"RESIDUE ARITHMETIC IN DIGITAL COMPUTERS"

RAMESH CHANDRA DEBNATH, M.Sc.

BEING A THESIS SUBMITTED

FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

IN THE

DEPARTMENT OF ELECTRICAL ENGINEERING

THE UNIVERSITY OF ADELAIDE

MARCH 1979

I give consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

Date 5/3/79 Signed

SUMMARY

The number systems used have great impact on the speed of a computer, At present at least 4 number systems (binary, negabinary, ternary and residue) are in use either for general purposes or for special purposes. The carry propagation delay which is inherent to all weighted systems (binary, negabinary and ternary) is a limiting factor on the speed of computations. Unlike the weighted system, the residue system is carry free which is the main attractive feature of this system and therefore its potentiality in a general purpose computer, in the light of present day technology, has been investigated.

As the background of the investigation the binary and negabinary arithmetic have been thoroughly studied. A new method for fractional negabinary division, and a correction on negabinary multiplication have been proposed. It has been observed that the binary arithmetic operations are faster than those in negabinary system.

The residue system has got problems too. The sign detection, operations involving sign detection (comparison, overflow detection, division) and the residue-to-decimal conversion are difficult. Moreover, the residue system, being an integer system, makes floating point arithmetic operations very difficult. The details of the existing algorithms/

methods to solve those problems have been thoroughly investigated and drawbacks of those algorithms/methods have been discussed.

This thesis proposes a new method on multiplicative overflow detection in the residue system, a technique for the determination of the first approximation for residue integer division, algorithms for floating point arithmetic operations in the residue system and a method for the residue-to-decimal conversion.

In order to verify those proposed methods and algorithms a 16-15 moduli residue arithmetic unit has been designed using LS1 components. The unit together with (in this case) an SDK-80 microcomputer can perform all the integer and floating point arithmetic operations.

It has been observed that the binary system still maintains its predominance over the residue system in general purpose digital computers. However, the residue system is faster where there is no operation involving sign detection, and also the carry free property, in very large systems, is of great importance in minimising the chip connections and inter chip delays. In future, investigation should be carried out on the possibility of using VLS1 components which may upset the present conclusions which are generally in favour of the binary system.

DECLARATION

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University, and to the best of the author's knowledge and belief contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

R.C. DEBNATH

ACKNOWLEDGEMENT

The author likes to acknowledge the contributions of those who have, directly or indirectly, assisted in this research and preparation of this thesis. The author is pleased to thank Dr D.A. Pucknell, the project supervisor, for his invaluable guidance and encouragement.

The author likes to thank the Radio Research Board of Australia and the Colombo Plan Authority for their financial assistance for this project.

The author would like to thank the technical staff of the Department of Electrical Engineering, University of Adelaide and specially Mr R.C. Nash for his helping hand in the computer laboratory.

The author is pleased to thank Mr D.C. Pawsey, senior lecturer and Mr C.K. Linke, laboratory manager, of the Department of Electrical Engineering, University of Adelaide, for their positive help in making a pleasant stay in Australia.

The author likes to thank Mrs Mercia Fuss for her excellent typing.

Finally, the author wishes to thank his wife, Mrs Asha Rani Debnath, for her encouragement throughout the research.

PAPER WRITTEN BY THE AUTHOR

with D.A. Pucknell, "On multiplicative overflow detection in residue number system". Electronics Letters, Vol.14, No.5, March 1978, pp.129-130.

ERRATA: "On multiplicative overflow detection in residue number system", Electronics Letters, Vol.14, No.12, June 1978, p.385.

NOTATIONS

$[X]$	:	The highest integer value of X, but less than X. (X may be an arithmetic expression also.)
Fl(A±B)	:	Floating point addition/ subtraction with operands A and B.
Fl(A·B)	:	Floating point multiplication with operands A and B.
Fl(A|B)	:	Floating point division with operands A and B.
$\overleftarrow{T(n)}$	:	Shift T to n positions left.
Pol $(\overleftarrow{T(n)})$	:	Polarise $\overleftarrow{T(n)}$.
$X \leftarrow Y$	:	Replace the value of X with Y. (Y may be an arithmetic or a logical expression.)
IN(X, Y,....)	:	Inputs X, Y, ...
SX or $S(X)$	:	Sign of X.
$X \rightleftharpoons Y$	:	Swap the values of X and Y.

$$E(X) = \left[\log_2(X)\right]$$

$$G(X) = 2^{E(X)}$$

$\|X\|_m$ or $[X]_m$	:	Residue of X with respect to module m.
$\|X\|$	:	Absolute value of X.
Gate $(a,b,c,\ldots k,\ell)$	:	Gate with inputs b, c, ...k and output ℓ at the location a on the residue arithmetic unit (RAU) board.
Coeff(A) => a_2, a_1	:	a_2 and a_1 are the co-efficients of A.
Exp(X)	:	Exponent of X.
Mnt(X)	:	Mantissa of X.
$\supset$	:	Contains.

CONTENTS

1. INTRODUCTION

1.1 Introductory Note

The number system has a great impact on the speed of computer arithmetic which, together with other factors - access time of memories, memory hierarchy and computer architecture, influence the speed of a computer as a whole. Scientific problems such as the solution of large sets of linear equations, problems embracing the field of structural design, traffic flow, signal processing etc. are associated with a great deal of computation. A number system with its associated arithmetic algorithms can greatly effect the speed of computation.

The choice of a number system depends on the availability of efficient arithmetic algorithms. Technology also influences the choice a great deal. For example, a decimal system would be preferable to a binary system if technology could provide low cost reliable ten valued logic circuits. One number system can offer advantages on a particular operation, but may be unsuitable from the consideration of overall speed of computation. For example, in a residue system, multiplication is faster, however, other operations such as sign detection, comparison and overflow detection are slow processes. It is, therefore, worthwhile to investigate which of the existing number systems can offer the greatest overall

speed advantage in computer arithmetic for general purpose computers. The rest of this chapter is an overview of the existing number systems and their advantages and disadvantages.

Chapter 2 discusses the arithmetic operations in binary and negabinary systems. The existing arithmetic algorithms in the residue system have been discussed in Chapter 3. A few improved techniques and algorithms for residue arithmetic have been proposed in Chapter 4 and the design of a residue arithmetic computer, on the basis of the proposed techniques and algorithms, has been shown in Chapter 5. Chapter 6 gives conclusions on the comparative studies of binary and residue systems.

1.2 Number Systems - an overview

To represent all numbers, a number system must have unique sets of representation. A number system may be a weighted system or a nonweighted system. Some examples of the weighted systems are the binary system, negabinary system, and ternary system; whereas the residue system is a nonweighted system.

A number in a weighted system can be expressed as

$$x = \sum_{i=o}^{N} a_i w_i$$

where a_i are the sets of integers and w_i are the sets of weights. If w_i are the successive power of radix w, the number system is called a fixed radix system; otherwise it is a mixed radix system. The binary system ($w=2$), negabinary

system ($w=-2$) and ternary system ($w=3$) are in the family of fixed radix systems. Examples of a mixed radix system have been given in Appendix 5.

Number systems which have found use in digital computers are the binary system, negabinary system [1], [2], ternary system [3] and the residue system. In subsequent sections brief discussions now follow on these number systems.

1.2.1 Binary Number System

The binary number system is a two digit (0,1) number system having a radix +2. The decimal value of a binary number depends on the position of 1. Thus for example

$$(11011)_2 = 1.2^4+1.2^3+0.2^2+1.2^1+1.2^0$$
$$= (27)_{10}$$

Due to easy conversion from binary number to octal number ($w=8$) and vice versa, some digital computers such as the IBM-7099, Data General Nova-1, Nova-2 and many others, use the octal system for easy input-output operation. The conversion of binary number to octal number is done by forming groups of three consecutive bits and expressing each group as an octal digit (0 through 7). Thus above binary number can be expressed in octal form as

$$(011,\ 011)_2 = (33)_8 = 3.8^1+3.8^0 = (27)_{10}$$

The octal number is converted to binary number by simply expressing the octal digits in binary form. Thus

$$(75)_8 = (111,101)_2.$$

Presently some computers - IBM-360, IBM-370 and many microprocessors make use of hexadecimal numbers ($w=16$). The sixteen digits of hexadecimal numbers are 0,1,2,3,4,5,6,7,8, 9,A,B,C,D,E,F. A binary number is converted to hexadecimal form by taking groups of four consecutive bits and representing each group by a hexadecimal digit. Thus

$$(1101,\ 1111)_2 = (DF)_{16} = 13.16^1+15.16^0 = (223)_{10}$$

When hexadecimal digits are represented in binary form, it gives equivalent binary number. For example

$$(9A)_{16} = (1001,\ 1010)_2.$$

Another code which is often used is the binary coded decimal number (BCD). This number is really a decimal number, the digits of which are expressed in binary form. Thus

$$(789)_{10} = (0111,\ 1000,\ 1001)_{BCD}$$

Binary states (0,1) are easily realisable in a physical system and is one of the main advantages of binary systems and this is the reason why almost all present day computers use binary number systems. The obvious disadvantage which is inherent in all weighted systems is the carry propagation delay which limits the speed of computation - for example, in addition and especially in multiplication which is a process of additions and shifts. Another disadvantage is the sign correction required in complement (1's or 2's) multiplication. These are the reasons why the potentiality of other number systems should be investigated. The speed of arithmetic is particularly important in the growing field of real time processing in which a wide range of digital computers/microprocessors are being applied.

1.2.2 Negabinary Number System

A negabinary system, like a binary system, is a two digit system. But unlike the binary system the radix is -2. For example

$$(a)\ (11011)_{-2} = 1\cdot(-2)^4+1\cdot(-2)^3+0\cdot(-2)^2+1\cdot(-2)^1+1\cdot(-2)^0$$
$$= 16-8+0-2+1 = (7)_{10}$$

$$(b)\ (01001)_{-2} = 0\cdot(-2)^4+1\cdot(-2)^3+0\cdot(-2)^2+1\cdot(-2)^0$$
$$= 0-8+0+0+1 = (-7)_{10}$$

The advantage of this system is that no sign correction is needed for multiplication. The disadvantages are: (i) during addition double carries are generated (to be discussed in Chapter 2) and therefore it is difficult to maintain speedy propagation of carries, (ii) from examples (a) and (b), it is seen that when the most significant 1 is in an odd position, the number is negative and if it is in an even position, the number is positive. This means that the sign is implicit. The detection of implicit sign is not so easy as that of explicit sign as in the case of the binary system, and (iii) from the example (a) it is clear that to represent $(7)_{10}$, this system needs 5 bits, whereas the binary system needs only 4 bits (3 bits for magnitude + 1 bit for sign). Furthermore, some of the arithmetic operations are not easy. Chapter 2 will discuss its arithmetic operations.

1.2.3 Ternary Number Systems

A ternary integer number can be expressed as

$$x = \sum_{i=o}^{N} a_i 3^i$$

where a_i (i=o, 1, ...,N) are the coefficients and can have any value of the set (0,1,2) in an ordinary ternary system or any value of the set (-1,0,1) in a symmetric system. This number system, specially the symmetric one, offers potential advantages for the following reasons (i) sign conversion is easy (simple inversion) (ii) rounding is done by simply truncating the number to the required number of digits (iii) since the ternary system needs 58% less digits than equivalent binary number, addition and multiplication will be faster [4], [5] and there will be a potential reduction in wiring cost [6].

The Russian 'SETUN' is an example of a symmetric ternary computer. It used magnetic cores [22] to realize three state logic.

A handful of papers is available on arithmetic algorithms [4], [5], [7] and ternary switching functions [8]-[12].

The ternary system sounds promising for the future and interest is rapidly growing in the hardware realization of three valued logic, [6], [13] - [24].

1.2.4 Residue Number System

The residue number system is not a weighted system. Therefore its arithmetic operations are carry free which is the main attractive feature of the system. Since carry does not propagate from one residue to the next one, the speed of addition, subtraction and multiplication is faster than the equivalent binary system. Moreover, in the case of multiplication, the storing of the partial product is not required.

However, sign detection and operations involving sign detection (comparison, overflow detection and division) are not so easy as in the binary system. Furthermore, since residue arithmetic is an integer arithmetic, floating point arithmetic is very difficult. So far as it is known there is only one residue computer developed by RCA [28]. Limited uses of residue numbers are found in error detection and correction algorithms [25] - [27]. The details of residue arithmetic operations will be discussed in Chapter 3.

2. INTRODUCTION TO BINARY AND NEGABINARY NUMBER SYSTEMS

As has been discussed earlier, the binary number system is a two valued (0,1) number system with its radix (base) 2; whilst the negabinary system is a two valued system with radix -2. Since two state physical entities are readily available the binary system has an advantage over all other systems. However, carry propagation delay is the main disadvantage of the binary system; specially in multiplication, which involves a series of additions and speed is limited by the carry propagation delay in the additions. For complement numbers, multiplication needs sign correction for negative operand/operands.

In the negabinary system the sign is implicit therefore multiplication does not need correction for negative operands. However, this system has got a twin carry problem which is even worse than that of the binary system, and also due to the implicit nature of the sign, the sign detection is not so easy as in the binary system.

Later sections give a review of arithmetic in binary and negabinary systems.

2.1 Binary Arithmetic

Any of the standard books [36, 37] can give the background to binary arithmetic. Binary numbers are presented

either in signed 1's complement form, or signed magnitude form or 2's complement form. However, whatever be the form of representation, the sign is always represented by the most significant bit (MSB).

2.1.1 Binary Addition and Subtraction

Tables 2.1, 2.2 and 2.3 show the algorithms of binary addition/subtraction of numbers in 1's complement form, signed magnitude form and 2's complement form respectively.

Table 2.1

Algorithms for binary addition/subtraction in 1's complement form

Method	Operation	Remark
(i) Direct Addition	$Z = X + Y$	Treat sign bit as a number as long as the register capacity for the number is not exceeded.
(ii) Direct Subtraction	$Z = X - Y$	
(iii) Addition by Subtraction	$Z = X - \overline{Y}$	Add (or subtract) end round carry (or borrow), whenever it occurs, to/from least significant bit of sum (or difference).
(iv) Subtraction by Addition	$Z = X + \overline{Y}$	

$\overline{Y}$ is 1's complement of Y.

Example: Addition of $(+9)_{10}$ and $-(6)_{10}$ in 1's complement.

```
                    0,1001
                    1,1001
End Round Carry →1  0 0010
                        1
                    ------
                    0 0011    (sum).
```

Table 2.2

Algorithms for binary addition/subtraction in signed magnitude form

	Signs of X and Y	Operation	Sign of Sum
Addition	Same	$Z^* = X^* + Y^*$	sign (z) = sign (x)
	Different	$Z^* + X^* - Y^*$	sign (z) = sign (x), if it does not overflow
			sign (z) = $\overline{\text{sign (x)}}$, if it overflows
Subtraction	Same	$Z^* = X^* - Y^*$	sign (z) = sign (x), if it does not overflow
			sign (z) = $\overline{\text{sign (x)}}$, if it overflows
	Different	$Z^* = X^* + Y^*$	sign (z) = sign (x)

In Table 2.2 negative number Z^* (after operation) is in 2's complement form. To change negative Z^* to the signed magnitude form the magnitude of $Z^*$is 2's complemented.

Example: Addition of X = +0101 (+5) and Y = -1001 (-9).

```
      0101
      1001
    1 1100
overflow
```

Therefore, the sign of Z^* is 1 [= $\overline{\text{sign (x)}}$] and the sum is 2's complement form of -4. This must now be converted back to signed magnitude form, i.e. to -0100.

Table 2.3

Algorithms for binary addition/subtraction in 2's complement form

Method	Operation	Sign bit
(i) Direct Addition	$Z = X + Y$	Treat sign bit as a number bit as long as register capacity for number does not exceed.
(ii) Direct Subtraction	$Z = X - Y$	
(iii) Addition by subtraction of 2's complement	$Z = X - \bar{Y}$	
(iv) Subtraction by Addition	$Z = X + \bar{Y}$	

$\bar{Y}$ is the 2's complement of Y.

Example: Addition : $+(10)_{10}$ and $-(7)_{10}$

$$\begin{array}{l} (0,1010)_2 \\ \underline{(1,1001)_2} \\ (0,0011)_2 = \quad 3. \end{array}$$

From the study of algorithms in Tables 2.1, 2.2 and 2.3 it is obvious that 2's complement addition/subtraction is easier than in any other form of representation. However, it is more difficult to form than 1's complement representation.

2.1.2 Binary Multiplication

Binary multiplication is generally a repeated addition and shifting process. Multiplication can be performed by using one of the following widely used methods [36].

(i) Direct Multiplication Method

(ii) Burks - Goldstine - Von Neumann Method

(iii) Robertson's First Method

(iv) Robertson's Second Method

(v) Booth's Method

(vi) A Short Cut Multiplication Method.

Binary multiplication with signed magnitude form of number is much easier since it does not need sign correction; the sign of the result is the Exclusive OR function of the signs of the operands. The magnitude of the result is simply the product of two magnitudes (operands). In the case of signed complement numbers, the system needs correction if any one/both of the operands are negative. The Direct Multiplication Method and Short Cut Multiplication Method are most suitable for signed magnitude multiplication. All the other methods are suitable for signed complement multiplication and these algorithms have the property of needing correction for signed complement multiplication.

A variation of the above techniques is a multiple digits multiplication method [36]. In this method the two LSB of the multiplier are examined at a time and a decision of operation/no operation is taken according to the Table 2.4. This technique is suitable for signed magnitude multiplication.

Table 2.4

Table for decision modes in multiple digits multiplication

Multiplier digits	Operation	Shift Right
00	No operation	2
01	Add multiple	2
10	Add 2 · (multiplicand)	2
11	Add 3 · (multiplicand)	2

In quatenary techniques [37] three consecutive bits are examined and decision of operation/no operation is taken according to the Table 2.5. An extra one bit of storage is needed at the LSD position of multiplier which is shifted two bits right after each examination; while the partial product is not shifted.

Table 2.5

Table for decision modes in quatenary multiplication

Multiplier bits	Operation
000	No operation
001	Add 1 · (multiplicand)
010	Add 2 · (multiplicand)
100	Subtract 2 · (multiplicand)
101	Subtract 1 · (multiplicand)
110	Subtract 1 · (multiplicand)
111	No operation

Example: Quatenary Multiplication with

Multiplicand $X = 1100$: Multiplier $Y = 1111$

Initial Partial Product: 00000000

Multiplier Register:	Extra storage		
1111	0	11110100	Subtract X
0111	1	11110100	No operation
0000	1	10110100	Addition of 16X

Simultaneous multiplication [36] is a method of hardware multiplication in which the shifted multiplicand is simultaneously added to the previous partial product. This technique is fast but expensive. Plenty of literature references [38] - [52] are available on this technique.

ROM multiplication [53], [54] is another approach to multiplication.

Integrated circuits for multiplication [55] - [58] are also available. These I.Cs can be cascaded to accommodate numbers of large word length.

2.1.3 Binary Division

Binary division is commonly performed using one of three methods [36] ---- (i) Comparison Method, (ii) Restoring Method (iii) Non-restoring Method. The most commonly used method is the non-restoring method. The algorithm for the non-restoring method has been given in Table 2.6.

Table 2.6

Algorithm for binary division by the non-restoring method

Remainder	$R_n = 2^{-n} r_n$
Partial remainder	$r_n = 2r_{n-1} + (1-2q_n)Y$
	$q_n = 1$ if r_{n-1} and Y are of same sign
	$q_n = 0$ if r_{n-1} and Y are of different sign
Quotient	$Q = (-1+2^{-n}) + \sum_{i=1}^{n} q_i \, 2^{-(i-1)}$
Correction	Add $(-1+2^{-n})$

Array divisions implementing non-restoring division method have been proposed in [59] - [64]. An integrated circuit for binary division [77] is also available.

2.2 The Negabinary Number System

2.2.1 Negabinary Number Representation (Integers)

General expression for a negative radix integer number is

$$a(-\beta) = \sum_{i=o}^{n} a_i \ (-\beta)^i \qquad \ldots\ldots (2.1)$$

where $0 \leq a_i \leq \beta-1$

For negabinary the expression 2.1 becomes

$$a(-2) = \sum_{i=o}^{n} a_i \ (-2)^i$$

where $0 \leq a_i \leq 1$

Thus $(11011)_{-2} = 1x(-2)^4+1x(-2)^3+0x(-2)^2+1x(-2)^1+1x(-2)^0$

$= 16-8+0-2+1 = (7)_{10}$

and $(01111)_{-2} = 0x(-2)^4+1x(-2)^3+1x(-2)^2+1x(-2)^1+1x(-2)^0$

$= -8+4-2+1 = -(5)_{10}$

From above two examples, it is seen that, if the non-zero MSB is in an even position, then the number is positive; and if the non-zero MSB is in an odd position, then the number is negative.

The conversion of integer number of positive base to negative base and vice versa has been shown in [66] - [68].

2.2.2 Negabinary Number Representation (Fraction)

No literature on the representation of fractional negabinary number is available, however, the general expression of fractional number in negabinary system is given by

$$a(-2) = \sum_{i=o}^{n} a_{-i+1} \ (-2)^{-i+1} \qquad \ldots\ldots (2.2)$$

where $a_i \supset (0,1)$

The representation of fractional negabinary number thus needs two extra bits before negabinary point.

Thus

$$(11.0111)_{-2} = 1x(-2)^1+1x(-2)^0+0x(-2)^{-1}+1x(-2)^{-2} +1x(-2)^{-3}+1x(-2)^{-4}$$

$$= -\left(\frac{13}{16}\right)_{10} .$$

2.2.3 Polarisation of Negabinary Number

Similar to the 'complement'of Binary Numbers there exists a 'Polarisation' [70] of Negabinary Numbers. The procedure to polarise a negabinary number can be stated as follows: Starting from the LSB complement immediate next bit following a '1'.

Example: (a) Polarisation of $(5)_{10} = (0101)_{-2}$ is

$$(\underline{1}1\underline{1}1)_{-2} = -(5)_{10}$$

(b) Polarisation of $(-3)_{10} = (1101)_{-2}$ is

$$(\underline{0}1\underline{1}1)_{-2} = (3)_{10}$$

2.2.4 Negabinary Addition/Subtraction

A truth table for negabinary addition, as presented in [65], has been shown in Table 2.7.

Table 2.7

Logic table for negabinary addition

Number X	Number y	Sum, S	Carry C_1	Carry C_0	
0	0	0	0	0	
0	1	1	0	0	
1	0	1	0	0	
1	1	0	1	1	Twin Carry

The twin carry grows to unmanageable numbers as the addition moves towards the MSB position. However, carry-borrow techniques [69] and [72] can minimize the twin carry accumulation. The technique can be summarised as: A carry generated in any bit position is considered to be a borrow to the next bit position; a borrow generated in any bit position is considered to carry to the next bit position. Example:

	$(5)_{10}$	=	0	0	1	0	1
+	$(3)_{10}$	=	0	0	1	1	1
	$(8)_{10}$		1	1	0	0	0

Truth tables of addition and subtraction, according to carry-borrow technique, have been shown in Tables 2.8 and 2.9 respectively.

Table 2.8

Addition - Truth - Table

Numbers		Carry	Borrow	Sum	Generated Carry	Generated Borrow
X	Y	C_i	B_i	S	C_{i+1}	B_{i+1}
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	1	0	1	0	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	1
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	1
1	1	0	0	0	0	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1

Table 2.9

Subtraction - Truth - Table

Numbers		Carry	Borrow	Sum	Generated Carry	Generated Borrow
X	Y	C_i	B_i	S	C_{i+1}	B_{i+1}
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	1
1	1	0	0	0	0	0
1	1	0	1	1	1	0
1	1	1	0	1	0	0

Subtraction can also be performed by the method of addition of the polarised subtrahend [72]. This technique is like subtraction by addition of a complemented subtrahend in binary subtraction.

The addition and subtraction methods discussed so far have been of a 'ripple carry' type. For fast addition, a carry look-ahead adder has been proposed by Agrawal [76].

2.2.5 Negabinary Multiplication

The methods of multiplication in negative base have been discussed in [70] and [74]. However, a variable-bit-shift negabinary multiplication process [73] is an interesting method. In this method two consecutive least significant bits of the multiplier are examined at a time and the decision of arithmetic operation/no operation is taken according to the Table 2.10.

Table 2.10

Table for decision modes in negabinary multiplication

Two bits of multiplier y_{i+1}	y_i	Arithmetic operation required	Number of right shift
0	0	No operation	2
0	1	Add Multiplicand	2
1	0	No operation	1
1	1	Subtract Multiplicand	2

However, this method does not work when two bits of the multiplier are 10. Multiplication of a non-zero multiplicand with the multiplier 10, will produce zero result according to table 2.10; whereas the actual result should be (-2 x multiplicand). Therefore, this method needs correction. A correction has been proposed in section 2.2.7.

2.2.6 Negabinary Division

Elaborate discussions on integer division in the negative base system have been given in [70] and [71].

Agrawal has presented the method of negabinary integer division in [75], and this has been described below.

Let a, $\bar{b}$, and q be the non-zero dividend, non-zero polarised divisor and quotient respectively. The a, $\bar{b}$ and q can be expressed as

$$a = \sum_{i=o}^{m} a_i(-2)^i$$

$$b = \sum_{i=o}^{m} \bar{b}_i(-2)^i$$

$$q = \sum_{o}^{m-n+1} q_i(-2)^i$$

The division is started with a leading zero added to a and the most significant bits (MSBs) of $\bar{b}$ aligned to a. The quotient bit is evaluated after shifting the polarised divisor one bit to the right. The quotient is taken to be 1 if the addition of the polarised divisor is possible. Generally any one of the following conditions exists after shifting the polarised-divisor one bit right when the i^{th} quotient is being evaluated.

(i) $\quad o \quad a_{i+n-1} \quad \ldots\ a_i \quad \ldots\ a_0$

$\quad\quad \bar{b}_n \quad \bar{b}_{n-1} \quad \ldots\ \bar{b}_0$

(ii) $\quad a_{i+n} \quad a_{i+n-1} \quad a_{i+n-2} \quad \ldots\ a_i \ \ldots\ a_0$

$\quad\quad \bar{b}_n \quad \bar{b}_{n-1} \quad \bar{b}_{n-2} \quad \ldots\ \bar{b}_0$

(iii) $\quad a_{i+n+1} \quad a_{i+n} \quad a_{i+n-1} \quad a_{i+n-2} \quad \ldots\ a_0$

$\quad\quad\quad\quad \bar{b}_n \quad \bar{b}_{n-1} \quad \bar{b}_{n-2} \quad \ldots\ \bar{b}_0$

If condition (iii) exists, the polarised addition is always possible. If condition (ii) exists, the polarised addition is never possible. If condition (i) exists, polarised addition is possible when the magnitude of P_{cri} is equal to or

smaller than the dividend (or the partial remainder). P_{cri} is given by the multiplication of $\bar{b}$ by $\bar{q}_{cri}$, where q_{cri} is given as:

Bit No	i	i-1	i-2	i-3	2	1	0
Value of q_{cri}	o	1	1	1	1	1	1

the rightmost bit is 1 when i is even; otherwise it is zero.

$$P_{cri} = \bar{b} \cdot \bar{q}_{cri}$$

However, the method described above is not applicable to negabinary fractional division. Therefore, one method for negabinary fractional division has been proposed in section 2.2.8.

2.2.7 A correction proposed on variable shift negabinary multiplication

Except for one correction, the method is the same as that discussed in section 2.2.5.

Table 2.11

Algorithm for negabinary multiplication

Two bits of multiplier y_{i+1}	y_i	Operation	Right Shift
0	0	No operation	2
0	1	Add Multiplicand	2
1	0	Add (-2·Multiplicand)	2
1	1	Subtract Multiplicand	2

[-2·(multiplicand) is obtained by shifting multiplicand one position left.]

The correction is only for $y_{i+1}\ y_i = 10$.

Example: Multiplication of $(3)_{10}$ by $-(10)_{10}$.

$$(3)_{10} = (0111)_{-2}$$
$$(-10)_{10} = (1010)_{-2}$$
$$01110$$
$$1110$$
(Negabinary Addition) $(100110)_{-2} = (-30)_{10}$

According to the proposed correction, the multiplier is a 2-shift multiplier and algorithm is same as that of binary multiple digits multiplication (Table 2.4).

2.2.8 Method Proposed on Negabinary Fraction Division

The proposed method assumes that both divisor and dividend are fractional and divisor is greater than dividend (i.e. $X<Y$). The quotient Q can be written as (from negabinary representation of fractional number section 2.2.2).

$$Q = a_1(-2)^1+a_0(-2)^0+a_{-1}(-2)^{-1}+a_{-2}(-2)^{-2}+ \ldots. +a_{-n}(-2)^{-n}$$

$$= -a_1(2)^1+a_0(2)^0+\sum_{i=1}^{n} a_{-i}(-2)^{-i}$$

Let $-a_1(2)^1+a_0(2)^0$ be represented by two bits q_1q_0 Therefore

$$Q = q_1q_0+\sum_{i=1}^{n} a_{-i}(-2)^{-i}$$

$$= q_1q_0+\sum_{i=1}^{n}(-1)^{-i}\,a_{-i}(2)^{-i}$$

If $a_{-i} = 1$, then

(i) $(-1)^{-i}\,a_{-i} = -1$ if i is an odd number.

(ii) $(-1)^{-i}\,a_{-i} = 1$ if i is an even number.

If $a_{-i} = o$, then

(iii) $(-1)^{-i}\,a_{-i} = o$

In the negabinary system -1 is represented by two bits, 1 1. Therefore, at least two bits are needed to represent the set (0,1,-1) in negabinary system.

In other words Q can be written as

$$Q = q_1 q_0 + \sum_{i=1}^{n} [a_1(-2)^1 + a_0(-2)^0] \cdot 2^{-i}$$

$$= q_1 q_0 + \sum_{i=1}^{n} (-a_1 2^{-i+1} + a_0 2^{-i})$$

$-a_i 2^{-i+1} + a_0 2^{-i}$ can be denoted by two bits

q_{-i+1} q_{-i}

Then $Q = q_1 q_0 + \sum_{i=1}^{n} q_{-i+1} q_{-i}$

(The negabinary point lies just after q_0 bit)

Fig 2.1 shows the flow chart of negabinary fractional division process, and it is similar to the non-restoring fractional division in the binary system.

Example: Divide $-\left(\frac{15}{64}\right)_{10}$ by $\left(\frac{3}{8}\right)_{10}$

$$X = -\left(\frac{15}{64}\right)_{10} = (00.110001)_{-2}$$

$$Y = \left(\frac{3}{8}\right)_{10} = (01.101)_{-2}$$

00.11001
01.101 Addition, since sign does not agree
00.011001 sign agrees with the sign of y. Therefore
$q_1 q_0 = 1\ 1$
left shift
00.11001
polarisation
00.01011
01.101 subtraction
00.00111 sign disagrees with that of y. Therefore
$q_0 q_{-1} = 00$
left shift
00.0111
polarisation
00.1101
01.101 addition
00.0111 sign agrees with that of y. Therefore
$q_{-1} q_{-2} = 01$
left shift
00.111
polarisation
01.101
01.101 subtraction
00.000 (for zero remainder $q_{-2} q_{-3} = 1\ 1$, since i=3 is an odd number)

Therefore $Q = q_1 q_0 + q_0 q_{-1} + q_{-1} q_{-2} + q_{-2} q_{-3}$

$= 11.$
$0\ 0$
01
11
$00.101 = -\frac{5}{8}.$

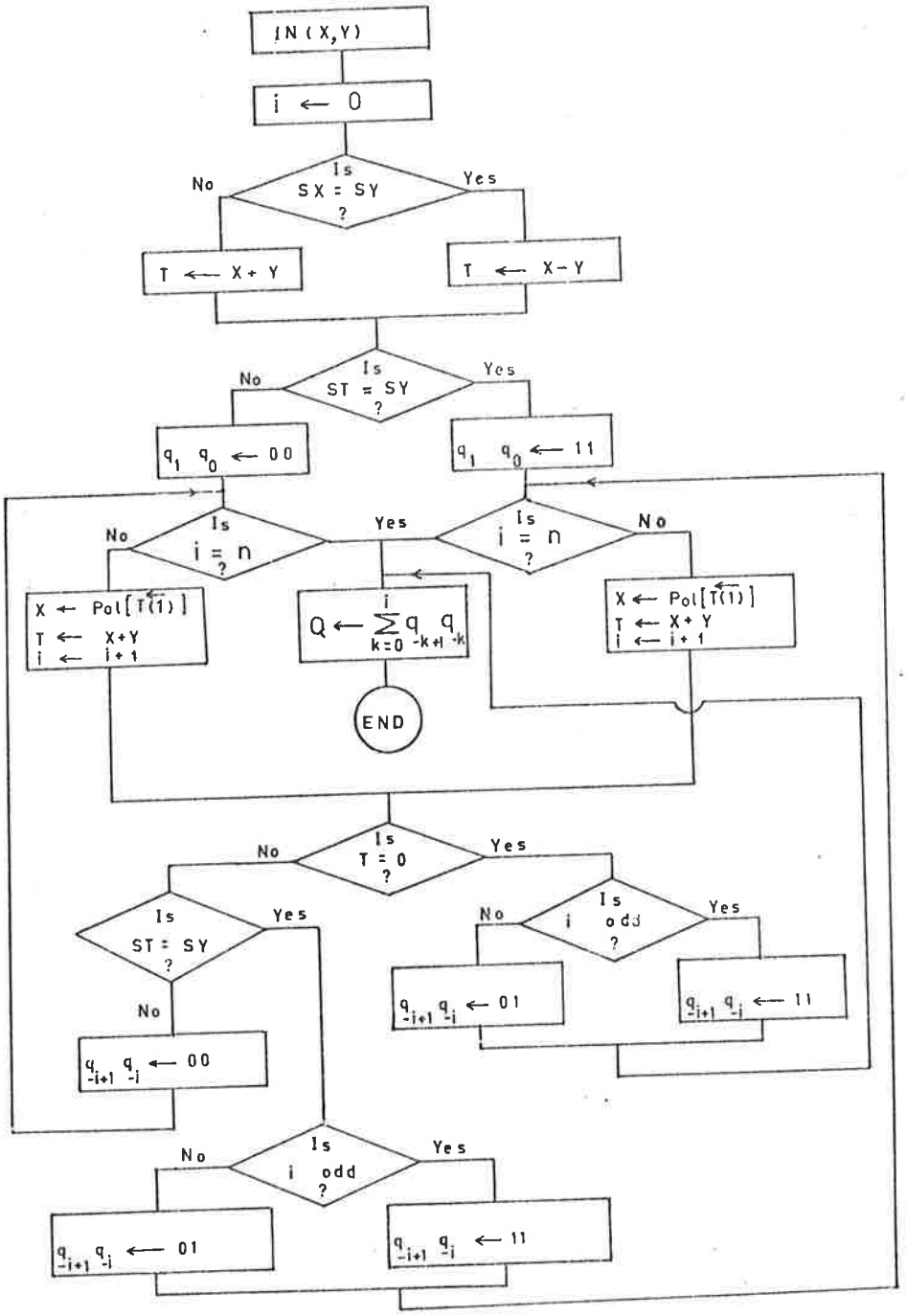


Fig.2.1 Flow chart of negabinary fractional division proccess.

2.3 Conclusion

The details of binary and negabinary arithmetic methods have been discussed in the preceding sections. It is clear from this study that all the methods of binary arithmetic can, with slight modification, be used in negabinary arithmetic. However, the twin carry propagation is the main disadvantage of the negabinary system. The carry-borrow technique of addition/subtraction is serial in nature; therefore, it is inherently slow. The negabinary carry look ahead adder is complex and has no speed advantage over a binary adder. Polarisation, similar to complement in the binary system, is a serial type of operation; therefore, it is a slow process. Negabinary multiplication does not need correction, but additions of partial products (PP) are slower, since negabinary addition is slower than binary addition. The proposed negabinary fractional division method is similar to the non-restoring binary fractional division, however, since polarisation and negabinary addition are slow in comparison with the complement and binary addition, the process of negabinary division is slower than that in the binary system. Above all it can be concluded that the negabinary system has no speed advantage over a binary system. Therefore, other number systems which are carry free are well worth investigating.

3. RESIDUE ARITHMETIC - - LITERATURE REVIEW

3.1 Int oduction

Due to the carry free property of residue arithmetic, addition, subtraction and multiplication are all faster in the residue system than in the binary system. However, sign detection, operations involving sign detection (comparison, overflow detection) and division are slow processes. Also since residue arithmetic is an integer arithmetic, residue floating point arithmetic is very difficult. Similarly output translation (residue to decimal) which, although less frequently used, is still a problem. However, due to the carry free properties, residue arithmetic systems are well worth investigating as an alternative to the conventional (binary) system. A comprehensive knowledge of residue arithmetic can be gained from [28], [29].

To be of use in any general purpose computer any number system needs efficient algorithms on sign detection, overflow detection, division, floating point arithmetic and output translation. Later work in this thesis investigates suitable improvement in some of these areas for the residue system.

The remaining sections of this chapter discuss residue addition, subtraction, multiplication, existing algorithms on division, overflow detection, floating point arithmetic and output translation.

3.2 Integer Residue Arithmetic - an outline

3.2.1 Addition, Subtraction and Multiplication

If $(p_{1x}, p_{2x}, p_{3x}, \ldots\ldots, p_{nx})$ and

$$(p_{1y}, p_{2y}, p_{3y}, \ldots\ldots, p_{ny})$$

are the residues of the operands X and Y respectively in the residue system of moduli m_i (i=1,2,3, ..., n), then addition/subtraction and multiplication can be expressed as

(i) Addition/subtraction:

$$|X \pm Y|_M = |p_{1x} \pm p_{1y}|_{m_1}, \quad |p_{2x} \pm p_{2y}|_{m_2},$$
$$|p_{3x} \pm p_{3y}|_{m_3}, \ldots\ldots, |p_{nx} \pm p_{ny}|_{m_n}$$

where $M = \pi_{i=1}^{n} m_i$

(ii) Multiplication:

$$|X \cdot Y|_M = |p_{1x} \cdot p_{1y}|_{m_1}, \quad |p_{2x} \cdot p_{2y}|_{m_2},$$
$$|p_{3x} \cdot p_{3y}|_{m_3}, \ldots\ldots, |p_{nx} \cdot p_{ny}|_{m_n}$$

Combinational logic or look up tables in suitable memories can be used to implement addition-subtraction and multiplication processes. The modulus which needs the most time in operation determines the speed of each overall operation.

3.2.2 Division

3.2.2a The types of division

There are three types of division [28] each of which must be considered in contemplating possible implementations.

Category 1: Division Remainder Zero : In this category of division, the dividend is a multiple of divisor.

Category 2: Scaling : Division by factor/factors of M.

Category 3: General Division : Division by any arbitrary numbers.

General Division is of particular interest, since in a general purpose computer both divisor and dividend are arbitrary numbers.

3.2.2b Methods of General Division

There are at least two methods for accomplishing this in a residue system - General Division I [28], [30] and General Division II [28].

General Division II process is slower than General Division I and is restricted by a special requirement. General Division I needs a look up table which contains the numbers 2^i (i=0,1,2, ..., N) in residue form. (N is the highest power to 2 contained in $(\frac{M}{2} - 1)$). The look up table is formed in memories, where the address represents the exponent (power) to 2 and the contents give the value of 2^i in residue form. This method assumes that the dividend X and the divisor Y are positive and nonzero. The details of the procedures are given below.

(i) Find $G(Y)=2^k$. (Starting with the highest address number (N), the contents of each location of the table is compared with Y, until $2^k \leq Y < 2^{k+1}$).

(ii) Find the first approximation, $F = \frac{2^N}{G(Y)}$. (Since 2^N and $G(Y)$ are the numbers of powers of 2 and $G(Y) \leq 2^N$, F can be found by division of Category 1.)

(iii) Follow the iterative process shown with the flow chart of Fig. 3.1.

To determine $G(Y)$ by logarithmic search technique [30], it requires on the average

$$\frac{n}{2} \log_2 c \quad \text{modular operations [30]} \quad \ldots\ldots \quad (3.1)$$

(Where n is the number of moduli; $c \simeq \log_2 (M/2-1)$)

After $G(Y)$ has been obtained, to find F it requires (on the average) additional

$$(n+2) \quad \text{modular operations} \quad \ldots\ldots \quad (3.2)$$

Iterative process needs on the average

$$\frac{c}{2} (n+1) \quad \text{modular operations} \quad \ldots\ldots \quad (3.3)$$

Therefore total division process requires, on the average

$$\frac{n}{2} \log_2 c + n + 2 + \frac{c}{2} (n+1)$$

$$\text{modular operations} \quad \ldots\ldots \quad (3.4)$$

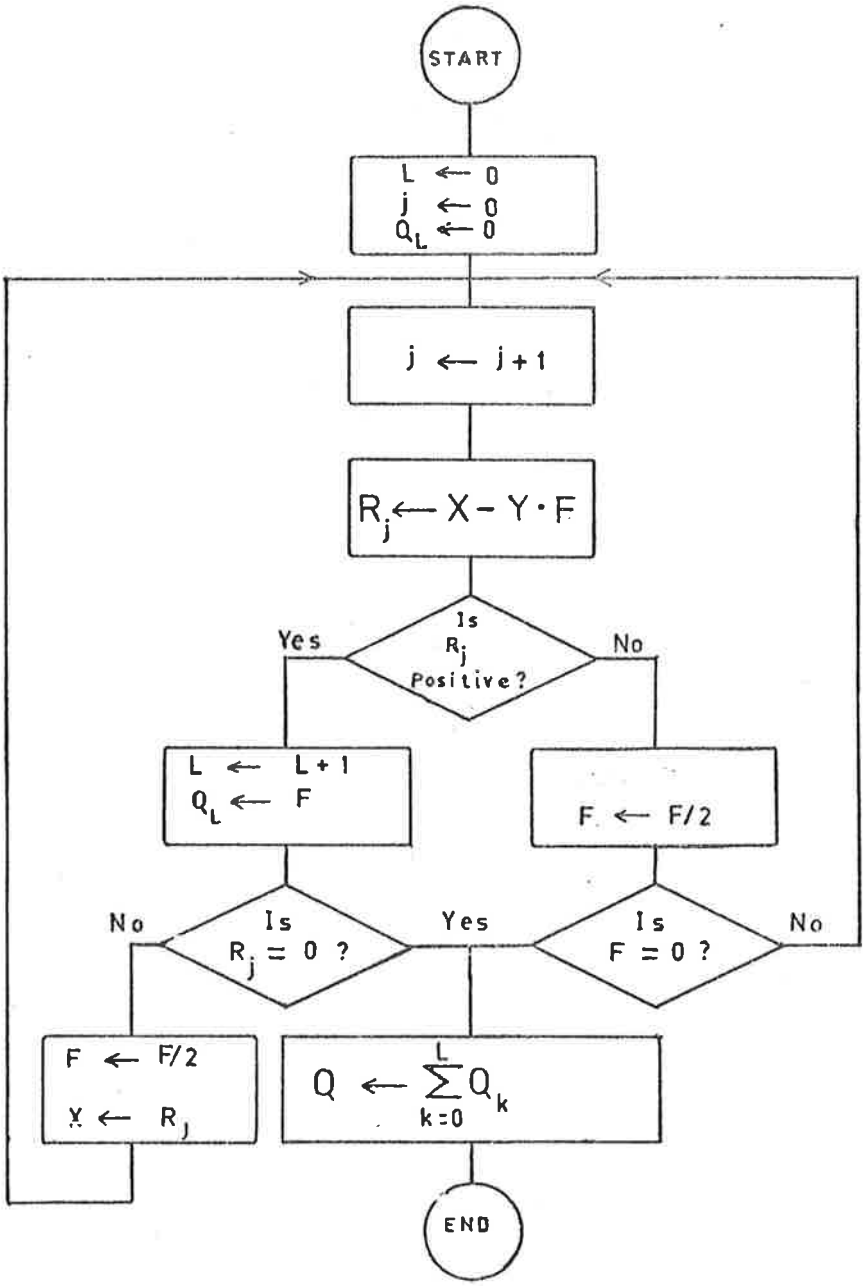


Fig. 3.1 Flow Chart Of Iterative Process in residue integer division.

The first approximation F can be determined by fast division technique [30] from the value of E(X) and E(Y). The steps that are followed are:

(i) Convert X to its mixed radix form (Appendix-5) and pick up the most significant nonzero digit a_{ux} (say).

(ii) Find $G(\lfloor a_{ux}+1 \rfloor \pi_p)$ and $G(a_{ux} \cdot \pi_p)$ from a look up table which contains the value of 2^i $(i=1,2, \ldots N)$, where N is such that $2^N \leq (M-1) < 2^{N+1}$. $(\pi_p = \prod_{i=1}^{p} m_i)$. Also find $E(\lfloor a_{ux}+1 \rfloor \pi_p)$ and $E(a_{ux}\pi_p)$ from another look up table.

(iii) Find $X' = X - G(\lfloor a_{ux}+1 \rfloor \pi_p)$ and determine the sign of X' by another mixed radix conversion. If sign is negative $G(X) = G(a_{ux}\pi_p)$ and $E(X) = E(a_{ux}\pi_p)$. Otherwise $G(X) = G(\lfloor a_{ux}+1 \rfloor \pi_p)$ and $E(X) = E(\lfloor a_{ux}+1 \rfloor \pi_p)$ $(E(X) = i$ where $i = 1,2,3, \ldots N)$. Similarly find E(Y).

The F is given by

$$F = 2^{E(X)-E(Y)} \quad \text{(in residue form)}$$

If two mixed radix converters and associated look up tables are available, F can be obtained in 2n modular operations.

3.3 Overflow

If m_i $(i=1,2,3, \ldots n)$ are the moduli of a particular residue system and the moduli are pairwise relatively prime, then the range of the numbers that can be uniquely represented

is given by $M = \pi_{i=1}^{n} m_i$. If one of the moduli is even, then the range of the positive numbers (positive half) and that of negative numbers (negative half) are defined by

{0 to M/2 -1} and {M/2 to M-1} respectively.

An overflow is said to have occurred if the result of an arithmetic operation exceeds M or the range of positive half or negative half.

3.3.1 Overflow due to addition/subtraction

An additive overflow can occur if two operands X and Y are of same signs. If operands are positive and overflow occurs, the result falls in the negative half. For negative operands, the result falls in the positive half. Therefore additive overflow can be easily detected by detecting the incorrect sign of the result. Subtractive overflow can occur if the signs of the operands differ. Since subtraction can be reduced to addition $(X + (-Y))$, subtractive overflow detection is similar to the additive overflow detection.

3.3.2 Multiplicative Overflow

When multiplicative overflow occurs, one of the two following phenomena occurs - (i) the result is of incorrect sign (exceeds the range of positive half or negative half) or (ii) the result is of correct sign, but incorrect magnitude (exceeds M).

The overflow of first category can be detected by checking the sign of the product (whether the sign is incorrect). When overflow is of second category, the following

relation [28] is valid.

$$G\left(\frac{|XY|_M}{X}\right) \neq G(Y) \qquad \ldots\ldots \text{(3.5a)}$$

First $\frac{|XY|_M}{X}$ is calculated by techniques of General Division I. Then $G\left(\frac{|XY|_M}{X}\right)$ and $G(Y)$ are obtained by logarithmic search technique. Since $\frac{|XY|_M}{X}$ is a division, more time will be needed to find it. However, if the expression (3.5a) is written as (3.5b), the division $\frac{|XY|_M}{X}$ can be avoided.

$$\frac{G((XY)_M)}{G(X)} \neq G(Y) \qquad \ldots\ldots \text{(3.5b)}$$

$G((XY)_M)$, $G(X)$ and $G(Y)$ can be obtained by fast technique proposed in [30]. If two mixed radix converters are available, the overflow of second category can be detection with

$$4n \quad \text{modular operations} \qquad \ldots\ldots \text{(3.5c)}$$

Multiplicative overflow can also be detected by using a redundant modulus [28]. The time required is at the most 3 MRC (Mixed Radix Conversion; Appendix-5) with $n+1$ moduli plus 1 modular cperation. This is equivalent to $3n+1$ modular operations. (1MRC is equivalent to $(n-1)$ modular operations [Appendix-5] with n moduli).

3.4 Floating Point Arithmetic in a Residue Number System

At present it is difficult to perform floating point arithmetic in a residue number system. There is a limited amount of published work on this problem, for example, [33]

has proposed 2 as the base of the exponent, while [32] has proposed a base of 10. However, no details of floating point arithmetic operations have been given. The algorithms proposed in [31] are of particular interest, since details of arithmetic operations have been presented with illustration. Furthermore, the format used is different from the conventional format. The details of the algorithms have been given in the following sections.

3.4.1 The representation of floating point numbers

A floating number can be represented as

$$k \cdot M^{e} \cdot M_{\delta}$$

where $|k| \leq \frac{1}{2}(M-1)$; e is an integer

$$M_{\delta} = \prod_{i=1}^{\delta} m_i \qquad (\delta \neq 0)$$

$$M_{\delta} = 1 \qquad (\delta = 0)$$

$$M = \prod_{i=1}^{n} m_i$$

and $m_1 < m_2 < m_3 < \ldots\ldots < m_n$.

3.4.2 Floating Point Arithmetic

Let $X = k_x M^{e_x} M_{\delta x}$ and $Y = k_y M^{e_y} M_{\delta y}$

and $Z = Fl(X \odot Y) = k_z M^{e_z} M_{\delta z}$ where $\odot$ denotes arithmetic operations (addition, subtraction, multiplication or division).

3.4.2a Algorithms for floating point addition/subtraction

(It is assumed that $e_x \geq e_y$ and X, Y are positive)

(1) Find $\Delta e = e_x - e_y$. If $\Delta e \geq 2$, $Fl(X \pm Y) = X$.

(2) If $\Delta e = 1$, test whether $\delta_x \geq \delta_y$. If it is true $Fl(X \pm Y) = X$, otherwise scale [28] k_y by $m_1\ m_2\ \ldots\ m_{\delta x}\ m_{\delta y+1}\ m_{\delta y+2}\ \ldots\ m_n$. Let the result be K_R.

(3) If $\Delta e = 0$, test whether $\delta_x = \delta_y$. If it is true $K_R = k_y$. Otherwise, scale k_y by $m_{\delta y+1}\ m_{\delta y+2}\ \ldots\ m_{\delta x}$ and the result be K_R.

(4) Find $k_z = k_x \pm K_R$ and set $e_z = e_x$; $\delta_z = \delta_x$.

[Addition is done using redundant modulus m_{n+1}. m_{n+1} is so chosen that $(M-1) \leq \frac{1}{2}(M \cdot m_{n+1} - 1)$ and $m_n < m_{n+1}$]

(5) Find the mixed radix digits a_i ($i = 1,2,3, \ldots, n+1$) of k_z which is associated with moduli m_i. If a_i are zeros, set $e_z = 0$, $\delta_z = 0$.

(6) Otherwise, denote by a_u the most significant non-zero digit. If $U = n+1$, scale k_z by $m_{\delta z+1}$ and increase δ_z by 1. If $\delta_z = n$, set $\delta_z = 0$ and increase e_z by 1.

(7) If $U < n$, multiply k_y by $\prod_{i=\delta z+u+1}^{\delta z+n} m_i$ which can be read from a permanent table by using U and δ_z, decrease e_z by 1, and increase δ_z by U. If $\delta_z < n$, go to step 5. If $\delta_z \geq n$, increase e_z by 1 and decrease δ_z by n, go to step 5.

(8) If U = n, test whether $\delta_z = 0$. If it is true, k_z is normalized. If $\delta_z \neq 0$, fetch $\frac{1}{2}((M|m_{\delta z})+1)$ from look up table. Find $|k_z| - \frac{1}{2}((M|m_{\delta z})+1)$ and determine the sign. If sign is negative multiply k_z by m_{δ_z} and decrease δ_z by 1.

3.4.2b Algorithms for floating point multiplication

(1) Find $|k_x|_{m_i}$ and $|k_y|_{m_i}$ $(i = n+1, n+2, \ldots, n+p)$ by method of base extension [28].

[Multiplication is done using p numbers of redundant moduli to prevent loss of information due to multiplicative overflow. The redundant moduli are so chosen that $p \leq n$ and

$$\frac{1}{4}(M-1)^2 \cdot m_1 \leq \frac{1}{2}(M \cdot m_{n+1} \cdot m_{n+2} \cdots m_{n+p} - 1).$$

and $m_{n+1} < m_{n+2} < \ldots < m_{n+p}$]

(2) Set $e_z = e_x + e_y$; $\delta_z = \delta_x$.

(3) If $\delta_y = 0$, find $k_z = k_x \cdot k_y$, and if $\delta_y = 1$, find $k_z = k_x \cdot k_y \cdot m_1$, where the residue representation of m_1 is read from look up table. Otherwise, increase e_z by 1, find $k_x \cdot k_y$ and scale the result by $m_{\delta_y+1} \cdot m_{\delta_y+2} \cdots m_n$ to get k_z.

(4) Find the mixed radix digits (Appendix-5) a_i $(i = 1, 2, \ldots, n+p)$ of k_z. If all the digits are zeros, set $\delta_z = 0$, $e_z = 0$. Otherwise denote the most significant non-zero digit by a_u.

(5) If $U \geq n+2$, replace k_z by $\left| k_z \right|_{\prod_{i=\delta_z+1}^{\delta_z+u-n} m_i}$

and increase δ_z by $u-n$. If $\delta_z = n$, set $\delta_z = 0$ and increase e_z by 1. Go to step 4.

(6) If U = n+1, replace k_z by $(k_z|m_{\delta_{z+1}})$ and increase δ_z by 1. If $\delta_z = n$, set $\delta_z = 0$ and increase e_z by 1. Fetch $\frac{1}{2}$ (M-1) (in residue form) from a table, compute $|k_z| - \frac{1}{2}$ (M-1) and find the sign. If sign is positive, replace k_z by $(k_z|m_{\delta_{z+1}})$ and increase δ_z by 1. If $\delta_z = n$, set δ_z to zero and increase e_z by 1.

3.4.2c Algorithms for floating point division

Let X and Y be the dividend and divisor respectively.

(1) Find $|k_x|_{m_i}$ and $|k_y|_{m_i}$ (i = n+1, n+2, ... n+p) by method of base-extension.

(2) Multiply k_x by A which can be read from a look up table.

[The value of A depends on δ_x and δ_y.

a) If $\delta_x \geq \delta_y$ and $\delta_y = 0$, A = M

b) If $\delta_x \geq \delta_y$ and $\delta_y \neq 0$, A = B.C

where $B = \pi_{i=\delta_y+1}^{n} m_i$ and $C = \pi_{i=\delta_x-\delta_y+1}^{\delta_x} m_i$

c) If $\delta_x = \delta_y$ and $\delta_y \neq 0$, A = M

d) If $\delta_x < \delta_y$, $A = M_{\delta_x} \cdot B \cdot D$ where

$$D = \pi_{i=n+\delta_x-\delta_y+1}^{n} m_i \quad]$$

(3) If $\delta_x < \delta_y$, decrease e_z by 1 and increase δ_z by n.

(4) Find $k_z = (k_x . A|k_y)$ by scaling.

(If $k_z = 0$, division is undefined)

3.5 Translation of Residue Numbers to Decimal Numbers

A number in residue form is not readily understandable. Therefore the residue system needs a provision for translating residue numbers to decimal numbers. Since output translation is less frequently used, more time can be allotted for translation than for arithmetic operations.

The Chinese Remainder Theorem [28] can be used for output translation provided that the system has the facility of mod-M operation. The A(X) method [34] does not need mod-M operation, however, it needs super modulus to determine A(X), and the method is applicable only if the base w (in this case 10) is pairwise relatively prime with m_i. Benerjee and Brzozowski [35] pointed out that the method of base extension [28] can be used provided (i) gcd $(w, m_i) = 1$ or (ii) gcd $(w, m_i) \neq 1$ for only one modulus and $w \geq m_i$ for that particular modulus.

[gcd stands for greatest common divisor.]

3.6 Conclusion

It is obvious from above discussions that addition/subtraction and multiplication are straight forward operations; and delay is equal to the delay of one modular operation. The operations can easily be implemented with memories (look up tables). Therefore there is no real problem in practical implementation of these operations.

Division, however, is not so easy as addition/subtraction and multiplication. Determination of the first approx-

imation F which needs $\left\lceil \frac{n}{2} \log_2 c + n+2 \right\rceil$ modular operations using logarithmic search technique is obviously a time-consuming operation. However, fast division technique can reduce that time (for determination of F) to 2n modular operations. This means that the division process will be faster. Still an investigation, whether a fast division technique can be implemented in a different way with speed improvement, is worthwhile.

Multiplicative overflow detection in redundant residue system is faster (3n + 1 modular operations) than in non-redundant system (4n modular operations). However, a redundant system needs an extra modular operation which may not be desirable. The choice of redundant modulus may be critical. This means that if the redundant modulus gets large, implementation of operation with that modulus may be difficult and operation may get slow. Again, the introduction of a redundant residue decreases the available range of the system which may not be appreciated. Therefore an alternate approach is to choose the non-redundant system and carry out an investigation for faster overflow detection technique.

As discussed earlier floating point multiplication and division need p numbers of redundant moduli to prevent loss of information due to multiplicative overflow. This compels the system to have the capacity of (n+p) modular operation units and the capacity of MRC with (n+p) moduli. If p is large e.g. (p=n), then MRC with (n+p=2n) may be difficult. If p is chosen to be smaller, each redundant

modulus will get larger (to meet the requirement stated before) and mechanisation will be difficult. Therefore it is worthwhile to investigate alternative floating point algorithms and particularly those which can be implemented within the integer residue arithmetic environment.

In output translation, the Chinese Remainder Theorem needs arithmetic with mod-M which is a costly affair. The A(X) method is restricted by the requirement that (w, m_i) must be pairwise relatively prime. In conversion of residue numbers to decimal numbers, the base is 10 which is a composite of 2 and 5; and a residue system generally has one even modulus to have well-defined positive half and negative half; therefore the above requirements cannot be fulfilled. The method of using base extension covers a wide range of situations. However, in a system where $w < m_i$ and gcd $(w, m_i) \neq 1$, the base extension method is not applicable. In that situation an alternative method must be investigated.

The next chapter proposes a few algorithms relating to the above problems.

APPEND TO CHAPTER 3 OF THESIS

"Residue Arithmetic in Digital Computers"

R.C. Debnath

FURTHER RELEVANT PAPERS

Jullien [82] establishes that the residue operations like addition, subtraction, multiplication and scaling can be easily implemented with look-up tables in currently available ROMs. He also contends that scaling by the metric vector estimate process is either equally or more highly efficient than the exact division technique. Residue arithmetic operations (addition, subtraction, multiplication and scaling) implemented with look-up tables in high density ROM can allow for a high speed digital processor.

The conversion of a residue number to a binary number is not an easy process. The method based on the Chinese Remainder Theorem requires residue arithmetic operation with Modulo-M, and is therefore a costly method. A more efficient method is to use the Mixed Radix Decoding technique [83]. This technique needs less memory locations and can also be implemented with currently available ROMs or PROMs.

Additional References:-

[82]. G.A. Jullien, "Residue number scaling and other operations using ROM arrays", IEEE Trans. on Computers, Vol.C-27, No.4, April 1978, pp.325-336.

[83]. A. Baraniecka and G.A. Jullien, "On decoding techniques for residue number system realizations of digital signal processing hardware", IEEE Trans.on Circuits and Systems, Vol.-CAS 25, No.11, No. 1978, pp.935-936.

from the following groupings of the numbers from 0 to M/2.

Category 1 : Group-0 $\supset$ (0,1);

Group-1 $\supset$ $(2, 3, \ldots, (M)^{\frac{1}{2}}-1)$

Group-2 $\supset$ $((M)^{\frac{1}{2}}, (M)^{\frac{1}{2}}+1, \ldots, M/2)$

Category 2 : Group-0 $\supset$ (0,1);

Group-1 $\supset$ $(2, 3, \ldots, (\frac{M}{2})^{\frac{1}{2}}-1)$

Group-2 $\supset$ $((\frac{M}{2})^{\frac{1}{2}}, (\frac{M}{2})^{\frac{1}{2}}+1, \ldots, M/2)$

From the groupings of the numbers, it is clear that no overflow occurs for (Group-0) • {(Group-1) or (Group-2)}; but overflow will occur for (Group-2) • (Group-2). However, there is no certainty whether overflow will occur for (Group-1) • {(Group-1) or (Group-2)}. Even if overflow occurs, it is not clear whether the product exceeds P(½) or N(½) or (M-1). This is due to poor resolution of numbers in groups 1 and 2. Therefore a 'Logarithmic Distribution' which gives high resolution of numbers in groups is proposed.

4.2.1 Logarithmic Distribution of residue numbers

The numbers 0 to M/2 (positive half and the highest negative number) are distributed among several groups according to the table 4.1.

If $g(x)$ and $g(Y)$ are the group numbers of X and Y respectively, then it can be shown from Table 4.1 [Appendix-1; (AP1(a))] that

Case (i) : if $g(X) + g(Y) < y$, no overflow occurs;

Case (ii) ; if $g(X) + g(Y) > y$, overflow must occur.

Case (iii) : if $g(X) + g(Y) = y$, overflow may or may not

occur; however, if it occurs, the product will be of incorrect sign.

Table 4.1

Distribution of Numbers

Numbers	Group
0	0
1	1
$W.2^{-\delta+2} \longrightarrow W.2^{-\delta+3}-1$	2
$W.2^{-\delta+3} \longrightarrow W.2^{-\delta+4}-1$	3
$W.2^{-\delta+(\delta-1)} \longrightarrow W.2^{-\delta+\delta}-1$	$\delta-1$
$W.2^{-\delta+\delta} = W \longrightarrow W.2^{-\delta+(\delta+1)}-1$	δ
$W.2^{-\delta+(\delta+1)} \longrightarrow W.2^{-\delta+(\delta+2)}-1$	$\delta+1$
$W.2^{-\delta+(y-1)} \longrightarrow M/2$	$y-1$

[Where $y = n+1$, such that $2^n \leq M < 2^{n+1}$; in other words y is the highest power to 2 contained in 2M. When y is even, $W = (M)^{\frac{1}{2}}$, and $\delta = y/2 + 1$. When y is odd, $W = (\frac{M}{2})^{\frac{1}{2}}$, and $\delta = (y+1)/2$.]

However, in most cases $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ is not an integer number. In those cases the closest higher integer value of $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ is chosen to be the value of W. The lowest number in the $(\delta-i)^{th}$ group ($i = 1, 2, \ldots, \delta-1$) is the rounded integer value of L/2, where L is the lowest integer number in $(\delta-i+1)^{th}$ group. The lowest number in the $(\delta+k)^{th}$

group (k = 1,2, ... , δ-3) is given by 2^k.W. However, in Case (iii) in some situations, multiplication of two highest numbers in two separate groups may be equal to or greater than M (due to rounding error and choice of the closest higher integer value of $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ for W). Under these circumstances a minor adjustment of the highest integer is needed. The highest number of the higher group of the two is decreased by a minimum number, say α, such that the product of the decreased number and the other highest number becomes less than M. The lowest number of next higher group (with respect to the higher group of the two) is decreased by the same number α.

If $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ is an integer number (i.e. M or M/2 is a rational number) and even, then the lowest number of $(\delta-1)^{th}$ group is to be increased by 1 and the highest number of the $(\delta-2)^{th}$ group is increased by 1.

Example 1: If $m_4 = 16$, $m_3 = 15$, $m_2 = 13$ and $m_1 = 11$, construct a 'group table'.

Solution:

(i) M = 16x15x13x11 = 34320

(ii) $2^{15} < M < 2^{16}$; therefore $y = 16$.

(iii) $(M)^{\frac{1}{2}} = 185{\cdot}256$; therefore W = 186.

(iv) $\delta = 16/2 + 1 = 9$

(v) M/2 = 17160

(vi) Table 4.2

Table 4.2

Group table for Example 1.

Numbers		Group
0		0
1		1
2		2
3 →	5	3
6 →	11	4
12 →	23	5
24 →	46	6
47 →	92	7
93 →	185	8
186 →	371	9
372 →	743	10
744 →	1487	11
1488 →	2975	12
2976 →	5951	13
5952 →	11903	14
11904 →	17160	15

Example 2: For $m_3 = 2$, $m_2 = 3$, and $m_4 = 13$, construct a 'group table'.

Solution:

(i) $M = 2x3x13 = 78$

(ii) $2^6 < M < 2^7$; therefore $y = 7$

(iii) $(\frac{M}{2})^{\frac{1}{2}} \simeq 6{\cdot}24$; therefore $W = 7$

(iv) $\delta = (y+1)/2 = 4$

(v) $M/2 = 55$

(vi) Table 4.2(a)

Table 4.2(a)

Group table for Example 2.

Numbers	Group
0	0
1	1
2 ⟶ 3	2
4 ⟶ 6	3
7 ⟶ 13	4
14 ⟶ 27	5
28 ⟶ 55	6

It is seen from the table 4.2(a) that the product of 6 (the highest number in 3^{rd} group) and 13 (the highest number in 4^{th} group) is equal to 78(=M). Therefore the highest number of 4^{th} group is replaced by 12 and the lowest number of 5^{th} group is replaced by 13. The correct 'group table' is shown in table 4.2(b).

Table 4.2(b)

Correct group table for Example 2.

Numbers	Group
0	0
1	1
2 ⟶ 3	2
4 ⟶ 6	3
7 ⟶ 12	4
13 ⟶ 27	5
28 ⟶ 55	6

Example 3: If $m_3 = 2$, $m_2 = 11$, and $m_1 = 5$, construct a 'group table'.

Solution:

(i) $M = 2x11x5 = 110$

(ii) $2^6 < M < 2^7$; therefore $y = 7$

(iii) $(\frac{M}{2})^{\frac{1}{2}} = 7{\cdot}41$; therefore $W = 8$

(iv) $\delta = (y+1)/2 = 4$

(v) $M/2 = 55$

(vi) Construct Table 4.3

Table 4.3

Group Table for Example 3.

Numbers	Group
0	0
1	1
2 → 3	2
4 → 7	3
8 → 15	4
16 → 31	5
32 → 55	6

Therefore multiplicative overflows can be detected by (a) testing for incorrect sign of the product, and (b) comparing the sum of group numbers of multiplicand and multiplier with the total numbers of groups y. If sum is greater than y, overflow must occur; if it is less than y, overflow will not occur.

4.2.2 Design Ideas for Overflow Detection

Since the group numbers described previously are assigned to the positive numbers and to M/2, and a number may be positive or negative, the information of sign and magnitude of the number is essential. The MRC (Appendix-5) is the only way to get sign and magnitude information. From the mixed radix digits (MRD) of a negative number it is possible to form the MRDs of its positive 'counterpart' [Appendix-1, (AP1(b))].

When number of MRDs gets larger, a partition between them is essential. This partition divides the MRDs into two parts - the most significant part (MSP) and the least significant part (LSP). If the modulus M_n is chosen to be even, and is placed at the most significant digit (MSD) position a_n, then a_n contains the information of sign and falls in the MSP. With the sign information and MRDs of the LSP, a look up table is formed for the group numbers of positive/'counterpart' positive numbers in the LSP. (The table is formed from the decimal value (DV) of the MRDs.)

When the number is negative the MSP needs the information whether the LSP is zero to form the MRD of the positive 'counterpart'. From this information (zero-LSP), the sign and the MRD of MSP a look up table is formed, assuming the MRDs of the LSP are zeros. (The table is formed from the DV of the MRDs.)

The group number of a number depends on the DV of all MRDs (MSP + LSP). From the observation of the range of

numbers in a group of the MSP and the DV of the MSP, it will be clear that the group number of certain MRDs of the MSP will transit to the next higher group number if certain minimum number from the LSP is added to the DV of those particular MRDs. Those MRDs may be termed as 'transition' MRDs and the minimum number required for transition as 'transition' number. The transition number is identified by observing whether the number, obtained as a result of subtraction of the DV of the MRDs in the MSP from the lowest number in the next higher group (with respect to the group of the MRDs in the MSP), falls within the range of the LSP. Therefore along with the group tables for the MSP and LSP, there must be one table containing the transition numbers and another table giving the negative binary values (2's complement) of the LSP. (The transition number of the non-transition MRDs is taken to be zero.) The transition number is added to the negative value of the binary number of the LSP. The carry from the most significant bit (MSB) position indicates that the group number of the MSP must transit to the next higher group. Therefore the carry is added to the group number of the MSP. The group number of a number is given by the group number of the MSP, or the group number of the MSP plus one if transtion occurs, or the group number of the LSP if the group number of the MSP is zero.

Overflow is detected by comparing the sum of the group numbers of operands with the total number of groups and detecting the incorrect sign of the product.

4.2.3 Design Example on Overflow Detection

The moduli $m_4 = 16$, $m_3 = 15$, $m_2 = 13$ and $m_1 = 11$ have been chosen for this design example. The MRDs are designated by a_4, a_3, a_2 and a_1. The range of the LSP is 13x11 = 143 (0 to 142) and that of the MSP is 16x15x13x11 - 13x11 = 34177 i.e. from 143 to 34177. The highest number in the LSP is 142 and the MSP is non-zero for a number greater than 142.

Examining the Table 4.2 and the DV of the MRDs in the MSP, Table 4.4 which shows the 'transition' MRDs and 'transition' numbers, can be formed.

Table 4.4

Transition MRDs and transition numbers

Transition MRDs a_4, a_3	The DV of the Transition MRDs = a_4.15.13.11 + a_3.13.11	Corresponding Group of the MRD	The lowest number in the next higher group	Transition Number
0, 1	143	8	186	186- 143 = 43
0, 2	286	9	372	372- 286 = 86
0, 5	715	10	744	744- 715 = 29
0, 10	1430	11	1488	1488- 1430 = 58
1, 5	2860	12	2976	2976- 2860 =116
2, 11	5863	13	5952	5952- 5863 = 83
5, 8	11869	14	11904	11904-11869 = 35

Figure 4.1 shows the circuit for multiplicative overflow detector. The circuit in the solid lined box detects incorrect sign of the product. If the Exclusive OR of the sign of the X, Y and Z (SX, SY, and SZ) is '0', the sign of

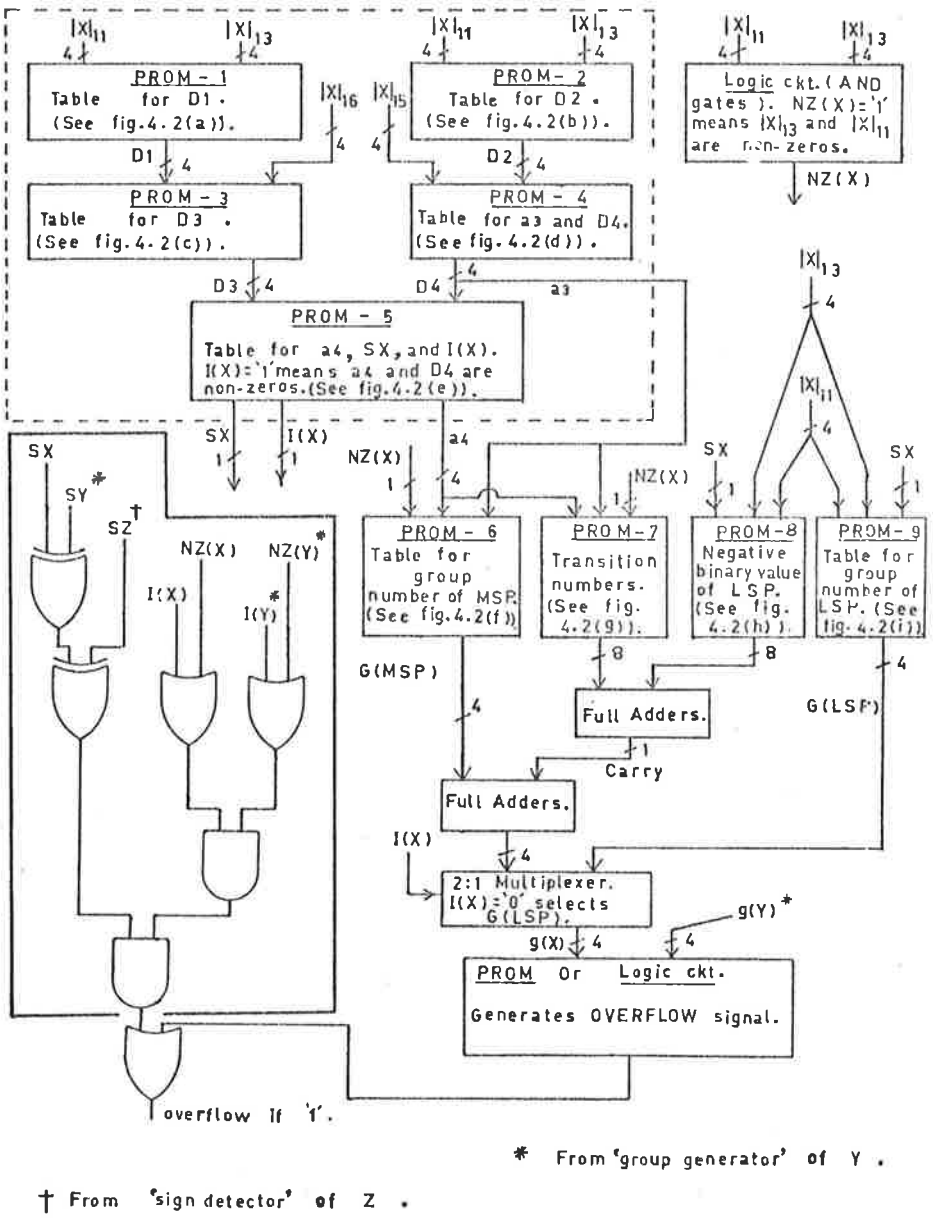


Fig.4.1 Schematic diagram for multiplicative overflow detection in the 16-15-13-11 residue system.

Function of PROM-1 ; (256x4)

It forms a table of D_1, where

$D_1 = |a_2 \cdot 11 + a_1|_{16}$,

where a_2 and a_1 are the MRDs of the LSP.

$$a_2 = \left[\left[|X|_{13} - \left[|X|_{11} \right]_{13} \right]_{13} \cdot \left| \frac{1}{11} \right|_{13} \right]_{13}$$

$$a_1 = |X|_{11}$$

Fig 4.2(a) : Function of PROM-1

Function of PROM-2 ; (256x4)

It forms a table of D_2, where

$D_2 = |a_2 \cdot 11 + a_1|_{15}$, where

a_2 and a_1 are same as in Fig 4.2(a).

Fig 4.2(b) : Function of PROM-2

Function of PROM-3 ; (256x4)

It forms a table of D_3, where

$$D_3 = \left[\left| \frac{1}{13x11} \right|_{16} \cdot \left| |X|_{16} - D_1 \right|_{16} \right]_{16}$$

Fig 4.2(c) : Function of PROM-3

Function of PROM-4 ; (256x4)

It forms a table of a_3, where

$$a_3 = \left[\left| \frac{1}{13.11} \right|_{15} \cdot \left| |X|_{15} - D_1 \right|_{15} \right]_{15}$$

$$D_4 = |a_3|_{16} = a_3, \text{ since } 15 < 16$$

Fig 4.2(d) : Function of PROM-4

Function of PROM-5 ; (256x8)

It forms a table of a_4, SX, $I(X)$, where

$$a_4 = \left[\left| \frac{1}{15} \right|_{16} \cdot |D_3 - D_4|_{16} \right]_{16}$$

SX = '1' if $a_4 > 7$; $I(X)$ = '1', if

D_3 and D_4 are nonzero

Fig 4.2(e) : Function of PROM-5

Function of PROM-6 ; (512x4)

It forms a table of group number of the MSP. If $NZ(X)$ = '0' or '1' and SX = '0', the decimal value, DV(0) = a_4.15.13.11 + D_4.13.11. If SX = '1', $NZ(X)$ = '0' and D_4 is nonzero, the decimal value, DV(1) = (15-a_4) .15.13.11 + (15-D_4).13.11. If SX = '1', $NZ(X)$ = '0' and D_4 is zero, the decimal value, DV(2) = (16-a_4) .15.13.11 + (15-D_4).13.11. If SX = '1', $NZ(X)$ = '1', the decimal value, DV(3) = (15-a_4).15.13.11 + (14-D_4) .13.11. The group numbers are assigned according to the Table 4.2.

Fig 4.2(f) : Function of PROM-6

Function of Prom-7 ; (512x8)

It forms a table of 'transition' number according to the table 4.4.

Fig 4.2(g) : Function of PROM-7

Function of PROM-8 ; (512x8)

It forms a table of negative binary value of the LSP from its decimal value. If SX = '0', the decimal value, DV(4) = a_2.11 + a_1. If SX = '1' and a_1 is zero, the decimal value, DV(5) = (13-a_2).11. If SX = '1' and a_1 is nonzero, the decimal value DV(6) = (12-a_2).11 + (11-a_1). a_2 and a_1 are same as shown in Fig 4.2(a)

Fig 4.2(h) : Function of PROM-8

Function of PROM-9 ; (512x4)

It forms a table of group number of the LSP from its decimal value (shown in Fig 4.2(h) according to the Table 4.2.

Fig 4.2(i) : Function of PROM-9

Function of PROM-10 ; (256x4)

It forms a table of overflow (not final overflow) according to

(i) Overflow = '0', if $g(X) = g(Y) \leq 16$

(ii) Overflow = '1', if $g(X) = g(Y) > 16$

Fig 4.2(j) : Function of PROM-10

the product is correct; otherwise the sign is incorrect. However, false indication of overflow is given if one operand is negative and another is zero. The circuit right side of the circuit in solid-lined box detects whether any of the operands is zero.

Since $|X|_{13}$ and $|X|_{11}$ contain the information on a_2 and a_1 it is not necessary to generate a_2 and a_1. This saves use of one PROM. The circuit in the dash lined box generates the sign of X (SX), the MRD (a_4) and the information of nonzero a_4 and D_4. Two additional circuits are required - one is for generation of the sign of Y (SY), the group number of Y ($g(Y)$) and $NZ(Y)$ and the other one is for the sign of the result Z (SZ).

Performance:

Delay in overflow detection is the delay through the longest path which, for the circuit in Fig 4.1, is through the circuit required for overflow detection by the 'grouping technique'. Therefore

Delay, D = (D_1 = delay in sign detection)

+ (D_2 = delay in getting Group Number of the LSP)

+ (D_3 = 8 bits adder delay + 4 bits adder delay + 2:1 multiplexer delay)

+ (D_4 = delay in PROM-10)

+ (D_5 = delay in OR-gate at the last stage).

Again $D_1 = 2T_1 + T_2$ where

T_1 = access time of (256x4) PROM

T_2 = access time of (256x8) PROM

Let T_{ADD} = delay in a 4-bit full adder.

T_{MUX} = delay in 2:1 multiplexer.

T_{OR} = delay in OR-gate.

Therefore

$$D = 2T_1 + T_2 + T_3 + 3T_{ADD} + T_{MUX} + T_1 + T_{OR}$$

$$= 3T_1 + T_2 + T_3 + 3T_{ADD} + T_{MUX} + T_{OR} \quad \ldots\ldots (4.1)$$

where T_3 = access time of (512x8) PROM

However, delay in generating group number (Dg) is the delay in generating $g(X)$. Therefore Dg can be expressed as

$$Dg = D_1 + D_2 + D_3$$

$$= 2T_1 + T_2 + T_3 + 3T_{ADD} + T_{MUX} \quad \ldots\ldots (4.1a)$$

Using TTL components (Table 6.3(a)), the delay in multiplicative overflow detection is given by

$$D = 3 \cdot 40 + 60 + 60 + 3 \cdot 7 + 4 + 4$$

$$= 120 + 120 + 21 + 8$$

$$= 269 \text{ ns.} \qquad \ldots\ldots (4.1b)$$

The delay in generating group number is given by

$$Dg = 2 \cdot 40 + 60 + 60 + 3 \cdot 7 + 4$$

$$= 80 + 60 + 60 + 21 + 4$$

$$= 225 \text{ ns.} \qquad \ldots\ldots (4.1c)$$

4.3 General Division

The method of division consists of two processes - (i) determination of first approximation F, and then (ii) iterations using successive approximation. (The successive approximation is the number which is equal to the previous approximation divided by 2). Since the group number contains the information on the magnitude of the number, it is possible to use the same piece of hardware,used for generation of group numbers, for determination of first approximation.

The F determined from the group numbers of the operands is always positive, however, division is not limited to positive operands only. The detail of the technique for determination of the first approximation and successive approximation has been given in section 4.3.2.

4.3.1 A Division Process

The division process can be best described with a flow chart in Fig 4.3. The sign of the result of division is the Exclusive OR function of the signs of the operands X and Y. (X is the dividend and Y is the divisor.)

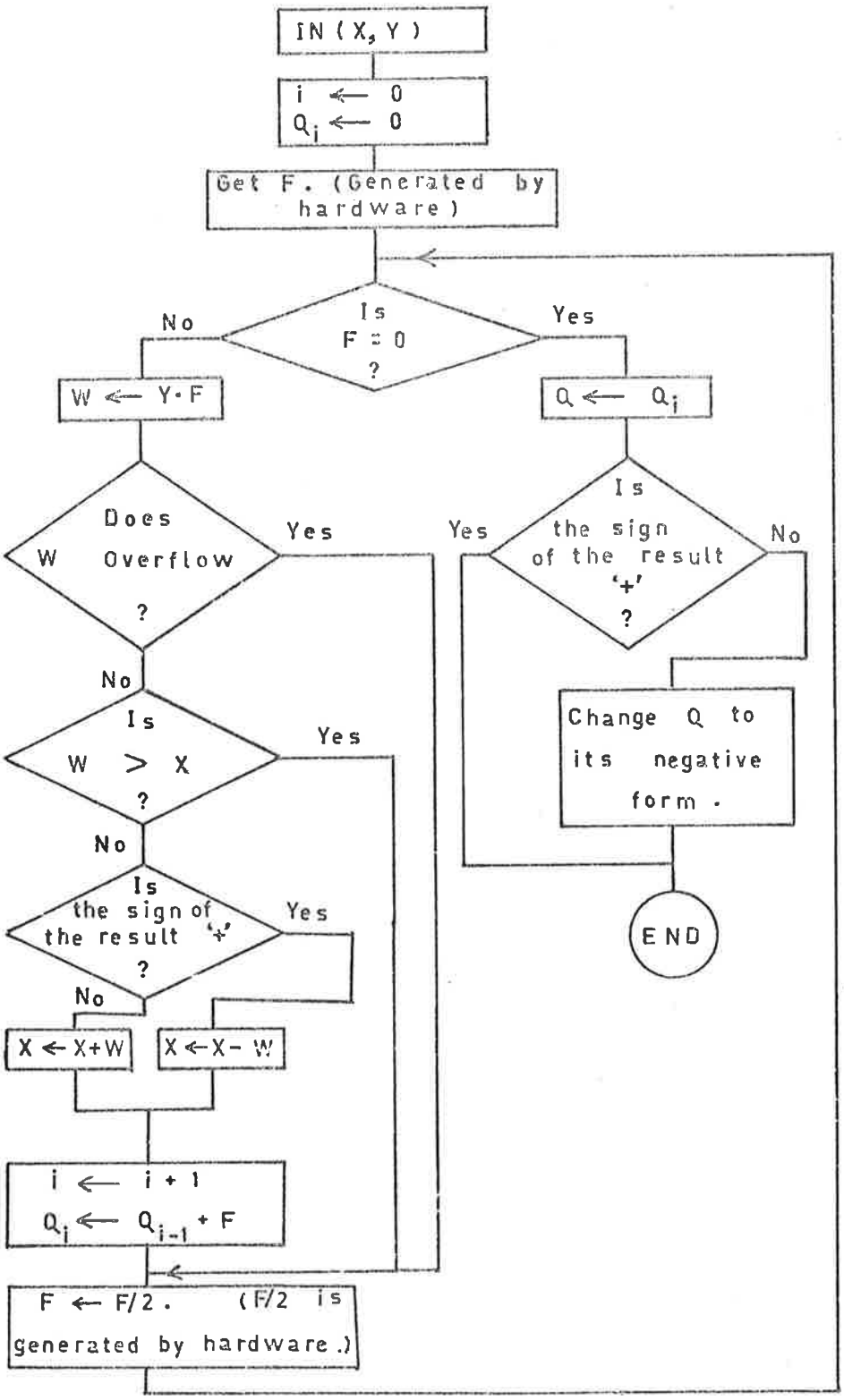


Fig. 4.3 Flow chart of residue integer division process.

4.3.2 How to find the first approximation and successive approximations

Two look-up tables are required to obtain F and successive approximation; one is used to get F and other one to get successive approximation. The value of F depends on the group numbers $g(X)$ and $g(Y)$ as shown below.

(i) If $g(X) \geq g(X)$; $F = 2^{g(X) - g(Y)}$

(ii) If $g(X) < g(Y)$; $F = 0$

(F or successive approximation is tabled in residue form.)

Fig 4.4 is a schematic diagram of the computer circuitry required to get F and successive approximation. The dotted line in Fig 4.4 indicates that either X or Y data bus can be used to issue 'previous approximation' to the multiplexer. The function of the computer is to perform iterations and to issue X and Y to get F or successive approximation from previous approximation.

If the address lines of an available are equal to or less than the sum of bits of each of the moduli, then the circuitry in Fig 4.4 is not feasible. An alternate approach which needs a smaller number of address lines to PROM is shown in Fig 4.5. In this case the PROM forms table of F (which also gives successive approximation), according to the number output from the subtractor. If the $(n+1)^{th}$ bit is '1' (i.e. negative), the PROM gives zero value; otherwise it gives the residue number of 2^k (where k is the positive output from the subtractor). The computer receives two sets of information - one is F or successive approximation and the other is the value of C´. For i^{th} iteration the

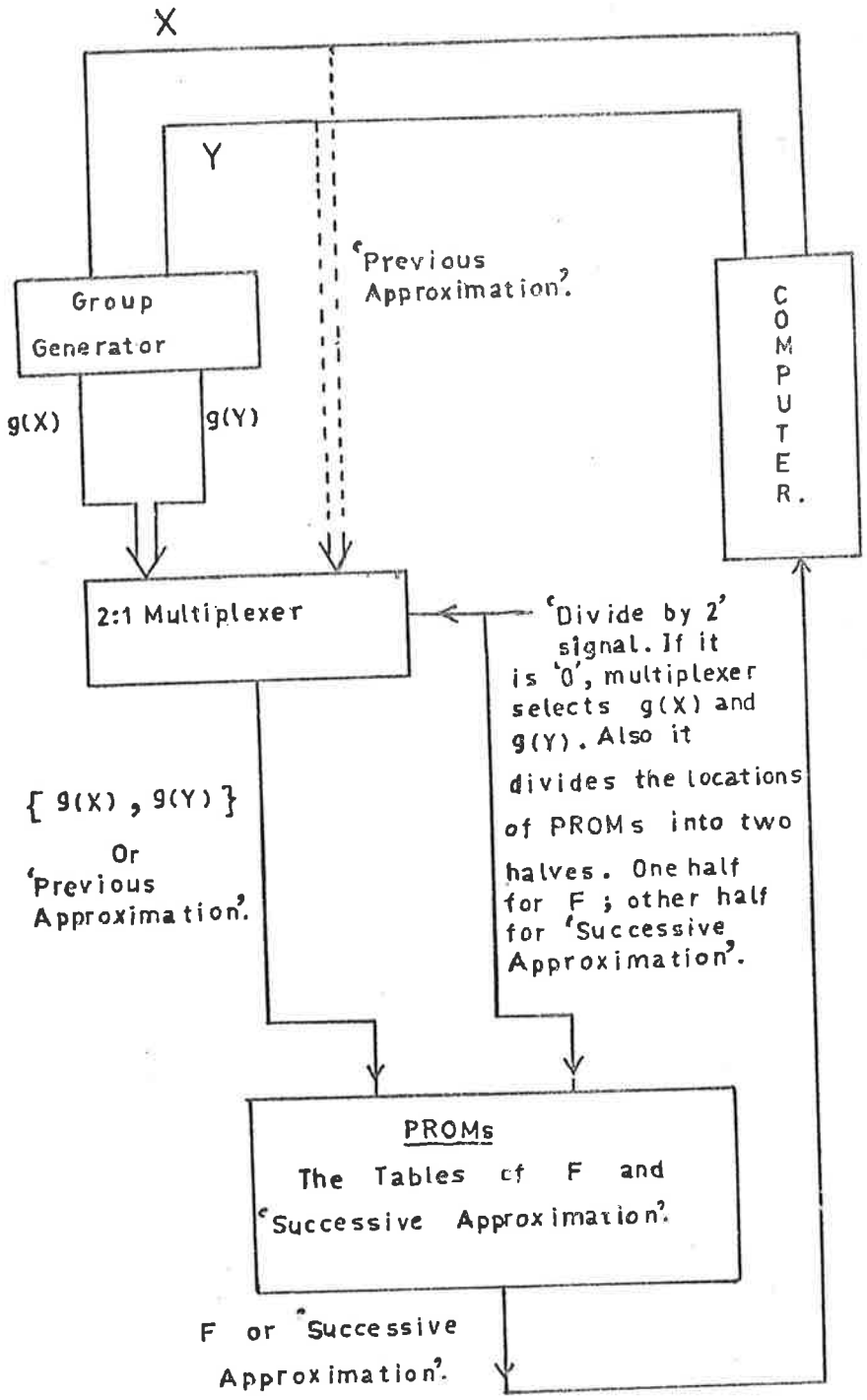


Fig. 4.4: Schematic diagram for generation of F and 'Successive Approximation'.

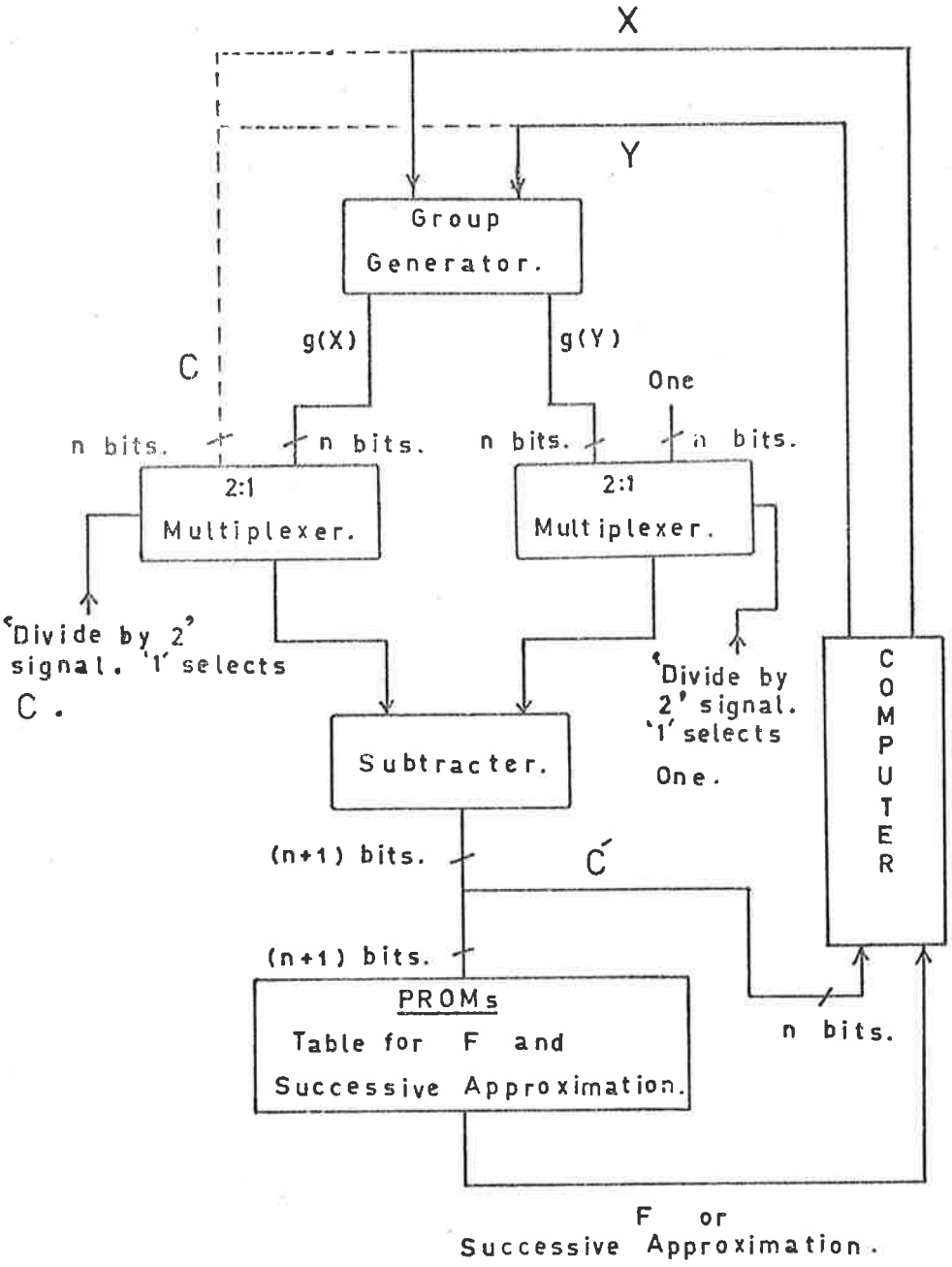


Fig.4.5 Schematic diagram of an alternate circuit for F and Successive Approximation.

computer issues the value of C (through the data bus X or Y) which is equal to C′ obtained during $(i-1)^{th}$ iteration and receives a new value of C′ which is (C-1), and the approximation.

Since the number of bits required to represent the group number is far less than the sum of bits of each modulus, it is possible to use PROM with less address lines. For example, in the case of the residue system of moduli 16, 15, 13 and 11, the sum of bits of each modulus is 16. Since a 64k PROM (16 address lines) is not yet available, the circuit in Fig 4.4 is not feasible. Whereas the circuit in Fig 4.5 can be implemented with a PROMs with 5/more than 5 address lines.

Performance

Delay in obtaining F by the circuit in Fig 4.4 is given by

$$D(1) = D_g + T_{MUX} + T_F \quad \ldots\ldots (4.2)$$

where D_g = delay in forming group.

T_{MUX} = delay in 2:1 multiplexer.

T_F = access time of the PROMs.

According to the circuit in Fig 4.5, delay is given by

$$D(2) = D_g + T_{MUX} + T_{SUB} + T_F \quad \ldots\ldots (4.3a)$$

where T_{SUB} = delay in subtraction.

Again $T_{SUB} = T_{INV} + T_{ADD}$.

where T_{INV} = delay in Inverter; and T_{ADD} = delay in addition.

Therefore equ. (4.3a) becomes

$$D(2) = Dg + T_{MUX} + T_{INV} + T_{ADD} + T_F \quad \ldots\ldots \quad (4.3b)$$

For 16-15-13-11 moduli system, and using TTL components (Table 6.3(a)), Dg = 225 ns (from 4.1c) and

$$D(2) = 225 + 4 + 3 + 7 + (T_F = 35)$$
$$= 274 \text{ ns} \quad \ldots\ldots \quad (4.3c)$$

* [T_F = 35 ns for DM7577, a (32 x 8) PROM [80]]

4.4 Floating Point Arithmetic

This section proposes a new form of representation of floating point number; it is represented by an integer multiplied by a suitable base of exponent depending on the moduli of the residue system. An advantage of this form is that floating arithmetic can be performed within the integer residue arithmetic environment without the need for any redundant modulus. The implementation is therefore economical.

4.4.1 Theoretical concept

If R is the highest absolute value of mantissa of a computing system, and β is a positive integer such that $(\beta-1)^2 < R \leq \beta^2$, then any number A within the range from 0 to R can be represented by

$$A = \{[A|\beta]\}\beta^1 + \{A-([A|\beta])\beta\}\beta^0$$
$$= a_2\ \beta^1 + a_1\ \beta^0 \quad \ldots\ldots \quad (4.4)$$

Where a_2 and a_1 are the co-efficients of β^1 and β^0, i.e. $a_2 = [A|\beta]$ and $a_1 = (A-\{[A|\beta]\}\beta)$.

The range of the co-efficients is from 0 to $(\beta-1)$. Similarly R can be represented by

$$R = r_2 \beta + r_1 \qquad \ldots\ldots (4.5)$$

Case (i) : For $\beta^2 = R$, $r_2 = \beta$ and $r_1 = 0$.

Case (ii) : For $(\beta-1)^2 < R < \beta^2$, the maximum and minimum values of R that can satisfy the above condition of β can be denoted by R_{MAX} and R_{MIN} respectively and expressed as

$$R_{MAX} = (\beta-1)\beta + (\beta-1) \qquad \ldots\ldots (4.6)$$

i.e. $r_2 = (\beta-1)$ ..... (4.6a)

$r_1 = (\beta-1)$ ..... (4.6b)

$$R_{MIN} = (\beta-1)^2 + 1 = (\beta-2)\beta + 2 \qquad \ldots\ldots (4.7)$$

i.e. $r_2 = (\beta-2)$ ..... (4.7a)

$r_1 = 2$ ..... (4.7b)

4.4.2 Floating point representation of a number

A number X can be represented in floating point form by

$$X = A\beta^n \qquad \ldots\ldots (4.8)$$

Where A is the mantissa, and lies within the range from 0 to R. β is the base of the exponent which must satisfy the conditions stated previously, and n is the exponent.

4.4.3 Floating Point Arithmetic with zero-exponent

4.4.3a Floating point addition/subtraction : with zero-exponent

$$X = A\beta^0 = A = a_2 \beta + a_1 \quad \text{(say)} \quad \ldots\ldots (4.9)$$

$$Y = B\beta^0 = B = b_2 \beta + b_1 \quad \text{(say)} \quad \ldots\ldots (4.10)$$

Therefore

$$Z = Fl(X \pm Y) = (a_2 \pm b_2)\ \beta + (a_1 \pm b_1)$$

$$Z = c_2\ \beta + c_1 \qquad \ldots\ldots (4.11)$$

Where

$$c_2 = (a_2 \pm b_2) \qquad \ldots\ldots (4.11a)$$

$$c_1 = (a_1 \pm b_1) \qquad \ldots\ldots (4.11b)$$

4.4.3a(i) Floating point normalisation : (definition)

A number is said to be normalised if overflow occurs when the number is multiplied by the base of the exponent.

4.4.3a(ii) Normalisation in floating point addition/ subtraction (zero-exponent)

The process of normalisation in floating point addition/subtraction has been shown with the flow chart in Fig 4.6. The exponent generated during normalisation may be termed as 'generated exponent' and denoted by g. In case of addition nonzero value of g is 1.

4.4.3b Floating point multiplication (zero exponent)

$$Z = Fl\ (X \cdot Y) \qquad \ldots\ldots (4.12)$$

From (4.9) and (4.10)

$$Z = (a_2 \cdot b_2)\beta^2 + (a_2 \cdot b_1)\beta + (a_1 \cdot b_2)\beta + (a_1 \cdot b_1)$$

$$= A_4\beta^2 + A_3\beta + A_2\beta + A_1 \qquad \ldots\ldots (4.13)$$

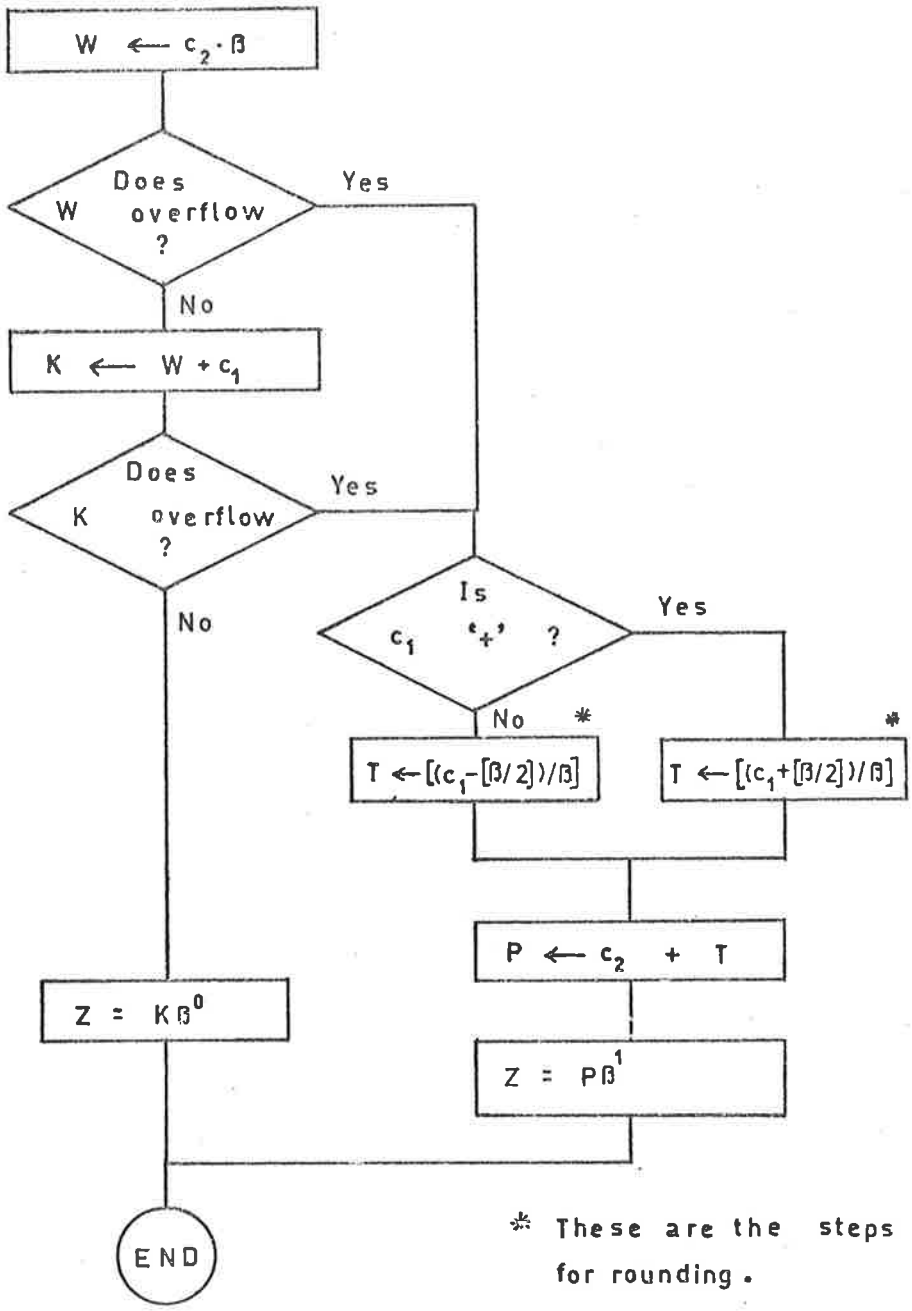


Fig.4.6 Process of normalisation in floating point residue adddition / subtraction .

Where
$$\left.\begin{aligned} A_4 &= a_2 \cdot b_2 \\ A_3 &= a_2 \cdot b_1 \\ A_2 &= a_1 \cdot b_2 \\ A_1 &= a_1 \cdot b_1 \end{aligned}\right\} \quad \ldots\ldots (4.13a)$$

Again Z can be written as

$$Z = (A_4 + D_1 + D_2)\beta^2 + c_2' \beta + c_1' \quad \ldots\ldots (4.13b)$$

Where
$$\left.\begin{aligned} D_1 &= [A_3|\beta] \\ D_2 &= [A_2|\beta] \\ c_2' &= [A_3 - D_1\beta] + [A_2 - D_2\beta] \\ c_1' &= A_1 \end{aligned}\right\} \quad \ldots\ldots (4.13c)$$

Finally $Z = c_3 \beta^2 + c_2 \beta + c_1$ ..... (4.14)

Where
$$\left.\begin{aligned} c_3 &= (A_4 + D_1 + D_2) \\ c_2 &= c_2' + [c_1'|\beta] \\ c_1 &= c_1' - [c_1'|\beta] \cdot \beta \end{aligned}\right\} \quad \ldots\ldots (4.14a)$$

4.4.3b(i) Normalisation in floating point multiplication (zero-exponent)

Figure 4.7 shows the flow chart of normalisation for floating point multiplication. The generated exponent, g, in case of multiplication, ranges from 0 to 2.

4.4.3c Floating point division (zero-exponent)

$$Z = Fl(X|Y) = A|B = A \cdot (1|B) \quad \ldots\ldots (4.15)$$

$$1|B = K\beta^{-x} \quad \text{(say)}$$

Where K is the mantissa (within the range from 0 to R), and x is the exponent. The values of x (except for B = 0 or 1) lie between 2 and 3. The conversion of 1|B to its equivalent floating point number can be done by a look-up table.

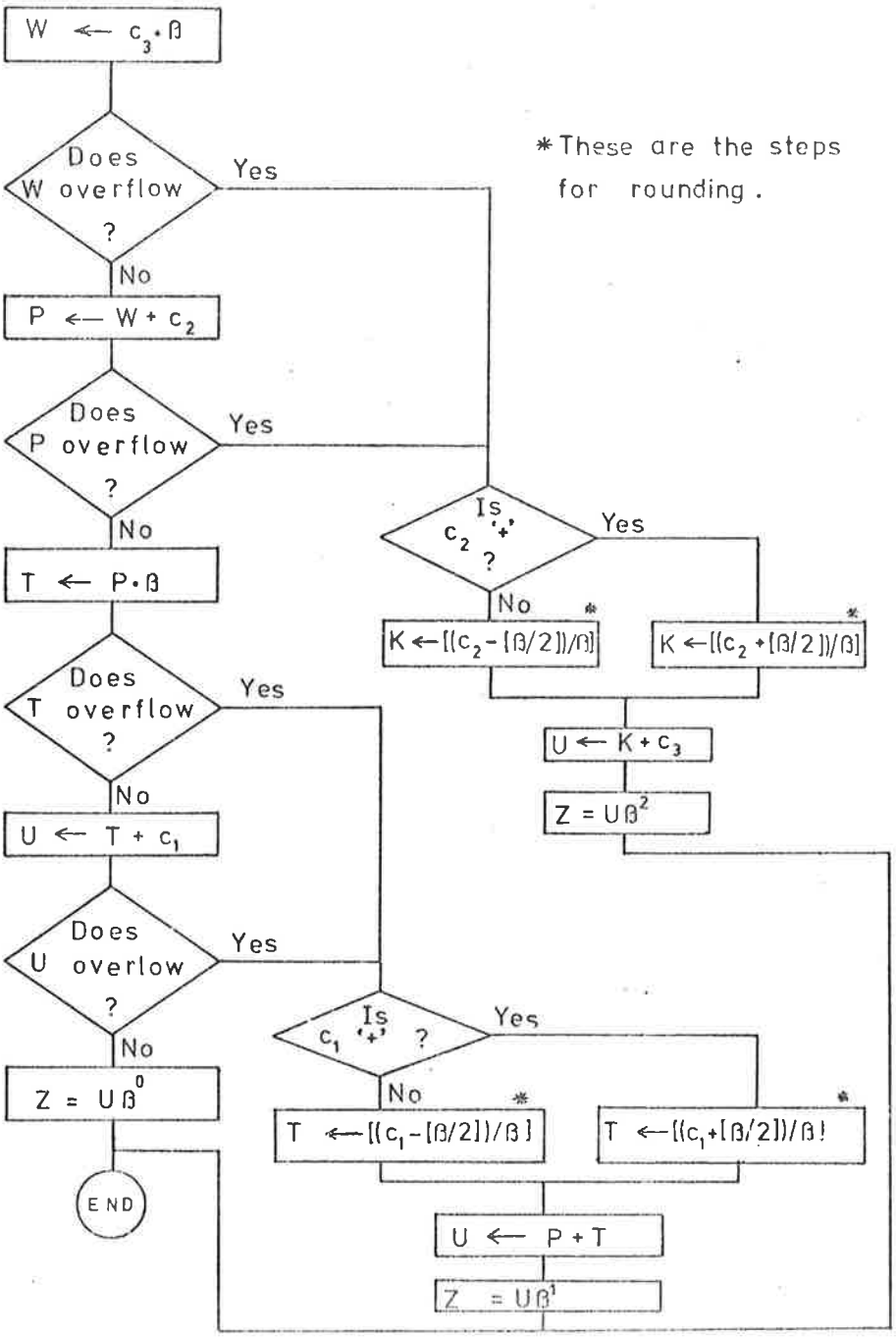


Fig.4.7 Process of normalization in floating point residue multiplication.

[See Appendix-2 for the evaluation of 1|B.]

Therefore $Z = A \cdot K\beta^{-x}$

$= (A \cdot K)\beta^{-x}$ ..... (4.16)

(A · K) can be evaluated by floating point multiplication. That is

$Fl\ (A \cdot K) = V\beta^{y}$ (say) ..... (4.17)

Therefore $Z = V\beta^{y} \cdot \beta^{-x} = V\beta^{y-x} = V\beta^{m}$

..... (4.18)

Where $m = y-x$

4.4.4 Floating point arithmetic with non-zero exponent

Let $X = A\beta^{\delta}$ and $Y = B\beta^{t}$

where δ and t are the exponents of X and Y respectively.

Also X and Y can be written as

$X = (a_2\beta + a_1)\beta^{\delta}$ ..... (4.19)

$Y = (b_2\beta + b_1)\beta^{t}$ ..... (4.19a)

Without loss of generality it can be assumed that

$\delta \geq t$

4.4.4a Algorithms for alignment and floating point addition/subtraction (non-zero exponent)

Let $k = \delta - t$

Case (i) : if k = 0, then

1st step : Compute $D = Fl(A \pm B)$

$= C\beta^{g}$ (say)

2nd step : Compute $p = \delta + g$.

$[Z = C\beta^{p}]$.

END

Case (ii) : If k = 1, then

*1st step : Compute $D = \left[\{b_1 + [\pm\beta|2]\}|\beta\right]$

(If b_1 is negative, add $[-\beta|2]$;
otherwise $[+\beta|2]$)

*2nd step : Compute $D' = a_2 + D$; [The aligned mantissa takes the form of $(0\cdot\beta + D')$.]

3rd step : Compute $T = Fl(A \pm D') = C\beta^{g}$ (say)

4th step : Compute $p = \delta + t$; $[Z = C\beta^{p}]$.

END

Case (iii) : If k = 2, then

*1st step : Compute $D = \left[\{b_1 + [\pm\beta|2]\}|\beta\right]$

[If b_1 is negative, add $[-\beta|2]$;
otherwise $[+\beta|2]$. The aligned mantissa takes the form of $(0\cdot\beta + D)$].

2nd step : Compute $T = Fl(A \pm D) = C\beta^{g}$ (say)

3rd step : Compute $p = \delta + g$; $[Z = C\beta^{p}]$.

END

* (These are the steps for making the aligned mantissa a rounded figure.)

Case (iv) : If $k \geq 3$, then $Z = A\beta^{\delta}$.

4.4.4b Floating point multiplication algorithms (non-zero exponent)

$$Z = Fl(X \cdot Y) = (A \cdot B)\beta^{\delta+t}$$

1st step : Compute $D = Fl(A \cdot B) = C\beta^{g}$ (say)

2nd step : Compute $p = \delta+t+g$; $[Z = C\beta^{p}]$.

END

4.4.4c Floating point division algorithms (non-zero exponent)

$$Z = Fl(X|Y)$$

1st step : Compute $D = Fl(A|B) = C\beta^{g}$ (say)

2nd step : Compute $p = \delta+g-t$; $[Z = C\beta^{p}]$.

END

4.4.5 How to find the co-efficients of a mantissa

The co-efficients of a mantissa A (in residue form) can be obtained by following steps:

1st step : Compute $a_2 = \left[\frac{A}{\beta}\right]$ (residue integer division)

2nd step : Compute $a_1 = A - a_2$ (residue subtraction)

4.4.6 Error in the proposed floating point arithmetic operations

Errors in floating point addition/subtraction, and multiplication are introduced during rounding process (Fig 4.6 and Fig 4.7) in normalisation. In floating point division error may originate from (i) error in approximation of floating point number of $\frac{1}{B}$ and (ii) error in floating point multiplication.

4.4.6a Error in floating point addition/ subtraction

The maximum relative error , Max(e) in floating addition is

$$\text{Max}(e) = \frac{[\beta|2]}{R} \qquad \ldots\ldots (4.20)$$

(Appendix-4 ; (9))

4.4.6b Error in floating point multiplication

The maximum relative error in floating point multiplication, when β is odd, is given by

$$\left(U(eo)\right)_{MAX} = \frac{2([\beta|2])^2 + 3[\beta|2]}{R} \qquad \ldots\ldots (4.21)$$

(Appendix-4 ; (26))

When β is even, the maximum relative error is given by

$$\left(U(ee)\right)_{MAX} = \frac{2([\beta|2])^2}{R} \qquad \ldots\ldots (4.22)$$

(Appendix-4 ; (27))

4.5 Output Translation

The method proposed here requires that $w \leq m_i$ for at least one of the moduli. (w is the base to which conversion is to be done.) However, there is no restriction whether $gcd(w, m_i) = 1$ or not. The method can be applied for conversion to any base provided above condition is satisfied.

4.5.1 Theoretical Aspects

Any integer I_o can be expressed in base w as

$$I_o = a_N w^N + a_{N-1} w^{N-1} + \ldots\ldots + a_i w^i + \ldots\ldots + a_1 w + a_0$$

$$= \sum_{n=o}^{N} a_n \, .$$

The coefficients or digits a_n (n=0,1, ... N) are given by $a_{n-1} = I_{n-1} - wI_n$ for n = 1,2, ..., N

$$\text{where } I_n = \left[\frac{I_{n-1}}{w}\right] .$$

$$\text{Special case } a_N = \left[\frac{I_{N-1}}{w}\right]$$

'N+1' gives the number of digits required to represent the maximum range of a computing system.

4.5.2 Conversion Procedure

The co-efficient (digits) in the base w can be found by repeating the following 3 steps N times.

For n = 1, 2, ..., N

1st step : Find $I_n = \left[\frac{I_{n-1}}{w}\right]$

2nd step : Find $I_{nb} = wI_n$

3rd step : Find $a_{n-1} = I_{n-1} - I_{nb}$

Last step (special) $a_N = \left[\frac{I_{n-1}}{w}\right]$

To convert a residue number to its equivalent decimal number at least one of the moduli must be greater than or equal to 10. The residue of that particular modulus will represent the co-efficient (digit) after third step of the procedure. Therefore only one residue is required to be extracted.

Example:

Let 8, 7 and 11 be the moduli of a residue system. The highest positive and negative numbers that can be handled are $(307)_{10}$ and $(308)_{10}$ respectively which are three digits figures.

Take for an example a residue number (0,5,10) which represents 208 in decimal system.

The procedure to get decimal digits out of the residue number is

(The residue representation of 10 is (2,3,10))

For n = 1

First step : $I_1 = (0,5,10) \,|\, (2,3,10) = (4,6,9) = (20)_{10}$

Second step: $I_{1b} = (4,6,9) \times (2,3,10) = (0,4,2) = (200)_{10}$

Third step : $a_0 = (0,5,10) - (0,4,2) = (0,1,\underset{\uparrow}{8}) = (8)_{10}$

For n = 2

First step : $I_2 = (4,6,9) \,|\, (2,3,10) = (2,2,2) = (2)_{10}$

Second step: $I_{2b} = (2,2,2) \times (2,3,10) = (4,6,9) = (20)_{10}$

Third step : $a_1 = (4,6,9) - (4,6,9) = (0,0,\underset{\uparrow}{0}) = (0)_{10}$

Last step for 3rd digit is

$$a_2 = (2,2,\underset{\uparrow}{2}) \qquad \left[a_N = \left| \frac{I_{N-1}}{w} \right| \right]$$

Therefore the decimal number is 208 $(= a_2 \times 10^2 + a_1 \times 10^1 + a_0 \times 10^0)$

[Division in the first step in each iteration is a general residue division. The arrow indicates the digits in the base 10.]

The arithmetic operations required for conversion are multiplication, division and subtraction. Therefore arithmetic unit must have the provision for subtraction, multiplication and division in the residue number system.

4.6 Conclusion

According to the proposed method, multiplicative overflow detection in 16-15-13-11 moduli system is considerably fast (269 ns). In conventional fast technique, using PROMs of 40 ns access time, the delay would be 640 ns (= 4.4.40). The hardware requirement is not insignificant, however, since same piece of hardware can be shared for sign detection and determination of first approximation, the overall cost will not be much. The delay in obtaining the first approximation, according to the proposed technique is 274 ns (eq 4.3c); whereas in conventional fast division technique, the delay would be 2.4.40 = 320 ns.

Floating point arithmetic which was difficult in residue system, now can easily be implemented using proposed method. The preliminary requirement is the existence of residue integer addition, subtraction, multiplicative, division, comparison and multiplicative overflow detector. Provided circuits for those operations and a look-up table for approximate floating point number of $\frac{1}{B}$ exist, floating point arithmetic operation can be performed in software (residing permanently in PROMs). Therefore implementation of floating point operation will be cost effective. One departure from the conventional floating point format (where 2 or 10 is chosen to be the base of the exponent) is the

choice of the base of the exponent which depends on the moduli of the residue system and will differ from system to system. The main advantage of this choice of the base is that addition/subtraction/multiplication among the co-efficients does not produce overflow. Therefore the cost of using redundant moduli is saved.

Since output translation is a less frequently used operation, more time can be allotted for this operation. Where there exists all the basic residue integer operations, the proposed method for output translation can be implemented with no extra cost of hardware. Moreover, in a situation where base extension cannot be applied, the proposed method is the only choice.

5. DESIGN OF A REAL TIME RESIDUE ARITHMETIC COMPUTER

5.1 Introduction

The aim of the design of the Residue Arithmetic Computer (RAC) is to verify the algorithms proposed in the Chapter 4, and study the feasibility of residue system in a general purpose computer. The functional blocks of the RAC (Fig 5.1) are (i) Intel SDK-80, 8 bit microcomputer (Fig 5.2(a)) and (ii) Residue Arithmetic unit. The microcomputer acquires and manipulates data, while the RAU does all the basic residue arithmetic operations. The design of the RAU, floating point arithmetic and residue to decimal conversion are the main topics of discussion in this chapter.

From the consideration of bit efficiency, moduli 16 and 15 (total 8 bits) have been chosen for the test system. This gives the range of the system from 0 to 239 (M=16x15= 240). The positive half starts from 0 to 119, and negative half from 120 to 239 representing -120 to -1 respectively.

The RAU has been designed on (22.5cm x 17.8cm) veroboard (Fig 5.2(b)) with wire wrapping technique (Fig 5.2(c)). There are altogether fifty IC chips and seventy-six resistors (Fig 5.3 and Table 5.1).

The microcomputer communicates with the RAU through a programmable peripheral interface (PPI) of the microcomputer

(Fig 5.4). The A-port of PPI issues operation codes (Table 5.2) and operands, while B-port receives the results from the RAU. The lower three lines (C0, C1 and C2) of C-port gives the information of the sign of the result (C0), the overflow (C1) and the Exclusive OR of the sign of divisor and dividend (C2), and the higher three lines (C4, C5 and C6) address the latches (Table 5.3 and Fig 5.5) assigned to the operands and operation codes. The operation codes and operands are each eight bits long. Of the eight bits of each operands, the most significant 4 bits represent the residue of module 16 (mod-16) and the least significant 4 bits represent the residue of module 15 (mod-15).

The I/O device specifies operations with alphabetic symbols (Table 5.4) and issues operands to the microcomputer which makes decision on the type of operation (decode), enters the operands and operation code to the RAU and then gets the result from the RAU. The final result is displayed on the I/O device in both residue and decimal forms with overflow character (if any by '?') and sign character ('+' or '-'). In case of comparison, the character > or =, or <, representing X is greater than, or equal to or less than Y, is displayed.

The programmes required for all operations reside in PROMs from locations 400H to 49F H and from 800 H to FEFH, of the SDK-80 microcomputer memory.

Later sections discuss the RAU in detail.

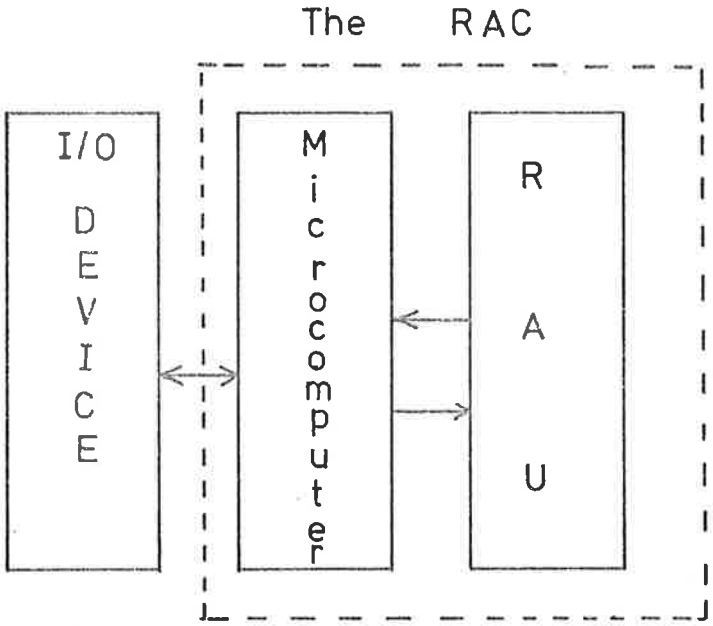


Fig. 5.1 Functional blocks of the RAC .

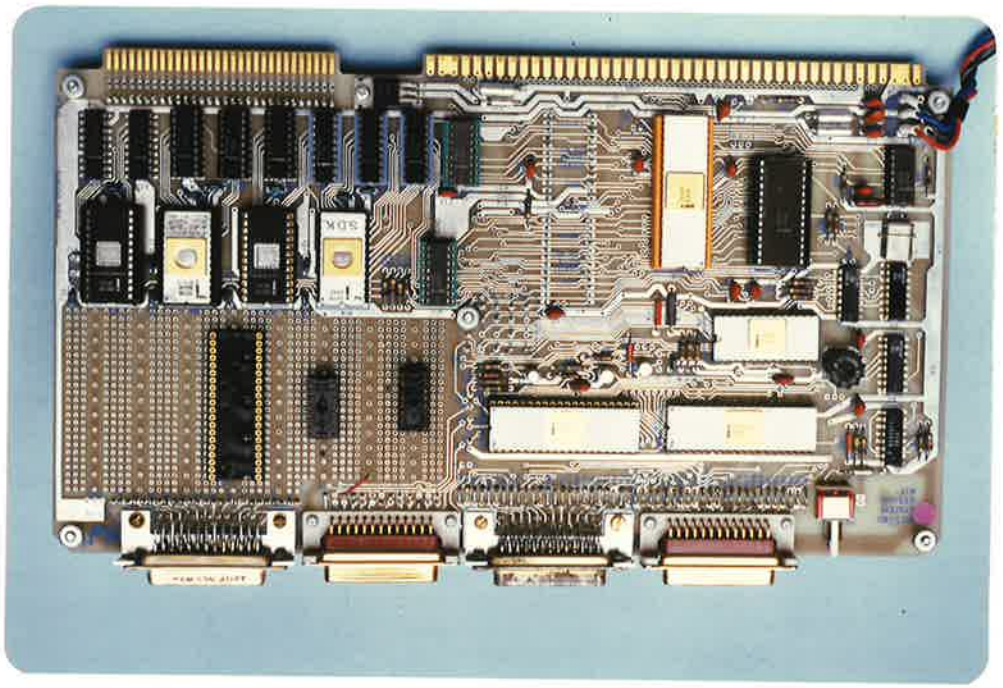


Fig. 5.2 (a) The SDK-80 microcomputer.

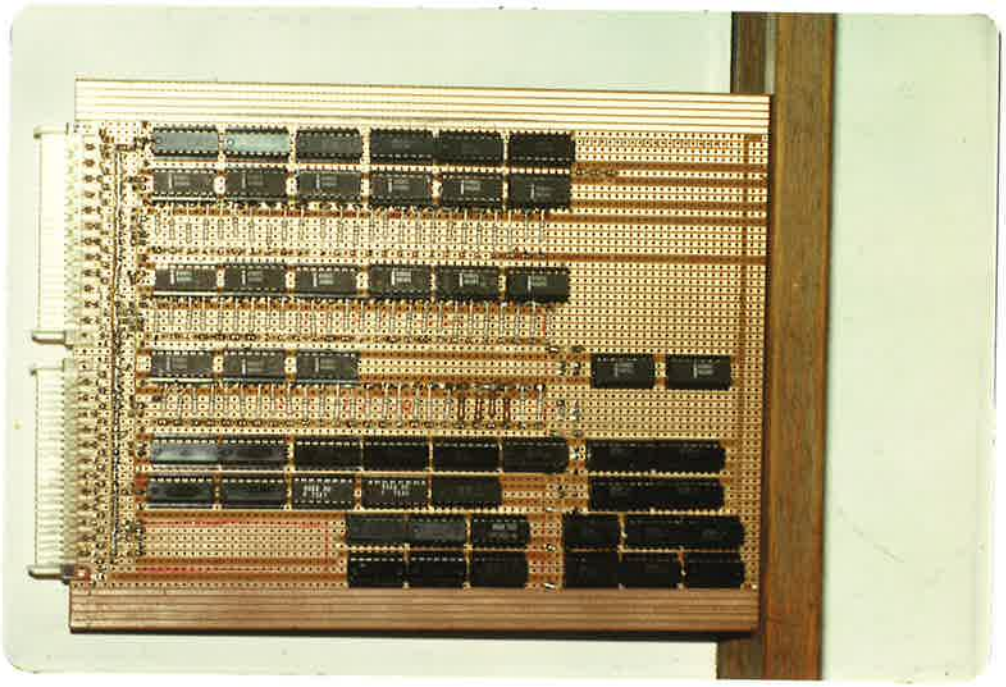


Fig. 5.2(b) Top view of the RAU board.

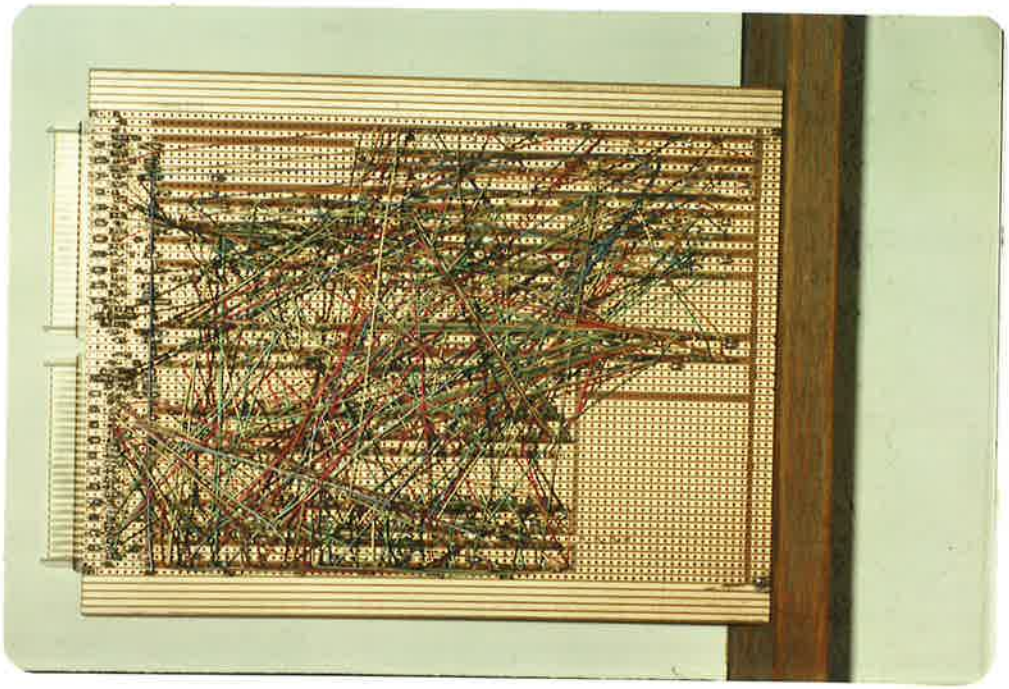


Fig. 5.2(c) Back view of the RAU board.

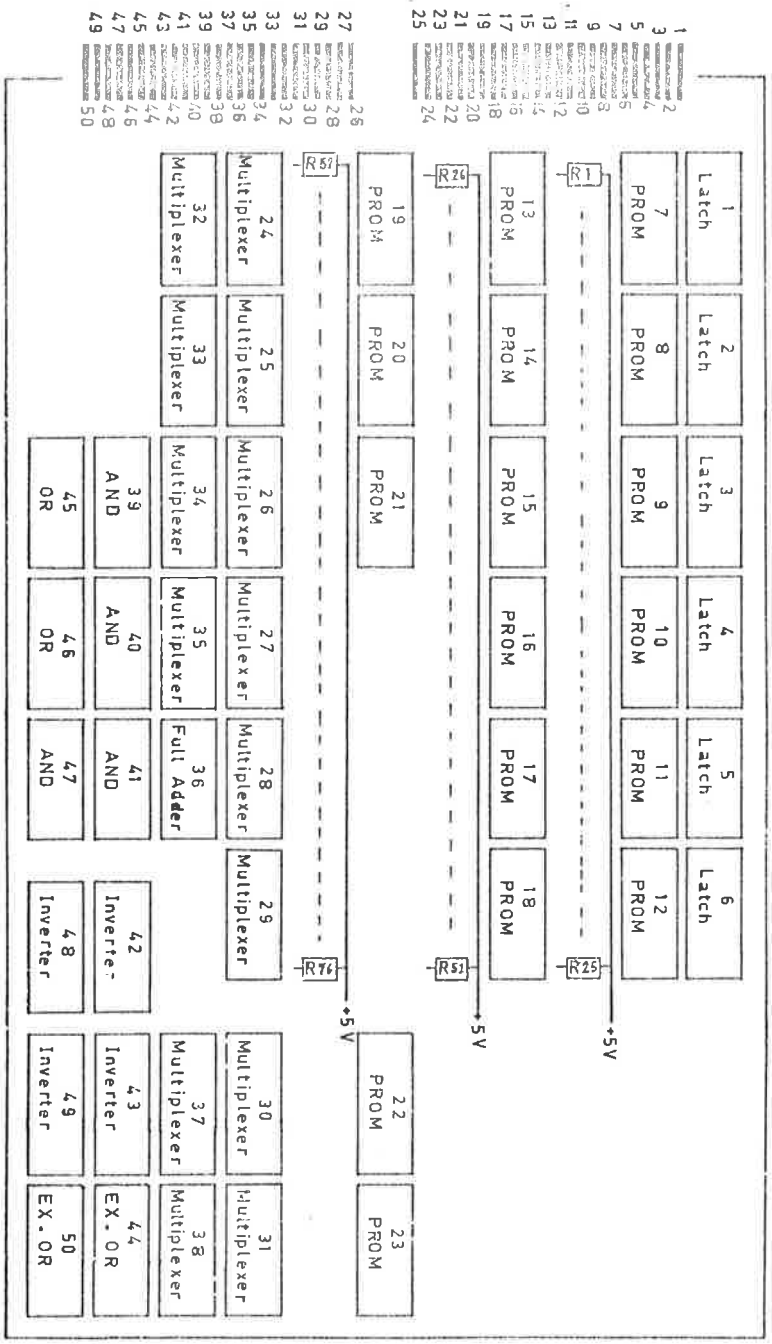


Fig. 5.3 Organisation of components on the RAU board.

Table 5.1

The list of components and their type number/specification

Functional Description of Components	Type Number/ Specification	Location on the RAU Board	Quantity
Quad Latch	DM 7475	(1) — (6)	6
Intel Bipolar Prom [79]	3601	(7) — (23)	17
4 Bit Full ADDER	DM 74LS83	(36)	1
Dual 4:1 Multiplexer	DM 74153	(30),(31),(37),(38)	4
Quad 2-input Multiplexer	9322	(34),(35)	2
8 Channel Multiplexer	DM 74151	(24) — (29), (32),(33)	8
Quad 2-input AND	DM 74H08	(39),(40),(47)	3
Triple 3-input AND	DM 74S11	(41)	1
Quad 2-input OR	DM 7432	(45) — (47),(50)	4
Quad 2-input Exclusive OR	DM 74S86	(43),(44)	2
Hex Inverter	DM 74S04	(42),(48),(49)	3
Resistors	1.8 K Ω (¼W)	R1 — R76	76

[All the I.C. components used, except PROMs, are of National Semiconductor [81].]

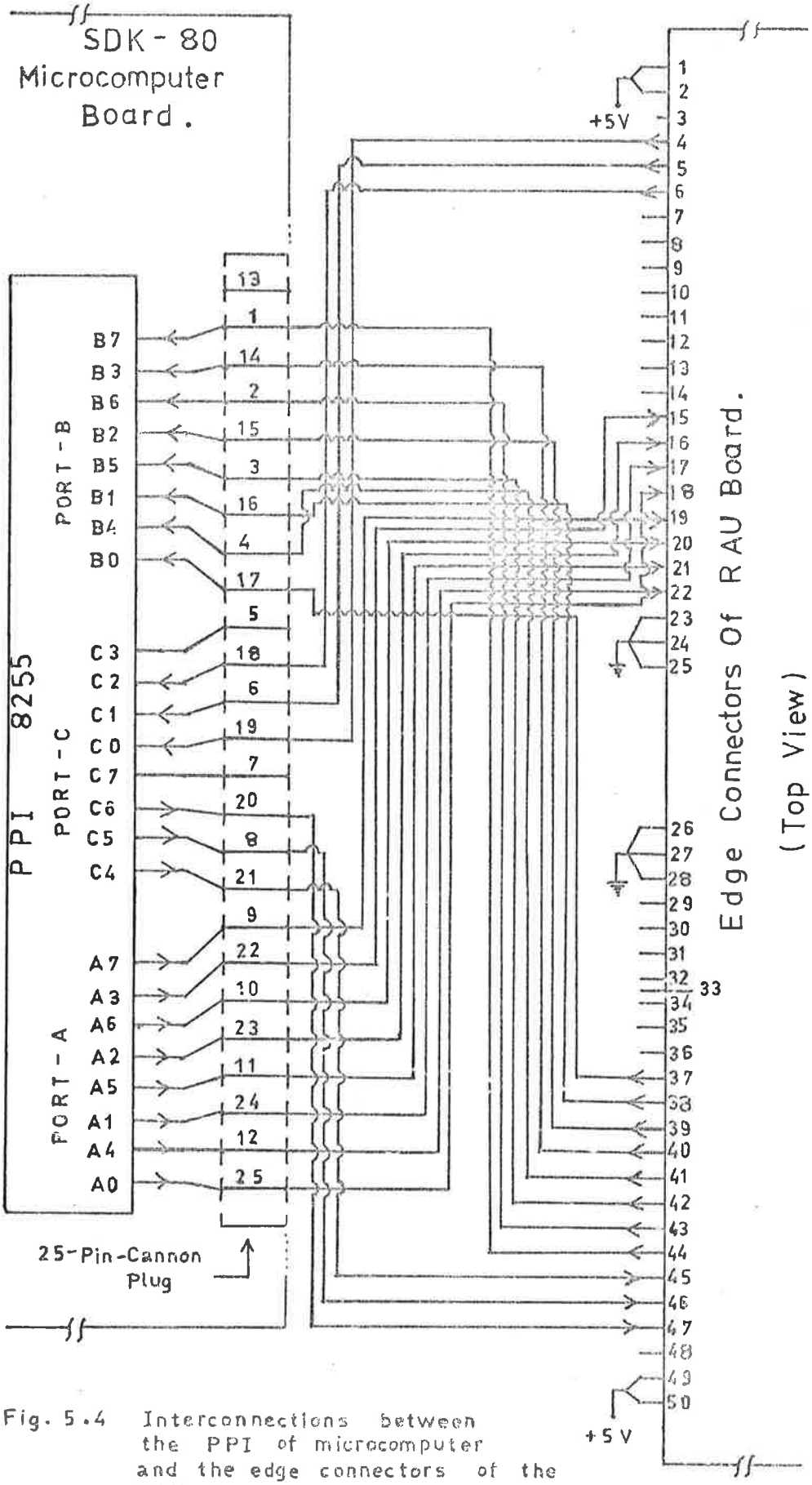


Fig. 5.4 Interconnections between the PPI of microcomputer and the edge connectors of the RAU board.

Table 5.2

Operation Codes and their definition

Operation Codes								Definition
ϕ7,	ϕ6,	ϕ5,	ϕ4,	ϕ3,	ϕ2,	ϕ1,	ϕ0	
0	0	0	0	0	1	1	0	(i) Add
0	0	0	0	0	1	0	1	(ii) Subtract
0	0	0	0	1	0	1	1	(iii) Multiply
0	1	0	0	0	0	0	0	(iv) Compare (Unsigned)
1	0	0	0	0	0	0	0	(v) Compare (Signed)
0	0	0	0	0	0	0	1	(vi) Get First Approximate of Quotient (Integer Division)
0	0	0	0	1	0	0	1	(vii) Get a number divided by 2
0	0	0	0	0	1	1	1	(viii) Get the equivalent mantissa of the reciprocal of divisor (floating point division)
0	0	0	0	0	0	1	0	(ix) Get the exponent of the reciprocal of the divisor

Table 5.3

Addresses of latches

Latches	Address C7,	C6,	C5,	C4,	C3,	C2,	C1,	C0
Latch of X	0	1	1	0	0	0	0	0
Latch of Y	0	1	0	1	0	0	0	0
Latch of Operation Code	0	0	1	1	0	0	0	0

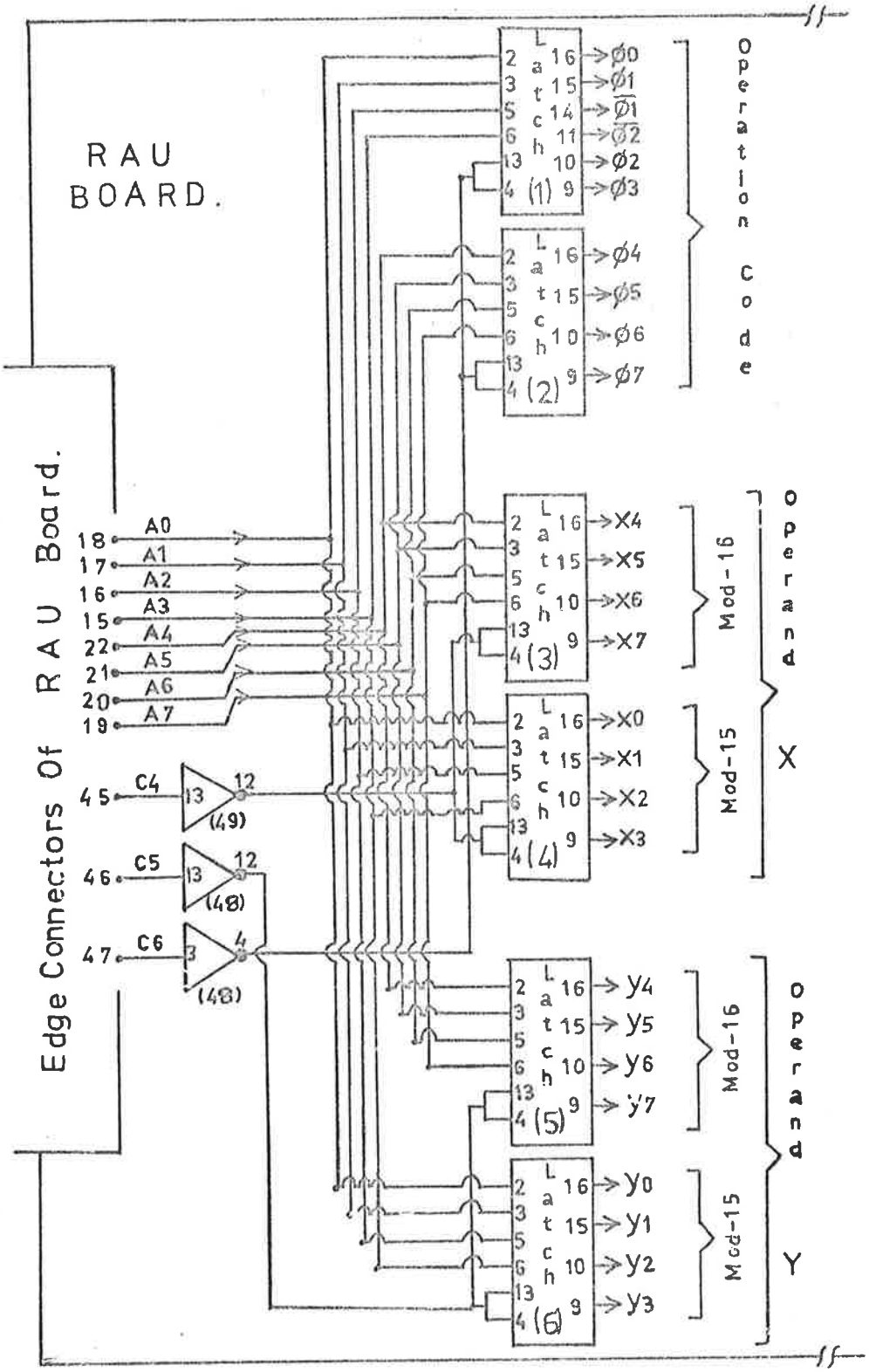


Fig. 5.5 Latching of operation code and operands.

Table 5.4

Alphabetic symbols and operations

Alphabet	Operation
A	Addition (Integer)
B	Subtraction (Integer)
C	Multiplication (Integer)
D	Division (Integer)
E	Comparison (unsigned, Integer)
F	Comparison (signed, Integer)
G	Addition (Floating Point)
H	Subtraction (Floating Point)
I	Multiplication (Floating Point)
J	Division (Floating Point)

5.2 Addition, Subtraction and Multiplication (Integer)

Addition, subtraction and multiplication are performed simultaneously by means of six look up tables (Tables 5.5(a), 5.5(b), 5.5(c), 5.5(d), 5.5(e) and 5.5(f)) stored in six (256x4) Intel Bipolar PROM (3601), each having a 70 ns access time. Two PROMs are assigned to each operation; one is for mod-16 and the other is for mod-15 operations (Fig 5.6(a), 5.6(b) and 5.6(c)).

The lower 4 address lines (pins 5,6,7 and 4) are connected to the 4 data lines of operand X; while the higher 4 address lines (pins 3,2,1 and 15) are connected to the 4 data lines (corresponding residue) of operand Y.

[The program listings have been shown in Appendix-6; AP 6(a).]

Table 5.5(a)

Look up Table (Addition in Mod-16)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1
3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
4	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3
5	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4
6	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5
7	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
A	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9
B	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A
C	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B
D	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C
E	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D
F	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E

Table 5.5(b)

Look up Table (Addition in Mod-15)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0	1
2	2	3	4	5	6	7	8	9	A	B	C	D	E	0	1	2
3	3	4	5	6	7	8	9	A	B	C	D	E	0	1	2	3
4	4	5	6	7	8	9	A	B	C	D	E	0	1	2	3	4
5	5	6	7	8	9	A	B	C	D	E	0	1	2	3	4	5
6	6	7	8	9	A	B	C	D	E	0	1	2	3	4	5	6
7	7	8	9	A	B	C	D	E	0	1	2	3	4	5	6	7
8	8	9	A	B	C	D	E	0	1	2	3	4	5	6	7	8
9	9	A	B	C	D	E	0	1	2	3	4	5	6	7	8	9
A	A	B	C	D	E	0	1	2	3	4	5	6	7	8	9	A
B	B	C	D	E	0	1	2	3	4	5	6	7	8	9	A	B
C	C	D	E	0	1	2	3	4	5	6	7	8	9	A	B	C
D	D	E	0	1	2	3	4	5	6	7	8	9	A	B	C	D
E	E	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0

Table 5.5(c)

Look up Table (Subtraction in Mod-16)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
2	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D
3	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C
4	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B
5	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A
6	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9
7	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6
A	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5
B	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4
C	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3
D	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
E	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1
F	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0

Table 5.5(d)

Look up Table (Subtraction in Mod-15)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	E	D	0
1	E	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
2	D	E	0	1	2	3	4	5	6	7	8	9	A	B	C	D
3	C	D	E	0	1	2	3	4	5	6	7	8	9	A	B	C
4	B	C	D	E	0	1	2	3	4	5	6	7	8	9	A	B
5	A	B	C	D	E	0	1	2	3	4	5	6	7	8	9	A
6	9	A	B	C	D	E	0	1	2	3	4	5	6	7	8	9
7	8	9	A	B	C	D	E	0	1	2	3	4	5	6	7	8
8	7	8	9	A	B	C	D	E	0	1	2	3	4	5	6	7
9	6	7	8	9	A	B	C	D	E	0	1	2	3	4	5	6
A	5	6	7	8	9	A	B	C	D	E	0	1	2	3	4	5
B	4	5	6	7	8	9	A	B	C	D	E	0	1	2	3	4
C	3	4	5	6	7	8	9	A	B	C	D	E	0	1	2	3
D	2	3	4	5	6	7	8	9	A	B	C	D	E	0	1	2
E	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0	1
F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0

Table 5.5(e)

Look up Table (Multiplication in Mod-16)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	0	2	4	6	8	A	C	E
3	0	3	6	9	C	F	2	5	8	B	E	1	4	7	A	D
4	0	4	8	C	0	4	8	C	0	4	8	C	0	4	8	C
5	0	5	A	F	4	9	E	3	8	D	2	7	C	1	6	B
6	0	6	C	2	8	E	4	A	0	6	C	2	8	E	4	A
7	0	7	E	5	C	3	A	1	8	F	6	D	4	B	2	9
8	0	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8
9	0	9	2	B	4	D	6	F	8	1	A	3	C	5	E	7
A	0	A	4	E	8	2	C	6	0	A	4	E	8	2	C	6
B	0	B	6	1	C	7	2	D	8	3	E	9	4	F	A	5
C	0	C	8	4	0	C	8	4	0	C	8	4	0	C	8	4
D	0	D	A	7	4	1	E	B	8	5	2	F	C	9	6	3
E	0	B	C	A	8	6	4	2	0	E	C	A	8	6	4	2
F	0	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1

Table 5.5(f)

Look up Table (Multiplication in Mod-15)

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	0
2	0	2	4	6	8	A	C	E	1	3	5	7	9	B	D	0
3	0	3	6	9	C	0	3	6	9	C	0	3	6	9	C	0
4	0	4	8	C	1	5	9	D	2	6	A	E	3	7	B	0
5	0	5	A	0	5	A	0	5	A	0	5	A	0	5	A	0
6	0	6	C	3	9	0	6	C	3	9	0	6	C	3	9	0
7	0	7	E	6	D	5	C	4	B	3	A	2	9	1	8	0
8	0	8	1	9	2	A	3	B	4	C	5	D	6	E	7	0
9	0	9	3	C	6	0	9	3	C	6	0	9	3	C	6	0
A	0	A	5	0	A	5	0	A	5	0	A	5	0	A	5	0
B	0	B	7	3	E	A	6	2	D	9	5	1	C	8	4	0
C	0	C	9	6	3	0	C	9	6	3	0	C	9	6	3	0
D	0	D	B	9	7	5	3	1	E	C	A	8	6	4	2	0
E	0	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

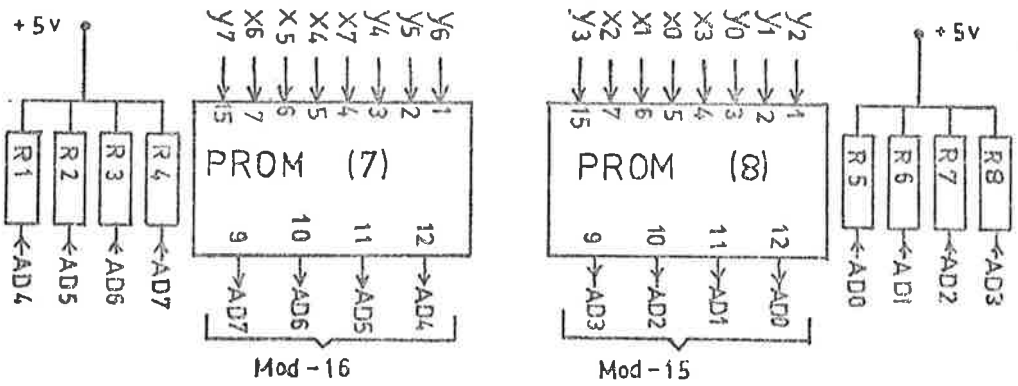


Fig. 5.6(a) Circuit for the 16-15 moduli addition.

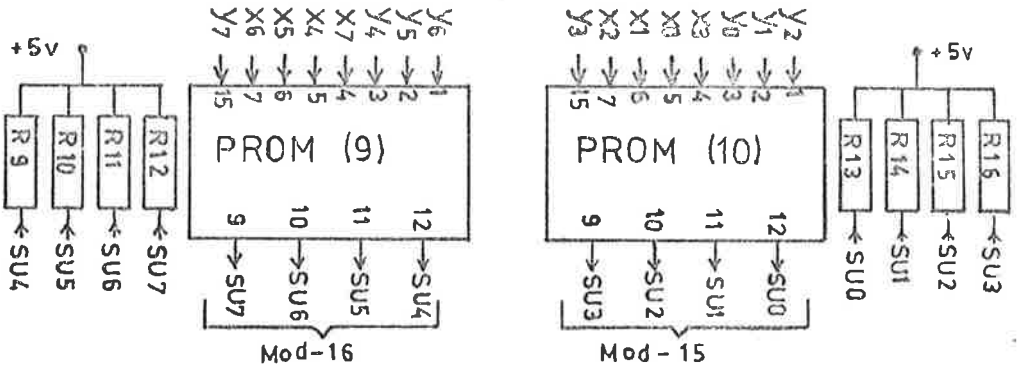


Fig. 5.6(b) Circuit for the 16-15 moduli subtraction.

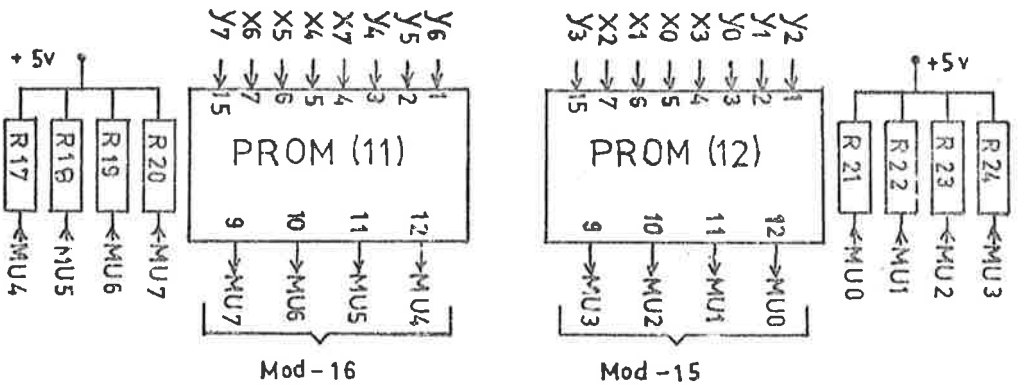


Fig. 5.6(c) Circuit for the 16-15 moduli multplication.

5.2 (a) Delay in Addition, Subtraction and Multiplication

The delay in addition/subtraction/multiplication

= 1 PROM delay

= 70 ns (access time of 3601 PROM)

5.3 Comparison

There are two types of comparisons - comparisons with signed numbers and comparisons with unsigned numbers. Comparisons with signed numbers involve either no operation or subtraction; whereas comparisons with unsigned numbers involve either addition or subtraction. Table 5.6 is a truth table showing the comparisons, operations required and decision modes. It is clear from the Table 5.6 that subtraction is required (in both types of comparisons) if the signs of X and Y agree; while addition is required (comparison with usnigned number) if the signs of X and Y do not agree.

Table 5.6

Truth table for decision modes in comparison

Signed Numbers	Unsigned Numbers	Sign of X	Sign of Y	Operation	Zero Result	Sign of the result	Decision
1	0	0	0	SUB.	0	0	X > Y
1	0	0	0	"	0	1	X < Y
1	0	0	0	"	1	0	X = Y
1	0	0	1	No Op.	-	-	X > Y
1	0	1	0	No Op.	-	-	X < Y
1	0	1	1	SUB.	0	0	X > Y
1	0	1	1	"	0	1	X < Y
1	0	1	1	"	1	0	X = Y
0	1	0	0	"	0	0	X > Y
0	1	0	0	"	0	1	X < Y
0	1	0	0	"	1	0	X = Y
0	1	0	1	ADD.	0	0	X > Y
0	1	0	1	"	0	1	X < Y
0	1	0	1	"	1	0	X = Y
0	1	1	0	"	0	0	X < Y
0	1	1	0	"	0	1	X > Y
0	1	1	0	"	1	0	X = Y
0	1	1	1	SUB.	0	0	X < Y
0	1	1	1	"	0	1	X > Y
0	1	1	1	"	1	0	X = Y

[Note: SUB = Subtraction; ADD = Addition]

Comparisons require the sign of X, Y and the sign of the result of addition or subtraction. However, the results of addition, subtraction and multiplication are generated simultaneously. Therefore, depending on signs of X and Y, the result of addition/subtraction is to be selected for sign detection.

The sign is detected by a look up table (Table 5.7) implemented with a PROM. There are altogether three sign detectors; one is for X, one for Y and the other one for the result (one of the results of addition, subtraction and multiplication).

Fig 5.7(a) and 5.7(b) show the circuits used for sign detection of X and Y respectively. The data lines of the residue of mod-16 are connected to the lower 4 address lines of the PROMs and that of mod-15 to the higher 4 address lines. The most significant bit of the output indicates the sign ('1' for negative, '0' for positive) and the least significant bit indicates zero/nonzero of the number ('1' for zero-number). Thus SX and ZX give the information sign and zero/nonzero of X respectively. Similarly so for Y.

The appropriate result is multiplexed to the sign detector by two bits ($\phi 3$, $\overline{\phi 1}$) of the respective operation code (Fig 5.8). However, $\overline{\phi 1}$ is not connected directly to the selection pin (pin-14) of the multiplexers (two dual 4:1 multiplexers), since in case of comparison, the result of either addition or subtraction is to be selected. Table 5.8 shows the states of the selection pins and the selected inputs.

Table 5.7

Look up table for the sign and zero-nonzero of a number

Each hexadecimal digit represents 'lower 4 address bits'

Each hexadecimal digit represents 'higher 4 address bits'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0
1	0	0	8	8	8	8	8	8	8	8	0	0	0	0	0	0
2	0	0	0	8	8	8	8	8	8	8	8	0	0	0	0	0
3	0	0	0	0	8	8	8	8	8	8	8	8	0	0	0	0
4	0	0	0	0	0	8	8	8	8	8	8	8	8	0	0	0
5	0	0	0	0	0	0	8	8	8	8	8	8	8	8	0	0
6	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8	0
7	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8
8	8	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8
9	8	8	0	0	0	0	0	0	0	0	8	8	8	8	8	8
A	8	8	8	0	0	0	0	0	0	0	0	8	8	8	8	8
B	8	8	8	8	0	0	0	0	0	0	0	0	8	8	8	8
C	8	8	8	8	8	0	0	0	0	0	0	0	0	8	8	8
D	8	8	8	8	8	8	0	0	0	0	0	0	0	0	8	8
E	8	8	8	8	8	8	8	0	0	0	0	0	0	0	0	8
F	8	8	8	8	8	8	8	8	0	0	0	0	0	0	0	0

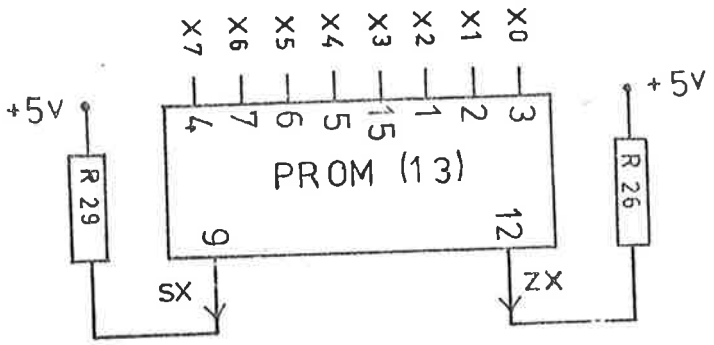


Fig. 5.7(a) Circuit for sign detection of X in the 16-15 residue system.

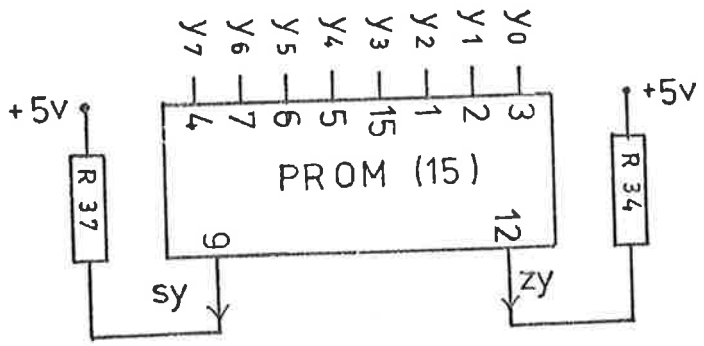


Fig. 5.7(b) Circuit for sign detection of Y in the 16-15 residue system.

Table 5.8

The states of the selection pins and corresponding selected inputs

States of Selected Pins 2, 14		Selected Inputs	'ϕ3' and '$\overline{\phi 1}$' (inverted 'ϕ1') of operation codes (Table 5.2) 'ϕ3', 'ϕ1'	
0	0	Addition	0	0
0	1	Subtraction	0	1
1	0	Multiplication	1	0
1	1	Not used		

From the Table 5.8 it is clear that 'ϕ3' and '$\overline{\phi 1}$' can be directly connected to the selection pins 2 and 14 respectively. However, since in comparisons the sign of the result of addition (if the signs of X and Y do not agree) / subtraction (if the signs of X and Y agree) is required and the 'ϕ3' bit of addition, subtraction and comparison codes (Table 5.2) is same (zero), only 'ϕ3' bit is directly connected; whereas '$\overline{\phi 1}$' is gated through the Exclusive OR (44, 1,2,3) to change conditionally the logic state of 'ϕ1' to select the result of addition/subtraction for sign detection during comparison.

For addition, subtraction and multiplication 'ϕ7' and 'ϕ6' of the operation codes (Table 5.2) are zeros; therefore the output of the AND gate (39,2,1,3) is '0'. The '$\overline{\phi 1}$' passes out through the Exclusive OR (44,1,2,3) without inversion. In case of comparison, subtraction and division, the '$\overline{\phi 1}$' which is the input to the Exclusive ORs (44,5,4,6) and (44,1,2,3) is '1'. The output from the Exclusive OR (44,5,

4,6) is therefore the inverted SY. The actual Exclusive OR of the signs of X and Y (SX,SY) is the output from the inverter (48,1,2), since '$\overline{SC}$' = $\overline{1 \oplus SY \oplus SX} = \overline{SY \oplus SX} = SY \oplus SX$. Therefore the '$\overline{SC}$' is also the Exclusive OR of the signs of divisor and dividend. In comparisons, the output of the OR gate (45,1,2,3) is '1'. If the signs of the X and Y does not agree '$\overline{SC}$' will be '1' and the output of the AND gate will be '1' which inverts '$\overline{\phi 1}$' (='1' in comparison) to '0'. Therefore the states of the selection pins 2 and 14 are '0' and '0' respectively and the result of addition will be selected. However, if the signs of X and Y agrees '$\overline{SC}$' will be '0' and so the output from the AND. Therefore the output of the Exclusive OR (44,1,2,3) will be '$\overline{\phi 1}$' (='1' in comparison); the states of the selection pines 2 and 14 are '0' and '1' respectively and the result of subtraction will be selected for sign detection.

The outputs from the multiplexers are connected to the sign detector (PROM (14)). 'SZ' gives the sign of the result (addition/subtraction/multiplication) Z and ZZ gives the information of zero/nonzero of the result.

[Note that the Exclusive OR (44,5,4,6) and the inverter (48, 1,2) could be omitted by connecting SY to the pin 10 of the Exclusive OR (44,10,9,8) and the output from it ('SC') to the pin 2 of the AND gate. However, it has not omitted, since this circuit is also part of overflow detector (Fig 5.10) where the sign of the subtrahend is to be inverted by '$\overline{\phi 1}$' (='1' in subtraction) for subtractive overflow detection].

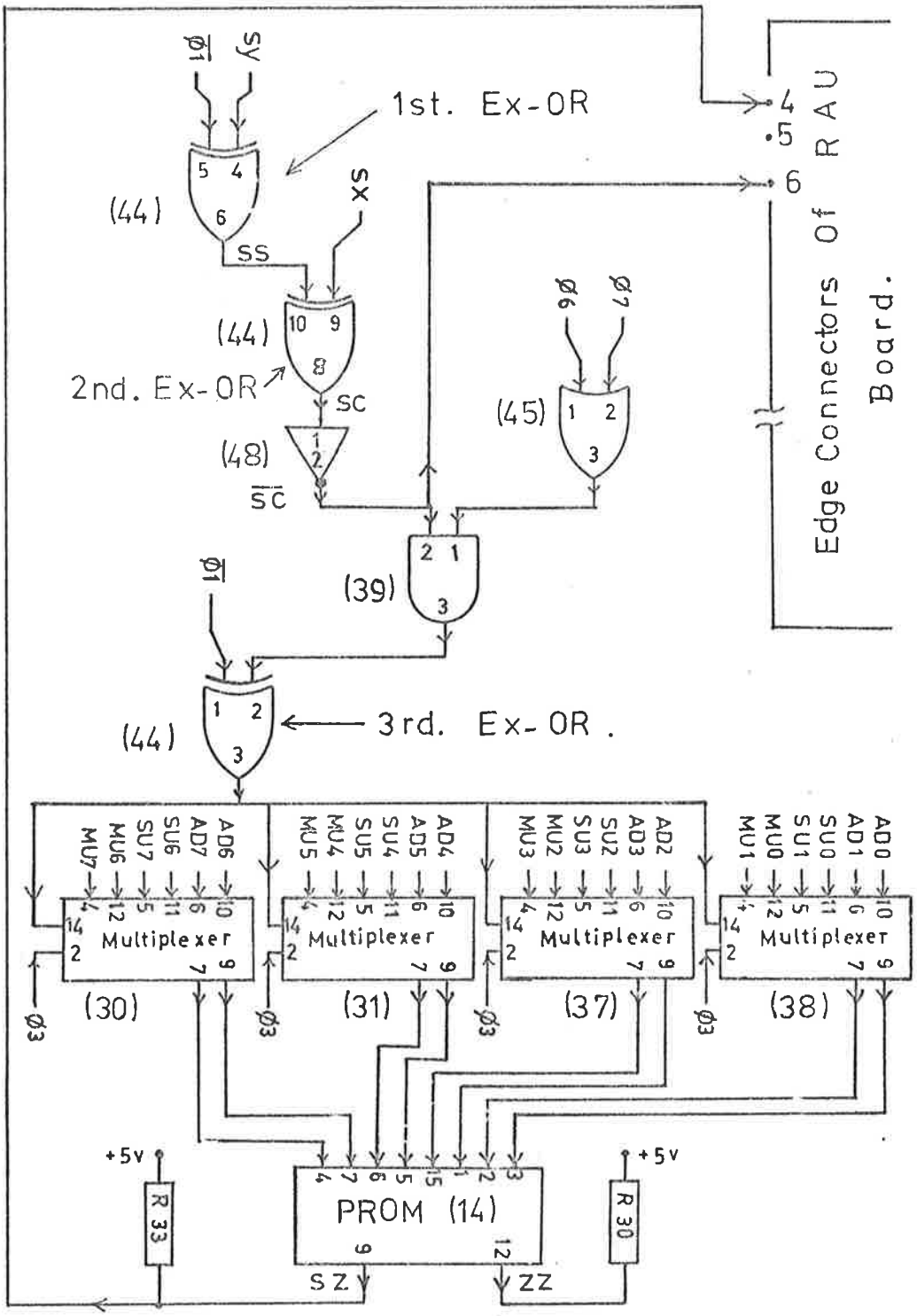


Fig. 5.8 Circuit for sign detection of the result of addition/subtraction/multiplication in the 16-15-residue system.

The final decision in comparison is done with a look up table (Table 5.9) formed in the PROM (23) (Fig 5.9). Altogether 32 locations (locations DOH to DFH and FOH to FFH) are required. The contents of the odd locations give the decision modes of comparison with signed numbers, whereas the contents of the even locations give the decision modes of comparison with unsigned numbers. The 'ϕ7' bit of the operation code determines whether the locations read are even or odd. If 'ϕ7' is '1', the odd locations are selected; otherwise the even locations.

The output Cϕ0 = '1' indicates X is less than Y; Cϕ1 = '1' indicates X is greater than Y; while Cϕ2 = '1' indicates X is equal to Y.

Table 5.9

Look up table for decision modes in comparison

4 bits of each hexadecimal digit represent 'SY', 'SZ', 'ZZ' and 'ϕ7'

The second bit of each hexadecimal digit represents 'SX'		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	D	2	2	4	4	1	1	0	0	2	2	2	4	2	1	0	0
	F	1	1	1	4	1	2	0	0	2	1	4	4	1	2	0	0

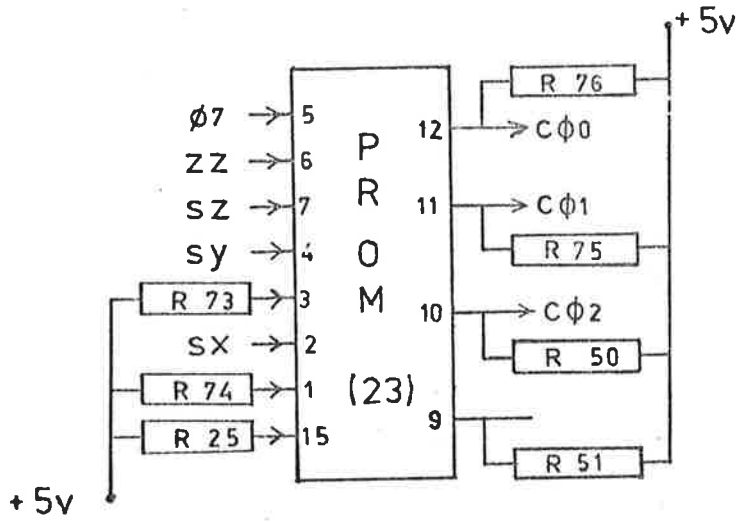


Fig. 5.9 Circuit for decision modes in comparison in 16-15-residue system.

5.3(a) Delay in Comparison

From Fig 5.8 and 5.9,

Delay in comparison = 1 PROM delay for operation of addition/subtraction.
+ 1 Multiplexer delay
+ (3 Exclusive OR + 1 inverter + 1 AND) delay.
+ 1 PROM delay for sign detection
+ 1 PROM delay for decision modes.

= (70+20+3x7+10+9+70+70) ns

= 270 ns (for the elements used in the test facilities).

5.4 Overflow Detection

Fig 5.10 shows the circuit for overflow detection in addition, subtraction and multiplication. The circuits in the dash-lined box detect additive/subtractive overflow; while the rest of the circuit is needed to detect multiplicative overflow.

5.4(a) Multiplicative Overflow Detection

When multiplicative overflow occurs the product may be of incorrect sign or may be outside the range of the system.

An overflow characterised by incorrect sign is detected by the circuit shown in the solid-lined box. The signal '*SC*' is the Exclusive OR of the signs of *X* and *Y* (Fig 5.8). In Fig 5.8, '$\overline{\phi 1}$' is '0' for multiplication and addition. Therefore the output '*SS*' from the Exclusive OR (44,5,4,6) is really *SY* (sign of *Y*) and the output '*SC*' from the Exclus-

ive OR (44,10,9,8) is the Exclusive OR of 'SY' and 'SX' (the signs of X and Y). Table 5.10 is the truth table of overflow characterised by incorrect sign.

Table 5.10

Truth table of overflow characterised by incorrect sign

Sign of X, SX	sign of Y, SY	sign of Z, SZ	Overflow
0	0	0	0
0	0	1	1
0	1	0	1
0	1	0	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

From the Table 5.10 the overflow $\mathcal{O}(\phi')$ is given by

$$\mathcal{O}(\phi') = \overline{SX}.\overline{SY}.SZ + \overline{SX}.SY.\overline{SZ} + SX.\overline{SY}.\overline{SZ} + SX.SY.SZ$$

$$= SX \oplus SY \oplus SZ \qquad \ldots\ldots (5.1)$$

From Fig 5.8 $\quad SC = '\overline{\phi 1}' \oplus 'SY' \oplus 'SX' \qquad \ldots\ldots (5.2)$

For multiplication and addition, '$\overline{\phi 1}$' is '0', therefore

$$SC = SY \oplus SX.$$

The expression (5.1) becomes

$$\mathcal{O}(\phi') = SC \oplus SZ \qquad \ldots\ldots (5.3).$$

In Fig 5.10, the output from the Exclusive OR (43,1, 2,3) is the Exclusive OR of 'SC' and 'SZ'. However, a false signal of overflow occurs when a negative number is multiplied by zero. The zero operands are detected by the inverters (49,3,4) and (49,5,6) and AND gate (39,6,7,8). If one of the operands is zero the output from the AND gate (39,6,7,8) will be '0'; so is the output from the 3-input AND gate (41,5,3,4,6) ($\overline{\phi 1}$ is '1' for multiplication). This means no overflow indication will be given.

The circuit outside the boxes (Fig 5.10) detects overflow by 'group technique'. The PROMs (18) and (16) form look up tables (Table 5.12) of group numbers of Y and X respectively (where X and Y are the numbers from 0 to +119 and -1 to -120) corresponding to the Table 5.11. (The number of groups is 8, since $2^7 < 240 < 2^8$ and $w = 16$.)

Table 5.11

Table of numbers and their corresponding group number

Numbers	Group Number
0	0
1	1
2 ⟶ 3	2
4 ⟶ 7	3
8 ⟶ 15	4
16 ⟶ 31	5
32 ⟶ 63	6
64 ⟶ 120	7

Table 5.12

Look up table of group number of X or Y

Each hexadecimal digit represents the residue of Mod-16

Each hexadecimal digit represents the residue of mod-15.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	4	5	6	6	7	7	7	7	7	7	7	6	6	5	4
1	5	1	4	5	6	6	7	7	7	7	7	7	7	6	6	5
2	6	5	2	4	5	6	6	7	7	7	7	7	7	7	6	6
3	6	6	5	2	4	5	6	6	7	7	7	7	7	7	7	6
4	7	6	6	5	3	4	5	6	6	7	7	7	7	7	7	7
5	7	7	6	6	5	3	4	5	6	6	7	7	7	7	7	7
6	7	7	7	6	6	5	3	4	5	6	6	7	7	7	7	7
7	7	7	7	7	6	6	5	3	4	5	6	6	7	7	7	7
8	7	7	7	7	7	6	6	5	4	3	5	6	6	7	7	7
9	7	7	7	7	7	7	6	6	5	4	3	5	6	6	7	7
A	7	7	7	7	7	7	7	6	6	5	4	3	5	6	6	7
B	7	7	7	7	7	7	7	7	6	6	5	4	3	5	6	6
C	6	6	7	7	7	7	7	7	7	6	6	5	4	2	5	6
D	6	6	6	7	7	7	7	7	7	7	6	6	5	4	2	5
E	5	6	6	6	7	7	7	7	7	7	7	6	6	5	4	1
F	0	4	5	6	6	7	7	7	7	7	7	7	6	6	5	4

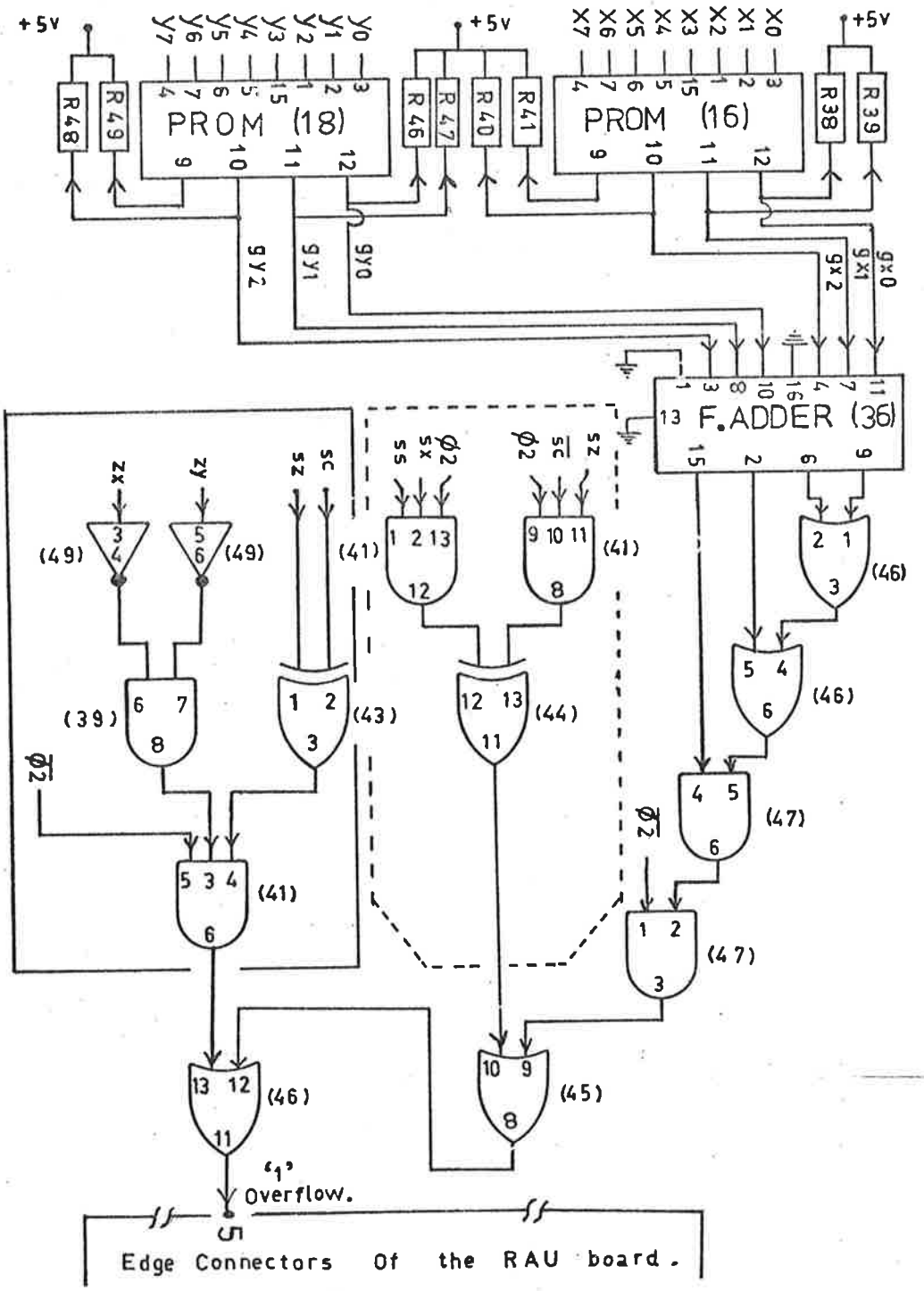


Fig. 5.10 Circuit for overflow detection in the 16-15 residue system .

If the sum of the group numbers of X and Y is greater than 8, overflow must occur, if less than 8 overflow does not occur, if equal to 8 overflow may or may not occur, but if overflow occurs, the sign will be incorrect.

The group numbers are added with the Full Adder. If the sum exceeds 8, the M.S.B. (at Pin 15) will be '1' and at least one of other three bits will be nonzero. The first two OR gates (46,2,1,3) and (46,5,4,6) detect whether the least significant three bits is nonzero. If it is nonzero and the M.S.B. is '1', the output from the AND gate (47,5,6,6) will be '1' i.e. overflow has occurred.

In multiplication '$\phi 2$' is '0'. This means that the output from the additive/subtractive overflow detectors (dash-lined box) is '0'. However, '$\overline{\phi 1}$' opens the AND gate (47,1,2,3). The output from the OR gate (46,13,12,11) gives the information of multiplicative overflow ('1' indicates overflow).

5.4(b) Additive and Subtractive Overflow Detection

After inverting the sign of the subtrahend, the subtractive overflow is same as the additive overflow detection. The inversion is done by the Exclusive OR (44,5,4,6) of Fig 5.8. In subtraction '$\overline{\phi 1}$' is '1'; therefore the output 'SS' = $\overline{SY}$. In addition '$\overline{\phi 1}$' is '0'; therefore the output is 'SS' = SY. An additive overflow is said to occur if the sign of the operands are same and the sign of result is incorrect. The Table 5.13 is a truth table of additive/subtractive overflow decision modes.

Table 5.13

Truth table of additive/subtractive overflow decision modes

'SS' = SV/$\overline{SV}$	Sign of X, SX	Sign of Z, SZ	Overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

From the truth table the additive overflow $\delta(\phi)$ can be expressed as

$$\delta(\phi) = 'SS'.\ \overline{SX}.\ SZ + 'SS'.\ SX.\ \overline{SZ} \quad \ldots\ldots (5.4)$$

This function is to be ANDed with '$\phi 2$' to signify that the overflow is due to addition/subtraction. Therefore the actual expression becomes

$$\delta(0) = '\phi 2'.\ \delta(\phi) = '\phi 2'.\ ('SS'.\ \overline{SX}.\ SZ + 'SS'.\ SX.\ \overline{SZ}) \quad \ldots\ldots (5.5)$$

The implementation of the expression (5.5) needs - three inverters, two 3-input AND gates, one 2-input OR gate and one 2-input AND gate. Assuming 'SS' exists, the total delay, according to the delay characteristics of the components used in the test circuit, is --- One 2-input AND gate delay (8·8 ns) + One 3-input AND gate delay (4·75 ns) + One 2-input OR

gate delay (12 ns) ≃ 25 ns.

The actual circuit of additive/subtractive overflow is shown in the dash-lined box of Fig 5.10. The logic expression $\delta(A)$ of the circuit is

$$\delta(A) = (\text{'}\phi 2\text{'}.\ SX.\ \text{'}SS\text{'}) \oplus (\text{'}\phi 2\text{'}.\ \overline{SC}.\ SZ) \qquad \ldots\ldots (5.6)$$

where $\overline{SC} = \overline{\text{'}SS\text{'} \oplus SX}$ (Fig 5.8)

To form ('SS' $\oplus$ SX) does not need any extra Exclusive OR, since this signal already exists (for multiplicative overflow detection). Only requirement is to invert to form $\overline{\text{'}SS\text{'} \oplus SX}$.

Therefore implementation of expression (5.6) needs -- One Inverter, two 3-input AND gates, and One 2-input Exclusive OR. The gate counts are less than that would be required to implement the expression (5.5). Assuming 'SS' exists, the delay is --- One Inverter delay (3 ns) + One 3-input AND gate delay (4·75 ns) + One Exclusive OR delay (7 ns) ≃ 15 ns. The delay is less than that would be for the expression (5.5).

However, the expression (5.6) can be reduced to the expression (5.5)

$$(\text{'}\phi 2\text{'}.\ SX.\ \text{'}SS\text{'}) \oplus (\text{'}\phi 2\text{'}.\ \overline{SC}.\ SZ)$$

$$= \text{'}\phi 2\text{'}.\ [(SX.\ \text{'}SS\text{'}) \oplus (\overline{SC}.\ SZ)]$$

$$= \text{'}\phi 2\text{'}.\ [\overline{SX.\ \text{'}SS\text{'}}.\ (\overline{SC}.\ SZ) + (SX.\ \text{'}SS\text{'}).\ \overline{\overline{SC}.\ SZ}]$$

$$= \text{'}\phi 2\text{'}.\ [(\overline{SX} + \text{'}\overline{SS}\text{'}) \cdot (\overline{SX \oplus SS}).\ SZ + SX.\ \text{'}SS\text{'}.\ (SX \oplus \text{'}SS\text{'} + \overline{SZ})]$$

$$= '\phi 2'.\ [(\overline{SX} + '\overline{SS}')\ (\overline{SX}.\ 'SS' + SX.\ '\overline{SS}').$$
$$SZ + SX.\ 'SS'\ (\overline{SX}.\ 'SS' + SX.\ '\overline{SS}' + \overline{SZ})\ ]$$
$$= '\phi 2'.\ [\overline{SX}.\ '\overline{SS}'.\ SZ + SX.\ SS.\ \overline{SZ}].$$

In addition/subtraction '$\overline{\phi 2}$' is '0'. therefore the output from the multiplicative overflow detector is '0'. The output from the OR gate (46,13,13,11) gives the information of additive/subtractive overflow ('1' indicates overflow).

5.4(c) Delay in Multiplicative Overflow Detection

Multiplicative overflow detection needs two processes (i) detection of incorrect sign of the result and (ii) detection whether the product is outside the range of system (using group technique) -

(i) The detection of incorrect sign of the product involves in detection of the sign of the result, 'SZ'.

Delay in finding 'SZ' is (from Fig 5.8)

$$D_{SZ} = 2\ T_3 + 3\ T_{XOR} + T_I + T_{AND} + T_{MUX} \quad \ldots\ldots\ (5.7)$$

(Of 2 T_3, one T_3 is needed to form product term)

where T_3 = the access time of (256x4) PROM.

T_{XOR} = the delay in the Exclusive OR.

T_I = the delay in the Inverter.

T_{AND} = the delay in 2-input AND gate.

T_{MUX} = the delay in 2:1 multiplexer.

For the components used in the circuit in Fig 5.8

T_3 = 70 ns ; T_{XOR} = 7 ns, T_I = 10 ns ; T_{AND} = 8.8 ns

T_{MUX} = 20 ns. Therefore

D_{SZ} = 140+21+10+8.8+20 ≃ 200 ns.

From Fig 5.10 (circuit in the solid-lined Box), the delay in detecting incorrect sign, D_{IS} is given by

$$D_{IS} = D_{SZ} + T_{3AND} + T_{XOR} + T_{OR} \quad \ldots\ldots (5.7a)$$

where T_{3AND} = the delay in 3-input AND gate.

T_{OR} = the delay in OR gate.

(Since delay in forming 'SC' (2 T_{XOR}) is less than the delay in detecting 'SZ', delay in forming 'SC' has been omitted.)

T_{OR} = 14 ns (for the OR-gate used in the circuit 5.10)

Therefore

$$D_{IS} = 200+7+8.8+14 \simeq 230 \text{ ns} \quad \ldots\ldots (5.7b)$$

(ii) Delay in overflow detection by 'grouping technique' D_{GT} is given by (from Fig 5.10)

$$D_{GT} = T_3 + T_{ADD} + 4\,T_{OR} + 2\,T_{AND} \quad \ldots\ldots (5.7c)$$

where T_{ADD} = the delay in adding 4 bits. For the adder used in Fig 5.10, T_{ADD} = 25 ns. Therefore

$$D = 70+25+4x14+2x8.8$$

$$\simeq 169 \text{ ns} \quad \ldots\ldots (5.7d)$$

From (5.7b) and (5.7d), it is seen that the overflow detection by 'group technique' is faster than by 'incorrect sign detection' technique. This is due to the fact that the sign detection is not needed for the formation of group number. The (5.7a) gives the actual expression for delay in multiplicative overflow in the present system.

5.4(d) Delay in additive/subtractive overflow detection

Delay in additive/subtractive overflow detection D_{ADD} is given by (from Fig 5.10)

$$D_{ADD} = D_{SZ} + T_{XOR} + 2\ T_{OR} \quad \ldots\ldots (5.7e)$$

$$= 200 + 7 + 2 \times 14$$

$$= 235 \text{ ns} \quad \text{(for the components used in Fig 5.10)}$$

5.5 Integer Division

Integer division needs a first approximation of the quotient and then proceeds by successive approximations each of which is a number equal to the preceding approximation divided by 2. The first approximation is formed from the information of the group number, $g(X)$ of the dividend and the group number, $g(Y)$ of the divisor according to the Table 5.14. (The first approximation and the successive approximations are positive numbers.)

Table 5.14

Table of first approximation of quotient

Condition	First Approximation
If $g(X) \geq g(Y)$	$2^{g(X) - g(Y)}$
If $g(X) < g(Y)$	Zero

(Division of zero or by zero is aborted.)

Table 5.15 shows the possible values of first approximations (in power of 2) and their residue forms in mod-16 and mod-15.

Table 5.15

Possible values of first approximations

First Approximation	First Approximation in residue form	
	mod-16	mod-15
2^7	0	8
2^6	0	4
2^5	0	2
2^4	0	1
2^3	8	8
2^2	4	4
2^1	2	2
2^0	1	1
0	0	0

Since the successive approximation is the number equal to the preceding number divided by 2, Table 5.15 also gives the possible values of successive approximation (except 2^7).

For the successive approximations, the number to be divided by 2 is supplied by the X-data bus (Fig 5.11). However, all eight bits need not be multiplexed. From the Table 5.15, it is clear that the residue in mod-16 is either zero or equal to the residue in mod-15. Therefore only six bits of information -- 4 bits of the residue in mod-15, 1 bit of zero-nonzero information of the residue in mod-16 and one extra bit signifying 'division by 2', are required. The requirement of 'division by 2' is commucicated by the 'ϕ3' bit which is connected to the input pin 13 of the multiplexer (34) and to the selection pin 1 of the multiplexer (34) and (35).

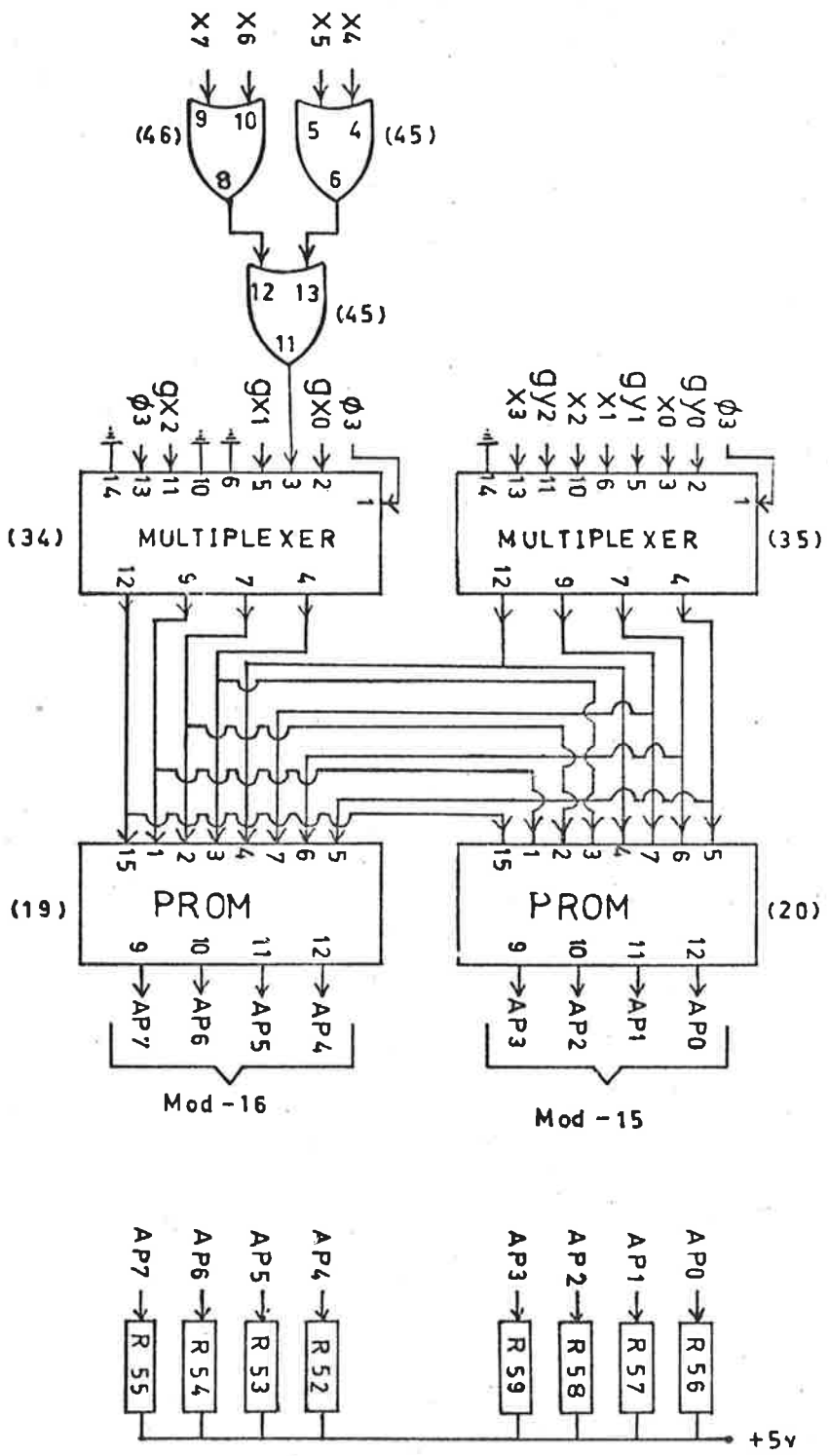


Fig. 5.11 Circuit for first approximation and successive approximations of quotient for the 16-15 residue system.

When 'ϕ3' goes high, four bits of the residue ($X3, X2, X1, X0$) in mod-15, one bit of zero/nonzero information of the residue ($X7, X6, X5, X4$) in mod-16 and 'ϕ3' (= '1') are selected to the output of the multiplexers (2:1 multiplexer). When 'ϕ3' goes low the group number of X, ($gX0, gX1, gX2$) and the group number of Y ($gY0, gY1, gY2$) are selected instead. (The group number of X and Y are generated by the PROMs (16) and (18) respectively of the circuit in Fig 5.10.)

The multiplexed outputs are fed to the PROMs (19) and (20) which form two look up tables of the first approximation and successive approximations. One table (Table 5.16(a)) gives the residue in mod-16 and the other table (Table 5.16 (b)) gives the residue in mod-15. Locations (00H to 07H), (10H to 17H), (20H to 27H), ... (70 H to 77 H) altogether 64 locations are allotted for first approximations and locations (80 H to 9F) altogether 32 locations are allotted for successive approximations.

The integer division is an iterative process; and it can be shown with the flow chart of Fig 5.12.

Table 5.16(a)

Look up table for first approximation and successive approximation in mod-16

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0								
1	0	1	0	0	0	0	0	0								
2	0	2	1	0	0	0	0	0								
3	0	4	2	1	0	0	0	0								
4	0	8	4	2	1	0	0	0			Z	E	R	O		
5	0	0	8	4	2	1	0	0								
6	0	0	0	8	4	2	1	0								
7	0	0	0	0	8	4	2	1								
8	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0
A																
B																
C																
D							Z	E	R	O						
E																
F																

Table 5.16(b)

Look up table for first approximation and successive approximation in mod-15

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0								
1	0	1	0	0	0	0	0	0								
2	0	2	1	0	0	0	0	0								
3	0	4	2	1	0	0	0	0			Z	E	R	O		
4	0	8	4	2	1	0	0	0								
5	0	1	8	4	2	1	0	0								
6	0	2	1	8	4	2	1	0								
7	0	4	2	1	8	4	2	1								
8	0	8	1	0	2	0	0	0	4	0	0	0	0	0	0	0
9	0	0	1	0	2	0	0	0	4	0	0	0	0	0	0	0
A																
B																
C																
D							Z	E	R	O						
E																
F																

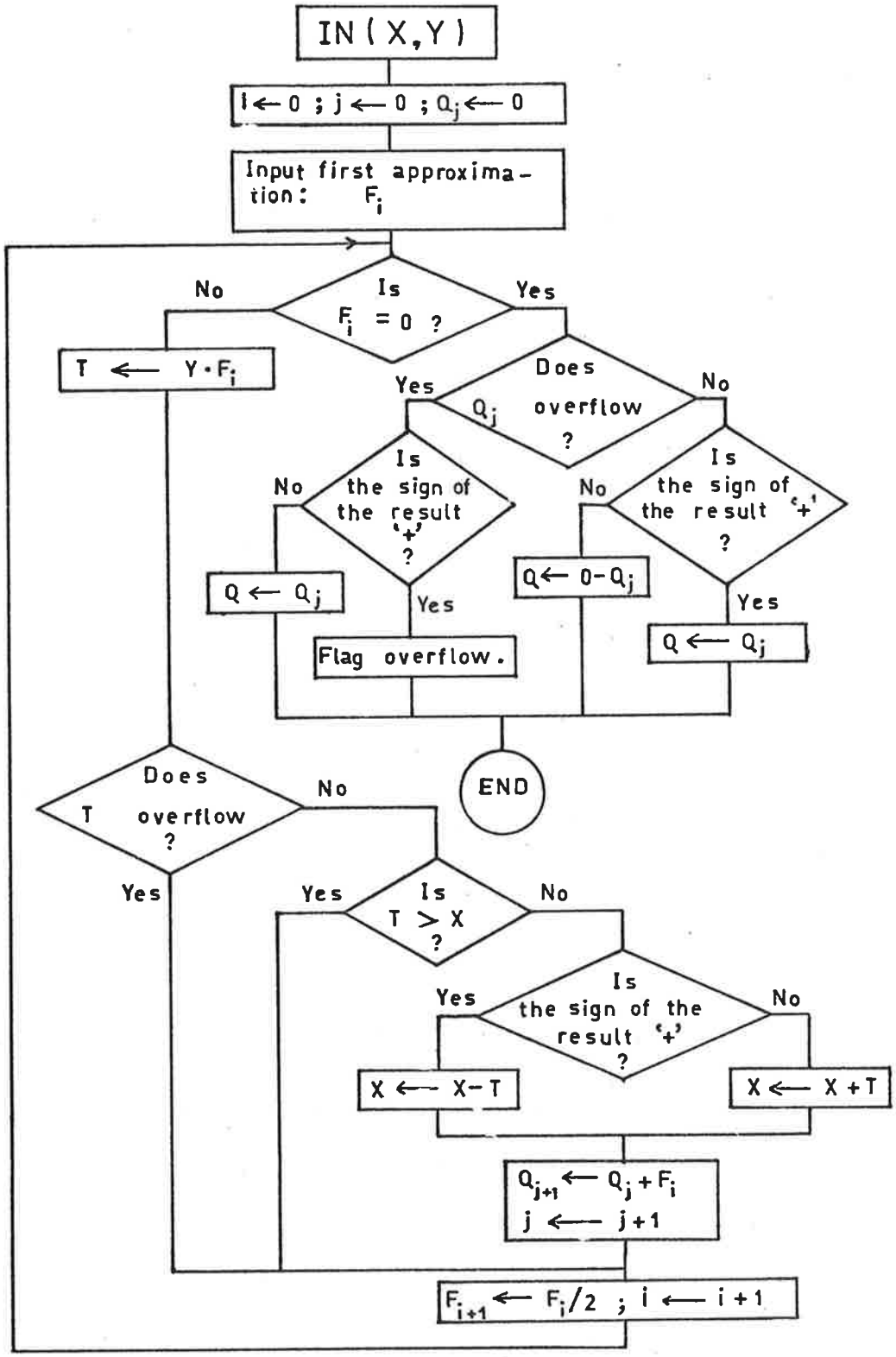


Fig. 5.12 Process of integer division in the 16-15 residue system.

Iterative processes continue such that the iterative values of X continue to decrease. Therefore T (in Fig 5.12) is added or subtracted from the iterative values of X depending on the sign of the result (Exclusive OR of the signs of X and Y i.e. $SX \oplus SY$). From Table 5.17, it is obvious that T is added if the sign of the result is negative; otherwise T is subtracted. (The first approximations and the successive approximations are positive.)

Table 5.17

Table to decide whether T is to be added or subtracted

SY	SX	Sign of T	Sign of the result $SX \oplus SY$	Operation
0	0	0	0	Subtraction
0	1	0	1	Addition
1	0	1	1	Addition
1	1	1	0	Subtraction

(The sign of T follows SY.)

The quotient generated is in positive form. Therefore it is converted to negative form (subtracting from zero), if the sign of the result is negative.

The division of M/2 (120 in this case) by -1 presents a problem. M/2 is a negative number and there is no 'counterpart' positive M/2. The sign of the result of above division is positive; whereas quotient (M/2) is negative.

However, this problem can be solved by software. When the positive approximations are added successively, the last addition will show overflow when M/2 is divided by ±1. Therefore when sign of the result is positive and overflow occurs, then overflow signal is to be flagged. If the sign of the result is negative and overflow occurs, the quotient is M/2. (The software listings have been shown in Appendix-6; AP6(a).)

5.5(a) Delay in finding approximations

Delay in finding the first approximation, D_{FA} is given by (Fig 5.11)

D_{FA} = Delay in finding the group number of X and Y
+ 1 multiplexers delay (Fig 5.11)
+ 1 PROM delay (Fig 5.11)

= 1 PROM delay (Fig 5.10)
+ 1 multiplexer delay + 1 PROM delay.

For the components used for the test circuit,

$$D_{FA} = 70 \text{ ns} + 17 \text{ ns} + 70 \text{ ns} = 157 \text{ ns} \quad \ldots\ldots (5.8)$$

Delay in finding a successive approximation D_{SA} is given by (from Fig 5.11)

D_{SA} = 2 OR gate delay
+ 1 multiplexer delay + 1 PROM delay.
= 2x14+17+70 = 115 ns
(for the components used in the circuit).

5.6 Floating Point Arithmetic

In this case the base of the exponent is chosen to be $\beta = (M/2)^{\frac{1}{2}} = (120)^{\frac{1}{2}} \simeq 11$. A number X is expressed as

$$X = A.11^{n} \qquad \ldots\ldots (5.9)$$

where A is the mantissa and n is the exponent, the range of which is from 0 to +119 and from -1 to -120.

In floating point operations each mantissa is separated into two terms such that

$$A = a_2.11 + a_1 \qquad \ldots\ldots (5.9a)$$

a_2 and a_1 are the co-efficients of 1-power and 0-power of 11 respectively and the range is from 0 to 10. The steps to find the co-efficients are

(i) $a_2 = \left[\frac{A}{11} \right]$ (Integer Division)

(ii) $a_1 = A - a_2.11$

During normalisation a co-efficient a_i is needed to be rounded. The steps that are required for rounding are

(i) Find $T = \left[\frac{a_{i-1} \pm 5}{11} \right]$

(If a_{i-1} is negative (-5) is added; otherwise +5)

(ii) Find the rounded $a_i = a_i + T$

$$\left(\left[\frac{11}{2} \right] = 5 \right)$$

Floating point arithmetic is done by sottware (Appendix-6(b)) which resides in the PROMs of SDK-80 microcomputer.

5.6(a) Floating Point Addition

The floating point addition can be described with the flow chart of Fig 5.13(a), 5.13(b) and 5.13(c).

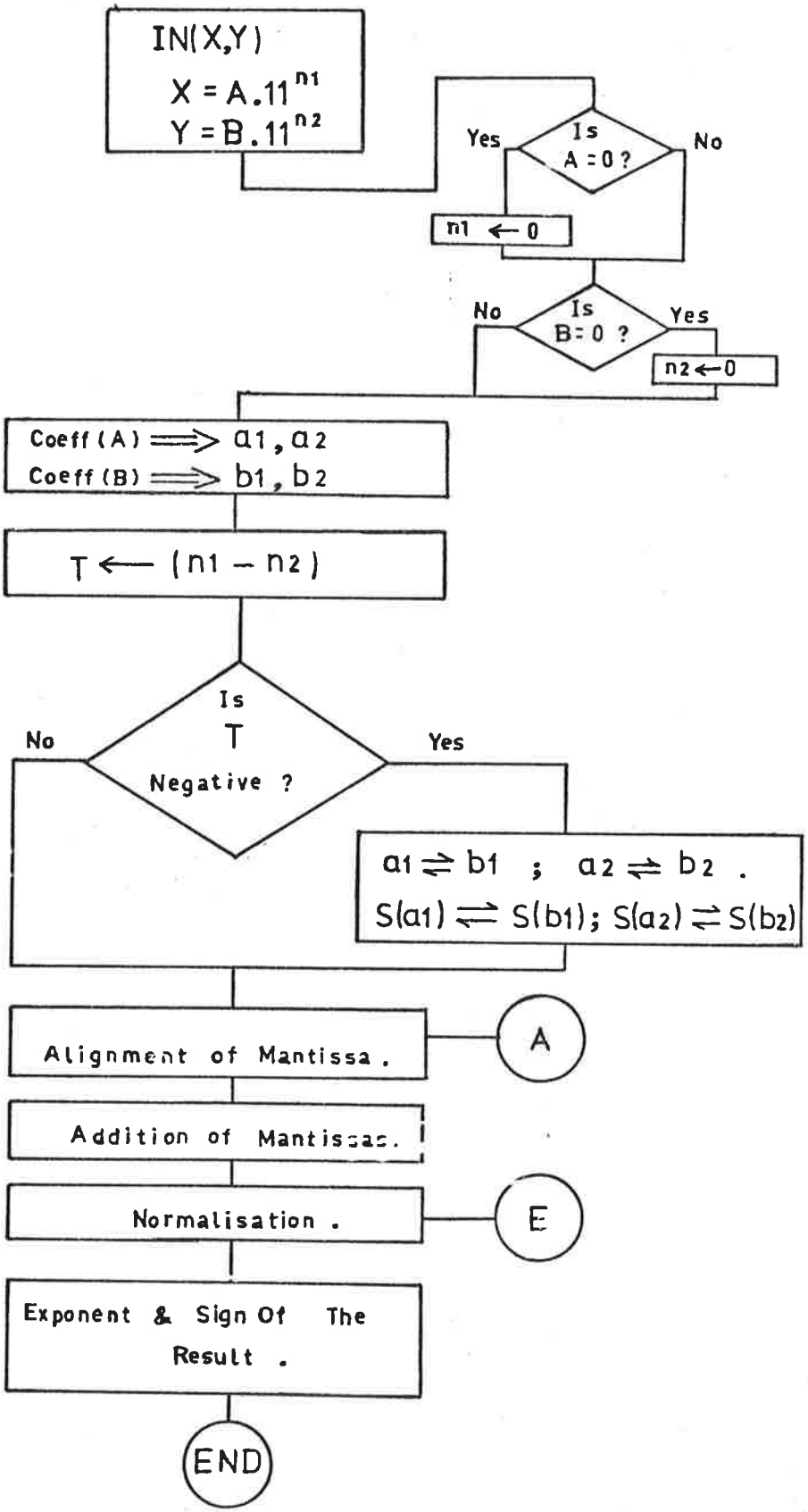
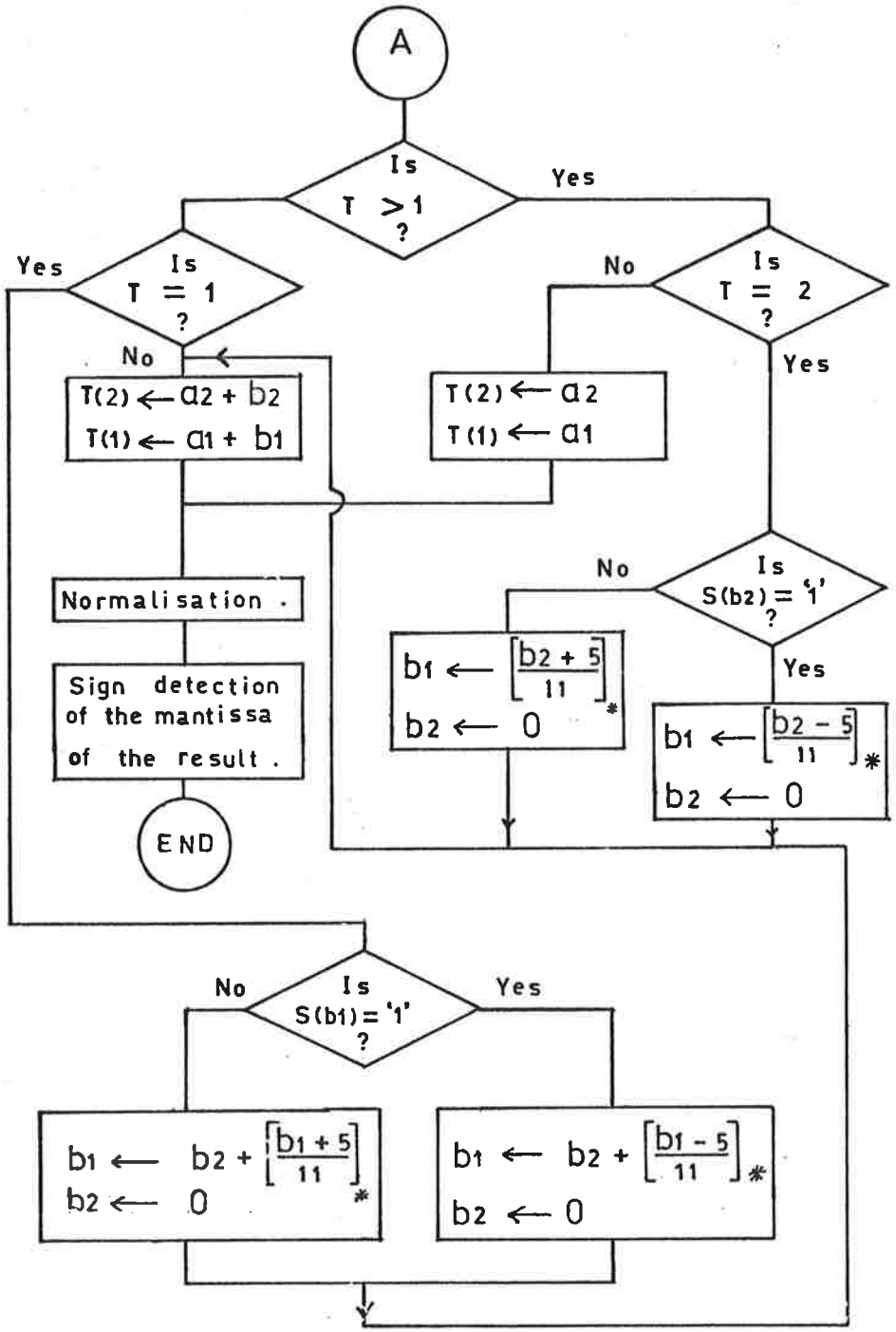


Fig. 5.13(a): Flow Chart Of Floating Point Addition In the 16-15 residue system .



* Process of rounding.

Fig. 5.13(b) Process of alignment and floating point addition in the 16-15 residue system .

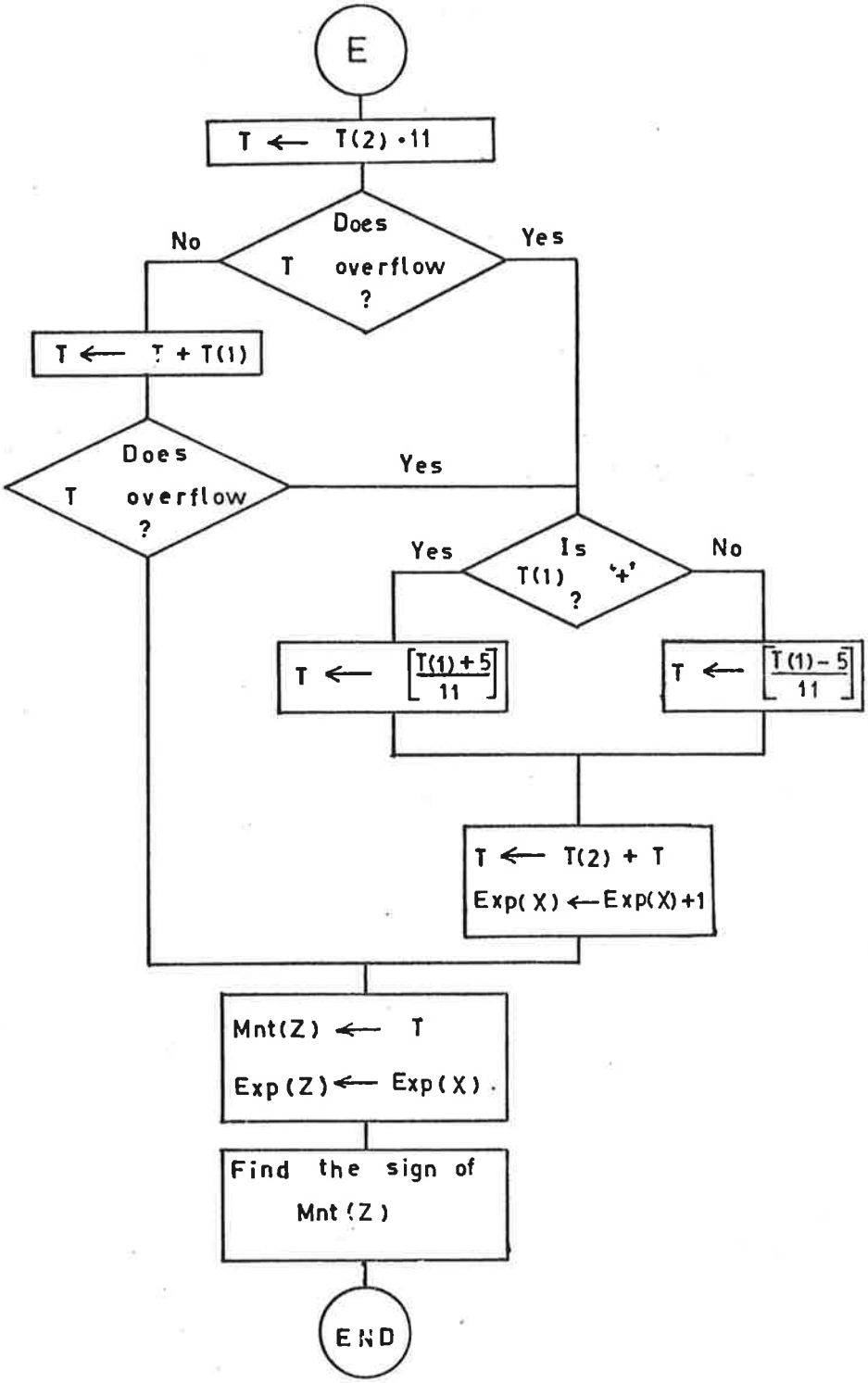


Fig. 5.13(c) Process of normalisation in floating point addition in the 16-15 residue system.

5.6(b) Floating Point Subtraction

After the subtraction of the mantissa of subtrahend from zero, the floating point is same as floating point addition. The flow chart of Fig 5.14 shows the floating point subtraction.

5.6(c) Floating Point Multiplication

The theory behind the floating point multiplication has been given in sections 4.4.3(b) and 4.4.4(b). The floating point multiplication in 16-15 moduli system can be described with the flow chart of Fig 5.15(a), 5.15(b).

To find the exponent of the result, Exp(Z) is a bit complicated process. The Exp(Z) can be written as

$$Exp(Z) = Exp(X) + Exp(Y) + g$$

where g is the 'generated exponent' (section 4.4.4 and Fig 5.15(b))

When Exp(X) and Exp(Y) are positive, the overflow in Exp(X) + Exp(Y) indicates that Exp(Z) is outside the range. (g is always positive.). When Exp(X) and Exp(Y) are negative, Exp(X) + Exp(Y) may overflow, however, Exp(Z) may or may not remain within the range due to the addition of g.

In later case it has been observed that when Exp(Z) is really within the range, and if Exp(X) + Exp(Y) overflows, addition g makes a second overflow such that Exp(Z) comes within the range.

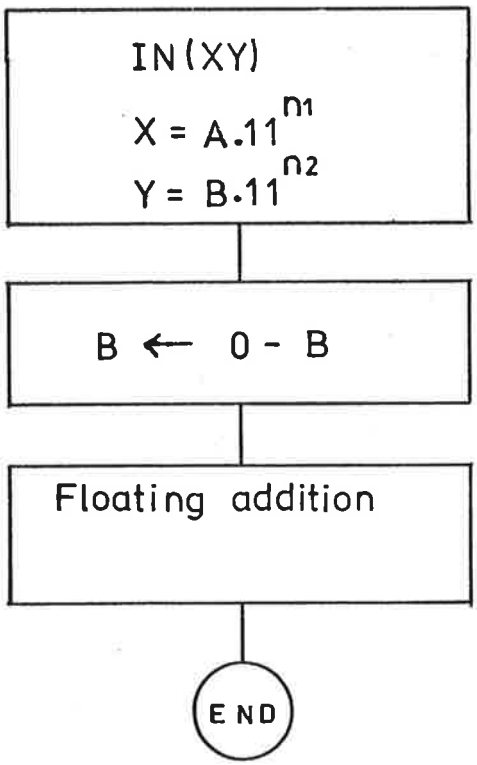


Fig. 5.14 Flow chart of floating point subtraction in the 16-15 residue system .

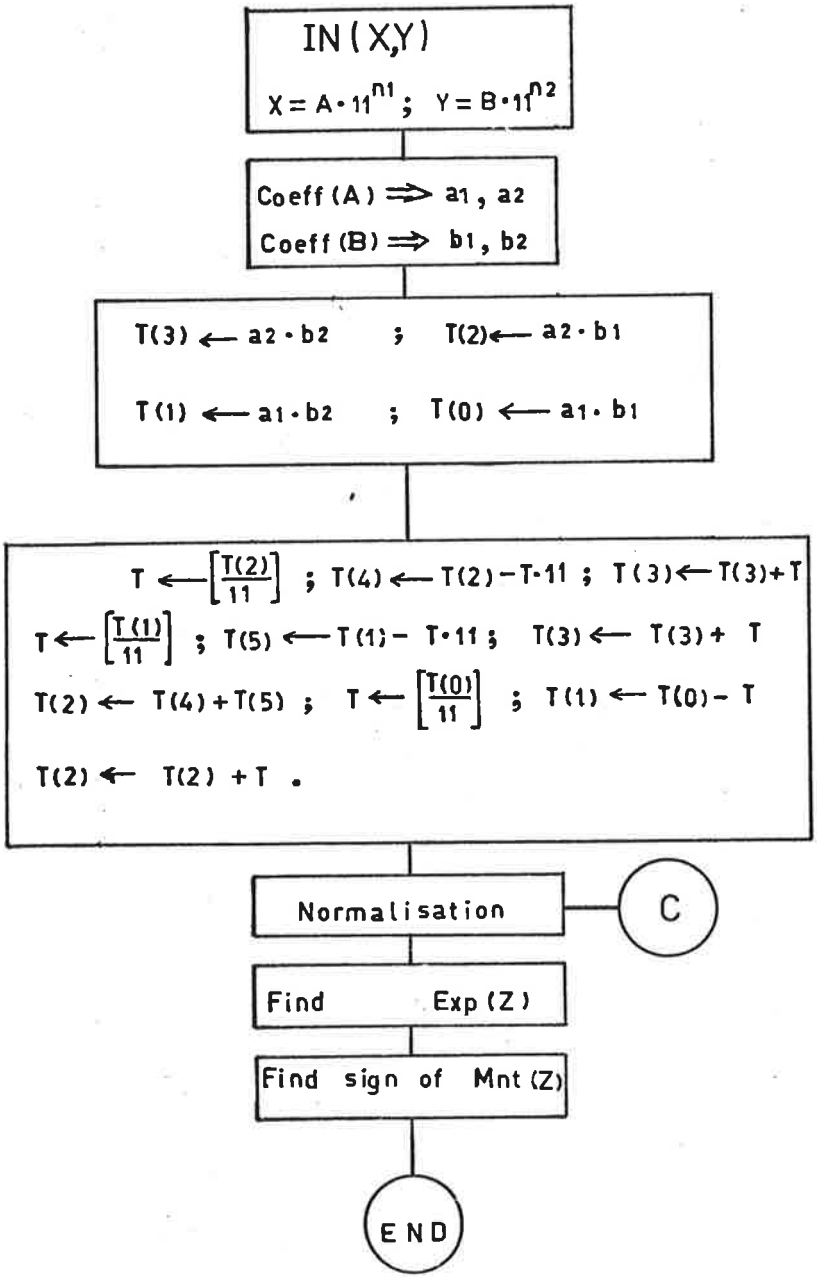


Fig. 5.15(a) Flow chart of floating point multiplication in the 16-15 residue system.

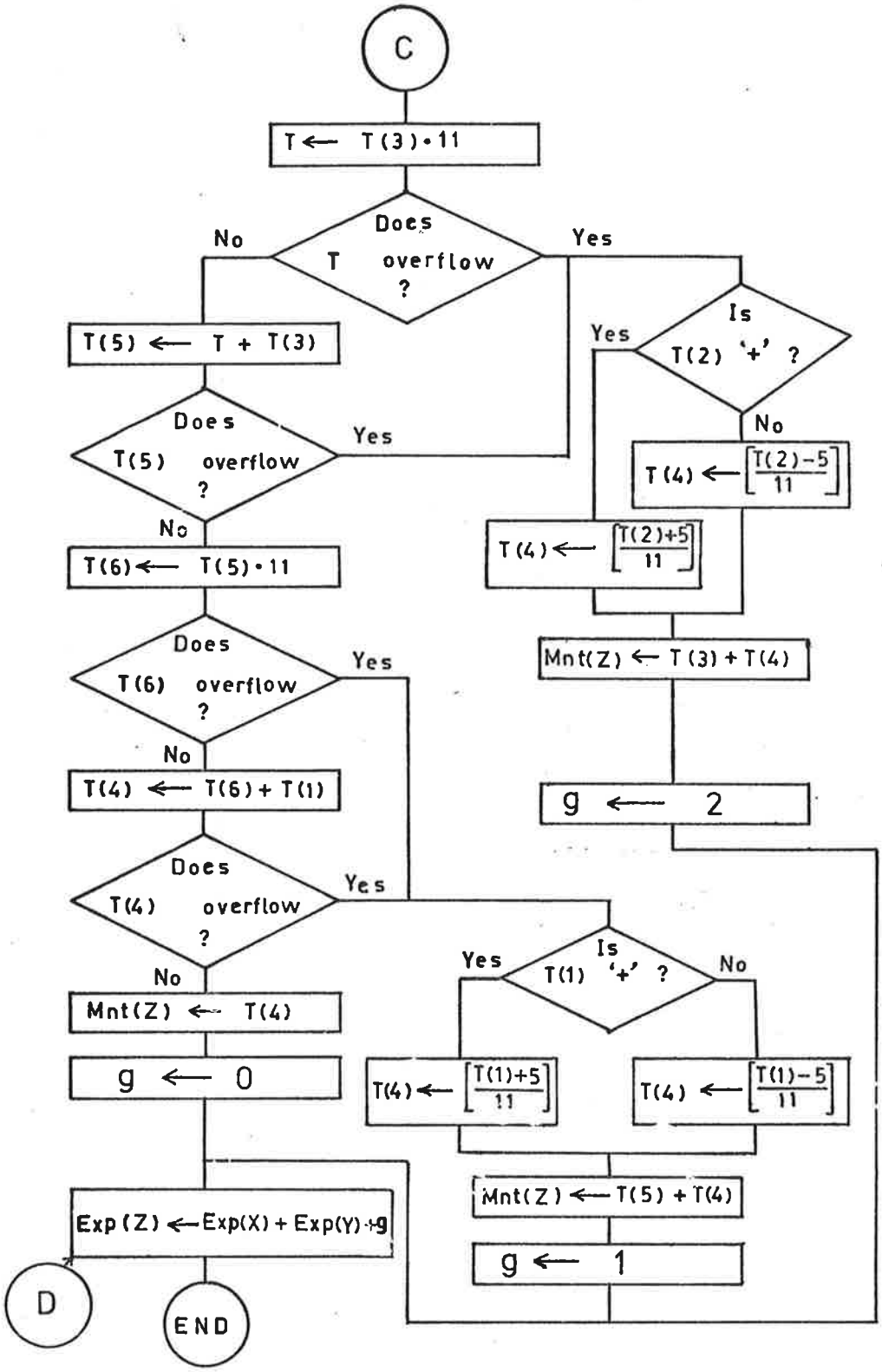


Fig. 5.15(b) Flow chart of normalisation in the 16-15 moduli floating point multiplication.

Therefore if overflow occurs twice or overflow does not occur at all, Exp(Z) is within the range, otherwise EXP(Z) is outside the range. The flow chart of Fig 5.15(c) explains how to determine whether Exp(Z) overflows or not.

5.5(d) Floating Point Division

The flow chart of Fig 5.16(a) shows the floating point division. (The division of zero or by zero is aborted.)

The mantissa and exponent of $\frac{1}{B}$ are generated with two look up tables; one is for the mantissa of $\frac{1}{B}$ and the other for the exponent of $\frac{1}{B}$.

The mantissa of the result, Mnt (Z) is found by floating point multiplication of the mantissa of X and that of $\frac{1}{B}$. This multiplication results in a 'generated exponent', g.

The calculation of the exponent of floating point division is complicated too.

$$\text{Exp}(Z) = [\text{Exp}(X) - \text{Exp}(Y)] + [g + EE] \quad \text{(Fig 5.16(a))}$$

Exp(X)-Exp(Y) may overflow; however, Exp(Z) may remain within the range due to addition of the term $[g+EE]$ which may be positive or negative and does not overflow.

(g = 0 or 1 or 2 and EE = 0 or -1 or -2 or -3)

Applying the same reasonings as given in determining the overflow condition of Exp(Z) in case of floating multiplication, the overflow condition of Exp(Z) in case of floating point division can be shown with the flow chart of Fig 5.16(b).

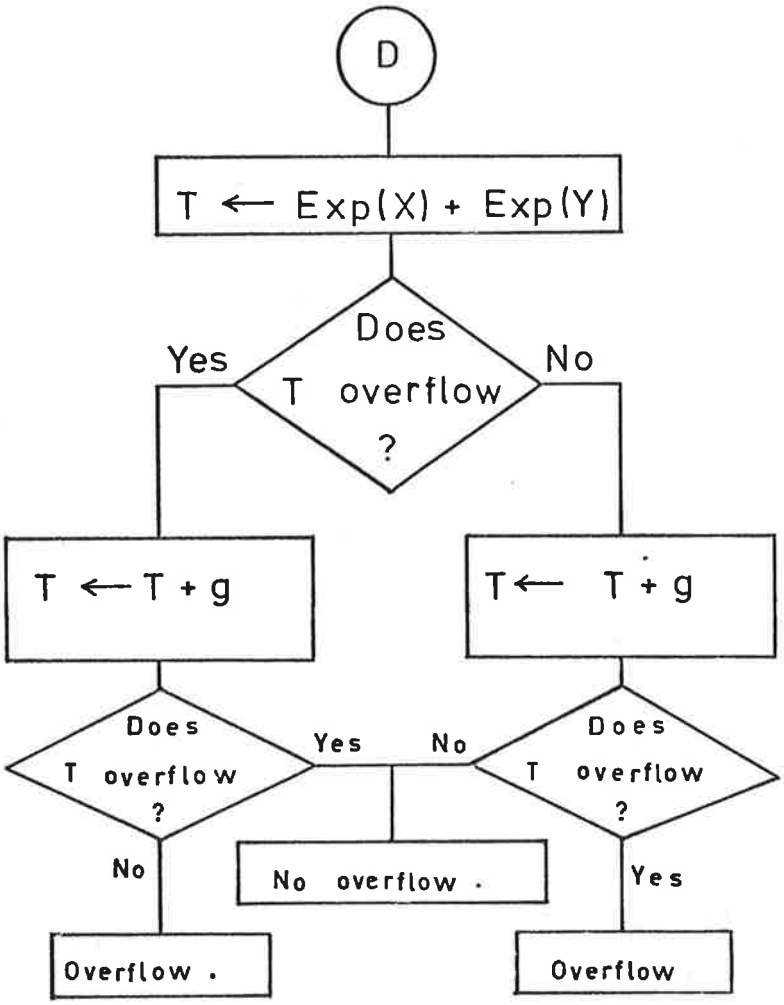


Fig. 5.15(c) Overflow detection of Exp(Z) in floating point multiplication in the 16-15 residue system .

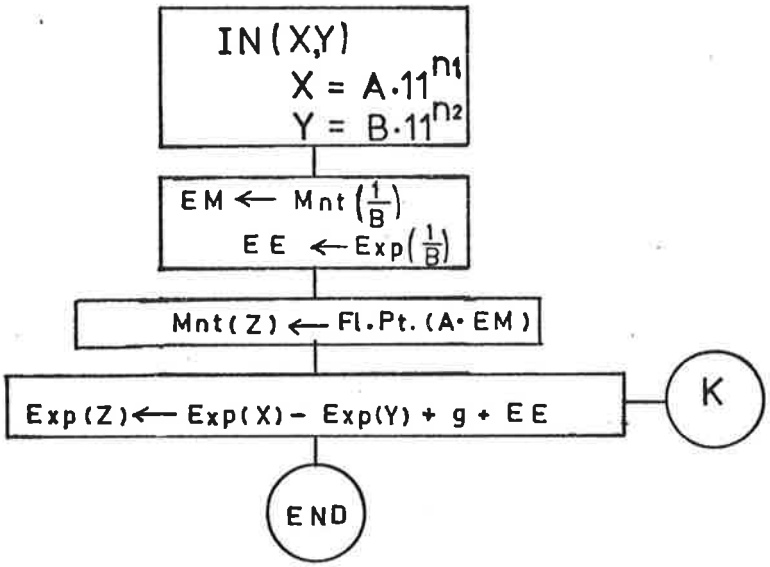


Fig. 5.16(a) Flow chart of floating point division in the 16-15 residue system.

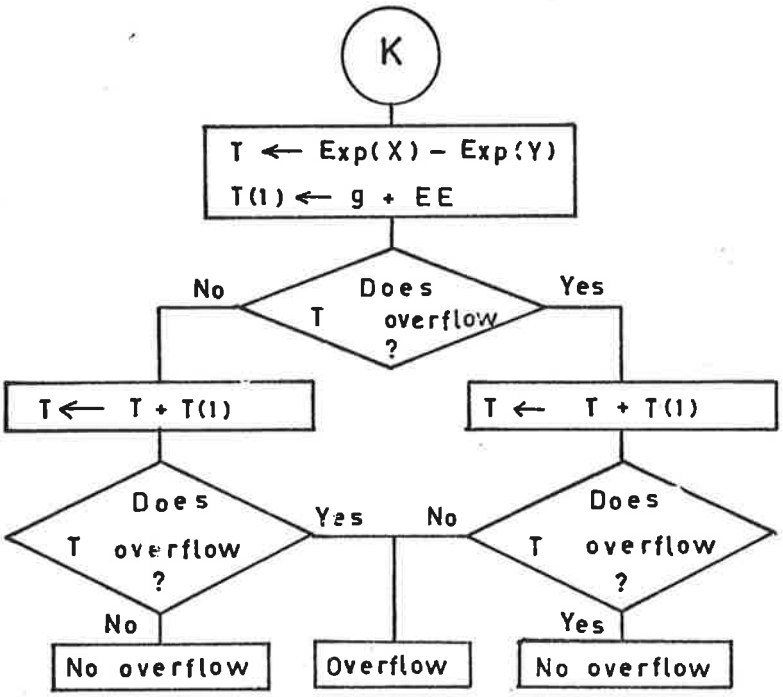


Fig. 5.16(b) Overflow detection of Exp(Z) in floating point division in the 16-15 residue system.

5.6(e) Generation of mantissa and exponent of $\frac{1}{B}$

For a base of exponent β, a general method to calculate the mantissa and exponent of $\frac{1}{B}$ has been described in Appendix -2. Applying that method the values of mantissa and exponent of $\frac{1}{B}$ for the base of exponent β=11 (eleven) have been tabulated in decimal and residue forms shown in Appendix-3.

The mantissa and exponent of $\frac{1}{B}$ in residue form are stored in PROMs (Fig 5.17). The value B is fed to the PROM (17), (21) and (23) through the X-data bus and PROM (17) and (21) output the mantissa in mod-16 and mod-15 respectively, while the PROM (23) gives the value of exponent.

The exponent of $\frac{1}{B}$ may be of any value of -1, -2 and -3. The residue form of those values are shown in Table 5.19

Table 5.19

Residue form of the exponent of $\frac{1}{B}$

Exponent in Decimal form	Exponent in Residue form			
	In Hex		In Binary	
	mod-16	mod-15	mod-16	mod-15
-1	F	E	1111	1110
-2	E	D	1110	1101
-3	D	C	1101	1100

From Table 5.19 it is seen that two most significant bits of the residues (in binary) in mod-16 and mod-15 are same and constant (11). Therefore in the formation of a look up table, storing of these two bits is not necessary; only two least

significant bits of each residue are required. Therefore only one (256x4) PROM is needed.

Two M.S.Bs are generated at the output circuit of the RAU and this will be discussed in section 5.7.

The PROM(23) (Fig 5.17) forms the look up table for the generation of the two least significant bits of each residue of the exponent of $\frac{1}{B}$.

The Tables 5.20(a), 5.20(b) and 5.20(c) show the contents of the PROMs (17), (21) and (23) respectively.

5.6(f) Delay in finding the mantissa and exponent of $\frac{1}{B}$

Delay = 1 PROM delay (from Fig 5.17)

= 70 ns (for PROM, Intel 3601).

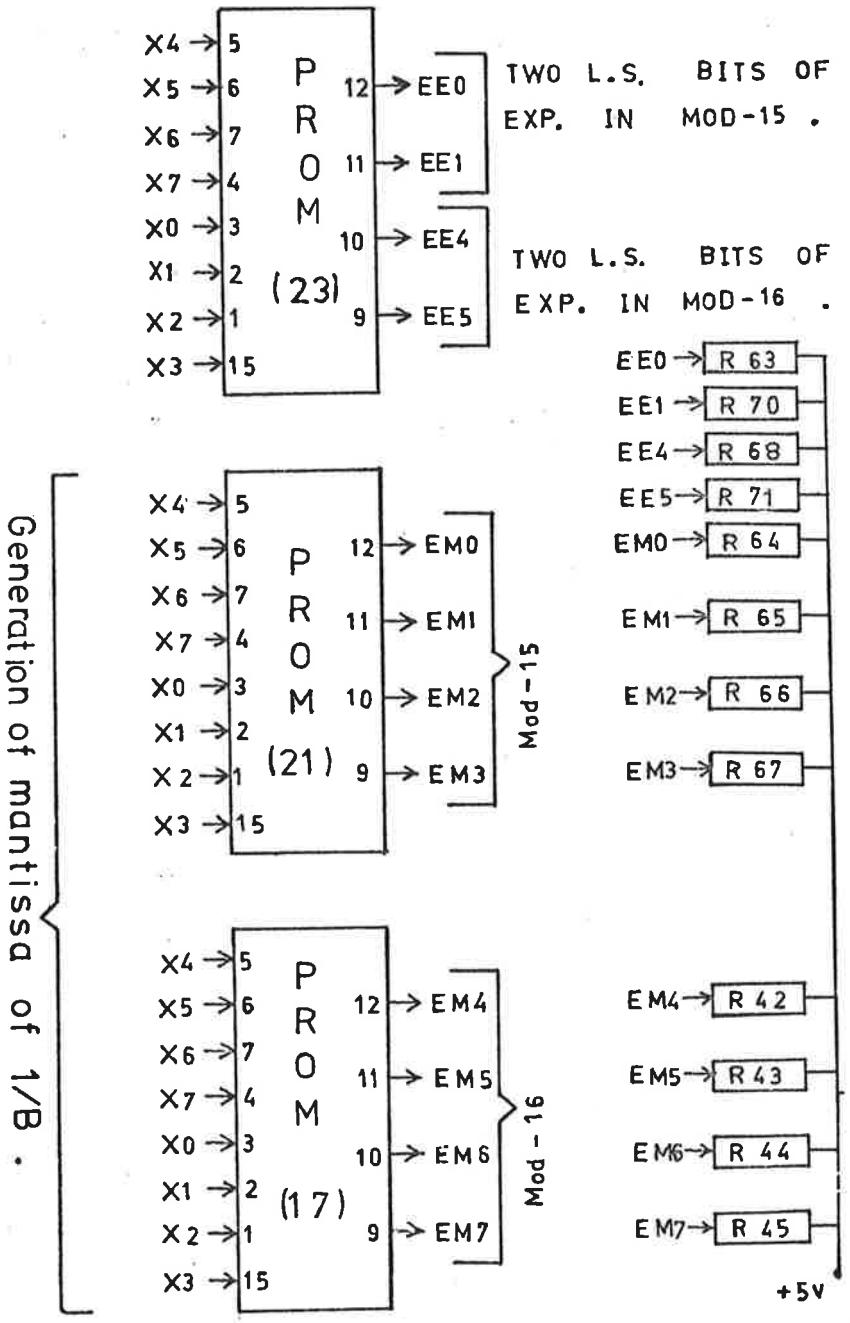


Fig. 5.17 Circuit for approximate floating point number of 1/B in the 16-15 residue system .

Table 5.20(a)

Look up table for the mantissa of $\frac{1}{B}$ in mod-16

Each hex digit represents the residue of B in mod-16. (Lower 4 address bits)

Each hex digit represents the residue of y in mod-15. (Higher 4 address bits).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	7	4	2	A	E	1	3	5	D	F	2	6	E	C	9
1	3	B	1	2	2	9	E	1	3	5	D	F	2	6	D	B
2	A	E	D	A	0	1	9	E	1	3	5	C	E	1	5	C
3	C	8	A	8	1	F	0	9	E	1	3	5	C	E	1	5
4	5	B	7	6	E	5	6	0	8	D	1	3	5	4	2	F
5	1	4	B	6	3	8	4	B	F	8	D	0	3	4	C	E
6	E	0	4	A	5	F	4	3	9	E	7	D	0	3	4	C
7	C	E	0	4	A	4	D	1	1	6	D	7	C	0	2	4
8	4	C	E	0	4	9	3	A	F	F	3	C	6	C	0	2
9	2	4	C	D	0	3	9	2	7	D	C	1	B	6	C	0
A	F	2	4	C	D	0	3	8	1	5	C	8	D	A	5	C
B	B	F	2	4	B	D	F	3	8	0	A	B	2	A	9	5
C	4	B	F	2	4	B	D	F	2	7	0	1	F	8	6	8
D	6	4	B	F	2	4	B	D	F	2	7	F	0	6	3	2
E	D	5	3	A	E	1	3	B	D	F	2	7	E	E	F	5
F	0	7	4	2	A	E	1	3	5	D	F	2	6	E	C	9

Table 5.20(b)

Look up table for the mantissa of $\frac{1}{B}$ in mod-15

Each hex digit represents the residue of B in mod-16. (Lower 4 address bits)

Each hex digit represents the residue of Y in mod-15. (Higher 4 address bits).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	1	0	8	C	0	2	4	D	0	3	7	0	E	E
1	8	B	A	E	0	7	C	0	2	4	D	0	3	7	E	D
2	C	3	1	3	C	E	7	C	0	2	4	C	E	2	6	D
3	D	A	E	A	9	B	D	7	C	0	2	4	C	E	2	6
4	6	C	9	A	0	4	4	D	6	B	0	2	4	3	1	D
5	2	5	C	8	7	9	3	7	C	6	B	E	2	3	C	E
6	E	1	5	B	7	3	5	2	5	B	5	B	E	2	3	C
7	C	E	1	5	B	6	1	2	0	2	A	5	A	E	1	3
8	3	C	E	1	5	A	5	D	0	D	E	9	4	A	E	1
9	1	3	C	D	1	4	A	4	A	D	A	C	8	4	A	E
A	D	1	3	C	D	1	4	9	3	8	C	6	8	7	3	A
B	9	D	1	3	B	D	0	4	9	2	B	B	0	5	6	3
C	2	9	D	1	3	B	D	0	3	8	2	4	6	5	1	5
D	3	2	9	D	1	3	B	D	0	3	8	1	3	C	E	C
E	7	2	1	8	C	0	2	B	D	0	3	8	0	1	5	4
F	0	1	1	0	8	C	0	2	4	D	0	3	7	0	E	E

Table 5.20(c)

Look up table for two least significant bits of each residue (in mod-15 and mod-16) of $\frac{1}{B}$

Each hex digit represents the residue of B in mod-16. (Lower 4 address bits)

Each hex digit represents the residue of Y in mod-15. (Higher 4 address bits).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
1	4	E	4	4	4	4	4	4	4	4	4	4	4	4	4	4
2	4	4	9	4	4	4	4	4	4	4	4	4	4	4	4	4
3	4	4	4	9	4	4	4	4	4	4	4	4	4	4	4	4
4	4	4	4	4	9	9	4	4	4	4	4	4	4	4	4	4
5	4	4	4	4	4	9	9	4	4	4	4	4	4	4	4	4
6	4	4	4	4	4	4	9	9	4	4	4	4	4	4	4	4
7	4	4	4	4	4	4	4	9	9	4	4	4	4	4	4	4
8	4	4	4	4	4	4	4	4	9	9	4	4	4	4	4	4
9	4	4	4	4	4	4	4	4	4	9	9	4	4	4	4	4
A	4	4	4	4	4	4	4	4	4	4	9	9	4	4	4	4
B	4	4	4	4	4	4	4	4	4	4	4	9	9	4	4	4
C	4	4	4	4	4	4	4	4	4	4	4	4	4	9	4	4
D	4	4	4	4	4	4	4	4	4	4	4	4	4	4	9	4
E	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	E
F	0	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

5.6.1 Error in Floating Point Addition/ Subtraction and Multiplication

For $\beta = 11$, $\left[\beta/2 \right] = 5$. The highest positive number for the system under consideration is +119; and the highest negative number is -120.

5.6.1(a) Error in Floating Point Addition/ Subtraction

From (4.20), the maximum relative error for positive numbers is

$$\text{(i)} \quad \text{Max } (e) = \frac{\left[\beta/2 \right]}{R} = \frac{5}{119} = .042$$

The maximum relative error for negative numbers is

$$\text{(ii)} \quad \text{Max } (e) = \frac{5}{120} \simeq .042$$

5.6.1(b) Error in Floating Point Multiplication

From (4.21), the maximum relative error for positive numbers is

$$\left(U_{(eo)} \right)_{max} = \frac{2 \cdot \left(\left[\beta/2 \right] \right)^2 + 3 \cdot \left[\beta/2 \right]}{R}$$

$$= \frac{2 \cdot 25 + 3 \cdot 5}{119} = \frac{65}{119} \simeq .55$$

The maximum relative error for negative numbers is

$$\left(U_{(eo)} \right)_{max} = \frac{65}{120} = .54$$

5.7 Output Circuit of the RAU

The outputs from the RAU are the results of addition, subtraction, multiplication, comparison, approximate quotient for integer division, and mantissa and exponent of $\frac{1}{B}$ for floating division.

The sign of the result of addition, subtraction and multiplication, overflow signal and the Exclusive OR of the sign of the divisor and dividend are connected to the pins 4, 5 and 6 respectively of the RAU board (Fig 5.8 and 5.10) and are eventually connected to the C-port (lower part - C0, C1, C2) of the SDK-80 microcomputer (Fig 5.4).

The result of addition, subtraction, multiplication, comparison, approximate quotient, and the mantissa and exponent of $\frac{1}{B}$ are multiplexed by three least significant bits of the operation codes (Fig 5.18).

The pin 2 of the multiplexers (24), (25), (28) and (29) are connected to +5V through 1.8 K resistors to generate two most significant bits for each residue of exponent of $\frac{1}{B}$.

The pin 4 of the multiplexers (24), (25), (26), (27) and (28) are connected to ground which generates zeros in the 5 MSB positions of the comparison signals; only the three least significant bits give the information of comparison.

Table 5.21 shows the states of the selection pins and the selected inputs.

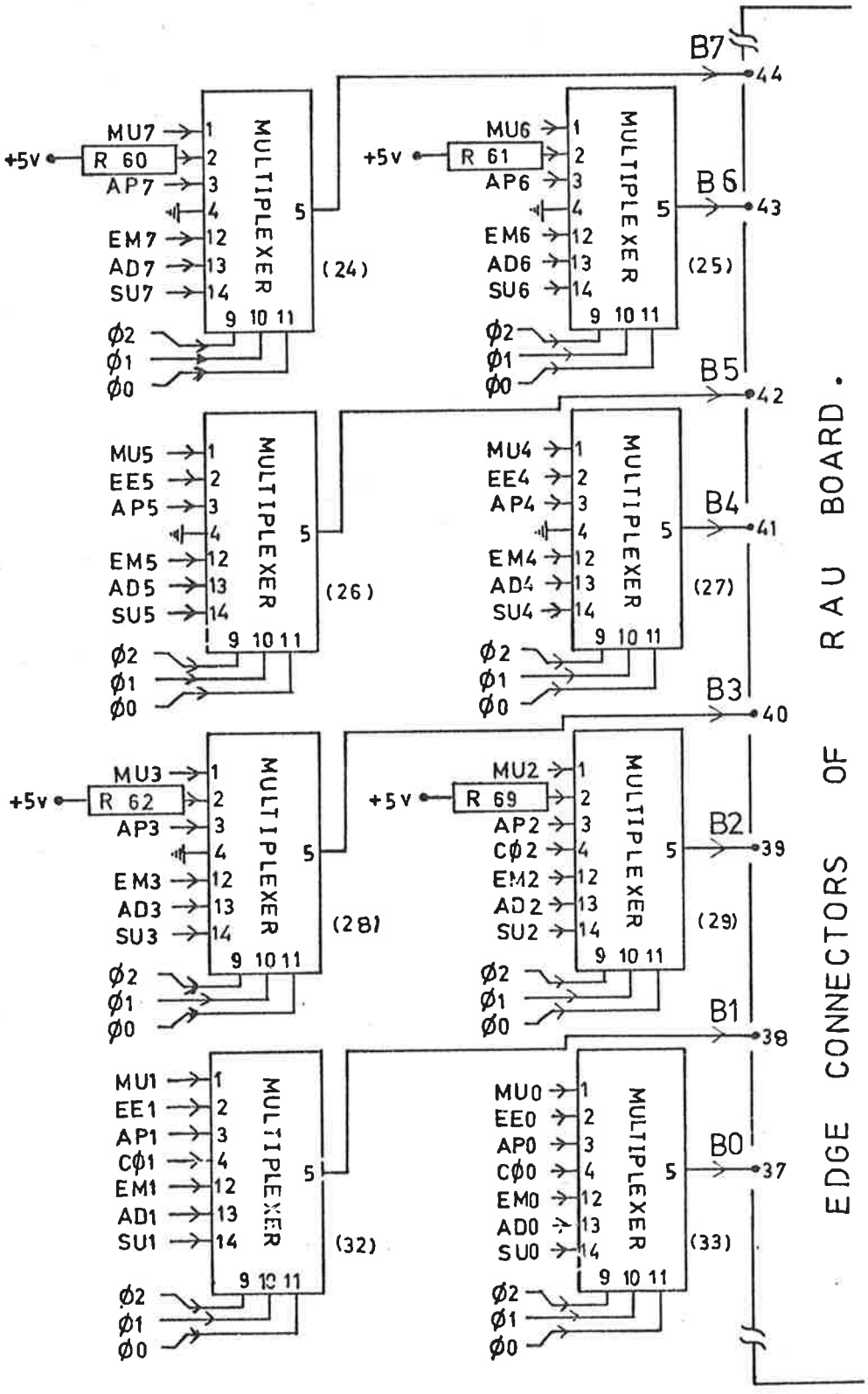


Fig. 5.18 Output circuit of the RAU.

Table 5.21

States of selection pins and selected inputs

States of selection pins $\phi2, \phi1, \phi0$ (3 bits of op. code)	Selected inputs
1 1 0	Addition
1 0 1	Subtraction
0 1 1	Multiplication
0 0 1	Approximate quotient
0 0 0	Comparison
1 1 1	Mantissa of $\frac{1}{B}$
0 1 0	Exponent of $\frac{1}{B}$

The outputs from the multiplexers are connected to the pins 37 through 44 of edge connectors of the RAU board. The outputs are eventually connected to the B-port of SDK-80 microcomputer (Fig 5.4). The most significant 4 bits of the outputs represent the residue in mod-16 and the least significant 4 bits represent the residue in mod-15. In case of comparison, three least significant bits gives the information of the results of comparison.

5.8 Conversion of Residue Numbers to Decimal Numbers

The 16-15 residue system is a three-digit decimal system (the highest positive number is +119). The theory behind the conversion has been described in section 4.5.

Fig 5.19 shows the residue operations required to find one decimal digit (residue form) of a positive number X.

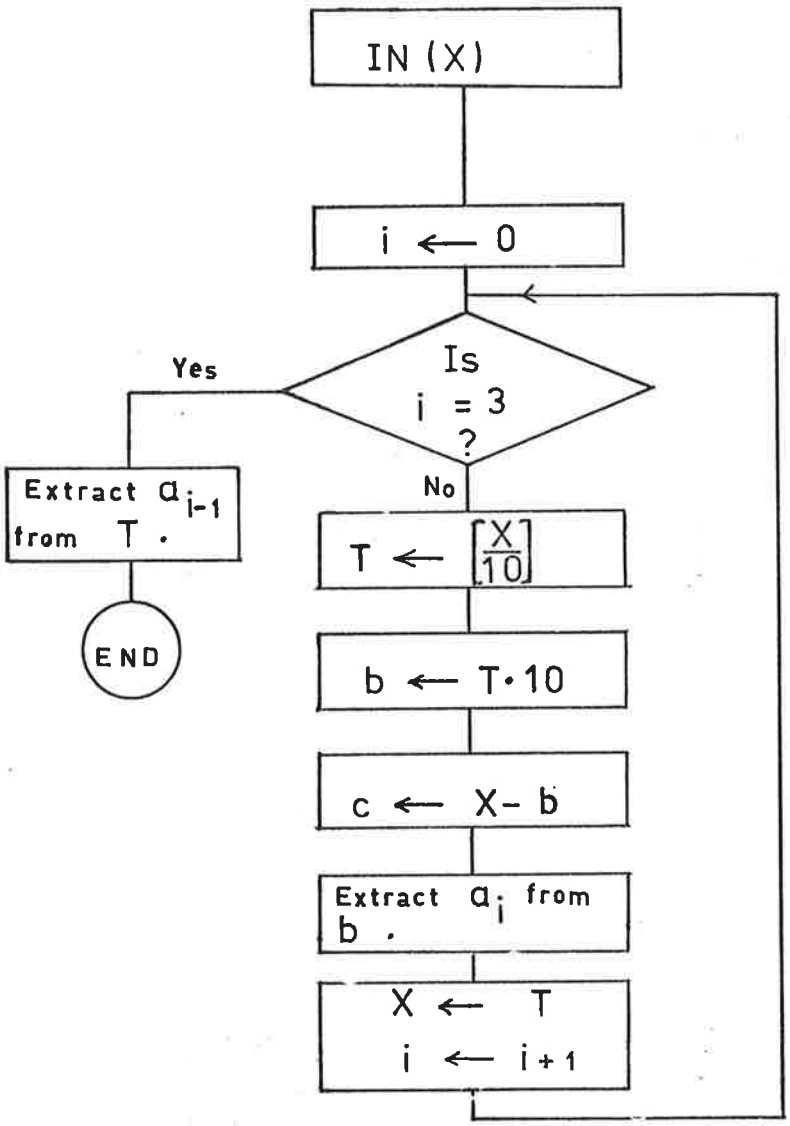


Fig. 5.19 Process of residue to decimal conversion in the 16-15 residue system .

Since 16 and 15 are greater than 10, the residue of the decimal digit in mod-16 is equal to the residue in mod-15 and each residue is equal to the decimal digit. Only one residue is needed to be extracted. Decimal numbers are represented in signed magnitude form. Therefore if X is negative, it is converted to its positive form by residue subtraction from zero or by residue multiplication with -1.

Successive digits are found by taking T (in Fig 5.19) as a new value of X.

The first decimal digit is extracted by residue multiplication with 01 and second digit by residue multiplication with 10. The residue addition of first and second extracted digits gives two consecutive decimal digits which are stored at an 8-bit-byte location. The third digit (M.S.D) is extracted by residue multiplication with 01 and is stored at another location. The contents of those locations, in proper order, give the decimal number of X.

(The program listings have been given in Appendix -6 (c).)

5.9 How the Residue Arithmetic Computer Works

The RAU together with the SDK-80 microcomputer forms a real time residue arithmetic computer (RAC). The communication between the RAC and outside is thus made through a teletype or VDU using SDK-80 programmes.

With the starting command - G 800↓ (↓= carriage return), the RAC undergoes the phases of operations shown by the flow chart of Fig. 5.20.

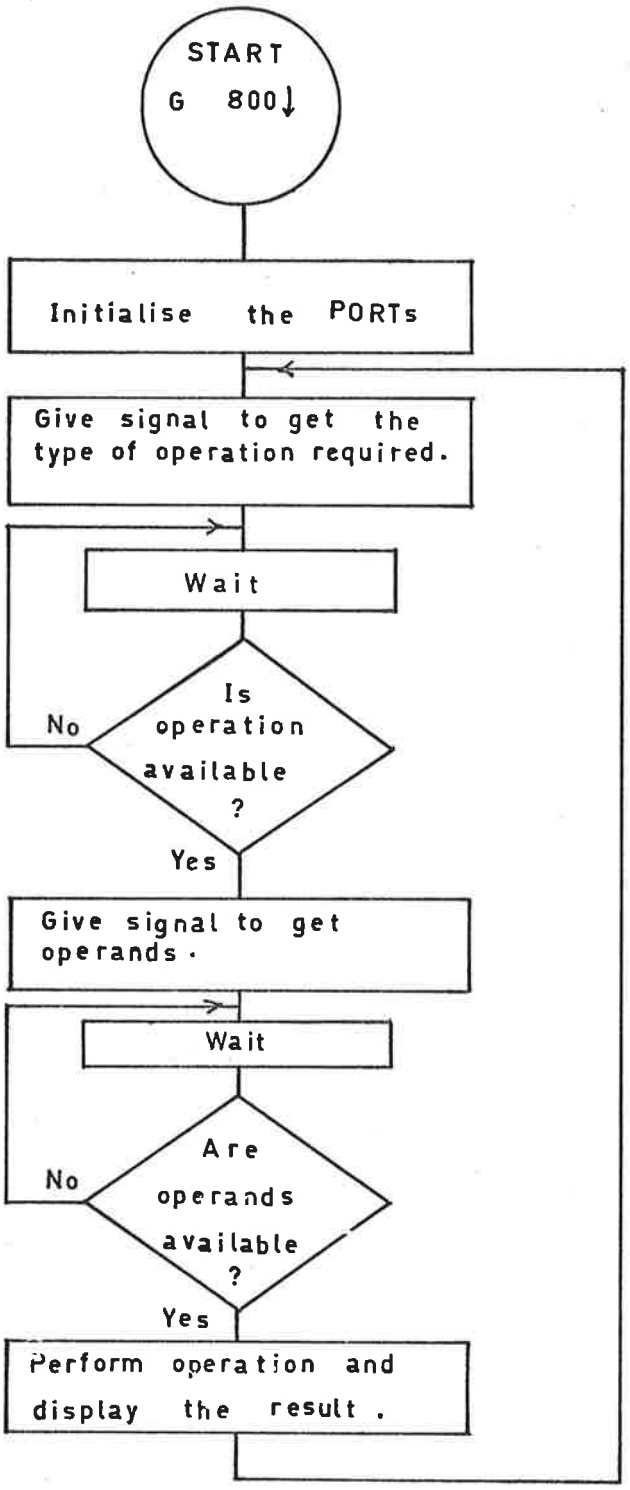


Fig. 5.20 Programe loops of the RAC .

5.10 How to operate the RAC

With the teletype or VDU connected to SDK-80 microcomputer and power on to the SDK-80 and RAU, the SDK-80 is 'reset'. Then the procedures that are followed are

(i) Key in G 800 ↓

The computer responds with

0 = (for type of operation).

(ii) Key in any alphabet A through J for the required operation. (See Table 5.4)

If the operation is an integer operation the computer responds with

M = (for first operand X).

(iii) Key in two hex numbers (residue number).

The computer responds with

M = (for second operand Y).

(iv) Key in two hex numbers (residue number).

The computer displays the result, first in residue form and second in decimal form with overflow (if any) and sign characters. (The overflow character is '?'.) If operation is a comparison one, the display is > or = or < to indicate X is greater than or equal to or less than Y. The computer signals

0 = (for further operation).

If operation is a floating point one, the computer responds with

M = (for the mantissa of X).

(iii) Key in two hex numbers (residue number).

The computer responds with

E = (for the exponent of X).

(iv) Key in two hex numbers (residue number).

The computer responds with

M = (for the mantissa of Y).

(v) Key in two hex numbers.

The computer responds with

E = (for the exponent of Y).

(vi) Key in two hex numbers.

The computer displays the result as

M = - - E = - - (in residue form)

M = - - - - E = - - - - (in decimal form).

(The dash-line indicates the digits. The results are displays with proper sign and overflow (if any) characters.)

Then the computer responds with

0 = (for further operation).

5.11 Conclusions

The delay in multiplicative overflow detection, according to the proposed technique, is 230 ns; whereas delay, according to conventional fast technique [using PROMs of 70 ns access time (Intel Bipolar PROM, 3601)] would be 4.2.70 = 560 ns. Therefore multiplicative overflow detection is considerably faster, according to the proposed technique. In determination of the first approximation for residue integer division, the proposed technique introduces a delay of 157 ns; whereas conventional fast division technique would need 2.2.70 = 280 ns. Therefore determination of first approximation is also faster in the proposed technique.

Until now there has been no residue computer which performs floating point operations. The introduction of floating point arithmetic in 16-15 moduli system reveals the fact that floating point arithmetic operations are feasible in residue system, and that these operations are cost-effective. The maximum relative errors that are introduced in addition and multiplication [section 5.6.1] are not really significant and quite normal for a single precision machine. Therefore the proposed algorithms on floating point arithmetic have great implication in the residue system.

In the residue-to-decimal conversion process, where the conversion radix is 10 and the moduli are 16 and 15, neither A(X) method nor base extension is applicable. Therefore the proposed technique is the only choice. Moreover there is no reason for not using the proposed technique where

all the integer arithmetic operations exist and more time can be allotted for output translation.

CHAPTER VI

6. CONCLUSIONS

This concluding chapter presents a comparative study between residue and binary number systems, and puts forward a suggestion on further study on residue number systems.

The bases of comparison are the speed, cost and power consumption in performing the basic operations - multiplication, addition/subtraction including overflow detection.

For the purpose of comparison 16-15 and 16-15-13-11 moduli systems have been considered and the fastest available logic components of both TTL and ECL technology have been chosen when dealing with implementations.

The range of 16-15 moduli system (16x15=240) is less than 2^8, however, greater than 2^7; likewise the range 16-15-13-11 system (16x15x13x11 = 34320) is less than 2^{16}, however, greater than 2^{15}. In this comparison, however, a 16-15 residue system has been compared with an 8-bit binary system; while the 16-15-13-11 residue system has been compared with a 16-bit binary system.

In multiplication the 16-15-13-11 system needs four (256x4) PROMs. The component requirement for multiplicative overflow detection (on the basis of Fig 4.1) has been shown in Table 6.1(a).

In the circuit of Fig 4.1 the generation of the group number of the operand X, $g(X)$ has been shown. The generation of $g(Y)$ needs similar sorts of components as required for $g(X)$. Again for the generation of $g(X)$ the information of $NZ(X)$ is required; and $NZ(X)$ can be formed with one 8 input NAND gate and one inverter with two levels of gate delay. Similar type of components are needed for $NZ(Y)$. The sign detection (by mixed radix conversion) of the product (SZ) needs five (256x4) PROMs. The circuit configuration is similar to the circuit used for SX, except that (256x8) PROM is replaced by a (256x4) PROM.

The requirement when using ECL components is same as that of TTL components, however, since (256x8), (512x8) and (512x4) ECL PROMs are not available, those PROMs are to be built from available (256x4) PROMs. The (256x8) is a PROM with 8-bit address lines and 8-bit data lines. The function of that PROM can be configured by using two (256x4) PROMs. One is used for four least significant data outputs, and other one is used for four most significant data outputs. The (512x8) PROM can be built from two (256x8) PROMs (constructed by above method). The 8-bit address lines are used to select any location of (256x8) PROM and the remaining address bit selects one of two (256x8) PROMs. The outputs from (256x8) PROMs are wire-ORed. Therefore four (256x4) PROMs are needed to construct an equivalent (512x8) PROMs. Construction of a (512x4) PROM needs two (256x4) PROMs. The 8-bit address lines select any location of (256x4) PROM, whilst the remaining address line selects one of two (256x4) PROMs. The outputs of (256x4) PROMs are wire-ORed. Table

6.1(b) shows ECL required for multiplication and its overflow detection in 16-15-13-11 moduli system. In place of a 4-bit full adder (ECL) which is not available, a 4-bit Arithmetic Logic Unit (ALU) has been chosen. Since an 8 input ECL NAND is not available, 2 input AND gates are to be used for the generation of $NZ(X)$ and $NZ(Y)$. A total of fourteen 2-input AND gates are needed and the arrangement has three levels of gate delay using currently available components.

In a 16-15 moduli system, multiplication needs two (256x4) PROMs. The components for multiplicative overflow detection can be calculated from the circuits in Fig 5.10 and 5.8. The circuit in Fig 5.8 has been used for sign SZ of the result Z. Table 6.2(a) shows the TTL components required for multiplication and its overflow detection in a 16-15 moduli system.

The ECL components needed are the same as the TTL components, however, since ECL 3-input AND gates are not available, 2-input AND gates are to be used. Therefore for three 3-input AND gates, six 2-input AND gates are substituted. Table 6.2(b) shows the ECL components needed for multiplication and overflow detection in a 16-15 moduli system.

Of the TTL family Schottky TTL [57] is the fastest; therefore Schottky TTL components have been chosen. Of the ECL family [58], ECL-III is the fastest; however, ECL-III has a limited number of functions, therefore ECL-10000 has been chosen for the ECL components. Tables 6.3(a) and 6.3(b) show the characteristics regarding delay, cost and power

consumption of TTL and ECL components respectively.

Table 6.1(a)

TTL components required for multiplication and its overflow detection in a 16-15-13-11 moduli system. (Yaking into account of generation of $g(Y)$ and SZ.)

Function	Component Required	Type No.	Number of Components	Number of Packages
Multiplication	(256x4) PROM	82S27	4	4
Overflow Detection: (According to the circuit in Fig 4.1, requirement for generation of $g(Y)$ and sign detection of Z i.e. SZ.)	(256x4) PROM	82S27	14	14
	(256x8) PROM	82S114	2	2
	(512x4) PROM	82S130	4	4
	(512x8) PROM	82S115	4	4
	4-bit Full Adder	SN74S283	6	6
	2:1 Multiplexer	SN74S158	2	2
	2-input Exclusive OR	SN74S86	2	1
	2-input AND	SN74S08	2	1
	2-input OR	SN74S32	3	1
	[8-input NAND	SN74S30	2	2
	Inverter for generation of $NZ(X)$, $NZ(Y)$.]	SN74S04	2	1
				42 (Total package count)

Table 6.1(b)

ECL components required for multiplication and its overflow detection in a 16-15-13-11 moduli system.

Function	Component Required	Type No.	Number of Components	Number of Packages
Multiplication	(256x4) PROM	10149	4	4
Overflow Detection: (According to the circuit in Fig 4.1, the requirement for generation of $g(Y)$ and the sign detection of Z i.e. SZ)	(256x4) PROM	10149	42	42
	4-bit Arithmetic Logic Unit	MC 10181	6	6
	2:1 Multiplexer	MC 10173	2	2
	2-input Exclusive OR	MC 10107	2	1
	2-input AND	MC 10104	16	3 (Hex AND)
	2-input OR	MC 10103	3	1
				59 (Total package count)

Table 6.2(a)

TTL components required for multiplication and its overflow detection in a 16-15 moduli system.

Function	Component Required	Type No.	Number of Components	Number of Packages of gates
Multiplication (Fig 5.6(c))	(256x4) PROM	82S27	2	2
Overflow Detection: (According to the circuits in Fig 5.8 and 5.10.)	(256x4) PROM	82S27	3	3
	4-bit Full Adder	SN74S283	1	1
	2-input Exclusive OR	SN74S86	5	2
	2-input OR	SN74S32	5	2
	2-input AND	SN74S08	4	1
	Inverter	SN74S04	3	1
	4:1 Multiplexer	SN74S153	4	4
	3-input AND	SN74S11	3	1
				17
				(Total package count)

Table 6.2(b)

ECL components required for multiplication and its overflow detection in a 16-15 moduli system.

Function	Component Required	Type No.	Number of Components	Number of Packages of gates
Multiplication	(256x4) PROM	10149	2	2
Overflow Detection: (According to the circuits in Fig 5.8 and 5.10.)	(256x4) PROM	10149	3	3
	4-bit ALU	MC 10181	1	1
	2-input Exclusive OR	MC 10107	5	2
	2-input OR	MC 10103	5	2
	2-input AND	MC 10104	10	2 (Hex AND)
	Inverter	MC 10195	3	1
	4:1 Multiplexer	MC 10174	4	4
				17 (Total package count)

Table 6.3(a)

Characteristics of TTL components required for Residue Multiplication and its overflow detection

	Components	Type Number	Delay in ns	Power	Price*	Manufacturers
(i)	(256x4) PROM	82S27	40	.6 mw/Bit	25.25	Signetics [78]
(ii)	(256x8) PROM	82S114	60	165 µw/Bit	19.00	"
(iii)	(512x4) PROM	82S130	80	.3 mw/Bit	6.50	"
(iv)	(512x8) PROM	82S115	60	165 µw/Bit	17.00	"
(v)	4-Bit Adder	SN74S283	7	510 mw	2.00	Texas Instrument [57]
(vi)	3-input AND	SN74S11	4.75	31 mw/gate	.57	" "
(vii)	Hex Inverter	SN74S04	3	19 mw/gate	1.08	" "
(viii)	2-input AND	SN74S08	4.75	32 mw/gate	.57	" "
(ix)	2-input OR	SN74S32	4	35 mw/gate	.57	" "
(x)	Quad Exclusive OR	SN74S86	7	250 mw	.85	" "
(xi)	2:1 Multiplexer	SN74S158	4	195 mw	2.39	" "
(xii)	8-input NAND	SN74S30	3	19 mw/gate	.86	" "
(xiii)	4:1 Multiplexer	SN74S153	6	225 mw	2.39	" "
					79.03	

* All prices are in Australian dollars and small quantity price.

Table 6.3(b)

Characteristics of ECL components required for Residue Multiplication and its overflow detection

Function	Type Number	Delay in ns	Power	Price*	Manufacturers
(i) (256x4) PROM	10149	20	.66 mw/Bit	22.32	Signetics [78]
(ii) 4-bit ALU	MC 10181	7	600 mw	35.20	Motorola [58]
(iii) Hex Inverter	MC 10195	2	100 mw/Pkg	1.31	"
(iv) Hex 2-input AND	MC 10104	2.7	140 mw/Pkg	.51	"
(v) 2:1 Multiplexer	MC 10173	2.5	270 mw	7.10	"
(vi) Triple Exclusive OR	MC 10107	2.5	110 mw/Pkg	.55	"
(vii) Quad 2-input OR	MC 10103	2	100 mw/Pkg	.51	"
(viii) Dual 4:1 Multiplexer	MC 10174	3.5	305 mw	7.06	"
				74.56	

* All prices are in Australian dollars and small quantiti price.

Presently the MPY/HJ family [56] of binary multipliers are the fastest. Table 6.4(a) shows comparisons of a 16 x 16 multiplier Vs. 16-15-13-11 moduli multiplier and an 8 x 8 bits multiplier Vs. 16-15 moduli multiplier using TTL technology. The delay in overflow detection in a 16-15-13-11 moduli multiplication has been calculated using the expression (4.1); whilst the expression (5.7) and (5.7a) have been used for calculation of delay in overflow detection in the 16-15 moduli multiplication. Table 6.4(b) shows similar comparisons using ECL components.

From the examination of Table 6.4(a) and 6.4(b) it is clear that residue multiplication is much faster and more cost-effective, specially using ECL technology. However, when residue multiplication is associated with overflow detection, the system gets slower and more costly. However, since the same piece of hardware, used for overflow detection, can be used for sign detection and generation of first approximation for residue integer division, the cost is not as much of a concerning factor as would at first appear.

Table 6.4(a)

Comparison of (i) 16x16 binary multiplier Vs 16-15-13-11 mod multiplier including its overflow detection

(ii) 8x8 binary multiplier Vs 16-15 mod multiplier including its overflow detection (TTL components.)

Function	Component Required	Number of Components	Delay in ns	Price*	Power
(1) 16 x 16 bits multiplier	TRW/ MPY-16 HJ	1	100	$265.00	3 W
(2) Residue Multiplication (16-15-13-11-mod)	See Table 6.1(a)	See Table 6.1(a)	40	$101.00	2.56 W
(3) Overflow Detection	See Table 6.1(a)	See Table 6.1(a)	269	$506.21	18 W
(4) 8 x 8 bits multiplier	TRW/ MPY-8 HJ-1	1	45	$120	1 W
(5) Residue Multiplication (16-15 mod)	See Table 6.2(a)	See Table 6.2(a)	40	$ 50.50	1.78 W
(6) Overflow Detection	See Table 6.2(a)	See Table 6.2(a)	129	$ 93.00	4 W

* All prices are in Australian dollars.

Table 6.4(b)

Comparison of (i) 16x16 binary multiplier Vs 16-15-13-11 mod multiplication including overflow detection

(ii) 8x8 binary multiplication Vs 16-15 mod multiplication including overflow detection. (Using ECL.)

Function	Component Required	Number of Components	Delay in ns	Price*	Power
(1) 16 x 16 binary multiplication	Motorola/10183 (2x4 multiplier)	32	100	@ 30.84ea 986.88	25.6 W
(2) Residue Multiplication (16-15-13-11)	See Table 6.1(b)	See Table 6.1(b)	20	89.28	2.7 W
(3) Overflow Detection	See Table 6.1(b)	See Table 6.1(b)	126	1165.39	33.16 W
(4) 8 x 8 binary Multiplier	Motorola/10183	8	50	@ 30.84ea 246.72	6.4 W
(5) Residue Multiplication (16-15-mod)	See Table 6.2(b)	See Table 6.2(b)	20	44.64	1.35 W
(6) Overflow Detection	See Table 6.2(b)	See Table 6.2(b)	65	320.00	5 W

* All prices are in Australian dollars.

A comparison of a 4-bit binary adder (TTL and ECL) with a 4-bit residue PROM adder has been shown in Table 6.5. From the table it is apparent that in TTL technology a 4-bit binary adder is six times faster than a 4-bit residue PROM adder. Therefore in addition of equivalent 24 (=6x4) bits or higher, residue PROM will be faster, provided the access time of the PROMs used is 40 ns. However, since the cost of one (256x4) PROM (TTL) is twelve times higher than that of a 4-bit binary adder (TTL), the residue adder is not cost effective.

In ECL technology, a 4-bit binary adder is three times faster than a 4-bit residue PROM adder. With ECL technology addition of equivalent 12 (=3x4) bits or higher will be faster in a residue system assuming that the access time of the PROMs used is 20 ns. However, sign detection in 3 moduli (each module having 4 bits) system will need 40 ns [= (3-1)·20; see Appendix-5]; whereas the sign detection in a 12 bit binary system will need 3x7 = 21 ns.

From what has been discussed above, it is clear that present day technology still is not favourable towards the implementation of residue in general purpose computers. The binary system will maintain its predominance. However, in special applications where there is no use of sign and over-flow detection, residue multiplication will be the first choice and may also be a contender in very fast and large systems where the carry free property is of great importance in minimising the chip connections and inter chip delays.

Table 6.5

Comparison of a 4 bit binary adder (TTL and ECL) Vs. a 4 bit residue PROM adder (TTL and ECL).

Function	Technology	Components	Number of Components	Delay in ns.	Price in $.A.
A 4 bit binary adder	TTL	SN74S283	1	7	2.00
	ECL	MC 10181	1	7	35.20
A 4 bit residue PROM adder	TTL	82S27 (256x4) PROM	1	40	25.25
	ECL	10149 (256x4) PROM	1	20	22.32

As it has been discussed earlier that the residue floating point arithmetic can be performed in software provided all the residue integer arithmetic operations are available. In fast processors the residue floating point operations will be fast. However, in comparison with binary floating point arithmetic the residue floating point arithmetic is slower since (i) the mantissas of the residue floating point operands are to be broken down into two co-efficients and (ii) in normalisation process the multiplicative overflow information is required. However, at least it has been shown that the floating point arithmetic can be performed in the residue system, and the operations are cost-effective, since the only requirements are PROMs for generation of floating point numbers of the reciprocal of the mantissa of divisor and memories to store the programme required for residue floating point operations.

It is clear from the discussion above that the multiplicative overflow detection is the speed limiting factor in the residue system. Also sign detection is a problem. Therefore more efficient algorithms for sign detection and overflow detection should be investigated and also the possibility of using logic gates or FPLAs (instead of PROMs) should be looked at. In the near future VLSI circuit may open the way to implementation of overflow detectors with combinational logic and a different comparison between the systems may then result.

The residue PROM adder/subtractor is slow. The combinational logic is another possible way for implementing residue addition/subtraction and should be a matter of further investigation.

BIBLIOGRAPHY

1. Pawlak, Z. "An Electronic digital computer based on '-2' system", Bull. Acad. Polonaise Sci. Serie des sciences techniques. Vol-7, No.12, Dec. 1959, pp.713-721.

2. Pawlak, Z. and Wakuliz A. "Use of expansions with a negative basis in arithmometer of a digital computer". Bull. Acad. Polonaise Sci. C1-3, Vol-5, No.3, March 1957, pp.233-236.

3. Brusenzov, N.P., Zhogolev, Y.A., Verigin, S.P., Maslov, S.P. and Tishulina, A.M. "The SETUN small automatic digital computer" (in Russian), Moscow University Vestnik, no.4, 1962, pp.3-12.

4. Israel, Halpern and Yoeli, M. "Ternary arithmetic unit", Proc. IEE, Vol-115, No.10, October 1968, pp.1385-1388.

5 Vranesic Z.G. and Hamachar, V.C. "Ternary Logic in Parallel Multipliers", The Computer Journal, Vol-15, No.3, Nov. 1972, pp.254-258.

6. Vranesic Z.G. and Smith, K.C. "Engineering Aspects of Multivalued Logic Systems", Computer, Sep. 1974, pp.34-41.

7. Mine, H. and Hasegawa, T. "Four Ternary Arithmetic Operations", System. Computers. Controls. Vol-2, No.1, 1971, pp.46-54.

8. Ataka, H. "On the Definition of Ternary Threshold Functions", System. Computer. Controls. Vol-2, No.1, 1971, pp.92-93.

9. Merril, R.D. Jr. "A Tabular Minimization Procedure for Ternary Switching Functions", IEEE trans on Electronic Computers. Vol-E6-15, No.4, August 1966, pp.578-585.

10. Mukhapadhyay, A. "Summetric Ternary Switching Functions", IEEE trans. on Electronic Computers, Vol-E6-15, No.5, Oct. 1966, pp.731-739.

11. Santos J. and Arango H. "On the Analysis and Synthesis of Three Valued Digital Systems", Proceedings Spring Joint Computer Conference, 1964, pp.463-475.

12. Porat, D.I. "Three Valued Digital Systems", Proc. IEE, Vol-116, No.6, June 1969, pp.947-950.

13. Mouftah H.T. and Jordan, I.B. "Design of Ternary COS/MOS Memory and Sequential Circuits", IEEE trans. on computers, Vol-C-26, March 1977, pp. 281-288.

14. Mouftah, H.T. "Three valued C.M.O.S. Cycling Gates" Electronic Letters, 19th Jan. 1978, Vol-14, No.2, pp.36-37.

15. Huertas, J.L., Acha, J.I. and Carmona, J.M. "Implementation of some ternary operations with CMOS integrated circuits", Electronic Letters, Vol-12, July 1976, pp.385-386.

16. Carmona, J.M., Huertas, J.L. and Acha, J.I. "Realization of three valued C.M.O.S. Cycling Gates", Electronic Letters, Vol-14, No.9, 24th April 1978, pp.288-290.

17. Kaniel, A. "Trilogic, a three levels logic system provides greater memory density", EDN, 1973, pp.80-83.

18. Mouftah, H.T. and Jordan, I.B. "Integrated Circuits for Ternary Logic", Proceedings of the International Symposium on Multivalued Logic, May 1974, pp.285-302.

19. Etiemble, D. and Israel, M. "Implementation of Ternary Circuits with Linear Integrated Circuits", IEEE trans. on Computer, 1977, C-26, pp.1222-1223.

20. Higuchi, T. and Kamayama, M. "Static Hazard Free T-gate for Ternary Memory Element and its application to Ternary Computer". IEEE trans. on Computers, Vol-26, No.12, Dec. 1977, pp.1212-1221.

21. Tichdao, T., McCluskay, E.J. and Russel, L.K. "Multivalued Integrated Injection Logic", IEEE trans. on Computer, Vol-26, No.12, Dec. 1977, pp. 1233-1241.

22. Anderson, D.J. and Dietmeyer, D.L. "A Magnetic Ternary Device", IEEE trans. on Electronic Computers, Dec. 1963, pp.911-914.

23. Pugh, A. "Application of binary device and Boolean Algebra to the realization of 3-valued Logic circuits", Proc. IEE 1967, 114, pp.335-338.

24. Rath, S.S. "A ternary flip-flop circuit", Int. J. Electronics, 1975, Vol-38, No.1, pp.41-47.

25. Watson, R.W. and Hastings, C.W. "Self-checked computation using residue number arithmetic", Proc. IEEE, Vol-54, Dec. 1966, pp.1920-1931.

26. Mandelbaum, D. "Error correction in residue arithmetic", IEEE trans. on Comp. Vol-C-21, June 1972, pp.538-545.

27. Barsi, F. and Maestrini, P. "Error correcting properties of redundant residue number systems", IEEE trans. Computer, Vol-C-22, March 1973, pp.307-315.

28. Szabo, N.S. and Tanaka, R.I. "Residue Arithmetic and its application to computer Technology", McGraw Hill Book Company, 1967.

29. Garner, H.L. "The Residue Number System", IRE trans. on Electronic Computers, Vol-EC-8, June 1959, pp.140-147.

30. Keir, Y.A., Cheney, P.W. and Tannenbaum, M. "Division and Overflow Detection in Residue Number Systems", IRE Trans. on Electronic Computers, Vol-EC-11, August 1962, pp.501-507.

31. Kinoshita, E., Kosako H. and Kojima, Y. "Floating Point Arithmetic Algorithms in the Symmetric Residue Number System", IEEE Trans. on Computers, Vol-C-23, No.1., January 1974, pp.9-20.

32. Sasaki, A. "The Basis of Implementation of Additive Operations in the Residue Number System", IEEE Trans. on Computers, Vol-C-17, Nov. 1968, pp. 1066-1078.

33. Svoboda, A. "Digitale Informationswandler", Braunschweig Germany : Vieweg and Sohn, 1960.

34. Flores, Ivan. "The Logic of Computer Arithmetic", Prentice Hall Inc., 1963.

35. Banerjee, D.K. and Brzozowski, J.A. "On Translation Algorithms in Residue Number Systems", IEEE Trans. on Computers, Vol-C-21, Dec. 1972, pp. 1281-1285.

36. Chu, Y. "Digital Computer Design Fundamentals", McGraw-Hill Book Company Inc., 1962.

37. Lewin, D. "Theory and Design of Digital Computers", Nelson, 1972.

38. Deverall, J. "The design of cellular arrays for arithmetic", The Radio and Electronic Engineer, Vol-44, January 1974, pp.21-26.

39. Jayashri, T. and Basu, D. "Cellular arrays for multiplication of signed binary numbers", The Radio and Electronic Engineer, Vol-44, No.1, January 1974, pp.18-20.

40. Wallace, C.S. "A suggestion for a Fast Multiplier", IEEE Trans. on Electronic Computers, Vol-EC-13, Feb. 1964, pp.14-17.

41. Guild, H.H. "Fully Iterative Fast Array for Binary Multiplication and Addition", Electronics Letters, 12th June 1969, Vol-5, No.12, pp.263.

42. Majithia, J.C. and Kitai, R. "An Iterative Array for Multiplication of Signed Binary Numbers", IEEE Trans. on Computers, Vol-C-20, Feb. 1971, pp.214-216.

43. Fenwick, P.M. "Binary Multiplication with Overlapped Addition Cycles", IEEE Trans. on Computers, Vol-C-19, January 1969, pp.71-74.

44. Motorola Semiconductor Products Inc., "High Speed Ripple-Through Arithmetic Processor:, AN487. (Application Note).

45. De Mori, R. "Suggestion for an I.C. Fast Parallel Multiplier", Electronics Letters, 6th February 1969, Vol-5, No.3, pp.50-51.

46. Thomson, P.M. and Belanger, A. "Digital arithmetic units for a high data rate", The Radio and Electronic Engineer, Vol-45, No.3, March 1975, pp.116-120.

47. Bandyopadhyay, S., Basu, S. and Choudhury, A.K. "An Iterative Array for multiplication of signed Binary Numbers", IEEE Trans. on Computers, Vol-C-21, August 1972, pp.921-922.

48. Hallin, T.G. and Flynn, M.J. "Pipelining of Arithmetic Functions", IEEE Trans. on Computers, Vol-21, August 1972, pp.880-886.

49. Habibi, A. and Wintz, P.A. "Fast Multipliers", IEEE Trans. on Computers, Vol-C-19, Feb. 1970, pp.153-1957.

50. Toma, C.I., "Cellular Logic Array for High-Speed Signed Binary Number Multiplication", IEEE Trans on Computers, Vol-C-23, 1975, pp.932-935.

51. Motorola Semiconductor Products Inc., "High Speed Binary Multiplication using the MC 10181", AN-566, 1972. (Application Note).

52. Pezaris, S.D. "A 40-ns 17-Bit by 17-Bit Array Multiplier", IEEE Trans. on Computers, Vol-C-20, April 1972, pp.442-447.

53. Chung, T.J. and Bedrosian, S.D. "Iterative Digital Multiplier Based on Cellular Arrays of ROMs", Electronics Letters, Vol-11, No.18, Sept. 1975, pp.426-428.

54. Knigh, R.B.D. and Stockton, J.R. "Development of Chung and Bedrosian Iterative Cellular Multiplier", Electronics Letters, Vol-12, No.8, April 1976, pp.182-183.

55. "MPY-Series Multipliers", TRW, Redondo Beach, California, 1976.

56. "MPY/HJ family of Multipliers", TRW, Redondo Beach, California, 1978.

57. The TTL DATA Book for Design Engineer, Texas Instrument Inc., Dallas, 1976.

58. "MECL Data Book", Motorola, Phoenix, Arizona, 1976.

59. Jacobsohn, D.H. "A Combinatoric Division Algorithm for fixed Integer Divisors", IEEE Trans. on Computers, pp.608-610.

60. Dean, K.J. "Binary Division Using a Data-Dependent Iterative Array", Electronics Letters, 12th July 1968, Vol-4, No.14, pp.283-284.

61. Guild, H.H. "Some Cellular Logic Arrays for Non-Restoring Binary Division", The Radio and Electronic Engineer, Vol-39, No.6, June 1970, pp. 345-348.

62. Dean, K.J. "Cellular Arrays for Binary Division", Proc. IEE. Vol-117, No.5, May 1970, pp.917-920.

63. Majetha, J.C. "Nonrestoring Division Using a Cellular Array", Electronics Letters, 14th May 1970, Vol-6, No.10, pp.303-304.

64. Cappa, M. and Hamachar, V.C. "An Augmented Iterative Array for High-Speed Binary Division", IEEE Trans. on Computers, Vol-C-22, No.2, Feb. 1973.

65. Wadel, L.B. "Negative Base Number Systems", IRE Trans. on Computers (Corresp.) Vol-EC-6, June 1957, p.123.

66. ——— "Conversion from Conventional Binary to Negative-base Number Representation", IRE Trans. on Electronic Computer, (Corresp.) Vol-EC-10, Dec. 1961, p.779.

67. Zohar, S. "Negative Radix Conversion", IEEE Trans. on Computers, Vol-C-19, March 1970, pp.222-226.

68. Krishnamurthy, E.V. "Complementary Two-way Algorithm for negative radix conversions", IEEE Trans. on Computers, Vol-C-20, No.5, May 1971, pp.543-550.

69. Yen, C.K. "A note on Base -2 Arithmetic Logic", IEEE Trans. on Computers, Vol-C-24, 1975, pp. 325-329.

70. Sankar, P.V., Chakrabarti, S. and Krishnamurthy, E.V. "Arithmetic Algorithms in a negative base", IEEE Trans. on Computers, Vol-C-22, Feb. 1973, pp.120-125.

71. ——— "Deterministic Division Algorithm in a negative base", IEEE Trans. on Computers, Vol-C-22, Feb. 1973, pp.125-128.

72. Rao, G.S., Rao, M.N. and Krishnamurthy, E.V. "Logical Design of a negative binary adder-subtractor", Int. J. Electronics, Vol-36, No.4, 1974, pp.537-542.

73. ——— "A variable shift negabinary multiplier", Int. J. Electronics, 1974, Vol-36, No.6, pp.749-751.

74. Agrawal, D.P. "Arithmetic algorithms in a negative base", IEEE Trans. onComputers, Vol-C-24, October 1975, pp.998-1000.

75. ——— "Negabinary Division and Square-rooting", Digital Process, Vol-1,Autumn 1975, pp.267-274.

76. ——— "Negabinary Carry-look-ahead adder and fast multiplier", Electronics Letters, Vol-10, July 1975, pp.312-313.

77. Waser, S. "State-of-the-Art in High Speed Arithmetic Integrated Circuits", Computer Design, July 1978, pp.67-75.

78. "Signetics Bipolar and MOS Memory Data Manual", 1977. 811 East Arques Avenue, Sunny Vale, California 94086.

79. "Intel Data Catalog", 1975, Intel Corporation, 3065, Bower Ave., Santa Clara, CA - 95051.

80. "Memory Data Book", 1976, National Semiconductor Corp., 2900 Semiconductor Drive, Santa Clara, CA - 95051.

81. "TTL Data Book", 1976, National Semiconductor, 2900 Semiconductor Drive, Santa Clara, California - 95051.

APPENDIX - 1

Some Important Proofs

AP1(a):

If $2^n < M < 2^{n+1}$, $y = n+1$, and $g(X)$ and $g(Y)$ are the group numbers of X and Y respectively, prove that

(i) if $g(X)+g(Y) > y$, overflow will occur.

(ii) if $g(X)+g(Y) < y$, overflow will not occur.

(iii) if $g(X)+g(Y) = y$, overflow may or may not occur. If it occurs, the sign of the product will be incorrect.

Proof:

The multiplication of the lowest numbers in two groups is given by (from Table 4.1)

$$Z = W \cdot 2^{-\delta+g(X)} \cdot W \cdot 2^{-\delta+g(Y)} \quad \ldots\ldots (1)$$

$$Z = W^2 \cdot 2^{-2\delta+g(X)+g(Y)} \quad \ldots\ldots (1a)$$

Let $g(X)+g(Y) = y+1$. If y is an even number, $W=(M)^{\frac{1}{2}}$ and $\delta = y/2+1$. Therefore (1a) can be written as

$$Z = M \cdot 2^{-2(y/2+1)+y+1} = M/2 \quad \ldots\ldots (2a)$$

If y is an odd number, $W=(\frac{M}{2})^{\frac{1}{2}}$ and $\delta = (y+1)/2$. Therefore (1a) can be written as

$$Z = (M/2) \cdot 2^{-2\{(y+1)/2\}+y+1} = M/2 \quad \ldots\ldots (2b)$$

The equations (2a) and (2b) indicate that overflow will occur when the lowest positive numbers in two groups, sum of which is greater than y, is multiplied. [The

equations (2a) and (2b) do not indicate overflow if one of the lowest numbers is negative, since M/2 is within the range of negative half. However, in most cases $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ is not an integer number. Therefore error will be introduced in determination of W, and this error will make Z in equations (2a) and (2b) greater than M/2, and therefore the product of the lowest numbers one of which is negative will overflow. The method of determination of W and construction of group table when $(M)^{\frac{1}{2}}$ or $(\frac{M}{2})^{\frac{1}{2}}$ is not an integer number has been discussed in section 4.2.1.]

Let $g(X)+g(Y) = y$. If y is an even number, then (1a) can be written as

$$Z = M \cdot 2^{-2(y/2+1)+y} = M \cdot 2^{-2} \quad \ldots\ldots \text{(3a)}$$

If y is an odd number, (1a) becomes

$$Z = (M/2) \cdot 2^{-2\{(y+1)/2\}+y} = M \cdot 2^{-2} \quad \ldots\ldots \text{(3b)}$$

The equations (3a) and (3b) indicate overflow will not occur.

The multiplication of the highest numbers of two groups $g(X)$ and $g(Y)$ is given by

$$Z = \left(W \cdot 2^{-\delta+g(X)+1} - 1\right) \cdot \left(W \cdot 2^{-\delta+g(Y)+1} - 1\right)$$

$$Z = W^2 \cdot 2^{-2\delta+g(X)+g(Y)+2} - W \cdot 2^{-\delta+g(X)+1} - W \cdot 2^{-\delta+g(Y)+1} + 1 \quad \ldots\ldots \text{(4)}$$

$$= W^2 \cdot 2^{-2\delta+g(X)+g(Y)+1} \left(2 - \frac{1}{W} \cdot 2^{\delta-g(Y)} - \frac{1}{W} \cdot 2^{\delta-g(X)}\right) + 1$$

$$= W^2 \cdot 2^{-2\delta+g(X)+g(Y)+1} \left(1 + 1 - \frac{1}{W} \cdot 2^{\delta-g(Y)} - \frac{1}{W} \cdot 2^{\delta-g(X)}\right) + 1$$

..... (4a)

Since group 1 contains number 1, overflow will never occur on multiplication of number in group 1 with the number in any other groups. Therefore overflow for the groups greater than 1 will be considered.

Let $g(X)+g(Y) = y$. The equation (4a) can be written as

$$Z = W^2 \cdot 2^{-2\delta+y+1} \left(1 + 1 - \frac{1}{W} \cdot 2^{\delta-g(X)} - \frac{1}{W} \cdot 2^{\delta-g(Y)}\right) + 1 \quad ..... \quad (4b)$$

The minimum value of Z depends on the minimum value of

$$1 - \frac{1}{W} \cdot 2^{\delta-g(X)} - \frac{1}{W} \cdot 2^{\delta-g(Y)} \quad ..... \quad (4c)$$

The expression (4c) is minimised when $g(X)$ or $g(Y)$ is equal to 2. Let $g(X) = 2$. Therefore

$$g(Y) = y-2 \quad ..... \quad (4d)$$

From the value of M, W can be expressed as

$$2^{n/2} < W < 2^{\frac{n+1}{2}} \quad ..... \quad (4e)$$

If y is an even number, then from (4c) and (4e)

$$1 - \frac{1}{W} \cdot 2^{\delta-2} - \frac{1}{W} \cdot 2^{\delta-y+2} < 1 - \frac{1}{2^{n/2}} \cdot 2^{y/2+1-2} - \frac{2^{y/2+1-y+2}}{2^{n/2}}$$

$$= 1 - \frac{2^{(n+1)/2+1-2}}{2^{n/2}} - \frac{2^{(n+1)/2+1-(n+1)+2}}{2^{n/2}}$$

$$= 1 - \frac{1}{2} - 2^{(-2n+1)/2+3}$$

However, for $n > 4$, $2^{(-2n+1)/2+3} < \frac{1}{2}$.

Therefore the minimum value of (4c) is a positive number for $n > 4$.

In other words for $n > 4$

$$Z_{min} > W^2 \cdot 2^{-2\delta+y+1} \qquad \ldots\ldots (4f)$$

or $$Z_{min} > M \cdot 2^{-2(y/2+1)+y+1}$$

$$= M \cdot 2^{-1} \qquad \ldots\ldots (5)$$

The expression (5) indicates that when the sum of group numbers is equal to y, multiplication of the highest numbers in two groups will produce overflow.

In similar way the expressed (5) can be proved to be valid for odd value of y. Again from (4)

$$Z < W^2 \cdot 2^{-2\delta+g(X)+g(Y)+2} \qquad \ldots\ldots (6)$$

Therefore for $g(X)+g(Y) = y$ and even value of y, (from (6))

$$Z < M \qquad \ldots\ldots (6a)$$

If $g(X)+g(Y) = y-1$, and y is an even number then from (6)

$$Z < W^2 \cdot 2^{-2(y/2+1)+y+1+2}$$

or $$Z < M \cdot 2^{-1} \qquad \ldots\ldots (6b)$$

The expression (6) indicates, when the sum of two groups is less than y, the product of the highest number of two groups does not overflow. Similarly it can be proved that expression (6b) is also valid for the odd value of y.

Combining (2a), (2b) and (5), it can be written that if $g(X)+g(Y) > y$, overflow must occur.

Combining (3a), (3b) and (6b), it can be concluded that

if $g(X) +g(Y) < y$, overflow will not occur.

Combining (3a), (3b), (5) and (6a), it can be concluded that

if $g(X)+g(Y) = y$, overflow may or may not occur. If overflow occurs, the product will be of incorrect sign.

AP1(b)

If a_n, a_{n-1}, ..., a_2, a_1 are the mixed radix digits of a number (positive or negative) in the system of moduli m_n, m_{n-1}, ..., m_2, m_1 respectively, then the mixed radix digits b_n, b_{n-1}, ..., b_2, b_1 of the 'counterpart' number (negative or positive number) are given by

$$b_i = \{(m_i - 1) - a_i\} \text{ for } a_i \neq 0 \quad (i=n, n+1, ..., k+1)$$

$$b_k = \{m_k - a_k\} \text{ and } b_p = 0 \text{ , for } a_k \neq 0 \text{ and } a_p = 0$$

$$(p=k-1, k-2, ..., 2, 1).$$

Proof:

The 'counterpart' number is formed by subtraction of the number from zero i.e.

$$(0_n\ 0_{n-1}\ 0_{k+1}\ 0_k\ 0_{k-1} \cdots 0_2\ 0_1)$$
$$- (a_n\ a_{n-1} \cdots a_{k+1}\ a_k\ a_{k-1} \cdots a_2\ a_1)$$

Since a_1 to a_{k-1} are zeros, the k^{th} digit is formed by $(m_k - a_k)$. This subtraction produces a borrow which propagates up to n^{th} digit. Therefore other digits are formed by

$$b_{k+1} = (m_{k+1} - a_{k+1} - 1) = \{(m_{k+1} - 1) - a_{k+1}\}$$
$$b_{k+2} = (m_{k+2} - a_{k+2} - 1) = \{(m_{k+2} - 1) - a_{k+2}\}$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$b_n = (m_n - a_n - 1) = \{(m_n - 1) - a_n\}.$$

$$b_p = 0 \quad \text{for } p = k-1,\ k-2,\ \ldots,\ 2,\ 1.$$

Example:

If $m_3 = 2$, $m_2 = 7$, $m_1 = 3$, and $a_3 = 1$, $a_2 = 5$, $a_1 = 0$, then the MRDs of the 'counterpart' number is given by

$$b_3 = (2-1)-1 = 0$$
$$b_2 = (7-5) = 2$$
$$b_1 = 0.$$

APPENDIX - 2

AP2: Floating Point value of $\frac{1}{B}$

For $B > 1$ and positive,

$$\frac{1}{B} = \frac{\beta^2}{B} \cdot \beta^{-2} \qquad \ldots\ldots (1)$$

Let $$\frac{\beta^2}{B} = Q_B + \frac{r_B}{B} \qquad \ldots\ldots (2)$$

where Q_B and r_B are the quotient and remainder of the left hand side division in (2).

Case (i) If $Q_B \cdot \beta \geq R$, then $\left[\beta/2\right]$ is added to r_B and then divided by B.

$$\frac{r_B + \left[\beta/2\right]}{B} = Q_r + \frac{r_r}{B} \qquad \ldots\ldots (3)$$

The rounded Q_B is given by K where

$$K = Q_B + Q_r \qquad \ldots\ldots (3a)$$

Hence $$\frac{1}{B} \simeq K\beta^{-2} \qquad \ldots\ldots (3b)$$

Case (ii) If $Q_B \cdot \beta < R$, then

$$\frac{1}{B} = \frac{\beta^3}{B} \cdot \beta^{-3} \qquad \ldots\ldots (5)$$

Let $$\frac{\beta^3}{B} = Q_p + \frac{r_p}{B} \qquad \ldots\ldots (6)$$

$\left[\beta/2\right]$ is added to r_p and then divided by B i.e.

$$\frac{r_p + \left[\beta/2\right]}{B} = Q_t + \frac{r_t}{B} \qquad \ldots\ldots (6a)$$

The rounded Q_p is given by H, where

$$H = Q_p + Q_t \qquad \ldots\ldots (6b)$$

Hence $$\frac{1}{B} \simeq H \cdot \beta^{-3} \qquad \ldots\ldots (6c)$$

If $H > R$, then H is taken to be equal to R.

For $B = 1$, $\frac{1}{B} = \beta \cdot B^{-1}$

The floating point value of negative $\frac{1}{B}$ is the negative form of mantissa of positive $\frac{1}{B}$. The exponents of both positive and negative $\frac{1}{B}$ are same.

Table 5.18: Table of mantissa and exponent of $(\frac{1}{B})$ for B = 1 to 120. (11 is the base of exponent)

Decimal Numbers	Residue Numbers Mod-15	Residue Numbers Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mantissa in Residue Form Mod-16	Exponent in Residue Form Mod-15	Exponent in Residue Form Mod-16
1	1	1	11	-1	B	B	E	F
2	2	2	61	-2	D	1	D	E
3	3	3	40	-2	8	A	D	E
4	4	4	30	-2	0	E	D	E
5	5	5	24	-2	9	8	D	E
6	6	6	20	-2	5	4	D	E
7	7	7	17	-2	2	1	D	E
8	8	8	15	-2	0	F	D	E
9	9	9	13	-2	D	D	D	E
10	A	A	12	-2	C	C	D	E
11	B	B	11	-2	B	B	D	E
12	C	C	111	-3	6	F	C	D
13	D	D	102	-3	12	6	C	D
14	E	E	95	-3	5	F	C	D
15	F	F	89	-3	E	9	C	D
16	1	0	83	-3	8	3	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mod-16	Exponent in Residue Form Mod-15	Mod-16
17	2	1	78	-3	3	E	C	D
18	3	2	74	-3	E	A	C	D
19	4	3	70	-3	A	6	C	D
20	5	4	67	-3	7	3	C	D
21	6	5	63	-3	3	F	C	D
22	7	6	61	-3	1	D	C	D
23	8	7	58	-3	D	A	C	D
24	9	8	55	-3	A	7	C	D
25	A	9	53	-3	8	5	C	D
26	B	A	51	-3	6	3	C	D
27	C	B	49	-3	4	1	C	D
28	D	C	48	-3	3	0	C	D
29	E	D	46	-3	1	E	C	D
30	0	E	44	-3	E	C	C	D
31	1	F	43	-3	D	B	C	D
32	2	0	42	-3	C	A	C	D
33	3	1	40	-3	A	8	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mod-16	Exponent in Residue Form Mod-15	Mod-16
34	4	2	39	-3	9	7	C	D
35	5	3	38	-3	8	6	C	D
36	6	4	37	-3	7	5	C	D
37	7	5	36	-3	6	4	C	D
38	8	6	35	-3	5	3	C	D
39	9	7	34	-3	4	2	C	D
40	A	8	33	-3	3	1	C	D
41	B	9	32	-3	2	0	C	D
42	C	A	32	-3	2	0	C	D
43	D	B	31	-3	1	F	C	D
44	E	C	30	-3	0	E	C	D
45	0	D	30	-3	0	E	C	D
46	1	E	29	-3	E	D	C	D
47	2	F	28	-3	D	C	C	D
48	3	0	28	-3	D	C	C	D
49	4	1	27	-3	C	B	C	D
50	5	2	27	-3	C	B	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mod-16	Exponent in Residue Form Mod-15	Mod-16
51	6	3	26	-3	B	A	C	D
52	7	4	26	-3	B	A	C	D
53	8	5	25	-3	A	9	C	D
54	9	6	25	-3	A	9	C	D
55	A	7	24	-3	9	8	C	D
56	B	8	24	-3	9	8	C	D
57	C	9	23	-3	8	7	C	D
58	D	A	23	-3	8	7	C	D
59	E	B	23	-3	8	7	C	D
60	0	C	22	-3	7	6	C	D
61	1	D	22	-3	7	6	C	D
62	2	E	21	-3	6	5	C	D
63	3	F	21	-3	6	5	C	D
64	4	0	21	-3	6	5	C	D
65	5	1	20	-3	5	4	C	D
66	6	2	20	-3	5	4	C	D
67	7	3	20	-3	5	4	C	D
68	8	4	20	-3	5	4	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mod-16	Exponent in Residue Form Mod-15	Mod-16
69	9	5	19	-3	4	3	C	D
70	A	6	19	-3	4	3	C	D
71	B	7	19	-3	4	3	C	D
72	C	8	18	-3	3	2	C	D
73	D	9	18	-3	3	2	C	D
74	E	A	18	-3	3	2	C	D
75	0	B	18	-3	3	2	C	D
76	1	C	18	-3	3	2	C	D
77	2	D	17	-3	2	1	C	D
78	3	E	17	-3	2	1	C	D
79	4	F	17	-3	2	1	C	D
80	5	0	17	-3	2	1	C	D
81	6	1	16	-3	1	0	C	D
82	7	2	16	-3	1	0	C	D
83	8	3	16	-3	1	0	C	D
84	9	4	16	-3	1	0	C	D
85	A	5	16	-3	1	0	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-16	Mod-16	Exponent in Residue Form Mod-15	Mod-16
86	B	6	15	-3	0	F	C	D
87	C	7	15	-3	0	F	C	D
88	D	8	15	-3	0	F	C	D
89	E	9	15	-3	0	F	C	D
90	0	A	15	-3	0	F	C	D
91	1	B	15	-3	0	F	C	D
92	2	C	14	-3	E	E	C	D
93	3	D	14	-3	E	E	C	D
94	4	E	14	-3	E	E	C	D
95	5	F	14	-3	E	E	C	D
96	6	0	14	-3	E	E	C	D
97	7	1	14	-3	E	E	C	D
98	8	2	14	-3	E	E	C	D
99	9	3	13	-3	D	D	C	D
100	A	4	13	-3	D	D	C	D
101	B	5	13	-3	D	D	C	D

Continued

Decimal Numbers	Residue Numbers Mod-15	Mod-16	Mantissa in Decimal Form	Exponent in Decimal Form	Mantissa in Residue Form Mod-15	Mod-16	Exponent in Residue Form Mod-15	Mod-16
102	C	6	13	-3	D	D	C	D
103	D	7	13	-3	D	D	C	D
104	E	8	13	-3	D	D	C	D
105	0	9	13	-3	D	D	C	D
106	1	A	13	-3	D	D	C	D
107	2	B	12	-3	C	C	C	D
108	3	C	12	-3	C	C	C	D
109	4	D	12	-3	C	C	C	D
110	5	E	12	-3	C	C	C	D
111	6	F	12	-3	C	C	C	D
112	7	0	12	-3	C	C	C	D
113	8	1	12	-3	C	C	C	D
114	9	2	12	-3	C	C	C	D
115	A	3	12	-3	C	C	C	D
116	B	4	11	-3	B	B	C	D
117	C	5	11	-3	B	B	C	D
118	D	6	11	-3	B	B	C	D
119	E	7	11	-3	B	B	C	D
120	0	8	-11	-3	4	5	C	D

APPENDIX - 4

AP4: Error Analysis of Floating Point Addition and Multiplication Operations

In case of floating point addition/subtraction, and multiplication, the errors are introduced during rounding process in normalisation. In floating point division error is introduced from two sources - (i) error from approximation of floating point value of $\frac{1}{B}$ and (ii) the error from floating point multiplication of Fl(A.K). However, errors only due to floating addition and multiplication will be evaluated, assuming the exponents of the operands are zero.

AP4(a): Error in floating point additon

The actual value of Z is given by

$$Z = c_2\ \beta + c_1 \qquad \text{(from (4.11))}.$$

The positive error occurs if

$$(c_2 \cdot \beta + c_1) > R \quad \text{or} \quad c_2 \cdot \beta \geq R$$

and

$$\begin{cases} c_1 > \left[\beta/2\right] & \text{if } \beta \text{ is odd.} \\ \text{or} & \\ c_1 \geq \left[\beta/2\right] & \text{if } \beta \text{ is even.} \end{cases}$$

Under these conditions the normalised rounded Z (Fig 4.6) becomes

$$Z' = (c_2 + 1)\beta$$

Therefore positive error $e+ = Z' - Z$

$$= \beta - c_1 \quad \ldots\ldots (1)$$

Case (i) If β is odd, then

$$\left[\beta/2\right] = \frac{\beta-1}{2}$$

$$\text{or} \quad \beta = 2 \cdot \left[\beta/2\right] + 1 \quad \ldots\ldots (1a)$$

Under error condition since $c_1 > \left[\beta/2\right]$, and c_1 is an integer, therefore the minimum value of c_1 can be given by

$$(c_1)_{min} = \left[\beta/2\right] + 1 \quad \ldots\ldots (1b)$$

For $c_1 = (c_1)_{min}$, $e+$, in equation (1), is maximised, therefore

$$(e+)_{max} = \beta - (c_1)_{min}$$

$$= \beta - \left[\beta/2\right] - 1$$

$$= 2 \cdot \left[\beta/2\right] + 1 - \left[\beta/2\right] - 1$$

$$= \left[\beta/2\right] \quad \ldots\ldots (2)$$

Case (ii) If β is even, then

$$\left[\beta/2\right] = \beta/2$$

$$\text{or} \quad \beta = 2 \cdot \left[\beta/2\right] \quad \ldots\ldots (3)$$

Under error condition since $c_1 \geq \left[\beta/2\right]$ and c_1 is an integer, therefore the minimum value of c_1, in this case, is given by

$$(c_1)_{min} = \left[\beta/2\right] \quad \ldots\ldots (4)$$

For $c_1 = (c_1)_{min}$, $e+$, in equation (1), is maximised; therefore

$$(e+)_{max} = \beta - (c_1)_{min}$$

$$= 2 \cdot \left[\beta/2\right] - \left[\beta/2\right]$$

$$= \left[\beta/2\right] \qquad \ldots\ldots (5)$$

Therefore in both cases, positive maximum error is same.

The relative positive error $U(e+)$ is given by

$$U(e+) = \frac{e+}{c_2\ \beta + c_1}$$

The $U(e+)$ is maximised when $e+ = (e+)_{max}$ and $c_2\ \beta + c_1 = R$. Therefore the maximum value of $U(e+)$ is given by

$$U(e+)_{max} = \frac{(e+)_{max}}{R} = \frac{\left[\beta/2\right]}{R} \qquad \ldots\ldots (6)$$

The negative error occurs if

$$(c_2 \cdot \beta + c_1) > R \quad \text{or} \quad c_2 \cdot \beta \geq R$$

and

$$\begin{cases} c_1 \leq \left[\beta/2\right] & \text{if } \beta \text{ is odd.} \\ \quad \text{or} & \\ c_1 < \left[\beta/2\right] & \text{if } \beta \text{ is even.} \end{cases}$$

Under those conditions the normalised Z becomes

$$Z' = c_2\ \beta$$

Therefore the absolute value of negative error is given by

$$|e-| = |Z' - Z| = c_1$$

If β is odd, $|e-| \leq \left[\beta/2\right]$ and the maximum relative negative error, $U(e-)_{max}$ is given by

$$U(e-)_{max} = \frac{\left[\beta/2\right]}{R} \qquad \ldots\ldots (7)$$

If β is even, $|e-| < \lceil \beta/2 \rceil$ and maximum $|e-|$ is given by

$$(e-)_{max} = \lceil \beta/2 \rceil - 1. \quad \text{(Since } |e-| \text{ is an integer.)}$$

The maximum relative negative error, when β is even, is given by

$$U(e-)_{max} = \frac{\lceil \beta/2 \rceil - 1}{R} \qquad \ldots\ldots (8)$$

From (6), (7) and (8), it is clear that the maximum error, Max(e) is given by (6) or (7), therefore

$$\text{Max}(e) = U(e+)_{max} = \frac{\lceil \beta/2 \rceil}{R} \qquad \ldots\ldots (9)$$

AP4(b): Error in floating point multiplication

Actual value of Z is given by

$$Z = c_3 \beta^2 + c_2 \beta + c_1 \qquad \text{(from 4.14).}$$

Case (i) If $c_3 = 0$, Z is reduced to Z_R, where

$$Z_R = c_2 \beta + c_1 \qquad \ldots\ldots (10)$$

In this case, maximum error is same as the maximum error in floating point addition.

Case (ii) If c_3 and c_1 are nonzero, then positive error will occur if

$$\{(c_3 \cdot \beta + c_2) \text{ or } c_3 \cdot \beta\} \geq R$$

and $\begin{cases} c_2 > \lceil \beta/2 \rceil & \text{if } \beta \text{ is odd.} \\ \text{or} \\ c \geq \lceil \beta/2 \rceil & \text{if } \beta \text{ is even.} \end{cases}$

In this case the normalised rounded Z (Fig 4.7) becomes

$$Z' = (c_3 + 1)\beta^2$$

Therefore positive error $e+ = Z' - Z$

$$= \beta^2 - (c_2 \cdot \beta + c_1) \quad \ldots\ldots (11)$$

$$\text{or } e+ < \beta^2 - c_2 \cdot \beta \quad \ldots\ldots (12)$$

Case (iii) If c_3 is nonzero and c_1 is zero, then positive error will occur if

$$c_3 \cdot \beta + c_2 > R \quad \text{or} \quad c_3 \cdot \beta \geq R$$

$$\text{and} \begin{cases} c_2 > \lceil \beta/2 \rceil & \text{if } \beta \text{ is odd.} \\ c_2 \geq \lceil \beta/2 \rceil & \text{if } \beta \text{ is even.} \end{cases}$$

Under these conditions, the expression for positive error can be derived from (11) as

$$e'+ = \beta^2 - c_2 \cdot \beta \quad \ldots\ldots (13)$$

From (13) and (12), it is seen that $e' > e$. Therefore to evaluate maximum positive error, the expression (13) will be used.

$e+$ is maximised if $c_2 = (c_2)_{min}$.

If <u>β is odd</u>, $c_2 > \lceil \beta/2 \rceil$ (under the error conditions given above). Therefore the minimum value of c_2 can be written as

$$(c_2)_{min} = \lceil \beta/2 \rceil + 1 \quad \ldots\ldots (14)$$

If <u>β is even</u>, $c_2 \geq \lceil \beta/2 \rceil$, therefore $(c_2)_{min}$, in this case is (under error condition)

$$(c_2)_{min} = \lceil \beta/2 \rceil \quad \ldots\ldots (15)$$

Therefore if <u>β is odd</u>, the maximum positive error is given by

$$(e'+)_{max} = \beta^2 - (c_2)_{min} \cdot \beta \qquad \text{from (13)}$$

$$= \beta^2 - \left(\left[\beta/2\right] + 1 \right) \beta \qquad \text{from (14)}$$

$$= \left(2 \cdot \left[\beta/2\right] + 1 \right)^2 - \left(\left[\beta/2\right] + 1 \right) \left(2 \cdot \left[\beta/2\right] + 1 \right) \qquad \text{from (1a)}$$

Therefore when β is odd, $(e'+)_{max} = \left[\beta/2\right] \cdot \left(2 \cdot \left[\beta/2\right] + 1 \right)$ ..... (16)

When β is even, the maximum positive error is given by

$$(e'+)_{max} = \beta^2 - (c_2)_{min} \cdot \beta \qquad \text{from (13)}$$

$$= \beta^2 - \left[\beta/2\right] \cdot \beta \qquad \text{from (15)}$$

$$= \left(2 \cdot \left[\beta/2\right] \right)^2 - \left[\beta/2\right] \cdot \left(2 \cdot \left[\beta/2\right] \right) \qquad \text{from (3)}$$

$$= 2\left(\left[\beta/2\right] \right)^2 \qquad \text{..... (17)}$$

The negative error occurs if

$$\{ (c_3 \cdot \beta + c_2) \text{ or } c_3 \cdot \beta \} \geq R$$

and $\begin{cases} c_2 \leq \left[\beta/2\right] & \text{if } \beta \text{ is odd.} \\ c_2 < \left[\beta/2\right] & \text{if } \beta \text{ is even.} \end{cases}$

(For c_3 and c_1 are nonzero)

Under these conditions, the normalised rounded Z becomes

$$Z' = c_3 \cdot \beta^2$$

The absolute value of negative error $|e-|$ is given by

$$|e-| = |Z' - Z| = c_2 \cdot \beta + c_1 \quad .....(18)$$

The maximum value of c_1, $(c_1)_{max} = (\beta-1)$ ..... (19)

If β is odd, under the above condition,

$$c_2 \leq \left[\beta/2\right]$$

Therefore the maximum value of c_2 can be written as

$$(c_2)_{max} = \left[\beta/2\right] \quad ..... (19a)$$

Therefore the maximum absolute value of negative error, when β is odd, is given by

$$\begin{aligned}(e-)_{max} &= (c_2)_{max} \cdot \beta + (c_1)_{max} \\ &= \left[\beta/2\right] \cdot \beta + (\beta-1) \quad \text{from (19a) and (19)} \\ &= \left[\beta/2\right] \cdot \left(2 \cdot \left[\beta/2\right]+1\right) + \left(2 \cdot \left[\beta/2\right]+1-1\right) \\ &= 2\left(\left[\beta/2\right]\right)^2 + 3 \cdot \left[\beta/2\right] \quad ..... (20)\end{aligned}$$

If β is even, under the above condition

$$c_2 < \left[\beta/2\right].$$

Therefore the maximum value of c_2 can be written as

$$(c_2)_{max} = \left[\beta/2\right] - 1 \quad ..... (21)$$

Therefore the maximum absolute value of negative error, when β is even, is given by

$$\begin{aligned}|e-|_{max} &= |c_2|_{max} \cdot \beta + |c_1|_{max} \\ &= \left(\left[\beta/2\right]-1\right) \cdot \beta + (\beta-1) \quad \text{from (19) and (21)} \\ &= \left(\left[\beta/2\right]-1\right) \cdot 2 \cdot \left[\beta/2\right] + 2 \cdot \left[\beta/2\right] - 1 \\ &\qquad \text{from (3)}\end{aligned}$$

$$= 2\left(\left[\beta/2\right] \right)^2 - 1 \qquad \ldots\ldots (22)$$

Comparing (16) and (20), the maximum error, when <u>β is odd</u> is given by (20), therefore

$$(e(o))_{max} = 2\left(\left[\beta/2\right] \right)^2 + 3 \cdot \left[\beta/2\right] \qquad \ldots\ldots (23)$$

Comparing (22) and (17), the maximum error, when <u>β is even</u>, is given by (17), therefore

$$(e(e))_{max} = 2\left(\left[\beta/2\right] \right)^2 \qquad \ldots\ldots (24)$$

When c_3 is nonzero, the minimum value of Z is

$$(Z)_{min} = R \qquad \ldots\ldots (25)$$

The maximum relative error, when <u>β is odd</u>, is given by

$$(U(eo))_{max} = \frac{(e(o))_{max}}{(Z)_{min}}$$

$$= \frac{2\left(\left[\beta/2\right] \right)^2 + 3 \cdot \left[\beta/2\right]}{R} \qquad \ldots\ldots (26)$$

from (23) and (25)

The maximum relative error, when <u>β is even</u>, is given by

$$(U(ee))_{max} = \frac{(e(e))_{max}}{(Z)_{min}}$$

$$= \frac{2\left(\left[\beta/2\right] \right)^2}{R} \qquad \ldots\ldots (27)$$

from (24) and (25).

APPENDIX - 5

AP5: Mixed Radix Conversion

For a set of moduli m_i (i=1,2,3, ..., n) a number X can be expressed in mixed radix form [28] as

$$X = a_n \prod_{i=1}^{n-1} + \ldots\ldots + a_3 m_2 m_1 + a_2 m_1 + a_1 \ldots\ldots \quad (1)$$

where a_i (i=1,2,3, ..., n) are the mixed radix digits. The digits can be found by

$$a_1 = |X|_{m_1}$$

$$a_2 = \left|\frac{X-a_1}{m_1}\right|_{m_2}$$

$$a_3 = \left|\frac{\frac{X-a_1}{m_1} - a_2}{m_2}\right|_{m_3}$$

In this way, by successive subtraction and division, all the digits can be obtained. Conversion of a number X (residue form) to mixed radix form (shown in (1)) is called mixed radix conversion (MRC).

Example:

For $m_3 = 16$, $m_2 = 15$, $m_1 = 13$, find mixed radix digits of $X \longrightarrow \{7,5,8\}$.

Solution:

Moduli:	16	15	13	
Residue of X	7	5	8	$a_1 = 8$
Subtract a_1	8	8		
	15	12		
* Multiply by $\left\|\frac{1}{13}\right\|_{m_i}$	5	7		
	11	9		$a_2 = 9$
Subtract a_2	9			
	2			
* Multiply by $\left\|\frac{1}{15}\right\|_{16}$	15			
	14			$a_3 = 14$

From example, it can be shown that total parallel residue operations required are -
(i) (n-1) subtractions and (ii) (n-1) multiplication. If these operations are carried out with PROMs (by look up table) then subtractions and multiplication can be over-lapped (since the multipliers are constant numbers). In this case total operations are (n-1). In other words, delay is (n-1) PROM-delay (access time).

The decimal value (DV) of $X = 14x15x13 + 9x13+8$

$= 2855$

* $\left[\left|\frac{1}{13}\right|_{m_i}, \left|\frac{1}{15}\right|_{16}\right.$ denotes multiplicative inverse of 13 and 15 with respective to m_i (15 and 16) and 16 respectively.]

APPENDIX - 6

PROGRAM LISTINGS

Appendix-6 consists of the program listings for the RAC and is divided into three parts - AP6(a), AP6(b) and AP6(c). The AP6(a) contains the program listings for integer arithmetic operations together with initialisation of the ports of the microcomputer, decoding of the operation codes etc.; the AP6(b) contains the program listings for floating point arithmetic operations and the AP6(c) contains the program listings for the conversion of residue numbers to decimal numbers.

The programs are the subroutines nested together. The labels of the subroutines and routines, in alphabetical order, are given below.

[S = Subroutine : R = Routine]

* Resides in the Monitor Prom of the SDK microcomputer.

AP-6(a): Listings of Integer Arithmetic Operations

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*************************************************************
;PROGRAME FOR INTEGER ADDITION,SUBTRACTION
;MULTIPLICATION AND COMPARISON.
;RESIDUE ARITHMETIC (MODULI: 16 AND 15)
;*************************************************************
;
;*************************************************************
;PROGRAME STARTS FROM LOC. #0800 OF THE SDK-80
;MICROCOMPUTER.
;*************************************************************
;
;*************************************************************
                        ORG #800
;-------------------------------------------------------------
;                       FROM SETS
       CNVBN: SET #01DA
        ECHO: SET #01F4
       GETCH: SET #021B
       NMOUT: SET #02C3
         DIV: SET #09B0
         FAD: SET #0C5E
         FSU: SET #0CEB
         FMU: SET #0D20
         FDI: SET #0400
        IBCD: SET #0ED0
;*************************************************************
;
;*************************************************************
;FOUR OF THE SUBROUTINES CALLED ARE STORED
;IN THE MONITOR 'FROM' OF THE SDK-80.
;FOUR SUBROUTINES ARE - 1)'CNVBN' -CONVERTS
;ASCII REPRESENTATION OF A NUMBER INTO ITS
;BINARY VALUE (HEX NUMBER).
;2)'ECHO' -TAKES SINGLE CHARACTER AS INPUT
;AND, VIA MONITOR SENDS THAT CHARACTER TO
;THE USER TERMINAL.
;3)'GETCH' -RETURNS THE CHARACTER IN THE
;INPUT STREAM TO THE CALLING PROGRAME.
;4)'NMOUT' -CONVERTS 8-BIT UNSIGNED INTEGER
;IN A-REGISTER INTO TWO ASCII CHARACTERS AND
;SENDS TO THE CONSOLE AT THE CURRENT PRINT
;POSITION OF THE CONSOLE.
;*************************************************************
```

```
;*************************************************************
;THIS ROUTINE INITIALISES THE PORTS OF PPI
;A-PORT AND C-PORT (UPPER) ARE SET TO OUTPUT
;MODE. B-PORT AND C-PORT (LOWER) ARE SET TO
;INPUT MODE.
;-------------------------------------------------------------
INI: LXI SP,#13FF   ;STACK POINTER.
     MVI A,#83      ;'#83' IS THE CONTROL DATA
                    ;WHICH SETS THE PORTS TO
                    ;AVOBE MODES.
     OUT #17        ;SEND IT TO PPI.
     JMP TEST
;*************************************************************
;
;*************************************************************
;THESE ARE THE SUBROUTINES WHICH OUTPUT THE FOLLOWING
;CHARACTERS:-'CR','LF','O','M','E','SPACE','-','+',
;'=','<','>','?' .
;-------------------------------------------------------------
;   '#' BEFORE A NUMBER INDICATES HEX NUMBER.
;-------------------------------------------------------------
CAR: MVI C,#0D      ;'#0D' IS THE ASCII CODE FOR
                    ;CARRIAGE RETURN.
     CALL ECHO      ;SIGNAL.
     MVI C,#0A      ;'#0A' IS THE ASCII CODE FOR
                    ;LINE FEED.
SET: CALL ECHO      ;SIGNAL.
     RET
OPC: MVI C,#4F      ;'#4F' IS THE ASCII CODE FOR
                    ;THE LETTER 'O'.
TTT: CALL ECHO      ;DISPLAY.
     JMP EQ
NEG: MVI C,#2D      ;'#2D' IS THE ASCII CODE FOR
                    ;THE SIGN '-'.
     JMP SET
POS: MVI C,#2B      ;'#2B' IS THE ASCII CODE FOR
                    ;THE SIGN '+'.
     JMP SET
MNT: MVI C,#4D      ;'#4D' IS THE ASCII CODE FOR
                    ;THE LETTER 'M'.
     JMP TTT
EPT: MVI C,#45      ;'#45' IS THE ASCII CODE FOR
                    ;THE LETTER 'E'.
     JMP TTT
 EQ: MVI C,#3D      ;'#3D' IS THE ASCII CODE FOR
                    ; THE SIGN '='.
     JMP SET
 GT: MVI C,#3E      ;'#3E' IS THE ASCII CODE FOR
                    ;THE SIGN '>'.
     JMP SET
 LT: MVI C,#3C      ;'#3C' IS THE ASCII CODE FOR
                    ;THE SIGN '<'.
```

```
            JMP SET
      BLN:  MVI C,#20     ;'#20' IS THE ASCII CODE FOR
                          ;THE 'SPACE'.
            JMP SET
     XTAR:  MVI C,#3F     ;'#3F' IS THE ASCII CODE FOR
                          ;THE SIGN '?'.
            JMP SET
;*************************************************
;
;*************************************************
;THIS SUBROUTINE GETS TWO ASCII CHARACTERS FROM
;CONSOLE DEVICE AND CONVERTS THEM INTO ONE BYTE
;(8-BITS HEX) DATA.
;-------------------------------------------------
      GET:  CALL GETCH    ;GET ONE ASCII CHARACTER.
            CALL ECHO     ;DISPLAY.
            CALL CNVBN    ;CONVERT IT INTO ITS.
                          ;BINARY VALUE.
            RLC
            RLC
            RLC
            RLC           ;ALTOGETHER FOUR LEFT SHIFTS.
            STA #1360     ;STORE AT LOC. #1360.
            CALL GETCH    ;GET ANOTHER ASCII CHARACTER
            CALL ECHO     ;DISPLAY.
            CALL CNVBN    ;CONVERT IT INTO ITS
                          ;BINARY VALUE.
            MOV C,A       ;MOVE IT TO C-REGISTER.
            LDA #1360     ;LOAD A- REGISTER FROM
                          ;LOC. #1360
            ADD C         ;ADD WITH C-REGISTER.
            RET
;*************************************************
;
;*************************************************
;THIS SUBROUTINE LOADS OPERANDS AND OPERATION
;CODE TO RESIDUE ARITHMETIC UNIT(RAU) AND
;GETS THE RESULT OUT OF IT.
;-------------------------------------------------
     LOAD:  MVI A,#60     ;'#60' IS THE ADDRESS
                          ;OF THE LATCH ASSIGNED
                          ;FOR FIRST OPERAND (X).
            OUT #16       ;SEND TO C-PORT (UPPER).
            LDA #1361     ;LOAD FIRST OPERAND FROM
                          ;LOC.#1361.
            OUT #14       ;SEND IT TO A-PORT.
            MVI A,#50     ;'#50' IS THE ADDRESS
                          ;OF THE LATCH ASSIGNED
                          ;FOR SECOND OPERAND (Y).
            OUT #16       ;SEND TO C-PORT (UPPER).
            LDA #1363     ;LOAD SECOND OPERAND FROM
                          ;LOC.#1363.
            OUT #14       ;SEND IT TO A-PORT.
```

```
                MVI A,#30     ;'#30' IS THE ADDRESS
                              ;OF THE LATCH ASSIGNED
                              ;FOR OPERATION CODE.
                OUT #16       ;SEND TO C-PORT (UPPER).
                LDA #1365     ;LOAD OPERATION CODE FROM
                              ;LOC.#1365.
                OUT #14       ;SEND IT TO A-PORT.
                IN #15        ;GET THE RESULT FROM B-PORT.
                STA #1370     ;STOE IT AT LOC.#1370.
                MOV B,A       ;MOVE IT TO B-REG. .
                IN #16        ;GET SIGN AND OVERFLOW
                              ;INFORMATION FROM C-PORT (LOWER).
                STA #1371     ;STORE IT AT LOC.#1371.
                RET
;*************************************************************
;
;*************************************************************
;THESE ARE THE SUBROUTINES FOR ADDITION,SUBTRACTION
;MULTIPLICATION,COMPARISON, AND FOR GETTING
;FIRST APPROXIMATION FOR INTEGER DIVISION.
;-------------------------------------------------------------
          MU: MVI  A,#0B    ;'#0B' IS THE CODE FOR
                            ;MULTIPLICATION.
         JPM: STA #1365     ;STORE AT #1365.
              CALL LOAD
              RET
          AD: MVI A,#06     ;'#06' IS THE CODE FOR
                            ;ADDITION.
              JMP JPM
          SU: MVI A,#05     ;'#05' IS THE CODE FOR
                            ;SUBTRACTION.
              JMP JPM
          CM: MVI A,#40     ;'#40' IS THE CODE FOR
                            ;COMPARISON (UNSIGNED
                            ;NUMBER).
              JMP JPM
          CV: MVI A,#80     ;'#80' IS THE CODE FOR
                            ;COMPARISON (SIGNED
                            ;NUMBER).
              JMP JPM
          DI: MVI A,#01     ;'#01' IS THE CODE FOR
                            ;GETTING APPROXIMATE
                            ;QUOTIENT (FIRST
                            ;APPROXIMATION).
              JMP JPM
;*************************************************************
;
;*************************************************************
;THIS SUBROUTINE SIGNALS TO THE CONSOLE DEVICE
;TO GET OPERANDS.
;-------------------------------------------------------------
        TAKE: CALL MNT      ;SIGNAL 'M='.
              CALL GET      ;GET ONE BYTE DATA.
```

```
              STA #1361     ;STORE AT #1361.
              CALL BLN      ;GIVE SPACE.
              CALL MNT      ;SIGNAL 'M='.
              CALL GET      ;GET ANOTHER BYTE OF DATA.
              STA #1363     ;STORE AT #1363.
              RET
;*************************************************************
;
;*************************************************************
;THIS SUBROUTINE SIGNALS OVERFLOW (?),POSITIVE (+),
;NEGATIVE (-),AND SENDS THE RESULT TO THE CONSOLE
;DEVICE.
;-------------------------------------------------------------
       OSIGN: LDA #1371     ;LOAD SIGN AND OVERFLOW
                            ;INFORMATION FROM #1371.
              MOV D,A       ;SAVE IT IN D-REG. .
              ANI #02       ;TEST IF RESULT
                            ;OVERFLOWS.
              JZ COR        ;IF NOT, JUMP TO 'COR'.
              CALL XTAR     ;OTHERWISE SIGNAL '?' .
        COR:  MOV A,D       ;GET BACK INFORMATION
                            ;FROM D-REG. .
              ANI #01       ;TEST IF RESULT IS
                            ;NEGATIVE..
              JZ PSI        ;IF NOT, JUMP TO 'PSI'.
              CALL NEG      ;OTHERWISE SIGNAL '-' .
              JMP PSI+3
        PSI:  CALL POS      ;SIGNAL '+' .
              LDA #1370     ;LOAD THE RESULT FROM #1370.
              CALL NMOUT    ;CONVERT THE RESULT
                            ;INTO ACSII FORM.
              RET
;*************************************************************
;
;*************************************************************
;THIS SUBROUTINE SIGNALS COMPARISON CHARACTERS
;TO THE CONSOLE DEVICE.
;-------------------------------------------------------------
       CCHR:  LDA #1370     ;LOAD THE RESULT  OF
                            ;COMPARISION.
              MOV B,A       ;SAVE IN B-REGISTER.
              ANI 01        ;TEST WHETHER LESSER.
              JNZ LLT       ;IF YES, JUMP TO 'LLT' .
              MOV A,B       ;GET THE INFORMATION
                            ;FROM B-REG. .
              ANI #02       ;TEST WHETHER GREATER.
              JNZ GGT       ;IF YES, JUMP TO 'GGT' .
              JMP EEQ       ;OTHERWISE JUMP TO 'EEQ'.
        LLT:  CALL LT       ;SIGNAL '<'.
              RET
        GGT:  CALL GT       ;SIGNAL '>'.
              RET
```

```
        EEQ: CALL EQ      ;SIGNAL '='.
             RET
;*************************************************
;
;*************************************************
;THIS PROGRAME ASKS FOR OPERATION CODES AND TEST
; THE TYPE OF OPERATION REQUIRED.
; CONSOLE DEVICE ISSUES THE FOLLOWING
;CODES:- A (ADDITION),B (SUBTRACTION)
;C (MULTIPLICATION),D (DIVISON),E (COMPARISION
;OF UNSIGNED NUMBER),F (COMPARISON OF SIGNED
;NUMBER),G (FLOATING POINT ADDITION),H (FLOATING
;POINT SUBTRACTION),I (FLOATING POINT MULTIFLI-
;CATION),J (FLOATING POINT DIVISION).
;-------------------------------------------------
       TEST: CALL CAR     ;CARRIAGE RETURN AND
                          ;LINE FEED.
             CALL OPC     ;SIGNAL O= TO GET
                          ;OPERATION CODE.
             CALL GETCH   ;GET OPERATION CODE.
             CALL ECHO    ;DISPLAY.
             MOV D,C      ;SAVE IT  AT D-REG. .
             CALL BLN     ;GIVE SPACE
             MOV A,D
         AP: CPI 41H       ;TEST IF 'A' .
             JZ ADD       ;IF YES JUMP TO 'ADD'.
         SB: CPI 42H       ;OTHERWISE, TEST IF 'B'.
             JZ SUB       ;IF YES JUMP TO 'SUB'.
         MM: CPI 43H       ;OTHERWISE TEST IF 'C'.
             JZ MUL       ;IF YES JUMP TO 'MUL'.
        DSV: CPI 44H       ;OTHERWISE TEST IF 'D'.
             JZ DIV       ;IF YES JUMP TO 'DIV'.
       COUS: CPI 45H       ;OTHERWISE TEST IF 'E'.
             JZ COMU      ;IF YES JUMP TO 'COMU'.
       COSI: CPI 46H       ;OTHERWISE TEST IF 'F'.
             JZ COMS      ;IF YES JUMP TO 'COMS'.
         FA: CPI 47H       ;OTHERWISE TEST IF 'G'.
             JZ FAD       ;IF YES JUMP TO 'FAD'.
         FS: CPI 48H       ;OTHERWISE TEST IF 'H'.
             JZ FSU       ;IF YES JUMP TO 'FSU'.
         FM: CPI 49H       ;OTERWISE TEST IF 'I'.
             JZ FMU       ;IF YES JUMP TO 'FMU'.
         FD: CPI 4AH       ;OTHERWISE TEST IF 'J'.
             JZ FDI       ;IF YES JUMP TO 'FDI'.
             JMP TEST      ;OTHERWISE ASK FOR
                          ;OPERATION CODE.
;*************************************************
```

```
;*************************************************
;THE MAIN PROGRAMES FOR INTEGER ARITHMETIC OPERATIONS.
;-------------------------------------------------
        ADD: CALL TAKE    ;GET TWO OPERANDS.
             CALL AD      ;DO ADDITION.
        A12: CALL CAR     ;CARRIAGE RETURN AND
                          ;LINE FEED.
             CALL OSIGN   ;SIGNALS OVERFLOW (IF ANY)
                          ;SIGN AND DISPLAY
                          ;THE RESULT
                          ;IN RESIDUE FORM.
             JMP IBCD     ;DISPLAY THE RESULT
                          ;IN DECIMAL FORM.
        SUB: CALL TAKE    ;GET OPERANDS.
             CALL SU      ;DO SUBTRACTION.
             JMP A12
        MUL: CALL TAKE    ;GET OPERANDS.
             CALL MU      ;DO MULTIPLICATION.
             JMP A12
       COMS: CALL TAKE    ;GET OPERANDS.
             CALL CV      ;COMPARE THE SIGNED
                          ;NUMBERS.
             JMP A30
       COMU: CALL TAKE    ;GET OPERANDS.
             CALL CM      ;COMPARE THE UNSIGNED
                          ;NUMBERS.
        A30: CALL CAR
             CALL CCHR    ;SIGNAL COMPARISION
                          ;CHARACTER.
             JMP TEST     ;ASK FOR OPERATION
                          ;CODE.
;*************************************************
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                          END
```

```
;^D$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;****************************************************
;PRORAME FOR INTEGER RESIDUE DIVISION.
;RESIDUE ARITHMECTIC (MODULI: 16 &15 ).
;****************************************************
;THIS PROGRAME STARTS FROM LOC. #09B0 OF
;THE SDK-80 MICROCOMPUTER.
;****************************************************
;
;****************************************************
                    ORG #09B0
;****************************************************
;                   PROM SETS
;====================================================
        MU: SET #0890
        AD: SET #0899
        SU: SET #089E
        CM: SET #08A3
        DI: SET #08AD
       A12: SET #0954
      TEST: SET #0908
      LOAD: SET #0869
      TAKE: SET #08B2
;****************************************************
;*********** MAIN PROGRAME *************************
;====================================================
       DIV: CALL TAKE    ;GET TWO BYTES OF DATA.
            CALL SDIV    ;DO DIVISION.
       PET: JMP A12      ;DISPLAY THE OVERFLOW
                         ;(IF ANY), THE SIGN AND
                         ;THE RESULT IN BOTH
                         ;RESIDUE AND DECIMAL FORM.
;****************************************************
;THIS SUBROUTINE DIVIDES TWO INTEGER NUMBERS.
;----------------------------------------------------
      SDIV: LDA #1361    ;LOAD THE DIVIDEND FROM
                         ;LOC. #1361.
            STA #1340    ;STORE IT AT #1340 .
            STA #134F
            MVI A,00
            STA #1380    ;INITIALISE THE LOC. OF
                         ;SIGN,#1380 TO ZERO.
            STA #1384    ;INITIALISE THE LOC. OF
                         ;QUOTIENT ,#1384 TO ZERO.
            LDA #1363    ;LOAD THE DIVISOR FROM
                         ;LOC #1363.
            STA #1342    ;STORE AT #1342 .
       TAB: CALL DI      ;GET APPROXIMATE QOUTIENT.
                         ;(FIRST APPROXIMATION).
            ANI 04       ;FIND THE SIGN OF THE RESULT.
            STA #1380    ;STORE IT AT LOC. #1380.
```

```
TAC: MOV A,B      $MOVE THE APPROXIMATE
                  $QUOTIENT FROM
                  $B-REG. TO A-REG. .
     STA $1381    $STORE AT $1381.
     ANI $FF      $TEST IF APPROXIMATE
                  $QUOTIENT IS ZERO.
     JZ ZERO      $IF ZERO, JUMP TO ZERO.
     STA $1363    $OTHERWISE STORE IT
                   $AT $1363 .
     LDA $1342    $LOAD  THE DIVISOR FROM
                  $$1342.
     STA $1361
     CALL MU      $MULTIPLY APPROXIMATE QUOT.
                  $AND THE DIVISOR.
     ANI 02      $TEST IF IT OVERFLOWS.
     JNZ SSC     $IF YES, JUMP TO 'SSC'.
     MOV A,B
     STA $1363
     LDA $1340    $LOAD DIVIDEND FROM$1340.
     STA $1361
     CALL CM      $COMPARE THE RESULT OF
                  $MULTIPLICATION WITH THE
                  $DIVIDEND.
     MOV A,B
     ANI 01       $TEST IF (PRODUCT) > (DIVIDEND).
     JNZ SSC      $IF YES, JUMP TO 'SSC'.
     LDA $1380    $GET THE SIGN OF THE
                  $RESULT FROM $1380.
     ANI $04      $TEST WHETHER POSITIVE.
     JNZ PASS     $IF NOT,JUMP TO 'PASS'.
     CALL SU      $SUBTRACT THE PRODUCT TERM
                  $FROM THE DIVIDEND.
TAD: MOV A,B      $MOVE THE RESULT OF SUB-
                  $TRACTION TO A-REG. .
     STA $1340    $STORE IT AS A NEW DIVIDEND
                  $AT $1340.
     LDA $1384    $LOAD THE PARTIAL QUOT.
                  $FROM $1384.
     STA $1363
     LDA $1381    $LOAD THE APPROXIMATE QUOT.
                  $FROM $1381.
     STA $1361
     CALL AD      $ADD TO FORM A NEW PARTIAL
                  $QUOTIENT.
     STA $1316    $STORE SIGN &OVERFLOW.
     MOV A,B
     STA $1384    $STORE IT AT $1384.
     JMP SSC      $GET THE QUOTIENT DIVIDED
                  $BY TWO (IT IS THE NUMBER
                  $GENERATED AFTER DIVIDING
                  $THE NUMBER STORED AT $1381
                  $BY TWO).
```

```
ZERO: LDA #1380     ;LOAD THE SIGN OF THE
                    ;RESULT FROM #1380.
      RRC
      RRC           ;ALTOGETHER TWO RIGHT SHIFTS.
      MOV B,A
      LDA #1316
      ANI 02
      ADD B
      STA #1371
      CPI 03        ;TEST IF A-REG.
                    ;CONTAINS 03
      JZ KATH       ;IF YES, JUMP TO 'KATH'.
      CPI 02        ;TEST IF 02.
      JZ KATR       ;IF YES , JUMP TO 'KATR'.
      ANI 01        ;TEST IF THE RESULT
                    ;IS POSITIVE.
      JZ KAT        ;IF YES, JUMP TO 'KAT'.
      LDA #1384     ;OTHERWISE LOAD THE
                    ;QUOT. FROM #1384.
      STA #1363
      MVI A,00
      STA #1361
      CALL SU       ;SUBTRACT THE QUOT.
                    ;FROM ZERO TO FORM
                    ;NEGATIVE FORM OF QUOT. .
      MOV A,B
      STA #1370
      RET
KATH: MVI A,01       ;THIS IS THE SIGN AND
                     ;OVERFLOW INFORMATION.
      STA #1371
      JMP KAT
KATR: MVI A,03       ;THIS IS THE SIGN AND
                     ;OVERFLOW INFORMATION.
      STA #1371
 KAT: LDA #1384     ;LOAD THE QUOT.
                    ;FROM #1384
      STA #1370
      RET
;*****************************************************
;
;*****************************************************
;THIS PROGRAME ADDS THE PRODUCT TERM WITH
;THE DIVIDEND,IF THE SIGN OF THE RESULT
;IS NEGATIVE, TO FORM A NEW PARTIAL REMAINDER.
;-----------------------------------------------------
 PASS: CALL AD
       JMP TAD
;*****************************************************
```

```
;*************************************************
;THIS PROGRAME GETS A NUMBER DIVIDED BY TWO.
;-------------------------------------------------
        SSC: LDA #1381    ;LOAD THE NUMBER
                          ;TO BE DIVIDED FROM
                          ;LOC. #1381.
             STA #1361
             MVI A,#09     ;'#09' IS THE OPERATION
                           ;CODE FOR DIVIDE BY 2.
             STA #1365
             CALL LOAD
             JMP TAC
;*************************************************
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                          END
```

AP-6(b): Listings of Floating Point Operations

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*****************************************************
;FLOATING POINT ADDITION WITH ELEVEN AS THE
;THE BASE OF EXPONENT ('#BB' IS THE REPRESENTATION
;OF ELEVEN IN RESIDUE SYSTEM OF MODULI 16 &15
;*****************************************************
;PROGRAME STARTS FROM LOC. #0AA0 OF THE SDK-80
;MICROCOMPUTER.
;*****************************************************
                        ORG #0AA0
;*****************************************************
;                       FROM SETS
;-----------------------------------------------------
        MNT: SET #0827
        EPT: SET #082C
        BLN: SET #0840
        GET: SET #084A
         MU: SET #0890
         AD: SET #0899
         SU: SET #089E
         CM: SET #08A3
       OVER: SET #0C00
       SSSP: SET #0FB3
       SDIV: SET #09B9
;*****************************************************
;
;*****************************************************
;THE SUBROUTINE 'CAD' SIGNALS TO GET
;MANTISSAS AND EXPONENTS.
;-----------------------------------------------------
        CAD: MVI D,02      ;LOAD D-REG. WITH #02.
             LXI H,#1350      ;LOAD H AND L REGISTERS
                              ;WITH #1350.
        DAD: CALL MNT         ;SIGNAL 'M='.
             CALL GET         ;GET MANTISSA.
             MOV M,A          ;STORE THE MANTISSA AT
                              ;THE LOC. FORMED BY THE
                              ;CONTENTS OF H AND L
                              ;REGISTERS.
             INR L            ;INCREASE THE CONTENTS
                              ;OF THE REGISTER L.
             CALL BLN         ;GIVE 'SPACE'.
             CALL EPT         ;SIGNAL 'E='.
             CALL GET         ;GET THE EXPONENT.
             MOV M,A          ;STORE IT AT THE LOC.
                              ;FORMED BY THE CONTeNS
                              ;OF THE REGISTER PAIR
                              ;(H,L).
             INR L            ;INCREASE THE CONTENTS OF
                              ;THE REGISTER L.
             CALL BLN         ;GIVE 'SPACE'.
             DCR D            ;DECREASE D-REG. .
```

```
              JNZ DAD          $IF NON ZERO, GET ANOTHER
                               $SET OF MANTISSA AND
                               $EXPONENT.
              RET              $OTHERWISE RETURN.
$*************************************************
$THIS SUBROUTINE SEPARATES A RESIDUE NUMBER INTO
$TWO TERMS.
$A= A(2)*(#BB)^1 + A(1)*(#BB)^0.
$IN SAME WAY IT CAN BE WRITTEN THAT
$B= B(2)*(#BB)^1 +B(1)*(#BB)^0.
$C=C(2)*(#BB)^1+C(1)*(#BB)^0.
$WHERE A = FIRST MANTISSA. B=SECOND MANTISSA.
$C = THE MANTISSA OF THE RESULT.
$THIS SUBROUTINE CALLS THE SUBROUTINE 'SSSP'
$WHICH GIVES THE QUOTIENT AND REMAINDER
$WHEN A NUMBER AT LOC. #1391 IS DIVIDED BY
$THE NUMBER AT LOC. #1390.THE LOC. OF
$QUOTIENT IS #1392 AND OF REMAINDER IS #1393.
$-------------------------------------------------
      SSEP: MVI A,#BB     $'#BB' IS THE RESIDUE
                          $REPRESENTATION OF ELEVEN.
            STA #1390
            CALL SSSP     $DIVIDE THE CONTENS AT
                          $LOC. #1391 BY ELEVEN.
                          $AND GET QUOT. AND
                          $REMAINDER.
            LDA #1346     $LOAD THE SIGN OF THE
                           $THE COEFFICIENT OF
                           $0-POWER OF (#BB) (ELEVEN).
            ANI 01        $TEST IF NEGATIVE.
            JZ CAT        $IF NOT JUMP TO 'CAT'.
            STA #1345     $OTHERWISE STORE  THE SIGN OF
                          $THE MANTISSA AT #1345.
       CAT: LDA #1393     $LOAD THE COEFFICIENT OF
                          $0-POWER OF (#BB) (ELEVEN).
            MOV B,A
            LDA #1392     $LOAD THE COEFFICIENT OF
                          $1-POWER OF (#BB) (ELEVEN).
            RET
$*************************************************
$
$*************************************************
$THE SUBROUTINE 'SELE' ALIGNS THE MANTISSA AND
$CONVERTS THE ALIGNED MANTISSA A ROUNDED FIGURE.
$-------------------------------------------------
      SELE: ANI 01        $FIND THE SIGN OF THE
                          $EXP.-DIFF. FROM THE SIGN
                          $INFORMATION IN A-REG. .
            CNZ TAG       $IF NEGATIVE, JUMP 'TAG'.
       GAT: LDA #1351     $LOAD THE EXPONENT OF
                          $SMALLER OPERAND.
```

```
          STA #1375    ;STORE IT AT #1375.
   SAT:   LDA #1370    ;LOAD THE EXPONENT
                       ;DIFFERENCE FROM #1370.
          STA #135F    ;SAVE AT #135F.
          STA #1363
          MVI A,#11
          STA #1361
          CALL CM      ;COMPARE THE EXP.-DIFF.
                       ;WITH ONE (#11).
          MOV A,B
          ANI 01       ;TEST IF THE EXP.-DIFF.
                       ;IS GREATER.
          JNZ FINIS    ;IF YES, JUMP 'FINIS'.
          MOV A,B
          ANI 04       ;OTHERWISE, TEST IF THE
                       ;EXP.-DIFF. IS EQUAL.
          JNZ SEND     ;IF YES, JUMP 'PEND'.
          RET
   SEND:  CALL PEND
          MVI A,00
          STA #135C
          RET
;*************************************************
;(NOTE: EXP.-DIFF. STANDS FOR EXPONENT DIFFERENCE).
;*************************************************
;
;
;-------------------------------------------------
   FINIS: MVI A,#22
          STA #1361
          CALL CM      ;COMPARE THE EXP.-DIFF.
                       ;WITH #22 (TWO).
          MOV A,B
          ANI 04       ;TEST IF EQUAL.
          JNZ FIXE     ;IF YES, JUMP 'FIXE'.
          MVI A,00    ;OTHERWISE STORE
          STA #135C    ;ZERO AT LOC.#135C & #135D.
          STA #135D
          RET
   FIXE:  LDA #135C    ;LOAD THE COEFFICIENT
                      ;OF 1-POWER OF^H#BB)
                       ;(SMALLER OPERAND)
          STA #135D    ;STORE AT #135D
          MVI A,00
          STA #135C    ;STORE ZERO AT #135C.
;-------------------------------------------------
;THIS SUBROUTINE, 'PEND' IS ALSO CALLED FOR
;ROUNDING A NUMBER DURING NORMALISATION AFTER
;FLOATING POINT MULTIPLICATION AND ADDITION.
;-------------------------------------------------
   PEND:  LDA #135B    ;LOAD SIGN OF THE
                       ;MANTISSA WHICH IS TO BE
                       ;ROUNDED.
```

```
        ANI 01          $TEST THE SIGN.
        JNZ NEGA        $IF NEGATIVE, JUMP 'NEGA'.
        MVI A,$55       $'$55' (FIVE) IS THE
                        $TRUNCATED NUMBER OF
                        $HALF OF ELEVEN.
FEND:   STA $1361
        LDA $135D       $LOAD THE NUMBER WHICH IS TO
                         $BE ROUNDED FROM $135D.
        STA $1363
        CALL AD          $ADD TO THE
                         $CONTENTS OF LOC.$1363.
        MOV A,B
        STA $1391
        CALL SSEP        $DIVIDE THE RESULT OF
                         $ADDITION BY ELEVEN.
                         $AND GET QUOT. &
                         $REMAINDER.
        STA $1361        $STORE THE QUOT.
                         $AT $1361.
        LDA $135C        $LOAD COEFFICIENT OF
                          $1-POWER OF $BB (ELEVEN).
        STA $1363
        CALL AD          $ADD THE QUOT. (GENERATED
                         $IN 'SSEP').
        MOV A,B
        STA $135D        $STORE AS THE COEFF.
                         $OF ZERO POWER OF $BB
                         $(ELEVEN).
        RET
NEGA:   MVI A,$BA        $'$BA' IS THE NEGATIVE
                         $FORM OF FIVE IN
                         $RESIDUE SYSTEM (MOD
                         $16,15)
        JMP FEND
 TAG:   LDA $1351        $LOAD THE EXPONENT OF
                         $FIRST OPERAND.
        MOV B,A
        LDA $1353        $LOAD THE EXPONENT OF
                         $SECOND OPERAND.
        STA $1351        $STORE IT AT $1351.
        MOV A,B
        STA $1353        $STORE IT AT $1353.
                         $(EXCHANGE OF THE
                         $CONTENS BETWEEN $1351
                         $AND $1353).
        LDA $1358        $LOAD THE SIGN OF THE
                         $FIRST OPERAND.
        MOV B,A
        LDA $135B        $LOAD THE SIGN OF THE
                         $SECOND OPERAND FROM $135B.
```

```
                STA #1358       ;STORE IT AT #1358.
                MOV A,B
                STA #135B       ;STORE IT AT #135B.
                                ;(EXCHANGE OF THE CONTENS
                                ;BETWEEN #1358 AND #135B).
                LDA #1359       ;LOAD A(2) OF FIRST
                                ;MANTISSA FROM #1359.
                MOV B,A
                LDA #135C       ;LOAD B(2) OF SECOND
                                ;MANTISSA FROM #135C.
                STA #1359       ;STORE IT AT #1359.
                MOV A,B
                STA #135C       ;STORE IT AT #135C.
                                ;(EXCHANGE OF THE CONTENTS
                                ;BETWEEN #1359 AND #135C).
                LDA #135A       ;LOAD A(1) OF FIRST
                                ;MANTISSA FROM #135A
                MOV B,A
                LDA #135D       ;LOAD B(1) OF SECOND
                                ;MANTISSA FROM #135D.
                STA #135A       ;STORE IT AT #135A.
                MOV A,B
                STA #135D       ;STORE IT AT #135D.
                                ;(EXCHANGE OF CONTENTS
                                ;BETWEEN #135A AND #135D).
                RET
;*****************************************************
;
;*****************************************************
;THIS SUBROUTINE NORMALISES THE RESULT AFTER
;ADDITION / SUBTRACTION.
;-----------------------------------------------------
        NORM: LDA #13A0         ;LOAD C(2) OF THE
                                ;RESULT FROM #13A0.
                STA #1361
                MVI A,#BB
                STA #1363
                CALL MU         ;MULTIPLY C(2) OF
                                ;THE RESULT BY ELEVEN.
                ANI 02          ;TEST IF IT OVERFLOWS.
                JNZ OVER        ;IF YES, JUMP 'OVER'.
                MOV A,B         ;OTHERWISE, MOVE THE
                                ;PRODUCT FROM B-REG.
                                ;A-REG. .
```

```
                STA #1361
                LDA #1377       ;LOAD C(1) OF THE
                                ;RESULT FROM #1377.
                STA #1363
                CALL AD         ;ADD C(1) AND THE
                                ;PRODUCT.
                ANI 02          ;TEST IF OVERFLOW
                JNZ OVER        ;IF YES , JUMP 'OVER'.
                MOV A,B
                STA #1376
                RET
;*************************************************
;
;*************************************************
;THIS SUBROUTINE FORMS EXPONENT DIFFERENCE
;AND CALLS 'SELE' FOR ALIGNMENT OF THE
;MANTISSAS.
;-------------------------------------------------
        EXPO: LDA #1351         ;LOAD THE EXPONENT OF
                                ;FIRST OPERAND FROM
                                ;#1351.
                STA #1361
                LDA #1353       ;LOAD THE EXPONENT OF
                                ;SECOND OPERAND FROM
                                ;#1353.
                STA #1363
                CALL SU         ;SUBTRACT THE EXPONENT
                                ;OF FIRST OPERAND FROM
                                ;THE EXPONENT OF SECOND
                                ;OPERAND.
                CALL SELE
                RET
;*************************************************
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                                END
```

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*****************************************************
;CONTINUATION OF FLOATING POINT ADDITION AND
;INTRODUCTION OF SUBTRACTION.
;RESIDUE ARITHMETIC. (MODULI: 16 & 15)
;****************************************************
;
;****************************************************
;PROGRAME STARTS FROM LOC. #0C00 OF THE
;SDK-80 MICROCOMPUTER.
;****************************************************
;
;****************************************************
                ORG #0C00
;****************************************************
;               FROM SETS
;----------------------------------------------------
      CAR: SET #080A
      MNT: SET #0827
      EPT: SET #082C
      BLN: SET #0840
     FBCD: SET #0EE5
    TZERO: SET #0E9B
     NORM: SET #0B9B
     EXPO: SET #0BC5
     PEND: SET #0B2F
      CAD: SET #0AA0
     SSEP: SET #0AC0
       AD: SET #0899
       SU: SET #089E
    OSIGN: SET #08C8
     SDIV: SET #09B9
;****************************************************
;THIS SUBROUTINE TAKES CARE OF ROUNDING A
;NUMBER DURING NORMALISATION.
;----------------------------------------------------
     OVER: LDA #1375    ;LOAD THE EXPONENT OF
                        ;HIGHER OPERAND FROM #1375.
           STA #1361
           MVI A,#11
           STA #1363
           CALL AD      ;ADD #11 (ONE) TO THE EXP. .
           ANI 03       ;FIND THE SIGN AND OVERFLOW
                        ;INFORMATION.
           STA #1374    ;STORE IT AT #1374.
           MOV A,B
           STA #1375    ;STORE THE RESULT OF
                        ;ADDITION AT #1375.
           LDA #137F    ;LOAD THE SIGN OF
                        ;THE RESULT FROM #137F.
           STA #135B    ;STORE IT AT #135B.
           LDA #1377    ;LOAD THE COEFFICIENT OF
                        ;ZERO POWER OF #BB
                        ;(ELEVEN) FROM #1377.
           STA #135D    ;STORE IT AT #135D.
```

```
            LDA #1376    $LOAD THE COEFFICIENT OF
                         $FIRST POWER OF #BB
                         $(ELEVEN) FROM #1376.
            STA #135C    $STORE IT AT #135C.
            CALL PEND    $CALL 'PEND' FOR ROUNDING.
            LDA #135D    $LOAD THE ROUNDED NUMBER
                         $FROM #135D.
            STA #1376
            RET
$*************************************************
$
$*************************************************
$THIS SUBROUTINE DISPLAYS THE RESULT OF FL. PT.
$ARITHMETIC OPERATIONS.
$-------------------------------------------------
     OUTPU: CALL CAR     $CARRIAGE RETURN AND
                         $LINE FEED.
            CALL MNT     $SIGNAL 'M='.
            LDA #1376    $LOAD THE MANTISSA OF
                         $THE RESULT FROM #1376.
            STA #1370
            LDA #137F    $LOAD SIGN AND OVERFLOW
                         $INFORMATION\OF THE
                         $MANTISSA OF THE RESULT
                         $FROM #137F.
            STA #1371
            CALL OSIGN   $SIGNAL OVERFLOW (IF ANY)
                         $AND SIGN OF THE RESULT.
                         $AND THE RESULT (MANTISSA)
                         $IN RESIDUE FORM.
            CALL BLN     $GIVE 'SPACE' .
           CALL EPT     $SIGNAL 'E='.
            LDA #1375    $LOAD THE EXP. OF THE
                         $THE RESULT.
            STA #1370
            LDA #1374    $LOAD THE SIGN AND
                         $OVERFLOW INFORMATION
                         $OF THE
                         $EXP. OF THE RESULT.
            STA #1371
            CALL OSIGN
            RET
$*************************************************
$
$*************************************************
$THIS IS THE MAIN FROGRAME OF FL. PT.
$ADDITION.
$-------------------------------------------------
       FAD: CALL CAD     $GET TWO FL. PT. NUMBERS.
            CALL TZERO   $TEST FOR ZERO
                         $MANTISSAS.
```

```
                CALL KATCH    $FIND THE COEFFICIENTS
                              $OF (#BB) &(#BB)
                              $FROM MANTISSAS.
                CALL EXPO     $FIND EXP.-DIFF. AND
                              $ALIGN THE COEFFICIENTS .
                CALL MAD      $ADD THE COEFICIENTS.
                CALL TAD      $FIND THE SIGN OF EXP. .
                CALL NORM     $NORMALISE.
                CALL TAP      $FIND THE SIGN OF THE
                              $RESULT.
                CALL OUTPU    $DISPLAY OVERFLOW
                              $(IF ANY) ,SIGN AND
                              $RESULT IN RESIDUE FORM.
                JMP FBCD      $DISPLAY OVERFLOW
                              $(IF ANY) ,SIGN AND
                               $RESULT IN DECIMAL FORM.
$*************************************************
$
$*************************************************
$THIS SUBROUTINE SEPARATES EACH OF THE
$MANTISSAS INTO TWO TERMS. THIS MEANS
$A= A(2)*(#BB)^1 + A(1)*(#BB)^0.
$B=B(2)*(#BB)^1 +B(1)*(#BB)^0.
$IN OTHERWORDS THIS SUBROUTINE FINDS
$A(2) ,A(1) ,B(2) & B(1) .
$-------------------------------------------------
     KATCH: LDA #1350     $LOAD MANTISSA OF FIRST
                          $OPERAND.
            STA #1391
            CALL SSEP     $GET TWO TERMS.
            STA #1359     $STORE THE COEFFICIENT
                          $( A(2) ) OF
                          $FIRST POWER OF #BB
                          $(ELEVEN) AT #1359.
            MOV A,B
            STA #135A     $STORE THE COEFFICIENT
                          $( A(1) )
                          $OF ZERO POWER OF #BB
                          $(ELEVEN) AT #135A.
            LDA #1345     $LOAD THE SIGN OF THE
                          $FIRST OPERAND (MANTISSA).
            STA #1358     $STORE IT AT #1358.
            LDA #1352     $LOAD THE MANTISSA OF
                          $SECOND OPERAND.
            STA #1391
            CALL SSEP     $GET TWO TERMS.
            STA #135C     $STORE THE COEFFICIENT
                          $( B(2) ) OF FIRST POWER
                          $OF #BB (ELEVEN) AT #135C.
            MOV A,B
            STA #135D     $STORE THE COEFFICIENT
                          $( B(1) ) OF ZERO POWER
                          $OF #BB (ELEVEN).
            LDA #1345     $LOAD THE SIGN OF THE
                          $SECOND OPERAND (MANTISSA).
            STA #135B     $STORE IT AT #135B.
            RET
$*************************************************
```

```
;*************************************************
;THIS SUBROUTINE ADDS THE COEFFICIENTS
;THAT MEANS  ( A(2)+B(2) ) AND
;( A(1)+B(1) ).
;-------------------------------------------------
       MAD: LDA #1359    ;LOAD THE COEFF. A(2).
            STA #1361
            LDA #135C    ;LOAD THE COEFF. B(2).
            STA #1363
            CALL AD      ;ADD A(2) AND B(2).
            STA #137F    ;STORE THE SIGN AND OVERFLOW
                         ;INFORMATION AT #137F.
            MOV A,B
            STA #1376    ;STORE THE RESULT OF
                         ;A(2)+B(2) AT #1376
            STA #13A0    ;AND ALSO AT #13A0.
            LDA #135A    ;LOAD THE COEFF. A(1).
            STA #1361
            LDA #135D    ;LOAD THE COEFF. B(1).
            STA #1363
            CALL AD      ;ADD A(1) AND B(1).
            MOV A,B
            STA #1377    ;STORE THE RESULT OF
                         ;A(1)+B(1) AT #1377.
            STA #13A1    ;AND ALSO AT #13A1.
            RET
;*************************************************
;
;*************************************************
;THIS SUBROUTINE FINDS THE SIGN
;OF  THE EXPONENT OF THE RESULT.
;-------------------------------------------------
       TAD: LDA #1375    ;LOAD THE EXP. OF THE
                         ;RESULT.
            STA #1361
            MVI A,00
            STA #1363
            CALL AD      ;ADD THE EXP. WITH ZERO.
            STA #1374    ;STORE THE SIGN AT #1374.
            RET
;*************************************************
```

```
$*********************************************
$THIS SUBROUTINE SUBTRACTS THE MINUEND
$(MANTISSA) FROM ZERO. THEN CALLS
$FLOATING POINT ADDITION.
$THE WHOLE PROGRAME IS USED FOR
$FLOATING POINT SUBTRACTION.
$---------------------------------------------
        FSU: CALL CAD     $GET TWO OPERANDS.
             LDA #1352    $LOAD MINUEND (MANTISSA)
                          $FROM #1352.
             STA #1363
             MVI A,00
             STA #1361
             CALL SU      $SUBTRACT THE MINUEND FROM
                          $ZERO.
             MOV A,B
             STA #1352     $STORE THE RESULT OF
                          $SUBTRACTION.
             JMP FAD+3
$*********************************************
$
$*********************************************
$THIS SUBROUTINE DETERMINES
$THE SIGN OF MANTISSA OF
$THE RESULT
$---------------------------------------------
        TAP: LDA #1376    $LOAD THE MANTISSA OF
                          $THE RESULT.
             STA #1361
             MVI A,00
             STA #1363
             CALL AD      $ADD THE MANTISSA
                          $WITH ZERO.
             STA #137F    $STORE THE SIGN AT #137F.
             RET
$*********************************************
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                          END
```

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*********************************************************
;FLOATING POINT MULTIPLICATION
;RESIDUE ARITHMETIC (MODULI: 16 &15)
;
;*********************************************************
;PROGRAME STARTS FROM LOC. #0D20 OF THE
;SDK-80 MICROCOMPUTER.
;*********************************************************
;
;*********************************************************
                        ORG #0D20
;*********************************************************
;                       FROM SETS
;---------------------------------------------------------
        FMT: SET #0456
        CAD: SET #0AA0
        MAD: SET #0CA9
        TAP: SET #0D03
         MU: SET #0890
         AD: SET #0899
       SSEP: SET #0AC0
       PEND: SET #0B2F
       FBCD: SET #0EE5
      OUTPU: SET #0C33
      KATCH: SET #0C7C
;*********************************************************
;
;*********************************************************
;THE MAIN PROGRAME FOR FL. PT. MULTIPLICATION.
;---------------------------------------------------------
        FMU: CALL CAD     ;GET TWO OPERANDS
             CALL FMT     ;MULTIPLY AND NORMALISE
             CALL EXP     ;FIND THE EXP. OF THE
                          ;RESULT.
             CALL OUTPU   ;DISPLAY THE OVERFLOW
                          ;(IF ANY) , THE SIGN
                          ;AND THE RESULT IN
                          ;RESIDUE FORM.
             JMP FBCD     ;DISPLAY THE RESULT IN
                          ;DECIMAL FORM.
;*********************************************************
```

```
;*************************************************
;THIS SUBROUTINE MULTIPLIES TWO MANTISSAS
;A AND B.  WHERE A=A(2)*(#BB)^1 + A(1)*(#BB)^0.
;AND B=B(2)*(#BB)^1 + B(1)*(#BB)^0.
;A*B =C = A(2)*B(2)*(#BB)^2 + A(2)*B(1)*(#BB)^1
;  +A(1)*B(2)*(#BB)^1 + A(1)*B(1)*(#BB)^0.
;OR C = C(3)*(#BB)^2 + C(2)*(#BB)^1 +C(1)*(#BB)^0.
;WHERE C(3) = A(2)*B(2) (THE COEFF. OF
;  2-POWER OF #BB), C(2)= A(2)*B(1) + A(1)*B(2)
;(THE COEFF. OF 1-POWER OF #BB)
;C(1) = A(1)*B(1) (THE COEFF. OF 0-POWER OF
;#BB).
;-------------------------------------------------

PEG: LDA #1359     ;LOAD THE COEFF. A(2)
                   ;FROM #1359.
     STA #1330     ;SAVE IT AT #1330.
     STA #1361
     LDA #135C     ;LOAD THE COEFF. B(2)
                   ;FROM #135C.
     STA #1332     ;SAVE IT AT #1332.
     STA #1363
     CALL MU       ;MULTIPLY A(2) AND B(2)
                   ;TO FORM THE COEFF. OF
                   ;2-POWER OF #BB (ELEVEN).
     MOV A,B       ;MOVE THE RESULT OF
                   ;MULTIPLICATION TO A-REG.
     STA #1320     ;STORE IT AT #1320.
     LDA #135D     ;LOAD THE COEFF. B(1)
                   ;FROM #135D.
     STA #1333     ;SAVE IT AT #1333.
     STA #1363
     CALL MU       ;MULTIPLY A(2) AND B(1)
                   ;TO FORM THE COEFF. OF
                   ;1-POWER OF #BB (ELEVEN).
     MOV A,B
     STA #1321      ;STORE THE RESULT OF
                   ;MULTIPLICATION AT #1320.
     STA #1350     ;STORE IT AT #1350 ALSO.
     LDA #135A     ;LOAD THE COEFF. A(1)
                   ;FROM #135A.
     STA #1331     ;SAVE IT AT #1331.
     STA #1361
     CALL MU       ;MULTIPLY A(1) AND B(1)
                   ;TO FORM THE COEFF. OF
                   ;0-POWER OF #BB (ELEVEN).
     MOV A,B
     STA #1323     ;STORE IT AT #1323.
     LDA #135C     ;LOAD THE COEFF. B(2)
                   ;FROM #135C.
     STA #1363
     CALL MU       ;MULTIPLY A(1) AND B(2)
                   ;TO FORM THE COEFF. OF
                   ;1-POWER OF #BB (ELEVEN).
```

```
MOV A,B
STA #1322     ;STORE IT AT #1322.
STA #1352
CALL KATCH    ;SEPARATE EACH OF THE
              ;COEFFICIENTS OF
              ;1-POWER OF #BB (ELEVEN)
              ;INTO TWO TERMS.
CALL MAD      ;ADD THE COEFFICIENT,
              ;GENERATED IN 'KATCH'.
              ;THIS ADDITION GENERATES
              ;TWO COEFF. (1) 1-POWER
              ;OF #BB, (2) 0-POWER OF
              ;#BB (ELEVEN).
              ;(THE COEFFICIENTS OF
              ;1-POWER AND 0-POWER OF
              ;#BB IN 'MAD' IS REALY
              ;THE COEFFICIENTS OF
              ;2-POWER AND 1-POWER OF
              ;#BB RESPECTIVELY, IN
              ;THE PARTIAL PRODUCTS -'P.P').
LDA #1376     ;LOAD THE COEFF. OF
              ;1-POWER OF #BB
              ;GENERATED IN 'MAD'.
STA #1361
LDA #1320     ;LOAD THE COEFF. OF
              ;2-POWER (IN PARTIAL
              ;PRODUCT).
STA #1363
CALL AD       ;ADD THE COEFF. OF
              ;2-POWER (IN PARTIAL
              ;PRODUCT) AND 1-POWER
              ;(GENERATED IN 'MAD').
MOV A,B
STA #137A     ;STORE THE RESULT
              ;OF ADDITION AT #137A.
LDA #1323     ;LOAD THE COEFF. OF
              ;0-POWER (IN P.P)
              ;ROM #1323.
STA #1391
CALL SSEP     ;FIND TWO COEFFICIENTS
              ;OF THE NUMBER AT #1391.
STA #1361     ;STORE THE COEFF. OF
              ;1-POWER (GENERATED IN
              ;'SSEP') AT #1361.
LDA #1345     ;LOAD THE SIGN OF
              ;THE COEFF. OF
              ;0-POWER OF #BB(IN P.P).
              ;FROM #1345.
STA #137D     ;STORE IT AT #137D.
MOV A,B       ;MOVE THE COEFF. OF
              ;0-POWER OF #BB
              ;(GENERATED IN 'SSEP').
              ;FROM B-REG. TO A-REG.
STA #137E     ;STORE IT AT #137E.
LDA #1377     ;LOAD THE COEFF.
              ;OF 0-POWER OF #BB
              ;(GENERATED IN 'MAD')
```

```
                STA #1363
                CALL AD         ;ADD THE COEFF. OF
                                ;0-POWER OF #BB
                                ;(GENERATED IN 'MAD')
                                ;AND THE COEFF. OF
                                ;1-POWER OF #BB
                                ;(GENERATED IN 'SSEP')
                STA #137B       ;STORE THE SIGN OF THE
                                ;ADDITION AT #137B.
                MOV A,B
                STA #137C       ;STORE THE FINAL COEFF.
                                ;OF 1-POWER OF #BB
                                ;AT #137C. (THIS IS
                                ;THE FINAL COEFF. OF
                                ;1-POWER OF #BB IN P.P).
                RET
;*************************************************************
;THIS SUBROUTINE NORMALISES
;-------------------------------------------------------------
        NORL: LDA #137A         ;LOAD THE COEFF. OF
                                ;2-POWER OF #BB (IN P.P)
                                ;FROM #137A.
                STA #1361
                MVI A,00
                STA #1334       ;STORE ZERO AT #1334
                                ;(LOC. OF GENERATED EXP.)
                MVI A,#BB
                STA #1363
                CALL MU         ;MULTIPLY THE COEFF. OF
                                ;2-POWER OF #BB (IN P.P)
                                ;WITH #BB (ELEVEN).
                STA #133F       ;STORE THE SIGN AND OVERFLOW
                                ;INFORMATION AT #133F.
                ANI 02          ;TEST IF OVERFLOW.
                JNZ POVER       ;IF YES , JUMP 'POVER'.
                MOV A,B
                STA #1361
                LDA #137C       ;LOAD THE COEFF. OF
                                ;1-POWER OF #BB (IN P.P)
                STA #1363
                CALL AD         ;ADD THE PREVIOUS
                                ;PRODUCT TERM TO THE COEFF.
                                ;OF 1-POWER OF #BB (INP.P).
                STA #133F       ;STORE THE SIGN AND OVERFLOW
                                ;INFORMATION OF ADDITION
                                ;AT #133F.
                ANI 02          ;TEST IF OVERFLOW.
                JNZ POVER       ;IF YES, JUMP 'POVER'.
                MOV A,B
                STA #137A       ;STORE THE RESULT OF
                                ;ADDITION AT #137A.
```

```
              STA #1361
              MVI A,#BB
              STA #1363
              CALL MU       ;MULTIPLY THE RESULT OF
                            ;ADDITION BY #BB (ELEVEN).
              STA #133F     ;STORE THE SIGN AND OVERFLOW
                            ;INFORMATION OFMULTIPLICATION
                            ;AT #133F.
              ANI 02        ;TEST IF OVERFLOW.
              JNZ COVER     ;IF YES, JUMP 'COVER'.
              MOV A,B
             STA #1361     ;STORE THE RESULT OF
                            ;MULTIPLICATION AT #1361.
              LDA #137E     ;LOAD THE COEFF. OF
                            ;0-POWER OF #BB (IN PP).
              STA #1363
              CALL AD       ;ADD THE PREVIOUS RESULT
                            ;OF MULTIPLICATION TO
                            ;ZERO POWER OF #BB (IN P.P).
              STA #133F     ;STORE THE SIGN AND OVERFLOW
                            ;INFORMATION OF ADDITION
                            ;AT #133F.
              ANI 02        ;TEST IF OVERFLOW.
              JNZ COVER     ;IF YES, JUMP 'COVER'.
              MOV A,B
              STA #1376     ;STORE THE NORMALISED
                            ;NUMBER AT #1376.
              LDA #133F     ;LOAD THE SIGN AND
                            ;OVERFLOW INFORMATION
                            ;FROM #133F.
              ANI 01        ;FIND THE SIGN OF
                            ;THE FINAL RESULT.
              STA #137F     ;STORE IT AT #137F.
              RET
;*************************************************
;
;*************************************************
;THIS ROUTINE TAKES CARE OF ROUNDING A
;NUMBER. (AFTER FIRST OR SECOND OVERFLOW
;DETECTION IN THE SUBROUTINE 'NORL').
;-------------------------------------------------
      POVER: MVI A,#22      ;'#22' (TWO) IS THE
                            ;GENERATED EXPONENT.
              STA #1334     ;STORE IT AT #1334.
              LDA #137B     ;LOAD THE SIGN OF THE
                            ;COEFF.OF 1-POWER OF
                            ;#BB (IN P.P).FROM #137B.
              STA #135B     ;STORE IT AT #135B.
              LDA #137C     ;LOAD THE COEFF. OF
                            ;1-POWER OF #BB (IN P.P)
                            ;FROM #137C.
```

```
              STA #135D     ;STORE IT AT #135D.
              LDA #137A     ;LOAD THE COEFF. OF
                            ;2-POWER OF #BB (IN P.P)
                            ;FROM #137A.
              STA #135C     ;STOR IT AT #135C.
              CALL PEND     ;GET ROUNDED NUMBER
                            ;OF THE COEFFICIENT
                            ;OF 2-POWER OF #BB.
       CAP:   LDA #135D     ;LOAD THE ROUNDED
                            ;NUMBER FROM #135D.
              STA #1376     ;STORE IT AT #1376.
              CALL TAP      ;FIND THE SIGN OF
                            ;THE MANTISSA OF THE
                           ;RESULT.
              STA #137F     ;STORE IT AT #137F.
              RET
;*************************************************
;
;*************************************************
;THIS PROGRAME TAKES CARE OF ROUNDING
;A NUMBER AFTER THIRD OR FOURTH OVERFLOW
;DETECTION IN THE SUBROUTINE 'NORL'.
;-------------------------------------------------
      COVER:  MVI A,#11     ;'#11' (ONE) IS THE
                            ;GENERATED EXPONENT.
              STA #1334     ;STORE IT AT #1334.
              LDA #137D     ;LOAD THE SIGN OF THE
                            ;COEFF. OF 0-POER OF
                            ;#BB (IN P.P) FROM
                            ;#137D.
              STA #135B     ;STORE IT AT #135B.
              LDA #137E     ;LOAD THE COEFF. OF
                            ;0-POWER OF #BB (INP.P)
                            ;FROM #137E.
              STA #135D     ;STORE IT AT #135D.
              LDA #137A     ;LOAD THE COEFF. OF
                            ;1-POWER OF #BB (IN P.P)
                            ;FROM #137A.
              STA #135C     ;STORE IT AT #135C.
              CALL PEND     ;GET ROUNDED NUMBER
                            ;OF THE COEFF. OF
                            ;1-POWER OF #BB.
              JMP CAP
;*************************************************
;THIS SUBROUTIE FINDS THE EXPONENT OF THE
;MANTISSA OF THE RESULT.
;-------------------------------------------------
        EXP:  LDA #1351     ;LOAD THE EXP. OF
                            ;THE FIRST OPERAND
                            ;FROM #1351.
              STA #1361
```

```
          LDA #1353     $LOAD THE EXP. OF
                        $THE SECOND OPERAND.
                        $FROM #1353.
          STA #1363
          CALL AD       $ADD TWO EXPONENTS.
          ANI 02        $TEST IF OVERFLOW.
          MOV C,A       $MOVE THE OVERFLOW
                        $INFORMATION FROM A-REG.
                        $TO C-REG.
          MOV A,B       $MOVE THE RESULT OF ADDITION
                        $FROM B-REG. TO A-REG.
          STA #1363
          LDA #1334     $LOAD THE GENERATED EXP.
                        $FROM #1334.
          STA #1361
          CALL AD       $ADD THE GENERATED EXP.
          MOV H,A       $MOVE THE SIGN AND
                        $OVERFLOW INFORMATION
                        $FROM A-REG. TO H-REG.
          ANI 02        $TEST IF OVERFLOW.
          MOV E,A       $MOVE THE OVERFLOW
                        $INFORMATION FROM A-REG.
                        $TO E-REG.
          MOV A,B       $MOVE THE RESULT OF
                        $ADDITION FROM B-REG.
                        $TO A-REG.
          STA #1375     $STORE IT (FINAL EXP.)
                        $AT #1375.
          MOV B,C       $MOVE THE OVERFLOW
                        $INFORMATION (AFTER
                        $FIRST ADDITION) FROM C-REG.
                        $TO B-REG.
          MOV A,E       $MOVE THE OVERFLOW
                        $INFORMATION (AFTER
                        $SECOND ADDITION)
                        $FROM E-REG. TO A-REG.
          XRA B         $EXCLUSIVE OR WITH
                        $A-REG  AND B-REG.
          MOV B,A       $MOVE FINAL OVERFLOW
                       $INFORMATION OF EXP.
                        $FROM A-REG. TO B-REG.
          MOV A,H      $MOVE THE SIGN AND OVERFLOW
                        $INFORMATION (AFTER
                        $SECOND ADDITION) FROM H-REG.
                        $TO A-REG.
          ANI 01        $FIND THE SIGN OF EXP.
          ADD B         $FORM SIGN AND OVERFLOW
                        $INFORMATION OF EXP.
          STA #1374
          RET
$*************************************************
```

```
;*************************************************
;THIS SUBROUTINE TESTS WHETHER ANY ONE OF THE
;MANTISSAS IS ZERO.IF ZERO THE EXPONENT OF THE
;CORRESPONDING MANTISSA IS SET TO ZERO.
;THIS SUBROUTINE IS CALLED DURING FL. PT.
;ADDITION/SUBTRACTION FOR TESTING ZERO-
;MANTISSAS.
;-------------------------------------------------
      TZERO: LDA #1350    ;LOAD THE MANTISSA OF
                          ;FIRST OPERAND FROM #1350.
             CPI 00       ;TEST IF ZERO IS OF THE
                          ;FORM '00'.
             CZ RZERO     ;IF ZERO, CALL 'RZERO.
             CPI #0F      ;TEST IF ZERO IS OF THE
                          ;FORM '0F'.
             CZ RZERO     ;IF ZERO, CALL 'RZERO'.
             LDA #1352    ;LOAD THE MANTISSA OF
                          ;SECOND OPERAND FROM #1352.
             CPI 00       ;TEST IF ZERO IS OF THE
                          ;FORM '00'.
             JZ QZERO     ;IF ZERO, JUMP 'QZERO'.
             CPI #0F      ;TEST IF ZERO IS OF THE
                          ;FORM '0F'.
             JZ QZERO     ;IF ZERO, JUMP TO'QZERO'.
             RET
      RZERO: MVI A,00
             STA #1351    ;STORE ZERO AT LOC. OF THE
                          ;EXP. OF THE FIRST
                          ;OPERAND (#1351).
              RET
      QZERO: MVI A,00
             STA #1353    ;STORE ZERO AT LOC. OF THE
                          ;EX. OF THE SECOND
                         ;OPERAND (#1353).
             RET
;*************************************************
;     NOTE:- 'P.P' STANDS  FOR 'PARTIAL PRODUCT'.
;*************************************************
                   END
```

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*************************************************************
;FLOATING POINT DIVISION.THE EXPRESSION USED IS
;RESIDUE ARITHMETIC. (MODULI: 16 & 15)
;*************************************************************
;
;PROGRAME STARTS FROM LOC. #0400 OF THE SDK-80
;MICROCOMPUTER.
;*************************************************************
;
;*************************************************************
                ORG #0400
;*************************************************************
;               FROM SETS
;-------------------------------------------------------------
          SU: SET #089E
          AD: SET #0899
         FEG: SET #0D2F
         CAD: SET #0AA0
        FBCD: SET #0EE5
        TEST: SET #0908
        NORL: SET #0DBB
       KATCH: SET #0C7C
       OUTPU: SET #0C33
;*************************************************************
;THE MAIN PROGRAME FOR FL. PT. DIVISION.
;-------------------------------------------------------------
        FDI: CALL CAD    ;GET TWO OPERANDS.
             JMP ZRET     ;TEST IF THE MANTISSAS
                          ;ARE ZERO.
        FFC: CALL OVFL   ;GET THE VALUE OF '1/B'.
        SDK: CALL FMT    ;PERFORM FL. PT.
                         ;MULTIPLICATION.
             CALL EXT    ;FIND THE EXPONENT.
             CALL OUTPU  ;DISPLAY THE RESULT TO
                         ;THE OUTPUT DEVICE.
                         ;(IN RESIDUE FORM ).
             JMP FBCD    ;DISPLAY THE RESULT
                         ;IN DECIMAL FORM.
;*************************************************************
;
;*************************************************************
;THIS SUBROUTINE FINDS THE EXPONENT OF THE
;RESULT.
;-------------------------------------------------------------
        EXT: LDA #1351   ;LOAD THE EXP. OF DIVIDEND.
             STA #1361
             LDA #1353   ;LOAD THE EXP.  OF DIVISOR.
             STA #1363
             CALL SU     ;SUBTRACT THE EXP. OF
                         ;DIVISOR FROM THAT OF
                         ;DIVIDEND.
```

```
        ANI 02          $TEST IF OVERFLOW.
        MOV C,A         $MOVE THE OVERFLOW
                        $INFORMATION FROM A-REG.
                        $TO C-REG. .
        MOV D,B         $MOVE THE RESULT OF
                       $SUBTRACTION FROM B-REG.
                        $TO D-REG. .
        LDA #1334       $LOAD THE EXP. GENERATED
                        $DURING FL. PT. MULTIPLI
                        $-CATION FROM #1334.
       STA #1361
        LDA #1398       $LOAD THE EXP. OF '1/B'.
                        $FROM #1398.
        STA #1363
        CALL AD         $ADD THE EXP. GENERATED
                        $DURING FL. PT.
                        $MULTIPLICATION TO THE
                        $EXP. OF '1/B'.
        MOV A,B         $MOVE THE RESULT OF
                        $ADDITION FROM B-REG.
                        $TO A-REG. .
        STA #1363
        MOV A,D         $MOVE THE RESULT OF
                        $SUBTRACTION FROM D-REG.
                        $TO A-REG. .
        STA #1361
        CALL AD         $ADD
        MOV H,A         $SAVE THE SIGNAND OVERFLOW
                        $INFORMATION IN H-REG.
        ANI 02          $TEST IF OVERFLOW.
        MOV E,A         $MOVE THE OVERFLOW
                        $INFORMATION FROM A-REG.
                        $TO E-REG. .
KKK:    MOV A,B         $MOVE THE EXP. OF THE
                        $RESULT FROM B-REG.
                        $TO A-REG. .
        STA #1375       $STORE IT AT #1375.
        MOV B,C         $MOVE THE OVERFLOW
                        $INFORMATION (AFTER
                        $SUBTRACTION ) FROM
                        $C-REG. TO B-REG. .
        MOV A,E         $MOVE THE OVERFLOW
                        $INFORMATION (AFTER
                        $ADDITION ) FROM
                        $E-REG. TO A-REG. .
        XRA B           $EXCLUSIVE OR A-REG.
                        $WITH B-REG. .
        MOV B,A         $MOVE IT TO B-REG. .
        MOV A,H         $MOVE THE SIGN AND OVERFLOW
                        $INFORMATION (AFTER ADDITION)
                        $FROM H-REG TO A-REG. .
```

```
                ANI 01          ;FIND THE SIGN OF THE RESULT.
                ADD B           ;FORM THE SIGN AND OVERFLOW
                                ;INFORMATION.
                STA #1374       ;STORE IT AT #1374
                RET
;*****************************************************
;
;*****************************************************
;THIS SUBROUTINE PERFORM FL. PT. MULTIPLICATION.
;-----------------------------------------------------
        FMT: CALL KATCH     ;FIND THE COEFFICIENTS
                           ;OF (#BB) & (#BB)
                            ;FROM THE MANTISSAS.
             CALL PEG       ;MULTIPLY (FLOATING POINT).
             CALL NORL      ;NORMALISE.
             RET
;*****************************************************
;
;*****************************************************
;THIS SUBROUTINE GETS THE VALUE OF 1/B
;FROM THE TABLE STORED IN PROM.
;-----------------------------------------------------
       OVFL: MVI A,#60      ;'#60' IS THE ADDRESS OF
                            ;THE LATCH ASSIGNED FOR
                            ;DIVISOR.
             OUT #16
             LDA #1352      ;LOAD THE MANTISSA OF
                            ;DIVISOR FROM #1352.
             OUT #14
             MVI A,#30      ;'#30' IS THE ADDRESS
                            ;OF THE LATCH WHICH
                            ;LATCHES THE CODE FOR GETT-
                            ;ING EXP. OF '1/B'.
             OUT #16
             MVI A,#02      ;'02' IS THE CODE FOR
                            ;GETTING THE EXP. OF '1/B'.
             OUT #14
             IN #15         ;GET THE EXP. OF '1/B'.
             STA #1398      ;STORE IT AT #1398.
             MVI A,#07      ;'#07' IS THE CODE FOR
                            ;GETTING THE MANTISSA
                            ;PART OF '1/B'.
              OUT #14
              IN #15         ;GET THE MANTISSA PART
                             ;OF '1/B'.
              STA #1352
              RET
;*****************************************************
```

```
;*************************************************************
;THIS PROGRAME TESTS IF ANY ONE OF THE MANTISSAS
;IS ZERO. IF ZERO ,NO OPERATION.
;-------------------------------------------------------------
      ZRET: LDA #1350     ;LOAD THE MANTISSA OF
                          ;DIVIDEND FROM #1350.
            CPI 00        ;TEST IF ZEROIS OF THE
                          ;FORM '00'.
            JZ RRET       ;IF YES, JUMP TO 'RRET'.
            CPI #0F       ;TEST IF ZERO IS OF THE
                          ;FORM '0F'.
            JZ RRET       ;IF YES, JUMP TO 'RRET'.
            LDA #1352     ;LOAD THE MANTISSA OF
                          ;DIVISOR FROM #1352.
            CPI 00        ;TEST IF ZERO IS OF THE
                          ;FORM '00'.
            JZ RRET       ;IF YES, JUMP TO 'RRET'.
            CPI #0F       ;TEST IF ZERO IS OF THE
                          ;FORM '0F'
            JZ RRET       ;IF YES, JUMP TO 'RRET'.
            JMP FFC
      RRET: JMP TEST      ;SIGNAL TO GET OPERANDS.
;*************************************************************
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                    END
```

AP-6(c): Listings of Residue to Decimal Conversion

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
;*************************************************************
;THIS PROGRAME CONVERTS A RESIDUE NUMBER
;TO A DECIMAL NUMBER.
;(RESIDUE ARITHMETIC. MODULI: 16 & 15 )
;*************************************************************
;
;*************************************************************
;PROGRAME STARTS FROM LOC. #0ED0 OF THE SDK-80
;MICROCOMPUTER.
;
;*************************************************************
                ORG #0ED0
;*************************************************************
;               FROM SETS
;-------------------------------------------------------------
        SDIV: SET #09B9
          SU: SET #089E
          MU: SET #0890
          AD: SET #0899
         CAR: SET #080A
        XTAR: SET #0845
         POS: SET #0822
         NEG: SET #081D
       NMOUT: SET #02C3
        TEST: SET #0908
         MNT: SET #0827
         BLN: SET #0840
         EPT: SET #082C
;*************************************************************
;
;*************************************************************
;'IBCD' IS THE PROGRAME WHICH DISPLAYS THE RESULT
;OF INTEGER ARITHMETIC OPERATIONS IN DECIMAL
;FORM.
;-------------------------------------------------------------
        IBCD: CALL CAR      ;CARRIAGE RETURN
                            ;AND LINE FEED.
              LDA #1370     ;LOAD THE SIGN AND OVERFLOW
                            ;INFORMATION FROM #1370
              STA #1376     ;STORE IT AT #1376
              CALL SCHRT    ;SIGNAL OVERFLOW (IF
                            ;ANY) AND SIGN.
              CALL CBCD     ;FIND DECIMAL NUMBER
                            ;OF RESIDUE NUMBER.
              CALL NCHRT    ;PRESENT THE RESULT
                            ;IN DECIMAL FORM.
              JMP TEST      ;SIGNAL TO GET OPERATION.
;*************************************************************
```

```
;*****************************************************
;'FBCD' IS THE PROGRAME WHICH DISPLAYS THE RESULT
;FLOATING POINT ARITHMETIC OPERATIONS IN DECIMAL
;FORM.
;-----------------------------------------------------

      FBCD: CALL CAR      ;CARRIAGE RETURN AND
                          ;LINE FEED.
            CALL MNT      ;SIGNAL 'M='.
            LDA #137F     ;LOA THE SIGN OF
                          ;THE MANTISSA OF THE
                          ;RESULT FROM #137F.
            STA #1371     ;STORE IT AT #1371.
            CALL SCHRT    ;SIGNAL OVERFLOW (IF
                          ;ANY) AND SIGN.
            CALL CBCD     ;FIND THE DECIMAL OF
                          ;THE RESIDUE NUMBER.
            CALL NCHRT    ;PRESENT THE RESULT IN
                          ;DECIMAL FORM.
            CALL BLN      ;GIVE 'SPACE',
            CALL EPT      ;SIGNAL 'E='.
            LDA #1374     ;LOAD THE SIGN OF EXP.
                          ;OF THE RESULT.
            STA #1371     ;STORE IT AT #1371.
            LDA #1375     ;LOAD THE EXP. OF
                          ;THE RESULT FROM #1375.
            STA #1376     ;STORE IT AT #1376.
            CALL SCHRT    ;SIGNAL OVERFLOW (IF
                          ;ANY) AND SIGN OF EXP. .
            CALL CBCD     ;FIND THE DECIMAL NUMBER
                          ;THE EXP. .
            CALL NCHRT    ;PRESENT THE EXP. IN
                          ;DECIMAL FORM.
            JMP TEST      ;SIGNAL TO GET OPERATIONS.
;*****************************************************
;
;*****************************************************
;THE SUBROUTINE 'CBCD' CONVERTS A RESIDUE NUMBER
;TO A DECIMAL NUMBER.
;-----------------------------------------------------

      CBCD: LDA #1376     ;LOAD THE RESIDUE NUMBER
                          ;WHICH IS TO BE CONVERTED
                          ;INTO DECIMAL NUMBER
                          ;FROM #1376.
            STA #1391     ;STORE IT AT #1391.
            MVI A,#AA     ;'#AA' (TEN) IS THE DIVISOR.
            STA #1390     ;STORE IT AT #1390.
            CALL SSSP     ;GET THE QUOTIENT AND
                          ;THE REMAINDER.
            LDA #135A     ;LOAD THE SIGN OF
                          ;THE DIVIDEND (THE
                          ;RESIDUE NUMBER WHICH
                          ;IS TO BE CONVERTED
                          ;TO DECIMAL NUMBER).
```

```
          ANI 01        $TEST IF NEGATIVE.
          CNZ SUM       $IF YES, JUMP TO 'SUM'.
          LDA #1393     $LOAD THE REMAINDER
                       $FROM #1393.
          STA #1363
          MVI A,#01     $'#01' IS THE RESIDUE
                        $REPRESENTATION OF SIXTEEN.
          STA #1361
          CALL MU       $MULTIPLY '#01' AND THE
                        $REMAINDER TO FORM THE
                        $RESIDUE OF THE FORM 'OX'
                        $WHERE X = 0 TO 9.
          MOV E,B       $MOVE THE PRODUCT TERM
                        $FROM B-REG TO E-REG. .
          LDA #1392     $LOAD THE QUOTIENT
                        $FROM #1392.
          STA #1391     $STORE IT AT #1391.
          CALL SSSP     $GET THE QUOTIENT AND
                        $REMAIBDER OF THE DIVIDEND
                        $AT LOC. #1391.
          STA #1363     $STORE THE REMAINDER
                        $AT #1363.
          MVI A,#10     $'#10' IS THE RESIDUE
                        $REPRESENTATION OF(-15).
          STA #1361
          CALL MU       $MULTIPLY THE REMAINDER
                        $BY '#10' TO FORM THE
                        $RESIDUE OF THE FORM
                        $'XO' WHERE X= 0 TO 9.
          MOV A,B       $MOVE THE PRODUCT TERM
                        $FROM B-REG. TO A-REG. .
          STA #1361
          MOV A,E       $MOVE THE PREVIOUS PRO-
                        $DUCT TERM FROM E-REG.
                        $TO A-REG. .
          STA #1363
          CALL AD       $ADD
          MOV E,B       $MOVE THE RESULT OF ADDITION
                        $FROM B-REG. TO E-REG .
          LDA #1392     $LOAD THE QUOTIENT FROM #1392.
          STA #1361     $STORE IT AT #1361.
          MVI A,01
          STA #1363
          CALL MU       $MULTIPLY '01' (SIXTEEN)
                        $AND THE QUOTIENT TO
                        $FORM THE RESIDUE OF THE
                        $FORM 'OX' WHERE X=0 TO 9.
          MOV H,B
          RET
$*************************************************************
```

```
;*************************************************
;THE SUBROUTINE 'SUM' CONVERTS A NEGATIVE
;NUMBER TO ITS POSITIVE FORM.
;-------------------------------------------------
        SUM: LDA #1392    ;LOAD THE QUOTIENT FROM
                          ;#1392.
             STA #1363
             MVI A,#FE    ;'#FE' IS THE RESIDUE
                          ;FORM OF MINUS ONE.
             STA #1361
             CALL MU      ;MULTIPLY THE QUOTIENT
                          ;AND -1 (#FE) TO
                          ;CONVERT NEGATIVE FORM
                          ;OF RESIDUE TO POSITIVE
                          ;FORM.
             MOV A,B
             STA #1392
             LDA #1393    ;LOAD THE REMAINDER
                          ;FROM #1393.
             STA #1363
             CALL MU      ;MULTIPLY THE REMAIN-
                          ;DER AND -1 (#FE) TO
                          ;CONVERT THE NEGATIVE
                          ;FORM OF REMAINDER TO
                          ;ITS POSITIVE FORM.
             MOV A,B
             STA #1393
             RET
;*************************************************
;
;*************************************************
;THE SUBROUTINE 'SCHRT' SIGNALS THE SIGN AND
;OVERFLOW (IF ANY) TO OUTPUT DEVICE.
;-------------------------------------------------
      SCHRT: LDA #1371    ;LOAD SIGN AND OVERFLOW
                          ;INFORMATION FROM #1371.
             STA #135A    ;STORE IT AT #135A.
             MOV D,A      ;MOVE IT TO D-REG. .
             ANI 02       ;TEST IF OVERFLOW.
             JZ TOR       ;IF NO, JUMP 'TOR'.
             CALL XTAR    ;OTHERWISE, SIGNAL '?' .
        TOR: MOV A,D
             ANI 01       ;TEST IF NEGATIVE.
             JZ RSI       ;IF NO, JUMP TO 'RSI'.
             CALL NEG     ;OTHERWISE, SIGNAL '-' .
             RET
        RSI: CALL POS     ;SIGNAL '+' .
             RET
;*************************************************
```

```
;*************************************************
;THE SUBROUTINE 'NCHRT' DISPLAYS DECIMAL NUMBERS.
;-------------------------------------------------
      NCHRT: MOV A,H      ;MOVE THE MOST SIGNIFICANT
                          ;DECIMAL DIGIT FROM H-REG.
                          ;TO A-REG. .
             CALL NMOUT   ;DISPLAY THE SIGNIFICANT
                          ;DIGITS.
             MOV A,E      ;MOVE TWO LEAST SIGNIFICANT
                          ;DECIMAL DIGITS FROM E-REG.
                          ;TO A-REG. .
             CALL NMOUT   ;DISPLAY TWO LEAST SIGNIFICANT
                          ;DECIMAL DIGITS TO OUTPUT
                          ;DEVICE.
             RET
;*************************************************
;
;*************************************************
;THE SUBROUTINE 'SSSP' FINDS THE QUOTIENT AND
;REMAINDER OF THE DIVIDEND (AT LOC. #1391)
;AND THE DIVISOR (AT LOC. #1390).
;-------------------------------------------------
       SSSP: LDA #1390    ;LOAD THE DIVISOR FROM
                          ;#1390.
             STA #1363
             LDA #1391    ;LOAD THE DIVIDEND FROM
                          ;#1391.
             STA #1361
             CALL SDIV    ;DIVIDE.
             LDA #1371    ;LOAD THE SIGN OF THE
                          ;QUOTIENT FROM #1371.
             STA #1345    ;STORE IT AT #1345.
             LDA #1370    ;LOAD THE QUOTIENT FROM
                          ;#1370.
             STA #1392    ;STORE IT AT #1392.
             STA #1361
             LDA #1390    ;LOAD THE DIVISOR FROM
                          ;#1390.
             STA #1363
             CALL MU      ;MULTIPLY DIVISOR AND
                          ;QUOTIENT.
             MOV A,B      ;MOVE THE PRODUCT TERM
                          ;FROM B-REG. TO A-REG. .
             STA #1363
             LDA #1391    ;LOAD THE DIVIDEND FROM
                          ;#1391.
             STA #1361
             CALL SU      ;SUBTRACT DIVIDEND FROM
                          ;PRODUCT TERM.
```

```
                STA #1346     ;STORE THE SIGN OF
                              ;THE RESULT OF SUBTRACTION
                              ;AT #1346.
                MOV A,B
                STA #1393     ;STORE THE REMAINDER
                              ;(THE RESULT OF SUBTRACTION)
                              ;AT #1393.
                MOV D,A       ;MOVE THE REMAINDER FROM
                              ;A-REG. TO D-REG. .
                RET
;*********************************************************
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                              END
```