

见下页

**mul** rd, rs1, rs2

$$x[rd] = x[rs1] \times x[rs2]$$

乘 (*Multiply*). R-type, RV32M and RV64M.

把寄存器  $x[rs2]$  乘到寄存器  $x[rs1]$  上，乘积写入  $x[rd]$ 。忽略算术溢出。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	000	rd	0110011	

**mulh** rd, rs1, rs2

$$x[rd] = (x[rs1]_s \times_s x[rs2]) \gg_s XLEN$$

高位乘 (*Multiply High*). R-type, RV32M and RV64M.

把寄存器  $x[rs2]$  乘到寄存器  $x[rs1]$  上，都视为 2 的补码，将乘积的高位写入  $x[rd]$ 。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	001	rd	0110011	

**mulhsu** rd, rs1, rs2

$$x[rd] = (x[rs1]_s \times_u x[rs2]) \gg_s XLEN$$

高位有符号-无符号乘 (*Multiply High Signed-Unsigned*). R-type, RV32M and RV64M.

把寄存器  $x[rs2]$  乘到寄存器  $x[rs1]$  上， $x[rs1]$  为 2 的补码， $x[rs2]$  为无符号数，将乘积的高位写入  $x[rd]$ 。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	010	rd	0110011	

**mulhu** rd, rs1, rs2

$$x[rd] = (x[rs1]_u \times_u x[rs2]) \gg_u XLEN$$

高位无符号乘 (*Multiply High Unsigned*). R-type, RV32M and RV64M.

把寄存器  $x[rs2]$  乘到寄存器  $x[rs1]$  上， $x[rs1]$ 、 $x[rs2]$  均为无符号数，将乘积的高位写入  $x[rd]$ 。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	011	rd	0110011	

**mulw** rd, rs1, rs2

$$x[rd] = \text{sext}((x[rs1] \times x[rs2])[31:0])$$

乘字 (*Multiply Word*). R-type, RV64M only.

把寄存器  $x[rs2]$  乘到寄存器  $x[rs1]$  上，乘积截为 32 位，进行有符号扩展后写入  $x[rd]$ 。忽略算术溢出。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	000	rd	0111011	

**mv** rd, rs1

$$x[rd] = x[rs1]$$

移动 (*Move*). 伪指令 (Pseudoinstruction), RV32I and RV64I.

把寄存器  $x[rs1]$  复制到  $x[rd]$  中。实际被扩展为 **addi** rd, rs1, 0

**neg** rd, rs2  $x[rd] = -x[rs2]$   
取反(Negate). 伪指令(Pseudoinstruction), RV32I and RV64I.  
把寄存器  $x[rs2]$  的二进制补码写入  $x[rd]$ 。实际被扩展为 **sub** rd, x0, rs2。

---

**neg** rd, rs2  $x[rd] = \text{sext}((-x[rs2])[31:0])$   
取非字(Negate Word). 伪指令(Pseudoinstruction), RV64I only.  
计算寄存器  $x[rs2]$  对于 2 的补码, 结果截为 32 位, 进行符号扩展后写入  $x[rd]$ 。实际被扩展为 **subw** rd, x0, rs2。

---

**nop** Nothing  
无操作(No operation). 伪指令(Pseudoinstruction), RV32I and RV64I.  
将 *pc* 推进到下一条指令。实际被扩展为 **addi** x0, x0, 0。

---

**not** rd, rs1  $x[rd] = \sim x[rs1]$   
取反(NOT). 伪指令(Pseudoinstruction), RV32I and RV64I.  
把寄存器  $x[rs1]$  对于 1 的补码 (即按位取反的值) 写入  $x[rd]$ 。实际被扩展为 **xori** rd, rs1, -1。

---

**or** rd, rs1, rs2  $x[rd] = x[rs1] | x[rs2]$   
取或(OR). R-type, RV32I and RV64I.  
把寄存器  $x[rs1]$  和寄存器  $x[rs2]$  按位取或, 结果写入  $x[rd]$ 。  
压缩形式: **c.or** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	110	rd	0110011	

**ori** rd, rs1, immediate  $x[rd] = x[rs1] | \text{sext}(\text{immediate})$   
立即数取或(OR Immediate). R-type, RV32I and RV64I.  
把寄存器  $x[rs1]$  和有符号扩展的立即数 *immediate* 按位取或, 结果写入  $x[rd]$ 。  
压缩形式: **c.or** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
Immediate[11:0]	rs2	rs1	110	rd	0010011	

**rdcycle** rd  $x[rd] = \text{CSRS}[\text{cycle}]$   
读周期计数器(Read Cycle Counter). 伪指令(Pseudoinstruction), RV32I and RV64I.  
把周期数写入  $x[rd]$ 。实际被扩展为 **csrrs** rd, cycle, x0。

---

rdcycleh<sub>rd</sub>x[rd] = CSRs[cycleh]  
读周期计数器高位(*Read Cycle Counte High*). 伪指令(Pseudoinstruction), RV32I only.  
把周期数右移 32 位后写入 x[rd]。实际被扩展为 **csrrs** rd, cycleh, x0。

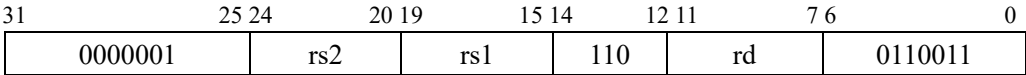
rdinstret<sub>rd</sub>x[rd] = CSRs[instret]  
读已完成指令计数器(*Read Instruction-Retired Counter*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
把已完成指令数写入 x[rd]。实际被扩展为 **csrrs** rd, instret, x0。

rdinstreth<sub>rd</sub>x[rd] = CSRs[instreth]  
读已完成指令计数器高位(*Read Instruction-Retired Counter High*). 伪指令(Pseudoinstruction), RV32I only.  
把已完成指令数右移 32 位后写入 x[rd]。实际被扩展为 **csrrs** rd, instreth, x0。

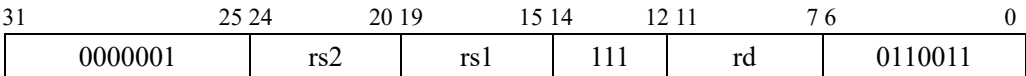
rdtime<sub>rd</sub>x[rd] = CSRs[time]  
读取时间(*Read Time*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
把当前时间写入 x[rd]，时间频率与平台相关。实际被扩展为 **csrrs** rd, time, x0。

rdtimeh<sub>rd</sub>x[rd] = CSRs[timeh]  
读取时间高位(*Read Time High*). 伪指令(Pseudoinstruction), RV32I only.  
把当前时间右移 32 位后写入 x[rd]，时间频率与平台相关。实际被扩展为 **csrrs** rd, timeh, x0。

rem<sub>rd, rs1, rs2</sub>x[rd] = x[rs1] %<sub>s</sub> x[rs2]  
求余数(*Remainder*). R-type, RV32M and RV64M.  
x[rs1]除以 x[rs2]，向 0 舍入，都视为 2 的补码，余数写入 x[rd]。



remu<sub>rd, rs1, rs2</sub>x[rd] = x[rs1] %<sub>u</sub> x[rs2]  
求无符号数的余数(*Remainder, Unsigned*). R-type, RV32M and RV64M.  
x[rs1]除以 x[rs2]，向 0 舍入，都视为无符号数，余数写入 x[rd]。



**remuw** rd, rs1, rs2  $x[rd] = \text{sext}(x[rs1][31:0]) \%_u x[rs2][31:0]$   
 求无符号数的余数字(*Remainder Word, Unsigned*). R-type, RV64M only.  
 x[rs1]的低 32 位除以 x[rs2]的低 32 位, 向 0 舍入, 都视为无符号数, 将余数的有符号扩展写入 x[rd]。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	111	rd	0111011	

**remw** rd, rs1, rs2  $x[rd] = \text{sext}(x[rs1][31:0]) \%_s x[rs2][31:0]$   
 求余数字(*Remainder Word*). R-type, RV64M only.  
 x[rs1]的低 32 位除以 x[rs2]的低 32 位, 向 0 舍入, 都视为 2 的补码, 将余数的有符号扩展写入 x[rd]。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	110	rd	0111011	

**ret**  $pc = x[1]$   
 返回(*Return*). 伪指令(*Pseudoinstruction*), RV32I and RV64I.  
 从子过程返回。实际被扩展为 **jalr** x0, 0(x1)。

**sb** rs2, offset(rs1)  $M[x[rs1] + \text{sext}(\text{offset})] = x[rs2][7:0]$   
 存字节(*Store Byte*). S-type, RV32I and RV64I.  
 将 x[rs2]的低位字节存入内存地址 x[rs1]+sign-extend(offset)。

31	25 24	20 19	15 14	12 11	7 6	0
offset[11:5]	rs2	rs1	000	offset[4:0]	0100011	

**sc.d** rd, rs2, (rs1)  $x[rd] = \text{StoreConditional64}(M[x[rs1], x[rs2]])$   
 条件存入双字(*Store-Conditional Doubleword*). R-type, RV64A only.  
 如果内存地址 x[rs1]上存在加载保留, 将 x[rs2]寄存器中的 8 字节数存入该地址。如果存入成功, 向寄存器 x[rd]中存入 0, 否则存入一个非 0 的错误码。

31	27 26	25 24	20 19	15 14	12 11	7 6	0
00011	aq	rl	rs2	rs1	011	rd	0101111

**SC.W** rd, rs2, (rs1)  $x[rd] = \text{StoreConditonal32}(M[x[rs1]], x[rs2])$

条件存入字(*Store-Conditional Word*). R-type, RV32A and RV64A.

内存地址  $x[rs1]$  上存在加载保留, 将  $x[rs2]$  寄存器中的 4 字节数存入该地址。如果存入成功, 向寄存器  $x[rd]$  中存入 0, 否则存入一个非 0 的错误码。

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00011	aq	rl	rs2	rs1	010	rd	0101111						

**sd** rs2, offset(rs1)  $M[x[rs1] + \text{sext}(\text{offset})] = x[rs2][63:0]$

存双字(*Store Doubleword*). S-type, RV64I only.

将  $x[rs2]$  中的 8 字节存入内存地址  $x[rs1] + \text{sign-extend}(\text{offset})$ 。

压缩形式: **c.sdsp** rs2, offset; **c.sd** rs2, offset(rs1)

31	25	24	20	19	15	14	12	11	7	6	0
offset[11:5]	rs2	rs1	011	offset[4:0]	0100011						

**seqz** rd, rs1  $x[rd] = (x[rs1] == 0)$

等于 0 则置位(*Set if Equal to Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.

如果  $x[rs1]$  等于 0, 向  $x[rd]$  写入 1, 否则写入 0。实际被扩展为 **sltiu** rd, rs1, 1。

**sext.w** rd, rs1  $x[rd] = \text{sext}(x[rs1][31:0])$

有符号字扩展(*Sign-extend Word*). 伪指令(Pseudoinstruction), RV64I only.

读入  $x[rs1]$  的低 32 位, 有符号扩展, 结果写入  $x[rd]$ 。实际被扩展为 **addiw** rd, rs1, 0。

**sfence.vma** rs1, rs2 Fence(Store, AddressTranslation)

虚拟内存屏障(*Fence Virtual Memory*). R-type, RV32I and RV64I 特权指令。

根据后续的虚拟地址翻译对之前的页表存入进行排序。当  $rs2=0$  时, 所有地址空间的翻译都会受到影响; 否则, 仅对  $x[rs2]$  标识的地址空间的翻译进行排序。当  $rs1=0$  时, 对所选地址空间中的所有虚拟地址的翻译进行排序; 否则, 仅对其中包含虚拟地址  $x[rs1]$  的页面地址翻译进行排序。

31	25	24	20	19	15	14	12	11	7	6	0
0001001	rs2	rs1	000	00000	1110011						

**sgtz** rd, rs2  $x[rd] = (x[rs1] >_s 0)$

大于 0 则置位(*Set if Greater Than Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.

如果  $x[rs2]$  大于 0, 向  $x[rd]$  写入 1, 否则写入 0。实际被扩展为 **slt** rd, x0, rs2。

**sh** rs2, offset(rs1)  $M[x[rs1] + sext(offset) = x[rs2][15:0]$   
 存半字(*Store Halfword*). S-type, RV32I and RV64I.  
 将  $x[rs2]$  的低位 2 个字节存入内存地址  $x[rs1] + sign-extend(offset)$ 。

31	25 24	20 19	15 14	12 11	7 6	0
offset[11:5]	rs2	rs1	001	offset[4:0]	0100011	

**SW** rs2, offset(rs1)  $M[x[rs1] + sext(offset) = x[rs2][31:0]$   
 存字(*Store Word*). S-type, RV32I and RV64I.  
 将  $x[rs2]$  的低位 4 个字节存入内存地址  $x[rs1] + sign-extend(offset)$ 。  
 压缩形式: **c.swsp** rs2, offset; **c.sw** rs2, offset(rs1)

31	25 24	20 19	15 14	12 11	7 6	0
offset[11:5]	rs2	rs1	010	offset[4:0]	0100011	

**sll** rd, rs1, rs2  $x[rd] = x[rs1] \ll x[rs2]$   
 逻辑左移(*Shift Left Logical*). R-type, RV32I and RV64I.  
 把寄存器  $x[rs1]$  左移  $x[rs2]$  位, 空出的位置填入 0, 结果写入  $x[rd]$ 。  $x[rs2]$  的低 5 位 (如果是 RV64I 则是低 6 位) 代表移动位数, 其高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	001	rd	0110011	

**slli** rd, rs1, shamt  $x[rd] = x[rs1] \ll shamt$   
 立即数逻辑左移(*Shift Left Logical Immediate*). I-type, RV32I and RV64I.  
 把寄存器  $x[rs1]$  左移  $shamt$  位, 空出的位置填入 0, 结果写入  $x[rd]$ 。对于 RV32I, 仅当  $shamt[5]=0$  时, 指令才是有效的。

压缩形式: **c.slli** rd, shamt

31	26 25	20 19	15 14	12 11	7 6	0
0000000	shamt	rs1	001	rd	0010011	

**slliw** rd, rs1, shamt  $x[rd] = sext((x[rs1] \ll shamt)[31:0])$   
 立即数逻辑左移字(*Shift Left Logical Word Immediate*). I-type, RV64I only.  
 把寄存器  $x[rs1]$  左移  $shamt$  位, 空出的位置填入 0, 结果截为 32 位, 进行有符号扩展后写入  $x[rd]$ 。仅当  $shamt[5]=0$  时, 指令才是有效的。

31	26 25	20 19	15 14	12 11	7 6	0
0000000	shamt	rs1	001	rd	0011011	

**slw** rd, rs1, rs2

$$x[rd] = \text{sext}((x[rs1] \ll x[rs2][4:0])[31:0])$$

逻辑左移字(*Shift Left Logical Word*). R-type, RV64I only.

把寄存器  $x[rs1]$  的低 32 位左移  $x[rs2]$  位, 空出的位置填入 0, 结果进行有符号扩展后写入  $x[rd]$ 。  $x[rs2]$  的低 5 位代表移动位数, 其高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	001	rd	0111011	

**slt** rd, rs1, rs2

$$x[rd] = (x[rs1] <_s x[rs2])$$

小于则置位(*Set if Less Than*). R-type, RV32I and RV64I.

比较  $x[rs1]$  和  $x[rs2]$  中的数, 如果  $x[rs1]$  更小, 向  $x[rd]$  写入 1, 否则写入 0。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	010	rd	0110011	

**slti** rd, rs1, immediate

$$x[rd] = (x[rs1] <_s \text{sext}(\text{immediate}))$$

小于立即数则置位(*Set if Less Than Immediate*). I-type, RV32I and RV64I.

比较  $x[rs1]$  和有符号扩展的 *immediate*, 如果  $x[rs1]$  更小, 向  $x[rd]$  写入 1, 否则写入 0。

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	010	rd	0010011	

**sltiu** rd, rs1, immediate

$$x[rd] = (x[rs1] <_u \text{sext}(\text{immediate}))$$

无符号小于立即数则置位(*Set if Less Than Immediate, Unsigned*). I-type, RV32I and RV64I.

比较  $x[rs1]$  和有符号扩展的 *immediate*, 比较时视为无符号数。如果  $x[rs1]$  更小, 向  $x[rd]$  写入 1, 否则写入 0。

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	011	rd	0010011	

**sltu** rd, rs1, rs2

$$x[rd] = (x[rs1] <_u x[rs2])$$

无符号小于则置位(*Set if Less Than, Unsigned*). R-type, RV32I and RV64I.

比较  $x[rs1]$  和  $x[rs2]$ , 比较时视为无符号数。如果  $x[rs1]$  更小, 向  $x[rd]$  写入 1, 否则写入 0。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	011	rd	0110011	



**sltz** rd, rs1  $x[rd] = (x[rs1] <_s 0)$   
 小于 0 则置位(*Set if Less Than to Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
 如果  $x[rs1]$  小于 0, 向  $x[rd]$  写入 1, 否则写入 0。实际扩展为 **slt** rd, rs1, x0。

**snez** rd, rs2  $x[rd] = x[rs2] \neq 0$   
 不等于 0 则置位(*Set if Not Equal to Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
 如果  $x[rs1]$  不等于 0, 向  $x[rd]$  写入 1, 否则写入 0。实际扩展为 **sltu** rd, x0, rs2。

**sra** rd, rs1, rs2  $x[rd] = (x[rs1] \gg_s x[rs2])$   
 算术右移(*Shift Right Arithmetic*). R-type, RV32I and RV64I.  
 把寄存器  $x[rs1]$  右移  $x[rs2]$  位, 空位用  $x[rs1]$  的最高位填充, 结果写入  $x[rd]$ 。  $x[rs2]$  的低 5 位 (如果是 RV64I 则是低 6 位) 为移动位数, 高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	101	rd	0110011	

**srai** rd, rs1, shamt  $x[rd] = (x[rs1] \gg_s shamt)$   
 立即数算术右移(*Shift Right Arithmetic Immediate*). I-type, RV32I and RV64I.  
 把寄存器  $x[rs1]$  右移  $shamt$  位, 空位用  $x[rs1]$  的最高位填充, 结果写入  $x[rd]$ 。对于 RV32I, 仅当  $shamt[5]=0$  时指令有效。

压缩形式: **c.srai** rd, shamt

31	26 25	20 19	15 14	12 11	7 6	
010000	shamt	rs1	101	rd	0010011	

**sraiw** rd, rs1, shamt  $x[rd] = sext(x[rs1][31:0] \gg_s shamt)$   
 立即数算术右移字(*Shift Right Arithmetic Word Immediate*). I-type, RV64I only.  
 把寄存器  $x[rs1]$  的低 32 位右移  $shamt$  位, 空位用  $x[rs1][31]$  填充, 结果进行有符号扩展后写入  $x[rd]$ 。仅当  $shamt[5]=0$  时指令有效。

压缩形式: **c.srai** rd, shamt

31	26 25	20 19	15 14	12 11	7 6	
010000	shamt	rs1	101	rd	0011011	

**sraw** rd, rs1, rs2

$$x[rd] = \text{sext}(x[rs1][31:0] \gg_s x[rs2][4:0])$$

算术右移字(*Shift Right Arithmetic Word*). R-type, RV64I only.

把寄存器  $x[rs1]$  的低 32 位右移  $x[rs2]$  位, 空位用  $x[rs1][31]$  填充, 结果进行有符号扩展后写入  $x[rd]$ 。 $x[rs2]$  的低 5 位为移动位数, 高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	101	rd	0111011	

**sret**

ExceptionReturn(Supervisor)

管理员模式例外返回(*Supervisor-mode Exception Return*). R-type, RV32I and RV64I 特权指令。

从管理员模式的例外处理程序中返回, 设置  $pc$  为  $CSRs[spec]$ , 权限模式为  $CSRs[sstatus].SPP$ ,  $CSRs[sstatus].SIE$  为  $CSRs[sstatus].SPIE$ ,  $CSRs[sstatus].SPIE$  为 1,  $CSRs[sstatus].spp$  为 0。

31	25 24	20 19	15 14	12 11	7 6	0
0001000	00010	00000	000	00000	1110011	

**srl** rd, rs1, rs2

$$x[rd] = (x[rs1] \gg_u x[rs2])$$

逻辑右移(*Shift Right Logical*). R-type, RV32I and RV64I.

把寄存器  $x[rs1]$  右移  $x[rs2]$  位, 空出的位置填入 0, 结果写入  $x[rd]$ 。 $x[rs2]$  的低 5 位 (如果是 RV64I 则是低 6 位) 代表移动位数, 其高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	101	rd	0110011	

**srlrli** rd, rs1, shamt

$$x[rd] = (x[rs1] \gg_u \text{shamt})$$

立即数逻辑右移(*Shift Right Logical Immediate*). I-type, RV32I and RV64I.

把寄存器  $x[rs1]$  右移  $\text{shamt}$  位, 空出的位置填入 0, 结果写入  $x[rd]$ 。对于 RV32I, 仅当  $\text{shamt}[5]=0$  时, 指令才是有效的。

压缩形式: **c.srlrli** rd, shamt

31	26 25	20 19	15 14	12 11	7 6	0
0000000	shamt	rs1	101	rd	0010011	

**srlrliw** rd, rs1, shamt

$$x[rd] = \text{sext}(x[rs1][31:0] \gg_u \text{shamt})$$

立即数逻辑右移字(*Shift Right Logical Word Immediate*). I-type, RV64I only.

把寄存器  $x[rs1]$  右移  $\text{shamt}$  位, 空出的位置填入 0, 结果截为 32 位, 进行有符号扩展后写入  $x[rd]$ 。仅当  $\text{shamt}[5]=0$  时, 指令才是有效的。

31	26 25	20 19	15 14	12 11	7 6	0
0000000	shamt	rs1	101	rd	0011011	

**srlw** rd, rs1, rs2  $x[rd] = \text{sext}(x[rs1][31:0] \gg_u x[rs2][4:0])$

逻辑右移字(*Shift Right Logical Word*). R-type, RV64I only.

把寄存器  $x[rs1]$  的低 32 位右移  $x[rs2]$  位, 空出的位置填入 0, 结果进行有符号扩展后写入  $x[rd]$ 。  $x[rs2]$  的低 5 位代表移动位数, 其高位则被忽略。

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	101	rd	0111011	

**sub** rd, rs1, rs2  $x[rd] = x[rs1] - x[rs2]$

减(*Subtract*). R-type, RV32I and RV64I.

$x[rs1]$  减去  $x[rs2]$ , 结果写入  $x[rd]$ 。忽略算术溢出。

压缩形式: **c.sub** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	000	rd	0110011	

**subw** rd, rs1, rs2  $x[rd] = \text{sext}((x[rs1] - x[rs2])[31:0])$

减去字(*Subtract Word*). R-type, RV64I only.

$x[rs1]$  减去  $x[rs2]$ , 结果截为 32 位, 有符号扩展后写入  $x[rd]$ 。忽略算术溢出。

压缩形式: **c.subw** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	000	rd	0111011	

**tail** symbol  $pc = \&\text{symbol}; \text{clobber } x[6]$

尾调用(*Tail call*). 伪指令(*Pseudoinstruction*), RV32I and RV64I.

设置  $pc$  为  $\text{symbol}$ , 同时覆写  $x[6]$ 。实际扩展为 **auipc**  $x6$ ,  $\text{offsetHi}$  和 **jalr**  $x0$ ,  $\text{offsetLo}(x6)$ 。

**wfi**  $\text{while}(\text{noInterruptPending}) \text{idle}$

等待中断(*Wait for Interrupt*). R-type, RV32I and RV64I 特权指令。

如果没有待处理的中断, 则使处理器处于空闲状态。

31	25 24	20 19	15 14	12 11	7 6	0
0001000	00101	00000	000	00000	1110011	

**XOR** rd, rs1, rs2

$$x[rd] = x[rs1] \wedge x[rs2]$$

异或(*Exclusive-OR*). R-type, RV32I and RV64I.

$x[rs1]$ 和  $x[rs2]$ 按位异或，结果写入  $x[rd]$ 。

压缩形式: **c.xor** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	100	rd	0110011	

**XORI** rd, rs1, immediate

$$x[rd] = x[rs1] \wedge sext(immediate)$$

立即数异或(*Exclusive-OR Immediate*). I-type, RV32I and RV64I.

$x[rs1]$ 和有符号扩展的 *immediate* 按位异或，结果写入  $x[rd]$ 。

压缩形式: **c.xor** rd, rs2

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	100	rd	0010011	