Residue Number Systems (RNS) provide a means of implementing high-speed integer arithmetic. The core function provides an easily implemented and efficient means for performing the traditionally difficult residue operations of sign detection, magnitude comparison, overflow detection, integer division, and decoding. Besides these, the core function can also be applied to evaluating a very broad class of functions in a very efficient manner. This paper describes a general purpose arithmetic logic unit (ALU) based on core evaluations. Its instruction set can be dynamically altered to fit the algorithm and the application.

Let \( L \) = the number of moduli, \( \{m_1, m_2, \ldots, m_L\} \) = the set of relatively prime odd moduli, \( a! = a! \mod m_i \), \( M = \text{the product of the moduli}, M_i = M/m_i \), and \( B_i = M_i(M_i^{-1} \mod m_i) \mod M \). Addition and multiplication in RNS can be done via the Chinese Remainder Theorem isomorphism \( \mathbb{Z}/M \mathbb{Z} \cong \mathbb{Z}/m_1 \mathbb{Z} \times \cdots \times \mathbb{Z}/m_L \mathbb{Z} \). This allows a residue representation to be converted back to binary (decoded) using the relation

\[
\ll||M = \ll \sum a_i B_i \ll M.
\]

A standard method to do addition and multiplication in \( \mathbb{Z}/m_i \mathbb{Z} \) is a lookup table implemented as a Random Access or Read Only Memory (RAM or ROM) in VLSI.

The core function is important because it can be used to approximately decode a residue value in \( \log_2(L) \) steps, as opposed to other methods which convert back to binary and/or require more steps such as mixed radix conversion. Two important linear and non-linear core functions are defined by

\[
\sum_{i=1}^{L} \left( M_i B_i \right) a_i
\]

The multiple core floating representation (MCFR) is a means of obtaining approximate relative magnitude information for use in algorithms such as sign detection and division. The MCFR maps residues into pairs \((a, e)\) of mantissas and exponents. The sign of a number is given by the sign of \( a \). Division can be computed by an iterative approximation scheme based on the Euclidean algorithm.

Figure 1 gives a block diagram of multiple core floating conversions.

The techniques used in division can be generalized to a large class of iterative algorithms. One major example is Newton iteration. If \( F(z) \) is a polynomial in one or more variables and \( J \) is its Jacobian, then the iterative step

\[
x_{n+1} = x_n - J^{-1}(x_n)F(x_n)
\]

to find a root of \( F \) can be approximated by storing an “approximate inverse Jacobian” in a lookup table. Convergence will be somewhat slower than with the exact value of \( J \), but a careful design can often keep the convergence rate comparable. Thus the advantages of RNS can carry over to some very non-linear calculations. Note that by storing \( J^{-1} \), we have essentially created an additional custom instruction.

Figure 2 shows an overall functional design of a core-based RNS ALU which is capable of executing all the instructions described above, as well as additional customized instructions loaded into lookup tables.