The GNU C/C++ compiler has a unique feature that allows coding any of these schemes as a macro, giving in-line code for the function references [Stall]. This feature allows statements, including declarations, to be inserted in code where an expression is called for. The sequence of statements would usually end with an expression, which is taken to be the value of the construction. Such a macro definition is shown below, for the first single-precision variation. (In C, it is customary to use uppercase for macro names.)

Click here to view code image

```
#define NLZ(kp) \
   ({union {unsigned _asInt; float _asFloat;}; \
     unsigned _k = (kp), _kk = _k & ~(_k >> 1); \
     _asFloat = (float)_kk + 0.5f; \
     158 - (_asInt >> 23);})
```

The underscores are used to avoid name conflicts with parameter `kp`; presumably, user-defined names do not begin with underscores.

Comparing the Number of Leading Zeros of Two Words

There is a simple way to determine which of two words x and y has the larger number of leading zeros [Knu5] without actually computing nlz(x) or nlz(y). The methods are shown in the equivalences below. The three relations not shown are, of course, obtained by complementing the sense of the comparison on the right.

$$\begin{aligned}
&\text{nlz}(x) = \text{nlz}(y) \quad \text{if and only if} \quad (x \oplus y) \overset{u}{\leq} (x \,\&\, y)\\
&\text{nlz}(x) < \text{nlz}(y) \quad \text{if and only if} \quad (x \,\&\, \neg y) \overset{u}{<} y\\
&\text{nlz}(x) \leq \text{nlz}(y) \quad \text{if and only if} \quad (y \,\&\, \neg x) \overset{u}{\leq} x
\end{aligned}$$

Relation to the Log Function

The "nlz" function is, essentially, the "integer log base 2" function. For unsigned $x \neq 0$,

$$\begin{aligned}
\lfloor \log_2(x) \rfloor &= 31 - \text{nlz}(x), \text{ and}\\
\lceil \log_2(x) \rceil &= 32 - \text{nlz}(x-1).
\end{aligned}$$

See also Section 11–4, "Integer Logarithm," on page 291.

Another closely related function is *bitsize*, the number of bits required to represent its argument as a signed quantity in two's-complement form. We take its definition to be

$$\text{bitsize}(x) = \begin{cases}
1, & x = -1 \text{ or } 0,\\
2, & x = -2 \text{ or } 1,\\
3, & -4 \leq x \leq -3 \text{ or } 2 \leq x \leq 3,\\
4, & -8 \leq x \leq -5 \text{ or } 4 \leq x \leq 7,\\
\ldots & \ldots\\
32, & -2^{31} \leq x \leq -2^{30}+1 \text{ or } 2^{30} \leq x \leq 2^{31}-1.
\end{cases}$$

From this definition, bitsize(x) = bitsize($-x-1$). But $-x - 1 = \neg x$, so an algorithm for bitsize is (where the shift is signed)