CHAPTER 7

Bitwise and Logical Functions

And sprinkled just a bit
Over each banana split.

—Tom Lehrer, "In My Home Town"

"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be: but as it isn't, it ain't. That's logic."

—Lewis Carroll, *Through the Looking-Glass*

IN THIS CHAPTER WE SHALL present functions that carry out bitwise operations on `CLINT` objects, and we shall also introduce functions for determining the equality and size of `CLINT` objects, which we have already used quite a bit.

Among the bitwise functions are to be found the shift operations, which shift a `CLINT` argument in its binary representation by individual bit positions, and certain other functions taking two `CLINT` arguments that enable the direct manipulation of the binary representation of `CLINT` objects. How such operations can be applied to arithmetic purposes is most clearly seen in the shift operations described below, but we have also seen, in Section 4.3, how the bitwise AND operation can be used in reduction modulo a power of two.

7.1 Shift Operations

Necessity devises all manner of shifts.

—Rabelais

The simplest way to multiply a number a with the representation $a = (a_{n-1}a_{n-2}\ldots a_0)_B$ to the base B by a power B^e is to "shift a to the left by e digits." This works with the binary representation exactly as it does in our familiar decimal system:

$$aB^e = (\hat{a}_{n+e-1}\hat{a}_{n+e-2}\ldots\hat{a}_e\hat{a}_{e-1}\ldots\hat{a}_0)_B\,,$$

where

$$\hat{a}_{n+e-1} = a_{n-1}, \quad \hat{a}_{n+e-2} = a_{n-2}, \quad \ldots,$$
$$\hat{a}_e = a_0, \qquad \hat{a}_{e-1} = 0, \qquad \ldots, \quad \hat{a}_0 = 0.$$

For $B = 2$ this corresponds to multiplication of a number in binary representation by 2^e, while for $B = 10$ it corresponds to multiplication by a power of ten in the decimal system.

In the analogous procedure for whole-number division by powers of B the digits of a number are "shifted to the right":

$$\left\lfloor \frac{a}{B^e} \right\rfloor = (\hat{a}_{n-1} \ldots \hat{a}_{n-e} \hat{a}_{n-e-1} \hat{a}_{n-e-2} \ldots \hat{a}_0)_B \,,$$

where

$$\hat{a}_{n-1} = \cdots = \hat{a}_{n-e} = 0, \quad \hat{a}_{n-e-1} = a_{n-1}, \quad \hat{a}_{n-e-2} = a_{n-2}, \ldots, \hat{a}_0 = a_e.$$

For $B = 2$ this corresponds to integer division of a number in binary representation by 2^e, and the analogous result holds for other bases.

Since the digits of `CLINT` objects are represented in memory in binary form, `CLINT` objects can easily be multiplied by powers of two by shifting left, where the next digit to the right is shifted into each place where a digit has been shifted left, and the binary digits left over on the right are filled with zeros.

In an analogous way `CLINT` objects can be divided by powers of two by shifting each binary digit to the right into the next lower-valued digit. Digits left free at the end are either filled with zeros or ignored as leading zeros, and at each stage in the process (shifting by one digit) the lowest-valued digit is lost.

The advantage of this process is clear: Multiplication and division of a `CLINT` object a by a power of two 2^e are simple, and they require at most $e \lceil \log_B a \rceil$ shift operations to shift each `USHORT` value by one binary digit. Multiplication and division of a by a power B^e uses only $\lceil \log_B a \rceil$ operations for storing `USHORT` values.

In the following we shall present three functions. The function `shl_l()` executes a rapid multiplication of a `CLINT` number by 2, while the function `shr_l()` divides a `CLINT` number by 2 and returns the integer quotient.

Lastly, the function `shift_l()` multiplies or divides a `CLINT` type a by a power of two 2^e. Which operation is executed is determined by the sign of the exponent e of the power of two that is passed as argument. If the exponent is positive, then the operation is multiplication, while if it negative, then division is carried out. If e has the representation $e = Bk + l, l < B$, then `shift_l()` carries out the multiplication or division in $(l + 1) \lceil \log_B a \rceil$ operations on `USHORT` values.

All three functions operate modulo $(N_{\max} + 1)$ on objects of `CLINT` type. They are implemented as accumulator functions, and thus they change their `CLINT` operands in that they overwrite the operand with the result of the operation. The functions test for overflow, respectively underflow. However, in shifting, underflow cannot really arise, since in those cases where more positions are to