The Alpha AXP: Part 12: How you detect carry on a processor with no carry?

devblogs.microsoft.com/oldnewthing/20170822-00

August 22, 2017

Raymond Chen

The Alpha AXP has no corresponding trap variant for arithmetic carry. So how would you detect carry?[1]

Answer: The same way you detect carry in C, or pretty much any other programming language that doesn't support carry.

To detect carry during addition, you check whether the sum is less than either addend. If the sum is less than one addend, then it will also be less than the other addend, so use whichever addend is most convenient.

```
; Rc = Ra + Rb, with Rd receiving carry
; Assumes Rc is not the same as Ra
ADDx    Ra, Rb, Rc      ; Rc = Ra + Rb
CMPULT  Ra, Rc, Rd      ; Rd = carry

; Rc = Ra + Rb, with Rd receiving carry
; Assumes Rc is not the same as Rb
ADDx    Ra, Rb, Rc      ; Rc = Ra + Rb
CMPULT  Rb, Rc, Rd      ; Rd = carry

; Rc = Rc + Rc, with Rd receiving carry
; Assumes Rd is distinct from Rc
BIS     Rd, Rc, Rc      ; Rd = Rc
ADDx    Rc, Rc, Rc      ; Rc = Rc + Rc
CMPULT  Rd, Rc, Rd      ; Rd = carry
```

The last case is where the output overwrites both inputs, so we have to stash one of the inputs in *Rd* so we can compare it to the result afterwards.

To detect borrow during subtraction, you check whether the subtrahend is greater than the minuend.

```
; Rc = Ra - Rb, with Rd receiving borrow
; Assumes Rd is distinct from both inputs
CMPULT  Ra, Rb, Rd      ; Rd = borrow
SUBx    Ra, Rb, Rc      ; Rc = Ra - Rb
```

To detect carry during multiplication, you capture the upper bits of the extended result.

```
; Rc = Ra *U Rb, with Rd receiving carry; 32-bit multiply
ZAPNOT  Ra, #15, Ra     ; zero-extend Ra from 32 to 64 bits
ZAPNOT  Rb, #15, Rb     ; zero-extend Rb from 32 to 64 bits
MULQ    Ra, Rb, Rc      ; Rc = Ra *U Rb (64-bit multiply)
SRA     Rc, #32, Rd     ; Rd = excess to carry forward
ADDL    Rc, zero, Rc    ; Convert Rc to canonical form

; Rc = Ra *U Rb, with Rd receiving carry; 64-bit multiply
; Assumes Rd is distinct from both inputs
UMULH   Ra, Rb, Rd      ; Rd = excess to carry forward
MULQ    Ra, Rb, Rc      ; Rc = Ra *U Rb (64-bit multiply)
```

In the subtraction and multiplication sequences above, you can elide the final instruction if *Rd* is identical to *Rc*. (In other words, if you care only about the carry and not the arithmetic result.)

Exercise: Why did I sometimes calculate *Rd* early and sometimes late?

Exercise 2: Why didn't I have to convert *Rd* to canonical form at the end of the 32-bit multiply?

[1] The Itanium processor also doesn't have a flags register, but nobody seemed to be upset that it didn't provide a way to detect arithmetic carry or overflow.

Raymond Chen

Follow