Instruction Set
Architecture
draft-ietf-bpf-isa
Dave Thaler <dthaler@microsoft.com>
Changes per IETF 117 discussion

• Moved ABI-specific text out of ISA and into ABI document
• Added IANA Considerations per discussion at IETF 117:
  • Permanent: Standards action or IESG Review
  • Provisional: Specification required
  • Historical: Specification required
• Adopted ISA document as WG document
Changes based on IETF 117 list discussion

• IETF 117 meeting discussed IANA registry for instructions:
  • Option 1: Single table with multiple key fields
    • \{opcode, src, imm, offset\} tuple where src and/or imm can be wildcards
  • Option 2: Multiple tables
    • BPF opcode table
    • Separate table per opcode with multiple instructions
• Meeting discussion was that they were equivalent and #2 preferred
• Subsequent list discussion pointed out they are NOT equivalent, so left as single table
Option 1: Multiple key fields for BPF instructions

• BPF instructions are identified by (opcode, src, imm, offset) tuple
  • Where src, imm can be wildcards

Examples:

<table>
<thead>
<tr>
<th>opcode</th>
<th>src</th>
<th>imm</th>
<th>offset</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x17</td>
<td>0x0</td>
<td>any</td>
<td>0</td>
<td>dst -= imm</td>
</tr>
<tr>
<td>0x0f</td>
<td>any</td>
<td>0x00</td>
<td>0</td>
<td>dst += src</td>
</tr>
<tr>
<td>0x30</td>
<td>any</td>
<td>0x00</td>
<td>1</td>
<td>dst = (src != 0) ? (dst s/ src) : 0</td>
</tr>
</tbody>
</table>
Option 2: Multiple tables

- BPF opcode table
- Separate table per opcode with multiple instructions

<table>
<thead>
<tr>
<th>opcode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x17</td>
<td>dst -= imm</td>
</tr>
<tr>
<td>0x18</td>
<td>See “64-bit immediate instructions” registry</td>
</tr>
<tr>
<td>0x1f</td>
<td>dst -= src</td>
</tr>
</tbody>
</table>

64-bit immediate instructions:

<table>
<thead>
<tr>
<th>src</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>dst = imm64</td>
</tr>
<tr>
<td>0x1</td>
<td>dst = map_by_fd(imm)</td>
</tr>
<tr>
<td>0x2</td>
<td>dst = mva(map_by_fd(imm)) + next_imm</td>
</tr>
</tbody>
</table>

...
Other changes since IETF 117

• Added explanation of “u32” type convention [Will Hawkins]
• Added glossary definition of “sign extend” [Will Hawkins]
• Corrected definition of BPF_NEG instruction [Jose Marchesi]
• Corrected definition of BPF_CALL instruction [Will Hawkins]
• Use “BPF” (vs “eBPF”) consistently throughout (was previously a mix)
• Fixed a few typos in opcode appendix
• Added new instructions [Yonghong Song] (see later slides)
New instructions

- Signed division
- Signed modulo (using truncated division)
- Move with sign extension
- Load with sign extension
- Unconditional byte swap
- Jump with 32-bit offset (existing instruction only allowed 16-bit)

Implementations:
- LLVM (clang) compiler added these as cpu “v4” instructions
- GCC compiler then began adding support
- Linux kernel verifier / JIT compiler support added
- ... others in progress
“ISA RFC compliance question” thread

• Instruction generators:
  • Compilers (clang, gcc, …)
  • Some applications
  • Test suites

• Instruction parsers:
  • Verifiers
  • JIT compilers
  • Interpreters
  • Disassemblers

• Need some way to express what instructions are supported, to enable interoperability
  • Also potentially enables version/capability negotiation ability
Existing instructions aren’t all “mandatory”

• Optimizations potentially independent of source code:
  • Immediate instructions for maps & variables (opcode 0x18)
  • Signed division & modulo
  • Move & load with sign extension
  • Unconditional byte swap
  • Jump with 32-bit offset

• Support for specific source code constructs:
  • Atomic instructions (opcode 0xdb)
  • Call local (non-inlined) functions
  • Call runtime-exported functions by BTF ID

• Support for above categories varies by generator & parser
• Want to keep existing deployments “compliant”
Some possible units of granularity

• Individual instructions
  • Impractical to have a huge number of combinations

• Clang cpu versions
  • Do they apply to other compilers, e.g., gcc, rust-to-bpf compilers (Aya, RedBPF, rebpf), etc.?
  • Some instructions don’t correlate directly to cpu version, e.g. BPF_ALU:
    • -Xclang -target-feature -Xclang +alu32
  • They don’t correlate to logical units of functionality

• Logical units of functionality
  • But you don’t necessarily need all to “work” per se
  • Doesn’t match historical practice
Gcc & clang compiler options

- cpu=v2
  - jmpext
- cpu=v3
  - jmp32
  - alu32
  - v3-atomics
- cpu=v4
  - bswap
  - sdiv
  - smov
Strawman Proposal

• Create an IANA registry of “conformance group” string labels
  • Possible examples: alu32, cpu=v3, cpu=v4
• Each label corresponds to a set of instructions that are MANDATORY
  • I.e., each instruction has one or more labels that it is part of
• An implementation supports a set of labels {“cpu=v2”, “alu32”}
• Groups can be nested (newer “cpu=v3” includes older “alu32”)
• A specification defines one or more conformance groups
  • Base spec would of course define multiple
  • Any instructions that are like a SHOULD need group(s) different from MUSTs
• Same IANA allocation policies as for instructions
Example IANA tables

Conformance groups:

<table>
<thead>
<tr>
<th>Label</th>
<th>Includes</th>
<th>Excludes</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>legacy</td>
<td>-</td>
<td>-</td>
<td>[RFCxxxx]</td>
</tr>
<tr>
<td>alu32</td>
<td>-</td>
<td>-</td>
<td>[RFCxxxx]</td>
</tr>
<tr>
<td>cpu=v3</td>
<td>cpu=v2, alu32</td>
<td>legacy</td>
<td>[RFCxxxx]</td>
</tr>
</tbody>
</table>

Instructions:

<table>
<thead>
<tr>
<th>opcode</th>
<th>src</th>
<th>imm</th>
<th>offset</th>
<th>description</th>
<th>conf. group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>0x0</td>
<td>any</td>
<td>0</td>
<td>dst = (u32)((u32)dst + (u32)imm)</td>
<td>alu32</td>
</tr>
<tr>
<td>0x20</td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>(deprecated, impl.-specific)</td>
<td>legacy</td>
</tr>
<tr>
<td>0xc3</td>
<td>any</td>
<td>0x00</td>
<td>any</td>
<td>*(u32 *)(dst + offset) += src</td>
<td>cpu=v2</td>
</tr>
<tr>
<td>0xc3</td>
<td>any</td>
<td>0xa0</td>
<td>any</td>
<td>*(u32 *)(dst + offset) ^= src</td>
<td>cpu=v3</td>
</tr>
</tbody>
</table>
Questions

• What conformance group(s) do we use as the spec baseline?
  • cpu=v3? cpu=v4?
  • Should in-market systems be considered compliant or not?

• Should support for some constructs be in own groups? Or conditionally mandatory in same group? E.g.:
  • Call helper function by BTF ID
  • Call helper function by address
  • Get address of runtime platform variable
  • Get address of first value of map
Next steps

• Volunteers for co-editor?
  • Ideally someone with commit privs in Linux kernel repo