**Unofficial eBPF spec**

The [official documentation for the eBPF instruction set](https://www.kernel.org/doc/Documentation/networking/filter.txt) is in the Linux repository. However, while it is concise, it isn't always easy to use as a reference. This document lists each valid eBPF opcode.

**Instruction encoding**

An eBPF program is a sequence of 64-bit instructions. This project assumes each instruction is encoded in host byte order, but the byte order is not relevant to this spec.

All eBPF instructions have the same basic encoding:

msb lsb

+------------------------+----------------+----+----+--------+

|immediate |offset |src |dst |opcode |

+------------------------+----------------+----+----+--------+

From least significant to most significant bit:

* 8 bit opcode
* 4 bit destination register (dst)
* 4 bit source register (src)
* 16 bit offset
* 32 bit immediate (imm)

Most instructions do not use all of these fields. Unused fields should be zeroed.

The low 3 bits of the opcode field are the "instruction class". This groups together related opcodes.

LD/LDX/ST/STX opcode structure:

msb lsb

+---+--+---+

|mde|sz|cls|

+---+--+---+

The sz field specifies the size of the memory location. The mde field is the memory access mode. uBPF only supports the generic "MEM" access mode.

ALU/ALU64/JMP opcode structure:

msb lsb

+----+-+---+

|op |s|cls|

+----+-+---+

If the s bit is zero, then the source operand is imm. If s is one, then the source operand is src. The op field specifies which ALU or branch operation is to be performed.

**ALU Instructions**

**64-bit**

| **Opcode** | **Mnemonic** | **Pseudocode** |
| --- | --- | --- |
| 0x07 | add dst, imm | dst += imm |
| 0x0f | add dst, src | dst += src |
| 0x17 | sub dst, imm | dst -= imm |
| 0x1f | sub dst, src | dst -= src |
| 0x27 | mul dst, imm | dst \*= imm |
| 0x2f | mul dst, src | dst \*= src |
| 0x37 | div dst, imm | dst /= imm |
| 0x3f | div dst, src | dst /= src |
| 0x47 | or dst, imm | dst |= imm |
| 0x4f | or dst, src | dst |= src |
| 0x57 | and dst, imm | dst &= imm |
| 0x5f | and dst, src | dst &= src |
| 0x67 | lsh dst, imm | dst <<= imm |
| 0x6f | lsh dst, src | dst <<= src |
| 0x77 | rsh dst, imm | dst >>= imm (logical) |
| 0x7f | rsh dst, src | dst >>= src (logical) |
| 0x87 | neg dst | dst = -dst |
| 0x97 | mod dst, imm | dst %= imm |
| 0x9f | mod dst, src | dst %= src |
| 0xa7 | xor dst, imm | dst ^= imm |
| 0xaf | xor dst, src | dst ^= src |
| 0xb7 | mov dst, imm | dst = imm |
| 0xbf | mov dst, src | dst = src |
| 0xc7 | arsh dst, imm | dst >>= imm (arithmetic) |
| 0xcf | arsh dst, src | dst >>= src (arithmetic) |

**32-bit**

These instructions use only the lower 32 bits of their operands and zero the upper 32 bits of the destination register.

| **Opcode** | **Mnemonic** | **Pseudocode** |
| --- | --- | --- |
| 0x04 | add32 dst, imm | dst += imm |
| 0x0c | add32 dst, src | dst += src |
| 0x14 | sub32 dst, imm | dst -= imm |
| 0x1c | sub32 dst, src | dst -= src |
| 0x24 | mul32 dst, imm | dst \*= imm |
| 0x2c | mul32 dst, src | dst \*= src |
| 0x34 | div32 dst, imm | dst /= imm |
| 0x3c | div32 dst, src | dst /= src |
| 0x44 | or32 dst, imm | dst |= imm |
| 0x4c | or32 dst, src | dst |= src |
| 0x54 | and32 dst, imm | dst &= imm |
| 0x5c | and32 dst, src | dst &= src |
| 0x64 | lsh32 dst, imm | dst <<= imm |
| 0x6c | lsh32 dst, src | dst <<= src |
| 0x74 | rsh32 dst, imm | dst >>= imm (logical) |
| 0x7c | rsh32 dst, src | dst >>= src (logical) |
| 0x84 | neg32 dst | dst = -dst |
| 0x94 | mod32 dst, imm | dst %= imm |
| 0x9c | mod32 dst, src | dst %= src |
| 0xa4 | xor32 dst, imm | dst ^= imm |
| 0xac | xor32 dst, src | dst ^= src |
| 0xb4 | mov32 dst, imm | dst = imm |
| 0xbc | mov32 dst, src | dst = src |
| 0xc4 | arsh32 dst, imm | dst >>= imm (arithmetic) |
| 0xcc | arsh32 dst, src | dst >>= src (arithmetic) |

**Byteswap instructions**

| **Opcode** | **Mnemonic** | **Pseudocode** |
| --- | --- | --- |
| 0xd4 (imm == 16) | le16 dst | dst = htole16(dst) |
| 0xd4 (imm == 32) | le32 dst | dst = htole32(dst) |
| 0xd4 (imm == 64) | le64 dst | dst = htole64(dst) |
| 0xdc (imm == 16) | be16 dst | dst = htobe16(dst) |
| 0xdc (imm == 32) | be32 dst | dst = htobe32(dst) |
| 0xdc (imm == 64) | be64 dst | dst = htobe64(dst) |

**Memory Instructions**

| **Opcode** | **Mnemonic** | **Pseudocode** |
| --- | --- | --- |
| 0x18 | lddw dst, imm | dst = imm |
| 0x20 | ldabsw src, dst, imm | See kernel documentation |
| 0x28 | ldabsh src, dst, imm | ... |
| 0x30 | ldabsb src, dst, imm | ... |
| 0x38 | ldabsdw src, dst, imm | ... |
| 0x40 | ldindw src, dst, imm | ... |
| 0x48 | ldindh src, dst, imm | ... |
| 0x50 | ldindb src, dst, imm | ... |
| 0x58 | ldinddw src, dst, imm | ... |
| 0x61 | ldxw dst, [src+off] | dst = \*(uint32\_t \*) (src + off) |
| 0x69 | ldxh dst, [src+off] | dst = \*(uint16\_t \*) (src + off) |
| 0x71 | ldxb dst, [src+off] | dst = \*(uint8\_t \*) (src + off) |
| 0x79 | ldxdw dst, [src+off] | dst = \*(uint64\_t \*) (src + off) |
| 0x62 | stw [dst+off], imm | \*(uint32\_t \*) (dst + off) = imm |
| 0x6a | sth [dst+off], imm | \*(uint16\_t \*) (dst + off) = imm |
| 0x72 | stb [dst+off], imm | \*(uint8\_t \*) (dst + off) = imm |
| 0x7a | stdw [dst+off], imm | \*(uint64\_t \*) (dst + off) = imm |
| 0x63 | stxw [dst+off], src | \*(uint32\_t \*) (dst + off) = src |
| 0x6b | stxh [dst+off], src | \*(uint16\_t \*) (dst + off) = src |
| 0x73 | stxb [dst+off], src | \*(uint8\_t \*) (dst + off) = src |
| 0x7b | stxdw [dst+off], src | \*(uint64\_t \*) (dst + off) = src |

**Branch Instructions**

| **Opcode** | **Mnemonic** | **Pseudocode** |
| --- | --- | --- |
| 0x05 | ja +off | PC += off |
| 0x15 | jeq dst, imm, +off | PC += off if dst == imm |
| 0x1d | jeq dst, src, +off | PC += off if dst == src |
| 0x25 | jgt dst, imm, +off | PC += off if dst > imm |
| 0x2d | jgt dst, src, +off | PC += off if dst > src |
| 0x35 | jge dst, imm, +off | PC += off if dst >= imm |
| 0x3d | jge dst, src, +off | PC += off if dst >= src |
| 0xa5 | jlt dst, imm, +off | PC += off if dst < imm |
| 0xad | jlt dst, src, +off | PC += off if dst < src |
| 0xb5 | jle dst, imm, +off | PC += off if dst <= imm |
| 0xbd | jle dst, src, +off | PC += off if dst <= src |
| 0x45 | jset dst, imm, +off | PC += off if dst & imm |
| 0x4d | jset dst, src, +off | PC += off if dst & src |
| 0x55 | jne dst, imm, +off | PC += off if dst != imm |
| 0x5d | jne dst, src, +off | PC += off if dst != src |
| 0x65 | jsgt dst, imm, +off | PC += off if dst > imm (signed) |
| 0x6d | jsgt dst, src, +off | PC += off if dst > src (signed) |
| 0x75 | jsge dst, imm, +off | PC += off if dst >= imm (signed) |
| 0x7d | jsge dst, src, +off | PC += off if dst >= src (signed) |
| 0xc5 | jslt dst, imm, +off | PC += off if dst < imm (signed) |
| 0xcd | jslt dst, src, +off | PC += off if dst < src (signed) |
| 0xd5 | jsle dst, imm, +off | PC += off if dst <= imm (signed) |
| 0xdd | jsle dst, src, +off | PC += off if dst <= src (signed) |
| 0x85 | call imm | Function call |
| 0x95 | exit | return r0 |