N.B. A pdf version is also available.

My girlfriend gave me Warren's Hacker's Delight for Christmas. It's really a nice compendium of tricks that are usually available on the web, but strewn across a dozen websites.

I only started reading it this morning, and I figured I'd put some of my notes here instead of leaving them in the margins. The page numbers refer to the ninth printing.

1 2-5: Sign Extension (p. 18)

For sign extension (i.e. replicate the k^{th} bit to the left), Warren suggests (for sign extension of a byte into a word):

1. $((x + \texttt{0x00000080})\ \&\ \texttt{0x000000FF}) - \texttt{0x00000080}$
2. $((x\ \&\ \texttt{0x000000FF}) \oplus \texttt{0x00000080}) - \texttt{0x00000080}$

When one knows that the higher bits of x are all zero, the second variant becomes $(x \oplus \texttt{0x00000080}) - \texttt{0x00000080}$. A similar variant is $x | -(x\ \&\ \texttt{0x00000080})$.

Warren's variant doesn't require any temporary register, but needs a single constant twice. Mine only requires that constant once, but needs a temporary register. On `x86`, with its good support for constant operands, Warren's is probably preferable. With a RISCier ISA, the other version could be useful.

2 2-9: Decoding a "Zero Means `2**n`" Field (p. 20)

The idea here is that we have a field which will never take a value of 0; it could however, take any value from 1 to 2^n. We obviously want to pack this into exactly n bits. A simple encoding would simply map 0 to 1, 1 to 2, etc. For various reasons, we're sometimes stuck with an encoding where everything except 0 maps to itself, and 0 to 2^n.

Notice that $0 \equiv 2^n \mod 2^n$. What we want to do is perform an identity modulo 2^n, but skip the modulo on the final result. Obvious candidates are $x-1+1$, $x+1-1$ and $0 - -x$ (and since we're working modulo 2^n, $-1 \equiv 2^n - 1$ and $0 \equiv 2^n$).

From Warren's list of eight "identities" (for $2^n = 8$), three clearly fall from the above:

1. $((x - 1)\ \&\ 7) + 1$
2. $8 - (-x\ \&\ 7)$
3. $((x + 7)\ \&\ 7) + 1$

Interestingly, those involving $\mid -8$ also do! $x \mid -8$ computes $(x \,\&\, 7) - 8$: it's sending x to a representative from its equivalence class modulo 8, but to the smallest negative value, instead of the smallest positive value. The intuition is that, like masking with 7, all but the three low bits are discarded; however, instead of filling the rest with 0s, like & 7, $\mid -8$ fills them with 1s.

3 Extra! Extra!

This entry is more markup-heavy than usual. That would be because I'm actually typing this in LaTeX, while a Lisp script drives the conversion (via tex4ht) into XHTML for pyblosxom. You can find the script at `http://discontinuity.info/~pkhuong/tex2blosxom.lisp`. It's a hack, but it works!