Instruction Set Architecture
draft-ietf-bpf-isa-01

Dave Thaler <dave.thaler.ietf@gmail.com>

History

- Oct 2023: Draft -00 posted
- Nov 2023: IETF 118
- Discussion on list and patches merged to kernel repository
- Mar 3, 2024: Draft -01 posted
- Mar 7, 2024 – Apr 5 (or later): WGLC

Issue tracking

- https://datatracker.ietf.org/doc/draft-ietf-bpf-isa/ links to GitHub Repository https://github.com/ietf-wg-bpf/ebpf-docs as proposed back in BPF BoF
- GitHub repository is currently for any WG drafts, not just ISA
- Issues tagged “ISA” are for ISA document
- 19 “ISA” issues were addressed since IETF 118
- A couple opened (mostly typos) since WGLC started

Editorial nits

- #57: width issues in TXT format of Appendix
- #65: appendix definitions of some lock instructions are incorrect
  - A couple of rows had descriptions that didn't match the main text
- #72: name of "opcode" field
  - A diagram incorrectly said "code"
- #78: "imm32" undefined (should be "imm")
- #92: "BPF ADD" should be "BPF_ADD"
- #93: incorrect use of "src" (being the value of the source operand) vs "src_reg" (being the # of the source register)
  - Some places incorrectly said "src" rather than "src_reg"

Other editorial changes

- #89: Document title changed, for consistency with charter
  - -00: BPF Instruction Set Specification, v1.0
  - -01: BPF Instruction Set Architecture (ISA)

Making legacy instructions more specific

- #80: verify which opcodes had legacy packet instructions
  - -00 deprecated all possible ABS and IND opcodes, including ones never used
- #96: narrow the definition of legacy opcodes
  - -00 deprecated all values of dst_reg, offset, and src_reg for ABS and IND
  - In reality, dst_reg and offset are 0, and src_reg is 0 for ABS

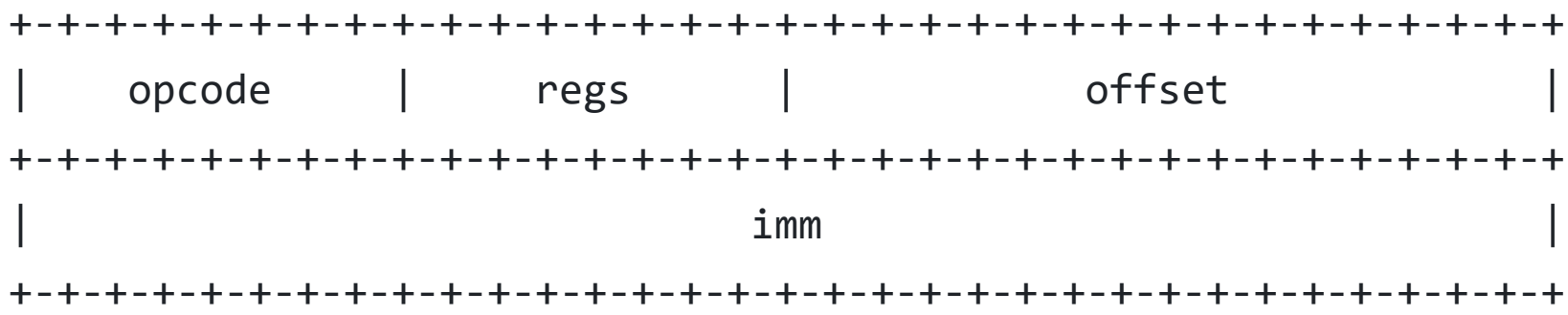
- -01 only defines (and deprecates) ABS and IND combinations that were actually used
  - This leaves the unused space available for future assignment

Technical clarifications

- #69: Incomplete text about MOVSX
  - MOVSX: only source=X (1), meaning register
- #73: Clarify jump instructions
  - EXIT: only source=K (0)
  - CALL: only class=JMP, source=K
  - JA: only source=K (was implied but not explicitly stated)
- #74: Clarify NEG operation
  - NEG: only source=K

#68: Inconsistent definitions of code field

- Aoyang Fang pointed out that some 4-bit fields had 8 bit values, etc.
  - E.g., BPF_SUB = 0x10
- List discussion was then to use IETF packet format in IETF standard
  - To avoid name collision, dropped BPF_ prefix from fields
  - SUB = 0x1 does not conflict with BPF_SUB = 0x10
- Reformatted all field descriptions to use IETF packet format, e.g.:



#75: Clarify 64-bit immediate instructions

- -00 implied all 64-bit immediate instructions use imm64, but only one does
  - 2 use imm and next_imm for different purposes
  - 4 require next_imm = 0 since unused
- -01 moves imm64 construction into the single instruction using it
  - There is no “imm64” definition any more

```
The 64 bits following the basic instruction format contain a pseudo instruction with 'opcode',
'dst_reg', 'src_reg', and 'offset' all set to zero. This is depicted in the following figure:

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    opcode     |     regs      |            offset             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                imm                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            next_imm                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

#50 & #102: Conformance groups

Specification and implementation status

Conformance groups

- Per discussion at IETF 118:
  - Use “logical units of functionality” not clang CPU version numbers
    - Current Linux kernel is RFC compliant
    - *Older* versions of Linux, Windows, etc. are NOT RFC compliant, even for the most basic conformance group
  - Initially each instruction is in one group, but over time can be in multiple groups due to addition/deprecation mechanisms (see later slide)
- Permanent/provisional/historical status is now per group not per instruction
  - Per discussion on issue #79 (should individual instructions have a status)

Defined conformance groups

Name	Description	Includes	status
atomic32	32-bit atomic instructions	-	Permanent
atomic64	64-bit atomic instructions	atomic32	Permanent
base32	32-bit base instructions	-	Permanent
base64	64-bit base instructions	base32	Permanent
divmul32	32-bit division, multiplication, and modulo	-	Permanent
divmul64	64-bit division and modulo	divmul32	Permanent
packet	Legacy packet instructions	-	**Historical**

Conformance groups: implementation status

- https://github.com/Alan-Jowett/bpf_conformance
  - Open source project for ISA conformance tests
  - As of Feb. 2024, now supports conformance groups
  - User specifies what groups to include/exclude during a conformance test run
  - By default uses the set of groups supported by the latest Linux kernel
  - Used for testing at least 5 other open source projects
- https://github.com/vbpf/ebpf-verifier
  - Open source project for BPF verifier (PREVAIL)
  - Used by ebpf-for-windows as well as other runtimes besides the Linux kernel
  - PR in review that adds support for conformance groups
  - Runtime specifies what groups it supports and verifier uses that set
- The above projects were used to help validate the correctness of the table in the appendix that lists instructions and their groups

Summary of bpf_conformance support today

- base32: missing tests for calling a helper function by BTF ID
- base64: missing tests for LDDW with src_reg > 0
- divmul{32,64}: supported
- atomic{32,64}: supported
- packet: not supported, but deprecated

Summary of PREVAIL ISA conformance today

- base32:
  - Issue #451 (Prevail does not support bpf2bpf calls)
  - Issue #590 (Add support for calling a helper function by BTF ID)
- base64:
  - Issue #533 (Add support for LDDW with src reg > 1)
- divmul{32,64}: supported
- atomic{32,64}: supported
- packet: partial, but deprecated

Defined process for adding instructions

In IANA Considerations section:

- A specification may add additional instructions to the BPF Instruction Set registry. Once a conformance group is registered with a set of instructions, **no further instructions can be added to that conformance group**.
- A specification should instead **create a new conformance group** that includes the original conformance group, plus any newly added instructions.
- Inclusion of the original conformance group is done via the "includes" column of the BPF Instruction Conformance Group Registry, and **inclusion of newly added instructions is done via the "groups" column** of the BPF Instruction Set Registry.

Example: add some instructions to “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent

Instructions:

opcode	…	description	groups
aaa	…	Example instruction 1	example
bbb	…	Example instruction 2	example

Example: add some instructions to “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent
examplev2	**Newer set of example instructions**	**example**	**-**	**Permanent**

Instructions:

opcode	…	description	groups
aaa	…	Example instruction 1	example
bbb	…	Example instruction 2	example
ccc	**…**	**Example instruction 3**	**examplev2**
ddd	**…**	**Example instruction 4**	**examplev2**

Defined process for deprecating instructions

- Deprecating instructions that are part of an existing conformance group can be done by
  - defining a new conformance group for the newly deprecated instructions, and
  - defining a new conformance group to supercede the existing conformance group containing the instructions, where
  - the new conformance group *includes* the existing one and *excludes* the deprecated instruction group.

Example: deprecate some instrs in “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent

Instructions:

opcode	…	description	groups
aaa	…	Good example instruction 1	example
bbb	…	Good example instruction 2	example
ccc	…	Bad example instruction 3	example
ddd	…	Bad example instruction 4	example

Example: deprecate some instrs in “example”

Conformance groups:

name	description	includes	excludes	status
example	Example instructions	-	-	Permanent
legacyexample	**Legacy example instructions**	**-**	**-**	**Historical**
examplev2	**Example instructions**	**example**	**legacyexample**	**Permanent**

Instructions:

opcode	…	description	groups
aaa	…	Good example instruction 1	example
bbb	…	Good example instruction 2	example
ccc	…	Bad example instruction 3	example**, legacyexample**
ddd	…	Bad example instruction 4	example**, legacyexample**

WGLC and next steps

Issues raised during WGLC

- #108: atom32 should be atomic32
  - IANA considerations section doesn't have final conformance group names
- #110: opcode 0x87 in appendix should be in base64 conformance group
  - One error in Appendix with IANA table incorrectly has base32 instead of base64 for one instruction

#76: Pointer/address clarification (1/2)

- Call instruction has “call helper function by address” in 32-bit imm
- **Q1**: Since helper function “address” fits in a 32-bit imm, are *all* addresses 32-bit? Is the answer up to the runtime? If not all 32-bit, just helper functions addresses have to fit in in the lower 32-bits?
  - By contrast we have `map_by_idx(imm)` and `var_addr(imm)` to get address from 32-bit map index or variable id, but not `helper_address_by_idx(imm)`
- Example:

```
<entry>:                          ← R6 currently contains ctx pointer
  0: w7 = w6                      ← 32-bit move from R6 to R7
  1: r1 = *(u32 *)(r6 + 0)        ← is this a valid pointer dereference?
```

#76: Pointer/address clarification (2/2)

- 64-bit immediate instruction:

src_reg	pseudocode	imm type	dst type
0x4	dst = code_addr(imm)	Integer	code pointer

```
code_addr(imm) gets the address of the instruction at a specified
relative offset in number of (64-bit) instructions
```

- The term “code pointer” is not defined, nor is anything mentioned that uses code pointers
  - Jump instructions use offsets, not code pointers
  - Call instruction has “call helper function by address” in 32-bit imm
- **Q2**: Possible ambiguity: are “address” and “code pointer” synonymous? (seems so)
  - Should we just say “code address” instead of “code pointer”? Or ok as is?
- Can leave more detailed discussion of use for separate document, e.g.:
  - “[PS] cross-platform helper functions” document, or one discussing `callx` (see callx presentation, #82)