目录

第一部分：RV32I 指令集 2
一、RV32I 指令列表 2
二、RV32I 指令格式 3
三、寄存器定义 3
四、RV32I 指令功能介绍 3
1. Load/Store 指令 3
2. 位运算指令 4
3. 算术指令 5
4. 逻辑指令 5
5. 比较指令 5
6. 分支指令 6
7. 跳转指令 6
8. 同步指令 7
9. 环境调用和断点 7
10. HINT 指令 7
第二部分：扩展指令 8
一、 M 扩展 8
二、 C 扩展 8

第一部分：RV32I 指令集

一、RV32I 指令列表

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

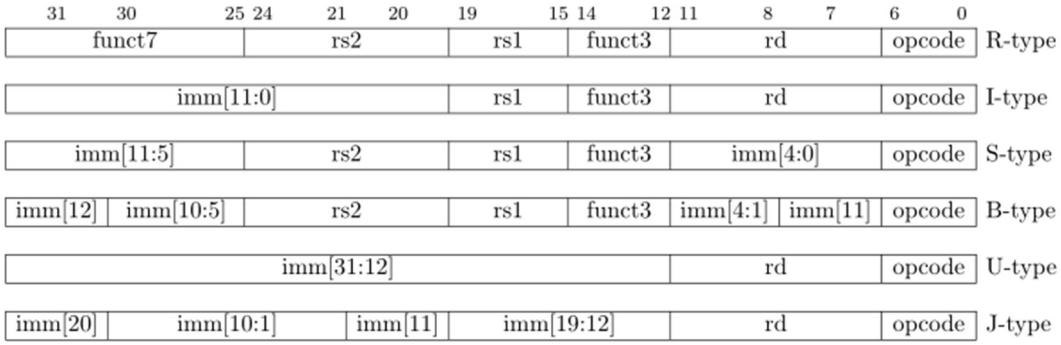
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

表 1 RV32I 指令

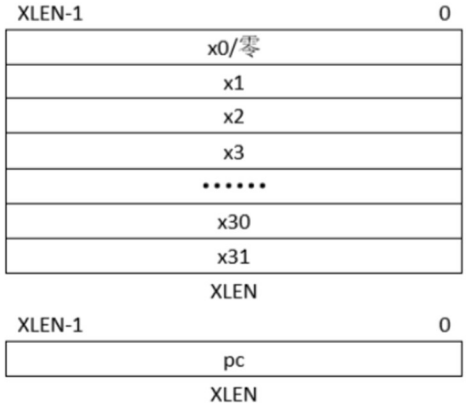
二、RV32I 指令格式

指令格式核心主要包括四种（R/I/S/U/B/J），具体格式如下图所示。



三、寄存器定义

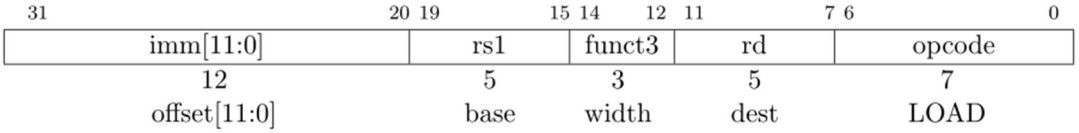
RV32I 共设置 32 个寄存器 x0-x31，其中 x0 硬连线为 0，x1-x31 为通用寄存器。另外有一个 pc 的寄存器，用来保存当前指令的地址。位宽为 32 位。x1 作为返回地址寄存器，x2 用作栈指针寄存器，x5 作为备用链路寄存器。



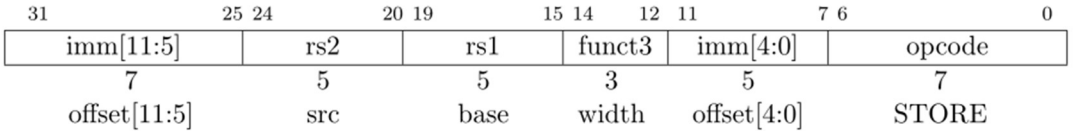
四、RV32I 指令功能介绍

1. Load/Store 指令

RV32I 中只有 load 和 store 类指令可以访问存储器，指令格式如下图,需要注意的是 imm 字段为有符号的立即数。



31–20	19–15	14–12	11–7	6–0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD



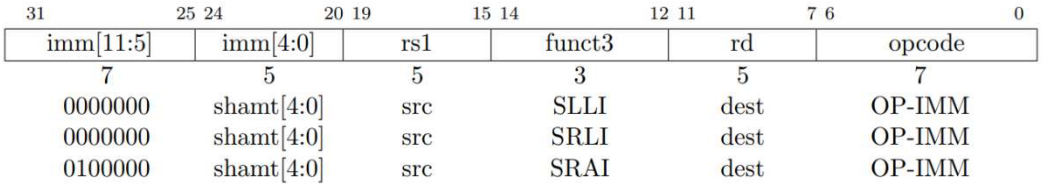
31–25	24–20	19–15	14–12	11–7	6–0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

Load 为 I 类格式,Load 类指令的储存器地址是通过 rs1+imm 的偏移得到的，指令功能都是将存储器的值复制到 rd 寄存器中。Load 类指令拥有五条指令，分别是 LB/LH/LW/LBU/LHU，LW(load word)将 32 位值读取到 rd 寄存器中，LH(load half word)从储存器中读取 16 位，然后将其符号扩展到 32 位，保存到 rd 寄存器中。LHU(load harf word unsigned)指令读取存储器 16 位，高位补零后保存到 rd 寄存器中。相应的，LB/LBU(load byte/load byte unsigned)则是读取 8 位，并进行相应的扩展保存操作。

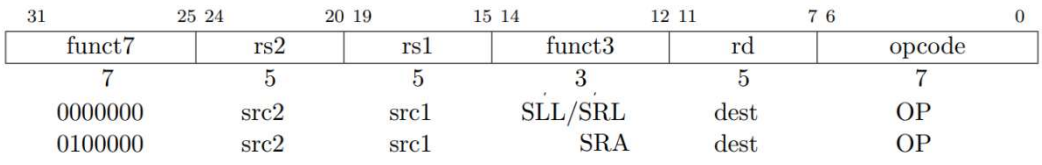
Store 为 S 类指令格式，Store 将 rs2 对应的寄存器中的值存储到 rs1+imm 地址的存储器单元中。Store 类指令共有三条指令，分别为 SW/SH/SB(store word/store half word/store byte)，SW/SH/SB 分别将寄存器 rs2 中的低 32/16/8/位保存到 rs1+imm 地址的存储器中。

2. 位运算指令

位移指令主要分为左移指令和右移指令，其中左移指令有两条，分别为 SLL/SLLI(shift left logic/shift left logic immediate)，右移指令有四条，分别是 SRL/SRLI/SRA/SRAI(shift right logic/shift right logic immediate/shift right arithmetic/shift right arithmetic immediate)，相应的指令个数如下图所示。



31–25	24–20	19–15	14–12	11–7	6–0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

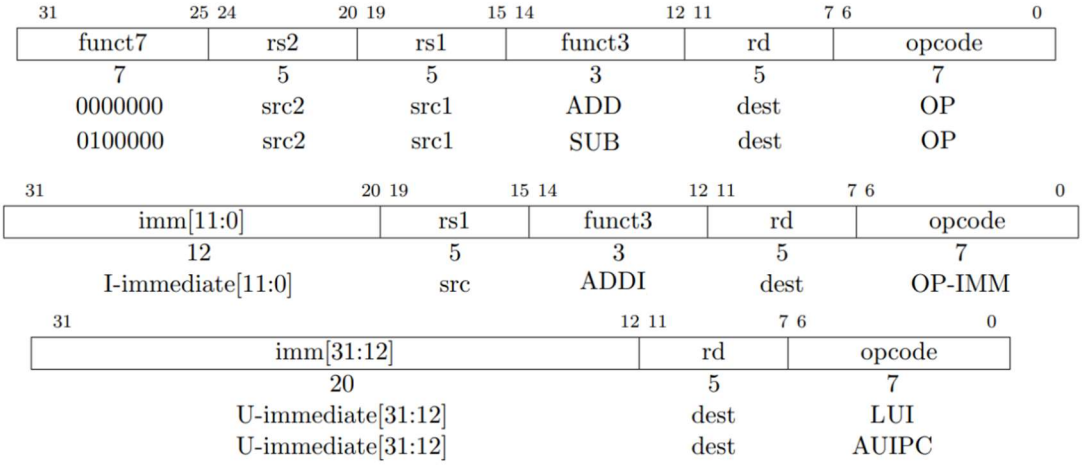


31–25	24–20	19–15	14–12	11–7	6–0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SRA	dest	OP

SLL 指令将 rs1 寄存器左移 rs2 寄存器值的次数，空位补零；SLLI 指令将 rs1 寄存器左移立即数次数，空位补零；SRL 指令将 rs1 寄存器右移 rs2 寄存器值的次数，空位补零；SRLI 将 rs1 寄存器右移立即数次数，空位补零；SRA 指令将 rs1 寄存器右移 rs2 寄存器值的次数，空位用最高位填充；SRAI 指令将 rs1 寄存器右移立即数次数，空位用最高位填充。

3. 算术指令

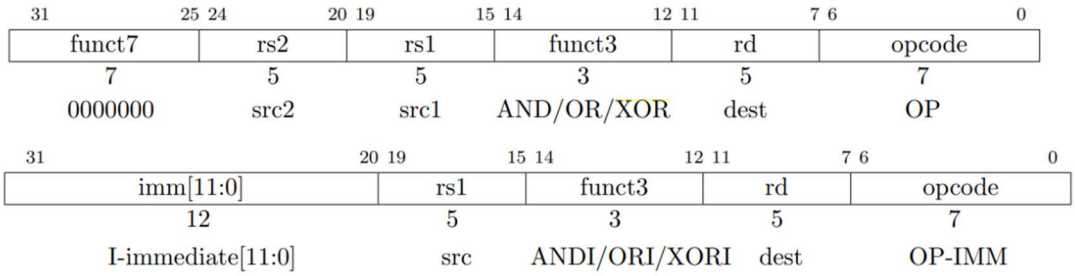
算术类指令主要包括五条指令，分别是 ADD/ADDI/SUB/LUI/AUIPC(add/add immediate/subtract/load upper immediate/add upper immediate to PC)。其对应的指令格式如下图所示。



ADD: x[rd] = x[rs1] + x[rs2] 忽略算术溢出。
ADDI: x[rd] = imm(12bit)+x[rs1] 忽略算术溢出。
SUB: x[rd] = x[rs1] - x[rs2] 忽略算术溢出。
LUI: x[rd] = imm << 12，低位补零。
AUIPC: x[rd] = PC[AUIPC] + (imm << 12)(低位补零)。

4. 逻辑指令

逻辑类指令主要包括六条指令，分别是 XOR/XORI/OR/ORI/AND/ANDI，代表了按位异或操作，按位异或立即数、按位或操作、按位或立即数操作、按位与操作、按位与立即数操作，指令格式如下图所示。

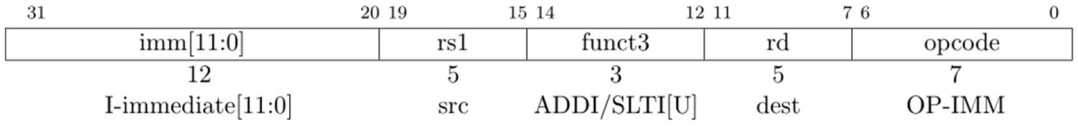


ANDI、ORI、XORI 均为逻辑操作，将 rs1 寄存器和符号扩展位上的 12bit 按位与、或、异或操作。

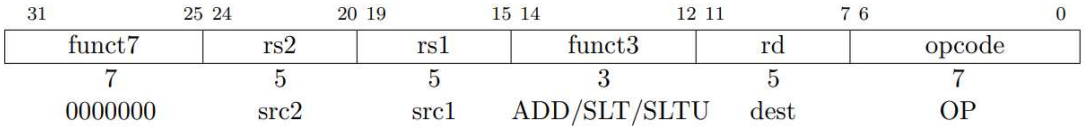
5. 比较指令

比较类指令主要包括四条指令，分别是 SLT/SLTI/SLTU/SLTIU(set less than/set less than

immediate/set less than unsigned/set less than immediate unsigned)。



31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM

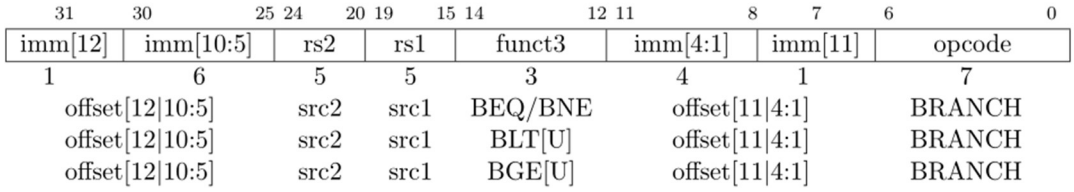


31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP

SLTI　　if(x[rs1]<imm) x[rd]=1; else x[rd]=0;
SLTIU 功能同 SLTI，不过 x[rs1]和 imm 被当作无符号数
SLT　　有符号位比较，如果 x[rs1]<x[rs2]，x[rd]=1，否则 x[rd]=0.
SLTU　　无符号位比较，如果 x[rs1]<x[rs2]，x[rd]=1，否则 x[rd]=0.

6. 分支指令

分支类指令主要包括六条指令，分别是 BEQ/BNE/BLT/BGE/BLTU/BGEU(branch equal/branch not equal/branch less than/branch greater than or equal/branch less than unsinged/branch greater than or equal unsigned)，其指令格式如下图所示。

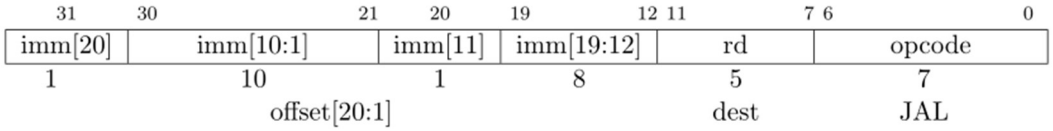


31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

BEQ/BNE　　if (x[rs1] ==/!= x[rs2]) then pc += offset
BLT/BLTU　　if ((unsinged)x[rs1] < (unsinged)x[rs2]) then pc += offset
BGE/BGEU　　if ((unsinged)x[rs1] >= (unsinged)x[rs2]) then pc += offset

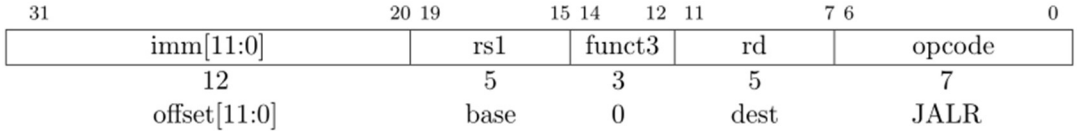
7. 跳转指令

跳转类指令主要包括两条指令，分别是 JAL/JALR(jump and link/jump and link register)



31	30 21	20	19 12	11 7	6 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7
	offset[20:1]			dest	JAL

JAL 在立即数处编码了一个有符号偏移量，这个偏移量加到 pc 上后形成跳转目标地址，并将跳转指令后面指令的地址(pc+4)加载到 rd，跳转范围为±1MB。标准软件调用约定使用寄存器 x1 来作为返回地址寄存器。



31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	0	dest	JALR

JALR(jump and link register) 通过有符号立即数加上 rs1,然后将结果的最低位设置为 0，作为目标地址，将跳转指令后面的地址存到 rd 中。

如果目标地址没有对齐到 32 位，JAL 和 JALR 指令均会产生一个非对齐指令取址异常。

所有无条件跳转指令都是用 pc 相对寻址，有助于支持位置无关代码。JALR 可以用来跳转到任何 32 位绝对地址空间。

首先 LUI 将目标地址的高 20 位加载到 rs1 中，然后 JALR 可以加上低 12 位。事实上，绝大多数 JALR 指令的使用要么是一个立即数 0，要么就是配合 LUI 或者 AUIPC 来跳转到 32 位地址空间

8. 同步指令

31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
fm	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
4	1	1	1	1	1	1	1	1	5	3	5	7
FM	predecessor				successor				0	FENCE	0	MISC-MEM

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
0	0	FENCE.I	0	MISC-MEM

主要用于其他 RISCV harts 和外部设备或者协处理查看的设备输入/输出和内存访问。与内存一致性模型相关。

9. 环境调用和断点

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

ECALL 和 EBREAK 与特权级相关，当前软核暂且使用不到。

10.HINT 指令

RV32I 为 HINT 指令预留了很大的编码空间，通常用于向微体系结构传递性能提示。不会改变任何体系结构上可见的状态。

第二部分：扩展指令

经小组讨论，为实现功能完整性和充分发挥 RISC-V 性能优势，决定扩展指令使用 M 和 C 指令的内容。

RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

一、 M 扩展

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
MULDIV	multiplier	multiplicand	MUL/MULH[[S]U]	dest	OP
MULDIV	multiplier	multiplicand	MULW	dest	OP-32

（1）MUL x[rd]=x[rs1]*x[rs2]，并将低位 XLEN 长度数据存入目标寄存器。

（2）MULH、MULHU、MULHSU 执行相同的乘法，但分别返回完整的 2*XLEN 位乘法的高 XLEN 位，用于有符号*有符号、无符号*无符号和有符号*无符号乘法。

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
MULDIV	divisor	dividend	DIV[U]/REM[U]	dest	OP
MULDIV	divisor	dividend	DIV[U]W/REM[U]W	dest	OP-32

（3）DIV/DIVU 有符号/无符号 rs1/rs2

（4）REM/REMU 有符号/无符号取余

二、 C 扩展

C 表示标准压缩指令集扩展。它通过常见操作添加短的 16 位指令编码来减少静态和动态代码大小。通常来说，一个程序中 50%-60%的 RISC-V 指令可以被 RVC 指令替换，导致代码大小减少 25%-30%。

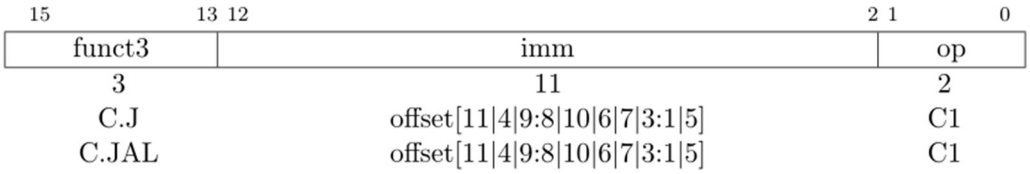
Format	Meaning	15 14 13 12	11 10 9 8 7	6 5 4 3 2	1 0
CR	Register	funct4	rd/rs1	rs2	op
CI	Immediate	funct3 | imm	rd/rs1	imm	op
CSS	Stack-relative Store	funct3	imm	rs2	op
CIW	Wide Immediate	funct3	imm	rd′	op
CL	Load	funct3 | imm	rs1′ | imm	rd′	op
CS	Store	funct3 | imm	rs1′ | imm	rs2′	op
CA	Arithmetic	funct6	rd′/rs1′ | funct2	rs2′	op
CB	Branch	funct3 | offset	rs1′	offset	op
CJ	Jump	funct3	jump target		op

15 13	12	11 7	6 2	1 0
funct3	imm	rd	imm	op
3	1	5	5	2
C.LWSP	offset[5]	dest≠0	offset[4:2|7:6]	C2
C.LDSP	offset[5]	dest≠0	offset[4:3|8:6]	C2
C.LQSP	offset[5]	dest≠0	offset[4|9:6]	C2
C.FLWSP	offset[5]	dest	offset[4:2|7:6]	C2
C.FLDSP	offset[5]	dest	offset[4:3|8:6]	C2

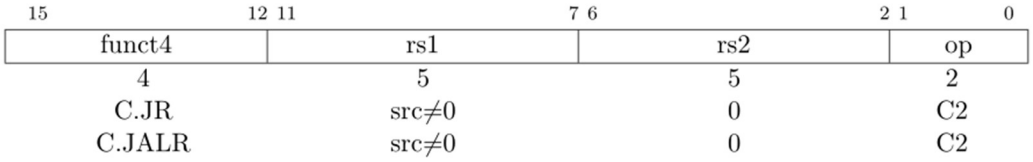
15 13	12 7	6 2	1 0
funct3	imm	rs2	op
3	6	5	2
C.SWSP	offset[5:2|7:6]	src	C2
C.SDSP	offset[5:3|8:6]	src	C2
C.SQSP	offset[5:4|9:6]	src	C2
C.FSWSP	offset[5:2|7:6]	src	C2
C.FSDSP	offset[5:3|8:6]	src	C2

15 13	12 10	9 7	6 5	4 2	1 0
funct3	imm	rs1′	imm	rd′	op
3	3	3	2	3	2
C.LW	offset[5:3]	base	offset[2|6]	dest	C0
C.LD	offset[5:3]	base	offset[7:6]	dest	C0
C.LQ	offset[5|4|8]	base	offset[7:6]	dest	C0
C.FLW	offset[5:3]	base	offset[2|6]	dest	C0
C.FLD	offset[5:3]	base	offset[7:6]	dest	C0

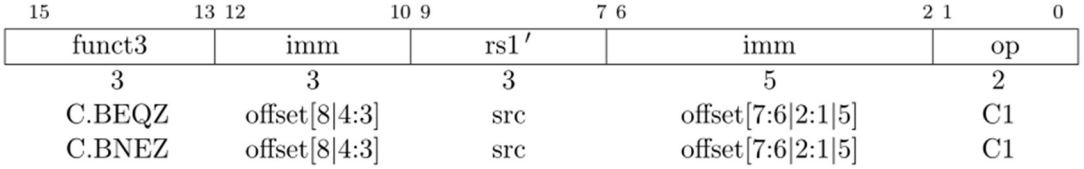
15 13	12 10	9 7	6 5	4 2	1 0
funct3	imm	rs1′	imm	rs2′	op
3	3	3	2	3	2
C.SW	offset[5:3]	base	offset[2|6]	src	C0
C.SD	offset[5:3]	base	offset[7:6]	src	C0
C.SQ	offset[5|4|8]	base	offset[7:6]	src	C0
C.FSW	offset[5:3]	base	offset[2|6]	src	C0
C.FSD	offset[5:3]	base	offset[7:6]	src	C0



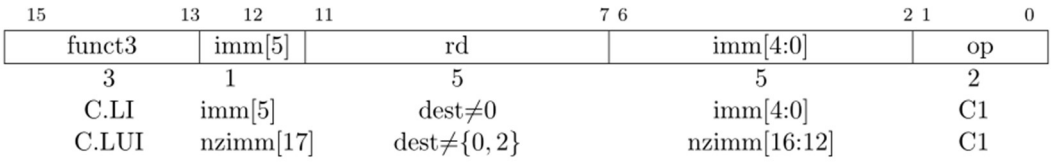
15	13 12	2 1	0
funct3	imm	op	
3	11	2	
C.J	offset[11|4|9:8|10|6|7|3:1|5]	C1	
C.JAL	offset[11|4|9:8|10|6|7|3:1|5]	C1	



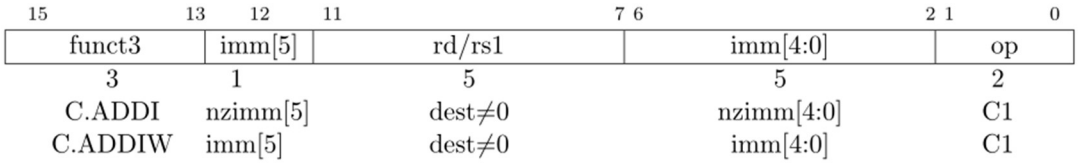
15	12 11	7 6	2 1	0
funct4	rs1	rs2	op	
4	5	5	2	
C.JR	src≠0	0	C2	
C.JALR	src≠0	0	C2	



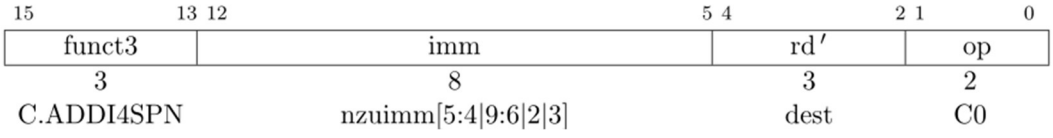
15	13 12	10 9	7 6	2 1	0
funct3	imm	rs1′	imm	op	
3	3	3	5	2	
C.BEQZ	offset[8|4:3]	src	offset[7:6|2:1|5]	C1	
C.BNEZ	offset[8|4:3]	src	offset[7:6|2:1|5]	C1	



15	13	12	11	7 6	2 1	0
funct3	imm[5]		rd	imm[4:0]	op	
3	1		5	5	2	
C.LI	imm[5]		dest≠0	imm[4:0]	C1	
C.LUI	nzimm[17]		dest≠{0, 2}	nzimm[16:12]	C1	



15	13	12	11	7 6	2 1	0
funct3	imm[5]		rd/rs1	imm[4:0]	op	
3	1		5	5	2	
C.ADDI	nzimm[5]		dest≠0	nzimm[4:0]	C1	
C.ADDIW	imm[5]		dest≠0	imm[4:0]	C1	
C.ADDI16SP	nzimm[9]		2	nzimm[4|6|8:7|5]	C1	



15	13 12	5 4	2 1	0
funct3	imm	rd′	op	
3	8	3	2	
C.ADDI4SPN	nzuimm[5:4|9:6|2|3]	dest	C0	