目录

[第一部分：RV32I指令集 2](#_Toc74732468)

[一、RV32I指令列表 2](#_Toc74732469)

[二、RV32I指令格式 3](#_Toc74732470)

[三、寄存器定义 3](#_Toc74732471)

[四、RV32I指令功能介绍 3](#_Toc74732472)

[1. Load/Store指令 3](#_Toc74732473)

[2. 位运算指令 4](#_Toc74732474)

[3. 算术指令 5](#_Toc74732475)

[4. 逻辑指令 5](#_Toc74732476)

[5. 比较指令 5](#_Toc74732477)

[6. 分支指令 6](#_Toc74732478)

[7. 跳转指令 6](#_Toc74732479)

[8. 同步指令 7](#_Toc74732480)

[9. 环境调用和断点 7](#_Toc74732481)

[10. HINT指令 7](#_Toc74732482)

[第二部分：扩展指令 8](#_Toc74732483)

[一、 M扩展 8](#_Toc74732484)

[二、 C扩展 8](#_Toc74732485)

# 第一部分：RV32I指令集

## 一、RV32I指令列表

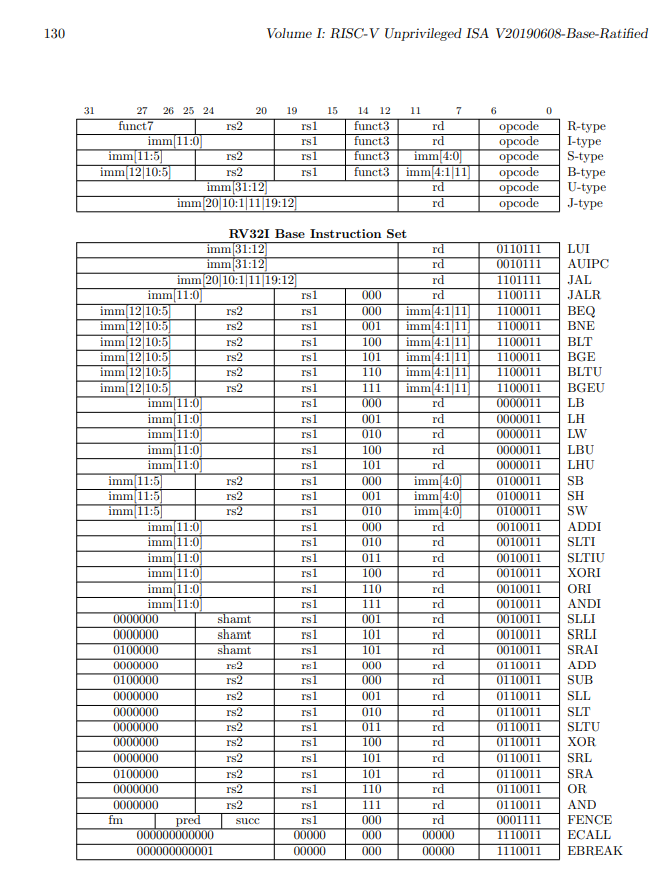
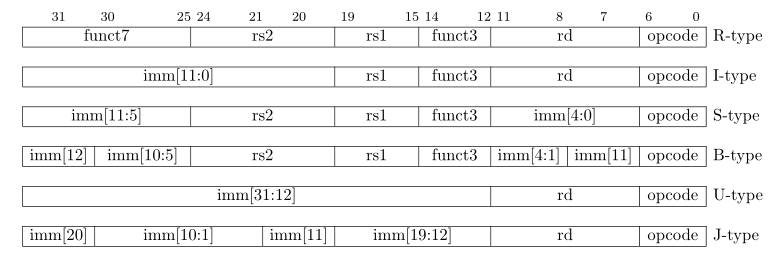


表 1 RV32I指令

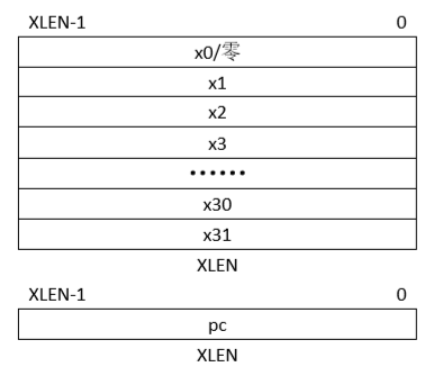
## 二、RV32I指令格式

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

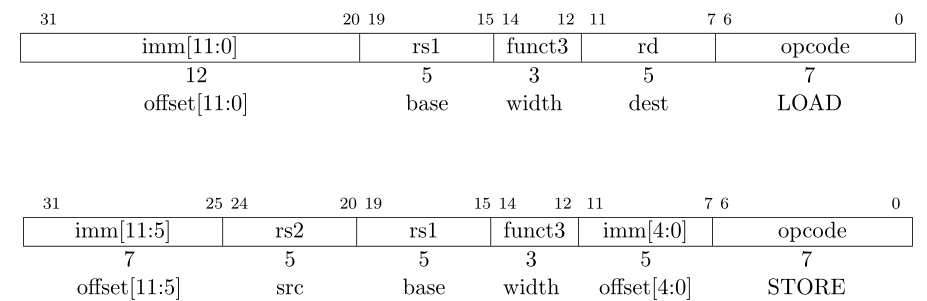
RV32I共设置32个寄存器x0-x31，其中x0硬连线为0，x1-x31为通用寄存器。另外有一个pc的寄存器，用来保存当前指令的地址。位宽为32位。x1作为返回地址寄存器，x2用作栈指针寄存器，x5作为备用链路寄存器。



## 四、RV32I指令功能介绍

### 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。

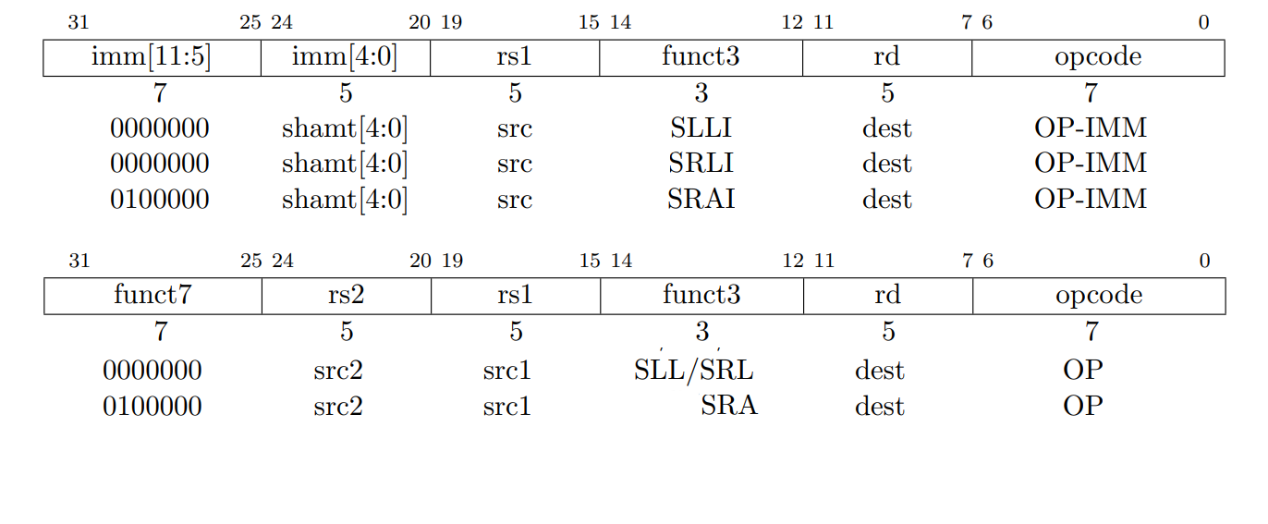


Load 为I类格式，Load类指令的储存器地址是通过rs1+imm的偏移得到的，指令功能都是将存储器的值复制到 rd 寄存器中。Load类指令拥有五条指令，分别是LB/LH/LW/LBU/LHU, LW(load word)将32位值读取到 rd 寄存器中，LH(load half word)从储存器中读取16位，然后将其符号扩展到32位，保存到 rd 寄存器中。LHU(load harf word unsigned)指令读取存储器16位，高位补零后保存到rd寄存器中。相应的，LB/LBU(load byte/load byte unsigned)则是读取8位，并进行相应的扩展保存操作。

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。

### 位运算指令

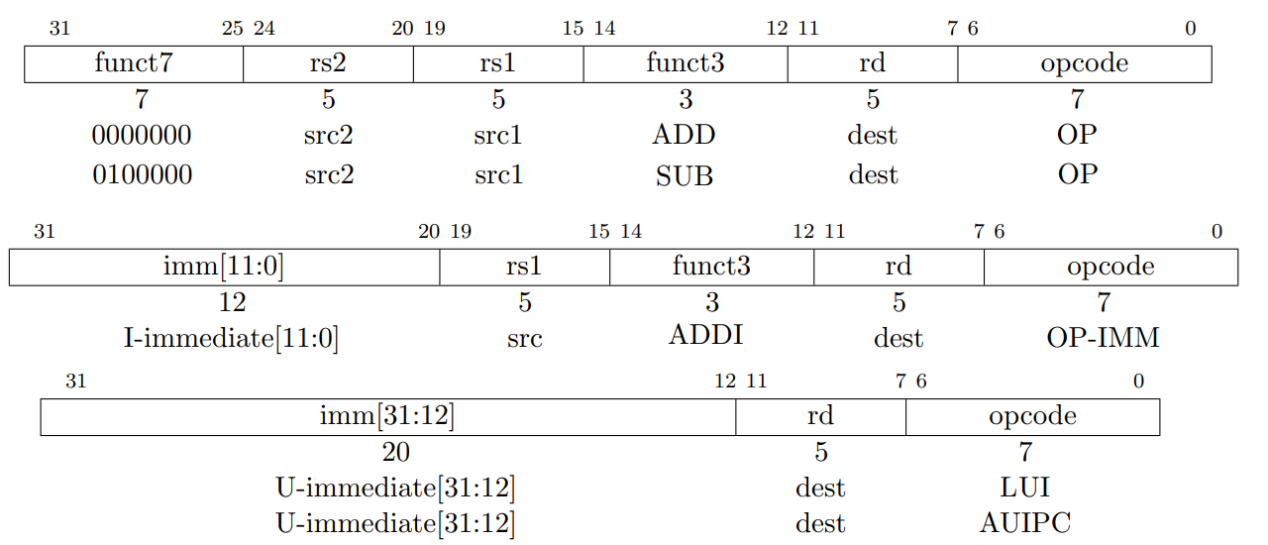
位移指令主要分为左移指令和右移指令，其中左移指令有两条，分别为SLL/SLLI(shift left logic/shift left logic immediate)，右移指令有四条，分别是SRL/SRLI/SRA/SRAI(shift right logic/shift right logic immediate/shift right arithmetic/shift right arithmetic immediate )，相应的指令个数如下图所示。



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。

### 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。

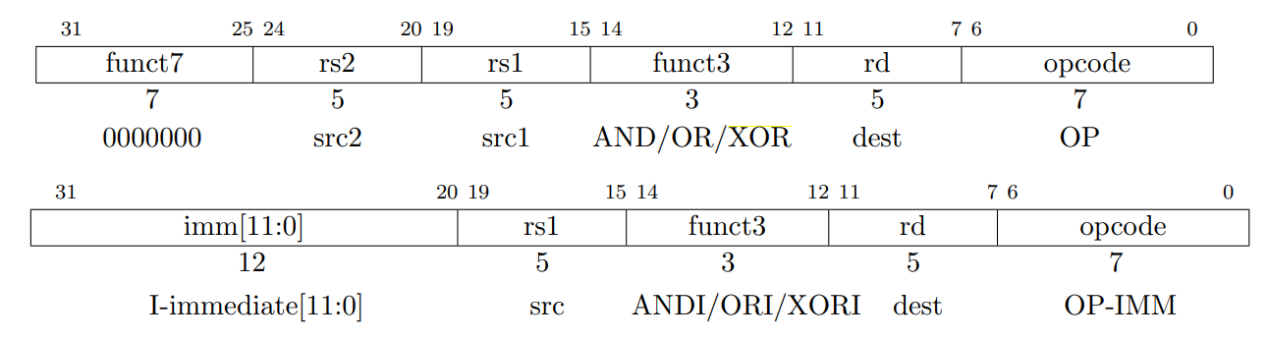
SUB: x[rd] = x[rs1] – x[rs2] 忽略算术溢出。

LUI: x[rd] = imm << 12，低位补零。

AUIPC: x[rd] = PC[AUIPC] + (imm << 12)(低位补零)。

### 逻辑指令

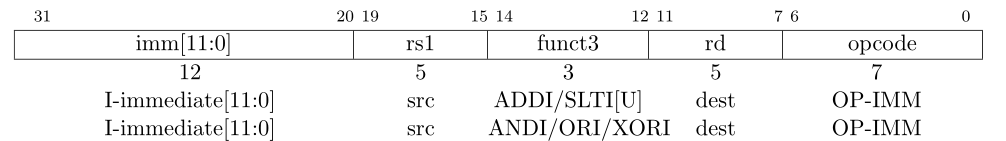
逻辑类指令主要包括六条指令，分别是XOR/XORI/OR/ORI/AND/ANDI，代表了按位异或操作，按位异或立即数、按位或操作、按位或立即数操作、按位与操作、按位与立即数操作，指令格式如下图所示。

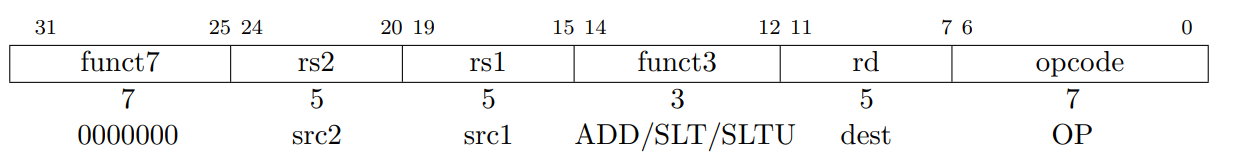


ANDI、ORI、XORI均为逻辑操作，将rs1寄存器和符号扩展位上的12bit按位与、或、异或操作。

### 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)。





SLTI if(x[rs1]<imm) x[rd]=1; else x[rd]=0;

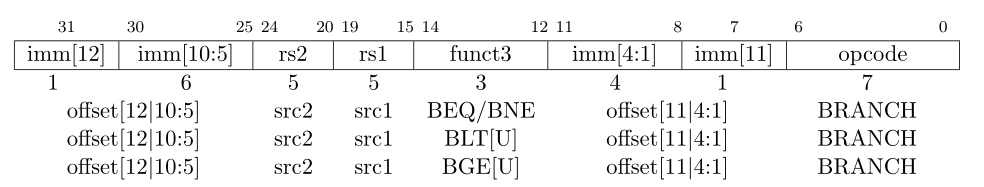
SLTIU功能同SLTI，不过x[rs1]和imm被当作无符号数

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.

### 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。



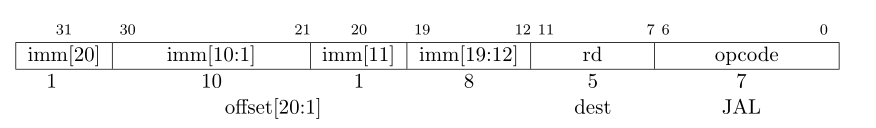
BEQ/BNE if (x[rs1] ==/!= x[rs2]) then pc += offset

BLT/BLTU if ((unsinged)x[rs1] < (unsinged)x[rs2]) then pc += offset

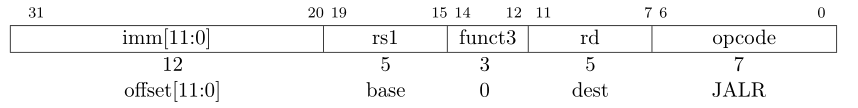
BGE/BGEU if ((unsinged)x[rs1] >= (unsinged)x[rs2]) then pc += offset

### 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)



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。



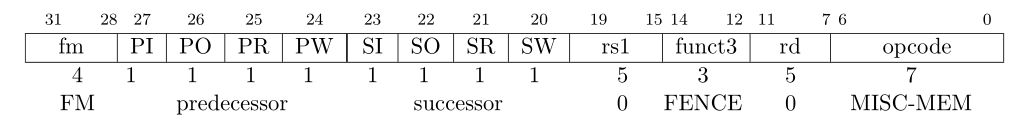
JALR(jump and link register) 通过有符号立即数加上rs1,然后将结果的最低位设置为0，作为目标地址，将跳转指令后面的地址存到rd中。

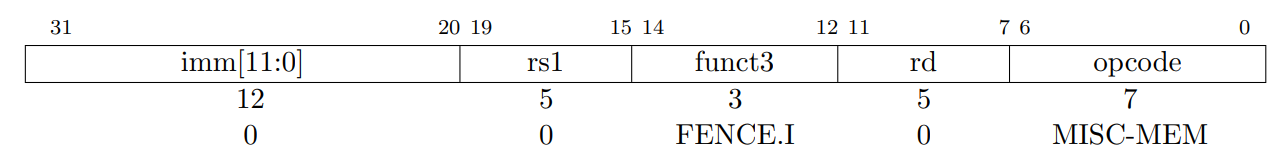
如果目标地址没有对齐到32位，JAL和JALR指令均会产生一个非对齐指令取址异常。

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

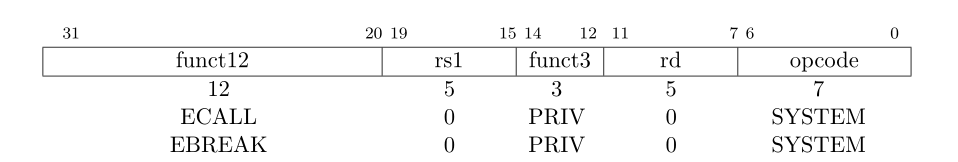
### 同步指令





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。

### 环境调用和断点



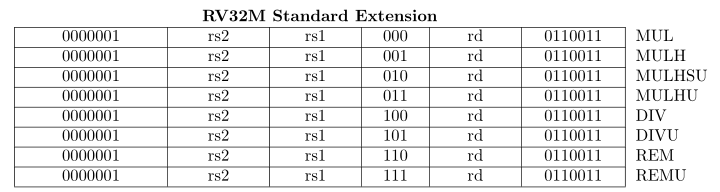
ECALL和EBREAK与特权级相关，当前软核暂且使用不到。

### HINT指令

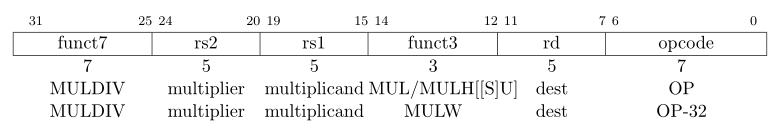
RV32I为HINT指令预留了很大的编码空间，通常用于向微体系结构传递性能提示。不会改变任何体系结构上可见的状态。

# 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。

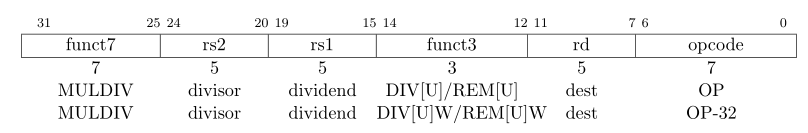


## M扩展



（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。



（3）DIV/DIVU 有符号/无符号rs1/rs2

（4）REM/REMU 有符号/无符号取余

## 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%。

