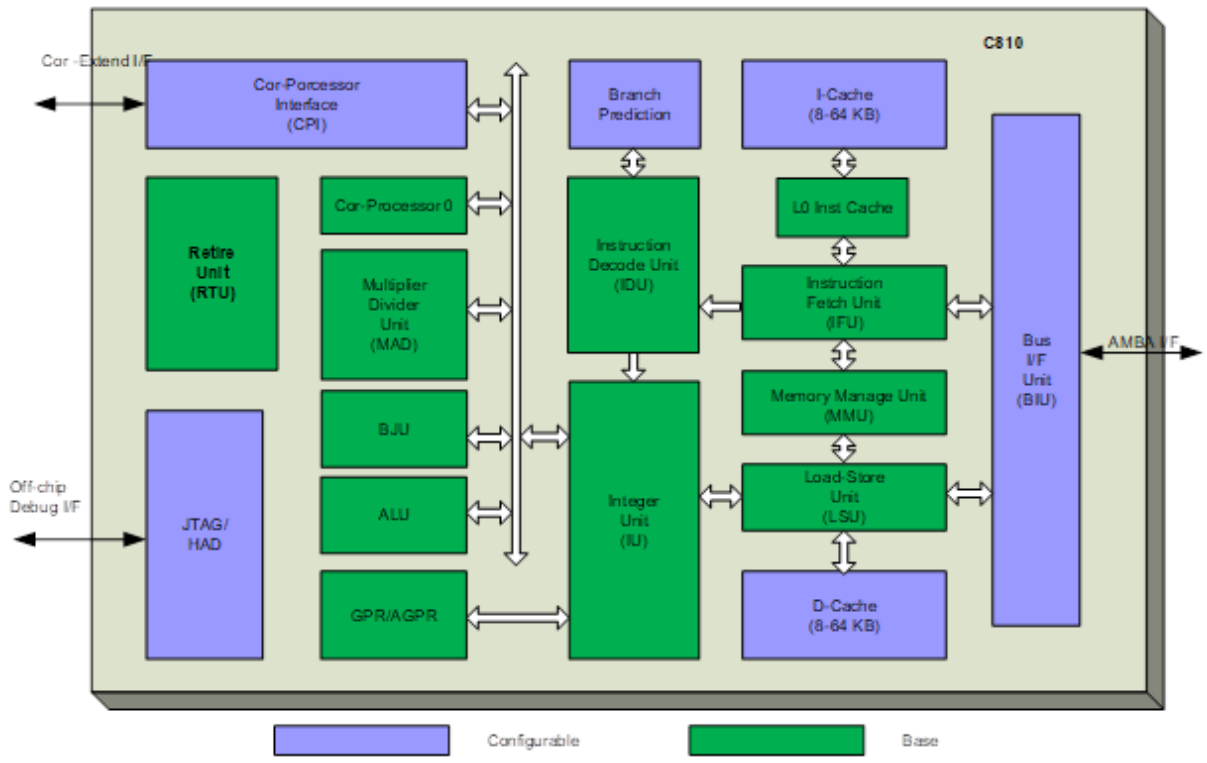


图 1.1: C810 结构图

C810 面向高性能进行流水线体系结构设计优化，设计有：指令 L1 Cache 访问 1、指令 L1 Cache 访问 2、L0 Cache 访问、预解码、解码、寄存器预取、地址计算、数据 L1 Cache 访问第 1 级、数据 L1 Cache 访问第 2 级、结果回写，共 10 级流水线。

指令提取单元，一次可最多提取四条指令并对其并行处理；可以配备高速缓存，在高速缓存缺失时采用关键指令先取和发射，以及后续指令旁路技术；设计一组直接映射的 L0 指令高速缓存器，当指令预取命中该缓存器后，可避免对 4 路组相连的 L1 指令 Cache 的访问提高性能并降低功耗；配备指令暂存器缓存预取指令；采用先进二级指令分支跳转预测，可最多同时预测四条分支指令，预测精度高。整个指令提取单元拥有低功耗，高指令预取效率的特点。

指令译码单元可以同时对两条指令进行译码，并检测出指令间的数据相关性。指令译码单元根据后继流水线执行情况，及时更新指令的数据相关性信息，并将指令乱序发送至下级流水线执行。指令译码单元支持多达 12 条指令的乱序调度，能充分缓解因数据相关性造成的性能损失。除此之外，指令译码单元还能够分解 LDM/STM 等复杂指令，简化执行逻辑。

内存管理单元（MMU），具有 4 表项全相联的数据 µTLB、2 表项全相联的指令 µTLB 和 64/128 表项 2 路组相联 jTLB 及 4 表项全相连 vTLB。µTLB 提升转换速度，对用户透明；jTLB 的大小可配置。如果在 µTLB 中有 TLB 匹配，当前周期从 uTLB 获取物理地址；如果在 µTLB 中没有匹配到 TLB，但在 jTLB 或在 vTLB 中成功 TLB，需要 5 个时钟周期产生物理地址；在 MMU 硬件回填没使能时，如果在 jTLB 和 vTLB 中都存在 TLB 失配，抛出异常并在软件支持下填充 jTLB；如果 MMU 硬件回填使能，jTLB 和 vTLB 产生 TLB 失配，硬件控制逻辑直接访问存储器填充 jTLB（通过 BIU 取回内存中的 TLB 表项）。

存储载入单元支持存储/加载指令的乱序执行，支持高速缓存的非阻塞访问。具有内部前馈机制，消除存储指令回写数据的相关性。支持字节、半字、字和双字的存储/载入指令，并支持字节和半字的载入指令的符号位和 0 扩展。存储/加载指令可以流水执行，使得数据吞吐量达到一个周期存取一个数据。Cache 缺失后，指令被重新排序，按照程序的顺序访问片外存储器。

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

执行单元包含 2 条整型流水线和 1 条存储流水线。整型流水线包含 2 个算术逻辑单元（ALU），1 个乘除法单元 MAD 和 1 个分支跳转单元 BJU。ALU 执行 32 位整数操作，大部分常用指令均为单周期指令，如加减法、移位、逻辑运算等；少部分指令为双周期指令，若比较、提取、插入指令等。ALU 支持快速查找 1/0 算法（FF1/FF0），支持移位加操作（IXH，IXW，IXD）等。一个条件/进位位（C）用于条件测试和多于 32 位的算术逻辑运算。通常，C 位只在显式的测试/比较运算中被改变，而不是正常指令运算的副作用。ALU 通过操作数前馈减少数据真相关，单周期 ALU 指令不存在数据真相关停顿延时。MAD 支持 16*16,16*32,32*32 整数乘法，支持乘累加、乘累减操作。除法器的设计采用了快速算法，执行周期 4~36 不等。BJU 支持快速分支预测错误处理，加速处理器性能。

指令退休单元包括一个 16 表项的重排序缓冲器与一个 16 表项的结果缓冲队列，最多支持 16 条指令的并行乱序执行。其中，重排序缓冲器负责指令的乱序回收与按序退休，结果缓冲队列负责结果的乱序回收与按序回写。通过支持指令并行回收与快速退休提高指令退休效率。指令退休单元每个时钟周期按序并行退休与写回两条指令，保证精确异常，支持普通中断和快速中断。16 个硬件可配置的可选择寄存器用于减少在中断异常处理程序的上下文切换的时间，支持矢量和自动矢量中断。

硬件辅助调试单元（HAD）支持各种调试方式，包括软件设置断点方式、硬件设置断点方式、单步和多

步指令跟踪、跳转指令跟踪等 6 种方式，可以在线调试 CPU、通用寄存器（GPR）、可选择寄存器（AGPR）、协处理器 0（CP0）和内存。

1.3 编程模型

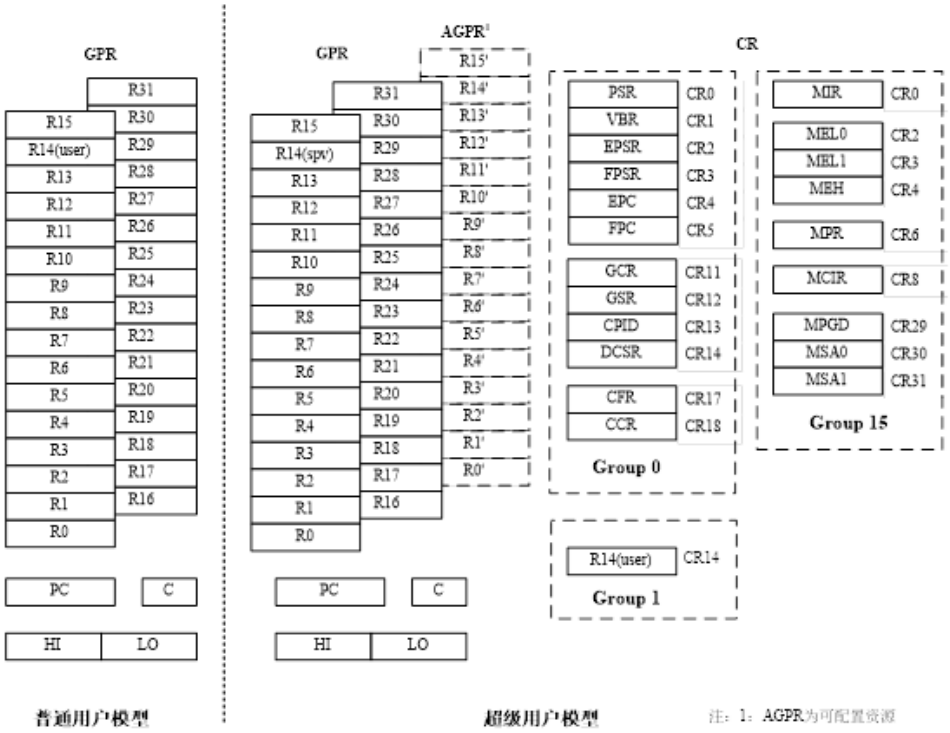


图 1.2: 编程模型

C810 定义了两种编程权限：超级用户模式和普通用户模式。当 PSR 内的 S 位被置位，处理器就在超级用户模式下执行程序。

两种运行模式对应不同的操作权限，区别主要体现在两个方面：1）对寄存器的访问；2）特权指令的使用。

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系统软件则可以访问所有的寄存器，使用控制寄存器来进行超级用户操作。普通用户程序因此就可以避免接触那些特权信息，而操作系统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。

普通用户模式和超级用户模式的 R14 是独立的，在普通用户模式下只能访问 R14（user）；在超级用户模式下，不仅可以访问 R14（spv），还可以访问普通用户的 R14（user）。通常，R14 用于当前的堆栈指针。

C810 可根据不同的应用配置一组可选择通用寄存器 AGPR：R0’ ~R15’。通过 PSR 中的 AF 位来选择访问 GPR 或者 AGPR。当 PSR 中的 AF 位置位时，处理器从可选择通用寄存器组中取数进行运算；当 AF 位被清零时，处理器从通用寄存器组中取数据进行运算。可选择通用寄存器组可用来减少在实时事件处理时的上下文切换时间。

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，psrlcr，rte，rfi。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。

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。异常处理把当前的 PSR 的值存放在 EPSR 或 FPSR 影子控制寄存器中，然后在 PSR 中设置 S 位，强制处理器进入超级用户模式。在异常处理后，为返回到以前的工作模式，系统函数执行 rte(从异常返回) 或 rfi（从快速中断返回），清空流水线，并从中断发生的地方重新取指执行。

普通用户模式下，条件/进位位（C）位于 PSR 的最低位，可以被普通用户指令访问和更改，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操作控制和状态信息的 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 和一套用来节省在快速中断中上下文切换时间的快速中断影子寄存器 FPSR、FPC。超级用户程序还可以利用一个寄存器 VBR 保存中断向量表的基地址、一个全局状态寄存器 GSR 和一个全局控制寄存器 GCR，以及其他相关控制寄存器。

1.4 数据格式

在 C810 处理器中，寄存器内部的值并没有大小端之分，只有有符号和无符号的区别。其格式均为从右至左表示逻辑低位到高位的排布，如下图所示。

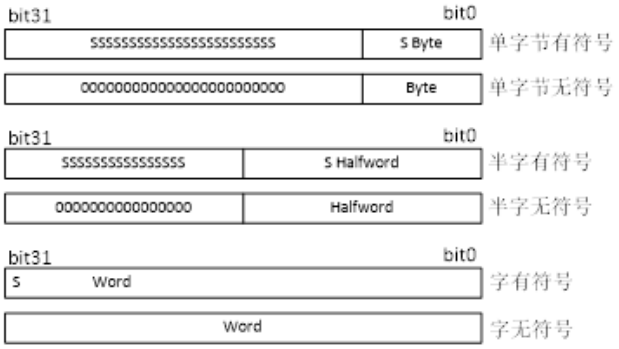


图 1.3: 寄存器中的数据组织结构

大小端的概念是相对于存储器数据存储的格式而提出的，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，如图所示。C810 CPU 在字节和半字访问时有大小端之分，但字访问时无大小端之分（格式相同）。

C810 支持标准补码的 2 进制整数。每个指令操作数的长度可以明确地编码在程序中（load/store 指令），也可以隐含在指令操作中（index operation, byte extraction）。通常，指令接收 32 位操作数，产生 32 位结果。

C810 的存储器可以配置成大端模式或小端模式。在大端模式（缺省模式）下，字 0 的最高位字节放在地址 0 上。而在小端模式下，字 0 的最高位字节放在地址 3 上。在寄存器中，第 31 位是最高位, 习惯上，不管是什么 endian 模式，寄存器的 0 字节代表数据最高字节。这个问题只在执行 xtrb[0-3] 时需要考虑。

图 1.4: 内存中的数据组织形式

1.5 指令集一览

C810 的指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。

图表 表 1.1 列出了 C810 指令集中所有 16 位 32 位指令。

表 1.1: C810 的指令集

汇编指令	32 位	16 位	指令说明
ADDU	√	√	无符号加法指令
ADDC	√	√	无符号带进位加法指令
ADDI	√	√	无符号立即数加法指令
SUBU	√	√	无符号减法指令
SUBC	√	√	无符号带借位减法指令
SUBI	√	√	无符号立即数减法指令
RSUB	√	×	反向减法指令
IXH	√	×	索引半字指令
IXW	√	×	索引字指令
IXD	√	×	索引双字指令
INCF	√	×	C 为 0 立即数加法指令
INCT	√	×	C 为 1 立即数加法指令
DECF	√	×	C 为 0 立即数减法指令
DECT	√	×	C 为 1 立即数减法指令
DECGT	√	×	减法大于零置 C 位指令
DECLT	√	×	减法小于零置 C 位指令
DECNE	√	×	减法不等于零置 C 位指令
AND	√	√	按位与指令
ANDI	√	×	立即数按位与指令
ANDN	√	√	按位非与指令

下页继续

表 1.1 – 续上页

ANDNI	√	×	立即数按位非与指令
OR	√	√	按位或指令
ORI	√	×	立即数按位或指令
XOR	√	√	按位异或指令
XORI	√	×	立即数按位异或指令
NOR	√	√	按位或非指令
NOT	√	√	按位非指令
LSL	√	√	逻辑左移指令
LSLI	√	√	立即数逻辑左移指令
LSLC	√	×	立即数逻辑左移至 C 位指令
LSR	√	√	逻辑右移指令
LSRI	√	√	立即数逻辑右移指令
LSRC	√	×	立即数逻辑右移至 C 位指令
ASR	√	√	算术右移指令
ASRI	√	√	立即数算术右移指令
ASRC	√	×	立即数算术右移至 C 位指令
ROTL	√	√	循环左移指令
ROTLI	√	×	立即数循环左移指令
XSR	√	×	扩展右移指令
CMPNE	√	√	不等比较指令
CMPNEI	√	√	立即数不等比较指令
CMPHS	√	√	无符号大于等于比较指令
CMPHSI	√	√	立即数无符号大于等于比较指令
CMPLT	√	√	有符号小于比较指令
CMPLTI	√	√	立即数有符号小于比较指令
TST	√	√	零测试指令
TSTNBZ	√	√	无字节等于零寄存器测试指令
MOV	√	√	数据传送指令
MOVF	√	×	C 为 0 数据传送指令
MOVT	√	×	C 为 1 数据传送指令
MOVI	√	√	立即数数据传送指令
MOVIH	√	×	立即数高位数据传送指令
LRW	√	√	存储器读入指令
GRS	√	×	符号产生指令
MVCV	√	√	C 位取反传送指令
MVC	√	×	C 位传送指令
CLRF	√	×	C 为 0 清零指令
CLRT	√	×	C 为 1 清零指令

下页继续

表 1.1 – 续上页

BCLRI	√	√	立即数位清零指令
BSETI	√	√	立即数位置位指令
BTSTI	√	×	立即数位测试指令
ZEXT	√	×	位提取并无符号扩展指令
SEXT	√	×	位提取并有符号扩展指令
INS	√	×	位插入指令
ZEXTB	√	√	字节提取并无符号扩展指令
ZEXTH	√	√	半字提取并无符号扩展指令
SEXTB	√	√	字节提取并有符号扩展指令
SEXTH	√	√	半字提取并有符号扩展指令
XTRB0	√	×	提取字节 0 并无符号扩展指令
XTRB1	√	×	提取字节 1 并无符号扩展指令
XTRB2	√	×	提取字节 2 并无符号扩展指令
XTRB3	√	×	提取字节 3 并无符号扩展指令
BREV	√	×	位倒序指令
REVB	√	√	字节倒序指令
REVH	√	√	半字内字节倒序指令
MULT	√	√	乘法指令
MULSH	√	√	16 位有符号乘法指令
DIVU	√	×	无符号除法指令
DIVS	√	×	有符号除法指令
ABS	√	×	绝对值指令
FF0	√	×	快速找 0 指令
FF1	√	×	快速找 1 指令
BMASKI	√	×	立即数位屏蔽产生指令
BGENR	√	×	寄存器位产生指令
BGENI	√	×	立即数位产生指令
BT	√	√	C 为 1 分支指令
BF	√	√	C 为 0 分支指令
BEZ	√	×	寄存器等于零分支指令
BNEZ	√	×	寄存器不等于零分支指令
BHZ	√	×	寄存器大于零分支指令
BLSZ	√	×	寄存器小于等于零分支指令
BLZ	√	×	寄存器小于零分支指令
BHSZ	√	×	寄存器大于等于零分支指令
BR	√	√	无条件跳转指令
BSR	√	√	跳转到子程序指令
JMPI	√	×	间接跳转指令

下页继续

表 1.1 – 续上页

JSRI	√	×	间接跳转到子程序指令
JMP	√	√	寄存器跳转指令
JSR	√	√	寄存器跳转到子程序指令
RTS	√	√	链接寄存器跳转指令
LD.B	√	√	无符号扩展字节加载指令
LD.BS	√	×	有符号扩展字节加载指令
LD.H	√	√	无符号扩展半字加载指令
LD.HS	√	×	有符号扩展半字加载指令
LD.W	√	√	字加载指令
LD.D	√	×	双字加载指令
ST.B	√	√	字节存储指令
ST.H	√	√	半字存储指令
ST.W	√	√	字存储指令
ST.D	√	×	双字存储指令
LDR.B	√	×	寄存器移位寻址无符号扩展字节加载指令
LDR.BS	√	×	寄存器移位寻址有符号扩展字节加载指令
LDR.H	√	×	寄存器移位寻址无符号扩展半字加载指令
LDR.HS	√	×	寄存器移位寻址有符号扩展半字加载指令
LDR.W	√	×	寄存器移位寻址字加载指令
STR.B	√	×	寄存器移位寻址字节存储指令
STR.H	√	×	寄存器移位寻址半字存储指令
STR.W	√	×	寄存器移位寻址字存储指令
LRS.B	√	×	字节符号加载指令
LRS.H	√	×	半字符号加载指令
LRS.W	√	×	字符号加载指令
SRS.B	√	×	字节符号存储指令
SRS.H	√	×	半字符号存储指令
SRS.W	√	×	字符号存储指令
LDQ	√	×	连续四字加载指令
LDM	√	×	连续多字加载指令
STQ	√	×	连续四字存储指令
STM	√	×	连续多字存储指令
POP	√	√	出栈指令
PUSH	√	√	压栈指令
MFCR	√	×	控制寄存器读传送指令
MTCR	√	×	控制寄存器写传送指令
PSRSET	√	×	PSR 位置位指令
PSRCLR	√	×	PSR 位清零指令

下页继续

表 1.1 – 续上页

WAIT	√	×	进入低功耗等待模式指令
DOZE	√	×	进入低功耗睡眠模式指令
STOP	√	×	进入低功耗暂停模式指令
RTE	√	×	异常和普通中断返回指令
RFI	√	×	快速中断返回指令
SYNC	√	×	CPU 同步指令
BKPT	×	√	断点指令
SCE	√	×	条件执行设置指令
IDLY	√	×	中断识别禁止指令
TRAP	√	×	无条件操作系统陷阱指令
PLDR	√	×	读数据预取指令
PLDW	√	×	写数据预取指令
*MTHI	√	×	累加器高位写传送指令
*MTLO	√	×	累加器低位写传送指令
*MFHI	√	×	累加器高位读传送指令
*MFLO	√	×	累加器低位读传送指令
*MVTC	√	×	溢出位复制到 C 位指令
*MULU	√	×	无符号数乘法指令
*MULS	√	×	有符号数乘法指令
*MULUA	√	×	无符号数乘累加指令
*MULSA	√	×	有符号数乘累加指令
*MULUS	√	×	无符号数乘累减指令
*MULSS	√	×	有符号数乘累减指令

注意: √ 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。

第二章　命名规则

2.1　符号

本文档用到的标准符号和操作符如图所示。

符号	功能
+	加
−	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
1/4	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 2.1: 符号

2.2 术语

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

第三章　寄存器描述

本章主要介绍 C810 通用寄存器和控制寄存器在普通用户模式和超级用户模式下的组织结构。

3.1　普通用户编程模式

图表 3.1 列出了普通用户编程模式下的一些寄存器：

- 32 个 32 位通用寄存器（R31~R0）；
- 32 位程序计数器（PC）；
- 条件码 / 进位标志位（C bit）。

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)	堆栈指针（普通用户编程模式）
R15	链接寄存器
R16	不确定

下页继续

表 3.1 – 续上页

名称	功能
R17	不确定
R18	不确定
R19	不确定
R20	不确定
R21	不确定
R22	不确定
R23	不确定
R24	不确定
R25	不确定
R26	不确定
R27	不确定
R28	不确定
R29	不确定
R30	不确定
R31	线程本地储存寄存器

表 3.2: 普通用户编程模式寄存器 (表二)

PC	程序计数器
PSR C 位	参见 处理器状态寄存器（*PSR*，*CR<0,0>*）

3.1.1 通用寄存器

这些通用寄存器包含了指令操作数和结果以及地址信息。硬软件上惯例就是用这些通用寄存器来做子程序的链接调用，参数传递以及堆栈指针。

其中用于堆栈指针的寄存器 R14(user) 为普通用户编程模式下的寄存器，索引方式与其余通用寄存器相同。

3.1.2 程序计数器

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

3.1.3 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清楚地被设置，或者作为另一些高精度算术或逻辑指令的结果而不确定地被设置。另外，特殊的指令如 DEC[GT,LT,NE]，以及 XTRB[0-3] 等将会影响条件码 / 进位标志位的值。

3.2 超级用户编程模式

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

超级用户编程模式由普通用户下的那些寄存器和以下寄存器组成，如图表 3-2 所示：

- 1 个超级用户编程模式通用寄存器（R14(spv)）
- 16 个 32 位可选择寄存器 *；
- 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)；
- 快速中断保留程序计数器 (FPC)；
- 快速中断保留处理器状态寄存器 (FPSR)；
- 32 位全控制寄存器 (GCR)；
- 32 位全状态寄存器 (GSR)；
- 产品序号寄存器 (CPIDR)；
- DSP 控制状态寄存器（DCSR）；
- 高速缓存功能设置寄存器（CFR）；
- 高速缓存配置寄存器 (CCR)；
- MMU 索引寄存器（MIR）；
- MMU EntryLo0 寄存器（MEL0）；
- MMU EntryLo1 寄存器（MEL1）；
- MMU EntryHi/BAD VPN 寄存器（MEH）；
- MMU 页掩码寄存器（MPR）；
- MMU 控制指令寄存器（MCIR）；
- MMU PGD 基址寄存器（MPGD）；
- MMU SSEG0 配置寄存器（MSA0）；

- MMU SSEG1 配置寄存器（MSA1）。

注解： 注：可选择寄存器仅在特定配置时有效。

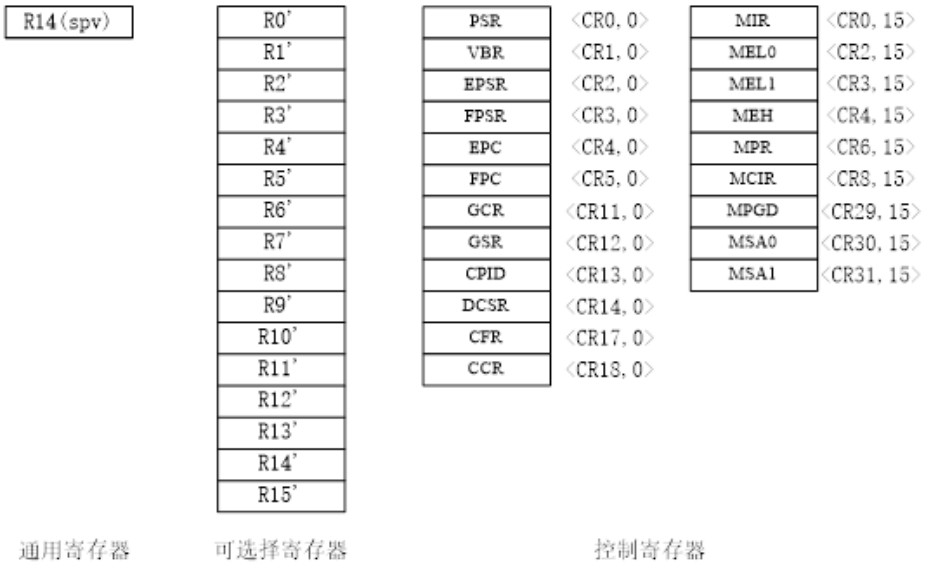


图 3.1: 超级用户编程模式附加资源

3.2.1 超级用户编程模式的通用寄存器 R14(spv)

在超级用户编程模式下，对通用寄存器 R14 的索引将会默认使用超级用户编程模式的寄存器 R14(spv)，这个通用寄存器用于超级用户编程模式的堆栈指针。

此时无法直接访问普通用户编程模式的通用寄存器 R14(user)。若要在超级用户模式下访问 R14(user)，可通过 mfcr/mtcr 访问 CR<14,1> 完成。

3.2.2 可选择寄存器

在特定配置下，超级用户编程模式下的附加资源包含 16 个可选择寄存器。在转向时间关键任务时可选择寄存器可用来减少因当前内容转换和保护现场引起的响应延迟时间。当 PSR（AF）为 1，则可选择寄存器被选中，所有的指令以前使用通用寄存器 R0~R15 现在都将使用可选择寄存器 R0’~R15’。反之，若 PSR（AF）为 0，则可选择寄存器不被选中。一些重要的参数和指针值可以保存在这些可选择寄存器中，只要在优先级高的任务执行时可选择寄存器被选中，这些重要的数据就可用了。

在异常出现并执行异常服务程序时，异常服务程序矢量入口值的低位将被拷贝到 AF 位中去以用来选择哪一组寄存器。

通用寄存器 R16~R31 没有对应的可选择寄存器。

3.2.3 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。在超级用户编程模式下，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），以及通用寄存器或者可选择寄存器 *（AF 位）。它们同样也指出了异常保留寄存器是否可用来保存当前相应的内容，以及中断申请是否有效。

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

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

图 3.2: 处理器状态寄存器

S-超级用户模式设置位：

当 S 为 0 时，处理器工作在普通用户模式；

当 S 为 1 时，处理器工作在超级用户模式。

S 位在被 reset 和进入异常处理时由硬件置 1。

VEC[7:0]-异常事件向量值：

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

TM[1:0]-跟踪模式位：

在指令跟踪模式下，每一条指令执行完后，C810 都将会进入跟踪异常服务程序；在跳转跟踪模式下，当碰到内含有跳转（不管是跳转还是不跳转）的指令执行完，C810 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 3.3 表示出 TM［1:0］编码与相对应的工作模式。

表 3.3: TM[1:0] 编码与相对应的工作模式

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

跟踪模式具体的操作请参考异常处理 。

TE-传输控制位：

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

MM-不对齐异常掩盖位：

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

当 MM 为 1 时，读取或存储的地址不对齐异常将会被掩盖，支持内存的不对齐访问。

该位不能掩盖 jmpi 或 jsri、ldm、stm、ldq、stq、push、pop 等指令的不对齐异常的发生，该位也不能掩盖 strong order 区的不对齐异常的发生，且该位不受其它异常影响，但会被 reset 清零。

矢量读取或者存储指令，size 为四字时，按照双字来判断是否非对齐。同时会受到 MM 位影响。

EE-异常有效控制位：

当 EE 为 0 时，异常无效，此时除了普通中断和快速中断之外的任何异常一旦发生，都会被 C810 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-中断控制位：

当 IC 为 0 时，中断只能在指令之间被响应；

当 IC 为 1 时，表明中断可在长时间、多周期的指令执行完之前被响应；

会被 reset 清零，不受其它异常影响。

IE-中断有效控制位：

当 IE 为 0 时，中断无效，即 pad_biu_int_b 不起作用，EPC 和 EPSR 都无效；

当 IE 为 1 时，中断有效，即 pad_biu_int_b 起作用；

该位会被 reset 清零，也在进入异常服务程序时被清零。

FE-快速中断有效控制位：

当 FE 为 0 时，快速中断无效，即 pad_biu_fint_b 不起作用，FPC 和 FPSR 都无效；

当 FE 为 1（不必考虑 EE 位）时，快速中断有效，即 pad_biu_fint_b 起作用；

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

AF-可选择寄存器有效控制位 *：

当 AF 为 0 时，通用寄存器被选中，可选择寄存器不被选中；

当 AF 为 1 时，通用寄存器不被选中，可选择寄存器被选中；

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

注解: 注：AF 位仅在配置可选择寄存器时有效，否则该位保留为零。

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 或 FPSR 之后不确定。

3.2.3.1 更新 PSR

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

- 异常响应和异常处理更新 PSR：

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

- RTE 和 RFI 指令更新 PSR：

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

- MTCR 指令更新 PSR：

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

- PSRCLR、PSRSET 指令更新 PSR：

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

3.2.4 向量基址寄存器（VBR，CR<1,0>）

VBR 寄存器用来保存异常向量的基址。该寄存器包含 22 个高位有效位，10 个保留位（其值为 0）。VBR 的复位值为 0X00000000。

3.2.5 异常保留寄存器（CR<2,0> ~ CR<5,0>）

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

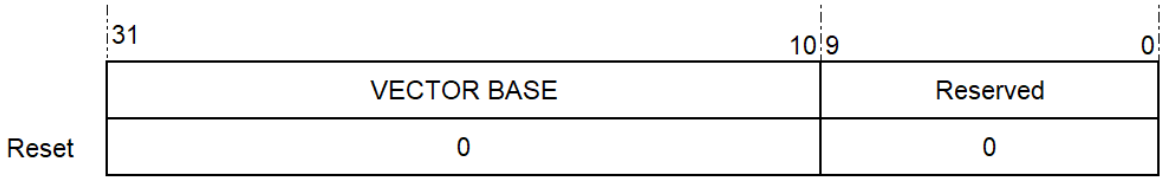


图 3.3: 基址向量寄存器

3.2.6 存储寄存器 SS0~SS4（CR<6,0> ~ CR<10,0>）

C810 系列包含了 5 个 32 位超级用户模式下的存储寄存器，超级用户模式可以使用这些寄存器来存储数据、指针等，用于辅助异常处理以及避免受普通用户模式影响。软件确定了这些寄存器的使用的功能和内容。所有的寄存器可以通过 mfcr 和 mtcr 指令来访问或者修改其中的内容。

3.2.7 全局控制寄存器（GCR，CR<11,0>）

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

3.2.8 全局状态寄存器（GSR，CR<12,0>）

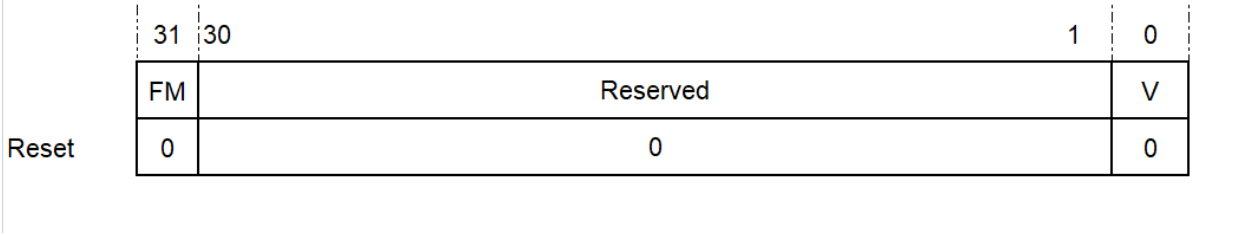
12 位的全局状态寄存器是用来标记外围设备和事件的。它通过芯片口上提供的 12 位平行输入接口将外部状态送入到 C810 内部，从而实现监测。一般来说，可以通过查看 GSR 来检测外围设备状态和事件。全状态寄存器是只读的。

3.2.9 产品序号寄存器（CPUIDRR，CR<13,0>）

该寄存器用于存放玄铁 CPU 产品的内部编号。产品序号寄存器是只读的，其复位值由产品本身决定。

3.2.10 DSP 控制状态寄存器（DCSR，CR<14,0>）

C810 提供一个额外的控制状态寄存器来控制 C810 所支持的 DSP 指令的执行，同时用来表征这些 DSP 指令执行结果的是否出现溢出。



	31	30 ... 1	0
	FM	Reserved	V
Reset	0	0	0

图 3.4: DSP 控制状态寄存器

FM-小数运算模式：

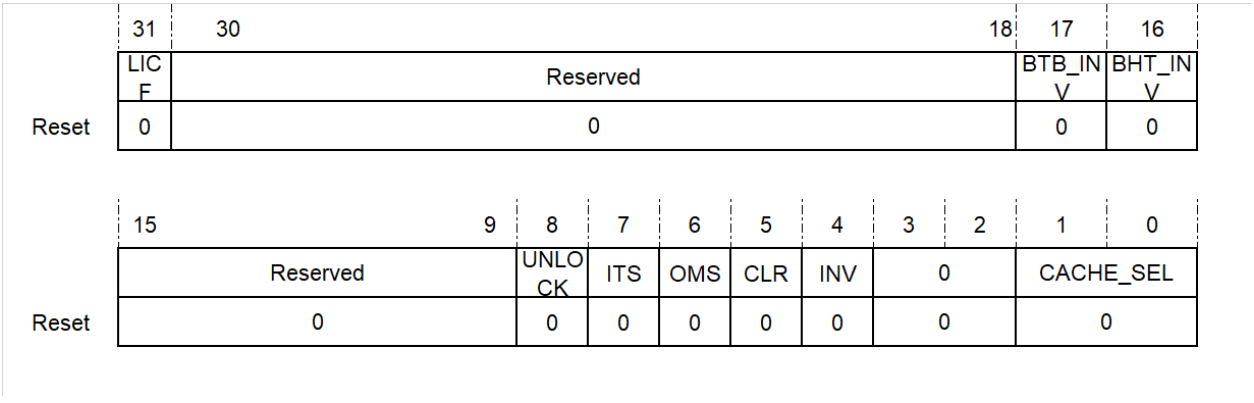
当 FM 位被设置成 1 时，所有的乘法运算被认为是小数运算。

V-溢出状态位：

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

3.2.11 高速缓存功能设置寄存器（CFR，CR<17,0>）

高速缓存功能设置寄存器用来控制高速缓存，让相应高速缓存内的数据全部无效。



	31	30 ... 18	17	16
	LICF	Reserved	BTB_INV	BHT_INV
Reset	0	0	0	0

	15 ... 9	8	7	6	5	4	3 ... 2	1 ... 0
	Reserved	UNLOCK	ITS	OMS	CLR	INV	0	CACHE_SEL
Reset	0	0	0	0	0	0	0	0

图 3.5: 高速缓存功能设置寄存器

LICF –缓存行 INV/CLR 失败状态位：

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

BTB_INV-BTB 无效设置位：

当 BTB_INV 为 1 时，分支目标缓冲器内的数据将失效。

BHT_INV-BHT 无效设置位：

当 BHT_INV 为 1 时，分支历史表内的数据将失效。

UNLOCK-数据高速缓存解锁设置位：

当 UNLOCK 为 1 时，数据高速缓存中的缓存行被解锁，允许缓存行被替换。

OMS–Cache INV/CLR 模式位：

当 OMS 为 0 时，高速缓存的 CLR 和 INV 操作将对高速缓存的所有缓存行起作用。

当 OMS 为 1 时，高速缓存的 CLR 和 INV 操作将对高速缓存的一个缓存行起作用。并且使用 CIR 作为 cache INV/CLR 操作的索引。

ITS–Cache INV/CLR 模式位：

当 ITS 为 0 时，CIR 将被用于虚拟地址索引。

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

CLR-脏表项清除设置位：

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

INV-无效设置位：

当 INV 为 1 时，高速缓存内的数据将失效。

CACHE_SEL-高速缓存选择位：

当 SEL 为 01 时，选中指令高速缓存；

当 SEL 为 10 时，选中数据高速缓存；

当 SEL 为 11 时，选中数据和指令高速缓存

3.2.12 高速缓存配置寄存器（CCR，CR<18,0>）

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

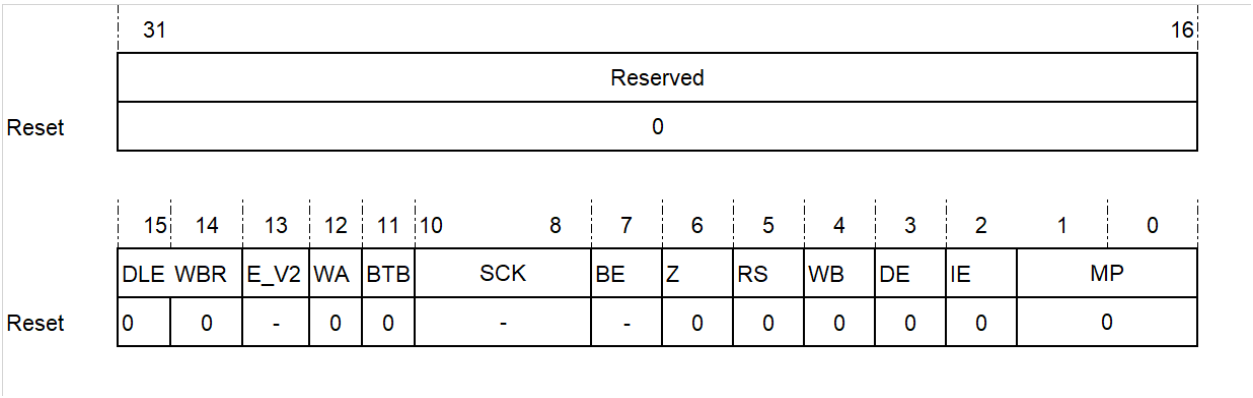


图 3.6: 高速缓存配置寄存器

DLE-数据高速缓存 Lock 使能设置位：

该位有效时，缓存行回填数据高速缓存时，该缓存行会在数据高速缓存中被锁存，后续缓存行缺失不会替换被锁存的缓存行。

WBR-写突发传输设置位：

该位有效时，cache 中 dirty line 的回写要求总线采用突发传输的方式。

E_V2-Endian 版本位：

当 E_V2 为 0 时，endian 采用 V1 版本；

当 E_V2 为 1 时，endian 采用 V2 版本。

E_V2 在 reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

WA-高速缓存写分配有效设置：

该位表示处理器执行 ST 指令且数据缓冲器缺失时，是否在数据高速缓存中分配缺失的缓存行。

当 WA 为 0 时，数据高速缓存为 write non-allocate 模式；

当 WA 为 1 时，数据高速缓存为 write allocate 模式。

BTB-BTB 有效设置：

该位有效时，分支目标缓冲器使能。

SCK-系统和处理器的时钟比：

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

BE-Endian 模式：

当 BE 为 0 时，Little endian；

当 BE 为 1 时，Big endian。

BE 在 reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

Z-允许预测跳转设置位：

当 Z 为 0 时，预测跳转关闭；

当 Z 为 1 时，预测跳转开启。

RS-地址返回栈设置位：

当 RS 为 0 时，返回栈关闭；

当 RS 为 1 时，返回栈开启。

WB-高速缓存写回设置位：

当 WB 为 0 时，数据高速缓存为写直模式；

当 WB 为 1 时，数据高速缓存为写回模式。

DE-数据高速缓存设置位：

当 DE 为 0 时，数据高速缓存关闭；

当 DE 为 1 时，数据高速缓存开启。

IE-指令高速缓存设置位：

当 IE 为 0 时，指令高速缓存关闭；

当 IE 为 1 时，指令高速缓存开启。

MP-内存保护设置位：

MP 用来设置 MMU 是否有效，如下表：

表 3.4: C810M 内存保护设置

MP	功能
00	MMU 无效
01	MMU 有效

3.2.13 高速缓存索引寄存器 (CIR，CR<22,0>)

当 CFR 的 ITS 被配置为 0 时，CIR 的各个位如下：

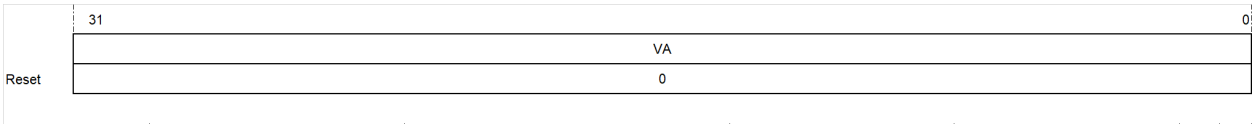
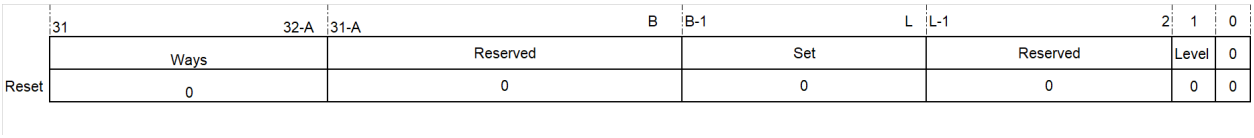


图 3.7: 高速缓存索引寄存器-1

当 CFR 的 ITS 被配置为 1 时，CIR 的各个位如下：

其中，A/B/L 由不同的高速缓存自身属性决定。其中：



	31 32-A	31-A B	B-1 L	L-1 2	1	0
	Ways	Reserved	Set	Reserved	Level	0
Reset	0	0	0	0	0	0

图 3.8: 高速缓存索引寄存器-2

A = Log2(ASSOCIATIVITY)

B = (L + S)

L = Log2(LINELEN)

S = Log2(SETS)

Level：指示操作的高速缓存 level，只用于指令缓存 L0/L1。

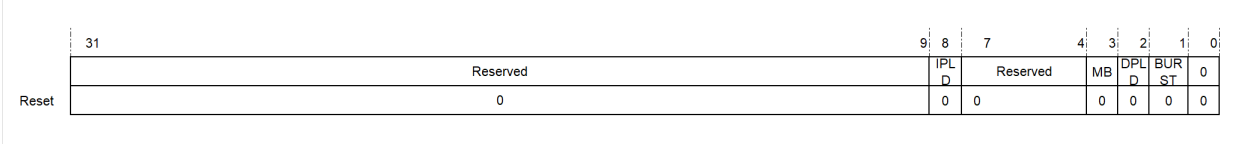
Set：指示操作的块位于高速缓存第几组。

Way：指示操作的块位于高速缓存第几路。

LINELEN：块的长度。

ASSOCIATIVITY：关联度。

3.2.14 CPU HINT 寄存器（HINT，CR<31,0>）



	31 9	8	7 4	3	2	1	0
	Reserved	IPLD	Reserved	MB	DPLD	BURST	0
Reset	0	0	0	0	0	0	0

图 3.9: CPU HINT 寄存器

IPLD-指令 Cache 读预取加速有效设置：

该位有效时，CPU 在检测到连续的指令 Cache 缓存行缺失时，会自动装载接下来的缓存行；此功能可用于加速指令的预取。

MB-写合并有效设置：

该位有效时，CPU 在检测到对于顺序 32 个字范围内的可高缓地址的写操作时，会自动将这些写操作合并成一次总线的突发传输，以提升总线的吞吐量；注意此功能的开启需要系统的 slave 支持 strb 传输。

DPLD-数据 Cache 读预取加速有效设置：

该位有效时，CPU 在检测到连续的数据 Cache 缓存行缺失时，会自动装载接下来的缓存行；此功能可用于加速内存拷贝的性能。

BURST-AHB 读传输 BURST 使能位：

CPU 在配置 AHB 总线，该位有效时缓存行的读访问采用 BURST 方式传输，否则采用 SINGLE 方式传输；当 CPU 配置位 AXI 总线，缓存行的读访问采用 BURST 方式传输，不受该位控制。

3.2.15 MMU 使用操作

玄铁 CPU 的虚拟内存地址空间可运行在两个权限级别上：普通用户模式和超级用户模式，如下图左所示。普通用户只能访问 0x00000000~0x7FFFFFFF 表示的 2GB 空间，在访问 0x80000000~0xFFFFFFFF 表示的 2GB 空间时，出现访问错误；超级用户可以访问完整的 4GB 空间。

从虚拟地址映射的角度，4GB 空间被划分为 4 个区：

USEG（0x00000000 ~ 0x7FFFFFFF）：2GB 用户模式下的虚拟地址访问空间。用户在 MMU 被初始化之前不能使用这个区域。

SSEG0（0x80000000 ~ 0x9FFFFFFF）：512MHz 内核模式下的不可映射、可 Cache 空间。默认情况下，这个空间不可映射指落入这个区的虚拟地址会被强制减掉 0x80000000 后作为物理地址直接返回（即物理地址是通过虚拟地址减去 0x80000000 获得）。如果用户需要修改这个基本属性，可以通过配置 MSA0（CR<30，15>）来配置 SSEG0 区域的起始地址，但空间大小（512M）不可以配置。

SSEG1（0xA0000000 ~ 0xBFFFFFFF）：512MHz 内核模式下的不可映射、不可 Cache 空间。默认情况下，这个空间的不可映射指落入该区域的虚拟地址会被强制减掉 0xA0000000 后作为物理地址直接返回（即物理地址是通过虚拟地址减去 0xA0000000 获得）。由于它的不可 Cache 属性，该区通常作为外设的映射空间（若原外设的物理地址为 0x10000000，其虚拟地址应该设置为 0xB0000000）。如果用户需要修改这个基本属性，可以通过配置 MSA1（CR<31，15>）来配置 SSEG0 区域的起始地址，但空间大小（512M）不可以配置。

SSEG2（0xC0000000 ~ 0xFFFFFFFF）：1GB 内核模式下的可映射、可 Cache 空间。这个空间允许通过 MMU 的 TLB 进行物理地址的灵活映射。

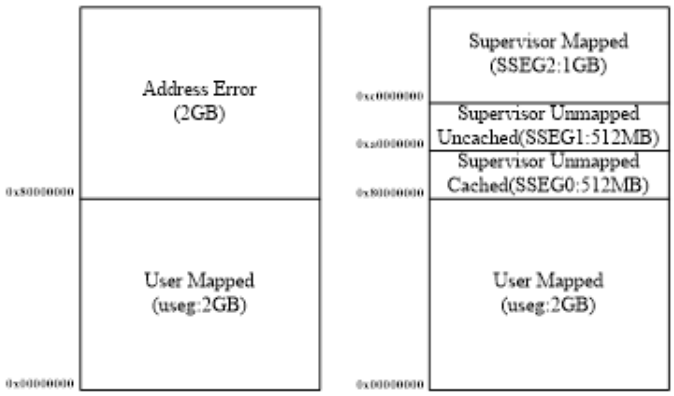


图 3.10: MMU 使用操作

C810M 通过第 15 组控制寄存器来实现内存管理单元（MMU）的功能。系统编程员在超级用户编程模式下来通过设置与 MMU 相关的状态寄存器，来进行 MMU 的管理和操作。这些寄存器包括：

- MMU 索引寄存器 (MIR)；
- MMU EntryLo0 寄存器 (MEL0)；
- MMU EntryLo1 寄存器 (MEL1)；
- MMU EntryHi/Bad VPN 寄存器 (MEH)；

- MMU 页掩码寄存器 (MPR)；
- MMU 控制指令寄存器 (MCIR)；
- MMU PGD 基址及硬件回填使能控制寄存器（MPGD）；
- MMU SSEG0 区域配置寄存器（MSA0）；
- MMU SSEG1 区域配置寄存器（MSA1）。

3.2.16 MMU 索引寄存器 (MIR，CR<0,15>)

MIR 寄存器用于指示读写操作的 jTLB 表项，或存放 TLBP 指令匹配的结果，其各位如下图所示：

	31	30	29 ... 10	9 ... 0
	P	TF	Reserved	Index
Reset	0	0	0	0

图 3.11: MMU 索引寄存器

P -匹配失败标志：

当执行 TLBP 指令时，硬件写该位指示 jTLB 是否匹配成功。

当 P 为 0 时，jTLB 匹配成功；

当 P 为 1 时，jTLB 匹配失败。

TF-TLB 不可恢复异常标志：

该位用于表征是否出现 TLB 不可恢复异常；当 tfatal 为 1 时，表示出现 TLB 不可恢复异常。

Index-jTLB 索引：

软件写该域以提供 TLB 表项的索引，参考 TLBR 和 TLBWI 用法说明。

当执行 TLBP 指令时，硬件用匹配的 jTLB 表项的索引写该域。如果 TLBP 指令匹配失败，该域的内容不可预知。

C810M，jTLB 表可配置为 64 或 128 个表项，vTLB 固定为 4 个表项，jTLB 表项的索引号为 0 至 63/127，vTLB 表项的索引号固定为 512、513、514、515；sTLB 表项的索引号固定为 520、521、522、523，每个 sTLB 表项有独立的 page 大小，通过自身的 page mask 值来配置。

3.2.17 MMU EntryLo0 和 EntryLo1 寄存器 (MEL0 & MEL1，CR<2,15> & CR<3,15>)

MEL 寄存器保存 jTLB 访问的物理地址高位和页面属性信息，其各位如图表所示。

PFN-页帧号：

该域对应于物理地址的 31 位到 12 位。

	31 12	11 7	6	5	4	3	2	1	0
	PFN	Reserved	B	SO	SEC	C	D	V	0
Reset	0	0	0	0	0	0	0	0	0

图 3.12: MMU EntryLo0 和 EntryLo1 寄存器

B-页 buffer 位：

指示当前页是否可 buffer。

SO-Strong Order 位:

指示当前页数据读写访问顺序是否和程序流的读写访问顺序相同：

当 SO 为 0 时，当前页数据读写访问顺序不受程序流的读写访问顺序限制；

当 SO 为 1 时，当前页数据读写访问顺序必须与程序流的读写访问顺序相同。

SEC-安全访问位：

指示当前页是否支持安全访问。

C-可高缓位：

指示当前页是否可高缓：

当 C 为 0 时，当前页不可高缓；

当 C 为 1 时，当前页可高缓。

D-Dirty 位：

指示当前页是否可写。

当 D 为 1 时，当前页可写；

当 D 为 0 时，当前页不可写，写该页将产生 TLB 修改异常。

V-有效位：

指示 jTLB 表项虚拟页映射是否有效。

当 V 为 1 时，该页有效，可访问；

当 V 为 0 时，该页无效，不可访问，访问该页将产生 TLB 无效异常。

3.2.18 MMU EntryHi/Bad VPN 寄存器 (MEH，CR <4,15>)

MEH 寄存器有两个功能：包含 TLB 访问的虚拟地址信息和 TLB 异常时当前 VPN 的值。ASID 表示当前页面对应的进程号。其各位如下图所示：

VPN-虚拟页序号：

该域在 TLB 读和 TLB 异常时由硬件写，在软件写 TLB 表项之前由软件预先写入。

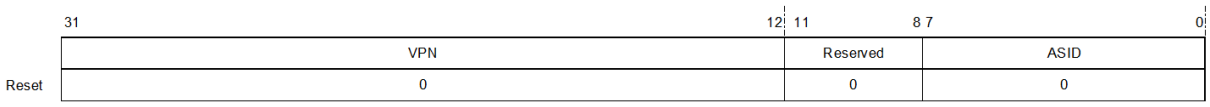


图 3.13: MMU EntryHi/Bad VPN 寄存器

注意：在 TLB 访问时，该域的最低位无意义。在 TLB 异常时，该域的最低位有意义。软件将通过该位分辨哪个页面出现异常。例如，当产生 TLB 修改异常时，软件将通过该位找到对应的发生修改异常的页面，然后设置相应的 Dirty 位。

ASID-地址空间标识：

该域通常用来保存操作系统所看到的当前地址空间的标识，异常不会改变其值，因此在 TLB 异常发生后，它依然保存着当前运行进程的正确地址空间标识。

绝大部分软件系统会把当前进程的地址空间标识写入该域。在使用 TLBR 指令读取 TLB 表项时必须要小心，该操作会覆盖整个 MEH 寄存器，因此在执行该操作之后必须恢复当前进程地址空间的标识。

在软件读写 TLB 表项时，该域用于指示当前表项所属的进程号；在 MMU 进行地址映射时，该域用于指示当前进程号。

该域在软件读 TLB 表项时由硬件设置；在软件写 TLB 表项时由软件设置。

3.2.19 MMU 页掩码寄存器 (MPR，CR<6,15>)

MPR 寄存器用于配置页面的大小，其各位如图所示。

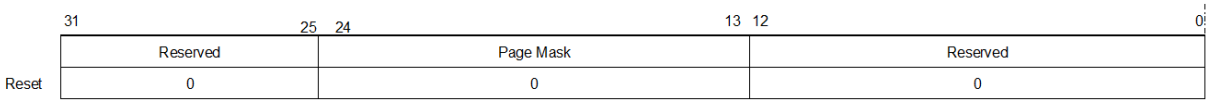


图 3.14: MMU 页掩码寄存器

Mask-页掩码：

该域的位为 1 表示虚拟地址的对应位不用于 TLB 匹配。

页掩码的编码如图表所示：

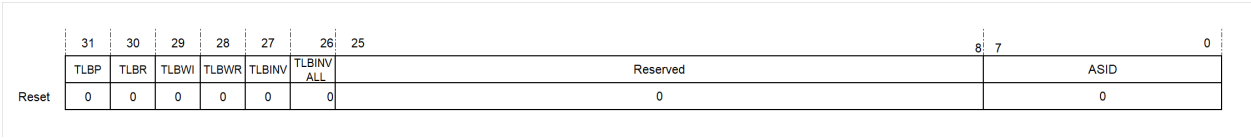
表 3.5: 页掩码编码

页掩码	页大小	奇偶块选择位
0000_0000_0000	4KB	VAddr[12]
0000_0000_0011	16KB	VAddr[14]
0000_0000_1111	64KB	VAddr[16]
0000_0011_1111	256KB	VAddr[18]
0000_1111_1111	1MB	VAddr[20]
0011_1111_1111	4MB	VAddr[22]
1111_1111_1111	16MB	VAddr[24]

如果软件装载到 MPR 寄存器的值与上面列出的值都不一样，处理器的操作将不可预知。

3.2.20 MMU 控制指令寄存器 (MCIR, CR<8,15>)

MCIR 寄存器实现支持操作 MMU 的命令，包括 TLB 查找、TLB 读、TLB 写索引、TLB 写随机、TLB 无效、TLB 无效全部表项。这些操作可以通过 MTCR 指令实现。其各位如图表所示。



	31	30	29	28	27	26	25 … 8	7 … 0
	TLBP	TLBR	TLBWI	TLBWR	TLBINV	TLBINV ALL	Reserved	ASID
Reset	0	0	0	0	0	0	0	0

图 3.15: MMU 控制指令寄存器

TLBP-TLB 查找：

该控制指令用于在由 MEH 寄存器 VPN 域低位指定的两路 jTLB 和全部 vTLB 表项中查找虚拟页号和 ASID（G 位有效时不比较 ASID）跟 MEH 寄存器中内容相匹配的表项，并把相应表项的索引值存入 MIR 寄存器。如果匹配失败，MIR 寄存器的 P 位被置 1。

TLBR-TLB 读：

该控制指令用于从 TLB 读一个表项。由 MIR 指向的 jTLB 表项的内容被装载到 MEH、MEL0 和 MEL1 寄存器。如果 MIR 寄存器的 Index 值无效时，处理器的操作将不可预知。

TLBWI-TLB 写索引：

该控制指令用于写 MIR 寄存器指向的 TLB 表项，即将 MEH、MEL0、MEL1 和 MPR 寄存器的内容写入到由 MIR 寄存器指向的 TLB 表项。如果 MIR 寄存器的 Index 值无效时，处理器的操作将不可预知。

TLBWR-TLB 写随机：

该控制指令用于写 MEH 寄存器 VPN 域低位指向的 jTLB 表项，即将 MEH、MEL0、MEL1 和 MPR 寄存器的内容写入到 MEH 寄存器 VPN 域低位指向的 jTLB 表项。在写入到 jTLB 前，把 jTLB 的对应 TLB 表项读出来，该表项有效时转存到 vTLB 中。

TLBINV-TLB 无效：

该控制指令用于删除与 ASID 域值相同的所有 TLB 表项。

TLBINV_ALL-TLB 无效全部表项：

该控制指令用于删除与 jTLB 和 vTLB 中的所有 TLB 表项。

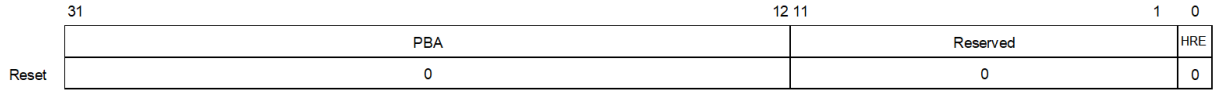
ASID-再循环 ASID：

该域仅仅对 TLB 无效指令有意义。如果存在多于 256 个进程，可能需要再循环 ASID。

MMU 指令的优先级：TLBINV_ALL>TLBINV>TLBP>TLBWI>TLBWR>TLBWR。当同时设置两个以上的指令域时，按照优先级仅且一个 MMU 指令有效。

3.2.21 MMU PGD 基址寄存器 (MPGD，CR<29,15>)

MPGD 寄存器包含 PGD_BASE_address（物理地址）和硬件回填使能控制位。在 TLB 缺失且硬件回填使能时，不产生 TLB 失配异常，操作系统从内存中读取回填失配的 TLB。其各位如下图所示：



	31 … 12	11 … 1	0
	PBA	Reserved	HRE
Reset	0	0	0

图 3.16: MMU PGD 基址寄存器

PBA-PGD 表项基地址：

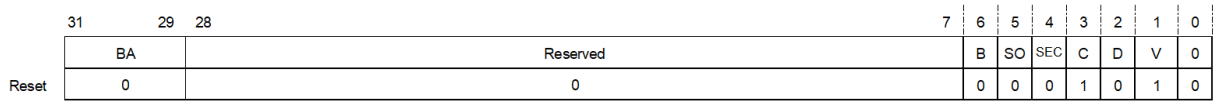
此为 PGD 表的基地址，为物理地址。

HRE-硬件回填使能位：

此为硬件回填使能控制位，当有效时 TLB 发生了失配，MMU 单元会通过 BIU 单元到内存中读取相关表项。

3.2.22 MMU SSEG0 配置寄存器 (MSA0，CR<30,15>)

MSA0 寄存器配置 SSEG0 区映射的物理地址和该区属性的信息，如果用户要改变 SSEG0 映射关系或者映射区域的属性，可以通过配置 MSA0 寄存器实现。默认情况下，该寄存器具有初始值，虚拟地址空间的 0x80000000 – 0xA000 0000 会被强制映射到物理空间的 0x0000 0000 – 0x20000000。其各位及初始值如下图所示：



	31 … 29	28 … 7	6	5	4	3	2	1	0
	BA	Reserved	B	SO	SEC	C	D	V	0
Reset	0	0	0	0	0	1	0	1	0

图 3.17: MMU EntryLo0 和 EntryLo1 寄存器-1

BA-SSEG0 映射的物理地址：

该域对应于物理地址的 31 位到 29 位。

B-页可缓存位：

指示 SSEG0 区是否可 buffer。

SO-Strong Order 位:

指示 SSEG0 区的数据读写访问顺序是否和程序流的读写访问顺序相同：

当 SO 为 0 时，SSEG0 区数据读写访问顺序不受程序流的读写访问顺序限制；

当 SO 为 1 时，SSEG0 区数据读写访问顺序必须与程序流的读写访问顺序相同。

默认 SSEG0 区的数据读写访问顺序不受程序流的读写访问顺序限制。

SEC-安全访问位：

指示当前页是否支持安全访问。

C-可高缓位：

指示 SSEG0 区是否可高缓：

当 C 为 0 时，SSEG0 区不可高缓；

当 C 为 1 时，SSEG0 区可高缓。

默认 SSEG0 可高缓。

D–脏（Dirty）位：

指示 SSEG0 区是否可写。

标志位 D 硬件固定为 1，这一位用户不可修改。

V-有效位：

指示 SSEG0 区映射是否有效。

超级用户模式下 V 为 1 时，SSEG0 被映射到物理地址 {BA，29 ‘b0} 上，V 为 0 这个映射关系失效，虚拟地址会到 jTLB 中查询，若 miss 则出 TLB miss 异常，否则根据 TLB 表项中的内容转换出物理地址。强烈推荐 V 位为 1，此时通过直接映射实现对 SSEG0 的虚实地址映射管理。

普通用户模式下 V 为 1 则访问此区域出 access error 异常；V 为 0，则访问 sseg0 区域时不出 access error，会转到 TLB 表项中查询，若 TLB 表项中没给该段地址分配物理表页，此时正常出 TLB miss，否则按 TLB 的对应关系转出物理地址

3.2.23 MMU SSEG1 配置寄存器 (MSA1，CR<31,15>)

MSA1 寄存器配置 SSEG1 区映射的物理地址和该区属性的信息，如果用户要改变 SSEG1 映射关系或者映射区域的属性，可以通过配置 MSA1 寄存器实现。默认情况下，该寄存器具有初始值，虚拟地址空间的 0xA0000000 – 0xC000 0000 会被强制映射到物理空间的 0x0000 0000 – 0x20000000。其各位及初始值如图表所示。

	31 29	28 7	6	5	4	3	2	1	0
	BA	Reserved	B	SO	SEC	C	D	V	0
Reset	0	0	0	1	0	0	0	1	0

图 3.18: MMU EntryLo0 和 EntryLo1 寄存器-2

BA-SSEG1 映射的物理地址：

该域对应于物理地址的 31 位到 29 位。

B–页可缓存位：

指示 SSEG1 区是否可 buffer。

SO-Strong Order 位:

指示 SSEG1 区的数据读写访问顺序是否和程序流的读写访问顺序相同：

当 SO 为 0 时，SSEG1 区数据读写访问顺序不受程序流的读写访问顺序限制；

当 SO 为 0 时，SSEG1 区数据读写访问顺序必须与程序流的读写访问顺序相同；

默认 SSEG1 区的数据读写访问顺序必须与程序流的读写访问顺序相同。

SEC-安全访问位：

指示当前页是否支持安全访问。

C-可高缓位：

指示 SSEG1 区是否可高缓：

当 C 为 0 时，当前页不可高缓；

当 C 为 1 时，当前页可高缓。

默认 SSEG1 不可高缓。

D-脏（Dirty）位：

指示 SSEG1 区是否可写。

标志位 D 硬件固定为 1，这一位用户不可修改

V-有效位：

超级用户模式下 V 为 1 时，SSEG1 被映射到物理地址 {BA，29 ‘b0} 上，V 为 0 这个映射关系失效，虚拟地址会到 jTLB 中查询，若 miss 则出 TLB miss 异常，否则根据 TLB 表项中的内容转换出物理地址。强烈推荐 V 位为 1，此时通过直接映射实现对 SSEG1 的虚实地址映射管理。

普通用户模式下 V 为 1 则访问此区域出 access error 异常；V 为 0，则访问 sseg1 区域时不出 access error，会到 TLB 表项中查询，若 TLB 表项中没给该段地址分配物理表页，此时正常出 TLB miss，否则按 TLB 的对也关系转出物理地址；

对于 SSEG2 区域，超级用户模式下能正常进行表项建立；普通用户模式下只有当 sseg0 和 sseg1 的 V 都为 0 时，sseg2 会变成可访问，可以通过配置 jTLB 表项来对该区域进行访问，若 TLB 表项中没有配置它的对应关系则会出 TLB miss 异常。

3.2.24 TLB 表项结构

TLB 包含两部分：比较部分和物理转换部分。比较部分包括表项的虚拟页序号（因为每个表项都映射到两个物理页，所以事实上是虚拟页序号/2，VPN2），ASID 和 G 位。物理转换部分包括一对表项，每个表项都包含物理页帧序号 (PFN)，有效 (V) 位，脏 (D) 位和可高缓位 (C)。每个 jTLB 表项对应两个物理页面并包含一个表项有效位。TLB 表项的位定义如下图所示。

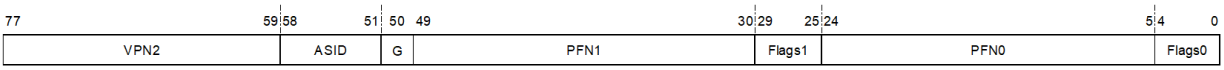


图 3.19: jTLB 表项

第四章　32 位指令

本章主要介绍 C810 实现的玄铁 CPU V2 的 32 位指令集，包括 32 位指令集的功能分类、编码方式和寻址模式等。

4.1　32 位指令功能分类

C810 实现的玄铁 CPU V2 的 32 位指令集按照指令实现的功能来划分，可以分为：

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

4.1.1　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 4.1: 32 位加减法指令列表

指令名称	指令说明
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 4.2: 32 位逻辑操作指令列表

指令名称	指令说明
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 4.3: 32 位移位指令列表

指令名称	指令说明
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 4.4: 32 位比较指令列表

指令名称	指令说明
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 4.5: 32 位数据传输指令列表

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

比特操作指令：

表 4.6: 32 位比特操作指令列表

指令名称	指令说明
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 4.7: 32 位提取插入指令列表

指令名称	指令说明
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 4.8: 32 位乘除法指令列表

指令名称	指令说明
MULU32	无符号数乘法指令
MULS32	有符号数乘法指令
MULUA32	无符号数乘累加指令
MULSA32	有符号数乘累加指令
MULUS32	无符号数乘累减指令
MULSS32	有符号数乘累减指令
MULT32	乘法指令
MULSH32	16 位有符号乘法指令
DIVU32	无符号除法指令
DIVS32	有符号除法指令

杂类运算指令:

表 4.9: 32 位杂类运算指令列表

指令名称	指令说明
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

4.1.2 分支跳转指令

分支跳转指令可以进一步分为:

分支指令:

表 4.10: 32 位分支指令列表

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

跳转指令:

表 4.11: 32 位跳转指令列表

指令名称	指令说明
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

4.1.3 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 4.12: 32 位立即数偏移存取指令列表

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

向量寄存器偏移存取指令：

表 4.13: 32 位向量寄存器偏移存取指令列表

指令名称	指令说明
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 4.14: 32 位多寄存器存取指令列表

指令名称	指令说明
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 4.15: 32 位符号存取指令列表

指令名称	指令说明
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

4.1.4 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 4.16: 32 位控制寄存器操作指令列表

指令名称	指令说明
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 4.17: 32 位低功耗指令列表

指令名称	指令说明
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 4.18: 32 位异常返回指令列表

指令名称	指令说明
RTE32	异常和普通中断返回指令
RFI32	快速中断返回指令

4.1.5 特殊功能指令

特殊功能指令具体包括：

表 4.19: 32 位特殊功能指令列表

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

4.2 32 位指令编码方式

玄铁 CPU V2 32 位指令集在编码风格上可以分为 3 大类，分别为:

- 跳转类型（J 型）
- 立即数类型（I 型）
- 寄存器类型（R 型）

4.2.1 跳转类型

32 位指令跳转类型（J 型）的编码方式如下所示：

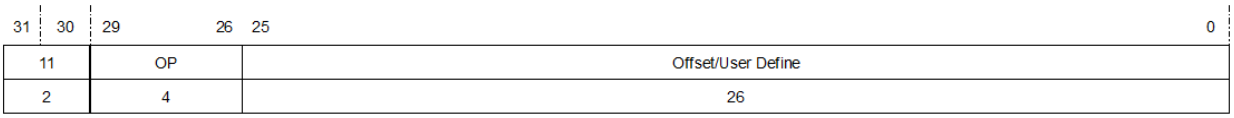


图 4.1: 跳转类型

OP 域为主操作码，通过 4 位主操作码可以识别该编码类型的指令；Offset/User Define 域为跳转指令的偏移量或者供用户自定义的保留域。

4.2.2 立即数类型

32 位指令立即数类型（I 型）包含 18 位立即数、16 位立即数和 12 位立即数三种编码方式。

18 位立即数的编码方式如下所示：

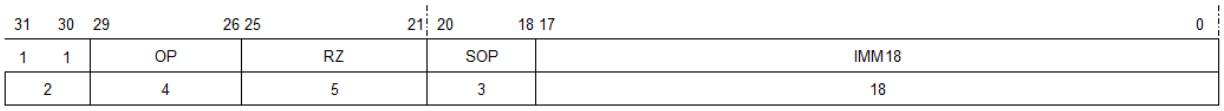


图 4.2: 立即数类型-18 位立即数的编码

OP 域为主操作码，通过 4 位主操作码可以识别指令或者指令类型；RZ 域为目的寄存器域；SOP 域为子操作码域；IMM18 域为 18 位立即数。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

16 位立即数的编码方式如下所示：

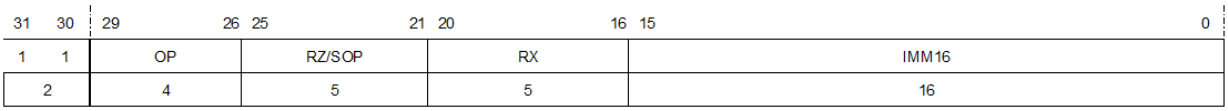
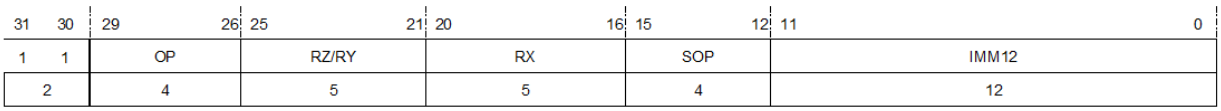


图 4.3: 立即数类型-16 位立即数的编码

OP 域为主操作码，通过 4 位主操作码可以识别指令或者指令类型；RZ/SOP 域为目的寄存器域或者子操作码域；RX 域为第一源寄存器；IMM16 域为 16 位立即数。

12 位立即数的编码方式如下所示：



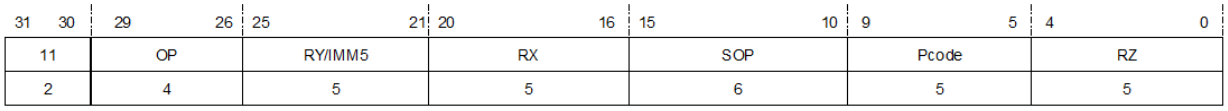
31 30	29 26	25 21	20 16	15 12	11 0
1 1	OP	RZ/RY	RX	SOP	IMM12
2	4	5	5	4	12

图 4.4: 立即数类型-12 位立即数的编码

OP 域为主操作码，通过 4 位主操作码可以识别指令或者指令类型；RZ/RY 域为目的寄存器域或者第二源寄存器域；RX 域为第一源寄存器；SOP 域为子操作码域；IMM12 域为 12 位立即数。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

4.2.3 寄存器类型

32 位指令寄存器类型（R 型）的编码方式如下所示：



31 30	29 26	25 21	20 16	15 10	9 5	4 0
11	OP	RY/IMM5	RX	SOP	Pcode	RZ
2	4	5	5	6	5	5

图 4.5: 寄存器类型

OP 域为主操作码，通过 4 位主操作码可以识别指令的类型；RY/IMM5 域为第二源寄存器域或者 5 位立即数；RX 为第一源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域；RZ 域为目的寄存器域。部分指令经过主操作码 OP 译码之后得出指令类型，再经过子操作码 SOP 译码之后得出指令子类，最后通过并行操作码 Pcode 译码识别出具体指令。Pcode 采用独热（one-hot）编码方式。

4.3 32 位指令操作数寻址模式

玄铁 CPU V2 的 32 位指令集总体遵循三种指令编码方式，每种编码方式都有自己特有的操作数寻址模式，下面将分别介绍各种操作数寻址模式。

4.3.1 跳转类型编码指令寻址方式

玄铁 CPU V2 中跳转类型编码的 32 位指令只有一种寻址方式。

4.3.1.1 二十六位立即数寻址方式

26 位立即数寻址方式指令中，有一段 26 比特长的立即数域。此域可以被认为是偏移量（Offset），用于运算产生目标地址。采用这种格式的指令有 bsr32。

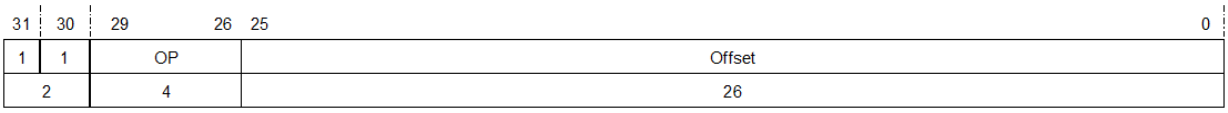


图 4.6: 二十六位立即数寻址方式

4.3.2 立即数类型编码指令寻址方式

玄铁 CPU V2 中立即数类型编码的 32 位指令有五种寻址方式。

4.3.2.1 一元寄存器十八位立即数寻址方式

一元寄存器十八位立即数寻址方式指令中，RZ 域为目的寄存器域或者第二源寄存器域；SOP 域为子操作码域；IMM18 域可以作为 18 位相对偏移量，用于运算产生目标地址。采用这种格式的指令包括 lrs32.b、lrs32.h、lrs32.w、grs32、srs32.b、srs32.h、srs32.w、addi32。

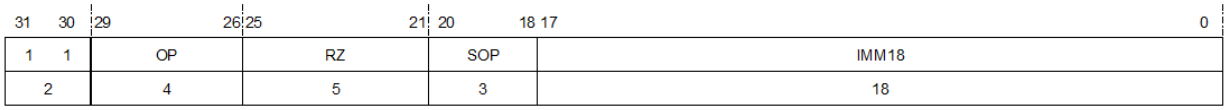


图 4.7: 一元寄存器十八位立即数寻址方式

4.3.2.2 二元寄存器十六位立即数寻址方式

二元寄存器十六位立即数寻址方式指令中，两个寄存器域 RX、RZ 分别为源寄存器域和目的寄存器域；IMM16 域作为 16 位立即数直接参与数据运算。采用这种格式的指令包括 ori32。

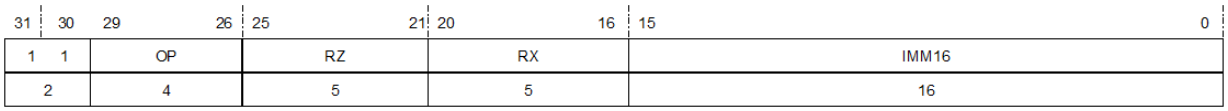


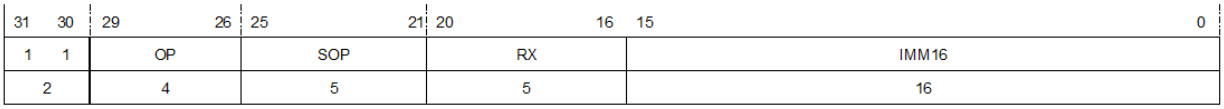
图 4.8: 二元寄存器十六位立即数寻址方式

4.3.2.3 一元寄存器十六位立即数寻址方式

一元寄存器十六位立即数寻址方式指令中，可以进一步分为两种格式。

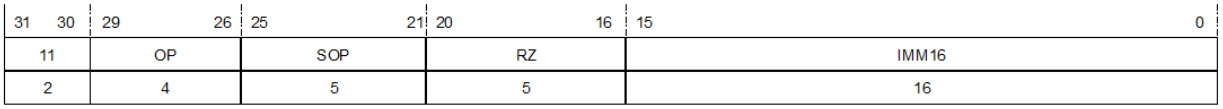
如 图 4.9 , 第一种格式中，SOP 域为子操作码域；RX 域为源寄存器域；IMM16 域可以作为 16 位相对偏移量，用于运算产生目标地址；IMM16 域也可以作为 16 位立即数直接参与数据运算。采用这种格式的指令包括 bez32、bnez32、bhz32、blsz32、blz32、bhsz32、cmphsi32、cmplti32、cmpnei32。

如 图 4.10 , 第二种格式中，SOP 域为子操作码域；RZ 域为目的寄存器域；IMM16 域为 16 位立即数直接参与数据运算或者留给用户自行定义。采用这种格式的指令包括 movi32、movih32、lrw32。



31 30	29 26	25 21	20 16	15 0
1 1	OP	SOP	RX	IMM16
2	4	5	5	16

图 4.9: 一元寄存器十六位立即数寻址方式-格式一

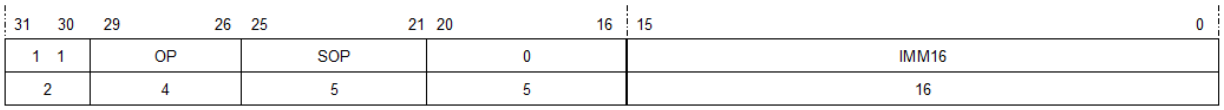


31 30	29 26	25 21	20 16	15 0
11	OP	SOP	RZ	IMM16
2	4	5	5	16

图 4.10: 一元寄存器十六位立即数寻址方式-格式二

4.3.2.4 十六位立即数寻址方式

16 位立即数寻址方式指令中, 有一段 16 比特长的立即数域。此域可以被认为是偏移量（Offset），用于运算产生目标地址。采用这种格式的指令有 br32、bf32、bt32、jmpi32、jsri32。



31 30	29 26	25 21	20 16	15 0
1 1	OP	SOP	0	IMM16
2	4	5	5	16

图 4.11: 十六位立即数寻址方式

4.3.2.5 二元寄存器十二位立即数寻址方式

如 图 4.12 ，二元寄存器十二位立即数寻址方式中，RZ 域为目的寄存器域或者第二源寄存器域；RX 域为第一源寄存器域；SOP 域为子操作码域；IMM12 域可以作为 12 位相对偏移量，用于运算产生目标地址。采用这种格式的指令包括 ld32.b、ld32.h、ld32.w、ld32.d、ld32.bs、ld32.hs、pldr32、st32.b、st32.h、st32.w、st32.d、pldw、addi32、subi32、andi、andni、xori。

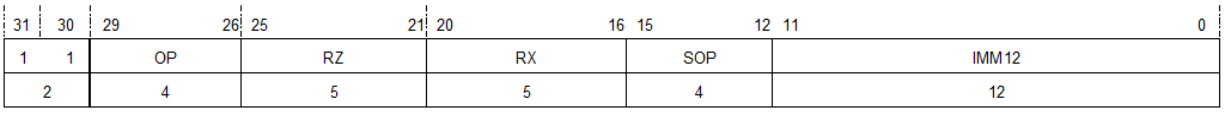


图 4.12: 二元寄存器十二位立即数寻址方式

4.3.3 寄存器类型编码指令寻址方式

玄铁 CPU V2 中寄存器类型编码的 32 位指令有五种寻址方式。

4.3.3.1 三元寄存器寻址方式

如 图 4.13 ，三元寄存器寻址方式中，RY 为第二源寄存器域；RX 域为第一源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域；RZ 为目的寄存器域。采用这种格式的指令包括 addu32、addc32、subu32、subc32、ixh32、ixw32、ixd32、and32、andn32、or32、xor32、nor32、lsl32、lsr32、asr32、rotl32、ldr32.b、ldr32.h、ldr32.w、ldr32.bs、ldr32.hs、str32.b、str32.h、str32.w、divu32、divs32、mult32、mulsh32。

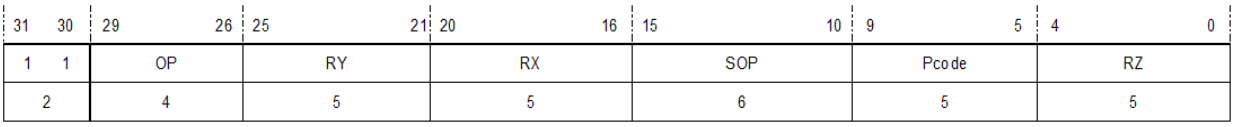


图 4.13: 三元寄存器寻址方式

4.3.3.2 二元寄存器五位立即数寻址方式

二元寄存器五位立即数寻址方式中，可以进一步分为两种格式。

如 图 4.14 ，第一种格式中，IMM5 域为 5 位立即数，作为源操作数；RX 域为源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域；RZ 域为目的寄存器域。采用这种格式的指令包括 decgt32、declt32、decne32、lsli32、lsri32、asri32、rotli32、lslc32、lsrc32、asrc32、xsr32、bclri32、bseti32。

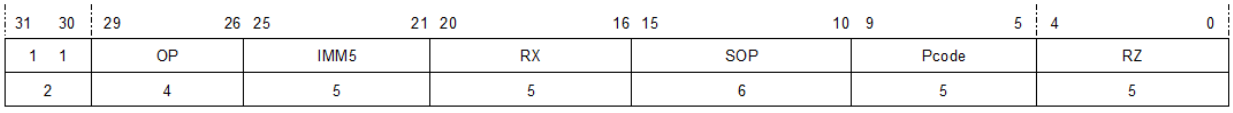
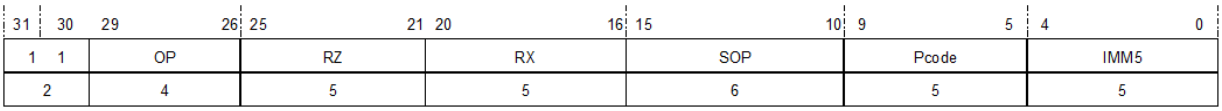


图 4.14: 二元寄存器五位立即数寻址方式-格式一

如 图 4.15 ，第二种格式中，IMM5 域为 5 位立即数，作为源操作数；RX 域为源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域；RZ 域为目的寄存器域和第二源寄存器域。采用这种格式的指令包括 incf32、inct32、decf32、dect32。



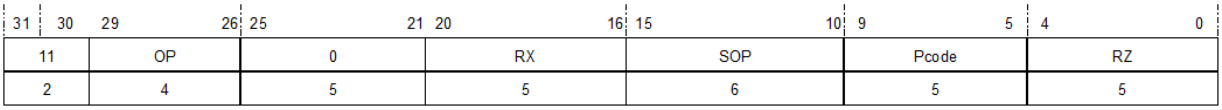
31	30	29 26	25 21	20 16	15 10	9 5	4 0
1	1	OP	RZ	RX	SOP	Pcode	IMM5
2		4	5	5	6	5	5

图 4.15: 二元寄存器五位立即数寻址方式-格式二

4.3.3.3 二元寄存器寻址方式

二元寄存器寻址方式指令中，可以进一步分为两种格式。

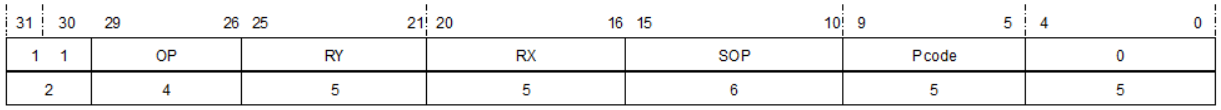
第一种格式中，RZ 域为目的寄存器域；RX 域为源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域。采用这种格式的指令包括 bgenr32、xtrb0.32、xtrb1.32、xtrb2.32、xtrb3.32、brev32、revb32、revh32、abs32、ff0.32、ff1.32。



31	30	29 26	25 21	20 16	15 10	9 5	4 0
11		OP	0	RX	SOP	Pcode	RZ
2		4	5	5	6	5	5

图 4.16: 二元寄存器五位立即数寻址方式-格式一

第二种格式中，RY 域为第二源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域；RX 域位第一源寄存器域。采用这种格式的指令包括 cmpne32、cmphs32、cmplt32、tst32、mulu32、mulua32、mulus32、muls32、mulsa32、mulss32。



31	30	29 26	25 21	20 16	15 10	9 5	4 0
1	1	OP	RY	RX	SOP	Pcode	0
2		4	5	5	6	5	5

图 4.17: 二元寄存器五位立即数寻址方式-格式二

4.3.3.4 一元寄存器五位立即数寻址方式

一元寄存器五位立即数寻址方式中指令中，可以进一步分为两种格式。

如 图 4.18 , 第一种格式中，IMM5 域为 5 位立即数，作为源操作数；RX 域为源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域。采用这种格式的指令包括 btsti32。

如 图 4.19 , 第二种格式中，IMM5 域为 5 位立即数，作为源操作数；SOP 域为子操作码域；Pcode 为并行操作码域；RZ 域为目的寄存器域。采用这种格式的指令包括 bmaski32。

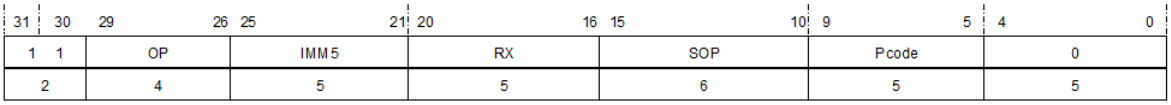


图 4.18: 一元寄存器五位立即数寻址方式-格式一

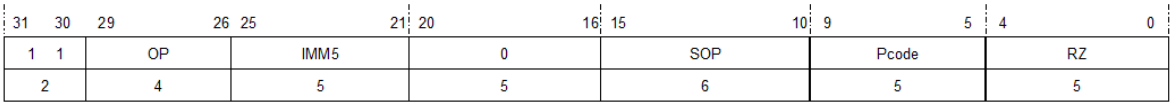


图 4.19: 一元寄存器五位立即数寻址方式-格式二

4.3.3.5 一元寄存器寻址方式

一元寄存器寻址方式中，可以进一步分为三种格式。

如 图 4.20 , 第一种格式中，RZ 为目的寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域。采用这种格式的指令包括 mvc32、mvcv32。

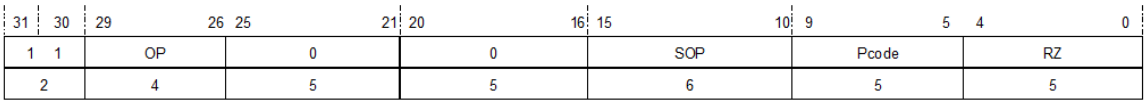


图 4.20: 一元寄存器寻址方式-格式一

如 图 4.21 , 第二种格式中，RX 为源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域。采用这种格式的指令包括 tstnbz32。

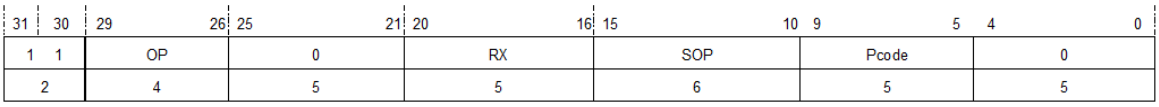


图 4.21: 一元寄存器寻址方式-格式二

如 图 4.22 , 第三种格式中，RZ 为目的寄存器域和源寄存器域；SOP 域为子操作码域；Pcode 为并行操作码域。采用这种格式的指令包括 clrf32、clrt32.

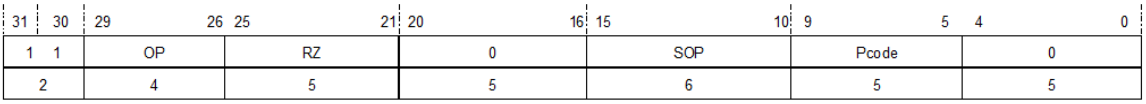


图 4.22: 一元寄存器寻址方式-格式三

第五章　16 位指令

本章主要介绍玄铁 CPU V2 的 16 位指令集，包括 32 位指令集到 16 位指令集的映射方式，以及 16 位指令集的功能分类，编码方式和寻址模式等。

5.1　32/16 指令映射方式

玄铁 CPU V2 中 16 位指令集是 32 位指令集的一个子集，每条 16 位指令都有对应的 32 位指令。具体的 32 位指令和 16 位指令之间的映射方式见下表：

表 5.1: 32/16 指令映射表

汇编指令	32 位	16 位	指令描述
ADDU	√	×	无符号加法指令
ADDC	√	√	无符号带进位加法指令
ADDI	√	√	无符号立即数加法指令
SUBU	√	√	无符号减法指令
SUBC	√	√	无符号带借位减法指令
SUBI	√	√	无符号立即数减法指令
RSUB	√	×	反向减法指令
IXH	√	×	索引半字指令
IXW	√	×	索引字指令
IXD	√	×	索引双字指令
INCF	√	×	C 为 0 立即数加法指令
INCT	√	×	C 为 1 立即数加法指令
DECF	√	×	C 为 0 立即数减法指令
DECT	√	×	C 为 1 立即数减法指令
DECGT	√	×	减法大于零置 C 位指令
DECLT	√	×	减法小于零置 C 位指令
DECNE	√	×	减法不等于零置 C 位指令
AND	√	√	按位与指令
ANDI	√	×	立即数按位与指令

下页继续

表 5.1 – 续上页

ANDN	√	√	按位非与指令
ANDNI	√	×	立即数按位非与指令
OR	√	√	按位或指令
ORI	√	×	立即数按位或指令
XOR	√	√	按位异或指令
XORI	√	×	立即数按位异或指令
NOR	√	√	按位或非指令
NOT	√	√	按位非指令
LSL	√	√	逻辑左移指令
LSLI	√	√	立即数逻辑左移指令
LSLC	√	×	立即数逻辑左移至 C 位指令
LSR	√	√	逻辑右移指令
LSRI	√	√	立即数逻辑右移指令
LSRC	√	×	立即数逻辑右移至 C 位指令
ASR	√	√	算术右移指令
ASRI	√	√	立即数算术右移指令
ASRC	√	×	立即数算术右移至 C 位指令
ROTL	√	√	循环左移指令
ROTLI	√	×	立即数循环左移指令
XSR	√	×	扩展右移指令
CMPNE	√	√	不等比较指令
CMPNEI	√	√	立即数不等比较指令
CMPHS	√	√	无符号大于等于比较指令
CMPHSI	√	√	立即数无符号大于等于比较指令
CMPLT	√	√	有符号小于比较指令
CMPLTI	√	√	立即数有符号小于比较指令
TST	√	√	零测试指令
TSTNBZ	√	√	无字节等于零寄存器测试指令
MOV	√	√	数据传送指令
MOVF	√	×	C 为 0 数据传送指令
MOVT	√	×	C 为 1 数据传送指令
MOVI	√	√	立即数数据传送指令
MOVIH	√	×	立即数高位数据传送指令
LRW	√	√	存储器读入指令
MTHI	√	×	累加器高位写传送指令
MTLO	√	×	累加器低位写传送指令
MFHI	√	×	累加器高位读传送指令
MFLO	√	×	累加器低位读传送指令

下页继续

表 5.1 – 续上页

MVCV	√	√	C 位取反传送指令
MVC	√	×	C 位传送指令
MVTC	√	×	溢出位复制到 C 位指令
CLRF	√	×	C 为 0 清零指令
CLRT	√	×	C 为 1 清零指令
BCLRI	√	√	立即数位清零指令
BSETI	√	√	立即数位置位指令
BTSTI	√	×	立即数位测试指令
ZEXT	√	×	位提取并无符号扩展指令
SEXT	√	×	位提取并有符号扩展指令
INS	√	×	位插入指令
ZEXTB	√	√	字节提取并无符号扩展指令
ZEXTH	√	√	半字提取并无符号扩展指令
SEXTB	√	√	字节提取并有符号扩展指令
SEXTH	√	√	半字提取并有符号扩展指令
XTRB0	√	×	提取字节 0 并无符号扩展指令
XTRB1	√	×	提取字节 1 并无符号扩展指令
XTRB2	√	×	提取字节 2 并无符号扩展指令
XTRB3	√	×	提取字节 3 并无符号扩展指令
BREV	√	×	位倒序指令
REVB	√	√	字节倒序指令
REVH	√	√	半字内字节倒序指令
MULU	√	×	无符号数乘法指令
MULS	√	×	有符号数乘法指令
MULUA	√	×	无符号数乘累加指令
MULSA	√	×	有符号数乘累加指令
MULUS	√	×	无符号数乘累减指令
MULSS	√	×	有符号数乘累减指令
MULT	√	√	乘法指令
MULSH	√	√	16 位有符号乘法指令
MULSH	√	√	16 位有符号乘法指令
DIVU	√	×	无符号除法指令
DIVS	√	×	有符号除法指令
ABS	√	×	绝对值指令
FF0	√	×	快速找 0 指令
FF1	√	×	快速找 1 指令
BMASKI	√	×	立即数位屏蔽产生指令
BGENR	√	×	寄存器位产生指令

下页继续

表 5.1 – 续上页

BGENI	√	×	立即数位产生指令
BT	√	√	C 为 1 分支指令
BF	√	√	C 为 0 分支指令
BEZ	√	×	寄存器等于零分支指令
BNEZ	√	×	寄存器不等于零分支指令
BHZ	√	×	寄存器大于零分支指令
BLSZ	√	×	寄存器小于等于零分支指令
BLZ	√	×	寄存器小于零分支指令
BHSZ	√	×	寄存器大于等于零分支指令
BR	√	√	无条件跳转指令
BSR	√	√	跳转到子程序指令
JMP	√	√	寄存器跳转指令
JSR	√	√	寄存器跳转到子程序指令
RTS	√	√	链接寄存器跳转指令
LD.B	√	√	无符号扩展字节加载指令
LD.BS	√	×	有符号扩展字节加载指令
LD.H	√	√	无符号扩展半字加载指令
LD.HS	√	×	有符号扩展半字加载指令
LD.W	√	√	字加载指令
LD.D	√	×	双字加载指令
ST.B	√	√	字节存储指令
ST.H	√	√	半字存储指令
ST.W	√	√	字存储指令
ST.D	√	×	双字存储指令
LDR.B	√	×	寄存器移位寻址无符号扩展字节加载指令
LDR.BS	√	×	寄存器移位寻址有符号扩展字节加载指令
LDR.H	√	×	寄存器移位寻址无符号扩展半字加载指令
LDR.HS	√	×	寄存器移位寻址有符号扩展半字加载指令
LDR.W	√	×	寄存器移位寻址字加载指令
STR.B	√	×	寄存器移位寻址字节存储指令
STR.H	√	×	寄存器移位寻址半字存储指令
STR.W	√	×	寄存器移位寻址字存储指令
LRS.B	√	×	字节符号加载指令
LRS.H	√	×	半字符号加载指令
LRS.W	√	×	字符号加载指令
SRS.B	√	×	字节符号存储指令
SRS.H	√	×	半字符号存储指令
SRS.W	√	×	字符号存储指令

下页继续

表 5.1 – 续上页

LDQ	√	√	连续四字加载指令
LDM	√	√	连续多字加载指令
STQ	√	√	连续四字存储指令
STM	√	√	连续多字存储指令
POP	√	√	出栈指令
PUSH	√	√	压栈指令
MFCR	√	×	控制寄存器读传送指令
MTCR	√	×	控制寄存器写传送指令
PSRSET	√	×	PSR 位置位指令
PSRCLR	√	×	PSR 位清零指令
WAIT	√	×	进入低功耗等待模式指令
DOZE	√	×	进入低功耗睡眠模式指令
STOP		×	进入低功耗暂停模式指令
RTE	√	×	异常和普通中断返回指令
RFI	√	×	快速中断返回指令
SYNC	√	×	CPU 同步指令
BKPT	×	√	断点指令
SCE	√	×	条件执行设置指令
IDLY	√	×	中断识别禁止指令
TRAP	√	×	无条件操作系统陷阱指令
PLDR	√	×	读数据预取指令
PLDW	√	×	写数据预取指令

注解: √ 表示相应指令集中存在该指令，× 表示相应指令集中不存在该指令。

5.2 32/16 指令执行延时

表 5.2: 32/16 指令执行延时

汇编指令	**32 位指令 执行延时**	**32 位指令 执行延时**	**备注**
ADDU	1	×	-
ADDC	1	1	-
ADDI	1	1	-
SUBU	1	1	-
SUBC	1	1	-

下页继续

表 5.2 – 续上页

SUBI	1	1	-
RSUB	1	×	-
IXH	1	×	-
IXW	1	×	-
IXD	1	×	-
INCF	2	×	-
INCT	2	×	-
DECF	2	×	-
DECT	2	×	-
DECGT	2	×	-
DECLT	2	×	-
DECNE	2	×	-
AND	1	1	-
ANDI	1	×	-
ANDN	1	1	-
ANDNI	1	×	-
OR	1	1	-
ORI	1	×	-
XOR	1	1	-
XORI	1	×	-
NOR	1	1	-
NOT	1	1	-
LSL	1	1	-
LSLI	1	1	-
LSLC	1	×	-
LSR	1	1	-
LSRI	1	1	-
LSRC	1	×	-
ASR	1	1	-
ASRI	1	1	-
ASRC	1	×	-
ROTL	1	1	-
ROTLI	1	×	-
XSR	1	×	-
CMPNE	1	1	-
CMPNEI	1	1	-
CMPHS	1	1	-
CMPHSI	1	1	-

下页继续

表 5.2 – 续上页

CMPLT	1	1	-
CMPLTI	1	1	-
TST	1	1	-
TSTNBZ	1	1	-
MOV	1	1	-
MOVF	1	×	-
MOVT	1	×	-
MOVI	1	1	-
MOVIH	1	×	-
LRW	4	4	-
MTHI	1	×	-
MTLO	1	×	-
MFHI	1	×	-
MFLO	1	×	-
MVCV	1	1	-
MVC	1	×	-
MVTC	1	×	-
CLRF	1	×	-
CLRT	1	×	-
BCLRI	1	1	-
BSETI	1	1	-
BTSTI	1	×	-
ZEXT	2	×	-
SEXT	2	×	-
INS	2	×	-
ZEXTB	1	1	-
ZEXTH	1	1	-
SEXTB	1	1	-
SEXTH	1	1	-
XTRB0	1	×	-
XTRB1	1	×	-
XTRB2	1	×	-
XTRB3	1	×	-
BREV	1	×	-
REVB	1	1	-
REVH	1	1	-
MULU	3	×	-
MULS	3	×	-

下页继续

表 5.2 – 续上页

MULUA	4	×	-
MULSA	4	×	-
MULUS	4	×	-
MULSS	4	×	-
MULT	3	3	-
MULSH	3	3	-
DIVU	1-33	1-33	-
DIVS	1-33	×	-
ABS	2	×	-
FF0	1	×	-
FF1	1	×	-
BMASKI	2	×	-
BGENR	2	×	-
BGENI	1	×	-
BT	1	×	-
BF	1	1	-
BEZ	1	1	-
BNEZ	1	×	-
BHZ	1	×	-
BLSZ	1	×	-
BLZ	1	×	-
BHSZ	1	×	-
BR	1	×	-
BSR	1	1	-
JMP	1	×	-
JSR	1	1	-
RTS	1	1	-
LD.B	4	4	-
LD.BS	4	4	-
LD.H	4	×	-
LD.HS	4	4	-
LD.W	4	×	-
LD.D	5	5	-
ST.B	2	×	-
ST.H	2	2	-
ST.W	2	2	-
ST.D	2	2	-
LDR.B	4	×	-

下页继续

表 5.2 – 续上页

LDR.BS	4	×	-
LDR.H	4	×	-
LDR.HS	4	×	-
LDR.W	4	×	-
STR.B	2	×	-
STR.H	2	×	-
STR.W	2	×	-
LRS.B	4	×	-
LRS.H	4	×	-
LRS.W	4	×	-
SRS.B	2	×	-
SRS.H	2	×	-
SRS.W	2	×	-
LDQ	N+3	×	拆分为 N 条 LD 指令
LDM	N+3	N+3	拆分为 N 条 LD 指令
STQ	N+1	N+1	拆分为 N 条 ST 指令
STM	N+1	N+1	拆分为 N 条 ST 指令
POP	N+3	N+3	拆分为 N 条 LD 指令
PUSH	N+4	N+4	拆分为 N 条 ST 指令
MFCR	1	1	-
MTCR	1	×	-
PSRSET	1	×	-
PSRCLR	1	×	-
WAIT	1	×	-
DOZE	1	×	-
STOP	1	×	-
RTE	1	×	-
RFI	1	×	-
SYNC	1	×	-
BKPT	1	×	-
SCE	1	1	-
IDLY	1	×	-
TRAP	1	×	-
PLDR	4	×	-
PLDW	4	×	-

5.3 16 位指令功能分类

玄铁 CPU V2 的 16 位指令集按照指令实现的功能来划分，可以分为:

- 数据运算指令
- 分支跳转指令
- 内存存取指令

5.3.1 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 5.3: 16 位加减法指令列表

指令名称	指令说明
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 5.4: 16 位逻辑操作指令列表

指令名称	指令说明
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 5.5: 16 位移位指令列表

指令名称	指令说明
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

下页继续

表 5.5 – 续上页

指令名称	指令说明
ASRI16	立即数算术右移指令
ROTL16	循环左移指令

比较指令：

表 5.6: 16 位比较指令列表

指令名称	指令说明
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 5.7: 16 位数据传输指令列表

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

比特操作指令：

表 5.8: 16 位比特操作指令列表

指令名称	指令说明
BCLRI16	立即数位清零指令
BSETI16	立即数位置位指令

提取插入指令：

表 5.9: 16 位提取插入指令列表

指令名称	指令说明
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 5.10: 16 位乘法指令列表

指令名称	指令说明
MULT16	乘法指令
MULSH16	16 位有符号乘法指令

5.3.2 分支跳转指令

分支跳转指令可以进一步分为:

分支指令:

表 5.11: 16 位分支指令列表

指令名称	指令说明
BT16	C 为 1 分支指令
BF16	C 为 0 分支指令

跳转指令:

表 5.12: 16 位跳转指令列表

指令名称	指令说明
BR16	无条件跳转指令
BSR16	跳转到子程序指令
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

5.3.3 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 5.13: 16 位立即数偏移存取指令列表

指令名称	指令说明
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 5.14: 16 位多寄存器存取指令列表

指令名称	指令说明
POP16	出栈指令
PUSH16	压栈指令

5.4 16 位指令编码方式

玄铁 CPU V2 的 16 位指令集在编码风格上基本和 32 指令子集保持一致，可以分为 3 大类，分别为:

- 跳转类型（J 型）
- 立即数类型（I 型）
- 寄存器类型（R 型）

5.4.1 跳转类型

跳转类型（J 型）的编码方式如下图：

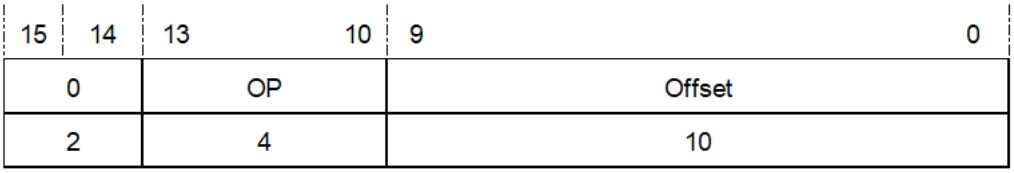


图 5.1: 跳转类型

OP 域为主操作码，通过 4 位主操作码可以识别该编码类型的指令；Offset 域为跳转指令的偏移量。

5.4.2 立即数类型

立即数类型（I 型）包含 3 位立即数、5 位立即数、7 位立即数和 8 位立即数四种编码方式。

3 位立即数的编码方式如下图：

15	14	13 11	10 8	7 5	4 2	1 0
1		OP	RX	RZ	IMM3	SOP
2		3	3	3	3	2

图 5.2: 立即数类型-3 位立即数的编码

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；RZ 域为目的寄存器域；IMM3 域为 3 位立即数；SOP 域为子操作码域。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

5 位立即数的编码方式有三种格式，第一种格式如下图：

15	14	13 11	10 8	7 5	4 0
1		OP	RX	RZ	IMM5
2		3	3	3	5

图 5.3: 立即数类型-5 位立即数的编码-格式一

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域为 5 位立即数。

5 位立即数的第二种编码方式如下图：

15 14	13 11	10 8	7 5	4 0
10	OP	RX	RZ	IMM5
2	3	3	3	5

图 5.4: 立即数类型-5 位立即数的编码-格式二

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域为 5 位立即数。

5 位立即数的第三种编码方式如下 图 5.5 所示。

15 14	13 11	10 8	7 5	4 0
0	OP	RX	SOP	IMM5
2	3	3	3	5

图 5.5: 立即数类型-5 位立即数的编码-格式三

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；SOP 域为子操作码域；IMM5 域为 5 位立即数。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

7 位立即数的编码方式如下图：

15 14	13 10	9 8	7 5	4 0
0	OP	IMM2	SOP/RZ	IMM5
2	4	2	3	5

图 5.6: 立即数类型-7 位立即数的编码

OP 域为主操作码，通过 4 位主操作码可以识别指令或者指令类型；IMM2 域和 IMM5 域分别为 7 位立即数的高 2 位和低 5 位；SOP/RZ 域为子操作码域或目的寄存器域。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

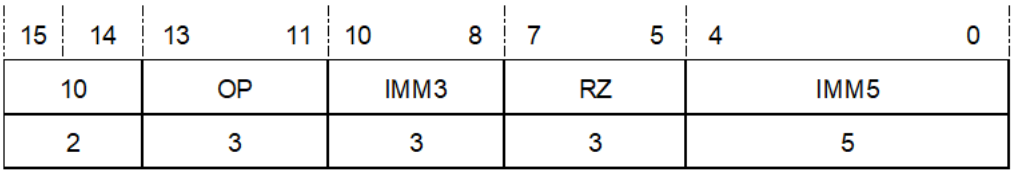
8 位立即数的编码方式有两种格式，第一种格式如下图：

15 14	13 11	10 8	7 0
0	OP	RX/RZ	IMM8
2	3	3	8

图 5.7: 立即数类型-8 位立即数的编码-格式一

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RZ/RX 域为目的寄存器域或者源寄存器域；IMM8 域为 8 位立即数。

8 位立即数的第二种编码方式如下 图 5.8 所示。



15	14	13 11	10 8	7 5	4 0
10		OP	IMM3	RZ	IMM5
2		3	3	3	5

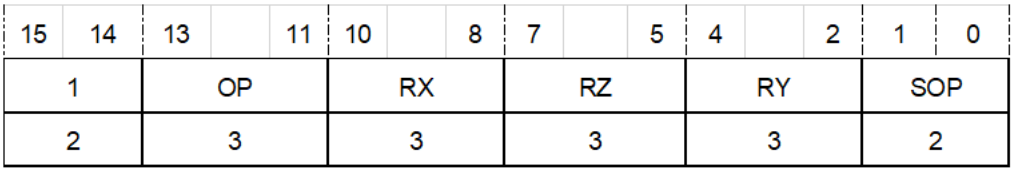
图 5.8: 立即数类型-8 位立即数的编码-格式二

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；IMM3 域和 IMM5 域分别为 8 位立即数的高 3 位和低 5 位；RZ 域为目的寄存器域。

5.4.3 寄存器类型

寄存器类型（R 型）包含 3 操作数和 2 操作数两种编码方式。

3 操作数的编码方式如下图：

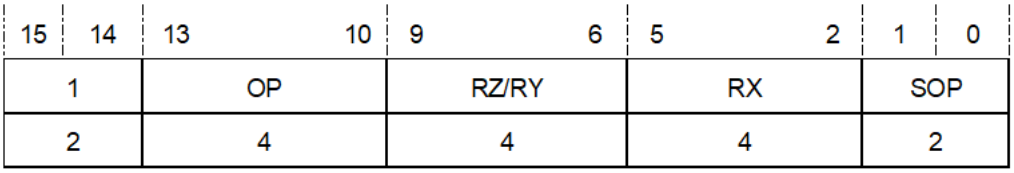


15	14	13 11	10 8	7 5	4 2	1	0
1		OP	RX	RZ	RY	SOP	
2		3	3	3	3	2	

图 5.9: 寄存器类型-3 操作数的编码

OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为第一源寄存器域；RZ 域为目的寄存器域；RY 域为第二源寄存器域；SOP 域为子操作码域。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

2 操作数的编码方式如下图：



15	14	13 10	9 6	5 2	1	0
1		OP	RZ/RY	RX	SOP	
2		4	4	4	2	

图 5.10: 寄存器类型-2 操作数的编码

OP 域为主操作码，通过 4 位主操作码可以识别指令或者指令类型；RZ/RY 域为目的寄存器域或者第二源寄存器域；RX 域为第一源寄存器域；SOP 域为子操作码域。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

5.5 16 位指令操作数寻址模式

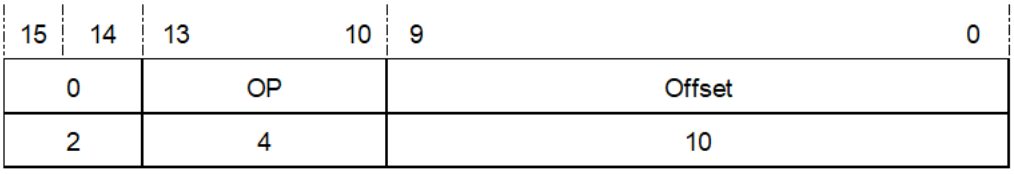
玄铁 CPU V2 的 16 位指令集总体遵循三种指令编码方式，每种编码方式都有自己特有的操作数寻址模式，下面将分别介绍各种操作数寻址模式。

5.5.1 跳转类型编码指令寻址方式

玄铁 CPU V2 中跳转类型编码的 16 位指令只有一种寻址方式。

5.5.1.1 十位立即数寻址方式

十位立即数寻址方式指令中，有一段 10 比特长的立即数域。此域作为偏移量（Offset），用于运算产生目标地址。采用这种格式的指令有 br16、bsr16、bt16、bf16。



15	14	13 10	9 0
0		OP	Offset
2		4	10

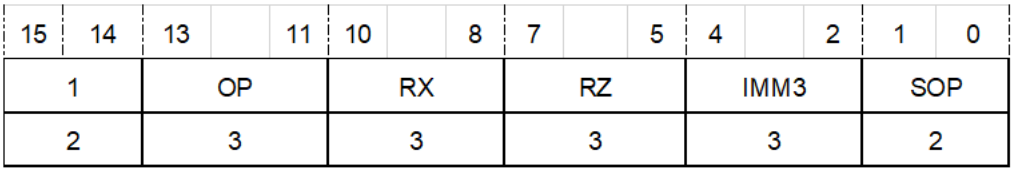
图 5.11: 十位立即数寻址方式

5.5.2 立即数类型编码指令寻址方式

玄铁 CPU V2 中立即数类型编码的 16 位指令有六种寻址方式。

5.5.2.1 二元寄存器三位立即数寻址方式

二元寄存器三位立即数寻址方式指令中，RX 域为源寄存器域；RZ 域为目的寄存器域；IMM3 域也可以作为 3 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 addi16、subi16。



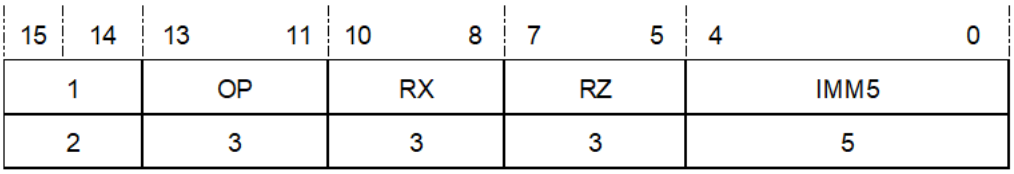
15	14	13 11	10 8	7 5	4 2	1 0
1		OP	RX	RZ	IMM3	SOP
2		3	3	3	3	2

图 5.12: 二元寄存器三位立即数寻址方式

5.5.2.2 二元寄存器五位立即数寻址方式

二元寄存器五位立即数寻址方式指令中，可以进一步分为两种格式。

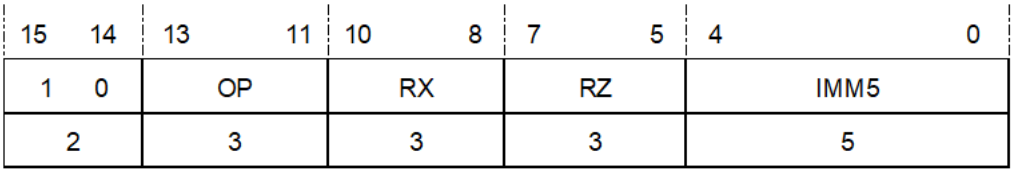
第一种格式中，RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域也可以作为 5 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 lsli16、lsri16、asri16。



15	14	13 11	10 8	7 5	4 0
1		OP	RX	RZ	IMM5
2		3	3	3	5

图 5.13: 二元寄存器五位立即数寻址方式-格式一

第二种格式中，RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域也可以作为 5 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 st16.b、st16.h、st16.w、ld16.b、ld16.h、ld16.w。

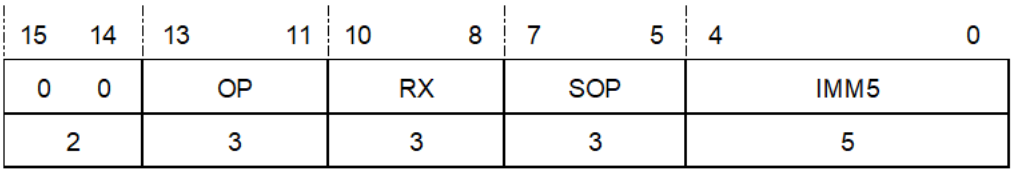


15 14	13 11	10 8	7 5	4 0
1 0	OP	RX	RZ	IMM5
2	3	3	3	5

图 5.14: 二元寄存器五位立即数寻址方式-格式二

5.5.2.3 一元寄存器五位立即数寻址方式

一元寄存器五位立即数寻址方式指令中，RX 域为源寄存器域或者目的寄存器；SOP 域为子操作码域。采用这种格式的指令包括 cmphsi16、cmplti16、cmpnei16、bclri16、bseti16。



15 14	13 11	10 8	7 5	4 0
0 0	OP	RX	SOP	IMM5
2	3	3	3	5

图 5.15: 一元寄存器五位立即数寻址方式

5.5.2.4 一元寄存器七位立即数寻址方式

一元寄存器七位立即数寻址方式指令中，RZ 域为目的寄存器域；IMM2 域和 IMM5 域，可以合并作为 7 位立即数直接参与数据运算。采用这种格式的指令包括 lrw16。

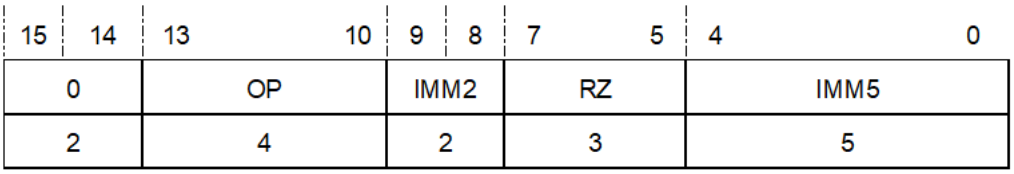


图 5.16: 一元寄存器七位立即数寻址方式

5.5.2.5 七位立即数寻址方式

七位立即数寻址方式指令中，IMM2 域和 IMM5 域，可以合并作为 7 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 push16、pop16、addi16(SP)、subi16(SP)。

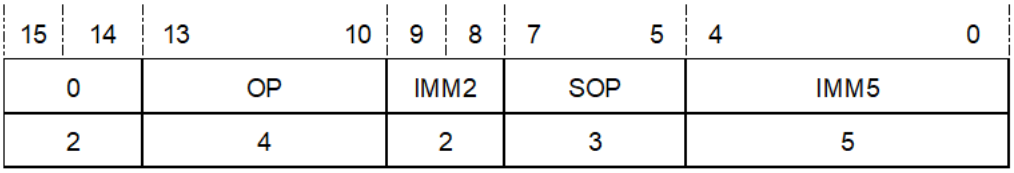


图 5.17: 七位立即数寻址方式

5.5.2.6 一元寄存器八位立即数寻址方式

一元寄存器八位立即数寻址方式指令中，可以进一步分为三种格式。

第一种格式中，RZ 域为目的寄存器域；IMM8 域也可以作为 8 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 addi16(SP)、subi16(SP)、movi16。

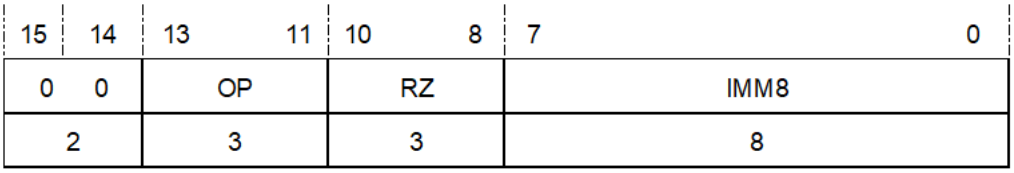


图 5.18: 一元寄存器八位立即数寻址方式-格式一

第二种格式中，RZ 域为源寄存器域和目的寄存器；IMM8 域也可以作为 8 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 addi16、subi16。

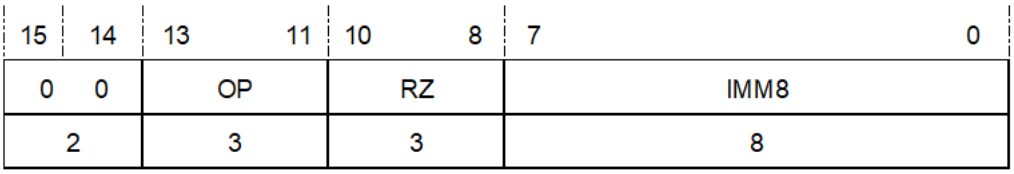


图 5.19: 一元寄存器八位立即数寻址方式-格式二

第三种格式中，RZ 域为源寄存器或者目的寄存器域；IMM3 域和 IMM5 域，可以合并作为 8 位立即数直接参与数据运算。采用这种格式的指令包括 st16.w(SP)、ld16.w(SP)。

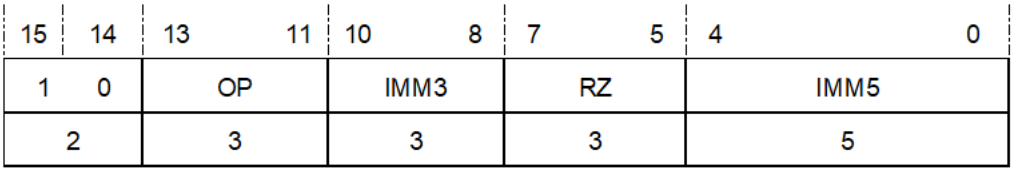


图 5.20: 一元寄存器八位立即数寻址方式-格式三

5.5.3 寄存器类型编码指令寻址方式

玄铁 CPU V2 中寄存器类型编码的 16 位指令有三种寻址方式。

5.5.3.1 三元寄存器寻址模式

三元寄存器寻址方式指令中，两个寄存器域 RX、RY 分别为第一源寄存器域和第二源寄存器域；RZ 域为目的寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 addu16、subu16。

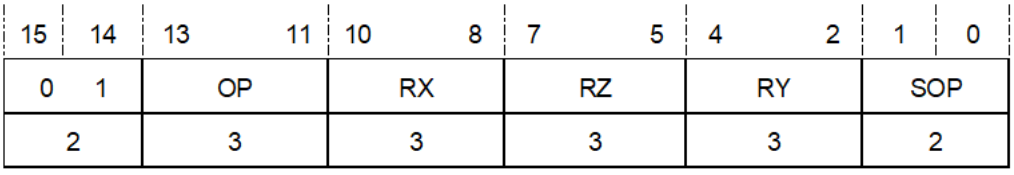


图 5.21: 三元寄存器寻址模式

5.5.3.2 二元寄存器寻址模式

二元寄存器寻址方式指令中，可以进一步分为三种格式。

第一种格式中，两个寄存器域 RX、RY 分别为第一源寄存器域和第二源寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 cmphs16、cmplt16、cmpne16、tst16。

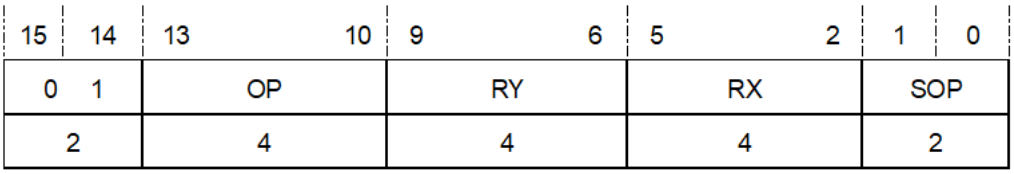


图 5.22: 二元寄存器寻址模式-格式一

第二种格式中，RZ 域为目的寄存器域；RX 为源寄存器；SOP 域为子操作码域。采用这种编码方式的指令有 mov16、zextb16、zexth16、sextb16、sexth16、revb16、revh16。

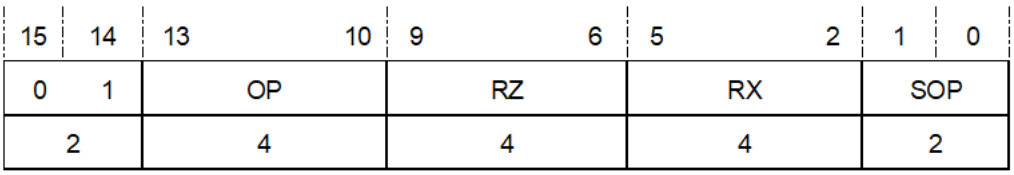


图 5.23: 二元寄存器寻址模式-格式二

第三种格式中，RZ 域为目的寄存器域和第二源寄存器域；RX 为第一源寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 addu16、addc16、subu16、subc16、and16、andn16、or16、xor16、nor16、lsl16、lsr16、asr16、rotl16、mult16、mulsh16。

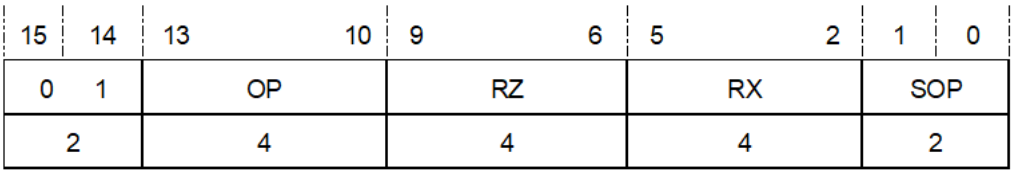


图 5.24: 二元寄存器寻址模式-格式三

5.5.3.3 一元寄存器寻址模式

一元寄存器寻址方式指令中，可以进一步分为两种格式。

第一种格式中，RX 域为源寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 tstnbz16、jmp16、jsr16。

15	14	13 10	9 6	5 2	1 0
0 1		OP	0	RX	SOP
2		4	4	4	2

图 5.25: 一元寄存器寻址模式-格式一

第二种格式中，RZ 域为目的寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 mvcv16。

15	14	13 10	9 6	5 2	1 0
0 1		OP	RZ	0	SOP
2		4	4	4	2

图 5.26: 一元寄存器寻址模式-格式二

第六章　指令流水线

本章介绍关于 C810 的指令流水线和指令时序信息。

C810 微处理器有 10 级流水线：即指令 Cache Tag 访问、指令 Cache 数据访问、指令获取、指令封装、指令译码、寄存器组访问、指令执行 I、指令执行 II/数据高速缓存访问、数据对齐、数据回写 WB。10 级流水线的作用如图。

表 6.1: 各级流水线作用

流水线名称	缩写	流水线作用
指令 Cache 访问 1	IC1	指令 Cache Tag 访问；
指令 Cache 访问 2	IC2	指令 Cache 数据访问；
指令获取	IF	1、访问第 0 级指令高速缓存，获取 64 位指令数据； 2、程序计数器（PC）由虚拟地址到物理地址转换； 3、访问分支目标缓冲器，提前获取分支指令的目标地址。
指令封装	IP	1. 提取并封装两条指令； 2. 访问分支历史表，进行分支指令预测； 3. 指令预译码。
指令译码	ID	1. 指令译码； 2. 寄存器重命名； 3. 指令相关性分析； 4. 指令发射。
访问寄存器组	RF	1. 从寄存器组中读取源操作数； 2. 操作数前馈。
指令执行 I	EX1/AG	1. 指令执行的第一阶段（大多数 ALU 指令在此级完成）； 2. Load /Store 指令的数据地址产生。

下页继续

表 6.1 – 续上页

流水线名称	缩写	流水线作用
指令执行 II/访问数据高缓	EX2/DC	1. 指令执行的第二阶段（少数 ALU 指令在此级完成）; 2. 访问数据高速缓存; 3. 数据地址由虚拟地址到物理地址转换。
数据对齐	DA	1. 依据大小端格式进行数据对齐; 2. 完成 Load/Store 指令。
数据回写	WB	1、指令退休 2、将指令执行结果回写到寄存器组。

C810 设计有 L0 指令 Cache，所以在通常情况下，取指单元直接从 IF 级的 L0 Cache 中获得，无需经过 IC1/IC2 两个流水级，故核心流水线为 8 级，以下的描述均描述核心流水线上的行为。指令访问单元每个周期从第 0 级指令高速缓存中访问获得 64 位指令数据并缓存于指令缓冲中。指令封装单元每个周期从指令缓冲中提取并封装两条指令用于指令执行。后级译码单元对这两条执行进行相关性分析与发射控制。下图显示了指令执行时的数据流向。

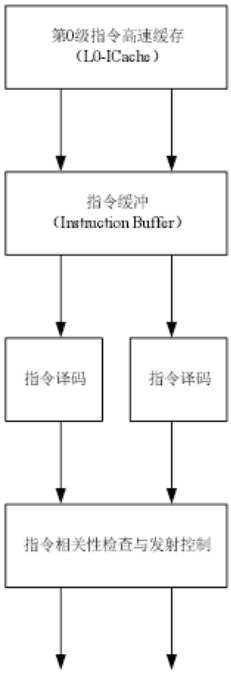


图 6.1: 指令执行时的数据流向

C810 引入了分支历史表与分支目标缓冲器，支持指令访问阶段进行分支指令预测与指令预取，降低分支跳转造成的流水线气泡与性能损失。同时，C810 采用非阻塞发射技术，允许指令之间的乱序发射与执行。相关性指令被发送至指令队列缓存，而后续指令被提前发射与执行，实现了指令之间的乱序发射与执行。为了降低数据相关性对于处理器性能的影响，C810 实现数据前馈技术。一旦产生运算数据，数据前馈技术立即将执行结果反馈到数据前馈逻辑，使得后面的指令不用等到前一条指令回写即可开始执行。

单周期指令流水线重叠执行顺序如 图 6.2 所示，大多数的算术和逻辑指令都属于这类指令。

IF	IP	ID	RF	EX1	WB		
	IF	IP	ID	RF	EX1	WB	
		IF	IP	ID	RF	EX1	WB

图 6.2: 单周期指令流水线重叠执行

如 图 6.3 所示，个别的算术和逻辑指令需要两个执行周期才能完成，具体有 decgt、declt、decne、sext、zex、ins、abs、bgenr、maski。此外，特权指令也需要两个执行周期完成。

图 6.3: 双周期指令执行过程

乘法指令，除法指令，至少需要 4 个执行周期才能完成。乘法指令 mulsh、mult 等需要四个执行周期完成，如下 图 6.4 所示。

图 6.4: 乘法指令 mulsh 和特权模式指令的执行过程

乘累加指令 mulua、mulus、mulsa 等需要 5 个执行周期完成，如下 图 6.5 所示。

图 6.5: 乘法指令 mult 的执行过程

除法指令 divs、divu 需要 4-36 个执行周期完成，如下 图 6.6 所示。

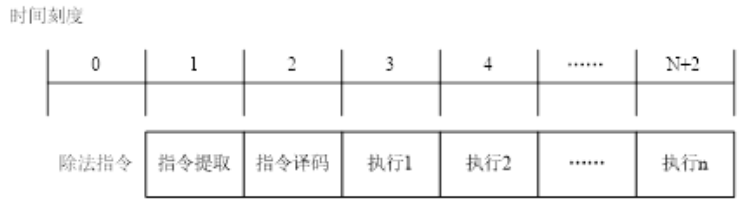


图 6.6: 除法指令的执行过程

br, bsr 指令的跳转和 bt,bf 指令预测跳转且预测正确时需要两个周期来填充流水线，其执行过程如下 图 6.7 所示。

图 6.7: BR, BSR 指令的跳转和 BT, BF 指令预测跳转且预测正确时执行过程

bt, bf 指令预测不跳转且预测正确时，流水线不停顿，其执行过程如下 图 6.8 所示。

图 6.8: BT, BF 指令预测不跳转且预测正确时执行过程

bt、bf 指令预测不正确和 jmp（除 jmpr15 外）、jsr 指令的跳转需要 6 个周期来填充流水线，其执行过程如下 图 6.9 所示。

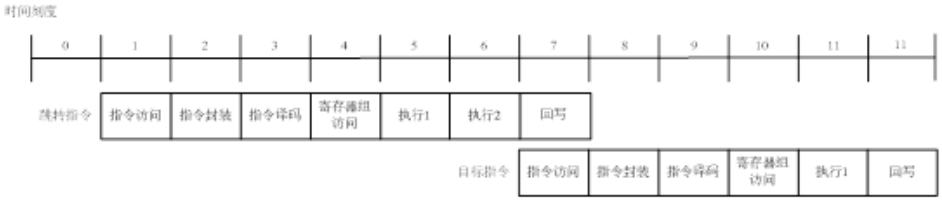


图 6.9: JMP, BT 和 BF 指令预测不正确时的执行过程

对于函数返回指令 jmp r15，C810 采用返回栈加速函数返回的速度。当 jmpr15 指令从返回站获取跳转目标地址且预取正确时，该跳转指令仅需两个周期填充流水线。若目标地址预测错误则需要 6 个周期填充流水线。

由于 jsri 指令的执行被拆分为一条数据加载指令 LD 与一条寄存器跳转指令 JMP，因此 JSRI 跳转需要至少 10 个周期来填充流水线，故通常情况下，采用 bsr 指令代替 jsri 指令功能。其执行过程如下 图 6.10 所示。

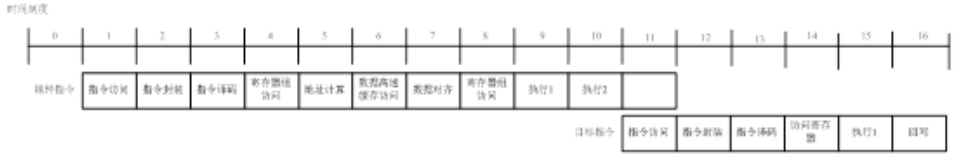


图 6.10: JSRI 指令的执行过程

极端情况下，当跳转指令的目标指令是双字未对齐的 32 位指令时，取指需要两次访问指令总线，于是存在至少一个时钟周期的延迟。图 6.11 显示了一条 BR 指令在目标指令是一条 32 位双字未对齐指令 ADD 的执行过程：

图 6.11: 跳转指令的目标指令是 32 位双字未对齐指令的执行过程

有一些多周期的指令可以流水执行，从而使得每条指令的有效执行时间小于这些指令执行时间的和。这种执行方法的限制是指令和指令之间不能有数据的相关性，并且指令必须按顺序结束和回写。当多周期指令后面有一条单周期指令时，该单周期指令必须等到前一条指令结束之后才能结束，以满足指令按序退休的需要。图 6.12 显示了两条没有数据相关性的 LD/ST 指令后跟一条单周期指令 ADD 的执行过程：

图 6.12: 简单的指令流水线执行过程

对于访问存储区的指令，可能有等待状态。这会导致所有在访问存储区指令之后的指令处于停止状态，因为必须等到访问存储区的指令完成之后这些指令才能完成，其执行过程如图所示。

图 6.13: 带有等待状态的指令流水执行过程

lrw 指令需要 3 个周期才能完成，其目标地址计算在执行级完成。下图显示了执行两条 lrw 指令和一条没有数据相关性的访问存储区指令的执行过程。

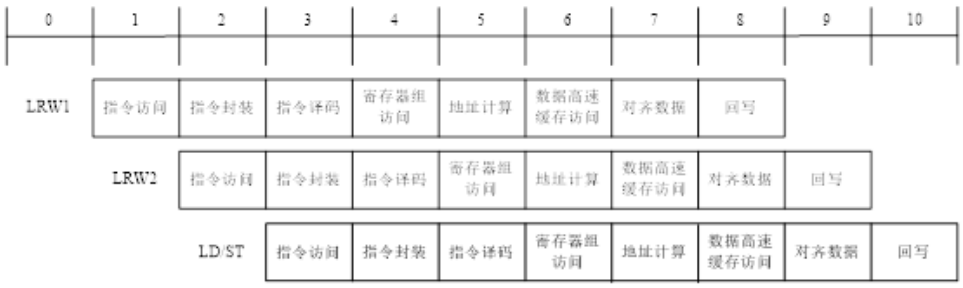


图 6.14: LRW 指令与访问存储指令混合的执行过程

第七章　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，用来使处理器转入对这些事件的处理。这些事件包括硬件错误，指令执行错误，和用户请求服务等等。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。

7.1　异常处理概述

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序。引起异常的外部事件包括：外部设备的中断请求、硬断点请求、读写访问错误和硬件重启；引起异常的内部事件包括：非法指令、不对齐错误（misaligned error）、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、load 和 store 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和一组影子寄存器跳转到异常服务程序的入口。

异常处理的关键就是在异常发生时, 保存 CPU 当前指令运行的状态, 在退出异常处理时恢复异常处理前的状态。异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。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 IF 阶段（未对齐访问、访问错误）或 RF 阶段（非法指令、trap 和 bkpt）被识别，异常也要在相应的指令退休时才会被处理。为了异常处理不影响 CPU 的性能,CPU 在异常处理结束后要避免重复执行以前的指令。C810 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，处理器更新 PSR 中的 VEC，然后保存 PSR 和 PC 到影子寄存器中。对于快速中断，PSR 和 PC 被保存到 FPSR 和 FPC 中；对于其它的异常，它们被保存到 EPSR 和 EPC 中。如果 PSR 的 EE 位被清零了，异常事件（除了普通中断和快速中断）将导致不可恢复的错误异常。PSR 和 PC 被保存后，PSR 的 EE 位被清零，处理器就只能处理不可恢复的错误异常。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第二步，处理器设置 PSR 的 S 位进入超级用户模式，并且把 PSR 的 TM 位清零防止异常服务程序被跟踪。中断使能位 PSR 的 IE 位也被清零禁止响应中断。如果异常是快速中断或重启，快速中断使能 PSR 的 FE 位会被清零，但是其它的异常不影响 PSR 的 FE 位。

传输控制位 PSR 的 TE 位被清零防止外部存储器管理单元对在异常入口地址计算时进行地址转译，使下面的访问以非转译的方式进行。

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

最后一步，处理器计算异常服务程序的第一条指令的地址并将 CPU 的控制权转交给异常服务程序。处理器把异常向量乘以 4 再加上异常向量基准地址（存在向量基准地址寄存器中）就得到了异常向量表中对应的异常入口地址。处理器用这个入口地址从存储器中读取一个字，这个异常入口地址的 [31:1] 转载到程序计数器中作为异常服务程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值的最低位无关）。在特定硬件配置下，处理器把这个异常入口地址的最低位装载到 PSR（AF）位，它用来控制异常处理程序使用哪一个寄存器组。就这样，处理器开始执行新的指令。

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

C810 支持 1024 个字节的向量表包含 256 个异常向量（见下图）。开始的 31 个向量是用作在处理器内部识别的向量。第 32 个向量是留给软件的，用作指向系统描述符的指针。其余的 224 个向量是留给外部设备的。外部设备通过 8 位的中断向量和中断请求使处理器响应中断服务。处理器响应中断请求时锁存这个中断向量。对那些不能提供中断向量的设备，处理器为一般中断和快速中断提供了自动向量。

表 7.1: 异常向量分配

向量号	向量偏移（十六进制）	向量分配
0	000	重启异常。
1	004	未对齐访问异常。
2	008	访问错误异常。
3	00C	除以零异常。
4	010	非法指令异常。
5	014	特权违反异常。
6	018	跟踪异常。
7	01C	断点异常。
8	020	不可恢复错误异常。
9	024	Idly 异常。
10	028	普通中断。
11	02C	快速中断。
12	030	保留（HAI）。
13	034	TLB 不可恢复异常。
14	038	TLB 失配异常，仅在 TLB 软件回填功能使能情况下存在
15	03C	TLB 修改异常。
16 - 19	040 - 04C	陷阱指令异常（TRAP # 0-3）。

下页继续

表 7.1 – 续上页

向量号	向量偏移（十六进制）	向量分配
20	050	TLB 读无效异常。
21	054	TLB 写无效异常。
20 - 29	058 - 074	保留。
30	078	保留
31	07C	保留（系统描述符指针）。
32 - 255	080 - 3FC	保留给向量中断控制器使用。

7.2 异常类型

本节描述外部中断异常和在 C810 内部产生的异常。C810 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/TLB 不可恢复异常；
- Idly4 异常；
- 普通中断；
- 快速中断；
- TLB 失配异常；
- TLB 修改异常；
- 陷阱指令异常；
- TLB 读无效异常；
- TLB 写无效异常。

7.2.1 重启异常（向量偏移 0X0）

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

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

7.2.2 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。

未对齐访问异常也可能发生在数据上。

7.2.3 访问错误异常（向量偏移 0X8）

如果 pad_biu_hresp[1:0]=1，就意味着发生了访问错误异常。当访问 MPU 保护的区域出现访问错误时，发生访问错误异常。

总线上的错误都会引起访问错误异常，使处理器进行异常处理。

7.2.4 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。

7.2.5 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。

7.2.6 特权违反异常（向量偏移 0X14）

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

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

7.2.7 跟踪异常（向量偏移 0X18）

为了便于程序开发调试，C810 提高了对每条指令或对改变控制流指令的跟踪能力。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（branch，jmp，等等）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。

如果处理器工作在跳转跟踪模式下，以下指令都会引起跟踪异常：jmp，jsr，jmpi，jsri，br，bt，bf，bsr，be，bne，bez，bnez，bhz，blsz，blz。bhsz。无论条件跳转指令的执行结果如何，跟踪异常都被使能了。

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，同时等待处理标志 TP 不置位。如果在指令完成时外部中断被探测到了，跟踪异常将延时处理，EPSR 中的等待处理标志 TP 位被置位，在中断服务程序返回时处理跟踪异常。

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

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

以下控制相关的指令不能被跟踪：rte，rfi，trap，stop，wait，doze 和 bkpt。如果 EPSR（TP）有效，跟踪异常作为 rte 或 rfi 正常执行的一部分被处理，而不管 PSR 或 EPSR 的 TM 位的状态。

如果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 或 rfi 退休时，等待处理的跟踪异常会被处理。等待处理的跟踪异常是用来跟踪前面的指令的，这条指令已经退休，为了避免这个异常被丢失，它的优先级是最高的仅次于重启。

7.2.8 断点异常（向量偏移 0X1C）

C810 提供了断点指令 bkpt 和硬断点请求管脚（pad_biu_brkrq_b[1:0]），它们被分配同一个断点异常向量。请参考接口总线协议接口操作中 pad_biu_brkrq_b[1:0] 管脚的操作。为了尽量避免硬断点异常被丢掉，硬断点异常被赋予了比中断更高的优先级。如果 pad_biu_brkrq_b[1:0] 设置在指令访问上，相应的指令不会被执行，当它退休时，处理器响应硬断点异常。如果 pad_biu_brkrq_b[1:0] 设置在数据访问上，相应的指令被允许完成，当它退休时，硬断点异常被响应。如果断点异常是由于 bkpt 指令或指令访问时 pad_biu_brkrq_b[1:0] 设置，EPC 指向该指令。如果数据访问时 pad_biu_brkrq_b[1:0] 设置，EPC 指向它的下一条指令。

7.2.9 不可恢复错误异常（向量偏移 0X20）

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

由于所写的软件在 PSR（EE）为零时应该排除了异常事件发生的可能，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。

7.2.10 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。

7.2.11 中断异常

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

尽管处理器可以通过允许多周期指令被中断的方式来缩短中断响应的延时，一般中断在指令的边界上被确认。如果 PSR（IC）位设置了，divs，divu，ldm，stm，ldq 和 stq 可以被中断而不等它们完成。

C810 提供了两个中断请求信号，支持自动提供中断向量号和由外设显式地提供中断向量号的方式。

下图显示了和中断相关的到处理器核的接口信号。为了支持向量化的中断，外设在中断请求时，用 7 位的中断向量信号提供中断向量号，或设置 pad_biu_avec_b 使用预先定义好的中断向量号。如果 pad_biu_avec_b 有效，pad_biu_vec_b[7:0] 上的输入信号被忽略。pad_biu_int_b，pad_biu_fint_b 和 pad_biu_avec_b 都是低电平有效。

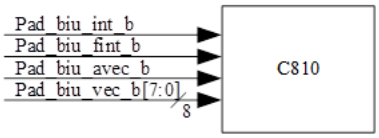


图 7.1: 中断接口信号

7.2.11.1 普通中断（INT）

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

7.2.11.2 快速中断（FINT）

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

7.2.11.3 中断处理过程

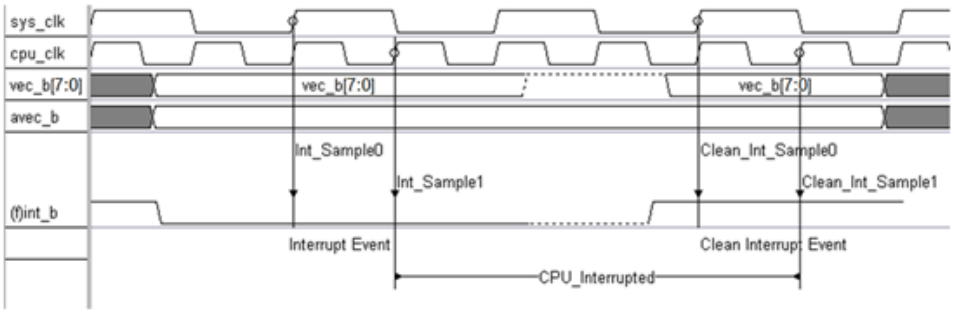


图 7.2: 中断处理过程

在上图中，当中断向量号 pad_biu_vec_b[7:0] 和自动中断向量 pad_biu_avec_b 都已经准备好时，拉低普通中断信号线 pad_biu_int_b，该信号经系统时钟 sys_clk 和 CPU 内部时钟 cpu_clk 的上升沿依次采样后，CPU 内部收到中断，并根据 pad_biu_avec_b 的设置取得中断向量，进入中断服务程序。在中断服务程序中，应该清除外部中断，即拉高 pad_biu_int_b，此信号亦需经此两个时钟依次采样后才会退出中断。快速中断 (fint) 的操作亦是如此。

7.2.12 TLB 不可恢复异常（向量偏移 0X34）

在物理地址转换或 jTLB 查找过程中，如果发现多个匹配的表项存在，将发生 TLB 不可恢复异常。

7.2.13 TLB 硬件回填访问错误异常（向量偏移 0X34）

在 TLB 表项硬件回填过程中出现了访问错误。

7.2.14 TLB 失配异常（向量偏移 0X38）

在 CPU 取指或数据访问时，如果 jTLB 中没有与虚拟地址匹配的表项，将发生 TLB 失配异常，软件需要负责再填充 jTLB。

C810 了增加了 TLB 硬件回填的功能，当 MMU 的 TLB 硬件回填功能使能时，该异常不发生。

7.2.15 TLB 修改异常（向量偏移 0X3C）

在 CPU 写数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面 dirty 位时发生 TLB 修改异常。

7.2.16 陷阱指令异常（向量偏移 0X40 - 0X4C）

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

7.2.17 TLB 读无效异常（向量偏移 0X50）

在 CPU 取指或读数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面有效位时发生 TLB 读无效异常。

7.2.18 TLB 写无效异常（向量偏移 0X54）

在 CPU 写数据时，如果 MMU 的 jTLB 表项与虚拟地址匹配，但是匹配的表项关闭了页面有效位时发生 TLB 写无效异常。

7.3 异常优先级

如图所示，根据异常的特性和被处理的先后关系，C810 把优先级分为 10 级。在图 表 7.2 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 C810 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 7-4 中的优先级关系进行处理。

如果 PSR（EE）被清零了，当异常发生时，处理器处理的是不可恢复异常。

如果多个异常同时发生，拥有最高优先级的异常最先被处理。处理器在异常返回后，重新执行产生异常的指令时，其余的异常可以重现。

表 7.2: 异常向量分配

优先级	异常与它相关的优先级	特征
1	重启异常	处理器中止所有程序运行，初始化系统。
2	待处理的跟踪异常	如果 TP= 1，在 rte 或 rfi 指令退休后，处理器处理待处理的跟踪异常。
3	硬断点请求	在相关的指令退休后，硬断点请求被处理。
4	Idly4 错误	在相关的指令退休后，处理器保存上下文并处理异常。
5	不对齐错误	在相关的指令退休后，处理器保存上下文并处理异常。
6.0 6.1	快速中断 普通中断	如果 IC=0，中断在指令退休后被响应；如果 IC=1，处理器允许中断在指令完成之前就被响应。
7.0 7.1 7.2 7.3 7.4	不可恢复错误异常/TLB 不可恢复异常 TLB 读无效异常/TLB 写无效异常 TLB 修改异常 TLB 失配异常 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。

下页继续

表 7.2 – 续上页

优先级	异常与它相关的优先级	特征
8	非法指令 特权异常 禁止硬件加速器 除以零 陷阱指令 断点指令	在相关的指令退休后，处理器保存上下文并处理异常。
9	跟踪异常	在相关的指令退休后，处理器保存上下文并处理异常。

7.3.1 发生待处理的异常时调试请求

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

7.4 异常返回

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

第八章　接口信号

本章介绍 C810 处理器接口信号，并详细说明信号的特性。其中主要包括信号名，信号类型（输入、输出、双向输入输出等），并简要描述信号的功能。

8.1　时钟信号

C810 的时钟信号有三个，一个是处理器时钟 (cpuclk), 这是处理器内部逻辑的工作时钟，一个是系统时钟 (sysclk)，这是处理器的总线接口模块工作的时钟，主要负责与总线进行交互，另外一个是调试时钟 (jtagclk)，这是处理器的调试模块工作的时钟。

其中，cpuclk 和 sysclk 是同源不同频的时钟，频率上呈现倍数的关系，jtagclk 与 cpuclk/sysclk 呈现异步的关系。(cpuclk 频率/sysclk 频率）可以为 1/2/3/4/5/6/7/8 中的任意值。为了简化 C810 在后端实现过程中的时钟树处理，C810 采用了 cpuclk 配合着 clk_en 产生 sysclk 的行为，具体请参见《C810 集成手册》。

SoC 设计中由时钟生成模块（Clock Generator）产生 cpuclk 和 sysclk，并且根据 cpuclk 和 sysclk 产生 clk_en 信号。其中，cpuclk 和 clk_en 输入 C810，system 提供给总线 IP 和其它系统 IP。具体如 图 8.1 所示。

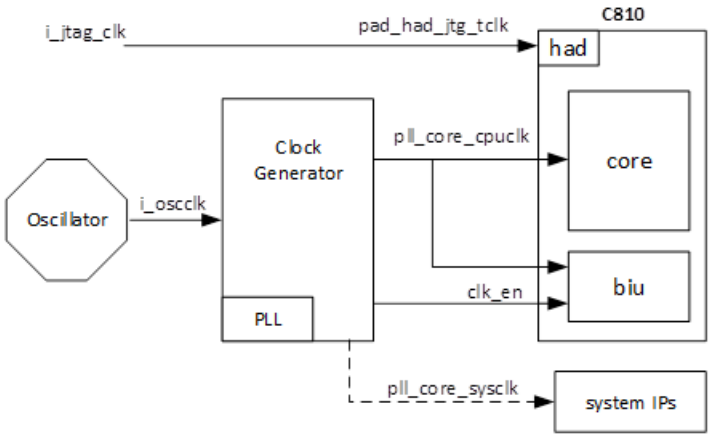


图 8.1: C810 的同步时钟管理方式

8.2 接口信号

本节中将解释，可能和其他系统模块和包进行集成的所有输入、输出和双向输入输出端口。

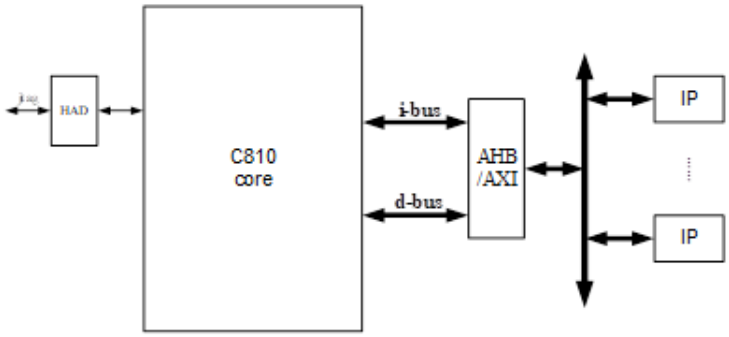


图 8.2: C810 存储器接口

8.2.1 信号命名规则

为了易于识别记忆，C810 接口信号使用了图表 表 8.1 的命名规则：

表 8.1: C810 接口信号命名规则

符号	含义
*	多组信号。
bist	BIST(Build-in Self Test) 相关信号。
biu	BIU(Bus Interface Unit) 相关信号。
core	CORE 相关信号。
had	HAD(Hardware Assist Debug) 相关信号。
jdb	JTAG(Joint Test Action Group) 硬件协同调试相关信号。
jtg	JTAG(Joint Test Action Group) 相关信号。
pad	PAD 相关信号。
pll	PLL(Phase-Locked Loop) 相关信号。
smbist	SMBIST(Simple Memory Build-in Self Test) 相关信号。
yy	多个目标单元的信号。

注意： _b 后缀用于表示低电平有效，没有 _b 后缀的信号高电平有效。

8.2.2 信号类型

图 8.3 根据功能对 C810 的总线和控制信号进行了分类。

8.2.3 信号总表

表 8.2 列出了 C810 各个信号的功能、类型，信号定义和 reset 之后的值。

表 8.2: C810 各个信号的功能、类型、定义

信号名	I/O	Reset	定义
AHB 总线主接口信号：			
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： 指示传输是一次突发传输的一部分： 000：SINGLE； 001：INCR； 010：WRAP4。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 指示传输的数据宽度： 000：byte； 001：halfword； 010：word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 指示当前总线周期的传输类型： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。
biu_pad_hwrite	O	0	读写表示信号： 指示当前 CPU 是读取总线数据还是写总线： 1 ： 指示是写总线传输； 0 ： 指示是读总线传输。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_hprot[3:0]	O	-	保护控制信号: 指示当前总线周期进行的数据传输的特性: ***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 C PU 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: 传输应答: 00: OKAY; 01: ERROR; 10: RETRY; 11: SPLIT。
复位控制信号:			
pad_cpu_rst_b	I	-	处理器复位信号: 低电平时，初始化 C810 内部及调试模块，C8 10 采用异步复位方式。
外部中断控制和低功耗握手信号:			
biu_pad_te_b	O	1	外部传输控制信号: 指示所要读取的地址应该经过一个外部的 MMU 转译。其对应于 P SR（处理器状态寄存器）的 TE 位，可以通过对 PSR 的修改控制传输。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_int_ack	O	0	C PU 中断响应信号信号: 指示 CPU 响应中断，即将进入中断服务程序处理中断。
pad_biu_int_b	I	-	中断请求信号: 低电平时表示进行普通中断申请。
pad_biu_fint_b	I	-	快速中断请求信号: 低电平时表示进行快速中断申请。
pad_biu_intraw_b	I	-	异步唤醒信号: 不请求进入中断，用于将 CPU 从低功耗模式中唤醒。这个信号会使 biu_pad_wakeup_b 和 biu_pad_ipend_b 有效。
pad_biu_fintraw_b	I	-	快速异步唤醒信号: 不请求进入中断，用于将 CPU 从低功耗模式中唤醒。这个信号会使 biu_pad_wakeup_b 和 biu_pad_ipend_b 有效。
biu_pad_ipend_b	O	1	异步唤醒确认信号: 指示异步唤醒请求，已经被处理器确认。
pad_biu_vec_b[7:0]	I	-	中断矢量序号信号: 提供 core 进行中断处理的向量号，如果 pad_biu_avec_b 为低电平，这个向量被忽略。
pad_biu_avec_b	I	-	中断向量有效控制信号: 如果为低电平，则普通中断和快速中断的向量入口取默认值；否则，普通中断和快速中断的向量入口取外部的输入（pa d_biu_vec_b[7:0]）。
功耗管理信号:			
biu_pad_lpmd_b[1:0]	O	11	低功耗模式状态信号: 当处理器执行 doze, stop 或 wait 指令时，biu_pad_lpmd_b[1:0] 被相应的改变: 00：STOP； 01：WAIT； 10：DOZE； 11：normal。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_wakeup_b	O	1	处理器唤醒指示信号： 低电平表示 CPU 被唤醒且进入正常工作模式，这个信号将会在 pad_biu_intraw_b， pad_biu_fintraw_b， pa d_biu_brkrq_b[1:0]， pad_biu_dbgrq_b 或 pad_had_jdb_req_b 有效之后有效。
状态和时钟信号：			
pll_core_cpuclk	I	-	内核工作时钟信号： 提供 CPU 内核工作的时钟。
clk_en	I	-	总线同步时钟信号： BIU 通过 clk_en 信号与 CPU 外部系统时钟保持同步。
p ad_biu_clkratio[2:0]	I	-	cpu 时钟与总线时钟比例信号： CPU 时钟频率/总线时钟频率 = pad_biu_clkratio +1： 000：1 比 1； 001：2 比 1； 011：4 比 1； 111：8 比 1； 其余：保留。
biu_pad_idly4_b	O	1	idly4 信号指示信号： 指示一个 idly4 指令序列。当处理器执行了 idly4 这条指令之后，这个信号被清零，直到 IDLY4 指令序列被执行完才释放。这个信号与系统时钟同步，这就意味着当 CLKRATIO 不等于 1(时钟比为 1:1) 时，这个信号就不一一对应于 CPU 指令执行的情况，用户应根据具体情况深刻理解这信号与内部指令执行的对应关系。
pad_biu_bigend_b	I	-	指令解析模式指示信号： 低电平时表示取回来的数据或指令是大端的字节顺序；这个信号为高电平时，表示取回来的数据或指令是小端的字节顺序。该信号只有在 CPU 复位的时候才可以设置，一旦 CPU 退出复位状态，这个信号就保持静态输入，不能发生变化。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
pad_biu_endian_v2	I	-	大小端版本指示信号： 1：v2 版本大小端； 0：v1 版本大小端。 注意：v1 版本和 v2 版本的小端完全一致。
全局状态和控制信号：			
biu_pad_gcb[31:0]	O	-	全局控制信号： 全局控制总线输出，用于控制外围设备和事件，它对应于 GCR 寄存器的低 12 位，因此可以通过 MTCR 修改 GCR 寄存器的值，达到控制外围设备的目的。当执行指令 MTCR 对全局控制寄存器进行更新时 biu_pad_g cb[11:0] 信号会改变。
pad_biu_gsb[31:0]	I	-	全局状态信号： 全局状态总线输入，用于检测外围设备和事件，它对应于 GSR 寄存器的低 12 位，但它只读。当执行指令 MFCR 将全局状态寄存器的值采样到一个通用寄存器中时，处理器对信号 pad_biu_gsb 进行采样。
cp0_pad_psr[31:0]	o	-	处理器状态信号： 处理器状态总线输出，使外围设备能监测处理器的状态，主要用于调试。
JTAG 支持信号：			
pad_had_jtg_trst_b	I	-	JTAG 测试复位信号： JTAG 复位信号，下跳变可以初始化和 JTAG 接口相关的触发器，正常工作情况下该信号需置成高电平。
pad_had_jtg_tclk	I	-	JTAG 测试时钟信号： JTAG 和 H AD 内部相关触发器的时钟信号，要求和 cpuclk 的频率比在 1：8 以上。
pad_had_jtg_tap_en	I	-	JTAG tap 控制器使能信号： 用于使能 JTAG ta p 控制器，另外信号 pad_had_jdb_req_b 也可以达到同样的使能效果。
had_pad_jtg_tap_on	O	-	表明 tap 控制器状态指示信号： 用于指示 tap 控制器是否工作，高电平有效。
pad_had_jtg2_sel	I	-	JTAG-5 JTAG-2 选择信号： 用于指示 jtag2 线五线选择，高电平表示 jtag 2，低电平表示 jtag5。
JTAG-5 支持信号：			

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
pad_had_jtg_tms	I	-	JT AG 测试模式选择信号: JTAG TAP 有限状态机状态跳转控制信号，该位可以控制命令的输入，数据的输入输出。
pad_had_jtg_tdi	I	-	JTAG 数据输入信号: JTAG 串行输入端口，包括控制命令，数据，输入顺序由低位到高位。
had_pad_jtg_tdo	O	-	JTAG 数据输出信号: JTAG 串行数据输出端口，输出顺序由低位到高位。
had_pad_jtg_tdo_en	O	-	TDO 输出使能信号: 用于支持多个并行的 JTAG 控制器，当 t ap 控制器处在 shift-dr 或 shift-ir 状态，并且 had_pad_jtg_tap_on 有效时，这个信号有效。
JTAG-2 接口信号:			
pad_had_jtg_tms_i	I	-	JTAG -2 串行数据输入信号: HAD 端在 JTAG 时钟信号（tc lk）的上升沿对其采样，而外部调试器在 tclk 的下降沿设置该信号。 空闲时，最好将该信号保持为高电平，同时停止 tclk。如果 tclk 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 HAD。
had_pad_jtg_tms_o	O		JTAG -2 串行数据输出信号: HAD 端在 tclk 的下降沿对其设置，而外部调试器在 tc lk 的上升沿对其采样。
had_pad_jtg_tms_oe	O	0	JTAG-2 串行数据输出有效指示信号: 要求 CPU 外部利用该信号通过一个三态门将 pad_had_jtg_tms_i 和 had _pad_jtg_tms_o 信号合为一个双向端口信号。
调试支持信号:			
pad_biu_dbgrq_b	I	-	外部调试请求信号: 该信号是外部调试请求信号，低电平有效，同步于 sysclk，使能处理器进入调试模式。 不用该信号时，需要接 1。
pad_biu_brkrq_b[1:0]	I	-	硬件断点请求: 该信号是外部硬件断点请求信号，低电平有效，可以使能处理器进入调试模式，同步于 sysclk。 不用该信号时，需要接 1。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
biu_pad_dbg_b	O	1	表明处理器进入调试模式： 该位信号指示处理器是否处于调试模式，该信号低电平有效，同步于 sysclk。
pad_had_jdb_req_b	I	-	外部调试请求信号： 该信号是外部调试请求信号，异步于 tclk，低电平有效。所以在 CPU 处于低功耗状态或者复位状态时，该信号有效可以快速请求 CPU 进入调试模式。 不用该信号时，需要接 1。
had_pad_jdb_ack_b	O	1	处理器响应进入调试模式： 处理器进入调试模式后，该响应信号保持两个 tclk 的低电平。
had_pad_jdb_pm[1:0]	O	00	处理器工作模式指示信号： 这些输出信号表明处理器的工作模式，异步于 pad_had_jtg_tclk。 00: normal 模式； 01: STOP/DOZE/WAIT 模式； 10：调试模式； 11：保留。
测试支持信号：			
pad_yy_test_mode	I	-	进入测试模式： 使处理器进入测试模式，此时的 cpu 时钟和系统时钟均为测试时钟 （pad_had_jtg_tclk）， 只有处理器进入测试模式， 并且 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。
px_t op_si_12···px_top_si_1	I	-	扫描链 scan in
px_t op_so_12···px_top_so_1	O	-	扫描链 scan out
pad_yy_scan_enable	I	-	扫描使能: 扫描链的使能控制信号，高电平有效。 不用该信号时，需要接 0。
pad_yy_gate_clk_en_b	I	-	门控时钟使能信号: 处理器的门控时钟使能控制信号，只有当这个信号有效时，cpu 的内部模块的门控时钟才能有效。 不用该信号时，需要接 1。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
Memory Bist 信号 *:（视处理器配置而定）			
b ht_array_smbist_done	O	0	bist 测试完成信号: 0：表明 BHT array memory 自测试未完成; 1：表明 BHT array memory 自测试完成。
b ht_array_smbist_fail	O	0	bist 测试失败信号: 0:表明 BHT array memory 自测试成功,memory 工作正常; 1：表明 BHT array memory 自测试失败。
btb_data_smbist_done	O	0	bist 测试完成信号: 0：表明 BTB data memory 自测试未完成; 1：表明 BTB data memory 自测试完成。
btb_data_smbist_fail	O	0	bist 测试失败信号: 0:表明 BTB data memory 自测试成功, memory 工作正常; 1：表明 BTB data memory 自测试失败。
btb_tag_smbist_done	O	0	bist 测试完成信号: 0：表明 BTB tag memory 自测试未完成; 1：表明 BTB tag memory 自测试完成。
btb_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明 BTB tag memory 自测试成功，memory 工作正常; 1：表明 BTB tag memory 自测试失败。
dcache_* da_array_smbist_done	O	0	bist 测试完成信号: 0：表明数据 cache 的 data memory 自测试未完成; 1：表明数据 cache 的 data memory 自测试完成。
dcache_* da_array_smbist_fail	O	0	bist 测试失败信号: 0：表明数据 cache 的 data memory 自测试成功，memory 工作正常; 1：表明数据 cache 的 data memory 自测试失败。
dcac he_dirty_smbist_done	O	0	bist 测试完成信号: 0：表明数据 cache 的 dirty 位 memory 自测试未完成; 1：表明数据 cache 的 dirty 位 memory 自测试完成。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
dcac he_dirty_smbist_fail	O	0	bist 测试失败信号: 0：表明数据 cache 的 d irty 位 memory 自测试成功，memory 工作正常； 1：表明数据 cache 的 dirty 位 memory 自测试失败。
dc ache_tag_smbist_done	O	0	bist 测试完成信号: 0 ： 表明数据 cache 的 tag memory 自测试未完成； 1 ： 表明数据 cache 的 tag memory 自测试完成。
dc ache_tag_smbist_fail	O	0	bist 测试失败信号: 0 ： 表明数据 cache 的 tag memory 自测试成功，memory 工作正常； 1 ： 表明数据 cache 的 tag memory 自测试失败。
icache_da ta_array_smbist_done	O	0	bist 测试完成信号: 0：表明指令 cache 的 data memory 自测试未完成； 1：表明指令 cache 的 data memory 自测试完成。
icache_da ta_array_smbist_fail	O	0	bist 测试失败信号: 0：表明指令 cache 的 data memory 自测试成功，memory 工作正常； 1：表明指令 cache 的 data memory 自测试失败。
ic ache_tag_smbist_done	O	0	bist 测试完成信号: 0 ： 表明指令 cache 的 tag memory 自测试未完成； 1 ： 表明指令 cache 的 tag memory 自测试完成。
ic ache_tag_smbist_fail	O	0	bist 测试失败信号: 0 ： 表明指令 cache 的 tag memory 自测试成功，memory 工作正常； 1 ： 表明指令 cache 的 tag memory 自测试失败。
jtlb_da_smbist_done	O	0	bist 测试完成信号: 0：表明 jtlb 的 data memory 自测试未完成； 1 ： 表明表明 jtlb 的 data memory 自测试完成。
jtlb_da_smbist_fail	O	0	bist 测试失败信号: 0 ： 表明表明 jtlb 的 data memory 自测试成功，memory 工作正常； 1 ： 表明表明 jtlb 的 data memory 自测试失败。

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
jtlb_tg_smbist_done	O	0	bist 测试完成信号: 0：表明 jtlb 的 tag memory 自测试未完成; 1：表明 jtlb 的 tag memory 自测试完成。
jtlb_tg_smbist_fail	O	0	bist 测试失败信号: 0:表明 jtlb 的 tag memory 自测试成功, memory 工作正常; 1：表明 jtlb 的 tag memory 自测试失败。
l0_ica che_data_smbist_done	O	0	bist 测试完成信号: 0：表明 l0 指令 cache 的 data memory 自测试未完成; 1：l0 指令 cache 的 data memory 自测试完成。
l0_ica che_data_smbist_fail	O	0	bist 测试失败信号: 0：表明 l0 指令 cache 的 data memory 自测试成功，memory 工作正常; 1：表明 l0 指令 cache 的 data memory 自测试失败。
l0_ic ache_tag_smbist_done	O	0	bist 测试完成信号: 0：表明 l0 指令 cache 的 tag memory 自测试未完成; 1：表明 l0 指令 cache 的 tag memory 自测试完成。
l0_ic ache_tag_smbist_fail	O	0	bist 测试失败信号: 0：表明 l0 指令 cache 的 tag memory 自测试成功，memory 工作正常; 1：表明 l0 指令 cache 的 tag memory 自测试失败。
pad_yy_bist_tst_en	I	-	bist 测试使能信号: 不用该信号时，需要接 0。
指令退休和结果回写信号:			
rtu_pad_retire0	O	0	第 0 路指令退休指示信号: 0：当前周期没有指令退休 1 ： 当前周期有指令退休
rtu_* pad_retire0_pc[31:0]	O	-	第 0 路退休指令的 PC: 表明当前正在退休的指令的 PC
rtu_pad_retire1	O	0	第 1 路指令退休指示信号: 0：当前周期没有指令退休 1 ： 当前周期有指令退休

下页继续

表 8.2 – 续上页

信号名	I/O	Reset	定义
rtu_* pad_retire1_pc[31:0]	O	-	第 1 路退休指令的 PC: 表明当前正在退休的指令的 PC
rtu_pad_wb0_c	O	-	第 0 路回写的 C 位: 表明第 0 路当前回写的 C 位的值
rtu_pad_wb0_c_vld	O	0	第 0 路回写 C 位有效位: 0：表明当前周期 C 位的值无效 1：表明当前周期 C 位的值有效
rtu_pad_wb1_c	O	-	第 1 路回写的 C 位: 表明第 0 路当前回写的 C 位的值
rtu_pad_wb1_c_vld	O	0	第 1 路回写 C 位有效位: 0：表明当前周期 C 位的值无效 1：表明当前周期 C 位的值有效
rt u_pad_wb0_data[31:0]	O	-	第 0 路回写数据: 表明第 0 路当前的回写数据
rtu_pad_wb0_data_vld	O	0	第 0 路回写数据有效位: 0：表明当前周期的回写数据无效 1：表明当前周期的回写数据有效
rt u_pad_wb1_data[31:0]	O	-	第 1 路回写数据: 表明第 1 路当前的回写数据
rtu_pad_wb1_data_vld	O	0	第 1 路回写数据有效位: 0：表明当前周期的回写数据无效 1：表明当前周期的回写数据有效
rt u_pad_wb0_index[4:0]	O	-	第 0 路回写寄存器索引: 表明当前回写的寄存器索引
rt u_pad_wb1_index[4:0]	O	-	第 1 路回写寄存器索引: 表明当前回写的寄存器索引
64bit 累加器信号:			
mad_pad_hi[31:0]	O	-	高 32bit 累加器数据: 表示给总线的高 32bit 累加器数据
mad_pad_hi_wb_vld	O	0	高 32bit 累加器数据有效信号: 0：表示给总线的高 3 2bit 累加器数据无效; 1：表示给总线的高 32bit 累加器数据有效
mad_pad_lo[31:0]	O	-	低 32bit 累加器数据: 表示给总线的低 32bit 累加器数据
mad_pad_lo_wb_vld	O	0	低 32bit 累加器数据有效信号: 0：表示给总线的低 3 2bit 累加器数据无效; 1：表示给总线的低 32bit 累加器数据有效

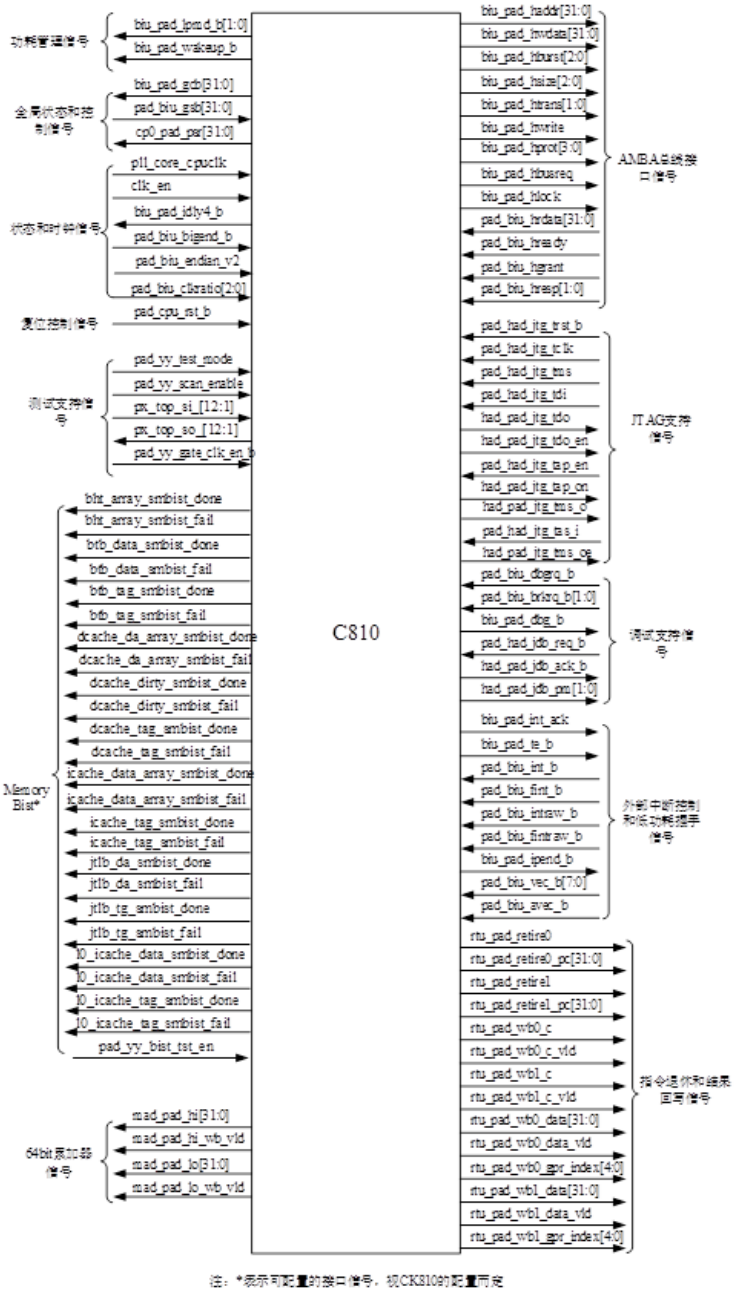


图 8.3: C810 顶层接口信号

第九章　接口总线协议

本章描述了 C810 的处理器接口总线协议。C810 的总线支持在处理器和其他外围设备之间同步传输数据。包括数据存取、地址控制和总线控制。

9.1　总线接口信号列表

图表 表 9.1 列出了 C810 处理器总线接口信号。

表 9.1: 总线接口信号描述列表

信号名	简称	I/O	注释
biu_pad_haddr[31:0]	HADDR	O	32 位地址总线。
biu_pad_hwdata[31:0]	HWDATA	O	32 位写数据。
biu_pad_hburst[2:0]	HBURST	O	指示传输是否 burst 传输及 burst 类型: 000:SINGLE Single transfer;
biu_pad_hsize[2:0]	HSIZE	O	指示传输的尺寸。
biu_pad_htrans[1:0]	HTRANS	O	指示传输类型，应当是 00:IDLE 01:BUSY 10:NONSEQ 11:SEQ 中的一种。
biu_pad_hwrite	HWRITE	O	1 - 指示是写传输; 0 - 指示是读传输。
biu_pad_hprot[3:0]	HPROT	O	保护控制信号。
biu_pad_hbusreq	HBUSREQ	O	总线请求信号。
biu_pad_hlock	HLOCK	O	表示总线已被 CPU 锁定。
pad_biu_hrdata[31:0]	HRDATA	I	32 读数据。
pad_biu_hready	HREADY	I	指示当前传输完成。
pad_biu_hgrant	HGRANT	I	指示 CPU 当前已占有总线。

下页继续

表 9.1 – 续上页

信号名	简称	I/O	注释
pad_biu_hresp[1:0]	HRESP	I	传输应答: 00: OKAY 01: ERROR 10: RETRY 11: SPLIT。

注意: I 表示输入，O 表示输出，带有 _b 后缀的表示低电平有效。

9.2 系统总线传输协议

C810 的外部总线传输协议和 AMBA2.0 的 AHB 兼容，所以我们可以参考 AMBA 2.0 规格说明(AMBA™ Specification Rev 2.0)。

9.2.1 概览

总线接口单元负责 CPU Core 和 AHB 之间的地址控制和数据传输，其基本特点有:

- 与 AMBA2.0 总线协议兼容;
- 支持 32 位和 64 位两种总线宽度;
- 32 位总线支持 SINGLE、INCR 和 WRAP8 三种传输模式;
- 64 位总线支持 SINGLE 和 WRAP4 两种传输模式;
- 通过配置不同 Core-to-bus 时钟频率比可动态变换时钟频率;
- 当执行 idly 指令时锁住总线;
- 所有的信号都 flopped out 以获得较好的时序;
- 只有时钟上升沿时有操作;
- 没有三态执行;
- 支持 RETRY 和 SPLIT 响应。

9.2.2 一些典型的传输模式

CPU 发出 HBUSREQ 请求，等仲裁器给 CPU HGRANT 应答后，CPU 就占有了总线，可以向从设备发起传输。一个 AHB 传输包括两个不同的阶段:

- adress phase: 只持续一个时钟周期;

- data phase：通过使用 HREADY 信号可以持续多个时钟周期。

9.2.2.1 基本传输

下图表示了一个最简单的传输，没有等待状态。

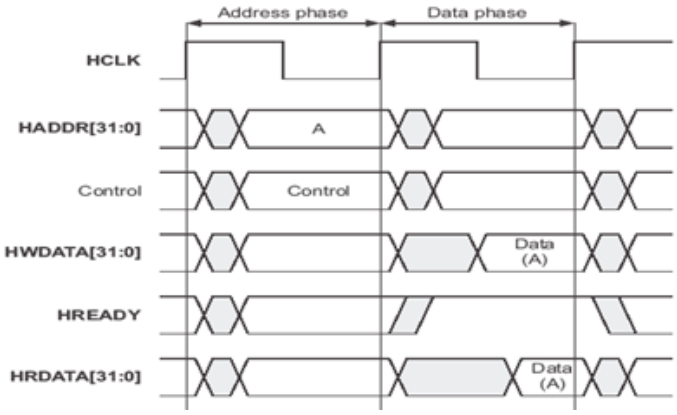


图 9.1: 简单传输

在一个没有等待状态的简单传输中：

- 主设备（C810，下同）在 HCLK 的上升沿后向总线驱动地址和控制信号。
- 从设备在下一个 clock 的上升沿时获取地址和控制信息。
- 之后次设备开始发出合适的响应，总线主设备在 clock 的第三个上升沿获取这个响应。

实际上，任何传输的 adress phase 在前一次传输的 data phase 就已产生了。这个地址和数据的重叠是这个总线的流水线特性的基础以获得高性能操作，同时仍可提供充分的时间给次设备发出传输的响应。

一个次设备可以向任何传输插入等待状态，如 图 9.2 所示，延长了传输完成的时间。

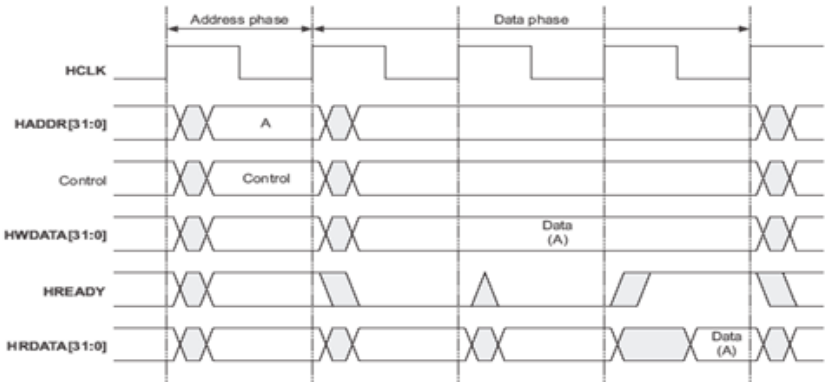


图 9.2: 有等待状态的传输

对于写操作主设备会在扩展的时钟周期里保持数据稳定，对于读传输次设备不需要提供有效数据直到传输快要完成。

当传输以这种方法扩展时它会产生扩展下一个传输的 address phase 的副作用。图 9.3 展示了对不相关的 A，B，C，三个地址的传输。

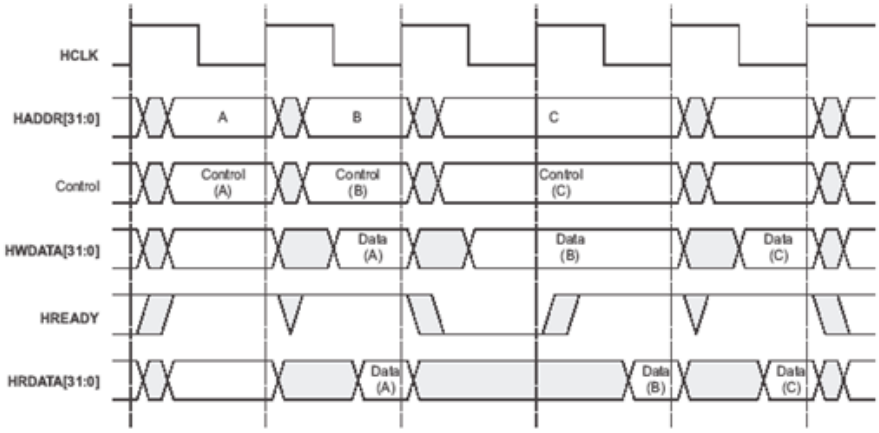


图 9.3: 多重传输

在上图中，向地址 A 和 C 的传输是零等待状态，向地址的传输是有一个等待状态，扩展向地址 B 的传输的 data phase 扩展了向地址 C 传输的 address phase。

9.2.2.2 传输类型

每次传输可以归类于四种不同传输类型中的一种，由 HTRANS[1:0] 信号指示，具体见 表 9.2 。

表 9.2: 传输类型编码

HTRANS[1:0]	类型	描述
00	IDLE	指示此时没有数据传输被请求。当总线主设备已占有总线但还没有数据传输时，便是 IDLE 传输类型。次设备必须总是提供一个零等待状态 OKAY 响应给 I DLE 传输，并且这个传输应被次设备忽略。
01	BUSY	BUSY 传输类型允许总线主设备在突发传输中插入 IDLE 周期。这种传输类型指示总线主设备在继续一个突发传输中，但是下一个不能立即发生。当主设备使用 BUSY 传输类型地址和控制信号必须反映出突发传输模式中的下一个传输。这种传输应被次设备忽略。次设备必须总是提供一个零等待状态 OKAY 响应，和在 IDLE 传输中的情况一样。
10	NONSEQ	指示突发传输中的第一个传输或者一单个传输。地址和控制信号和前次传输无关。总线上的单个传输被认为单个的突发传输，所以传输类型是 NONSE-QUENTIAL。

下页继续

表 9.2 – 续上页

HTRANS[1:0]	类型	描述
11	SEQ	突发传输模式里剩下的传输是 SEQUENTIAL 并且地址和先前的传输相关。控制信息和先前的传输相同。address 等于先前的 address 加上传输 size（以字节单位）。在回绕突发传输模式中传输的地址在地址界限处回绕。

9.2.2.3 传输响应

CPU 开始一次传输之后，次设备决定传输如何进行。每当次设备被访问他必须提供一个响应指示传输的状态。HREADY 信号用来扩展传输，它和给 CPU 传输的响应的 HRESP[1:0] 一起工作。

表 9.3: HRESP[1:0] 响应类型描述

HRESP[1:0]	响应类型	描述
00	OKAY	当 HREADY 变高 OKAY 表示传输成功完成。OKAY 循环也用在任何插入增加的循环，先于其他三个响应给出。
01	ERROR	这个响应表示有一个错误发生了。错误状态需要告知 CPU 使之知道传输不成功。ERROR 响应需要保持两个时钟周期。
10	RETRY	RETRY 响应表示传输尚未完成，所以 CPU 需要重试这次传输直到完成传输。RETRY 响应需要保持两个时钟周期。
11	SPLIT	传输尚未成功完成。CPU 当它下次被授予总线访问权时必须重试传输。当传输能完成的时候，次设备将会代表主设备请求对总线的访问。SPLIT 响应需要保持两个时钟周期（对此响应的操作 C810 可扩展）。

只有 OKAY 响应可只在单个周期给出。ERROR，SPLIT 和 RETRY 响应需要至少两个周期。为完成这些响应在倒数第二周期次设备驱动 HRESP[1:0] 指示 ERROR，SPLIT 或 RETRY 同时驱动 HREADY 为低电平以扩展一额外的传输周期。最后驱动 HREADY 为高电平以结束传输同时响应置为 OKAY。

对于 SPLIT 和 RETRY 接下来的传输必须被取消。对于 ERROR，由于进行中的传输不被重复，是否进行接下来的传输是可选择的。

图 9.4 表示在一次传输中次设备需要一个周期决定发出哪个响应（这时 HRESP 指示 OKAY）然后次设备以两个周期的 ERROR 响应结束了传输。CPU 收到 ERROR 响应后将跳到异常服务程序里。

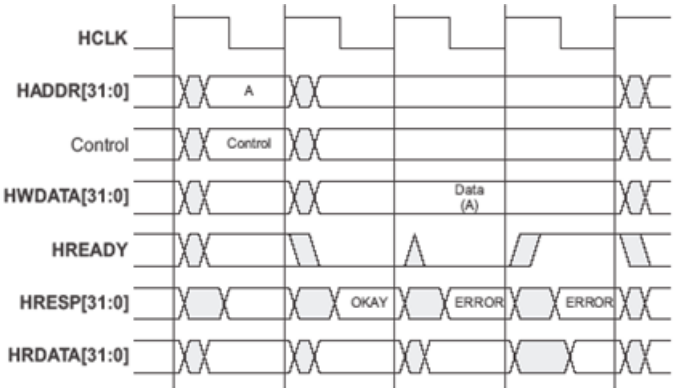


图 9.4: 错误响应

9.2.2.4 突发传输操作

HBURST[2:0] 指示了传输是否是 burst 传输及 burst 类型。

32 位总线支持以下三种传输：

- 当 HBURST[2:0] 为 000 时，代表单个传输（SINGLE）；
- 当 HBURST[2:0] 为 001 时，代表未定长度增量突发传输（INCR）；
- 当 HBURST[2:0] 为 100 时，代表八拍回绕突发传输（WRAP8）。

64 位总线支持以下两种传输：

- 当 HBURST[2:0] 为 000 时，代表单个传输（SINGLE）；
- 当 HBURST[2:0] 为 010 时，代表四拍回绕突发传输（WRAP4）。

未定长度增量突发传输 (INCR) 访问连续的位置，burst 中每次传输的地址是前一次传输地址的增量。一次增量突发传输可以是任意长的，上限是不可超过 1kB 地址范围。可以使用未定长度增量突发传输模式只执行一次传输以执行单个的传输。

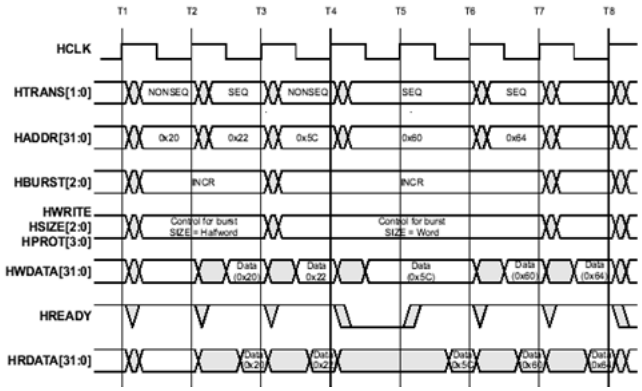


图 9.5: 增量突发传输

在上图中，两次 halfword 的传输开始于地址 0x20，halfword 传输地址加 2。word 传输开始于地址 0x5C，地址增量是 4。

对于回绕突发传输，如果传输的开始地址和 burst 的总传输比特数不对齐，当传输地址达到此次 burst 地址界限时将回绕。对于 C810 来说，一个以 word（4byte）为单位的 4 beat wrapping burst 将在 16byte 界限处回绕。就是说，如果如果传输的开始地址是 0x38，它包含的 4 个传输地址顺序就是 0x38，0x3C，0x30 和 0x34。如下图中所展示的那样。

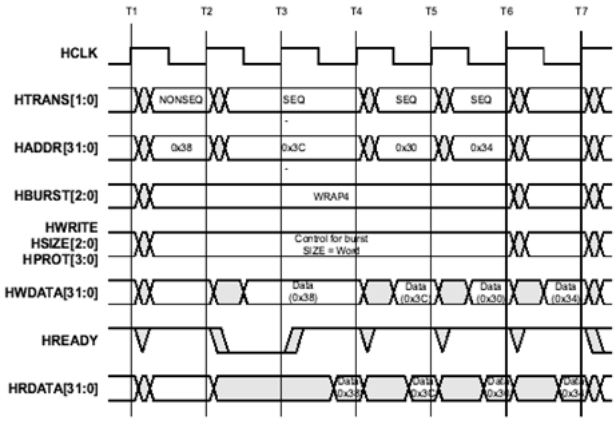


图 9.6: 四拍回绕突发传输

如果 CPU 由于失去了总线所有权不能完成一个 burst，当它下次能访问总线时会以合适的方法重建 burst。例如，如果 CPU 只完成了 4beat burst 中的 1 beat，然后它会以 SINGLE 传输的方式执行剩下的 3 次传输。

9.2.3 总线异常

表 9.4 表示了总线上出现异常时 CPU 的行为：

表 9.4: 总线异常控制循环

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

第十章　工作模式转换

C810 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之处在于工作时停止的时钟不同。本章将详细解析 CPU 的工作模式以及模式之间的转换。

10.1　C810 工作模式及其转换

如图所示，C810 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 biu_pad_jdb_pm[1:0] 信号得到。

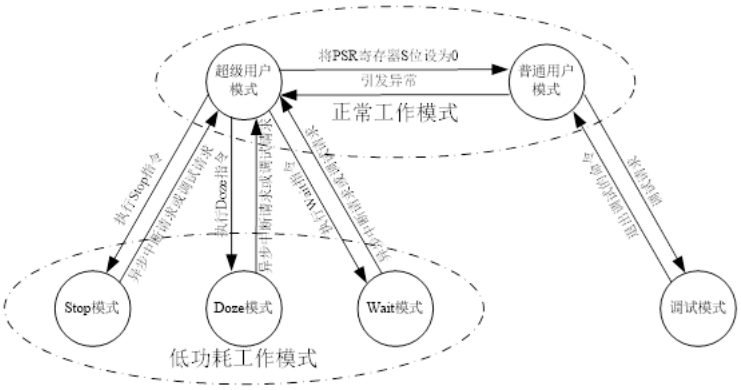


图 10.1: CPU 的各种工作状态示意图

10.1.1　正常工作模式

CPU 的正常工作模式可以分为两种，超级用户模式和普通用户模式，CPU 处于哪种正常工作模式通过查询 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。

10.1.2 低功耗模式

当 CPU 执行完低功耗指令 (STOP、DOZE、WAIT) 之后，CPU 将进入相应的低功耗模式。CPU 进入低功耗模式之后，CPU 的时钟被停止，只有异步中断请求（pad_biu_intraw_b 和 pad_biu_fintraw_b 信号）或者调试请求才能正常地退出低功耗模式。当前的 CPU 工作于哪种低功耗模式可以通过查询信号线 biu_pad_lpmd_b[1:0] 来得到。

10.1.2.1 DOZE 工作模式

当 CPU 执行完 DOZE 指令后，进入 DOZE 低功耗模式，此时 CPU 的时钟停止，哪些外设的时钟停掉与实现相关。

10.1.2.2 STOP 工作模式

当 CPU 执行完 STOP 指令后，进入 STOP 低功耗模式，此时 CPU 和大多数外设的时钟都被停止

10.1.2.3 WAIT 工作模式

当 CPU 执行完 WAIT 指令之后，进入 WAIT 低功耗模式，此时 CPU 停止工作，大多数的外设仍然在运行并可以产生中断。

10.1.3 调试模式

10.1.3.1 进入调试模式

当 CPU 接到调试请求之后，进入调试模式，其中的调试请求源可以有以下 7 种：

- 在低功耗模式下，可以通过使能 pad_had_jdb_req_b 信号进入调试模式；
- 在正常工作模式下，可以通过使能 pad_biu_dbgrq_b 信号进入调试模式；
- 当 C810 CSR 的 FDB 位有效时，pad_biu_brkrq_b[1:0] 信号有效进入调试模式；
- 当 C810 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式；
- 当 C810 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 C810 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 C810 存储器断点调试模式下，当 MBCA 的值为 0 时，如果当前执行的指令符合断点要求，处理器进入调试模式。

10.1.3.2 退出调试模式

如果 C810 HACR 的 GO、EX 位被置为 1 时，并且当前的操作是对 CPUSCR 或者是对“没有选中任何寄存器”进行读写，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，CPUSCR 中的值必须是刚进入调试模式之时保存过的值。

第十一章　C810 的双总线架构（可选）

为了满足 SoC 多总线架构设计的需求，简化多总线架构的设计复杂度，C810 在原有单条总线的基础上，增加了双总线接口的配置。下图分别描述了 LSU 和多个 BIU 以及 IFU 与多个 BIU 的内部连接示意图。

11.1　LSU 的多总线访问

当 LSU 发起一次总线访问请求时，在单条总线的架构下，会直接通过 BIU 接口，根据 SoC 的总线类型，转换成相应的总线传输行为进行传输。当有多条总线的架构时，LSU 发起一次总线访问请求时，通过内部的分路器，根据地址区间，按照如下的逻辑进行仲裁。

biu1_hit = (lsu_request_addr & pad_div_region1_page_mask) == pad_div_region1_base_addr;

当命中在 biu1 上时（biu1_hit 为 1’ b1），LSU 所发起的读写请求均会通过 BIU1 接口向核外传输；如果 biu1 不命中（biu1_hit 为 1’ b0），LSU 所发起的读写请求均会通过 BIU0 接口向核外发起请求。

注意： SoC 的设计者需要确保 pad_div_region1_base_addr 与 pad_div_region1_page_mask 对齐。

11.2　IFU 的多总线访问

当 IFU 发起一次总线读请求时，在单条总线的架构下，会直接通过 BIU 接口，根据 SoC 的总线类型，转换成相应的总线传输行为进行读传输。当有多条总线的架构时，IFU 发起一次总线读请求时，通过内部的分路器，根据地址区间，按照如下的逻辑进行仲裁。

biu1_hit = (ifu_request_addr & pad_div_region1_page_mask) == pad_div_region1_base_addr;

当命中在 biu1 上时（biu1_hit 为 1’ b1），IFU 所发起的读请求均会通过 BIU1 接口向核外传输；如果 biu1 不命中（biu1_hit 为 1’ b0），LSU 所发起的读请求均会通过 BIU0 接口向核外发起请求。

注意： SoC 的设计者需要确保 pad_div_region1_base_addr 与 pad_div_region1_page_mask 对齐。

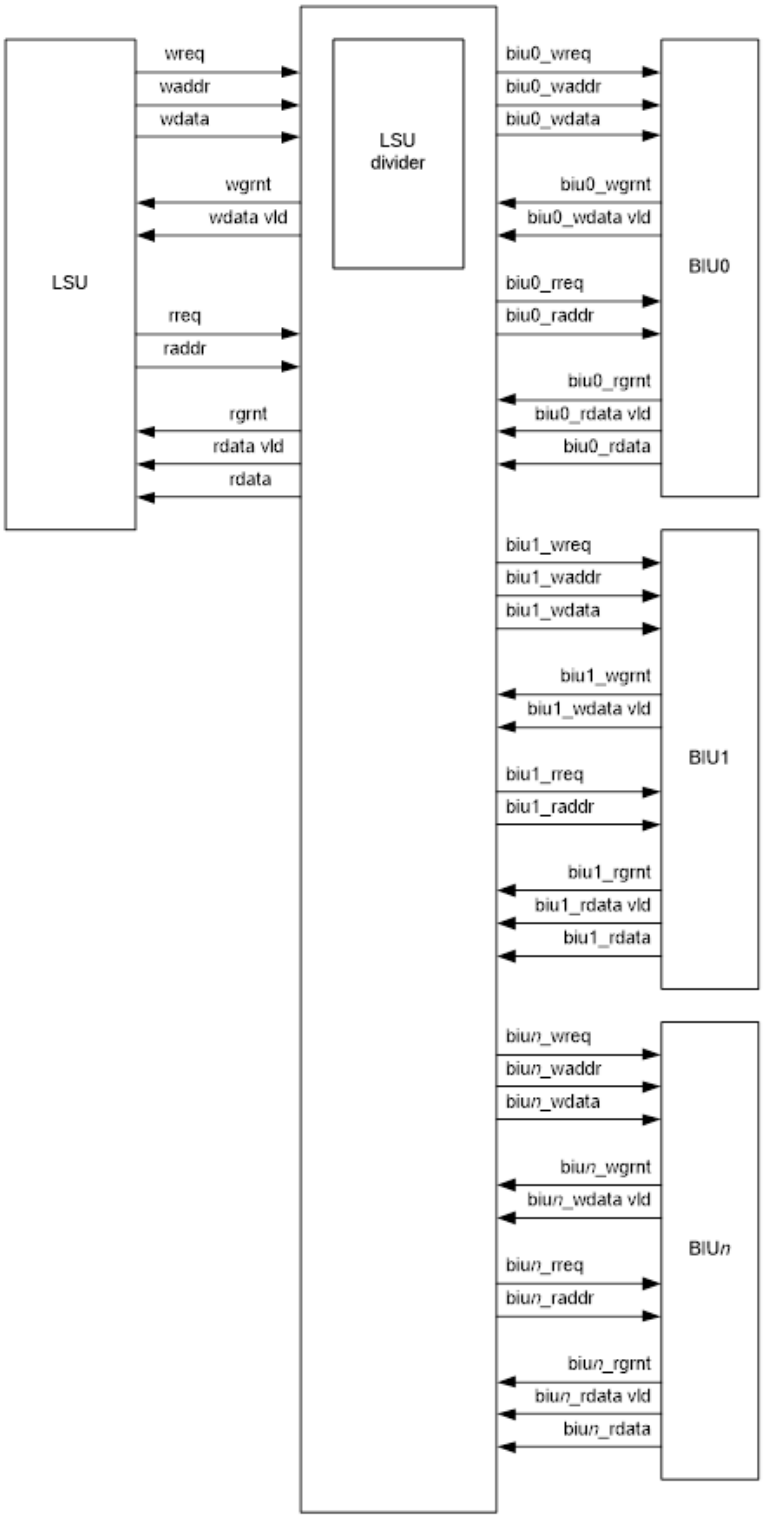


图 11.1: LSU 与多个 BIU 的连接示意图

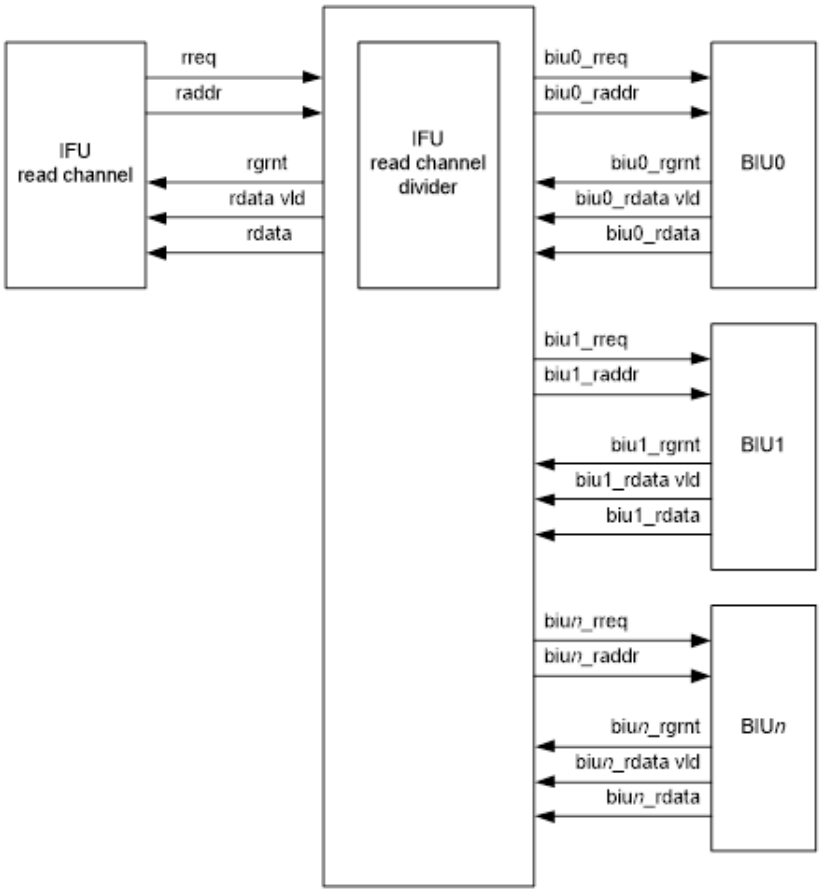


图 11.2: IFU 与多个 BIU 的连接示意图

第十二章　附录 A MMU 设置示例

```
/*************************************************************************
* Function: An example of set C810 CPU’ s MMU TLB.
* Enable/disable C810 CPU data/instruction cache.
* Memory space: Virtual address <-> physical address.
*
* virtual address: 0K-4K <-> physical address: 0K-4K
* virtual address: 4K-8K <-> physical address: 4K-8K
* virtual address: 8K-12K <-> physical address: 16K-20K
* virtual address: 12K-16K <-> physical address: 8K-12K
*************************************************************************/
/invalid all MMU TLB Entry/
movi r1,0x0
btsti r1,26
mctr r1,cr<8,15>
/* virtual address: 0K-4K <-> physical address: 0K-4K*/
/* virtual address: 4K-8K <-> physical address: 4K-8K*/
movi r1,0x1e
mtcr r1,cr<2,15> //mel0,PPN0:0x0
lrw r1,0x101e
mtcr r1,cp<3,15> //mel1,PPN1:0x1
movi r1,0
mtcr r1,cr<4,15> //meh,VPN:0x0 & 0x1
mtcr r1,cr<6.15> //page size:4K
```

```
bseti r1,28
mtcr r1,cr<8,15> //write to jTLB
/* virtual address: 8K-12K <-> physical address: 16K-20K*/
/* virtual address: 12K-16K <-> physical address: 8K-12K*/
movi r1,0x401e
mtcr r1,cr<2,15> //mel0,PPN0:0x4
lrw r1,0x201e
mtcr r1,cr<3,15> //mel1,PPN1:0x2
lrw r1,0x2000
mtcr r1,cr<4.15> //meh,VPN:0x2 & 0x3
movi r1,0
mtcr r1,cr<6,15> //page size:4K
bseti r1,28
mtcr r1,cr<8,15> //write to jTLB
/* enable mmu*/
mfcr r1,cr18
bseti r1,0x0
mtcr r1,cr18 //enable mmu
```

注意: 注意：使能 MMU 的这条 mtcr 语句所在的 PC 地址空间必须是映射过的，并已写入 TLB，保证当 MMU 开启时，指令所在的 PC 地址空间可以完成异常，不出异常。

第十三章　附录 B 指令术语表

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

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

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

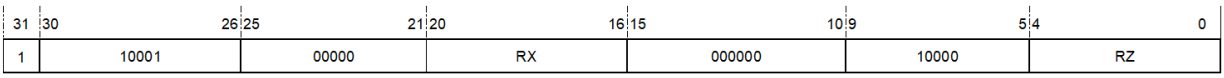
其中，指令中文名称中带 # 的为伪指令。

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：

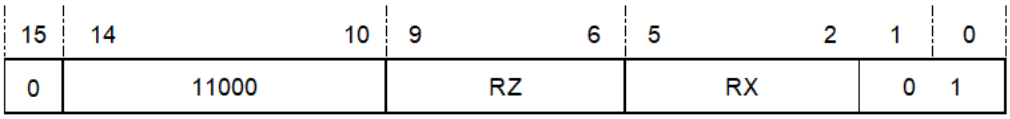


图 13.1: ADDC-1

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：

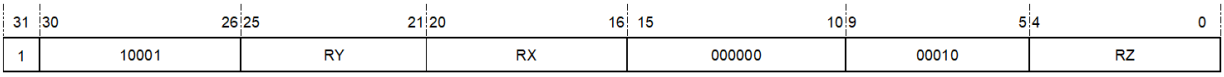


图 13.2: ADDC-2

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：

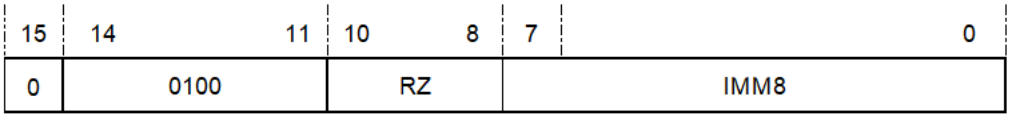


图 13.3: ADDI-1

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：

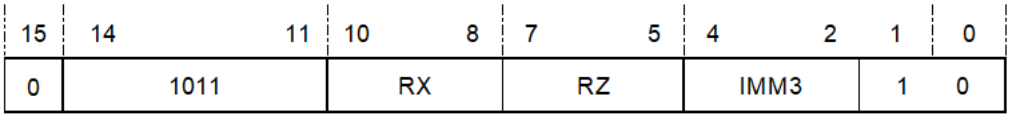


图 13.4: ADDI-2

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：

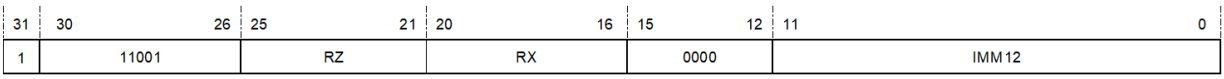


图 13.5: ADDI-3

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：

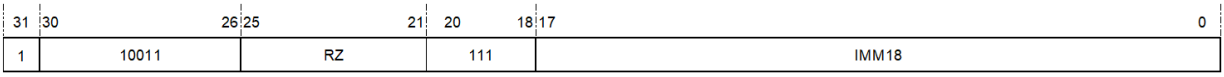


图 13.6: ADDI-4

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：

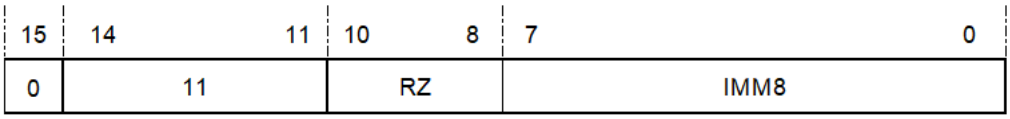


图 13.7: ADDI(SP)-1

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：

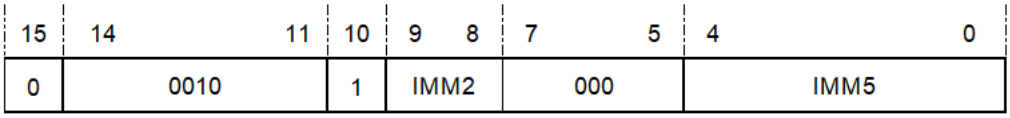


图 13.8: ADDI(SP)-2

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：

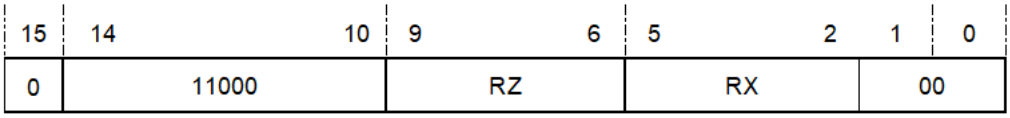


图 13.9: ADDU-1

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：

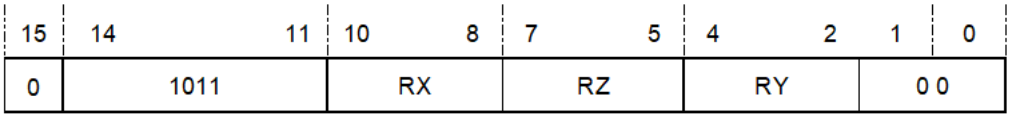


图 13.10: ADDU-2

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：

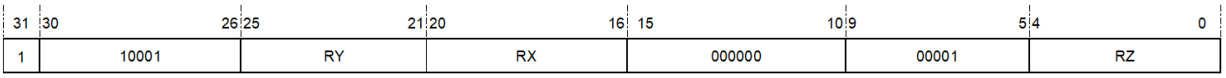


图 13.11: ADDU-3

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：

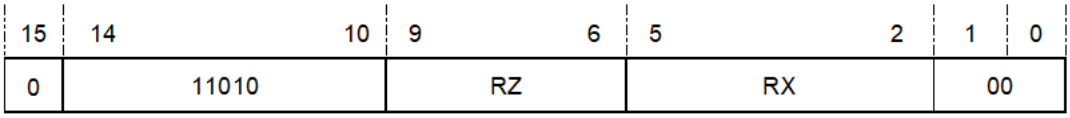


图 13.12: AND-1

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：

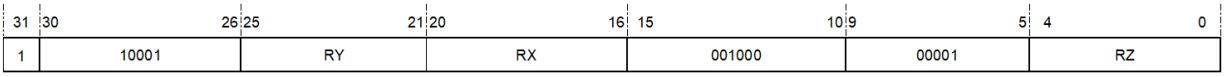


图 13.13: AND-2

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：

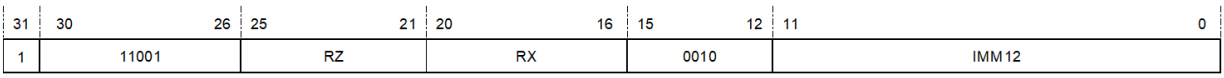


图 13.14: ANDI

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：

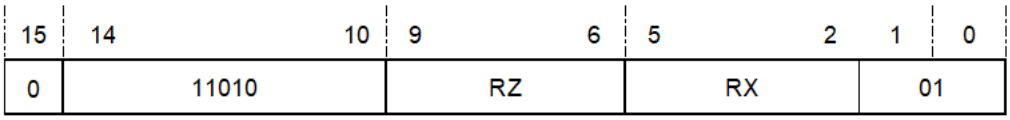


图 13.15: ANDN-1

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：

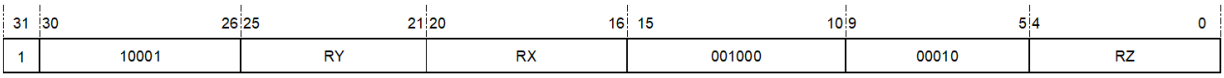


图 13.16: ANDN-2

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：

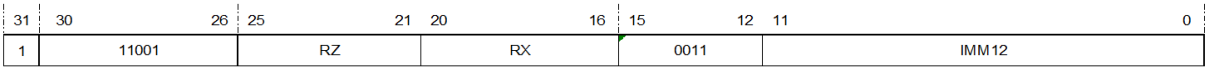


图 13.17: ANDNI

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：

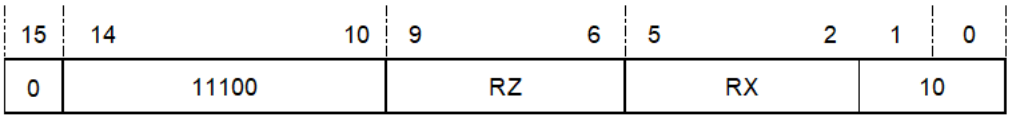


图 13.18: ASR-1

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：

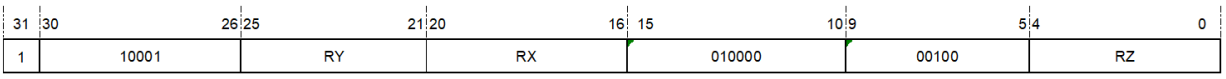


图 13.19: ASR-2

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：

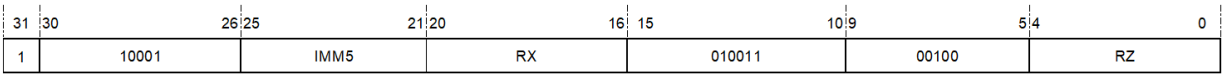


图 13.20: ASRC

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：

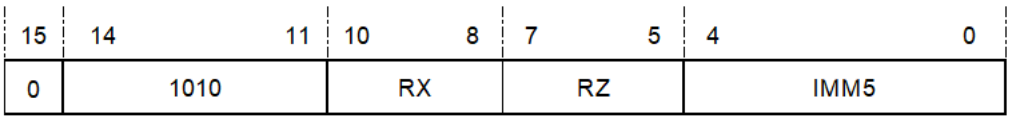


图 13.21: ASRI-1

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：

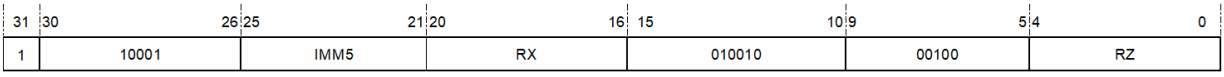


图 13.22: ASRI-2

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：

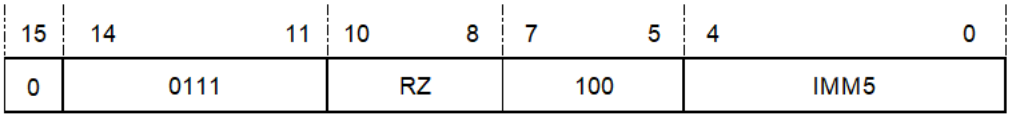


图 13.23: BCLRI-1

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：

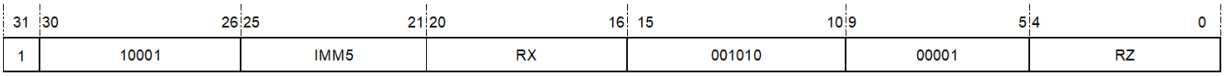


图 13.24: BCLRI-2

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：

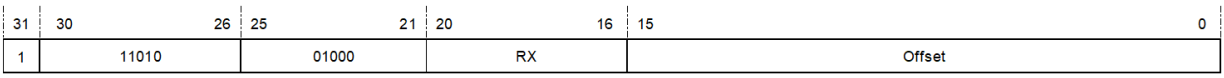


图 13.25: BEZ

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：

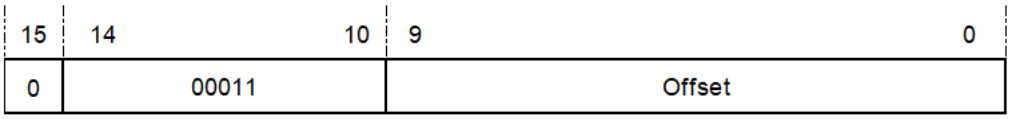


图 13.26: BF-1

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：

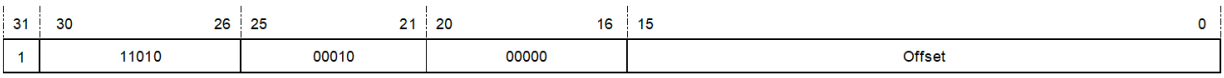


图 13.27: BF-2

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：

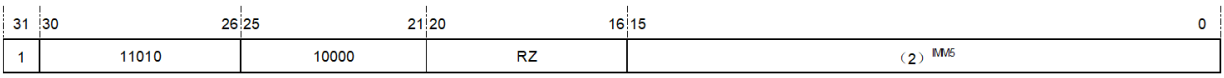


图 13.28: BGENI-1

如果 IMM5 大于等于 16：

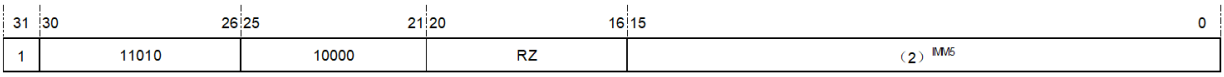


图 13.29: BGENI-2

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：

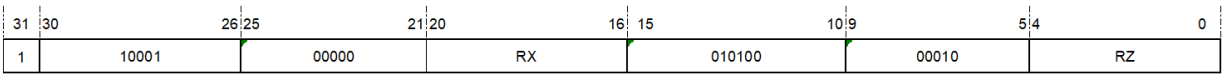


图 13.30: BGENR

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：

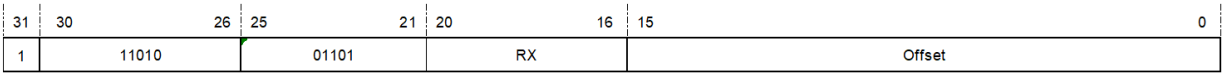


图 13.31: BHSZ

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：

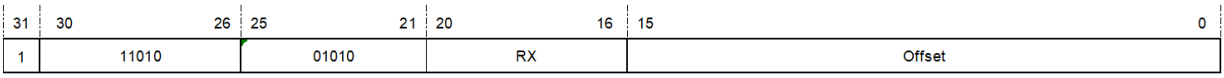


图 13.32: BHZ

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：

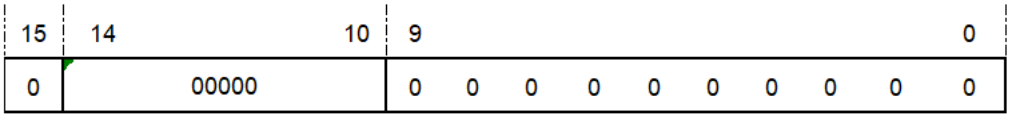


图 13.33: BKPT

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：

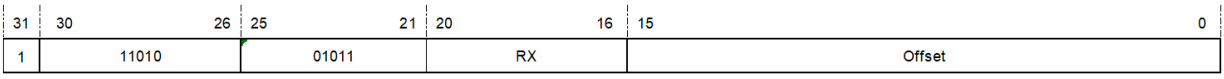


图 13.34: BLSZ

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：

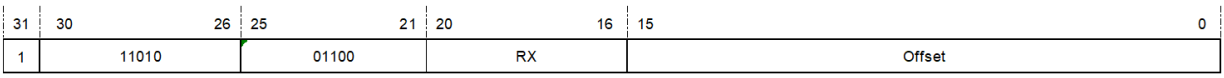


图 13.35: BLZ

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：

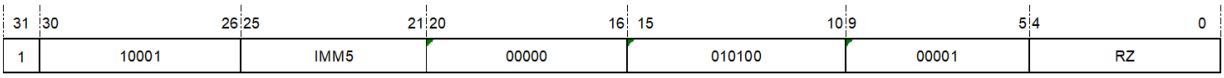


图 13.36: BMASKI

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：

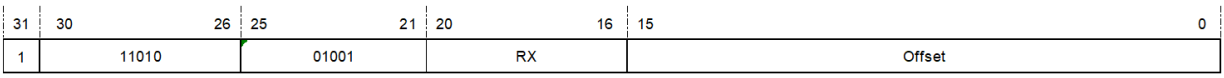


图 13.37: BNEZ

13.25 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：

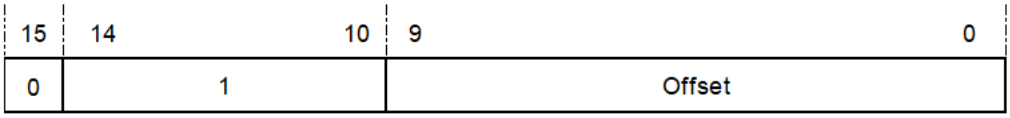


图 13.38: BR-1

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

32 位指令格式：

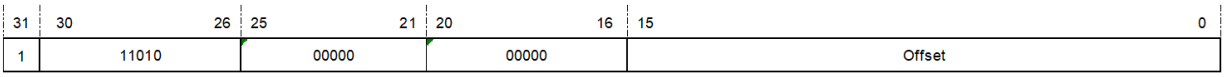


图 13.39: BR-2

13.26 BREV——位倒序指令

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

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

指令格式：

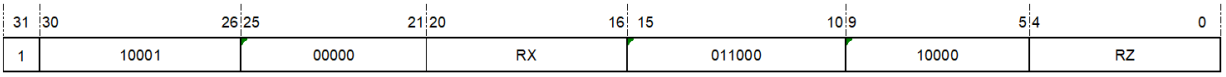


图 13.40: BREV

13.27 BSETI——立即数位置位指令

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

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

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

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

16 位指令格式：

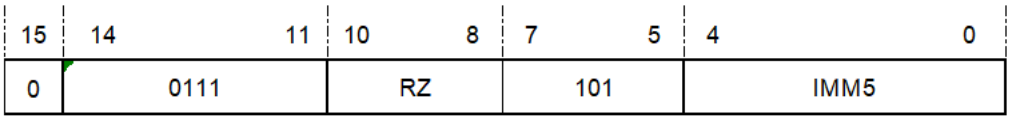


图 13.41: BSETI-1

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

32 位指令格式：

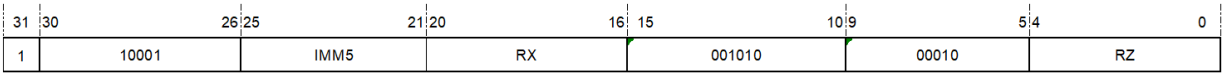


图 13.42: BSETI-2

13.28 BSR——跳转到子程序指令

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

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

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

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

指令格式:

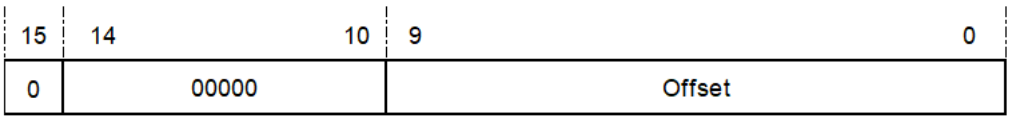


图 13.43: BSR-1

Offset 域——指定跳转的相对偏移量。

注意：跳转的相对偏移量（Offset）不能为 0x0。

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

指令格式:

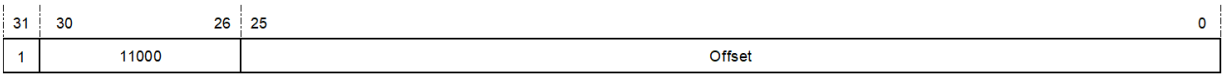


图 13.44: BSR-2

13.29 BT——C 为 1 分支指令

统一化指令	
语法	bt label

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

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

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

16 位指令	

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

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

16 位指令格式：

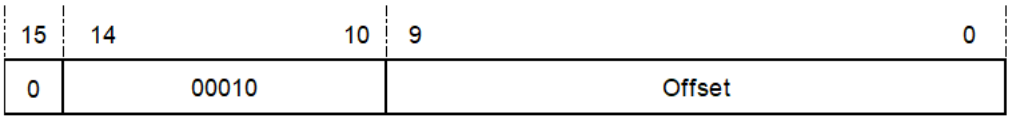


图 13.45: BT-1

32 位指令	

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

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

32 位指令格式：

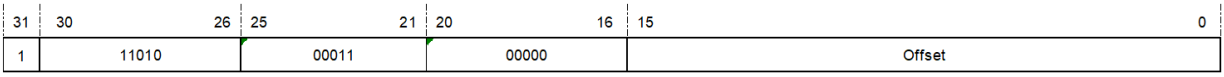


图 13.46: BT-2

13.30 BTSTI——立即数位测试指令

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

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

32 位指令格式：

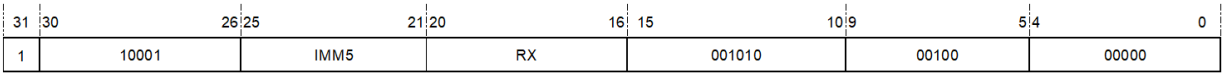


图 13.47: BTSTI

13.31 CLRF——C 为 0 清零指令

统一化指令	
语法	clrf rz

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

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

32 位指令	

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

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

32 位指令格式：

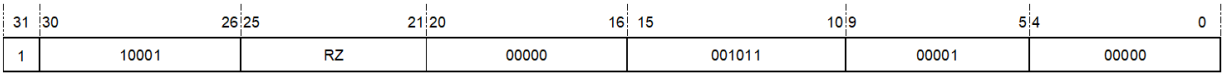


图 13.48: CLRF

13.32 CLRT——C 为 1 清零指令

统一化指令	
语法	clrt rz

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

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

32 位指令	

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

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

32 位指令格式：

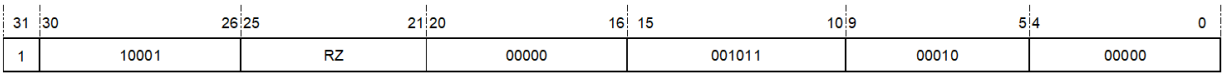


图 13.49: CLRT

13.33 CMPHS——无符号大于等于比较指令

统一化指令	
语法	cmphs rx, ry

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

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

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

16 位指令	

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

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

16 位指令格式：

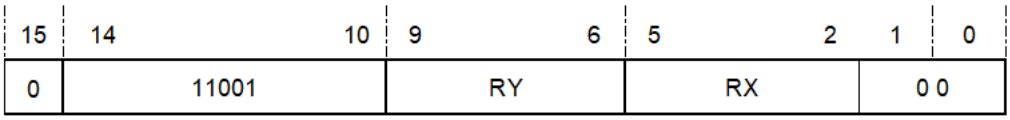


图 13.50: CMPHS-1

32 位指令	

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

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

32 位指令格式：

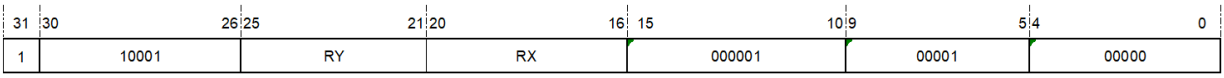


图 13.51: CMPHS-2

13.34 CMPHSI——立即数无符号大于等于比较指令

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

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

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

16 位指令	

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

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

16 位指令格式：

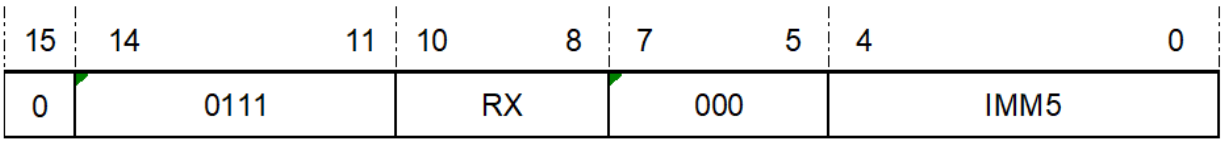


图 13.52: CMPHSI-1

IMM5 域：

指定不带偏置立即数的值。

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

00000：

与 1 比较

00001：

与 2 比较

……

11111：

与 32 比较

32 位指令	

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

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

32 位指令格式：

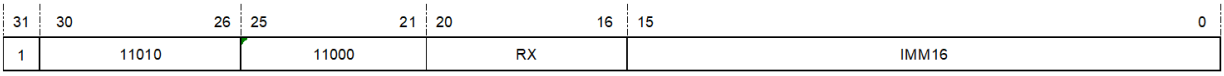


图 13.53: CMPHSI-2

IMM16 域：

指定不带偏置立即数的值。

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

0000000000000000：

与 0x1 比较

0000000000000001：

与 0x2 比较

……

1111111111111111：

与 0x10000 比较

13.35 CMPLT——有符号小于比较指令

统一化指令	
语法	cmplt rx, ry

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

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

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

16 位指令	

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

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

16 位指令格式：

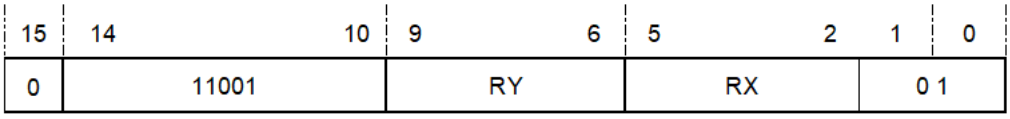


图 13.54: CMPLT-1

32 位指令	

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

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

32 位指令格式：

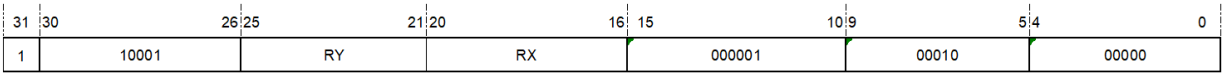


图 13.55: CMPLT-2

13.36 CMPLTI——立即数有符号小于比较指令

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

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

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

16 位指令	

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

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

16 位指令格式：

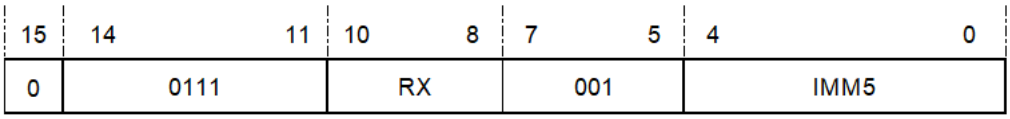


图 13.56: CMPLT-1

IMM5 域：

指定不带偏置立即数的值。

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

00000：

与 1 比较

00001：

与 2 比较

……

11111：

与 32 比较

32 位指令	

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

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

32 位指令格式：

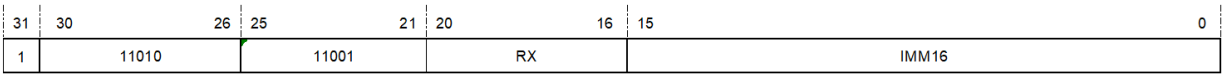


图 13.57: CMPLT-2

IMM16 域：

指定不带偏置立即数的值。

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

0000000000000000：

与 0x1 比较

0000000000000001：

与 0x2 比较

……

1111111111111111：

与 0x10000 比较

13.37 CMPNE——不等比较指令

统一化指令	
语法	cmpne rx, ry

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

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

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

16 位指令	

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

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

16 位指令格式：

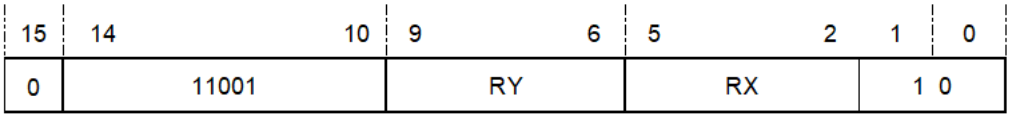


图 13.58: CMPNE-1

32 位指令	

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

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

32 位指令格式：

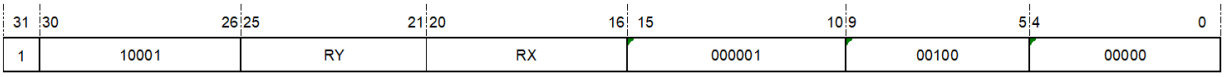


图 13.59: CMPNE-2

13.38 CMPNEI——立即数不等比较指令

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

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

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

16 位指令	

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

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

16 位指令格式：

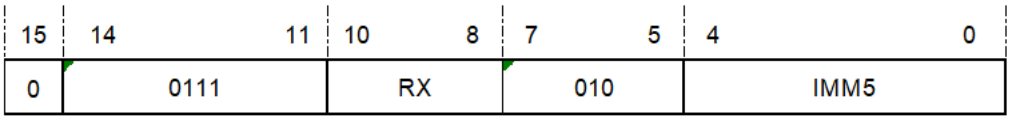


图 13.60: CMPNEI-1

32 位指令	

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

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

32 位指令格式：

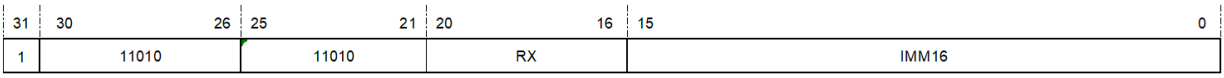


图 13.61: CMPNEI-2

13.39 DECF——C 为 0 立即数减法指令

统一化指令	
语法	decf rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

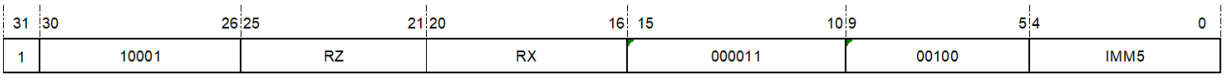


图 13.62: DECF

13.40 DECGT——减法大于零置 C 位指令

统一化指令	
语法	decgt rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

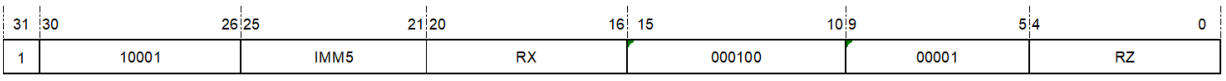


图 13.63: DECGT

13.41 DECLT——减法小于零置 C 位指令

统一化指令	
语法	declt rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

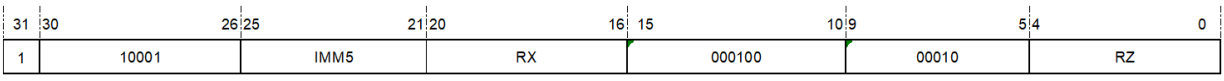


图 13.64: DECLT

13.42 DECNE——减法不等于零置 C 位指令

统一化指令	
语法	decne rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

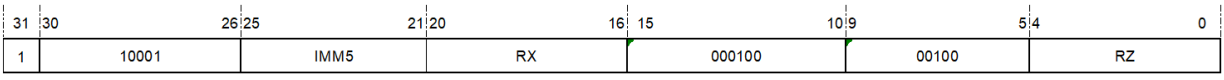


图 13.65: DECNE

13.43 DECT——C 为 1 立即数减法指令

统一化指令	
语法	dect rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

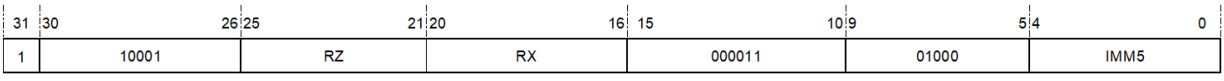


图 13.66: DECT

13.44 DIVS——有符号除法指令

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

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

32 位指令格式：

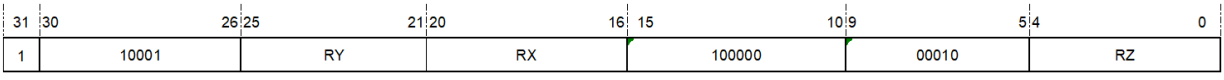


图 13.67: DIVS

13.45 DIVU——无符号除法指令

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

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

32 位指令格式：

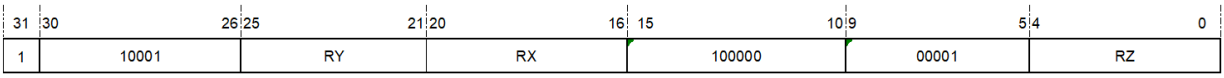


图 13.68: DIVU

13.46 DOZE——进入低功耗睡眠模式指令

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

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

32 位指令格式：

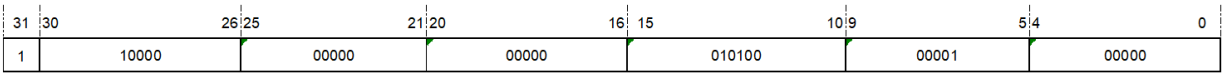


图 13.69: DOZE

13.47 FF0——快速找 0 指令

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

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

指令格式：

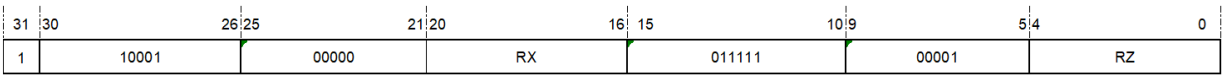


图 13.70: FF0

13.48 FF1——快速找 1 指令

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

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

指令格式：

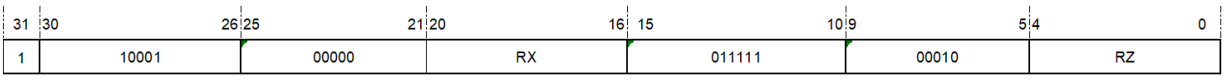


图 13.71: FF1

13.49 GRS——符号产生指令

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

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

指令格式：

图 13.72: GRS

13.50 IDLY——中断识别禁止指令

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

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

32 位指令格式：

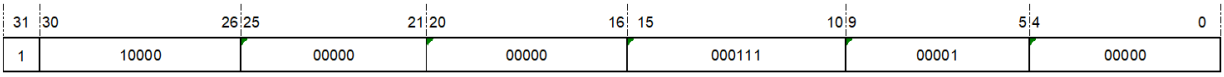


图 13.73: IDLY

13.51 INCF——C 为 0 立即数加法指令

统一化指令	
语法	incf rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

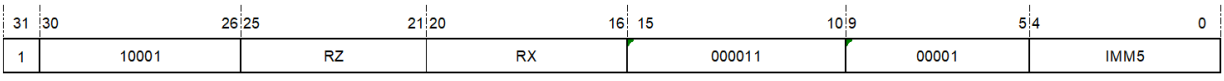


图 13.74: INCF

13.52 INCT——C 为 1 立即数加法指令

统一化指令	
语法	inct rz, rx, imm5

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

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

32 位指令	

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

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

32 位指令格式：

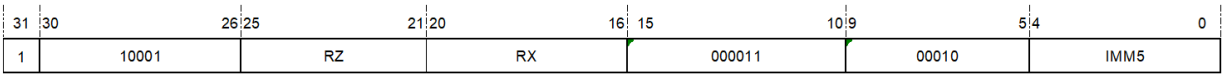


图 13.75: INCT

13.53 INS——位插入指令

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

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

32 位指令格式：

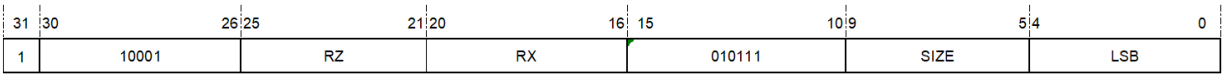


图 13.76: INS

SIZE 域：

指定插入位的宽度。

注意：二进制操作数 SIZE 等于 MSB-LSB。

00000：

1

00001：

2

……

11111：

32

LSB 域：

指定插入结束的位。

00000：

0 位

00001：

1 位

……

11111：

31 位

13.54 IXH——索引半字指令

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

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

32 位指令格式：

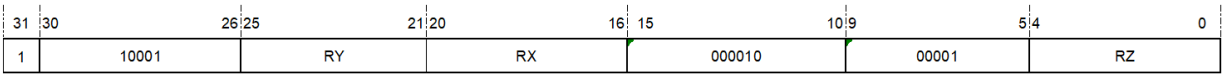


图 13.77: IXH

13.55 IXW——索引字指令

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

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

32 位指令格式：

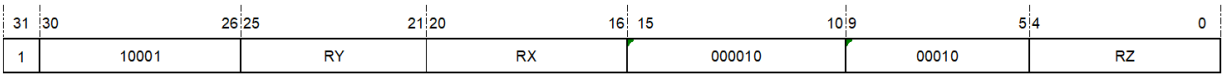


图 13.78: IXW

13.56 IXD——索引双字指令

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

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

32 位指令格式：

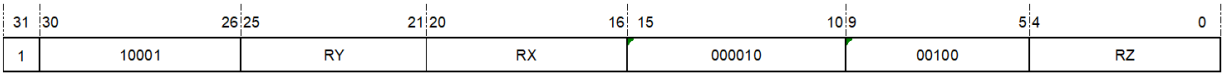


图 13.79: IXD

13.57 JMP——寄存器跳转指令

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

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

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

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

16 位指令格式：

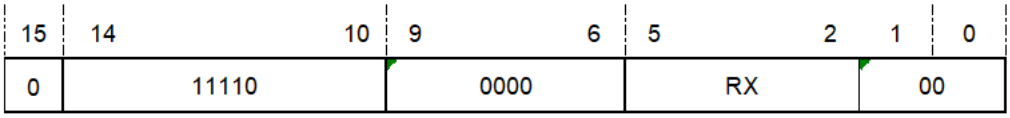


图 13.80: JMP-1

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

32 位指令格式：

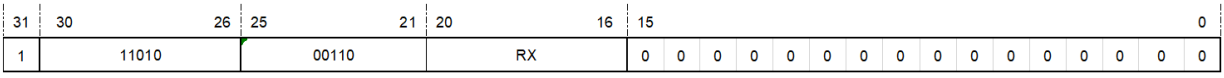


图 13.81: JMP-2

13.58 JMPI——间接跳转指令

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

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

32 位指令格式：

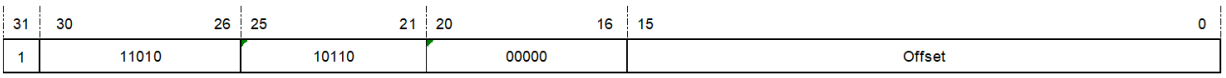


图 13.82: JMPI

13.59 JSR——寄存器跳转到子程序指令

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

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

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

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

16 位指令格式：

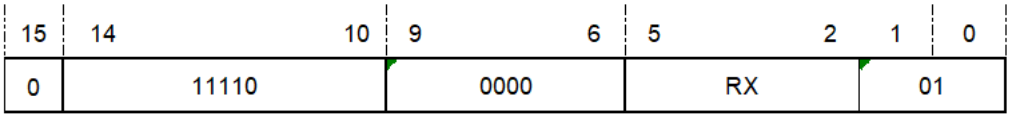


图 13.83: JSR-1

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

32 位指令格式：

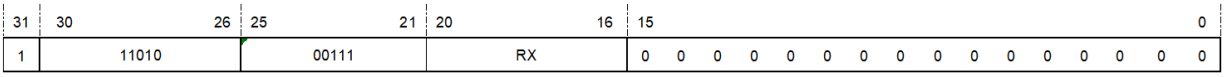


图 13.84: JSR-2

13.60 JSRI——间接跳转到子程序指令

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

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

32 位指令格式：

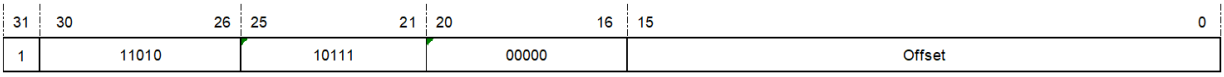


图 13.85: JSRI

13.61 LD.B——无符号扩展字节加载指令

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

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

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

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

16 位指令格式：

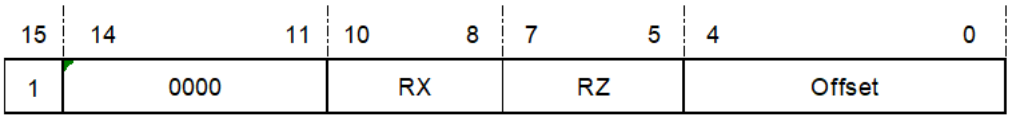


图 13.86: LD.B-1

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

32 位指令格式：

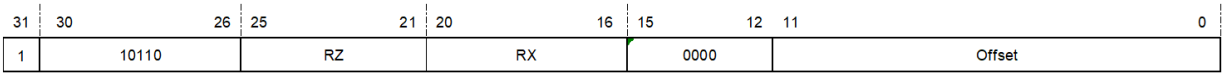


图 13.87: LD.B-2

13.62 LD.BS——有符号扩展字节加载指令

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

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

32 位指令格式：

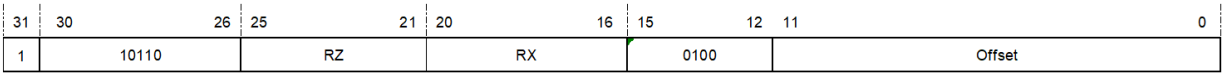


图 13.88: LD.BS

13.63 LD.D——双字加载指令

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

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

32 位指令格式：

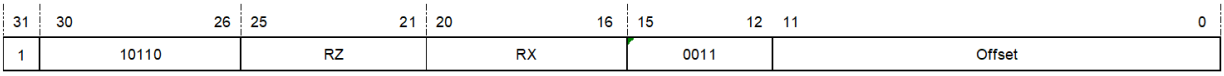


图 13.89: LD.D

13.64 LD.H——无符号扩展半字加载指令

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

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

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

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

16 位指令格式：

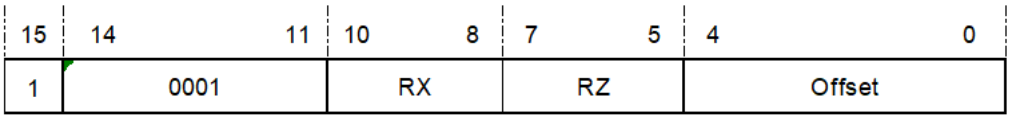


图 13.90: LD.H-1

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

32 位指令格式：

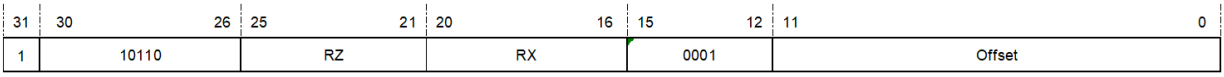


图 13.91: LD.H-2

13.65 LD.HS——有符号扩展半字加载指令

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

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

32 位指令格式：

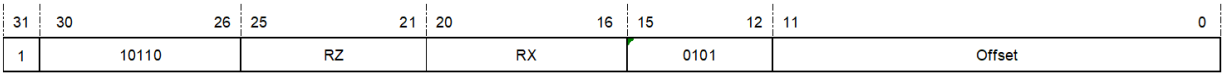


图 13.92: LD.HS

13.66 LD.W——字加载指令

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

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

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

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

16 位指令格式：

ld16.w rz, (rx, disp)

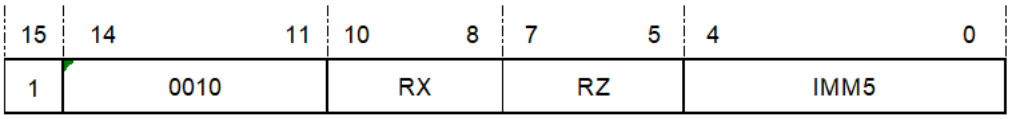


图 13.93: LD.W-1

ld16.w rz, (sp, disp)

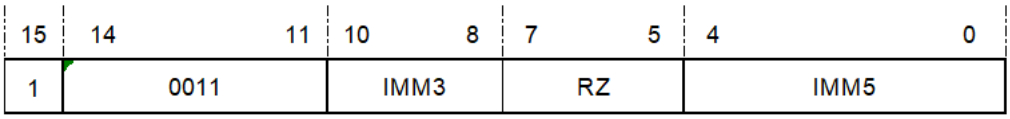


图 13.94: LD.W-2

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

32 位指令格式：

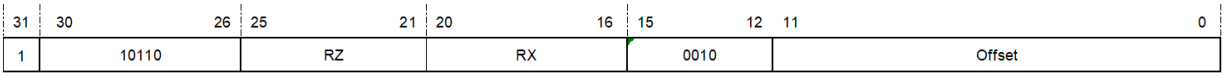


图 13.95: LD.W-3

13.67 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：

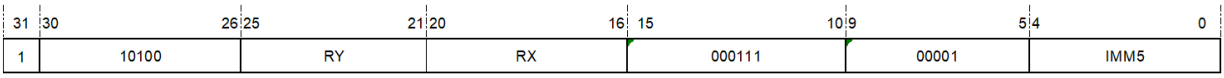


图 13.96: LDM

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

13.68 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：

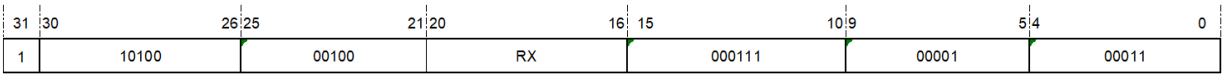


图 13.97: LDQ

13.69 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)

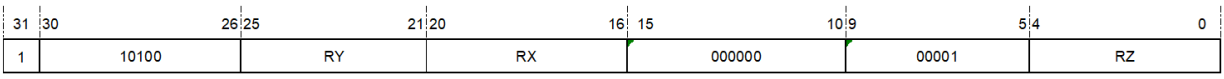


图 13.98: LDR.B-1

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

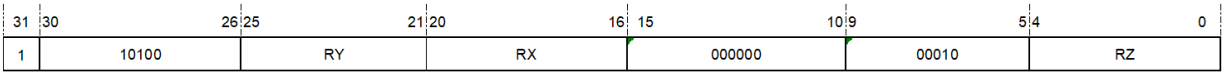


图 13.99: LDR.B-2

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

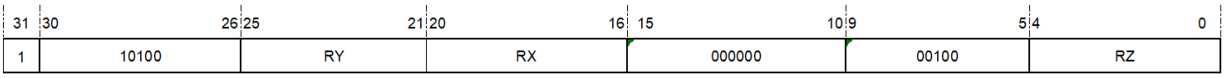


图 13.100: LDR.B-3

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

13.70 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

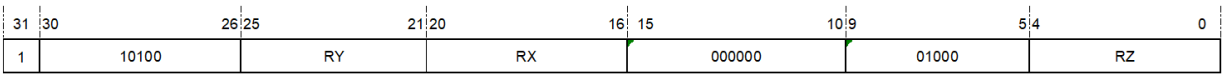


图 13.101: LDR.B-4

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)

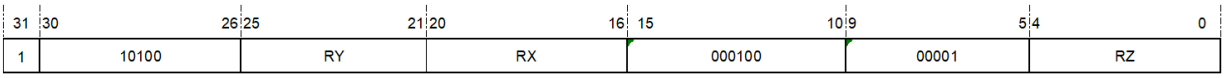


图 13.102: LDR.BS-1

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

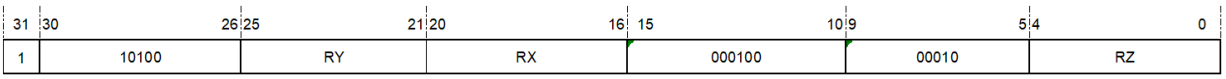


图 13.103: LDR.BS-2

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

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

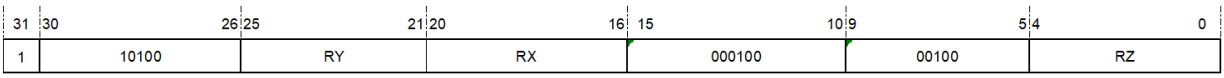


图 13.104: LDR.BS-3

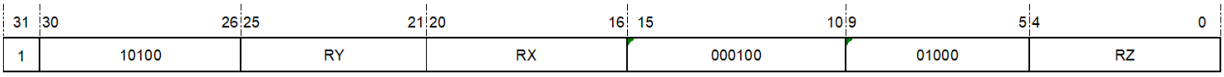


图 13.105: LDR.BS-4

13.71 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)

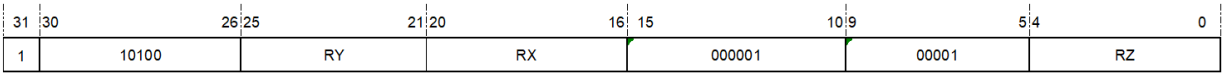


图 13.106: LDR.H-1

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

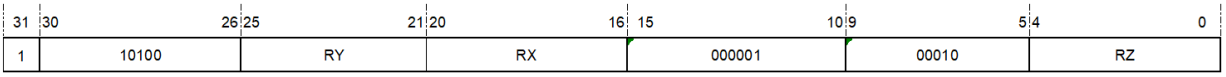


图 13.107: LDR.H-2

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

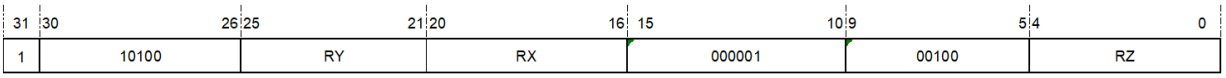


图 13.108: LDR.H-3

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

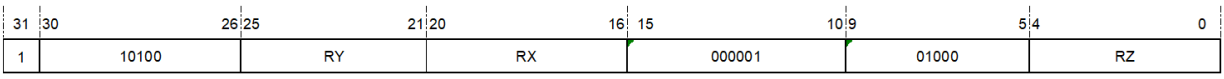


图 13.109: LDR.H-4

13.72 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)

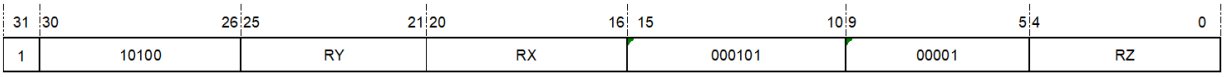


图 13.110: LDR.HS-1

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

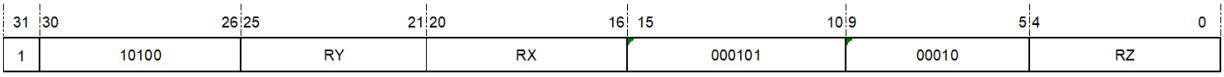


图 13.111: LDR.HS-2

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

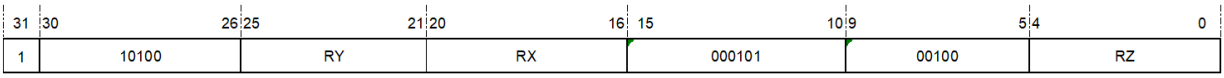


图 13.112: LDR.HS-3

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

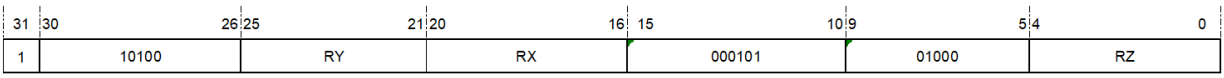


图 13.113: LDR.HS-4

13.73 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)

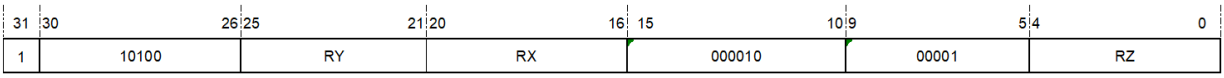


图 13.114: LDR.W-1

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

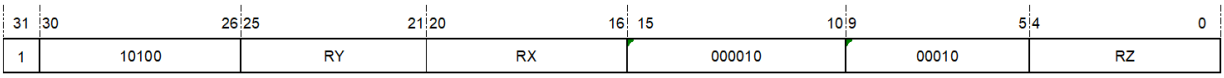


图 13.115: LDR.W-2

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

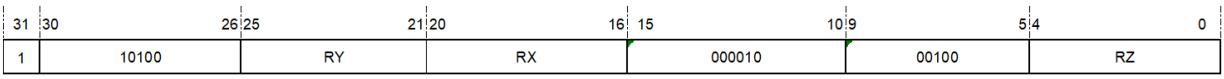


图 13.116: LDR.W-3

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

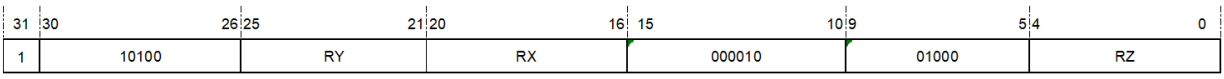


图 13.117: LDR.W-4

13.74 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：

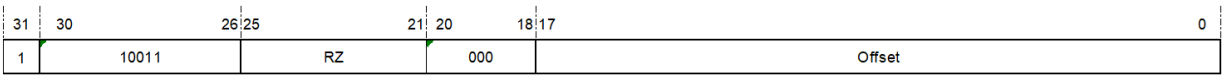


图 13.118: LRS.B

13.75 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：

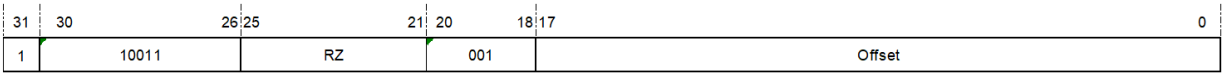


图 13.119: LRS.H

13.76 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：

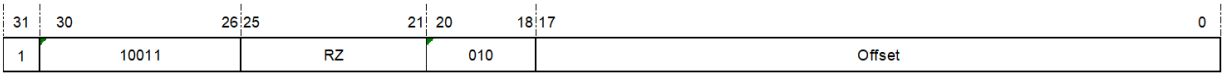


图 13.120: LRS.W

13.77 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：

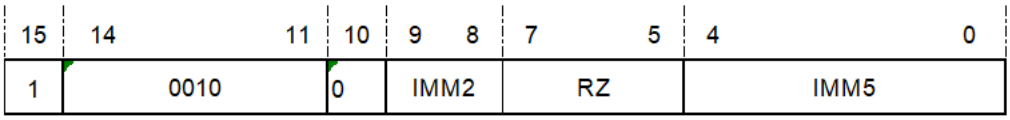


图 13.121: LRW-1

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：

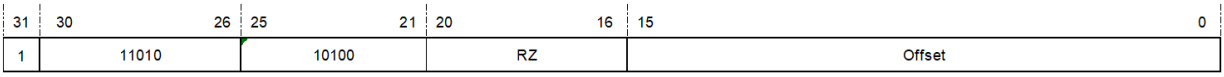


图 13.122: LRW-2

13.78 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：

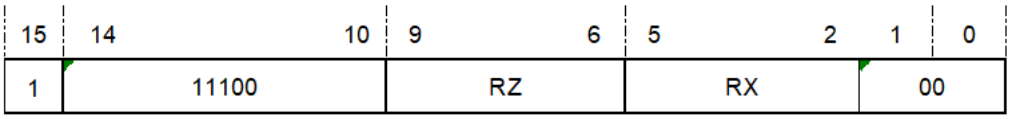


图 13.123: LSL-1

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：

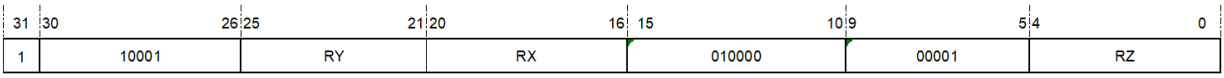


图 13.124: LSL-2

13.79 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：

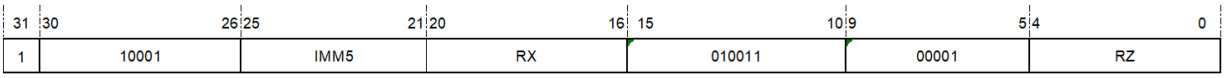


图 13.125: LSLC

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

13.80 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：

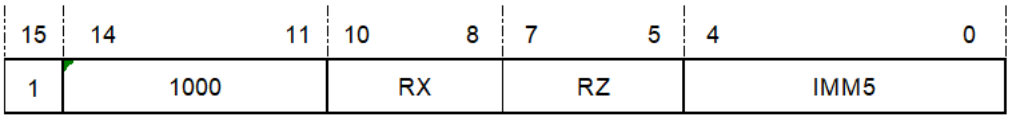


图 13.126: LSLI-1

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：

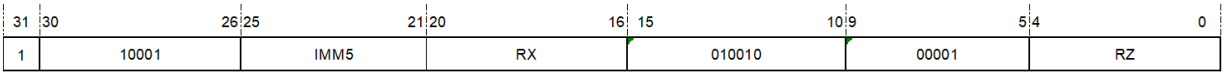


图 13.127: LSLI-2

13.81 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：

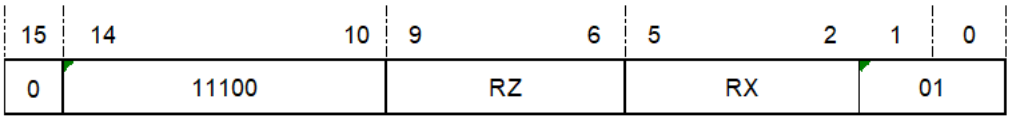


图 13.128: LSR-1

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：

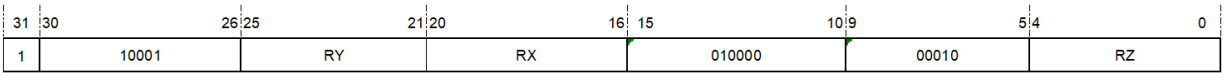


图 13.129: LSR-2

13.82 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：

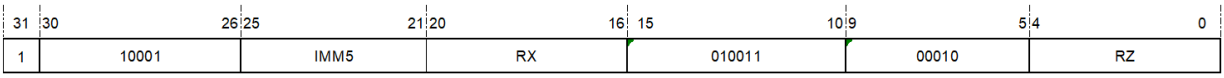


图 13.130: LSRC

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

13.83 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：

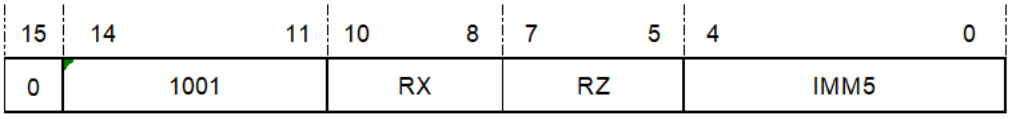


图 13.131: LSRI-1

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：

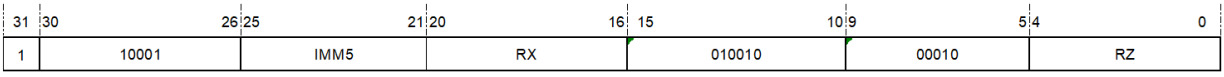


图 13.132: LSRI-2

13.84 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：

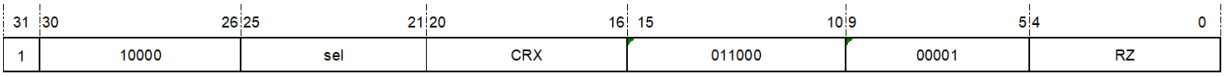


图 13.133: MFCR

13.85 MFHI——累加器高位读传送指令

统一化指令	
语法	mfhi rz
操作	将累加器高位寄存器的内容传送到通用寄存器中 RZ ← HI
编译结果	仅存在 32 位指令。 mfhi32 rz

说明:	**将 64 位累加器的高 32 位寄存器 HI 的内容传送到通用寄存器 RZ 中。**
影响标志位:	无影响
异常：	无

32 位指令	
操作:	将累加器高位寄存器的内容传送到通用寄存器中 RZ ← HI
语法:	mfhi32 rz
说明:	将 64 位累加器的高 32 位寄存器 HI 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常：	无

指令格式:

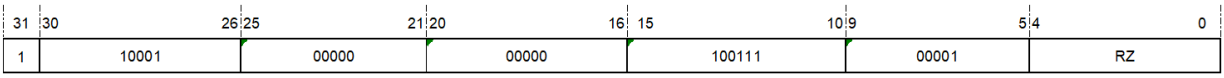


图 13.134: MFHI

13.86 MFLO——累加器低位读传送指令

统一化指令	
语法	mflo rz
操作	将累加器低位寄存器的内容传送到通用寄存器中。 RZ ← LO
编译结果	仅存在 32 位指令。 mflo32 rz

说明:	将 64 位累加器的低 32 位寄存器 LO 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将累加器低位寄存器的内容传送到通用寄存器中 RZ ← LO
语法:	mflo32 rz
说明:	将 64 位累加器的低 32 位寄存器 LO 的内容传送到通用寄存器 RZ 中。
影响标志位:	无影响
异常:	无

指令格式:

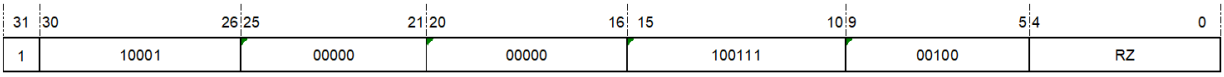


图 13.135: MFLO

13.87 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：

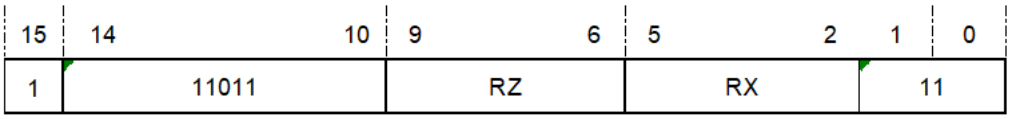


图 13.136: MOV-1

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：

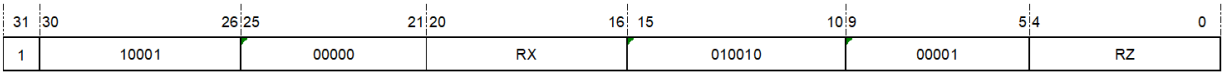


图 13.137: MOV-2

13.88 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：

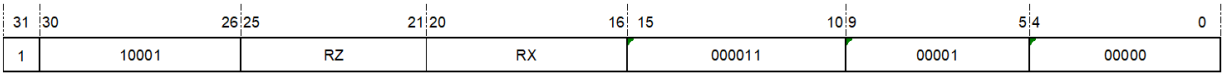


图 13.138: MOVF

13.89 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：

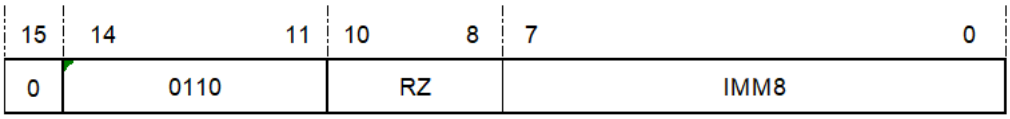


图 13.139: MOVI-1

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：

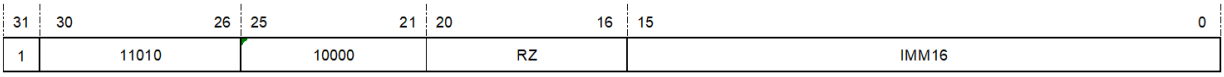


图 13.140: MOVI-2

13.90 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：

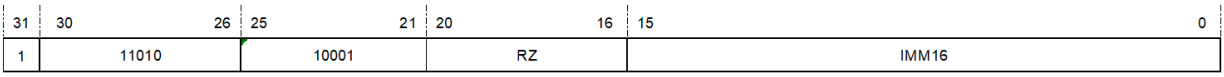


图 13.141: MOVIH

13.91 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：

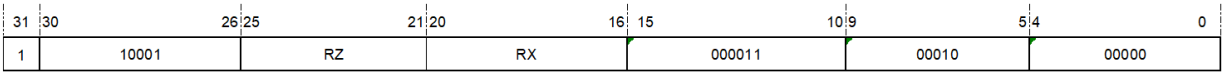


图 13.142: MOVT

13.92 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：

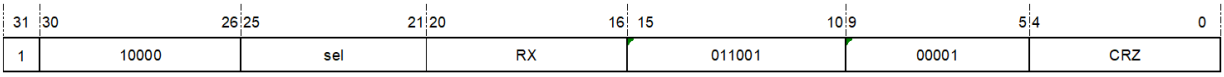


图 13.143: MTCR

13.93 MTHI——累加器高位写传送指令

统一化指令	
语法	mthi rx
操作	将通用寄存器的内容传送到累加器高位寄存器中. HI ← RX
编译结果	仅存在 32 位指令。 mthi32 rx

说明:	将通用寄存器 RX 的内容传送到 64 位累加器的高 32 位寄存器 HI 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将通用寄存器的内容传送到累加器高位寄存器中 HI ← RX
语法:	mthi32 rx
说明:	将通用寄存器 RX 的内容传送到 64 位累加器的高 32 位寄存器 HI 中。
影响标志位:	无影响
异常:	无

指令格式:

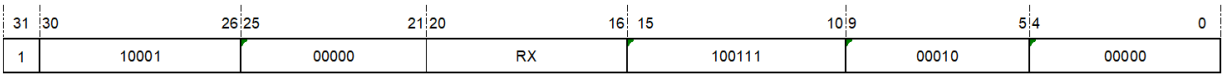


图 13.144: MTHI

13.94 MTLO——累加器低位写传送指令

统一化指令	
语法	mtlo rx
操作	将通用寄存器的内容传送到累加器低位寄存器中 LO ← RX
编译结果	仅存在 32 位指令。 mtlo32 rx

说明:	将通用寄存器 RX 的内容传送到 64 位累加器的低 32 位寄存器 LO 中。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	将通用寄存器的内容传送到累加器低位寄存器中 LO ← RX
语法:	mtlo32 rx
说明:	将通用寄存器 RX 的内容传送到 64 位累加器的低 32 位寄存器 LO 中。
影响标志位:	无影响
异常:	无

指令格式:

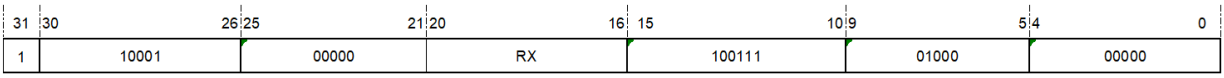


图 13.145: MTLO

13.95 MULS——有符号数乘法指令

统一化指令	
语法	muls rx, ry
操作	两个有符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
编译结果	仅存在 32 位指令。 muls32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个有符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
语法:	muls32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

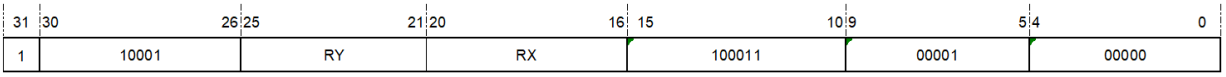


图 13.146: MULS

13.96 MULSA——有符号数乘累加指令

统一化指令	
语法	mulsa rx, ry
操作	两个有符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
编译结果	仅存在 32 位指令。 mulsa32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个有符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
语法:	mulsa32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

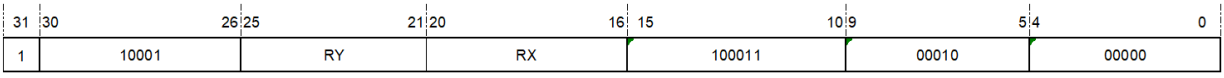


图 13.147: MULSA

13.97 MULSS——有符号数乘累减指令

统一化指令	
语法	mulss rx, ry
操作	累加器中的值减去两个有符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
编译结果	仅存在 32 位指令。 mulss32 rx, ry

说明:	**将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。**
影响标志位:	无影响
异常:	无

32 位指令	
操作:	累加器中的值减去两个有符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
语法:	mulss32 rx, ry
说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是有符号数。
影响标志位:	无影响
异常:	无

指令格式:

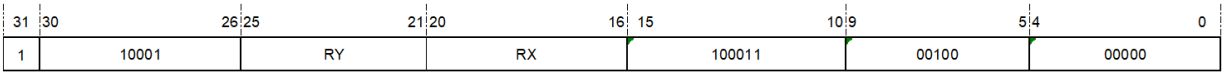


图 13.148: MULSS

13.98 MULU——无符号数乘法指令

统一化指令	
语法	mulu rx, ry
操作	两个无符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
编译结果	仅存在 32 位指令。 mulu32 rx, ry

说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个无符号数相乘，结果放入累加器中 {HI, LO} ← RX × RY
语法:	mulu32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后存放到 64 位累加器中，其中高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

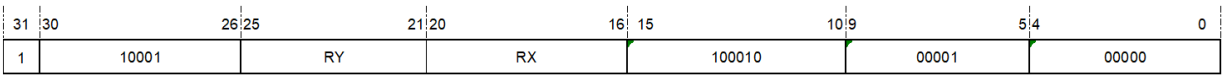


图 13.149: MULU

13.99 MULUA——无符号数乘累加指令

统一化指令	
语法	mulua rx, ry
操作	两个无符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
编译结果	仅存在 32 位指令。 mulua32 rx, ry

说明:	**将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。**
影响标志位:	无影响
异常:	无

32 位指令	
操作:	两个无符号数相乘再加上累加器中的值，结果放回累加器中 {HI, LO} ← {HI, LO} + RX × RY
语法:	mulua32 rx, ry
说明:	将通用寄存器 RX 和 RY 的内容相乘后再加上 64 位累加器中的值，结果放回到累加器中。乘累加的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

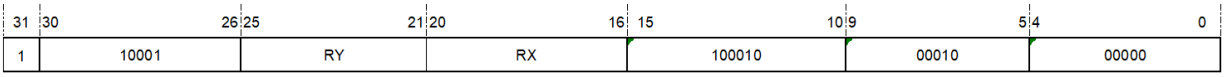


图 13.150: MULUA

13.100 MULUS——无符号数乘累减指令

统一化指令	
语法	mulus rx, ry
操作	累加器中的值减去两个无符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
编译结果	仅存在 32 位指令。 mulus32 rx, ry

说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

32 位指令	
操作:	累加器中的值减去两个无符号数相乘后的值，结果放回累加器中 {HI, LO} ← {HI, LO} - RX × RY
语法:	mulus32 rx, ry
说明:	将 64 位累加器中的值减去通用寄存器 RX 和 RY 相乘后的值，结果放回到累加器中。乘累减的结果的高 32 位存放于 HI 中，低 32 位存放于 LO 中。通用寄存器 RX 和 RY 以及 64 位累加器中的值均被认为是无符号数。
影响标志位:	无影响
异常:	无

指令格式:

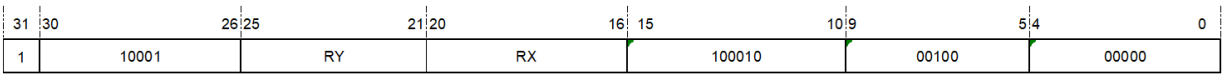


图 13.151: MULUS

13.101 MULSH——16 位有符号乘法指令

统一化指令	
语法	mulsh rz, rx
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RZ[15..0]

编译结果	根据寄存器的范围编译为对应的 16 位或 32 位指令。
	if (x<16) and (y<16), then
	mulsh16 rz, rx;
	else
	mulsh32 rz, rz, rx;

说明	将通用寄存器 RX 的低 16 位和 RZ/ RY 的低 16 位相乘后的结果存放到通用寄存器 RZ 中。寄存器的内容均被认为是有符号数，其中源寄存器的符号位是第 15 位，目标寄存器的符号位是第 31 位。
影响标志位	无影响
异常	无

16 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RZ[15..0]
语法	mulsh16 rz, rx
说明	将通用寄存器 RX 的低 16 位和 RZ 的低 16 位相乘后得到的 32 位结果存放到通用寄存器 RZ 中。寄存器 RX 和 RZ 的内容均被认为是有符号数，其中源操作数的符号位是寄存器的第 15 位，结果的符号位是寄存器的第 31 位。
影响标志位	无影响
限制	寄存器的范围为 r0-r15。
异常	无

16 位指令格式：

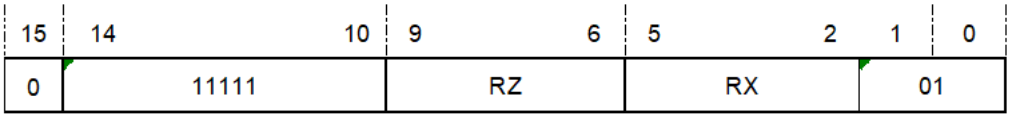


图 13.152: MULSH-1

32 位指令	
操作	两个 16 位有符号数相乘，结果放入通用寄存器中 RZ ← RX[15..0] × RY[15..0]
语法	mulsh32 rz, rx, ry
说明	将通用寄存器 RX 的低 16 位和 RY 的低 16 位相乘后的结果存放到通用寄存器 RZ 中。寄存器 RX、RY 和 RZ 的内容均被认为是有符号数，其中源寄存器 RX、RY 的符号位是第 15 位，目标寄存器 RZ 的符号位是第 31 位。
影响标志位	无影响
异常	无

32 位指令格式：

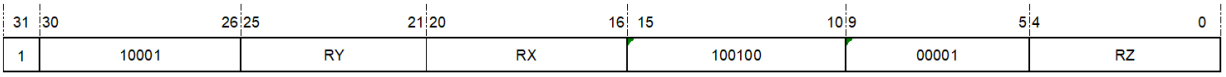


图 13.153: MULSH-2

13.102 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：

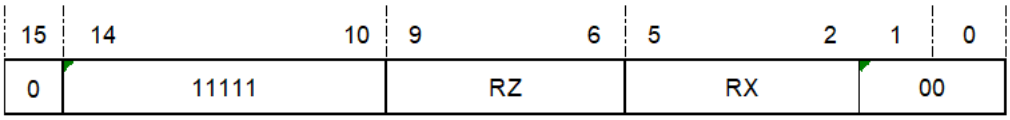


图 13.154: MULT-1

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：

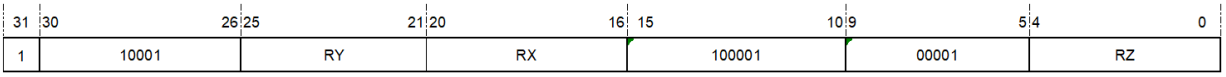


图 13.155: MULT-2

13.103 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：

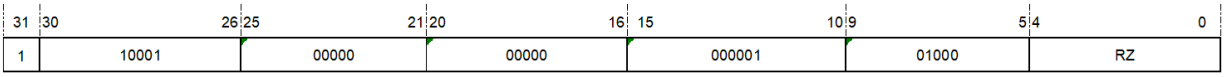


图 13.156: MVC

13.104 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：

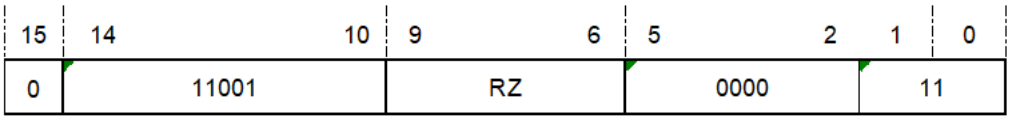


图 13.157: MVCV-1

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：

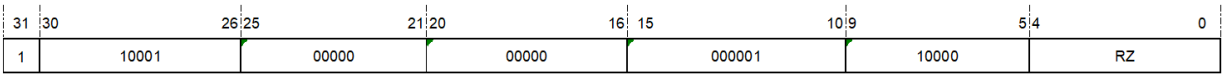


图 13.158: MVCV-2

13.105 MVTC——溢出位复制到 C 位指令

统一化指令	
语法	mvtc
操作	C ← V
编译结果	仅存在 32 位指令。 mvtc32;

说明:	将 DCSR(CR <14,0>) 的溢出位复制到 C 位。
影响标志位:	根据溢出位设置 C 位
异常:	无

32 位指令	
操作:	C ← V
语法:	mvtc32
说明:	将 DCSR(CR <14,0>) 的溢出位复制到 C 位。
影响标志位:	根据溢出位设置 C 位
异常:	无

指令格式：

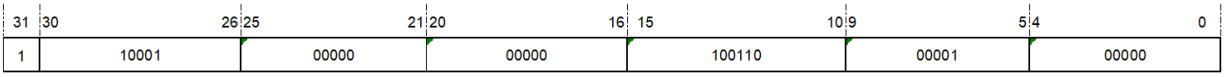


图 13.159: MVTC

13.106 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：

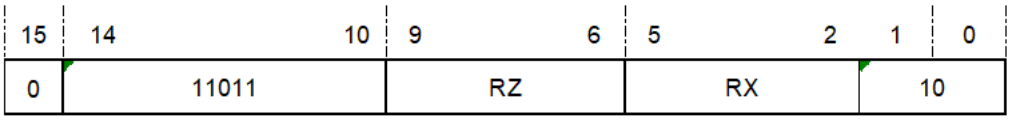


图 13.160: NOR-1

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：

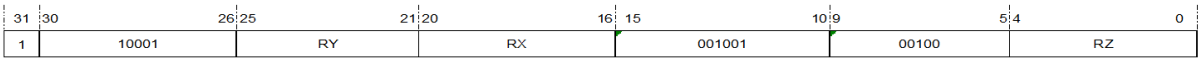


图 13.161: NOR-2

13.107 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：

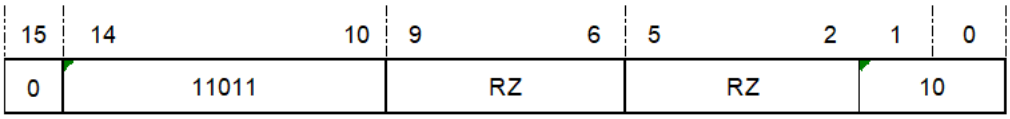


图 13.162: NOT-1

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：

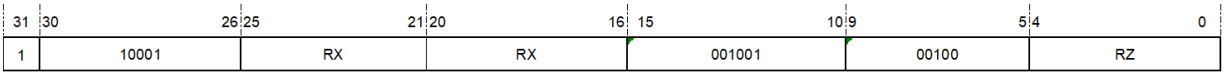


图 13.163: NOT-2

13.108 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：

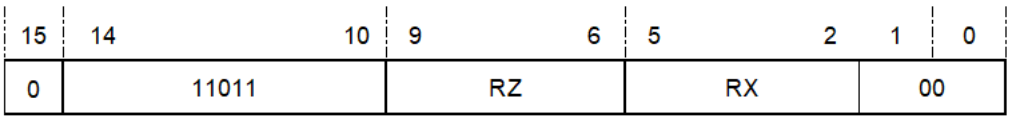


图 13.164: OR-1

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：

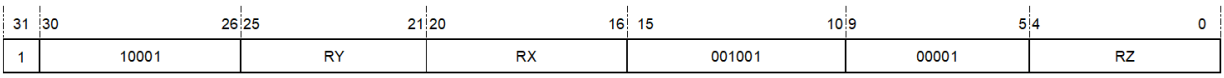


图 13.165: OR-2

13.109 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：

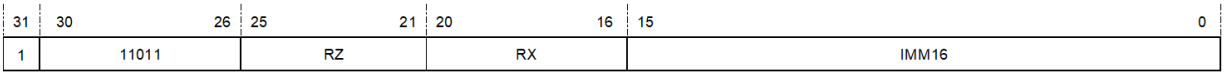


图 13.166: ORI

13.110 PLDR——读数据预取指令

统一化指令	
语法	pldr (rx, disp)
操作	读数据预取 MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 位指令。 pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令	
操作	读数据预取
语法	pldr32 (rx, disp)
说明	读数据预取指令的目的是加速数据的加载行为。在数据加载前，使用 PLDR 指令将该数据行读入到 Cache 中；在加载数据的 Load 指令执行时，命中在 D-Cache 中，从而加快数据加载的效率。读数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

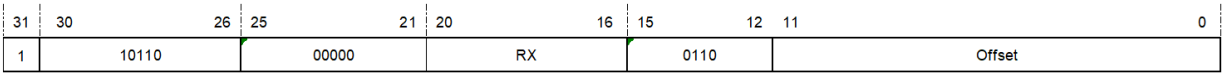


图 13.167: PLDR

13.111 PLDW——写数据预取指令

统一化指令	
语法	pldw (rx, disp)
操作	写数据预取 MEM[RX + zero_extend(offset << 2)]
编译结果	仅存在 32 指令。 pldw32 (r 位 x, disp)
说明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令	
操作	写数据预取
语法	pldw32 (rx, disp)
说明	写数据预取指令的目的是加速写分配 Cache 策略下数据的存储行为。在写分配 Cache 的策略下，存储指令 Cache 未命中会导致一次 Cache 回填操作。PLDW 指令在存储指令之前将即将存储的数据所在的行读入到 Cache；在存储数据的 Store 指令执行时，命中在 D-Cache 中，从而加快数据存储的效率。写数据预取的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	无

32 位指令格式：

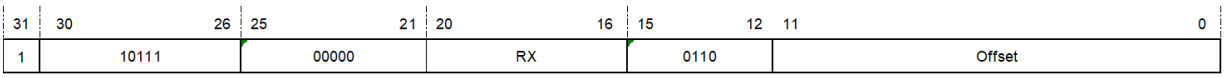


图 13.168: PLDW

13.112 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：

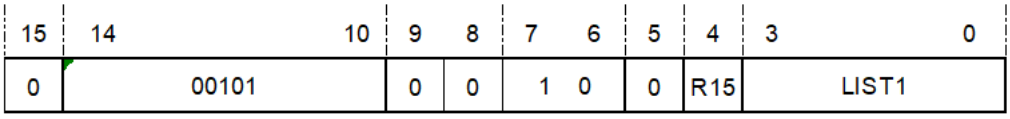


图 13.169: POP-1

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：

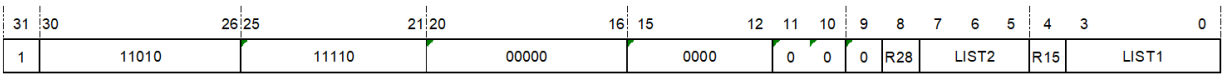


图 13.170: POP-2

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

13.113 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：

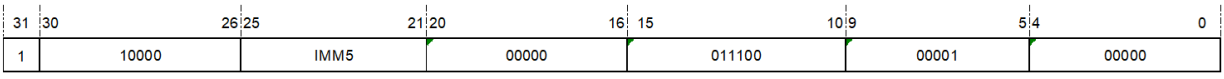


图 13.171: PSRCLR

13.114 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：

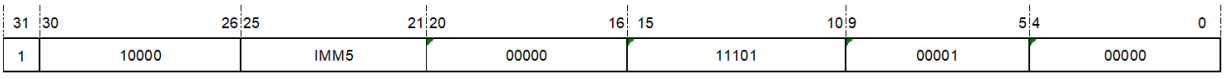


图 13.172: PSRSET

13.115 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：

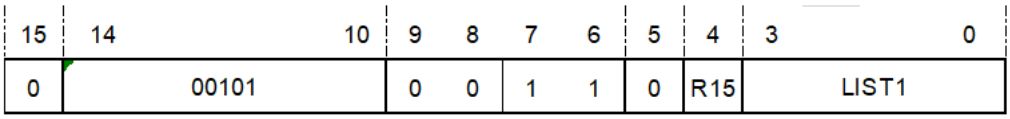


图 13.173: PUSH-1

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：

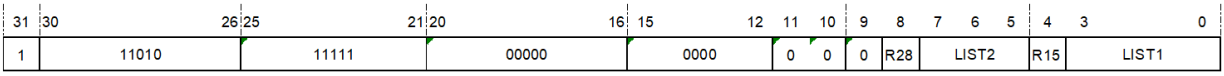


图 13.174: PUSH-2

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

13.116 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：

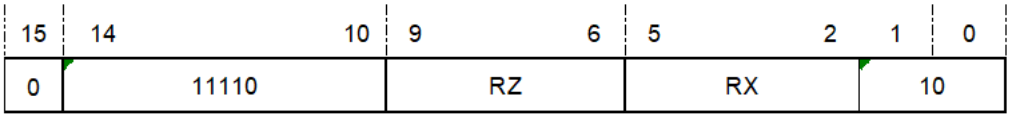


图 13.175: REVB-1

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：

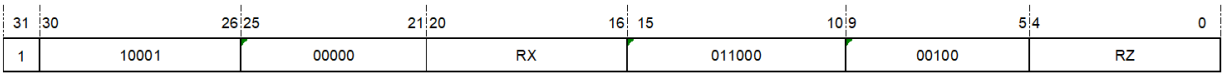


图 13.176: REVB-2

13.117 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：

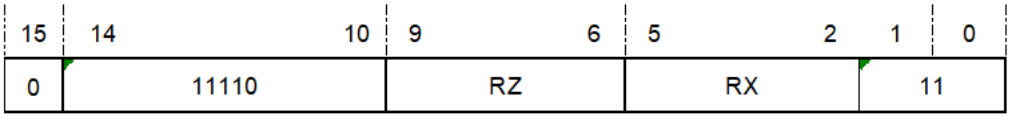


图 13.177: REVH-1

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：

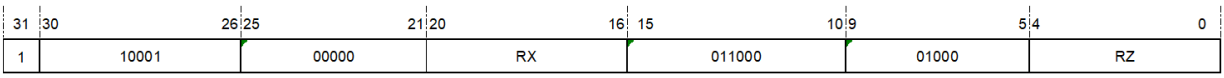


图 13.178: REVH-2

13.118 RFI——快速中断返回指令

统一化指令	
语法	rfi
操作	快速中断返回 PC ← FPC，PSR ← FPSR
编译结果	仅存在 32 位指令。 rfi 32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令	
操作	快速中断返回 PC ← FPC，PSR ← FPSR
语法	rfi32
属性：	特权指令
说明	PC 值恢复为保存在控制寄存器 FPC 中的值，PSR 值恢复为保存在 FPSR 的值，指令执行从新的 PC 地址处开始。
影响标志位	无影响
异常	特权违反异常

32 位指令格式：

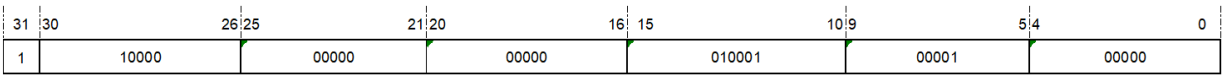


图 13.179: RFI

13.119 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：

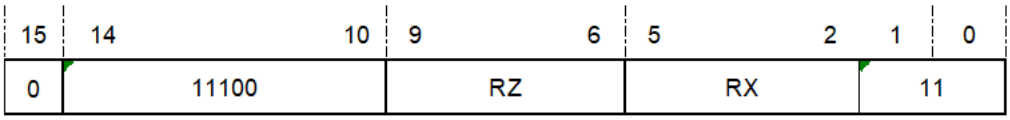


图 13.180: ROTL-1

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：

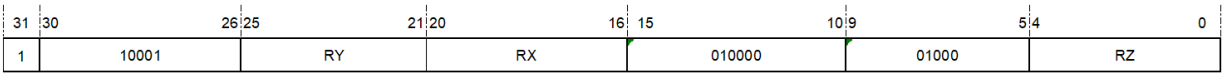


图 13.181: ROTL-2

13.120 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：

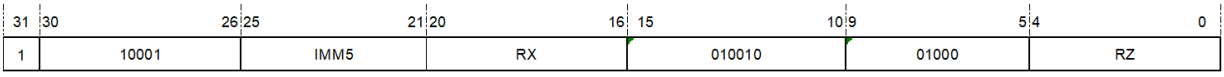


图 13.182: ROTLI

13.121 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：

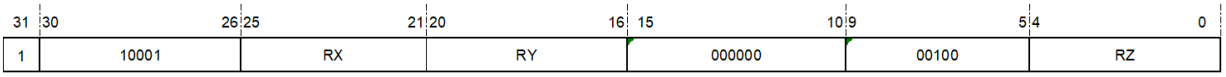


图 13.183: RSUB

13.122 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：

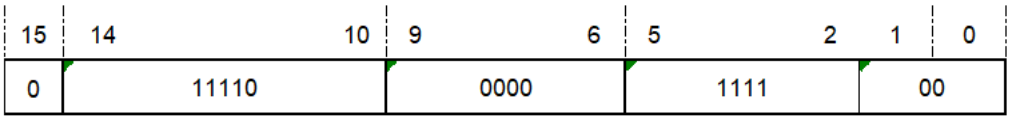


图 13.184: RTS-1

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：

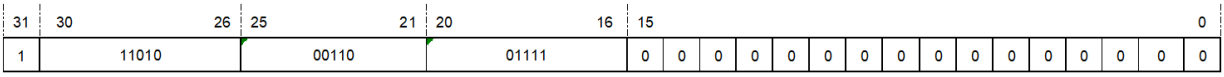


图 13.185: RTS-2

13.123 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：

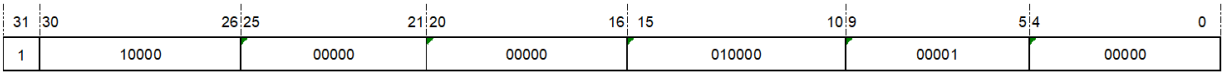


图 13.186: RTE

13.124 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:

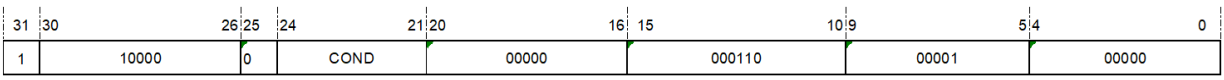


图 13.187: SCE

13.125 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：

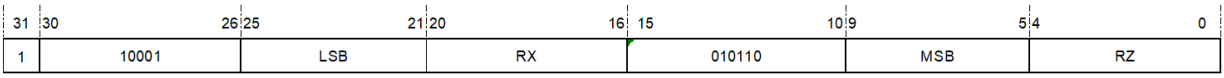


图 13.188: SEXT

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

13.126 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：

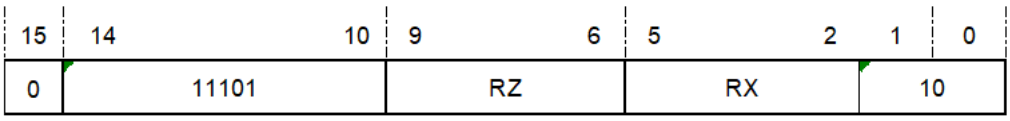


图 13.189: SEXTB-1

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：

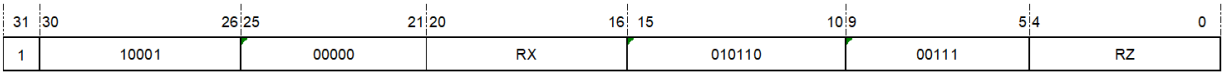


图 13.190: SEXTB-2

13.127 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：

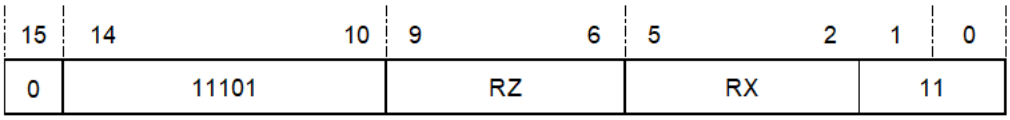


图 13.191: SEXTH-1

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：

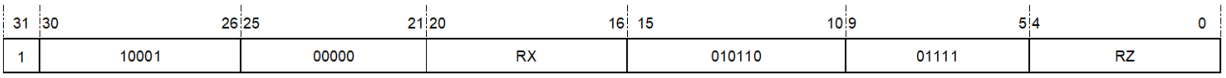


图 13.192: SEXTH-2

13.128 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：

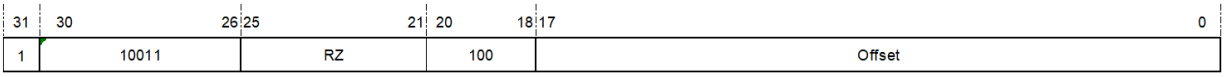


图 13.193: SRS.B

13.129 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：

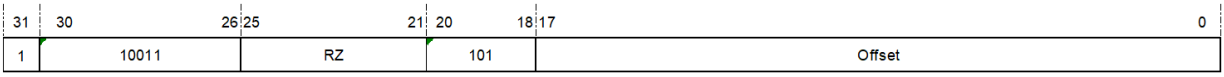


图 13.194: SRS.H

13.130 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：

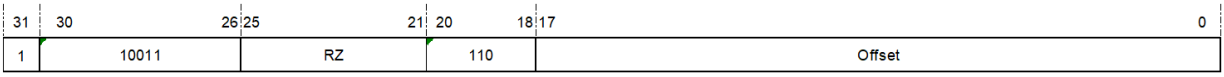


图 13.195: SRS.W

13.131 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：

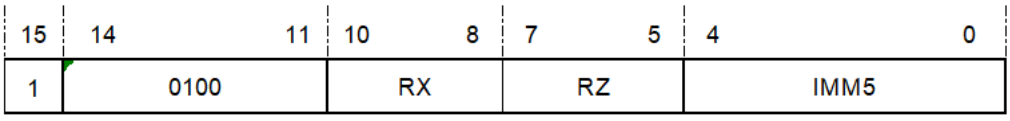


图 13.196: ST.B-1

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：

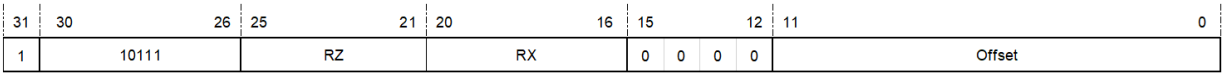


图 13.197: ST.B-2

13.132 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：

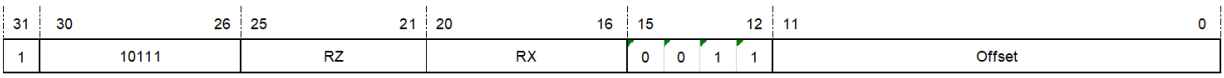


图 13.198: ST.D

13.133 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：

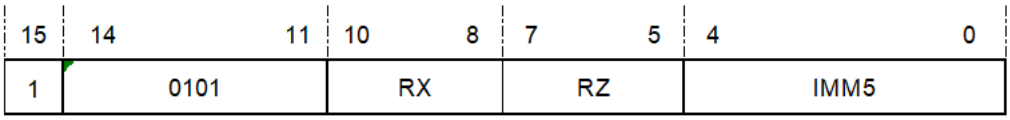


图 13.199: ST.H-1

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：

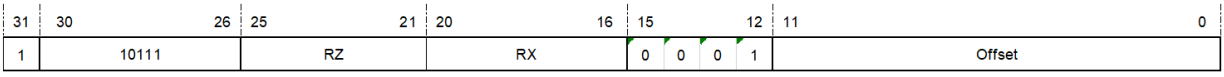


图 13.200: ST.H-2

13.134 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)

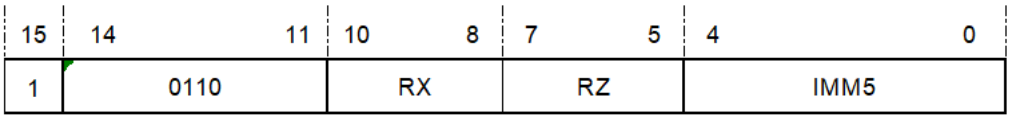


图 13.201: ST.W-1

st16.w rz, (sp, disp)

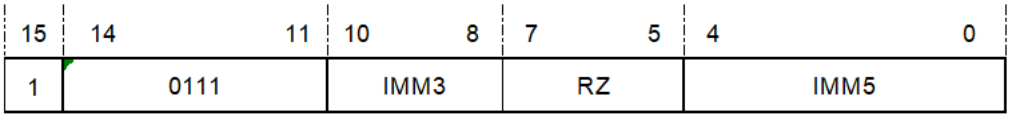


图 13.202: ST.W-2

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：

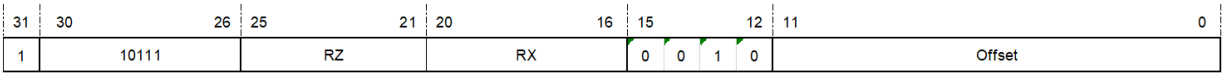


图 13.203: ST.W-3

13.135 STCPR——协处理器字存储指令

统一化指令	
语法	stcpr <cpid, cprz>, (rx, offset)
操作	将协处理器通用寄存器中的字存储到存储器中 MEM[RX + sign_extend(offset << 2)] ← CPRZ
编译结果	仅存在 32 位指令。 stcpr32 <cpid, cprz>, (rx, offset)
说明	将协处理器通用寄存器 CPRZ 中的字存储到存储器中。采用寄存器加立即数偏移量的寻址方式。指令低 12 位编码空间中 8~11 位约定为协处理器号，用于指定预操作的协处理器，其余 8 位为相对偏移量。存储器的有效地址由主流水线基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。STCPR 指令可以寻址 +1KB 地址空间。
影响标志位	无影响
异常	未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

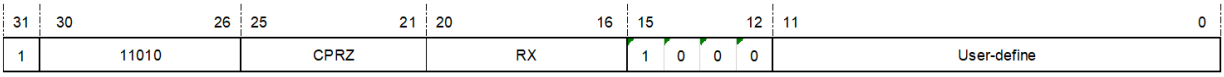


图 13.204: STCPR

13.136 STEX.W——独占式字存储指令

统一化指令	
语法	stex.w rz, (rx, disp)

操作	if 独占式字存储成功, then
	MEM[RX +
	sign_extend(offset << 2)] ← RZ;
	RZ ← 1;
	else
	RZ ← 0;

编译结果	仅存在 32 位指令。 stex32.w rz, (rx, disp)
说明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX.W 指令可以寻址 +16KB 空间。 该指令与 LDEX.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令	

操作	if 独占式字存储成功, then
	MEM[RX + sign_extend(offset << 2)] ← RZ;
	RZ ← 1;
	else
	RZ ← 0;

语法	stex32.w rz, (rx, disp)
说明	将通用寄存器 RZ 中的字存储到存储器中，若独占存储成功，则源寄存器 RZ 返回 1；否则源寄存器返回 0 表示独占存储失败。STEX32.W 采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 相对偏移量无符号扩展到 32 位后的值得到。STEX32.W 指令可以寻址 +16KB 空间。 该指令与 LDEX32.W 配对，用于多核通信时的原子“读存储器—修改—写存储器”操作。 注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。
影响标志位	无影响
异常	未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32 位指令格式：

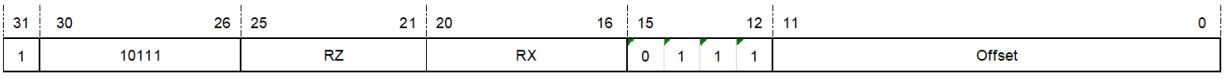


图 13.205: STEX.W

13.137 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：

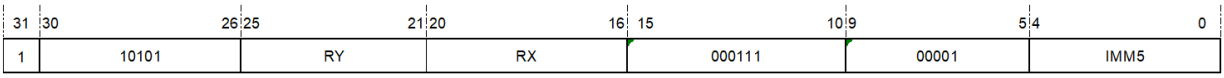


图 13.206: STM

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

13.138 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：

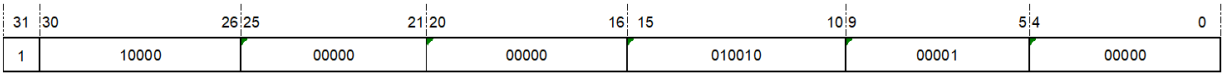


图 13.207: STOP

13.139 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：

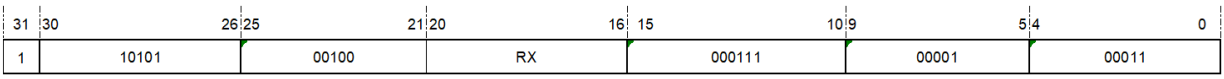


图 13.208: STQ

13.140 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)

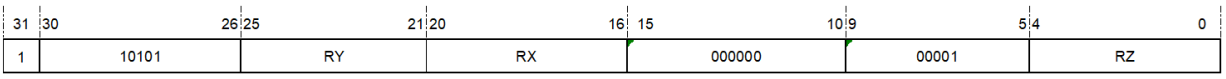


图 13.209: STR.B-1

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

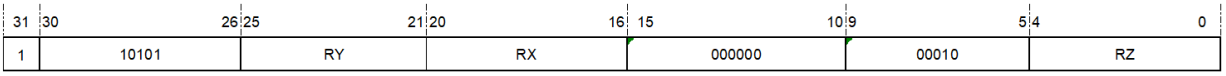


图 13.210: STR.B-2

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

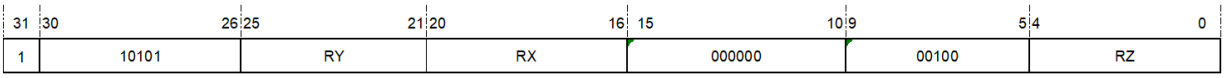


图 13.211: STR.B-3

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

13.141 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

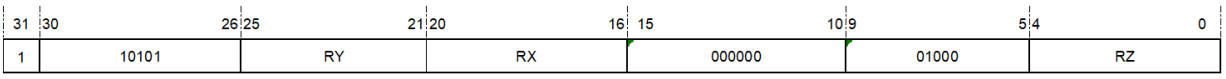


图 13.212: STR.B-4

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)

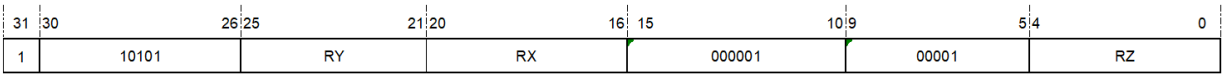


图 13.213: STR.H-1

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

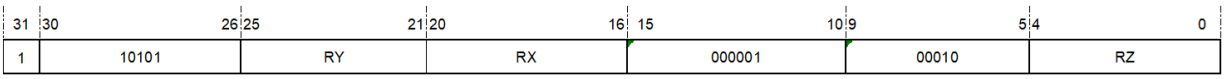


图 13.214: STR.H-2

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

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

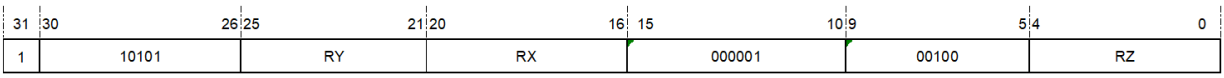


图 13.215: STR.H-3

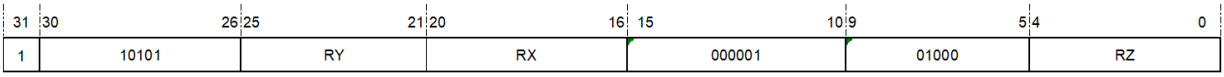


图 13.216: STR.H-4

13.142 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)

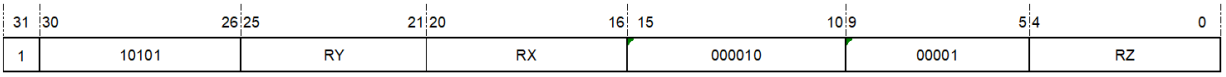


图 13.217: STR.W-1

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

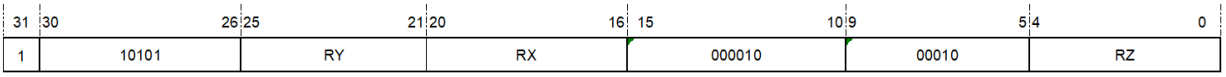


图 13.218: STR.W-2

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

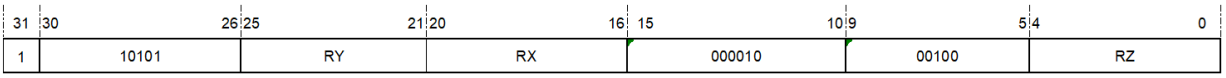


图 13.219: STR.W-3

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

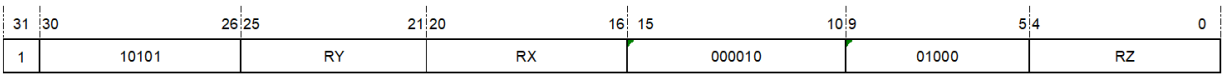


图 13.220: STR.W-4

13.143 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：

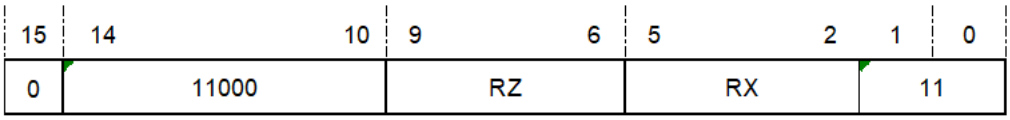


图 13.221: SUBC-1

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：

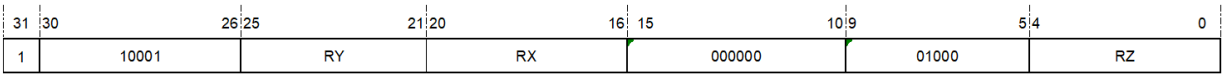


图 13.222: SUBC-2

13.144 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：

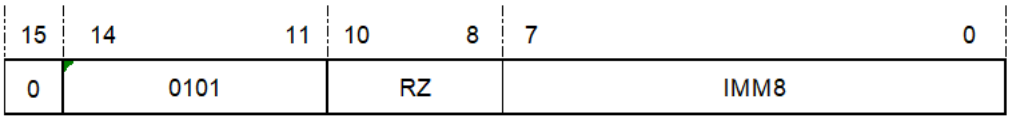


图 13.223: SUBI-1

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：

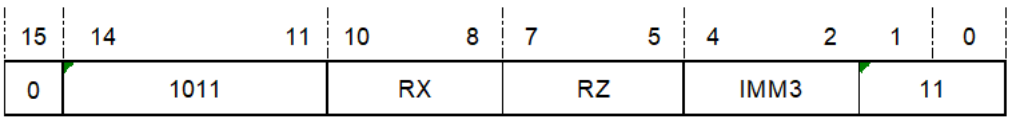


图 13.224: SUBI-2

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

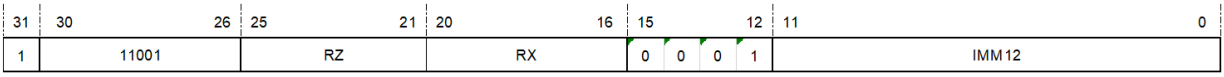
……

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

13.145 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：

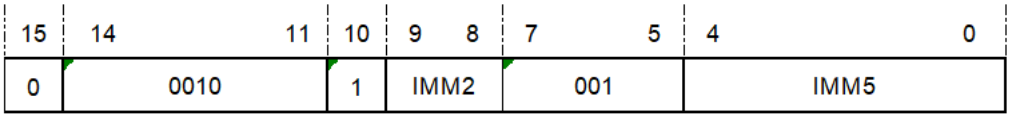


图 13.225: SUBI(SP)

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

13.146 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：

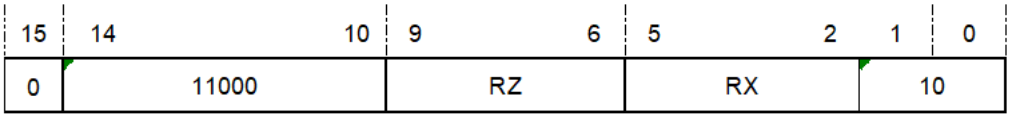


图 13.226: SUBU-1

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：

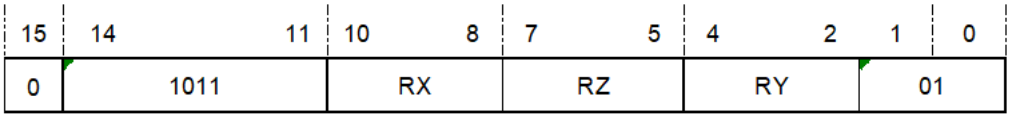


图 13.227: SUBU-2

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：

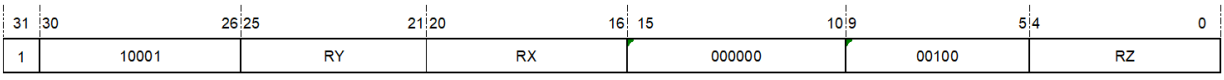


图 13.228: SUBU-3

13.147 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：

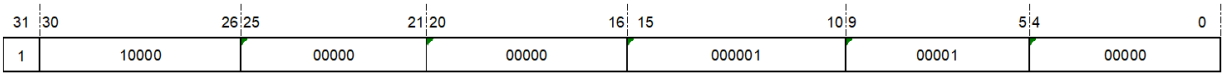


图 13.229: SYNC

13.148 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

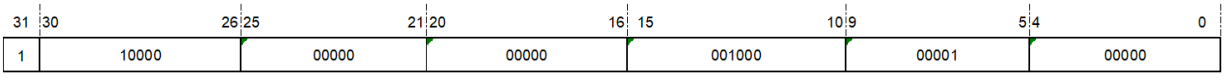


图 13.230: TRAP-1

trap32 1

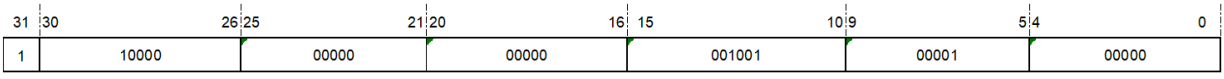


图 13.231: TRAP-2

trap32 2

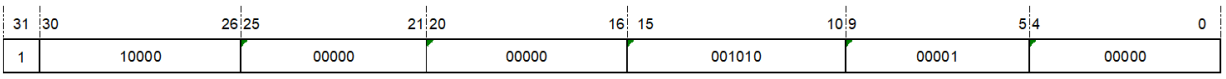


图 13.232: TRAP-3

trap32 3

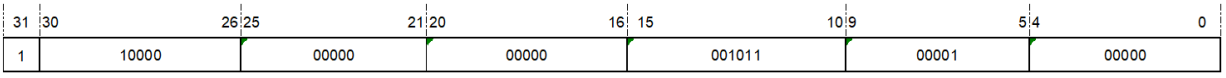


图 13.233: TRAP-4

13.149 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：

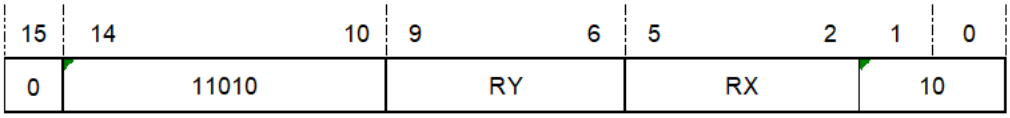


图 13.234: TST-1

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：

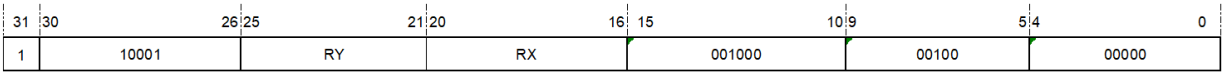


图 13.235: TST-2

13.150 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：

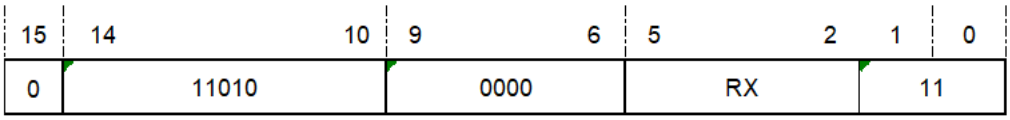


图 13.236: TSTNBZ-1

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：

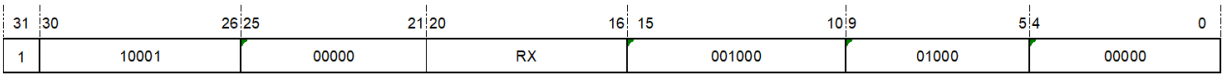


图 13.237: TSTNBZ-2

13.151 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：

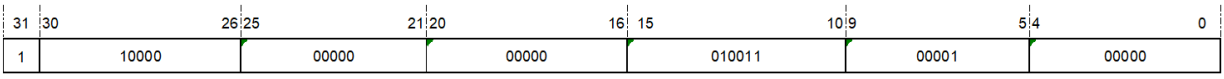


图 13.238: WAIT

13.152 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：

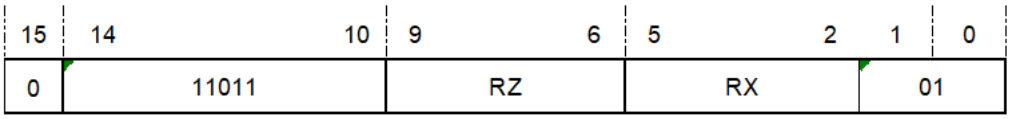


图 13.239: XOR-1

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：

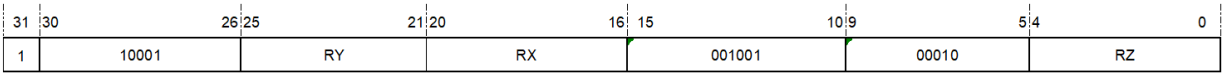


图 13.240: XOR-2

13.153 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：

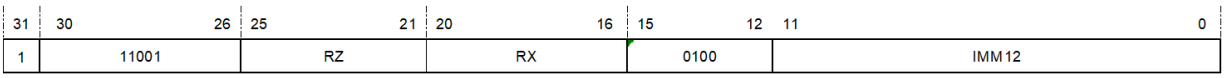


图 13.241: XORI

13.154 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：

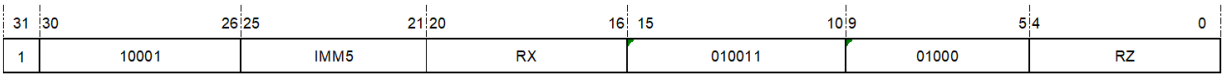


图 13.242: XSR

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

13.155 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：

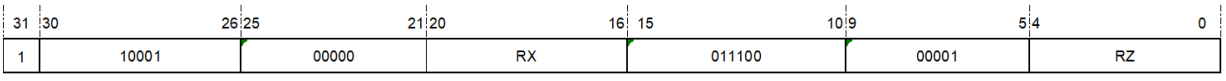


图 13.243: XTRB0

13.156 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：

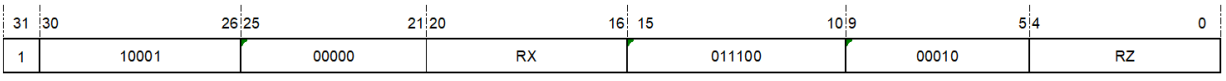


图 13.244: XTRB1

13.157 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：

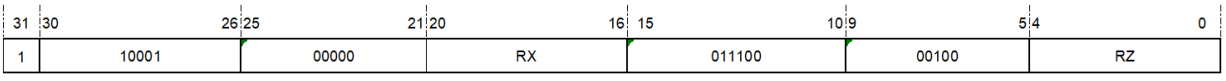


图 13.245: XTRB2

13.158 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：

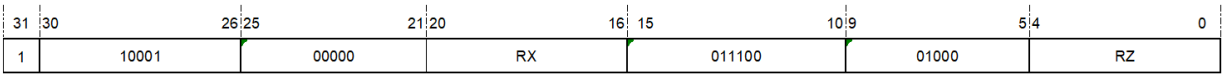


图 13.246: XTRB3

13.159 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：

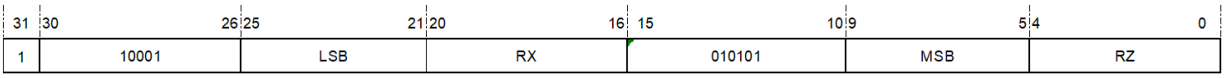


图 13.247: ZEXT

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

13.160 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：

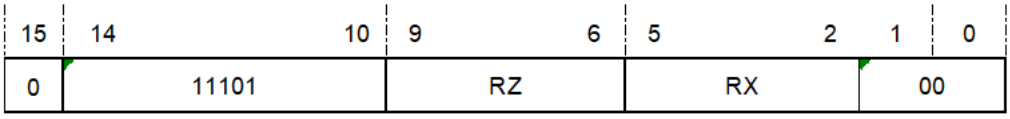


图 13.248: ZEXTB-1

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：

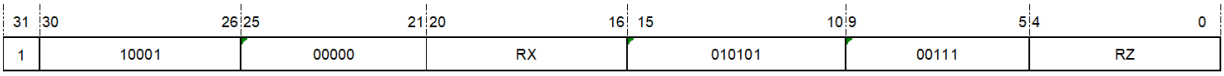


图 13.249: ZEXTB-2

13.161 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：

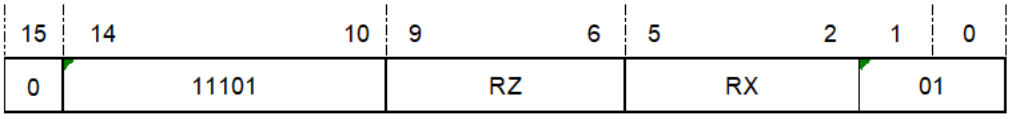


图 13.250: ZEXTH-1

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：

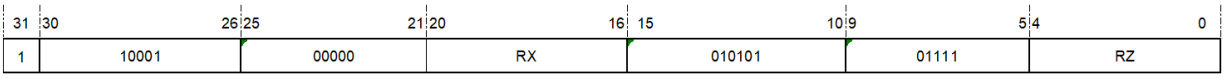


图 13.251: ZEXTH-2

