

## 附录 A RISC-V 指令列表

**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。



简约是一切真正优雅的要义。——Coco Chanel, 1923

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

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

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.addiw** rd, imm; **c.addi** rd, im; **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

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

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)

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	000	rd	0101111

---

## amoadd.w rd, rs2, (rs1)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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)

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  $t$ 。

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	aq	rl	rs2	rs1	011	rd	0101111						

---

## amoxor.w rd, rs2, (rs1)

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  $t$ 。

31	27	26	25	24	20	19	15	14	12	11	7	6	0
00100	aq	rl	rs2	rs1	010	rd	0101111						

---

## and rd, rs1, 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

31	25	24	20	19	15	14	12	11	7	6	0
0000000	rs2	rs1	111	rd	0110011						

---

## andi rd, rs1, 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.andiw** rd, imm

31	20	19	15	14	12	11	7	6	0
immediate[11:0]	rs1	111	rd	0010011					

---

## auipc rd, immediate

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  $x[rd]$ 。

31	12	11	7	6	0
immediate[31:12]	rd	0010011			

---

**beq** rs1, rs2, 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

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

可视为 **beq** rs1, x0, offset.

---

**bge** rs1, rs2, 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	

---

**bge** rs1, rs2, 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

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

可视为 **bge** rs1, x0, offset.

---

**bgt** rs1, rs2, offset

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

可视为 **blt** rs2, rs1, offset.

---

**bgtu** rs1, rs2, offset

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

可视为 **bltu** rs2, rs1, offset.

---

**bgtz**    *rs1, offset*  
大于零时分支 (*Branch if Greater Than Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
可视为 **blt** *x0, rs2, offset*.

---

**ble**    *rs1, rs2, offset*  
小于等于时分支 (*Branch if Less Than or Equal*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
可视为 **bge** *rs2, rs1, offset*.

---

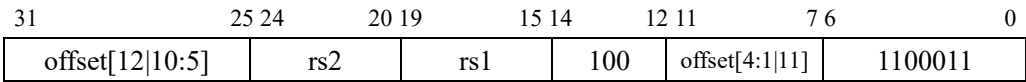
**bleu**    *rs1, rs2, offset*  
小于等于时分支 (*Branch if Less Than or Equal, Unsigned*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
可视为 **bgeu** *rs2, rs1, offset*.

---

**blez**    *rs2, offset*  
小于等于零时分支 (*Branch if Less Than or Equal to Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
可视为 **bge** *x0, rs2, offset*.

---

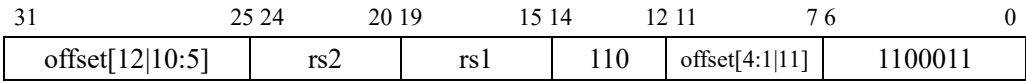
**blt**    *rs1, rs2, offset*  
小于时分支 (*Branch if Less Than*). B-type, RV32I and RV64I.  
若寄存器 *x[rs1]* 的值小于寄存器 *x[rs2]* 的值（均视为二进制补码），把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。



**bltz**    *rs2, offset*  
小于零时分支 (*Branch if Less Than Zero*). 伪指令(Pseudoinstruction), RV32I and RV64I.  
可视为 **blt** *rs1, x0, offset*.

---

**bltu**    *rs1, rs2, 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*。





**bne** rs1, rs2, 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

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

可视为 **bne** rs1, x0, offset.

**c.add** rd, 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

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	rs2	01		

**c.addi16sp** 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[4j6j8:7j5]	01		

**c.addi4spn** rd', 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

加立即数字 (*Add Word Immediate*). RV64IC.

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

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

---

**c.and** rd', rs2'

与 (*AND*). RV32IC and RV64IC.

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

15	10	9	7	6	5	4	2	1	0
100011			rd'		11	rs2'		01	

---

**c.addw** rd', rs2'

加字 (*Add Word*). RV64IC.

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

15	10	9	7	6	5	4	2	1	0
100111			rd'		01	rs2'		01	

---

**c.andi** rd', imm

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

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

15	13	12	11	10	9	7	6	2	1	0
100	imm[5]			10	rd'		imm[4:0]		01	

---

**c.beqz** rs1', 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'.

15	13	12	10	9	7	6	2	1	0
110	offset[8:4:3]			rs1'		offset[7:6:2:1:5]		01	

---

**c.bnez** rs1', 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'.

15	13	12	10	9	7	6	2	1	0
111	offset[8:4:3]			rs1'		offset[7:6:2:1:5]		01	

---

## c.ebreak

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

扩展形式为 **ebreak**.

15	13	12	11	7	6	2	1	0
100	1	00000	00000	10				

---

## c.fld rd', uimm(rs1')

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

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

15	13	12	10	9	7	6	5	4	2	1	0
001	uimm[5:3]	rs1'	uimm[7:6]	rd'	00						

---

## c.fldsp rd, uimm(x2)

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).

15	13	12	11	7	6	2	1	0
001	uimm[5]	rd	uimm[4:3 8:6]	10				

---

## c.flw rd', uimm(rs1')

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

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

15	13	12	10	9	7	6	5	4	2	1	0
011	uimm[5:3]	rs1'	uimm[2 6]	rd'	00						

---

## c.flwsp rd, uimm(x2)

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).

15	13	12	11	7	6	2	1	0
011	uimm[5]	rd	uimm[4:3 8:6]	10				

---

## c.fsd rs2', uimm(rs1')

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

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

15	13	12	10	9	7	6	5	4	2	1	0
101	uimm[5:3]	rs1'	uimm[7:6]	rs2'	00						

---

**c.fsdsp**     $rs2, uimm(x2)$

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**  $rs2, uimm(x2)$ .

15	13 12	7 6	2 1	0
101	$uimm[5:3 8:6]$	$rs2$	10	

---

**c.fsw**     $rs2', uimm(rs1')$

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

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

15	13 12	10 9	7 6	5 4	2 1	0
111	$uimm[5:3]$	$rs1'$	$uimm[2 6]$	$rs2'$	00	

---

**c.fswsp**     $rs2, uimm(x2)$

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

扩展形式为 **fsw**  $rs2, uimm(x2)$ .

15	13 12	7 6	2 1	0
111	$uimm[5:2 7:6]$	$rs2$	10	

---

**c.j**    offset

跳转 (*Jump*). RV32IC and RV64IC.

扩展形式为 **jal**  $x0, offset$ .

15	13 12	2 1	0
101	$offset[11 4 9:8 10 6 7 3:1 5]$	01	

---

**c.jal**    offset

链接跳转 (*Jump and Link*). RV32IC.

扩展形式为 **jal**  $x1, offset$ .

15	13 12	2 1	0
001	$offset[11 4 9:8 10 6 7 3:1 5]$	01	

---

**c.jalr**     $rs1$

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。

15	13	12	11	7 6	2 1	0
100	1	$rs1$	00000	10		

---

### c.jr 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。

15	13	12	11	7	6	2	1	0
100	0	rs1	00000	10				

---

### c.ld rd', uimm(rs1')

双字加载 (*Load Doubleword*). RV64IC.

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

15	13	12	10	9	7	6	5	4	2	1	0
011	uimm[5:3]	rs1'	uimm[7:6]	rd'	00						

---

### c.ldsp rd, uimm(x2)

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。

15	13	12	11	7	6	2	1	0
011	uimm[5]	rd	uimm[4:3 8:6]	10				

---

### c.li rd, imm

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

扩展形式为 **addi** rd, x0, imm.

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

---

### c.lui rd, imm

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。

15	13	12	11	7	6	2	1	0
011	imm[17]	rd	imm[16:12]	01				

---

### c.lw rd', uimm(rs1')

字加载 (*Load Word*). RV32IC and RV64IC.

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

15	13	12	10	9	7	6	5	4	2	1	0
010	uimm[5:3]	rs1'	uimm[2 6]	rd'	00						

---

**c.lwsp** rd, uimm(x2)

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。

15	13	12	11	7	6	2	1	0
010	uimm[5]	rd	uimm[4:2 7:6]	10				

---

**c.mv** rd, rs2

移动 (*Move*). RV32IC and RV64IC.  
扩展形式为 **add** rd, x0, rs2. rs2=x0 时非法。

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

---

**c.or** rd', rs2'

或 (*OR*). RV32IC and RV64IC.  
扩展形式为 **or** rd, rd, rs2, 其中 rd=8+rd', rs2=8+rs2'.

15	10	9	7	6	5	4	2	1	0
100011	rd'	10	rs2'	01					

---

**c.sd** rs2', uimm(rs1')

双字存储 (*Store Doubleword*). RV64IC.  
扩展形式为 **sd** rs2, uimm(rs1), 其中 rs2=8+rs2', rs1=8+rs1'.

15	13	12	10	9	7	6	5	4	2	1	0
111	uimm[5:3]	rs1'	uimm[7:6]	rs2'	00						

---

**c.sdsp** rs2, uimm(x2)

栈指针相关双字存储 (*Store Doubleword, Stack-Pointer Relative*). RV64IC.  
扩展形式为 **sd** rs2, uimm(x2).

15	13	12	7	6	2	1	0
111	uimm[5:3 8:6]	rs2	10				

---

**c.slli** 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.

15	13	12	11	7	6	2	1	0
000	uimm[5]	rd	uimm[4:0]	10				

---

### c.srai rd', 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  $rd=8+rd'$ .

15	13	12	11	10 9	7 6	2 1	0
100	uimm[5]	01	rd'	uimm[4:0]	01		

---

### c.srli rd', 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  $rd=8+rd'$ .

15	13	12	11	10 9	7 6	2 1	0
100	uimm[5]	00	rd'	uimm[4:0]	01		

---

### c.sub rd', rs2'

减 (*Subtract*). RV32IC and RV64IC.

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

15	10 9	7 6	5 4	2 1	0
100011	rd'	00	rs2'	01	

---

### c.subw rd', rs2'

减字 (*Subtract Word*). RV64IC.

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

15	10 9	7 6	5 4	2 1	0
100111	rd'	00	rs2'	01	

---

### C.SW rs2', uimm(rs1')

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

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

15	13 12	10 9	7 6	5 4	2 1	0
110	uimm[5:3]	rs1'	uimm[2:6]	rs2'	00	

---

### c.swsp rs2, uimm(x2)

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).

15	13 12	7 6	2 1	0
110	uimm[5:2 7:6]	rs2	10	

---

### C.XOR rd', rs2'

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

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

15	10 9	7 6	5 4	2 1	0
100011	rd'	01	rs2'	01	

### call rd, 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, offsetHi, 再加上一条 **jalr** rd, offsetLo(rd). 若省略了  $rd$ , 默认为  $x1$ .

### CSRR rd, 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

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  $csr$  的对应位清零, 等同于 **csrrc** x0, csr, rs1.

### CSRCi csr, zimm[4:0]

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  $csr$  的对应位清零, 等同于 **csrrci** x0, csr, zimm.

### CSRRC rd, csr, rs1

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  $x[rd]$ 。

31	20 19	15 14	12 11	7 6	0
csr	rs1	011	rd	1110011	



**csrrci** rd, csr, zimm[4:0]

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

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]

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]

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]

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

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

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

---

### csrci csr, zimm[4:0]

立即数置位控制状态寄存器 (*Control and Status Register Set Immediate*). 伪指令(Pseudoinstruction), 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* 的对应位清零, 等同于 **csrrsi**  $x0, csr, zimm$ .

---

### CSRW csr, rs1

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

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

---

### CSRwi csr, zimm[4:0]

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

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  $x[rd]$ 。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	100	rd	0110011	

---

### divu rd, rs1, 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  $x[rd]$ 。

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	101	rd	0110011	

---

## divuw rd, rs1, rs2

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

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

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
0000000000001	00000	000	00000	1110011	

---

## ecall

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
0000000000000	00000	000	00000	1110011	

---

## fabs.d rd, rs1

浮点数绝对值 (*Floating-point Absolute Value*). 伪指令 (Pseudoinstruction), 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

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

---