MULTI - INPUT RESIDUE ARITHMETIC UTILIZING

READ - ONLY ASSOCIATIVE MEMORY

Christos A. Papachristou

Computer Engineering and Science Department
Center for Automation and Intelligent System Research
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

In this paper an approach to residue arithmetic is presented using Read-Only-Associative-Memories (ROAMs), such as PLAs. These memories have considerable advantages for table lookup arithmmetic processing over the conventional ROMs in terms of their storage and time efficiency. In residue arithmetic, the ROAM storage required largely depends on the residue recurrencies in arithmetic tables modulo M. After reviewing recent results on the computation of recurrencies, a scheme is proposed for implementing residue arithmetic based on ROAMs. The efficiency of the scheme is established by comparison to conventional ROM-based table lookup techniques. Applications to residue-to-binary number system conversion are also discussed.

1. INTRODUCTION

It is known that associative memory processing is superior to conventional memory processing in such applications as searching, data retrieval, memory mapping and the like [1-3]. However, widespread applications of associative memories in computer systems has been hindered by their cost. Advances in technology may not alleviate this problem unless their production volume is significantly increased.

In addition to the above "macrolevel" applications, associative processing can also be employed at the "microlevel",i.e., in embedded computer system functions implemented in VLSI chips. In our approach, we envision the "microlevel" associative memories as non-volatile in the same sense as ROMs. The fundamental characteristic of both types of associative memories, macro and micro, remains the same: access by content rather than by location. Associative processing at the microlevel can be realized by array locic such as PLAs [4-5], however, other technologies may be more efficient in the future [6].

An important application of ROAMs is in direct implementation of functions by table lookup processsing [7]. This technique is particularly suitable in residue arithmetic tables as the latter are more compact than ordinary arithmetic tables. Moreover, residue arithmetic has the well known advantage over ordinary arithmetic: each digit of the result is derived by processing only the corresponding digits of the operands, hence, eliminating the time consuming carry generation [8-9]. Thus, the residue number system (RNS) can be very attractive when a large number of additions or multiplications is required such as in digital signal processing.

In this paper we investigate the direct implementation of D-operand RNS using ROAMs. The proposed scheme is detailed in Fig. 1 where each of the output bits corresponds to a distinct ROAM module, i.e., the output bits are treated independently due to the matching property of ROAMs. Each ROAM module stores only the truth-table row patterns corresponding to the 1-valued output bits. Associative processing here consists of comparing input patterns to prestored information words in the ROAM modules with the appropriate output bits specifying detected matchings. It was shown in [8-9] that the total ROAM storage area required for addition or multiplication modulo M depends on (a) the residue recurrencies (multiplicities) in the corrresponding tables mod M, and (b) on the weights (number of 1's) of the code words representing the residues mod M. This prompted a linear programming coding approach to optimize the storage [7].

In the following, we first review recent results concerning the evaluation of recurrencies in D-input addition and multiplication mod M. We then consider the implementation of RNS by the proposed scheme. Finally, we discuss the important application of RNS to binary conversion using ROAMs.

2. COMPUTATION OF RESIDUE RECURRENCIES: A REVIEW

2.1 Addition and Multiplication Mod M; M prime

It is known that residue functions such as addition and multiplication mod M may be depicted by multivalued truth tables, the so-called lookup tables [7], with the left-hand columns listing D-tuples of residues and the right-hand column being the corresponding residue function value. The recurrency f(k,D) of residue k with respect to a D-operand residue function is defined to be the number of appearances of k in the output column of the function lookup table. Due to the well known cyclic-shift property of residue addition tables, it has been shown in [9] that the recurrencies in D-operand addition mod M are each equal to MD-1.

With the exception of zero residue values, multiplication table rows, mod M, are also produced by cyclic shifts, as long as M is prime. It has been shown also in [9] that the recurrencies of non-zero residues in D-operand multiplication mod M, M prime, are each equal to $(M-1)^{D-1}$. The recurrency of zero is $M^D-(M-1)^D$.

2.2 Two-Operand Multiplication Mod M, M Non-Prime

It was shown in [8] that the non-zero residue recurrencies in 2-operand multiplication mod M are not equal when M is not a prime. These recurrencies are ultimately related to the factors of M, or M-factors, i.e. any number Q<=M that divides M. There is a recursive relationship among the M-factors in that some are factors of other M-factors, for example 2 is a factor of 4 which is a factor of 8, etc. These recursive relationships may be conveniently depicted by the M-factor graph (Fig. 4 of [8]). Briefly, the nodes of the M-factor graph correspond, one-by-one, to the M-factors and are produced, level-by-level, via division by the prime factors of M. Details are in [8].

Each node (factor) Q of the M-factor graph defines a predecessor, {↑Q;Q}, and successor, {↓Q;Q}, subgraph of Q, respectively. Clearly, these subgraphs consist of all predecessor and succesor nodes of Q, respectively.

Each M-factor of Q also defines a periodic set or period, R(Q), via multiplication mod M by each residue in the residue set S={Ø,1,...,M-1}. It has been shown that the periodic interval T(Q), i.e. the cardinality of R(Q) is T(Q)=M/Q. The periods mod M correspond, one-by-one, to the nodes of the above graph preserving the precedence relationships. This means that if Ri is predecessor of Rj then R(Qi)⊂ R(Qj), and vice versa.

An element x of a period R(Q) is called essential if x is not in any other period predecessor to R(Q). By definition, the set of all essential elements of R(Q) comprise the semiperiod, r(Q), of R(Q). The number of elements in r(Q) is called the semiperiodic interval t(Q). We now summarize the most important results in [8].

1) The semiperiods mod M are pairwise desjoint.

2) the union of all semiperiods over the predecessor subgraph of Q comprises the period P(Q).

3) The semiperiod r(Q) and the semiperiodic interval t(Q) may be computed by the following two recursive formulas, respectively

$$r(Q) = R(Q) - \bigcup r(Qi)$$

$$t(Q) = T(Q) - \sum t(Qi)$$

where Qi runs over {↑Q}, the set of all predecessors of Q, and T(Q)=M/Q.

4) Multiplication mod M of any essential element of a semiperiod r(Q) by every element of the residue set S={Ø,...,M-1} generates the period P(Q) with frequency Q.

5) Every element of a semiperiod r(Q) mod M has the same recurrency f(Q,2) which can be evaluated by the following formula

$$f(Q,2) = \sum Qi\ t(Qi)$$

where Qi runs over {↓Q;Q}, the successor subgraph of Q.

It is clear from the preceding review that the semiperiods mod M constitute recurrency equivalence classes for the elements of the residue set S. In other words, the elements of a semiperiod mod M have each the same recurrency f(Q,2) in 2-operand multiplication mod M.

Computation results concerning the evaluation of recurrencies for a number of moduli values are tabulated in [8].

2.3 D-operand Multiplication Mod M, D>2

The computation of D-operand recurrencies, D>2, for multiplication mod M is based on the construction of D-operand lookup tables. There are two types of such tables for every D>2: (a) the compact type, and (b) the expanded type, lookup table. The compact type, D>2, can be derived from the 2-operand multiplication table, recursively, by means of an isomorphic transformation. In this way, the rows of the 2-operand table are transformed into the generalized rows of the 3,4,..., D-operand compact-type table. The row elements of the compact (D-operand) are themselves rows of the (D-1)-operand compact type table. Apparently, the D-operand compact type multiplication tables mod M, D>2, are each isomorphic to the corresponding 2-operand table. The D-operand expanded table can be constructed iteratively from the corresponding D-operand compact table. The details of these transformations and constructions are in [9]. An example is illustrated in Fig. 2 for D=3, M=4.

The D-operand multiplication recurrency mod M of residue k, f(k,D), can now be defined as the number of appearances of k in the entire D-operand expanded table. On the basis of the previously mentioned isomorphism, it is possible to generalize the 2-operand M-factor graph approach, given in section 2.2, to the multi-operand case. The main result is in the following formula which computes recursively the D-operand recurrencies mod M

$$f(Q,D) = \sum Qi\ t(Qi)\ f(Qi,D-1)$$

where Qi runs over {↓Q;Q}, the successor graph of Q.

The computation of D-operand recurrencies can be simplified by developing a matrix notation for the basic formula above. To do this, let Q∅,Q1,...,Qm denote the nodes of the M-factor graph with Q∅=M and Qm=1. We define the (mxm) semiperiodic matrix Π mod M as follows: for i=∅,...,m and j=∅,...,m

$$\Pi_{ij} = \begin{cases} Qj\ t(Qj) & \text{if } Qi..\{\downarrow Q,Q\} \\ \emptyset & \text{otherwise} \end{cases}$$

We also let $\Phi(D)$ denote a (mx1) column vector such that

$$\Phi_i(D) = f(Qi,D) \quad i=\emptyset,...,m$$

The D-operand recurrencies may now be computed by the following linear recursion

$$\Phi(D) = \Pi * \Phi(D-1)$$

which accepts the following solution

$$\Phi(D) = (\Pi)^{D-1} * \Uparrow$$

where $\Uparrow$ designates the (mx1) column vector with all-unity components.

In conclusion, once the Π matrix mod M is constructed, the computation of the D-operand recurrencies reduces to ordinary matrix multiplication mod M.

3. MULTIOPERAND RNS ARITHMETIC

The preceding designs of multioperand addition and multiplication tables mod M can be applied to associative table lookup processing of the same operations in a residue number system (RNS) which employs a finite set of prime moduli M1,M2,M3,... . In D-operand residue addition (or multiplication), the i-th digit of the sum (or product) equals the sum (or product) of the D, i-th location, operands. This residue operation may be implemented by the multioperand structure of Fig. 3 where each device, as in Fig. 1, consists of ROAM modules that operate in parallel. Note the number of such modules in each device, i.e. the number of output bits of the i-th device, depends of the encoding scheme of the residue set S={∅,1,...,M-1} [7].

The complexity of the proposed RNS scheme should be defined as the totally required ROAM storage in the structure of Fig. 3. Note complexity values concerning addition and multiplication tables mod M have been tabulated in [6-7] for a number of M values and D=2. Due to the inherent parallelism of the residue operations, the complexity measure defined is simply equal to the summation of the corresponding ROAM storages taken over all moduli in the system.

An important consideration regarding the choice of the moduli set in the context of our scheme is established next. Let M be the product of n relative prime moduli, namely M = M1*M2*...*Mn. Let L(Mi,D) denote the complexity of a residue operation mod Mi in terms of the ROAM scheme. Then the following relation is valid for the RNS complexity L(M,D) with respect to both residue addition and multiplication

$$L(M,D) > L(M1,D) + ... + L(Mn,D)$$

This relation can be easily verified on the basis of the results tabulated in [7-8]; the proof is beyond the scope of this paper. The important conclusion, due to this relation is that a fragmentation of the moduli set reduces dramatically the required ROAM storage of our proposed scheme. In other words, it is much more efficient to use many small moduli rather than few large moduli within the same, approximately, dynamic range.

We shall now support quantitatively the main claim of this paper made in the introduction, i.e. the ROAM table lookup processing is more efficient than the one by conventional memories, at least as far as residue arithmetic is concerned. To do this, we shall first estimate the required storage for each memory type, namely ROAM and ROM (or RAM), then we shall compare these estimates. The computation of E(ROAM) and E(ROM), i.e. the ROAM and ROM estimates, respectively, is omitted here as being somewhat involved. The results are as follows

$$E(ROAM) \leq \sum_{1}^{n} M_i^{D-1} \mu_i 2^{\mu_i - 1}$$

$$E(ROM) = \sum_{1}^{n} \mu_i 2^{D\mu_i}$$

where $\mu_i = \lceil \log_2 M_i \rceil$. To show that E(ROM) > E(ROAM) it is sufficient to prove that

$$2^{(D-1)\mu_i} > \tfrac{1}{2} M_i^{D-1} .$$

the latter is true due to the definition of μ_i. We vave actually shown an even stronger case: E(ROAM) < E(ROM)/2.

We have thus established the following very important result: the proposed ROAM based scheme for multioperand RNS arithmetic in a given moduli set is at least twice more storage efficint than table lookup techniques based on conventional memories.

4. RNS TO BINARY CONVERSION

As an application, we suggest a conceptual design of a RNS to binary converter based on our multioperand scheme. The underlying theory is the well-known Chinese Remainder Theorem [10]. Adapting the notation of this reference, let M1,...,Mn be an RNS with M = M1*...*Mn. For an integer P, let P1,...,Pn be its representation in the given RNS where Pi = P mod Mi, i=1,...,n. Also, let Mi' denote the multiplicative inverse mod Mi of M/Mi, i.e. Mi'(M/Mi) mod Mi = 1, i=1,...,n. The conversion formula may now be expressed as follows:

$$(P1\ M1 + \ldots + Pn\ Mn) \bmod M = P \bmod M$$

where the n constants Mi = Mi'(M/Mi), that depend upon the given RNS, constitute the basis of the conversion.

To apply our technique, we note that the above formula resembles an RNS "sum-of-product" expression. This suggests an implementation using n, 2-operand, parallel multipliers, mod M, followed by one, n-operand, adder mod M. However, each 2-operand multiplier has one constant operand Mi and another operand Pi such that 0<=Pi<Mi where, in general, Mi<<M. To exploit this particular situation, we propose a PLA-based implementation of each multiplier with array dimensions: $\lceil \log_2 Mi \rceil$ address bits, $\lceil \log_2 M \rceil$ outputs and (Mi-1) PLA words (P-terms) i=1,...,n. No Mi-type input operands are required by the multipliers. More specifically, there are Mi-1 words embedded in each multiplier of the type (k,kMi) k=1,2,...,Mi-1 where k and kMi represent the search and information fields of each PLA word, respectively. This is a clercut case of associative table lookup processing.

The above proposal leads to the RNS-to-binary converter structure of Fig. 4 which implements example 2-9 of [11]. In this example, the RNS is (M1,M2,M3,M4)=(13,11,7,9), M=9009, P=277, (P1,P2,P3,P4)=(4,2,4,4). There are 4x14 I/O bits in each PLA multiplier and a total of (13-1)+(11-1)+(7-1)+(9-1)= 36 PLA words. The 4-operand ROAM-based adder which has 56x14 I/O bits may be implemented by the multioperand adder configuration of Fig. 1. This requires a total of 14 ROAM modules with 56x1 I/O bits per module. The registers in Fig. 4 are inserted to produce a pipeline version of this structure for recurrent RNS to binary conversions.

As a final note we consider the complexity of the above design. The multiplier complexity is rather small and, hence, these devices can be readily implemented in available technologies. However, the complexity of the multioperand adder is rather significant and its implementation can only be accomplished in VLSI technology. Each ROAM module could probably be implemented by many PLAs and the entire adder could be integrated in a single chip using appropriate VLSI CAD tools.

5. CONCLUSION

Recently, a different approach to residue arithmetic has been investigated via associative, table lookup, techniques [6-7]. The emphasis of this paper is on the application of these results to the RNS design by means of a ROAM based structure which is amenable to VLSI implementation.

Specifically, we first reviewed the work on the computation of recurrencies in multioperand addition and multiplication tables mod M. The complexity of residue operations by the proposed approach is directly related to these recurrencies as well as to the encoding scheme for the residue set. Then we applied this analysis to the implementation of a general RNS, by the proposed scheme, taking into account practical considerations. Concerning the choice of moduli, we showed that a fragmentation of the moduli set reduces substantially the required storage of the ROAM-based scheme. We also demonstrated the superiority of our RNS scheme, in terms of storage efficiency (at least 100% greater), versus conventional ROM lookup techniques. Finally, we discussed an important application: an RNS to binary converter based on the proposed multioperand ROAM scheme.

In conclusion, we have demonstrated the viability of the proposed scheme in performing table lookup processing for RNS arithmetic using Read-Only associative memories, that can now be implemented in VLSI technology [12]. Further, this approach is essentially technology-independent. In fact, future technologies such as electrooptics and the like may be more efficient to implement ROAM modules in terms of time and storage considerations [13].

REFERENCES

1. C.C. Foster, Content-Addressable Parallel Processors, New York: Van Norstrand Reinhold Company, 1976.

2. B. Parhami, "Associative memories and processors: an overview and selected bibliography," Proc. IEEE, Vol. 61, No. 6, pp. 722-730, June 1973.

3. T. Kohonen, Content-Addressable Memories, New York: Springer-Verlag, 1980.

4. O.P. Agrawal and D.R. Laws, "The role of programmable logic in system design," VLSI Design, Vol. V, No. 3, pp. 44-50, March 1984.

5. H. Fleisher and L.I. Maissel, "An introduction to array logic," IBM J. of Research and Development, Vol. 19, pp. 98-109, March 1975.

6. C.C. Guest, M.M. Mirsalehi and T.K. Gaylord, "Residue number system truth-table lookup processing: moduli selection and logical minimization," IEEE Trans. Computers, Vol. C-33, Oct. 1984.

7. C.A. Papachristou, "Direct implementation of discrete and residue-based functions via optimal encoding: a programmable aray logic approach," IEEE Trans. Computers, Vol. C-32, No. 10, pp. 961-968, October 1983.

8. C.A. Papachristou, "Complexity of table lookup processing for multioperand residue addition and multiplication," Research Annals, College of Engineering, University of Cincinnati, Vol. 82, No. ECE 132, Oct. 1982.

9. C.A. Papachristou, "The recurrency classes in multi-operand addition and multiplication modulo M," Proc. 1981 Conf. Information Systems and Science, Baltimore, MD: Johns Hopkins University, pp. 402-407, March 1981.

10. H.L. Garner, "The residue number system," IRE Trans. Electron. Computers, Vol. EC-8, pp. 140-147, June 1959.

11. N.S. Szabo and R.I. Tanaka, Residue Arithmetic and its Applications in Computer Technology, New York: McGraw-Hill, 1967.

12. C. Mead and L. Conway, Introduction to VLSI Systems, Reading, MA: Addison- Wesley, 1980.

13. J.N. Polky, D.D. Miller and R.L. Gutmann, "Optical residue arithmetic data processing," Boeing Aerospace Co., Wright-Patterson AFB, OH, Report AFWAL/AADO-2, November 1981.

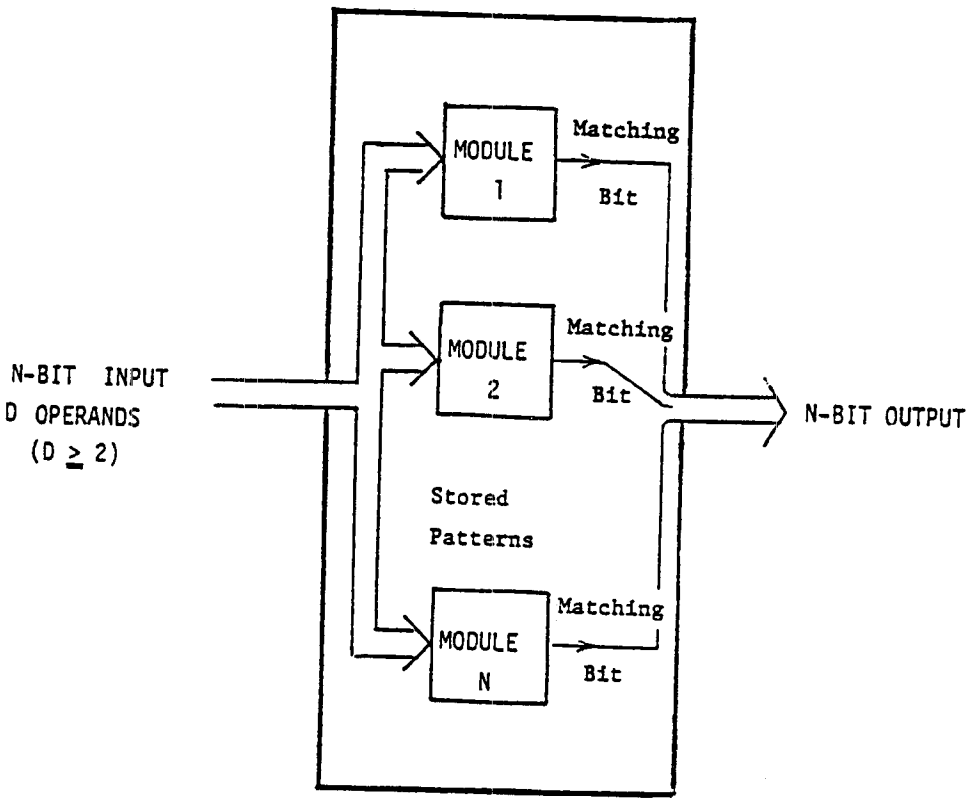

Figure 1 **Table-lookup implementation of an addition or multiplication** table mod M by an array of associative memory modules. Each of the D input operands is N-bit coded for a total of N × D input data bits. There are totally N output matching bits corresponding to the N memory modules.

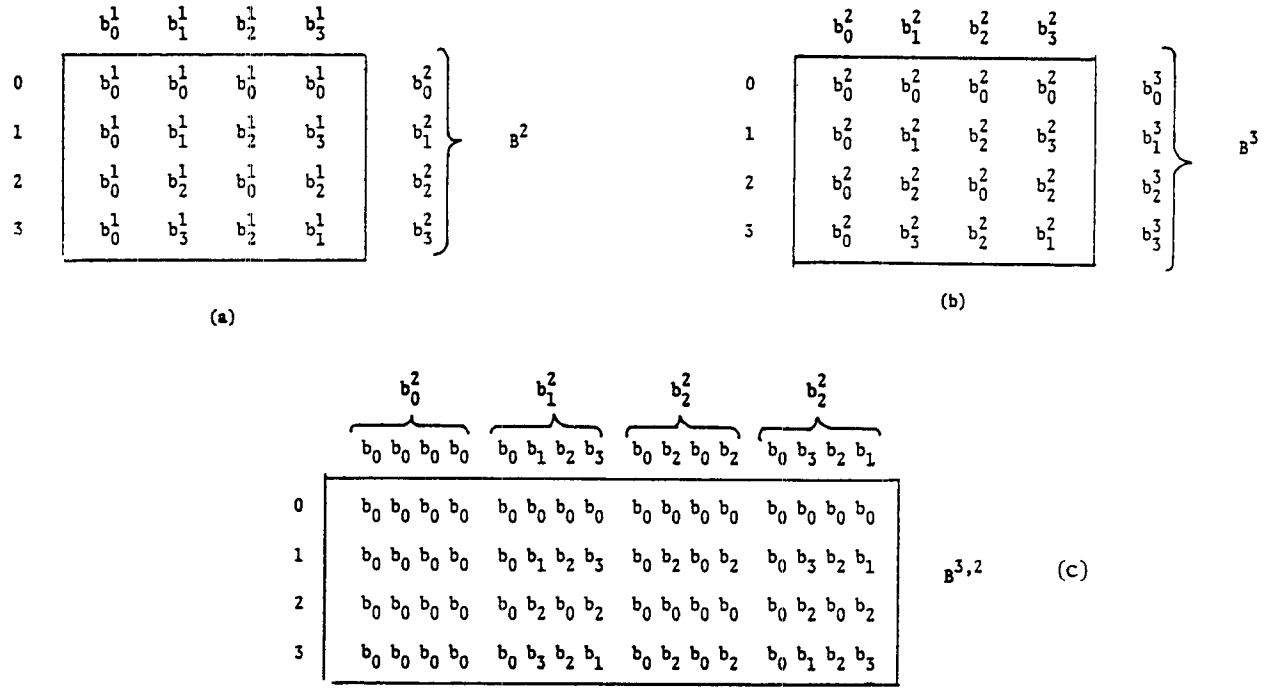

Figure 2: Multiplication tables mod 4

(a) 2-operand table, (b) 3-operand compact-type, (c) 3-operand expanded table note $b_k^1 = k$, k=0,1,2,3; b_k^2 is the k-th row of table (a), b_k^3 is the k-th row of table (b). Superscripts are omitted in table (c) for simplicity.

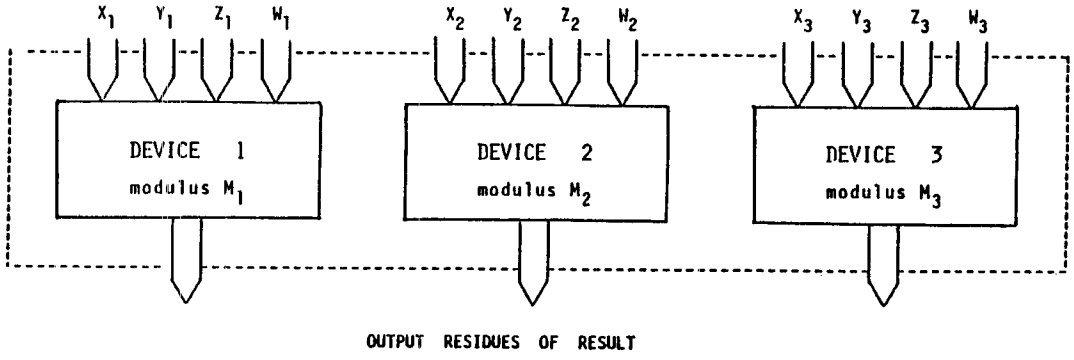

Figure 3 A Read-Only associative-memory processor for D-operand residue arithmetic. In this example the relative prime moduli are M_1, M_2, M_3 and the operands are X,Y,Z,W (D=4), $X_1 = X \bmod M_1, \ldots, W_3 = W \bmod M_3$.

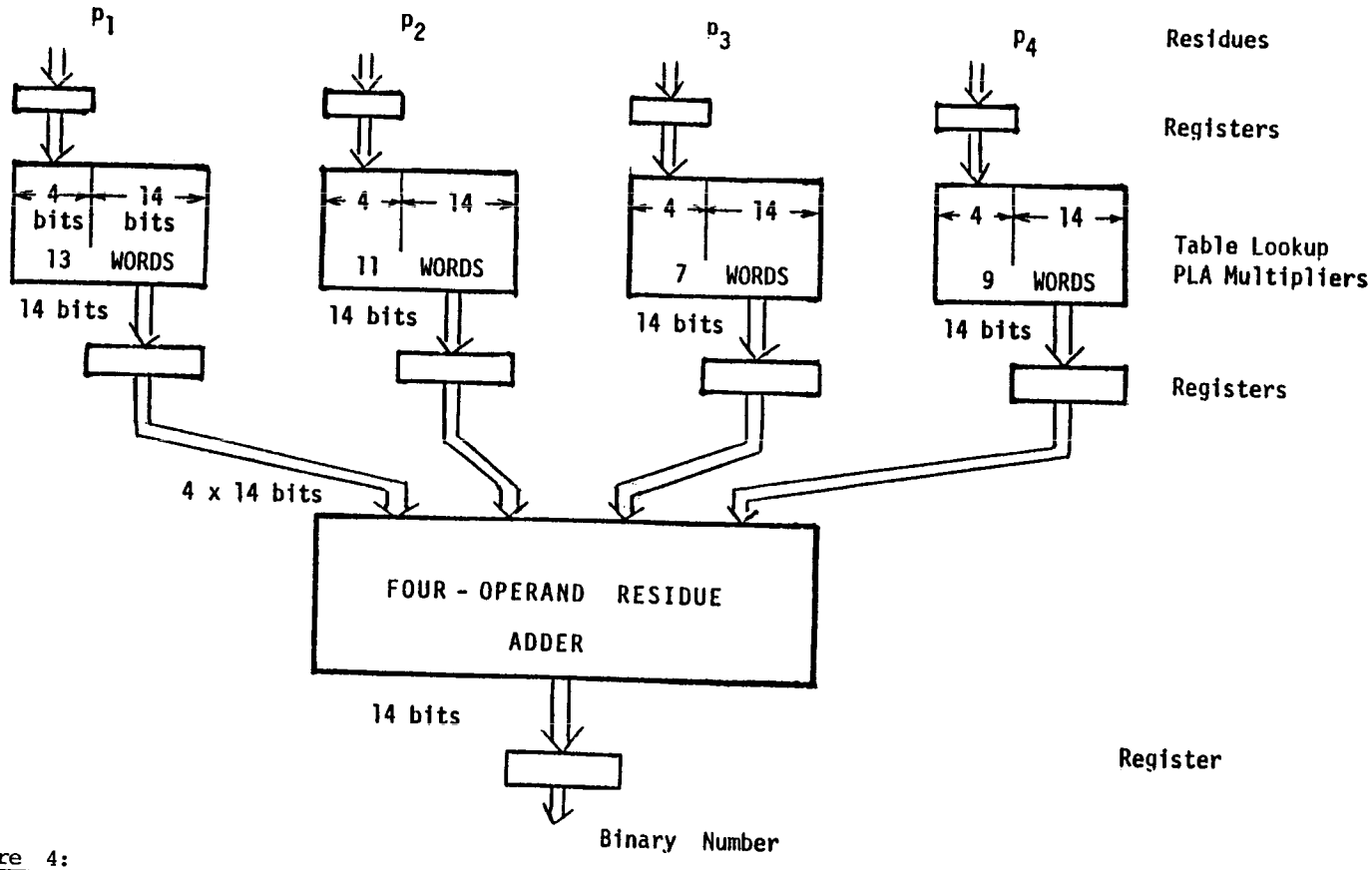

Figure 4:

RNS to binary conversion based on the Chinese remainder theorem. The sheme consists of table lookup PLA multipliers, at the 1st level, and a multioperand adder at the 2d level. The adder may be implemented by ROAMs in VLSI.