The AArch64 processor (aka arm64), part 11: Loading addresses

devblogs.microsoft.com/oldnewthing/20220809-00

August 9, 2022

Raymond Chen

Loading the address of a variable can be tricky on AArch64, seeing as you have to somehow get a 64-bit value into a register, and you don't even know the value until the module is loaded and relocated in memory.

The way to do this is with the help of the `ADR` and `ADRP` instructions.

```
; form pc-relative address to nearby label (±1MB)
; Xn = pc + offset
adr     Xn, label

; form pc-relative address to nearby 4KB page (±4GB)
; Xn = (pc & ~4095) + (offset * 4096)
adrp    Xn, label
```

The `ADR` instruction adds a 21-bit signed immediate to the current instruction's address.

The `ADRP` instruction takes a 21-bit signed immediate, shifts it left 12 positions, and then adds it to the address of the starting byte of the page the current instruction is on. The result is the address of the starting byte of a page nearby.

Since modules are unlikely to be bigger than 4GB, the ±4GB reach should be enough to cover accesses to any global variables in the module from any code in that same module.

You can use `ADRP` with offset addressing to access any global variable in two instructions.

```
; first, set x0 to point to the start of the page
; that holds the global
adrp    x0, global

; then use a positive offset to access the global
ldr     w0, [x0, #PageOffset(global)]
```

All of the register offset addressing modes support offsets up to 4095 (most go even higher), so you are sure to be able to get it in two instructions.

If all you need is the address of a global variable, rather that its value, then you can use an `ADD` instruction:

```
    ; first, set x0 to point to the start of the page
    ; that holds the global
    adrp    x0, global

    ; then add the page offset to get the address of the global
    add     x0, x0, #PageOffset(global)
```

Both the unsigned offset in the `[Rn, #imm]` addressing mode as well as the unsigned immediate in the `add` instruction are 12 bits long, which is certainly-not-coincidentally exactly the number of bits in a page offset.

The Microsoft compiler goes to some effort to consolidate address calculations for global variables. For example, if it knows for sure that two global variables are laid out in memory that is a known distance apart, then it will use an `ADRP` to get the address of one, and then use that fixed offset from the first variable to get the second.

```
// global variables
int a, b;

// code generation for calling
f(&a, b);

    ; x0 points to start of page containing variable "a"
    adrp    x0, a

    ; adjust to point directly at "a"
    add     x0, a, #PageOffset(a)

    ; "b" is right next to "a", so load w1 via register offset
    ldr     w1, [x0, #8]

    ; ready to call "f"
    bl      f
```

(We'll learn more about the calling convention later.)

Note that this optimization is not available if `a` and `b` were declared `extern`, since the compiler doesn't know anything about the layout of the memory in that case.

Using *pc*-relative addresses makes it easier to generate position-independent code. I don't know whether Windows requires AArch64 code to be position-independent, but doing so reduces the number of fixups needed, so it's still a good thing even if not required.

Raymond Chen

Follow