C-SKY CK802

用户手册

(2014.09 Ver 1.2.8)

杭州中天微系统有限公司

C-SKY Microsystems Co.,Ltd

声明：

杭州中天微系统有限公司(C-SKY Microsystems Co.,Ltd)保留本文档的所有权利。
本文档的内容有可能发生更改、更新、删除、变动，恕不另行通知。

公司地址：浙江省杭州市华星路 99 号东软创业大厦 A403 室
邮政编码：310012
电话：0571-88157060
传真：0571-88157058-8888
主页：www.c-sky.com
E-mail：info@c-sky.com

版本历史:

版本	日期	描述	作者
1.0	12/01/2011	第一版基本功能描述。	杭州中天微系统有限公司
1.1	12/15/2011	1、 删除第七章中对 DIV 指令的描述。 2、 修改 8.2.2 未对齐异常描述，修改为只能发生在数据上。 3、 删除 8.2.7 的跟踪异常，CK802 没有设计跟踪模式，即不会产生异常。 4、 删除第 8 章中除以 0 异常中的内容，CK802 不实现除法指令。	杭州中天微系统有限公司
1.2	2/13/2012	1、 修改对 32/16 的指令描述。 2、 修改对通用寄存器的描述。 3、 增加二进制代码转译功能的描述。	杭州中天微系统有限公司
1.2.1	3/1/2012	1、 修改 CK802 系统结构图。 2、 修改 CK802 信号框图。	杭州中天微系统有限公司
1.2.2	3/8/2012	1、 修改表达错误。	杭州中天微系统有限公司
1.2.3	1/13/2014	1、 增加 BPUSH.H、BPUSH.W、BPOP.H、BPOP.W。 2、 更新 MGU 相关控制位功能。 3、 增加 SEU 单元。 4、 增加部分 32 位指令。	杭州中天微系统有限公司
1.2.4	2/16/2014	1、 修正 CR21 名称表示错误 2、 修正总线结构 prot 信号表征含义 3、 加入 strong order 机制说明	杭州中天微系统有限公司
1.2.5	3/28/2014	1、 加入软件复位控制寄存器以及软件复位机制说明	杭州中天微系统有限公司
1.2.6	5/14/2014	1、 修改 MGU 配置寄存器相关说明	杭州中天微系统有限公司
1.2.7	6/30/2014	1、 修改 lrw16 编码，删除 bsr16，增加 btsti16	杭州中天微系统有限公司
1.2.8	9/3/2014	1、 删除部分 32 位指令 2、 删除 LDR 指令 3、 对异常处理机制进行补充说明 4、 对调试机制进行修改 5、 加入 cache 相关说明 6、 加入 MM 位对非对齐异常的控制说明	杭州中天微系统有限公司

目录：

图表目录：

1. 简介

CK802 是杭州中天微系统有限公司自主研发的极低功耗、极低成本嵌入式 CPU 核，以 8 位 CPU 的成本获得 32 位嵌入式 CPU 的运行效率与性能。CK802 基于 C-SKY V2 自主指令架构，采用 16/32 位混合编码系统，通过精心设计指令系统与流水线硬件结构，具备极低成本、极低功耗和高代码密度等优点。CK802 主要针对智能卡、智能电网、低成本微控制器、无线传感网络等嵌入式应用。

CK802 采用了 16/32 位混合编码的 RISC 指令集，实现了 C-SKY V2 指令架构中 65 条 16 位指令和部分 32 位指令。其中 16 位指令集的优势是低成本、高代码密度，缺点是索引和立即数范围较小；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，C-SKY 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

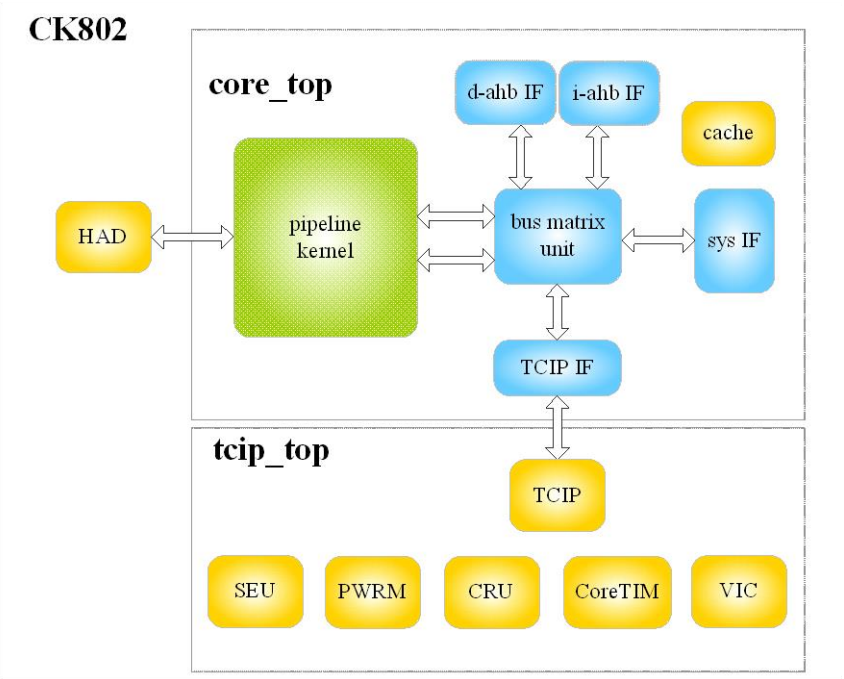
CK802 处理器体系结构的主要特点如下：

- RISC 精简指令结构；
- 高性能的 32 位数据长度，32 位或 16 位可变的指令长度；
- 2 级流水线；
- 可配置的硬件乘法器，支持 1-33 个周期；
- 可配置的快速退休机制；
- 单周期指令和数据存储器访问；
- 无延时的分支跳转；
- 支持硬件可配置内存保护区域（0-8）；
- 支持 AHB-Lite 总线协议，支持可配置的指令总线和数据总线
- 支持总线接口多种 CPU 到总线时钟比率；
- 支持大端模式和小端模式；
- 支持可配置的片上在线硬件调试接口。
- 支持可配置的二进制代码转译机制；
- 支持可配置的信息安全增强技术；
- 支持可配置的高速缓存器，高速缓存容量 2KB、4KB 和 8KB 硬件可配。

CK802 在 TSMC 0.18um 工艺下性能参数如下（频率优化）：

- 工作频率 50MHz（最恶劣情况）；
- CPU 核面积约 12K 等效门；
- 动态功耗小于 80 uW/MHz；
- 性能：0.95DMIPS/MHz。

1.2. 微体系结构



图表 1-1 CK802 结构图

CK802 处理器采用 2 级流水线结构。指令取指阶段主要负责从内存中获取指令，并对 16/32 位变长指令进行译码、复杂指令拆解和调度指令发射到下一级流水线；指令执行阶段主要负责指令的执行和结果的回写。CK802 中内存数据的存取划分为两个步骤，分别为地址的产生和内存的访问，最快支持在一个时钟周期内完成存储器的访问。

可配置的内存管理单元支持超级用户自定义内存空间的访问权限，权限划分为：不可读写/只读/可读写，可执行/不可执行，也可以设置为安全区与非安全区。

总线接口单元兼容 AHB-Lite 协议，设计有寄存器输出（Flop-out）和直接输出（Non-Flop-out）两种硬件配置。在寄存器输出配置下，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；在直接输出配置下，系统时钟与 CPU 时钟只能按照 1：1 工作。

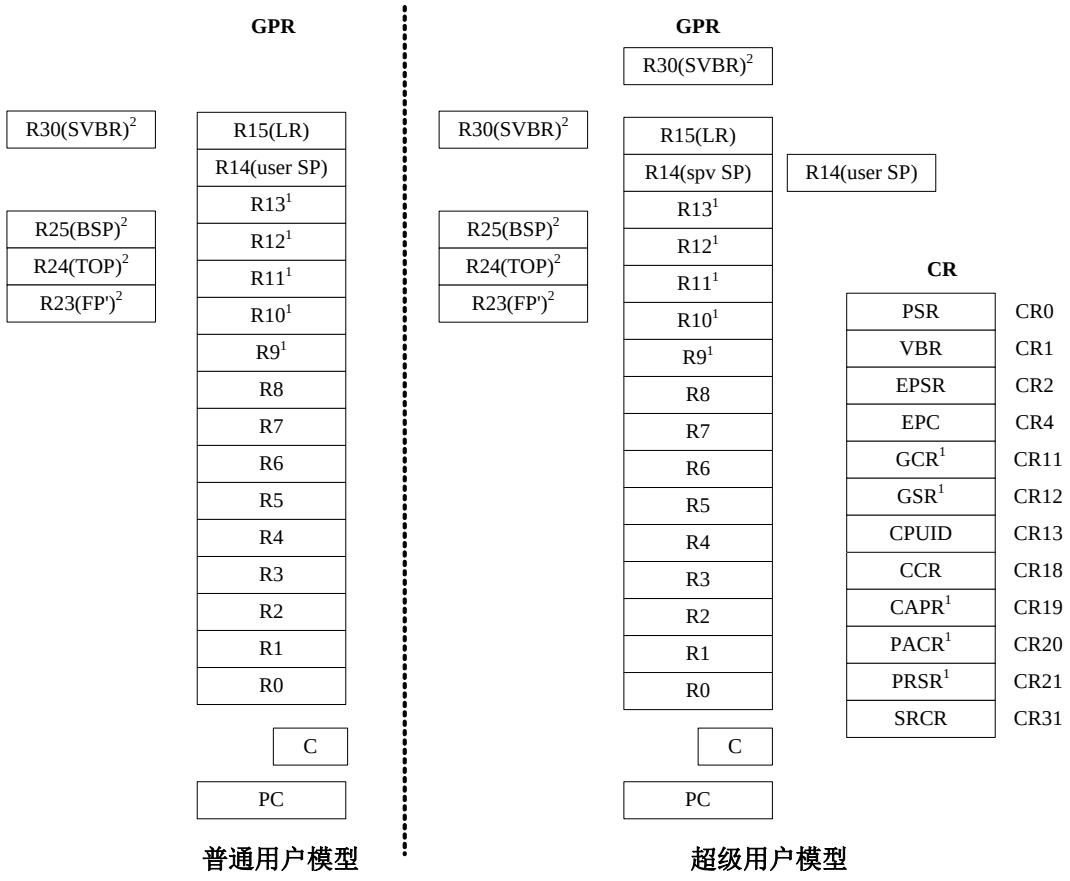
硬件辅助调试单元支持各种调试方式，包括软件设置断点方式、内存断点方式、单步和多步指令跟踪等 7 种方式，可在线调试 CPU、通用寄存器（GPR）、协处理器 0（CP0）和内存。

片内外存储资源包括紧耦合存储器接口、片上紧耦合的 IP 接口和系统总线接口。紧耦合存储器接口用于用户自定义功能扩展，片上紧耦合的 IP 接口下设计有中断控制器（VIC）、系统计时器（CoreTIM）、功耗管理单元（PWRM）、高速缓存控制寄存器单元（CRU）和安全扩展单元（SEU）。矢量中断控制器支持 1/2/4/8/16/24/32 个中断源，支持电平和脉冲两种中断方式。系统计时器提供 1 个 24 位的循环递减计数器，计数器按照 CPU 时钟或者外部参考时钟递减计数，计数到 0 时产生中断请求。功耗管理模块支持动态功耗和静态功耗的模式控制。高速缓存控制寄存器单元在配置有高速缓存（cache）时负责对其进行控制及操作。CK802 同时设计有针对信息安全应用的可配置模块。

CK802 处理器实现了二进制代码转译机制，支持对 JAVA 等解释性语言的加速。用户可以通过选用支持该功能的 CK802 核，实现对 JAVA 应用的加速。

*注：详细内容请参考紧耦合 IP 用户手册。

1.3. 编程模型



注：1、可配置资源；2、仅在二进制代码转译机制中实现

图表 1-2 编程模型

CK802 定义了两种编程模式：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。处理器复位后工作在超级用户模式下。

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）对控制寄存器的访问；2）特权指令的使用;3)对于紧耦合 IP 的控制寄存器访问。普通用户模式只允许访问通用寄存器；超级用户模式可以访问所有的通用寄存器和控制寄存器。用户程序因此可以避免接触特权信息，而操作系统通过协调与用户程序的行为来为用户程序提供管理和服务。超级用户模式下可以访问所有紧耦合 IP 的控制寄存器，用于调度 CPU 资源。

普通用户模式下可以访问普通用户堆栈指针（后续段落简称 user SP）。在普通用户模式下不能访问超级用户堆栈指针（后续段落简称 spv SP）。

普通用户模式下可以访问链接寄存器（后续段落简称 LR），该 LR 与超级用户模式共享。

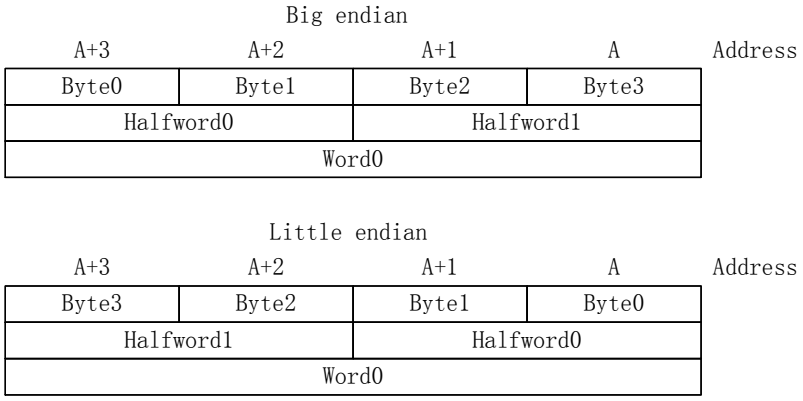
普通用户模式下，条件/进位位（C）位于 PSR 的最低位，可以被访问和更改，是 PSR 中唯一能在普通用户模式下被访问的数据位。

普通用户模式可以使用绝大多数的指令，除了对系统产生重大影响的特权指令如 stop，doze，wait，mfcr，mtcr，psrset，psrlcr，rte 之外。普通用户模式可以通过使用 trap #n 指令来进入超级用户模式。

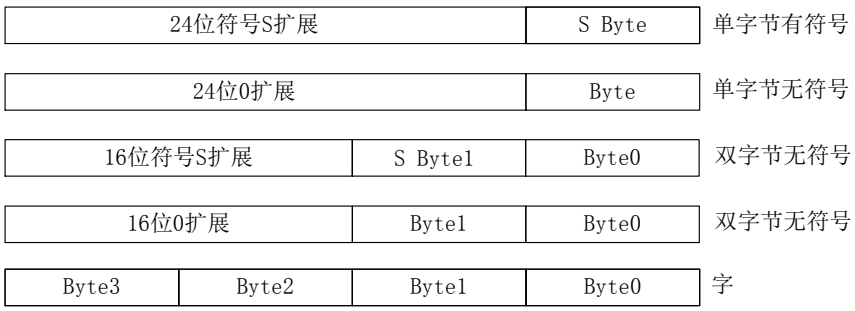
超级用户模式下可以访问所有通用寄存器以及控制寄存器。在超级用户模式下可以访问 user SP 和 spv SP。如果用户需要在超级用户模式下访问 uesr SP，则需要使用 mtcr rx cr<14,1>和 mfcr rz cr<14,1>来实现。

超级用户模式下可以使用 CK802 支持的所有指令。

1.4. 数据格式



图表 1-3 内存中的数据组织形式



图表 1-4 寄存器中的数据组织结构

CK802 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，指令使用 32 位操作数，产生 32 位结果。

CK802 的存储器可以配置成大端模式或小端模式。在大端模式下，字 0 的最高位字节放在地址 0 上。而在小端模式（缺省模式）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。

1.5. 指令集一览

CK802 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 1-5 列出了 CK802 指令集中所有 16 位 32 位指令 。

图表 1-5 CK802 的指令集

汇编指令	32 位	16 位	指令描述
ADDU	○	○	无符号加法指令
ADDC	○	○	无符号带进位加法指令

ADDI	○	○	无符号立即数加法指令
SUBU	○	○	无符号减法指令
SUBC	○	○	无符号带借位减法指令
SUBI	○	○	无符号立即数减法指令
RSUB	○	×	反向减法指令
IXH	○	×	索引半字指令
IXW	○	×	索引字指令
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
AND	○	○	按位与指令
ANDI	○	×	立即数按位与指令
ANDN	○	○	按位非与指令
ANDNI	○	×	立即数按位非与指令
OR	○	○	按位或指令
ORI	○	×	立即数按位或指令
XOR	○	○	按位异或指令
XORI	○	×	立即数按位异或指令
NOR	○	○	按位或非指令
NOT	○	○	按位非指令
LSL	○	○	逻辑左移指令
LSLI	○	○	立即数逻辑左移指令
LSLC	○	×	立即数逻辑左移至 C 位指令
LSR	○	○	逻辑右移指令
LSRI	○	○	立即数逻辑右移指令
LSRC	○	×	立即数逻辑右移至 C 位指令
ASR	○	○	算术右移指令
ASRI	○	○	立即数算术右移指令
ASRC	○	×	立即数算术右移至 C 位指令

ROTL	○	○	循环左移指令
ROTLI	○	×	立即数循环左移指令
XSR	○	×	扩展右移指令
CMPNE	×	○	不等比较指令
CMPNEI	○	○	立即数不等比较指令
CMPHS	×	○	无符号大于等于比较指令
CMPHSI	○	○	立即数无符号大于等于比较指令
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
MVCV	×	○	C 位取反传送指令
MVC	○	×	C 位传送指令
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
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

REVB	×	○	字节倒序指令
REVH	×	○	半字内字节倒序指令
MULT	○	○	乘法指令
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
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
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
ST.B	○	○	字节存储指令
ST.H	○	○	半字存储指令
ST.W	○	○	字存储指令
LDM	○	×	连续多字加载指令
STM	○	×	连续多字存储指令
PUSH	×	○	压栈指令
POP	×	○	出栈指令
*BPUSH.H	×	○	二进制转译半字压栈指令
*BPUSH.W	×	○	二进制转译字压栈指令
*BPOP.H	×	○	二进制转译半字出栈指令
*BPOP.W	×	○	二进制转译字出栈指令

**IPUSH	×	○	中断压栈指令
**IPOP	×	○	中断出栈指令
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
SYNC	○	×	CPU 同步指令
BKPT	×	○	断点指令
TRAP	○	×	无条件操作系统陷阱指令
**NIE	×	○	中断嵌套使能指令
**NIR	×	○	中断嵌套返回指令
*BMSET	○	×	BM 位置位指令
*BMCLR	○	×	BM 位清零指令
*JMPIX	×	○	寄存器索引跳转指令

注：○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。*表示二进制代码转译机制增加的指令，**表示中断嵌套增强指令。

2. 命名规则

2.1. 符号

本文档用到的标准符号和操作符如下表所示

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
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

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 代表最低有效字节,MSB 代表最高有效字节。
  存储单元和寄存器当“pad_sysio_bigend_b=0”时采用大端模式，其字节次序是高字节在最低位。一个字中的字节是从最高有效字节(第 31—24 位)开始往下排列。
- 当“pad_sysio_bigend_b=1”时，采用小端模式。
- 信号，位域，控制位的表示都使用一种通用的规则。
- 标识符后来跟着表示范围的数字，从高位到低位表示一组信号，比如 addr[4:0]就表示一组地址总线，最高位是 addr[4]，最低位是 addr[0]。
- 单个的标识符就表示单个信号，例如 pad_cpu_rst_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。

3. 寄存器描述

本章主要介绍 CK802 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：

- 16 个 32 位通用寄存器（R15~R0）；
- 32 位程序计数器（PC）；
- 条件码 / 进位标志位（C bit）。
- 二进制转译堆栈栈底寄存器 R23（二进制代码转译机制时实现）
- 二进制转译堆栈栈顶寄存器 R24（二进制代码转译机制时实现）
- 二进制转译堆栈栈针寄存器 R25（二进制代码转译机制时实现）
- 软件向量基址寄存器 R30（二进制代码转译机制时实现）

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(user)	链接寄存器
R23(fp')	二进制转译堆栈栈底寄存器（二进制代码转译机制时实现）
R24(top)	二进制转译堆栈栈顶寄存器（二进制代码转译机制时实现）

R25(bsp)	二进制转译堆栈栈针寄存器（二进制代码转译机制时实现）
R30(svbr)	软件向量基址寄存器（二进制代码转译机制时实现）

PC	程序计数器

C	条件码/进位标志

图表 3-1 普通用户编程模式寄存器

3.1.1.通用寄存器

通用寄存器包含了指令操作数和结果以及地址信息。软硬件上约定这些通用寄存器做为子程序的链接调用，参数传递以及堆栈指针等功能。

其中用于普通用户堆栈指针的寄存器 R14 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。程序计数器的值不能被指令直接修改，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，程序计数器中的最低位一直为零。

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进位或条件判断结果。条件码／进位标志位能够作为比较操作指令的结果被置位，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置位。另外，特殊的指令如 XTRB[0-3]等也会影响条件码／进位标志位的值。

3.2. 超级用户编程模式

系统程序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通用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堆栈指针寄存器（R14）
- 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)；
- 32 位全控制寄存器(GCR)（可配置宽度）*；
- 32 位全状态寄存器(GSR)（可配置宽度）*；
- 产品序号寄存器(CPUIDR)；
- 高速缓存配置寄存器(CCR)；
- 访问权限配置寄存器(CAPR) *；
- 保护区控制寄存器(PACR) *；
- 保护区选择寄存器(PRSR) *；

- 软件复位控制寄存器（SRCR）。

*注：可选择寄存器仅在特定配置时有效。

图表 3-2 超级用户编程模式附加资源

3.2.1.超级用户编程模式的堆栈指针寄存器 R14(spv SP)

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 SP)，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。若要在超级用户模式下访问 R14，可通过 mfcr/mtcr 访问 CR<14,1>完成。

3.2.2.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）。处理器状态寄存器指示处理器处于超级用户模式或者普通用户模式（S 位）。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等。

	31	30	28	27	24	23	16
	S	0		0		VEC［7：0］	
Reset	1	0				0	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		0	0	0		MM	EE	IC	IE	0	0	0	*BM	0	C
Reset	0		0	0	0		0	0	0	0	0	0	0	0	0	0

图表 3-3 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；

该位在被 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。

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，处理器会响应该非对齐异常；

当 MM 为 1 时，读取或存储的地址不对齐，处理器不会响应该非对齐异常：若处理器硬件支持非对齐访问，则处理器使用该非对齐地址对存储器进行非对齐访问；若处理器硬件不支持非对齐访问，则处理器将该非对齐地址的低位强行置 0 后对存储器进行对齐访问。在任何情况下，只要多周期内存访问指令（如 STM、LDM、PUSH、POP、NIE、NIR、IPUSH、IPOP 等）发生地址非对齐，处理器都要响应非对齐异常。

未对齐的具体操作实现如下：

1，地址为 1，2,3 的 word 读访问会在总线上出现两次 word 读操作，地址分别为 0 和 4。
2，地址为 1 的 half word 的读访问，会出现一次 word 读操作，地址为 0。
3，地址为 3 的 half word 的读访问，会出现两次 word 读操作，地址为 0 和 4。
4，地址为 1 的 word 写操作会在总线上出现地址为 1 的 byte 写，地址为 2 的 half word 写，地址为 4 的 byte 写。
5，地址为 2 的 word 写操作会在总线上出现地址为 2 的 half 写，地址为 4 的 half word 写。
6，地址为 3 的 word 写操作会在总线上出现地址为 3 的 byte 写，地址为 4 的 half word 写，地址为 6 的 byte 写。
7，地址为 1 的 half word 写操作会在总线上出现地址为 1 的 byte 写，地址为 2 的 byte 写。
8，地址为 3 的 half word 写操作会在总线上出现地址为 3 的 byte 写，地址为 4 的 byte 写。
该位会被 reset 清零。

EE-异常有效控制位：

当 EE 为 0 时，异常无效，此时除了普通中断之外的任何异常一旦发生，都会被 CK802 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。

该位会被 reset 清零，也在处理器响应异常时被清零。

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；

该位会被 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，EPC 和 EPSR 都无效；

当 IE 为 1 时，中断有效，（此时 EE 位也需要为 1，否则中断依然无效）；

该位会被 reset 清零，也在处理器响应异常时被清零。

***BM-二进制代码转译模式控制位：**

当 BM 为 0 时，处理器工作在正常模式；

当 BM 为 1 时，处理器工作在、二进制代码转译模式，影响 LD/ST/BPUSH/BPOP 等指令的执行；

该位会被 reset 清零，不受其它异常影响。

注：该位在处理器配置了二进制代码转译机制时实现，若处理器不支持该机制，该位恒为 0。

C-条件码 / 进位位

该位用作条件判断位为一些指令服务。

该位会被 reset 清零。

3.2.2.1. 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，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，VEC，IE，EE 位。对 S，VEC，IE， 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 位的改动优先于异常服务程序中的第一条指令的执行。

- RTE 指令更新 PSR：

更新 PSR 作为 rte 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，IE，EE、BM 的改动优先于对返回 PC 的取址，对 VEC, MM, IC 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。

- BMCLR、BMSET 指令更新 PSR：

更新 PSR 作为 bmclr 和 bmset 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。

3.2.3.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。

图表 3-4 基址向量寄存器

3.2.4.异常保留寄存器（CR<2,0>～CR<5,0>）

EPSR 和 EPC 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第六章异常处理。

3.2.5.全局控制寄存器（GCR，CR<11,0>）

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。在 CK802 中全局控制寄存器的位宽是硬件可配置的。

3.2.6.全局状态寄存器（GSR，CR<12,0>）

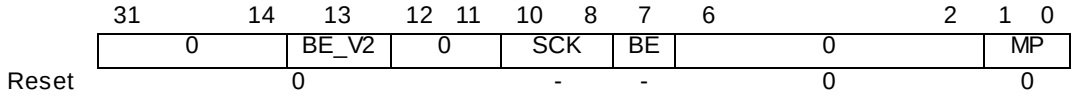
全局状态寄存器是用来标记外围设备和事件的。它通过芯片口上提供的输入接口将外部状态送入到 CK802 内部，从而实现监测。一般来说，可以通过查看 GSR 来检测外围设备状态和事件。全状态寄存器是只读的。在 CK802 中全局状态寄存器的位宽是硬件可配置的。

3.2.7.产品序号寄存器（CPUIDRR，CR<13,0>）

该寄存器用于存放杭州中天微系统有限公司产品的内部编号。产品序号寄存器是只读的，其复位值由产品本身决定。CK802 产品序号寄存器版本为 3.1 版，具体定义请参考《C-SKY 产品 ID 定义规范（3.1 版）》。

3.2.8.高速缓存配置寄存器（CCR，CR<18,0>）

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



图表 3-5 高速缓存配置寄存器

BE_V2-V2 版本大小端：

当 BE_V2 为 0 时，非 V2 版本大小端；

当 BE_V2 为 1 时，V2 版本大小端；

该位与 BE 一起决定处理器具体工作的大小端模式，该位仅在 BE 为 1 时起作用；

BE_V2 在 power on reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

SCK-系统和处理器的时钟比：

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

该域目前没有任何功能，只供软件查询。

BE-Endian 模式：

当 BE 为 0 时，小端；

当 BE 为 1 时，大端；

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

MP-内存保护设置位：

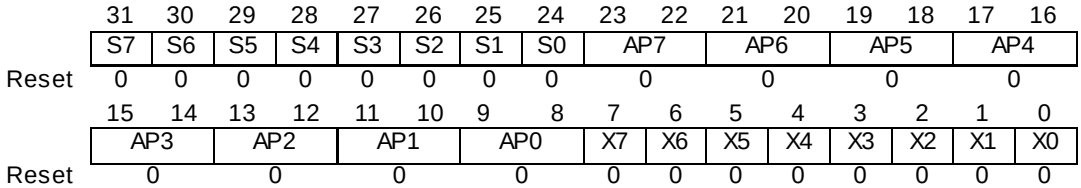
MP 用来设置 MPU 是否有效，如下表：

MP	功能
00	MPU 无效
01	MPU 有效

图表 3-6 CK802 内存保护设置

3.2.9.可高缓和访问权限配置寄存器（CAPR，CR<19,0>）

CAPR 的各位如下图所示：



图表 3-7 可高缓和访问权限配置寄存器

X0~X7-不可执行属性设置位：

当 X 为 0 时，该区为可执行区；

当 X 为 1 时，该区为不可执行区。

注：当处理器取指访问到不可执行的区域时，会出现访问错误异常。

S0~S7-安全属性设置位：

当 S 为 0 时，该区非安全区；

当 S 为 1 时，该区安全区。

注：该区域决定内存访问的安全属性，该属性会被传递到总线上。

AP0~AP7-访问权限设置位：

AP	超级用户权限	普通用户权限
00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

图表 3-8 访问权限设置

3.2.10. 保护区控制寄存器（PACR，CR<20,0>）

PACR 的各位如下图所示：

	31 … 10	9 … 6	5 … 1	0
	Base Address	0	Size	E
Reset	-	0	-	0

图表 3-9 保护区控制寄存器

Base Address-保护区地址的高位：

该寄存器指出了保护区地址的高位，但写入的基地址必须与设置的页面大小对齐，例如设置页面大小为 8M，CR<20,0>[22:10]必须为 0，各页面的具体要求见下表：图表 3-10 保护区大小配置和其对基址要求。

Size-保护区大小：

保护区大小从 4KB 到 4GB，它可以通过公式：保护区大小＝2^（Size+1）计算得到。因此 Size 便从 01011 到 11111，其它一些值都会造成不可预测的结果。

Size	保护区大小	对基地址低位的要求
00000—01010	保留	—
01001	1KB	无要求
01010	2KB	CR<20,0>.bit[10]=0
01011	4KB	CR<20,0>.bit[11:10]=0
01100	8KB	CR<20,0>.bit[12:10]=0
01101	16KB	CR<20,0>.bit[13:10] =0

Size	保护区大小	对基地址低位的要求
01110	32KB	CR<20,0>.bit[14:10] =0
01111	64KB	CR<20,0>.bit[15:10] =0
10000	128KB	CR<20,0>.bit[16:10] =0
10001	256KB	CR<20,0>.bit[17:10] =0
10010	512KB	CR<20,0>.bit[18:10] =0
10011	1MB	CR<20,0>.bit[19:10] =0
10100	2MB	CR<20,0>.bit[20:10] =0
10101	4MB	CR<20,0>.bit[21:10] =0
10110	8MB	CR<20,0>.bit[22:10] =0
10111	16MB	CR<20,0>.bit[23:10] =0
11000	32MB	CR<20,0>.bit[24:10] =0
11001	64MB	CR<20,0>.bit[25:10] =0
11010	128MB	CR<20,0>.bit[26:10] =0
11011	256MB	CR<20,0>.bit[27:10] =0
11100	512MB	CR<20,0>.bit[28:10] =0
11101	1GB	CR<20,0>.bit[29:10] =0
11110	2GB	CR<20,0>.bit[30:10] =0
11111	4GB	CR<20,0>.bit[31:10] =0

图表 3-10 保护区大小配置和其对基址要求

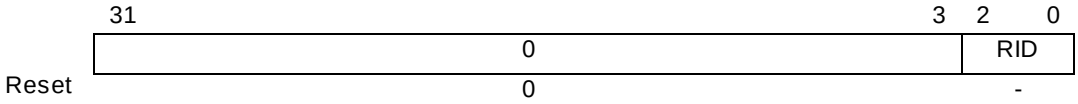
E-保护区有效设置：

当 E 为 0 时，保护区无效；

当 E 为 1 时，保护区有效。

3.2.11. 保护区选择寄存器（PRSR，CR<21,0>）

PRSR 用来选择当前操作的保护区，其各位如下图所示：



图表 3-11 保护区选择寄存器

RID-保护区索引值：

RID 表示所选择的对应的保护区，如 000 表示第 0 保护区。

3.2.12. MPU 使用操作

3.2.12.1.MPU 有效控制

CR<20,0>中的第 0 位是 MPU 有效控制位。在 MPU 有效之前，至少有一个区被指定以及它相应的 X、S 和 AP 也必须被设置。此外，这个让 MPU 有效的指令必须在指令地址访问有效的范围之内，即此指令所在的区域不可以在 MPU 中设置为拒绝访问。若不这么做，将会导致不可预测的结果。

3.2.12.2.内存访问处理

当 MPU 被使能后，在内存访问信号产生时，MPU 会检查当前访问的地址是否在这些保护区内：

如果访问的地址不在这些区中的任何一个，此内存访问会中途停止；

如果访问的地址在这些区中的一个或多个内，此访问被已使能的最高索引区（7 为最高，0 为最低）所控制。

3.2.12.3.内存访问起始地址设置

CR<20,0>中定义了四/八个保护区的起始地址和大小。保护区大小必须是 2 的幂，能从 1KB 到 4GB。起始地址必须与区大小对齐，比如：8KB 大小的保护区，起始地址可以是 32'h12346000，但是 16KB 大小的保护区，起始地址就不可以为这个值，可以是 32'h12344000。

3.2.13. 软件复位控制寄存器（SRCR，CR<31,0>）

CR<31,0>用以实现处理器软件复位功能，通过 MTCR 指令向该寄存器中写入特定值即可实现处理器的软件复位，该特定值可由用户通过静态配置的方式指定。

软件复位操作会复位处理器中的所有通用寄存器、控制寄存器，并使处理器向外发起一个软件复位标识信号。另外，若处理器在正常运行模式下执行软件复位指令，则处理器将进入复位异常处理器程序（即异常向量号为零的异常服务程序）执行相应操作；若在调试模式下执行该软件复位指令，则处理器保持在调试模式，但在退出调试模式后自动进入复位异常处理器程序（即异常向量号为零的异常服务程序）执行相应操作。

CR<31,0>的读操作将无条件返回 0，另外对该寄存器写入除软件复位对应的特定值之外的任何其他值，将不产生任何效果。

3.2.14. 普通用户模式堆栈指针寄存器 14（R14(user), CR<14,1>）

为简化硬件设计，普通用户模式通用寄存器 14 映射为控制寄存器 CR<14,1>。

3.3. 二进制代码转译模式

CK802 实现了二进制代码转译模式(Binary Code Translation Mode)，加速 JAVA 等解释性语言的执行。二进制代码转译功能在普通 CK802 核基础上增加了以下资源：

- 二进制转译堆栈栈底寄存器 FP'，位于第 23 号通用寄存器（R23）。
- 二进制转译堆栈栈顶寄存器 TOP，位于第 24 号通用寄存器（R24）。
- 二进制转译堆栈栈针寄存器 BSP，位于第 25 号通用寄存器（R25）。
- 软件矢量基址寄存器 SVBR，位于第 30 号通用寄存器（R30）。

- 二进制代码转译模式位 BM，位于 PSR[2]。
- 增加了 BMSET/BMCLR/JMPIX/BPUSH/BPOP 等指令。

普通用户设置 PSR 的 BM 位使得处理器运行于二进制代码转译模式，通过清除 BM 位使得处理器回复至正常模式运行。相比正常模式，二进制代码转译模式将影响加载存储操作的运行。

在二进制代码转译模式下，处理器对加载存储操作的基地址合法性检查。一旦二进制堆栈访问溢出（仅限于 BPUSH/BPOP 指令），则内存访问操作不执行并抛出软件异常，处理器将下条指令的 PC 保存至链接寄存器 R15，同时从 SVBR-12（针对二进制堆栈访问溢出）地址获取跳转入口地址并跳转执行。如果二进制堆栈访问未溢出，则内存访问操作正常完成。通过硬件监测，避免了软件在二进制转译中通过软件比较的操作，提高运行速度。

二进制代码转译模式新增的指令具体见附件 B 的指令术语表。

4. 32 位指令

本章主要介绍 CK802 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

CK802 的 32 位指令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：

图表 4-1 32 位加减法指令列表

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
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

逻辑操作指令：

图表 4-2 32 位逻辑操作指令列表

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：

图表 4-3 32 位移位指令列表

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：

图表 4-4 32 位比较指令列表

CMPNEI32	立即数不等比较指令
CMPHSI32	立即数无符号大于等于比较指令
CMPLTI32	立即数有符号小于比较指令

数据传输指令：

图表 4-5 32 位数据传输指令列表

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
MVC32	C 位传送指令
LRW32	存储器读入指令

比特操作指令：

图表 4-6 32 位比特操作指令列表

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：

图表 4-7 32 位提取插入指令列表

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

乘除法指令：

图表 4-8 32 位乘除法指令列表

MULT32	乘法指令

杂类运算指令：

图表 4-9 32 位杂类运算指令列表

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
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：

图表 4-10 32 位分支指令列表

BT32	C 为 1 分支指令
BF32	C 为 0 分支指令

跳转指令：

图表 4-11 32 位跳转指令列表

BR32	无条件跳转指令
BSR32	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：

图表 4-12 32 位立即数偏移存取指令列表

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

多寄存器存取指令：

图表 4-13 32 位多寄存器存取指令列表

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

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：

图表 4-14 32 位控制寄存器操作指令列表

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：

图表 4-15 32 位低功耗指令列表

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：

图表 4-16 32 位异常返回指令列表

RTE32	异常和普通中断返回指令

4.1.5. 特殊功能指令

特殊功能指令具体包括：

图表 4-17 32 位特殊功能指令列表

SYNC32	CPU 同步指令
TRAP32	无条件操作系统陷阱指令
BMSET32	BCTM 位置位指令
BMCLR32	BCTM 位清零指令

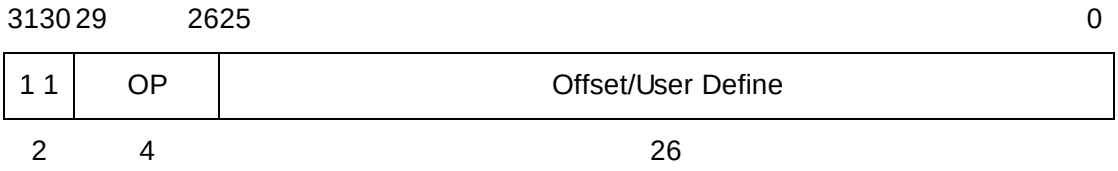
4.2. 32 位指令编码方式

CSKY V2 的 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）的编码方式如下图：

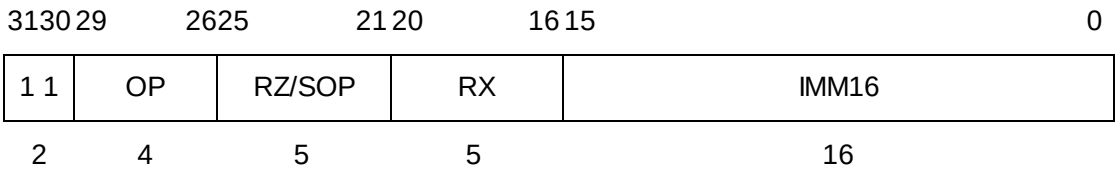


OP 域为主操作码，通过 4 位主操作码可以识别该编码类型的指令；Offset/User Define 域为跳转指令的偏移量或者供用户自定义的保留域。

4.2.2. 立即数类型

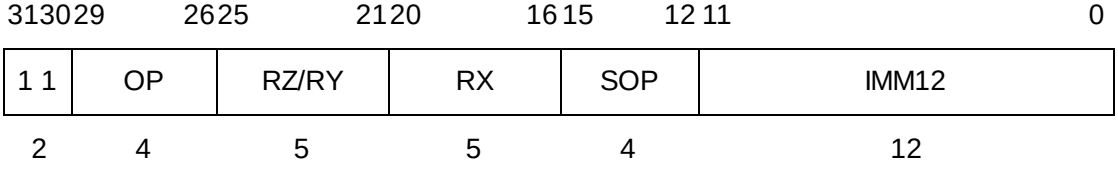
32 位指令立即数类型（I 型）包含 16 位立即数和 12 位立即数两种编码方式。

16 位立即数的编码方式如下图：



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图：



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）的编码方式如下图：

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

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

CSKY 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

CSKY 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。

31 30 29	26 25	0
1 1	OP	Offset
2	4	26

4.3.2. 立即数类型编码指令寻址方式

CSKY V2 中立即数类型编码的 32 位指令有四种寻址方式。

4.3.2.1. 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。

31 30 29	26 25	21 20	16 15	0

1 1	OP	RZ	RX	IMM16
2	4	5	5	16

4.3.2.2. 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，可以进一步分为二种格式。

第一种格式中，SOP 域为子操作码域；RX 域为源寄存器域；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 cmphsi32、cmplti32。

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

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	RZ	IMM16
2	4	5	5	16

4.3.2.3. 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。

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

4.3.2.4. 二元寄存器十二位立即数寻址方式

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.bs、ld32.hs、st32.b、st32.h、st32.w、addi32、subi32、andi32、andni32、xori32。

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

4.3.3. 寄存器类型编码指令寻址方式

CSKY 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

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、and32、andn32、or32、xor32、nor32、lsl32、lsr32、asr32、rotl32、mult32。

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

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。

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 lsli32、lsri32、asri32、rotli32、lslc32、lsrc32、asrc32、xsr32、bclri32、bseti32。

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

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

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，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 xtrb0.32、xtrb1.32、xtrb2.32、xtrb3.32、ff0.32、ff1.32。

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

2	4	5	5	6	5	5

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。

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。

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

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。

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

4.3.3.5. 一元寄存器寻址方式

一元寄存器寻址方式中，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。

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

5. 16 位指令

本章主要介绍 CK802 的 16 位指令集，包括 16 位指令集的功能分类，编码方式和寻址模式等。

5.1. 16 位指令功能分类

CK802 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

5.1.1. 数据运算指令

数据运算类指令可以进一步分为：

加减法指令：

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

图表 5-1 16 位加减法指令列表

逻辑操作指令：

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

图表 5-2 16 位逻辑操作指令列表

移位指令：

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

图表 5-3 16 位移位指令列表

比较指令：

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

图表 5-4 16 位比较指令列表

数据传输指令：

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

图表 5-5 16 位数据传输指令列表

比特操作指令：

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

图表 5-6 16 位比特操作指令列表

提取插入指令：

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

图表 5-7 16 位提取插入指令列表

乘除法指令：

MULT16	乘法指令

图表 5-8 16 位乘法指令列表

5.1.2. 分支跳转指令

分支跳转指令可以进一步分为：

分支指令：

BT16	C 为 1 分支指令
BF16	C 为 0 分支指令

图表 5-9 16 位分支指令列表

跳转指令：

BR16	无条件跳转指令
JMP16	寄存器跳转指令
JSR16	寄存器跳转到子程序指令
RTS16	链接寄存器跳转指令
JMPIX16	寄存器索引跳转指令

图表 5-10 16 位跳转指令列表

5.1.3. 内存存取指令

内存存取指令可以进一步分为：

立即数偏移存取指令：

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

图表 5-11 16 位立即数偏移存取指令列表

多寄存器存取指令：

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

图表 5-12 16 位多寄存器存取指令列表

注：NIE16 和 NIR16 需要在特权模式下执行。

16 位二进制转译堆栈指令：

BPUSH16.H	二进制转译半字压栈指令
BPUSH16.W	二进制转译字压栈指令
BPOP16.H	二进制转译半字出栈指令
BPOP16.W	二进制转译字出栈指令

图表 5-13 16 位二进制转译堆栈指令

5.1.4. 特权指令

特权指令具体包括：

NIE16	中断嵌套使能指令
NIR16	中断嵌套返回指令

图表 5-14 16 位特殊功能指令列表

注：NIE16 和 NIR16 同时也是多寄存器存取指令。

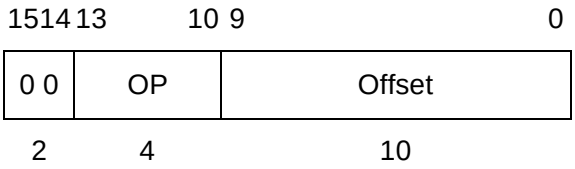
5.2. 16 位指令编码方式

CSKY 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）

5.2.1. 跳转类型

跳转类型（J 型）的编码方式如下图：

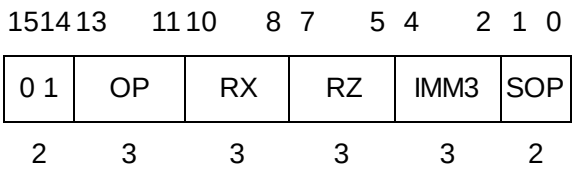


OP 域为主操作码，通过 4 位主操作码可以识别该编码类型的指令；Offset 域为跳转指令的偏移量。

5.2.2. 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。

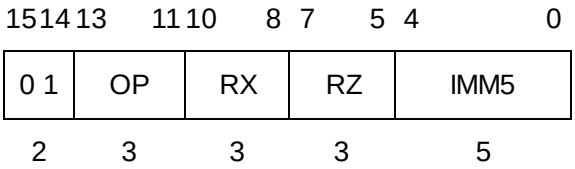
3 位立即数的编码方式如下图：



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

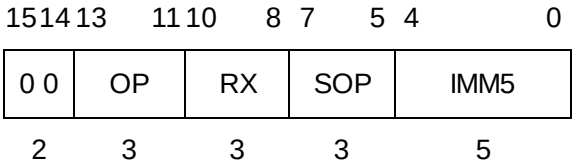
域；RZ 域为目的寄存器域； IMM3 域为 3 位立即数；SOP 域为子操作码域。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

5 位立即数的编码方式有三种格式，第一种格式如下图：



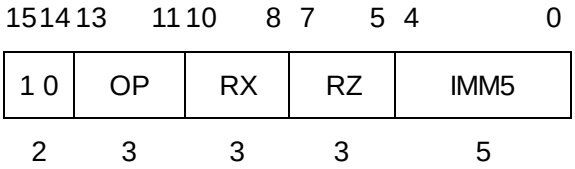
OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域为 5 位立即数。

5 位立即数的第二种编码方式如下图：



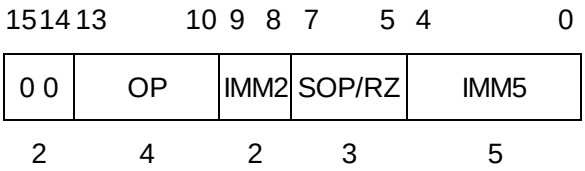
OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；SOP 域为子操作码域；IMM5 域为 5 位立即数。指令在经过主操作码 OP 的译码之后得出指令类型，需要对子操作码 SOP 的进一步译码才能得到具体指令。

5 位立即数的第三种编码方式如下图：



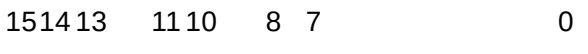
OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；RX 域为源寄存器域；RZ 域为目的寄存器域；IMM5 域为 5 位立即数。

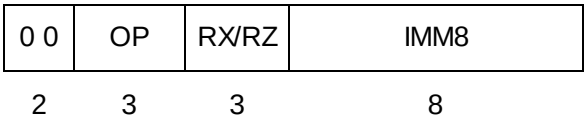
7 位立即数的编码方式如下图：



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。

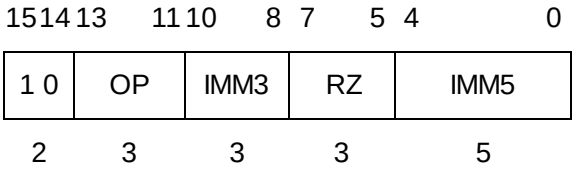
8 位立即数的编码方式有两种格式，第一种格式如下图：





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：

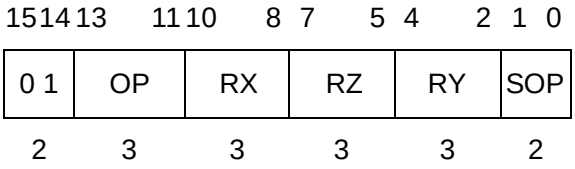


OP 域为主操作码，通过 3 位主操作码可以识别指令或者指令类型；IMM3 域和 IMM5 域分别为 8 位立即数的高 3 位和低 5 位；RZ 域为目的寄存器域。

5.2.3. 寄存器类型

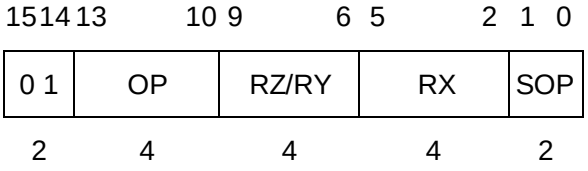
寄存器类型（R 型）包含 3 操作数和 2 操作数两种编码方式。

3 操作数的编码方式如下图：



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：



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。

5.3. 16 位指令操作数寻址模式

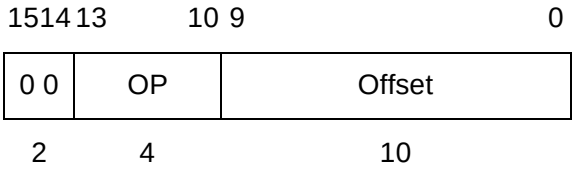
CSKY V2 的 16 位指令集总体遵循三种指令编码方式，每种编码方式都有自己特有的操作数寻址模式，下面将分别介绍各种操作数寻址模式。

5.3.1. 跳转类型编码指令寻址方式

CSKY V2 中跳转类型编码的 16 位指令只有一种寻址方式。

5.3.1.1. 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、bt16、bf16。

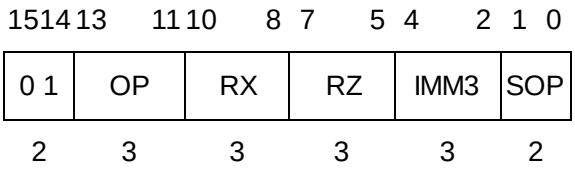


5.3.2. 立即数类型编码指令寻址方式

CSKY V2 中立即数类型编码的 16 位指令有六种寻址方式。

5.3.2.1. 二元寄存器三位立即数寻址方式

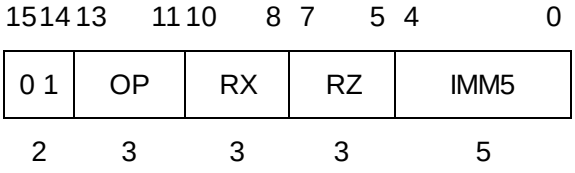
二元寄存器三位立即数寻址方式指令中，RX 域为源寄存器域；RZ 域为目的寄存器域；IMM3 域也可以作为 3 位立即数直接参与数据运算；SOP 域为子操作码域。采用这种格式的指令包括 addi16、subi16。



5.3.2.2. 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。



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。

2	3	3	3	5

5.3.2.3. 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、btsti16。

15 14 13 11 10 8 7 5 4 0

0 0	OP	RX	SOP	IMM5
2	3	3	3	5

5.3.2.4. 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。

15 14 13 10 9 8 7 5 4 0

0 0	OP	IMM2	RZ	IMM5
2	4	2	3	5

5.3.2.5. 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、bpush16.h、bpush16.w、bpop16.h、bpop16.w、addi16(SP)、subi16(SP)。

15 14 13 10 9 8 7 5 4 0

0 0	OP	IMM2	SOP	IMM5
2	4	2	3	5

5.3.2.6. 一元寄存器八位立即数寻址方式

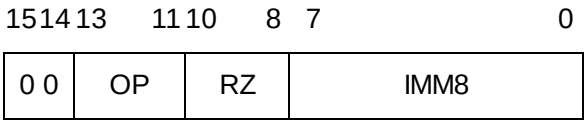
一元寄存器八位立即数寻址方式指令中，可以进一步分为三种格式。

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。采用这种格式的指令包括 addi16(SP)、subi16(SP)、movi16。

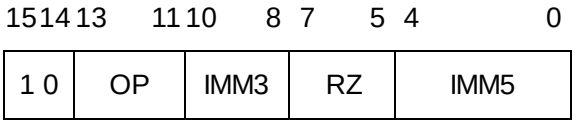
15 14 13 11 10 8 7 0

0 0	OP	RZ	IMM8
2	3	3	8

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。采用这种格式的指令包括 addi16、subi16。



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)。



5.3.3. 寄存器类型编码指令寻址方式

CSKY V2 中寄存器类型编码的 16 位指令有三种寻址方式。

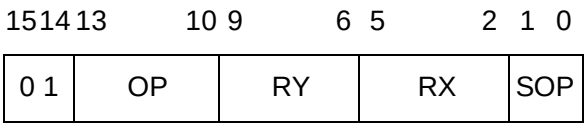
5.3.3.1. 三元寄存器寻址模式

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。

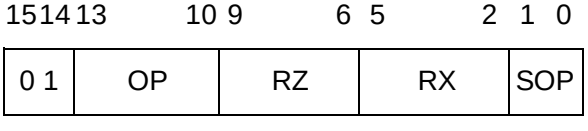
5.3.3.2. 二元寄存器寻址模式

二元寄存器寻址方式指令中，可以进一步分为三种格式。

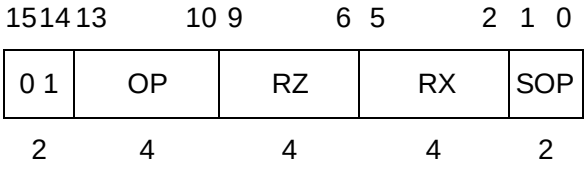
第一种格式中，两个寄存器域 RX、RY 分别为第一源寄存器域和第二源寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 cmphs16、cmplt16、cmpne16、tst16。



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。



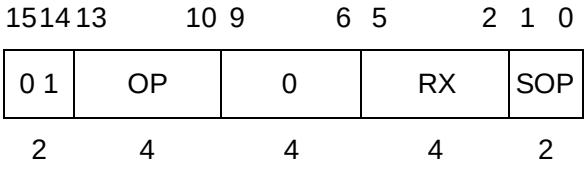
第三种格式中，RZ 域为目的寄存器域和第二源寄存器域；RX 为第一源寄存器域；SOP 域为子操作码域。采用这种编码方式的指令有 addu16、addc16、subu16、subc16、and16、andn16、or16、xor16、nor16、lsl16、lsr16、asr16、rotl16、mult16。



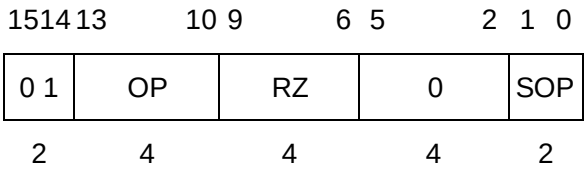
5.3.3.3. 一元寄存器寻址模式

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。



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。



6. 指令流水线

本章介绍关于 CK802 的指令流水线和指令时序信息。

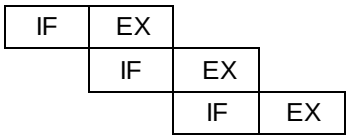
CK802 微处理器有 2 级流水线： 即指令提取与译码、指令执行与退休。2 级流水线的作用如图表 7-1：

流水线名称	缩写	流水线作用
指令提取与译码	IF	1、访问指令总线； 2、计算下一条指令的地址； 3、分支地址计算； 4、指令预译码； 5、指令译码； 6、复杂指令拆解；
指令执行与退休	EX	1、 指令发射； 2、 访问寄存器组； 3、 指令的执行； 4、 load/store 指令的数据地址的产生； 5、 访问数据总线； 6、 指令执行结果回写。

图表 6-1 各级流水线作用

在流水执行指令的过程中，采用单发射机制，即一个时钟周期至多发射一条指令；同时采用阻塞发射架构，即前一条指令没有执行完成时，后续指令不得发射到执行单元。

单周期指令流水线重叠执行顺序如图表 6-2 所示，load, store、算术和逻辑指令都属于这类指令。

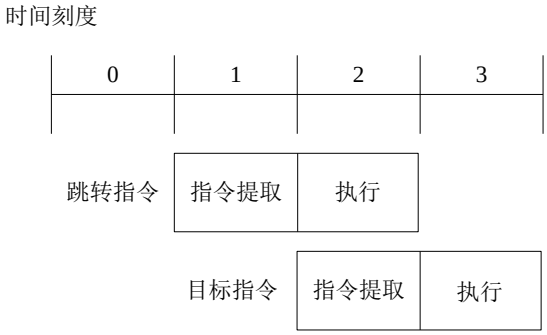


图表 6-2 单周期指令流水线重叠执行

乘法指令 mult 需要多个执行周期完成（不可流水操作），如图表 6-3 所示：

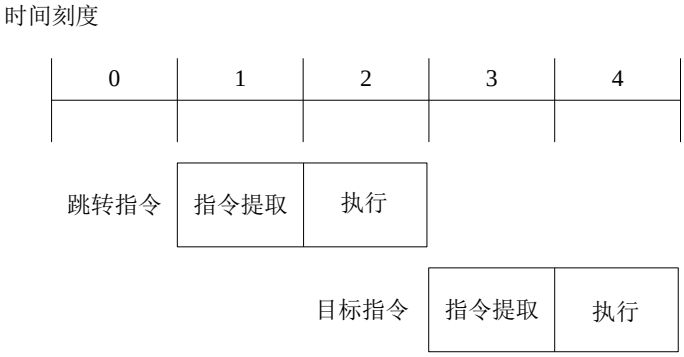
图表 6-3 乘法指令 mult 的执行过程

br, bsr 指令的跳转和条件分支指令，由于采用了提前获取条件位技术，即使出现跳转流水线也不停顿，其执行过程如图表 6-4 所示；



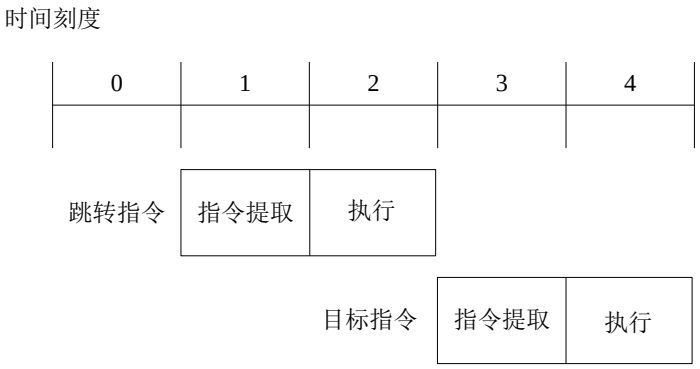
图表 6-4 BR, BSR 指令的跳转和条件指令预测正确时的执行过程

jmp 指令（jmp r15 指令除外）至少需要两个周期来填充流水线，其执行过程如图表 6-5 所示：



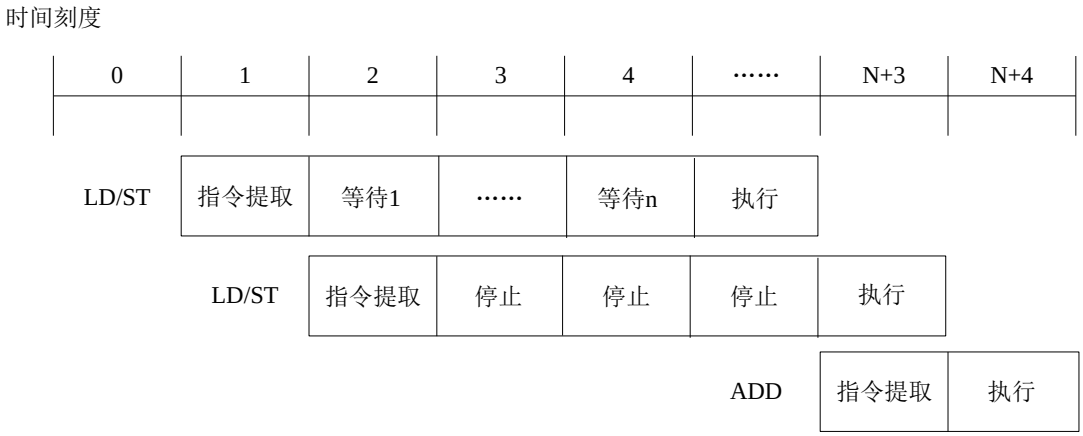
图表 6-5 JMP 指令执行过程

如果跳转指令的目标指令是字未对齐的 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。图表 6-6 显示了一条 BR 指令在目标指令是一条 32 位字未对齐指令 ADD 的执行过程：



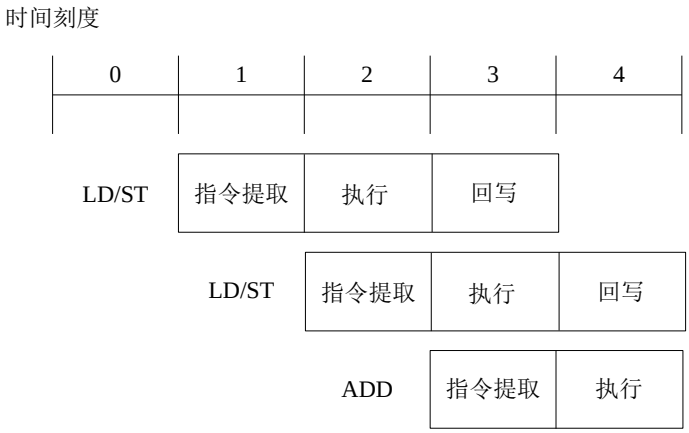
图表 6-6 跳转指令的目标指令是 32 位字未对齐指令的执行过程

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，因为采用的是阻塞发射机制，必须等到访问存储区的指令完成之后这些指令才能执行。图表 6-7 显示了一条有等待状态的 ld/st 后跟一条无等待状态的 ld/st 和一条单周期指令 ADD 的执行过程：



图表 6-7 带有等待状态的指令流水执行过程

对于配置 load/store 有快速退休机制的 CK802，且存储器访问都可以在单周期内完成，其时序图如图表 6-8 所示。



图表 6-8 具有快速退休功能的指令流水执行过程

7. 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、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、非对齐错误（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。即使异常指令退休前被识别，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。CK802 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（PC+2/PC+4，根据当前指令是 16 位或 32 位决定+2 或者+4）将被保存在异常地址寄存器（EPC）中作为中断返回时指令的入口；如果异常事件是由访问错误指令产生的，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)，这条访问错误地址指令的地址（PC）将被保存在异常地址寄存器（EPC）中，CPU 从中断服务程序返回时继续执行这条访问错误指令。

异常按以下步骤被处理：

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

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

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

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

第五步，将 PSR 中的中断使能位 IE 位清零，禁止响应中断。

以上 2-4 步，同时发生。

第六步，处理器首先根据 PSR 中的异常向量号计算得到异常入口地址，然后用该地址获得异常服务程序的第一条指令的地址。将异常向量乘以 4 后加上异常向量基准地址（存在向量基准地址寄存器 VBR 中，当 VBR 不存在时该值恒为零）即得到异常入口地址，以该异常入口地址从存储器中读取一个字，并将该字的[31:1]转载到程序计数器中作为异常服务程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值

的最低位无关）。对于向量中断，异常向量由外部的中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。

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

所有的异常向量存放在超级用户地址空间，并以指令空间索引访问。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，如果配置有 VBR，则允许异常向量表的基准地址被重载。

CK802 支持 256 个字节的向量表包含 64 个异常向量（见图表 7-1）。开始的 30 号向量是用作在处理器内部识别的向量。31 号向量是留给软件的，用作指向系统描述符的指针。其余的 32 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。

图表 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	保留。
4	010	非法指令异常。
5	014	特权违反异常。
6	018	保留。
7	01C	断点异常。
8	020	不可恢复错误异常。
9-15	024-03C	保留。
16－19	040－04C	陷阱指令异常（TRAP ＃0-3）。
20-30	050-078	保留。
31	07C	系统描述符指针。
32－255	080－FC	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 CK802 内部产生的异常。CK802 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；
- 非法指令异常；
- 特权违反异常；
- 断点异常；

- 不可恢复错误异常；
- 向量中断。

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。重启异常也会把 PSR（IE）和 PSR（EE）清零以禁止异常及中断响应。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），可以屏蔽该异常，屏蔽后处理器忽略对数据的对齐检查。MM 位被设置后，若处理器硬件支持非对齐访问，则处理器使用该非对齐地址对存储器进行非对齐访问；若处理器硬件不支持非对齐访问，则处理器将该非对齐地址的低位强行置 0 后对存储器进行对齐访问。EPC 指向试图进行未对齐访问的指令。CK802 的未对齐访问异常只可能发生在数据访问上。

在任何情况下，如果拆分数据访问指令（如 LDM、STM、PUSH、POP、NIE、NIR、IPUSH、IPOP 等）发生地址非对齐，处理器都要响应非对齐访问异常。

7.2.3.访问错误异常（向量偏移 0X8）

如果外部总线接口访问错误返回信号（如 pad_biu_hresp[1:0]=1），就意味着访问发生了异常。当访问 MPU 保护的区域出现访问错误时，也会产生访问错误异常。EPC 指向该次总线请求对应的指令。

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.非法指令异常（向量偏移 0X10）

如果在译码时发现了非法指令或没有实现的指令，CK802 不会执行该指令，而是响应非法指令异常。EPC 指向该非法指令。

7.2.5.特权违反异常（向量偏移 0X14）

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

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

7.2.6.断点异常（向量偏移 0X1C）

CK802 在 Had 控制和状态寄存器 CSR 的 FDB 位为 0 时执行到断点指令 bkpt 会响应

断点异常。EPC 指向触发本次断点异常的 bkpt 指令。

7.2.7.不可恢复错误异常（向量偏移 0X20）

当 PSR（EE）为零时，除复位异常外的其他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）可能由于不可恢复的错误而被重写。

由于软件在 PSR（EE）为零时应该排除了异常事件发生的可能，此不可恢复错误异常一般意味着有系统错误。在异常服务程序中，引起不可恢复错误异常的异常类型是不确定的。

7.2.8.中断异常

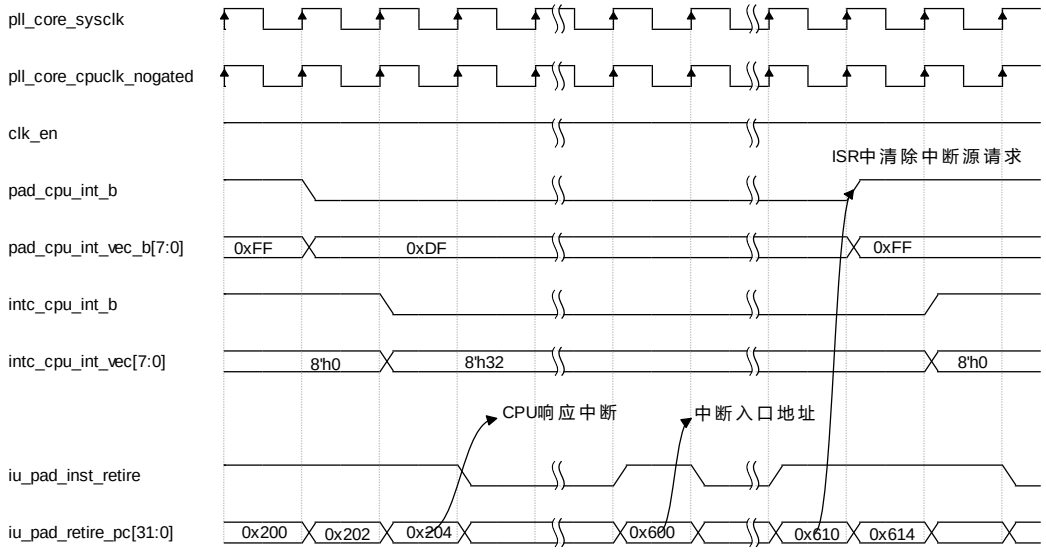
当外部设备需要向处理器请求服务或发送处理器需要的信息时，它可以用中断请求信号和相应的中断向量信号向处理器请求中断异常。

一般中断在指令的边界上被确认。如果 PSR（IC）位设置了，LDM、STM、PUSH、POP、IPUSH、IPOP 等多周期指令可以被中断而不等它们完成，从而缩短中断响应延时。多周期指令 NIE 不可响应中断，NIR 只在指令执行的末尾响应中断，不能被 PSR（IC）位打断。

7.2.8.1. 向量中断（INT）

如果 PSR（IE）被清零，中断输入信号被屏蔽，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。当中断有效时，处理器通过专门的信号提供中断向量号，它可以是 32－255 中的任意一个（不允许使用 0－31）。

7.2.8.2. 中断处理过程



图表 7-2 中断处理过程

在上图中，当中断向量号已经准备好时，拉低普通中断信号线，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，并根据向量信号取得中断向量，进入中断服务程序。在中断服务程序中，应该软件清除外部中断源，即拉高中断有效信号，此信号亦需经此两个时钟依次采样后才会退出中断。

更详细的中断机制及接口信号说明可参考集成手册以及紧耦合 IP 用户手册。

7.2.9.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。

7.3. 异常优先级

如图表 7-3 所示，根据异常的特性和被处理的先后关系，CK802 把优先级分为 5 级。在图表 7-3 中，1 代表最高优先级，5 代表了最低优先级。值得注意的是，在第 4,5 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 CK802 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 7-3 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。

图表 7-3 异常优先级

优先级	异常与它相关的优先级	特征
1	重启异常	处理器中止所有程序运行，初始化系统。
2	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。
3	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。
4	不可恢复错误异常 访问错误	在相关的指令退休后，处理器保存上下文并处理异常。
5	非法指令 特权异常 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。

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

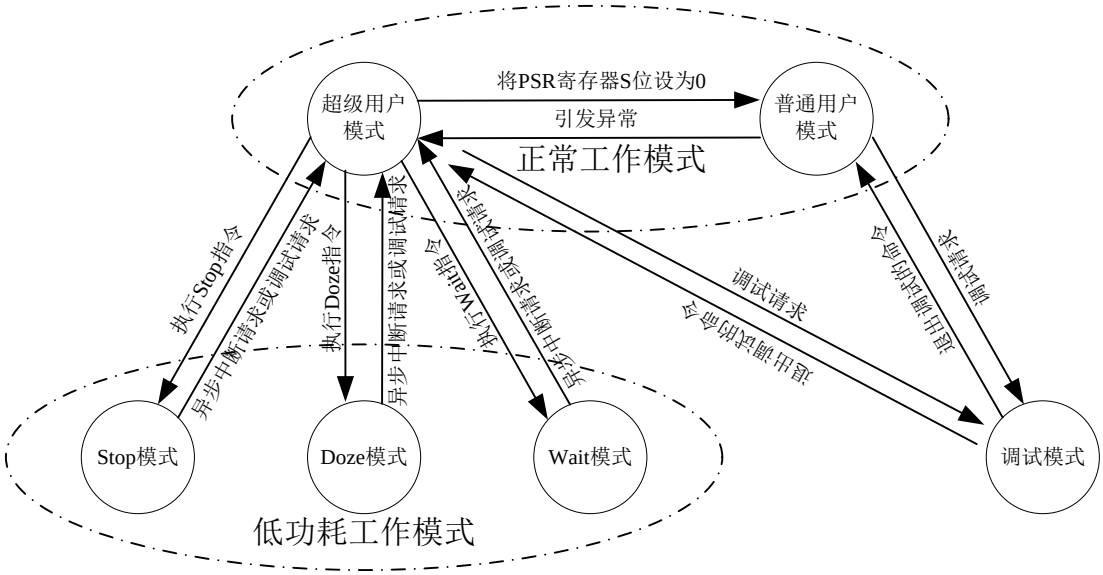
根据正在处理的异常类型，处理器通过执行 rte 指令从异常服务程序中返回。rte 指令利用保存在 EPSR 和 EPC 影子寄存器中的上下文从异常服务程序中返回。

8. 工作模式转换

CK802 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。

8.1. CK802 工作模式及其转换

如图表 10-1 所示，CK802 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 had_pad_jdb_pm[1:0]信号得到。



图表 8-1 CPU 的各种工作状态示意图

8.1.1.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。

8.1.2.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_sysio_intraw_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 sysio_pad_lpmd_b[1:0]来得到。

8.1.2.1. 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。

8.1.2.2. 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

8.1.2.3. 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。

8.1.3.调试模式

8.1.3.1. 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 5 种：

- 当 CK802 HCR 的 ADR 位有效时，处理器直接进入调试模式。
- 当 CK802 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 CK802 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式。
- 当 CK802 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 CK802 存储器断点调试模式下，当 BKPTA 或 BKPTB 被触发（即 MBCA 或 MBCB 的值为 0 时），或者 BKPTC—BKPTI 中有一个被触发，如果当前执行的指令符合断点要求，处理器进入调试模式。

当处理器处于低功耗模式时，通过设置 HCR 中的 ADR 以及 DR 的调试请求，可以使得处理器退出低功耗并进入调试模式。

8.1.3.2. 退出调试模式

如果 CK802 HACR 的 GO、EX 位被置为 1，同时 R/W 为 0（写操作），RS 选择的是 WBBR、PSR、PC、IR、CSR 或者 Bypass 寄存器，则执行指令时 CPU 退出调试模式，进入正常工作模式。

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

附录 A MPU 设置示例

```
/*****************************************************************************
* Function: An example of set CK802 MPU.
*           Enable/disable CK802 memory space region.
* Memory space: 0x28000000 ~ 0x29000000(16MBytes).
*
* Id  |        Memory Space          | Write  | Read  | Executable  | Security  |
* 0   | 0x00000000 ~ 0xFFFFFFFF      | Yes    | Yes   | NO          | NO        |
* 1   | 0x28000000 ~ 0x29000000      | Yes    | Yes   | Yes         | Yes       |
* 2   | 0x28000000 ~ 0x28100000      | No     | Yes   | Yes         | NO        |
* 3   | 0x28F00000 ~ 0x29000000      | Yes    | Yes   | No          | Yes       |
*
* Copyright (c) 2007 C-SKY Microsystems Co.,Ltd. All rights reserved.
*****************************************************************************/

    /* Enable/disable every memory area. */
    /* Set the access authorization.      */
    /* Set the executable or not.         */
    /* Set the security or not.           */
    movih     r10,0xa00
    ori       r10,r10,0xef06
    mtcr      r10,cr<19,0>

    /* The first area (0x00000000 ~ 0xFFFFFFFF) */
    movi      r10,0
    mtcr      r10,cr<21,0>
    movi      r10,0x3f              /* 4G space, Base address: 0x00000000 */
    mtcr      r10,cr<20,0>

    /* The second area (0x28000000 ~ 0x29000000) */
    movi      r10,1
    mtcr      r10,cr<21,0>

    movih     r10,0x2800
    ori       r10,r10,0x2f      /* 16M Space, Base address: 0x28000000 */
    mtcr      r10,cr<20,0>

    /* The third area (0x28000000 ~ 0x28100000) */
    movi      r10,2
    mtcr      r10,cr21
```

```
movih     r10,0x2800
ori       r10,r10,0x2f        /* 1M Space, Base address: 0x28000000 */
mtcr      r10,cr<20,0>

/* The fourth area (0x28F00000 ~ 0x29000000) */
movi      r10,3
mtcr      r10,cr21

movih     r10,0x2800
ori       r10,r10,0x27        /* 1M Space, Base address: 0x28F00000 */
mtcr      r10,cr<20,0>

/* Enable MPU */
mfcr      r7, cr<18,0>
bseti     r7, 0
bclri     r7, 1
mtcr      r7, cr<18,0>
```

附录 B 指令术语表

以下是每条 CK802 实现的 CSKY 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”），系统会自动将汇编为最优化的指令。

其中，指令中文名称中带#的为伪指令。

ADDC——无符号带进位加法指令

统一化指令

语法	操作	编译结果

addc rz, rx	RZ ← RZ + RX + C，
	C←进位
		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;

addc rz, rx, ry	RZ ← RX + RY + C，
	C←进位
		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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 0	RZ	RX	0 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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 0 0 1 0	RZ

ADDI——无符号立即数加法指令

统一化指令

语法	操作	编译结果

addi rz, oimm12	RZ ← RZ + zero_extend(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;

addi rz, rx, oimm12	RZ ← RX + zero_extend(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。

异常： 无

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

指令格式:

15	14 11	10 8	7 0
0	0 1 0 0	RZ	IMM8

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

指令格式:

15	14 10	9 8	7 5	4 2	1 0
0	1 0 1 1	RX	RZ	IMM3	1 0

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

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

指令格式：

31	30 26	25 21	20 16	15 12	11 0
1	1 1 0 0 1	RZ	RX	0 0 0 0	IMM12

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

ADDI(SP)——无符号（堆栈指针）立即数加法指令

统一化指令

语法	操作	编译结果
addi rz, sp, imm	RZ ← SP + zero_extend(IMM)	仅存在 16 位指令。 addi rz, sp, imm
addi sp, sp, imm	SP ← SP+ zero_extend(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

指令格式：

15 14 11 10 8 7 0

0	0 0 1 1	RZ	IMM8

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

指令格式：

15	14 13 12 11	10	9 8	7 6 5	4 3 2 1 0
0	0 0 1 0	1	IMM2	0 0 0	IMM5

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

ADDU——无符号加法指令

统一化指令

语法	操作	编译结果

addu rz, rx	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<16) and (x<16), then
		addu16 rz, rx;
		else
		addu32 rz, rz, rx;

addu rz, rx, ry	RZ ← RX + RY	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

指令格式：

1514 10 9 6 5 2 1 0

0	1 1 0 0 0	RZ	RX	0 0

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

指令格式：

15 14 11 10 8 7 5 4 2 1 0

0	1 0 1 1	RX	RZ	RY	0 0

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

指令格式：

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 0 0 0 1	RZ

AND——按位与指令

统一化指令

语法	操作	编译结果

and rz, rx	RZ ← RZ and RX	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;

and rz, rx, ry	RZ ← RX and RY	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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 0	RZ	RX	0 0

32位指令

操作： RZ ← RX and RY

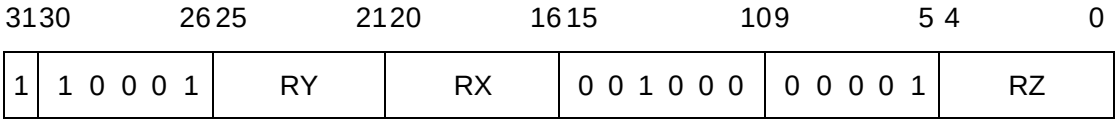
语法： and32　rz, rx, ry

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

指令格式：



ANDI——立即数按位与指令

统一化指令

语法	操作	编译结果
andi rz, rx, imm12	RZ ← RX and zero_extend(IMM12)	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

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 1 0 0 1		RZ		RX	0 0 1 0			IMM12	

ANDN——按位非与指令

统一化指令

语法	操作	编译结果

andn rz, rx	RZ ← RZ and (!RX)	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;

andn rz, rx, ry	RZ ← RX and (!RY)	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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 0	RZ	RX	0 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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 1 0 0 0	0 0 0 1 0	RZ

ANDNI——立即数按位非与指令

统一化指令

语法	操作	编译结果
andni rz, rx, imm12	RZ ← RX and !(zero_extend(IMM12))	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

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 1 0 0 1		RZ		RX		0 0 1 1		IMM12	

ASR——算术右移指令

统一化指令

语法	操作	编译结果

asr rz, rx	RZ ← RZ >>> RX[5:0]	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;

asr rz, rx, ry	RZ ← RX >>> RY[5:0]	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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 0	RZ	RX	1 0

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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 1 0 0	RZ

ASRC——立即数算术右移至 C 位指令

统一化指令

语法	操作	编译结果
asrc rz, rx, oimm5	RZ ← RX >>> OIMM5, C ← RX[OIMM5 - 1]	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 的符号位决定。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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 1	0 0 1 0 0	RZ

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

ASRI——立即数算术右移指令

统一化指令

语法	操作	编译结果

asri　rz, rx, imm5	RZ ← RX >>> IMM5	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

指令格式：

15 14　　11 10　8 7　5 4　0

0	1 0 1 0	RX	RZ	IMM5

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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 1 0 0	RZ

BCLRI——立即数位清零指令

统一化指令

语法	操作	编译结果

bclri rz, imm5	RZ ← RZ[IMM5]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;

bclri rz, rx, imm5	RZ ← RX[IMM5]清零	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

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RZ	1 0 0	IMM5

32位指令

操作： RZ ← RX[IMM5]清零

语法： bclri32　rz, rx, imm5

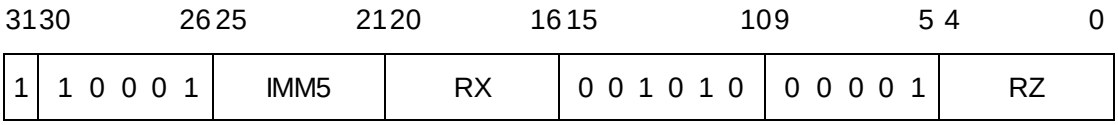
说明： 将 RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：



BF——C 为 0 分支指令

统一化指令

语法	操作	编译结果

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

指令格式：

15	14	10	9	0
0	0 0 0 1 1		Offset	

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 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。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

指令格式:

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		0 0 0 1 0		0 0 0 0 0		Offset	

BGENI——立即数位产生指令#

统一化指令

语法	操作	编译结果
bgeni rz, imm5	RZ ←（2）^{IMM5}	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

指令格式：

如果 IMM5 小于 16：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 0 0 0	RZ	（2）^{IMM5}

如果 IMM5 大于等于 16：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 0 0 1	RZ	（2）^{IMM5}

BKPT——断点指令

统一化指令

语法	操作	编译结果
bkpt	引起一个断点异常或者进入调试模式	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

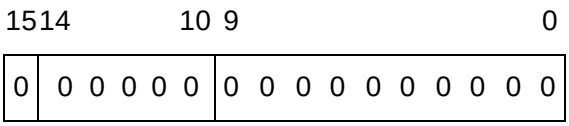
语法： bkpt16

说明： 断点指令

影响标志位： 无影响

异常： 断点异常

指令格式：



BMASKI——立即数位屏蔽产生指令

统一化指令

语法	操作	编译结果
bmaski rz, oimm5	RZ ← (2)^{OIMM5} - 1	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)^{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。

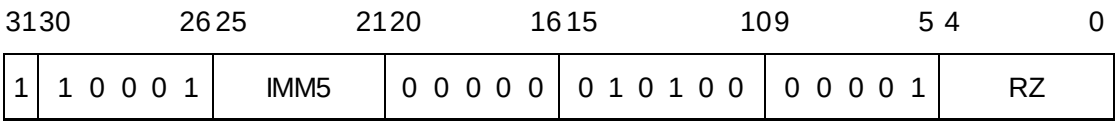
注意，OIMM5 为 1-16 时由 movi 指令执行；二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： 无影响

限制： 立即数的范围为 0，17-32；

异常： 无

指令格式：



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

BMCLR——BCTM 位清零指令

统一化指令

语法	操作	编译结果
bmclr	清除状态寄存器的 BM 位。 PSR(BM) ← 0	仅存在 32 位指令。 bmclr32

说明： PSR 的 BM 位被清零。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的CK802处理器。

32位指令

操作： 清除状态寄存器的 BM 位

PSR(BM) ← 0

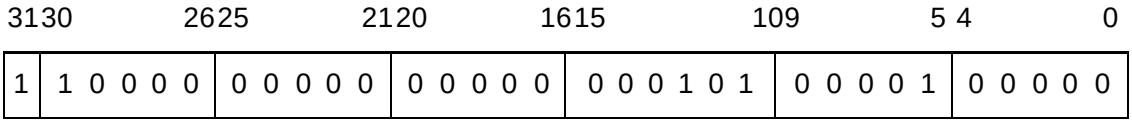
语法： bmclr32

说明： PSR 的 BM 位被清零。

影响标志位： 无影响

异常： 无

指令格式：



BPOP.H——二进制转译半字压栈指令

统一化指令

语法	操作	编译结果

bpop.h　rz	更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载半字到寄存器 RZ 中；
	if (BSP - 2 < FP')
	　R15 ← next PC
	　PC ← SVBR - 12
	else
	　BSP ← BSP - 2;
	　RZ ← zero_extend(MEM[BSP]);
		仅存在 16 位指令
		　bpop.h　rz;

说明： 将二进制转译堆栈指针寄存器（BSP）减 2 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 2 的值小于 FP'，则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 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的半字经过零扩展到 32 位后，加载到寄存器 RZ 中。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、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： 更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载半字到寄存器 RZ 中；

```
if (BSP - 2 < FP')
    R15 ← next PC
    PC ← SVBR - 12
else
  BSP ← BSP - 2;
  RZ ← zero_extend(MEM[BSP]);
```

语法： bpop16.h　rz

说明： 将二进制转译堆栈指针寄存器（BSP）减 2 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 2 的值小于 FP'，则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 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的半字经过零扩展到 32 位后，加载到寄存器 RZ 中。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 r0 – r7。

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

指令格式：

15	14	10	9	8	7	6	5	4	2	1	0
0	0 0 1 0 1		0	0	1	0	1	RZ		0	0

BPOP.W——二进制转译字压栈指令

统一化指令

语法	操作	编译结果

bpop.w　rz	更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载字到寄存器 RZ 中；
	if (BSP - 4 < FP')
	　R15 ← next PC
	　PC ← SVBR - 12
	else
	　BSP ← BSP - 4;
	　RZ ← MEM[BSP];
		仅存在 16 位指令
		　bpop.w　rz;

说明： 将二进制转译堆栈指针寄存器（BSP）减 4 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 4 的值小于 FP'，则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 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的字加载到寄存器 RZ 中。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、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： 更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载字到寄存器 RZ 中；

```
if (BSP - 4 < FP')
    R15 ← next PC
    PC ← SVBR - 12
else
   BSP ← BSP - 4;
   RZ ← MEM[BSP];
```

语法： bpop16.w　rz

说明： 将二进制转译堆栈指针寄存器（BSP）减 4 的值与二进制转译帧指

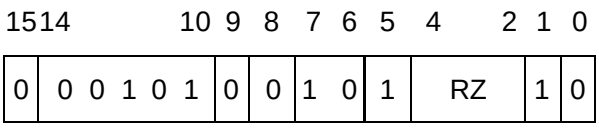
针寄存器（FP’）进行比较。如果 BSP 减 4 的值小于 FP’，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的字加载到寄存器 RZ 中。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

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

指令格式：



BPUSH.H——二进制转译半字压栈指令

统一化指令

语法	操作	编译结果

bpush.h rz	将寄存器 RZ 中的半字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；
	if (BSP + 2 > TOP)
	R15 ← next PC
	PC ← SVBR - 12
	else
	MEM[BSP] ← RZ[15:0];
	BSP ← BSP + 2;
		仅存在 16 位指令
		bpush.h rz;

说明： 将二进制转译堆栈指针寄存器（BSP）加 2 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 2 的值大于 TOP，则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 SVBR-12 处执行；否则，将寄存器 RZ 中的低半字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。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、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 RZ 中的半字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；

if (BSP + 2 > TOP)

R15 ← next PC

PC ← SVBR - 12

else

MEM[BSP] ← RZ[15:0];

BSP ← BSP + 2;

语法： bpush16.h rz

说明： 将二进制转译堆栈指针寄存器（BSP）加 2 的值与二进制转译栈顶

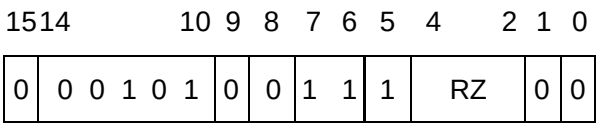
寄存器（TOP）进行比较。如果 BSP 加 2 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的低半字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

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

指令格式：



BPUSH.W——二进制转译字压栈指令

统一化指令

语法	操作	编译结果

bpush.w rz	将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；
	if (BSP + 4 > TOP)
	R15 ← next PC
	PC ← SVBR - 12
	else
	MEM[BSP] ← RZ[31:0];
	BSP ← BSP + 4;
		仅存在 16 位指令
		bpush.w rz;

说明： 将二进制转译堆栈指针寄存器（BSP）加 4 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 4 的值大于 TOP，则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 SVBR-12 处执行；否则，将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。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、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 RZ 中的字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；

if (BSP + 4 > TOP)

R15 ← next PC

PC ← SVBR - 12

else

MEM[BSP] ← RZ[31:0];

BSP ← BSP + 4;

语法： bpush16.w rz

说明： 将二进制转译堆栈指针寄存器（BSP）加 4 的值与二进制转译栈顶

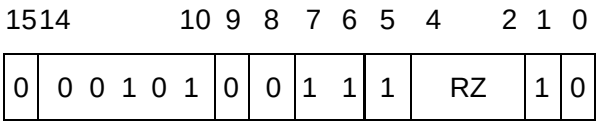
寄存器（TOP）进行比较。如果 BSP 加 4 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

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

指令格式：



BMSET——BCTM 位置位指令

统一化指令

语法	操作	编译结果
bmset	设置状态寄存器的 BM 位。 PSR(BM) ← 1	仅存在 32 位指令。 bmset32

说明： PSR 的 BM 位被置位。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的CK802处理器。

32位指令

操作： 设置状态寄存器的 BM 位

PSR(BM) ← 1

语法： bmset32

说明： PSR 的 BM 位被置位。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 1 0 0	0 0 0 0 1	0 0 0 0 0

BR——无条件跳转指令

统一化指令

语法	操作	编译结果

br label	PC ← PC + sign_extend(offset << 1)	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

指令格式：

15	14 10	9 0
0	0 0 0 0 1	Offset

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 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。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

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	0 0 0 0 0	0 0 0 0 0	Offset

BSETI——立即数位置位指令

统一化指令

语法	操作	编译结果

bseti rz, imm5	RZ ← RZ[IMM5]置位	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；

bseti rz, rx, imm5	RZ ← RX[IMM5]置位	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

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RZ	1 0 1	IMM5

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 0 1 0 1 0		0 0 0 1 0		RZ	

BSR——跳转到子程序指令

统一化指令

语法	操作	编译结果
bsr label	链接并跳转到子程序： R15 ← next PC PC ← PC + sign_extend(offset << 1)	仅存在于 32 位指令 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

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：

31	30 26	25 0
1	1 1 0 0 0	Offset

BT——C 为 1 分支指令

统一化指令

语法	操作	编译结果

bt label	if(C == 1)
	PC ← PC + sign_extend(offset << 1);
	else
	PC ← next PC;
		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 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。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

指令格式：

15	14 10	9 0
0	0 0 0 1 0	Offset

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 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。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

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	0 0 0 1 1	0 0 0 0 0	Offset

BTSTI——立即数位测试指令

统一化指令

语法	操作	编译结果
btsti rx, imm5	C ← RX[IMM5]	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

16位指令

操作： C ← RX[IMM5]

语法： Btsti16 rx, imm5

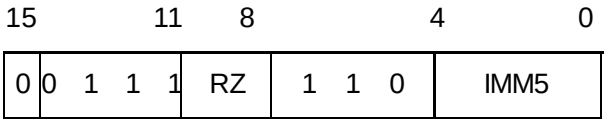
说明： 对由 IMM5 决定的 RX 的位（RX[IMM5]）进行测试，并使条件位 C 的值等于该位的值。

影响标志位： C ← RX[IMM5]

限制： 立即数的范围为 0-31。

异常： 无

指令格式：



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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 0 1 0 1 0		0 0 1 0 0		0 0 0 0 0	

CMPHS——无符号大于等于比较指令

统一化指令

语法	操作	编译结果

cmphs rx, ry	RX与RY作无符号比较。
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令 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。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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 1	RY	RX	0 0

CMPHSI——立即数无符号大于等于比较指令

统一化指令

语法	操作	编译结果

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

指令格式：

15	14 10	8 7 5	4 0
0	0 1 1 1	RX	0 0 0	IMM5

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

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 1 0 0 0	RX	IMM16

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

CMPLT——有符号小于比较指令

统一化指令

语法	操作	编译结果

cmplt rx, ry	RX与RY作有符号比较。
	If RX < RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令 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。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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 1	RY	RX	0 1

CMPLTI——立即数有符号小于比较指令

统一化指令

语法	操作	编译结果

cmplti rx, oimm16	RX与立即数作有符号比较。
	If RX < zero_extend(OIMM16),
	C ← 1;
	else
	C ← 0;
		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。cmplti16 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

指令格式：

15	14	10	8	7	5	4	0
0	0 1 1 1	RX		0 0 1		IMM5	

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。cmplti32 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

指令格式：

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		1 1 0 0 1		RX		IMM16	

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

CMPNE——不等比较指令

统一化指令

语法	操作	编译结果

cmpne rx, ry	RX与RY作比较。
	If RX != RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令
		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

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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 1	RY	RX	1 0

CMPNEI——立即数不等比较指令

统一化指令

语法	操作	编译结果

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

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RX	0 1 0	IMM5

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

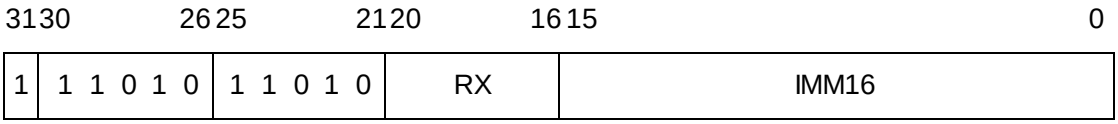
说明： 将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

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

指令格式：



DECF——C 为 0 立即数减法指令

统一化指令

语法	操作	编译结果

decf　rz, rx, imm5	if C==0, then
	　RZ ← RX - zero_extend(IMM5);
	else
	　RZ ← RZ;
		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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RZ		RX		0 0 0 0 1 1		0 0 1 0 0		IMM5	

DECT——C 为 1 立即数减法指令

统一化指令

语法	操作	编译结果

dect rz, rx, imm5	if C==1, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 1 0 0 0	IMM5

DOZE——进入低功耗睡眠模式指令

统一化指令

语法	操作	编译结果
doze	进入低功耗睡眠模式	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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	10000	00000	00000	010100	00001	00000

FF0——快速找 0 指令

统一化指令

语法	操作	编译结果
ff0　rz, rx	RZ ← find_first_0(RX);	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：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 1 1	0 0 0 0 1	RZ

FF1——快速找 1 指令

统一化指令

语法	操作	编译结果
ff1　rz, rx	RZ ← find_first_1(RX);	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：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 1 1	0 0 0 1 0	RZ

INCF——C 为 0 立即数加法指令

统一化指令

语法	操作	编译结果

incf　rz, rx, imm5	if C==0, then
	　RZ ← RX + zero_extend(IMM5);
	else
	　RZ ← RZ;
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 0 1	IMM5

INCT——C 为 1 立即数加法指令

统一化指令

语法	操作	编译结果

inct rz, rx, imm5	if C==1, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 1 0	IMM5

IPUSH——中断压栈指令

统一化指令

语法	操作	编译结果
ipush	将中断的通用寄存器现场{R0~R3, R12, R13 }存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端； MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;	仅存在 16 位指令 ipush16

说明： 将中断的通用寄存器现场{ R0~R3, R12, R13 }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

16位指令

操作： 将中断的通用寄存器现场{R0~R3, R12, R13 }存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端；
MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0};
SP←SP-24;

语法： IPUSH16

属性 无

说明： 将中断的通用寄存器现场{ R0~R3, R12, R13 }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

指令格式：

15	14 10	9	8	7 5	4 0
0	0 0 1 0 1	0	0	0 1 1	0 0 0 1 0

IPOP——中断出栈指令

统一化指令

语法	操作	编译结果
ipop	从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端； {R0~R3,R12,R13} ← MEM[SP]~MEM[SP+20]; SP←SP+24;	仅存在 16 位指令 ipop16

说明： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

16位指令

操作： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端；
{R0~R3,R12,R13}← MEM[SP]~MEM[SP+20];
SP←SP+24;

语法： IPOP16

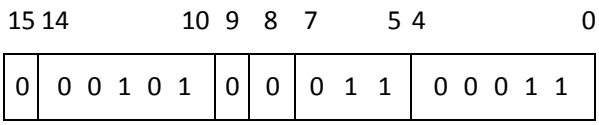
属性 无

说明： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

指令格式：



IXH——索引半字指令

统一化指令

语法	操作	编译结果
ixh rz, rx, ry	RZ ← RX + (RY << 1)	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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 0 0 1 0	0 0 0 0 1	RZ

IXW——索引字指令

统一化指令

语法	操作	编译结果
ixw rz, rx, ry	RZ ← RX + (RY << 2)	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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 0 0 1 0	0 0 0 1 0	RZ

JMP——寄存器跳转指令

统一化指令

语法	操作	编译结果
jmp　rx	跳转到寄存器指定的位置 PC ← RX & 0xfffffffe	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	0 0 0 0	RX	0 0

JMPIX——寄存器索引跳转指令

统一化指令

语法	操作	编译结果
jmpix rx, imm	跳转到寄存器索引指定的位置 PC ← SVBR + (RX & 0xff) * IMM	仅存在 16 位指令。 jmpix16 rx, imm;

说明： 程序跳转到 SVBR + RX[7:0] * IMM 的位置，IMM ∈ {16, 24, 32, 40}。RX 的高 24 位被忽略。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的CK802处理器。

16位指令

操作： 跳转到寄存器索引指定的位置

PC ← SVBR + (RX & 0xff) * IMM

语法： jmpix16 rx, imm

说明： 程序跳转到 SVBR + RX[7:0] * IMM 的位置，IMM ∈ {16, 24, 32, 40}。RX 的高 24 位被忽略。

影响标志位： 无影响

异常： 无

指令格式：

15 14 11 10 8 7 2 0

0	0 1 1 1	RX	1 1 1 0 0 0	IMM2

IMM2 域——指定立即数的值。

注意：二进制编码的 IMM2 值与此跳转指令中 IMM 值的对应关系如下：

2’b00——乘 16

2’b01——乘 24

2’b10——乘 32

2’b11——乘 40

JSR——寄存器跳转到子程序指令

统一化指令

语法	操作	编译结果
jsr　rx	链接并跳转到寄存器指定的子程序位置 R15 ← PC + 4， PC ← RX & 0xfffffffe	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	0 0 0 0	RX	0 1

LD.B——无符号扩展字节加载指令

统一化指令

语法	操作	编译结果

ld.b rz,(rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(offset)])	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。

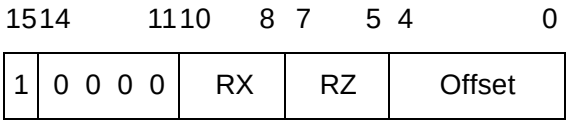
注意，偏移量 DISP 即二进制操作数 Offset。

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

指令格式：



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)

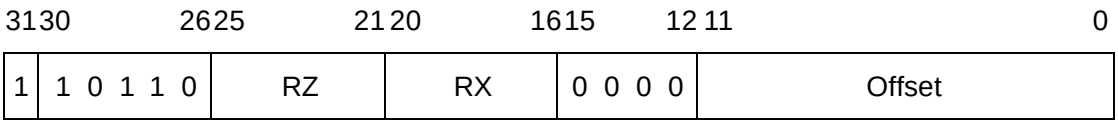
说明： 从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器RX加上12位相对偏移量无符号扩展到32位后的值得到。LD32.B指令可以寻址+4KB 地址空间。

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

指令格式：



LD.BS——有符号扩展字节加载指令

统一化指令

语法	操作	编译结果
ld.bs rz, (rx, disp)	RZ ← sign_extend(MEM[RX + zero_extend(offset)])	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)

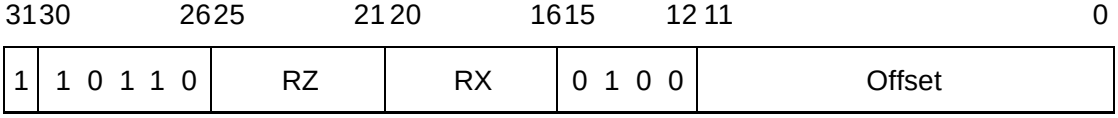
说明： 从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.BS 指令可以寻址+4KB 地址空间。

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

指令格式：



LD.H——无符号扩展半字加载指令

统一化指令

语法	操作	编译结果

ld.h rz, (rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])	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

指令格式：

1514 1110 8 7 5 4 0

1	0 0 0 1	RX	RZ	Offset

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

指令格式：

31 30 26 25 21 20 16 15 12 11 0

1	1 0 1 1 0	RZ	RX	0 0 0 1	Offset

LD.HS——有符号扩展半字加载指令

统一化指令

语法	操作	编译结果
ld.hs rz, (rx, disp)	RZ ← sign_extend(MEM[RX + zero_extend(offset << 1)])	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

指令格式：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	0	RZ					RX					0	1	0	1	Offset											

LD.W——字加载指令

统一化指令

语法	操作	编译结果

ld.w rz, (rx, disp)	RZ ← MEM[RX + zero_extend(offset << 2)]	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

指令格式：

ld16.w rz, (rx, disp)

15	14 11	10 8	7 5	4 0
1	0 0 1 0	RX	RZ	IMM5

ld16.w rz, (sp, disp)

15	14 11	10 8	7 5	4 0
1	0 0 1 1	IMM3	RZ	IMM5

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

指令格式：

31	30 26	25 21	20 16	15 12	11 0
1	1 0 1 1 0	RZ	RX	0 0 1 0	Offset

LDM——连续多字加载指令

统一化指令

语法	操作	编译结果

ldm ry-rz, (rx)	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;
	}
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 1 0 0	RY	RX	0 0 0 1 1 1	0 0 0 0 1	IMM5

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

LDQ——连续四字加载指令#

统一化指令

语法	操作	编译结果

ldq　r4-r7, (rx)	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;
	}
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 1 0 0	0 0 1 0 0	RX	0 0 0 1 1 1	0 0 0 0 1	0 0 0 1 1

LRW——存储器读入指令

统一化指令

语法	操作	编译结果

lrw rz, label
lrw rz, imm32
	从存储器加载字到寄存器
	RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])
		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 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 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，若 IMM1 为 1，相对偏移量 Offset 等于二进制编码{0，{IMM2，IMM5}}。

若 IMM1 为 0， 相对偏移量 Offset 等于二进制编码{1，{！{IMM2，IMM5}}}。

IMM1,IMM2,IMM5,RZ 同时为零则该指令为 BKPT 指令，并非 LRW16。

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

指令格式：

15	14 13	12	11 10	9 8	7 5	4 0
0	00	IMM1	0 0	IMM2	RZ	IMM5

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 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 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

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 1 0 0	RZ	Offset

LSL——逻辑左移指令

统一化指令

语法	操作	编译结果

lsl　rz, rx	RZ ← RZ << RX[5:0]	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;

lsl　rz, rx, ry	RZ ← RX << RY[5:0]	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

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 1 0 0		RZ		RX		0 0	

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 1 0 0 0 0		0 0 0 0 1		RZ	

LSLC——立即数逻辑左移至 C 位指令

统一化指令

语法	操作	编译结果
lslc rz, rx, oimm5	RZ ← RX << OIMM5, C ← RX[32 – OIMM5]	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 0 0 0 1		RZ	

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

LSLI——立即数逻辑左移指令

统一化指令

语法	操作	编译结果

lsli rz, rx, imm5	RZ ← RX << IMM5	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

指令格式：

15 14 11 10 8 7 5 4 0

0	1 0 0 0	RX	RZ	IMM5

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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 0 0 1	RZ

LSR——逻辑右移指令

统一化指令

语法	操作	编译结果

lsr　rz, rx	RZ ← RZ >> RX[5:0]	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;

lsr　rz, rx, ry	RZ ← RX >> RY[5:0]	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

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 1 0 0		RZ		RX		0 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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX	0 1 0 0 0 0			0 0 0 1 0		RZ	

LSRC——立即数逻辑右移至 C 位指令

统一化指令

语法	操作	编译结果
lsrc rz, rx, oimm5	RZ ← RX >> OIMM5, C ← RX[OIMM5 - 1]	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 0 0 1 0		RZ	

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

LSRI——立即数逻辑右移指令

统一化指令

语法	操作	编译结果

lsri rz, rx, imm5	RZ ← RX >> IMM5	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

指令格式：

15 14 11 10 8 7 5 4 0

0	1 0 0 1	RX	RZ	IMM5

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 0		0 0 0 1 0		RZ	

MFCR——控制寄存器读传送指令

统一化指令

语法	操作	编译结果
mfcr rz, cr<x, sel>	将控制寄存器的内容传送到通用寄存器中 RZ ← CR<X, sel>	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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 0	sel	CRX	0 1 1 0 0 0	0 0 0 0 1	RZ

MOV——数据传送指令#

统一化指令

语法	操作	编译结果
mov rz, rx	RZ ← RX	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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RX	1 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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 0 0 1 0	0 0 0 0 1	RZ

MOVF——C 为 0 数据传送指令#

统一化指令

语法	操作	编译结果

movf　rz, rx	if C==0, then
	　RZ ← RX;
	else
	　RZ ← RZ;
		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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 0 1	0 0 0 0 0

MOVI——立即数数据传送指令

统一化指令

语法	操作	编译结果

movi16 rz, imm16	RZ ← zero_extend(IMM16);	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

15	14 11	10 8	7 0
0	0 1 1 0	RZ	IMM8

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

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 0 0 0	RZ	IMM16

MOVIH——立即数高位数据传送指令

统一化指令

语法	操作	编译结果
movih rz, imm16	RZ ← zero_extend(IMM16) << 16	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, 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, 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

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 0 0 1	RZ	IMM16

MOVT——C 为 1 数据传送指令#

统一化指令

语法	操作	编译结果

movt　rz, rx	if C==1, then
	　RZ ← RX;
	else
	　RZ ← RZ;
		仅存在 32 位指令。
		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 inct　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==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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 1 0	0 0 0 0 0

MTCR——控制寄存器写传送指令

统一化指令

语法	操作	编译结果
mtcr rx, cr<z, sel>	将通用寄存器的内容传送到控制寄存器中 CR<Z, sel> ← RX	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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 0	sel	RX	0 1 1 0 0 1	0 0 0 0 1	CRZ

MULT——乘法指令

统一化指令

语法	操作	编译结果

mult rz, rx	两个数相乘，结果的低 32 位放入通用寄存器中
	RZ ← RX × RZ
		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;

mult rz, rx, ry	两个数相乘，结果的低 32 位放入通用寄存器中
	RZ ← RX × RY
		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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 1 1 1	RZ	RX	0 0

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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	1 0 0 0 0 1	0 0 0 0 1	RZ

MVC——C 位传送指令

统一化指令

语法	操作	编译结果
mvc rz	RZ ← C	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

指令格式：

31	30 ~ 26	25 ~ 21	20 ~ 16	15 ~ 10	9 ~ 5	4 ~ 0
1	1 0 0 0 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 1	0 1 0 0 0	RZ

MVCV——C 位取反传送指令

统一化指令

语法	操作	编译结果
mvcv　rz	RZ ← (!C)	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 0 1	RZ	0 0 0 0	1 1

NIE——中断嵌套使能指令

统一化指令

语法	操作	编译结果
nie	将中断的控制寄存器现场{EPSR, EPC}存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE； MEM[SP-4] ←EPC; MEM[SP-8] ←EPSR; SP←SP-8; PSR({EE,IE}) ← 1	仅存在 16 位指令。 nie16

说明： 将中断的控制寄存器现场{EPSR, EPC }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PSR.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

16位指令

操作： 将中断的控制寄存器现场{EPSR, EPC}存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE；

MEM[SP-4] ←EPC;

MEM[SP-8] ←EPSR;

SP←SP-8;

PSR({EE,IE}) ← 1

语法： NIE16

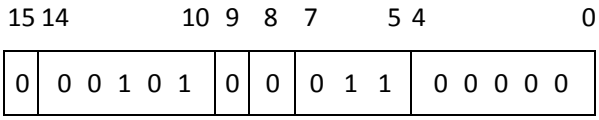
属性 特权指令

说明： 将中断的控制寄存器现场{EPSR, EPC }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PSR.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

指令格式：


15 14 10 9 8 7 5 4 0
0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0

NIR——中断嵌套返回指令

统一化指令

语法	操作	编译结果
nir	从堆栈存储器中载入中断的控制寄存器现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回 EPSR←MEM[SP] EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC	仅存在 16 位指令。 nir16

说明： 从堆栈存储器中载入中断的现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

16位指令

操作： 从堆栈存储器中载入中断的控制寄存器现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回

EPSR←MEM[SP]

EPC←MEM[SP+4];

SP←SP+8;

PSR←EPSR;

PC←EPC

语法： NIR16

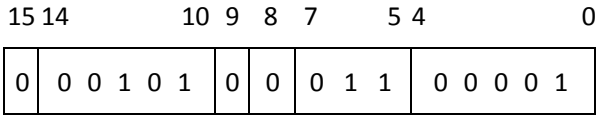
属性 特权指令

说明： 从堆栈存储器中载入中断的现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

指令格式：



NOR——按位或非指令

统一化指令

语法	操作	编译结果

nor rz, rx	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<16) and (z<16), then
		nor16 rz, rx;
		else
		nor32 rz, rz, rx;

nor rz, rx, ry	RZ ← !(RX | RY)	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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RX	1 0

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 1 0 0 1		0 0 1 0 0		RZ	

NOT——按位非指令#

统一化指令

语法	操作	编译结果

not rz	RZ ← !(RZ)	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;

not rz, rx	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), 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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RZ	1 0

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RX		RX		0 0 1 0 0 1		0 0 1 0 0		RZ	

OR——按位或指令

统一化指令

语法	操作	编译结果

or rz, rx	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<16) and (z<16), then
		or16 rz, rx ;
		else
		or32 rz, rz, rx;

or rz, rx, ry	RZ ← RX | RY	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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 1	RZ	RX	0 0

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

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 1 0 0 1	0 0 0 0 1	RZ

ORI——立即数按位或指令

统一化指令

语法	操作	编译结果
ori　rz, rx, imm16	RZ ← RX | zero_extend(IMM16)	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)

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。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	0
1	1 1 0 1 1		RZ		RX		IMM16	

POP——出栈指令

统一化指令

语法	操作	编译结果

pop reglist	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;
		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

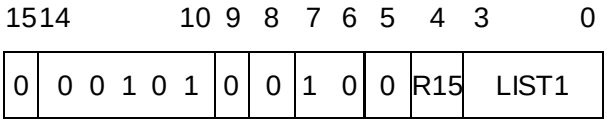
说明： 从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。

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

指令格式：



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

PSRCLR——PSR 位清零指令

统一化指令

语法	操作	编译结果
psrclr　ee, ie, fe, af 或者操作数也可以为ee、ie、fe、af的任意组合。	清除状态寄存器的某一位或几位。 PSR({EE，IE，FE，AF}) ← 0	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

指令格式:

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		IMM5		0 0 0 0 0		0 1 1 1 0 0		0 0 0 0 1		0 0 0 0 0	

PSRSET——PSR 位置位指令

统一化指令

语法	操作	编译结果
psrset　ee, ie, fe, af 或者操作数也可以为ee、ie、fe、af的任意组合。	设置状态寄存器的某几位 PSR({EE，IE，FE，AF}) ← 1	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

指令格式：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	IMM5					0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0

PUSH——压栈指令

统一化指令

语法	操作	编译结果

push reglist	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){
	addr ← addr - 4; MEM[addr] ← Rsrc;
	src ← next {reglist};
	}
	sp ← addr;
		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

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

指令格式：

15	14 10	9	8	7	6	5	4	3 0
0	0 0 1 0 1	0	0	1	1	0	R15	LIST1

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

REVB——字节倒序指令

统一化指令

语法	操作	编译结果
revb rz, rx	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];	仅存在 16 位指令 revb16 rz, rxjin

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	RZ	RX	1 0

REVH——半字字节倒序指令

统一化指令

语法	操作	编译结果
revh rz, rx	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	RZ	RX	1 1

ROTL——循环左移指令

统一化指令

语法	操作	编译结果

rotl rz, rx	RZ ← RZ <<<< RX[5:0]	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;

rotl rz, rx, ry	RZ ← RX <<<< RY[5:0]	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

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 1 0 0		RZ		RX		1	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 1 0 0 0 0		0 1 0 0 0		RZ	

ROTLI——立即数循环左移指令

统一化指令

语法	操作	编译结果
rotli rz, rx, imm5	RZ ← RX <<<< IMM5	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 0		0 1 0 0 0		RZ	

RSUB——反向减法指令#

统一化指令

语法	操作	编译结果
rsub rz, rx, ry	RZ ← RY - RX	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RX		RY		0 0 0 0 0 0		0 0 1 0 0		RZ	

RTS——子程序返回指令#

统一化指令

语法	操作	编译结果
rts	程序跳转到链接寄存器指定的位置 PC ← R15 & 0xfffffffe	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

指令格式：

15	14 13 12 11 10	9 8 7 6	5 4 3 2	1 0
0	1 1 1 1 0	0 0 0 0	1 1 1 1	0 0

RTE——异常和普通中断返回指令

统一化指令

语法	操作	编译结果
rte	异常和普通中断返回 PC ← EPC，PSR ← EPSR	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

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 0 0 0	0 0 0 0 1	0 0 0 0 0

SEXTB——字节提取并有符号扩展指令#

统一化指令

语法	操作	编译结果
sextb rz, rx	RZ ← sign_extend(RX[7:0]);	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 1	RZ	RX	1 0

SEXTH——半字提取并有符号扩展指令#

统一化指令

语法	操作	编译结果
sexth rz, rx	RZ ← sign_extend(RX[15:0]);	仅存在 16 位指令 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

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

指令格式

15	14 10	9 6	5 2	1 0
0	1 1 1 0 1	RZ	RX	1 1

ST.B——字节存储指令

统一化指令

语法	操作	编译结果

st.b rz, (rx, disp)	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]
		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

指令格式：

15	14	11	10	8	7	5	4	0
1	0 1 0 0		RX		RZ		IMM5	

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

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 0 1 1 1		RZ		RX	0 0 0 0		Offset		

ST.H——半字存储指令

统一化指令

语法	操作	编译结果

st.h rz, (rx, disp)	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]
		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

指令格式：

15 14 11 10 8 7 5 4 0

1	0 1 0 1	RX	RZ	IMM5

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

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 0 1 1 1		RZ		RX	0 0 0 1			Offset	

ST.W——字存储指令

统一化指令

语法	操作	编译结果

st.w rz, (rx, disp)	将寄存器中的字存储到存储器中
	MEM[RX + zero_extend(offset<< 2)] ← RZ[31:0]
		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

指令格式：

st16.w rz, (rx, disp)

15	14 13 12 11	10 9 8	7 6 5	4 3 2 1 0
1	0 1 1 0	RX	RZ	IMM5

st16.w rz, (sp, disp)

15	14 13 12 11	10 9 8	7 6 5	4 3 2 1 0
1	0 1 1 1	IMM3	RZ	IMM5

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

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
1	1 0 1 1 1	RZ	RX	0 0 1 0	Offset

STM——连续多字存储指令

统一化指令

语法	操作	编译结果

stm ry-rz, (rx)	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;
	}
		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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1		RY		RX		0 0 0 1 1 1		0 0 0 0 1		IMM5	

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

STQ——连续四字存储指令#

统一化指令

语法	操作	编译结果
stq r4-r7, (rx)	将寄存器 R4—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; }	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 R4—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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 1		0 0 1 0 0		RX		0 0 0 1 1 1		0 0 0 0 1		0 0 0 1 1	

STOP——进入低功耗暂停模式指令

统一化指令

语法	操作	编译结果
stop	进入低功耗暂停模式	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

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 1 0 0 1 0	0 0 0 0 1	0 0 0 0 0	

SUBC——无符号带借位减法指令

统一化指令

语法	操作	编译结果

subc　rz, rx	RZ ← RZ - RX -（!C），
	C ← 借位
		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;

subc　rz, rx, ry	RZ ← RX - RY -（!C），
	C ← 借位
		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 subc　rz, 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

指令格式：

15 14　　10 9　　6 5　　2 1 0

0	1 1 0 0 0	RZ	RX	1 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

指令格式：

31	30	29	28	27	26	25	21	20	16	15	10	9	5	4	0
1	1	0	0	0	1	RY		RX		0	0	0	0	0	0	0	1	0	0	0	RZ

SUBI——无符号立即数减法指令

统一化指令

语法	操作	编译结果

subi rz, oimm12	RZ ← RZ - zero_extend(OIMM12)	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;

subi rz, rx, oimm12	RZ ← RX - zero_extend(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

指令格式：

1514 1110 8 7 0

0	0 1 0 1	RZ	IMM8

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

指令格式：

1514 10 8 7 5 4 2 1 0

0	1 0 1 1	RX	RZ	IMM3	1 1

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。

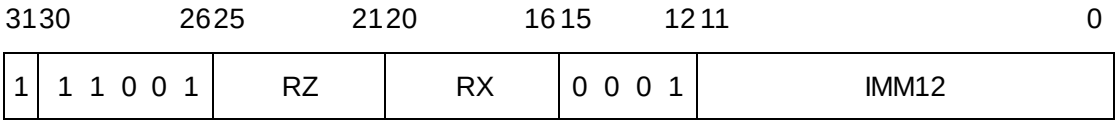
注意：二进制操作数 IMM12 等于 OIMM12 - 1。

影响标志位： 无影响

限制： 立即数的范围为 0x1-0x1000。

异常： 无

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

SUBI(SP)——无符号（堆栈指针）立即数减法指令

统一化指令

语法	操作	编译结果
subi sp, sp, imm	SP ← SP- zero_extend(IMM)	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

指令格式：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	IMM2		0	0	1	IMM5				

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

SUBU——无符号减法指令

统一化指令

语法	操作	编译结果

subu rz, rx
sub rz, rx
	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<16) and (x<16), then
		subu16 rz, rx;
		else
		subu32 rz, rz, rx;

subu rz, rx, ry	RZ ← RX - RY	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

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 0 0 0		RZ		RX		1 0	

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

指令格式：

15	14	11	10	8	7	5	4	2	1	0
0	1 0 1 1		RX		RZ		RY		0 1	

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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 0 0 0 0		0 0 1 0 0		RZ	

SYNC——CPU 同步指令

统一化指令

语法	操作	编译结果
sync	使 CPU 同步	仅存在 32 位指令。 sync32

说明： 当处理器碰到 sync 指令时，指令就会被悬挂起来直到所有外面的操作全都完成，即没有未完成的指令。

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： 当处理器碰到 sync 指令时，指令就会被悬挂起来直到所有外面的操作全都完成，即没有未完成的指令。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 0 0 0 1		0 0 0 0 1		0 0 0 0 0	

TRAP——操作系统陷阱指令

统一化指令

语法	操作	说明
trap 0, trap 1 trap 2, trap 3	引起陷阱异常发生	仅存在 32 位指令。 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

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

指令格式：

trap32 0

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9	8 7 6 5 4	3 2 1 0
1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 0 0	0 0 0 0 1	0 0 0 0 0

trap32 1

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9	8 7 6 5 4	3 2 1 0
1	1 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1 0 0 1	0 0 0 0 1	0 0 0 0 0

trap32 2

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 1 0		0 0 0 0 1		0 0 0 0 0	

trap32 3

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 1 1		0 0 0 0 1		0 0 0 0 0	

TST——零测试指令

统一化指令

语法	操作	编译结果

tst rx, ry	If (RX & RY) != 0, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令。
		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

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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 0	RY	RX	1 0

TSTNBZ——无字节等于零寄存器测试指令

统一化指令

语法	操作	编译结果
tstnbz16　rx	If　((RX[31:24] != 0) 　&(RX[23:16] != 0) 　&(RX[15: 8] != 0) 　&(RX[7 : 0] != 0)), then 　C ← 1; else 　C ← 0;	仅存在 16 位指令。 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

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

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 0 1 0		0 0 0 0		RX		1	1

WAIT——进入低功耗等待模式指令

统一化指令

语法	操作	编译结果
wait	进入低功耗等待模式	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

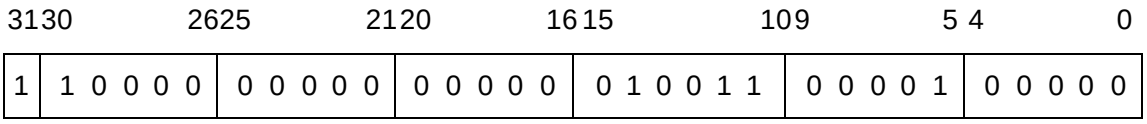
属性： 特权指令

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

指令格式：



XOR——按位异或指令

统一化指令

语法	操作	编译结果

xor rz, rx	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<16) and (z<16), then
		xor16 rz, rx;
		else
		xor32 rz, rz, rx;

xor rz, rx, ry	RZ ← RX ^ RY	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

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 1	RZ	RX	0 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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 1 0 0 1		0 0 0 1 0		RZ	

XORI——立即数按位异或指令

统一化指令

语法	操作	编译结果
xori rz, rx, imm16	RZ ← RX ^ zero_extend(IMM12)	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

指令格式：

31	30 26	25 21	20 16	15 12	11 0
1	1 1 0 0 1	RZ	RX	0 1 0 0	IMM12

XSR——扩展右移指令

统一化指令

语法	操作	编译结果
xsr rz, rx, oimm5	{RZ,C} ←{RX,C} >>>> OIMM5	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

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 1 0 0 0		RZ	

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

XTRB0——提取字节 0 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb0　rz, rx	RZ ← zero_extend(RX[31:24]);
	if (RX[31:24] == 0), then
	　C ← 0;
	else
	　C ← 1;
		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

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 0 0 1	RZ

XTRB1——提取字节 1 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb1　rz, rx	RZ ← zero_extend(RX[23:16]);
	if (RX[23:16] == 0), then
	　C ← 0;
	else
	　C ← 1;
		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

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 0 1 0	RZ

XTRB2——提取字节 2 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb2　rz, rx	RZ ← zero_extend(RX[15:8]);
	if (RX[15:8] == 0), then
	　C ← 0;
	else
	　C ← 1;
		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

指令格式：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	0	0	0	0	RX					0	1	1	1	0	0	0	0	1	0	0	RZ				

XTRB3——提取字节 3 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb3　rz, rx	RZ ← zero_extend(RX[7:0]);
	if (RX[7:0] == 0), then
	　C ← 0;
	else
	　C ← 1;
		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

指令格式：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	0	0	0	0	0	RX					0	1	1	1	0	0	0	1	0	0	0	RZ				

ZEXTB——字节提取并无符号扩展指令#

统一化指令

语法	操作	编译结果
zextb rz, rx	RZ ← zero_extend(RX[7:0]);	仅存在 16 位指令。 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

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

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 1	RZ	RX	0 0

ZEXTH——半字提取并无符号扩展指令#

统一化指令

语法	操作	编译结果
zexth rz, rx	RZ ← zero_extend(RX[15:0]);	仅存在 16 位指令。 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

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

指令格式

15 14 10 9 6 5 2 1 0

0	1 1 1 0 1	RZ	RX	0 1