BPF LLVM Relocations

This document describes LLVM BPF backend relocation types.

Relocation Record

LLVM BPF backend records each relocation with the following 16-byte ELF structure:

```
typedef struct
{
  Elf64_Addr    r_offset;  // Offset from the beginning of section.
  Elf64_Xword   r_info;    // Relocation type and symbol index.
} Elf64_Rel;
```

For example, for the following code:

```
int g1 __attribute__((section("sec")));
int g2 __attribute__((section("sec")));
static volatile int l1 __attribute__((section("sec")));
static volatile int l2 __attribute__((section("sec")));
int test() {
  return g1 + g2 + l1 + l2;
}
```

Compiled with `clang -target bpf -O2 -c test.c`, the following is the code with `llvm-objdump -dr test.o`:

```
 0:       18 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r1 = 0 ll
          0000000000000000:  R_BPF_64_64  g1
 2:       61 11 00 00 00 00 00 00 r1 = *(u32 *)(r1 + 0)
 3:       18 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r2 = 0 ll
          0000000000000018:  R_BPF_64_64  g2
 5:       61 20 00 00 00 00 00 00 r0 = *(u32 *)(r2 + 0)
 6:       0f 10 00 00 00 00 00 00 r0 += r1
 7:       18 01 00 00 08 00 00 00 00 00 00 00 00 00 00 00 r1 = 8 ll
          0000000000000038:  R_BPF_64_64  sec
 9:       61 11 00 00 00 00 00 00 r1 = *(u32 *)(r1 + 0)
10:       0f 10 00 00 00 00 00 00 r0 += r1
11:       18 01 00 00 0c 00 00 00 00 00 00 00 00 00 00 00 r1 = 12 ll
          0000000000000058:  R_BPF_64_64  sec
13:       61 11 00 00 00 00 00 00 r1 = *(u32 *)(r1 + 0)
14:       0f 10 00 00 00 00 00 00 r0 += r1
15:       95 00 00 00 00 00 00 00 exit
```

There are four relations in the above for four `LD_imm64` instructions. The following `llvm-readelf -r test.o` shows the binary values of the four relocations:

```
Relocation section '.rel.text' at offset 0x190 contains 4 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
0000000000000000  0000000600000001 R_BPF_64_64            0000000000000000 g1
0000000000000018  0000000700000001 R_BPF_64_64            0000000000000004 g2
0000000000000038  0000000400000001 R_BPF_64_64            0000000000000000 sec
0000000000000058  0000000400000001 R_BPF_64_64            0000000000000000 sec
```

Each relocation is represented by `Offset` (8 bytes) and `Info` (8 bytes). For example, the first relocation corresponds to the first instruction (Offset 0x0) and the corresponding `Info` indicates the relocation type of `R_BPF_64_64` (type 1) and the entry in the symbol table (entry 6). The following is the symbol table with `llvm-readelf -s test.o`:

```
Symbol table '.symtab' contains 8 entries:
   Num:    Value          Size Type    Bind   Vis       Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT   UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT   ABS test.c
     2: 0000000000000008     4 OBJECT  LOCAL  DEFAULT     4 l1
     3: 000000000000000c     4 OBJECT  LOCAL  DEFAULT     4 l2
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT     4 sec
     5: 0000000000000000   128 FUNC    GLOBAL DEFAULT     2 test
     6: 0000000000000000     4 OBJECT  GLOBAL DEFAULT     4 g1
     7: 0000000000000004     4 OBJECT  GLOBAL DEFAULT     4 g2
```

The 6th entry is global variable `g1` with value 0.

Similarly, the second relocation is at `.text` offset `0x18`, instruction 3, for global variable `g2` which has a symbol value 4, the offset from the start of `.data` section.

The third and fourth relocations refers to static variables `l1` and `l2`. From `.rel.text` section above, it is not clear which symbols they really refers to as they both refers to symbol table entry 4, symbol `sec`, which has `STT_SECTION` type and represents a section. So for static variable or function, the section offset is written to the original insn buffer, which is called `A` (addend). Looking at above insn `7` and `11`, they have section offset `8` and `12`. From symbol table, we can find that they correspond to entries `2` and `3` for `l1` and `l2`.

In general, the `A` is 0 for global variables and functions, and is the section offset or some computation result based on section offset for static variables/functions. The non-section-offset case refers to function calls. See below for more details.

Different Relocation Types

Six relocation types are supported. The following is an overview and `S` represents the value of the symbol in the symbol table:

```
Enum  ELF Reloc Type     Description      BitSize  Offset        Calculation
0     R_BPF_NONE         None
1     R_BPF_64_64        ld_imm64 insn    32       r_offset + 4  S + A
2     R_BPF_64_ABS64     normal data      64       r_offset      S + A
3     R_BPF_64_ABS32     normal data      32       r_offset      S + A
4     R_BPF_64_NODYLD32  .BTF[.ext] data  32       r_offset      S + A
10    R_BPF_64_32        call insn        32       r_offset + 4  (S + A) / 8 - 1
```

For example, `R_BPF_64_64` relocation type is used for `ld_imm64` instruction. The actual to-be-relocated data (0 or section offset) is stored at `r_offset + 4` and the read/write data bitsize is 32 (4 bytes). The relocation can be resolved with the symbol value plus implicit addend. Note that the `BitSize` is 32 which means the section offset must be less than or equal to `UINT32_MAX` and this is enforced by LLVM BPF backend.

In another case, `R_BPF_64_ABS64` relocation type is used for normal 64-bit data. The actual to-be-relocated data is stored at `r_offset` and the read/write data bitsize is 64 (8 bytes). The relocation can be resolved with the symbol value plus implicit addend.

Both `R_BPF_64_ABS32` and `R_BPF_64_NODYLD32` types are for 32-bit data. But `R_BPF_64_NODYLD32` specifically refers to relocations in `.BTF` and `.BTF.ext` sections. For cases like bcc where llvm `ExecutionEngine RuntimeDyld` is involved, `R_BPF_64_NODYLD32` types of relocations should not be resolved to actual function/variable address. Otherwise, `.BTF` and `.BTF.ext` become unusable by bcc and kernel.

Type `R_BPF_64_32` is used for call instruction. The call target section offset is stored at `r_offset + 4` (32bit) and calculated as `(S + A) / 8 - 1`.

Examples

Types `R_BPF_64_64` and `R_BPF_64_32` are used to resolve `ld_imm64` and `call` instructions. For example:

```
__attribute__((noinline)) __attribute__((section("sec1")))
int gfunc(int a, int b) {
  return a * b;
}
static __attribute__((noinline)) __attribute__((section("sec1")))
int lfunc(int a, int b) {
  return a + b;
}
int global __attribute__((section("sec2")));
int test(int a, int b) {
  return gfunc(a, b) +  lfunc(a, b) + global;
}
```

Compiled with `clang -target bpf -O2 -c test.c`, we will have following code with *llvm-objdump -dr test.o`*:

```
Disassembly of section .text:

0000000000000000 <test>:
       0:       bf 26 00 00 00 00 00 00 r6 = r2
       1:       bf 17 00 00 00 00 00 00 r7 = r1
       2:       85 10 00 00 ff ff ff ff call -1
                0000000000000010:  R_BPF_64_32  gfunc
       3:       bf 08 00 00 00 00 00 00 r8 = r0
       4:       bf 71 00 00 00 00 00 00 r1 = r7
       5:       bf 62 00 00 00 00 00 00 r2 = r6
       6:       85 10 00 00 02 00 00 00 call 2
                0000000000000030:  R_BPF_64_32  sec1
       7:       0f 80 00 00 00 00 00 00 r0 += r8
       8:       18 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r1 = 0 ll
                0000000000000040:  R_BPF_64_64  global
      10:       61 11 00 00 00 00 00 00 r1 = *(u32 *)(r1 + 0)
      11:       0f 10 00 00 00 00 00 00 r0 += r1
      12:       95 00 00 00 00 00 00 00 exit

Disassembly of section sec1:

0000000000000000 <gfunc>:
       0:       bf 20 00 00 00 00 00 00 r0 = r2
       1:       2f 10 00 00 00 00 00 00 r0 *= r1
       2:       95 00 00 00 00 00 00 00 exit
```

```
0000000000000018 <lfunc>:
       3:       bf 20 00 00 00 00 00 00 r0 = r2
       4:       0f 10 00 00 00 00 00 00 r0 += r1
       5:       95 00 00 00 00 00 00 00 exit
```

The first relocation corresponds to `gfunc(a, b)` where `gfunc` has a value of 0, so the `call` instruction offset is `(0 + 0)/8 - 1 = -1`. The second relocation corresponds to `lfunc(a, b)` where `lfunc` has a section offset `0x18`, so the `call` instruction offset is `(0 + 0x18)/8 - 1 = 2`. The third relocation corresponds to ld_imm64 of `global`, which has a section offset `0`.

The following is an example to show how R_BPF_64_ABS64 could be generated:

```
int global() { return 0; }
struct t { void *g; } gbl = { global };
```

Compiled with `clang -target bpf -O2 -g -c test.c`, we will see a relocation below in `.data` section with command `llvm-readelf -r test.o`:

```
Relocation section '.rel.data' at offset 0x458 contains 1 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
0000000000000000  0000000700000002 R_BPF_64_ABS64         0000000000000000 global
```

The relocation says the first 8-byte of `.data` section should be filled with address of `global` variable.

With `llvm-readelf` output, we can see that dwarf sections have a bunch of `R_BPF_64_ABS32` and `R_BPF_64_ABS64` relocations:

```
Relocation section '.rel.debug_info' at offset 0x468 contains 13 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
0000000000000006  0000000300000003 R_BPF_64_ABS32         0000000000000000 .debug_abbrev
000000000000000c  0000000400000003 R_BPF_64_ABS32         0000000000000000 .debug_str
0000000000000012  0000000400000003 R_BPF_64_ABS32         0000000000000000 .debug_str
0000000000000016  0000000600000003 R_BPF_64_ABS32         0000000000000000 .debug_line
000000000000001a  0000000400000003 R_BPF_64_ABS32         0000000000000000 .debug_str
000000000000001e  0000000200000002 R_BPF_64_ABS64         0000000000000000 .text
000000000000002b  0000000400000003 R_BPF_64_ABS32         0000000000000000 .debug_str
0000000000000037  0000000800000002 R_BPF_64_ABS64         0000000000000000 gbl
0000000000000040  0000000400000003 R_BPF_64_ABS32         0000000000000000 .debug_str
......
```

The .BTF/.BTF.ext sections has R_BPF_64_NODYLD32 relocations:

```
Relocation section '.rel.BTF' at offset 0x538 contains 1 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
0000000000000084  0000000800000004 R_BPF_64_NODYLD32      0000000000000000 gbl

Relocation section '.rel.BTF.ext' at offset 0x548 contains 2 entries:
    Offset             Info             Type               Symbol's Value  Symbol's Name
000000000000002c  0000000200000004 R_BPF_64_NODYLD32      0000000000000000 .text
0000000000000040  0000000200000004 R_BPF_64_NODYLD32      0000000000000000 .text
```