附录 A　RISC-V 指令列表

简约是一切真正优雅的要义。——Coco Chanel，1923

Coco Chanel（1883-1971）香奈儿时装品牌的创始人，她对昂贵的简约的追求塑造了 20 世纪的时尚。

本附录列出了 RV32/64I 的所有指令、本书中涵盖的所有扩展（RVM、RVA、RVF、RVD、RVC 和 RVV）以及所有伪指令。每个条目都包括指令名称、操作数、寄存器传输级定义、指令格式类型、中文描述、压缩版本（如果存在），以及一张带有操作码的指令布局图。我们认为这些摘要对于您了解所有的指令已经足够，但如果您想了解更多细节，请参阅 RISC-V 官方规范[Waterman and Asanovic 2017]。

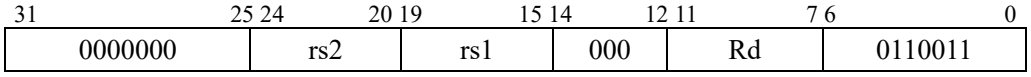
为了帮助读者在本附录中找到所需的指令，左侧（奇数）页面的标题包含该页顶部的第一条指令，右侧（偶数）页面的标题包含该页底部的最后一条指令。格式类似于字典的标题，有助于您搜索单词所在的页面。例如，下一个偶数页的标题是 **AMOADD.W**，这是该页的第一条指令；下一个奇数页的标题是 **AMOMINU.D**，这是该页的最后一条指令。如下是你能在这两页中找到的指令：amoadd.w、adoand.d、amoadn.w、amomax.d、amomax.w、amomaxu.d、amomaxu.w、amomin.d、amomin.w 和 amominu.d。

add rd, rs1, rs2 x[rd] = x[rs1] + x[rs2]

加 (Add). R-type, RV32I and RV64I.

把寄存器 x[*rs2*]加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.add** rd, rs2; **c.mv** rd, rs2



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	000	Rd	0110011

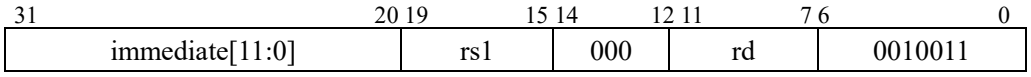
---

addi rd, rs1, immediate x[rd] = x[rs1] + sext(immediate)

加立即数(Add Immediate). I-type, RV32I and RV64I.

把符号位扩展的立即数加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.li** rd, imm; **c.addi** rd, imm; **c.addi16sp** imm; **c.addi4spn** rd, imm



31 20	19 15	14 12	11 7	6 0
immediate[11:0]	rs1	000	rd	0010011

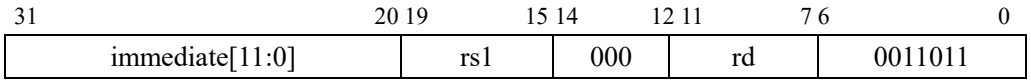
---

addiw rd, rs1, immediate x[rd] = sext((x[rs1] + sext(immediate))[31:0])

加立即数字(Add Word Immediate). I-type, RV64I.

把符号位扩展的立即数加到 x[*rs1*]，将结果截断为 32 位，把符号位扩展的结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.addiw** rd, imm



31 20	19 15	14 12	11 7	6 0
immediate[11:0]	rs1	000	rd	0011011

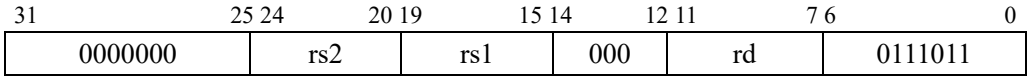
---

addw rd, rs1, rs2 x[rd] = sext((x[rs1] + x[rs2])[31:0])

加字(Add Word). R-type, RV64I.

把寄存器 x[*rs2*]加到寄存器 x[*rs1*]上，将结果截断为 32 位，把符号位扩展的结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.addw** rd, rs2



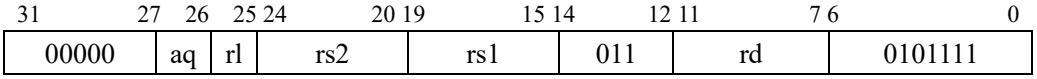
31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	000	rd	0111011

---

amoadd.d rd, rs2, (rs1) x[rd] = AMO64(M[x[rs1]] + x[rs2])

原子加双字(Atomic Memory Operation: Add Doubleword). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 *t*+x[*rs2*]，把 x[*rd*]设为 *t*。



31 27	26	25 24	20 19	15 14	12 11	7 6	0
00000	aq	rl	rs2	rs1	011	rd	0101111

---

amoadd.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] + x[rs2])

原子加字(Atomic Memory Operation: Add Word). R-type, RV32A and RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 *t*+x[*rs2*]，把 x[*rd*]设为符号位扩展的 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
00000	aq	rl	rs2	rs1	010	rd	0101111

---

amoand.d rd, rs2, (rs1)

x[rd] = AMO64(M[x[rs1]] & x[rs2])

原子双字与 (Atomic Memory Operation: AND Doubleword). R-type, RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]位与的结果，把 x[*rd*]设为 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
01100	aq	rl	rs2	rs1	011	rd	0101111

---

amoand.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] & x[rs2])

原子字与 (Atomic Memory Operation: AND Word). R-type, RV32A and RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]位与的结果，把 x[*rd*]设为符号位扩展的 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
01100	aq	rl	rs2	rs1	010	rd	0101111

---

amomax.d rd, rs2, (rs1)

x[rd] = AMO64(M[x[rs1]] MAX x[rs2])

原子最大双字(Atomic Memory Operation: Maximum Doubleword). R-type, RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]中较大的一个（用二进制补码比较），把 x[*rd*]设为 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	aq	rl	rs2	rs1	011	rd	0101111

---

amomax.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] MAX x[rs2])

原子最大字(Atomic Memory Operation: Maximum Word). R-type, RV32A and RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]中较大的一个（用二进制补码比较），把 x[*rd*]设为符号位扩展的 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10100	aq	rl	rs2	rs1	010	rd	0101111

---

amomaxu.d rd, rs2, (rs1)　　x[rd] = AMO64(M[x[rs1]] MAXU x[rs2])

原子无符号最大双字(Atomic Memory Operation: Maximum Doubleword, Unsigned). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]中较大的一个（用无符号比较），把 x[*rd*]设为 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11100	aq	rl	rs2	rs1	011	rd	0101111

---

amomaxu.w rd, rs2, (rs1)　　x[rd] = AMO32(M[x[rs1]] MAXU x[rs2])

原子无符号最大字(Atomic Memory Operation: Maximum Word, Unsigned). R-type, RV32A and RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]中较大的一个（用无符号比较），把 x[*rd*]设为符号位扩展的 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11100	aq	rl	rs2	rs1	010	rd	0101111

---

amomin.d rd, rs2, (rs1)　　x[rd] = AMO64(M[x[rs1]] MIN x[rs2])

原子最小双字(Atomic Memory Operation: Minimum Doubleword). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]中较小的一个（用二进制补码比较），把 x[*rd*]设为 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	aq	rl	rs2	rs1	011	rd	0101111

---

amomin.w rd, rs2, (rs1)　　x[rd] = AMO32(M[x[rs1]] MIN x[rs2])

原子最小字(Atomic Memory Operation: Minimum Word). R-type, RV32A and RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]中较小的一个（用二进制补码比较），把 x[*rd*]设为符号位扩展的 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
10000	aq	rl	rs2	rs1	010	rd	0101111

---

amominu.d rd, rs2,(rs1)　　x[rd] = AMO64(M[x[rs1]] MINU x[rs2])

原子无符号最小双字(Atomic Memory Operation: Minimum Doubleword, Unsigned). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]中较小的一个（用无符号比较），把 x[*rd*]设为 *t*。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
11000	aq	rl	rs2	rs1	011	rd	0101111

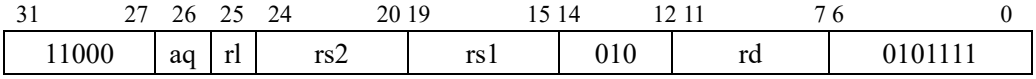
---

amominu.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] MINU x[rs2])

原子无符号最大字(Atomic Memory Operation: Minimum Word, Unsigned). R-type, RV32A and RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]中较小的一个（用无符号比较），把 x[*rd*]设为符号位扩展的 *t*。



31 27	26	25	24 20	19 15	14 12	11 7	6 0
11000	aq	rl	rs2	rs1	010	rd	0101111

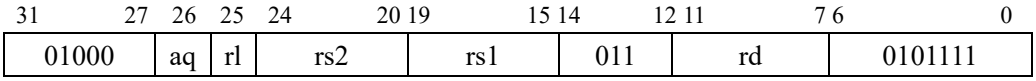
---

amoor.d rd, rs2, (rs1)

x[rd] = AMO64(M[x[rs1]] | x[rs2])

原子双字或 (Atomic Memory Operation: OR Doubleword). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]位或的结果，把 x[*rd*]设为 *t*。



31 27	26	25	24 20	19 15	14 12	11 7	6 0
01000	aq	rl	rs2	rs1	011	rd	0101111

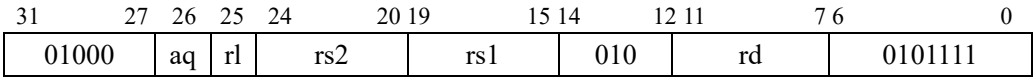
---

amoor.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] | x[rs2])

原子字或 (Atomic Memory Operation: OR Word). R-type, RV32A and RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]位或的结果，把 x[*rd*]设为符号位扩展的 *t*。



31 27	26	25	24 20	19 15	14 12	11 7	6 0
01000	aq	rl	rs2	rs1	010	rd	0101111

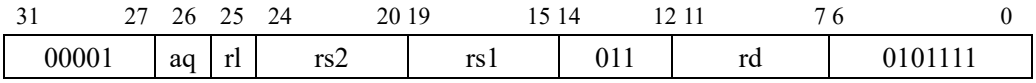
---

amoswap.d rd, rs2, (rs1)

x[rd] = AMO64(M[x[rs1]] SWAP x[rs2])

原子双字交换 (Atomic Memory Operation: Swap Doubleword). R-type, RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 x[*rs2*]的值，把 x[*rd*]设为 *t*。



31 27	26	25	24 20	19 15	14 12	11 7	6 0
00001	aq	rl	rs2	rs1	011	rd	0101111

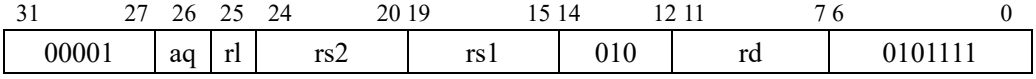
---

amoor.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] SWAP x[rs2])

原子字交换 (Atomic Memory Operation: Swap Word). R-type, RV32A and RV64A.

进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 x[*rs2*]的值，把 x[*rd*]设为符号位扩展的 *t*。



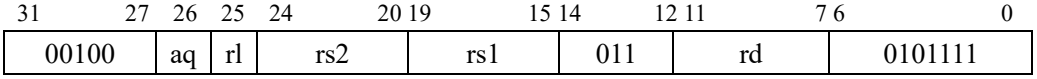
31 27	26	25	24 20	19 15	14 12	11 7	6 0
00001	aq	rl	rs2	rs1	010	rd	0101111

---

amoxor.d rd, rs2, (rs1)

x[rd] = AMO64(M[x[rs1]] ^ x[rs2])

原子双字异或 (Atomic Memory Operation: XOR Doubleword). R-type, RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的双字记为 *t*，把这个双字变为 t 和 x[*rs2*]按位异或的结果，把 x[*rd*]设为 *t*。

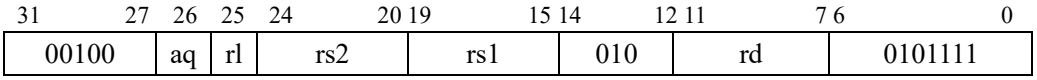


---

amoxor.w rd, rs2, (rs1)

x[rd] = AMO32(M[x[rs1]] ^ x[rs2])

原子字异或 (Atomic Memory Operation: XOR Word). R-type, RV32A and RV64A.
进行如下的原子操作：将内存中地址为 x[*rs1*]中的字记为 *t*，把这个字变为 t 和 x[*rs2*]按位异或的结果，把 x[*rd*]设为符号位扩展的 *t*。

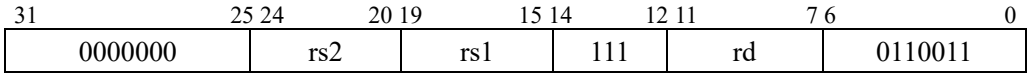


---

and rd, rs1, rs2

x[rd] = x[rs1] & x[rs2]

与 (And). R-type, RV32I and RV64I.
将寄存器 x[*rs1*]和寄存器 x[*rs2*]位与的结果写入 x[*rd*]。
压缩形式：**c.and** rd, rs2

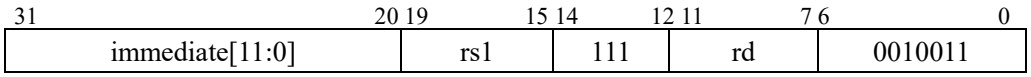


---

andi rd, rs1, immediate

x[rd] = x[rs1] & sext(immediate)

与立即数 (And Immediate). I-type, RV32I and RV64I.
把符号位扩展的立即数和寄存器 x[*rs1*]上的值进行位与，结果写入 x[*rd*]。
压缩形式：**c.andi** rd, imm

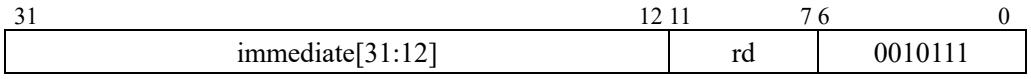


---

auipc rd, immediate

x[rd] = pc + sext(immediate[31:12] << 12)

PC 加立即数 (Add Upper Immediate to PC). U-type, RV32I and RV64I.
把符号位扩展的 20 位（左移 12 位）立即数加到 *pc* 上，结果写入 x[*rd*]。



---

beq rs1, rs2, offset　　if (rs1 == rs2) pc += sext(offset)

相等时分支 *(Branch if Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

压缩形式：**c.beqz** rs1, offset

31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	000	offset[4:1|11]	1100011

---

beqz rs1, offset　　if (rs1 == 0) pc += sext(offset)

等于零时分支 *(Branch if Equal to Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **beq** rs1, x0, offset.

---

bge rs1, rs2, offset　　if (rs1 $\geq_s$ rs2) pc += sext(offset)

大于等于时分支 *(Branch if Greater Than or Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]的值大于等于寄存器 x[*rs2*]的值（均视为二进制补码），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	101	offset[4:1|11]	1100011

---

bgeu rs1, rs2, offset　　if (rs1 $\geq_u$ rs2) pc += sext(offset)

无符号大于等于时分支 *(Branch if Greater Than or Equal, Unsigned)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]的值大于等于寄存器 x[*rs2*]的值（均视为无符号数），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	111	offset[4:1|11]	1100011

---

bgez rs1, offset　　if (rs1 $\geq_s$ 0) pc += sext(offset)

大于等于零时分支 *(Branch if Greater Than or Equal to Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bge** rs1, x0, offset.

---

bgt rs1, rs2, offset　　if (rs1 $>_s$ rs2) pc += sext(offset)

大于时分支 *(Branch if Greater Than)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **blt** rs2, rs1, offset.

---

bgtu rs1, rs2, offset　　if (rs1 $>_u$ rs2) pc += sext(offset)

无符号大于时分支 *(Branch if Greater Than, Unsigned)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bltu** rs2, rs1, offset.

---

bgtz rs1, offset　　if (rs2 $>_s$ 0) pc += sext(offset)

大于零时分支 *(Branch if Greater Than Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **blt** x0, rs2, offset.

---

ble rs1, rs2, offset　　if (rs1 $\leqslant_s$ rs2) pc += sext(offset)

小于等于时分支 *(Branch if Less Than or Equal)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bge** rs2, rs1, offset.

---

bleu rs1, rs2, offset　　if (rs1 $\leqslant_u$ rs2) pc += sext(offset)

小于等于时分支 *(Branch if Less Than or Equal, Unsigned)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bgeu** rs2, rs1, offset.

---

blez rs2, offset　　if (rs2 $\leqslant_s$ 0) pc += sext(offset)

小于等于零时分支 *(Branch if Less Than or Equal to Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bge** x0, rs2, offset.

---

blt rs1, rs2, offset　　if (rs1 $<_s$ rs2) pc += sext(offset)

小于时分支 *(Branch if Less Than)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]的值小于寄存器 x[*rs2*]的值（均视为二进制补码），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	100	offset[4:1|11]	1100011

---

bltz rs2, offset　　if (rs1 $<_s$ 0) pc += sext(offset)

小于零时分支 *(Branch if Less Than Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **blt** rs1, x0, offset.

---

bltu rs1, rs2, offset　　if (rs1 $<_u$ rs2) pc += sext(offset)

无符号小于时分支 *(Branch if Less Than, Unsigned)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]的值小于寄存器 x[*rs2*]的值（均视为无符号数），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	110	offset[4:1|11]	1100011

---

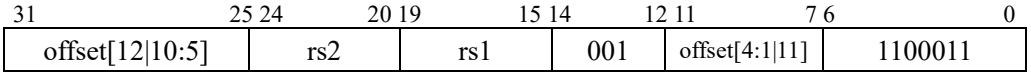
bne rs1, rs2, offset

if (rs1 ≠ rs2) pc += sext(offset)

不相等时分支 *(Branch if Not Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值不相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

压缩形式：**c.bnez** rs1, offset



31 25	24 20	19 15	14 12	11 7	6 0
offset[12|10:5]	rs2	rs1	001	offset[4:1|11]	1100011

---

bnez rs1, offset

if (rs1 ≠ 0) pc += sext(offset)

不等于零时分支 *(Branch if Not Equal to Zero)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

可视为 **bne** rs1, x0, offset.

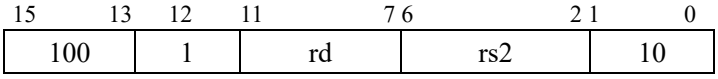
---

c.add rd, rs2

x[rd] = x[rd] + x[rs2]

加 *(Add)*. RV32IC and RV64IC.

扩展形式为 **add** rd, rd, rs2. rd=x0 或 rs2=x0 时非法。



15 13	12	11 7	6 2	1 0
100	1	rd	rs2	10

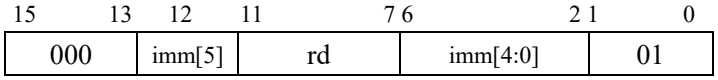
---

c.addi rd, imm

x[rd] = x[rd] + sext(imm)

加立即数 *(Add Immediate)*. RV32IC and RV64IC.

扩展形式为 **addi** rd, rd, imm.



15 13	12	11 7	6 2	1 0
000	imm[5]	rd	imm[4:0]	01

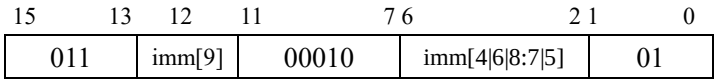
---

c.addi16sp imm

x[2] = x[2] + sext(imm)

加 16 倍立即数到栈指针 *(Add Immediate, Scaled by 16, to Stack Pointer)*. RV32IC and RV64IC.

扩展形式为 **addi** x2, x2, imm. imm=0 时非法。



15 13	12	11 7	6 2	1 0
011	imm[9]	00010	imm[4|6|8:7|5]	01

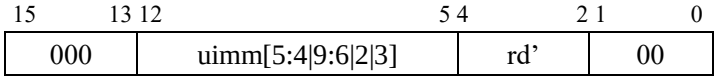
---

c.addi4spn rd', uimm

x[8+rd'] = x[2] + uimm

加 4 倍立即数到栈指针 *(Add Immediate, Scaled by 4, to Stack Pointer, Nondestructive)*. RV32IC and RV64IC.

扩展形式为 **addi** rd, x2, uimm, 其中 rd=8+rd'. uimm=0 时非法。



15 13	12 5	4 2	1 0
000	uimm[5:4|9:6|2|3]	rd'	00

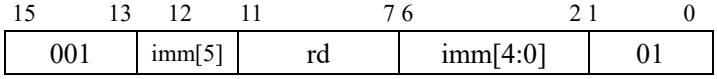
---

c.addiw rd, imm

x[rd] = sext((x[rd] + sext(imm))[31:0])

加立即数字 *(Add Word Immediate)*. RV64IC.

扩展形式为 **addiw** rd, rd, imm. rd=x0 时非法。



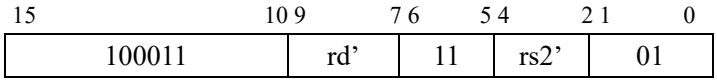
---

c.and rd', rs2'

x[8+rd'] = x[8+rd'] & x[8+rs2']

与 *(AND)*. RV32IC and RV64IC.

扩展形式为 **and** rd, rd, rs2, 其中 rd=8+rd', rs2=8+rs2'.



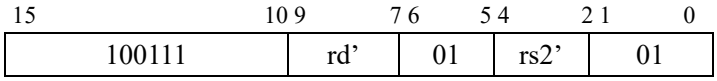
---

c.addw rd', rs2'

x[8+rd'] = sext((x[8+rd'] + x[8+rs2'])[31:0])

加字 *(Add Word)*. RV64IC.

扩展形式为 **addw** rd, rd, rs2, 其中 rd=8+rd', rs2=8+rs2'.



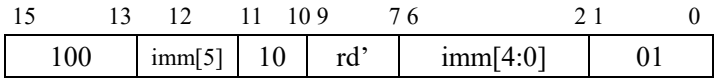
---

c.andi rd', imm

x[8+rd'] = x[8+rd'] & sext(imm)

与立即数 *(AND Immediate)*. RV32IC and RV64IC.

扩展形式为 **andi** rd, rd, imm, 其中 rd=8+rd'.



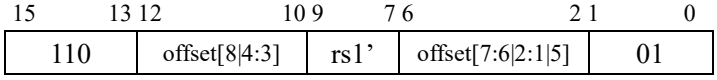
---

c.beqz rs1', offset

if (x[8+rs1'] == 0) pc += sext(offset)

等于零时分支 *(Branch if Equal to Zero)*. RV32IC and RV64IC.

扩展形式为 **beq** rs1, x0, offset, 其中 rs1=8+rs1'.



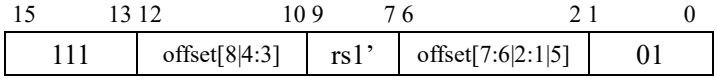
---

c.bnez rs1', offset

if (x[8+rs1'] ≠ 0) pc += sext(offset)

不等于零时分支 *(Branch if Not Equal to Zero)*. RV32IC and RV64IC.

扩展形式为 **bne** rs1, x0, offset, 其中 rs1=8+rs1'.



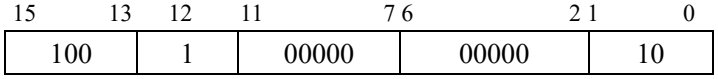
---

c.ebreak

RaiseException(Breakpoint)

环境断点 *(Environment Breakpoint)*. RV32IC and RV64IC.

扩展形式为 **ebreak**.



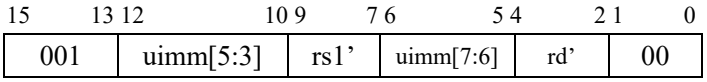
---

c.fld rd', uimm(rs1')

f[8+rd'] = M[x[8+rs1'] + uimm][63:0]

浮点双字加载 *(Floating-point Load Doubleword)*. RV32DC and RV64DC.

扩展形式为 **fld** rd, uimm(rs1), 其中 rd=8+rd', rs1=8+rs1'.



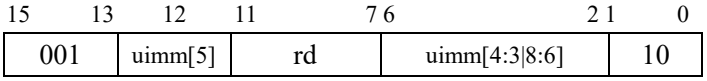
---

c.fldsp rd, uimm(x2)

f[rd] = M[x[2] + uimm][63:0]

栈指针相关浮点双字加载 *(Floating-point Load Doubleword, Stack-Pointer Relative)*. RV32DC and RV64DC.

扩展形式为 **fld** rd, uimm(x2).



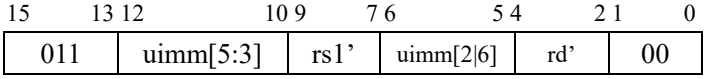
---

c.flw rd', uimm(rs1')

f[8+rd'] = M[x[8+rs1'] + uimm][31:0]

浮点字加载 *(Floating-point Load Word)*. RV32FC.

扩展形式为 **flw** rd, uimm(rs1), 其中 rd=8+rd', rs1=8+rs1'.



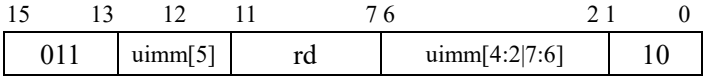
---

c.flwsp rd, uimm(x2)

f[rd] = M[x[2] + uimm][31:0]

栈指针相关浮点字加载 *(Floating-point Load Word, Stack-Pointer Relative)*. RV32FC.

扩展形式为 **flw** rd, uimm(x2).



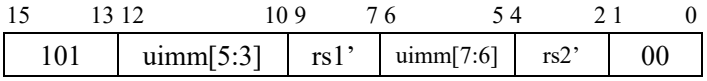
---

c.fsd rs2', uimm(rs1')

M[x[8+rs1'] + uimm][63:0] = f[8+rs2']

浮点双字存储 *(Floating-point Store Doubleword)*. RV32DC and RV64DC.

扩展形式为 **fsd** rs2, uimm(rs1), 其中 rs2=8+rs2', rs1=8+rs1'.



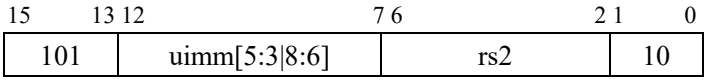
---

c.fsdsp rs2, uimm(x2)

M[x[2] + uimm][63:0] = f[rs2]

栈指针相关浮点双字存储 *(Floating-point Store Doubleword, Stack-Pointer Relative)*. RV32DC and RV64DC.

扩展形式为 **fsd** rs2, uimm(x2).



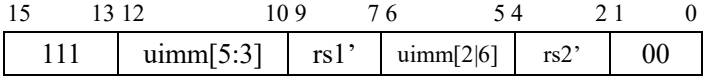
---

c.fsw rs2', uimm(rs1')

M[x[8+rs1'] + uimm][31:0] = f[8+rs2']

浮点字存储 *(Floating-point Store Word)*. RV32FC.

扩展形式为 **fsw** rs2, uimm(rs1), 其中 rs2=8+rs2', rs1=8+rs1'.



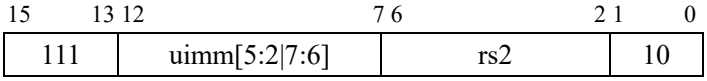
---

c.fswsp rs2, uimm(x2)

M[x[2] + uimm][31:0] = f[rs2]

栈指针相关浮点字存储 *(Floating-point Store Word, Stack-Pointer Relative)*. RV32FC.

扩展形式为 **fsw** rs2, uimm(x2).



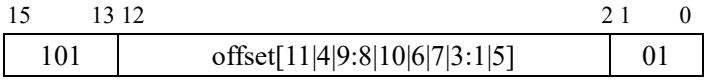
---

c.j offset

pc += sext(offset)

跳转 *(Jump)*. RV32IC and RV64IC.

扩展形式为 **jal** x0, offset.



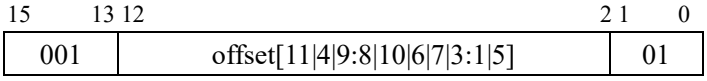
---

c.jal offset

x[1] = pc+2; pc += sext(offset)

链接跳转 *(Jump and Link)*. RV32IC.

扩展形式为 **jal** x1, offset.



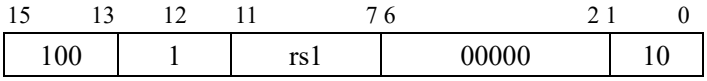
---

c.jalr rs1

t = pc+2; pc = x[rs1]; x[1] = t

寄存器链接跳转 *(Jump and Link Register)*. RV32IC and RV64IC.

扩展形式为 **jalr** x1, 0(rs1). 当 rs1=x0 时非法。



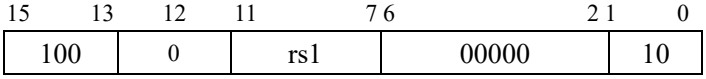
---

c.jr rs1

pc = x[rs1]

寄存器跳转 *(Jump Register)*. RV32IC and RV64IC.

扩展形式为 **jalr** x0, 0(rs1). 当 rs1=x0 时非法。



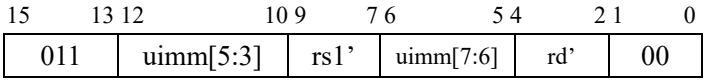
---

c.ld rd', uimm(rs1')

x[8+rd'] = M[x[8+rs1'] + uimm][63:0]

双字加载 *(Load Doubleword)*. RV64IC.

扩展形式为 **ld** rd, uimm(rs1), 其中 rd=8+rd', rs1=8+rs1'.



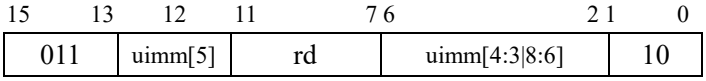
---

c.ldsp rd, uimm(x2)

x[rd] = M[x[2] + uimm][63:0]

栈指针相关双字加载 *(Load Doubleword, Stack-Pointer Relative)*. RV64IC.

扩展形式为 **ld** rd, uimm(x2). rd=x0 时非法。



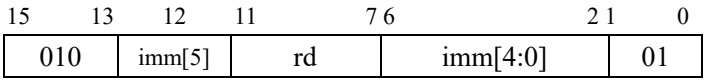
---

c.li rd, imm

x[rd] = sext(imm)

立即数加载 *(Load Immediate)*. RV32IC and RV64IC.

扩展形式为 **addi** rd, x0, imm.



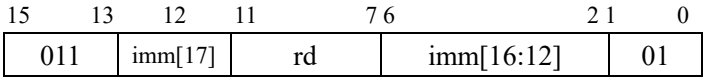
---

c.lui rd, imm

x[rd] = sext(imm[17:12] << 12)

高位立即数加载 *(Load Upper Immediate)*. RV32IC and RV64IC.

扩展形式为 **lui** rd, imm. 当 rd=x2 或 imm=0 时非法。



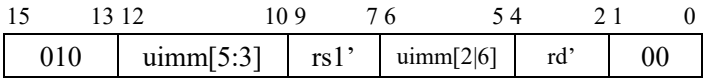
---

c.lw rd', uimm(rs1')

x[8+rd'] = sext(M[x[8+rs1'] + uimm][31:0])

字加载 *(Load Word)*. RV32IC and RV64IC.

扩展形式为 **lw** rd, uimm(rs1), 其中 rd=8+rd', rs1=8+rs1'.



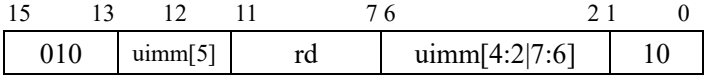
---

c.lwsp rd, uimm(x2)

x[rd] = sext(M[x[2] + uimm][31:0])

栈指针相关字加载 *(Load Word, Stack-Pointer Relative)*. RV32IC and RV64IC.

扩展形式为 **lw** rd, uimm(x2). rd=x0 时非法。



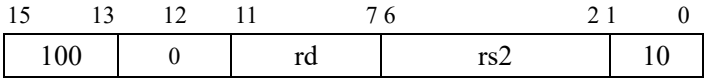
---

c.mv rd, rs2

x[rd] = x[rs2]

移动 *(Move)*. RV32IC and RV64IC.

扩展形式为 **add** rd, x0, rs2. rs2=x0 时非法。



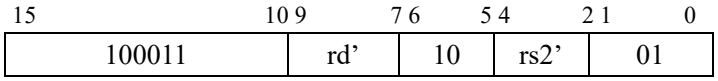
---

c.or rd', rs2'

x[8+rd'] = x[8+rd'] | x[8+rs2']

或 *(OR)*. RV32IC and RV64IC.

扩展形式为 **or** rd, rd, rs2, 其中 rd=8+rd', rs2=8+rs2'.



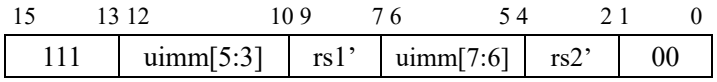
---

c.sd rs2', uimm(rs1')

M[x[8+rs1'] + uimm][63:0] = x[8+rs2']

双字存储*(Store Doubleword)*. RV64IC.

扩展形式为 **sd** rs2, uimm(rs1), 其中 rs2=8+rs2', rs1=8+rs1'.



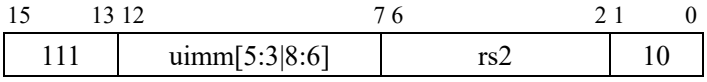
---

c.sdsp rs2, uimm(x2)

M[x[2] + uimm][63:0] = x[rs2]

栈指针相关双字存储 *(Store Doubleword, Stack-Pointer Relative)*. RV64IC.

扩展形式为 **sd** rs2, uimm(x2).



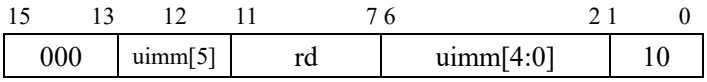
---

c.slli rd, uimm

x[rd] = x[rd] << uimm

立即数逻辑左移 *(Shift Left Logical Immediate)*. RV32IC and RV64IC.

扩展形式为 **slli** rd, rd, uimm.



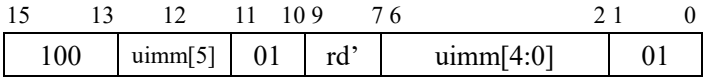
---

c.srai rd', uimm

x[8+rd'] = x[8+rd'] >>$_s$ uimm

立即数算术右移 *(Shift Right Arithmetic Immediate)*. RV32IC and RV64IC.

扩展形式为 **srai** rd, rd, uimm, 其中 rd=8+rd'.



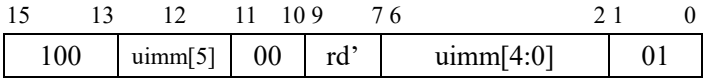
---

c.srli rd', uimm

x[8+rd'] = x[8+rd'] >>$_u$ uimm

立即数逻辑右移 *(Shift Right Logical Immediate)*. RV32IC and RV64IC.

扩展形式为 **srli** rd, rd, uimm, 其中 rd=8+rd'.



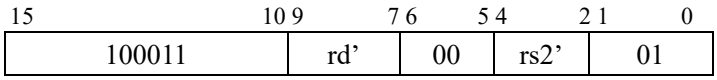
---

c.sub rd', rs2'

x[8+rd'] = x[8+rd'] - x[8+rs2']

减 *(Subtract)*. RV32IC and RV64IC.

扩展形式为 **sub** rd, rd, rs2. 其中 rd=8+rd', rs2=8+rs2'. .



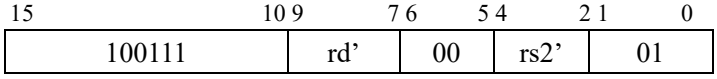
---

c.subw rd', rs2'

x[8+rd'] = sext((x[8+rd'] - x[8+rs2'])[31:0])

减字 *(Subtract Word)*. RV64IC.

扩展形式为 **subw** rd, rd, rs2. 其中 rd=8+rd', rs2=8+rs2'. .



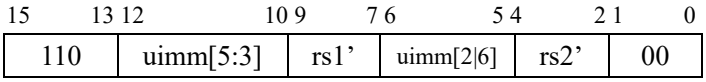
---

c.sw rs2', uimm(rs1')

M[x[8+rs1'] + uimm][31:0] = x[8+rs2']

字存储 *(Store Word)*. RV32IC and RV64IC.

扩展形式为 **sw** rs2, uimm(rs1), 其中 rs2=8+rs2', rs1=8+rs1'.



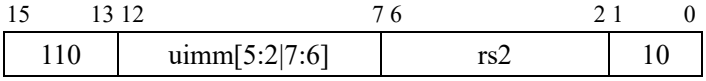
---

c.swsp rs2, uimm(x2)

M[x[2] + uimm][31:0] = x[rs2]

栈指针相关字存储 *(Store Word, Stack-Pointer Relative)*. RV32IC and RV64IC.

扩展形式为 **sw** rs2, uimm(x2).

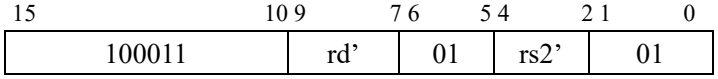


---

c.xor rd', rs2' x[8+rd'] = x[8+rd'] ^ x[8+rs2']

异或 *(Exclusive-OR)*. RV32IC and RV64IC.

扩展形式为 **xor** rd, rd, rs2, 其中 rd=8+rd', rs2=8+rs2'.



---

call rd, symbol x[rd] = pc+8; pc = &symbol

调用 *(Call)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

把下一条指令的地址（*pc*+8）写入 x[*rd*]，然后把 *pc* 设为 *symbol*。等同于 **auipc** rd, offestHi, 再加上一条 **jalr** rd, offsetLo(rd). 若省略了 *rd*，默认为 x1.

---

csrr rd, csr x[rd] = CSRs[csr]

读控制状态寄存器 *(Control and Status Register Read)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

把控制状态寄存器 *csr* 的值写入 x[*rd*]，等同于 **csrrs** rd, csr, x0.

---

csrc csr, rs1 CSRs[csr] &= ~x[rs1]

清除控制状态寄存器 *(Control and Status Register Clear)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

对于 x[*rs1*]中每一个为 1 的位，把控制状态寄存器 *csr* 的的对应位清零，等同于 **csrrc** x0, csr, rs1.

---

csrci csr, zimm[4:0] CSRs[csr] &= ~zimm

立即数清除控制状态寄存器 *(Control and Status Register Clear Immediate)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

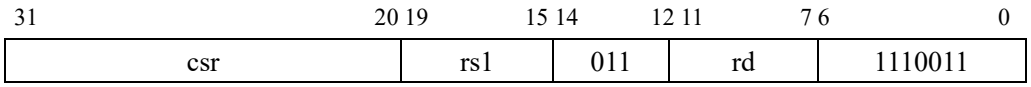
对于五位的零扩展的立即数中每一个为 1 的位，把控制状态寄存器 *csr* 的的对应位清零，等同于 **csrrci** x0, csr, zimm.

---

csrrc rd, csr, rs1 t = CSRs[csr]; CSRs[csr] = t &~x[rs1]; x[rd] = t

读后清除控制状态寄存器 *(Control and Status Register Read and Clear)*. I-type, RV32I and RV64I.

记控制状态寄存器 *csr* 中的值为 *t*。把 *t* 和寄存器 x[*rs1*]按位与的结果写入 *csr*，再把 *t* 写入 x[*rd*]。



---

csrrci rd, csr, zimm[4:0]　　t = CSRs[csr]; CSRs[csr] = t &~zimm; x[rd] = t

立即数读后清除控制状态寄存器 (Control and Status Register Read and Clear Immediate). I-type, RV32I and RV64I.

记控制状态寄存器 *csr* 中的值为 *t*。把 *t* 和五位的零扩展的立即数 *zimm* 按位与的结果写入 *csr*，再把 *t* 写入 x[*rd*]（*csr* 寄存器的第 5 位及更高位不变）。

31 20	19 15	14 12	11 7	6 0
csr	zimm[4:0]	111	rd	1110011

---

csrrs rd, csr, rs1　　t = CSRs[csr]; CSRs[csr] = t | x[rs1]; x[rd] = t

读后置位控制状态寄存器 (Control and Status Register Read and Set). I-type, RV32I and RV64I.

记控制状态寄存器 *csr* 中的值为 *t*。把 *t* 和寄存器 x[*rs1*]按位或的结果写入 *csr*，再把 *t* 写入 x[*rd*]。

31 20	19 15	14 12	11 7	6 0
csr	rs1	010	rd	1110011

---

csrrci rd, csr, zimm[4:0]　　t = CSRs[csr]; CSRs[csr] = t | zimm; x[rd] = t

立即数读后设置控制状态寄存器 (Control and Status Register Read and Set Immediate). I-type, RV32I and RV64I.

记控制状态寄存器 *csr* 中的值为 *t*。把 *t* 和五位的零扩展的立即数 *zimm* 按位或的结果写入 *csr*，再把 *t* 写入 x[*rd*]（*csr* 寄存器的第 5 位及更高位不变）。

31 20	19 15	14 12	11 7	6 0
csr	zimm[4:0]	110	rd	1110011

---

csrrw rd, csr, zimm[4:0]　　t = CSRs[csr]; CSRs[csr] = x[rs1]; x[rd] = t

读后写控制状态寄存器 (Control and Status Register Read and Write). I-type, RV32I and RV64I.

记控制状态寄存器 *csr* 中的值为 *t*。把寄存器 x[*rs1*]的值写入 *csr*，再把 *t* 写入 x[*rd*]。

31 20	19 15	14 12	11 7	6 0
csr	rs1	001	rd	1110011

---

csrrwi rd, csr, zimm[4:0]　　x[rd] = CSRs[csr]; CSRs[csr] = zimm

立即数读后写控制状态寄存器 (Control and Status Register Read and Write Immediate). I-type, RV32I and RV64I.

把控制状态寄存器 *csr* 中的值拷贝到 x[*rd*]中，再把五位的零扩展的立即数 *zimm* 的值写入 *csr*。

31 20	19 15	14 12	11 7	6 0
csr	zimm[4:0]	101	rd	1110011

---

csrc csr, rs1 CSRs[csr] |= x[rs1]

置位控制状态寄存器 *(Control and Status Register Set)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

对于 x[*rs1*]中每一个为 1 的位，把控制状态寄存器 *csr* 的的对应位置位，等同于 **csrrs** x0, csr, rs1.

---

csrci csr, zimm[4:0] CSRs[csr] |= zimm

立即数置位控制状态寄存器 *(Control and Status Register Set Immediate)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

对于五位的零扩展的立即数中每一个为 1 的位，把控制状态寄存器 *csr* 的的对应位清零，等同于 **csrrsi** x0, csr, zimm.

---

csrw csr, rs1 CSRs[csr] = x[rs1]

写控制状态寄存器 *(Control and Status Register Set)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

对于 x[*rs1*]中每一个为 1 的位，把控制状态寄存器 *csr* 的的对应位置位，等同于 **csrrs** x0, csr, rs1.

---

csrwi csr, zimm[4:0] CSRs[csr] = zimm

立即数写控制状态寄存器 *(Control and Status Register Write Immediate)*. 伪指令(Pesudoinstruction), RV32I and RV64I.

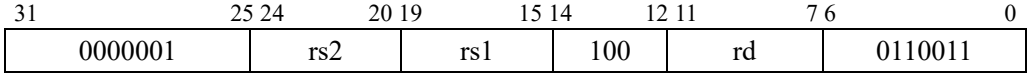
把五位的零扩展的立即数的值写入控制状态寄存器 *csr* 的，等同于 **csrrwi** x0, csr, zimm.

---

div rd, rs1, rs2 $x[rd] = x[rs1] \div_s x[rs2]$

除法*(Divide)*. R-type, RV32M and RV64M.

用寄存器 x[*rs1*]的值除以寄存器 x[*rs2*]的值，向零舍入，将这些数视为二进制补码，把商写入 x[*rd*]。

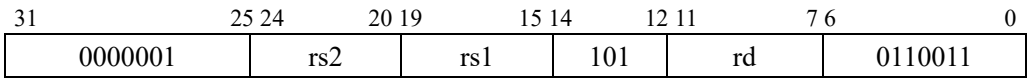


---

divu rd, rs1, rs2 $x[rd] = x[rs1] \div_u x[rs2]$

无符号除法*(Divide, Unsigned)*. R-type, RV32M and RV64M.

用寄存器 x[*rs1*]的值除以寄存器 x[*rs2*]的值，向零舍入，将这些数视为无符号数，把商写入 x[*rd*]。



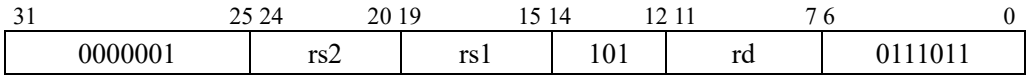
---

divuw rd, rs1, rs2

x[rd] = sext(x[rs1][31:0] $\div_u$ x[rs2][31:0])

无符号字除法(Divide Word, Unsigned). R-type, RV64M.

用寄存器 x[*rs1*]的低 32 位除以寄存器 x[*rs2*]的低 32 位，向零舍入，将这些数视为无符号数，把经符号位扩展的 32 位商写入 x[*rd*]。



31–25	24–20	19–15	14–12	11–7	6–0
0000001	rs2	rs1	101	rd	0111011

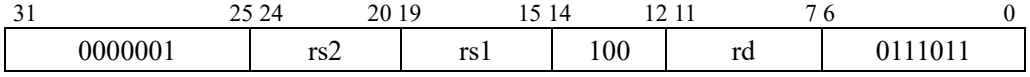
---

divw rd, rs1, rs2

x[rd] = sext(x[rs1][31:0] $\div_s$ x[rs2][31:0])

字除法(Divide Word). R-type, RV64M.

用寄存器 x[*rs1*]的低 32 位除以寄存器 x[*rs2*]的低 32 位，向零舍入，将这些数视为二进制补码，把经符号位扩展的 32 位商写入 x[*rd*]。



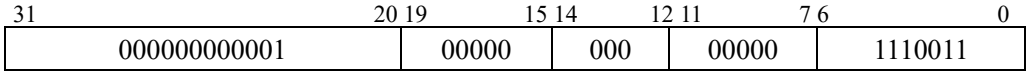
31–25	24–20	19–15	14–12	11–7	6–0
0000001	rs2	rs1	100	rd	0111011

Ebreak

RaiseException(Breakpoint)

环境断点 (Environment Breakpoint). I-type, RV32I and RV64I.

通过抛出断点异常的方式请求调试器。



31–20	19–15	14–12	11–7	6–0
000000000001	00000	000	00000	1110011

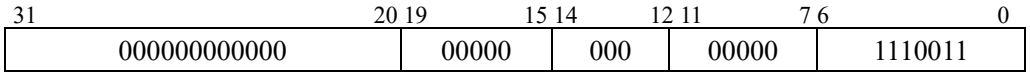
---

ecall

RaiseException(EnvironmentCall)

环境调用 (Environment Call). I-type, RV32I and RV64I.

通过引发环境调用异常来请求执行环境。



31–20	19–15	14–12	11–7	6–0
000000000000	00000	000	00000	1110011

---

fabs.d rd, rs1

f[rd] = |f[rs1]|

浮点数绝对值 (Floating-point Absolute Value). *伪指令*(Pesudoinstruction), RV32D and RV64D.

把双精度浮点数 f[*rs1*]的绝对值写入 f[*rd*]。

等同于 **fsgnjx.d** rd, rs1, rs1.

---

fabs.s rd, rs1

f[rd] = |f[rs1]|

浮点数绝对值 (Floating-point Absolute Value). *伪指令*(Pesudoinstruction), RV32F and RV64F.

把单精度浮点数 f[*rs1*]的绝对值写入 f[*rd*]。

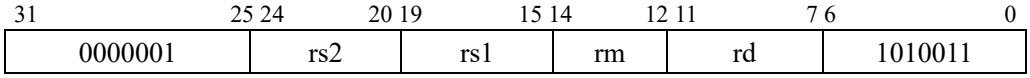
等同于 **fsgnjx.s** rd, rs1, rs1.

---

fadd.d rd, rs1, rs2

f [rd] = f [rs1] + f [rs2]

双精度浮点加*(Floating-point Add, Double-Precision)*. R-type, RV32D and RV64D.
把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相加，并将舍入后的和写入 f[*rd*]。



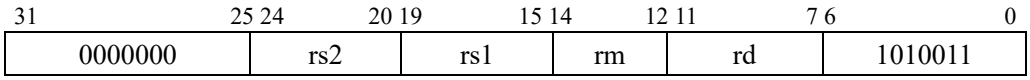
31 25	24 20	19 15	14 12	11 7	6 0
0000001	rs2	rs1	rm	rd	1010011

---

fadd.s rd, rs1, rs2

f [rd] = f [rs1] + f [rs2]

单精度浮点加*(Floating-point Add, Single-Precision)*. R-type, RV32F and RV64F.
把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相加，并将舍入后的和写入 f[*rd*]。



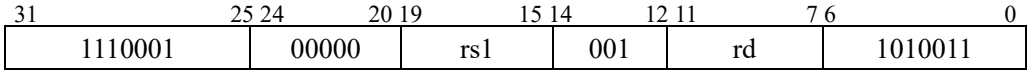
31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	rm	rd	1010011

---

fclass.d rd, rs1, rs2

x[rd] = classify$_d$(f[rs1])

双精度浮点分类*(Floating-point Classify, Double-Precision)*. R-type, RV32D and RV64D.
把一个表示寄存器 f[*rs1*]中双精度浮点数类别的掩码写入 x[*rd*]中。关于如何解释写入 x[*rd*]的值，请参阅指令 **fclass.s** 的介绍。



31 25	24 20	19 15	14 12	11 7	6 0
1110001	00000	rs1	001	rd	1010011

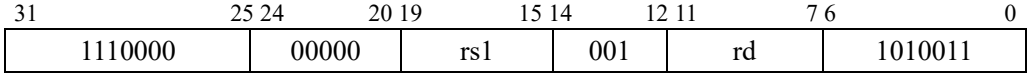
---

fclass.s rd, rs1, rs2

x[rd] = classify$_s$(f[rs1])

单精度浮点分类*(Floating-point Classify, Single-Precision)*. R-type, RV32F and RV64F.
把一个表示寄存器 f[*rs1*]中单精度浮点数类别的掩码写入 x[*rd*]中。x[*rd*]中有且仅有一位被置上，见下表。

x*[rd]*位	含义
0	f [*rs1*]为−∞。
1	f [*rs1*]是负规格化数。
2	f [*rs1*]是负的非规格化数。
3	f [*rs1*]是-0。
4	f [*rs1*]是+0。
5	f [*rs1*]是正的非规格化数。
6	f [*rs1*]是正的规格化数。
7	f [*rs1*]为+∞。
8	f [*rs1*]是信号(signaling)NaN。
9	f [*rs1*]是一个安静(quiet)NaN。



31 25	24 20	19 15	14 12	11 7	6 0
1110000	00000	rs1	001	rd	1010011

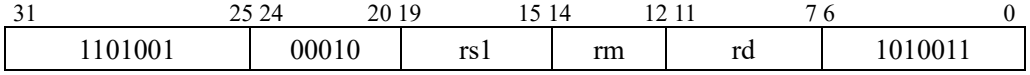
---

fcvt.d.l rd, rs1, rs2

f[rd] = $f64_{s64}$(x[rs1])

长整型向双精度浮点转换*(Floating-point Convert to Double from Long)*. R-type, RV64D.

把寄存器 x[*rs1*]中的 64 位二进制补码表示的整数转化为双精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101001	00010	rs1	rm	rd	1010011

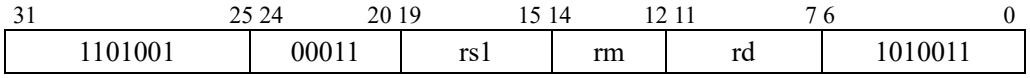
---

fcvt.d.lu rd, rs1, rs2

f[rd] = $f64_{u64}$(x[rs1])

无符号长整型向双精度浮点转换*(Floating-point Convert to Double from Unsigned Long)*. R-type, RV64D.

把寄存器 x[*rs1*]中的 64 位无符号整数转化为双精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101001	00011	rs1	rm	rd	1010011

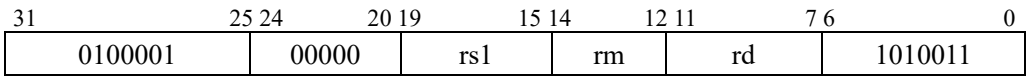
---

fcvt.d.s rd, rs1, rs2

f[rd] = $f64_{f32}$(f[rs1])

单精度向双精度浮点转换*(Floating-point Convert to Double from Single)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]中的单精度浮点数转化为双精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
0100001	00000	rs1	rm	rd	1010011

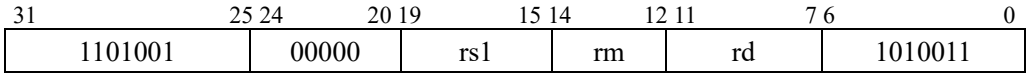
---

fcvt.d.w rd, rs1, rs2

f[rd] = $f64_{s32}$(x[rs1])

字向双精度浮点转换*(Floating-point Convert to Double from Word)*. R-type, RV32D and RV64D.

把寄存器 x[*rs1*]中的 32 位二进制补码表示的整数转化为双精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101001	00000	rs1	rm	rd	1010011

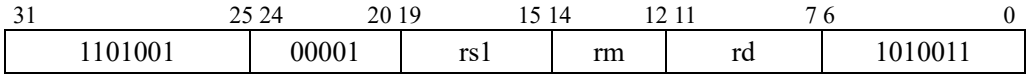
---

fcvt.d.wu rd, rs1, rs2

f[rd] = $f64_{u32}$(x[rs1])

无符号字向双精度浮点转换*(Floating-point Convert to Double from Unsigned Word)*. R-type, RV32D and RV64D.

把寄存器 x[*rs1*]中的 32 位无符号整数转化为双精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101001	00001	rs1	rm	rd	1010011

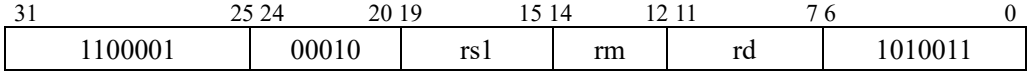
---

fcvt.l.d rd, rs1, rs2

x[rd] = $s64_{f64}$(f[rs1])

双精度浮点向长整型转换(Floating-point Convert to Long from Double). R-type, RV64D.

把寄存器 f[*rs1*]中的双精度浮点数转化为 64 位二进制补码表示的整数，再写入 x[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1100001	00010	rs1	rm	rd	1010011

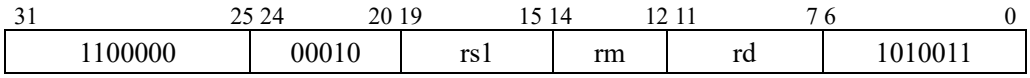
---

fcvt.l.s rd, rs1, rs2

x[rd] = $s64_{f32}$(f[rs1])

单精度浮点向长整型转换(Floating-point Convert to Long from Single). R-type, RV64F.

把寄存器 f[*rs1*]中的单精度浮点数转化为 64 位二进制补码表示的整数，再写入 x[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1100000	00010	rs1	rm	rd	1010011

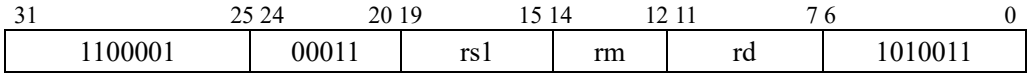
---

fcvt.lu.d rd, rs1, rs2

x[rd] = $u64_{f64}$(f[rs1])

双精度浮点向无符号长整型转换(Floating-point Convert to Unsigned Long from Double). R-type, RV64D.

把寄存器 f[*rs1*]中的双精度浮点数转化为 64 位无符号整数，再写入 x[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1100001	00011	rs1	rm	rd	1010011

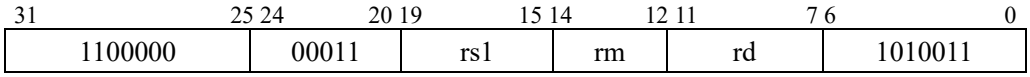
---

fcvt.lu.s rd, rs1, rs2

x[rd] = $u64_{f32}$(f[rs1])

单精度浮点向无符号长整型转换(Floating-point Convert to Unsigned Long from Single). R-type, RV64F.

把寄存器 f[*rs1*]中的单精度浮点数转化为 64 位二进制补码表示的整数，再写入 x[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1100000	00011	rs1	rm	rd	1010011

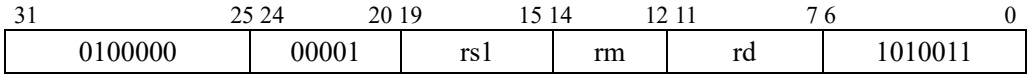
---

fcvt.s.d rd, rs1, rs2

f[rd] = $f32_{f64}$(f[rs1])

双精度向单精度浮点转换(Floating-point Convert to Single from Double). R-type, RV32D and RV64D.

把寄存器 f[*rs1*]中的双精度浮点数转化为单精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
0100000	00001	rs1	rm	rd	1010011

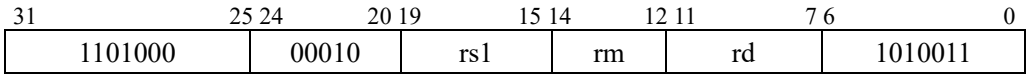
---

fcvt.s.l rd, rs1, rs2

f[rd] = $f32_{s64}$(x[rs1])

长整型向单精度浮点转换*(Floating-point Convert to Single from Long)*. R-type, RV64F.

把寄存器 x[*rs1*]中的 64 位二进制补码表示的整数转化为单精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101000	00010	rs1	rm	rd	1010011

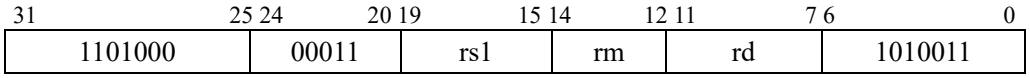
---

fcvt.s.lu rd, rs1, rs2

f[rd] = $f32_{u64}$(x[rs1])

无符号长整型向单精度浮点转换*(Floating-point Convert to Single from Unsigned Long)*. R-type, RV64F.

把寄存器 x[*rs1*]中的 64 位的无符号整数转化为单精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101000	00011	rs1	rm	rd	1010011

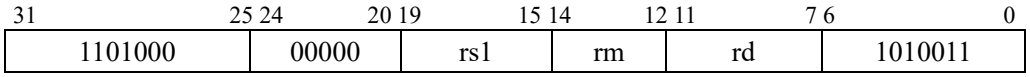
---

fcvt.s.w rd, rs1, rs2

f[rd] = $f32_{s32}$(x[rs1])

字向单精度浮点转换*(Floating-point Convert to Single from Word)*. R-type, RV32F and RV64F.

把寄存器 x[*rs1*]中的 32 位二进制补码表示的整数转化为单精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101000	00000	rs1	rm	rd	1010011

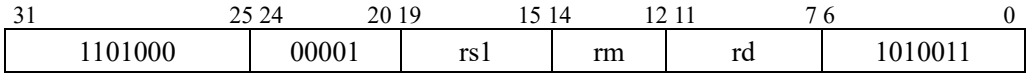
---

fcvt.s.wu rd, rs1, rs2

f[rd] = $f32_{u32}$(x[rs1])

无符号字向单精度浮点转换*(Floating-point Convert to Single from Unsigned Word)*. R-type, RV32F and RV64F.

把寄存器 x[*rs1*]中的 32 位无符号整数转化为单精度浮点数，再写入 f[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1101000	00001	rs1	rm	rd	1010011

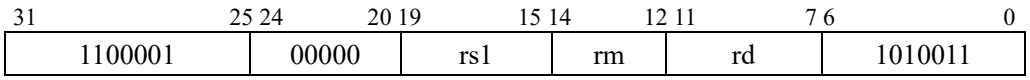
---

fcvt.w.d rd, rs1, rs2

x[rd] = sext($s32_{f64}$(f[rs1]))

双精度浮点向字转换*(Floating-point Convert to Word from Double)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]中的双精度浮点数转化为 32 位二进制补码表示的整数，再写入 x[*rd*]中。



31 25	24 20	19 15	14 12	11 7	6 0
1100001	00000	rs1	rm	rd	1010011

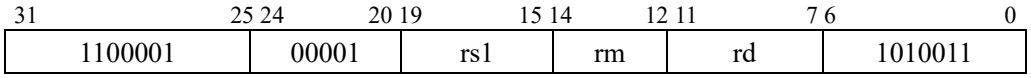
---

fcvt.wu.d rd, rs1, rs2

x[rd] = sext($u32_{f64}$(f[rs1]))

双精度浮点向无符号字转换(Floating-point Convert to Unsigned Word from Double). R-type, RV32D and RV64D.

把寄存器 f[*rs1*]中的双精度浮点数转化为 32 位无符号整数，再写入 x[*rd*]中。



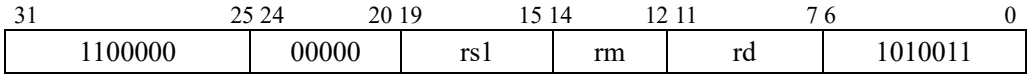
---

fcvt.w.s rd, rs1, rs2

x[rd] = sext($s32_{f32}$(f[rs1]))

单精度浮点向字转换(Floating-point Convert to Word from Single). R-type, RV32F and RV64F.

把寄存器 f[*rs1*]中的单精度浮点数转化为 32 位二进制补码表示的整数，再写入 x[*rd*]中。



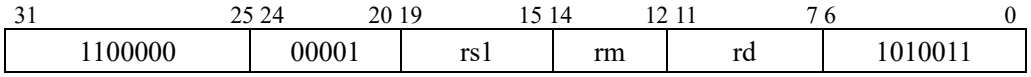
---

fcvt.wu.s rd, rs1, rs2

x[rd] = sext($u32_{f32}$(f[rs1]))

单精度浮点向无符号字转换(Floating-point Convert to Unsigned Word from Single). R-type, RV32F and RV64F.

把寄存器 f[*rs1*]中的单精度浮点数转化为 32 位无符号整数，再写入 x[*rd*]中。



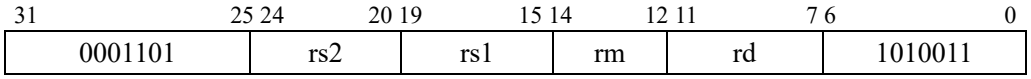
---

fdiv.d rd, rs1, rs2

f[rd] = f[rs1] ÷ f[rs2]

双精度浮点除法(Floating-point Divide, Double-Precision). R-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相除，并将舍入后的商写入 f[*rd*]。



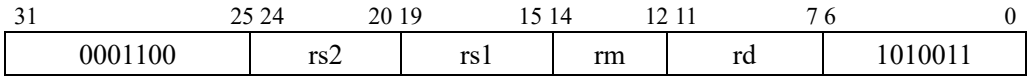
---

fdiv.s rd, rs1, rs2

f[rd] = f[rs1] ÷ f[rs2]

单精度浮点除法(Floating-point Divide, Single-Precision). R-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相除，并将舍入后的商写入 f[*rd*]。



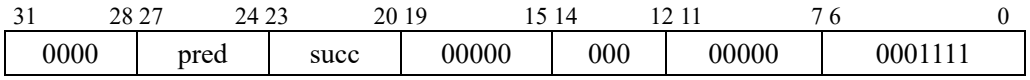
---

fence pred, succ

Fence(pred, succ)

同步内存和 I/O*(Fence Memory and I/O)*. I-type, RV32I and RV64I.

在后续指令中的内存和 I/O 访问对外部（例如其他线程）可见之前，使这条指令之前的内存及 I/O 访问对外部可见。比特中的第 3,2,1 和 0 位分别对应于设备输入，设备输出，内存读写。例如 **fence** r, rw，将前面读取与后面的读取和写入排序，使用 *pred*=0010 和 *succ*=0011 进行编码。如果省略了参数，则表示 **fence** iorw, iorw，即对所有访存请求进行排序。



31–28	27–24	23–20	19–15	14–12	11–7	6–0
0000	pred	succ	00000	000	00000	0001111

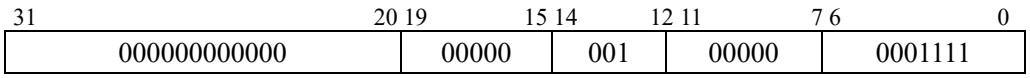
---

fence.i

Fence(Store, Fetch)

同步指令流*(Fence Instruction Stream)*. I-type, RV32I and RV64I.

使对内存指令区域的读写，对后续取指令可见。



31–20	19–15	14–12	11–7	6–0
000000000000	00000	001	00000	0001111

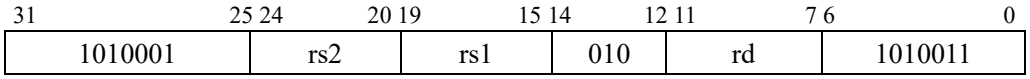
---

feq.d rd, rs1, rs2

x[rd] = f[rs1] == f[rs2]

双精度浮点相等*(Floating-point Equals, Double-Precision)*. R-type, RV32D and RV64D.

若寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相等，则在 x[*rd*]中写入 1，反之写 0。



31–25	24–20	19–15	14–12	11–7	6–0
1010001	rs2	rs1	010	rd	1010011

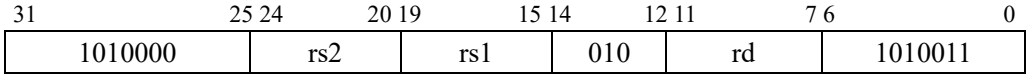
---

feq.s rd, rs1, rs2

x[rd] = f[rs1] == f[rs2]

单精度浮点相等*(Floating-point Equals, Single-Precision)*. R-type, RV32F and RV64F.

若寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相等，则在 x[*rd*]中写入 1，反之写 0。



31–25	24–20	19–15	14–12	11–7	6–0
1010000	rs2	rs1	010	rd	1010011

---

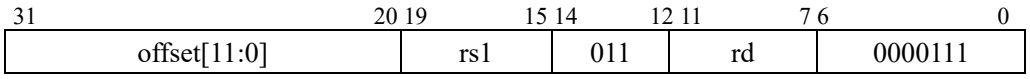
fld rd, offset(rs1)

f[rd] = M[x[rs1] + sext(offset)][63:0]

浮点加载双字*(Floating-point Load Doubleword)*. I-type, RV32D and RV64D.

从内存地址 x[*rs1*] + *sign-extend*(*offset*)中取双精度浮点数，并写入 f[*rd*]。

压缩形式：**c.fldsp** rd, offset; **c.fld** rd, offset(rs1)



31–20	19–15	14–12	11–7	6–0
offset[11:0]	rs1	011	rd	0000111

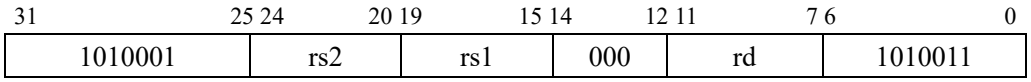
---

fle.d rd, rs1, rs2

x[rd] = f[rs1] ≤ f[rs2]

双精度浮点小于等于*(Floating-point Less Than or Equal, Double-Precision)*. R-type, RV32D and RV64D.

若寄存器 f[*rs1*]中的双精度浮点数小于等于 f[*rs2*]中的双精度浮点数，则在 x[*rd*]中写入 1，反之写 0。



31 25	24 20	19 15	14 12	11 7	6 0
1010001	rs2	rs1	000	rd	1010011

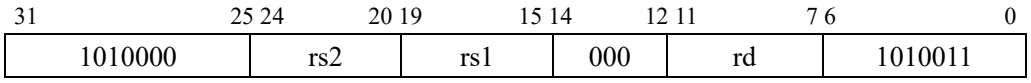
---

fle.s rd, rs1, rs2

x[rd] = f[rs1] ≤ f[rs2]

单精度浮点小于等于*(Floating-point Less Than or Equal, Single-Precision)*. R-type, RV32F and RV64F.

若寄存器 f[*rs1*]中的单精度浮点数小于等于 f[*rs2*]中的单精度浮点数，则在 x[*rd*]中写入 1，反之写 0。



31 25	24 20	19 15	14 12	11 7	6 0
1010000	rs2	rs1	000	rd	1010011

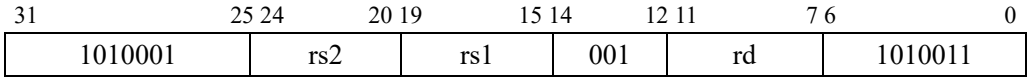
---

fle.d rd, rs1, rs2

x[rd] = f[rs1] < f[rs2]

双精度浮点小于 *(Floating-point Less Than, Double-Precision)*. R-type, RV32D and RV64D.

若寄存器 f[*rs1*]中的双精度浮点数小于 f[*rs2*]中的双精度浮点数，则在 x[*rd*]中写入 1，反之写 0。



31 25	24 20	19 15	14 12	11 7	6 0
1010001	rs2	rs1	001	rd	1010011

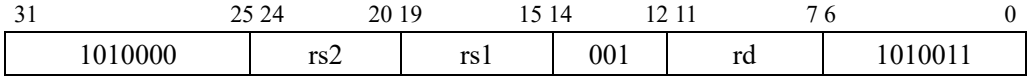
---

fle.s rd, rs1, rs2

x[rd] = f[rs1] < f[rs2]

单精度浮点小于 *(Floating-point Less Than, Single-Precision)*. R-type, RV32F and RV64F.

若寄存器 f[*rs1*]中的单精度浮点数小于 f[*rs2*]中的单精度浮点数，则在 x[*rd*]中写入 1，反之写 0。



31 25	24 20	19 15	14 12	11 7	6 0
1010000	rs2	rs1	001	rd	1010011

---

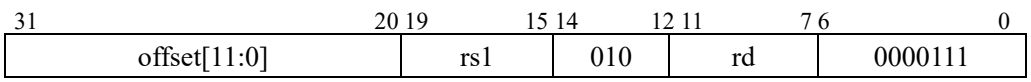
flw rd, offset(rs1)

f[rd] = M[x[rs1] + sext(offset)][31:0]

浮点加载字*(Floating-point Load Word)*. I-type, RV32F and RV64F.

从内存地址 x[*rs1*] + *sign-extend*(*offset*)中取单精度浮点数，并写入 f[*rd*]。

压缩形式：**c.flwsp** rd, offset; **c.flw** rd, offset(rs1)



31 20	19 15	14 12	11 7	6 0
offset[11:0]	rs1	010	rd	0000111

---

fmadd.d rd, rs1, rs2, rs3 f[rd] = f[rs1]×f[rs2]+f[rs3]

双精度浮点乘加*(Floating-point Fused Multiply-Add, Double-Precision)*. R4-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相乘，并将未舍入的积和寄存器 f[*rs3*]中的双精度浮点数相加，将舍入后的双精度浮点数写入 f[*rd*]。

31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	01	rs2	rs1	rm	rd	1000011

---

fmadd.s rd, rs1, rs2, rs3 f[rd] = f[rs1]×f[rs2]+f[rs3]

单精度浮点乘加*(Floating-point Fused Multiply-Add, Single-Precision)*. R4-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相乘，并将未舍入的积和寄存器 f[*rs3*]中的单精度浮点数相加，将舍入后的单精度浮点数写入 f[*rd*]。

31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	00	rs2	rs1	rm	rd	1000011

---

fmax.d rd, rs1, rs2 f[rd] = max(f[rs1], f[rs2])

双精度浮点最大值*(Floating-point Maximum, Double-Precision)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数中的较大值写入 f[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
0010101	rs2	rs1	001	rd	1010011

---

fmax.s rd, rs1, rs2 f[rd] = max(f[rs1], f[rs2])

单精度浮点最大值*(Floating-point Maximum, Single-Precision)*. R-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数中的较大值写入 f[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
0010100	rs2	rs1	001	rd	1010011

---

fmin.d rd, rs1, rs2 f[rd] = min(f[rs1], f[rs2])

双精度浮点最小值*(Floating-point Minimum, Double-Precision)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数中的较小值写入 f[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
0010101	rs2	rs1	000	rd	1010011

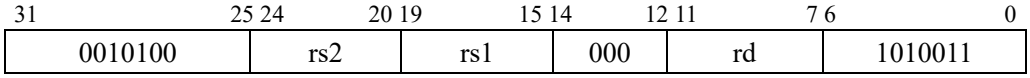
---

fmin.s rd, rs1, rs2

f[rd] = min(f[rs1], f[rs2])

单精度浮点最小值*(Floating-point Minimum, Single-Precision)*. R-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数中的较小值写入 f[*rd*]中。



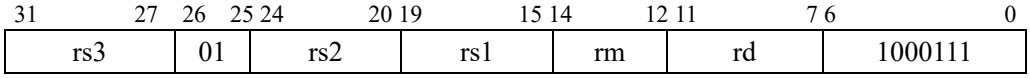
---

fmsub.d rd, rs1, rs2, rs3

f[rd] = f[rs1]×f[rs2]-f[rs3]

双精度浮点乘减*(Floating-point Fused Multiply-Subtract, Double-Precision)*. R4-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相乘，并将未舍入的积减去寄存器 f[*rs3*]中的双精度浮点数，将舍入后的双精度浮点数写入 f[*rd*]。



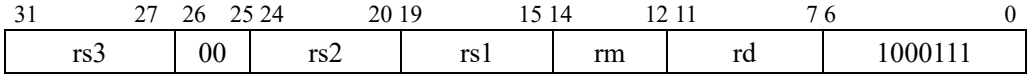
---

fmsub.s rd, rs1, rs2, rs3

f[rd] = f[rs1]×f[rs2]-f[rs3]

单精度浮点乘减*(Floating-point Fused Multiply-Subtarct, Single-Precision)*. R4-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相乘，并将未舍入的积减去寄存器 f[*rs3*]中的单精度浮点数，将舍入后的单精度浮点数写入 f[*rd*]。



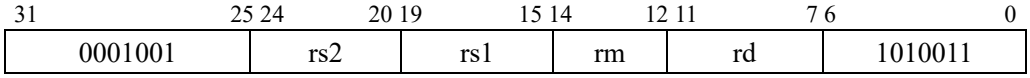
---

fmul.d rd, rs1, rs2

f[rd] = f[rs1] × f[rs2]

双精度浮点乘*(Floating-point Multiply, Double-Precision)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相乘，将舍入后的双精度结果写入 f[*rd*]中。



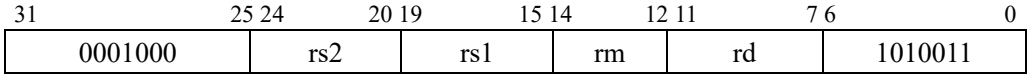
---

fmul.s rd, rs1, rs2

f[rd] = f[rs1] × f[rs2]

单精度浮点乘*(Floating-point Multiply, Single-Precision)*. R-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相乘，将舍入后的单精度结果写入 f[*rd*]中。



---

fmv.d rd, rs1

f[rd] = f[rs1]

双精度浮点移动 (Floating-point Move). *伪指令*(Pesudoinstruction), RV32D and RV64D.
把寄存器 f[*rs1*]中的双精度浮点数复制到 f[*rd*]中，等同于 **fsgnj.d** rd, rs1, rs1.

---

fmv.d.x rd, rs1, rs2

f[rd] = x[rs1][63:0]

双精度浮点移动(Floating-point Move Doubleword from Integer). R-type, RV64D.
把寄存器 x[*rs1*]中的双精度浮点数复制到 f[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
1111001	00000	rs1	000	rd	1010011

---

fmv.s rd, rs1

f[rd] = f[rs1]

单精度浮点移动 (Floating-point Move). *伪指令*(Pesudoinstruction), RV32F and RV64F.
把寄存器 f[*rs1*]中的单精度浮点数复制到 f[*rd*]中，等同于 **fsgnj.s** rd, rs1, rs1.

---

fmv.d.x rd, rs1, rs2

f[rd] = x[rs1][31:0]

单精度浮点移动(Floating-point Move Word from Integer). R-type, RV32F and RV64F.
把寄存器 x[*rs1*]中的单精度浮点数复制到 f[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
1111000	00000	rs1	000	rd	1010011

---

fmv.x.d rd, rs1, rs2

x[rd] = f[rs1][63:0]

双精度浮点移动(Floating-point Move Doubleword to Integer). R-type, RV64D.
把寄存器 f[*rs1*]中的双精度浮点数复制到 x[*rd*]中。

31 25	24 20	19 15	14 12	11 7	6 0
1110001	00000	rs1	000	rd	1010011

---

fmv.x.w rd, rs1, rs2

x[rd] = sext(f[rs1][31:0])

单精度浮点移动(Floating-point Move Word to Integer). R-type, RV32F and RV64F.
把寄存器 f[*rs1*]中的单精度浮点数复制到 x[*rd*]中，对于 RV64F，将结果进行符号扩展。

31 25	24 20	19 15	14 12	11 7	6 0
1110000	00000	rs1	000	rd	1010011

---

fneg.d rd, rs1 f[rd] = -f[rs1]

双精度浮点取反 *(Floating-point Negate)*. 伪指令(Pesudoinstruction), RV32D and RV64D.
把寄存器 f[*rs1*]中的双精度浮点数取反后写入 f[*rd*]中，等同于 **fsgnjn.d** rd, rs1, rs1.

---

fneg.s rd, rs1 f[rd] = -f[rs1]

单精度浮点取反 *(Floating-point Negate)*. 伪指令(Pesudoinstruction), RV32F and RV64F.
把寄存器 f[*rs1*]中的单精度浮点数取反后写入 f[*rd*]中，等同于 **fsgnjn.s** rd, rs1, rs1.

---

fnmadd.d rd, rs1, rs2, rs3 f[rd] = f[rs1]×f[rs2]+f[rs3]

双精度浮点乘取反加*(Floating-point Fused Negative Multiply-Add, Double-Precision)*. R4-type, RV32D and RV64D.
把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相乘，将结果取反，并将未舍入的积和寄存器 f[*rs3*]中的双精度浮点数相加，将舍入后的双精度浮点数写入 f[*rd*]。

31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	01	rs2	rs1	rm	rd	1001111

---

fnmadd.s rd, rs1, rs2, rs3 f[rd] = -f[rs1]_f[rs2]-f[rs3]

单精度浮点乘取反加*(Floating-point Fused Negative Multiply-Add, Single-Precision)*. R4-type, RV32F and RV64F.
把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相乘，将结果取反，并将未舍入的积和寄存器 f[*rs3*]中的单精度浮点数相加，将舍入后的单精度浮点数写入 f[*rd*]。

31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	00	rs2	rs1	rm	rd	1001111

---

fnmsub.d rd, rs1, rs2, rs3 f[rd] = -f[rs1]_f[rs2]+f[rs3]

双精度浮点乘取反减*(Floating-point Fused Negative Multiply-Subtract, Double-Precision)*. R4-type, RV32D and RV64D.
把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相乘，将结果取反，并将未舍入的积减去寄存器 f[*rs3*]中的双精度浮点数，将舍入后的双精度浮点数写入 f[*rd*]。

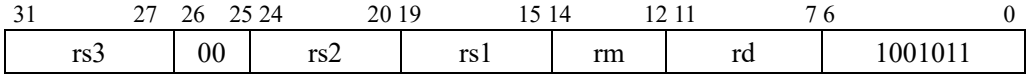
31 27	26 25	24 20	19 15	14 12	11 7	6 0
rs3	01	rs2	rs1	rm	rd	1001011

---

fnmsub.s rd, rs1, rs2, rs3 f[rd] = -f[rs1]×f[rs2]+f[rs3]

单精度浮点乘取反减(Floating-point Fused Negative Multiply-Subtract, Single-Precision). R4-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相乘，将结果取反，并将未舍入的积减去寄存器 f[*rs3*]中的单精度浮点数，将舍入后的单精度浮点数写入 f[*rd*]。



---

frcsr rd x[rd] = CSRs[fcsr]

浮点读控制状态寄存器 (Floating-point Read Control and Status Register). 伪指令(Pseudoinstruction), RV32F and RV64F.

把浮点控制状态寄存器的值写入 x[*rd*]，等同于 **csrrs** rd, fcsr, x0.

---

frflags rd x[rd] = CSRs[fflags]

浮点读异常标志 (Floating-point Read Exception Flags). 伪指令(Pseudoinstruction), RV32F and RV64F.

把浮点异常标志的值写入 x[*rd*]，等同于 **csrrs** rd, fflags, x0.

---

frrm rd x[rd] = CSRs[frm]

浮点读舍入模式 (Floating-point Read Rounding Mode). 伪指令(Pseudoinstruction), RV32F and RV64F.

把浮点舍入模式的值写入 x[*rd*]，等同于 **csrrs** rd, frm, x0.

---

fscsr rd, rs1 t = CSRs[fcsr]; CSRs[fcsr] = x[rs1]; x[rd] = t

浮点换出控制状态寄存器 (Floating-point Swap Control and Status Register). 伪指令(Pseudoinstruction), RV32F and RV64F.

把寄存器 x[*rs1*]的值写入浮点控制状态寄存器，并将浮点控制状态寄存器的原值写入 x[*rd*]，等同于 **csrrw** rd, fcsr, rs1。*rd* 默认为 x0。

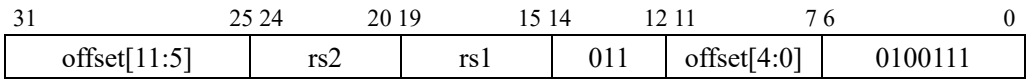
---

fsd rs2, offset(rs1) M[x[rs1] + sext(offset)] = f[rs2][63:0]

双精度浮点存储(Floating-point Store Doubleword). S-type, RV32D and RV64D.

将寄存器 f[*rs2*]中的双精度浮点数存入内存地址 x[*rs1*] + *sign-extend*(*offset*)中。

压缩形式：**c.fsdsp** rs2, offset; **c.fsd** rs2, offset(rs1)



---

fsflags rd, rs1 t = CSRs[fflags]; CSRs[fflags] = x[rs1]; x[rd] = t

浮点换出异常标志 *(Floating-point Swap Exception Flags)*. 伪指令(Pseudoinstruction), RV32F and RV64F.

把寄存器 x[*rs1*]的值写入浮点异常标志寄存器，并将浮点异常标志寄存器的原值写入 x[*rd*]，等同于 **csrrw** rd, fflags, rs1。*rd* 默认为 x0。

---

fsgnj.d rd, rs1, rs2 f[rd] = {f[rs2][63], f[rs1][62:0]}

双精度浮点符号注入*(Floating-point Sign Inject, Double-Precision)*. R-type, RV32D and RV64D.

用 f[*rs1*]指数和有效数以及 f[*rs2*]的符号的符号位，来构造一个新的双精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010001	rs2	rs1	000	rd	1010011

---

fsgnj.s rd, rs1, rs2 f[rd] = {f[rs2][31], f[rs1][30:0]}

单精度浮点符号注入*(Floating-point Sign Inject, Single-Precision)*. R-type, RV32F and RV64F.

用 f[*rs1*]指数和有效数以及 f[*rs2*]的符号的符号位，来构造一个新的单精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010000	rs2	rs1	000	rd	1010011

---

fsgnjn.d rd, rs1, rs2 f[rd] = {~f[rs2][63], f[rs1][62:0]}

双精度浮点符号取反注入*(Floating-point Sign Inject-Negate, Double-Precision)*. R-type, RV32D and RV64D.

用 f[*rs1*]指数和有效数以及 f[*rs2*]的符号的符号位取反，来构造一个新的双精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010001	rs2	rs1	001	rd	1010011

---

fsgnjn.s rd, rs1, rs2 f[rd] = {~f[rs2][31], f[rs1][30:0]}

单精度浮点符号取反注入*(Floating-point Sign Inject-Negate, Single-Precision)*. R-type, RV32F and RV64F.

用 f[*rs1*]指数和有效数以及 f[*rs2*]的符号的符号位取反，来构造一个新的单精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010000	rs2	rs1	001	rd	1010011

---

fsgnjx.d rd, rs1, rs2 f[rd] = {f[rs1][63] ^ f[rs2][63], f[rs1][62:0]}

双精度浮点符号异或注入*(Floating-point Sign Inject-XOR, Double-Precision)*. R-type, RV32D and RV64D.

用 f[*rs1*]指数和有效数以及 f[*rs1*]和 f[*rs2*]的符号的符号位异或，来构造一个新的双精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010001	rs2	rs1	010	rd	1010011

---

fsgnjx.s rd, rs1, rs2 f[rd] = {f[rs1][31] ^ f[rs2][31], f[rs1][30:0]}

单精度浮点符号异或注入*(Floating-point Sign Inject-XOR, Single-Precision)*. R-type, RV32F and RV64F.

用 f[*rs1*]指数和有效数以及 f[*rs1*]和 f[*rs2*]的符号的符号位异或，来构造一个新的单精度浮点数，并将其写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0010000	rs2	rs1	010	rd	1010011

---

fsqrt.d rd, rs1, rs2 f[rd] = $\sqrt{f[rs1]}$

双精度浮点平方根*(Floating-point Square Root, Double-Precision)*. R-type, RV32D and RV64D.

将 f[*rs1*]中的双精度浮点数的平方根舍入和写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0101101	00000	rs1	rm	rd	1010011

---

fsqrt.s rd, rs1, rs2 f[rd] = $\sqrt{f[rs1]}$

单精度浮点平方根*(Floating-point Square Root, Single-Precision)*. R-type, RV32F and RV64F.

将 f[*rs1*]中的单精度浮点数的平方根舍入和写入 f[*rd*]。

31 25	24 20	19 15	14 12	11 7	6 0
0101100	00000	rs1	rm	rd	1010011

---

fsrm rd, rs1 t = CSRs[frm]; CSRs[frm] = x[rs1]; x[rd] = t

浮点换出舍入模式 *(Floating-point Swap Rounding Mode)*. 伪指令(Pseudoinstruction), RV32F and RV64F.

把寄存器 x[*rs1*]的值写入浮点舍入模式寄存器，并将浮点舍入模式寄存器的原值写入 x[*rd*]，等同于 **csrrw** rd, frm, rs1。*rd* 默认为 x0。

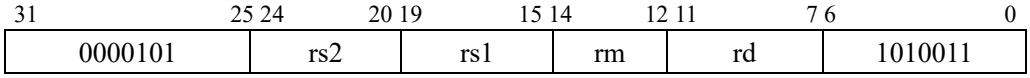
---

fsub.d rd, rs1, rs2

f[rd] = f[rs1] - f[rs2]

双精度浮点减*(Floating-point Subtract, Double-Precision)*. R-type, RV32D and RV64D.

把寄存器 f[*rs1*]和 f[*rs2*]中的双精度浮点数相减，并将舍入后的差写入 f[*rd*]。



31 25	24 20	19 15	14 12	11 7	6 0
0000101	rs2	rs1	rm	rd	1010011

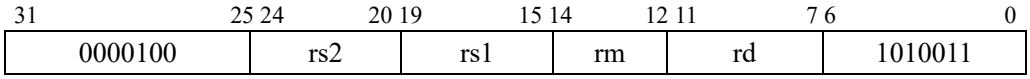
---

fsub.s rd, rs1, rs2

f[rd] = f[rs1] - f[rs2]

单精度浮点减*(Floating-point Subtract, Single-Precision)*. R-type, RV32F and RV64F.

把寄存器 f[*rs1*]和 f[*rs2*]中的单精度浮点数相减，并将舍入后的差写入 f[*rd*]。



31 25	24 20	19 15	14 12	11 7	6 0
0000100	rs2	rs1	rm	rd	1010011

---

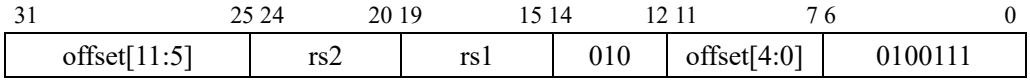
fsw rs2, offset(rs1)

M[x[rs1] + sext(offset)] = f[rs2][31:0]

单精度浮点存储*(Floating-point Store Word)*. S-type, RV32F and RV64F.

将寄存器 f[*rs2*]中的单精度浮点数存入内存地址 x[*rs1*] + *sign-extend*(*offset*)中。

压缩形式：**c.fswsp** rs2, offset; **c.fsw** rs2, offset(rs1)



31 25	24 20	19 15	14 12	11 7	6 0
offset[11:5]	rs2	rs1	010	offset[4:0]	0100111

---

j offset

pc += sext(offset)

跳转 *(Jump)*. 伪指令(Pseudoinstruction), RV32I and RV64I.

把 *pc* 设置为当前值加上符号位扩展的 *offset*，等同于 **jal** x0, offset.

---

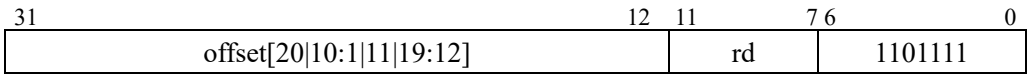
jal rd, offset

x[rd] = pc+4; pc += sext(offset)

跳转并链接 *(Jump and Link)*. J-type, RV32I and RV64I.

把下一条指令的地址(*pc+4*)，然后把 *pc* 设置为当前值加上符号位扩展的 *offset*。*rd* 默认为 x1。

压缩形式：**c.j** offset; **c.jal** offset



31 12	11 7	6 0
offset[20|10:1|11|19:12]	rd	1101111

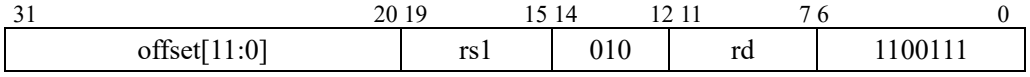
---

jalr rd, offset(rs1)　　t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t

跳转并寄存器链接 *(Jump and Link Register)*. I-type, RV32I and RV64I.

把 *pc* 设置为 x[*rs1*]+*sign-extend*(*offset*)，把计算出的地址的最低有效位设为 0，并将原 *pc+4* 的值写入 f[*rd*]。*rd* 默认为 x1。

压缩形式：**c.jr** rs1; **c.jalr** rs1



---

jr rs1　　pc = x[rs1]

寄存器跳转 *(Jump Register)*. 伪指令(Pseudoinstruction), RV32I and RV64I.

把 *pc* 设置为 x[*rs1*]，等同于 **jalr** x0, 0(rs1)。

---

la rd, symbol　　x[rd] = &symbol

地址加载 *(Load Address)*. 伪指令(Pseudoinstruction), RV32I and RV64I.

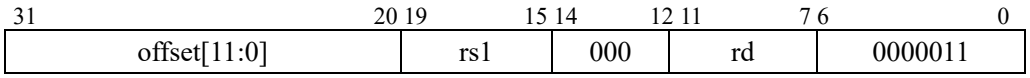
将 *symbol* 的地址加载到 x[*rd*]中。当编译位置无关的代码时，它会被扩展为对全局偏移量表(Global Offset Table)的加载。对于 RV32I，等同于执行 **auipc** rd, offsetHi，然后是 **lw** rd, offsetLo(rd)；对于 RV64I，则等同于 **auipc** rd，offsetHi 和 **ld** rd, offsetLo(rd)。如果 offset 过大，开始的算加载地址的指令会变成两条，先是 **auipc** rd, offsetHi 然后是 **addi** rd, rd, offsetLo。

---

lb rd, offset(rs1)　　x[rd] = sext(M[x[rs1] + sext(offset)][7:0])

字节加载 *(Load Byte)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取一个字节，经符号位扩展后写入 x[*rd*]。

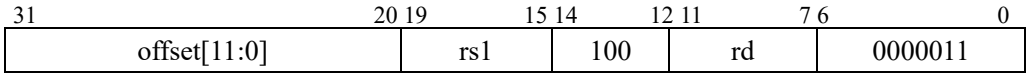


---

lbu rd, offset(rs1)　　x[rd] = M[x[rs1] + sext(offset)][7:0]

无符号字节加载 *(Load Byte, Unsigned)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取一个字节，经零扩展后写入 x[*rd*]。



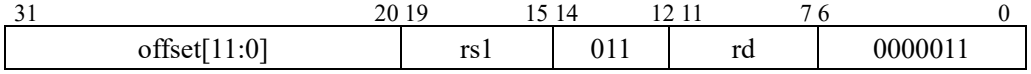
---

ld rd, offset(rs1)　　x[rd] = M[x[rs1] + sext(offset)][63:0]

双字加载 *(Load Doubleword)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取八个字节，写入 x[*rd*]。

压缩形式：**c.ldsp** rd, offset; **c.ld** rd, offset(rs1)

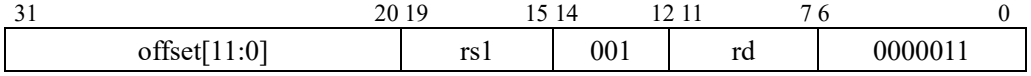


---

lh rd, offset(rs1)　　x[rd] = sext(M[x[rs1] + sext(offset)][15:0])

半字加载 *(Load Halfword)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取两个字节，经符号位扩展后写入 x[*rd*]。

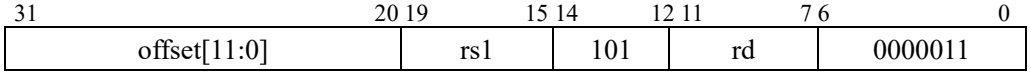


---

lhu rd, offset(rs1)　　x[rd] = M[x[rs1] + sext(offset)][15:0]

无符号半字加载 *(Load Halfword, Unsigned)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取两个字节，经零扩展后写入 x[*rd*]。



---

li rd, immediate　　x[rd] = immediate

立即数加载 *(Load Immediate)*. 伪指令(Pseudoinstruction), RV32I and RV64I.

使用尽可能少的指令将常量加载到 x[*rd*]中。在 RV32I 中，它等同于执行 **lui** 和/或 **addi**；对于 RV64I，会扩展为这种指令序列 **lui, addi, slli, addi, slli, addi ,slli, addi**。

---

lla rd, symbol　　x[rd] = &symbol

本地地址加载 *(Load Local Address)*. 伪指令(Pseudoinstruction), RV32I and RV64I.

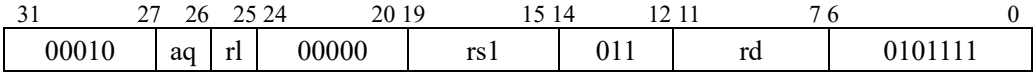
将 *symbol* 的地址加载到 x[*rd*]中。等同于执行 **auipc** rd, offsetHi，然后是 **addi** rd, rd, offsetLo。

---

lr.d rd, (rs1)　　x[rd] = LoadReserved64(M[x[rs1]])

加载保留双字*(Load-Reserved Doubleword)*. R-type, RV64A.

从内存中地址为 x[*rs1*]中加载八个字节，写入 x[*rd*]，并对这个内存双字注册保留。

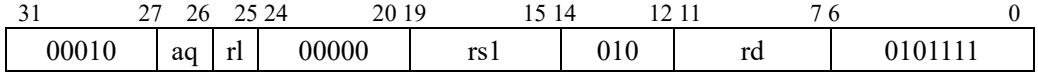


---

lr.w rd, (rs1)　　x[rd] = LoadReserved32(M[x[rs1]])

加载保留字*(Load-Reserved Word)*. R-type, RV32A and RV64A.

从内存中地址为 x[*rs1*]中加载四个字节，符号位扩展后写入 x[*rd*]，并对这个内存字注册保留。



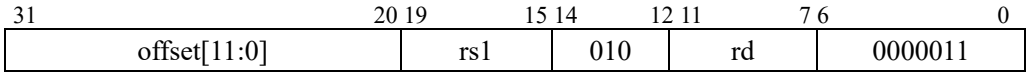
---

lw rd, offset(rs1)　　x[rd] = sext(M[x[rs1] + sext(offset)][31:0])

字加载 *(Load Word)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取四个字节，写入 x[*rd*]。对于 RV64I，结果要进行符号位扩展。

压缩形式：**c.lwsp** rd, offset; **c.lw** rd, offset(rs1)

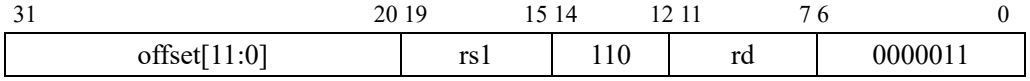


---

lwu rd, offset(rs1)　　x[rd] = M[x[rs1] + sext(offset)][31:0]

无符号字加载 *(Load Word, Unsigned)*. I-type, RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取四个字节，零扩展后写入 x[*rd*]。



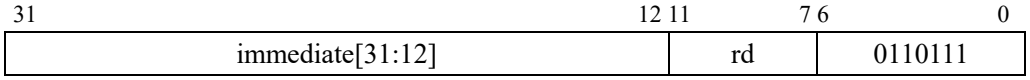
---

lui rd, immediate　　x[rd] = sext(immediate[31:12] << 12)

高位立即数加载 *(Load Upper Immediate)*. U-type, RV32I and RV64I.

将符号位扩展的 20 位立即数 *immediate* 左移 12 位，并将低 12 位置零，写入 x[*rd*]中。

压缩形式：**c.lui** rd, imm

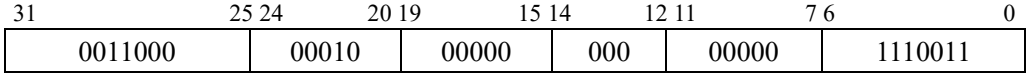


---

mret　　ExceptionReturn(Machine)

机器模式异常返回*(Machine-mode Exception Return)*. R-type, RV32I and RV64I 特权架构

从机器模式异常处理程序返回。将 *pc* 设置为 CSRs[mepc]，将特权级设置成 CSRs[mstatus].MPP, CSRs[mstatus].MIE 置成 CSRs[mstatus].MPIE，并且将 CSRs[mstatus].MPIE 为 1;并且，如果支持用户模式，则将 CSR [mstatus].MPP 设置为 0。



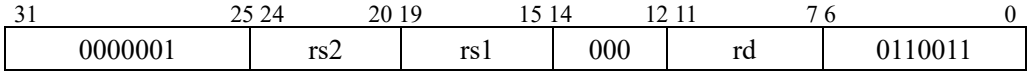
---

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

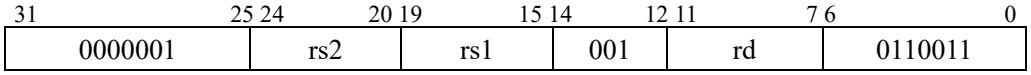
---

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

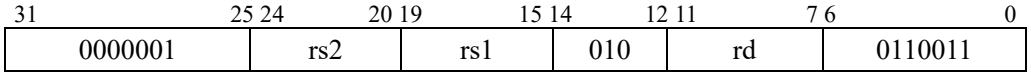
---

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

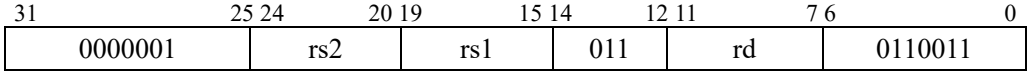
---

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

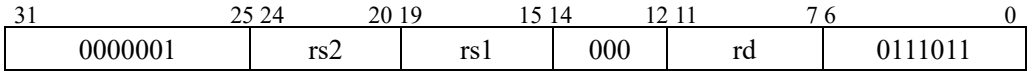
---

mulw rd, rs1, rs2

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

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。

---

negw rd, rs2

$x[rd] = sext((-x[rs2])[31:0])$

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。

---

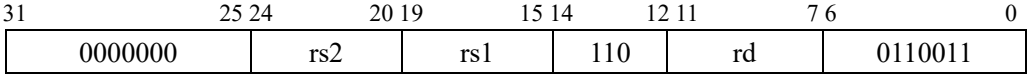
or rd, rs1, rs2

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

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



---

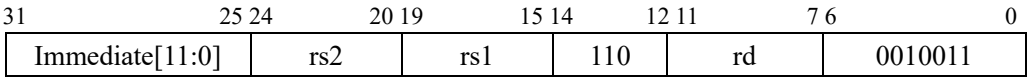
ori rd, rs1, immediate

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

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



---

rdcycle rd

$x[rd] = CSRS[cycle]$

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 rd

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 rd

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 rd

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 rd

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 rd

x[rd] = CSRs[timeh]

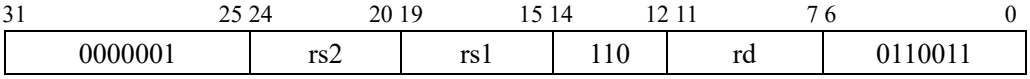
读取时间高位*(Read Time High)*. 伪指令(Pseudoinstruction), RV32I only.
把当前时间右移 32 位后写入 x[*rd*]，时间频率与平台相关。实际被扩展为 **csrrs** rd, timeh, x0。

---

rem rd, rs1, rs2

$x[rd] = x[rs1] \%_s x[rs2]$

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*]。

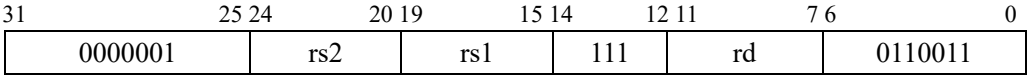


---

remu rd, rs1, rs2

$x[rd] = x[rs1] \%_u x[rs2]$

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*]。



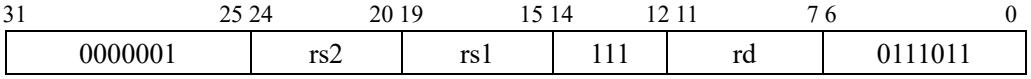
---

remuw rd, rs1, rs2

$x[rd] = sext(x[rs1][31:0] \%_u x[rs2][31:0])$

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

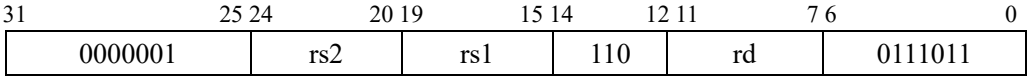
---

remw rd, rs1, rs2

$x[rd] = sext(x[rs1][31:0] \%_s x[rs2][31:0])$

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)。

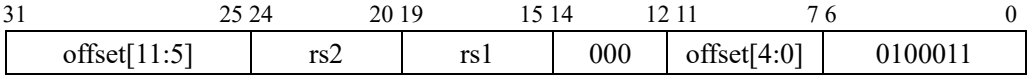
---

sb rs2, offset(rs1)

$M[x[rs1] + sext(offset) = x[rs2][7:0]$

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

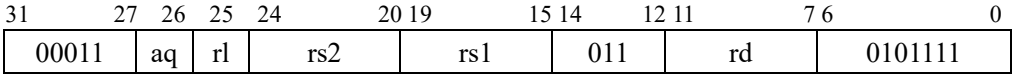
---

sc.d rd, rs2, (rs1)

$x[rd] = StoreConditonal64(M[x[rs1], x[rs2])$

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

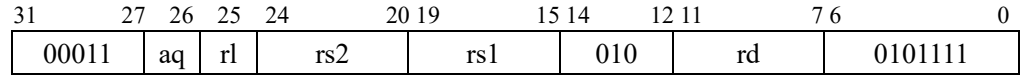
---

sc.w rd, rs2, (rs1)

$x[rd] = StoreConditonal32(M[x[rs1], x[rs2])$

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。



---

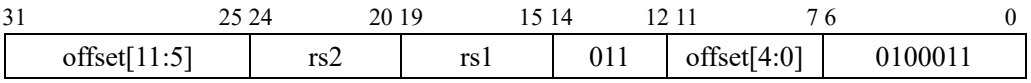
sd rs2, offset(rs1)

$M[x[rs1] + sext(offset) = x[rs2][63:0]$

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*]+*sign-extend(offset)*。

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



---

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

$x[rd] = sext(x[rs1][31:0])$

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。

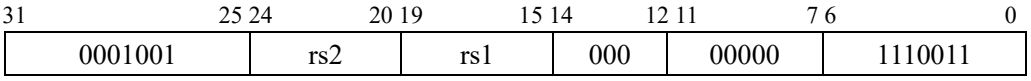
---

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。



---

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.

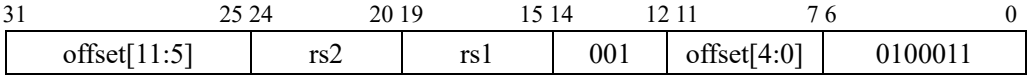
如果 x[*rs2*]大于 0，向 x[*rd*]写入 1，否则写入 0。实际被扩展为 **slt** rd, x0, rs2。

---

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)*。



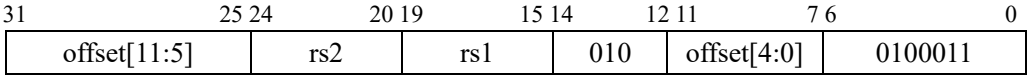
---

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)

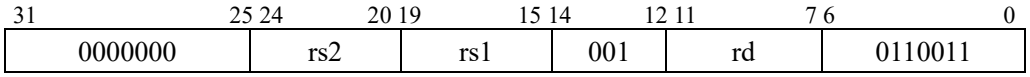


---

sll rd, rs1, rs2 x[rd] = x[rs1] ≪ 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。



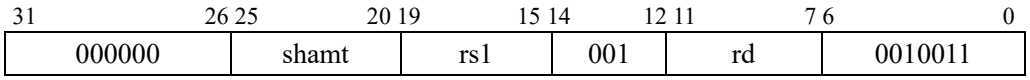
---

slli rd, rs1, shamt x[rd] = x[rs1] ≪ 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

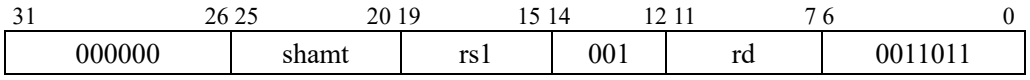


---

sllw rd, rs1, shamt x[rd] = sext((x[rs1] ≪ 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。

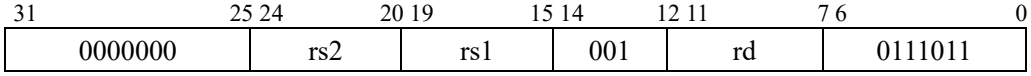


---

sllw rd, rs1, rs2 $x[rd] = sext((x[rs1] \ll x[rs2][4:0])[31:0])$

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。

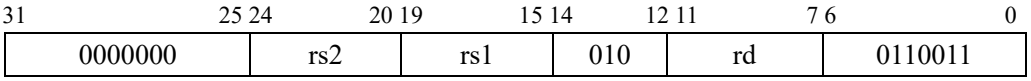


---

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。

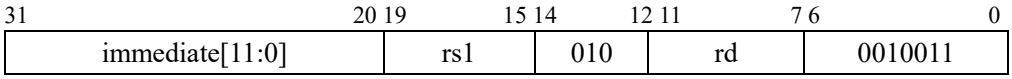


---

slti rd, rs1, immediate $x[rd] = (x[rs1] <_s sext(immediate))$

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。

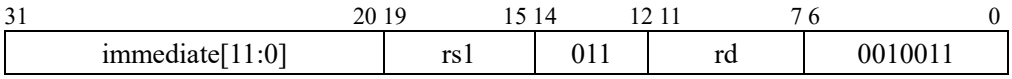


---

sltiu rd, rs1, immediate $x[rd] = (x[rs1] <_u sext(immediate))$

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。

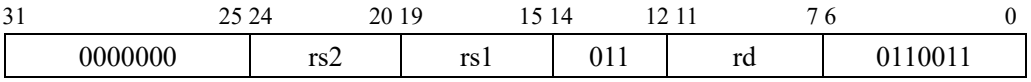


---

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。



---

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 0
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 0
010000	shamt	rs1	101	rd	0011011

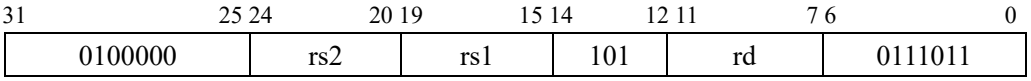
---

sraw rd, rs1, rs2

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

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

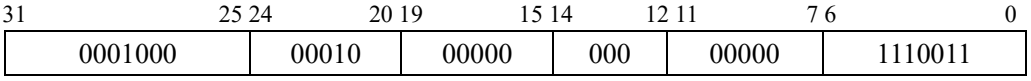
---

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

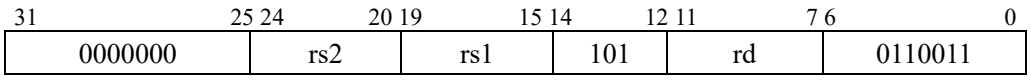
---

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

---

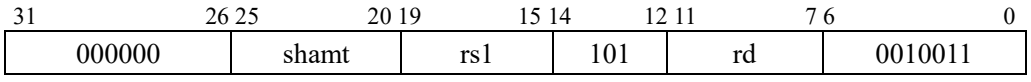
srli rd, rs1, shamt

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

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

把寄存器 x[*rs1*]右移 *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.srli** rd, shamt



31 26	25 20	19 15	14 12	11 7	6 0
000000	shamt	rs1	101	rd	0010011

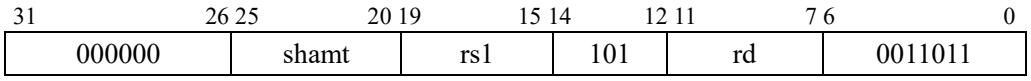
---

srliw rd, rs1, shamt

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

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 *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
000000	shamt	rs1	101	rd	0011011

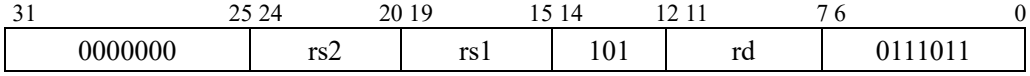
---

srlw rd, rs1, rs2

$x[rd] = sext(x[rs1][31:0] \gg_u x[rs2][4:0])$

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

---

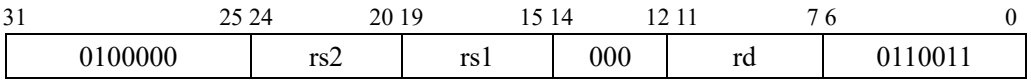
sub rd, rs1, rs2

$x[rd] = x[rs1] - x[rs2]$

减(Substract). 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

---

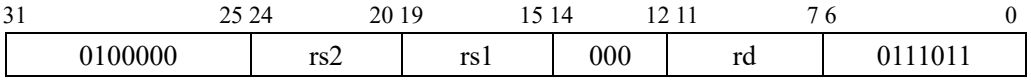
subw rd, rs1, rs2

$x[rd] = sext((x[rs1] - x[rs2])[31:0])$

减去字(Substract 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 = &symbol; clobber x[6]

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

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

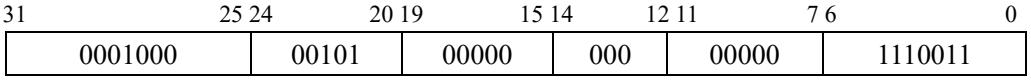
---

wfi

while (noInterruptPending) 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

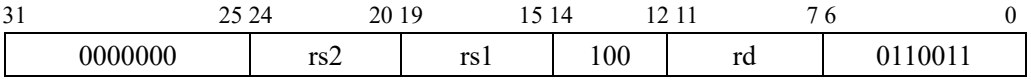
---

xor rd, rs1, rs2 x[rd] = x[rs1] ^ 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



---

xori rd, rs1, immediate x[rd] = x[rs1] ^ 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