The AArch64 processor (aka arm64), part 9: Sign and zero extension

devblogs.microsoft.com/oldnewthing/20220805-00

August 5, 2022

Raymond Chen

AArch64 does not have dedicated sign and zero-extension instructions because they can be synthesized as pseudo-instructions from the bitfield extraction instructions.

```
; unsigned extend byte
; Rd = (uint8_t)Rn
uxtb    Rd/zr, Rn/zr        ; ubfx Rd, Rn, #0, #8

; unsigned extend halfword
; Rd = (uint16_t)Rn
uxth    Rd/zr, Rn/zr        ; ubfx Rd, Rn, #0, #16

; signed extend byte
; Rd = (int8_t)Rn
sxtb    Rd/zr, Rn/zr        ; sbfx Rd, Rn, #0, #8

; signed extend halfword
; Rd = (int16_t)Rn
sxth    Rd/zr, Rn/zr        ; sbfx Rd, Rn, #0, #16

; unsigned extend word
mov     Wd, Wn

; signed extend word
sxtw    Xd/zr, Xn/zr        ; sbfx Xd, Xn, #0, #32
```

The odd man out here is the lack of an unsigned extend word pseudo-instruction, but a `MOV` works just as well, because the rule for 32-bit destinations is that they are zero-extended to a 64-bit value. So just moving a 32-bit register secretly zero-extends it. In practice, you can usually get the zero-extension for free by using a 32-bit register as the destination for the original calculation.

You can avoid having using these instructions if you can merge it into a subsequent extended register operation:

```
; as two instructions, using r3 as scratch register
uxtb    r3, r2              ; r3 = (uint8_t)r2
add     r0, r1, r3          ; r0 = r1 + r3

; merged into one instruction, avoids scratch register
add     r0, r1, r2, uxtb    ; r0 = r1 + (uint8_t)r2
```

For extending a word to a doubleword, you can synthesize that easily enough:

```
; unsigned extend word in Xd to doubleword in Xd/X(d+1)
mov     X(d+1), #0

; signed extend word in Xd to doubleword in Xd/X(d+1)
asr     X(d+1), Xd, #63 ; copy sign bits to all bits
```

Next time, we'll look at ways of loading constants.

Raymond Chen

Follow