The Alpha AXP, part 3: Integer constants

devblogs.microsoft.com/oldnewthing/20170809-00

August 9, 2017

Raymond Chen

The Alpha AXP does not have a "load immediate 32-bit integer" instruction. If you need to load an immediate 32-bit integer, you need to use some tricks.

We saw last time that loading 8-bit constants can be done by using the `ADD` and `SUB` instructions. But there are also instructions that can be repurposed to generate signed 16-bit constants.

Effective address instructions are basically arithmetic operations disguised as memory operations. (Yes, I know we haven't learned about memory operations yet.)

```
LDA     Ra, disp16(Rb)  ; Ra = Rb + (int16_t)disp16
LDAH    Ra, disp16(Rb)  ; Ra = Rb + (int16_t)disp16 * 65536
```

The first instruction applies a signed 16-bit displacement to a value in a register and puts the result in the *Ra* register.

The second one is a little trickier. It takes the signed 16-bit displacement and shifts it left 16 positions before adding it to the *Rb* register.

Both of these operations operate on the full 64-bit register, so they can produce non-canonical results.

The basic idea behind loading a 32-bit constant (in canonical form) is as follows:

1. Use the `LDAH` relative to the *zero* register to load the high-order 48 bits of the 32-bit constant.
2. Use the `LDA` instruction relative to the destination register of the previous instruction to load the low-order 16 bits.

However, the fact that the 16-bit values are sign-extended makes things a bit more complicated.

Let's say that the 32-bit constant we want to load into the *to* register is `0xXXXXYYYY`.

Let `xxxx` be the result you get when you treat `XXXX` as a signed 16-bit value. Similarly, `yyyy` and `YYYY`.

Let `S` be the sign bit of `XXXX`. The canonical form of the constant we want to load is `0xSSSSSSSS`XXXXYYYY``.

If `yyyy` is nonnegative, then we can just load up the two halves of our constant and they won't interact with each other.

```
    LDAH    t0, XXXX(zero)      ; t0 = 0xSSSSSSSS`XXXX0000
    LDA     t0, YYYY(t0)        ; t0 = 0xSSSSSSSS`XXXXYYYY
```

(Throughout, I will leave out the obvious simplifications if `XXXX` or `YYYY` is zero.)

If `yyyy` is negative, then the `LDA` is going to undershoot by `0x10000`, so we compensate by adding one more to `xxxx`.

```
    LDAH    t0, xxxx+1(zero)    ; t0 = 0xSSSSSSSS`XXXX0000 + 0x10000
    LDA     t0, yyyy(t0)        ; t0 = 0xSSSSSSSS`XXXXYYYY
```

Aha, but this trick doesn't work if `xxxx` is exactly `0x7FFF`, because `0x7FFF` + 1 = `0x8000`, which has the wrong sign bit. In that case, we need a final adjustment step to put the result into canonical form.

```
    LDAH    t0, -32768(zero)    ; t0 = 0xFFFFFFFF`80000000
    LDA     t0, yyyy(t0)        ; t0 = 0xFFFFFFFF`7FFFYYYY
    ADDL    zero, t0, t0        ; t0 = 0x00000000`7FFFYYYY
```

Constants that are in the range `0x7FFF8000` to `0x7FFFFFFF` suffer from this problem.[1]

All of this hassle about creating 32-bit constants has consequences for the Windows NT memory manager, as I discussed a few years ago.

Okay, so that's it for loading constants. Next time, we'll start looking at memory access.

[1] There is a special shortcut for the value `0x7FFFFFFF`:

```
    LDA    t0, -1(zero)         ; t0 = 0xFFFFFFFF`FFFFFFFF
    SRL    t0, #33, t0          ; t0 = 0x00000000`7FFFFFFF
```

Raymond Chen

Follow