M. Tech. (Computer Science) Dissertation Series

Design and Implementation of a Redundant Radix-4 Coprocessor with Binary Interface

a dissertation submitted in the partial fulfilment of the requirement for the M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

Subhasis Bhattacharjee

under the supervision of

Dr. B. P. Sinha
Professor & Head
Advance Computing & Microelectronics Unit

INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road
Calcutta - 700 035

1998

Indian Statistical Institute

203, Barrackpore Trunk Road

Calcutta = 700 035

Certificate Of Approval

This is to certify that the dissertation work entitled "Design & Implementation of a Redundant Radix-4 Coprocessor with Binary Interface" submitted by Subhasis Bhattacharjee, in partial fulfillment of the requirements for M. Tech in *Computer Science* degree of the *Indian Statistical Institute* is an acceptable work for the award of the degree.

[illegible]

Prof. Bhabani P. Sinha
Professor & Head
Advance Computing & Microelectronics Unit

ACKNOWLEDGEMENTS

I would like to acknowledge my guide, Dr. B. P. Sinha for providing me invaluable support throughout this project. During the course of this project he has always given me helpful suggestions whenever I was in trouble.

I would like to thank Prof. Mrs. Susmita Sur-Koley and Prof. Mrs. Mallika De and all other members of ACM unit , for helping me directly or indirectly throughout the work.

Finally I would like to thank all my teachers and batch-mates for their support throughout this project.

Subhasis Bhattacharjee.

ABSTRACT

Arithmetic Operations in Redundant Radix-4 require lesser time and hardware complexity with respect to those in binary number system. The proposal deals with the Design and Implementation of a coprocessor for some arithmetic and logical operations which can be integrated with the main processor operating in binary number system. The implementation is likely to be done using FPGA modules.

CONTENTS

Sl. No.	Topic	Page No
1.	Introduction	1
2.	Redundant Radix-4 Number System	2
3.	Design of Coprocessor	7
4.	Implementation	14
5.	Conclusion	15
6.	Reference	16

1. Introduction

Coprocessors are very common in today's computing environment whether they are built in single chip microprocessor or attached separately. Coprocessor runs parallely with the master microprocessor to serve fast arithmetic computations. The data representation inside the processor may significantly reduce the time complexity of the operation. That's what happens when data in the coprocessor is represented in redundant radix-4 number system.

Encouraged by "Fast Parallel Multiplication using redundant Quaternary Number system" a paper by Prof. Mallika De & Prof. B. P. Sinha , this dissertation is on the Design and Implementation of Redundant Radix-4 Coprocessor with Binary Interface which runs at a much faster speed than the conventional one.

The advantages of RR4 number system for arithmetic computations are many. The redundancy in RR4 number system helps to perform the carry-propagation-free addition of two numbers which leads to the constant time addition. Also subtraction reduces to addition if we negate the number to be subtracted. Multiplication of two RR4 numbers can be performed in $\lceil 1/2 \ \log_2 m \rceil$ time. Again as division can be done by repeated multiplication, the division operation can be achieved in $O(\log^2 m)$ time.

To make this coprocessor compatible with the conventional microprocessor which usually runs on binary number system , interface is organised in binary number system. The exchange of the data between microprocessor and coprocessor is only in binary number system. The data in the coprocessor is first converted into redundant radix-4(RR-4) number system. Subsequent processing is done in RR-4 number system and the result is again converted into binary number system.

The conversion of binary number into RR-4 can be done in constant time and re-conversion time is $O(\log n)$. Though this seems to increase the overall time complexity but using suitable algorithms for different arithmetic operations in RR-4 number system the total time complexity can be reduced.

For certain logical operations like AND, OR ,EXCLUSIVE-OR etc. suitable RR4 equivalence are not known. Operation like SHIFT & ROTATE have RR4 equivalences , but they create unnecessary circuitry complexity and take little more time than that of in binary. As there is a suitable interface, logical operations are done in binary.

2.Redundant Radix - 4 Number System

A radix-4 number system uses four distinct digits for coefficients and the coefficients are multiplied by powers of 4. Here coefficients are all positive only. In redundant radix-4 (RR-4) number system each coefficients can also be negative. So thereby in RR-4, we have coefficients ranging from -3 to 3 including 0.

To encode these coefficients into binary computer we require three binary digits. If we denote *msb* of the encoded symbol to represent sign of the coefficients, where 0 and 1 represents positive and negative sign , the remaining two digits will represent the magnitude. This encoding maps set of coefficients {-3,-2,-1,0,1,2,3} to the set of binary symbols {111,110,101,000,001,010,011}.

2.1. Binary to RR4 Conversion

If the number is given in sign-magnitude form , then we can convert it n RR-4 number system by grouping every pair of bits of magnitude part starting from the least significant bit(*lsb*) side (padding a '0' bit at the extreme left, if the number of bits in the magnitude part is odd), and then attaching the sign bit ('0' if the number is positive and '1' otherwise) to every such group. Each such resulting group of 3 bits will constitute one digit in equivalent RR-4 number system.

If binary number is given in 2's complement form, as it is commonly used, then first check its most significant bit(*msb*). If it is '0', then proceed with the remaining bits of the number in the same way as above to get the required number in RR-4 system. If the msb is '1', then complement the remaining bits of the binary number and then group them pair wise starting from the lsb side(by padding a '0' at the left if necessary). If any group of bits is '11' then the corresponding RR-4 digit will be -1, along with the generation of an RR-4 carry digits '1' for next higher digit position. Collect the carry digits from each such group of '11' bits to construct a carry vector in RR-4 system. The RR-4 digits corresponding to the other groups will be obtained in the same way as discussed in the preceding paragraph. Now add the three RR-4 numbers : (i) the number obtained from the pairs of bits as above, (ii) the carry vector and (*iii*) a carry of '1' at the least significant RR-4 digit position, to get a new number in RR-4 system. Finally the sign bit of each RR-4 digit of this sum is complemented to get the required equivalent RR-4 number.

Based on the above principle following logic has been developed –

Let $A_7A_6......A_0$ be a 8 bit 2's complement binary number, which is to be converted in RR4 number system, where A_7 is the sign bit. Taking pair wise bits from binary number (starting from *lsb*) and using the following logic B_k and C_l are generated for each pair where k=0 to 3 and l=0 to 4. Let B_k be represented as sba

$$L_l: \quad a = \overline{A_7}A_0 + A_7\overline{A_0}$$
$$b = \overline{A_7}A_1 + A_7\overline{A_1}\overline{A_0}$$
$$s = A_7\overline{A_1}\overline{A_0}$$
$$C_l = A_7\overline{A_1 A_0}$$

Using the logic below Bk & Cl (for each k =l) generate Dk . Let Dk be represented as rde, then,

$$L_2: \quad e = a \oplus C0$$
$$d = b \oplus aC0$$
$$r = C_0(e + d) \oplus s$$

Therefore the RR4 equivalence of the binary number $A_7A_6.....A_0$ is $D_3....D_0$.

2.2. Redundant Radix-4 to Binary Conversion

An RR4 number may contain both positive and negative digits. If there is no negative digit then to convert it into binary each if the digits of RR$ number is changed to binary deleting its sign bit. But if the RR4 number has negative digit then to convert it into binary we do as follows:

Two vectors are generated , one with the positive digits putting 0 in the place of negative digits & the other with negative digits putting 0 in the place of positive digit . The second vector is subtracted (using 2'scomplement addition)from the first one to get the binary number equivalent to the given RR4 number .

Based on the above principle following logic has been developed :-

Let $A_3,..,A_0$ be a RR4 number ,where each A_k is represented as $sa1a_0$ .Let P & N be two binary vectors with 4 pair of bits is generated from theRR4 numbers. Let each pair is represented by p_1p_0,same is the case for N.

$$L_3: \quad p_0 = \bar{s}a_0$$
$$p_1 = \bar{s}a_1$$
$$L_4: \quad n_0 = \overline{a_0} + \bar{s}$$
$$n_1 = \overline{a_1} + \bar{s}$$

To perform the subtraction operation intermediate sum vector D and pre-carry vector E are generated. D is a binary vector with 4-pair of bits and let each pair be represented by d_1d_0. E is a binary vector with 8-pair of bits & let each pair be represented by e_1e_0.

$$\begin{aligned} L_5: \quad & d_0 = p_0 \oplus n_0 \\ & d_1 = p_1 \oplus n_1 \\ L_6: \quad & e_0 = p_0n_0 + p_0c_{in} + n_0c_{in} \\ & e_1 = c_{in}\overline{p_0}\overline{n_0} + \overline{c_{in}}\,\overline{p_0}n_0 + \overline{c_{in}}p_0\overline{n_0} \\ & e_{i0} = p_in_i \quad \text{for } i\neq 0 \\ & e_{i1} = p_i \oplus n_i \quad \text{for } i\neq 0 \end{aligned}$$

The pre-carry vector subsequently processed in $\log_2$(2*no of RR4 digits) (here 3) stages to generate final carry vector F which is clear from fig. 3(c). Each *circle* of the said figure takes four input lines & those from two different pair of E or its successor and two output lines to produce its successor. *Circle* of figure 3(c) takes 4 inputs (f_3,..,f_0) & generates 2 outputs (g_1,g_0) which implements the relation ,

$$g_0 = f_2 + f_3f_0$$
$$g_1 = f_3f_1$$

The required binary equivalent is obtained by doing Ex-OR to D & F.

2.3. Arithmetic Logic in RR-4

Four separate arithmetic logic are designed for I) Addition ii) Subtraction iii)Multiplication iv) Division.

2.3.1. Logic for Addition

In carry-propagation-free addition of RR4 numbers, four such numbers can be added at a time. Time requirement for such addition is same for 2,3 or 4 RR4 numbers. Hence the logic for addition of four RR4 numbers is taken into consideration.

The addition of four RR4 numbers X,Y,U,V is performed in two steps. In the first step the intermediate sum S_i and carry C_i at each digit position satisfying the relation $x_i+y_i+u_i+v_i=4C_i+S_i$ (i=0 to m) are determined and all C_i's & S_i's can be computed parallely. To add four RR4 numbers for each digit position 12 bit input is needed to produce an output of 6 bits(3 bit each for C_i & S_i). Let S_i & C_i be denoted as $S_ss_0s_1$ and $S_cc_0c_1$ respectively. Let S_{cp} denote sign of carry digit of the previous digit position i.e S_c of C_{i-1}. Let the rectified sum $S'i$ and carry $C'i$ for each digit position generated as per following logic be denoted as $S'_sS'_0s'_1$ & $S'_cc'_0c'_1$ respectively.

$$\begin{aligned} L_7: \quad & S'_c = S_c \\ & c'_0 = c_0 + c_1\overline{(S_c \oplus S_{cp})}(s_0 + s_1) \\ & c'_1 = c_1 \oplus \overline{(S_c \oplus S_{cp})}(s_0 + s_1) \\ & S'_s = c_1 \oplus \overline{(S_s \oplus S_{cp})}(s_0 + s_1) \end{aligned}$$

$$s'_1 = s_1$$
$$s'_0 = s_0 + s_1\overline{(S_i \oplus S_{cp})}(s_0 + s_1)$$

Now S'$_i$ & C'$_{i-1}$ are added to get the final sums Ai (represented as S0a0a1) for each digit position as per logic below. Letting C'-1 equals to 0 and C'm will be the carry RR4 digit of the final sum.

2.3.2. Logic for Subtraction

The subtraction of two RR4 numbers X & Y, (where X-Y is defined) is equivalent to the addition of RR4 no X & $\overline{Y}$. To convert Y to $\overline{Y}$, the sign bit of each RR4 digit position is reversed.

2.3.3. Logic for Multiplication

Let A & B be two RR4 numbers that are to be multiplied. Then , Pi = b$_i$A is the ith. partial product .Let jth. digit of this partial product be represented as p$_{ij}$(which is nothing but b$_i$a$_j$). Each p$_{ij}$ (for all i & j) can have a maximum value of 32, which needs two digits in RR4 system. Now the aim is to generate a digit pair [c$_{ij}$(2) c$_{ij}$(1)] with weight of c$_{ij}$(2) four times that of c$_{ij}$(1) for each digit-product p$_{ij}$ in such a way that the sum of c$_{ij-1}$(2) and c$_{ij}$(1) becomes carry-propagation-free for all j, $1 \leq j \leq m-1$.

Let S$_1$B$_1$A$_1$ and S$_2$B$_2$A$_2$ be two RR-4 digits to be multiplied, where for k=1 ,2, S$_k$ is the sign bit, B$_k$ and A$_k$ are the most and the least significant bits respectively of the magnitude of the digit. Let c$_{ij}$(1) & c$_{ij}$(2) be the pair of digits to be generated , corresponding to the product of the digits S$_1$B$_1$A$_1$ & S$_2$B$_2$A$_2$,as per the following logic. Let c$_{ij}$(1) & c$_{ij}$(2) be represented as S$_{d1}$d$_{12}$d$_{11}$ and S$_{d2}$d$_{22}$d$_{21}$ respectively, taking S$_{cp}$ as the sign of the carry digit from the previous digit position.

L$_9$: $S_{d2} = S_1 \oplus S_2$

$$S_{d2} = S_1 \oplus S_2 + (B_2\overline{A_2} + B_1\bar{A_1})\overline{(S_1 \oplus S_2 \oplus S_{cp})} + A_1A_2(B_1 \oplus B_2)$$

$$d_{11} = A_1A_2$$

$$d_{12} = A_2B_2\overline{A_1} + B_2\overline{A_2}A_1$$
$$d_{21} = (B_2\overline{A_2} + B_1\overline{A_1})\overline{[(S_1 \oplus S_2 \oplus S_{cp})(B_2A_2 + B_1A_1)} + (S_1 \oplus S_2 \oplus S_{cp})$$
$$(B_2A_2 + B_1A_1)] + A_1A_2(B_1 \oplus B_2) + B_2\overline{A_2}B_1\overline{A_1}$$
$$d_{22} = B_1B_2A_1A_2 + B_1B_2(A_1 \oplus A_2)\overline{(S_1 \oplus S_2 \oplus S_3)}$$

2.3.4. Logic For Division

Let A & B are two RR4 numbers and A/B (taking defined) is to be computed. Now the fraction A/B is normalised in such a manner that the decimal point can be assumed to the left of the first significant digit from msb of B. As a result A changes to A'. Now B, being a decimal fraction , can be put into the form (1-C) , where C is another decimal fraction. Therefore,

$$\begin{aligned} A/B &= A'/(1-C) \\ &= A'(1+C)(1+C_2)(1+C_4).....(1+C_n)]/(1-C_{2n}). \end{aligned}$$

Now as $n \to \infty$, $C_{2n} \to 0$, therefore ,
$A/B = A'(1+C)(1+C_2)(1+C_4).....(1+C_k)$.

If the value of A/B is required correct up to jth. Significant digit, the value of k will be given by the relation $k/2 < j < k$ where $k=2^m$ for some integer m.
By complementing the sign bit of each digit position of B , C is obtained.

3. Design of Coprocessor

Design of a 32-bit coprocessor with binary interface is given below.

3.1. Architecture

Input unit consists of four registers viz. AB,BB,CB,DB and output unit consists of two output registers viz. PB,QB. Contents of all these registers are defined in 32-bit binary number system. Data is loaded from microprocessor or memory to any or some the input registers. Next interface unit converts the binary data into RR4 equivalent, then pass to the arithmetic unit for the said arithmetic operation. Arithmetic unit sends the result of the operation to the interface unit for re-conversion of RR4 to binary, which subsequently goes to the output register(s) (Ref. Fig. 1(a)).

For logical operations only AB,BB serves as input registers. In this case data passes into logical unit directly without going through interface unit, which subsequently goes directly to the output register PB.

The flag register FB is a part of output unit whose content is defined in the fig. 1(b).

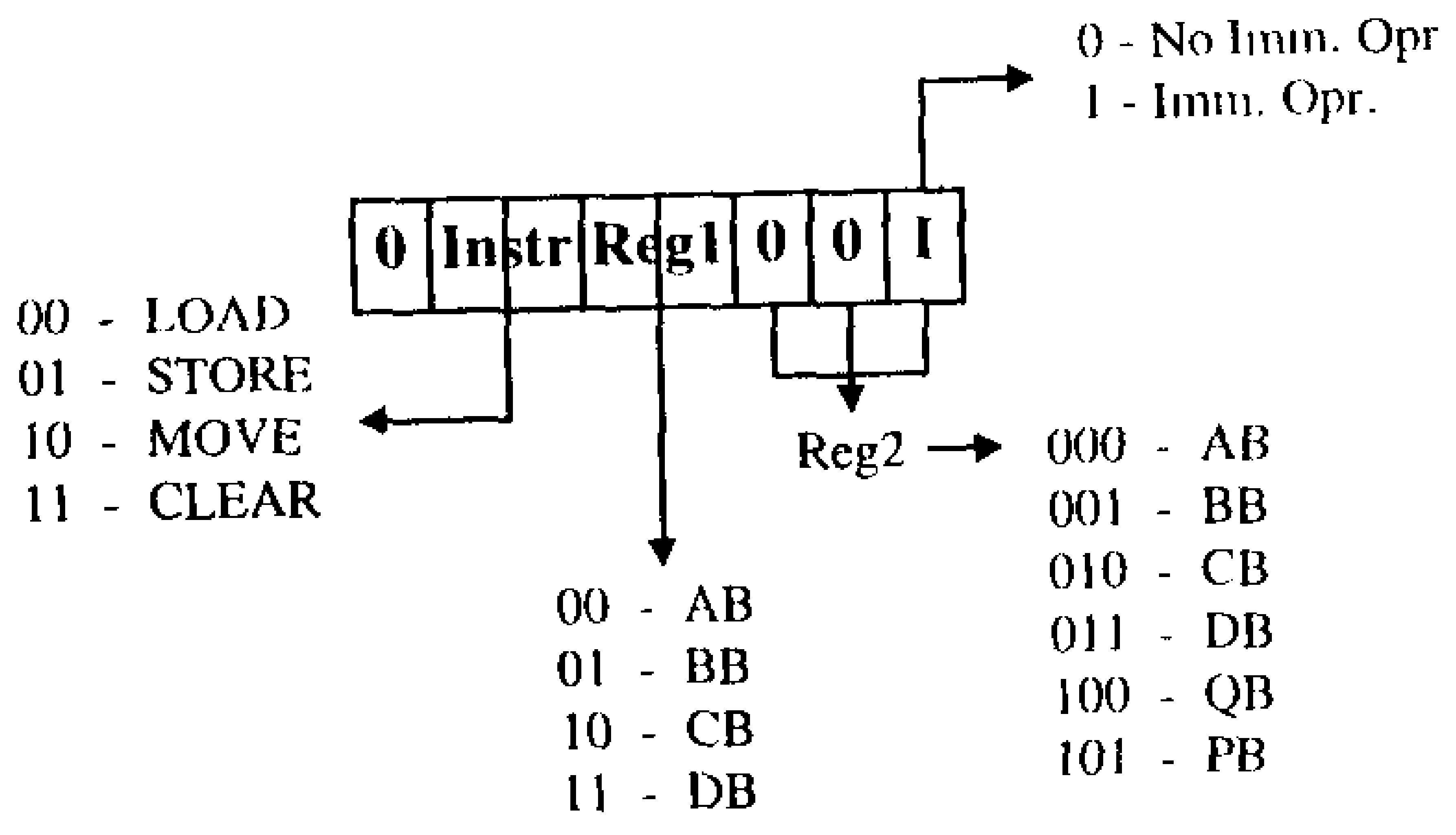
3.2. Instruction Set

Type	Instruction	Operand(s)	Description	Change of flag(s)
Data Transfer	LOAD	reg , data	reg can be any of the input registers, data is 32-bit immediate [reg] ← data	No
	STORE	reg , addr	reg can be any of the output registers, addr is 16-bit absolute address. [addr] ← (reg).	
	MOV	Rd , Rs	Rs can be any of 6 regs, but Rd can take only input regs. Rd ← (Rs)	
	CLEAR	reg	reg ← 0	
Arithmetic	ADD		[PB:QB] ← AB+BB+CB+DB	
	SUB		QB ← AB-BB	
	MUL		[PB:QB] ← AB*CB	
	DIV		[PB:QB] ← AB/CB	

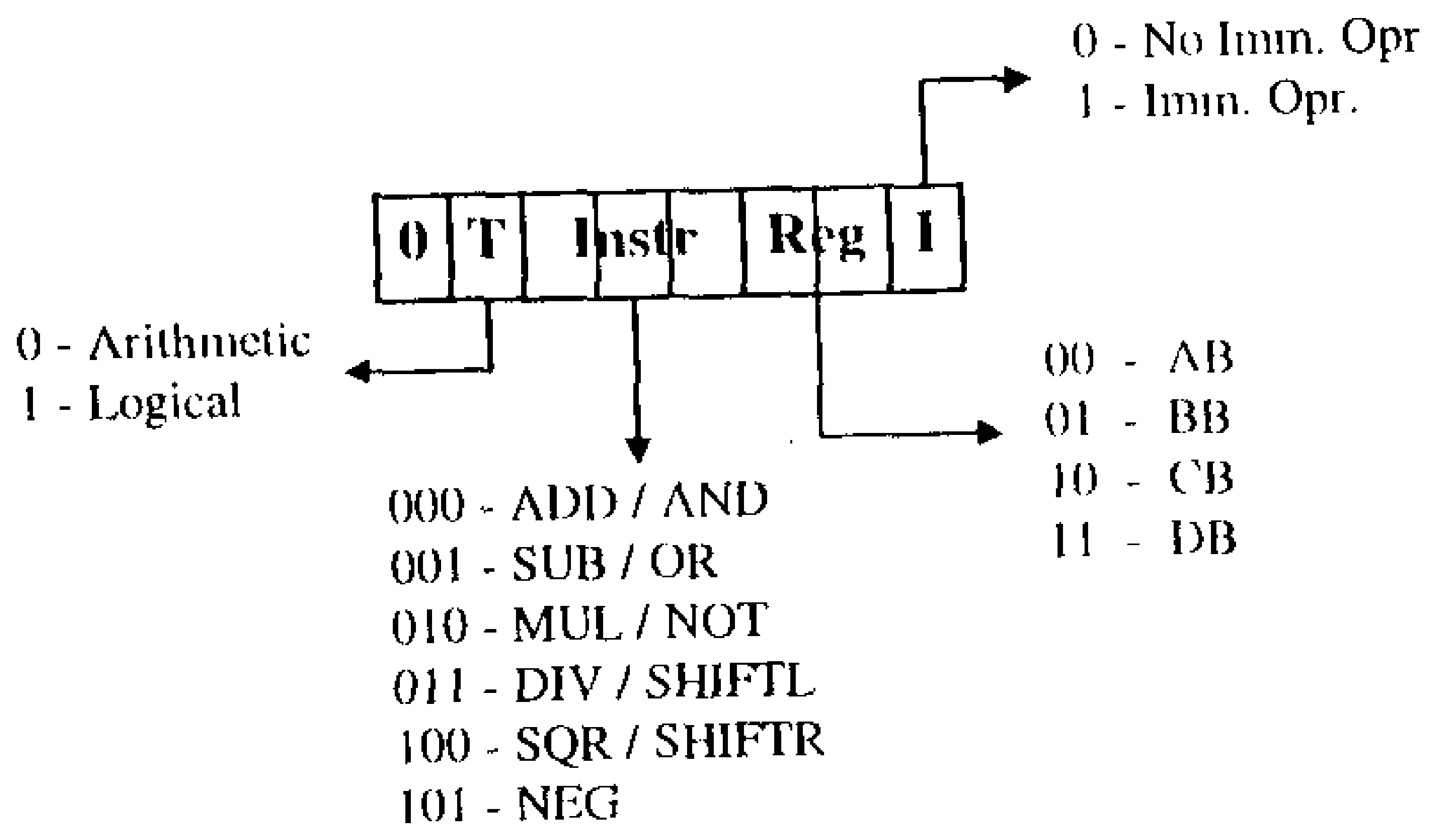
			Characteristics in PB, Fraction in QB.
	SQR		[PB:QB]← AB*AB
	NEG	reg	reg ← -reg.
Logical	AND		PB← AB AND BB
	OR		PB←AB OR BB
	NOT	reg	reg ← !reg
	SHIFTL	reg , count	reg← left shifted reg count is no of bit to shift
	SHIFTR	reg , count	reg← right shifted reg
Special	STF	addr	16bit absolute address

3.3. Instruction Encoding

For Data Transfer instruction :



For Arithmetic and Logical Instructions :



For the instruction STF :

1	0	1	1	1	0	0	1

3.4. Internal Details of Coprocessor

Coprocessor has Interface unit, Arithmetic unit, Logical unit and Control unit besides Input and Output unit. Here the units are discussed one by one with reference to the corresponding figures and schematic diagrams.

3.4.1. Binary to RR4 Interface (Ref. Fig. 2):

Fig. 2(a) show schematic diagram where AR is a 16-digit RR4 register. Fig. 2(b) shows the detailed diagram for AB to AR conversion. B_5 is the implementation of L1 of Sec. , where

{AB(0),AB(1),AB(31)} $\leftarrow$ {A0,A1,A7}
& {R33(2),R33(1),R33(0)} $\leftarrow$ {s, b, a}
& x0 $\leftarrow$ C0.

Again B6 is the implementation of L2 of Sec. , where
{AR(2),AR(1),AR(0)} $\leftarrow$ {r, d, e}

3.4.2. RR4 to Binary Interface(Ref. Fig. 3) :

Fig. 3(a) shows the schematic diagram where PR & QR are two 16-digits RR4 registers and PB & QB are two 32-bit binary registers. Fig. 3(b) explains the detailed diagram for the above conversion. B20 & B21 are the implementation of L3 & L4 of Sec. ,where

{QR(2),QR(1),QR(0)} $\leftarrow$ {s, a1, a0}
& {QP(0),QP(1)} $\leftarrow$ {p0, p1}
& {QN(0),QN(1)} $\leftarrow$ {n0, n1}

B22 & B23 are the implementation of L5 and L6 of Sec. , where
{D(0),D(1)} $\leftarrow$ {d0, d1}
& {E(0),E(1)} $\leftarrow$ {e0, e1}

Fig. 3(c) shows final carry generation from pre-carry vector for 4-digit RR4 number. This is to be extended for 16-digit RR4 number.

When PR & QR together hold the data , B25 generates final carry for MSB which is clear from fig. 3(d). B26 consists of sixteen Ex-OR , where QBi = Di Ex-OR Fi.

3.4.3. Arithmetic Unit :

Fig. 4(a) shows schematic diagram of 16-digit RR4 adder block. Fig. 4(b) explains the details of above. B_1 & B_2 are the implementation of L_7 & L_8 of Sec. , where

$\{R5(2),R5(1),R5(0)\} \leftarrow \{S_s, s_0,s_1\}$
$\{R6(2),R6(1),R6(0)\} \leftarrow \{S_c, c_0,c_1\}$
$\{R7(2),R7(1),R7(0)\} \leftarrow \{S'_s, s'_0,s'_1\}$
$\{R8(2),R8(1),R8(0)\} \leftarrow \{S'_c, c'_0,c'_1\}$
$\{R9(2),R9(1),R9(0)\} \leftarrow \{S_a, a_0,a_1\}$
$\{R10(2),R10(1),R10(0)\} \leftarrow \{R8(47),R8(46),R8(45)\}$

Following is for the ROM table for the adder.

Dsi	Ci and Si for negative DSi-1	
12	3	0
11	2	3
10	2	2
9	2	1
8	2	0
7	1	3
6	1	2
5	1	1
4	1	0
3	0	3
2	0	2
1	0	1
0	0	0

Following are the some entries in the ROM table for adder.

Address	Ci	Si
000001010011	001	010
010000010011	001	011
011011011011	011	000
001111011101	000	000
110110001010	000	101
101101101101	101	101
111111111111	111	00

Fig 5(a) shows schematic diagram of 32-digit RR4 adder using 16-digit RR4 adder of Fig. 4.

Fig 5(b) shows the details of above. R11 is a 1-digit RR4 register. R12 to R16 each is a 32-digit RR4 register which are again halved into Lower(L) and Higher(H) parts.

Fig 6(a) shows schematic diagram of 16-digit RR4 multiplier which uses 16-digit RR4 numbers from R17 & R18 to produce 32-digit RR4 number in R16.

Fig. 6(b0) & 6(b1) give the details of first & second partial product generation (P0,P1). B3 is the implementation of L9 of Sec. , where

$\{R17(2), R17(1), R17(0)\} \leftarrow \{S1, B1, A1\}$
& $\{R18(2), R18(1), R18(0)\} \leftarrow \{S2, B2, A2\}$
& $\{R19(2), R19(1), R19(0)\} \leftarrow \{Sd2, d22, d21\}$
& $\{R'19(2), R'19(1), R'19(0)\} \leftarrow \{Sd1, d12, d11\}$

Scp is generated inside B3. B4 is generated in the same way as B2.

R20 keeps the partial product vector (Pi) in expanded 32-digit form.

Fig. 6(c) shows how to add these partial product vectors to generate final result of multiplication which is kept by R16 register.

Fig 7(a) shows a schematic diagram of generation of 32-digit RR4 fractional multiplier using 16-digit RR4 multiplier of fig. 6. Fig. 7(b) shows the details of above. R21 to R32 are all 16-digit RR4 register.

Fig. 8 shows normalisation procedure during division. CR , DR are capable of simultaneous action and can shift right together. R34 is a parallely loaded 4-bit counter.

Fig. 9 shows a schematic diagram of division unit rather the complete arithmetic unit , using 32-digit RR4 fraction multiplier.

In all figures M stands for Multiplexer and DM stands for De-Multiplexer.

3.4.4. Logical Unit : Conventional.

3.4.5. Control Unit:
It has three registers viz. IR,PC and MAR (usual meaning) . Registers PC and MAR has the capability read , write (parallely) and counting , but IR has read & write capabilities.

The instruction execution cycle is divided into 4 basic operation namely, Instruction Decode(ID), Data conversion i.e. binary to RR-4(BRR), Compute(CM) and Re-conversion (RRB).

Data transfer and logical instructions does not enter into conversion and re-conversion phases. After decoding , they directly enter into CM.

For some instructions the timing & control flows are given below.

1. LOAD reg,data

```
ID.t0 : LOAD←1,reg←1,PC←PC+1,Busy←1
   t1 : RD PC, ECC←CM, t← t0

LOAD.CM.t0 :      RD PC
         t1:      WR MAR
         t2:      RD MAR, M←1,RD←1
         t3:      WR reg[7:0]
         t4:      RD reg[7:0], PC←PC+1, MAR←MAR+1
         t5:      RD MAR, M←1,RD←1
         t6:      WR reg[15:8]
         t7:      RD reg[15:8], PC←PC+1, MAR←MAR+1
         t8:      RD MAR, M←1,RD←1
         t9:      WR reg[23:16]
         t10:     RD reg[23:16], PC←PC+1, MAR←MAR+1
         t11:     RD MAR, M←1,RD←1
         t12:     WR reg[31:24]
         t13:     RD reg[31:24], PC←PC+1, MAR←MAR+1
         t14:     RD PC, ECC←ID, Busy←0, t←t0
```

2. ADD

```
ID.t0 : ADD←1,PC←PC+1,Busy←1,type←0
   t1 : RD PC, ECC←BRR, t← t0
   BRR.t0:   RD AB,BB,CB,DB
       t1:   WR R33
       t2:   RD R33
       t3:   WR AR,BR,CR,DR
       t4:   RD  AR,BR,CR,DR, ECC← CM, t←t0

ADD.CM.t0:   [M1,M2,M3,M4]S1S0←10
       t1:   WR R1 to R4, CLEAR R0
       t2:   RD R1 to R4
       t3:   WR R5,R6
       t4:   RD R5,R6
```

```
t5:   WR R7,R8
t6:   RD R7,R8
t7:   WR R9,R10
t8:   RD R9,R10 , $ECC \leftarrow RRB$, $t \leftarrow t0$
```

From these control flow, the control signals have beeen generated for different registers and counters.

4. Implementation

The part of the design is verified using verilog HDL. The Binary to **RR4** conversion and the re-conversion is running properly.

For other part of the coprocessor ,adder takes significant amount of memory to keep the ROM table. Also entries in the ROM table has to be done. The simulation is giving exact result. The coprocessor is designed for 32-bit binary number, but for the sake of simplicity, simulation is done for 8-bit coprocessor.

As the exact way to simulate on FPGA and proper connections are not known, as it is generally assisted with software tool, it could not be implemented on FPGA.

5. Conclusion

This design of the coprocessor can execute a small set of operations. This show the scope to design a full phase coprocessor which will run in RR-4 number system in near future.

By hand designing of the coprocessor can not assure the optimality of the resources used. The time optimality is also to be considered for future design.

6. References

[1] Mallika De and Bhabani P.Sinha, *Fast Parallel Multiplication Using Redundant Quaternary Number System* : IEEE 1989.
[2] P. Abouzeid, Michel Crastes, *Input-Driven Partitioning Method and Application to Synthesis on Table-Lookup-Based FPGA.: IEEE 1993.*
[3] J.P.Hayes, Computer Architecture.

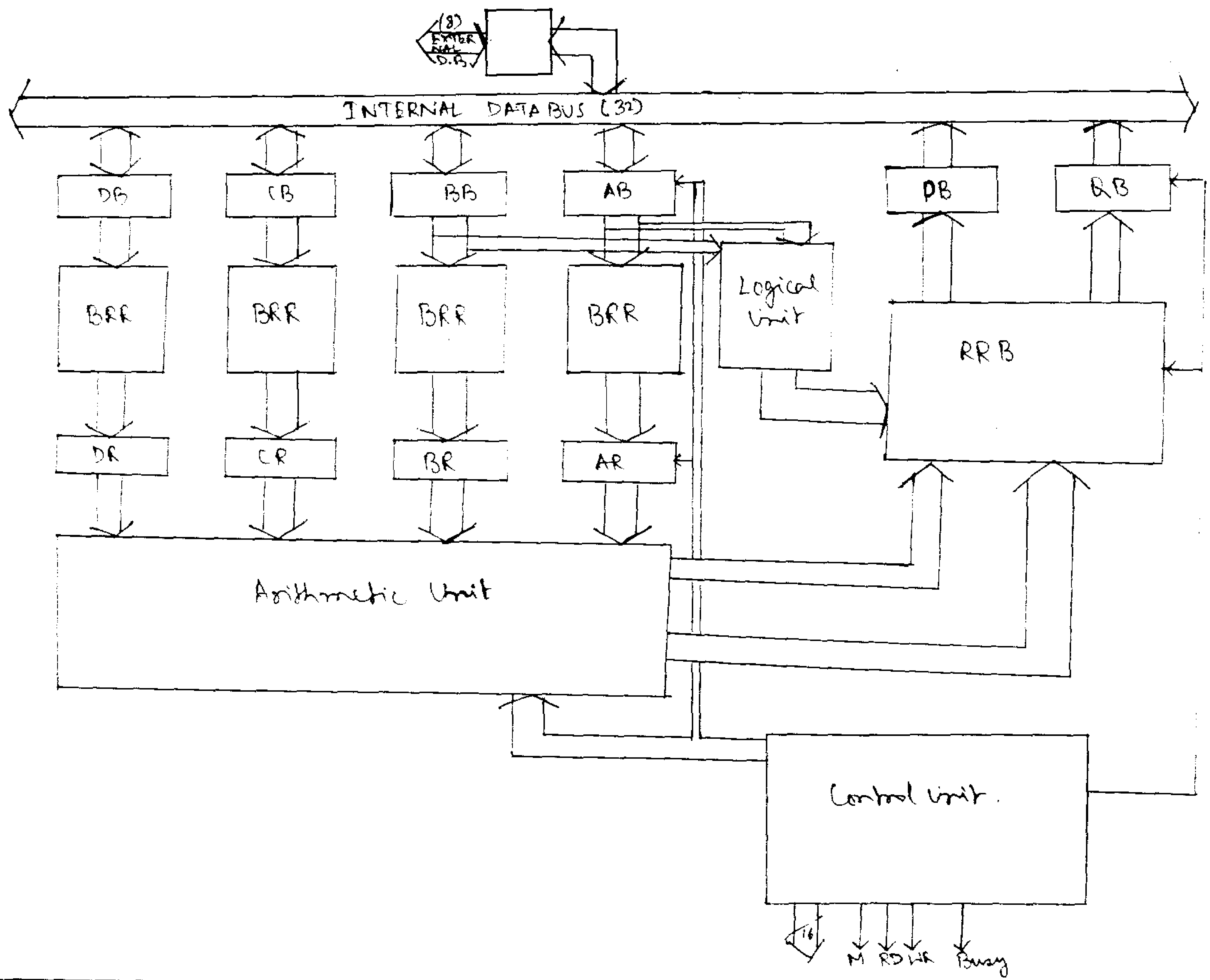

(8)
EXTERNAL D.B.
INTERNAL DATABUS (32)
DB
CB
BB
AB
PB
QB
BRR
BRR
BRR
BRR
Logical Unit
RRB
DR
CR
BR
AR
Arithmetic Unit
Control Unit.
16
M
RD
WR
Busy

Fig. 1

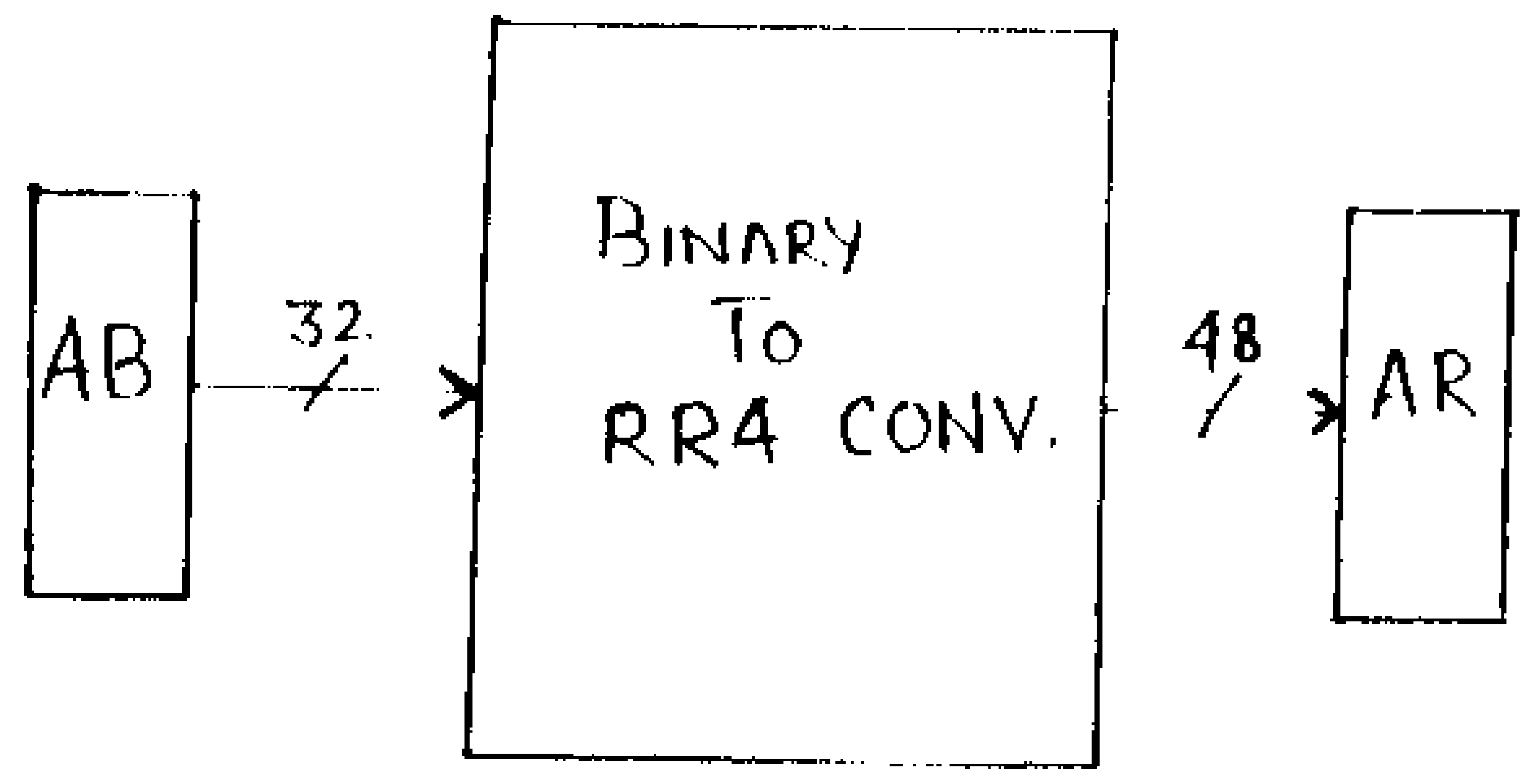

AB
32
BINARY To RR4 CONV.
48
AR

Fig. 2(a)

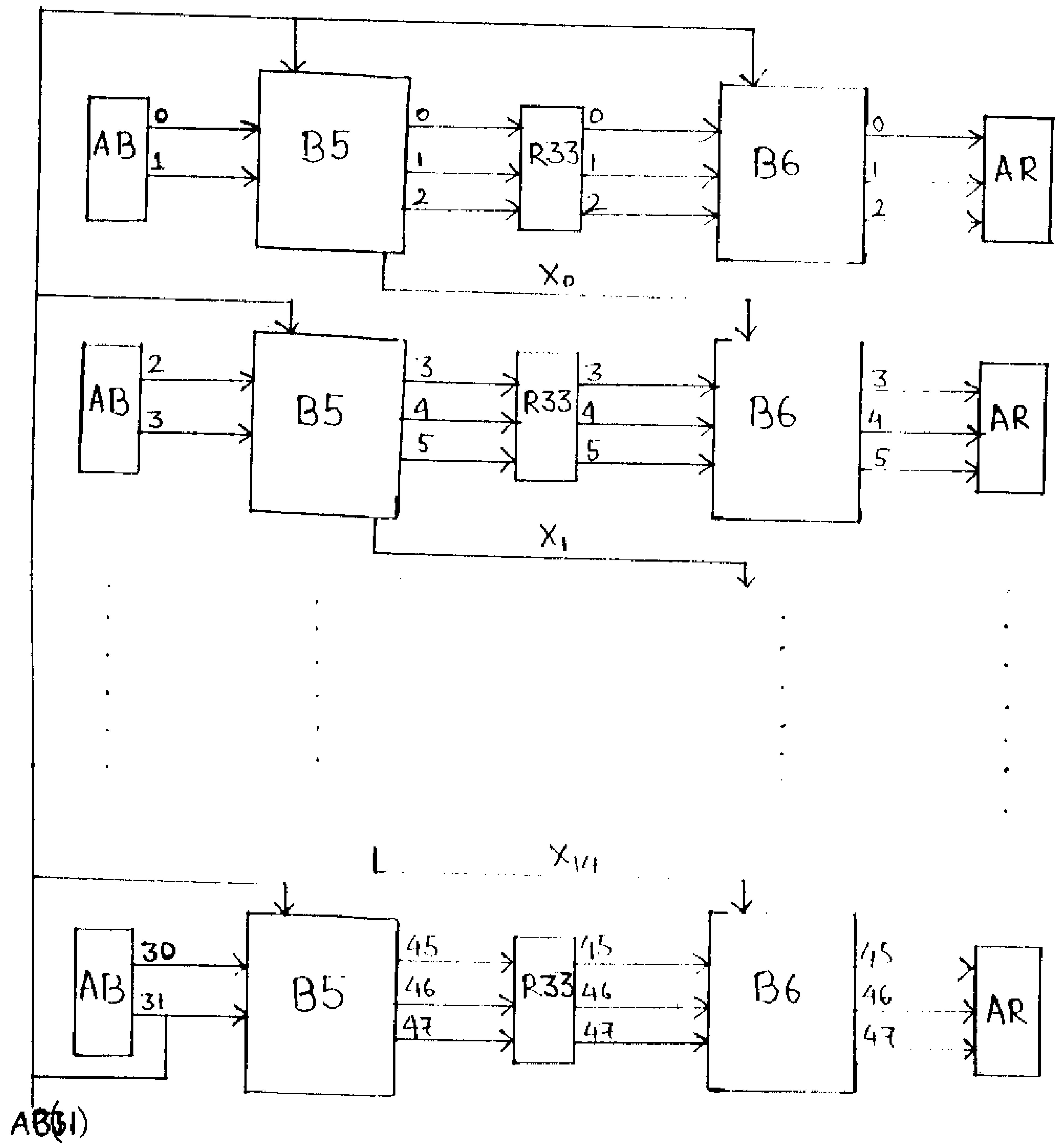
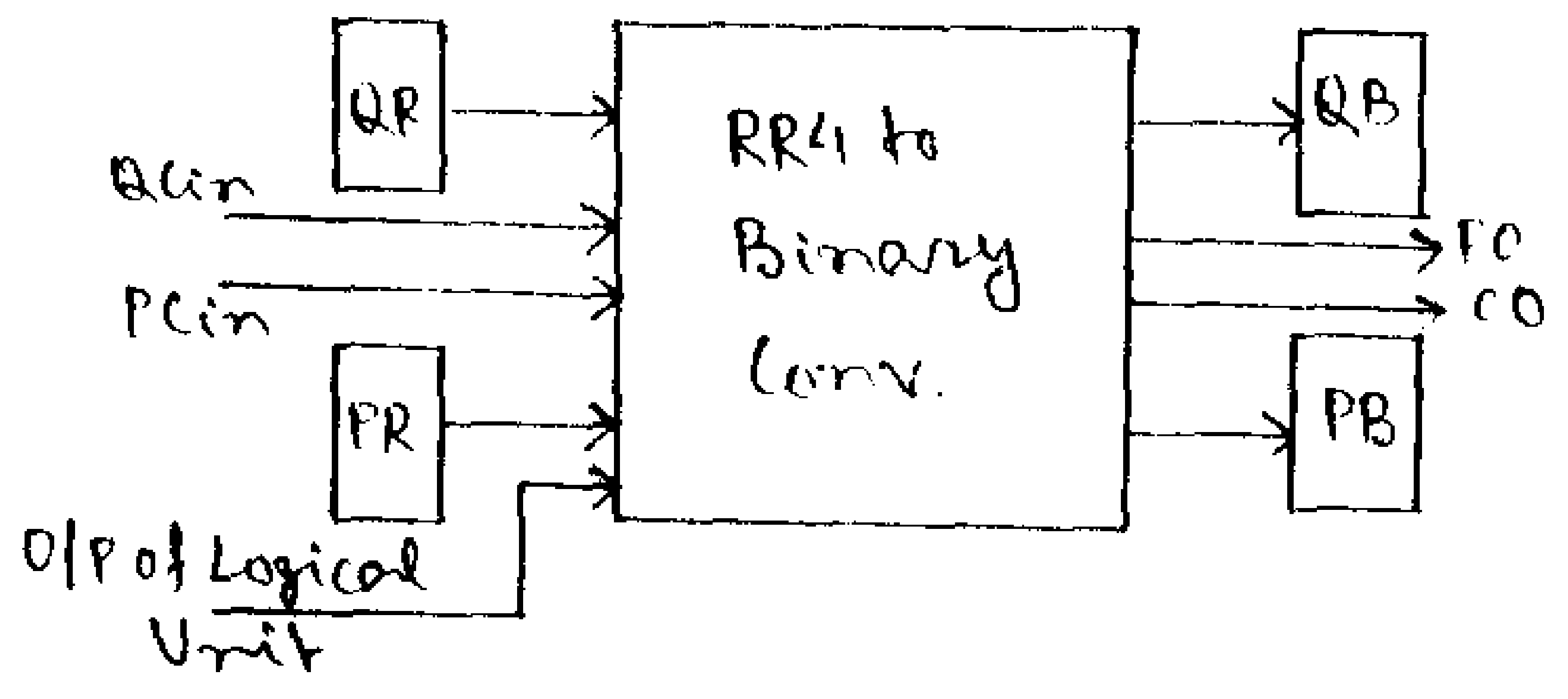
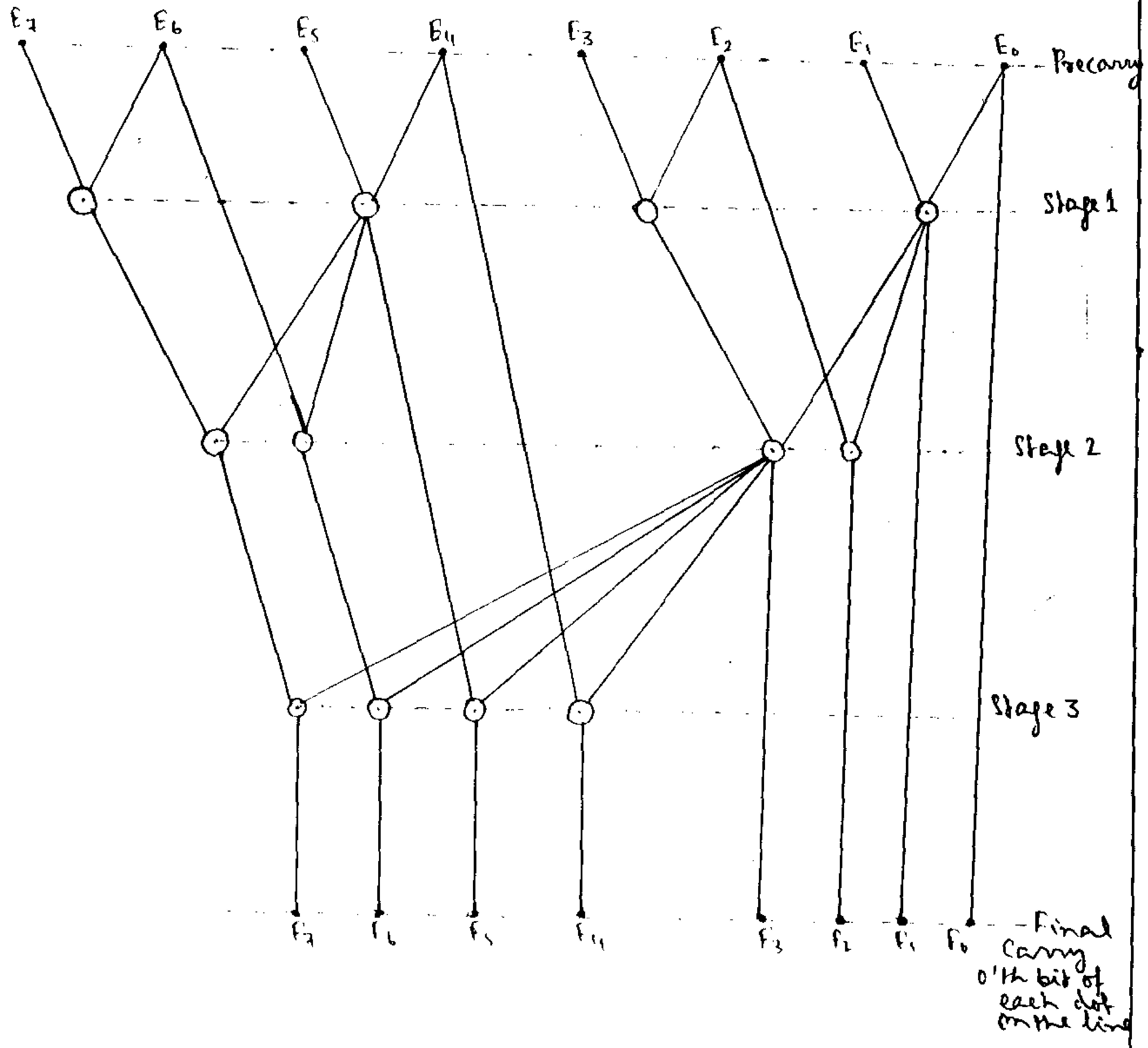

AB
0
1
B5
R33
B6
AR
2
X_0
3
4
5
X_1
X_{14}
30
31
45
46
47
AB(31)

Fig. 2(b)



QR
QCin
PCin
PR
O/P of Logical Unit
RR4 to Binary Conv.
QB
PC
CO
PB

Fig. 3 (a)



E7
E6
E5
E4
E3
E2
E1
E0
Precarry
Stage 1
Stage 2
Stage 3
F7
F6
F5
F4
F3
F2
F1
F0
Final carry
O'th bit of each dot on the line

Fig. 3(C)

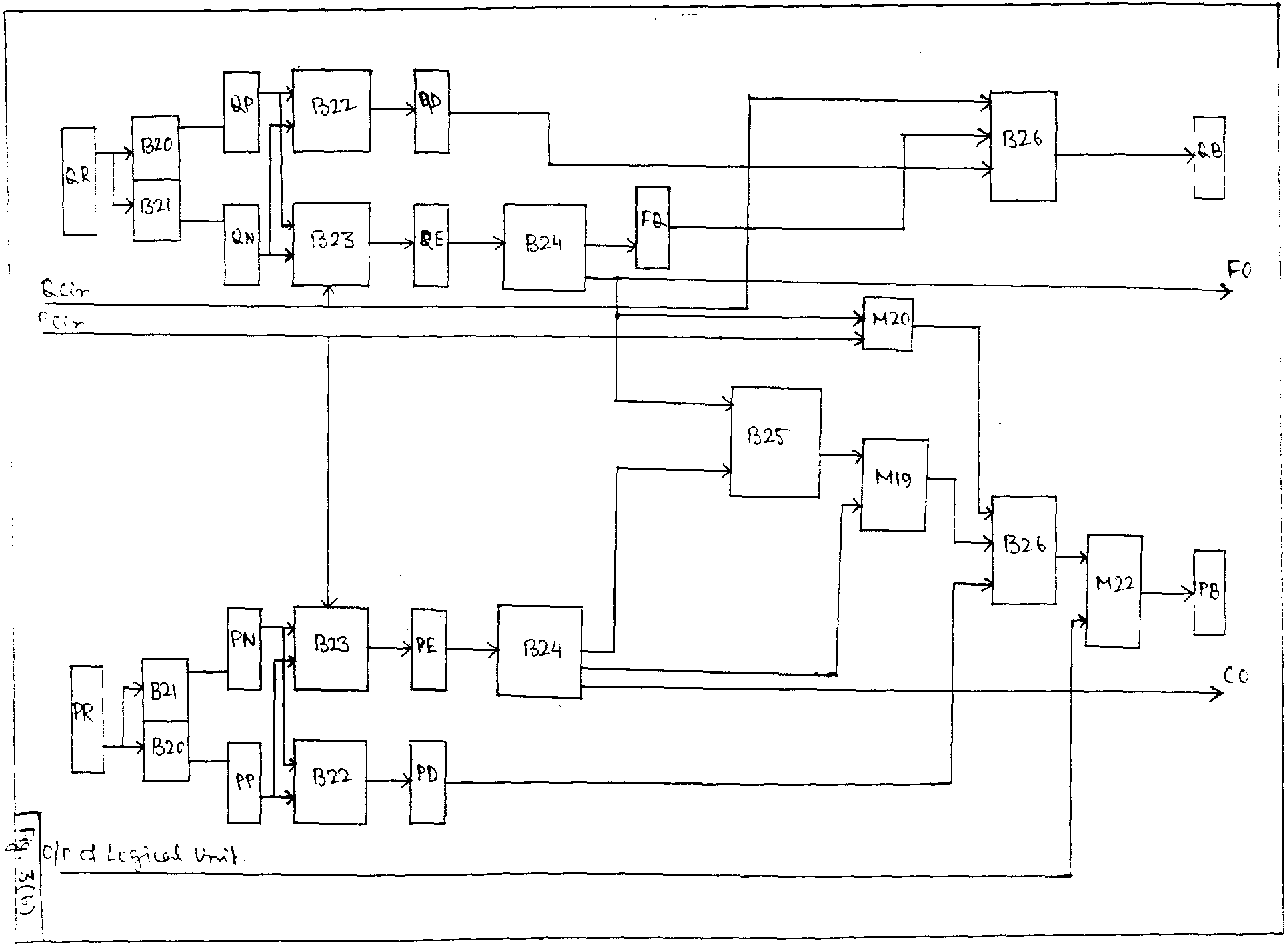

QR
B20
B21
QP
QN
B22
B23
QD
QE
B24
FQ
B26
QB
FO
Qcir
Pcir
M20
B25
M19
B26
M22
PB
CO
PR
B21
B20
PN
PP
B23
B22
PE
PD
B24
O/P of Logical Unit.

Fig. 3(b)

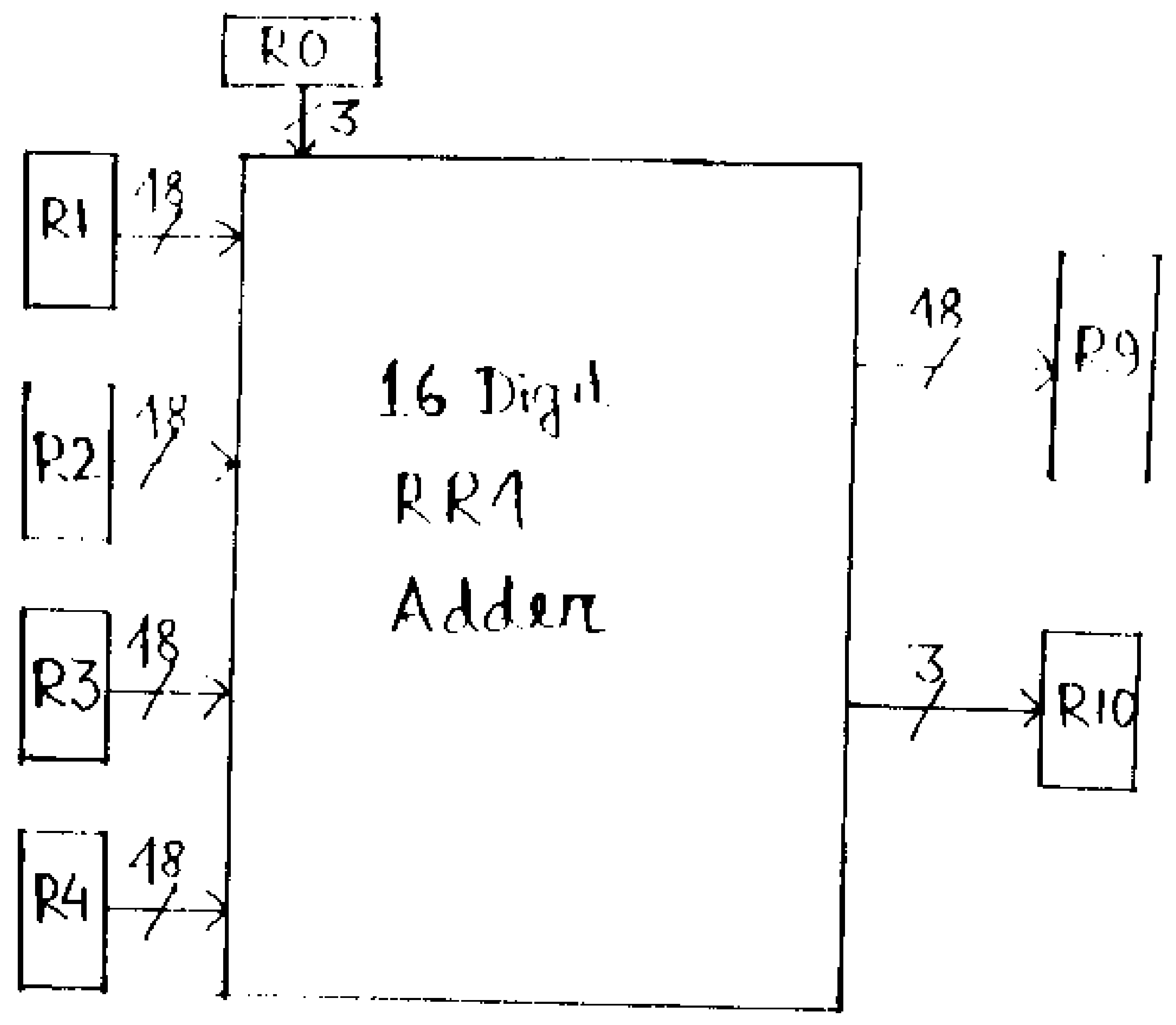

R0
3
R1
18
R2
18
R3
18
R4
18
16 Digit
RR1
Adder
18
R9
3
R10

Fig 4(a)

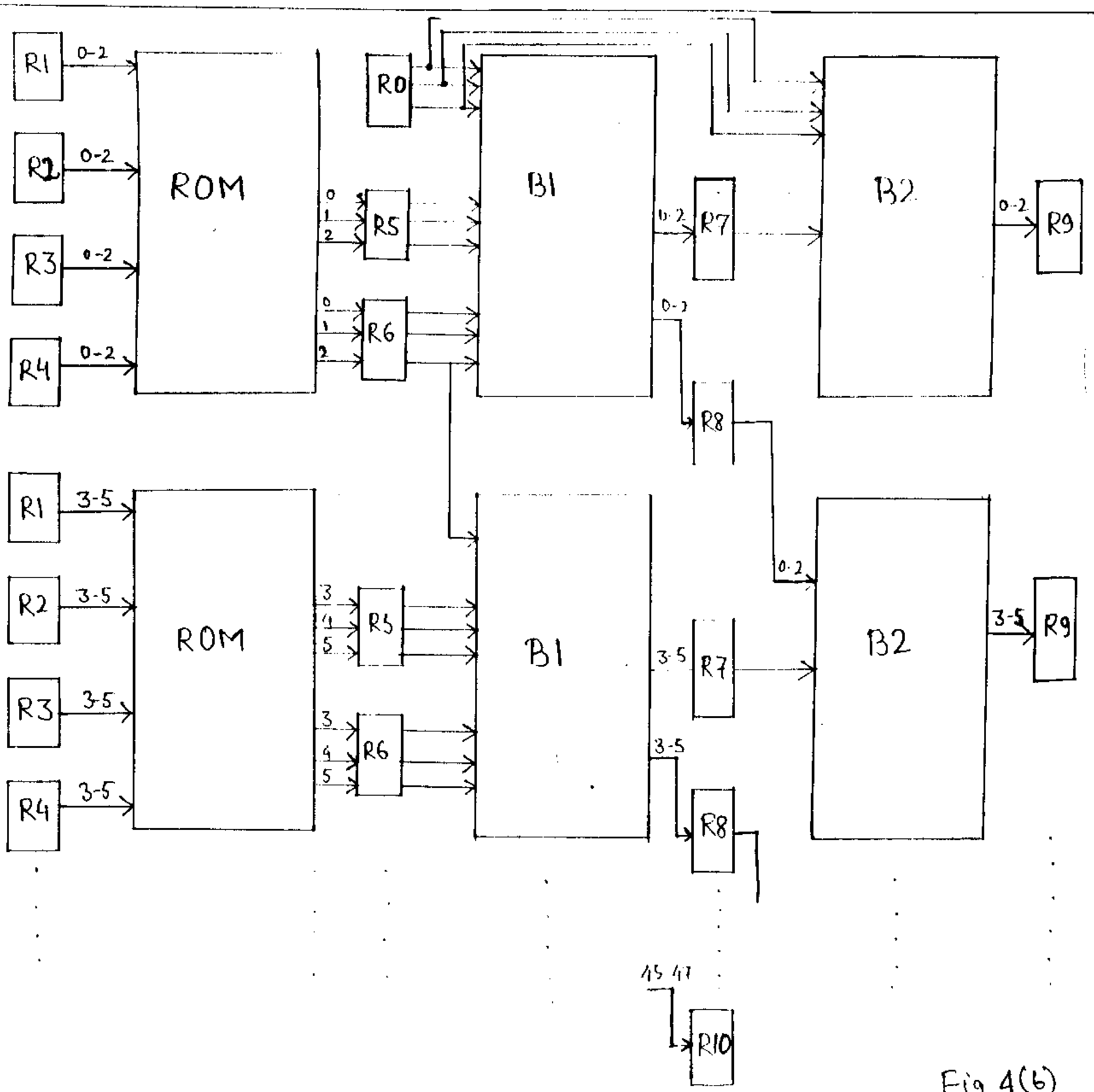

R1
0-2
R2
0-2
R3
0-2
R4
0-2
ROM
R0
R5
R6
B1
0-2
R7
0-2
R8
B2
0-2
R9
R1
3-5
R2
3-5
R3
3-5
R4
3-5
ROM
R5
R6
B1
3-5
R7
3-5
R8
0-2
B2
3-5
R9
45-47
R10

Fig. 4(b)

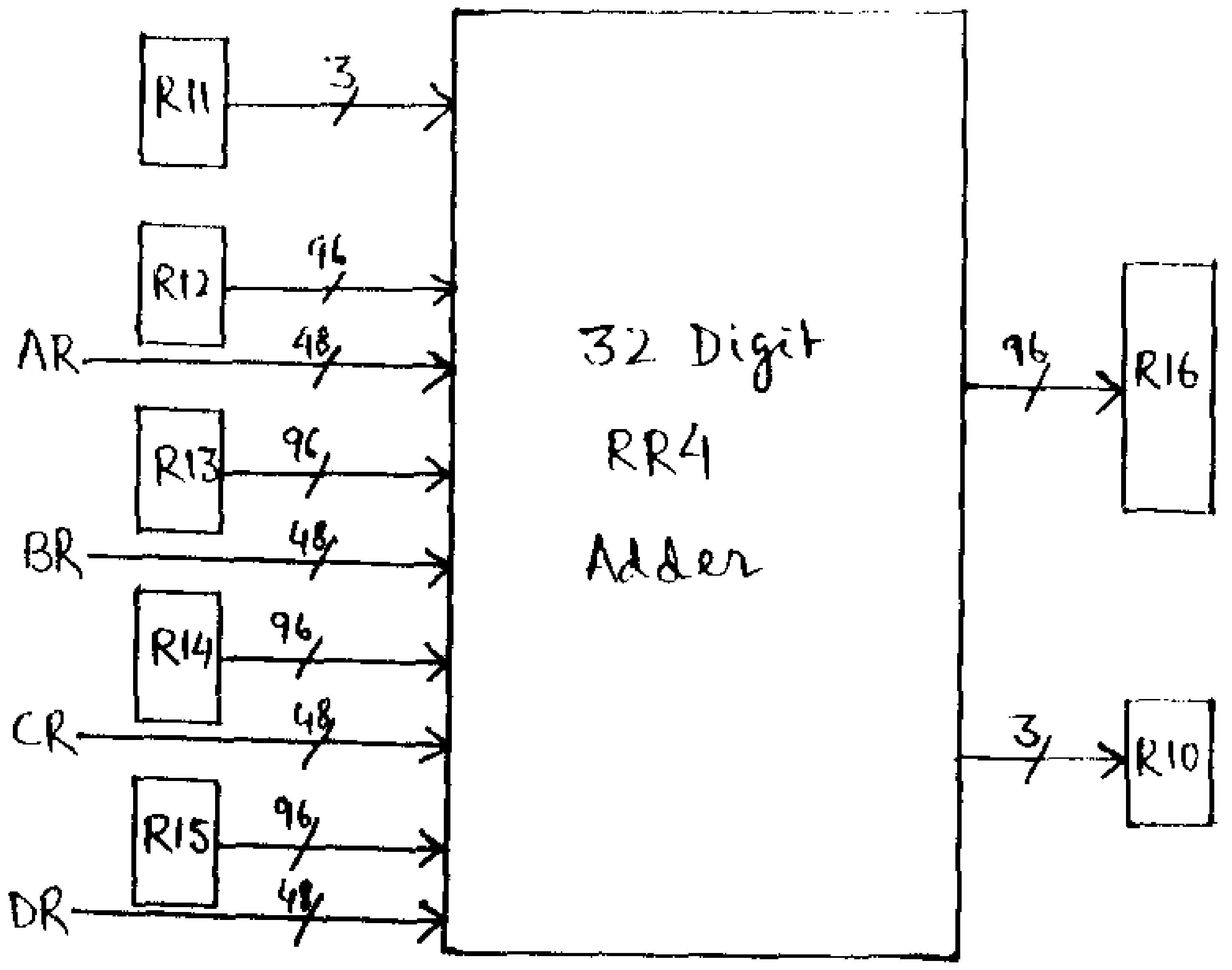

R11
3
R12
96
AR
48
R13
96
BR
48
R14
96
CR
48
R15
96
DR
48
32 Digit RR4 Adder
96
R16
3
R10

Fig. 5(a)

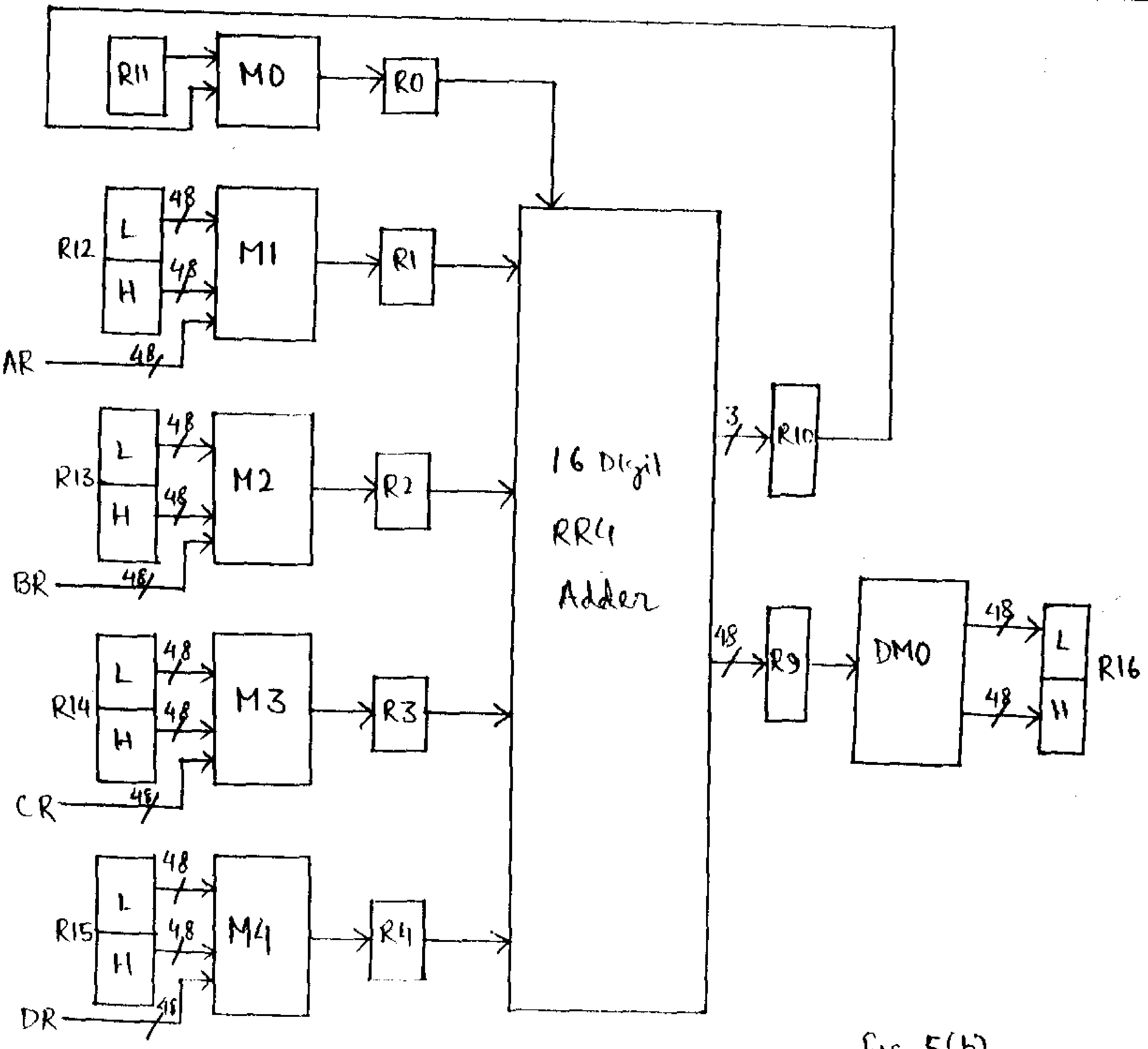

R11
M0
R0
R12
L
H
48
48
M1
R1
AR
48
R13
L
H
48
48
M2
R2
BR
48
R14
L
H
48
48
M3
R3
CR
48
R15
L
H
48
48
M4
R4
DR
48
16 Digit RR4 Adder
3
R10
48
R9
DM0
48
48
L
H
R16

Fig. 5(b)

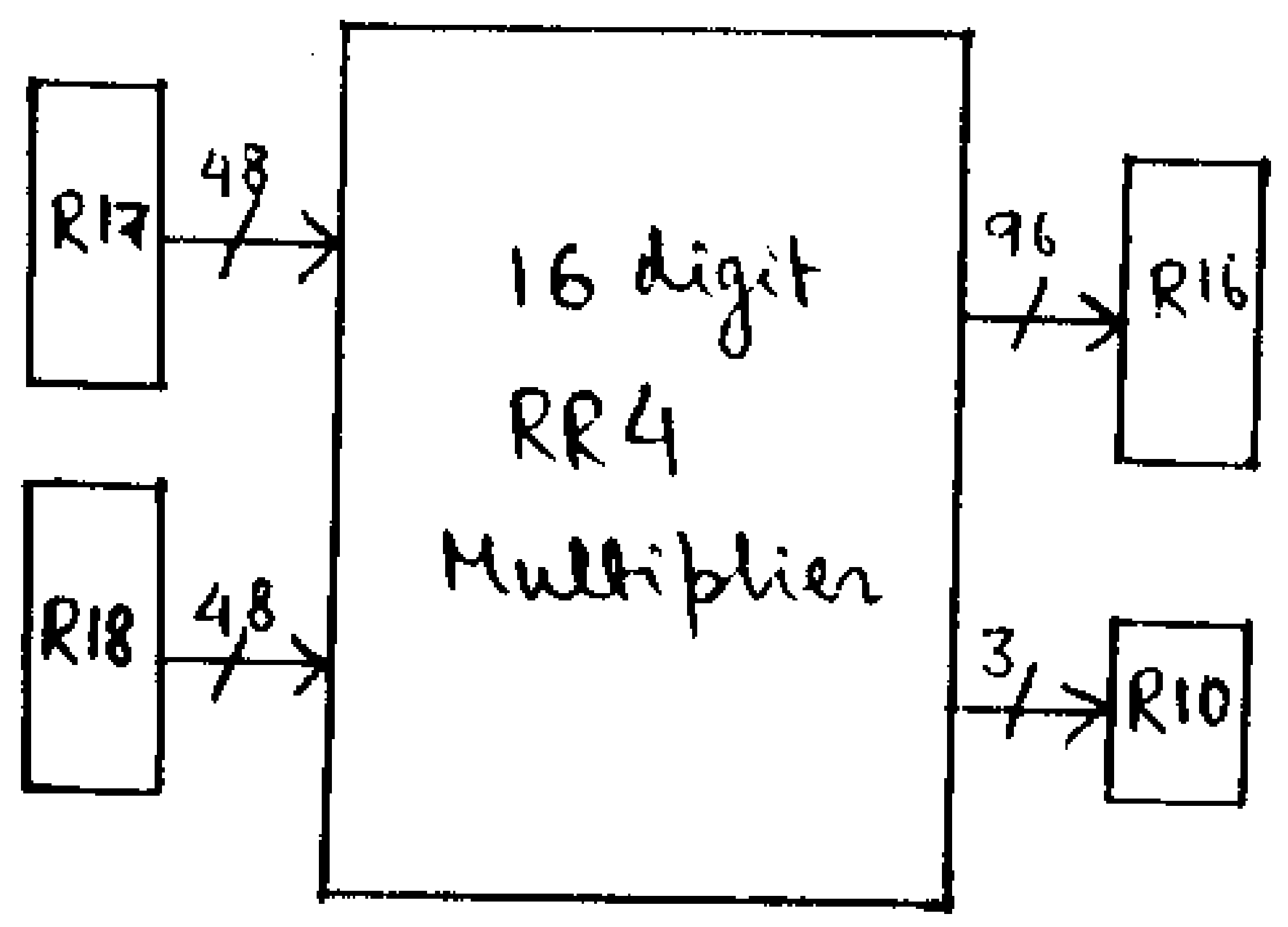

R17
48
R18
48
16 digit RR4 Multiplier
96
R16
3
R10

Fig. 6 (a)

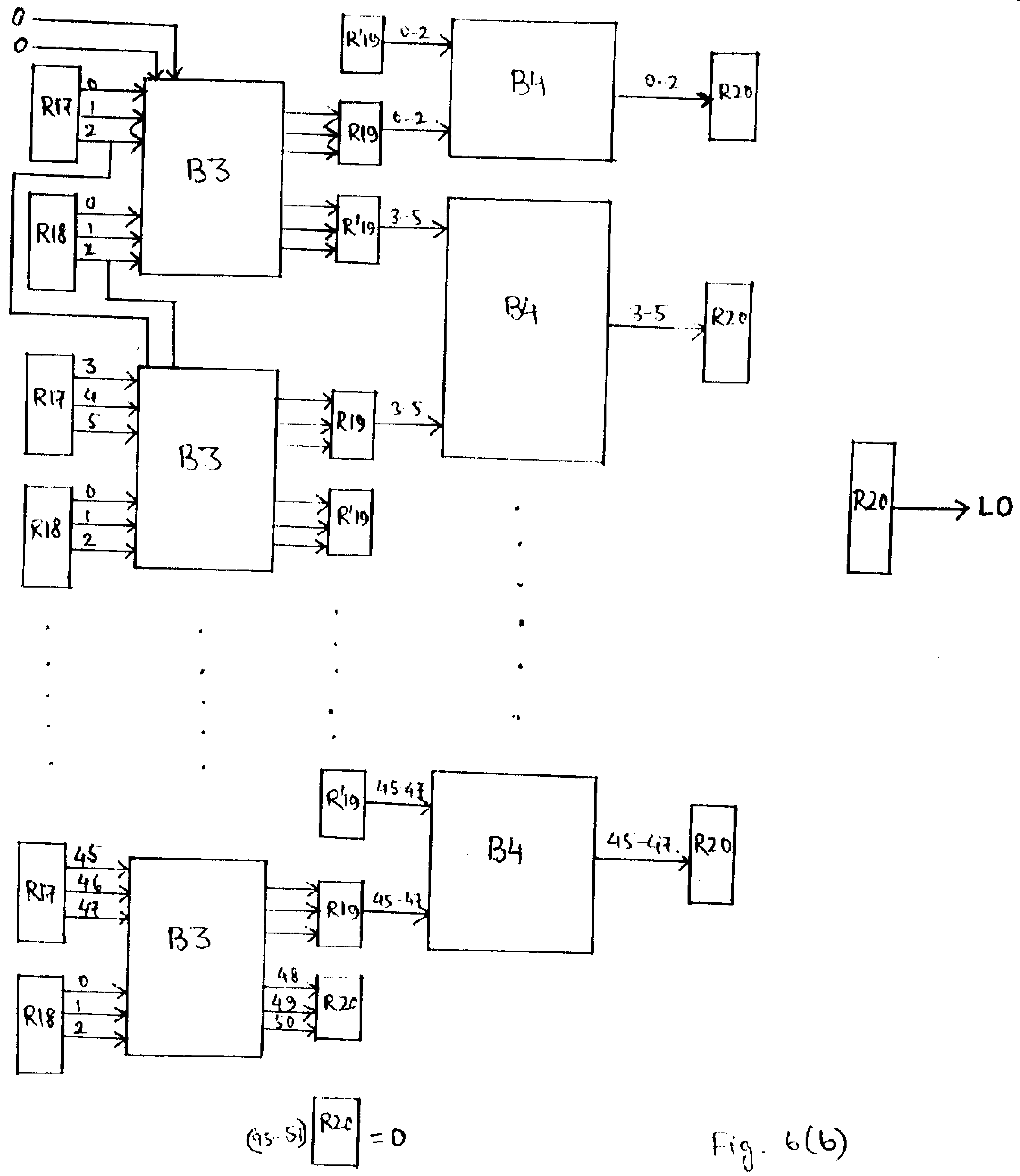
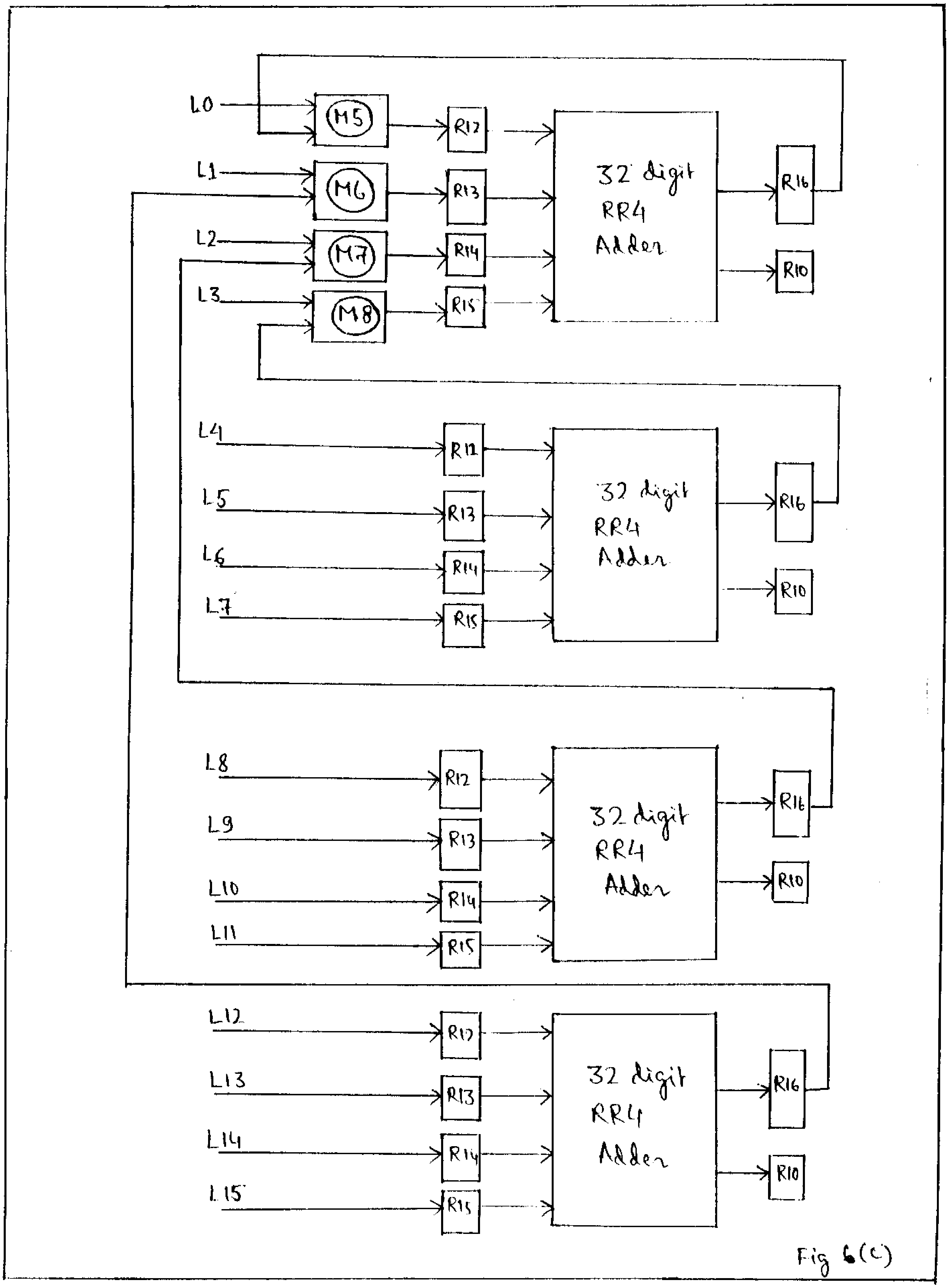

0
0
R17
0
1
2
R18
0
1
2
B3
R'19
0-2
R19
0-2
B4
0-2
R20
R'19
3-5
B4
3-5
R20
R17
3
4
5
R18
0
1
2
B3
R19
3-5
R'19
R20
LO
R'19
45-47
B4
45-47
R20
R17
45
46
47
R18
0
1
2
B3
R19
45-47
48
49
50
R20
(95-51) R20 = 0

Fig. 6(b)


L0
L1
L2
L3
M5
M6
M7
M8
R12
R13
R14
R15
32 digit RR4 Adder
R16
R10
L4
L5
L6
L7
R12
R13
R14
R15
32 digit RR4 Adder
R16
R10
L8
L9
L10
L11
R12
R13
R14
R15
32 digit RR4 Adder
R16
R10
L12
L13
L14
L15
R12
R13
R14
R15
32 digit RR4 Adder
R16
R10
Fig 6(c)

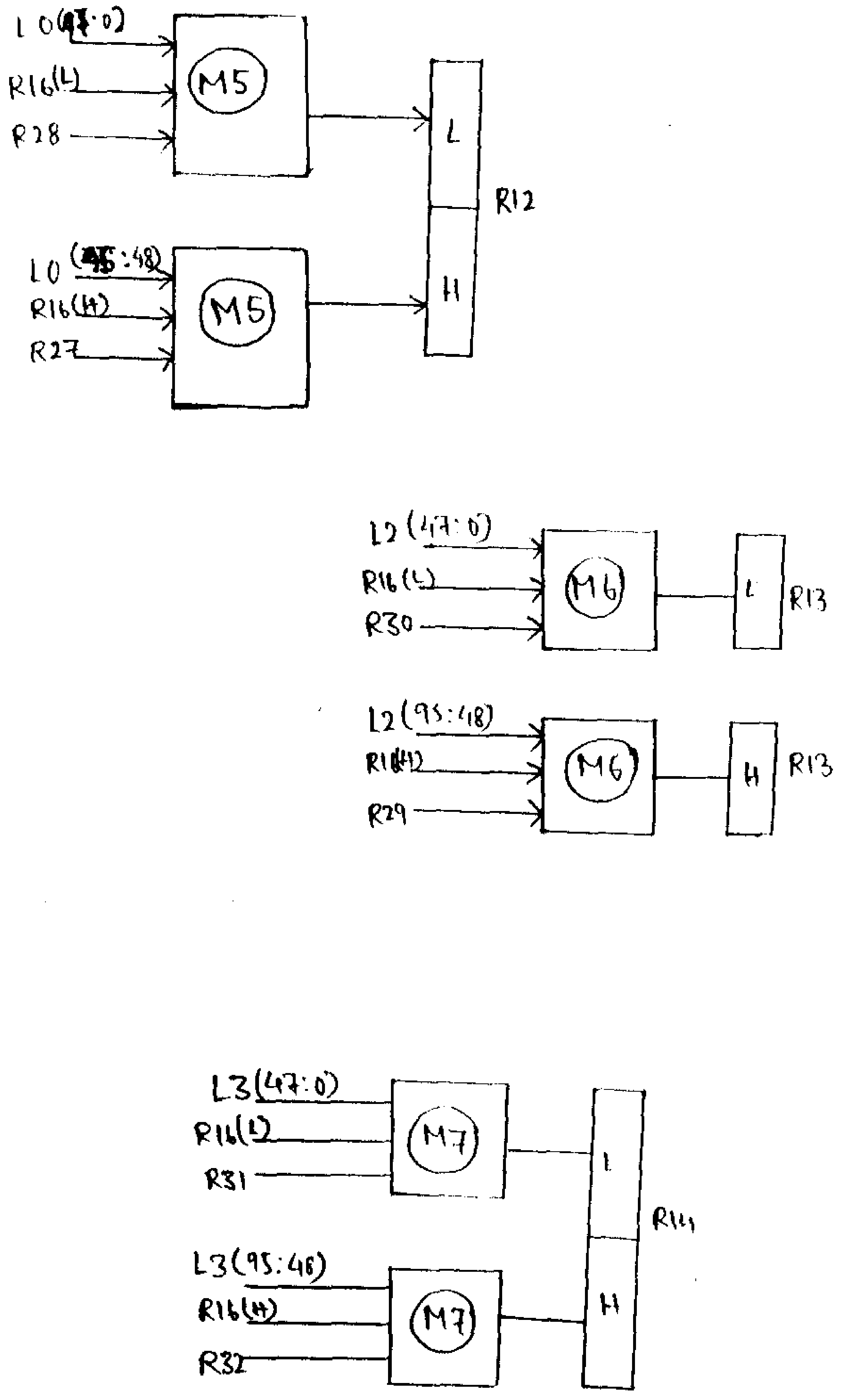
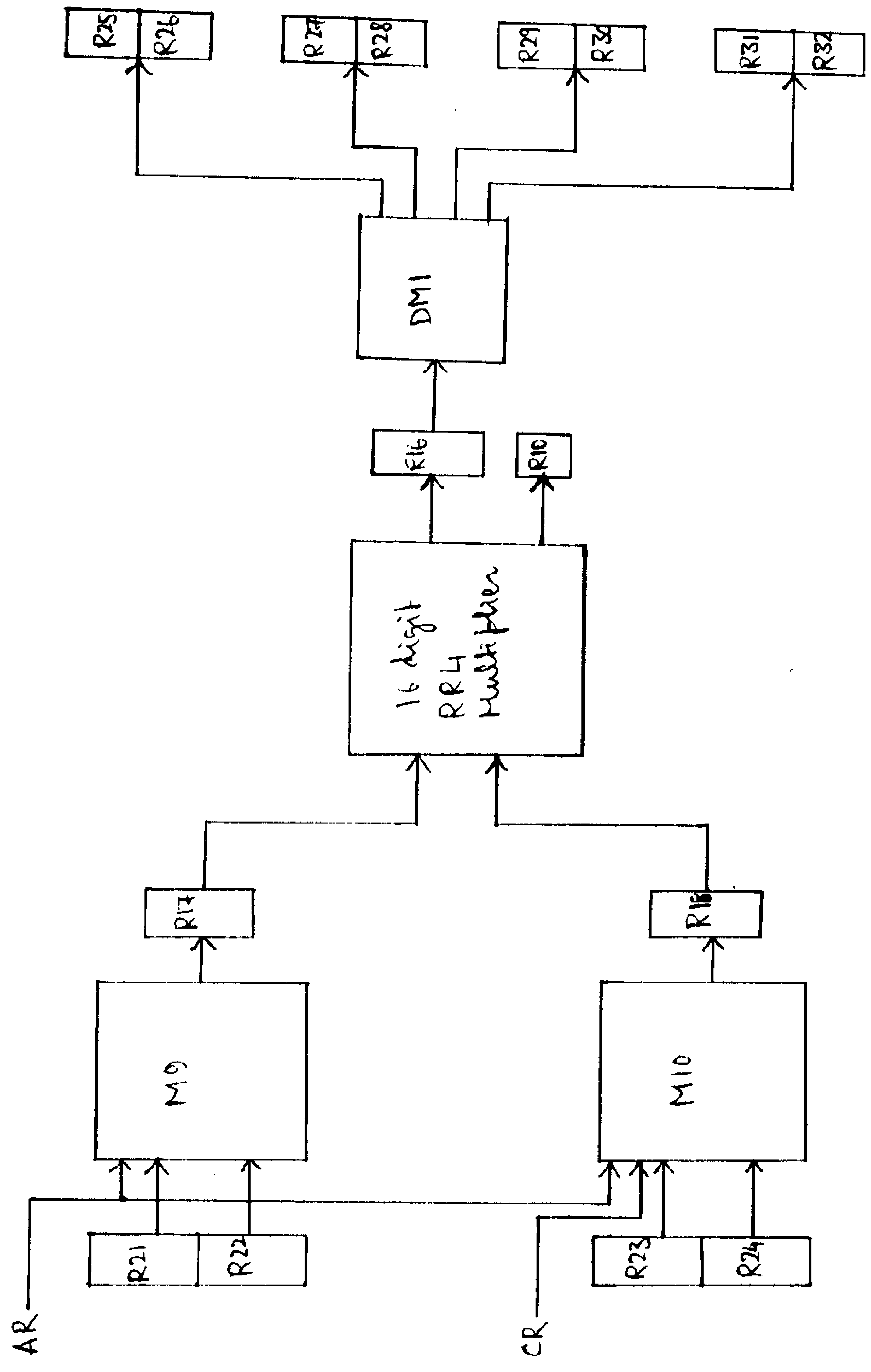

L0(47:0)
R16(L)
R28
M5
L
R12
H
L0(95:48)
R16(H)
R27
M5
L2(47:0)
R16(L)
R30
M6
L
R13
L2(95:48)
R1(H)
R29
M6
H
R13
L3(47:0)
R16(L)
R31
M7
L
R14
L3(95:48)
R16(H)
R32
M7
H

Fig 6(d)



AR
R21
R22
M9
R17
CR
R23
R24
M10
R18
16 digit RR4 Multiplier
R16
R10
DM1
R25
R26
R27
R28
R29
R30
R31
R32

Fig 7(b)

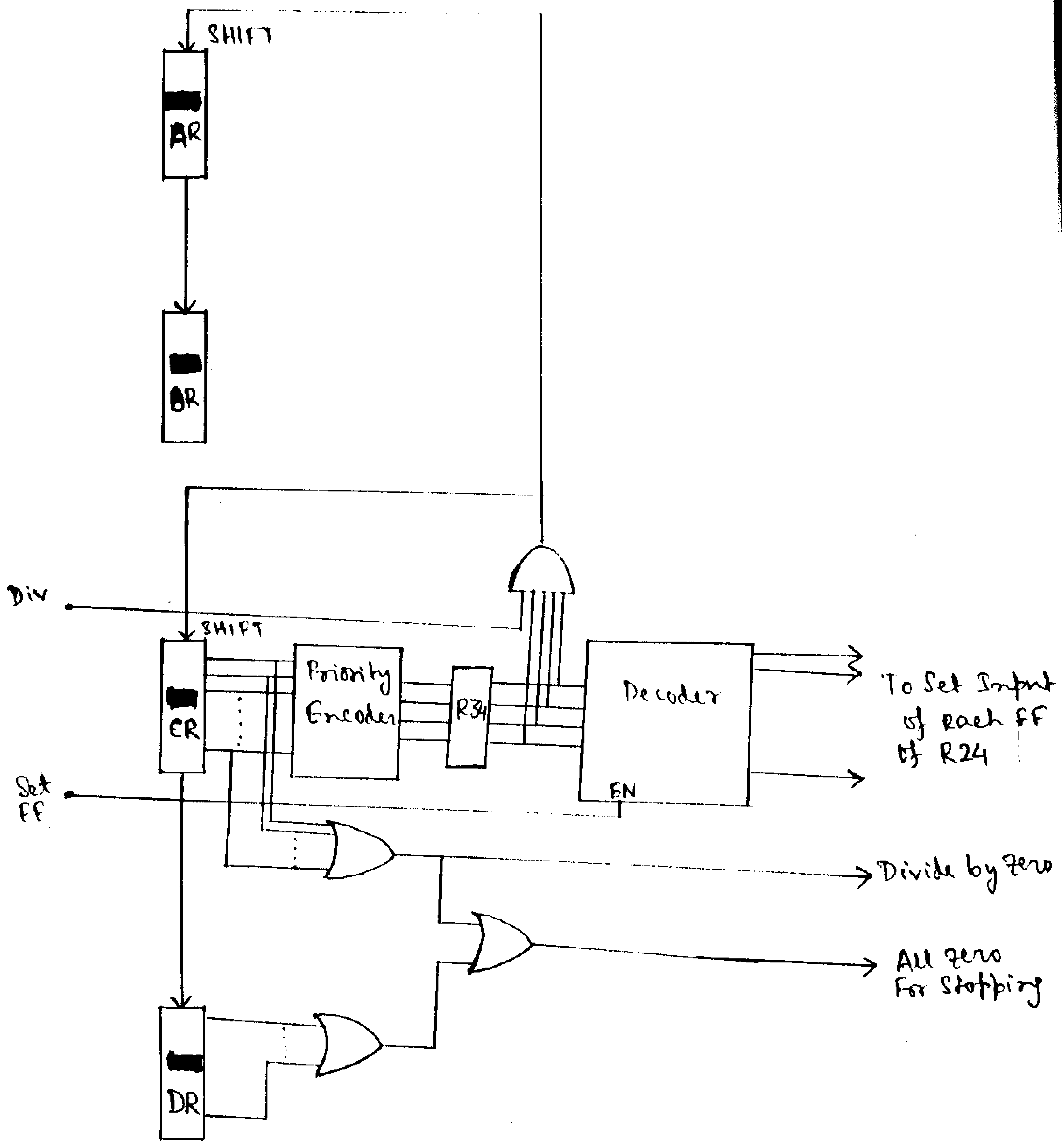

SHIFT
[redacted]
AR
[redacted]
BR
Div
SHIFT
[redacted]
CR
Priority Encoder
R34
Decoder
EN
Set FF
To Set Input of each FF of R24
Divide by zero
All zero For Stopping
[redacted]
DR

Fig. 8.

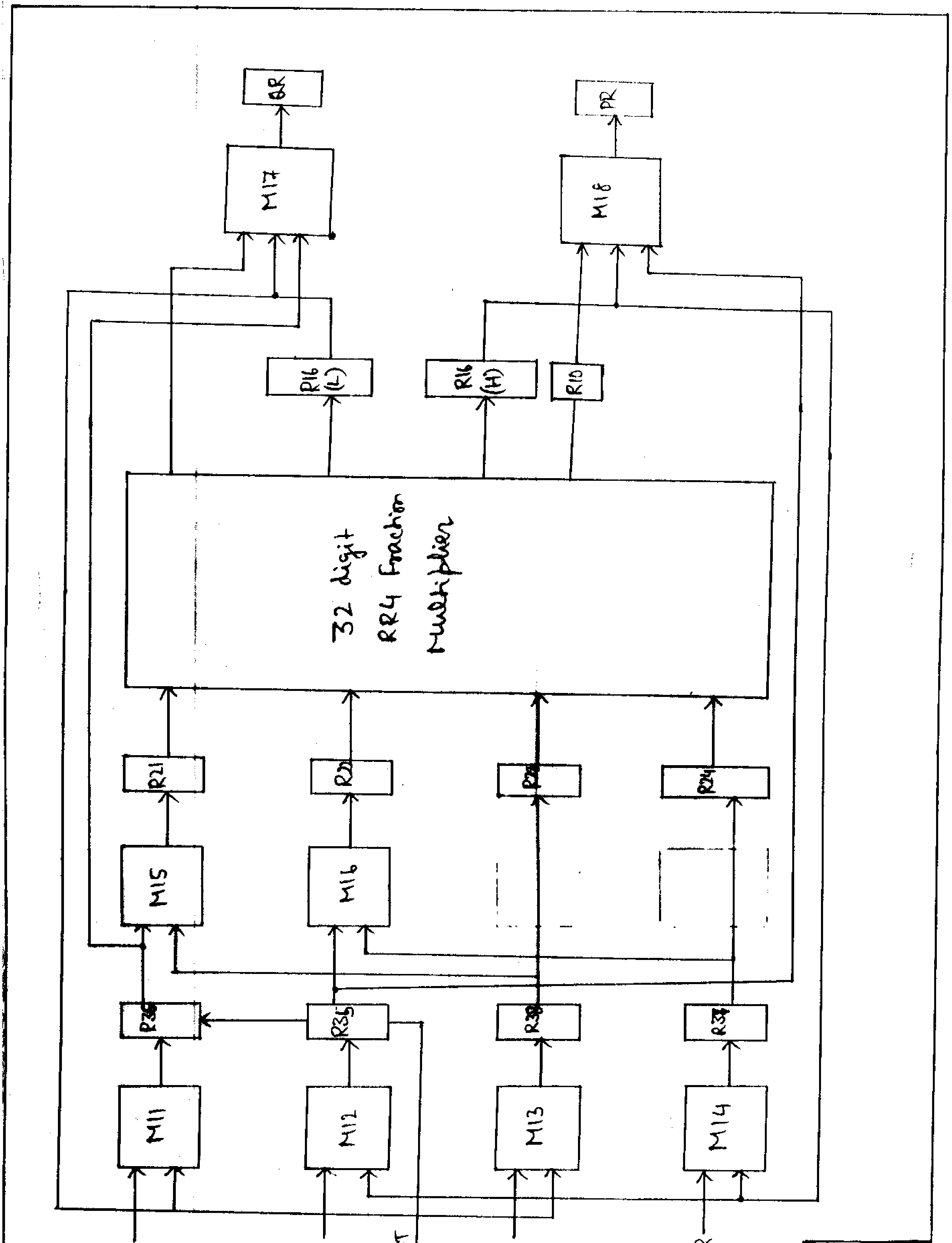

QR
M17
PR
M18
R16 (L)
R16 (H)
R10
32 digit RR4 Fraction Multiplier
R21
R22
R24
M15
M16
R35
M11
M12
M13
M14
BR
AR
SHIFT
DR
CR

Fig. 9