
# 数字信号处理器 QXS320C28x
# 指令手册

v2.3

合肥乾芯科技有限公司

表 1：版本历史

| 版本号 | 日期 | 备注 |
|---|---|---|
| 1.0 | 2019/06 | 初始版本 |
| 1.2 | 2020/03 | 增加VIL、VIH指令，修正部分指令功能描述错误 |
| 1.3 | 2020/04 | 修正部分指令功能描述错误 |
| 1.4 | 2021/09 | 增加浮点矢量运算指令 |
| 1.5 | 2022/03 | 依流片版本信息更新文档 |
| 1.6 | 2022/06 | 更新C2000-3槽指令 |
| 1.7 | 2024/09 | 添加VCU指令 |
| 1.8 | 2024/10 | 添加部分64位指令 |
| 1.12 | 2024/12 | 修正MUL64指令行为、修正FSEQ指令语法、更新TMU指令说明、删除未实现的指令 |
| 2.0 | 2025/05/27 | 添加v2内核内容 |
| 2.1 | 2025/06/24 | 添加INTR指令说明 |
| 2.2 | 2025/08/19 | 添加v3内核说明及指令 |
| 2.3 | 2025/08/26 | 区分部分TMU指令的v1/v2版本 |

# 目 录

# 1. 概述

本文档为 QXS320C28x 指令手册，描述处理器指令及指令功能。

目前，QXS320C28x 内核有三个迭代版本：

v1：2024 年量产

v2：2025 年量产

- 在 v1 内核的基础上新增指令
- 扩充控制寄存器（CR）定义

v3：2025 年量产

- 在 v2 内核的基础上新增双精度浮点指令 和 CLA 任务触发指令

当前基于 QXS320C28x 内核的微控制器型号及内核版本如表 2 所示。

表 2：微控制器及内核版本

| 索引 | 微控制器 | 内核版本 |
|---|---|---|
| 1 | QXS320F280049RevA | v1 |
| 2 | QXS320F28377LRevA | |
| 3 | QXS320F280049RevB | v2 |
| 4 | QXS320F28377LRevB | |
| 5 | QXS320F2800137 | |
| 6 | QXS320F28034 | |
| 7 | QXS320F28P650 | v3 |
| 8 | QXS320F28377D | |

# 2. VLIW 槽定义

QXS320C28x 采用 VLIW 结构，并包含 3 个指令槽（slot0、slot1、slot2），每个指令槽可以容纳一条指令。每个 VLIW 可能包含的指令数量为 1~3，对应 VLIW 指令长度可能为 32~96 位。

每个槽能容纳的指令类型为：

Slot0：特殊指令、控制流指令、数据传送指令、标量运算指令、FPU/TMU/VCU 指令

Slot1：NOP、加载指令

Slot2：NOP、存储指令

# 3. 寄存器说明

## 3.1. 寄存器堆描述

| 标记 | 名称 | 宽度 | 数量 |
|---|---|---|---|
| GR | 通用寄存器 | 32 | 32 |
| OFF | 偏移寄存器 | 32 | 4 |
| BAR | 基址寄存器 | 32 | 4 |
| MR | 取模寄存器 | 32 | 4 |

## 3.2. 控制寄存器（CR: Control Register）

### 3.2.1. v1 内核

CR 可以通过 MOVG2C、MOVC2G 进行设置及保存

- [13:12]OVF：溢出标志位，10/11/00 三种情况对应上/下/无溢出。
- [7] OF: 溢出标志位，产生溢出，该标志位设置为 1，否则设置为 0。
- [4] CON：标量比较指令的结果，如果比较结果成立则该标志位设置为 1，否则设置为 0；如果当前不是比较指令，则该标志位保持不变。CR[4]对应 slot0。
- [0] CF：标量运算的最高位是否产生进位/借位， CR [0]对应 slot0。

### 3.2.2. v2/v3 内核

CR 可以通过 MOVG2C、MOVC2G 进行设置及保存

- [14]EALLOW：系统中的一些寄存器被 EALLOW 保护机制保护，防止伪 CPU 写入。
- [13:12]OVF：溢出标志位，10/11/00 三种情况对应上/下/无溢出。
- [7] OF: 溢出标志位，产生溢出，该标志位设置为 1，否则设置为 0。
- [4] CON：标量比较指令的结果，如果比较结果成立则该标志位设置为 1，否则设置为 0；如果当前不是比较指令，则该标志位保持不变。CR[4]对应 slot0。
- [2]LVF FPU/TMU 上溢
- [1]LUF FPU/TMU 下溢
- [0] CF：标量运算的最高位是否产生进位/借位， CR [0]对应 slot0。

## 3.3. 中断返回地址寄存器（ICA）

保存中断返回的地址，为特殊寄存器。寄存器地址为：0x007F0718。

## 3.4. 通用寄存器使用惯例

| 通用寄存器 | 使用惯例 | caller/callee-saved |
|---|---|---|
| GR0 | 中断向量表基地址 | n/a |
| GR1 | 通用寄存器 | n/a |
| GR2 | 传递函数返回值 | caller-saved |
| GR3 | 传递函数返回值（64bit 返回值的高 32bit） | |
| GR4 | 传递函数第一个实参 | |
| GR5 | 传递函数第二个实参 | |
| GR6 | 传递函数第三个实参 | |
| GR7 | 传递函数第四个实参 | |
| GR8 | 通用寄存器 | |
| GR9 | 通用寄存器 | |
| GR10 | 通用寄存器 | |
| GR11 | 通用寄存器 | |
| GR12 | 通用寄存器 | |
| GR13 | 通用寄存器 | |
| GR14 | 通用寄存器 | |
| GR15 | 通用寄存器 | |
| GR16 | 通用寄存器 | callee-saved |
| GR17 | 通用寄存器 | |
| GR18 | 通用寄存器 | |
| GR19 | 通用寄存器 | |
| GR20 | 通用寄存器 | |
| GR21 | 通用寄存器 | |
| GR22 | 通用寄存器 | |
| GR23 | 通用寄存器 | |
| GR24 | 通用寄存器 | caller-saved |
| GR25 | 通用寄存器 | |
| GR26 | 通用寄存器 | |
| GR27 | 通用寄存器 | |
| GR28 | 通用寄存器 | |
| GR29 | 通用寄存器 | callee-saved |
| GR30 | 栈指针（SP） | n/a |
| GR31 | 函数返回地址（LR） | callee-saved |

# 4. 基础指令

## 4.1. 特殊指令

### 4.1.1. NOP

**内核：**v1

**格式：**

NOP

**目的：**

空操作

**描述：**

不执行任何操作，不影响寄存器值和处理器状态。

**限制条件：**

;

**操作：**

;

**异常：**

;

**备注：**

;

### 4.1.2. IDLE

**内核：**v1

**格式：**

IDLE

**目的：**

进入低功耗模式

**描述：**

有三种低功耗模式，IDLE、STANDBY、HALT，三种低功耗模式通过 IDLE 指令进行进入，低功耗模式的种类通过 LPMCR 寄存器进行配置。

**限制条件：**

**操作：**

IDLE 是 C2000 CPU 的标准功能。在这种模式下，CPU 时钟是门控的，而所有外围时钟都保持运行。因此，IDLE 可以用于在 CPU 等待外设事件时节省 power。

任何启用的中断都会将 CPU 从 IDLE 模式唤醒。

1.设置 LPMCR， LPM 到 0x0 并执行 IDLE 指令，则进入到 IDLE_MODE。

2.当被启用中断事件发生时，进入到 IDLE_EXIT 状态，如果此时无 IDLE 指令，则进入 NORMAL，若中断未被屏蔽则会进入中断，否则 CPU 将恢复正常的操作。

STANDBY 是一种更激进的低功耗模式，它对 CPU 时钟和从 CPU 的 SYSCLK 派生的任何外设时钟进行门控，看门狗仍可处于 Active 状态，其他 CPU 子系统及其所有外设会被同步控制进入低功耗。STANDBY 最适合唤醒信号来自外部系统而不是外设输入的应用，看门狗中断（可选），GPIO0-63 中的任何一个可以被配置为在它们被驱动为低驱动时唤醒 CPU 子系统。在唤醒时，CPU 接收 WAKEINT 中断。

要进入 STANDBY 模式：(不包含 flash 和 pll 相关)

1.将 LPMCR.LPM 设置为 0x1。

2.启用 PIE 中的 WAKEINT 中断。

3.对于看门狗中断唤醒，将 LPMCR.WDINTE 设置为 1，并配置看门狗以生成中断。

4.对于 GPIO 唤醒，设置 GPIOLPMSEL0 和 GPIOLPMSEL1 以将所选 GPIO 连接到 LPM 模块。

5.执行 IDLE 指令。。

要从 STANDBY 模式唤醒：

1.配置所需的 GPIO 以触发唤醒。

2.将所选 GPIO 低电平驱动至少 5us，这将激活 WAKEINT PIE 中断； WAKEINT 中断被锁存在 PIE 块中。WAKEINT 中断也可以由看门狗中断触发。

3.若此时无 IDLE 指令，则退出低功耗模式进入 NORMAL 状态，执行 WAKEINT 中断，芯片开始正常工作。

CPU 现在已脱离 STANDBY 模式，可以恢复正常执行。

HALT 是一种全局低功耗模式，几乎可以门控所有系统时钟，并允许 XTAL 和模拟块断电。与其他 C2000 设备不同，HALT 模式不会在进入 HALT 时自动关闭 XTAL。

对于在 HALT 模式期间的最小功耗，应用程序软件应在进入 HALT 之前关闭 XTAL。如果 OSCCLK 源配置为 XTAL，则在设置 XTALCR.OSCOFF 之前，应用程序应首先将 OSSCLK 源切换到 INTOSC1 或 INTOSC2。

GPIO0-63 可以配置为从 HALT 唤醒系统，且没有其他唤醒选项可用。在唤醒时，CPU 接收 WAKEINT 中断。

要进入 HALT 模式：

1.启用 PIE 中的 WAKEINT 中断。

2.将 LPMCR.LPM 设置为 0x2。设置 GPIOLPMSEL0 和 GPIOLPMSEL1，将所选 GPIO 连接到 LPM 模块。

3.将 CLKSRCCTL1.WDHALTI 设置为 1，以保持看门狗定时器处于活动状态，并且 INTOSC1 和 INTOSC2 在 HALT 中通电。

4.将 CLKSRCCTL1.WDHALTI 设置为 0 以禁用看门狗定时器，并在 HALT 中关闭 INTOSC1 和 INTOSC2。

5.执行 IDLE 指令以进入 HALT 模式。

要从 HALT 模式唤醒：

1.将所选 GPIO 低电平驱动至少 5us，这将激活 WAKEINT PIE 中断。

2.再次驱动唤醒 GPIO 为高电平，以启动 SYSPLL 的加电。

3.等待 16us 加 1024 个 OSCCLK 周期，以允许 PLL 锁定和 WAKEINT ISR 锁定。

4.执行 WAKEINT ISR。

**异常：**

**备注：**

### 4.1.3. EALLOW

**内核：**v2

**格式：**

EALLOW

**目的：**

打开对外设写寄存器的控制

**描述：**

将控制寄存器中 eallow 控制位置 1

**限制条件：**

**操作：**

**cpu_ctrlReg[14]** ← 1

**异常：**

**备注：**

；

### 4.1.4. EDIS

**内核：**v2

**格式：**

EDIS

**目的：**

关闭对外设写寄存器的控制

**描述：**

将控制寄存器中 eallow 控制位置 0

**限制条件：**

**操作：**

**cpu_ctrlReg[14]** ← 0

**异常：**

**备注：**

；

### 4.1.5. SETCINTM

**内核：**v2

**格式：**

SETCINTM

**目的：**

屏蔽全体外设中断

**描述：**

作用同向 intm 寄存器写 1

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

### 4.1.6. CLRCINTM

**内核：**v2

**格式：**

CLRCINTM

**目的：**

解除屏蔽全体外设中断

**描述：**

作用同向 intm 寄存器写 0

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

# 4.2. 控制流指令

## 4.2.1. JMP

**内核：** v1

**格式：**

JMP $addr_{21}$

**目的：**

无条件相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

PC ← PC + addr << 2

**异常：**

；

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.2. JC

**内核：**v1

**格式：**

JC $addr_{21}$

**目的：**

满足条件则相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

**IF** CON

PC ← PC + addr << 2

**异常：**

；

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.3. JNC

**内核：** v1

**格式：**

JNC addr$_{21}$

**目的：**

不满足条件则相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

**IF** ~CON

PC ← PC + addr << 2

**异常：**

;

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.4. CALL

**内核：**v1

**格式：**

CALL $addr_{21}$

**目的：**

调用子程序。

**描述：**

目的地址的有效范围：当前 PC + [-4MB, 4MB-4B]，地址偏移由 21-bit 的 addr 左移 2 bit 产生。当前 PC+8（CALL_PC + 12）保存到 GR[31]（Link Register）

**限制条件：**

imm 为 21 位有符号立即数

**操作：**

GR[31] ← PC + 8

PC ← PC + addr << 2

**异常：**

；

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.5. JMPR

**内核：**v1

**格式：**

JMPR Rs

**目的：**

无条件绝对跳转到目标地址。

**描述：**

目标地址为 Rs 中的数。

**限制条件：**

Rs 为 GR

**操作：**

PC ← Rs

**异常：**

；

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.6. CALLR

**内核：**v1

**格式：**

CALLR Rs

**目的：**

调用子程序。

**描述：**

子程序地址为 Rs 中的数据。调用目标地址处的子程序，将当前 PC 保存在 GR[31]（Link Register）中，以便于跳出子程序

**限制条件：**

Rd 为 GR

**操作：**

GR[31] ← PC+8

PC ← Rs

**异常：**

；

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.7. RET

**内核：**v1

**格式：**

RET

**目的：**

子程序返回。

**描述：**

返回的目的地址为 GR[31]（Link Register）中的内容。

**限制条件：**

;

**操作：**

PC ← GR[31]

**异常：**

;

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.8. RTT

**内核：**v1

**格式：**

RTT

**目的：**

中断返回。

**描述：**

返回的目的地址为中断地址寄存器 ICA 的内容

**限制条件：**

;

**操作：**

PC ← ICA

**异常：**

;

**备注：**

有两个延迟槽，由编译器填充指令

### 4.2.9. LOOP

**内核：**v1

**格式：**

LOOP Rd，$addr_{16}$

**目的：**

循环。

**描述：**

循环次数为 Rd 值加一，循环出口地址 loop end 为地址标签（addr），即循环体最后一条指令地址。loopbegin 由硬件生成。每次在执行到循环体最后一条指令的时候，判断 loop counter 是否为 0，如果是则跳出循环至 loop end，如果不是则跳回 loop begin 重新执行循环体。

**限制条件：**

Rd 为 GR，addr 为 16 位立即数

**操作：**

loopend ← PC + addr << 2

loopcounter ← Rd

**异常：**

；

**备注：**

；

### 4.2.10. SYNCH

**内核：**v1

**格式：**

SYNCH

**目的：**

同步操作

**描述：**

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

### 4.2.11. TRAP

**内核：**v1

**格式：**

TRAP $imm_{18}$

**目的：**

用于调试的自陷指令

**描述：**

遇到 TRAP 后，处理器暂停，将 imm 送至 debug 模块的 DRR 寄存器

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

### 4.2.12. SJMP

**内核：**v2

**格式：**

SJMP $addr_{21}$

**目的：**

无条件相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

PC ← PC + addr << 2

**异常：**

；

**备注：**

该跳转指令不具有延迟槽；

### 4.2.13. SJC

**内核：**v2

**格式：**

SJC addr$_{21}$

**目的：**

满足条件则相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

**IF** CON

PC ← PC + addr << 2

**异常：**

;

**备注：**

该跳转指令不具有延迟槽；

### 4.2.14. SJNC

**内核：**v2

**格式：**

SJNC addr$_{21}$

**目的：**

不满足条件则相对跳转到目标地址。

**描述：**

地址的有效范围是当前的 8M($2^{21+2}$B)区域。相对地址由 21 位的 Addr 域左移 2 位产生。

**限制条件：**

imm 为 21 位立即数

**操作：**

**IF** ~CON

PC ← PC + addr << 2

**异常：**

；

**备注：**

该跳转指令不具有延迟槽；

### 4.2.15. SCALL

**内核：**v2

**格式：**

SCALL $addr_{21}$

**目的：**

调用子程序。

**描述：**

目的地址的有效范围：当前 PC + [-4MB, 4MB-4B]，地址偏移由 21-bit 的 addr 左移 2 bit 产生。当前 PC+8（CALL_PC + 12）保存到 GR[31]（Link Register）

**限制条件：**

imm 为 21 位有符号立即数

**操作：**

GR[31] ← PC + 8

PC ← PC + addr << 2

**异常：**

；

**备注：**

该跳转指令不具有延迟槽；

### 4.2.16. SJMPR

**内核：**v2

**格式：**

SJMPR Rs

**目的：**

无条件绝对跳转到目标地址。

**描述：**

目标地址为 Rs 中的数。

**限制条件：**

Rs 为 GR

**操作：**

PC ← Rs

**异常：**

；

**备注：**

该跳转指令不具有延迟槽；

### 4.2.17. SCALLR

**内核：**v2

**格式：**

SCALLR Rs

**目的：**

调用子程序。

**描述：**

子程序地址为 Rs 中的数据。调用目标地址处的子程序，将当前 PC 保存在 GR[31]（Link Register）中，以便于跳出子程序

**限制条件：**

Rd 为 GR

**操作：**

GR[31] ← PC+8

PC ← Rs

**异常：**

；

**备注：**

该跳转指令不具有延迟槽；

### 4.2.18. SRET

**内核：**v2

**格式：**

SRET

**目的：**

子程序返回。

**描述：**

返回的目的地址为 GR[31]（Link Register）中的内容。

**限制条件：**

;

**操作：**

PC ← GR[31]

**异常：**

;

**备注：**

该跳转指令不具有延迟槽；

### 4.2.19. SRTT

**内核：**v2

**格式：**

SRTT

**目的：**

中断返回。

**描述：**

返回的目的地址为中断地址寄存器 ICA 的内容

**限制条件：**

;

**操作：**

PC ← ICA

**异常：**

;

**备注：**

该跳转指令不具有延迟槽；

### 4.2.20. INTR

**内核：**v2

**格式：**

INTR $imm_{18}$

**目的：**

软件触发外设中断

**描述：**

软件指令触发外设中断(设置 INT1~INT14)，不受 picmr 位影响

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

### 4.2.21. IACK

**内核：**v3

**格式：**

IACK $imm_{18}$

**目的：**

软件触发 CLA task

**描述：**

软件指令触发 CLA task

**限制条件：**

；

**操作：**

**异常：**

；

**备注：**

；

# 4.3. 数据传送指令

## 4.3.1. MOVIGH

**内核：**v1

**格式：**

MOVIGH Rd，$imm_{16}$

**目的：**

将一个 16 位的立即数复制到 rd 的高 16 位中

**描述：**

$Rd_{31..16}$ ← $imm_{16}$

**限制条件：**

Rd 为 GR, OFF, BAR, MR；imm 为 16 位立即数

**操作：**

$GR[Rd]_{31..16}$ ← $imm_{16}$

**异常：**

；

**备注：**

；

### 4.3.2. MOVIGL

**内核：**v1

**格式：**

MOVIGL　Rd，$imm_{16}$

**目的：**

将一个 16 位的立即数复制到 rd 的低 16 位中

**描述：**

$Rd_{15..0}$ ← $imm_{16}$

**限制条件：**

Rd 为 GR, OFF, BAR, MR；imm 为 16 位立即数

**操作：**

$GR[Rd]_{15..0}$ ← $imm_{16}$

**异常：**

；

**备注：**

；

### 4.3.3. MOVG2G

**内核：**v1

**格式：**

MOVG2G Rd，Rs

**目的：**

将 Rs 中的数移送至 Rd 中

**描述：**

Rd ← Rs

**限制条件：**

Rd 为 GR, OFF, BAR, MR； Rs 为 GR, OFF, BAR, MR

**操作：**

GR[Rd] ←GR[Rs]

**异常：**

；

**备注：**

；

### 4.3.4. MOVG2C

**内核：**v1

**格式：**

MOVG2C Rs

**目的：**

用 Rs 的内容设置控制寄存器

**描述：**

Control Register ← Rs

**限制条件：**

Rs 为 GR

**操作：**

Control Register ←GR[Rs]

**异常：**

；

**备注：**

；

### 4.3.5. MOVC2G

**内核：**v1

**格式：**

MOVC2G Rd

**目的：**

将控制寄存器的内容保存到 GR 中

**描述：**

Rs ← Control Register

**限制条件：**

Rd 为 GR

**操作：**

GR[Rs] ← Control Register

**异常：**

；

**备注：**

；

### 4.3.6. MOVIGLZ

**内核：**v2

**格式：**

MOVIGLZ Rd，$imm_{16}$

**目的：**

将 rd 的低 16 位写入 16 位立即数，将 rd 的高 16 位写入 0

**描述：**

$Rd_{31..0}$ ← {16'd0, $imm_{16}$}

**限制条件：**

Rd 为 GR, OFF, BAR, MR；imm 为 16 位立即数

**操作：**

$GR[Rd]_{31..0}$ ← {16'd0, $imm_{16}$}

**异常：**

；

**备注：**

；

### 4.3.7. MOVIGLX

**内核：**v2

**格式：**

MOVIGLX Rd，$imm_{16}$

**目的：**

将一个 16 位的有符号立即数进行符号位扩展至 32bit 后复制到 rd 中

**描述：**

$Rd_{31..0}$ ← sign_extend($imm_{16}$)

**限制条件：**

Rd 为 GR, OFF, BAR, MR；imm 为 16 位立即数

**操作：**

$GR[Rd]_{31..0}$ ← sign_extend($imm_{16}$)

**异常：**

；

**备注：**

；

# 4.4. 加载/存储指令

## 4.4.1. LOAD8

**内核：** v1

**格式：**

LOAD8 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 8bit 的数据。

**描述：**

从内存的目标地址处读取 8bit 的数据，零符号扩展到 32bit，复制到 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

$Rd_{7..0}$ ← zero_extend (MEM[Rs + $imm_9$])

**异常：**

；

**备注：**

；

### 4.4.2. LOAD16

**内核：**v1

**格式：**

LOAD16 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 16bit 的数据。

**描述：**

从内存的目标地址处读取 16bit 的数据，零符号扩展到 32bit，复制到 Rd 中。地址偏移量为 $imm_9$ 左移一位。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

$Rd_{15..0}$ ← MEM[Rs + $imm_9$ << 1]

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.3. LOAD32

**内核：**v1

**格式：**

LOAD32 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 32bit 的数据。

**描述：**

从内存的目标地址处读取 32bit 的数据，存放在 Rd 中。地址偏移量为 $imm_9$ 左移 2 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

Rd ← MEM[Rs + $imm_9$ << 2]

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

；

### 4.4.4. STORE8

**内核：**v1

**格式：**

STORE8　Rd, Rs, $imm_9$

**目的：**

写存储器，将 8bit 数据写入内存的目标地址处。

**描述：**

将 Rd 的低 8bit 数据写入内存的目标地址处。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$] ← $Rd_{7..0}$

**异常：**

；

**备注：**

；

### 4.4.5. STORE16

**内核：**v1

**格式：**

STORE16 Rd, Rs, $imm_9$

**目的：**

写存储器，将 16bit 数据写入内存的目标地址处。

**描述：**

将 Rd 的低 16bit 数据写入内存的目标地址处。地址偏移量为 $imm_9$ 左移 1 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$ << 1] ← $Rd_{15..0}$

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.6. STORE32

**内核：**v1

**格式：**

STORE32 Rd, Rs, $imm_9$

**目的：**

写存储器，将 32bit 数据写入内存的目标地址处。地址偏移量为 $imm_9$ 左移 2 位。

**描述：**

将 Rd 中数据写入内存的目标地址处。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$ << 2] ← Rd

**异常：**

访存地址最低两位不为 00，则发出异常。

**备注：**

；

### 4.4.7. LOADU8

**内核：**v1

**格式：**

LOADU8 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 8bit 的数据。并且更新基址寄存器。

**描述：**

从内存的目标地址处读取 8bit 的数据，存放在 Rd 的低位。同时更新基址寄存器 Rs

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

$Rd_{7..0}$ ← MEM[Rs + $imm_9$]

Rs ← Rs + $imm_9$

**异常：**

；

**备注：**

；

### 4.4.8. LOADU16

**内核：**v1

**格式：**

LOADU16 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 16bit 的数据。并且更新基址寄存器。

**描述：**

从内存的目标地址处读取 16bit 的数据，存放在 Rd 的低位，同时更新基址寄存器 Rs。地址偏移量为 $imm_9$ 左移 1 位

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

$Rd_{15..0}$ ← MEM[Rs + $imm_9$ << 1]

Rs ← Rs + $imm_9$ << 1

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.9. LOADU32

**内核：**v1

**格式：**

LOADU32 Rd, Rs, $imm_9$

**目的：**

访问存储器，读取目标地址出 32bit 的数据。并且更新基址寄存器。

**描述：**

从内存的目标地址处读取 32bit 的数据，存放在 Rd 中，同时更新基址寄存器 Rs. 地址偏移量为 $imm_9$ 左移 2 位

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

Rd ← MEM[Rs + $imm_9$ << 2]

Rs ← Rs + $imm_9$ << 2

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

；

### 4.4.10. STOREU8

**内核：**v1

**格式：**

STOREU8 Rd, Rs, $imm_9$

**目的：**

写存储器，将 8bit 数据写入内存的目标地址处。同时更新基址寄存器。

**描述：**

将 Rd 的低 8bit 数据写入内存的目标地址处，同时更新基址寄存器 Rs。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$] ← $Rd_{7..0}$

Rs ← Rs +$imm_9$

**异常：**

；

**备注：**

；

### 4.4.11. STOREU16

**内核：**v1

**格式：**

STOREU16　Rd, Rs, $imm_9$

**目的：**

写存储器，将 16bit 数据写入内存的目标地址处。同时更新基址寄存器。

**描述：**

将 Rd 的低 16bit 数据写入内存的目标地址处，同时更新基址寄存器 Rs。地址偏移量为 $imm_9$ 左移 1 位

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$ << 1] ← $Rd_{15..0}$

Rs ← Rs +$imm_9$ << 1

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.12. STOREU32

**内核：**v1

**格式：**

STOREU32 Rd, Rs, $imm_9$

**目的：**

写存储器，将 32bit 数据写入内存的目标地址处。同时更新基址寄存器。

**描述：**

将 Rd 的数据写入内存的目标地址处，同时更新基址寄存器 Rs。地址偏移量为 $imm_9$ 左移 2 位

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 9 位立即数

**操作：**

MEM[Rs + $imm_9$ << 2] ← Rd

Rs ← Rs +$imm_9$ << 2

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

；

### 4.4.13. LOADO16

**内核：** v1

**格式：**

LOADO16 Rd, Rs, Rt

**目的：**

循环访问存储器，读取目标地址出 16bit 的数据。

**描述：**

从内存的目标地址处读取 16bit 的数据，存放在 Rd 的低位。Rt 对应一组寄存器:BAR, OFF, MR。取完后需要递增 Rs，递增的步长为 OFF，但是需要保证 Rs 不超过 BAR + MR，如果超过则将模 MR。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 BAR、OFF、MR。

**操作：**

$Rd_{15..0}$ ← MEM[Rs]

**IF** Rs + OFF > MR + BAR
    Rs = Rs + OFF – MR
**ELSE IF** Rs + OFF < BAR
    Rs = Rs + OFF + MR
**ELSE**
    Rs = Rs + OFF

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.14. LOADO32

**内核：** v1

**格式：**

LOADO32 Rd, Rs, Rt

**目的：**

循环访问存储器，读取目标地址出 32bit 的数据。

**描述：**

从内存的目标地址处读取 32bit 的数据，存放在 Rd 中。Rt 对应一组寄存器:BAR, OFF, MR。取完后需要递增 Rs，递增的步长为 OFF，但是需要保证 Rs 不超过 BAR + MR，如果超过则将模 MR。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 BAR、OFF、MR。

**操作：**

Rd ← MEM[Rs]

**IF** Rs + OFF > MR + BAR
    Rs = Rs + OFF – MR
**ELSE IF** Rs + OFF < BAR
    Rs = Rs + OFF + MR
**ELSE**
    Rs = Rs + OFF

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

使用实例：loado32 gr0 gr1 bar1

其中，bar - base address，off – offset ,mr – modulus

三个 MOR 寄存器共有 4 组，见寄存器简述部分；

### 4.4.15. STOREO16

**内核：** v1

**格式：**

STOREO16 Rd, Rs, Rt

**目的：**

循环写存储器，向目标地址写 16bit 的数据。

**描述：**

将 Rd 的低 16 位写入内存的目标地址处。Rt 对应一组寄存器:BAR, OFF, MR。写完后需要递增 Rs，递增的步长为 OFF，但是需要保证 Rs 不超过 BAR + MR，如果超过则将模 MR

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 BAR、OFF、MR。

**操作：**

MEM[Rs] ← $Rd_{15..0}$

**IF** Rs + OFF > MR + BAR
    Rs = Rs + OFF – MR
**ELSE IF** Rs + OFF < BAR
    Rs = Rs + OFF + MR
**ELSE**
    Rs = Rs + OFF

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.16. STOREO32

**内核：** v1

**格式：**

STOREO32 Rd, Rs, Rt

**目的：**

循环写存储器，向目标地址写 32bit 的数据。

**描述：**

将 Rd 的数据写入内存的目标地址处。Rt 对应一组寄存器:BAR, OFF, MR。写完后需要递增 Rs，递增的步长为 OFF，但是需要保证 Rs 不超过 BAR + MR，如果超过则将模 MR

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 BAR、OFF、MR。

**操作：**

MEM[Rs] ← Rd

**IF** Rs + OFF > MR + BAR
    Rs = Rs + OFF – MR
**ELSE IF** Rs + OFF < BAR
    Rs = Rs + OFF + MR
**ELSE**
    Rs = Rs + OFF

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

；

### 4.4.17. LOADI8

**内核：**v2

**格式：**

LOADI8 Rd, $imm_{15}$

**目的：**

访问存储器，读取目标地址出 8bit 的数据。

**描述：**

从内存的目标地址处读取 8bit 的数据，零符号扩展到 32bit，复制到 Rd 中。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

$Rd_{7..0}$ ← zero_extend (MEM[$imm_{15}$])

**异常：**

；

**备注：**

；

## 4.4.18. LOADI16

**内核：**v2

**格式：**

LOADI16 Rd, $imm_{15}$

**目的：**

访问存储器，读取目标地址出 16bit 的数据。

**描述：**

从内存的目标地址处读取 16bit 的数据，零符号扩展到 32bit，复制到 Rd 中。地址偏移量为 $imm_{15}$ 左移一位。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

$Rd_{15..0}$ ← MEM[ $imm_{15}$ << 1]

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

### 4.4.19. LOADI32

**内核：**v2

**格式：**

LOADI32 Rd, $imm_{15}$

**目的：**

访问存储器，读取目标地址出 32bit 的数据。

**描述：**

从内存的目标地址处读取 32bit 的数据，存放在 Rd 中。地址偏移量为 $imm_{15}$ 左移 2 位。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

Rd ← MEM[$imm_{15}$ << 2]

**异常：**

访存地址最低两位不为 00，则发出异常；

**备注：**

；

## 4.4.20. STOREI8

**内核：**v2

**格式：**

STOREI8　Rd, $imm_{15}$

**目的：**

写存储器，将 8bit 数据写入内存的目标地址处。

**描述：**

将 Rd 的低 8bit 数据写入内存的目标地址处。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

MEM[$imm_{15}$] ← $Rd_{7..0}$

**异常：**

；

**备注：**

；

## 4.4.21. STOREI16

**内核：**v2

**格式：**

STOREI16 Rd, $imm_{15}$

**目的：**

写存储器，将 16bit 数据写入内存的目标地址处。

**描述：**

将 Rd 的低 16bit 数据写入内存的目标地址处。地址偏移量为 $imm_{15}$ 左移 1 位。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

MEM[$imm_{15}$ << 1] ← $Rd_{15..0}$

**异常：**

访存地址最低位不为 0，则发出异常；

**备注：**

；

## 4.4.22. STOREI32

**内核：**v2

**格式：**

STOREI32 Rd, $imm_{15}$

**目的：**

写存储器，将 32bit 数据写入内存的目标地址处。地址偏移量为 $imm_{15}$ 左移 2 位。

**描述：**

将 Rd 中数据写入内存的目标地址处。

**限制条件：**

Rd 为 GR，imm 为 15 位立即数

**操作：**

MEM[$imm_{15}$ << 2] ← Rd

**异常：**

访存地址最低两位不为 00，则发出异常。

**备注：**

；

# 4.5. 标量运算指令 – 比较指令

标量运算操作数 Rd, Rs, Rt 都是 GR，标量指令都是整数运算

## 4.5.1. EQI

**内核：** v1

**格式：**

EQI Rs，$imm_9$

**目的：**

有符号判断。比较两个 32 位数是否相等

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs == sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

```
IF GR[Rs] == sign_extend(imm9)
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

；

**备注：**

；

### 4.5.2. NEQI

**内核：**v1

**格式：**

NEQI Rs，$imm_9$

**目的：**

有符号判断。比较两个 32 位数是否相等

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs != sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

```
IF GR[Rs] != sign_extend(imm9)
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.3. GTI

**内核：**v1

**格式：**

GTI Rs，$imm_9$

**目的：**

有符号判断。比较 Rs 是否大于立即数 imm

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs > sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] > sign_extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

### 4.5.4. LTI

**内核：** v1

**格式：**

LTI Rs，$imm_9$

**目的：**

有符号判断。比较 Rs 是否小于立即数 imm

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs < sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

```
IF GR[Rs] < sign_extend(imm9)
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.5. GEI

**内核：**v1

**格式：**

GEI Rs，$imm_9$

**目的：**

有符号判断。比较 Rs 是否大于等于立即数 imm

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs >= sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] >= sign_extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

### 4.5.6. LEI

**内核：**v1

**格式：**

LEI Rs，$imm_9$

**目的：**

有符号判断。Rs 是否小于等于立即数 imm

**描述：**

$imm_9$ 有符号扩展至 32bit 进行比较

CON = (Rs <= sign_extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] <= sign_extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

### 4.5.7. EQ

**内核：**v1

**格式：**

EQ Rs，Rt

**目的：**

有符号判断。比较两个 32 位数是否相等

**描述：**

CON = (Rs == Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] == GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.8. NEQ

**内核：**v1

**格式：**

NEQ Rs，Rt

**目的：**

有符号判断。比较两个 32 位数是否不相等

**描述：**

CON = (Rs != Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] != GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.9. GT

**内核：**v1

**格式：**

GT Rs， Rt

**目的：**

有符号判断。比较 Rs 是否大于 Rt。

**描述：**

CON = (Rs > Rt)

**限制条件：**

Rs 为 GR， Rt 为 GR

**操作：**

```
IF GR[Rs] > GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

；

**备注：**

；

### 4.5.10. LT

**内核：**v1

**格式：**

LT Rs，Rt

**目的：**

有符号判断。比较 Rs 是否小于 Rt。

**描述：**

CON = (Rs < Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] < GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.11. GE

**内核：**v1

**格式：**

GE Rs， Rt

**目的：**

有符号判断。比较 Rs 是否大于等于 Rt。

**描述：**

CON = (Rs >= Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] >= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.12. LE

**内核：**v1

**格式：**

LE Rs，Rt

**目的：**

有符号判断。比较 Rs 是否小于等于 Rt

**描述：**

CON = (Rs <= Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] <= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

；

**备注：**

；

### 4.5.13. GTU

**内核：**v1

**格式：**

GTU Rs，Rt

**目的：**

无符号判断。比较 Rs 是否大于 Rt。

**描述：**

CON = (Rs > Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] > GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.14. LTU

**内核：**v1

**格式：**

LTU Rs，Rt

**目的：**

无符号判断。比较 Rs 是否小于 Rt。

**描述：**

CON = (Rs < Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] < GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.15. GEU

**内核：**v1

**格式：**

GEU Rs，Rt

**目的：**

无符号判断。比较 Rs 是否大于等于 Rt。

**描述：**

CON = (Rs >= Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] >= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.16. LEU

**内核：**v1

**格式：**

LEU Rs， Rt

**目的：**

无符号判断。比较 Rs 是否小于等于 Rt

**描述：**

CON = (Rs <= Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] <= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

### 4.5.17. TEST

**内核：**v1

**格式：**

TEST Rs，Rt

**目的：**

测试 Rs 中某一位是否为 1

**描述：**

测试 Rs 的第 Rt 位是否为 1（最低位定义为第 0 位）

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

**IF** $GR[Rs]_{Rt}$ == 1
    CON = 1;
**ELSE**
    CON = 0;

**异常：**

;

**备注：**

;

## 4.5.18. TESTI

**内核：**v1

**格式：**

TESTI Rs，$imm_5$

**目的：**

测试 Rs 中某一位是否为 1

**描述：**

测试 Rs 的第 $imm_5$ 位是否为 1（最低位定义为第 0 位）

**限制条件：**

Rs 为 GR，imm 为 5 位立即数

**操作：**

**IF** $GR[Rs]_{imm}$ == 1
    CON = 1;
**ELSE**
    CON = 0;

**异常：**

;

**备注：**

;

### 4.5.19. EQU

**内核：**v2

**格式：**

EQU Rs，Rt

**目的：**

无符号判断。比较两个 32 位数是否相等

**描述：**

CON = (Rs == Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] == GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

## 4.5.20. NEQU

**内核：**v2

**格式：**

NEQU Rs， Rt

**目的：**

无符号判断。比较两个 32 位数是否不相等

**描述：**

CON = (Rs != Rt)

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] != GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

;

**备注：**

;

## 4.5.21. EQIU

**内核：**v2

**格式：**

EQIU Rs，$imm_9$

**目的：**

无符号判断。比较两个 32 位数是否相等

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs == extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] == extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

## 4.5.22. NEQIU

**内核：**v2

**格式：**

NEQIU Rs，$imm_9$

**目的：**

无符号判断。比较两个 32 位数是否相等

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs != extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] != extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

## 4.5.23. GTIU

**内核：**v2

**格式：**

GTIU Rs，$imm_9$

**目的：**

无符号判断。比较 Rs 是否大于立即数 imm

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs > extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] > extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

## 4.5.24. LTIU

**内核：**v2

**格式：**

LTIU Rs，$imm_9$

**目的：**

无符号判断。比较 Rs 是否小于立即数 imm

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs < extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] < extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

## 4.5.25. GEIU

**内核：**v2

**格式：**

GEIU Rs，$imm_9$

**目的：**

无符号判断。比较 Rs 是否大于等于立即数 imm

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs >= extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] >= extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

## 4.5.26. LEIU

**内核：**v2

**格式：**

LEIU Rs，$imm_9$

**目的：**

无符号判断。Rs 是否小于等于立即数 imm

**描述：**

extend ($imm_9$) = {21'd0, $imm_9$}

CON = (Rs <= extend ($imm_9$))

**限制条件：**

Rs 为 GR，imm 为 9 位立即数

**操作：**

IF GR[Rs] <= extend($imm_9$)

CON = 1；

ELSE

CON = 0；

**异常：**

；

**备注：**

；

# 4.6. 标量运算指令 – 运算指令

## 4.6.1. ADDI

**内核：** v1

**格式：**

ADDI Rd，Rs，$imm_{11}$

**目的：**

将一个 32 位数与一个常数相加得到结果

**描述：**

Rd ← Rs + $imm_{11}$

Imm 进行符号扩展至 32 位(加法运算只是局限在低 11 位，即加到 0x7ff 时即会拓展到 0xffffffff，当 0xffffffff 时再加高于 11 位有效的值时不变，但是加 1 之后就会变为 0，即运算只是局限在低 11 位)。最高位产生进位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 11 位立即数

**操作：**

{CARRY,，GR[Rd]} ← GR[Rs] + sign_extend ($imm_{11}$)

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.2. ADDIC

**内核：**v1

**格式：**

ADDIC Rd，Rs，$imm_{11}$

**目的：**

将一个 32 位数和一个常数相加，低位再加上 CARRY 位得到结果。

**描述：**

Rd ← Rs + imm + CARRY

Imm 进行符号扩展至 32 位。最高位产生进位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，imm 为 11 位立即数

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] + sign_extend ($imm_{11}$)+ CARRY

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.3. ADD

**内核：**v1

**格式：**

ADD Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数相加得到结果

**描述：**

Rd ← Rs + Rt

最高位产生进位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] + GR[Rt]

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.4. ADDC

**内核：**v1

**格式：**

ADDC Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数相加，低位加上 CARRY 位得到结果

**描述：**

Rd ← Rs + Rt + CARRY

最高位产生进位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] + GR[Rt] + CARRY

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.5. SUB

**内核：**v1

**格式：**

SUB Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数相减得到结果

**描述：**

Rd ← Rs – Rt

最高位产生借位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] – GR[Rt]

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.6. SUBC

**内核：**v1

**格式：**

SUBC Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数相减，低位减去 CARRY 得到结果

**描述：**

Rd ← Rs – Rt – CARRY

最高位产生借位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] – GR[Rt] – CARRY

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.7. MUL32

**内核：**v1

**格式：**

MUL32 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数进行有符号相乘得到结果的低 32 位。

**描述：**

将 Rs 和 Rt 中的 32 位数据有符号相乘得到 64 位数据，截取低 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt]

GR[Rd] ← $temp_{31..0}$

**异常：**

；

**备注：**

；

### 4.6.8. MUL64

**内核：**v1

**格式：**

MUL64 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数有符号相乘得到结果的高 32 位。

**描述：**

将 Rs 和 Rt 中的 32 位数据有符号相乘得到 64 位数据，截取高 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt]

GR[Rd] ← $temp_{63..32}$

**异常：**

；

**备注：**

；

### 4.6.9. MULU32

**内核：**v1

**格式：**

MULU32 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数无符号相乘得到结果的低 32 位。

**描述：**

将 Rs 和 Rt 中的 32 位数据无符号相乘得到 64 位数据，将其低 32 位保存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt]

GR[Rd] ← $temp_{31..0}$

**异常：**

；

**备注：**

；

### 4.6.10. MAX

**内核：**v1

**格式：**

MAX Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数按照有符号数进行比较，得到较大的那个数

**描述：**

Rd ← Rs > Rt ? Rs : Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

**IF** GR[Rs] > GR[Rt]

GR[Rd] ← GR[Rs]

**ELSE**

GR[Rd] ← GR[Rt]

**异常：**

；

**备注：**

；

### 4.6.11. MIN

**内核：**v1

**格式：**

MIN Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数按照有符号数进行比较，得到较小的那个数

**描述：**

Rd ← Rs < Rt ? Rs : Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

**IF** GR[Rs] < GR[Rt]

GR[Rd] ← GR[Rs]

**ELSE**

GR[Rd] ← GR[Rt]

**异常：**

；

**备注：**

；

### 4.6.12. ABS

**内核：**v1

**格式：**

ABS Rd，Rs

**目的：**

得到一个 32 位有符号数的绝对值

**描述：**

Rd ← |Rs|

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← |GR[Rs]|

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.13. CBW

**内核：**v1

**格式：**

CBW Rd，Rs

**目的：**

整数有符号扩展。

**描述：**

将 Rs 的低 8 位进行有符号扩展至 32 位后保存在 Rd 中

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← sign_extend ($GR[Rs]_{7..0}$)

**异常：**

；

**备注：**

；

### 4.6.14. CHW

**内核：** v1

**格式：**

CHW Rd，Rs

**目的：**

整数有符号扩展。

**描述：**

将 Rs 的低 16 位进行有符号扩展至 32 位后保存在 Rd 中

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← sign_extended (GR[Rs]$_{15..0}$)

**异常：**

；

**备注：**

；

### 4.6.15. SAT64

**内核：**v1

**格式：**

SAT64 Rd，Rs

**目的：**

根据溢出结果对 64 位内容做饱和。

**描述：**

Rs+1 和 Rs 两个 32 位寄存器组合成 64 位源操作数，其中 Rs+1 存放 64 位数高 32 位，Rs 存放 64 位数低 32 位，比较溢出计数器 OVC 的值与 0，结果写入 Rd 表示写入 Rd+1 和 Rd 两个 32 位寄存器组合，作为目的寄存器，其中 Rd+1 存放 64 位结果高 32 位，Rd 存放 64 位结果低 32 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，且 Rd、Rs 必须为偶数

**操作：**

```
IF OVC > 0
    {GR[Rd+1], GR[Rd]} = 0x7FFF FFFF FFFF FFFF;
IF OVC < 0
    {GR[Rd+1], GR[Rd]} = 0x8000 0000 0000 0000;
IF OVC = 0
    {GR[Rd+1], GR[Rd]} = {GR[Rs+1], GR[Rs]};
```

**异常：**

;

**备注：**

;

### 4.6.16. NEG

**内核：**v2

**格式：**

NEG Rd

**目的：**

32 位有符号整数取负

**描述：**

对 GR[Rs]中存放的 32 位有符号整数取负（取反后＋1），结果存到 GR[Rd]。

**限制条件：**

Rd 为 GR

**操作：**

```
if (Rs == 0x80000000)
    Rd = 0x80000000;
else
    Rd = -Rs;
```

**异常：**

;

**备注：**

若操作数为 0x80000000（-2147483648），则设置溢出标志位为 1，否则置 0。

### 4.6.17. SUBI

**内核：**v2

**格式：**

SUBI Rd，Rs，$imm_{11}$

**目的：**

将一个 32 位数与一个 32 位数相减得到结果

**描述：**

Rd ← Rs – $imm_{11}$

Imm 进行符号扩展至 32 位，最高位产生借位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，$imm_{11}$ 为 11 为立即数

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] –sign_extend ($imm_{11}$)

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.18. SUBIC

**内核：**v2

**格式：**

SUBIC Rd，Rs，$imm_{11}$

**目的：**

将一个 32 位数与一个 32 位数相减，低位减去 CARRY 得到结果

**描述：**

Rd ← Rs – $imm_{11}$ – CARRY

Imm 进行符号扩展至 32 位。最高位产生进位则置 CARRY 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，$imm_{11}$ 为 11 为立即数

**操作：**

{CARRY，GR[Rd]} ← GR[Rs] – sign_extend ($imm_{11}$)– CARRY

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.6.19. SELECT

**内核：** v2

**格式：**

SELECT Rd, Rs, Rt

**目的：**

根据 CR 中的条件标志位，选择 GR 的数值来源

**描述：**

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

```
if CON
 {
    Rd <= Rs
 }
else
{
    Rd <= Rt
}
```

**异常：**

;

**备注：**

### 4.6.20. SETC

**内核：**v2

**格式：**

SETC Rd

**目的：**

根据 CR 中的条件标志位，选择 GR 的数值来源

**描述：**

**限制条件：**

Rd 为 GR

**操作：**

```
if CON
 {
    Rd <= 1
 }
else
{
    Rd <= 0
}
```

**异常：**

;

**备注：**

### 4.6.21. MULU64

**内核：**v2

**格式：**

MULU.32 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数无符号相乘得到结果的高 32 位。

**描述：**

将 Rs 和 Rt 中的 32 位数据无符号相乘得到 64 位数据，将其高 32 位保存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt]

GR[Rd] ← $temp_{63..32}$

**异常：**

**备注：**

；

## 4.6.22. MAC16

**内核：**v2

**格式：**

MAC16 Rd，Rs，Rt

**目的：**

有符号双 16 位定点整数乘累加指令

**描述：**

有符号双 16 位定点整数乘累加指令，对 Rs 和 Rd 的高 16 位和低 16 位分开进行 16 位乘法，即 Rs_h * Rt_h + Rd_h 和 Rs_l * Rt_l + Rd_l。得到的 64 位结果，高 32 位结果为 Rs_h * Rt_h + Rd_h，存放到 Rd+1；低 32 位结果为 Rs_l * Rt_l + Rd_l，存放到 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ←Rs_l * Rt_l + Rd_l

GR[Rd+1] ← Rs_h * Rt_h + Rd_h

**异常：**

；

**备注：**

## 4.6.23. MACU16

**内核：**v2

**格式：**

MACU16　Rd，Rs，Rt

**目的：**

无符号双 16 位定点整数乘累加指令

**描述：**

无符号双 16 位定点整数乘累加指令，对 Rs 和 Rd 的高 16 位和低 16 位分开进行 16 位乘法，即 Rs_h * Rt_h + Rd_h 和 Rs_l * Rt_l + Rd_l。得到的 64 位结果，高 32 位结果为 Rs_h * Rt_h + Rd_h，存放到 Rd+1；低 32 位结果为 Rs_l * Rt_l + Rd_l，存放到 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ←Rs_l * Rt_l + Rd_l

GR[Rd+1] ← Rs_h * Rt_h + Rd_h

**异常：**

**备注：**

### 4.6.24. MAC32

**内核：**v2

**格式：**

MAC32 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数进行有符号相乘后加上一个 32 位累加数得到结果的低 32 位。

**描述：**

32 位有符号数乘累加指令，取低 32 位结果。将 Rs 和 Rt 中的 32 位数据有符号相乘后再加上 Rd 中的 32 位数据得到 64 位数据，截取低 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt] + GR[Rd]

GR[Rd] ← $temp_{31..0}$

**异常：**

**备注：**

；

### 4.6.25. MAC64

**内核：**v2

**格式：**

MAC64 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数进行有符号相乘后加上一个 32 位累加数得到结果的高 32 位。

**描述：**

32 位有符号数乘累加指令，取高 32 位结果。将 Rs 和 Rt 中的 32 位数据有符号相乘后再加上 Rd 中的 32 位数据得到 64 位数据，截取高 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt] + GR[Rd]

GR[Rd] ← $temp_{63..32}$

**异常：**

**备注：**

## 4.6.26. MACU32

**内核：**v2

**格式：**

MACU32 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数进行无符号相乘后加上一个 32 位累加数得到结果的高=低 32 位。

**描述：**

32 位无符号数乘累加指令，取低 32 位结果。将 Rs 和 Rt 中的 32 位数据无符号相乘后再加上 Rd 中的 32 位数据得到 64 位数据，截取低 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt] + GR[Rd]

GR[Rd] ← $temp_{31..0}$

**异常：**

；

**备注：**

；

## 4.6.27. MACU64

**内核：**v2

**格式：**

MACU.64 Rd，Rs，Rt

**目的：**

将一个 32 位整数与一个 32 位整数进行无符号相乘后加上一个 32 位累加数得到结果的高 32 位。

**描述：**

32 位无符号数乘累加指令，取高 32 位结果。将 Rs 和 Rt 中的 32 位数据无符号相乘后再加上 Rd 中的 32 位数据得到 64 位数据，截取高 32 位存在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] * GR[Rt] + GR[Rd]

GR[Rd] ← $temp_{63..32}$

**异常：**

；

**备注：**

；

# 4.7. 标量运算指令 – 逻辑指令

## 4.7.1. AND

**内核：** v1

**格式：**

AND Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数按位与得到结果

**描述：**

Rd ← Rs & Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] & GR[Rt]

**异常：**

；

**备注：**

；

### 4.7.2. OR

**内核：**v1

**格式：**

OR Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数按位或得到结果

**描述：**

Rd ← Rs | Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] | GR[Rt]

**异常：**

；

**备注：**

；

### 4.7.3. XOR

**内核：**v1

**格式：**

XOR Rd，Rs，Rt

**目的：**

将一个 32 位数与一个 32 位数按位异或得到结果

**描述：**

Rd ← Rs ^ Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] ^ GR[Rt]

**异常：**

；

**备注：**

；

### 4.7.4. NOT

**内核：**v1

**格式：**

NOT Rd，Rs

**目的：**

将一个 32 位数按位取反得到结果

**描述：**

Rd ← ~Rs

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← ~GR[Rs]

**异常：**

；

**备注：**

；

### 4.7.5. SL

**内核：**v1

**格式：**

SL Rd，Rs，Rt

**目的：**

将一个 32 位数进行左移（空出的低位补零）得到结果，保存低 32 位

**描述：**

Rd ← Rs << Rt

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

temp ← GR[Rs] << GR[Rt]

GR[Rd] ← $temp_{31..0}$

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.7.6. SRA

**内核：**v1

**格式：**

SRA Rd，Rs，Rt

**目的：**

将一个 32 位数进行算数右移得到结果

**描述：**

Rd ← Rs >>> Rt

空出的高位补符号位

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd]← GR[Rs] >>> GR[Rt]

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.7.7. SRL

**内核：**v1

**格式：**

SRL Rd，Rs，Rt

**目的：**

将一个 32 位数进行逻辑右移得到结果

**描述：**

Rd ← Rs >> Rt

空出的高位补零

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd]← GR[Rs] >> GR[Rt]

**异常：**

根据结果是否溢出设置 OF；

**备注：**

；

### 4.7.8. NEG64

**内核：**v1

**格式：**

NEG64 Rd， Rs

**目的：**

将一个 64 位数按位取反。

**描述：**

(Rd+1) , Rd ← ~(Rs+1) , ~Rs

Rs+1 和 Rs 两个 32 位寄存器组合成 64 位源操作数，其中 Rs+1 存放 64 位数高 32 位，Rs 存放 64 位数低 32 位，将源操作数按位取反，结果写入 Rd 表示写入 Rd+1 和 Rd 两个 32 位寄存器组合，作为目的寄存器，其中 Rd+1 存放 64 位结果高 32 位，Rd 存放 64 位结果低 32 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，且 Rd、Rs 必须为偶数

**操作：**

{GR[Rd+1], GR[Rd]} = ~{GR[Rs+1], GR[Rs]};

**异常：**

;

**备注：**

该指令实际上应该命名为 NOT64；

### 4.7.9. SRA64

**内核：**v1

**格式：**

SRA64 Rd, Rs, Rt

**目的：**

将 64 位源操作数进行算数右移指定位。

**描述：**

将 Rs+1, Rs 中的 64 位源操作数进行算数右移 Rt 寄存器低 6 位指定的量，其中 Rs+1 存放 64 位数高 32 位，Rs 存放 64 位数低 32 位，结果保存在两个连续的 GR 中，其中 Rd+1 存放 64 位结果高 32 位，Rd 存放 64 位结果低 32 位。随着值的移位，最高有效位被符号扩展，最后移出的低位存储在进位标志位中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs 必须为偶数

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1],GR[Rs]}>>>GR[Rt]

CF = {GR[Rs+1],GR[Rs]} $_{GR[Rt]-1}$

**异常：**

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

当 Rt=0 时，CF 设置为 0

### 4.7.10. SL64

**内核：** v1

**格式：**

SL64 Rd, Rs, Rt

**目的：**

将 64 位源操作数进行逻辑左移指定位。

**描述：**

将 Rs+1, Rs 中的 64 位源操作数进行逻辑左移 Rt 寄存器低 6 位指定的量，其中 Rs+1 存放 64 位数高 32 位，Rs 存放 64 位数低 32 位，结果保存在两个连续的 GR 中，其中 Rd+1 存放 64 位结果高 32 位，Rd 存放 64 位结果低 32 位。随着值的移位，低位被 0 填充，最后移出的高位存储在进位标志位中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs 必须为偶数

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1],GR[Rs]}<<GR[Rt]

CF = {GR[Rs+1],GR[Rs]} $^{64\text{-}GR[Rt]}$

**异常：**

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

当 Rt=0 时，CF 设置为 0

### 4.7.11. SRL64

**内核：**v1

**格式：**

SRL64 Rd, Rs, Rt

**目的：**

将 64 位源操作数进行逻辑右移指定位。

**描述：**

将 Rs+1-Rs 中的 64 位源操作数进行逻辑右移 Rt 寄存器低 6 位指定的量，其中 Rs+1 存放 64 位数高 32 位，Rs 存放 64 位数低 32 位，结果保存在两个连续的 GR 中，其中 Rd+1 存放 64 位结果高 32 位，Rd 存放 64 位结果低 32 位。随着值的移位，高位被 0 填充，最后移出的低位存储在进位标志位中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs 必须为偶数

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1],GR[Rs]}>>GR[Rt]

CF = {GR[Rs+1],GR[Rs]} $_{GR[Rt]-1}$

**异常：**

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

当 Rt=0 时，CF 设置为 0

### 4.7.12. SLI

**内核：**v2

**格式：**

SLI Rd，Rs，$imm_6$

**目的：**

将一个 32 位数进行左移（空出的低位补零）得到结果，保存低 32 位，最后移出的高位存储在进位标志位中。

**描述：**

Rd ← Rs << $imm_6$

**限制条件：**

Rd 为 GR，Rs 为 GR，$imm_6$ 为 6 位立即数

**操作：**

temp ← GR[Rs] << $imm_6$

GR[Rd] ← $temp_{31..0}$

CF = GR[Rs] $_{32\text{-}imm6}$

**异常：**

；

**备注：**

；

## 4.7.13. SRAI

**内核：**v2

**格式：**

SRAI Rd， Rt， $imm_6$

**目的：**

将一个 32 位数进行算数右移得到结果，最后移出的低位存储在进位标志位中。

**描述：**

Rd ← Rs >>> $imm_6$

空出的高位补符号位

**限制条件：**

Rd 为 GR， Rs 为 GR，$imm_6$ 为 6 位立即数

**操作：**

GR[Rd]← GR[Rs] >>> $imm_6$

CF = GR[Rs] $_{imm6-1}$

**异常：**

；

**备注：**

；

### 4.7.14. SRLI

**内核：**v2

**格式：**

SRLI Rd， Rs，$imm_6$

**目的：**

将一个 32 位数进行逻辑右移得到结果，最后移出的低位存储在进位标志位中。

**描述：**

Rd ← Rs >> $imm_6$

空出的高位补零

**限制条件：**

Rd 为 GR， Rs 为 GR，$imm_6$ 为 6 位立即数

**操作：**

GR[Rd]← GR[Rs] >> $imm_6$

CF = GR[Rs] $_{imm6-1}$

**异常：**

；

**备注：**

# 4.8. 标量运算指令 – 比特位域指令

## 4.8.1. BFEXT

**内核：** v1

**格式：**

BFEXT Rd，Rs，imm0，imm1

**目的：**

比特域位提取。Rs 中的某一段比特域提取并保存

**描述：**

将 Rs 从 imm0 + imm1 - 1 位置开始（imm0 表示最低位置），宽度为 imm1 的比特域提取出来，高位进行符号扩展后存放在 Rd 中。

比特域提取后符号扩展至 32 位送入 Rd。实现指令功能时：

（1）需要的 mask 为 111111111110….0（比特域位置为 0）

（2）生成 mask ： *~(1 << imm1 – 1)* FFFFFF…FF << imm1

（3）Rs 右移 imm0

（4）根据 Rs[imm1-1]的值决定是 Rs | mask 还是 Rs & ~mask（高位补充符号位）

**限制条件：**

Rd 为 GR，Rs 为 GR，imm0，imm1 均为 5 位立即数

**操作：**

GR[Rd] ← sign_extend (GR[Rs]$_{(imm0 + imm1 – 1) .. imm0}$)

**异常：**

；

**备注：**

；

## 4.8.2. BFEXTU

**内核：**v1

**格式：**

BFEXTU Rd，Rs，imm0，imm1

**目的：**

比特域位提取。Rs 中的某一段比特域提取并保存

**描述：**

将 Rs 从 imm0 + imm1 - 1 位置开始（imm0 表示最低位置），宽度为 imm1 的比特域提取出来，高位进行零扩展后存放在 Rd 中。

比特域提取后符号扩展至 32 位送入 Rd。实现指令功能时：

（1）需要的 mask 为 00000001….1（比特域位置为 1）

（2）生成 mask ：*(1 << imm1 – 1)* ~( FFFFFF…FF << imm1)

（3）Rs 逻辑右移 imm0

（4）提取比特域：Rs & mask（高位为零，低位为提取的比特域）

**限制条件：**

Rd 为 GR，Rs 为 GR，imm0，imm1 均为 5 位立即数

**操作：**

GR[Rd] ← zero_extend (GR[Rs] $_{(imm0 + imm1 - 1) .. imm0}$)

**异常：**

；

**备注：**

；

### 4.8.3. BFST

**内核：** v1

**格式：**

BFST Rd，Rs，imm0，imm1

**目的：**

比特域位设置。Rd 中的某一段比特域设置成给定值 Rs

**描述：**

将 Rd 从 imm0 + imm1 - 1 位置开始（imm0 表示最低位置），宽度为 imm1 的比特域设置成 Rs 的低 imm1 位。实现指令功能时：

（1）需要的 mask 为 111111111110….0（比特域位置为 0）

（2）生成 mask ： *~(1 << imm1 – 1)* FFFFFF…FF << imm1

（3）目标域清空：Rd & ((mask << imm0) | ~mask2)，（目标域置 0，别忘了低位要保持原来的值）

（4）Rs 移位至目标位置：(Rs & ~mask) << imm0

（5）目标域设置：Rd | Rs

**限制条件：**

Rd 为 GR，Rs 为 GR，imm0，imm1 均为 5 位立即数

**操作：**

GR[Rd] $_{(imm0 + imm1 – 1) .. imm0}$ ← GR[Rs] $_{(imm1 – 1)..0}$

**异常：**

；

**备注：**

；

### 4.8.4. BST

**内核：**v1

**格式：**

BST Rd，$imm_5$

**目的：**

设置比特位。Rd 第 imm 位置 1

**描述：**

$Rd_{imm}$ = 1 （最低位定义为第 0 位）

**限制条件：**

Rd 为 GR，imm 为 5 位立即数

**操作：**

$GR[Rd]_{imm}$ ← 1

**异常：**

；

**备注：**

；

### 4.8.5. BCLR

**内核：**v1

**格式：**

BCLR Rd，$imm_5$

**目的：**

清空比特位设置。Rd 第 imm 位置 0

**描述：**

$Rd_{imm}$ = 0 （最低位定义为第 0 位）

**限制条件：**

Rd 为 GR，imm 为 5 位立即数

**操作：**

$GR[Rd]_{imm}$ ← 0

**异常：**

；

**备注：**

；

# 5. FPU 指令

## 5.1. 运算指令

### 5.1.1. FSMUL

**内核：**v1

**格式：**

FSMUL Rd，Rs，Rt

**目的：**

标量单精度浮点数乘法

**描述：**

Rs 的 32 位单精度浮点数为乘数，Rt 的 32 位单精度浮点数为被乘数，按 IEEE 754 标准执行单精度浮点乘法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] * GR[Rt]

**异常：**

### 5.1.2. FSMAC

**内核：**v1

**格式：**

FSMAC Rd，Rs，Rt

**目的：**

标量单精度浮点数乘累加

**描述：**

Rs 的 32 位单精度浮点数为乘数，Rt 的 32 位单精度浮点数为被乘数，Rd 的 32 位单精度浮点数为累加数，按 IEEE 754 标准执行单精度浮点乘法后再加上累加数，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] * GR[Rt] + GR[Rd]

**异常：**

**备注：**

### 5.1.3. FSEINV

**内核：** v1

**格式：**

FSEINV Rd，Rs

**目的：**

标量单精度浮点数倒数逼近

**描述：**

对 Rs 的 32 位单精度浮点数执行倒数查表操作。对于尾数部分，基于 ROM 构成一张 256 项、每项 8bit 的查找表。取 Rs [22:15]作为索引，根据索引值到 ROM 查找表中读出相应的 8bit 数据作为结果尾数的高 8 位，在其后添加 15 个 0 作为尾数中间结果。

对于指数部分，将 Rs 的指数部分进行右移一位操作，得到指数的中间结果。

将符号位、尾数、指数部分进行拼接和规格化操作，得到 32bit 结果存放在 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

W ← $GR[Rs]_{22..15}$

Temp_fraction ← $\{ROM_{W+7..W}, 15'b0\}$

Temp_exp ← $8'd255 + GR[Rs]_{30..23}$

Temp_signal ← $GR[Rs]_{31}$

GR[Rd] ← normalization{ Temp_fraction , Temp_exp, Temp_signal}

**异常：**

**备注：**

### 5.1.4. FSEISQRT

**内核：** v1

**格式：**

FSEISQRT Rd，Rs

**目的：**

标量单精度浮点数倒数平方根逼近

**描述：**

对 Rs 的 32 位单精度浮点数执行倒数平方根查表操作。根据指数部分奇偶性对尾数部分进行移位，指数为偶数时尾数不动，指数为奇数时尾数向右移动一位，指数加一。

对于尾数部分，基于 ROM 构成一张 256 项、每项 8bit 的查找表。取 Rs [22:15]作为索引，根据索引值到 ROM 查找表中读出相应的 8bit 数据作为结果尾数的高 8 位，在其后添加 15 个 0 作为尾数中间结果。

对于指数部分，将 Rs 的指数部分进行取反操作，得到指数的中间结果。

将符号位、尾数、指数部分进行拼接和规格化操作，得到 32bit 结果存放在 Rd 中

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

W ← GR[Rs]$_{23}$ ? GR[Rs]$_{22..15}$ : GR[Rs]$_{22..15}$ >> 1

Temp_fraction ← {ROM$_{W+7..W}$, 15'b0}

Temp_exp ← ((GR[Rs]$_{30..23}$ +GR[Rs]$_{23}$ ? 0 : 1)- 8'd255) >> 1 + 8'd255

Temp_signal ← GR[Rs]$_{31}$

GR[Rd] ← normalization{ Temp_fraction , Temp_exp, Temp_signal}

**异常：**

**备注：**

### 5.1.5. FSDIV

**内核：**v1

**格式：**

FSDIV Rd，Rs，Rt

**目的：**

标量单精度浮点数除法

**描述：**

Rs 的 32 位单精度浮点数为被除数，Rt 的 32 位单精度浮点数为除数，按 IEEE 754 标准执行单精度浮点除法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] / GR[Rt]

**异常：**

**备注：**

### 5.1.6. FSSQRT

**内核：**v1

**格式：**

FSSQRT Rd，Rs

**目的：**

标量单精度浮点数开方根

**描述：**

对 Rs 的 32 位单精度浮点数执行开方根运算，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← SquareRoot(GR[Rs])

**异常：**

**备注：**

### 5.1.7. FSADD

**内核：**v1

**格式：**

FSADD Rd，Rs，Rt

**目的：**

标量浮点单精度加法

**描述：**

Rs 和 Rd 的 32 位单精度浮点数分别作为两个源操作数，按 IEEE 754 标准执行单精度浮点加法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] + GR[Rt]

**异常：**

**备注：**

；

### 5.1.8. FSSUB

**内核：**v1

**格式：**

FSSUB Rd，Rs，Rt

**目的：**

标量浮点单精度减法

**描述：**

Rs 的 32 位单精度浮点数为被减数，Rt 的 32 位单精度浮点数为减数，按 IEEE 754 标准执行单精度浮点减法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] - GR[Rt]

**异常：**

**备注：**

；

### 5.1.9. FSABS

**内核：**v1

**格式：**

FSABS Rd，Rs

**目的：**

标量浮点单精度取绝对值

**描述：**

对 Rs 的 32 位单精度浮点数执行取绝对值操作，结果写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← ABS(GR[Rs])

**异常：**

**备注：**

；

### 5.1.10. FSMAX

**内核：**v1

**格式：**

FSMAX Rd，Rs，Rt

**目的：**

标量浮点单精度数据取较大值

**描述：**

Rs 的 32 位单精度浮点数值和 Rt 的 32 位单精度浮点数值分别作为两个源操作数，取两者中较大的一个写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] > GR[Rt] ? GR[Rs] : GR[Rt]

**异常：**

**备注：**

### 5.1.11. FSMIN

**内核：**v1

**格式：**

FSMIN Rd，Rs，Rt

**目的：**

标量浮点单精度数据取较小值

**描述：**

Rs 的 32 位单精度浮点数值和 Rt 的 32 位单精度浮点数值分别作为两个源操作数，取两者中较小的一个写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

GR[Rd] ← GR[Rs] < GR[Rt] ? GR[Rs] : GR[Rt]

**异常：**

**备注：**

### 5.1.12. FDMUL

**内核：**v3

**格式：**

FDMUL Rd，Rs，Rt

**目的：**

标量双精度浮点数乘法

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数为乘数，{Rt+1, Rt}的 64 位双精度浮点数为被乘数，按 IEEE 754 标准执行双精度浮点乘法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} * {GR[Rt+1], GR[Rt]}

**异常：**

### 5.1.13. FDMAC

**内核：**v3

**格式：**

FDMAC Rd，Rs，Rt

**目的：**

标量双精度浮点数乘累加

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数为乘数，{Rt+1, Rt}的 64 位双精度浮点数为被乘数，{Rd+1, Rd}的 64 位双精度浮点数为累加数，按 IEEE 754 标准执行双精度浮点乘法后再加上累加数，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} * {GR[Rt+1], GR[Rt]} + {GR[Rd+1], GR[Rd]}

**异常：**

**备注：**

### 5.1.14. FDADD

**内核：**v3

**格式：**

FDADD Rd，Rs，Rt

**目的：**

标量浮点双精度加法

**描述：**

{Rs+1, Rs}和{Rt+1, Rt}的 64 位双精度浮点数分别作为两个源操作数，按 IEEE 754 标准执行双精度浮点加法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} + {GR[Rt+1], GR[Rt]}

**异常：**

**备注：**

；

### 5.1.15. FDSUB

**内核：**v3

**格式：**

FDSUB Rd，Rs，Rt

**目的：**

标量浮点双精度减法

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数为被减数，{Rt+1, Rt}的 64 位双精度浮点数为减数，按 IEEE 754 标准执行双精度浮点减法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} - {GR[Rt+1], GR[Rt]}

**异常：**

**备注：**

；

### 5.1.16. FDABS

**内核：**v3

**格式：**

FDABS　Rd，Rs

**目的：**

标量浮点单精度取绝对值

**描述：**

对{Rs+1, Rs}的 64 位双精度浮点数执行取绝对值操作，结果写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← ABS({GR[Rs+1], GR[Rs]})

**异常：**

**备注：**

；

### 5.1.17. FDMAX

**内核：**v3

**格式：**

FDMAX Rd，Rs，Rt

**目的：**

标量浮点双精度数据取较大值

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数值和{Rt+1, Rt}的 64 位双精度浮点数值分别作为两个源操作数，取两者中较大的一个写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} > {GR[Rt+1], GR[Rt]} ? {GR[Rs+1], GR[Rs]} : {GR[Rt+1], GR[Rt]}

**异常：**

**备注：**

### 5.1.18. FDMIN

**内核：**v3

**格式：**

FDMIN Rd，Rs，Rt

**目的：**

标量浮点双精度数据取较小值

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数值和{Rt+1, Rt}的 64 位双精度浮点数值分别作为两个源操作数，取两者中较小的一个写入{Rd+1, Rd}。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← {GR[Rs+1], GR[Rs]} < {GR[Rt+1], GR[Rt]} ? {GR[Rs+1], GR[Rs]} : {GR[Rt+1], GR[Rt]}

**异常：**

**备注：**

# 5.2. 比较指令

## 5.2.1. FSEQ

**内核：** v1

**格式：**

FSEQ Rs，Rt

**目的：**

标量浮点单精度数据比较（相等）

**描述：**

若两个单精度浮点数相等，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] == GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.2. FSGT

**内核：**v1

**格式：**

FSGT Rs，Rt

**目的：**

标量浮点单精度数据比较（大于）

**描述：**

若 Rs 的 32 位单精度浮点数值大于 Rt 的 32 位单精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] > GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.3. FSLT

**内核：**v1

**格式：**

FSLT Rs，Rt

**目的：**

标量浮点单精度数据比较（小于）

**描述：**

若 Rs 的 32 位双精度浮点数值小于 Rt 的 32 位单精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] < GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.4. FSGE

**内核：**v1

**格式：**

FSGE Rs，Rt

**目的：**

标量浮点单精度数据比较（大于等于）

**描述：**

若 Rs 的 32 位单精度浮点数值大于等于 Rt 的 32 位单精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] >= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.5. FSLE

**内核：**v1

**格式：**

FSLE Rs，Rt

**目的：**

标量浮点单精度数据比较（小于等于）

**描述：**

若 Rs 的 32 位单精度浮点数值小于等于 Rt 的 32 位单精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF GR[Rs] <= GR[Rt]
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.6. FDEQ

**内核：**v3

**格式：**

FDEQ Rs，Rt

**目的：**

标量浮点双精度数据比较（相等）

**描述：**

若两个双精度浮点数相等，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF {GR[Rs+1], GR[Rs]} == {GR[Rt+1], GR[Rt]}
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.7. FDGT

**内核：**v3

**格式：**

FDGT Rs， Rt

**目的：**

标量浮点双精度数据比较（大于）

**描述：**

若{Rs+1, Rs}的 64 位双精度浮点数值大于{Rt+1, Rt}的 64 位双精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF {GR[Rs+1], GR[Rs]} > {GR[Rt+1], GR[Rt]}
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

## 5.2.8. FDLT

**内核：**v3

**格式：**

FDLT Rs， Rt

**目的：**

标量浮点双精度数据比较（小于）

**描述：**

若{Rs+1, Rs}的 64 位双精度浮点数值小于{Rt+1, Rt}的 64 位双精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR， Rt 为 GR

**操作：**

```
IF {GR[Rs+1], GR[Rs]} < {GR[Rt+1], GR[Rt]}
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.9. FDGE

**内核：**v3

**格式：**

FDGE Rs，Rt

**目的：**

标量浮点双精度数据比较（大于等于）

**描述：**

若{Rs+1, Rs}的 64 位双精度浮点数值大于等于{Rt+1, Rt}的 64 位双精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF {GR[Rs+1], GR[Rs]} >= {GR[Rt+1], GR[Rt]}
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

### 5.2.10. FDLE

**内核：**v3

**格式：**

FDLE Rs， Rt

**目的：**

标量浮点双精度数据比较（小于等于）

**描述：**

若{Rs+1, Rs}的 64 位双精度浮点数值小于等于{Rt+1, Rt}的 64 位双精度浮点数值，则相应 CON 设置为 1，否则设置为 0

**限制条件：**

Rs 为 GR，Rt 为 GR

**操作：**

```
IF {GR[Rs+1], GR[Rs]} <= {GR[Rt+1], GR[Rt]}
    CON = 1;
ELSE
    CON = 0;
```

**异常：**

**备注：**

# 5.3. 类型转换指令

## 5.3.1. FCVTSF

**内核：** v1

**格式：**

FCVTSF Rd，Rs

**目的：**

标量浮点单精度转有符号整型

**描述：**

Rs 的 32 位单精度浮点数值转换为 32 位有符号整型数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← SingleToSignedInt (GR[Rs])

**异常：**

**备注：**

### 5.3.2. FCVTFS

**内核：**v1

**格式：**

FCVTFS Rd，Rs

**目的：**

标量有符号整型转浮点单精度

**描述：**

Rs 的 32 位有符号整型数值转换为 32 位单精度浮点数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← SignedIntToSingle (GR[Rs])

**异常：**

**备注：**

### 5.3.3. FCVTSU

**内核：**v1

**格式：**

FCVTSU Rd， Rs

**目的：**

标量浮点单精度转无符号整型

**描述：**

Rs 的 32 位单精度浮点数值转换为 32 位无符号整型数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← SingleToUnsignedInt (GR[Rs])

**异常：**

**备注：**

### 5.3.4. FCVTUS

**内核：**v1

**格式：**

FCVTUS Rd， Rs

**目的：**

标量无符号整型转浮点单精度

**描述：**

Rs 的 32 位无符号整型数值转换为 32 位单精度浮点数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← UnsignedIntToSingle (GR[Rs])

**异常：**

**备注：**

### 5.3.5. FCVTDF

**内核：**v3

**格式：**

FCVTDF Rd， Rs

**目的：**

标量浮点双精度转有符号整型

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数值转换为 32 位有符号整型数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← DoubleToSignedInt ({GR[Rs+1], GR[Rs]))

**异常：**

**备注：**

### 5.3.6. FCVTFD

**内核：**v3

**格式：**

FCVTFD Rd， Rs

**目的：**

标量有符号整型转浮点双精度

**描述：**

Rs 的 32 位有符号整型数值转换为 64 位双精度浮点数后写入{Rd+1, Rd}中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← SignedIntToDouble (GR[Rs])

**异常：**

**备注：**

### 5.3.7. FCVTDU

**内核：**v3

**格式：**

FCVTDU Rd， Rs

**目的：**

标量浮点双精度转无符号整型

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数值转换为 32 位无符号整型数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← DoubleToUnsignedInt ({GR[Rs+1], GR[Rs]})

**异常：**

**备注：**

### 5.3.8. FCVTUD

**内核：**v3

**格式：**

FCVTUD Rd， Rs

**目的：**

标量无符号整型转浮点双精度

**描述：**

Rs 的 32 位无符号整型数值转换为 64 位双精度浮点数后写入{Rd+1, Rd}中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← UnsignedIntToDouble (GR[Rs])

**异常：**

**备注：**

### 5.3.9. FCVTDS

**内核：**v3

**格式：**

FCVTDS Rd， Rs

**目的：**

标量浮点双精度转浮点单精度

**描述：**

{Rs+1, Rs}的 64 位双精度浮点数值转换为 32 位单精度浮点数后写入 Rd 中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← DoubleToSingle ({GR[Rs+1], GR[Rs]})

**异常：**

**备注：**

### 5.3.10. FCVTSD

**内核：**v3

**格式：**

FCVTSD Rd， Rs

**目的：**

标量浮点单精度转浮点双精度

**描述：**

Rs 的 32 位单精度浮点数值转换为 64 位双精度浮点数后写入{Rd+1, Rd}中。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

{GR[Rd+1], GR[Rd]} ← SingleToDouble (GR[Rs])

**异常：**

**备注：**

# 6. TMU 指令

## 6.1. MPY2PIF32

**内核：**v2

**格式：**

MPY2PIF32 Rd，Rs

**目的：**

32 位浮点乘以 2pi

**描述：**

Rs 的 32 位单精度浮点数为乘数，2pi = 6.28318530718 = 1.570796326795*2^2 为被乘数，按 IEEE 754 标准执行单精度浮点乘法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← GR[Rs] * 2pi

**异常：**

根据结果是否溢出设置 LVF 和 LUF

**备注：**

由 FPU 乘法指令实现

## 6.2. DIV2PIF32

**内核：**v2

**格式：**

DIV2PIF32　Rd，Rs

**目的：**

32 位浮点除以 2pi

**描述：**

Rs 的 32 位单精度浮点数为被除数，2pi = 6.28318530718 = 1.570796326795*2^2 为除数，按 IEEE 754 标准执行单精度浮点除法，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

GR[Rd] ← GR[Rs] / 2pi

**异常：**

根据结果是否溢出设置 LVF 和 LUF

**备注：**

由 FPU 乘法指令实现

## 6.3. SINPUF32.v1

**内核：** v1

**格式：**

SINPUF32　Rd，Rs

**目的：**

标量浮点单精度正弦

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做正弦函数处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR , GR[Rs]∈[0,1]

**操作：**

GR[Rd] ← SIN( $\frac{\pi}{2}$ * GR[Rs])

**异常：**

**备注：**

## 6.4. SINPUF32.v2

**内核：**v2

**格式：**

SINPUF32　Rd，Rs

**目的：**

标量浮点单精度正弦

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做正弦函数处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR ，GR[Rs]为 IEEE754 单精度表示的规格化数

**操作：**

GR[Rd] ← SIN( 2π ・ GR[Rs])

**异常：**

**备注：**

# 6.5. COSPUF32

**内核：**v2

**格式：**

COSPUF32　Rd，Rs

**目的：**

标量浮点单精度余弦

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做余弦函数处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR ，GR[Rs]为 IEEE754 单精度表示的规格化数

**操作：**

GR[Rd] ← COS( 2π • GR[Rs])

**异常：**

**备注：**

## 6.6. ATANPUF32.v1

**内核：** v1

**格式：**

ATANPUF32　Rd，Rs

**目的：**

标量浮点单精度反正切

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做反正切函数处理，再对其单位化处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR , GR[Rs]∈[0,1]

**操作：**

GR[Rd] ← ATAN(GR[Rs])/2pi

**异常：**

**备注：**

# 6.7. ATANPUF32.v2

**内核：**v2

**格式：**

ATANPUF32　Rd，Rs

**目的：**

标量浮点单精度反正切

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做反正切函数处理，再对其单位化处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd。

**限制条件：**

Rd 为 GR，Rs 为 GR , GR[Rs] ∈ [-1,1]

**操作：**

GR[Rd] ← ATAN(GR[Rs])/2pi

**异常：**

根据结果是否溢出设置 LVF

**备注：**

# 6.8. QUADF

**内核：** v1

**格式：**

QUADF Rd， Rs， Rt

Rs,Rt 为源操作数寄存器，Rd 为目的寄存器。

**功能：**

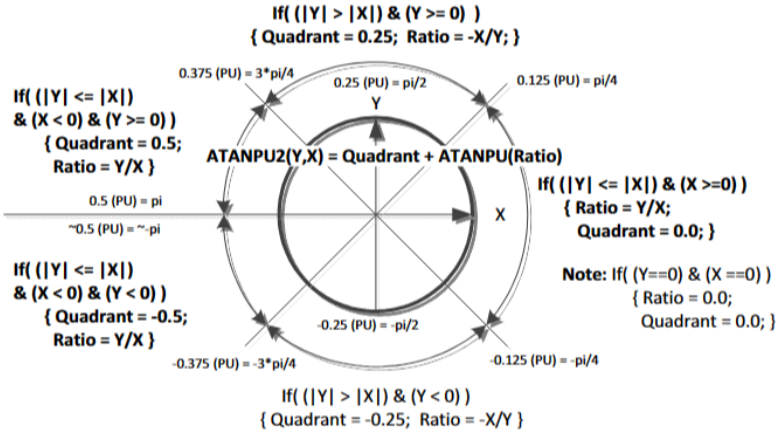
根据两个 32bit 单精度浮点输入值计算得到 ratio 值和 quadrant 值。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

以 y=x 和 y=-x 将空间分为四个象限并定义相应的象限值（0，±0.25，±0.5），输出 Rs,Rt 所处的象限值（quadrant）并输出 Rs,Rt 的比值（ratio）。将 ratio 值存入 GR[Rd],将 quadrant 值存入 GR[Rd+1]。




若|Rs|>|Rt|（|Y|>|X|）&|Rs|>=0（Y>=0），则 Rd = -Rt/Rs（-X/Y）, Rd+1 = 0.25;

若|Rs|>|Rt|（|Y|>|X|）&|Rs|<0（Y<0），则 Rd = -Rt/Rs（-X/Y）, Rd+1 = -0.25;

若|Rs|<=|Rt|（|Y|<=|X|）&|Rt|>=0（X>=0），则 Rd = Rs/Rt（Y/X）, Rd+1 = 0.0;

若|Rs|<=|Rt|（|Y|<=|X|）&|Rs|>=0（X<0& Y>=0），则 Rd = -Rt/Rs（Y/X）, Rd+1 = 0.5;

若|Rs|<=|Rt|（|Y|<=|X|）&|Rs|>=0（X<0& Y<0），则 Rd = -Rt/Rs（Y/X）, Rd+1 = -0.5;

**备注：**

v2 内核中，根据结果是否溢出设置 LVF 和 LUF

## 6.9. EXPPUF32

内核：v3

格式：

```
EXPPUF32  Rd，Rs
```

目的：

描述：

限制条件：

操作：

异常：

备注：

# 6.10. LOGPUF32

**内核：**v3

**格式：**

LOGPUF32　Rd，Rs

**目的：**

标量浮点单精度求对数(以二为底)。

**描述：**

Rs 的 32 位单精度浮点数为输入，取小数部分做对数函数处理，再对其单位化处理，并对结果进行规格化处理（包含舍入操作，根据移出或截断的最高位为 0 或者 1 来判断是否+1）后写入 Rd

**限制条件：**

**操作：**

**异常：**

**备注：**

# 7. VCU 指令

## 7.1. VSTATUS 寄存器

### 7.1.1. 配置方法

通过 spr 读写，地址为 0x007F_0B00。

### 7.1.2. 寄存器定义

| Filed | Name | Initial | R/W | 功能 |
|---|---|---|---|---|
| [31:14] | reserved | 18'd0 | R | 保留 |
| [13] | OVFI | 1'b0 | R/W | 虚数溢出位 |
| [12] | OVFR | 1'b0 | R/W | 实数溢出位 |
| [11] | RND | 1'b0 | R/W | 舍入位 |
| [10] | SAT_EN | 1'b0 | R/W | 饱和压缩使能位 |
| [9:5] | SHIFTL | 5'b0 | R/W | 左移位数 |
| [4:0] | SHIFTR | 5'b0 | R/W | 右移位数 |

# 7.2. 复数指令

## 7.2.1. VCADD

**内核：** v1

**格式：**

VCADD Rd, Rs, Rt

**目的：**

32 位整型复数加法

**描述：**

第一个操作数的虚部存储在 GR[Rs]，实部存储在 GR[Rs+1]。第二个操作数的虚部存储在 GR[Rt]，实部存储在 GR[Rt+1]。

将两个操作数的实部、虚部分别相加，计算结果的虚部存入 GR[Rd]，实部存入 GR[Rd+1]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs、Rt 必须为偶数

**操作：**

```
GR[Rd + 1] = GR[Rs + 1] + GR[Rt + 1];
GR[Rd] = GR[Rs] + GR[Rt];
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

根据实部运算结果（GR[Rd + 1]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd]）设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-

2147483648）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.2.2. VCSUB

**内核：** v1

**格式：**

VCSUB Rd, Rs, Rt

**目的：**

32 位整型复数减法

**描述：**

第一个操作数（被减数）的虚部存储在 GR[Rs]，实部存储在 GR[Rs+1]。第二个操作数的虚部存储在 GR[Rt]，实部存储在 GR[Rt+1]。

将两个操作数的实部、虚部分别相减，计算结果的虚部存入 GR[Rd]，实部存入 GR[Rd+1]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs、Rt 必须为偶数

**操作：**

```
GR[Rd + 1] = GR[Rs + 1] - GR[Rt + 1] ;
GR[Rd] = GR[Rs] - GR[Rt];
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

根据实部运算结果（GR[Rd + 1]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd]）设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-2147483648）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.2.3. VCDA16

**内核：**v1

**格式：**

VCDA16 Rd, Rs, Rt

**目的：**

16 位有符号整型复数与 32 位有符号整型复数加法，结果保留低 16 位

**描述：**

第一个操作数（16 位）的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。第二个操作数（32 位）的虚部存储在 GR[Rt]，实部存储在 GR[Rt+1]。

将两个操作数的实部、虚部分别相加，计算结果保留低 16 位，虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rt 必须为偶数

**操作：**

```
temp1 = sign_extend(GR[Rs][31:16]) + GR[Rt + 1];
temp2 = sign_extend(GR[Rs][15:0]) + GR[Rt];
if (SAT == 1)
{
    GR[Rd][31:16] = sat16(temp1);
    GR[Rd][15:0] = sat16(temp2);
}
else
{
    GR[Rd][31:16] = temp1[15:0];
    GR[Rd][15:0] = temp2[15:0];
}
```

**异常：**

**备注：**

第一个操作数的实部（GR[Rs][31:16]）和虚部（GR[Rs][15:0]）在运算前会符号扩展至 32 位。

根据实部运算结果（GR[Rd][31:16]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd][15:0]）设置 OVFI 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果（低 16 位）按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFF（32767）；若发生下溢，则将对应结果压缩为 0x8000（-32768）。

### 7.2.4. VCDS16

**内核：**v1

**格式：**

VCDS16 Rd, Rs, Rt

**目的：**

16 位有符号整型复数与 32 位有符号整型复数减法，结果保留低 16 位

**描述：**

第一个操作数（被减数，16 位）的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。第二个操作数（32 位）的虚部存储在 GR[Rt]，实部存储在 GR[Rt+1]。

将两个操作数的实部、虚部分别相减，计算结果保留低 16 位，虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rt 必须为偶数

**操作：**

```
temp1 = sign_extend(GR[Rs][31:16]) - GR[Rt + 1];
temp2 = sign_extend(GR[Rs][15:0]) - GR[Rt];
if (SAT == 1)
{
    GR[Rd][31:16] = sat16(temp1);
    GR[Rd][15:0] = sat16(temp2);
}
else
{
    GR[Rd][31:16] = temp1[15:0];
    GR[Rd][15:0] = temp2[15:0];
}
```

**异常：**

**备注：**

第一个操作数的实部（GR[Rs][31:16]）和虚部（GR[Rs][15:0]）在运算前会符号扩展至 32 位。

根据实部运算结果（GR[Rd][31:16]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd][15:0]）设置 OVFI 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果（低 16 位）按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFF（32767）；若发生下溢，则将对应结果压缩为 0x8000（-32768）。

### 7.2.5. VNEG

**内核：**v1

**格式：**

VNEG Rd, Rs

**目的：**

32 位有符号整数取负

**描述：**

对 GR[Rs]中存放的 32 位有符号整数取负（取反后＋1），结果存到 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

```
if (Rs == 0x80000000)
    if (SAT == 1)
        Rd = 0x7FFFFFFF;
    else
        Rd = 0x80000000;
else
    Rd = -Rs;
```

**异常：**

**备注：**

若操作数为 0x80000000（-2147483648），则设置 OVFR 溢出标志位为 1，否则置 0。

且当 SAT 为 1 时，结果为 0x7FFFFFFF（2147483647），否则保留原数 0x80000000。

## 7.2.6. VCMPY

**内核：** v1

**格式：**

VCMPY Rd, Rs

**目的：**

16 位有符号整型复数乘法，结果为 32 位

**描述：**

第一个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。第二个操作数的虚部存储在 GR[Rs+1][15:0]，实部存储在 GR[Rs+1][31:16]。

将两个操作数按复数乘法运算规则进行相乘，计算结果的虚部存入 GR[Rd]，实部存入 GR[Rd+1]。

**限制条件：**

Rd 为 GR，Rs 为 GR，且 Rs、Rd 必须为偶数

**操作：**

```
GR[Rd + 1] = GR[Rs][31:16] * GR[Rs+1][31:16] - GR[Rs][15:0] * GR[Rs+1][15:0];
GR[Rd] = GR[Rs][31:16] * GR[Rs+1][15:0] + GR[Rs][15:0] * GR[Rs+1][31:16];
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

根据实部运算结果（GR[Rd + 1]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd]）设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-2147483648）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

## 7.2.7. VCMPYAC

**内核：** v1

**格式：**

VCMPYAC Rd, Rs, Rt

**目的：**

16 位有符号整型复数乘积累加，结果为 32 位

**描述：**

乘法的第一个操作数的虚部存储在 GR[Rs+1][15:0]，实部存储在 GR[Rs+1][31:16]，第二个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。累加数的虚部存储在 GR[Rt]，实部存储在 GR[Rt+1]。

首先执行乘法操作，分别对乘法的第一第二操作数做复数乘法，再将乘法结果的实部和虚部分别与累加数的实部和虚部相加，计算结果实部存入 GR[Rd+1]，虚部存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rs、Rd、Rt 必须为偶数

**操作：**

```
temp1 = GR[Rs][31:16] * GR[Rs+1][31:16] - GR[Rs][15:0] * GR[Rs+1][15:0];
temp2 = GR[Rs][31:16] * GR[Rs+1][15:0] + GR[Rs][15:0] * GR[Rs+1][31:16];
GR[Rd + 1] = GR[Rt + 1] + temp1;
GR[Rd] = GR[Rt] + temp2;
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

乘累加运算后，根据实部运算结果（GR[Rd＋1]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd]）设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。

若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-2147483648）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

# 7.3. CRC 指令

## 7.3.1. VCRC8LL

**内核：** v1

**格式：**

VCRC8LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC8 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC8 校验码，CRC 校验多项式为 0x07。计算初始值取 Rt 寄存器中低 8 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 8 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 24'b0, CRC8 (GR[Rs][7:0], GR[Rt][7:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.2. VCRC8LH

**内核：** v1

**格式：**

VCRC8LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC8 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC8 校验码，CRC 校验多项式为 0x07。计算初始值取 Rt 寄存器中低 8 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 8 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 24'b0, CRC8 (GR[Rs][15:8], GR[Rt][7:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.3. VCRC8HL

**内核：** v1

**格式：**

VCRC8HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC8 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC8 校验码，CRC 校验多项式为 0x07。计算初始值取 Rt 寄存器中低 8 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 8 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 24'b0, CRC8 (GR[Rs][23:16], GR[Rt][7:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.4. VCRC8HH

**内核：** v1

**格式：**

VCRC8HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC8 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC8 校验码，CRC 校验多项式为 0x07。计算初始值取 Rt 寄存器中低 8 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 8 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 24'b0, CRC8 (GR[Rs][31:24], GR[Rt][7:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.5. VCRC16P1LL

**内核：** v1

**格式：**

VCRC16P1LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x8005。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][7:0], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.6. VCRC16P1LH

**内核：** v1

**格式：**

VCRC16P1LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x8005。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][15:8], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.7. VCRC16P1HL

**内核：** v1

**格式：**

VCRC16P1HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x8005。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][23:16], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.8. VCRC16P1HH

**内核：** v1

**格式：**

VCRC16P1HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x8005。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][31:24], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.9. VCRC16P2LL

**内核：** v1

**格式：**

VCRC16P2LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x1021。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][7:0], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.10. VCRC16P2LH

**内核：** v1

**格式：**

VCRC16P2LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x1021。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][15:8], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.11. VCRC16P2HL

**内核：** v1

**格式：**

VCRC16P2HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x1021。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][23:16], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.12. VCRC16P2HH

**内核：** v1

**格式：**

VCRC16P2HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC16 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC16 校验码，CRC 校验多项式为 0x1021。计算初始值取 Rt 寄存器中低 16 位所存放的数据。计算结果存放在 GR[Rd]寄存器的低 16 位中，Rd 寄存器的高位以 0 填充。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← { 16'b0, CRC16 (GR[Rs][31:24], GR[Rt][15:0]) };
```

**异常：**

;

**备注：**

;

### 7.3.13. VCRC32LL

**内核：** v1

**格式：**

VCRC32LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x04C11DB7。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][7:0], GR[Rt])
```

**异常：**

;

**备注：**

;

### 7.3.14. VCRC32LH

**内核：** v1

**格式：**

VCRC32LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x04C11DB7。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][15:8], GR[Rt])
```

**异常：**

;

**备注：**

;

### 7.3.15. VCRC32HL

**内核：** v1

**格式：**

VCRC32HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x04C11DB7。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][23:16], GR[Rt])
```

**异常：**

;

**备注：**

;

### 7.3.16. VCRC32HH

**内核：** v1

**格式：**

VCRC32HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x04C11DB7。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][31:24], GR[Rt])
```

**异常：**

；

**备注：**

；

# 7.4. Viterbi 指令

## 7.4.1. VITBM2

**内核：** v1

**格式：**

VITBM2 Rd, Rs

**目的：**

计算码率为 1/2 时的分支度量值。

**描述：**

将 Rs 的高 16 和低 16 位分别相加和相减，值分别存入 Rd 的高 16 和低 16 位。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

```
GR[Rd][15:0]= GR[Rs][15:0]+ GR[Rs][31:16];
GR[Rd] [31:16]= GR[Rs] [15:0]- GR[Rs] [31:16];
if (SAT == 1)
{
    GR[Rd] [15:0] = sat16(GR[Rd] [15:0]);
    GR[Rd] [31:16] = sat16(GR[Rd] [31:16]);
}
```

**异常：**

;

**备注：**

SAT 操作：两数相加设置 17bit 的中间量 tmp，对 tmp 进行饱和压缩运算，当 tmp>32767 时，压缩为 16'h3fff；当 tmp<-32768 时，压缩为 16'h8000；

同时进行溢出的判断，并会置 VSTATUS 中的实数溢出标志位 OVFR，（发生上溢或者下溢都会置位）

### 7.4.2. VITBM3

**内核：** v1

**格式：**

VITBM3 Rd，Rs，Rt

**目的：**

计算码率为 1/3 时的分支度量值。

**描述：**

将 GR[Rs], GR[Rs+1], GR[Rt]的低 16 位作为该计算的三个输入。计算得出 4 个 16bit 分支度量值分别存入 GR[Rd], GR[Rd+1]中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd、Rs 必须为偶数

**操作：**

```
GR[Rd][15:0]= GR[Rs][15:0]+ GR[Rs+1][15:0]+ GR[Rt][15:0];
GR[Rd][31:16]= GR[Rs][15:0]+ GR[Rs+1][15:0]- GR[Rt][15:0];
GR[Rd+1][15:0]= GR[Rs][15:0]- GR[Rs+1][15:0]+ GR[Rt][15:0];
GR[Rd+1][31:16]= GR[Rs][15:0]- GR[Rs+1][15:0]- GR[Rt][15:0];
if (SAT == 1)
{
   GR[Rd] [15:0] = sat16(GR[Rd] [15:0]);
   GR[Rd] [31:16] = sat16(GR[Rd] [31:16]);
}
```

**异常：**

；

**备注：**

SAT 操作：两数相加设置 17bit 的中间量 tmp，对 tmp 进行饱和压缩运算，当 tmp>32767 时，压缩为 16’h3fff；当 tmp<-32768 时，压缩为 16’h8000；

同时进行溢出的判断，并会置 VSTATUS 中的实数溢出标志位 OVFR，（发生上溢或者下溢都会置位）Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.3. VITDHAS

**内核：** v1

**格式：**

VITDHADDSUB Rd，Rs，Rt

**目的：**

计算 4 个路径度量。

**描述：**

Rs 的低 16 位和高 16 位分别是状态度量，Rt 高 16 位是分支度量

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数

**操作：**

```
GR[Rd][15:0] = GR[Rs][15:0] + GR[Rt][31:16];
GR[Rd][31:16]= GR[Rs][31:16]- GR[Rt][31:16];
GR[Rd+1][15:0]= GR[Rs][15:0]- GR[Rt][31:16];
GR[Rd+1][31:16]= GR[Rs][31:16]+ GR[Rt][31:16];
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.4. VITDHSA

**内核：** v1

**格式：**

VITDHSUBADD Rd，Rs，Rt

**目的：**

计算 4 个路径度量。

**描述：**

Rs 的低 16 位和高 16 位分别是状态度量 state metric0 和 state metric1，Rt 高 16 位是分支度量 Branch metric0。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数

**操作：**

```
GR[Rd][15:0]= GR[Rs][15:0]- GR[Rt][31:16];
GR[Rd][31:16]= GR[Rs][31:16]+ GR[Rt][31:16]
GR[Rd+1][15:0]= GR[Rs][15:0]+ GR[Rt][31:16]
GR[Rd+1][31:16]= GR[Rs][31:16]- GR[Rt][31:16]
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.5. VITDLAS

**内核：** v1

**格式：**

VITDLADDSUB Rd，Rs，Rt

**目的：**

计算出 4 个路径度量.

**描述：**

Rs 的低 16 位和高 16 位分别是状态度量 state metric 0 和 state metric1，Rt 低 16 位是分支度量 Branch metric 0

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数

**操作：**

```
GR[Rd][15:0]= GR[Rs][15:0]+ GR[Rt][15:0];
GR[Rd][31:16]= GR[Rs][31:16]- GR[Rt][15:0];
GR[Rd+1][15:0]= GR[Rs][15:0]- GR[Rt][15:0];
GR[Rd+1][31:16]= GR[Rs][31:16]+ GR[Rt][15:0];
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.6. VITDLSA

**内核：** v1

**格式：**

VITDLSUBADD Rd，Rs，Rt

**目的：**

计算 4 个路径度量.

**描述：**

Rs 的低 16 位和高 16 位分别是状态度量 state metric 0 和 state metric 1，Rt 低 16 位是分支度量 Branch metric 0

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数

**操作：**

```
GR[Rd][15:0]= GR[Rs][15:0]- GR[Rt][15:0];
GR[Rd][31:16]= GR[Rs][31:16]+ GR[Rt][15:0];
GR[Rd+1][15:0]= GR[Rs][15:0]+ GR[Rt][15:0];
GR[Rd+1][31:16]= GR[Rs][31:16]- GR[Rt][15:0];
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.7. VITHSEL

**内核：** v1

**格式：**

VITHSEL Rd, Rs, Rt

**目的：**

进行路径的选择

**描述：**

Rs 及 Rt 的高 16 位及低 16 位分别存储了 4 条路径的路径度量值，分别进行比较，得出两个状态度量 state metric 并更新到 Rd 和 Rd+1 寄存器的高 16 位中。同时产生两个指示路径选择的 transition bit：T0，T1，分别添加到 VT 寄存器的末尾。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数；VT0 和 VT1 为特殊寄存器

**操作：**

```
if (GR[Rt][15:0] > GR[Rt][31:16])
{
    GR[Rd+1][31:16]= GR[Rt][15:0];
    VT0 = VT0[30:0],1’b0;
}
else
{
    GR[Rd+1][31:16]= GR[Rt][31:16];
    VT0 = VT0[30:0],1’b1;
}
if (GR[Rs][15:0]> GR[Rs][31:16])
{
    GR[Rd][31:16]= GR[Rs][15:0];
    VT1 = VT1[30:0],1’b0;
}
else
{
    GR[Rd][31:16]= GR[Rs][31:16];
    VT1 = VT1[30:0],1’b1;
}
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.8. VITLSEL

**内核：** v1

**格式：**

VITLSEL Rd, Rs, Rt

**目的：**

进行路径的选择

**描述：**

Rs 及 Rt 的高 16 位及低 16 位分别存储了 4 条路径的路径度量值，分别进行比较，得出两个状态度量 state metric 并更新到 Rd 和 Rd+1 寄存器的低 16 位中。同时产生两个指示路径选择的 transition bit：T0，T1，分别添加到 VT 寄存器的末尾。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR，且 Rd 必须为偶数；VT0 和 VT1 为特殊寄存器

**操作：**

```
if (GR[Rt][15:0]> GR[Rt][31:16])
{
    GR[Rd+1][15:0]= GR[Rt][15:0];
    VT0 = VT0[30:0],1’b0;
}
else
{
    GR[Rd+1][15:0]= GR[Rt][31:16];
    VT0[0:0] = VT0[30:0],1’b1;
}
if (GR[Rs][15:0]> GR[Rs][31:16])
{
    GR[Rd][15:0]= GR[Rs][15:0];
    VT1[0:0] = VT1[30:0],1’b0;
}
else
{
    GR[Rd][15:0]= GR[Rs][31:16];
    VT1[0:0] = VT1[30:0],1’b1;
}
```

**异常：**

；

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

### 7.4.9. VTCLEAR

**内核：** v1

**格式：**

VTCLEAR

**目的：**

清除 Transition Bit Registers（VT 寄存器）

**描述：**

清除 Transition Bit Registers（VT 寄存器）

**限制条件：**

**操作：**

```
VT0 = 32’b0;
VT1 = 32’b0;
```

**异常：**

;

**备注：**

;

## 7.4.10. VTRACE

**内核：** v1

**格式：**

VTRACE Rd, Rs

**目的：**

进行路径回溯操作，并将路径存入寄存器中

**描述：**

根据 Rs 中的回溯状态，对存储在 VT0 和 VT1 寄存器中的 transition bits 进行回溯，将回溯值写入 Rd、回溯状态写入 Rd+1。VT0 和 VT1 寄存器中的 transition bits 由 VITLSEL 和 VITHSEL 指令以以下格式存储:

```
VT0[31] Transition bit [State 0]
VT0[30] Transition bit [State 1]
VT0[29] Transition bit [State 2]
... ...
VT0[0] Transition bit [State 31]
VT1[31] Transition bit [State 32]
VT1[30] Transition bit [State 33]
VT1[29] Transition bit [State 34]
... ...
VT1[0] Transition bit [State 63]
```

**限制条件：**

**操作：**

```
S = GR[Rs][5:0];
if (S < 32)
{
    temp[0] = VT0[31-S];
}
else
{
    temp[0] = VT1[63-S];
}
temp1[5:0] = 2* GR[Rs] [5:0] + temp[0];
GR[Rd] = 31’b0, temp[0];
```

```
GR[Rd+1] = 26’b0, temp1[5:0];
```

**异常：**

;

**备注：**

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

# 7.5. Viterbi II 指令

## 7.5.1. VDEC

**内核：** v3

**格式：**

VDEC Rd, Rs

**目的：**

16 位自减

**描述：**

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

```
GR[Rd] = GR[Rs] - 1;
```

**异常：**

### 7.5.2. VINC

**内核：**v3

**格式：**

VDEC Rd, Rs

**目的：**

16 位自增

**描述：**

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

```
GR[Rd] = GR[Rs] + 1;
```

**异常：**

### 7.5.3. VCCON

**内核：** v3

**格式：**

VCCON Rd, Rs

**目的：**

16 位整型复数共轭运算

**描述：**

操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

将操作数虚部取反，实部不变，计算结果的虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

```
GR[Rd] = {GR[Rs][31:16], -GR[Rs][15:0]} ;
if (SAT == 1)
{
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

不影响 OVFR 溢出标志位；根据虚部运算结果（GR[Rd][15:0]）设置 OVFI 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果 GR[Rd][15:0]压缩为 0x7FFF（32767）；若发生下溢，则将对应结果 GR[Rd][15:0]压缩为 0x8000（-32768）。

### 7.5.4. VCFLIP

**内核：** v3

**格式：**

VCFLIP Rd, Rs

**目的：**

16 位整型复数实部虚部交换

**描述：**

操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

将操作数虚部和实部交换，计算结果的虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

**限制条件：**

Rd 为 GR，Rs 为 GR

**操作：**

```
GR[Rd] = {GR[Rs][15:0], GR[Rs][31:16]} ;
if (SAT == 1)
{
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

不影响 OVFR、OVFI 溢出标志位，不进行饱和运算。

### 7.5.5. VCSHL

**内核：** v3

**格式：**

VCSHL Rd, Rs, Rt

**目的：**

16 位有符号整型复数的实部和虚部分别逻辑左移

**描述：**

操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

计算结果的虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

```
GR[Rd][31:16] ← GR[Rs][31:16] << GR[Rt][3:0]
GR[Rd][15:0]  ← GR[Rs][15:0]  << GR[Rt][3:0]
if (SAT == 1)
{
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

根据实部运算结果（GR[Rd][31:16]）设置 OVFR 溢出标志位，根据虚部运算结果（GR[Rd][15:0]）设置 OVFI 溢出标志位，发生溢出时均设置对应标志位为 1，否则置 0。不进行饱和运算。

### 7.5.6. VCSHR

**内核：**v3

**格式：**

VCSHR Rd, Rs, Rt

**目的：**

16 位有符号整型复数的实部和虚部分别算术右移

**描述：**

操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

将操作数虚部和实部交换，计算结果的虚部存入 GR[Rd][15:0]，实部存入 GR[Rd][31:16]。

空出的高位补符号位

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR

**操作：**

```
GR[Rd][31:16] ← GR[Rs][31:16] >>>GR[Rt][3:0]
GR[Rd][15:0]  ← GR[Rs][15:0]  >>>GR[Rt][3:0]
if (SAT == 1)
{
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

不影响 OVFR、OVFI 溢出标志位，不进行饱和运算。

### 7.5.7. VCMAG

**内核：** v3

**格式：**

VCMAG Rd, Rs

**目的：**

16 位有符号整型复数求复数模运算

**描述：**

操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

求模计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，且 Rs、Rd 必须为偶数

**操作：**

```
GR[Rd] = GR[Rs][15:0] * GR[Rs+1][15:0] + GR[Rs][31:16] * GR[Rs+1][31:16];
if (SAT == 1)
{
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

**备注：**

根据虚部运算结果（GR[Rd]）设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。不影响 OVFR

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-2147483648）。

### 7.5.8. VCRC24LL

**内核：**v3

**格式：**

VCRC32LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x5D6DCB。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][7:0], GR[Rt])
```

**异常：**

；

**备注：**

；

### 7.5.9. VCRC24LH

**内核：**v3

**格式：**

VCRC32LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x5D6DCB。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][15:8], GR[Rt])
```

**异常：**

；

**备注：**

；

## 7.5.10. VCRC24HL

**内核：**v3

**格式：**

VCRC32HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x5D6DCB。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][23:16], GR[Rt])
```

**异常：**

；

**备注：**

；

### 7.5.11. VCRC24HH

**内核：**v3

**格式：**

VCRC32HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x5D6DCB。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][31:24], GR[Rt])
```

**异常：**

；

**备注：**

；

## 7.5.12. VCRC32P2LL

**内核：**v3

**格式：**

VCRC32LL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[7:0]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x1EDC6F41。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][7:0], GR[Rt])
```

**异常：**

；

**备注：**

；

### 7.5.13. VCRC32P2LH

**内核：**v3

**格式：**

VCRC32LH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[15:8]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x1EDC6F41。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][15:8], GR[Rt])
```

**异常：**

;

**备注：**

;

## 7.5.14. VCRC32P2HL

**内核：**v3

**格式：**

VCRC32HL Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[23:16]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x1EDC6F41。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][23:16], GR[Rt])
```

**异常：**

；

**备注：**

；

## 7.5.15. VCRC32P2HH

**内核：**v3

**格式：**

VCRC32HH Rd, Rs, Rt

**目的：**

计算单字节数据的 CRC32 校验码。

**描述：**

以 Rt 寄存器指定的数据作为初始值计算 Rs 寄存器中[31:24]位指定字节数据的 CRC32 校验码，CRC 校验多项式为 0x1EDC6F41。计算初始值取 Rt 寄存器中所存放的数据。计算结果存放在 GR[Rd]寄存器中。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd] ← CRC32 (GR[Rs][31:24], GR[Rt])
```

**异常：**

；

**备注：**

；

## 7.5.16. VCFFT1

**内核：** v3

**格式：**

VCFFT1 Rd, Rs, Rt

**目的：**

计算 16 位有符号整型复数蝶形相乘运算，结果为 32 位数。

**描述：**

乘法的第一个操作数的虚部存储在 GR[Rs+1][15:0]，实部存储在 GR[Rs+1][31:16]，第二个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]。

首先执行乘法操作，分别对乘法的第一第二操作数做复数乘法，再将乘法结果的实部和虚部与累加数的实部和虚部进行相应的加法或减法计算，计算结果分别存入 GR[Rd+1]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rs+1 为 GR。

**操作：**

```
GR[Rd + 1] = GR[Rs][15:0] * GR[Rs+1][31:16] - GR[Rs][31:16] * GR[Rs+1][15:0];
GR[Rd] = GR[Rs][15:0] * GR[Rs+1][15:0] + GR[Rs][31:16] * GR[Rs+1][31:16];
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
```

**异常：**

;

**备注：**

根据 GR[Rd + 1]运算结果设置 OVFR 溢出标志位，根据 GR[Rd]运算结果设置 OVFI 溢出标志位。运算结果发生上溢（>2147483647）或下溢（<-2147483648）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应结果压缩为 0x7FFFFFFF（2147483647）；若发生下溢，则将对应结果压缩为 0x80000000（-2147483648）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

；

## 7.5.17. VCFFT6

**内核：**v3

**格式：**

VCFFT6 Rd, Rs, Rt,imm$_1$

**目的：**

计算 16 位有符号整型复数蝶形运算加减。

**描述：**

第一个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]，第二个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]。

分别对乘法的第一、第二操作数的实部和虚部进行相应的加法或减法计算，计算结果分别存入 GR[Rd+1]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd+1][31:16] = (GR[Rt][31:16] + GR[Rs][31:16])>>imm1;
GR[Rd+1][15:0] = (GR[Rt][15:0] - GR[Rs][15:0])>>imm1;
GR[Rd][31:16] = (GR[Rt][31:16] - GR[Rs][31:16])>>imm1;
GR[Rd][15:0] = (GR[Rt][15:0] + GR[Rs][15:0])>>imm1;
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
if (RND== 1)
{
    GR[Rd + 1] = RND(GR[Rd + 1]);
    GR[Rd] = RND(GR[Rd]);
}
```

**异常：**

；

**备注：**

根据 GR[Rd]运算结果设置 OVFI 溢出标志位，根据 GR[Rd+1]运算结果设置 OVFR 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应 16bit 结果压缩为 0x7FFF（32767）；若发生下溢，则将对应 16bit 结果 GR[Rd][15:0]压缩为 0x8000（-32768）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

## 7.5.18. VCFFT8

**内核：**v3

**格式：**

VCFFT8 Rd, Rs, Rt,$imm_1$

**目的：**

计算 16 位有符号整型复数蝶形运算加减。

**描述：**

第一个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]，第二个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]。

分别对乘法的第一、第二操作数的实部和虚部进行相应的加法或减法计算，计算结果分别存入 GR[Rd+1]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd+1][31:16] = (GR[Rs][31:16] - GR[Rt][31:16])>>imm1;
GR[Rd+1][15:0] = (GR[Rs][31:16] + GR[Rt][31:16])>>imm1;
GR[Rd][31:16] = (GR[Rs][15:0] - GR[Rt][15:0])>>imm1;
GR[Rd][15:0] = (GR[Rs][15:0] + GR[Rt][15:0])>>imm1;
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
if (RND == 1)
{
    GR[Rd + 1] = RND(GR[Rd + 1]);
    GR[Rd] = RND(GR[Rd]);
}
```

**异常：**

；

**备注：**

根据 GR[Rd]运算结果设置 OVFI 溢出标志位，根据 GR[Rd+1]运算结果设置 OVFR 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应 16bit 结果压缩为 0x7FFF（32767）；若发生下溢，则将对应 16bit 结果 GR[Rd][15:0]压缩为 0x8000（-32768）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

## 7.5.19. VCFFT9

**内核：** v3

**格式：**

VCFFT9 Rd, Rs, Rt,$imm_1$

**目的：**

计算 16 位有符号整型复数蝶形运算加减。

**描述：**

第一个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]，第二个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]，第三个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]，第四个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]。

分别对乘法的第一、第二操作数的实部和虚部进行相应的加法或减法计算，计算结果分别存入 GR[Rd+1]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd+1][31:16] = (GR[Rt][15:0] - GR[Rt+1][15:0])>>imm1;
GR[Rd+1][15:0] = (GR[Rs][15:0] - GR[Rs+1][15:0])>>imm1;
GR[Rd][31:16] = (GR[Rt][15:0] + GR[Rt+1][15:0])>>imm1;
GR[Rd][15:0] = (GR[Rs][15:0] + GR[Rs+1][15:0])>>imm1;
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
if (RND == 1)
{
    GR[Rd + 1] = RND(GR[Rd + 1]);
```

```
    GR[Rd] = RND(GR[Rd]);
}
```

**异常：**

```
    ;
```

**备注：**

根据 GR[Rd]运算结果设置 OVFI 溢出标志位，根据 GR[Rd+1]运算结果设置 OVFR 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应 16bit 结果压缩为 0x7FFF（32767）；若发生下溢，则将对应 16bit 结果 GR[Rd][15:0]压缩为 0x8000（-32768）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

## 7.5.20. VCFFT10

**内核：**v3

**格式：**

VCFFT10 Rd, Rs, Rt,imm$_1$

**目的：**

计算 16 位有符号整型复数蝶形运算加减。

**描述：**

第一个操作数的虚部存储在 GR[Rs][15:0]，实部存储在 GR[Rs][31:16]，第二个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]，第三个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]，第四个操作数的虚部存储在 GR[Rt][15:0]，实部存储在 GR[Rt][31:16]。

分别对乘法的第一、第二操作数的实部和虚部进行相应的加法或减法计算，计算结果分别存入 GR[Rd+1]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd+1][31:16] = (GR[Rt][31:16] - GR[Rt+1][31:16])>>imm1;
GR[Rd+1][15:0] = (GR[Rs][31:16] - GR[Rs+1][31:16])>>imm1;
GR[Rd][31:16] = (GR[Rt][31:16] + GR[Rt+1][31:16])>>imm1;
GR[Rd][15:0] = (GR[Rs][31:16] + GR[Rs+1][31:16])>>imm1;
if (SAT == 1)
{
    GR[Rd + 1] = sat32(GR[Rd + 1]);
    GR[Rd] = sat32(GR[Rd]);
}
if (RND == 1)
{
    GR[Rd + 1] = RND(GR[Rd + 1]);
```

```
    GR[Rd] = RND(GR[Rd]);
}
```

**异常：**

```
    ;
```

**备注：**

根据 GR[Rd]运算结果设置 OVFI 溢出标志位，根据 GR[Rd+1]运算结果设置 OVFR 溢出标志位。运算结果发生上溢（>32767）或下溢（<-32768）时，均设置对应溢出标志位为 1，否则置 0。

运算后，若 SAT 标志位为 1，则需要对运算结果按规则进行饱和操作。若发生上溢，将对应 16bit 结果压缩为 0x7FFF（32767）；若发生下溢，则将对应 16bit 结果 GR[Rd][15:0]压缩为 0x8000（-32768）。

Rd 用 0 槽寄存器写入通路，Rd+1 使用 1 槽寄存器写入通路。

## 7.5.21. VGFACC

**内核：** v3

**格式：**

VGFACC Rd, Rs, Rt

**目的：**

计算 8bit 伽罗华域加法

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，第三个操作数存储在 GR[Rs+1]。

对第一操作数和第二操作数进行伽罗华域加法计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
if (GR[Rs+1][0:0] == 1 )
    GR[Rd][7:0] = GR[Rs][7:0] ^ GR[Rt][7:0]
if (GR[Rs+1][1:1] == 1 )
    GR[Rd][7:0] = GR[Rs][7:0] ^ GR[Rt][15:8]
if (GR[Rs+1][2:2] == 1 )
    GR[Rd][7:0] = GR[Rs][7:0] ^ GR[Rt][23:16]
if (GR[Rs+1][3:3] == 1 )
    GR[Rd][7:0] = GR[Rs][7:0] ^ GR[Rt][31:24]
```

**异常：**

;

**备注：**

## 7.5.22. VGFADD

**内核：** v3

**格式：**

VGFADD Rd, Rs, Rt,$imm_4$

**目的：**

计算伽罗华域加法

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，第三个操作数存储在 GR[Rs+1]。

对第一操作数和第二操作数进行伽罗华域加法计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
if (imm1 == 1)
    GR[Rd][7:0] = GR[Rs][7:0] ^ GR[Rt][7:0]
else
    GR[Rd][7:0] = GR[Rs][7:0]

if (imm2 == 1)
    GR[Rd][15:8] = GR[Rs][15:8] ^ GR[Rt][15:8]
else
    GR[Rd][15:8] = GR[Rs][15:8]

if (imm3 == 1)
    GR[Rd][23:16] = GR[Rs][23:16] ^ GR[Rt][23:16]
else
    GR[Rd][23:16] = GR[Rs][23:16]

if (imm4 == 1)
```

```
        GR[Rd][31:24] = GR[Rs][31:24] ^ GR[Rt][31:24]
    else
        GR[Rd][31:24] = GR[Rs][31:24]
```

**异常：**

```
        ;
```

**备注：**

## 7.5.23. VREVB

**内核：**v3

**格式：**

VREVB Rd, Rs

**目的：**

计算字节倒序

**描述：**

操作数存储在 GR[Rs]，进行字节倒序计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR。

**操作：**

```
GR[Rs] = {B3,B2,B1,B0}
GR[Rd] = {B0,B1,B2,B3}
```

**异常：**

;

**备注：**

## 7.5.24. VPACK

**内核：** v3

**格式：**

VPACK Rd, Rs, $imm_2$

**目的：**

计算字节打包

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，进行字节打包计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
if (imm == 0)
    GR[Rd] = {GR[Rs][7:0], GR[Rs][7:0], GR[Rs][7:0], GR[Rs][7:0]}

else if (imm == 1)
    GR[Rd] = {GR[Rs][15:8], GR[Rs][15:8], GR[Rs][15:8], GR[Rs][15:8]}

else if (imm == 2)
    GR[Rd] = {GR[Rs][23:16], GR[Rs][23:16], GR[Rs][23:16], GR[Rs][23:16]}

else if (imm == 3)
    GR[Rd] = {GR[Rs][[31:24]], GR[Rs][31:24], GR[Rs][31:24], GR[Rs][31:24]}
```

**异常：**

;

**备注：**

## 7.5.25. VGFMPY

**内核：**v3

**格式：**

VGFMPY Rd, Rs， Rt

**目的：**

计算伽罗华域乘法

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，进行伽罗华域乘法计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd][7:0]   = GR[Rs][7:0]  * GR[Rt][7:0]
GR[Rd][15:8]  = GR[Rs][15:8] * GR[Rt][15:8]
GR[Rd][23:16] = GR[Rs][23:16] * GR[Rt][23:16]
GR[Rd][31:24] = GR[Rs][31:24] * GR[Rt][31:24]
```

**异常：**

；

**备注：**

## 7.5.26. VGFMAC

**内核：** v3

**格式：**

VGFMAC Rd, Rs， Rt

**目的：**

计算伽罗华域乘加

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，第三个操作数存储在 GR[Rs+1]，进行伽罗华域乘加计算，计算结果存入 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rd][7:0]   = (GR[Rs][7:0]  * GR[Rt][7:0]) ^ GR[Rs+1][7:0]
GR[Rd][15:8]  = (GR[Rs][15:8] * GR[Rt][15:8]) ^ GR[Rs+1][15:8]
GR[Rd][23:16] = (GR[Rs][23:16] * GR[Rt][23:16]) ^ GR[Rs+1][23:16]
GR[Rd][31:24] = (GR[Rs][31:24] * GR[Rt][31:24]) ^ GR[Rs+1][31:24]
```

**异常：**

；

**备注：**

## 7.5.27. VSHLMB

**内核：** v3

**格式：**

VSHLMB Rd, Rs, Rt

**目的：**

计算伽罗华域移位

**描述：**

第一个操作数存储在 GR[Rs]，第二个操作数存储在 GR[Rt]，进行伽罗华域乘法计算，计算结果存入 GR[Rd]和 GR[Rd]。

**限制条件：**

Rd 为 GR，Rs 为 GR，Rt 为 GR。

**操作：**

```
GR[Rs] = {B7,B6,B5,B4}
GR[Rt] = {B3,B2,B1,B0}

GR[Rd+1] = {B6,B5,B4,B3}
GR[Rd] = {B2,B1,B0,8'b0}
```

**异常：**

;

**备注：**