RISCV BPF JIT支持扩展调研报告

> *本报告基于：*
>
> *kernel mainline*
>
> *RISC-V Instruction Set Manual*
>
> *BPF Instruction Set Architecture (ISA) — The Linux Kernel documentation*

本报告将介绍目前在RISCV BPF JIT编译器中已支持的扩展及指令，并对未支持的指令分析其原因。

其中已支持的指令用“（已支持）”标注

1、RV64I

RV64I 为基本的整数运算指令集，包含了RV32I中所有指令。在RV64I中，寄存器宽度为 64 位，能够处理 64 位的数据操作和地址空间。

其中，RV32I指令如下：

1. Integer Register-Immediate Instructions（整数 寄存器-立即数指令）
    a. ADDI（已支持）：将符号扩展的12bit立即数加到rs1寄存器中
    b. SLTI：当rs1寄存器中的值和立即数都是有符号数，如果寄存器中的值比立即数小，则将rs1的值设为1，否则设为0
    c. SLTIU：功能同上，但对比为无符号数
    d. ANDI（已支持）：将rs1寄存器的值和12bit立即数进行按位与操作，结果写入rd寄存器
    e. ORI（已支持）：将rs1寄存器的值和12bit立即数进行按位或操作，结果写入rd寄存器
    f. XORI（已支持）：将rs1寄存器的值和12bit立即数进行按位异或操作，结果写入rd寄存器
    g. SLLI（已支持）：将rs1寄存器的值逻辑左移一个立即数，该值存放在立即数域的低五位

h. SRLI（已支持）：将rs1寄存器的值逻辑右移一个立即数，该值存放在立即数域的低五位

i. SRAI（已支持）：将rs1寄存器的值算数右移一个立即数，该值存放在立即数域的低五位

2. Integer Register-Register Operations（整数 寄存器-寄存器操作）

a. ADD（已支持）：将rs1和rs2寄存器中的值相加，写入rd寄存器中，溢出被忽略

b. SUB（已支持）：将rs1中的值减去rs2中的值，结果写入rd寄存器

c. SLT：当rs1和rs2中都为有符号数，且rs1<rs2，则将1写入rd，否则将0写入rd

d. SLTU（已支持）：功能同上，但进行无符号比较

e. AND（已支持）：对rs1和rs2寄存器进行按位与操作，结果写入rd寄存器

f. OR（已支持）：对rs1和rs2寄存器进行按位或操作，结果写入rd寄存器

g. XOR（已支持）：对rs1和rs2寄存器进行按位异或操作，结果写入rd寄存器

h. SLL（已支持）：将rs1寄存器中的值逻辑左移rs2寄存器的低5位

i. SRL（已支持）：将rs1寄存器中的值逻辑右移rs2寄存器的低5位

j. SRA（已支持）：将rs1寄存器中的值算数右移rs2寄存器的低5位

3. NOP Instruction（已支持）：编码为ADDI x0, x0, 0。

4. Control Transfer Instructions（控制转移指令）

a. Unconditional Jumps（非条件跳转）

i. JAL（已支持）：该指令将一个偏移值乘2字节加到当前指令地址来生成一个目标地址

ii. JALR（已支持）：该指令将一个符号扩展的12位立即数加到寄存器rs1上，然后将结果的最低有效位设为0

b. Conditional Branches（条件分支）

i. BEQ（已支持）：如果rs1和rs2寄存器中的值相等，则跳转到12bit的立即数乘2字节和当前指令地址相加的结果目标地址中

ii. BNE（已支持）：执行分支操作，但条件为rs1和rs2寄存器的值不相等时

iii. BLT（已支持）：执行分支操作，但条件为有符号的rs1小于有符号的rs2

iv. BLTU（已支持）：执行分支操作，但条件为无符号的rs1小于有符号的rs2

v. BGT（已支持）：执行分支操作，但条件为有符号的rs1大于有符号的rs2

vi. BGTU（已支持）：执行分支操作，但条件为无符号的rs1大于有符号的rs2

vii. BGE（已支持）：执行分支操作，但条件为有符号的rs1大于等于有符号的rs2

viii. BGEU（已支持）：执行分支操作，但条件为无符号的rs1大于等于有符号的rs2

ix. BLE（已支持）：执行分支操作，但条件为有符号的rs1小于等于有符号的rs2

x. BLEU（已支持）：执行分支操作，但条件为无符号的rs1大于等于有符号的rs2

5. Load and Store Instructions（加载和存储指令）

a. LW（已支持）：该指令从内存中取出32bit的数加载到rd寄存器中

b. LH（已支持）：该指令从内存中取出16bit的数，符号扩展到32bit后加载到rd寄存器中

c. LHU（已支持）：功能同上，但执行0扩展

d. LB（已支持）：功能同LH指令，但操作数为8bit数

e. LBU（已支持）：功能同上，但执行0扩展

f. SW（已支持）：将32bit数从rs2寄存器中存储到内存中

g. SH（已支持）：将rs2寄存器的低16bit存储到内存中

h. SB（已支持）：将rs2寄存器的低8bit存储到内存中

6. Memory Ordering Instructions（内存排序指令）

FENCE（已支持）：该指令用于确保在多处理器系统中某些操作的顺序性。在该指令之后的操作必须在该指令前的操作完成后才能被观察到

7. Environment Call and Breakpoints（环境调用和断点）

a. SYSTEM：该指令用于访问可能需要特权访问的系统功能。这些指令可以分为两个主要类别：原子性地读取-修改-写入控制和状态寄存器（CSRs）的指令，以及所有其他潜在的特权指令

b. ECALL：该指令用于向执行环境发出服务请求

c. EBREAK：该指令用于将控制权交还给调试环境

下面是除了RV32I外的指令：

1. Integer Computational Instructions（整数计算指令）

a. Integer Register-Immediate Instructions（整数 寄存器-立即数指令）

i. ADDIW（已支持）：将一个12bit的符号扩展立即数和rs1寄存器中的值相加，将符号扩展的32bit结果写入rd寄存器中。溢出会被忽略，32位结果被符号扩展到64位。

ii. SLLI（已支持）：将rs1寄存器中的值逻辑左移6bit的立即数，结果写入rd寄存器中

iii. SRLI（已支持）：将rs1寄存器中的值逻辑右移6bit的立即数，结果写入rd寄存器中

iv. SRAI（已支持）：将rs1寄存器中的值算术右移6bit的立即数，结果写入rd寄存器中

v. SLLIW（已支持）：是RV64I独有指令，功能与SLLI类似，但操作数是32bit数，并将结果符号扩展到64bit

vi. SRLIW（已支持）：是RV64I独有指令，功能与SRLI类似，但操作数是32bit数，并将结果符号扩展到64bit

vii. SRAIW（已支持）：是RV64I独有指令，功能与SRAI类似，但操作数是32bit数，并将结果符号扩展到64bit

viii. LUI（已支持）：将一个20bit立即数写入rd寄存器的高20位，将低12位设为0，结果符号扩展到64位

ix. AUIPC（已支持）：构成一个偏移量，其中低12为0，高20位为立即数，然后同pc相加，结果写入rd寄存器中

b. Integer Register-Register Operations（整数 寄存器-寄存器操作）

i. ADDW（已支持）：将rs1和rs2寄存器的低32位相加，结果截断为32位，将结果符号扩展到64位，写入rd寄存器

ii. SUBW（已支持）：用rs1寄存器的值减去rs2寄存器的值，结果截断为32位，将结果符号扩展到64位，写入rd寄存器

iii. SLLW（已支持）：进行逻辑左移，但操作数为32位，并将结果符号扩展到64位

iv. SRLW（已支持）：进行逻辑右移，但操作数为32位，并将结果符号扩展到64位

v. SRAW（已支持）：进行算术左移，但操作数为32位，并将结果符号扩展到64位

c. Load and Store Instructions（加载和存储指令）

i. LD（已支持）：从内存中取出64bit数加载到rd寄存器中

ii. LW（已支持）：从内存中取出32bit数，并符号扩展到64位，再加载到rd寄存器中

iii. LWU（已支持）：功能同上，但是将32bit数零扩展到64位

iv. LH（已支持）：功能同LW，但操作数为16位

v. LHU（已支持）：功能同上，但是进行零扩展

vi. LB（已支持）：功能同LW，但是操作数位8位

vii. LBU（已支持）：功能同上，但是进行零扩展

viii. SD（已支持）：将rs2寄存器中的值存储到内存中

ix. SW（已支持）：功能同上，但是操作数为rs2中低32位

x. SH（已支持）：功能同上，但是操作数为rs2中低16位

xi. SB（已支持）：功能同上，但是操作数为rs2中低8位

可以看到，除了SLTI、SLTIU和SLT指令外，其余指令均得到支持。

当前BPF指令中并没有寄存器和立即数，寄存器和寄存器比值大小的指令，因此这几条指令也无需实现。

2、M扩展

M扩展提供了乘法、除法、求余运算指令。具体包含以下指令：

1. Multiplication Operations（乘法操作）
    a. MUL（已支持）：将rs1和rs2寄存器中的值相乘，将结果截断到寄存器长度，将低XLEN写入rd寄存器中
    b. MULH（已支持）：将有符号的rs1和rs2寄存器中的值相乘，将结果截断到寄存器长度，将高XLEN写入rd寄存器中
    c. MULHU（已支持）：将无符号的rs1和rs2寄存器中的值相乘，将结果截断到寄存器长度，将高XLEN写入rd寄存器中
    d. MULHSU：将rs1中的有符号值和rs2中的无符号值相乘，将结果截断到寄存器长度，将高XLEN写入rd寄存器中
    e. MULW（已支持）：该指令为RV64指令。该指令计算原寄存器的低32的乘积，并将结果低32位进行符号扩展写入rd寄存器中
2. Division Operations（除法指令）
    a. DIV（已支持）：对rs1和rs2寄存器中的有符号数进行除法，并将结果向0舍入
    b. DIVU（已支持）：对rs1和rs2寄存器中的无符号数进行除法，并将结果向0舍入
    c. REM（已支持）：对有符号除法进行取余操作
    d. REMU（已支持）：对无符号除法进行取余操作
    e. DIVW（已支持）：该指令位RV64指令。对rs1和rs2寄存器的低32位进行有符号除法，将32位结果写入rd寄存器，并符号扩展到64位

f. DIVUW（已支持）：该指令位RV64指令。对rs1和rs2寄存器的低32位进行无符号除法，将32位结果写入rd寄存器，并符号扩展到64位

g. REMW（已支持）：对有符号除法进行取余操作，并将结果进行符号扩展到64位

h. REMUW（已支持）：对无符号除法进行取余操作，并将结果进行符号扩展到64位

除了MULHSU指令，其他指令均已支持。

在当前RISC-V JIT编译器中，BPF乘法指令被翻译为MUL和MULW指令，也就是只保留了结果的低XLEN位。MULHSU指令用于生成乘法结果的高XLEN位，因此若要保留乘法结果的高XLEN或完整的2XLEN位，则需要支持该指令。

3、A扩展

A扩展为原子指令扩展，由 Zaamo 扩展和 Zalrsc 扩展组成

3.1 Zaamo扩展

Zaamo扩展提供了对内存中数据进行原子操作的指令，包括交换、加法、按位与、按位或、按位异或、计算最大/小值等。

这些指令原子地将rs1寄存器中保存的地址中的值加载到rd寄存器中，并将刚刚的值和rs2寄存器中的值进行如下运算后，将结果写入到rs1寄存器中的地址处。具体包含以下操作：

1. AMOSWAP.W（已支持）：原子交换32bit数
2. AMOSWAP.D（已支持）：原子交换64bit数
3. AMOADD.W（已支持）：32bit数做原子加法
4. AMOADD.D（已支持）：64bit数做原子加法
5. AMOAND.W（已支持）：32bit数做原子按位与操作
6. AMOAND.D（已支持）：64bit数做原子按位与操作
7. AMOOR.W（已支持）：32bit数做原子按位或操作
8. AMOOR.D（已支持）：64bit数做原子按位或操作
9. AMOMAX[U].W：有[无]符号32bit数取最大值
10. AMOMAX[U].D：有[无]符号64bit数取最大值
11. AMOMIN[U].W：有[无]符号32bit数取最小值
12. AMOMIN[U].D：有[无]符号64bit数取最小值

除了计算最大/小值指令（AMOMAX、AMOMIN），其他指令均已支持。

当前BPF指令中没有对寄存器值取最大、最小值指令，因此上述指令也无需实现。

3.2 Zalrsc扩展

Zalrsc扩展提供了一对原子指令LR、SC。具体指令如下：

1. LR.D（已支持）：该指令用于从指定地址读取一个64bit值并保存到指定寄存器中
2. LR.W（已支持）：该指令功能同上，但操作数为32bit数
3. SC.D（已支持）：该指令将指定寄存器中的64bit值存储到指定地址中，如果自从上次执行 LR指令以来，内存地址没有被其他处理器或设备修改过，则存储操作成功，rd寄存器被设置为 0。否则，存储操作失败，rd寄存器 被设置为非零值（通常为 1）
4. SC.W（已支持）：该指令功能同上，但操作数为32bit数

这两条指令通常同时出现，在多线程同时访问共享内存时提供无锁同步机制。

4、Zacas扩展

Zacas扩展提供了原子CAS（Compare And Swap）指令。该指令在多线程同时访问共享内存时提供无锁同步机制。该扩展依赖于Zaamo扩展。具体指令如下：

1. AMOCAS.W（已支持）：从一个寄存器（rs1）中保存的地址中取出一个32bit数据，跟结果寄存器（rd）保存的预期值进行比较，如果两者相等，则将另一个寄存器（rs2）中的值写入到rs1寄存器中保存的内存地址处。
2. AMOCAS.D（已支持）：功能同上，但操作数为64bit数据
3. AMOCAS.Q：功能同上，但操作数为128bit数据

AMOCAS.Q指令尚未支持。

当前BPF指令未支持128操作数，因此上述指令无需实现。

5、B扩展

B扩展为原子指令扩展，由Zba、Zbb、Zbc和Zbs扩展组成，目前仅支持了Zba和Zbb中的部分指令

5.1 Zba扩展

Zba指令将索引值移位并加到基地址上，加速生成基本类型数组地址

该扩展包含以下指令：

1. ADD.UW：加无符号字
2. SH1ADD：左移一位再加
3. SH1ADD.UW：无符号字左移一位再加
4. SH2ADD（已支持）：左移两位再加
5. SH2ADD.UW：无符号字左移两位再加
6. SH3ADD（已支持）：左移三位再加
7. SH3ADD.UW：无符号字左移三位再加
8. SLLI.UW：无符号字左移立即数
9. ZEXT.W（已支持）：无符号字，是翻译为add.uw的伪指令

目前仅支持SH2ADD和SH3ADD指令和ZEXT.W伪指令

上述指令的主要用于加速地址的生成，其中ZEXT.W指令已经得到广泛应用。对于未实现的指令，这里推测是因为通过移位加和方法生成地址的场景较少，因此暂时不需要这些指令来加速，若后续上述场景增多，可考虑增加这些指令。

5.2 Zbb扩展

1. 简介：

   Zbb扩展为基本位操作扩展，包含以下指令：

   a. Logical with negate（逻辑与取反）

      i. ANDN：将rs2寄存器的值按位取反后，与rs1寄存器的值进行按位与运算

      ii. ORN：将rs2寄存器的值按位取反后，与rs1寄存器的值进行按位或运算

      iii. XNOR：将rs1和rs2寄存器的值按位异或后，将结果按位取反

   b. Count leading/trailing zero bits（计数前导0位/后缀0位）

i. CLZ：计算从最高位到第一个1之前0的个数

ii. CLZW：计算从第31位到第一个1之前0的个数

iii. CTZ：计算从最低位到第一个1之前0的个数

iv. CTZW：计算从最低位到第31位之间，第一个1之前0的个数

c. Count population（统计位1的个数）

i. CPOP：统计源寄存器中位1的数量

ii. CPOPW：统计源寄存器中最低有效字中位1的数量

d. Integer minimum/maximum（最小/最大整数）

i. MAX：返回两个有符号整数中最大的值

ii. MAXU：返回两个无符号整数中最大的值

iii. MIN：返回两个有符号整数中最小的值

iv. MINU：返回两个无符号整数中最小的值

e. Sign- and zero-extension（符合和零扩展）

i. SEXT.B（**已支持**）：将最低字节进行符号扩展

ii. SEXT.H（**已支持**）：将最低半字进行符号扩展

iii. ZEXT.H（**已支持**）：将最低半字进行零扩展

f. Bitwise rotation（按位旋转）

i. ROL：向左循环rs2的最低$log_2(XLEN)$位上的数

ii. ROLW：向左循环rs2的最低5位上的数，结果进行符号扩展

iii. ROR：向右循环rs2的最低$log_2(XLEN)$位上的数

iv. RORI：向右循环立即数的最低$log_2(XLEN)$位上的数

v. RORIW：向右循环立即数的最低$log_2(XLEN)$位上的数，结果进行符号扩展

vi. RORW：向右循环rs2的最低5位上的数，结果进行符号扩展

g. OR Combine（按位或合并）

i. OCR.B：检查源寄存器中的每个字节，如果字节上的位都为0，则结果寄存器对应的字节上的每一位也为0，否则都为1

h. Byte-reverse（字节反转）

i. REV8（**已支持**）：反转源寄存器中的字节序

目前仅支持SEXT.B、SEXT.H、ZEXT.H和REV8指令。

对于未支持的指令，暂时没有对应的BPF指令，因此无需实现。

6、C扩展

C扩展通过添加常用操作的16位指令，减少了动静态的代码体积。

该扩展包含以下几种指令格式：

1. CR：寄存器
2. CI：立即数
3. CSS：栈相关存储
4. CIW：宽立即数
5. CL：加载
6. CS：存储
7. CA：算数
8. CB：分支/算数
9. CJ：跳转

它包含以下指令:

1. Load and Store Instructions （加载存储指令）
    a. Stack-Pointer-Based Loads and Stores（基于栈指针的加载和存储）
        i. C.LWSP**(已支持)**：从内存加载32位值到寄存器中，将零扩展的偏移量乘4加到栈指针x2中计算地址。可扩展为lw rd, offset(x2)
        ii. C.LDSP**(已支持)**：是RV64C/RV128C独有指令。它从内存加载64位值到寄存器中，将零扩展的偏移量乘8加到栈指针x2中计算地址。可扩展为ld rd, offset(x2)
        iii. C.LQSP：是RV128C独有指令。它从内存加载128位值到寄存器中，将零扩展的偏移量乘16加到栈指针x2中计算地址。可扩展为lq rd, offset(x2)
        iv. C.FLWSP：是RV32FC独有指令。它从内存加载单精度浮点数到浮点寄存器中，将零扩展的偏移量乘4加到栈指针x2中计算地址。可扩展为flw rd, offset(x2)
        v. C.FLDSP：是RV32DC/RV64DC独有指令。它从内存加载双精度浮点数到浮点寄存器中，将零扩展的偏移量乘8加到栈指针x2中计算地址。可扩展为fld rd, offset(x2)
        vi. C.SWSP**(已支持)**：它从寄存器存储32位数到寄存器中，将零扩展的偏移量乘4加到栈指针x2中计算地址。可扩展为sw rs2, offset(x2)

vii. C.SDSP(已支持)：是RV64C/RV128C独有指令。它从寄存器存储64位数到寄存器中，将零扩展的偏移量乘8加到栈指针x2中计算地址。可扩展为sd rs2, offset(x2)

viii. C.SQSP：是RV128C独有指令。它从寄存器存储128位数到寄存器中，将零扩展的偏移量乘16加到栈指针x2中计算地址。可扩展为sq rs2, offset(x2)

ix. C.FSWSP：是RV32FC独有指令。它从浮点寄存器存储单精度浮点数到内存中，将零扩展的偏移量乘4加到栈指针x2中计算地址。可扩展为fsw rs2, offset(x2)

x. C.FSDSP：是RV32DC/RV64DC独有指令。它从浮点寄存器存储双精度浮点数到内存中，将零扩展的偏移量乘8加到栈指针x2中计算地址。可扩展为fsd rs2, offset(x2)

b. Register-Based Loads and Stores（基于寄存器的加载和存储）

i. C.LW(已支持)：从内存加载32位值到寄存器中，将零扩展的偏移量乘4加到rs1寄存器中计算地址。可扩展为lw rd, offset(rs1)

ii. C.LD(已支持)：是RV64C/RV128C独有指令。它从内存加载64位值到寄存器中，将零扩展的偏移量乘8加到rs1寄存器中计算地址。可扩展为ld rd, offset(rs1)

iii. C.LQ：是RV128C独有指令。它从内存加载128位值到寄存器中，将零扩展的偏移量乘16加到rs1寄存器中计算地址。可扩展为lq rd, offset(rs1)

iv. C.FLW：是RV32FC独有指令。它从内存加载单精度浮点数到浮点寄存器中，将零扩展的偏移量乘4加到rs1寄存器中计算地址。可扩展为flw rd, offset(rs1)

v. C.FLD：是RV32DC/RV64DC独有指令。它从内存加载双精度浮点数到浮点寄存器中，将零扩展的偏移量乘8加到rs1寄存器中计算地址。可扩展为fld rd, offset(rs1)

vi. C.SW(已支持)：它从寄存器存储32位数到寄存器中，将零扩展的偏移量乘4加到rs1寄存器中计算地址。可扩展为sw rs2, offset(rs1)

vii. C.SD(已支持)：是RV64C/RV128C独有指令。它从寄存器存储64位数到寄存器中，将零扩展的偏移量乘8加到rs1寄存器中计算地址。可扩展为sd rs2, offset(rs1)

viii. C.SQ：是RV128C独有指令。它从寄存器存储128位数到寄存器中，将零扩展的偏移量乘16加到rs1寄存器中计算地址。可扩展为sq rs2, offset(rs1)

ix. C.FSW：是RV32FC独有指令。它从浮点寄存器存储单精度浮点数到内存中，将零扩展的偏移量乘4加到rs1寄存器中计算地址。可扩展为fsw rs2, offset(rs1)

x. C.FSD：是RV32DC/RV64DC独有指令。它从浮点寄存器存储双精度浮点数到内存中，将零扩展的偏移量乘8加到rs1寄存器中计算地址。可扩展为fsd rs2, offset(rs1)

2. Control Transfer Instructions（控制转移指令）

a. C.J：pc寄存器加上符号扩展的偏移量生成跳转地址，因此可以访问±2KB范围。该指令可扩展为 jal x0, offset

b. C.JAL：为RV32C独有指令。该指令功能同上，另外将下一条指令的地址（pc+2）写入链接寄存器x1中。该指令可扩展为 jal x1, offset

c. C.JR(已支持)：跳转到rs1寄存器中的地址。该指令可扩展为 jalr x0, 0(rs1)

d. C.JALR(已支持)：功能同上条指令，另外将下一条指令的地址（pc+2）写入链接寄存器x1中。该指令可扩展为jalr x1, 0(rs1)

e. C.BEQZ：在rs1寄存器的值为0时进行分支，跳转地址为当前pc加符号扩展的偏移量，跳转范围为±256B。该指令可扩展为beq rs1′, x0, offset

f. C.BNEZ：在rs1寄存器的值非0时进行分支，跳转地址为当前pc加符号扩展的偏移量，跳转范围为±256B。该指令可扩展为bne rs1′, x0, offset

3. Integer Computational Instructions（整数计算指令）

a. Integer Constant-Generation Instructions（整数常量生成指令）

i. C.LI(已支持)：该指令加载6bit立即数到rd寄存器中。该指令可扩展为addi rd, x0, imm

ii. C.LUI(已支持)：该指令将非0的6bit立即数加载到rd寄存器的17-12位，清除底部的12位，将17位符号扩展至所有高位。该指令可扩展为lui rd, imm

b. Integer Register-Immediate Operations（整数 寄存器-立即数操作）

i. C.ADDI(已支持)：将非零的6bit符号扩展立即数和rd寄存器中的值相加，结果存储到rd寄存器中。可扩展为addi rd, rd, imm

ii. C.ADDIW(已支持)：是RV64C/RV128C独有指令，功能同上条指令类似，但会将32bit的结果符号扩展到64bit。可扩展为addiw rd, rd, imm.

iii. C.ADDI16SP(已支持)：该指令将一个6bit的非零符号扩展数增加到栈指针（x2）上，其中立即数以16的倍数进行缩放。可扩展为addi x2, x2, nzimm[9:4]

iv. C.ADDI4SPN(已支持)：该指令将一个非零的零扩展立即数乘4加到栈指针（x2）上，将结果存储到rd寄存器中。该指令用于生成指向栈分配变量的指针，可扩展为addi rd′, x2, nzuimm[9:2]

v. C.SLLI(已支持)：该指令将rd寄存器中的值逻辑左移，将结果写回rd寄存器中，移位量在shamt中。对于RV128C，移位量为0表示移动64位。该指令可扩展为slli rd, rd, shamt[5:0]

vi. C.SRLI(已支持)：该指令将rd寄存器中的值逻辑右移，将结果写回rd寄存器中，移位量在shamt中。对于RV128C，移位量为0表示移动64位。该指令可扩展为srli rd′, rd′, shamt

vii. C.SRAI(已支持)：该指令功能同上条类似，不过进行算数右移。该指令可扩展为srai rd′, rd′, shamt

viii. C.ANDI(已支持)：该指令对符号扩展的6bit立即数和rd寄存器中的值进行按位与操作，将结果写入rd寄存器中。可扩展为andi rd′, rd′, imm

c. Integer Register-Register Operations（整数 寄存器-寄存器操作）

i. C.MV(已支持)：该指令从rs2寄存器中拷贝值到rd寄存器中。可扩展为add rd, x0, rs2

ii. C.ADD(已支持)：该指令将rs2和rd寄存器中的值相加，结果写入rd寄存器中。可扩展为add rd, rd, rs2

iii. C.AND(已支持)：该指令将rs2寄存器和rd寄存器中的值进行按位与操作，并将结果写入rd寄存器中。可扩展为and rd, rd, rs2

iv. C.OR(已支持)：该指令将rs2寄存器和rd寄存器中的值进行按位或操作，并将结果写入rd寄存器中。or rd, rd, rs2

v. C.XOR(已支持)：该指令将rs2寄存器和rd寄存器中的值进行按位异或操作，并将结果写入rd寄存器中。xor rd, rd, rs2

vi. C.SUB(已支持)：该指令将rd寄存器中的值减去rs2寄存器中的值，将结果写入rd寄存器中。可扩展为sub rd′, rd′, rs2′

vii. C.ADDW：该指令为RV64C/RV128C独有指令，它将rd和rs2寄存器中的值相加，将结果低32位进行符号扩展，写入rd寄存器中。可扩展为addw rd′, rd′, rs2′

viii. C.SUBW(已支持)：该指令为RV64C/RV128C独有指令，它将rd寄存器中的值减去rs2寄存器中的值，将结果的低32位进行符号扩展，写入rd寄存器中。可扩展为subw rd′, rd′, rs2

d. NOP instruction：该指令不让处理器执行任何操作，可用于占位符、时间延迟、代码对齐等。可扩展为nop

e. Breakpoint Instruction（断点指令）：该指令将控制权转移回调试环境，可扩展为ebreak

大部分未支持的指令为浮点数指令和128操作数指令，而BPF指令中暂时不支持这两种操作数和指令，因此不需要这些指令。

未支持指令原因分析

综上，可以看到某些指令未得到支持的原因大致可以分为以下几点：

1. 部分BPF指令暂不支持128位操作数，因此没有需要支持128位的RV指令
2. 部分BPF指令暂不支持浮点数，因此没有需要支持浮点运算的RV指令
3. 部分RV指令暂无对应的BPF指令，因此不需要支持
4. 有些RV指令技术上可以支持，但尚未有人开展相关工作