The AArch64 processor (aka arm64), part 10: Loading constants

devblogs.microsoft.com/oldnewthing/20220808-00

August 8, 2022

Raymond Chen

Since AArch64 uses fixed-size 32-bit instructions, you have to exercise some creativity to load a 64-bit constant.

```
; move wide with zero
; Rd = imm16 << n
; n can be 0, 16, 32, or 48
movz    Rd, #imm16, LSL #n

; move wide with not
; Rd = ~(imm16 << n)
; n can be 0, 16, 32, or 48
movn    Rd, #imm16, LSL #n

; move wide with keep
; Rd[n+15:n] = imm16
movk    Rd, #imm16, LSL #n
```

The `MOVZ` instruction loads a 16-bit unsigned value into one of the four lanes of a 64-bit destination, or one of the two lanes of a 32-bit destination. All the remaining lanes are set to zero.

The `MOVN` instruction does the same thing as `MOVZ`, except the whole thing is bitwise negated. (Be careful not to confuse `MOVN` with `MVN`.)

The `MOVK` instruction does the same thing as `MOVZ`, except that instead of setting the other lanes to zero, the other lanes are left unchanged.

Loading a 32-bit value can be done in two instructions by using `MOVZ` to load 16 bits into half of the register, than the `MOVK` into the other half.

```
movz    r0, #0x1234             ; r0 = 0x00001234
movk    r0, #0xABCD, LSL #16    ; r0 = 0xABCD1234
```

This technique can be extended to load a 64-bit value in four steps, but that's getting quite unwieldy. The compiler is more likely to store the value in the code segment and use a *pc*-relative addressing mode to load it.

```
; special syntax for pc-relative loads
ldr     x0, =0x123456789ABCDEF0 ; load 64-bit value
ldr     w0, =0x12345678         ; load 32-bit value
```

As I noted in the discussion of addressing modes, the assembler and disassembler use this special equals-sign notation to represent a *pc*-relative load. It means that the value is stored in a *literal pool* in the code segment, and a *pc*-relative load is being used to fetch it. The assembler batches up all of these literals and emits them between functions. The *pc*-relative load has a reach of ±1MB, so you are unlikely to run into the problem that you had on AArch32, where the reach was only ±4KB, and you had to find a safe place to dump the literals in the middle of the function.

There are quite a number of instructions that generate constants, and if you use the `MOV` pseudo-instruction, the assembler will try to find one that works.

```
; load up a constant somehow
mov     Rd, #imm
```

Instruction	Used for
`add Rd, zr, #imm12`	`0x00000000`00000XXX`
`add Rd, zr, #imm12, LSL #12`	`0x00000000`00XXX000`
`sub Wd, wzr, #imm12`	`0x00000000`FFFFFXXX`
`sub Wd, wzr, #imm12, LSL #12`	`0x00000000`FFXXXFFF`
`sub Xd, xzr, #imm12`	`0xFFFFFFFF`FFFFFXXX`
`sub Xd, xzr, #imm12, LSL #12`	`0xFFFFFFFF`FFXXXFFF`
`movz Rd, #imm16`	`0x00000000`0000XXXX`
`movz Rd, #imm16, LSL #16`	`0x00000000`XXXX0000`
`movz Rd, #imm16, LSL #32`	`0x0000XXXX`00000000`

`movz Rd, #imm16, LSL #48`	`0xXXXX0000`00000000
`movn Wd, #imm16`	`0x00000000`FFFFXXXX
`movn Wd, #imm16, LSL #16`	`0x00000000`XXXXFFFF
`movn Xd, #imm16`	`0xFFFFFFFF`FFFFXXXX
`movn Xd, #imm16, LSL #16`	`0xFFFFFFFF`XXXXFFFF
`movn Xd, #imm16, LSL #32`	`0xFFFFXXXX`FFFFFFFF
`movn Xd, #imm16, LSL #48`	`0xXXXXFFFF`FFFFFFFF
`orr Xd, xzr, #imm`	Value can be expressed as a Bitwise operation constant
`orr Wd, wzr, #imm`	Value can be expressed as lower 32 bits of a Bitwise operation constant

A common type of sort-of constant is the address of a global variable. It's a constant whose value isn't discovered until runtime. We'll look at those next time.

Raymond Chen

Follow