TEMPORARY COVER SHEET

DOCUMENT NUMBER:	00000105
DOCUMENT TITLE:	The Illiac IV Processing Element VOL II
AUTHOR:	Theofanis Economidis
DATE ISSUED:	April 1974

INSTITUTE FOR ADVANCED COMPUTATION

IAC DOC NO. PO-I1100-VOL II-A

THE ILLIAC IV PROCESSING ELEMENT

VOLUME II

THEOFANIS ECONOMIDIS

OCTOBER 1973

REVISED: FEBRUARY, 1974

VOLUME II

TABLE OF CONTENTS

SECTION C: THEORY OF OPERATION 110

I. INTRODUCTION 110

A. Addition 110
B. Subtraction 179
C. Multiplication 180
D. Division 228

ACKNOWLEDGMENT 277

BIBLIOGRAPHY 278

VOLUME II

LIST OF FIGURES

FIGURE PAGE

32 Exponent Part of B Register in 64-Bit Mode for Exponent Correction 116
33 Exponent Part of B Register in 32-Bit Mode for Exponent Correction 117
34 Diagrammatic Representation of Rounding Procedure (Case #4) 132
35 Flow Chart of Actions at Time T1 139
36 Flow Chart of Actions at Time T2 141
37 Flow Chart of Actions at Time T3 143
38 Flow Chart of Actions at Time T4 145
39 Flow Chart of Actions at Time T5 147
40 Flow Chart of Actions at Time T6 (PART A) 149
41 Flow Chart of Actions at Time T6 (PART B) 150
42 Flow Chart of Actions at Time T7 152
43 Flow Chart of Actions at Time T1 154
44 Diagrammatic Representation of Shifting Operation at Time T2 155
45 Flow Chart of Actions at Time T2 156
46 Flow Chart of Actions at Time T3 (PART A) 159
47 Flow Chart of Actions at Time T3 (PART B) 160
48 Flow Chart of Actions at Time T4 162
49 Flow Chart of Actions at Time T5 (PART A) 165
50 Flow Chart of Actions at Time T5 (PART B) 166
51 Flow Chart of Actions at Time T6 168
52 Flow Chart of Actions at Time T7 (PART A) 171
53 Flow Chart of Actions at Time T7 (PART B) 172
54 Flow Chart of Actions at Time T8 174
55 Flow Chart of Actions at Time T9 177
56 Block Functional Diagram of the Registers Participating in Mantissa Manipulation 193
57 Multiplication Process (Mantissa) 195
58 Correction Bits for Mantissa Multiplication 197
59 Clock Time T1 209
60 Clock Time T2 211
61 Clock Time T3 213
62 Clock Time T4 215
63 Clock Time T5 217
64 Clock Time T6 219
65 Clock Time T7 221
66 Clock Time T8 224
67 Input Gating of the CPA, Bit Slice Diagram 225
68 Clock Time T9 227
69 Chart Showing the Area in which the Quotient is Valid 235

VOLUME II

LIST OF TABLES

TABLE		PAGE
33	Shifting in Normalization and Exponent Adjustment	120
34	Truth Table of Conditions in Subtraction (4)	179
35	Multiplier Bits to be Recoded	183
36	Recoding Multiplier Scheme	184
37	Definition of Signals Applied to PAT	190
38	Steps of Mantissa Manipulation in 64-Bit Mode Division	239
39	Steps of Exponent Manipulation in 64-Bit Mode Division	248
40	Procedure for Interchanging INNER & OUTER Mantissas of RGB	274
41	Procedure for Interchanging INNER & OUTER Mantissas of RGR	275
42	Procedure for Interchanging INNER & OUTER Remainder (RGB)	276

VOLUME II

LIST OF FIGURES

FIGURE		PAGE
32	Exponent Part of B Register in 64-Bit Mode for Exponent Correction	116
33	Exponent Part of B Register in 32-Bit Mode for Exponent Correction	117
34	Diagrammatic Representation of Rounding Procedure (Case #4)	132
35	Flow Chart of Actions at Time T1	139
36	Flow Chart of Actions at Time T2	141
37	Flow Chart of Actions at Time T3	143
38	Flow Chart of Actions at Time T4	145
39	Flow Chart of Actions at Time T5	147
40	Flow Chart of Actions at Time T6 (PART A)	149
41	Flow Chart of Actions at Time T6 (PART B)	150
42	Flow Chart of Actions at Time T7	152
43	Flow Chart of Actions at Time T1	154
44	Diagrammatic Representation of Shifting Operation at Time T2	155
45	Flow Chart of Actions at Time T2	156
46	Flow Chart of Actions at Time T3 (PART A)	159
47	Flow Chart of Actions at Time T3 (PART B)	160
48	Flow Chart of Actions at Time T4	162
49	Flow Chart of Actions at Time T5 (PART A)	165
50	Flow Chart of Actions at Time T5 (PART B)	166
51	Flow Chart of Actions at Time T6	168
52	Flow Chart of Actions at Time T7 (PART A)	171
53	Flow Chart of Actions at Time T7 (PART B)	172
54	Flow Chart of Actions at Time T8	174
55	Flow Chart of Actions at Time T9	177
56	Block Functional Diagram of the Registers Participating in Mantissa Manipulation	193
57	Multiplication Process (Mantissa)	195
58	Correction Bits for Mantissa Multiplication	197
59	Clock Time T1	209
60	Clock Time T2	211
61	Clock Time T3	213
62	Clock Time T4	215
63	Clock Time T5	217
64	Clock Time T6	219
65	Clock Time T7	221
66	Clock Time T8	224
67	Input Gating of the CPA, Bit Slice Diagram	225
68	Clock Time T9	227
69	Chart Showing the Area in which the Quotient is Valid	235

SECTION C: THEORY OF OPERATION

I. INTRODUCTION

A. Addition

1. Introduction. In order to present all phases of how addition of two numbers is performed by ILLIAC IV, the process and the details of floating point addition with the options of rounding and normalization will be described, since this is the longest and most complicated form of addition performed by the PE.

In floating point arithmetic, addition rules require that the numbers to be added have equal exponents. If the exponents are different, the difference of the exponents is found and the number (mantissa portion) with the smaller exponent is first shifted, as many places to the right as the difference of the exponents (bits shifted off are lost). This procedure is called alignment.

After the addition of two mantissas is completed, the leading ONE might not be in the most significant bit (MSB) position of the mantissa and the exponent must then be adjusted by subtracting from it the number of places the mantissa was shifted to the left. This procedure is called normalization.

2. Alignment. The Barrel Switch has been designed in such a way that it can shift words in 64-bit mode or words in 32-bit mode. For alignment the shifting is always to the right, which means that the shift counter receives the six least signifcant bits from the Carry Propagating Adder (CPA) as the result of the difference of the exponents of the two operands to be added and applies them to the leading one detector and Barrel Switch controls to be decoded and then applied to the proper levels of the Barrel Switch.

a) Implementation. The two operands, being 64 bits long, are brought into CPA as follows:

The True output of the exponent part of B register is enabled into CPA (bits 65 - 79) while the Complement output of the exponent part of A register is allowed into CPA (bits 65 - 79). The two quantities are added together and the result is sent to LOD #4 and LOD #6 (the six least significant bits go to LOD #4 while the nine most significant bits are sent to LOD #6). The output of LOD #6 goes to LOD #15 to determine whether the exponent difference is greater than 47 (PEXD1-L48L).

The output of LOD #4 (Shift Count Register) is sent to LOD #1, 2, 3, and 5 which control the levels of the Barrel Switch, which, in turn, shifts the mantissa of the operand with the smaller exponent to the right as many places as indicated by the difference of the exponents.

If the exponent of A register is greater than the exponent of B register there is no carry (indicates adder output is complement of ture difference) and therefore the Complement output of LOD #4 is sent to LOD #1, 2, 3, and 5. If the exponent of A register is less than the exponent of B register there is a carry (indicates adder output is true difference) and therefore the True output of LOD #4 is allowed into LOD #1, 2, 3, and 5.

b) Examples.

Example #1

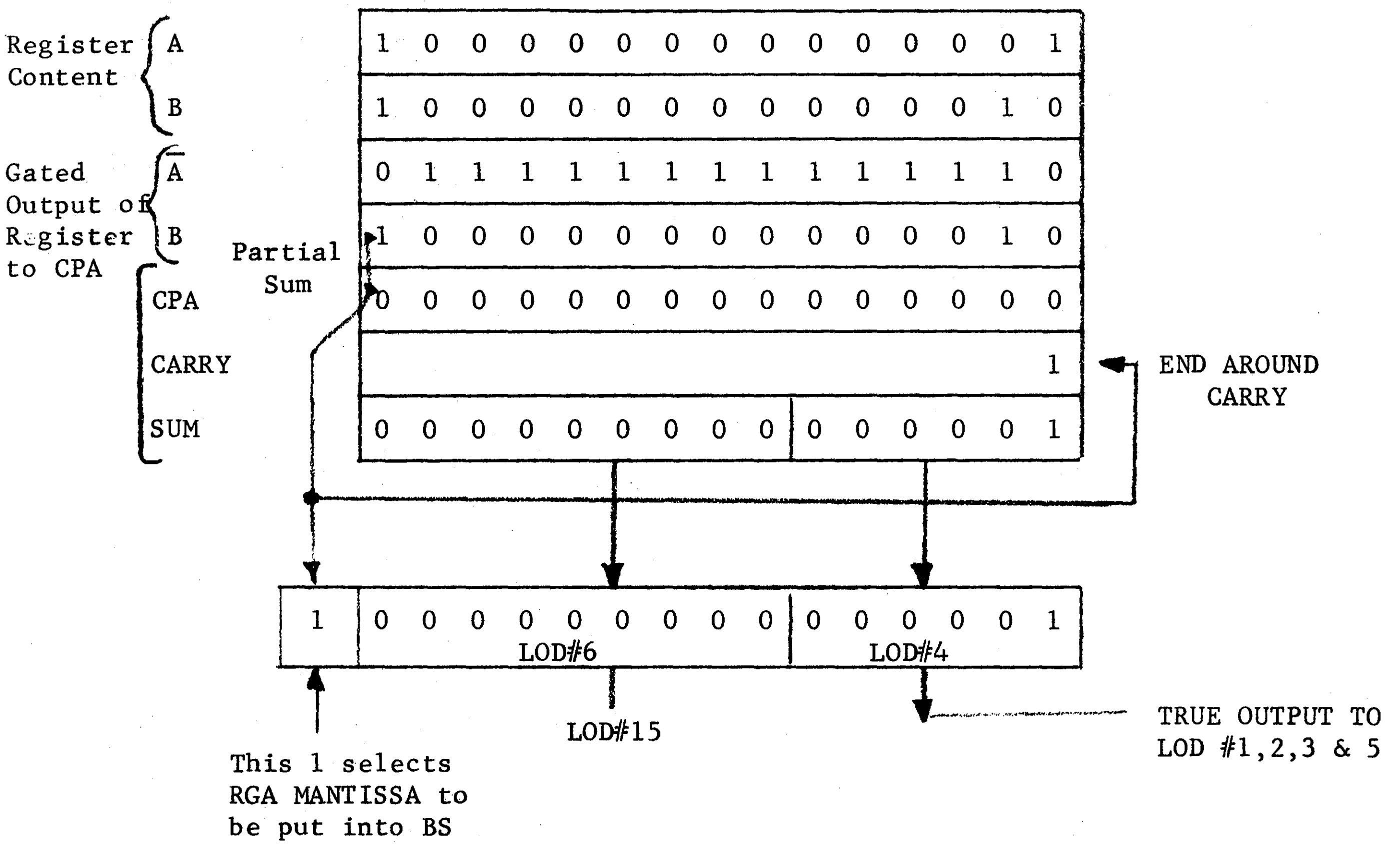
If the exponent of A register, which holds the Augend is Aexp = 100000000000001 and the exponent of B register, which holds the Addend is Bexp = 100000000000010, find the exponent difference.

Solution

$$\text{Aexp} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 = +\ 1)_{10}$$

$$\text{Bexp} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0 = +\ 2)_{10}$$

The difference is ONE and, because Bexp > Aexp, there is a carry and the True output of the Shift Count Register (SCR) (LOD #4) must be ONE.



Register Content	A		1	0	0	0	0	0	0	0	0	0	0	0	0	1
	B		1	0	0	0	0	0	0	0	0	0	0	0	1	0
Gated Output of Register to CPA	$\bar{A}$		0	1	1	1	1	1	1	1	1	1	1	1	1	0
	B		1	0	0	0	0	0	0	0	0	0	0	0	1	0
	CPA		0	0	0	0	0	0	0	0	0	0	0	0	0	0
	CARRY															1
	SUM		0	0	0	0	0	0	0	0	0	0	0	0	0	1
		1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

This 1 selects RGA MANTISSA to be put into BS

Example #2

If Aexp = 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1

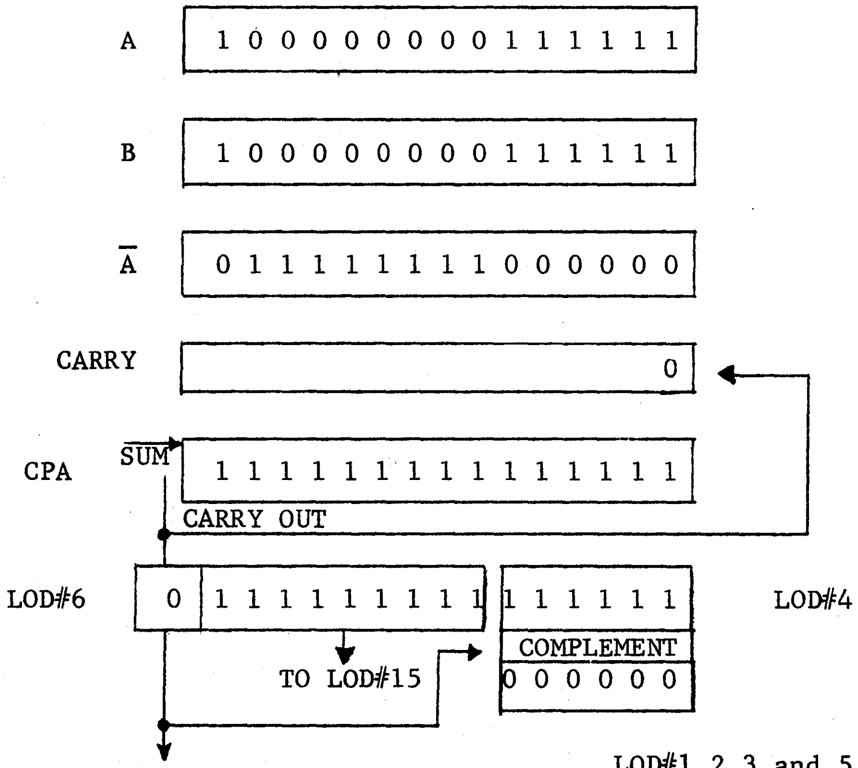
Bexp = 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1

Find the exponent difference.

Solution

Since Aexp = Bexp there is no carry and the Complement output of LOD#4 is enabled into LOD#1, 2, 3 and 5.

Because the exponents are equal, their difference is ZERO.



This "0" selects
RGB mantissa to be
put into BS

Example #3

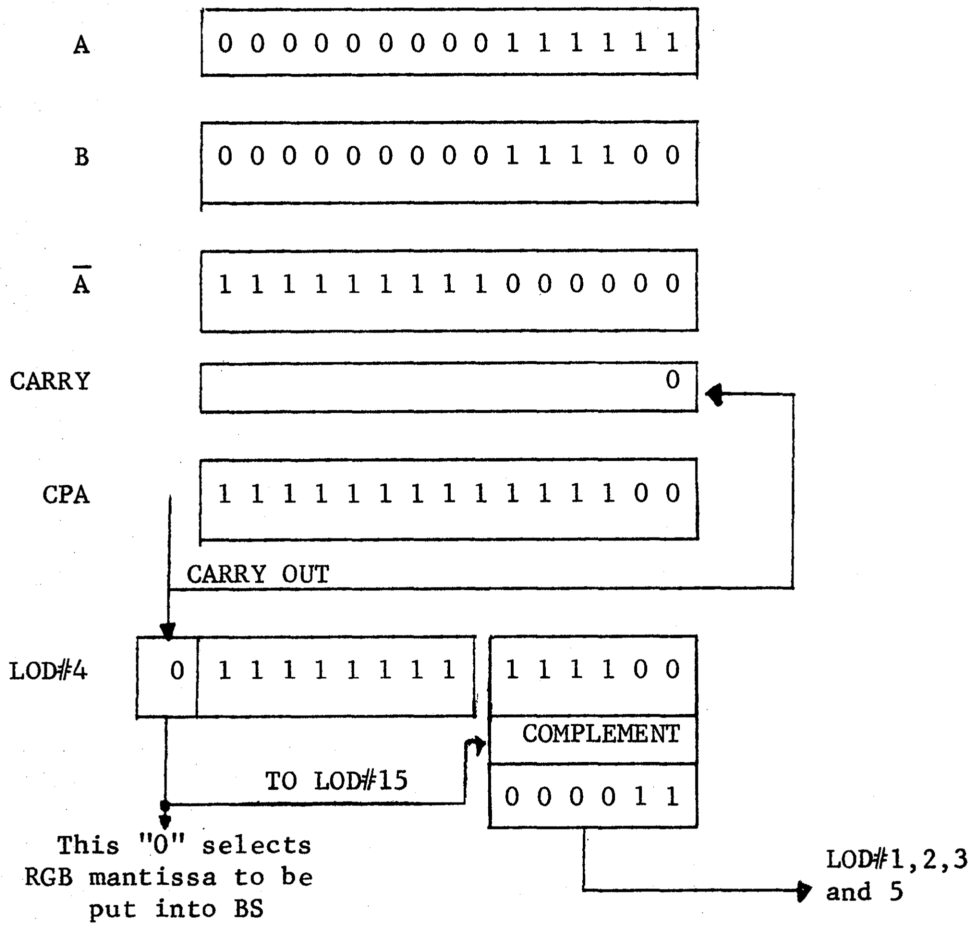
If Aexp = 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1

Bexp = 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0

Find the exponent difference.

Solution

Since Aexp > Bexp by 3 there is no carry and the complement output of LOD#4 is enabled into LOD #1, 2, 3 and 5.



If the exponent difference (EXPDIF) is less than 48_{10} then PEDS1-L48L (see LOD #15) is true and allows the register with the smaller exponent to receive the output of the barrel switch; otherwise the register mantissa is cleared. If rounding is desired, a pseudo-alignment is performed prior to actual alignment, wherein the most significant of the bits to be shifted off is saved in the INNER most significant bit latch (IMSB) for later use. The signal PEXD1-L65H is used during this pseudo-alignment cycle and, when true, allows the INNER most significant bit (IMSB) latch to hold the "shifted off" most significant bit for rounding. Operation of LOD #15 signals PEXD1-L48L and PEXD1-L65H is displayed below.

EXPDIF	PED1-L48L	PEXD1-L65H*	REMARKS
0 → 47	TRUE (allows register with smaller exponent to receive output of BSW)	TRUE (allows shifted off MSB into IMSB latch)	If rounding is preferred
48	FALSE (register with smaller exponent will be cleared)	TRUE (allows shifted off MSB into IMSB latch)	If rounding is preferred
48 - 63	FALSE (same as above)	TRUE (allows ZERO into IMSB since bits 0-15 are not enabled into BS)	If rounding is preferred
64 or greater	FALSE (same as above)	FALSE (input to IMSB is forced to ZERO)	If rounding is preferred

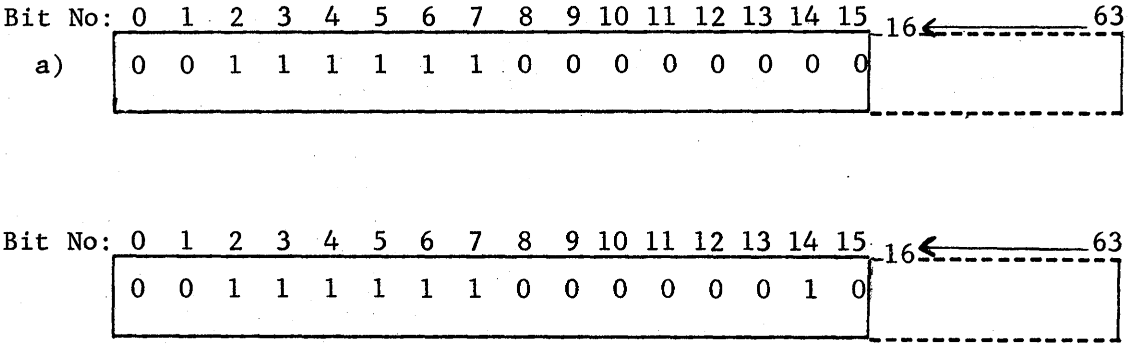
* Signal is misnamed; should be 64 instead of 65.

3. Normalization. For Normalization only, the leading ONE Detectors (LOD #1, 2, 3) are used to detect where the leading ONE is in the mantissa. When one of these LOD's detects a leading ONE it generates the proper controls (shift count) for selection of the proper level of the Barrel Switch for shifting the mantissa to the left as many places as required to place the leading ONE in bit position 16 (for 64-bit mode and Inner Word in 32-bit mode) or bit position 40 (for the Outer Word in 32-bit mode). If the leading one is in bit position 16 (in 64-bit mode or in 32-bit mode/ inner mantissa) it is already normalized and the LOD is not enabled. The BS is then set up for a zero shift. Normalization of the outer mantissa in 32-bit mode is handled differently. In this case the LOD sees the outer

mantissa with 24 leading zeros (inner mantissa is not gated out). The LOD will generate a left shift (of at least 24 which is compensated for by using the first level of the BSW (Byte Swapping Level)). This level moves the outer mantissa 24 bits right.

The correction of the exponent of the final sum as a result of the option of normalization takes place as follows:

The output of LOD #1, 2, 3 being enabled by LOD #5 is placed into bit positions 10 to 15 of the B register in case of 64-bit mode or 32-bit mode Inner word or into bit position 2 to 7 of B register in case of 32-bit mode Outer word.



Bit No:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16← 63
a)	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	

Bit No:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16← 63
	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	0	

Figure 32. Exponent Part of B Register in 64-Bit Mode for Exponent Correction

The mechanization of Exponent Correction, as indicated in Figure 32 (a) and (b), takes place as follows:

A fixed input of 0 0 1 1 1 1 1 1 is placed in bit positions 0 through 7 of the B register. The LOD #5 generates bits 8, 9 as zeros in case of overflow (OV1) or the Leading ONE is at bit position 16 and as ONE's in any other case. The LOD #5 also enables LOD #1, 2, 3 to generate the amount by which the exponent is to be reduced by placing it at bit positions 10 through 15 of the B register as follows:

0 0 0 0 0 0 if leading ONE is at bit position 16
0 0 0 0 0 1 if overflow (OV1) has occurred.

The output of B register is brought into CPA which receives, at the same time, the exponent from A register and adds the two quantities to form the final exponent.

The main criterion for selecting the output of B register to be enabled into CPA is the state of bits 8 and 9 (as a result of an overflow or when bit 16 contains a ONE). If these bits are zero then the complement of bits 1 to 7 of B register is brought into the CPA which means that the output of B register is:

01000000 00000000 if the leading ONE is at bit 16
01000000 00000001 if overflow (OV1) has occurred.

If bits 8 and 9 are ONE's, overflow has not occurred (OV1) and the leading ONE is not at bit 16. Therefore, the mantissa has to be shifted to the left a certain number of places, which implies that a number must be subtracted from the exponent of the final sum. The correction bits from the LOD #1, 2, 3 are placed in bit positions 10 through 15 of B register, but the TRUE output of B register is brought into the CPA. Bit 1 is then a ZERO which means that this exponent is negative. For a better understanding of the exponent manipulation see the discussion on page 19 concerned with the exponent.

In 32-bit mode the mechanization of exponent correction, as indicated in Figure 32 (a) and (b), takes place as follows:

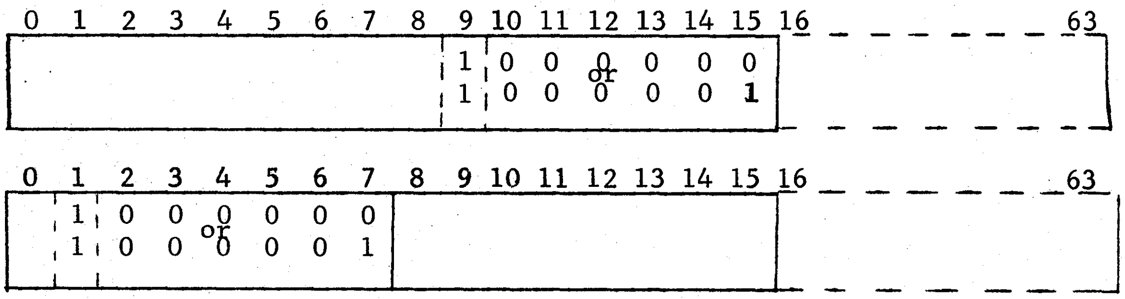


Figure 33. Exponent Part of B Register in 32-Bit Mode for Exponent Correction

LOD #5 generates bit 9 as a ONE in 32-bit mode for the Inner word if OV1 or bit 16 is a ONE and LOD #1, 2, 3 are not enabled. Bits 10-15 of the B register will contain all zeros or zeros and a low order one as follows:

0 0	0 0	0 0	if the leading ONE is at bit position 16
0 0	0 0	0 1	if overflow (OV1) has occurred.

In this case the TRUE output of B register is brought into the CPA since the correction bits comprise a positive exponent.

If bit 1 is ONE, the TRUE output of B register is enabled into CPA; if bit 1 is a zero the TRUE output of B register indicates a negative exponent as explained in the description of the 32-bit mode for Inner word. From the discussion so far it is evident that, for normalization and exponent correction, the shift counter (LOD #4) does not participate at all but instead LOD #1, 2, 3 and 5 control the shifting operation.

At this point a very interesting question arises. What happens if the program calls for normalization and the mantissa of the result is ZERO? It is apparent that in this case the LOD #1, 2, 3 will not detect any leading ONE and therefore the Barrel Switch will not perform any shifting of the mantissa. Also, the correction bits in B register will be all zeros which implies that the exponent should not be expected to be affected at all. Therefore, exponent Underflow will not occur if the above method was used when attempting to normalize a ZERO mantissa.

As discussed previously, when representing a ZERO number in floating point arithmetic the mantissa must be ZERO and the exponent must have the least value that the machine can hold. To accomplish this, when a mantissa is ZERO there is a signal called ZERO MANTISSA LEVEL which becomes true when this condition is detected and inhibits the load clocks to the exponent field of the A register, while at the same time the clear clocks are enabled into the exponent field of A register, thus forcing the exponent field to be filled with ZEROS (see the description of LOD #5).

Since the exponent is represented in excess code, ZEROS in the exponent field means that the number has the smallest possible exponent, which combined with the ZERO mantissa represents a ZERO result. In this way it is assured that the ZERO number is represented by "Clean Zeros".

Table 33 indicates the amount of shifting of the mantissa to the left required to bring the leading ONE to bit 16 for 64-bit mode or 32-bit mode for the Inner word or to bit 40 for the 32-bit mode Outer word. It also indicates the amount the exponent of the final sum has to be reduced when the option of normalization is used.

In the discussion of Alignment and Normalization it was said that in case of an overflow or underflow the sum is properly adjusted in order to represent a correct number. Before proceeding into the rounding procedure, a few explanatory remarks about overflow (OV) and underflow may help the reader not only to understand the mechanization of the entire procedure, but also to appreciate the importance of the logic involved for such an operation.

4. Overflow. If two numbers of the same sign are added, the magnitude of the result might be greater than what the register that is to contain the result can hold. This condition is called overflow.

In floating point arithmetic the difference between mantissa overflow and exponent overflow must be distinguished.

a) Mantissa Overflow can occur -

1) when the mantissas are added and the signs of addend and augend are the same. The overflow is indicated by a carry out of the most significant bit of the sum.

2) when rounding and the signs of addend and augend are equal, the most significant shifted off bit is added to the least significant bit of the sum.

If while adding this shifted off bit, the sum contains all ONES and this bit is a ONE, the result is mantissa overflow.

Table 33. Shifting in Normalization and Exponent Adjustment

Bit Position of Leading One	BARREL SWITCH LEVELS					EXPONENT ADJUSTMENT	
	FIRST LEVEL		SECOND LEVEL	THIRD LEVEL	FOURTH LEVEL	64-BIT MODE	32-BIT MODE
	64-Bit Mode 32 Inner	32-Bit Mode Outer					
OV1	0	-	0	0	1	0100000000000001	01000001
16	0	-	0	0	0	0100000000000000	01000000
17	0	-	48	12	3	0011111111111111	00111111
18	0	-	48	12	2	0011111111111110	00111110
19	0	-	48	12	1	0011111111111101	00111101
20	0	-	48	12	0	0011111111111100	00111100
21	0	-	48	8	3	0011111111111011	00111011
22	0	-	48	8	2	0011111111111010	00111010
23	0	-	48	8	1	0011111111111001	00111001
24	0	-	48	8	0	0011111111111000	00111000
25	0	-	48	4	3	0011111111110111	00110111
26	0	-	48	4	2	0011111111110110	00110110
27	0	-	48	4	1	0011111111110101	00110101
28	0	-	48	4	0	0011111111110100	00110100
29	0	-	48	0	3	0011111111110011	00110011
30	0	-	48	0	2	0011111111110010	00110010
31	0	-	48	0	1	0011111111110001	00110001
32	0	-	48	0	0	0011111111110000	00110000
33	0	-	32	12	3	0011111111101111	00101111
34	0	-	32	12	2	0011111111101110	00101110
35	0	-	32	12	1	0011111111101101	00101101
36	0	-	32	12	0	0011111111101100	00101100
37	0	-	32	8	3	0011111111101011	00101011
38	0	-	32	8	2	0011111111101010	00101010
39	0	-	32	8	1	0011111111101001	00101001

Table 33. (Continued) Shifting in Normalization and Exponent Adjustment

Bit Position of Leading One	BARREL SWITCH LEVELS					EXPONENT ADJUSTMENT	
	FIRST LEVEL						
	64-Bit Mode 32 Inner	32-Bit Mode Outer	SECOND LEVEL	THIRD LEVEL	FOURTH LEVEL	64-BIT MODE	32-BIT MODE
OV2	-	0	0	0	1	- - - - - - - - - - - - - - - -	01000001
40	0	24 Right	32	8	0	0011111111101000	01000000
41	0	24	32	4	3	0011111111100111	00111111
42	0	24	32	4	2	0011111111100110	00111110
43	0	24	32	4	1	0011111111100101	00111101
44	0	24	32	4	0	0011111111100100	00111100
45	0	24	32	0	3	0011111111100011	00111011
46	0	24	32	0	2	0011111111100010	00111010
47	0	24	32	0	1	0011111111100001	00111001
48	0	24	32	0	0	0011111111100000	00111000
49	0	24	16	12	3	0011111111011111	00110111
50	0	24	16	12	2	0011111111011110	00110110
51	0	24	16	12	1	0011111111011101	00110101
52	0	24	16	12	0	0011111111011100	00110100
53	0	24	16	8	3	0011111111011011	00110011
54	0	24	16	8	2	0011111111011010	00110010
55	0	24	16	8	1	0011111111011001	00110001
56	0	24	16	8	0	0011111111011000	00110000
57	0	24	16	4	3	0011111111000111	00101111
58	0	24	16	4	2	0011111111000110	00101110
59	0	24	16	4	1	0011111111000101	00101101
60	0	24	16	4	0	0011111111000100	00101100
61	0	24	16	0	3	0011111111000011	00101011
62	0	24	16	0	2	0011111111000010	00101010
63	0	24 ↓	16	0	1	0011111111000001	00101001

In the first case the overflow is cleared and loaded; in the second case the latch is simply loaded.

b) Exponent Overflow can occur -

1) when adding exponents in multiplication or subtracting exponents in division.

2) when mantissa overflow in floating point occurs and the exponent happens to contain all ONES.

5. Exponent Underflow. If the value of the exponent is reduced beyond the minimum value that the proper register can hold, it is said that an "underflow" has occurred. In Addition exponent underflow will occur only when normalization is performed. In this case, the number equal to the amount of places the mantissa is shifted to the left until the leading ONE is at bit position 16 for 64- or 32-bit mode for Inner word or bit 40 for 32-bit mode for the Outer word is effectively subtracted from the exponent.

There are two cases in which a fault may occur, in which, therefore, the F, F1 bits of mode register are set because of exponent underflow.

In one case the following conditions would exist:

a) The operand or result is any number other than zero.
b) Normalization takes place.
c) Exponent underflow occurs.
d) Set F bit in case of exponent underflow.

In the other case the following conditions would exist:

a) Exponent arithmetic only (operations involving the exponents only).
b) Exponent underflow occurs.
c) Set F bit in case of exponent underflow.

In both cases the resulting word is zero.

6. Rounding. In the description of the word formats in the 64-bit mode (subsection III B b) of Section A, a number in floating point arithmetic is described by

$$X = (-1)^{X_o} 2^T [\sum_{i=1}^{48} 2^{-i} X_i].$$

In operation of addition there are two operands involved and they may be described by

$$X = (-1)^{X_o} 2^s [\sum_{i=1}^{48} 2^{-i} X_i]$$

(64-bit mode)

$$Y = (-1)^{Y_o} 2^t [\sum_{j=1}^{48} 2^{-j} Y_j]$$

where X = Augend

Y = Addend

X_o = Sign of mantissa of Augend

Y_o = Sign of mantissa of Addend

2^s = Exponent value of Augend

2^t = Exponent value of Addend

$\sum_{i=1}^{48} 2^{-i} X_i$ = Mantissa part of Augend

$\sum_{j=1}^{48} 2^{-j} Y_j$ = Mantissa part of Addend

Because the mantissa part of the word in ILLIAC IV occupies bit positions 16 to 63 the above notations would be absolutely consistent with the word formats only if the subscripts i and j were considered as varying from 16 to 63 instead of 1 to 48. But since in this section the changes that the mantissas undergo when the option of rounding is used are being discussed, and because the mantissa field has 48 bits, the above notation setting the limits of the mantissa field between 1 and 48 is felt to be acceptable if only because of its convenience.

Rounding is an option that may be selected by the programmer. If the exponents of the two operands differ, the mantissa of the operand with the smaller exponent is shifted off the number of places indicated by the amount of the exponent difference. If the exponents are equal there is no need to shift any of the mantissas.

There are many factors that affect rounding operations; some of these factors are:

a) The value (1 or 0) of the saved bit.

b) Whether the arithmetic operation that requires rounding involves addition or subtraction.

c) If addition, whether or not there is overflow.

d) If subtraction, which operand is larger and which operand the saved bit came from.

e) If subtraction, whether or not the mantissa difference is zero.

Since the exponent difference constitutes the starting point of the rounding process, the following cases are examined.

a) $s > t$ and $X_o = Y_o$

b) $s > t$ and $X_o \neq Y_o$

c) $s < t$ and $X_o = Y_o$

d) $s < t$ and $X_o \neq Y_o$

i. Case where $s > t$ and $X_o = Y_o$:

The number with the smaller exponent is the Y operand and must be shifted right s-t places. It must have as exponent the exponent of X. Therefore,

$$Y \text{ shifted} = (-1)^{Y_o} 2^s \left[\sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} + \sum_{j=49}^{48+(s-t)} 2^{-j} Y_{j-(s-t)} \right] \qquad (1)$$

The mantissas of the two operands are then added as follows:

$$X + Y \text{ shifted} = (-1)^{X_o} 2^s \left[\sum_{i=1}^{48} 2^{-i} X_i + \sum_{j=(s-t)+1}^{48} {}^{-j} Y_{j-(s-t)} + \sum_{j=49}^{48+(s-t)} 2^{-j} Y_{j-(s-t)} \right] \qquad (2)$$

In reality since the adder is not extended to take care of the shifted off bits given by

$$\sum_{j=49}^{48+(s-t)} 2^{-j} Y_{j-(s-t)}$$

the most significant shifted off bit is added to the least significant bit of the adder and therefore

$$(X + Y \text{ shifted})_{\text{rounded}} = (-1)^{X_o} 2^s \left[\sum_{i=1}^{48} 2^{-i} X_i + \sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} + 2^{-48} Y_{49-(s-t)} \right] \qquad (3)$$

If the programmer wishes to truncate instead of rounding, then the result will be

$$(X + Y \text{ shifted})_{\text{truncated}} = (-1)^{X_o} 2^s \left[\sum_{i=1}^{48} 2^{-i} X_i + \sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} \right] \quad (4)$$

The procedure, therefore, for rounding in case #1 is as follows:

a. Determine which operand has the smaller exponent.

b. Determine the difference of the exponents of the two operands to be added.

c. Shift off the mantissa of the operand with the smaller exponent as many places as the difference of the two exponents (1).

d. Perform the summation of the mantissas of the two operands (2).

e. Check for overflow.

If there is no overflow, then add the most significant shifted off bit to the least significant bit of the sum (4).

NOTE: In order to add the 2^{-49} bit all zeros are forced into B register (Y operand).

ii. <u>Case where $s > t$ and $X_o \neq Y_o$:</u>

The operands are represented as in case #1. Because X is the largest operand, Y must be shifted s-t places end off. Therefore

$$\text{Y shifted} = (-1)^{Y_o} 2^s \Big[\sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} + \sum_{j=49}^{48+(s-t)} 2^{-j} Y_{j-(s-t)} \Big] \tag{5}$$

Since the signs are different the addend (Y_{sh}) is 1's complemented

$$\overline{\text{Y}} \text{ shifted} = (-1)^{Y_o} 2^s \Big\{ 1- \Big[\sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} + \sum_{j=49}^{48+(s-t)} 2^{-j} Y_{j-(s-t)} \Big] \Big\} \tag{6}$$

but only the most significant bit shifted off is saved to be subtracted from the sum later. The addend then becomes

$$\overline{\text{Y}} \text{ shifted} = (-1)^{Y_o} 2^s \Big\{ 1- \Big[\sum_{j=(i-t)+1}^{48} 2^{-j} Y_{j-(s-t)} + 2^{-48} Y_{49-(s-t)} \Big] \Big\} \tag{7}$$

Addition of the mantissas of X and Y_{sh} (complement) gives

$$(\text{X} + \overline{\text{Y}} \text{ shifted}) = (-1)^{X_o} 2^s \Big\{ \sum_{i=1}^{48} 2^{-i} X_i + 1 - \sum_{j=(s-t)+1}^{48} 2^{-j} Y_{j-(s-t)} - 2^{-49} Y_{49-(s-t)} + 2^{-48} \Big\} \tag{8}$$

The term 2^{-48} is the end around carry which exists only when the augend (X) is greater than the addend (Y) and then the sign of the sum is the sign of the augend.

The procedure for rounding in case #2 is as follows:

a. Determine which operand has the smaller exponent.

b. Determine the difference of the exponents of the two operands.

c. Save only the most significant shifted off bit.

d. Shift off the mantissa of the operand with the smaller exponent as many places as the difference of the exponents (5).

e. Take the 1's complement of the addend.

f. Perform the addition of the two mantissas as follows:

   1. Since $s > t$, the augend is greater than the addend and a carry (for normalized operands only) is to be expected.

   2. The sum is not complemented and it has the sign of augend.

   3. The most significant shifted off bit is subtracted from the least significant bit of the sum by adding all 1's to the sum--see equation (8).

Example

Given: $X = (-1)^0 \times 2^3 \times 1\ 1\ 1\ 0$ (Augend)

$Y = (-1)^1 \times 2^1 \times 1\ 1\ 1\ 0$ (Addend)

Show the mechanization of rounding procedure (case #2).

Solution

a. The addend has the smaller exponent.

b. The difference of the exponents is two ($s-t = 3-1 = 2$).

c. The addend after it has been shifted two places end off to the right looks like

$$Y = (-1)^1 \times 2^3 \times 0\ 0\ 1\ 1 \;\Big|\; 1\ 0$$

← end of register

← most significant shifted off bit

d. Take the 1's complement of the mantissa of Y

$$Y = (-1)^1 \times 2^3 \times 1\ 1\ 0\ 0 \;\Big|\; 1\ 0$$

← remains unchanged

e. The most significant shifted bit which is ONE is saved in the latch.

f. The two operands are added as follows:

$$\begin{array}{rlrl}
X = & (-1)^0 \times 2^3 \times & 1\ 1\ 1\ 0 & \\
Y = & (-1)^1 \times 2^3 \times & \underline{1\ 1\ 0\ 0} & \\
\text{SUM} = & (-1)^0 \times 2^3 & 1\ 0\ 1\ 0 & \\
 & & \xrightarrow{} \underline{1} & \\
 & & 1\ 0\ 1\ 1 & \\
 & & \underline{1\ 1\ 1\ 1} & \leftarrow \text{Add all ONES in order to subtract the most significant shifted off bit from the sum} \\
 & & 1\ 0\ 1\ 1 & \\
 & 1 & & \leftarrow \text{This ONE is ignored (Carry)}
\end{array}$$

The resulting carry that is produced when adding all ONES to the sum is ignored and therefore the final rounded sum is:

$$\text{SUM} = (-1)^0 \times 2^3 \times 1\ 0\ 1\ 0$$

NOTE: In order to subtract the 2^{-49} bit, all 1's are forced into the mantissa part of B input of the CPA (Y operand) and the contents of the A register are added to it. If there is any carry it is ignored.

To check the results:

The X mantissa = 14_{10}

The Y shifted mantissa = -3.5_{10}

Then $14 + (-3.5) = 10.5_{10}$

After rounding, and therefore subtracting .5 from the sum we get $10_{10} = (1\ 0\ 1\ 0)_2$ which is the same as the rounded mantissa part of the final sum above.

iii. Case where $s < t$ and $X_o = Y_o$

The X operand has the smaller exponent and must be shifted end off t-s places to the right. This gives the X operand the same exponent as the Y operand.

$$\text{X shifted} = (-1)^{X_o} 2^t \, [\sum_{i=(t-s)+1}^{48} 2^{-i} X_{i-(t-s)}$$

$$+ \; 2^{-49} X_{49-(t-s)} \;] \qquad (9)$$

The term $2 \times 2^{-49} \; X_{49-(t-s)}$ is the most significant bit shifted off and is saved to be added into the least significant bit of the sum. The mantissas of the two operands are then added.

$$(X \text{ shifted} + Y) = (-1)^{Y_o} 2^t \left[\sum_{i=(t-s)+1}^{48} 2^{-i} X_{i-(t-s)} + \sum_{j=1}^{48} 2^{-j} Y_j \right] + 2^{-48} X_{49-(t-s)} \qquad (10)$$

If the most significant bit $[2^{-49} X_{49-(t-s)}]$ shifted off and which has been saved for rounding is Zero, the sum is:

$$(X \text{ shifted} + Y) = (-1)^{Y_o} 2^t \left[\sum_{i=(t-s)+1}^{48} 2^{-i} X_{i-(t-s)} + \sum_{j=1}^{48} 2^{-j} Y_j \right] \qquad (11)$$

iv. Case where $s < t$ and $X_o \neq Y_o$ (see Figure 30)

Since the addend (Y) is the largest operand for normalized operands:

a. There is no overflow because the signs are different.

b. There is an end around carry.

c. The sum must be complemented.

d. The sign of the sum is the sign of Y.

e. The sum has as exponent the exponent of Y.

The mantissa of X must be shifted by t-s places to the right end off.

$$X \text{ shifted} = (-1)^{X_o} 2^t \left[\sum_{i=(t-s)+1}^{48} 2^{-i} X_{i-(t-s)} + \sum_{i=49}^{48} 2^{-i} X_{i-(t-s)} \right] \qquad (12)$$

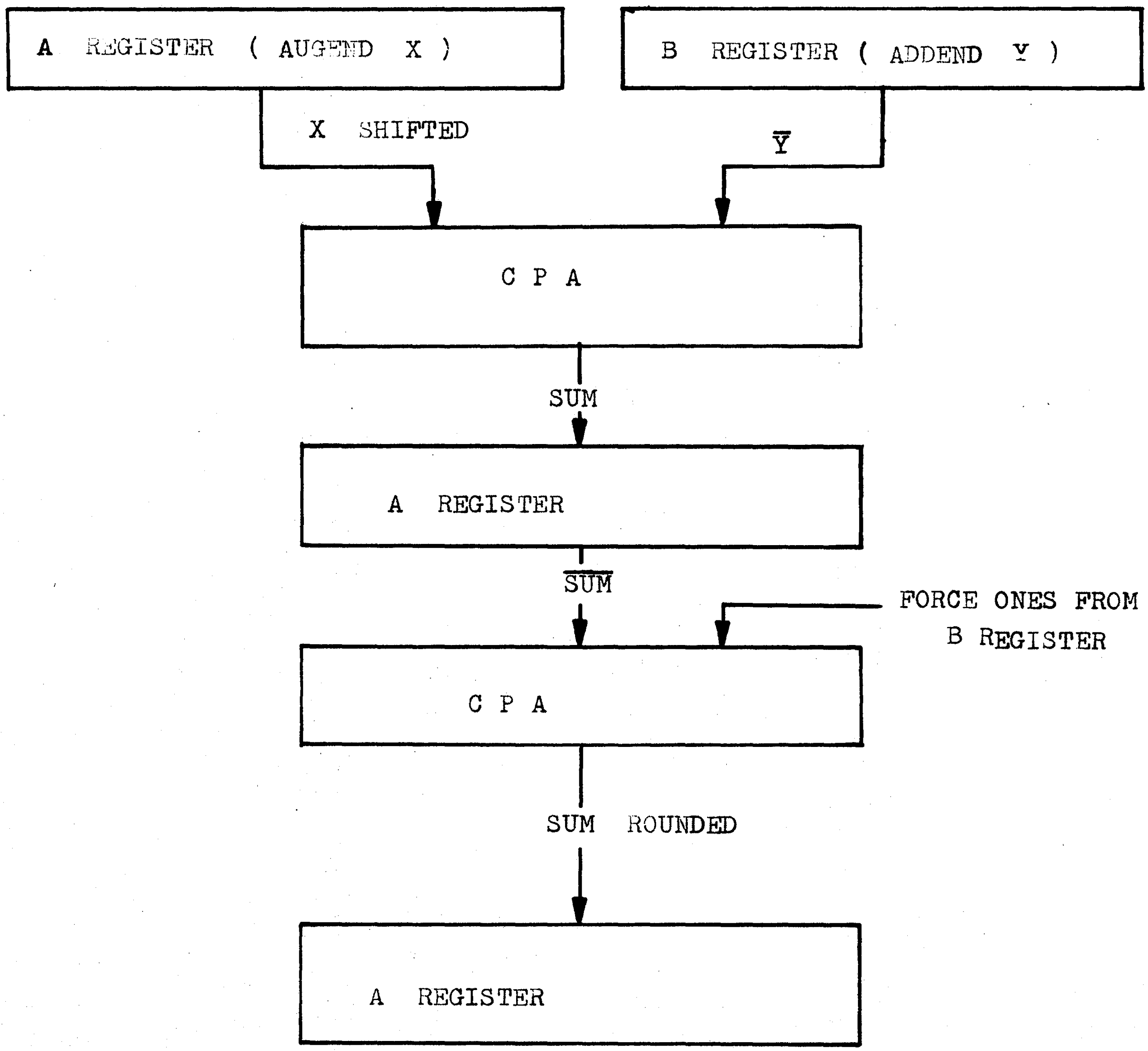


FIGURE 34. DIAGRAMMATIC REPRESENTATION OF ROUNDING PROCEDURE (CASE # 4)

Because the signs are different the 1's complement of the Addend (Y) is needed:

$$\overline{Y} = (-1)^{Y_o} 2^t [1-\sum_{j=1}^{48} 2^{-j} Y_j] \tag{13}$$

Addition of the two operands gives

$$(X \text{ shifted} + \overline{Y}) = (-1)^{Y_o} 2^t \left[\sum_{i=(t-s)+1}^{48} 2^{-1} X_{i-(t-s)} \right.$$

$$\left. + \quad (1- \sum_{j=1}^{48} 2^{-j} Y_j) \right] \tag{14}$$

where the term

$$\sum_{i=49}^{48+(t-s)} 2^{-i} X_{i-(t-s)}$$

is ignored except that, for rounding only, the most significant shifted off bit is saved to be subtracted from the least significant bit of the sum. This bit is represented by 2 x $2^{-49} Y_{49-(t-s)}$.

The sum is now complemented and the most significant shifted off bit is subtracted from the least significant bit of the sum.

$$(X \text{ shifted} + \overline{Y}) = (-1)^{Y_o} 2^t \left\{ 1- \left[\sum_{i=(t-s)+1}^{48} 2^{-i} X_{i-(t-s)} \right. \right.$$

$$\left. \left. + (1- \sum_{j=1}^{48} 2^{-j} Y_j) \right] - 2^{-48} Y_{49-(t-s)} \right\} \tag{15}$$

Example

Suppose the two operands to be added are

X = $(-1)^0$ x 2^1 x 1 1 1 0

Y = $(-1)^1$ x 2^3 x 1 1 1 1

Show the rounding procedure.

Solution

a. The augend has the smaller exponent.

b. The difference of the exponents is TWO (t-s = 3-1 = 2).

c. The augend is shifted end off to the right by TWO.

$$\text{X shifted} = (-1)^0 \text{ x } 2^3 \text{ x } 0\ 0\ 1\ 1 \ \Big|\ 1\ 0$$

Most significant shifted off bit

End of register

d. The mantissa of the addend is 1's complemented because the signs are different.

Y (comp.) = $(-1)^1$ x 2^3 x 0 0 0 0

e. Form the sum

Sum = X + Y comp. = $(-1)^1$ x 2^3 x 0 0 1 1

f. Complement the sum

Sum (comp.) = $(-1)^1$ x 2^3 x 1 1 0 0

g. Insert all ONES to subtract the most significant shifted off bit from the mantissa part of the Sum (comp.).

$$\begin{array}{rl} \text{Sum (comp.)} = & (-1)^1 \text{ x } 2^3 \text{ x } 1\ 1\ 0\ 0 \\ & \phantom{(-1)^1 \text{ x } 2^3 \text{ x }} \underline{1\ 1\ 1\ 1} \\ \text{Final rounded sum} = & (-1)^1 \text{ x } 2^3 \text{ x } 1\ 0\ 1\ 1 \end{array}$$

1

This ONE is ignored (carry)

7. Summary of the Procedure for Addition of Two Operands in Floating Point Arithmetic.

a) Find the magnitude of the difference of the two exponents.

b) Shift the mantissa of the operand with the smaller exponent to the right end off as many places as the exponent difference (alignment).

c) Save the most significant shifted off bit of the operand with the smaller exponent (if rounding is used).

d) The exponent of the operand with the larger exponent becomes common for both operands and therefore remains the same for the final sum unless normalization takes place later.

e) Add the mantissas of both operands.

1) Mantissa of augend is added to the mantissa of addend (both in true form) if the signs of the two operands are the same.

2) True form of mantissa of augend is added to the complement of the mantissa of the addend, if the signs are different.

f) The resultant sum is taken in true form if the signs of the two operands are different and there is a carry out of the most significant bit as a result of the addition of the mantissas of the two operands. The sum is taken in complement form if the signs are different but there is no carry out of the most significant bit.

h) The resultant sum has as sign:

1) The sign which is the same for both operands or

2) The sign of the augend if there is a carry out of the most significant bit position (augend > addend) and the signs disagree or

3) The sign of the addend if there is no carry out of the most significant bit position (addend > augend) and the signs disagree.

i) Add the most significant shift off bit to the resultant sum as follows:

1) If the signs are the same, add most significant shifted off bit to the least significant bit of the unnormalized sum.

2) If the signs are different and Aexp > Bexp and the magnitude of mantissa of A > magnitude of mantissa of B after the alignment takes place or if the signs are different and Bexp > Aexp and the magnitude of mantissa of B > magnitude of mantissa of A after the alignment takes place, subtract the most significant shifted off bit from the least significant bit of the mantissa of the unnormalized sum.

3) If the signs are different and Aexp > Bexp and the magnitude of mantissa of B > magnitude of A after the alignment takes place or if the signs are different and Bexp > Aexp and the magnitude of mantissa of A > magnitude of the mantissa of B after the alignment takes place, add the most significant shifted off bit to the least significant bit of the mantissa of the unnormalized sum. In the special case where $|A| = |B|$ after alignment (no rounding) the result is 0 0 0 ... 0.

j) Detect the position of the leading ONE in the mantissa of the sum, shift to left the mantissa until a ONE is at bit position 16 in 64-bit mode and 32-bit mode for the Inner word or bit position 40 in 32-bit mode for the Outer word or shift to the right by one if mantissa overflow has occurred. (If the option of normalization is used). Mantissa overflow is corrected for even if not normalizing.

k) Adjust the exponent of the sum as follows:

   1) Add one to the exponent of the final sum if overflow has occurred.

   2) Subtract from the exponent of the final sum the number of bit positions the leading ONE was shifted to be placed at bit 16 for 64-bit mode or 32-bit mode for the Inner word or at bit position 40 in 32-bit mode for the Outer word.

   3) If the mantissa is all zeros, the result (if normalizing) is zero for the whole word (sign, exponent and mantissa).

ADDITION IN 64-BIT MODE

CLOCK TIME T1--Figure 35 (This clock time is skipped if fixed point arithmetic is used.)

<u>Exponent Difference</u> (Bexp-Aexp) $\text{Bexp}+\overline{\text{Aexp}} \rightarrow$ SCR and LOD #6

1. Enable true form of exponent of B (bits 1 through 15) out of B.
2. Enable complement form of exponent of A (bits 1 through 15) out of A.
3. Enable complement of exponent of A into CPA (bits 1 through 15).
4. Enable true form of exponent of B into CPA (bits 1 through 15).
5. Enable the sign of B mantissa into CPA (actually zero, since B sign is not gated out of B register; bit 0).
6. Enable the bit carries because of the addition of exponents of B and A into CPA.
7. Force zeros into the mantissa part of CPA (this transmits end around carry into exponent field).
8. Clear and load clocks into SCR.
9. Put exponent part of CPA into SCR (8-15)* (CPA bits 74 through 79).
10. Clear and load clocks into LOD (CPA bits 64 through 73 and store carry-out).
11. Clear and load the latch for OSEQ (stores operation, add or subtract).

*In reality bits 10-15 are put into SCR because only 6 bits are needed for a count number 0 - 63.

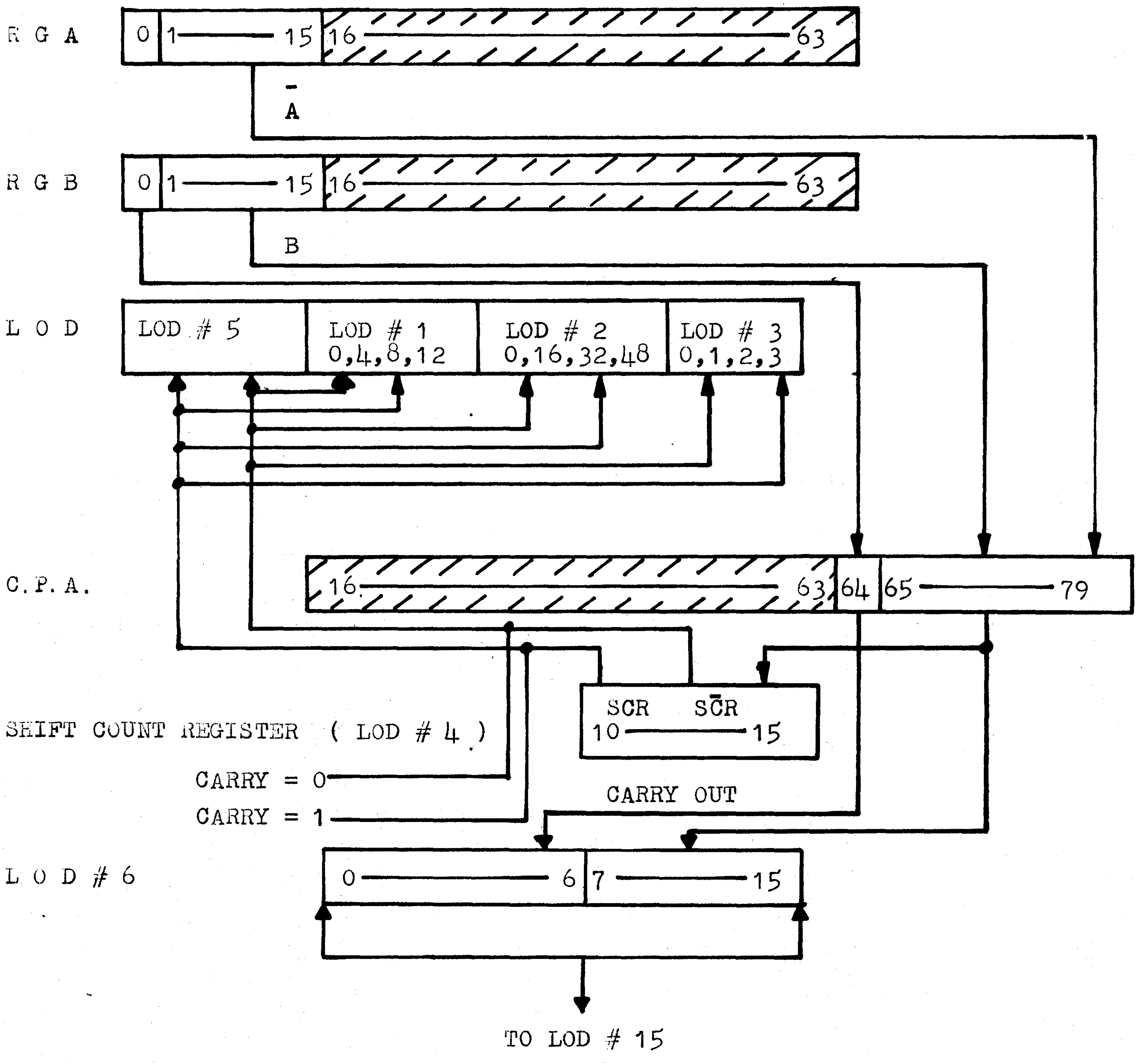


FIGURE 35. FLOW CHART OF ACTIONS AT TIME T 1

CLOCK TIME T2--Figure 36 (This clock time is skipped if rounding is not used or if fixed point arithmetic is used.)

Rounding (Optional)

1. Enable true output sign and mantissa of A
2. Enable true output sign and mantissa of B

} These enables are unnecessary in this clock time.

3. a. Enable A into LOG if there is (from exp. difference) a carry (exponent of B > exponent of A).

   b. Enable B into LOG if there is no carry (exponent A $\geq$ exponent of B).

4. Transfer LOG into BSW (bits 16-63).

5. Enable the 64-bit shift counter from SCR (true value if carry; complement if no carry).

6. Enable "force" shift left (control for end off; allows data in bit zero position).

7. Load most significant shifted off bit (appears in BSW bit zero position) into IMSB latch if exponent difference is less than 64.

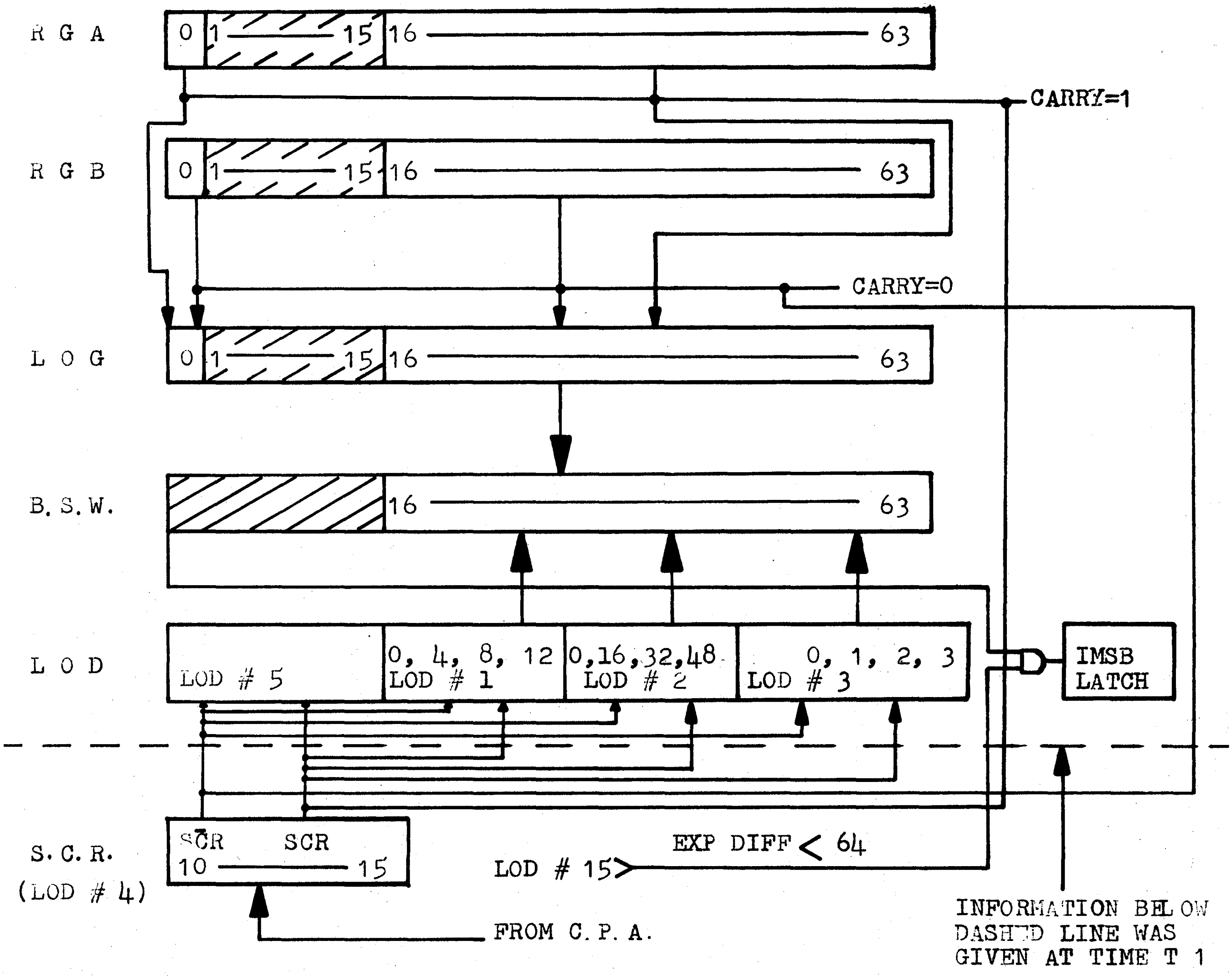


FIGURE 36. FLOW CHART OF ACTIONS
AT TIME T 2

CLOCK TIME T3--Figure 37

Align Mantissa with Smaller Exponent

1. Enable (select) complement of signs of A and B if unsigned and fixed point (when used with 2, forces signs to be equal--both ones).

2. Enable (select) true form of signs of A and B.

3. Clear and load the batch for OSEQ is signed or fixed point.

4. Enable (select) true form of mantissas of A and B.

5. Enable sign and mantissa of A into LOG if there was a carry.

6. Enable sign and mantissa of B into LOG if there was no carry.

7. Enable 64 shift counter from SCR

   a. True out of SCR if there was a carry

   b. Complement of SCR if there was no carry.

8. Transfer LOG into BSW (16-63 bits).

9. a. Transfer the BSW into A (16-64 bits) if there was a carry

   b. Transfer the BSW into B (0-15, 16-63 bits) if there was no carry

*10. a. Load A mantissa if there is a carry and exponent difference is less than 48

   b. Load B mantissa if there is no carry and exponent difference is less than 48.

11. Enable exponent of B (1-15 bits) out of B.

12. Enable original sign of A (0 bit) into sign bit position of A.

13. Enable B exponent into the exponent part of A.

14. Clear and load the sign and exponent part of A if there was a carry (Bexp > Aexp).

* Floating point only

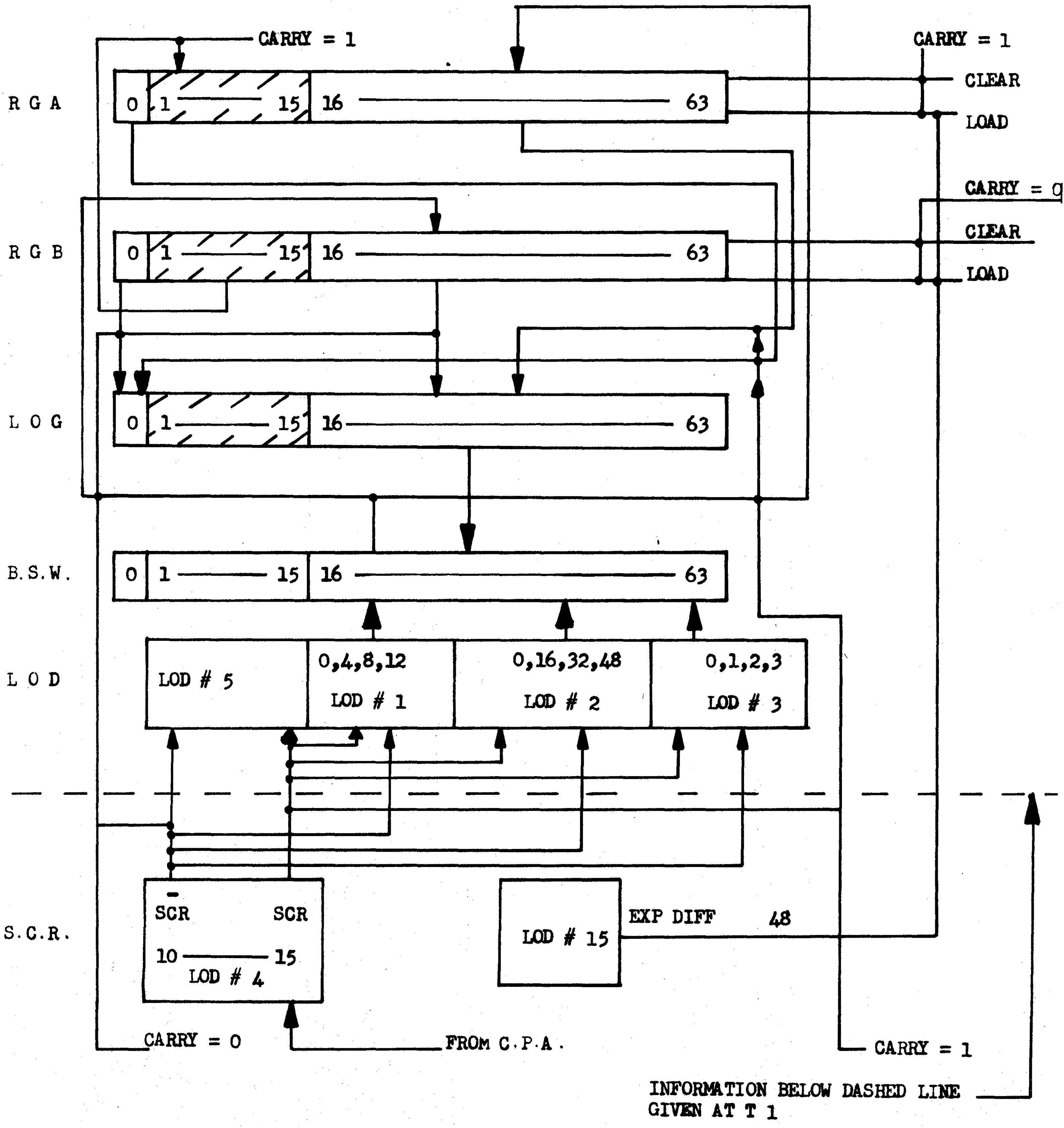


FIGURE 37. FLOW CHART OF ACTIONS AT TIME T 3

CLOCK TIME T4--Figure 38

Addition of Mantissa

1. Enable true form of mantissa out of B if mantissa signs are equal (OSEQ true).

2. Enable complement of mantissa out of B if mantissa signs are unequal (OSEQ false).

3. Enable true form of sign and mantissa out of A.

4. Transfer mantissa of A into CPA.

5. Transfer mantissa of B into CPA.

6. Inhibit EAC from exponent if signs are equal.

7. Enable bit carries into CPA (resulting from the addition of the mantissas).

8. Transfer sign of mantissa of B into CPA (actually zero)

9. Transfer the exponent of A into CPA (actually zero)

This puts transmits in bits 0 through 15 of CPA.

10. Transfer the mantissa of CPA into A

11. Set WCMP latch if there is no carry and signs are unequal.

12. Set overflow if the signs are equal and there is a carry (most significant bit of mantissa of CPA).

13. Clear and load clocks into OV1.

14. Transfer CPA equal signal (all transmits) to RGC (can only occur if signs are unequal) used for rounding.

15. Clear and load RGC.

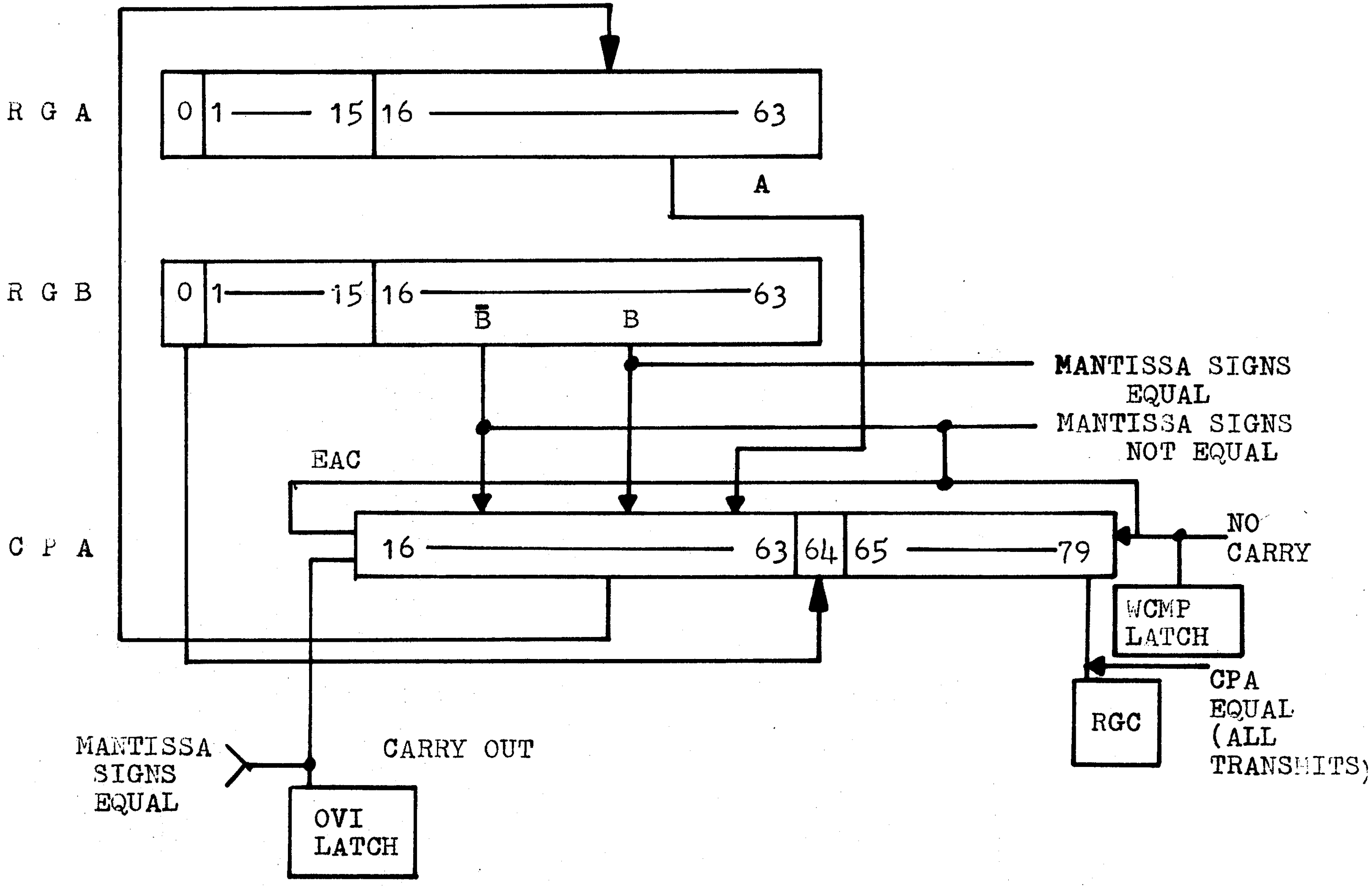


FIGURE 38. FLOW CHART OF ACTIONS AT TIME T 4

CLOCK TIME T5--Figure 39 (This clock time is skipped when not rounding or in fixed point.)

Complement, Round, Store Overflow

1. Enable the true form of mantissa out of A if WCMP latch is false.
2. Enable the complement of mantissa out of A if WCMP latch is true.
3. Transfer mantissa of A (1 or 2 above) into CPA.
4. Enable bit carries into CPA (0 - 15) and CPA (16 - 63).
5. Enable the exponent of A into CPA (65-79) (actually all zeros).
6. Clear and then load clocks into mantissa of A.
7. Force zeros into the mantissa part of CPA (from B).
8. a. Force SGE = 1 If ROUNDING and ADD 1 to MSB of mantissa and if WCMP latch is low

   b. Force SGE = 0 and STE = 0 If ROUNDING and subtract 1 from the MSB of mantissa if WCMP latch is low.
9. Set overflow OV1 using load clock only and if OSEQ is true.
10. Enable the mantissa sign of B into CPA.
11. Enable the mantissa part of CPA into A.

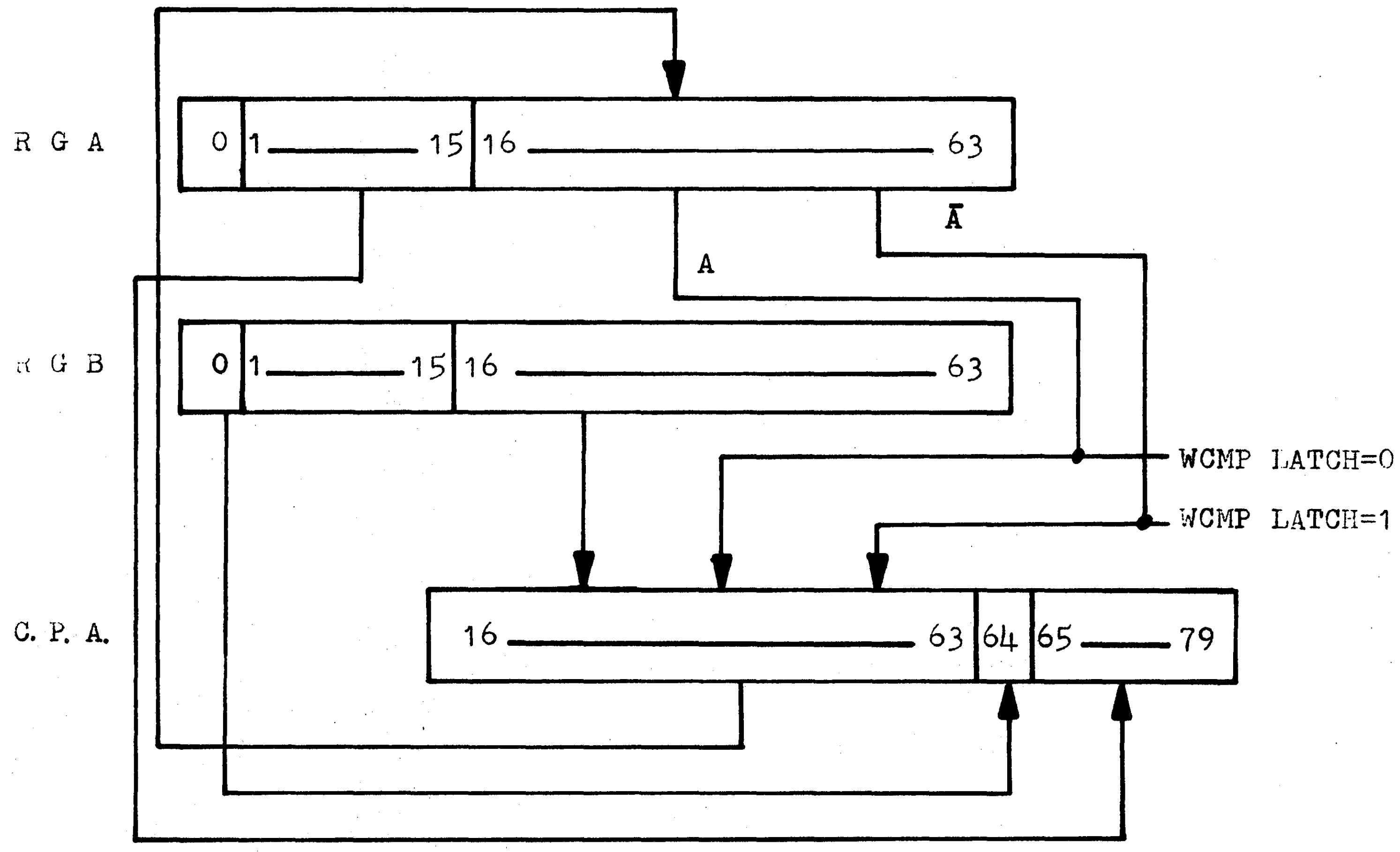


FIGURE 39. FLOW CHART OF ACTIONS AT TIME T 5

CLOCK TIME T6--Figures 40 and 41

(PART A) Complement, Normalize, Adjust Exponent, Determine Sign

1. Enable RGA (16-63) if WCMP latch = 0 or round variant is used.
2. Enable complement of RGA (16-63) if WCMP latch = 1 and rounding is not used.
3. Enable RGA (16-63) into LOD when normalizing.
4. Enable RGA (16-63) into LOG.
5. Enable LOG (16-63) into BSW.
6. Enable exponent correction bits into RGB (8-15) when normalizing.
7. Enable clear and load clocks into RGB (0-7).
8. Enable 00 111 111 into RGB (0-7) for exponent correction.
9. Enable clear clock for OV1.

(PART B) If Not Normalizing

1. Enable clear and load clocks for RGA (16-63).
2. Enable RGA (0-15).
3. Enable RGA (1-63) into CPA (65-79, 16-63).
4. Enable RGB (16-63) into CPA (16-63).
5. Enable bit carries into CPA (16-63) but disable CPA (65-79).
6. Compute correct sign of RGA.
7. Restore sign of RGA.
8. Enable CPA (1-15) into RGA (1-15).
9. Enable clear clock for OV1.

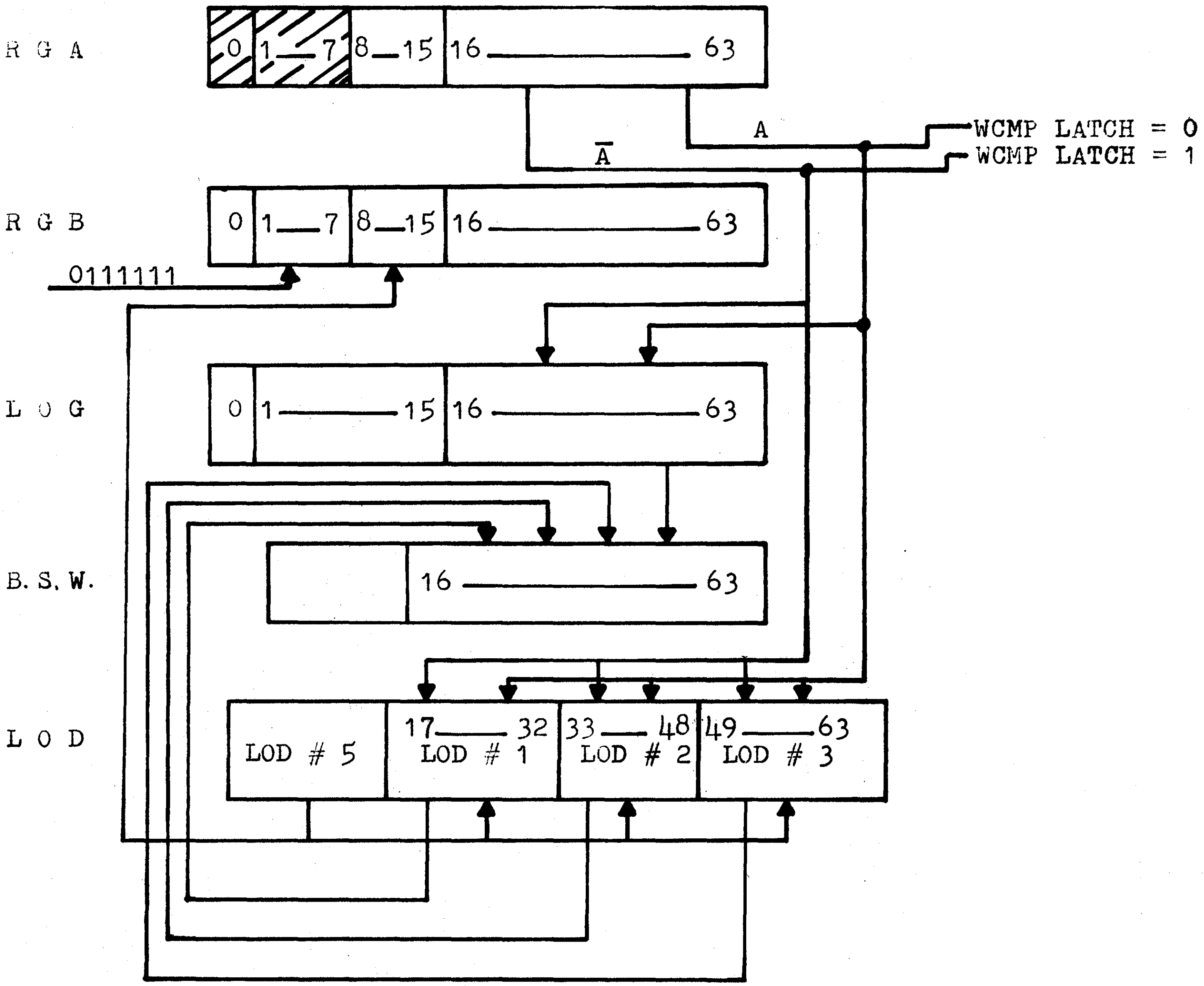


FIGURE40. FLOW CHART OF ACTIONS AT TIME T 6 (PART A)

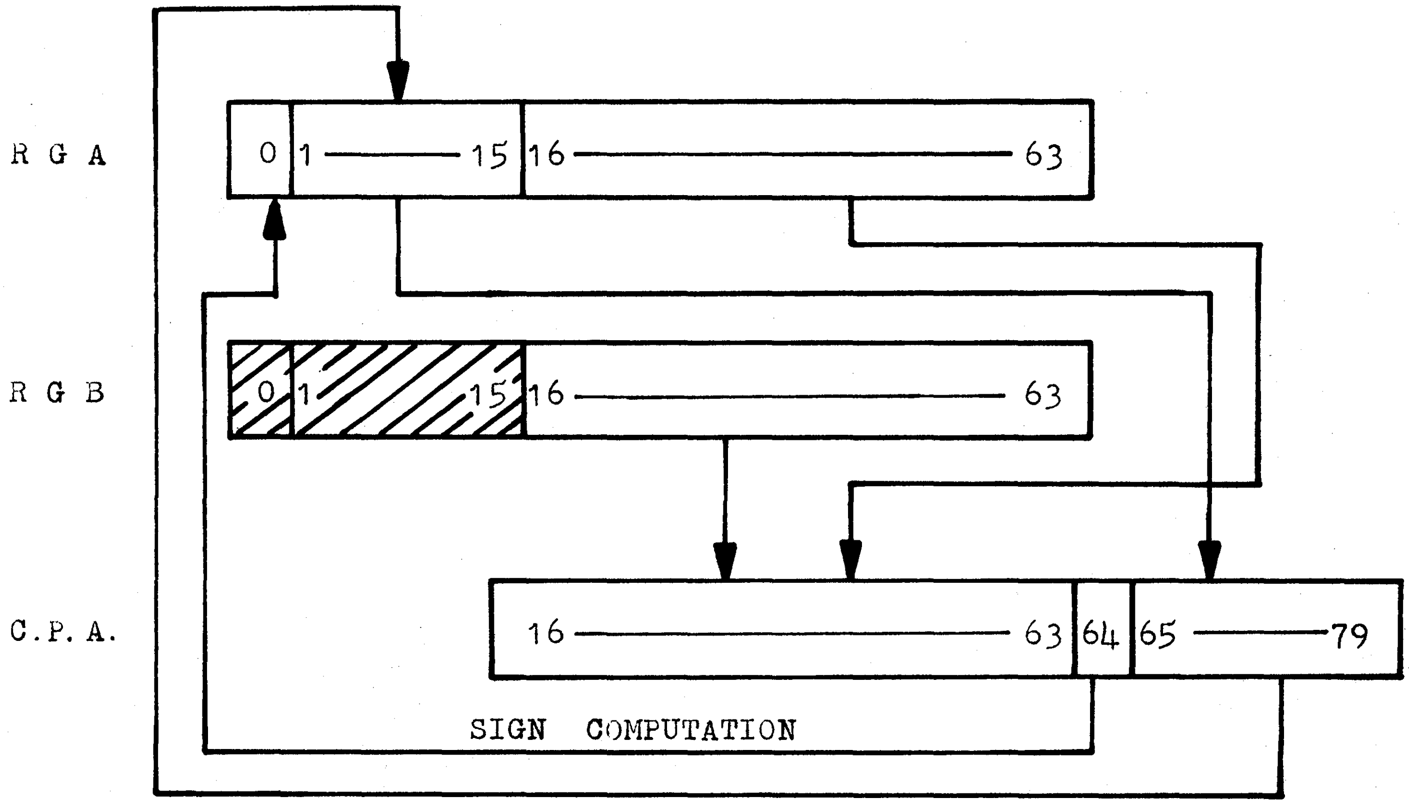


FIGURE 41. FLOW CHART OF ACTIONS
AT TIME T 6 (PART B)

CLOCK TIME T7--Figure 42 (This clock time is skipped if in fixed point.)

Correct Resultant Exponent

1. Enable true out of RGA (1-15).
2. Enable true out of RGB (8-15).
3. Enable true out of RGB (1-7) if normalize and there is:
   a. No overflow or
   b. Bit 16 of RGA was not a ONE prior to normalization.
4. Enable complement out of RGB (1-7) if there is:
   a. Overflow or
   b. Bit 16 was a ONE prior to normalization.
5. Enable RGA (1-63) into CPA (65-79, 16-63) (mantissa part is all zeros).
6. Enable RGB (1-63) into CPA (65-79, 16-63) (mantissa part is all zeros).
7. Enable bit carries into CPA (16-63, 65-79).
8. Clear mantissa of RGA if there is exponent underflow (conditionally).
9. Load clocks to RGA (0-15) or FYEASNOW-T and P-EX-UF--L and P-ZML--H-L.
10. Enable exponent overflow to mode REGISTER on FYEEXOFM-T and P----E----1.
11. Enable exponent underflow on FYENUF-M-T and P-ZML--H-L.
12. Clear and load clocks to F.
13. Set F on underflow or NO zero mantissa.
14. Clear clocks to OV1.
15. Restore the sign of RGA after computation.
16. Enable CPA (65-79) into RGA (1-15).

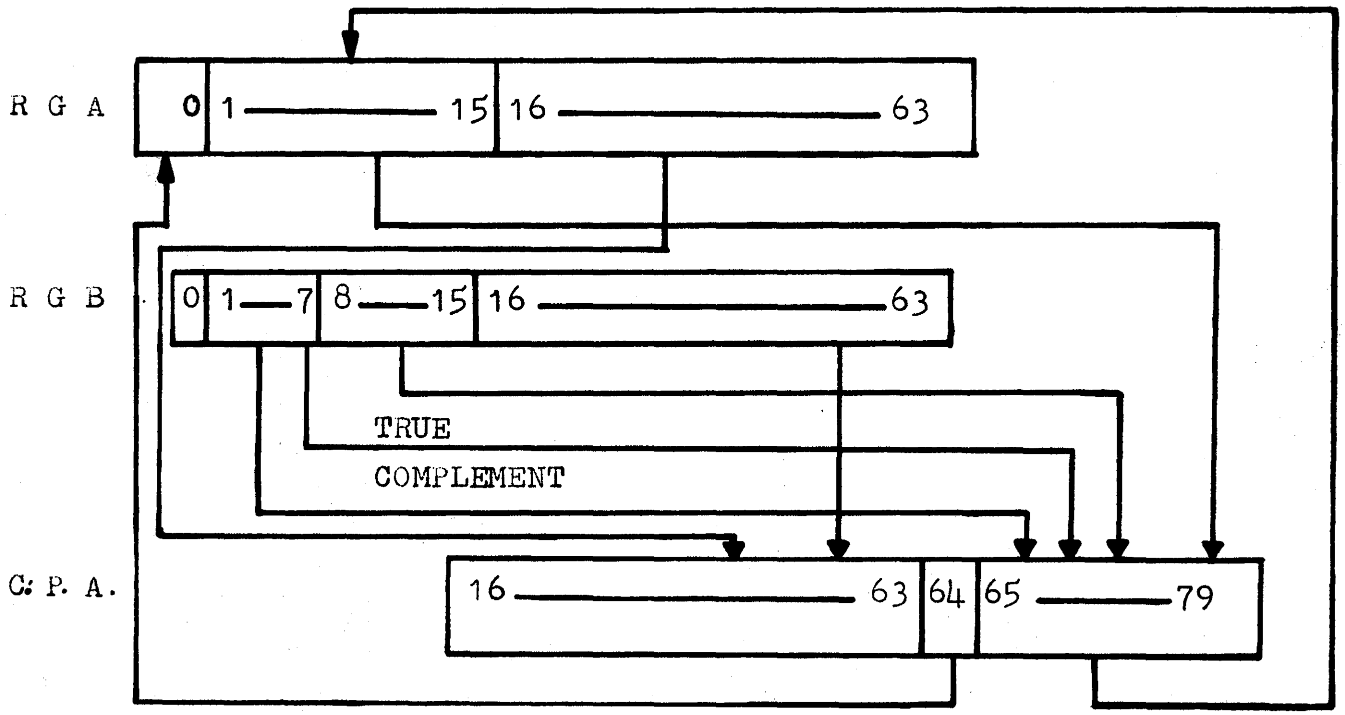


FIGURE 42. FLOW CHART OF ACTIONS
AT TIME T 7

ADDITION IN 32-BIT MODE

CLOCK TIME T1--Figure 43

Exponent Difference of Inner Word

1. Enable true out of RGB (9-15).
2. Enable complement out of RGA (9-15).
3. Enable RGA (0,1-7, 9-15) into CPA (64,65-71, 73-79).
4. Enable RGB (8,9-15) into CPA (72,73-79).
5. Enable WD4 inner and outer mantissa into CPA (16-63)*.
6. Enable bit carries into CPA (72,73-79).
7. Enable CPA sum of inner sign and exponent (72,73-79) into the BSW.
8. Enable clear and load clocks into the SCR.
9. Enable clear and load clocks into LOD.
10. Clear and load clocks into ISEQ and ØSEQ latch.
11. Enable signal to speed up path around the latch of stored carry.

*Since the part of LOG corresponding to the OUTER and INNER mantissa has not been enabled, the input to the CPA (16-63) looks like all zeros have been forced into it.

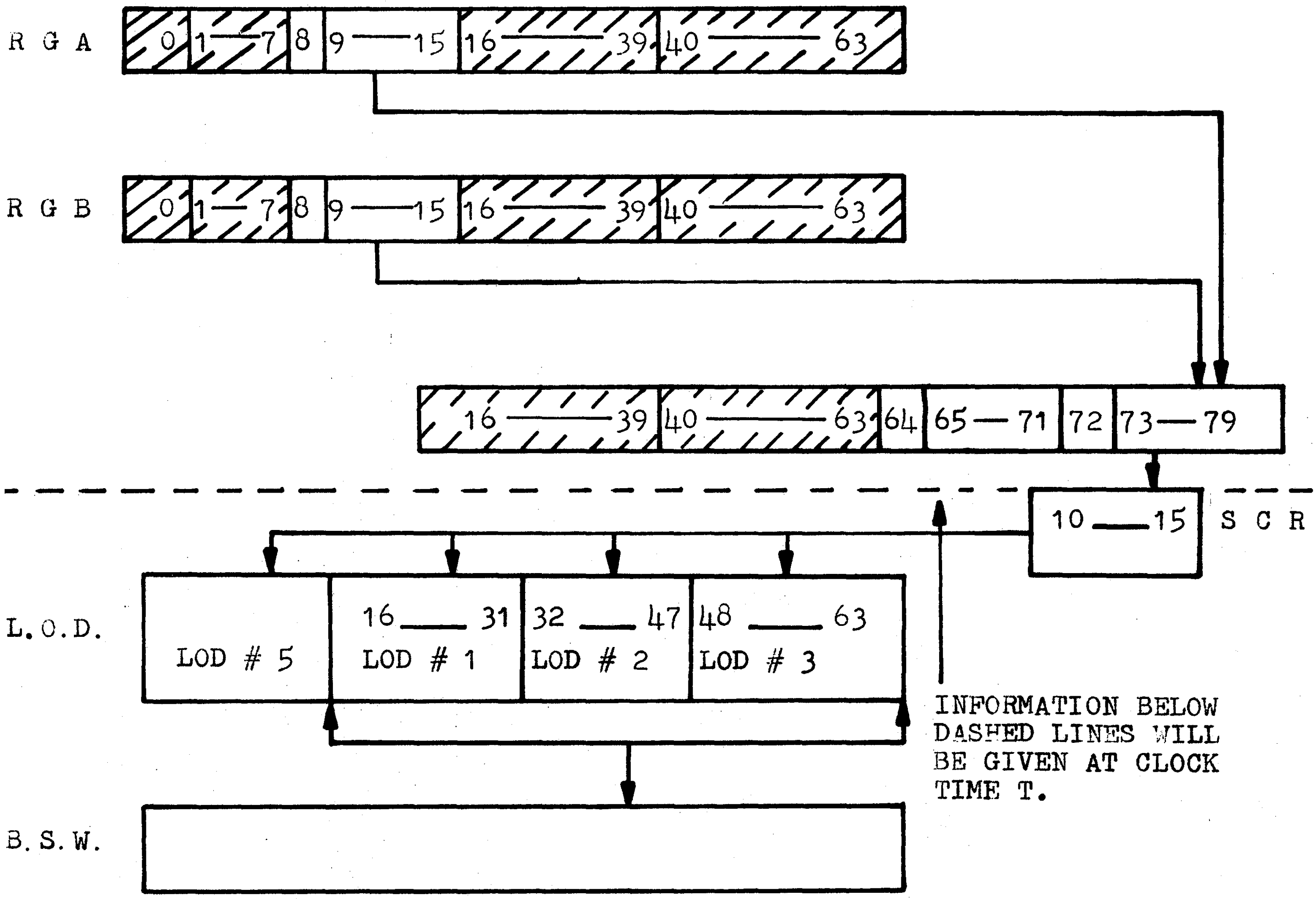


FIGURE 43. FLOW CHART OF ACTIONS AT TIME T 1

CLOCK TIME T2--Figures 44 and 45

Save MSB to be Shifted Off for Rounding in Inner Word

1. Enable true out of inner sign and inner mantissa of A (8,16-39).

2. Enable true out of inner sign and inner mantissa of B (8,16-39).

3. Enable RGA into LOG if there is a carry (Aexp < Bexp).

4. Enable RGB into LOG if there is no carry (Aexp ≥ Bexp).

5. Enable LOG (0-39) into BSW (24-63)*.

6. Enable CPA into shift count

   a. SCR true out if there is a carry

   b. SCR complement out if there is no carry.

7. Enable force shift left**.

8. Enable clear and load clocks into IMSB latch.

* Because of restrictions arising from signal controls LOG bits (0-39) are enabled into BSW, but bits (0-39) of LOG are effectively placed in the BSW (24-63 bits) by simply shifting each byte by 24 places to the right in order to be able to save the MSB in position 64.

** This is required in the case where there is no carry to complement the content of SCR into LOD and for shift to right to have the same result as left shift.

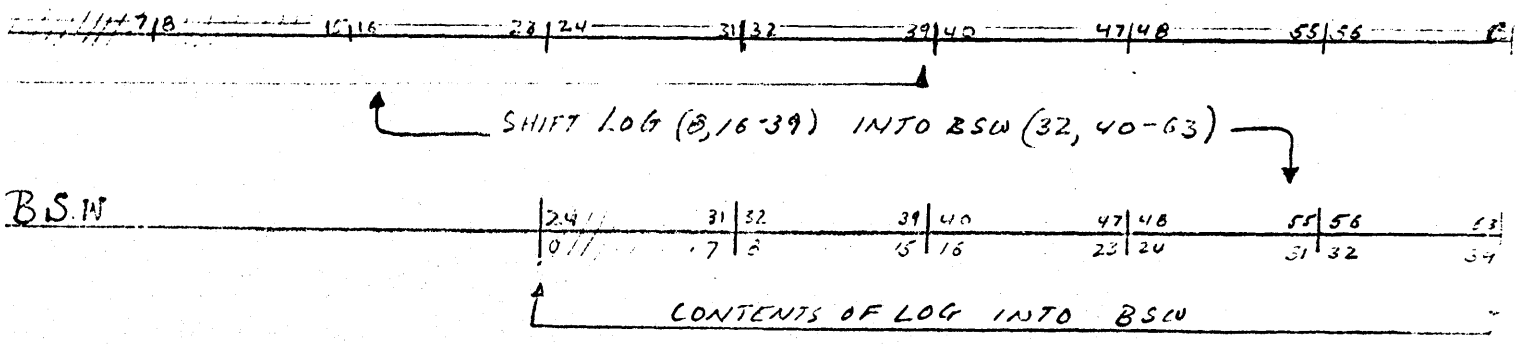


Figure 44. Diagrammatic Representation of Shifting Operation at Time T2

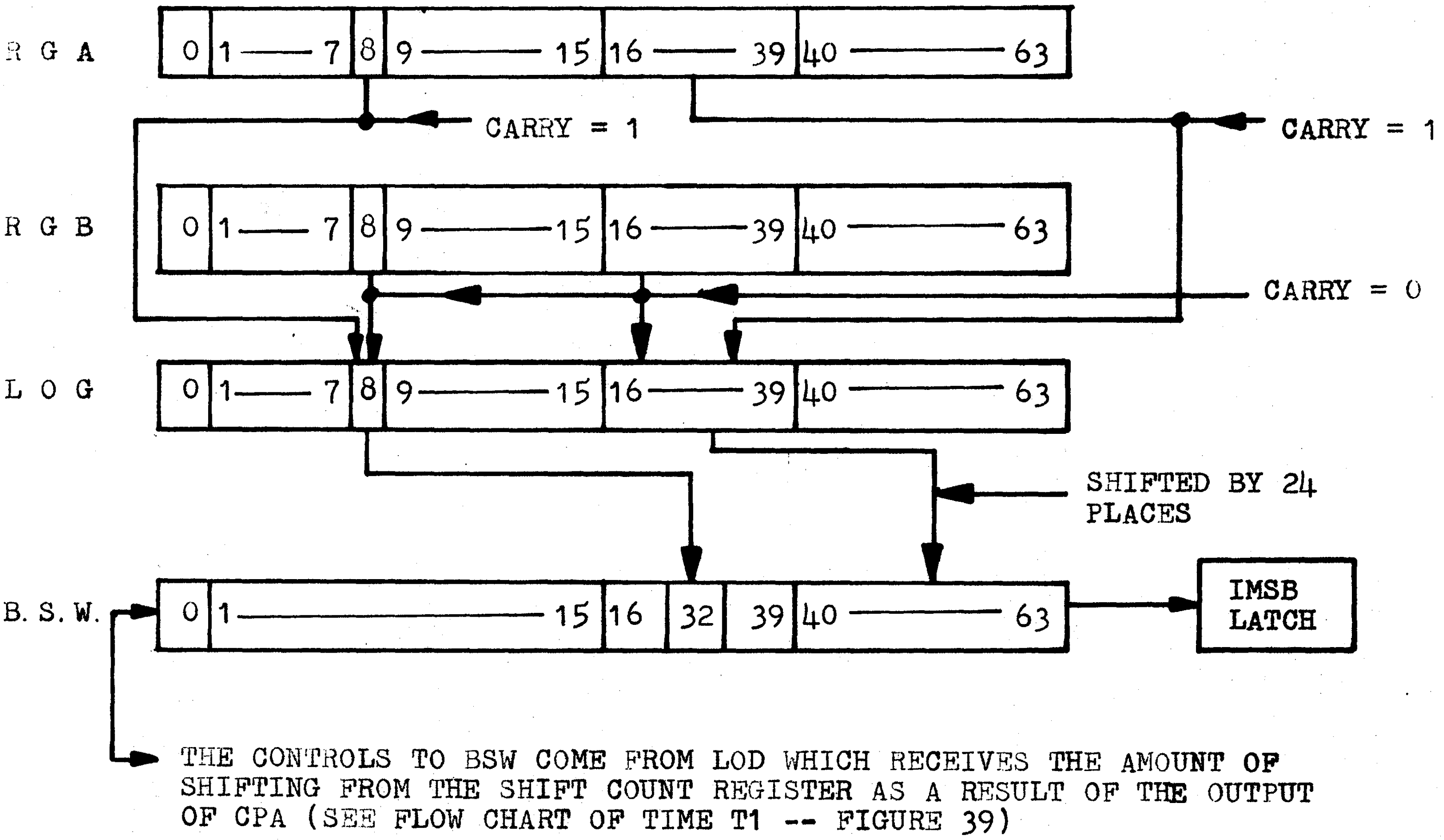


FIGURE 45. FLOW CHART OF ACTIONS AT TIME T 2

CLOCK TIME T3--Figures 46 and 47

(PART A) Difference of Exponents of Outer Word

1. Enable true out of RGB (1-7).
2. Enable complement out of RGA (0,1-7).
3. Enable RGA (1-7) into CPA (65-71).
4. Enable 01 111 111 into CPA (72-79).
5. Enable RGA (8-15) into CPA (72-79).
6. Enable RGB (0,1-7) into CPA (64,65-71).
7. Enable WD4 inner mantissa into CPA (16-39)*.
8. Enable bit carries into CPA (64,65-71).
9. Enable CPA sum of outer sign and exponent (64,65-71) into Barrel Switch (through the SCR).
10. Enable clear and load clocks into SCR.
11. Enable clear and load clocks into LOD.

(PART B) Align Mantissas of Inner Word with Smaller Exponent

1. Enable true out of RGA (8, 16-39).
2. Enable true out of RGB (8, 16-39).
3. Enable RGA (8,16-39) into LOG (8,16-39) if there is a carry.
4. Enable RGB (8,16-39) into LOG (8,16-39) if there is no carry.

* The WD4 inner mantissa is brought into CPA (16-39) to insert zeros to CPA (16-39) because since the part of MSG corresponding to WD4 has not been enabled, its output looks like a zero.

5. Enable CPA into SC from SCR depending upon the end around carry. Also enable SC > 48 detection for inner exponent

   a. SCR true out if there is a carry

   b. SCR complement out if there is no carry.

6. Enable LOG (16-39) into BSW (16-39).

7. Enable BSW (16-39) into RGA (16-39).

8. Enable BSW exponent into RGB exponent.

9. Enable BSW (16-39) into RGB (16-39).

10. Enable clear and load clocks into RGA (16-39) if there is a carry.

11. Enable clear and load clocks into RGB (16-39) if there is no carry.

12. Clear and load clocks to the ISEQ latch.

13. Enable true out of RGB (9-15).

14. Enable RGB exponent into RGA exponent.

15. Enable clear and load clocks into RGA (8-15) if there is a carry and E1 = 1.

16. Restore sign of RGA (8).

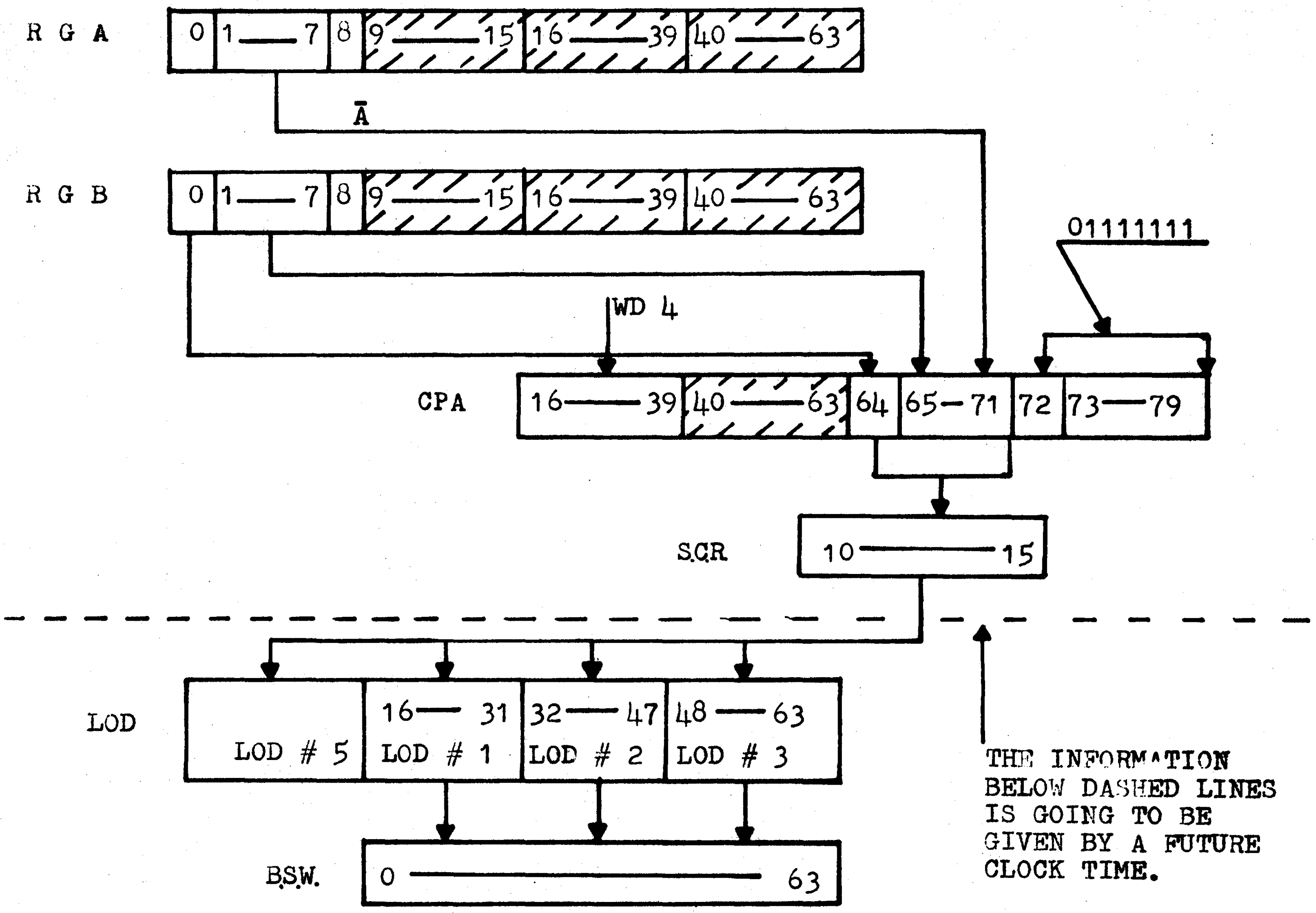


FIGURE 46. FLOW CHART OF ACTIONS AT TIME T 3 (PART A)

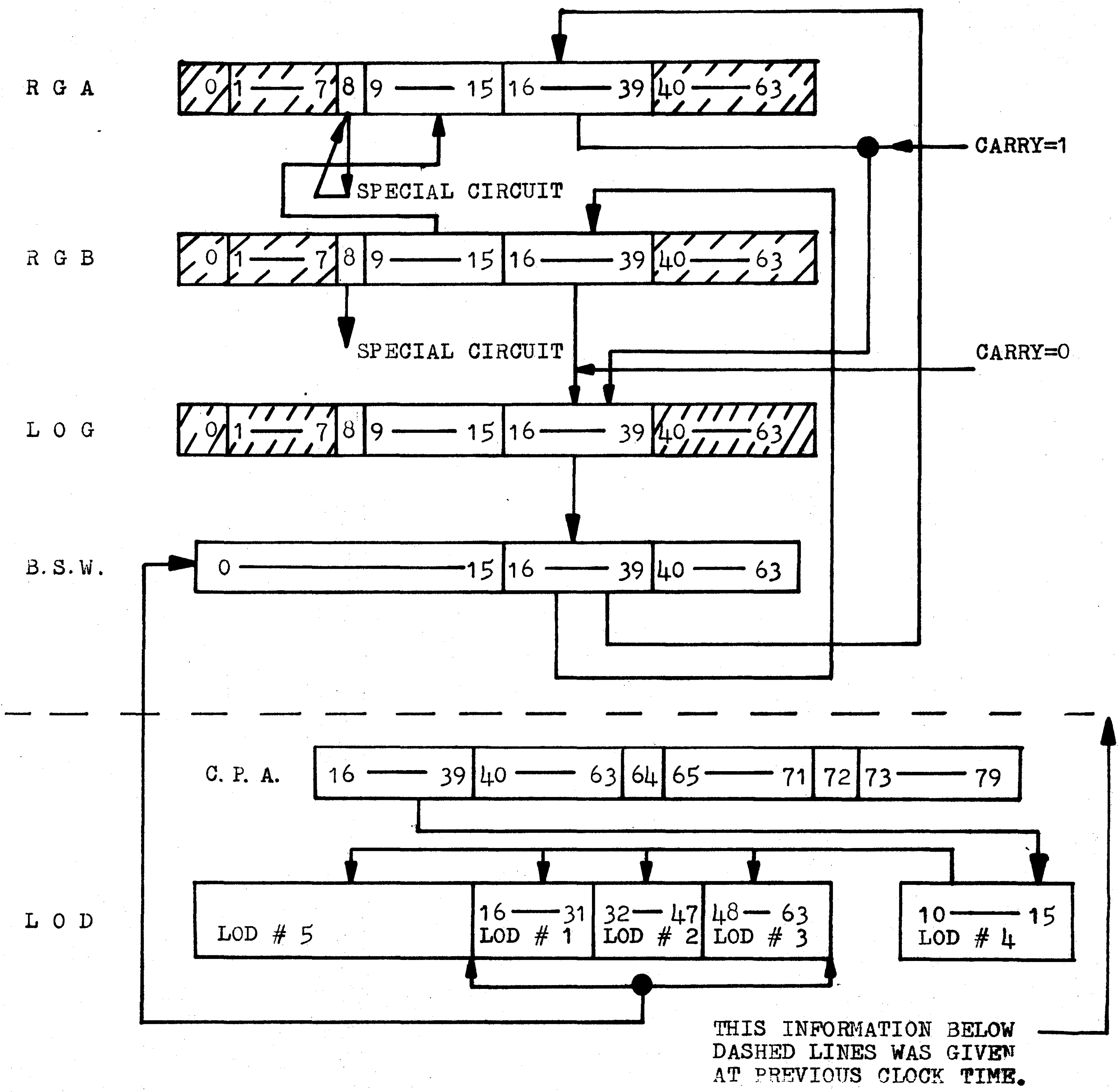


FIGURE 47. FLOW CHART OF ACTIONS
AT TIME T 3 (PART B)

CLOCK TIME T4--Figure 48

Save MSB of Bits to be Shifted Off of Outer Word

1. Enable true out of outer sign and mantissa of A (0,40-63).

2. Enable true out of outer sign and mantissa of B (0,40-63).

3. Enable A into LOG if there is a carry (Aexp < Bexp).

4. Enable B into LOG if there is no carry (Aexp > Bexp).

5. Enable LOG (40-63) into BSW (40-63).

6. Enable CPA out into SC from

   a. SCR true out if there is a carry

   b. SCR complement out if there is no carry.

7. Enable force shift left.

8. Enable clear and load clocks into OMSR latch.

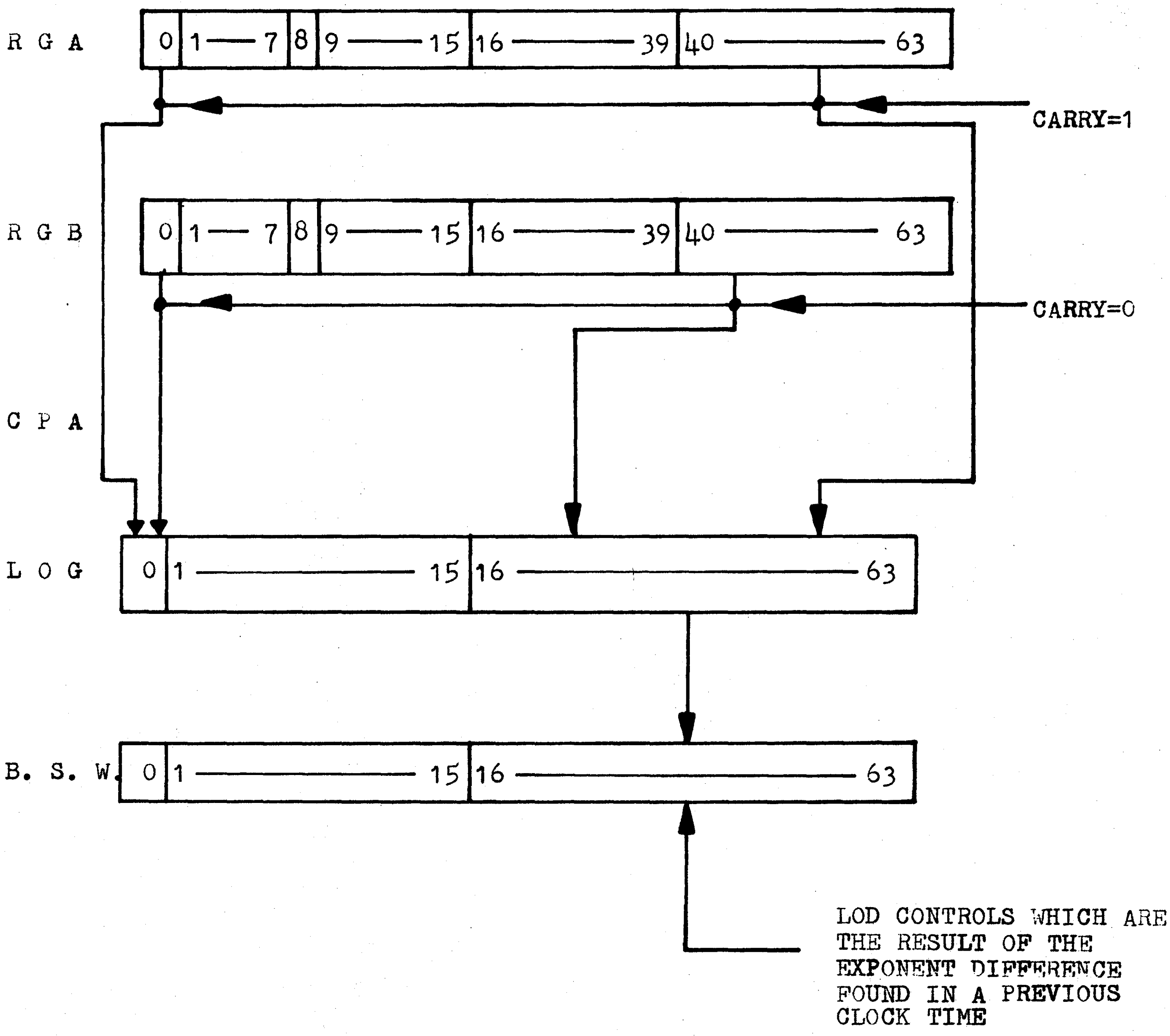


FIGURE 48. FLOW CHART OF ACTIONS AT TIME T 4

CLOCK TIME T5--Figures 49 and 50

(PART A) Align Mantissas of Outer Word with Smaller Exponent

1. Enable true out of outer sign and mantissa of A (0,40-63).
2. Enable true out of inner sign and mantissa of B (0,40-63).
3. Enable A into LOG (0,40-63) if there is a carry.
4. Enable B into LOG (0-,40-63) if there is no carry.
5. Enable CPA into shift counter and then
   a. SCR true out if there is a carry
   b. SCR complement if there is no carry.
6. Enable LOG into BSW (40-63).
7. Enable BSW (40-63) into A (40-63).
8. Enable BSW (0-15,40-63) into B (0-15,40-63).
9. Enable load clocks to OUTER mantissa of A if there is a carry and PEXDI-L48H is true.
10. Clear and load clocks to outer mantissa of B if there is no carry.
11. Enable outer exponent of B (1-7).
12. Clear and load clocks into outer sign and exponent of A (0-7).
13. Enable exponent of B into exponent of A (0-15).
14. Restore outer sign of A (0:1).

(PART B) Add Mantissas of Inner Word Store Overflow (OV1) (if there is any)

1. Enable clear and load clocks to outer sign and exponent of RGA if E = 1 and there is a carry.
2. For addition FYE-K----T is 0 from CU.
3. Enable true out of inner mantissa of RGB if P--ISEQ--H.

4. Enable complement out of inner mantissa of RGB if P--ISEQ--L.

5. Clear and load clocks to outer sign equal latch.

6. Enable true out of inner mantissa of RGA.

7. Enable the complement out of SCR.

8. Enable RGA (1-7,9-15,16-39) into CPA.

9. Enable RGB (8,16-39).

10. Force zeros into CPA (WD4 outer mantissa to CPA outer mantissa).

11. Enable the bit carries into CPA.

12. Inhibit end around carry through exponent if ISEQ high.

13. Enable CPA (16-39) into RGA (16-39).

14. Clear and load clocks into inner CMP latch.

15. Clear and load clocks into inner mantissa of RGA (16-39).

16. Enable to set OV1 (for inner word).

17. Enable true output of RGA inner sign.

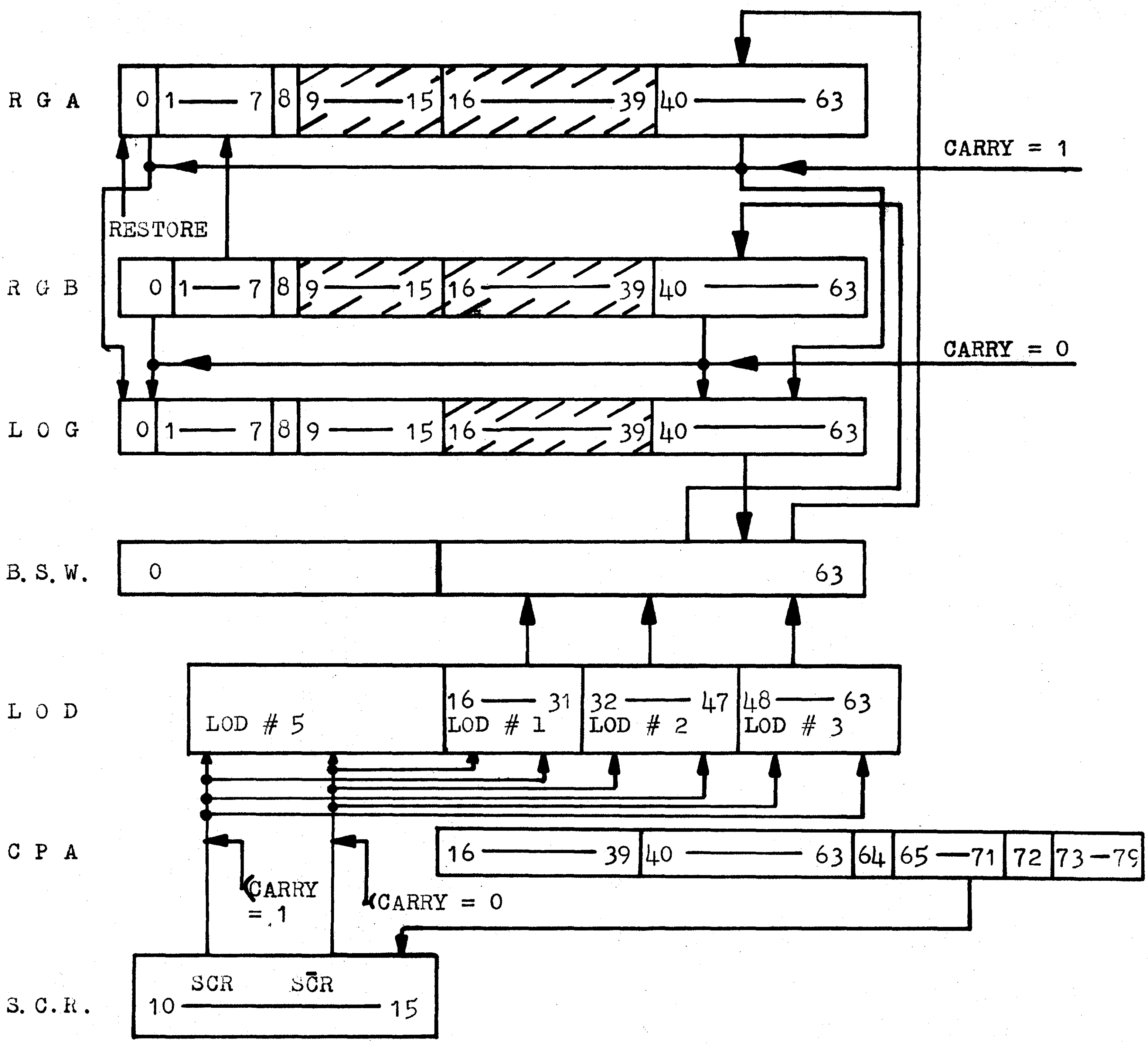


FIGURE 49. FLOW CHART OF ACTIONS AT TIME T 5 (PART A)

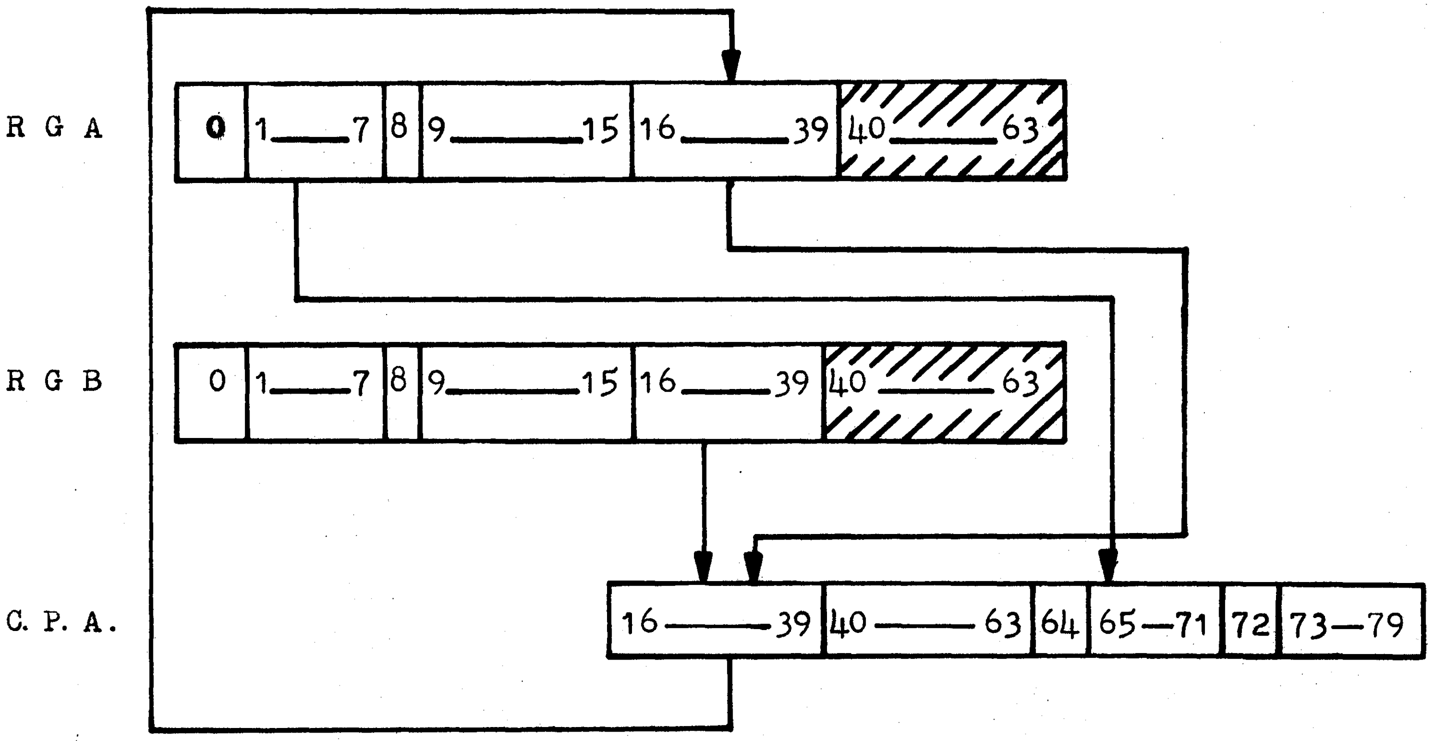


FIGURE 30. FLOW CHART OF ACTIONS
AT TIME T5 (PART B)

CLOCK TIME T6--Figure 51

Complement (if necessary), Round, Store Overflow (if there is any) of Inner Word

1. Enable true out of inner mantissa of RGA (16-39) if inner CMP latch output is low.
2. Enable complement of inner mantissa of RGA (16-39) if inner CMP latch output is high.
3. Enable RGA (16-39) into CPA (16-39).
4. Enable bit carries into CPA inner sign and exponent (72, 73-79).
5. Enable bit carries into CPA mantissa.
6. Enable zeros into CPA (enable RGA (40-63) into CPA (40-63)).
7. Enable to set OV1 (for inner word).
8. Enable RGB inner mantissa (16-39) into CPA (16-39).
9. Enable adder to round properly.
10. Enable RGB outer sign (0) to CPA outer sign (64).
11. Enable RGB outer exponent (1-7) to CPA (65-71).
12. Enable inner sign of RGB into CPA (72).
13. Enable RGA (9-15) into CPA (73-79).
14. Enable CPA (16-39) into RGA (16-39).

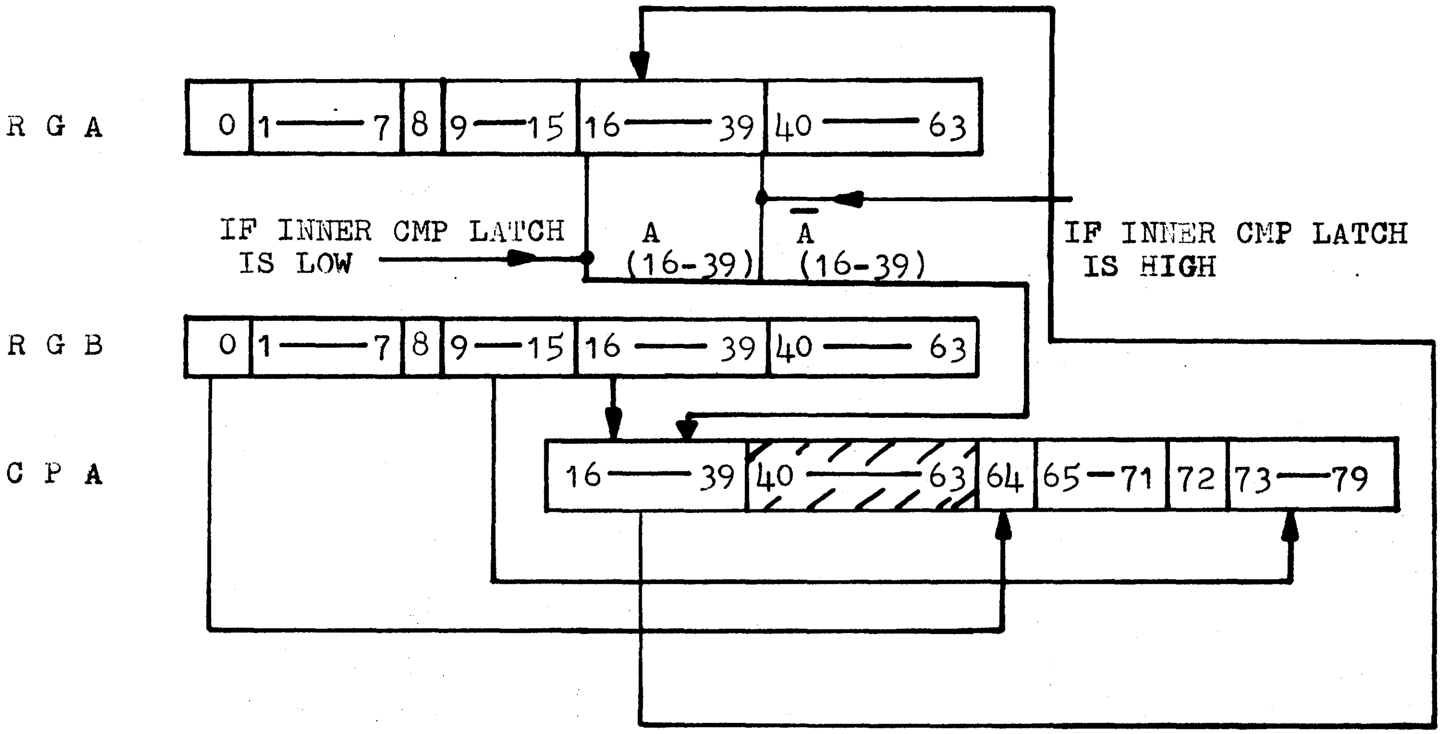


FIGURE 51. FLOW CHART OF ACTIONS
AT TIME T 6

CLOCK TIME T7--Figures 52 and 53

(PART A) Add Mantissas of Outer Word, Store Overflow (if there is any)

1. Enable true out of RGB (40-63) if outer sign equal latch output is high.
2. Enable complement out of RGB (40-63) if outer sign equal latch output is low.
3. Enable true out of RGA (40-63).
4. Enable RGA (0) to CPA (64).
5. Enable RGA (1-7,8) into CPA (65-71, 72).
6. Enable RGB (9-15) into CPA (73-79).
7. Enable RGA (40-63) into CPA (40-63).
8. Enable RGB (40-63) into CPA (40-63).
9. Enable zeros into CPA (16-39) (because the signal calls for WD4 inner mantissa into CPA inner mantissa).
10. Enable bit carries into CPA.
11. Inhibit end around carries through exponent if outer sign equal latch is high.
12. Enable CPA (40-63) to RGA (40-63).
13. Clear and load clocks to outer CMP latch.

(PART B) Complement (if needed), Normalize, Adjust Exponent of Inner Word

1. Clear and load clocks into RGA (40-63).
2. Enable true out of RGA (16-39) if inner sign equal latch output is low.
3. Enable complement out of RGA (16-39) if inner sign equal latch output is high.
4. Enable RGA (16-39) into LOG (16-39).

5. Enable LOG (16-39) into barrel switch (16-39).

6. Enable exponent adjustment into Inner exponent of RGB.

7. Enable clear and load clocks into LOD.

8. Clear OV2 if no overflow exists.

9. Load OV2 if there is overflow.

10. Clear and load RGA (16-39) and RGB (8-15).

11. Enable LOD (8,9-15) to RGB (8,9-15).

12. Shift right by ONE if overflow exists.

13. Enable (conditionally) to set F bit if F bit has been set and there is OV1.

14. Inhibit the clear clocks to RGD.

15. Enable barrel switch (16-39) into RGA (16-39).

16. Clear overflow (OV1).

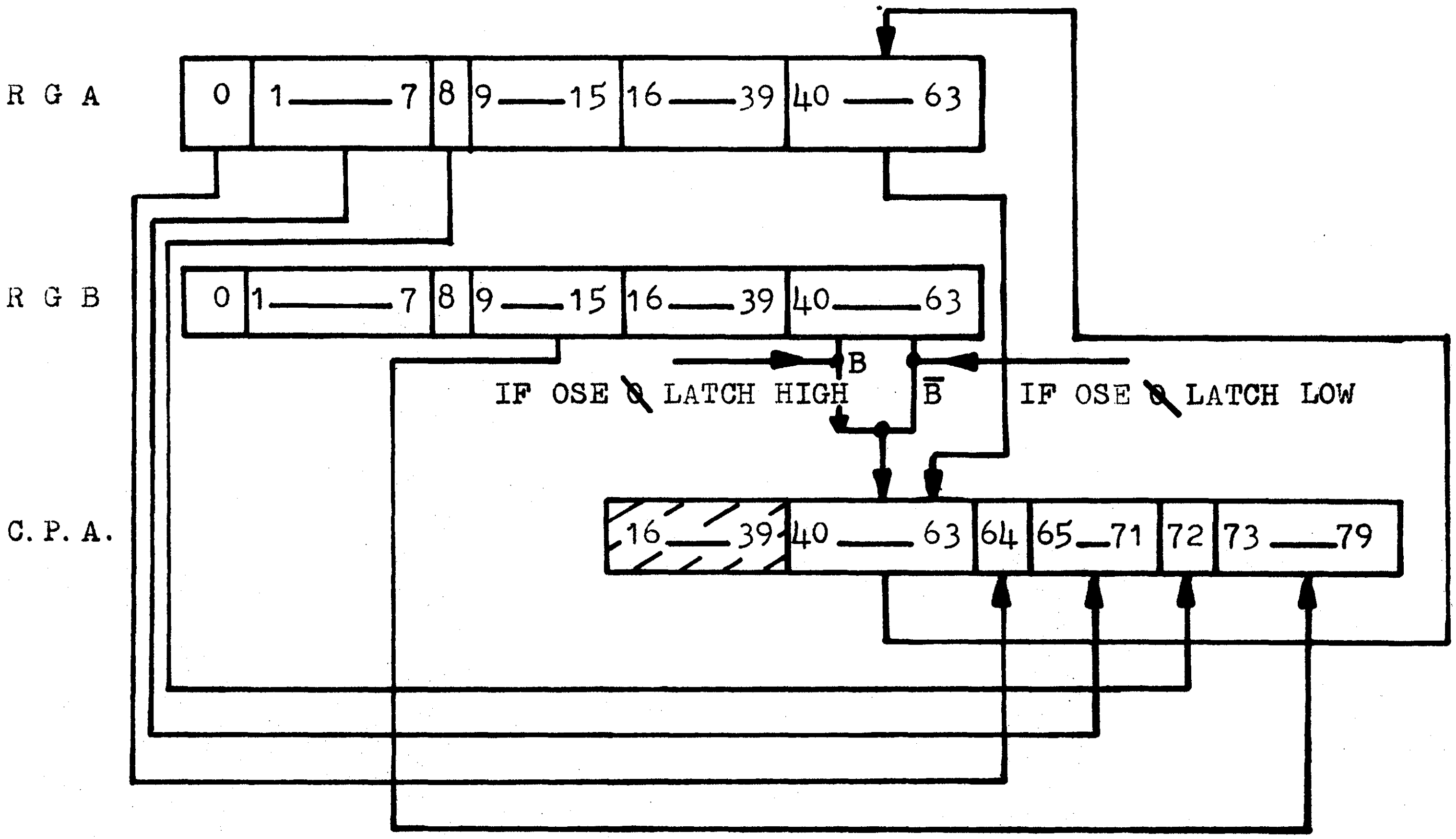
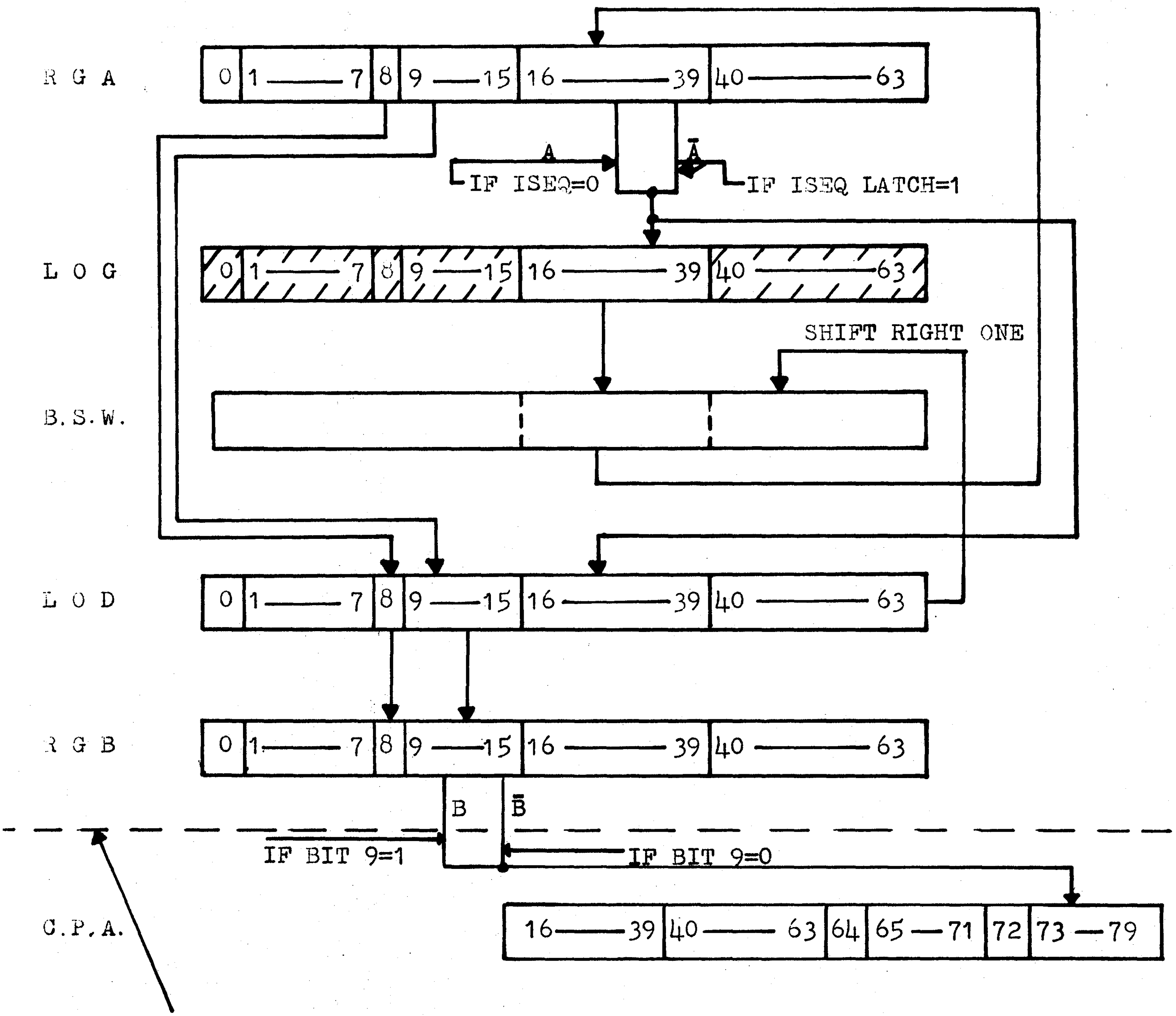


FIGURE 52. FLOW CHART OF ACTIONS AT TIME T 7 (PART A)



THE PART BELOW DASHED LINE DOES NOT BELONG TO TIME T7 BUT IS SHOWN FOR THE SAKE OF CLARITY OF PRESENTATION.

FIGURE 53. FLOW CHART OF ACTIONS AT TIME T 7 (PART B)

CLOCK TIME T8--Figure 54

Complement (if needed), Round, Adjust Exponent of Outer Word

1. Enable true out of RGA (40-63) if outer CMP is low.
2. Enable complement out of RGA (40-63) if outer CMP is high.
3. Enable RGA (40-63) into CPA (40-63).
4. Enable bit carries into CPA (64-71).
5. Enable RGB (0) into CPA (64).
6. Enable RGB (40-63) into CPA (40-63).
7. Enable RGA (1-39) into CPA (65-79; 16-39).
8. Clear and load clocks to RGA (40-63).
9. Round properly.
10. Enable to set OV2 (for outer word).
11. Enable CPA (40-63) to RGA (40-63).

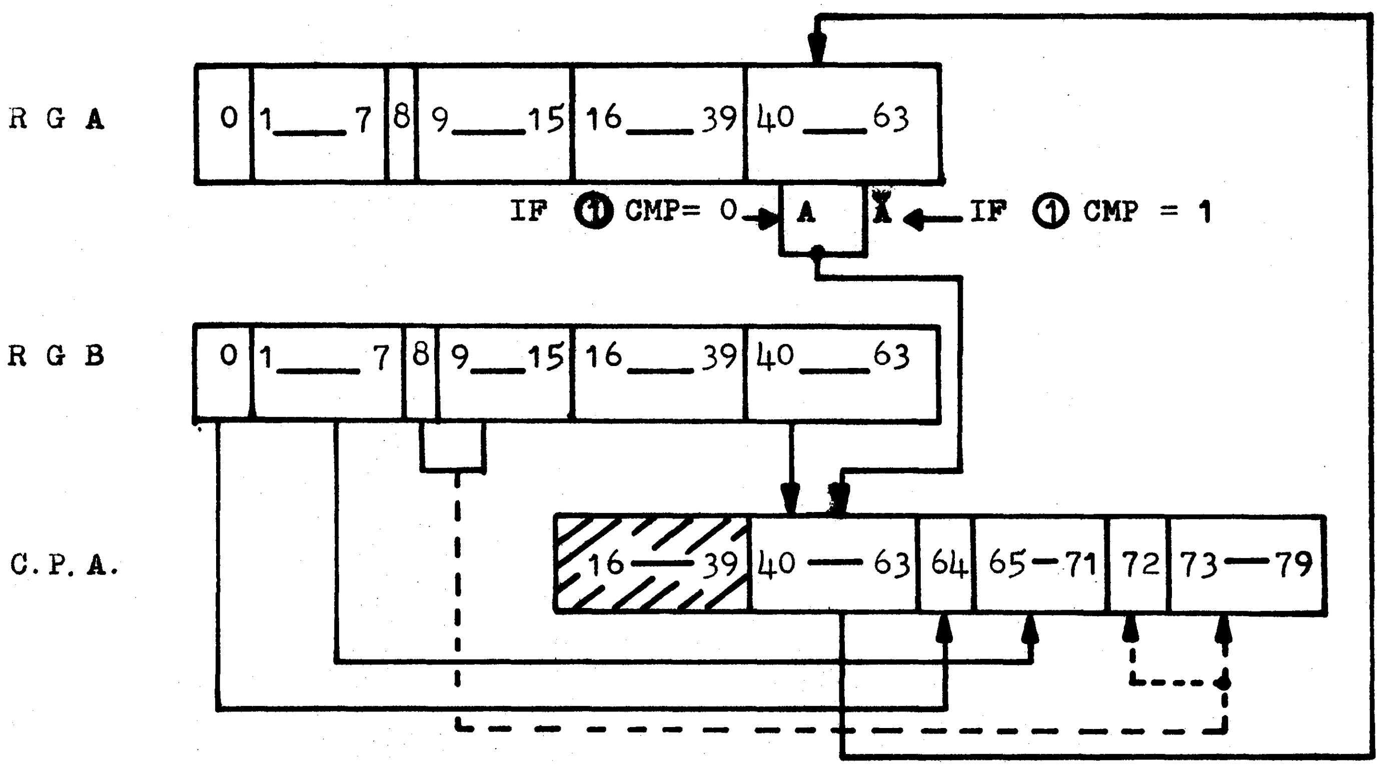


FIGURE 54. FLOW CHART OF ACTIONS AT TIME T 8

CLOCK TIME T9--Figure 55

Complement (if needed), Normalize, Adjust Exponent of Outer Word or Correct Exponent and Sign of Inner Word

1. Clear and load clocks into RGA (40-63).
2. Enable true out of RGA (40-63) if OCMP latch output is low.
3. Enable complement out of RGA (40-63) if OCMP latch output is high.
4. Enable RGA (40-63) into LOG (40-63).
5. Enable LOG (40-63) into barrel switch (40-63).
6. Enable load clocks to RGA (40-63).
7. Enable LOD (0-7) to RGB (0-7).
8. Enable (conditionally) underflow if mantissa is not ZERO.
9. Inhibit clear clocks to RGD.
10. Enable exponent adjustment RGB (8-15).
11. Clear and load RGB (0-15).
12. Enable RGA (9-15) into CPA (73-79).
13. Enable RGA (16-39) into CPA (16-39).
14. Enable RGB (16-39) into CPA (16-39).
15. Enable carries into CPA (16-39).
16. Enable CPA sum (72-79) to RGA (8-15).
17. Enable clear clocks to RGA (8-15) if E1 = 1.
18. Enable clear clock to RGA (16-39) if there is exponent overflow during normalization.
19. Enable exponent overflow into mode register.
20. Enable load clocks into RGA (8-15) in case of overflow or underflow of exponent.
21. Enable clears and loads to F1.

22. Restore sign of RGA (8).

23. Enable set of F1 if F1 bit has been set and there is OV2.

24. Compute correct sign of RGA (8).

25. Enable RGB (9-15) to CPA (73-79).

26. Enable bit carries into CPA (n, 73-79).

27. Enable CPA sum (72, 73-79) into RGA (8, 9-15).

28. Force a shift right ONE if overflow occurs.

29. Inhibit Section carries.

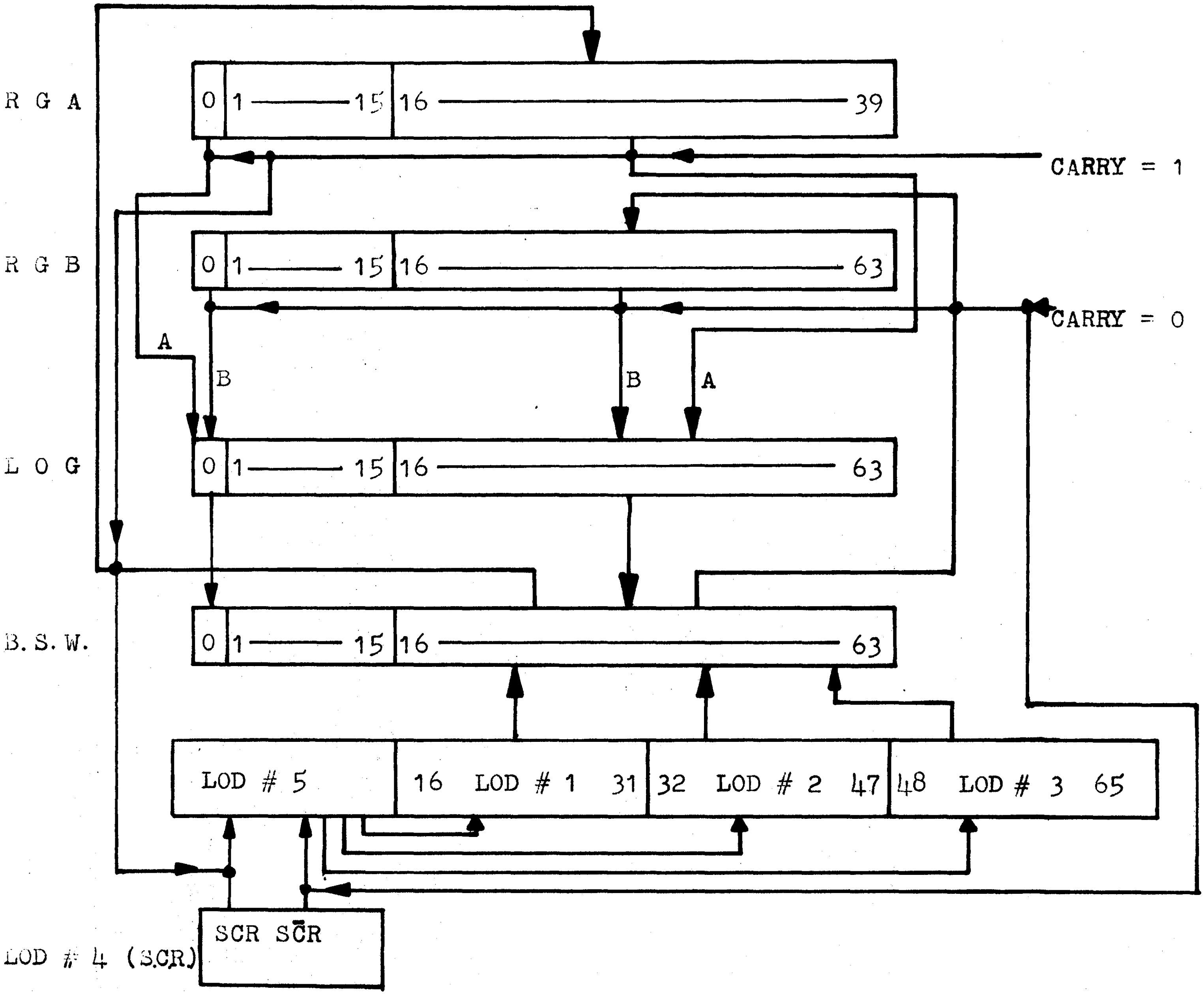


FIGURE 55. FLOW CHART OF ACTIONS
AT TIME T 9

CLOCK TIME T10

Correct Resultant Exponent and Sign of Outer Word

1. Enable RGA (0-7).
2. Enable true out of RGB (1-7) (for exponent adjustment).
3. Enable RGA (1-15) into CPA (65-79).
4. Enable RGB (1-15) into CPA (65-79).
5. Enable bit carries into CPA (72-79).
6. Enable CPA sum (64-71) to RGB (8-15).
7. Enable bit carries into CPA (64-71).
8. Enable clear clocks to RGA (0-7) if E = 1.
9. Enable clear clocks to RGA (40-63) if E = 1 and exp. UF.
10. Enable load clocks to RGA (1-7) if FYEASNOO-T is true which conditionally clears OUTER word of RGA in exponent overflow or underflow or if there is underflow and P-ZML--H-L is true.
11. Enable CPA sum (64-71) to RGA (0-7).
12. Clear the OV2 latch.
13. Enable (conditionally) underflow into RGD on E = 1 and when mantissa is not ZERO.
14. Enable exponent underflow or overflow if any of them occurs.
15. Enable clear and load clocks to F.
16. Inhibit clear clocks to RGD.
17. Restore the sign of RGA (0).
18. Compute correct of RGA (0).

B. Subtraction

This instruction is executed in a manner similar to that used for addition. The minuend is held in A register and the subtrahend specified by the content of the ADR field is held in B register. Before the process begins the sign of the subtrahend changes and the two operands are added exactly as in the implementation of the addition previously described.

Table 34 provides a complete summary of the actions during subtraction, but it is suggested that the information given in the latter part of addition (summary of actions) be thoroughly considered before attempting to analyze the contents of this table. It is felt that the following remarks might be of help in analyzing this table. They are consistent with the procedure of implementation of the addition instructions.

Table 34. Truth Table of Conditions in Subtraction (4)

CARRY OUT OF MSB OF MANTISSA	SIGN OF MANTISSA	SIGN OF SUBTRAHEND	END AROUND CARRY	COMPLEMENT SUBTRAHEND MANTISSA	COMPLEMENT MANTISSA OF SUM	DOES MANTISSA OVERFLOW OCCUR?	SIGN OF SUM	REMARKS
0	0	0	0	1	1	0	1	"0" denotes
0	0	1	0	0	0	0	0	FALSE, "1"
0	1	0	0	0	0	0	1	denotes TRUE
0	1	1	0	1	1	0	0	In all cases
1	0	0	1	1	0	0	0	except for
1	0	1	0	0	0	1	0	sign where
1	1	0	0	0	0	1	1	"0" = +
1	1	1	1	1	0	0	1	"1" = -

1. End Around Carry is used when the signs of the original operands (prior to changing the sign of subtrahend) do not disagree and there is a Carry out of the most significant bit of the sum when these two operands are added.

2. When the two operands are added the mantissa of the subtrahend is complemented (1's complement) only if the signs of both operands are the same.

3. The sum (mantissa) is complemented (1's complement) only if the signs of the original operands are the same and there is no Carry out of the most significant bit of the mantissa sum.

4. Mantissa overflow (sum) is expected to occur only when the signs of the original operands disagree and there is a Carry out of the most significant bit of the mantissa sum.

5. The sign of the sum is a function of the sign of the minuend and the complement of the sum (sign of minuend $\oplus$ complement mantissa of sum).

Since subtraction follows the same procedure as addition it is proper to say that it needs the same number of clock times to be implemented in both the 64- and 32-bit modes of operation. It is apparent, therefore, that if the option of Rounding is not used, clock times T2 and T5 in 64-bit mode and T2, T4, T6 and T8 in 32-bit mode are not used and thus the execution of addition and subtraction can be accomplished in 5 clock times in 64-bit mode and 6 clock times in 32-bit modes.

C. Multiplication

1. Introduction. Multiplication in general is the addition of partial sums which are the partial product of the multiplicand and one or more digits of the multiplier.

Since speed is important and since the multiplication time in the PE's is limited by the fact that the control unit proceeds into the next operation only when all PE's in the quadrant have completed the multiplication, acceleration of the process of multiplication is greatly dependent upon the reduction of the number of the partial sums, the speed with which these sums are formed and the speed with which these sums are added in order to give the final product of multiplicand and multiplier.

2. Implementation

a) Mantissa: In ILLIAC IV the bits of the multiplier are used in pairs to control the addition of the multiplicand as required by these pairs. These pair bits may also require subtraction or a left shift of the multiplicand.

When the pair of bits is 00 the multiplicand must be multiplied by zero; i.e., no addition is required.

When the pair of bits is 01 the multiplicand is to be multiplied by ONE (addition of the multiplicand to the partial product is required).

When the pair of bits is 10 multiplication of the multiplicand by TWO is required, which is in actuality the multiplicand itself shifted to the left one position.

When the pair of bits is 11 the multiplicand must be multiplied by THREE, which cannot be accomplished by either shifting or complementing the multiplicand. Shifting the multiplicand one position to the left has the effect of multiplying the multiplicand by TWO; shifting it two positions to the left has the effect of multiplying by four. Multiplication by three can be (and is) effected by multiplying by four in this way and then subtracting the multiplicand from this partial product.

The bits of the multiplier are recoded as X0, X1, X2, X-1 corresponding to 00, 01, 10, 11, respectively. When the recoded stage happens to be X-1 the multiplicand is just complemented and then added to the partial product. This is the same as subtracting the multiplicand once from the partial product. At a later time (in a manner that will be described subsequently), the multiplication by four is accomplished.

Suppose the selected mode is the 64-bit mode, which means that the mantissa has 48 bits. The mantissa can then be represented as

$$Y = Y_{(1+i)}\ Y_{(2+i)}\ Y_{(3+i)} \ldots\ldots\ldots\ldots Y_{(48+i)} \qquad (1)$$

where i = 15.

Assuming the mantissa is in fractional form (floating point arithmetic), and therefore the binary point is just left of the most significant bit position ($Y_{(1+i)}$), the weight of each position is 2^{-k}, where $1+i<k<48$.

The mantissa of an operand in floating point can be represented in a simpler form, as follows:

$$Y = \sum_{i=1}^{48} 2^{-i} Y_{i+15}$$

where 2^{-i} = weight of the vector Y_{i+15}

Y_{i+15} = the vector Y in the $(i+15)^{th}$ position of the register which can take on the binary values "0" or "1"

Bit 63 of RGB does not participate at all in the recoding process of the multiplier, but is used as a flag that determines whether the multiplicand is to be left in register A (if it happens to be a ONE) or that register is to be cleared (if it is a ZERO). The remaining 47 bits of the RGB (multiplier) mantissa are grouped by two from right to left and segregated into six sections (iterations), each section containing eight bits and therefore four bit pairs. Each bit pair controls a word (multiplicand times recoding of bit pair). Word 1 contains the two least significant bits of that particular section, while word 4 contains the two most significant bits of that particular section. Table 35 shows the multiplier bits that are to be recoded, with respect to word and iteration number, wherein:

	WRD#1	WRD#2	WRD#3	WRD#4	
i =	0	1	2	3	for Iteration #1
i =	4	5	6	7	for Iteration #2
i =	8	9	10	11	for Iteration #3
i =	12	13	14	15	for Iteration #4
i =	16	17	18	19	for Iteration #5
i =	20	21	22	23	for Iteration #6

Table 35. Multiplier Bits to be Recoded

ITERATION #	WORD #4		WORD #3		WORD #2		WORD #1	
	Y_{61-2i}	Y_{62-2i}	Y_{61-2i}	Y_{62-2i}	Y_{61-2i}	Y_{62-2i}	Y_{61-2i}	Y_{62-2i}
1	55	56	57	58	59	60	61	62
2	47	48	49	50	51	52	53	54
3	39	40	41	42	43	44	45	46
4	31	32	33	34	35	36	37	38
5	23	24	25	26	27	28	29	30
6		16	17	18	19	20	21	22

(For those who like closed expressions for variables, it is evident from the above that $i=4I+J-5$, when I = iteration number and J = word number. Thus:

$$\text{WORD \#J} = (Y_{71-8I-2J},\ Y_{72-8I-2J})$$

where $I = 1,2,3,4,5,6$ and $J = 1,2,3,4$.)

By taking out of the recoding scheme the 48th bit and by grouping the rest of the mantissa in groups of two bits, word 4 of the last section has bit 1 (with a value of either 0 or 1) and a zero (in bit 2). This zero is essential to handle a carry into word 4 from the previous word; if then bit Y_{y2-2i} of this word happens to be a ONE, word 4 will be 10 which means that there will be a request for addition two times of the multiplicand to the partial product. In this way there will be no way to have a carry out of the word 4 of the last section (remember bit Y_{62-2i} of last word corresponds to bit 16 of the register). Since bit numbers are assigned to each word and not bit location it can be said that the general recoding stage (four such stages per section) is a function of the pair of bits of each word and the carry into the stage.

In this notation the output of recoding stage RS_{47-21} can be 0, X1, X2, X-1.

Consider the pair of bits Y_{61-2i} and Y_{62-2i} with carry coming into the stage in the first step at a particular section (Table 36). In the second step not only that particular word, but any carry into the stage must be considered.

Table 36. Recoding Multiplier Scheme

STEP	Y_{61-2i}	Y_{62-2i}	CARRY IN	RS_{47-2i}	CARRY OUT
1	0	0	0	0	0
	0	1	0	X1	0
	1	0	0	X2	0
	1	1	0	X-1	1
2	0	0	1	X1	0
	0	1	1	X2	0
	1	0	1	X-1	1
	1	1	1	0	1

The recoding stage specifies how much the multiplicand must be multiplied (how many times the multiplicand must be added or subtracted from the partial product). From Table 36, the equations for the carry out from a particular stage and the equations for the (47-2i)th recoding stage corresponding to each of the four values (X0, X1, X2, X-1) that this stage can take can be derived.

$$\text{RS}_{47-2i} = \text{X0} = \overline{\text{Y}}_{61-2i}\cdot\overline{\text{Y}}_{62-2i}\cdot\overline{\text{Carry In}} + \text{Y}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\text{Carry In}$$

$$\text{RS}_{47-2i} = \text{X1} = \overline{\text{Y}}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\overline{\text{Carry In}} + \overline{\text{Y}}_{61-2i}\cdot\overline{\text{Y}}_{62-2i}\cdot\text{Carry In} \qquad (2)$$

$$\text{RS}_{47-2i} = \text{X2} = \text{Y}_{61-2i}\cdot\overline{\text{Y}}_{62-2i}\cdot\overline{\text{Carry In}} + \overline{\text{Y}}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\text{Carry In}$$

$$\text{RS}_{47-2i} = \text{X-1} = \text{Y}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\overline{\text{Carry In}} + \text{Y}_{61-2i}\cdot\overline{\text{Y}}_{62-2i}\cdot\text{Carry In}$$

$$\text{Carry Out} = \text{Y}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\overline{\text{Carry In}} + \text{Y}_{61-2i}\cdot\overline{\text{Y}}_{62-2i}\cdot\text{Carry In}$$

$$+ \text{Y}_{61-2i}\cdot\text{Y}_{62-2i}\cdot\text{Carry In}$$

$$= \text{Y}_{61-2i}\ \text{Y}_{62-2i}\ (\overline{\text{Carry In}} + \text{Carry In}) + \text{Y}_{61-2i}(\text{Carry In})\overline{\text{Y}}_{62-2i}$$

$$= \text{Y}_{61-2i}(\text{Y}_{62-2i} + \overline{\text{Y}}_{62-2i}\ \text{Carry In}) \qquad (3)$$

(because of the Boolean Identity $A + \overline{A} = 1$). From the Boolean Identity $A + \overline{A}B = A + B$:

$$\text{Carry Out} = \text{Y}_{61-2i}\cdot\text{Y}_{62-2i} + \text{Y}_{61-2i}\cdot\ \text{Carry In} \qquad (4)$$

Further study of Table 35 reveals that in the initial step there will be no case where a carry can come into the first recoding stage and also no carry can come out of the last stage of the last step.

The carry into a recoding stage is interpreted as a request for an addition of four times the multiplicand to the partial product in the next step, because at the current step, when the carry out is generated, the multiplicand is subtracted once from the partial product. Therefore any time the stage (or word) comes out to be 11 the multiplicand is subtracted from the partial product and a request is made for the multiplicand to be added four times in the next step (Iterative Cycle).

Every time the word is recoded the multiplicand is shifted a number of positions to the left but, at that point, it is necessary to determine which recoding stage is in effect (i.e., which multiple is used). If it is at X1 or X2 and word 1 of the first iteration (steps) is being dealt with, the multiplicand is placed into the pseudo adder tree (PAT) but shifted left one or two positions, respectively. Of course if it is X0 there is no problem, because that means that there is no request for addition of the multiplicand. If it is X-1 then the multiplicand is subtracted from the partial product. In the next step things are different because there may be a carry into the stage. If the stage is X0 which means no request for addition of the multiplicand, the carry in forces the multiplicand to be added one time to the partial product because carry into the stage is equivalent to a request for addition of the multiplicand four times. This is accomplished by shifting the multiplicand two positions relative to where the multiplicand was placed in the previous word. If the stage is X1, the appearance of the carry forces the multiplicand to be added two times. This is done by shifting the multiplicand two positions to the left compared to where the multiplicand was placed in the previous word. If the stage happens to be X2 which normally requires addition of the multiplicand two times, now the appearance of the carry forces it to look like X-1 which means subtraction of the multiplicand once from the partial product and a request to the next step for an addition of the multiplicand four times.

If the stage is X-1, which normally requires the addition of the multiplicand three times, the carry into the stage forces no operation at this time, but a request for the addition of the multiplicand four times to the partial product in the next step is made.

Table 35 shows that there are six sections (iterations) each of which contains four words. Each section requires one clock time for the formation of the partial product. Since there are eight bits in each section or four words of two bits each, the multiplier can be represented in terms of recoding stages as follows:

$$Y = Y_{63} \cdot 2^{-48} + \sum_{i=0}^{5} (RS_{47-8i} \cdot 2^{-(47-8i)} + RS_{45-8i} \cdot 2^{-(45-8i)}$$

$$+ RS_{43-8i} \cdot 2^{-(43-8i)} + RS_{41-8i} \cdot 2^{-(41-8i)}) \qquad (5)$$

Therefore the final product of the multiplicand X and multiplier Y is given by [4]:

$$XY = X [Y_{63} \cdot 2^{-48} + \sum_{i=o}^{5} (RS_{47-8i} \cdot 2^{-(47-8i)} + RS_{45-8i} \cdot 2^{-(45-8i)}$$

$$+ RS_{43-8i} \cdot 2^{-(43-8i)} + RS_{41-8i} \cdot 2^{-(41-8i)})] \qquad (6)$$

Dealing with positive numbers there are some cases which must be thoroughly investigated. What happens if the two bits of word 1 of the first section of the multiplier are 11? In this case the multiplicand is subtracted once from the partial product and a request is sent to the next step to add the multiplicand four times to the partial product. This subtraction should normally force the partial product to decrease more and more. If the multiplier happens to be such that the next pairs of bits are 10 the carry from the first pair will force the partial product to decrease more and more until it may become negative and also an extra clock time would be required to perform the request of adding the multiplicand four times to the negative partial product. This extra clock time is not desirable at all because of its costs in terms of speed. In order to cure this problem a ZERO is forced in front of bit position 16 so that if bit position 16 has a ONE and there is a carry from the previous pair, this carry will force the last pair to look like 10 which indicates addition of the multiplicand two times to the negative partial product. At this point, being in the last step of the last section, the multiplicand is shifted two places to the left compared to where the multiplicand was placed in the immediately previous level of the pseudo adder and, therefore, this number has a greater weight than the negative partial product, which makes the final product a positive number.

Since multiplication has been defined as successive additions of multiplicand, it is evident that the use of a very fast adder is critically essential. However, instead of using the old scheme of addition of two numbers in order to produce the sum, a new adder has been designed which adds three numbers and produces, instead of a single sum, two numbers (sum and carry) which are equivalent to the three numbers being added. This adder is called PSEUDOADDER (or Carry Save Adder).

The pseudoadder has three levels and can accommodate 56 bits. Its high speed is due to the fact that there is no carry propagation but instead, the carry constitutes one of the three inputs to each level of the pseudoadder, every level having as output the sum and a carry, which along with the multiple of the multiplicand, constitutes the three inputs to the next level.

The output of the third level of the pseudoadder is applied to the Carry Propagating Adder (used, in this case, as a Carry Save Adder) which along with the multiple of the multiplicand as a result of the recode stage (word 4) produces the final sum and carries of a particular iteration. The first 48 bits of the sum are stored in A register which holds the partial product, while the other 8 least significant bits are placed into B register. The B register is shifted through the BSW in every clock time 8 bits to the right end off. The carries from the carry propagating adder are stored in C register to be used as one of the inputs into the first level of the pseudoadder in the next iteration.

The correction bits (Table 37) are necessary in all levels of the PAT and for WORD #4 in the CPA, because, if any of the words happen to require the multiplicand to be multiplied by three, then instead of multiplication a subtraction of the multiplicand from the current partial product is performed. Since the sum is taken in 2's complement form the multiplicand is complemented (1's complement) and added to the partial sum. Also an extra bit is added to the least significant bit (bit 71). This extra bit is forced into bit position 71 of the PAT or CPA depending on which word requires such an addition, by the signal:

PW1-C----1 for WORD #1
PW2E-X-1-1 for WORD #2
PW3E-X-1-1 for WORD #3
PW4-C---1 for WORD #4

Details concerning the use of the correction bits are given later in Example #2.

In the process of multiplication it may happen that the multiplicand is ZERO and the multiplier is such that, in one of the iterations, one or more of its words calls for a multiplication by three. The expected result is ZERO but in this case it is necessary to get rid of the carry which resulted from the extra bit inserted in bit location 71 in order to obtain the result in 2's complement. There is a signal called PW1-K----1 which inserts eight ONES in the most significant bit locations of the PAT thus insuring that the final product will contain all ZEROS. This signal requires special logic, in order to cover all the possible cases and depends mainly on whether or not the multiplicand is normalized. The equation for this signal (K function) is as follows (i = 1,2,---,6):

$$\begin{aligned}\text{PW1-K{-}{-}{-}{-}1}_{Ti} = {} & R_{16}\ [\text{WORD \#4x-1} + \text{WORD \#4x2}\ (\text{WORD \#1x-1}+ \\ & +\text{WORD \#2x-1} + \text{WORD \#3x-1}) + \text{PW1-K{-}{-}{-}{-}1})_{To}\ ] \\ & +R_{16}\ [\text{WORD \#4x-1} + (\text{WORD \#1x-1} + \text{WORD \#2x-1} \\ & +\text{WORD \#3x-1}) + \text{PW1-K{-}{-}{-}{-}1})_{To}\ ] \qquad (7)\end{aligned}$$

Table 37. Definition of Signals Applied to PAT

SIGNAL NAME	DESCRIPTION
PW1 -WXX--1	WORD #1 bit XX from MSG into the first level of PAT
PW2 -WXX--1	WORD #2 bit XX from MSG into the second level of PAT
PW2 -WXX--1	WORD #3 bit XX from MSG into the third level of PAT
PW1EX1I X1Ø X-1 X2	
PW2EX1I X1Ø X-1 X2	These signals come from the MDG and are applied into the MSG through the CTL's. They represent the recoding scheme i.e. the amount of multiplication of multiplicand,
PW3EX1I X1Ø X-1 X2	
PW4EX1I X1Ø X-1 X2	
* PW1-C----1	Correction bits enable when WORD #1= -1
PW2ECX-1-1	Correction bits enable when WORD #2= -1
PW3ECX-1-1	Correction bits enable when WORD #3= -1
** PW4-C----1	Correction bits enable when WORD #4= -1
PPW-WXX--1	PAT SUM for bit XX of the 3rd level of PAT

* PW1-C----1 is used also to insert an extra bit ("1") at bit location 71 when WORD #1=X-1

** PW4-C----1 is used to insert an extra bit ("1") at bit location 71 when WORD #4=X-1

Table 37. (Continued) Definition of Signals Applied to PAT

P51-WXX--1	Carry out of 1st Level of PAT for bit XX
P52-WXX--1	Carry out of 2nd Level of PAT for bit XX
P5W-WXX--1	Carry out of 3rd Level of PAT for bit XX
P3R-WXX--1	RGC bit XX into the first Level of PAT
PAW-WXX--	RGA bit XX into the first Level of PAT
PW2E-1----	Enable of an extra bit at bit location 71 in the Second Level of PAT when WORD #2= -1
PW3E	Enable of an extra bit at bit location 71 in the third level of PAT when WORD #3= -1
PW1-K----1	A special enable of eight ONES to insure that when the multiplicand is ZERO, the result as U be ZERO too. See more details below.

It is evident that in the first iteration, if the multiplicand is not normalized and bit 16 happens to be ZERO, this signal is true and therefore will insert eight ONES (only if any one of the words calls for multiplication by three).

During the multiplication process, in addition to the registers shown on Figure 56, there are two more units used. These units are the logic unit (LOG) which feeds the barrel switch and the barrel switch and its associated controls which perform the shifting of 40 bits, 8 bits at a time, thus returning to B register only 32 bits. The reason for this is the fact that at the conclusion of every iteration, a space of 8 bits is needed (bits 16-23) to hold the extra least significant bits of the 56-bit partial product. However, at the beginning of each iteration the 8 least significant bits of B register are recoded and thus are not needed any more. Thus, from the end of clock time T2 until the end of clock time T6 the barrel switch receives 40 bits (once in every clock time) and returns to B register 32 bits (Figure 4 in Volume 1).

For an illustration of the multiplication process consider two operands of 13 bits each. In order to understand the process better assume that the two operands have filled in the registers A and B in such a way that the least significant bit of the 13 bit operands is at bit location 63 (Example #1 - page 187).

At the start the multiplicand is in register A and during T1 it is transferred into R. Bits 55 through 62 of the multiplier, which are in B register, have been recoded and since bit location 63 of the multiplier is a ONE the multiplicand in A register remains unchanged.

The multiplicand being in R register is applied to multiplicand select gates (MSG) and being ANDed with the selection controls leaves the MSG shifted to the left in the following manner. The multiplicand from A register occupies 48 bit positions (in 64-bit mode), the output of MSG is shifted to the left one position because word 1 is recoded as X1 and since this is the first word of the first section there is no carry to be added to them.

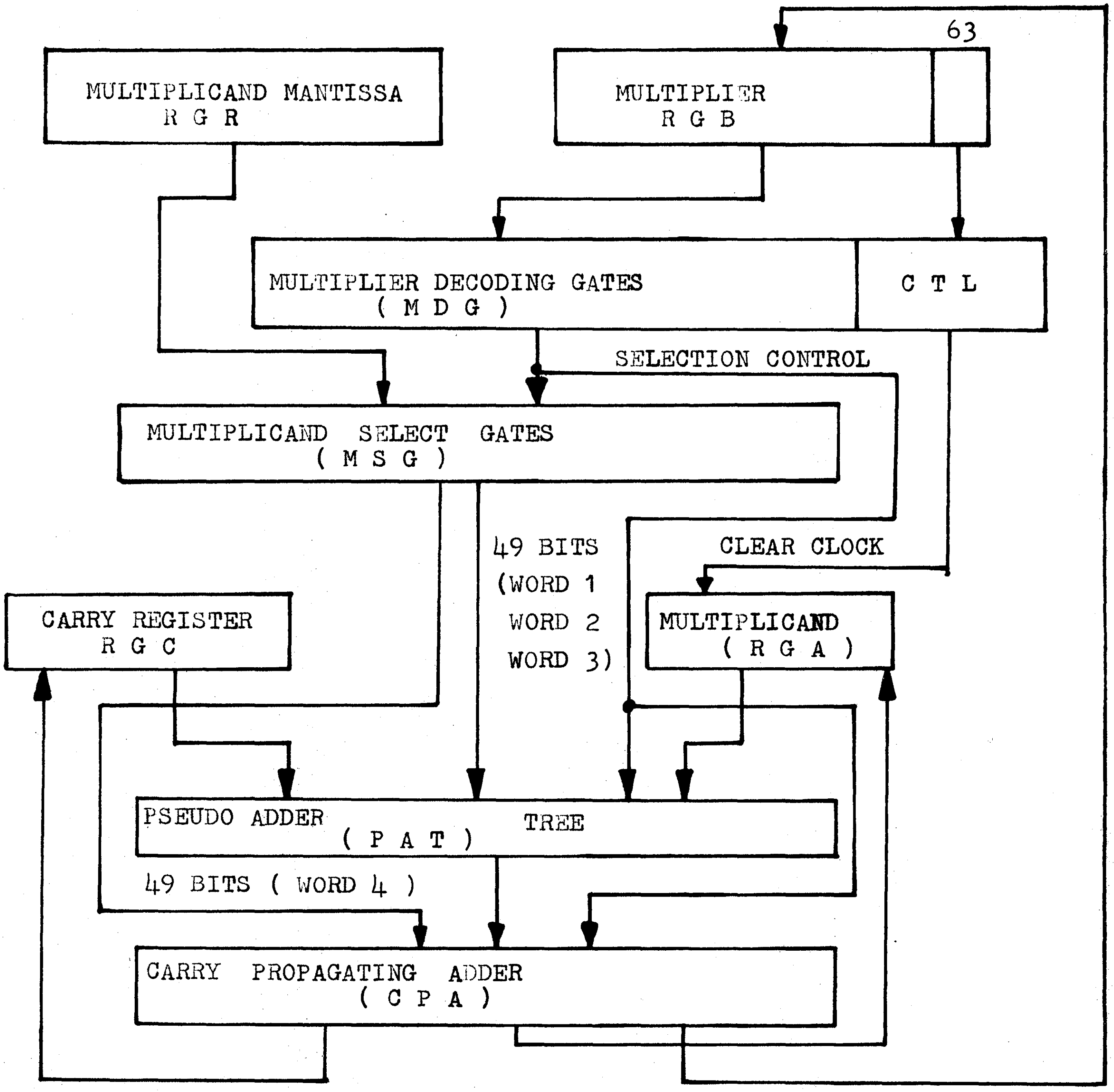


FIGURE 56. BLOCK FUNCTIONAL DIAGRAM OF THE REGISTERS PARTICIPATING IN MANTISSA MULTIPLICATION

The sum and carries out of the first level of the pseudoadder are placed into the second level of the pseudoadder. There, the output of the MSG corresponding to word 2 is also applied, but shifted just two positions to the left as compared to the output of MSG corresponding to word 1. The output of the second level (and therefore input to the third level of the pseudoadder) is the sum and carry which are added to the output of the MSG corresponding to word 3, which is shifted two positions to the left compared to that corresponding to word 2 of the second level of the pseudoadder. The output of the third level of the pseudoadder (sum and carry) is supplied to the carry propagating adder (CPA) along with the output of the MSG corresponding to word 4.

In this first iteration the sum, consisting of 56 bits, is placed into registers A and B in the following manner. The 48 most significant bits of the sum from the CPA are placed into A register (bits 16-63). The remaining 8 least significant bits of the sum are placed into bit positions 16 to 23 of B register (Figure 57). This space (16 to 23 bit positions of B register) is available because in the clock time 1 the mantissa of B register is shifted end off to the right. The shifted off 8 bits of the multiplier are not needed any more because they have already been recoded for the first iteration. At this time the carries of bit locations 65 to 72 are held in C register until the next clock time. At that time they are forced into the CPA at bit positions 72 to 79 to be added with the 8 bits of the sum being stored temporarily in bit positions 16 to 23 of B register. This new sum is brought back to B register, but in bit locations 24 to 31 because the bit locations 16 to 23 are needed for the 8 bit partial sum of the next iteration. At this point, if there is a carry because of the addition of the 8 bits of partial sum and carries, the carry is placed in bit position 72 of register C to be added to the partial sum of the next iteration in CPA bit positions 72 to 79 along with the rest of the 8 bit carries of this iteration. If there is a carry in bit position 64 of register C, this carry is kept in a special latch and placed in bit position 79 of CPA to be added to the sum of the next iteration in CPA bit positions 72 to 79 along with the rest of the 8 bit carries of this iteration. The reason for doing this is that the carry at bit position

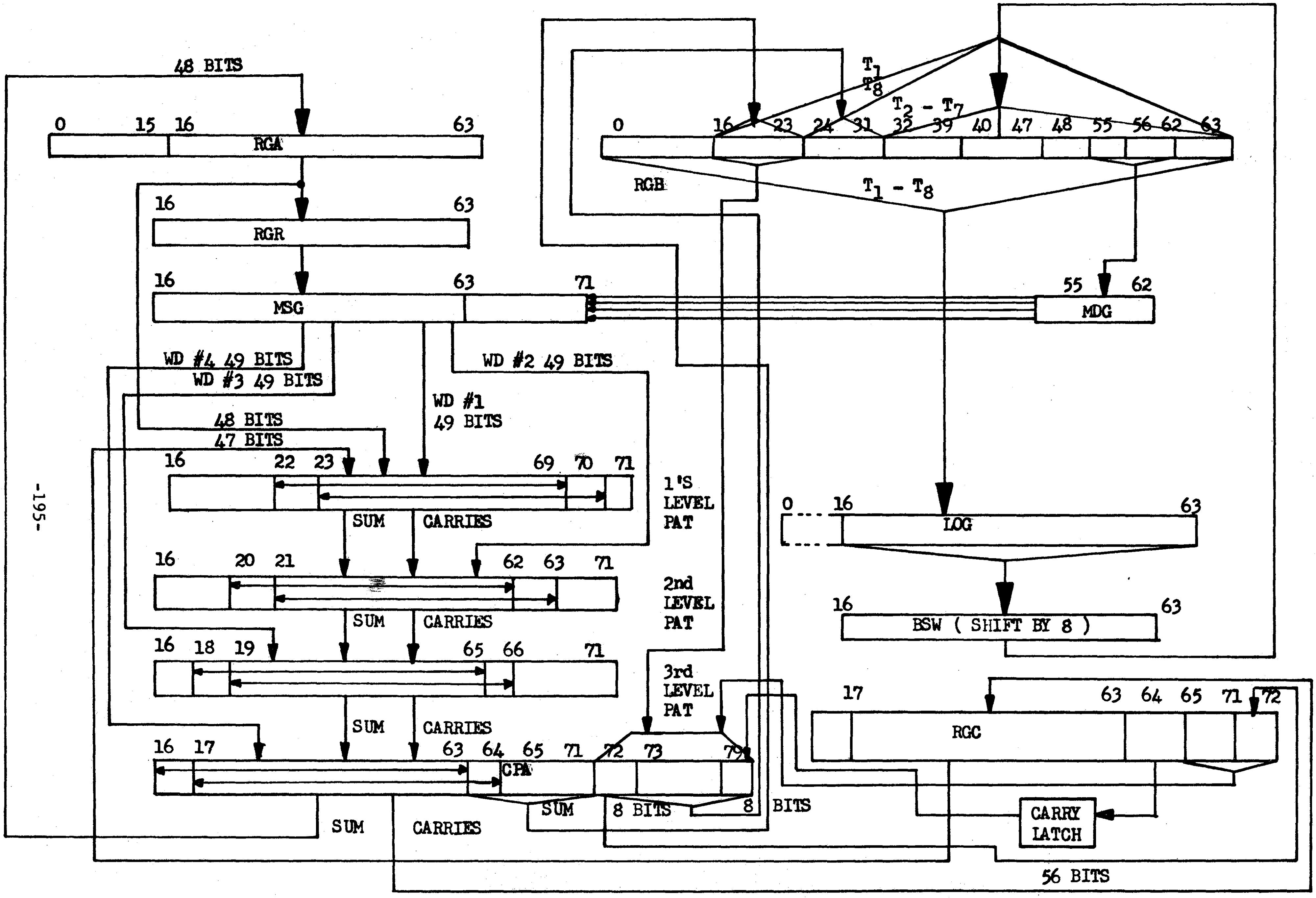


FIGURE 57. MULTIPLICATION PROCESS (MANTISSA)

64 is the least significant bit of the carries of the next iteration and the most significant bit of the carries (8) which are added to the partial sum of the present iteration.

It is evident that in every clock time the mantissa of B register is shifted off 8 bits so that it will provide, by the end of the last iteration, the space for the 48 least significant bits of the final product.

It was previously stated that every time one iteration is completed the partial product is 56 bits long. This is due to the following reasons. The multiplicand (Figure 58), due to the least significant bit of the multiplier (bit 63), will occupy bits 24 to 71 of pseudoadder. If then, word 1 is X1 the multiplicand will occupy bits 23 to 70 of 22 to 69 if word 1 is X2.

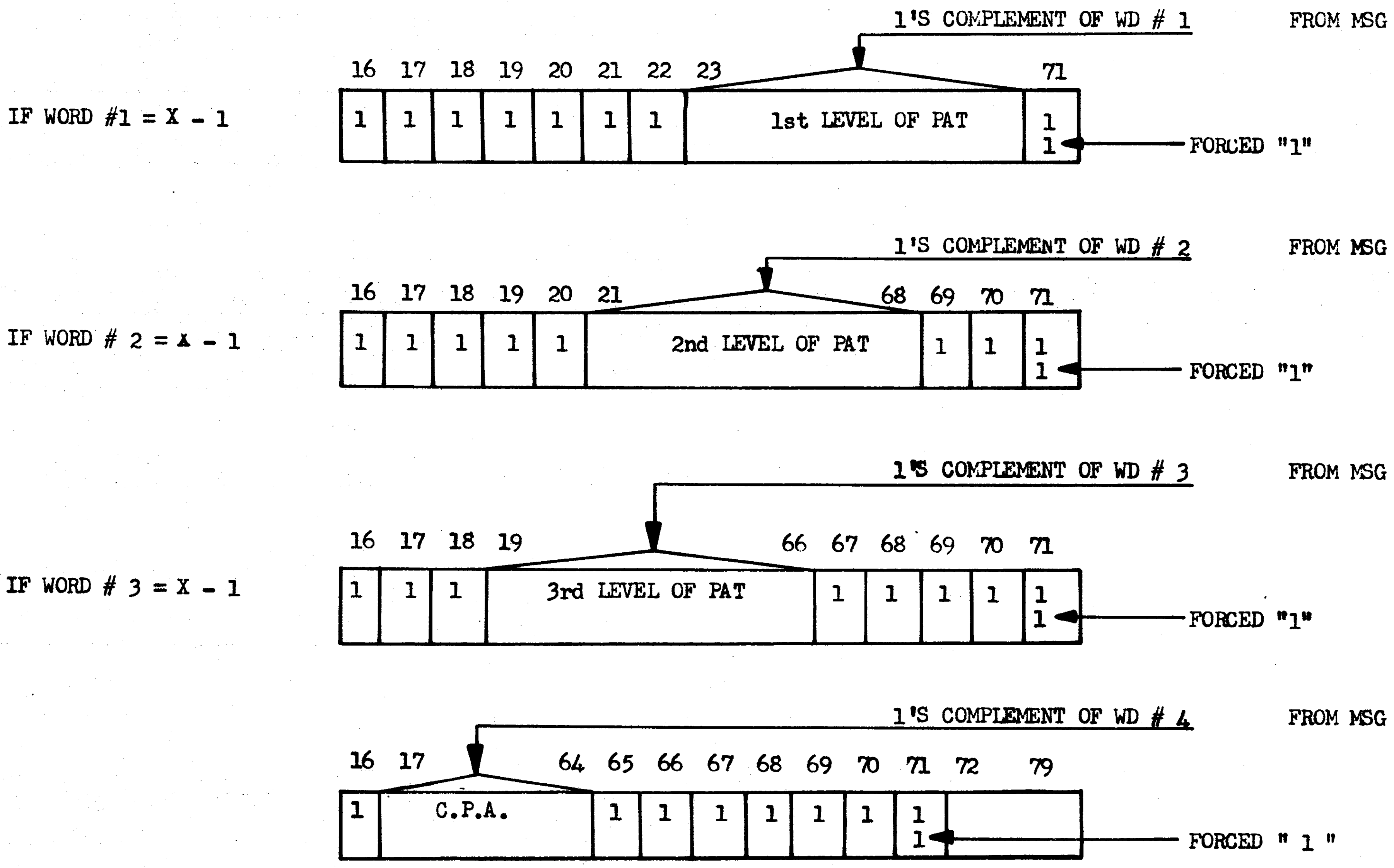
If word 2 is X1 the multiplicand will occupy bits 21 to 68; if word 2 is X2 it will occupy bits 20 to 67. If word 3 is X1 the multiplicand will occupy bits 19 to 66 or, if it is X2, 18 to 65. If word 4 is X1 the multiplicand will occupy bits 17 to 64 or 16 to 63 if word 4 is X2 in the CPA.

Therefore the multiplicand related to word 1, 2, 3 or 4 can be 49 bits long because, if word 1, 2, 3 or 4 is X2, the multiplicand is shifted one position to the left more than if the word is an X1. So wiring for 49 bits from the MSG to the pseudoadder (word 1, 2, 3) and to the CPA (word 4) is provided. In case any of the words (word 1, 2, 3, 4) is recoded to be X1 the most significant bit (the extra one provided) is brought into PAT and CPA as a ZERO.

Example #1

Given the multiplicand X = 1 0 1 0 1 0 1 0 1 0 1 0 1 and
the multiplier Y = 0 0 0 0 0 1 0 1 0 1 0 1 1

Find the final product of the first iteration.



NOTE: SEE EXAMPLE #2 ON A SUBSEQUENT PAGE FOR AN EXPLANATION OF THE "FORCED ONES".

FIGURE 58. CORRECTION BITS FOR MANTISSA MULTIPLICATION

Solution

63

RGA	1 0 1 0 1 0 1 0 1 0 1 0 1
RGC	0 0 0 0 0 0 0 0 0 0 0 0 0
RGB	0 0 0 0 0 1 0 1 0 1 0 1 1
RGR	1 0 1 0 1 0 1 0 1 0 1 0 1

WORD1 = X1
WORD2 = X1
WORD3 = X1
WORD4 = X1
Recoding Stage

RGA ·B63 (T1 only)	1 0 1 0 1 0 1 0 1 0 1 0 1	
RGC (ZERO INITIALLY)	0 0 0 0 0 0 0 0 0 0 0 0 0	CSA1
RGR.WORD1	1 0 1 0 1 0 1 0 1 0 1 0 1	

SUM	1 1 1 1 1 1 1 1 1 1 1 1 1 1	
CARRY	0 0 0 0 0 0 0 0 0 0 0 0 0 0	CSA2
RGR·WORD2	1 0 1 0 1 0 1 0 1 0 1 0 1	

SUM	1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 1	
CARRY	0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0	CSA3
RGR·WORD3	1 0 1 0 1 0 1 0 1 0 1 0 1	

Inputs to PSEUDOADDER

SUM	1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 1 1
CARRY	1 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0
RGR· (WORD 4)	1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0

PARTIAL SUM	1 0 0 1 1 1 1 0 1 0 1 0 ⋮ 1 0 0 0 0 1 1 1
CARRY	1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0

C P A (Used as CSA)

16 17 18 19 20 21 22 23

RGA	1 0 0 1 1 1 1 0 1 0 1 0	1 0 0 0 0 1 1 1	RGB

RGC	1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0

To see the result at this point, it is necessary to (carry propagate) add the carry to the partial sum; which gives:

RGA & RGB	1 0 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1
+ RGC	0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0
FINAL SUM	1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1

$$\begin{array}{r}
524288 \\
262144 \\
131072 \\
000000 \\
000000 \\
008192 \\
004096 \\
002048 \\
001024 \\
000512 \\
000256 \\
000128 \\
000064 \\
000000 \\
000000 \\
000000 \\
000004 \\
000002 \\
\underline{000001} \\
\hline\hline
933831_{10}
\end{array}$$

To check the result the equivalent decimal numbers of multiplicand and multiplier are found and multiplied to see if the result is the same as that found by the process of iteration through the computer.

MULTIPLICAND = 1 0 1 0 1 0 1 0 1 0 1 0 1

MULTIPLIER = 0 0 0 0 0 1 0 1 0 1 0 1 1

1 2 8
0 0 0
0 3 2
0 0 0
0 0 8
0 0 0
0 0 2
0 0 1

$1\,7\,1_{10}$

4 0 9 6
0 0 0 0
1 0 2 4
0 0 0 0
0 2 5 6
0 0 0 0
0 0 6 4
0 0 0 0
0 0 1 6
0 0 0 0
0 0 0 4
0 0 0 0
0 0 0 1

$5\,4\,6\,1_{10}$

$$5461_{10} \times 171_{10} = 933831_{10}$$

Example #2

Given the operands:

X = 110000001

Y = 010111101

Find the product (XY).

Solution

a. The operands are placed in the proper registers:

X → RGA and RGB	(Multiplicand)
Y → RGB	(Multiplier)

b. Recode the multiplier

1. 63 bit position of multiplier = 1
2. WD#1 = X2
3. WD#2 = X-1 (forces a carry into WD#3)
4. WD#3 = X2
5. WD#4 = X1

c. WD#2 = -1 means that the multiplicand has to be subtracted from the partial product as follows:

1. Place the 1's complement of the multiplicand in the second level of the pseudoadder tree starting at bit position 68 towards the left up to bit position 21.

2. Insert 1's from bit position 71 through 69 and from 59 through 55*.

3. Force a one into bit position 71 of the pseudoadder tree to form the 2's complement of the multiplicand**.

* See page 203

** See page 204

d. In the final product (sum and carry) neglect the end around carry because the product is in 2's complement.

Bit Position	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	
RGA									1	1	0	0	0	0	0	0	1	PAT
CARRY							0	0	0	0	0	0	0	0	0	0	0	
RGR$_{(WD\#1)}$(X2)							1	1	0	0	0	0	0	0	1			
P. SUM$_{(1)}$							1	1	1	1	0	0	0	0	1	0	1	
CARRY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
RGR$_{(WD\#2)}$(X-1)	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	
P. SUM$_{(2)}$	1	1	1	1	1	0	1	0	0	0	1	1	1	0	0	1	1	
CARRY	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	
RGR$_{(WD\#3)}$(X2)			1	1	0	0	0	0	0	0	1							
P. SUM$_{(3)}$	1	1	0	0	1	0	0	1	1	0	0	1	1	1	0	0	1	CPA
CARRY		1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	
RGR$_{(WD\#4)}$(X1)		1	1	0	0	0	0	0	0	1								
P. SUM$_{(4)}$	1	1	0	0	1	1	0	1	1	0	0	1	1	1	1	0	1	
CARRY	1	1	0	0	0	0	0	0	1									
FINAL PRODUCT	1	0	0	0	1	1	1	0	0	0	0	1	1	1	1	0	1	
	1 (neglected)																	

e. To check the result

1. Final product $= 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 1_2$

$$
\begin{array}{rl}
= & 0\ 0\ 0\ 0\ 1 \\
 & 0\ 0\ 0\ 0\ 4 \\
 & 0\ 0\ 0\ 0\ 8 \\
 & 0\ 0\ 0\ 1\ 6 \\
 & 0\ 0\ 0\ 3\ 2 \\
 & 0\ 1\ 0\ 2\ 4 \\
 & 0\ 2\ 0\ 4\ 8 \\
 & 0\ 4\ 0\ 9\ 6 \\
 & 6\ 5\ 5\ 3\ 6 \\
\hline
 & 7\ 2\ 7\ 6\ 5_{10}
\end{array}
$$

2. Multiplicand $= 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1_2$

   $= 3\ 8\ 5_{10}$

3. Multiplier $= 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1_2$

   $= 1\ 8\ 9_{10}$

4. 385 x 189 $= 7\ 2\ 7\ 6\ 5_{10}$

In general, whenever the bit pair is 11 (recoding stage) the 1's complement of the multiplicand is placed in the proper level of the PAT and at the bit position which normally corresponds to WD#1 = X1 (i=1,2,3,4). Since the partial product is 56 bits long, as previously explained, 8 ones are inserted into the same level as follows:

If WD# 1 = X-1 (a) Place 1's complement of multiplicand at bit positions 70 through 23.

(b) Insert a ONE at bit position 71 and 7 ONES in bit position 22 through 16.

If WD# 2 = X-1 (a) Place 1's complement of multiplicand in bit positions 68 through 21.

(b) Insert 3 ONES at bit position 71 through 69 and 5 ONES in bit position 20 through 16.

If WD# 3 = X-1 (a) Place 1's complement of multiplicand in bit positions 66 through 19.

(b) Insert 5 ONES at bit positions 71 through 67 and 3 ONES in bit positions 18 through 16.

If WD# 4 = X-1 (a) Place 1's complement of multiplicand in bit positions 64 through 17.

(b) Insert 7 ONES in bit positions 71 through 65 and a ONE in bit position 16.

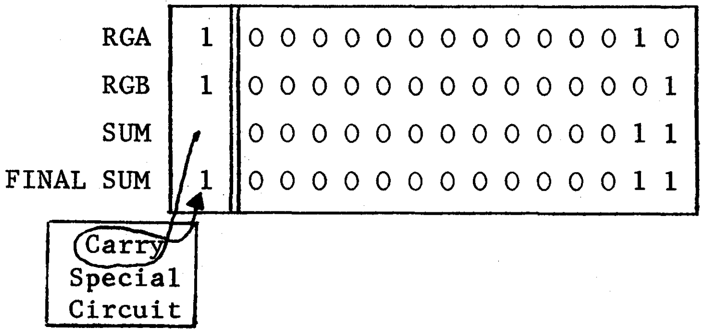
The reason for doing the above is that out of the 56 bits of the PAT, only the 48 bits of the multiplicand have been put in the 1's complement form but from bit 71 through 16 there are 8 bits (unoccupied) which look like zeros. Taking the 1's complement of the whole 56 bits, forces these 8 bits to become ONES.

By forcing a carry into the least significant bit of PAT (71) the partial sum in that particular level of PAT takes the form of the 2's complement. This is done so that in case there is an end around carry in the partial product, which is in 2's complement, the end around carry can be neglected.

b) Exponent: In multiplication, the exponents of the operands are added. There are a few points which should be brought up before attempting to compute the exponent.

The input to the CPA that is not enabled appears as all ONES. For this reason, any carries from Section #1 (Bit 16-31) into Section #4 (Bits 64-79) should be inhibited. The exponents of both operands are added, but the sign bit of the exponent is determined by the appearance or absence of a carry. This carry is placed in a special circuit and is brought back to the sign bit position as a 1 or 0 as illustrated by the following examples.

A



RGA	1	0 0 0 0 0 0 0 0 0 0 0 0 1 0
RGB	1	0 0 0 0 0 0 0 0 0 0 0 0 0 1
SUM		0 0 0 0 0 0 0 0 0 0 0 0 1 1
FINAL SUM	1	0 0 0 0 0 0 0 0 0 0 0 0 1 1

Aexp = +2

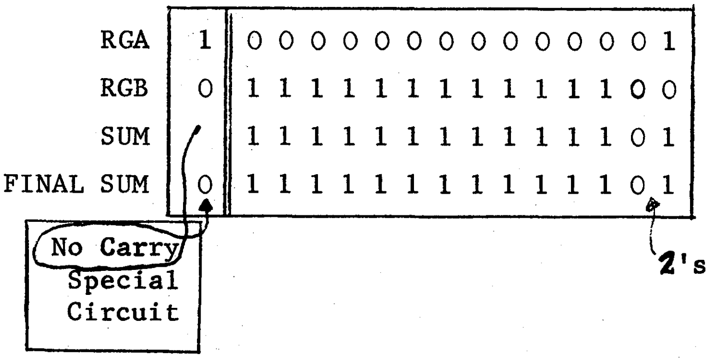
Bexp = +1

CPA

Sum = +3

Sum = +3

B



RGA	1	0 0 0 0 0 0 0 0 0 0 0 0 0 1
RGB	0	1 1 1 1 1 1 1 1 1 1 1 1 0 0
SUM		1 1 1 1 1 1 1 1 1 1 1 1 0 1
FINAL SUM	0	1 1 1 1 1 1 1 1 1 1 1 1 0 1

2's complement form

Aexp = +1

Bexp = -4

CPA

Sum = -3

Sum = -3

In this way, the carry from the most significant bit of the exponent (Bit 65) does not affect the sign of the mantissa. The mantissa sign being a function of $A_o \oplus B_o$ and the inverse of the enable for computation of the mantissa sign (FYEMDOSGNT = 1) depends on the sign of both operands as the following examples illustrate.

1) if $A_o = 0 = +$

if $B_o = 0 = +$

The mantissa sign must = +.

$(0 \cdot 1 + 1 \cdot 0) \cdot 1 = 0 = +$

2) if $A_o = 1 = -$ $A_o = 0 = +$

OR The mantissa sign must = −.

if $B_o = 0 = +$ $B_o = 1 = -$

$(1 \,.\, 1 + 0 \,.\, 0) \,.\, 1 = 1 = -$

3) if $A_o = 1 = -$

The mantissa sign must = +.

if $B_o = 1 = -$

$(1 \,.\, 0 + 0 \,.\, 1) \,.\, 1 = 0 = +$

MULTIPLICATION IN 64-BIT MODE

CLOCK TIME T1--Figure 59

Add Exponents - Recode 8 Bits of Multiplier Mantissa for First Iteration

1. Enable Registers A & B (bits 0-63).
2. Enable Register A into R (bits 16-63).
3. Enable Register R (bits 16-63).
4. Enable clear and load clocks into inner and outer word of register R.
5. Enable clear and load clocks in Register C.
6. Enable clear and load clocks to underflow latch.
7. Enable exponent underflow into mode register (D) if not normalizing and ACAR(9).
8. Enable overflow into Register D.
9. Inhibit clear clocks into Register D.
10. Enable clear and load clocks into F if floating point.
11. Enable clear clocks into Outer sign and exponent of register A (0-7) if E=1.
12. Enable clear clocks into Inner sign and exponent of register A (8-15) if E_1=1.
13. Enable clear clocks into inner mantissa of Register A (16-39) if E_1=1 and B63=0.
14. Enable clear clocks into outer mantissa of Register A (40-63) if E=1 and B63=0.
15. Enable load clocks into outer sign and exponent of Register A (0-7) if E=1.
16. Enable load clocks into inner sign and exponent of Register A (8-15) if E_1=1.

17. Enable clear clocks into inner and outer mantissa of Register B (16-63).

18. Enable load clocks into Inner and outer mantissa of Register B (16-63).

19. Enable the content of Register B into LOG.

20. Enable LOG (16-63) into barrel switch.

21. Enable a shift to the right by 8 into the barrel switch controls.

22. Enable Barrel Switch (16-63) into Register B (16-63).

23. Enable the complement output of the outer exponent of Register B (1-7)

24. Enable the complement output of the inner exponent of Register B (9-15)

For Fixed Point Operand

25. Enable Registers A and B (1-15) into CPA (65-79). B only if floating point.

26. Enable the bit carries into CPA (64-79) if floating point.

27. Inhibit section carries from Section 1.

28. Enable the outer sign and exponent of CPA into Register A (0-7).

29. Enable the inner sign and exponent of CPA into Register A (8-15).

30. Restore the sign of Register A (0) if unsigned.

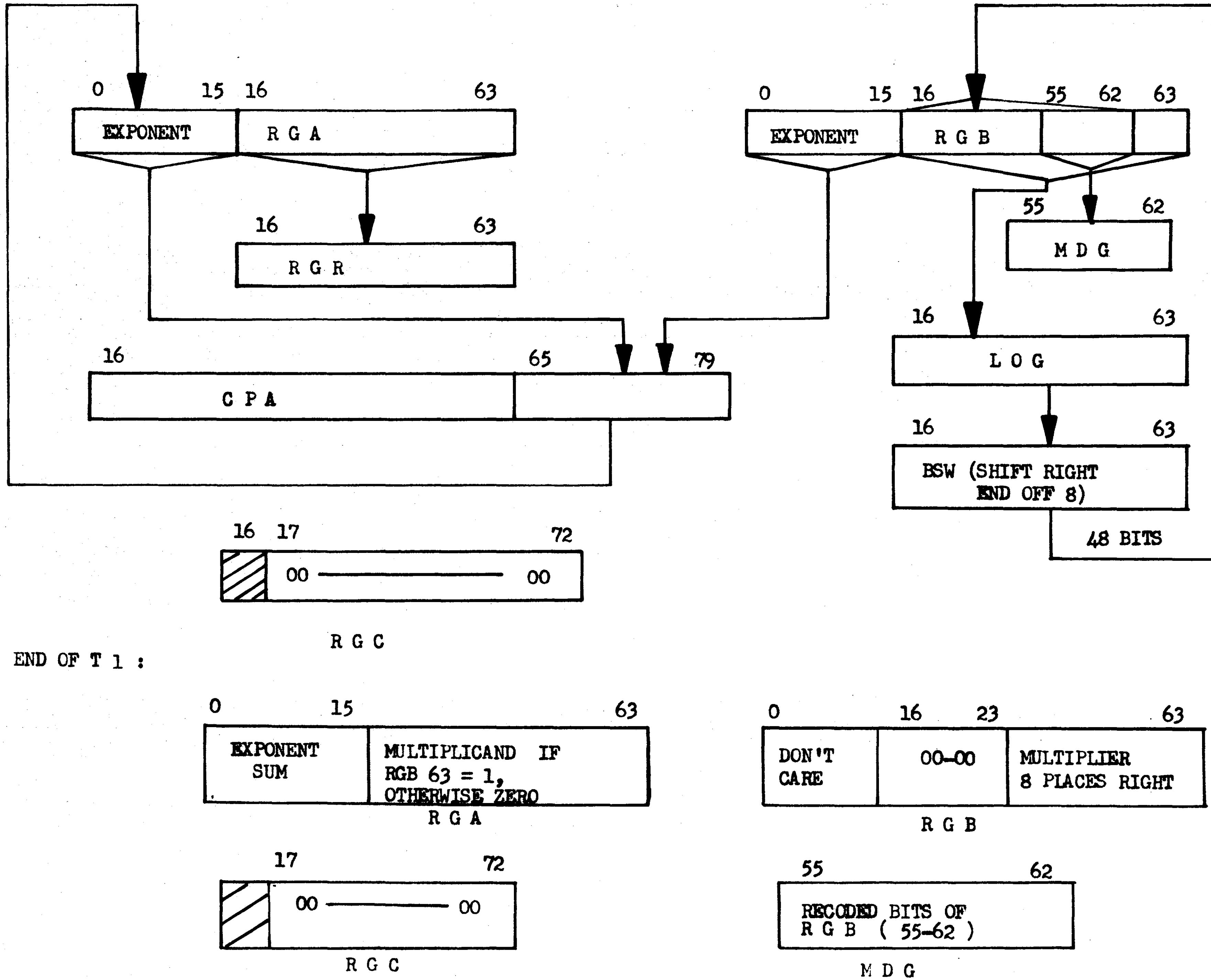


FIGURE 59. CLOCK TIME T 1

CLOCK TIME T2--Figure 60

First Iteration - Recode 8 Bits of Multiplier Mantissa for Second Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) to LOG (16-63).
5. Enable LOG (16-63) into BSW (16-63).
6. Force a shift right end off 8 places into the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable CPA sum (16-63) to RGA (16-63).
9. Enable CPA sum (64-71) to RGB (16-23).
10. Enable CPA sum (72-79) to RGB (24-31).
11. Enable BSW (32-63) into RGB (32-63).
12. Enable WD4 (16-63) into CPA (16-63).
13. Enable WD4 (64-71) into CPA (64-71).
14. Enable RGB (16-23) into CPA (72-79).
15. Enable RGC (65-72) into CPA (72-79).
16. Enable bit carries into CPA (72-79).
17. Inhibit section carries.
18. Enable clear clocks to RGA (16-63) if $E=E_1=1$.
19. Enable load clocks to RGA (16-63) if $E=E_1=1$.
20. Enable load clocks to RGB (16-63).
21. Enable clear and load clocks to RCC (CPA carries are the input).
22. Select K function.

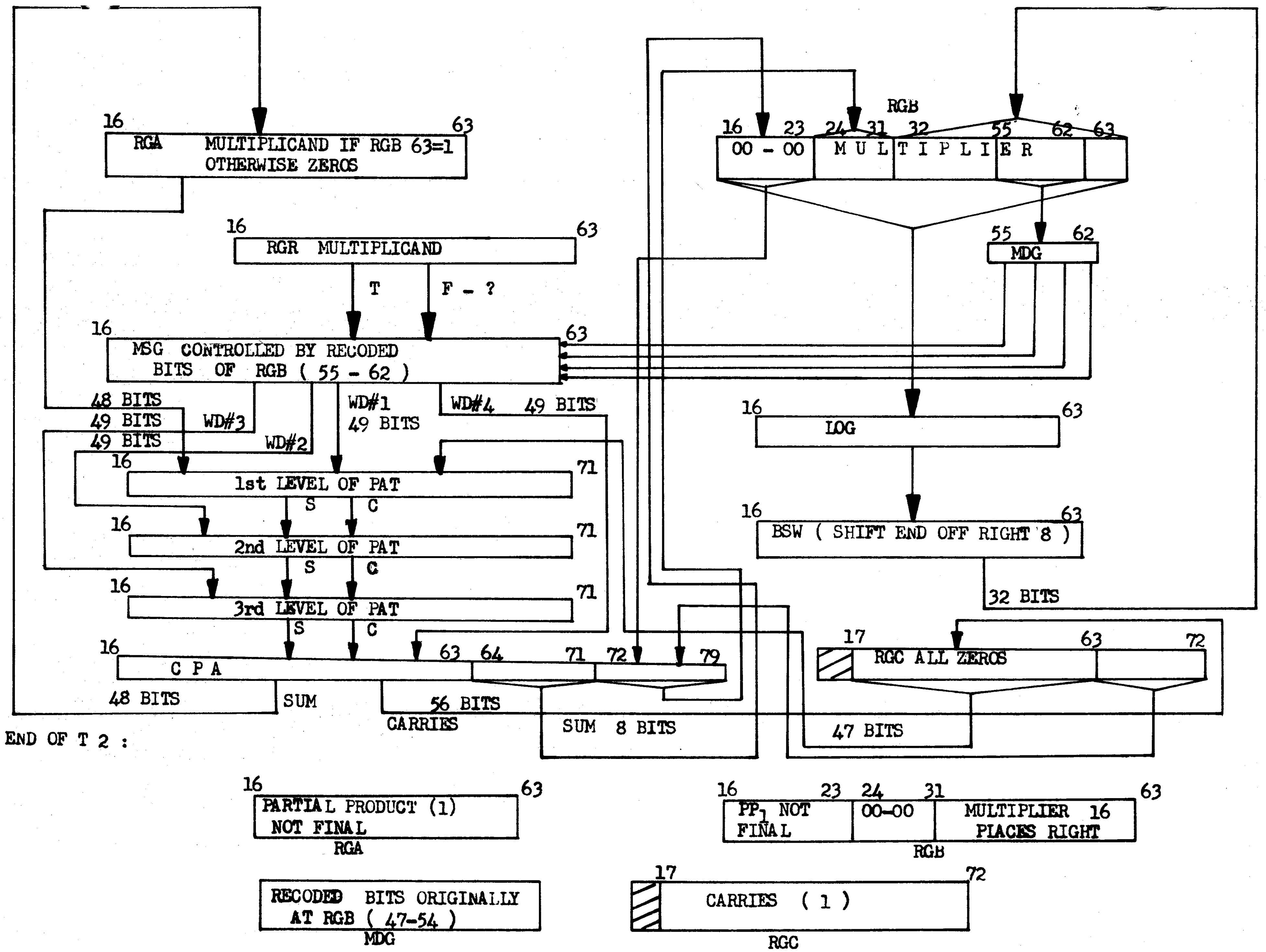


FIGURE 60. CLOCK TIME T 2

CLOCK TIME T3--Figure 61

Second Iteration - Record 8 Bits of Multiplier Mantissa for Third Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) to LOG (16-63)*.
5. Enable LOG (16-63) into BSW (16-63).
6. Force a shift to right 8 positions end off to the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable WD4 (16-63) into CPA (16-63).
9. Enable CPA sum (16-63) into RGA (16-63).
10. Enable BSW (32-63) into RGB (32-63).
11. Enable PAT sum and carry bits (16-71) to CPA.
12. Enable RGB (16-23) into CPA (72-79).
13. Enable RGC (65-72) into CPA (72-79).
14. Enable bit carries into CPA (72-79).
15. Inhibit section carries.
16. Enable load clocks to RGA (16-63) if $E=E_1=1$.
17. Enable clear clocks to RGB (16-63).
18. Enable load clocks to RGB (16-63).
19. Enable clear and load clocks to CPA carries.
20. Enable clear clocks to RGA (16-63) if $E=E_1=1$.
21. Select K function.

*Effectively RGB (0-63) is sent to LOG (0-63) but RGB (0-15) has no significance for the mantissa; there is one control signal allowing RGB (0-63) into LOG (0-63).

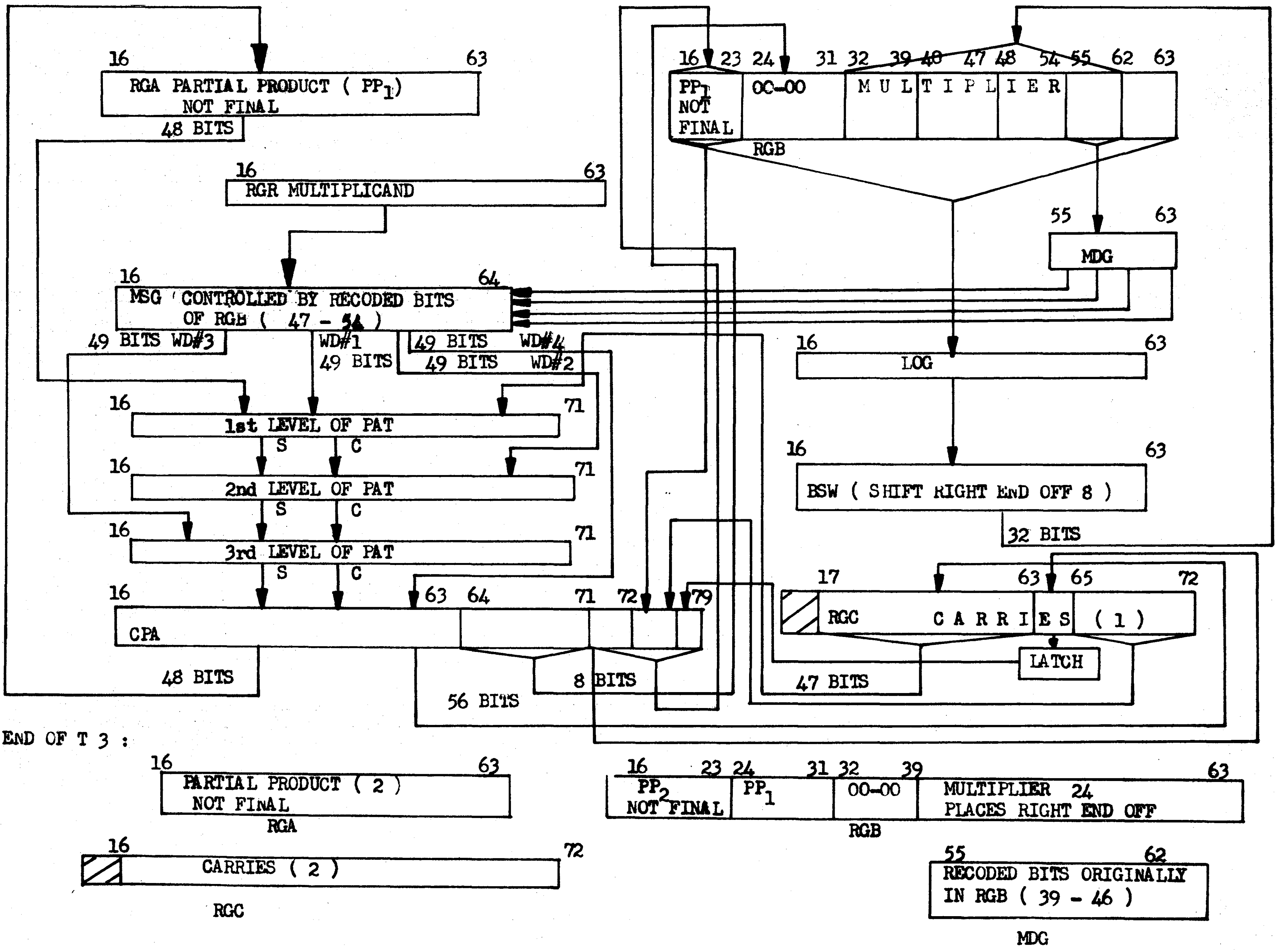


FIGURE 61. CLOCK TIME T 3

CLOCK TIME T4--Figure 62

Third Iteration - Recode 8 Bits of Multiplier Mantissa for Fourth Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) to LOG (16-63)--see footnote for clock time T3.
5. Enable LOG (16-63) into the BSW (16-63).
6. Force shift right end off 8 positions to the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable WD4 (16-64) into CPA (16-63).
9. Enable CPA sum (16-63) into RGA (16-63).
10. Enable BSW (32-63) into RGB (32-63).
11. Select K function.
12. Enable the stored carry in the latch to be placed in CPA bit 79.
13. Enable PAT sum and carry bits (16-71) into CPA.
14. Enable RGB (16-23) into CPA (72-79).
15. Enable RGC (65-72) into CPA (72-79).
16. Enable bit carries into CPA (72-79).
17. Inhibit section carries.
18. Enable clear clocks to RGA (16-63).
19. Enable load clocks to RGA (16-63).
20. Enable clear clocks to RGB (16-63).
21. Enable load clocks to RGB (16-63).
22. Enable clear and load clocks to CPA carries.

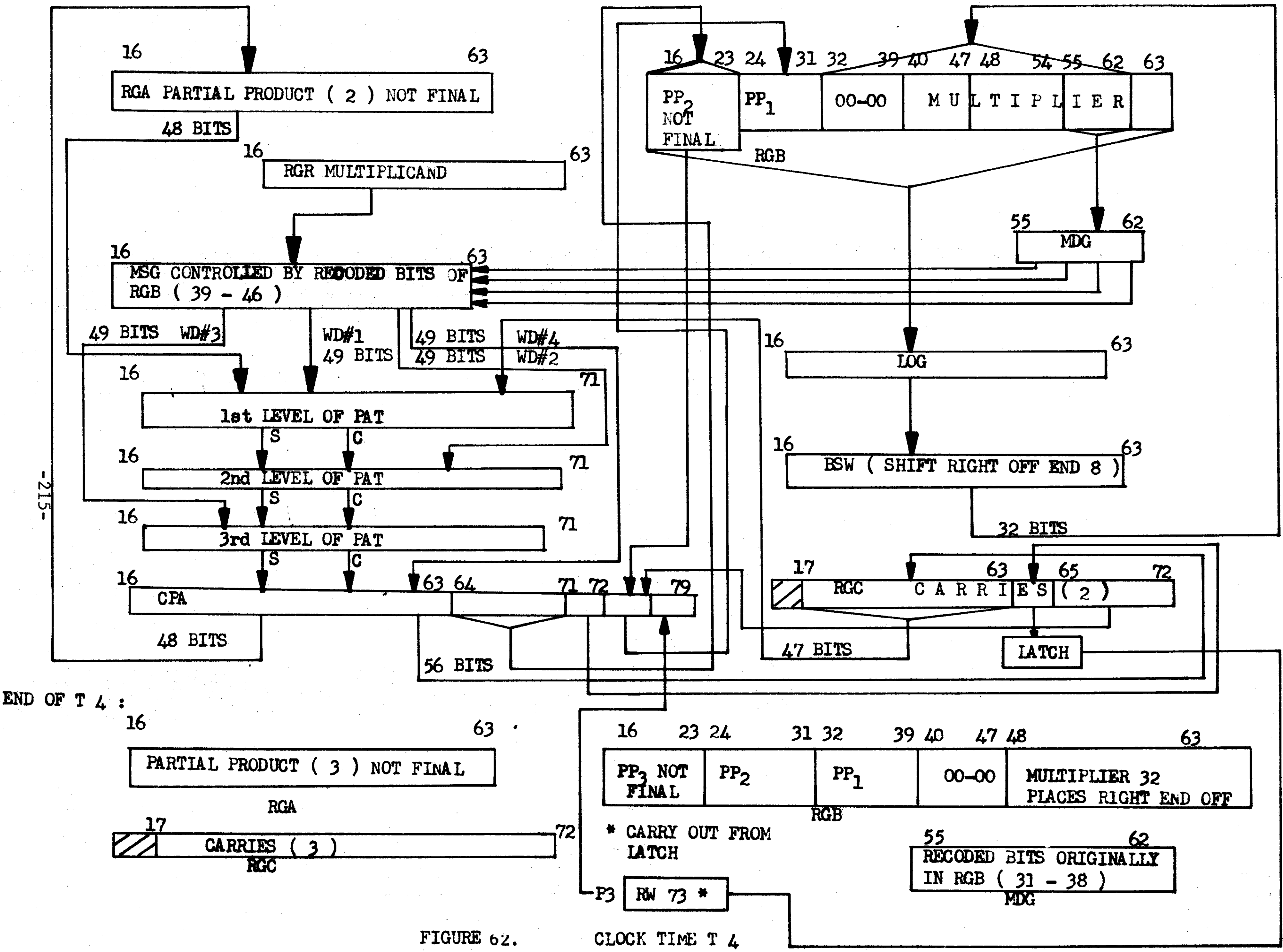


FIGURE 62. CLOCK TIME T 4

CLOCK TIME T5--Figure 63

Fourth Iteration - Recode 8 Bits of Multiplier Mantissa for Fifth Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) into LOG (16-63)--see footnote for clock time T3.
5. Enable LOG (16-63) into BSW (16-63).
6. Force shift right end off 8 positions to the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable BSW (32-63) into RGB (32-63).
9. Enable the stored carry in the latch to be placed in CPA bit 79.
10. Enable CPA sum (16-63) into RGA (16-63).
11. Enable PAT sum and carry bits (16-71) to CPA.
12. Enable WD4 (16-63) into CPA (16-63).
13. Enable RGB (16-23) into CPA (72-79).
14. Enable RGC (65-72) into CPA (72-79).
15. Enable bit carries into CPA (72-79).
16. Inhibit section carries.
17. Select K function.
18. Enable clear clocks to RGA (16-63).
19. Enable load clocks to RGA (16-63).
20. Enable clear clocks to RGB (16-63).
21. Enable load clocks to RGB (16-63).
22. Enable clear and load clocks to CPA carries.

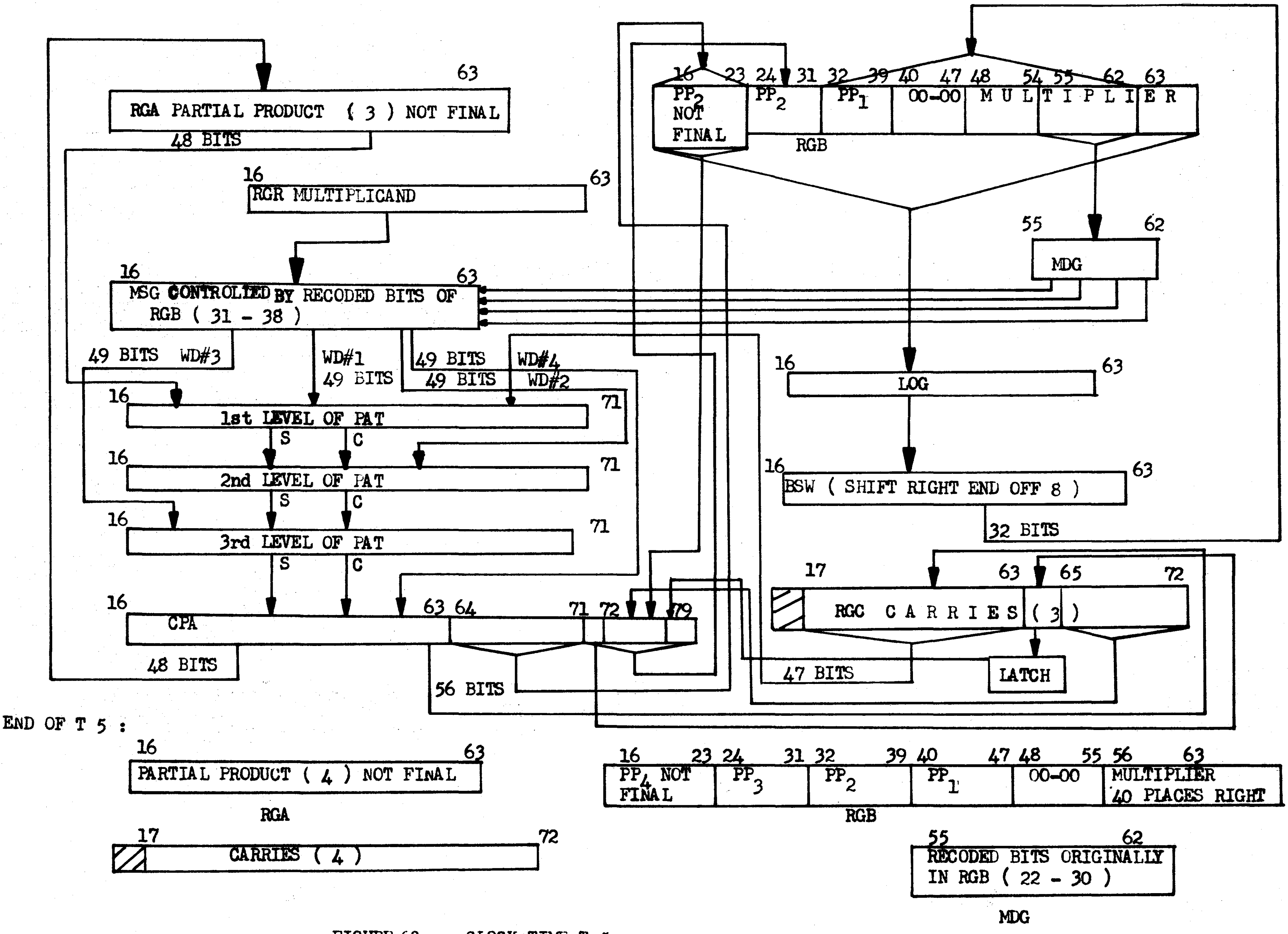


FIGURE 63. CLOCK TIME T 5

CLOCK TIME T6--Figure 64

Fifth Iteration - Recode 8 Bits of Multiplier Mantissa for Sixth Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) into LOG (16-63)--see footnote for clock time T3.
5. Enable LOG (16-63) into BSW (16-63).
6. Force shift right end off 8 positions to the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable BSW (32-63) into RGB (32-63).
9. Enable the stored carry in the latch to be placed in bit 79 of CPA.
10. Enable CPA sum (16-63) into RGA (16-63).
11. Enable PAT sum and carry bits (16-71).
12. Enable WD4 (16-63) into CPA (16-63).
13. Enable RGB (16-23).
14. Enable RGC (65-72) into CPA (72-79).
15. Enable bit carries into CPA (72-79).
16. Inhibit section carries.
17. Select K function.
18. Enable clear clocks to RGA (16-63).
19. Enable load clocks to RGA (16-63).
20. Enable clear clocks to RGB (16-63).
21. Enable load clocks to RGB (16-63).
22. Enable clear and load clocks to CPA carries.

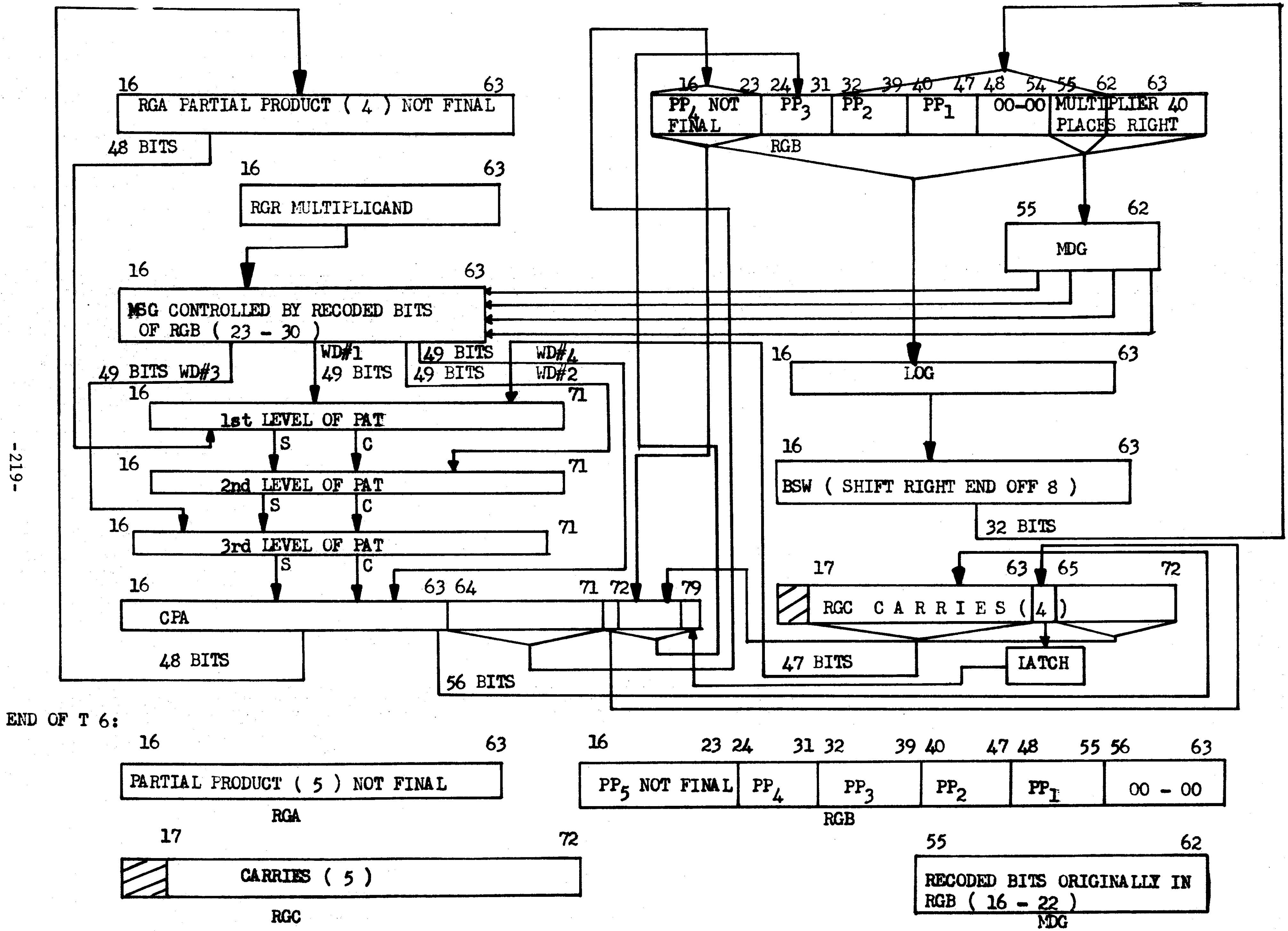


FIGURE 64. CLOCK TIME T 6

CLOCK TIME T7--Figure 65

Sixth Iteration

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable true out of RGR (16-63).
4. Enable RGB (16-63) into LOG (16-63)--see footnote for clock time T3.
5. Enable LOG (16-63) into BSW (16-63).
6. Force shift right end off 8 positions to the BSW controls.
7. Enable PAT sum and carry bits (16-71) to CPA.
8. Enable BSW (32-63) into RGB (32-63).
9. Enable the stored carry to be placed in bit 79 of CPA.
10. Enable CPA sum (16-63) into RGA (16-63).
11. Enable PAT sum and carry bits (16-71) to CPA.
12. Enable WD4 (16-63) into CPA (16-63).
13. Enable RGB (16-23) into CPA (72-79).
14. Enable RGC (65-72) into CPA (72-79).
15. Enable bit carries into CPA (72-79).
16. Inhibit section carries.
17. Select K function.
18. Round (optional).
19. Enable clear clocks to RGA (16-63).
20. Enable load clocks to RGA (16-63).
21. Enable clear clocks to RGB (16-63).
22. Enable load clocks to RGB (16-63).
23. Enable clear and load clocks to CPA carries.

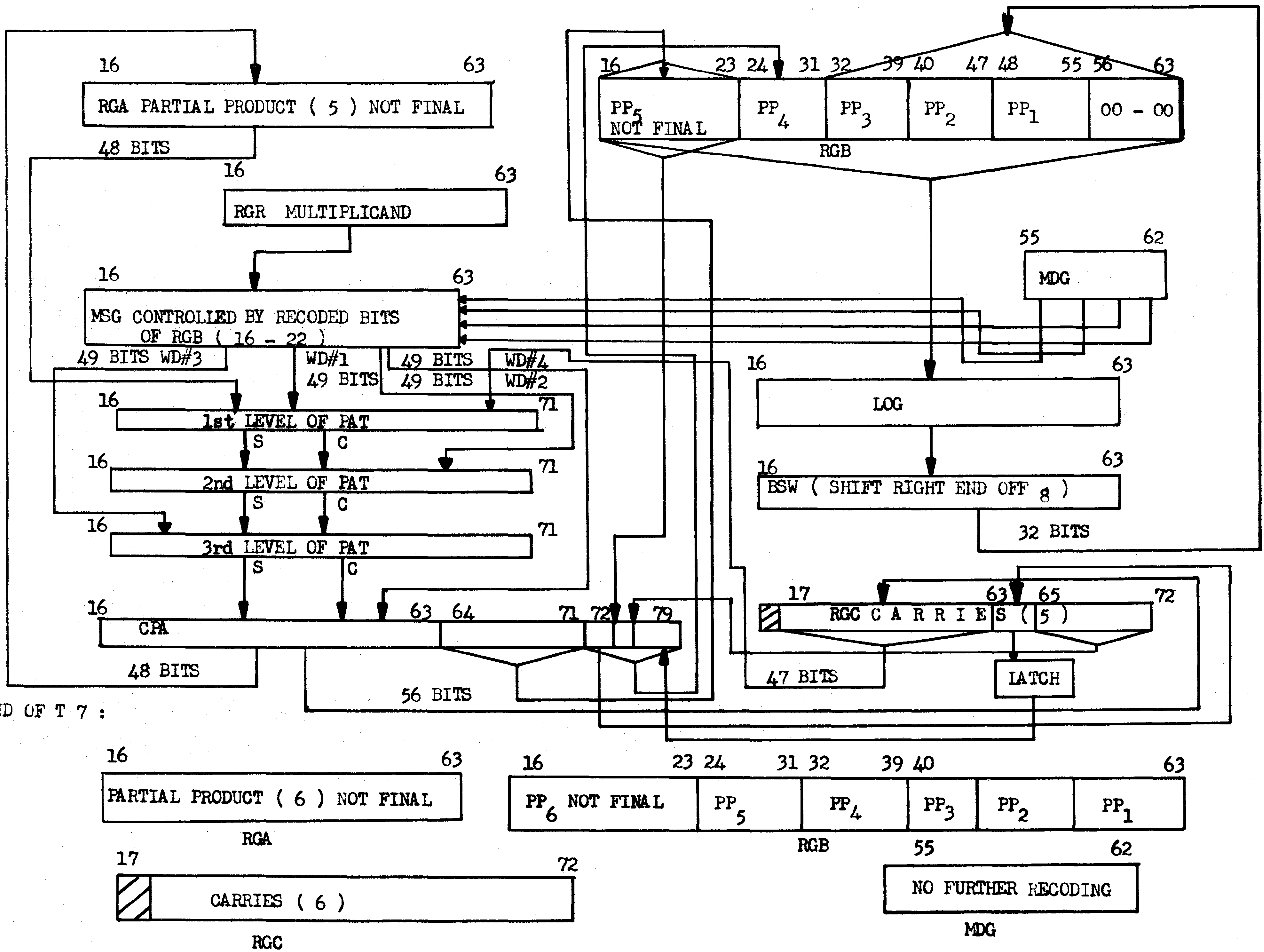


FIGURE 65. CLOCK TIME T 7

CLOCK TIME T8--Figures 66 and 67

Form the Final Product

1. Enable true out of RGA (16-63).
2. Enable true out of RGB (16-63).
3. Enable the group carries within the CPA.
4. Enable RGA (16-63) into CPA (16-63)--see footnote for clock time T3.
5. Enable bit carries into CPA (64-79).
6. Enable bit carries into CPA (16-63).
7. Enable RGB (16-23) into CPA (72-79).
8. Enable RGB (0-7) into CPA (64-71)*.
9. Enable RGC (65-72) into CPA (72-79).
10. Inhibit section carries.
11. Enable stored carry in latch to be placed in bit 79 of CPA.
12. Enable CPA (16-63) into RGA (16-63).
13. Enable RGB (16-63) into LOG (16-63).
14. Enable LOG (16-63) into BSW (16-63).
15. Enable CPA (72-79) into RGB (16-23).
16. Enable BSW (24-63) into RGB (24-63).
17. Enable 0 0 1 1 1 1 1 1 into RGB (0-7) for exponent correction in case of normalization.
18. Enable clear clocks to RGB (16-63).

*The gates in CPA (64-71) which receive RGB (0-7) are enabled, but RGB (0-7) are not enabled out of the B register. Thus all zeros are placed on one input to each of the corresponding CPA input gates that receive RGB (0-7). At the same time those gates are not enabled (S=1, see Figure 67, so that CPA (64-71) receives all ones.

19. Enable load clocks to RGB (16-63) if not rounding.

20. Clear RGA (0-15) on EXP UF and not normalized.

21. Clear RGA (16-63) on EXP UF and not normalized.

22. Load RGA (16-63) when normalizing and $E=E_1=1$.

23. Enable load and clear clocks to LOD latches.

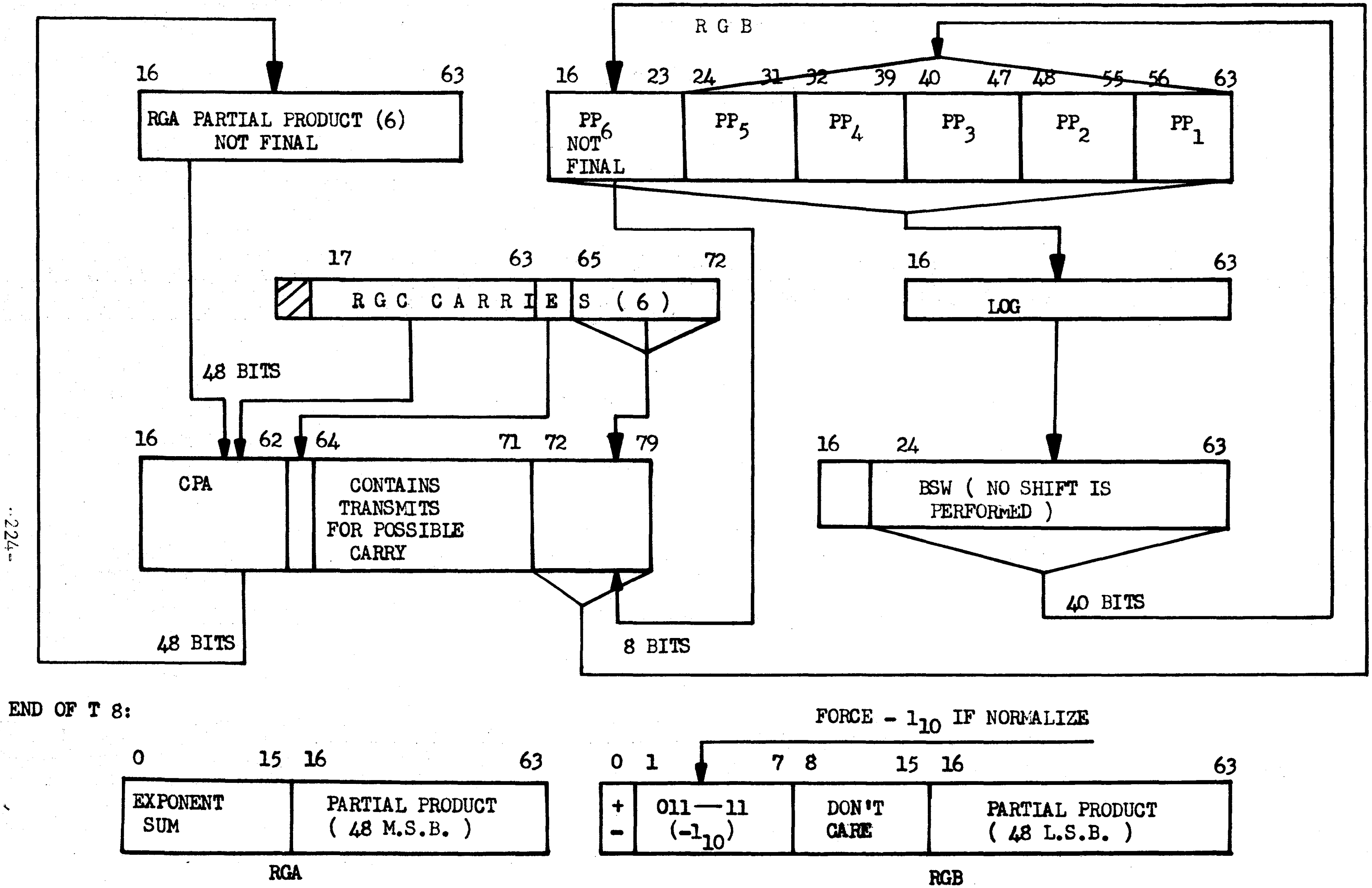
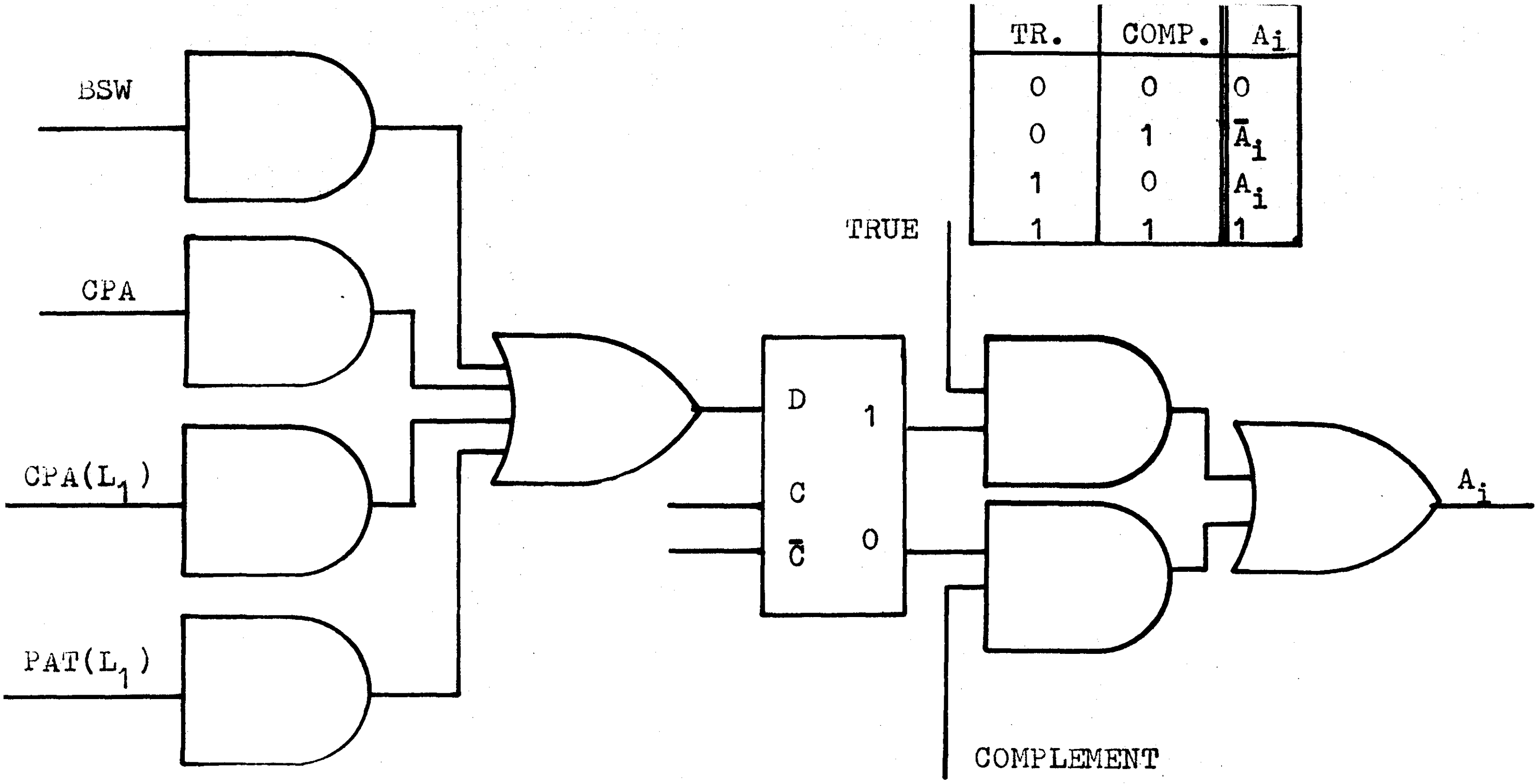
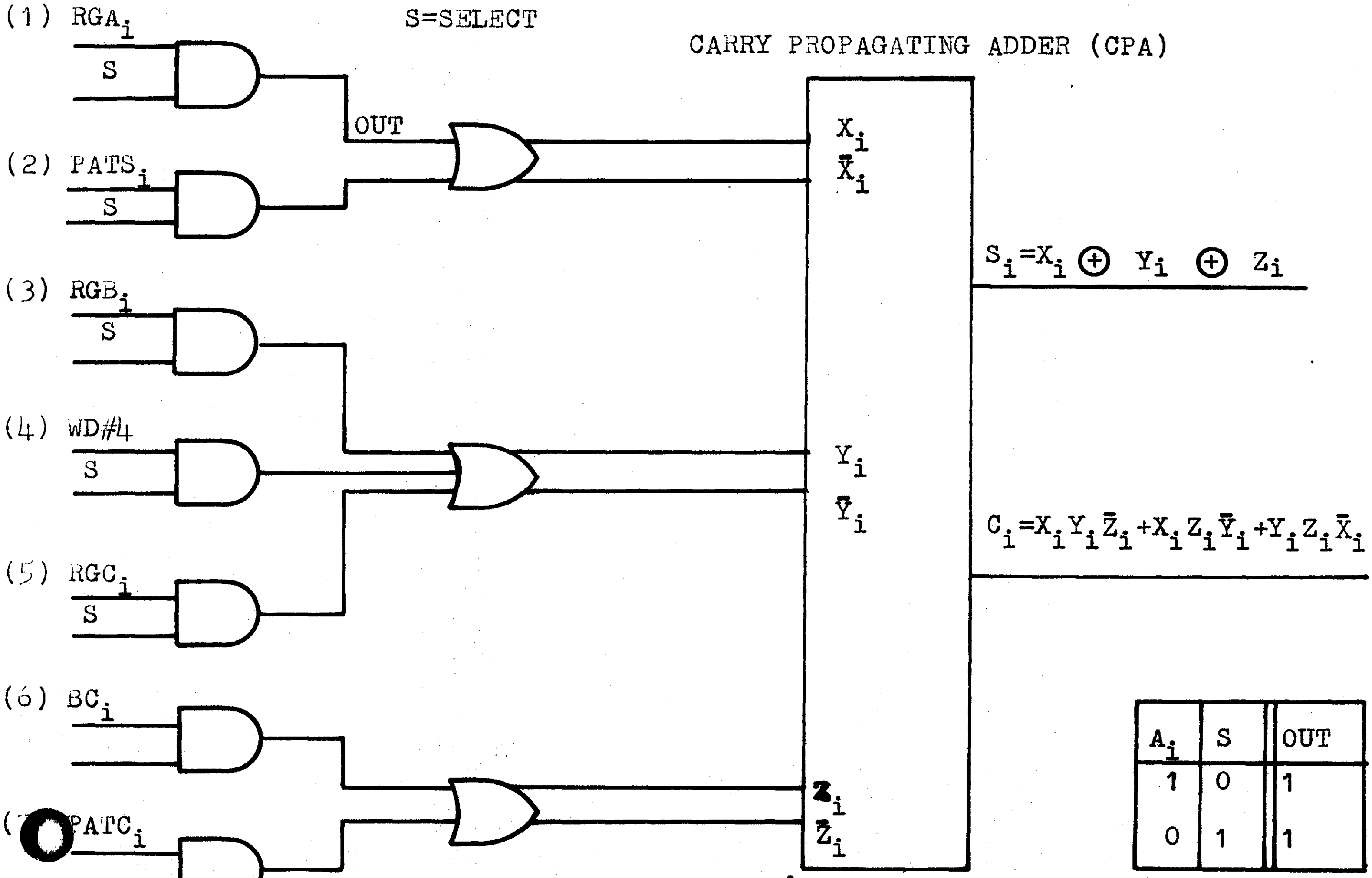


FIGURE 66. CLOCK TIME T 8



TR.	COMP.	A_i
0	0	0
0	1	$\bar{A}_i$
1	0	A_i
1	1	1



A_i	S	OUT
1	0	1
0	1	1

IN MULT (T_2-T_7) use (2), (4), (7),
(T_8) use (1), (5), (6)

FIGURE 67. INPUT GATING OF THE CPA, BIT SLICE DIAGRAM

CLOCK TIME T9--Figure 68

Normalize Final Product

1. Enable true out of RGA (0-63).

2. Enable true out of RGB (1-7, 16-63).

3. Enable CPA sum (64-79) to RGA (0-15).

4. Enable CPA sum (16-63) to RGA (16-63).

5. Clear RGA (0-63) on UFL or bit 16 is ZERO (UFL + A16).

6. Load RGA (0-63) on $\overline{\text{UFL}}$; $\overline{\text{UF}}$; A16; A17.

7. Enable clear and load clocks to F bit mode register.

8. Enable exponent underflow into mode register.

9. Inhibit clear clocks to RGD.

10. Enable RGA (1-63) into CPA (65-79, 16-63).

11. Enable RGB (1-7) into CPA (65-71).

12. Inhibit section carries.

13. Restore the sign of RGA (0).

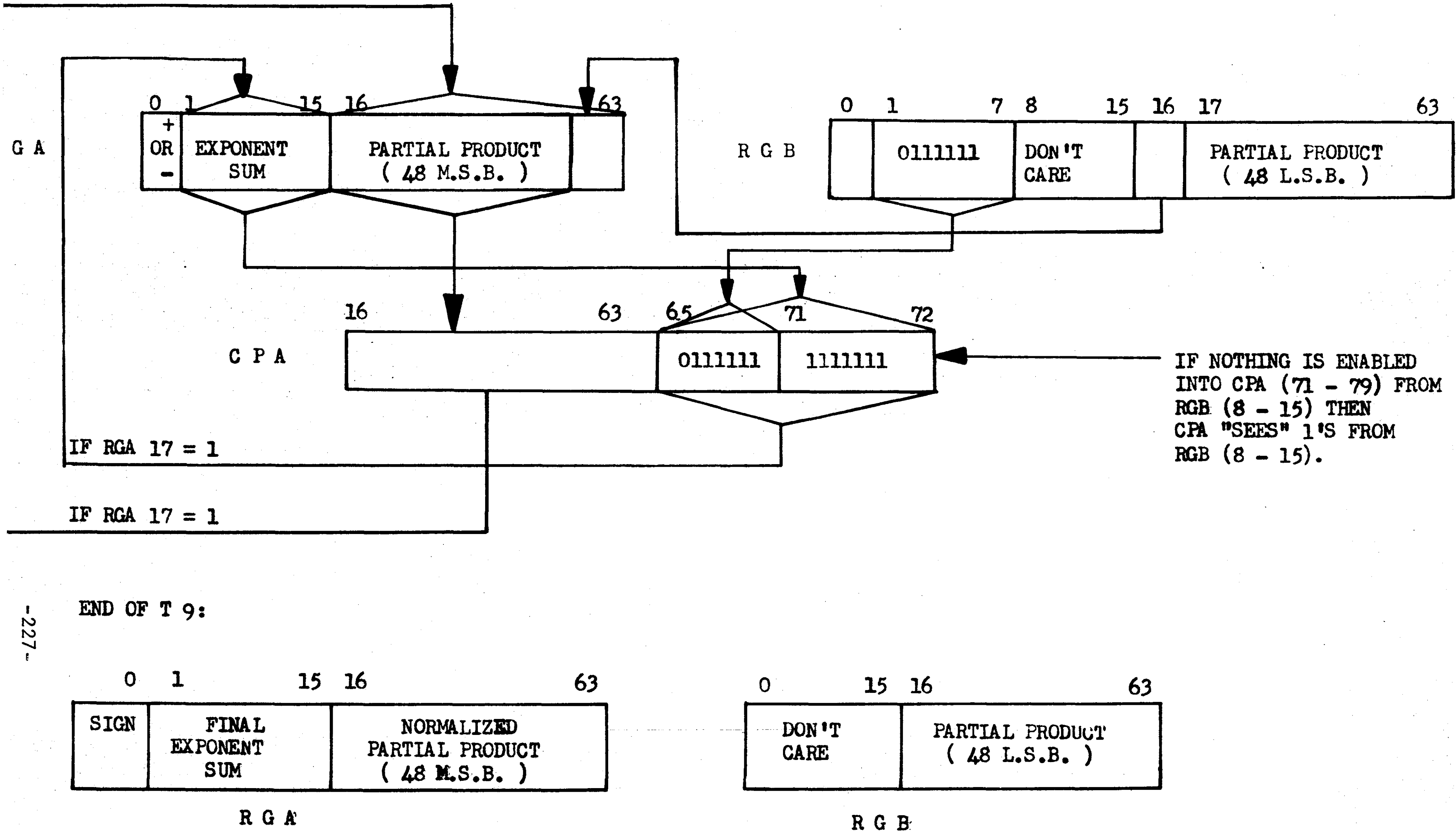

G A
0
1
15
16
63
+
OR
-
EXPONENT SUM
PARTIAL PRODUCT (48 M.S.B.)
R G B
0
1
7
8
15
16
17
63
0111111
DON'T CARE
PARTIAL PRODUCT (48 L.S.B.)
C P A
16
63
65
71
72
0111111
1111111
IF NOTHING IS ENABLED INTO CPA (71 - 79) FROM RGB (8 - 15) THEN CPA "SEES" 1'S FROM RGB (8 - 15).
IF RGA 17 = 1
IF RGA 17 = 1
END OF T 9:
0
1
15
16
63
SIGN
FINAL EXPONENT SUM
NORMALIZED PARTIAL PRODUCT (48 M.S.B.)
R G A
0
15
16
63
DON'T CARE
PARTIAL PRODUCT (48 L.S.B.)
R G B

FIGURE 68. CLOCK TIME T 9

D. Division

1. Introduction. In ordinary division, given two integers X and Y called dividend and divisor, respectively, two other integers are found; namely, the quotient Q and remainder R which satisfy the following conditions:

a. $R = X - YQ$

b. $|R| < Y$

Division is always performed by comparing the divisor with the dividend or the partial remainder and forming the quotient by guess. The correctness of the guess is determined by subtracting the product of the newly guessed digit of the quotient and the divisor from the dividend in the initial step and from the partial remainder in the consecutive steps. From the above relations it is concluded that if the signs of dividend and divisor are the same (+ or -), then the sign of the quotient is positive, otherwise it is negative. The sign of the remainder is always the same as the sign of the dividend. The factor that determines whether or not the newly selected digit of the quotient is valid is the sign of the result of the subtraction. If the sign of the result is the same as the sign of the dividend or partial remainder, then the selected quotient, is considered to be correct. If a sign change takes place, then the newly chosen digit (of the quotient) is not correct. In that case new guesses are made until the sign of the result of the subtraction is the same as the sign of dividend or partial remainder. In binary arithmetic the product of the quotient and the divisor is never greater than the divisor and thus only the divisor is subtracted from the dividend or partial remainders. Only one guess is required each time, because if the result has a different sign than that of the dividend or partial remainder, the guessed quotient bit had to be "1" and the proper choice is 0.

2. Methods of Division. There are several kinds of division; among them the common ones are: restoring, nonrestoring, and nonperforming.

In restoring division, the divisor is successively subtracted from the dividend or partial remainder to generate at least one quotient bit at a time. In single bit division which generates one quotient bit in each clock period, the quotient bit becomes ONE any time the result of the subtraction is positive and ZERO otherwise. The partial remainder or the dividend in the initial subtraction and the quotient are shifted to the left by 1 when the quotient becomes ZERO and the dividend or current partial remainder is restored.

In nonrestoring division, the cycle has two subcycles. The first one takes care of the subtraction of the divisor from the dividend or partial remainder, the second subcycle forms the quotient bit and shifts the quotient and partial remainder to the left by 1. In this method there is no restoration required due to a negative result in the cycle where the negative result was detected, but in the next cycle the divisor is added to instead of being subtracted from the new partial remainder. This, however, requires extra logic gates to pick up (locally) the TRUE or COMPLEMENT output of the register containing the divisor, which, from the hardware point of view, results in an increase in cost and complexity of the machine.

Nonperforming division is the sort used in the PE. As described above in the restoring division method if the sign of the quotient is positive and the result of the subtraction is positive a ONE is entered into the quotient; if the result is negative a ZERO is entered. If the sign of the quotient is negative, a ZERO is entered in the quotient bit if the result of the subtraction is positive, or a ONE otherwise. Nonperforming division is similar to restoring division, but if the sign of the result is the same as that of the current partial remainder then a ONE is entered in the quotient bit if the sign of the quotient is positive and a ZERO if the sign of the quotient is negative. If the sign of the result and that of the current partial remainder disagree, the result of the subtraction is ignored, the old partial remainder is shifted to the left by one place and if the quotient sign is positive a ZERO is inserted in the quotient register; a ONE otherwise.

3. Implementation.

a) Mantissa: The PE of ILLIAC IV has adopted the nonperforming division method because, from the hardware point of view, it is simpler and more economical to implement. The division can be characterized as a "long" operation because it requires a recursive process for the generation of the quotient field one bit at a time. This process can be described by the following general equation:

$$X_{i+1} = rX_i - Q_{i+1}Y \text{ , } i = 1,2, \ldots ,48 \qquad (1)$$

where:

X_{i+1} = Partial remainder after the $(i+1)^{th}$ step of the division

X_i = Partial remainder after the i^{th} step of the division

X_o = Dividend

Q_{i+1} = The $(i+1)^{th}$ bit of the quotient to the right of the binary point

Y = Divisor

r = Radix

The remainder of the first step is given by:

$$X_1 = X_0 - Q_1Y \qquad (2)$$

because at the start of the recursive process the dividend is not shifted to the left and, therefore, the first step of the division cannot be described by (1). Further study of (1) leads us to the mechanization of the recursive process as follows:

Since the remainder from the first step is given by:

$$X_1 = X_0 - Q_1Y$$

for i = 1, (1) becomes:

$$\begin{aligned} X_2 &= 2X_1 - Q_2Y \\ &= 2X_0 - 2Q_1Y - Q_2Y \end{aligned} \tag{3}$$

for i = 2, (1) becomes:

$$\begin{aligned} X_3 &= 2X_2 - Q_3Y \\ &= 2^2X_0 - (2^2Q_1Y + 2Q_2Y + Q_3Y) \end{aligned} \tag{4}$$

for i = 47, (1) becomes:

$$X_{48} = 2^{47}X_0 - (2^{47}Q_1Y + 2^{46}Q_2Y + ... + Q_{48}Y) \tag{5}$$

for i = 48, (1) becomes:

$$X_{49} = 2^{48}X_0 - (2^{48}Q_1Y + 2^{47}Q_2Y + ...+ Q_{49}Y) \tag{6}$$

The dividend being 96 bits long occupies the A and B registers with the most significant bits placed in A register. The dividend need not be normalized. The divisor, which is 48 bits, must be normalized before the division starts; it is placed in R register. Since the recursive process requires subtraction of the divisor from the dividend, the 1's complement of the divisor is taken into the CPA where an extra ONE is added to the least significant bit in order to form the 2's complement of the divisor and the result is added to the dividend. If the subtraction is "successful," in other words the result is positive, then the quotient bit is a ONE; if the result is negative the quotient bit is a ZERO. This is determined by Group Carry 16 only for the first execution because the "R sign" latch at the beginning contains a ZERO. In the remaining steps of the recursive process the quotient bit is determined by the content of "R sign" latch (ONE), or Group Carry 16 which is a result of an end-around carry from the subtraction of the divisor from the dividend or partial remainder. An overflow can also force the Group Carry 16 to be a ONE.

Since the state of the quotient bit Q_2 depends on the state of the Group Carry 16 (G.C. 16) and "R sign" latch, it is important to know what the variables are what set the "R sign" latch. At the beginning, the "R sign" latch is cleared and therefore

$$\text{R sign}_{(0)} = 0$$

From the logic diagram (card B06) the equation for setting the new "R sign" latch is given by:

$$\begin{aligned}\text{R sign}_{(i+1)} = {} & \text{A16 } (\overline{\text{Rsign}}_{(i)} \cdot \overline{\text{G.C.16}}) + \text{A16} \cdot \text{CPA16} \\ & + \text{A16} \cdot \text{CPA16} \cdot (\text{Rsign}_{(i)} + \text{G.C.16}) \\ & + \text{A16} \cdot \text{CPA16} \cdot (\overline{\text{Rsign}}_{(i)} \cdot \overline{\text{G.C.16}}) \\ & + \text{CPA16} \cdot (\text{Rsign}_{(i)} + \text{G.C.16}) \qquad (7)\end{aligned}$$

After minimization (7) is reduced to:

$$\text{Rsign}_{(i+1)} = \text{A16} \cdot (\overline{\text{Rsign}}_{(i)} \cdot \overline{\text{G.C.16}}) + \text{CPA16} \cdot (\text{Rsign}_{(i)} + \text{G.C.16}) \qquad (8)$$

where $i = 0, 1, \ldots., 48$

A16 = Bit position 16 of "A" register

CPA16 = Bit position 16 of Carry Propagating Adder

It can be said that $\text{Rsign}_{(i+1)}$ is set when

$$A16 \cdot Q_i \text{ or } CPA16 \cdot Q_i \text{ are true}$$

where $Q = \text{Rsign}_{(i+1)} + G.C.16$

This means that when $Q_i = 0$ the content of A register is shifted to the left by 1 through the PAT because the subtraction was unsuccessful. If A16 is a ONE, this sets the R sign latch. If $Q_1 = 1$, the result in CPA is shifted through the CPA to the left by 1 and it is brought back to A register. If CPA16 then happens to be a ONE, that sets the Rsign latch. In general, the Rsign latch looks like an extension of A register to the left by 1 position.

The status of quotient bit Q_1 determines the procedure in the second step of the recursive process. If $Q_1 = 1$, the subtraction was "successful" and, therefore, the result is positive. This result, which is the remainder, is taken through the CPA back to A register, but shifted to the left by one place. If $Q_1 = 0$, the subtraction was "unsuccessful," in which case the result of the subtraction is ignored and the dividend from A register is passed through the PAT and, after being shifted by one to the left, is placed back into A register. At the time that the shifting of the remainder or of the dividend takes place, the mantissa of B register is shifted to the left by one place directly to A register, thus bringing bit 16 of B register into bit 63 of A register. At the same time bit position 63 of B register receives the first quotient bit (Q_1). After the end of the second iteration, quotient bit Q_2 will be either ONE or ZERO, depending on the sign of the result of the subtraction of the divisor in 2's complement form from the first partial remainder shifted by one place to the left if $Q_1 = 1$ or from the dividend also being shifted by one to the left if $Q_1 = 0$.

For the third, fourth, etc., steps the process is repeated and performed in the same way as in the second step, but it should not be forgotten that if $Q_i = 0$, then $X_{i+1} = 2X_i$, which means that if the previous quotient bit was a ZERO, the new remainder for the next execution will be the remainder from which the divisor was subtracted to give $Q_i = 0$, but shifted through the PAT by one place to the left (which is the same as multiplying it by 2).

At the end of the 48th step, equation (5) indicates that the original dividend has been shifted 47 times to the left and the quotient bit Q_{48} has been formed and, therefore, inserted into bit location 63 of B register. The weight of this bit is 2^{-48}. Rewriting (5):

$$2^{-48}X_{48} = 2^{-1}X_0 - (2^{-1}Q_1Y + 2^{-2}Q_2Y + \dots. + 2^{-48}Q_{48}Y) \qquad (9)$$

In this step an additional operation takes place when a certain condition exists. This condition is a function of the magnitude of the dividend and divisor. It has been said that the dividend need not be normalized while the divisor must be normalized before the division begings. This means that:

$$0 \leq X_0 < 1 \qquad \text{and} \qquad \tfrac{1}{2} \leq Y < 1$$

This condition will force the quotient to be:

$$0 \leq Q < 1$$

because, as shown on the chart in Figure 69 (due to R. Davis), the quotient varies from 0 to 1 only when the divisor takes on values greater than or equal to ½ but less than 1.

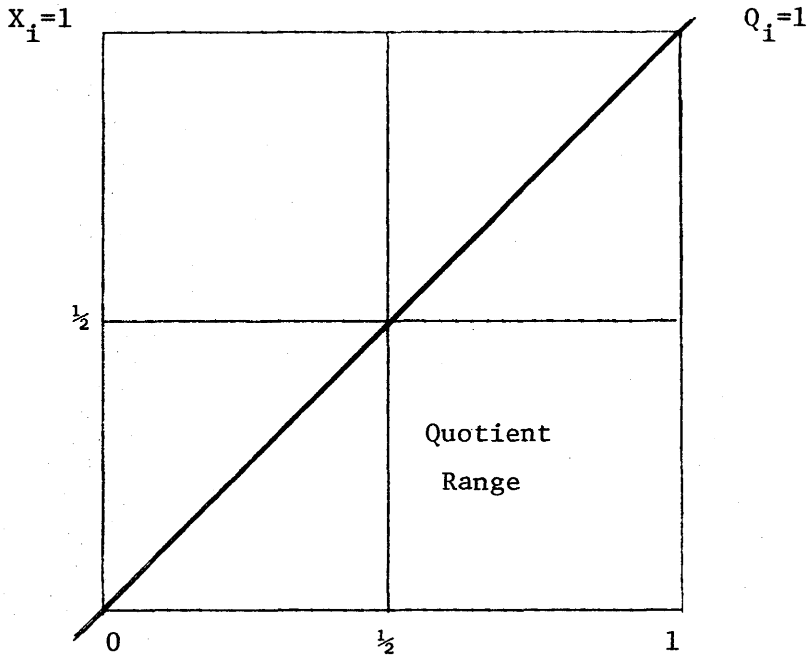


Figure 69. Chart Showing the Area in which the Quotient is Valid

When the dividend is greater than or equal to the divisor the quotient in this case would be $1 \leq 0 < 2$. To circumvent this the quotient is forced to take the value $\frac{1}{2} \leq 0 < 1$ by shifting it one place to the right and adding a ONE to the final exponent. The actual mechanization to take care of this case is performed in the following way.

At the beginning of the recursive process (Clock Time T4) when the divisor is subtracted from the dividend, the quotient bit Q_1 will be

ONE if the dividend is greater than or equal to the divisor,

ZERO if the dividend is less than the divisor.

At the end of the 48th step of the recursive process (Clock Time T51) the quotient bit Q_1 will be at bit position 16 of the B register because in every clock time there is a shift of the quotient bit Q_1 to the left by one place. The result of the subtraction has already determined the quotient bit Q_{48} and if Q_{48}=1, then the result is shifted one position to the left and is placed back into A register to become the partial remainder for the 49th execution of the recursive process. If Q_{48}=0 the previous remainder is shifted one place to the left and placed into A register through the PAT.

In the next clock time no "execution step" takes place, but a test is made to adjust the quotient field and force it to be in the range of $\frac{1}{2} \leq Q < 1$ in the case in which the dividend is greater than, or equal to, the divisor. If bit position 16 of B register contains a ONE, which means $Q_1=1$, the exponent which has already been formed (end of Clock Time T3) is increased by ONE. If bit position 16 of B register contains a ZERO, the exponent remains unchanged. In this clock time also, if an exponent overflow occurs or might occur because of the increase of the exponent by ONE, the mode register is armed to set the F bit for a fault indication.

At the end of the 49th step (Clock Time T53), if the bit in bit position 16 of the B register is a ONE, this register is blocked to prevent insertion of the final quotient bit into bit position 63 and no shifting of the quotient field is performed. If bit position 16 of B register is a ZERO, the final quotient bit is allowed to be inserted into bit position 63 of B register, while at the same time the quotient field is shifted to the left one place.

Rearrangement of equation (6) gives:

$$2^{-48}X_{49} = X_0 - (Q_1Y + 2^{-1}Q_2Y + 2^{-2}Q_3Y + \ldots + 2^{-48}Q_{49}Y) \qquad (10)$$

This equation indicates that Q_1 has a weight of 2^0 and, therefore, this bit in floating point arithmetic is to the left of the binary point. If $Q_1=0$ it is allowed to be shifted off the most significant position of the quotient field and, since Q_2 has a weight of 2^{-1}, this guarantees the range of the quotient field to be:

$$0 \leq Q < 1$$

because Q_2 can be either a ZERO or ONE. This matches the requirements set up by the chart of Figure 1 for a valid quotient. If $Q_1=1$ this implies that $X_o \geq Y$ and, therefore, the remainder will be invalid. In this case it was said that the exponent must be increased by one and the quotient should be shifted by one to the right. The exponent indeed is increased in Clock Time T52, but in the 49th step B register was blocked and even though the divisor was subtracted from the partial remainder (X_{48}) the quotient bit Q_{49} was neither inserted into B register nor was the quotient field shifted and $Q_1=1$ was forced to remain in the bit position of weight 2^{-1}, thus guaranteeing a valid quotient field:

$$\tfrac{1}{2} \leq Q < 1$$

The remainder in this case is not valid. The programmer has an option to force the dividend X_0 to be less than the divisor by shifting X_0 right one end off when $X_0 \geq Y$ assuming that the divisor has already been normalized. In this case he will guarantee $Q_1=0$, a quotient field $0 \leq Q < 1$ and a valid remainder. This shifting should be made prior to the beginning of the recursive process. If $Q_{49}=1$, the final remainder will be equal to the result of the subtraction of the divisor from the partial remainder of the 49th step, which is put into A register; if $Q_{49}=0$ it is equal to the remainder (X_{48}) of the 48th step which was placed into A register as previously described.

Therefore, the final remainder of the recursive process which has been shifted to the left 48 times may be described by:

$$2^{-48}X_{49} = X_0 - Y\sum_{i=1}^{48} 2^{-i}Q_i \qquad \text{if} \quad \begin{matrix} Q_1 = 1 \\ Q_{49} = 1 \end{matrix} \qquad (11)$$

$$2^{-48}X_{48} = X_0 - Y\sum_{i=1}^{48} 2^{-i+1}Q_{i+1} \qquad \text{if} \quad \begin{matrix} Q_1 = 0 \\ Q_{49} = 0 \end{matrix} \qquad (12)$$

Remember that when Q_{49} is not allowed to be gated into the least significant bit of B register the state of Q_{49} determines the remainder. If $Q_1=0$ and $Q_{49}=1$, then equation (12) becomes:

$$2^{-48}X_{49} = X_0 - Y\sum_{i=1}^{48} 2^{-i}Q_{i+1} \qquad (13)$$

In the next clock Time (T54) the final remainder which is in the mantissa part of A register is transferred into B register while, at the same time, the quotient which was placed one bit at a time into the mantissa part of B register is transferred into A register.

Table 38 summarizes the steps for the development of the quotient field. If the result of the subtraction of the divisor from the dividend or partial remainder is equal to or greater than ZERO, in which case the quotient bit is ONE, the new partial remainder is the result of the subtraction being shifted through the CPA by one place to the left and placed into A register. If the result of the subtraction is less than ZERO, in which case the quotient bit is ZERO, then the new partial remainder is the previous partial remainder, which after being shifted through the PAT to the left by one is placed in A register. The latter case can be interpreted as avoiding restoration of the remainder, a characteristic of restoring division, thus saving the considerable amount of time required for the restoration of the remainder. In this way the result of the subtraction is ignored because the subtraction is said to be "unsuccessful."

Because all PEs are subject to a lock-step synchronous operation, if the method of restoring division was used in ILLIAC IV, it would require an extra 49 clock times to perform the restoration, even though some of the PEs would not have to restore. This, in conjunction with the requirements of the nonrestoring division which has been previously explained, was the dominating factor for the adoption of the nonperforming method of division in ILLIAC IV.

An example of division in the ILLIAC IV PE is to be found on the next six pages.

Table 38. Steps of Mantissa Manipulation in 64-Bit Mode Division

CLOCK TIME	ACTION TAKING PLACE	REMARKS
T1	Clear "C" register for future use of PAT	
T2	Shift mantissa of R register through barrel switch right end off one place and place it into mantissa part of B register.	Only for rounding
T3	Set signs of A and B register equal. Clear R sign latch. Set controls for shift left by one into shift count register. Check bit 16 of R register and set F bit if bit 16 is zero.	
T4 ↕	If result of subtraction <0, shift mantissa of A register through PAT left by one end off. If result of subtraction ≥ 0, then shift it through CPA by one to the left. Shift the most significant bit of B register into the least significant bit of A register. Bring Q_i into the least significant bit of A register.	
T51	Bring Q_i into the least significant bit of B register.	
T52	No action concerned with the mantissa occurs during this clock time.	
T53	If result of subtraction ≥ 0, bring it through CPA back to A register. If $Q_i = 0$, then shift mantissa of B to the left by one and place Q_{49} into bit position 63 of B register.	
T54	Interchange mantissa of A and B registers.	
T55	Shift mantissa of A to the left through barrel switch until leading one is at bit 16 of A register.	Only when normalized
T56	Clear mantissa and insert zeros into sign and exponent part of A register if exponent underflow occurred.	

EXAMPLE

GIVEN:

The dividend X_0 of 96 bits long, the 48 most significant of which are placed in A register and the rest, 48 bits, in B register and the divisor Y is placed in R register.

FIND:

The quotient field and the remainder.

SOLUTION

1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	"A" Register	DIVIDEND
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	"B" Register	
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	"R" Register	DIVISOR

Q_i=	Rs			Clock Time
		1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Y	
		0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	$\overline{Y}$	
		1		
		1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\overline{Y}$ + 1	
		1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	X_0	
$Q_1$1	1	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	X_0 + $\overline{Y}$ + 1	T4
		1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	2 x 1	
$Q_2$1	1	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	2 x 1 + $\overline{Y}$ + 1	T5
		1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	2 x 2	
$Q_3$1	1	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	2 x 2 + $\overline{Y}$ + 1	T6
		1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	2 x 3	
$Q_4$1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	2 x 3 + $\overline{Y}$ + 1	T7
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	2 x 4	
$Q_5$0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	2 x 4 + $\overline{Y}$ + 1	T8
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	2 x 2 x 4	
$Q_6$0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	2 x 2 x 4 + $\overline{Y}$ + 1	T9
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	2^3x 4	
$Q_7$0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	2^3x 4 + $\overline{Y}$ + 1	T10
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	2^4x 4	
$Q_8$0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	2^4x 4 + $\overline{Y}$ + 1	T11
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	2^5x 4	
$Q_9$0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	2^5x 4 + $\overline{Y}$ + 1	T12
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	2^6x 4	
Q_{10}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	2^6x 4 + $\overline{Y}$ + 1	T13
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	2^7x 4	
Q_{11}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	2^7x 4 + Y + 1	T14

Rs = "R" Sign Latch

Q_i=	Rs			Clock Time
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	2^8x 4	
$Q_{12}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	2^8x 4 + $\overline{Y}$ + 1	T16
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	2^9x 4	
$Q_{13}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	2^9x 4 + $\overline{Y}$ + 1	T17
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	2^{10}x 4	
$Q_{14}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	2^{10}x 4 + $\overline{Y}$ + 1	T18
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{11}x 4	
$Q_{15}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{11}x 4 + $\overline{Y}$ + 1	T19
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{12}x 4	
$Q_{16}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{12}x 4 + $\overline{Y}$ + 1	T20
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{13}x 4	
$Q_{17}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{13}x 4 + $\overline{Y}$ + 1	T21
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{14}x 4	
$Q_{18}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{14}x 4 + $\overline{Y}$ + 1	T22
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{15}x 4	
$Q_{19}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{15}x 4 + $\overline{Y}$ + 1	T23
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{16}x 4	
$Q_{20}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{16}x 4 + $\overline{Y}$ + 1	T24
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{17}x 4	
$Q_{21}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{17}x 4 + $\overline{Y}$ + 1	T25
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{18}x 4	
$Q_{22}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{18}x 4 + $\overline{Y}$ + 1	T26
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{19}x 4	
$Q_{23}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{19}x 4 + $\overline{Y}$ + 1	T27
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{20}x 4	
$Q_{24}0$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{20}x 4 + $\overline{Y}$ + 1	T28

Rs = "R" Sign Latch

Q_i=	Rs			Clock Time
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{21}x 4	
Q_{25}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{21}x 4 + $\overline{Y}$ + 1	T29
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{22}x 4	
Q_{26}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{22}x 4 + $\overline{Y}$ + 1	T30
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{23}x 4	
Q_{27}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{23}x 4 + $\overline{Y}$ + 1	T31
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{24}x 4	
Q_{28}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{24}x 4 + $\overline{Y}$ + 1	T32
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{25}x 4	
Q_{29}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{25}x 4 + $\overline{Y}$ + 1	T33
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{26}x 4	
Q_{30}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{26}x 4 + $\overline{Y}$ + 1	T34
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{27}x 4	
Q_{31}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{27}x 4 + $\overline{Y}$ + 1	T35
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{28}x 4	
Q_{32}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{28}x 4 + $\overline{Y}$ + 1	T36
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{29}x 4	
Q_{33}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{29}x 4 + $\overline{Y}$ + 1	T37
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{30}x 4	
Q_{34}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{30}x 4 + $\overline{Y}$ + 1	T38
		0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{31}x 4	
Q_{35}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{31}x 4 + $\overline{Y}$ + 1	T39
		0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{32}x 4	
Q_{36}0	0	1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{32}x 4 + $\overline{Y}$ + 1	T40
		0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{33}x 4	
Q_{37}0	0	1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{33}x 4 + Y + 1	T41
		0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{34}x 4	
Q_{38}0	0	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{34}x 4 + $\overline{Y}$ + 1	T42

Rs = "R" Sign Latch

Q_i=	Rs			Clock Time
		0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{35}x 4	
Q_{39}0	0	1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{35}x 4 + $\overline{Y}$ + 1	T43
		0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{36}x 4	
Q_{40}0	0	1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{36}x 4 + $\overline{Y}$ + 1	T44
		0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{37}x 4	
Q_{41}0	0	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{37}x 4 + $\overline{Y}$ + 1	T45
		0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{38}x 4	
Q_{42}0	0	1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{38}x 4 + $\overline{Y}$ + 1	T46
		0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{39}x 4	
Q_{43}0	0	1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{39}x 4 + $\overline{Y}$ + 1	T47
		0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{40}x 4	
Q_{44}0	0	1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{40}x 4 + $\overline{Y}$ + 1	T48
		0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{41}x 4	
Q_{45}0	0	1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{41}x 4 + $\overline{Y}$ + 1	T49
		0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{42}x 4	
Q_{46}0	0	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{42}x 4 + $\overline{Y}$ + 1	T50
		0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{43}x 4	
Q_{47}0	0	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{43}x 4 + $\overline{Y}$ + 1	T51
		0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{44}x 4	
Q_{48}0	0	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2^{44}x 4 + $\overline{Y}$ + 1	T52
		1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	2^{45}x 4	
Q_{49}1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	2^{45}x 4 + $\overline{Y}$ + 1	T53

Rs = "R" Sign Latch

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}	Q_{19}	Q_{20}	Q_{21}	Q_{22}	Q_{23}	Q_{24}	Q_{25}	Q_{26}	Q_{27}	Q_{28}	Q_{29}	Q_{30}	Q_{31}	Q_{32}	Q_{33}	Q_{34}	Q_{35}	Q_{36}	Q_{37}	Q_{38}	Q_{39}	Q_{40}	Q_{41}	Q_{42}	Q_{43}	Q_{44}	Q_{45}	Q_{46}	Q_{47}	Q_{48}

"A" Register

T54

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
B_{16}	B_{17}	B_{18}	B_{19}	B_{20}	B_{21}	B_{22}	B_{23}	B_{24}	B_{25}	B_{26}	B_{27}	B_{28}	B_{29}	B_{30}	B_{31}	B_{32}	B_{33}	B_{34}	B_{35}	B_{36}	B_{37}	B_{38}	B_{39}	B_{40}	B_{41}	B_{42}	B_{43}	B_{44}	B_{45}	B_{46}	B_{47}	B_{48}	B_{49}	B_{50}	B_{51}	B_{52}	B_{53}	B_{54}	B_{55}	B_{56}	B_{57}	B_{58}	B_{59}	B_{60}	B_{61}	B_{62}	B_{63}

"B" Register

REMAINDER

Let us now check our answer:

$$\text{The dividend} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$$

$$= \frac{15}{16} \text{ (base 10)}$$

$$\text{The divisor} = \frac{1}{2} + \ldots\ldots\ldots = \frac{1}{2} \text{ (base 10)}$$

$$\text{The expected quotient} = \frac{\frac{15}{16}}{\frac{1}{2}} = \frac{15}{8} \text{ (base 10)} = 1\,\frac{7}{8} \text{ (base 10)} = 1.111_2$$

The quotient is greater than 1. Since $Q_1=1$ the 49th quotient bit is ignored, but at the same time the exponent is increased by one and $Q_1=1$ remains in position 16 of B register. Increasing the exponent by one is equivalent to multiplying the mantissa by 2.

The quotient obtained is:

$$\text{quotient} = .1111_2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} = \frac{15}{16} \text{ (base 10)}$$

but with the exponent increased by one, the result is:

$$\text{quotient} = \frac{15}{16} \times 2 = \frac{15}{8} \text{ (base 10)} = 1\,\frac{7}{8} \text{ (base 10)}$$

which is that expected.

b) Exponent: In division the exponent of the divisor is subtracted from the exponent of the dividend and the result is placed in the exponent part of A register. As stated previously, the dividend is placed in A and B registers and the divisor in R register. At the beginning of the division process (Clock Time T1) the exponent of the divisor, through the operand select gates (OSG), is gated into the exponent part of B register which has just been cleared and allowed to be loaded with the exponent of R register. In Clock Time T2, the exponent remains unchanged because this time is used only to adjust the mantissa of B register when the option of rounding is used. In Clock Time T3 the TRUE output of the exponent part of A register is taken into CPA, while at the same time the COMPLEMENT output of the exponent part of B register is brought into CPA. The result of the addition (difference of two exponents) is brought into the exponent part of A register. If the option of normalization is not used this exponent is the final exponent of the quotient, unless the first bit of the quotient happens to be a one, in which case this exponent has to be increased by one. If normalization is to take place in Clock Time T3, the binary number 0 111 111 is placed in the OUTER exponent part of B register for exponent adjustment as was explained in the description of addition. In Clock Time T56 the exponent of A register is adjusted by the amount of correction bits from the exponent part of B register, which is directly related to the amount of places the leading ONE of the quotient filed was moved to the left in order to occupy bit position 16 of A register. The leading ONE detection and the insertion of the correction bits into the exponent part of B register is done at Clock Time T55.

Since the exponent is formed and subject to changes in different clock times of the division process, Table 39 summarizes the steps for the formation of the exponent.

It was said that the divisor must be normalized before the recursive process begins while the dividend need not be normalized. The need for normalization of the divisor stems from the need to establish a fixed reference point for the alignment of divisor and dividend or partial remainder and to allow determination of the difference in their magnitudes. This reference point corresponds to the binary point of the A and R registers.

The way the machine has been implemented if the leading one of the divisor is not placed at least one place to the left of the leading one of the dividend the result of the division will not be correct. But

Table 39. Steps of Exponent Manipulation in 64-Bit Mode Division

CLOCK TIME	ACTION TAKING PLACE	REMARKS
T1	Exponent of R register is transferred to the exponent part of B register. ½	
T2	This clock time does not concern the exponent.	
T3	Exponent of A and B registers is brought into CPA. The result of addition is brought back to exponent of A . The binary number 0 111 111 is inserted into OUTER exponent of "B" register. Tests for exponent overflow / underflow are made.	
T4-T51 T53-T54	No action concerned with the exponent occurs during these clock times.	
T52	Increase exponent of A by 1 if Q_i = 1 and set F bit if exponent overflow occurred.	
T55	The leading one detector determines amount exponent is to be adjusted. This amount is inserted into the exponent of B.	Only if normalized
T56	The exponent of A and B registers is brought into CPA. The result is transferred back to the exponent part of A register.	

when the comparison and the proper shifting of the divisor (so that its leading one, compared to the leading one of the dividend, would be one place to the left) are attempted it is much simpler to normalize the divisor, because the magnitude of the divisor does not change. This is true because, when we normalize the mantissa, the exponent of the divisor is reduced as much as the leading one was shifted to the left in order to be at bit position 16 of the R register. Another advantage obtained by normalizing the divisor is that, when bit position 16 of R register and is found to be a ZERO, it indicates that the divisor is ZERO because the divisor is assumed to be normalized. In this case the F bit is set to indicate that there is an exponent and mantissa overflow and therefore the result of the division is not correct. Another approach to the assurance of a correct result that could be implemented is, instead of shifting the leading one of the divisor at least one place to the left of the leading one of the dividend, the leading one of the dividend could be shifted one place to the right of the leading one of the divisor, but such would require extra programming time. When the divisor is normalized the dividend does not have to be less than the divisor but in some cases it must be less than twice the divisor.

There will be a case, however, when the programmer may decide not to normalize the divisor, even though he is aware that the F bit will be set anyway, because the machine has been implemented in such a way that if bit 16 of R register (divisor) is ZERO, the F bit is set. The process may continue and the result can be considered as correct if $X_0 < 2Y$. Since the F bit is set also when the divisor is ZERO, the programmer can check his result. In this case the quotient field will contain all ONES, because the divisor in 2's complement will produce a carry which will result in setting the quotient bit to ONE in every execution of the recursive process. The same result will be obtained even if the dividend contains all ZEROS provided the divisor is a ZERO number also.

It was also said that the dividend being 96 bits long is placed in A and B registers and the divisor, which is 48 bits long, is placed in R register. The reader may wonder why the dividend must be twice as long as the divisor and also what happens if the dividend is restricted into a 48 bit long register.

The dividend represents a number which can be used either in floating point or fixed point arithmetic, and therefore, it is best to examine each case separately.

1) Floating point: In multiplication, the final product of two operands 48 bits long comes out to be 96 bits long. When a program calls for division, this operation is executed after the operation calling for multiplication, if there is any, the result of which is 96 bits long, with its most significant bits placed in A register and the least significant bits in B register. Recalling that by definition the accuracy of a number is the number of bits which have significance, the dividend being 96 bits long allows accuracy in the division process consistent with that of other arithmetic operations performed by the PE. We can arrive at the same conclusion through the argument that, since it is desired that the quotient field be a 48 bit number as a result of division of a dividend X by a divisor Y 48 bits long and since the dividend can be obtained by multiplying the quotient by the divisor, provided the remainder is ZERO, the dividend X must be a 96 bit number. If accuracy is not of great concern, the dividend can be a 96 bit number with the most significant bits in A register and all ZEROS in B register (case of rounding). In this case

$$\left|\text{dividend } X_0\right| < \left| 2 \times \text{divisor } Y\right|$$

2) <u>Fixed Point Arithmetic</u>:

- Fractional number: In this case the A register can be filled with ZEROS, while B register will contain the number representing the dividend. The binary point may be considered to be between A and B registers. In this case

$$\left|\text{dividend } X_0\right| < \left|\text{ divisor } Y\right|$$

Also, B register might contain all ZEROS in which case the binary point will be considered to be at the left of bit 16 of A register.

- Integer: In this case proper scaling of the dividend must be performed in order to be able to execute the recursive process. Since the

$$\left|\text{dividend } X_0\right| > \left|\text{ divisor}\right|$$

the number, after being scaled, will occupy both registers (A and B). It is evident that the quotient will be an integer which cannot be held in the proper register. By appropriate scaling, the dividend is shifted to the right as many places as the programmer feels is needed, so that the result will be a number that will not impose an overflow condition. Of course, it will be the programmer's responsibility to adjust the quotient by shifting to the left the quotient field as many places as were imposed by the scaling factor. The binary point in this case is considered to be at the left of bit position 16 of A register.

Throughout this subsection shifting the dividend or partial remainder one place to the left, either through the PAT or CPA, has been mentioned often. But so far there has been no reference to PAT or CPA in regards to their possession of the capability for shifting their contents. The reader should recall the function of both CPA and PAT during the multiplication process. The PAT has three levels, each level receiving three inputs; namely, the partial sum, carry and word #1, 2, 3 depending on the level of the PAT. In division and precisely in the first clock time (T1) the content of C register is cleared and therefore, there is no carry coming into the first level of PAT. In division also the register for the recoding scheme (MDG) is not accessed and therefore only the content of A register is allowed to come into the first level of PAT. The PAT is designed in such a way that the input of the first level of the PAT, which is hard wire connected directly to A register, on the absence of carry and word #1, 2, 3 is directed to the third level of the PAT; the output of PAT is hard wire connected back to A register, but one place to the left compared to the output of A register connected to the first level of PAT. In other words bit position 63 of A register comes into bit position 71 of the first level of PAT and gets out of bit position 71 of the third level of PAT to go back to bit 62 of A register. The interconnection of CPA and A register is similar. A-register is wired to the CPA, the output of which goes back to A register, but one place to the left.

DIVISION IN 64-BIT MODE

CLOCK TIME **T1**

Transfer Exponent of "R" into "B" - Prepare SCR for Shifting

1. Clear RGC (0-63).
2. Enable COMPLEMENT OUT OF RGR EXP (0-15)*.
3. Enable TRUE AND COMPLEMENT OUT OF RGR mantissa (16-63)*.
4. Enable RGR (0-63) into OSG.
5. Clear RGB exponent and sign (0-15).
6. Enable load clocks into RGB exponent and sign (0-15).
7. Enable OSG into RGB (0-15).

IF ROUNDING

8. Clear shift count register (SCR).
9. Enable load clocks into SCR.
10. Enable shift right one from Common Data Bus into OSG.
11. Enable OSG into Address Adder (ADA) (Outer Exponent **
12. Enable ADA into Barrel Switch.

* Since the contents of RGR pass through OSG which is an inverter, in order to have the TRUE form of RGR out of OSG the COMPLEMENT of RGR is gated into OSG.

** Steps 10, 11, 12 are necessary, because "shift right one enable" into the shift count register is a CU decision and this is the correct route.

CLOCK TIME T2

If Rounding Transfer Mantissa of RGR into RGB Shifted to the Right End Off by One

1. Enable COMPLEMENT of RGR (16-63).

2. Enable RGR (16-63) into OSG.*

3. Enable OSG into LOG.

4. Enable LOG into Barrel Switch (16-63).

5. Enable OUT from shift count register.

6. Clear mantissa of RGB (16-63).

7. Enable load clocks into mantissa of RGB (16-63).

8. Enable Barrel Switch into RGB (16-63).

* The whole word of RGR is enabled into OSG, but since only the mantissa of RGB is enabled, the exponent part of RGR is already in RGB from the previous clock time and need not be inserted again. Therefore the mantissa of RGR shifted to the right by one is allowed to come into RGB.

CLOCK TIME T3

Computation of the Exponent

1. Enable COMPLEMENT of RGR (16-63).
2. Enable the WORD # 4 x 2 path through MSG.
3. Enable TRUE out of sign and exponent of RGA (0-15).
4. Enable TRUE out of sign of RGB (0).
5. Enable COMPLEMENT out of exponent of RGB (1-15).
6. Enable exponent of RGA into CPA (65-79).
7. Enable exponent of RGB into CPA (65-79).
8. Enable bit carries into CPA (0 -15).
9. Compute sign of RGA.
10. Clear exponent and sign of RGA (0-15).
11. Enable load clocks into RGA (0-15).
12. Restore sign of RGA.
13. Enable CPA (64-79) into RGA (90-15).
14. Clear R sign latch.
15. Clear Barrel Switch (shift counter register).
16. Enable load clocks into Barrel Switch (shift count register).
17. Enable shift left end around from CDB into OSG.
18. Enable OSG into ADA.
19. Enable ADA into Barrel Switch (shift count register).
20. Inhibit clear clocks into mode register.
21. Enable exponent underflow depending on values of E, E1.
22. Enable exponent underflow into mode register (decision of CU).

23. Enable exponent overflow into mode register.

24. Enable clear and load clocks to F bit.

25. Enable clear and load clocks into the INNER and OUTER underflow latches.

26. Clear OUTER exponent and sign of RGB (0-7).

27. Enable load clocks to OUTER exponent and sign of RGB (0-7).

28. Enable 00111111 into OUTER exponent and sign of RGB (0-7) for exponent correction during normalization.

29. Initialize iteration counter 47 times.

CLOCK TIME T4 - T51

Form the Quotient Field

1. Enable TRUE out of mantissa of RGA (16-63).
2. Enable COMPLEMENT out of mantissa of RGR (16-63).
3. Enable the WORD #4 x 2 path through MSG.*
4. Enable mantissa of RGA (16-63) into CPA (16-63).
5. Enable WORD # 4 x 2 into CPA (16-63).
6. Enable exponent of RGA (1-15) into CPA (65-79).
7. Enable bit carries into CPA (64-79).
8. Enable bit carries into the mantissa of CPA (16-63).
9. Enable TRUE out of shift count register (SCR).
10. Enable TRUE out of mantissa of RGB (16-63).
11. Enable RGB (16-63) into LOG (16-63).
12. Enable LOG (16-63) into Barrel Switch.**
13. Clear mantissa of RGB (16-63).
14. Enable load clocks into mantissa of RGB (16-63).
15. Enable Barrel Switch into RGB (shifted left one).
16. Enable quotient bit into least significant bit of RGB (63).
17. Clear mantissa of RGA (16-63).
18. Enable load clocks into mantissa of RGA (16-63).

* In step 3 WORD # 4 x 2 path through MSG must be enabled because this is the only way to get RGR (16-63) into CPA.

** In step 16 the whole word of LOG is enabled into Barrel Switch but since the load clocks of mantissa of RGB must be enabled, in reality the mantissa part of RGB will pass through the Barrel Switch and will go back to RGB shifted left one.

19. Enable PAT sum [RGA (16-63) shifted left one] into mantissa of RGA (16-63) if difference < 0.

20. Enable CPA sum into RGA (16-63) shifted left one if difference > 0.

21. Enable clear clock to R sign latch.

22. Enable load clock to R sign latch.

23. Test iteration and if the iteration counter has not counted 47 iterations repeat all steps T_4 - T_{51}. If the counter has counted 47 interactions then go to T_{52}.

24. Increment iteration counter after the above testing.

CLOCK TIME T52

Increase Exponent of RGA by One if $Q_1 = 1$

1. Enable COMPLEMENT out of RGR mantissa (16-63).
2. Enable the WORD # 4 x 2 path through the MSG.
3. Enable TRUE out of sign and exponent of RGA (0-15).
4. Enable COMPELEMENT out of sign and OUTER exponent of RGR (0-15).
5. Enable TRUE out of INNER mantissa of RGB (16-39) in order to see if bit 16 of RGB is a ONE.
6. Restore sign of RGA (0).
7. Enable exponent of RGA into CPA (65-79).
8. Enable sign and exponent of RGB into CPA (65-79).
9. Enable bit carries into CPA exponent (64-79).
10. Clear exponent of RGA if $Q_1 = 1$ (RBG bit 16 must be ONE in this case).
11. Enable load clocks to RGA if $Q_1 = 1$.
12. Enable CPA sum (64-79) into RGA (0-15).
13. Inhibit clear clocks to mode register.
14. Enable clear and load clocks to F bit.
15. Enable exponent overflow into mode register.

CLOCK TIME T53

Test Q_1 in Order to Determine Use of Q_{49}

1. Enable TRUE out of mantissa of RGA (16-63).
2. Enable COMPLEMENT out of mantissa of RGA (16-63).
3. Enable the WORD # 4 x 2 path through MSG.
4. Enable TRUE AND COMPLEMENT out of sign of RGB (0).
5. Force ONE from RGB (8) conditionally on R sign if FYEDITER-T or P----7I--1 have been enabled.
6. Force ONE from RGB (8) conditionally on R sign if FYEDITER-T or P----7I--1 have been enabled.
7. Enable mantissa of RGA into CPA (16-63).
8. Enable WORD # 4 mantissa into CPA (16-63).
9. Enable exponent of RGB into CPA (65-79).
10. Enable bit carries into CPA (16-79).
11. Enable output of shift count register.
12. Enable TRUE out of mantissa of RGB (16-63).
13. Enable RGB (16-63) into LOG (16-63).
14. Enable LOG (16-63) into Barrel Switch (16-63).
15. Enable clear clocks to mantissa of RGB (16-63) if bit 16 of RGB is ZERO ($Q_1 = 0$).
16. Enable load clocks to mantissa of RGB (16-63) if bit 16 or RGB is ZERO ($Q_1 = 0$).
17. Enable Barrel Switch into mantissa of RGB (16-63).
18. Enable Quotient bit into least significant bit (bit 63) of RGB.
19. Enable clear clocks to mantissa of RGA (16-63) if the difference > 0.

20. Enable load clocks to mantissa of RGA (16-63) if the difference > 0.

21. Enable CPA sum directly to RGA mantissa (16-63).

22. Inhibit clear clocks to mode register.

23. Enable clear and load clocks to F bit.

24. Enable clear and load clocks into mantissa of B register if bit 16 of B register is a ZERO ($Q_1 = 0$).

CLOCK TIME T54

Interchange Mantissas of RGA and RGB

1. Enable TRUE from mantissa of RGA (16-63).
2. Enable RGA into LOG (16-63).*
3. Enable LOG into the Barrel Switch.
4. Enable TRUE from mantissa of RGB (16-63).
5. Enable RGB into CPA (16-63).
6. Enable clear clocks into RGB mantissa (16-63).
7. Enable load clocks into RGB mantissa (16-63).
8. Enable Barrel Switch (which contains RGA mantissa) into RGB mantissa (16-63).
9. Enable clear clocks to RGA mantissa (16-63).
10. Enable load clocks into RGA mantissa (16-63).**
11. Enable CPA sum (which contains RGB mantissa) into RGA mantissa (16-63).
12. Enable clear clocks into RGA mantissa (16-63).**
13. Enable load clocks into RGA mantissa (16-63).**
14. Enable COMPLEMENT out of RGR INNER mantissa (16-39) in order to test whether bit 16 is ZERO or 1 and therefore to detect if the divisor is normalized or not.
15. Enable RGR (COMPLEMENT) into mode register for unnormalized divisor.
16. Inhibit clear clocks into mode register.
17. Enable clear and load clocks into F bit.

* In actuality the whole word of RGA is enabled into LOG but, since only the mantissa of RGA was enabled, only the mantissa part of LOG is effectively used.

** From steps 10 and 13 above it can be seen that: The mantissa of RGB is allowed to be transferred into the mantissa of RGA only if normalization is performed or the exponent underflow latch is low (contains ZERO) and normalization is not performed. However, step 12 clears the mantissa of RGA and therefore the mantissa of RGA contains ZEROS only if the exponent underflow latch is HIGH (contains ONE) and normalization is not performed.

CLOCK TIME T55

If Normalize, Adjust Exponent in Two Clock Times (T_{55}, T_{56}) Detect the Leading ONE of Mantissa of RGA and Shift Accordingly

1. Enable the leading one detector (LOD) for divide-64.
2. Enable TRUE out of RGA mantissa (16-63).
3. Enable RGA into LOG (16-63).
4. Enable LOG into Barrel Switch (16-63).
5. Clear LOD.
6. Enable load clocks into LOD.
7. Clear sign and exponent of RGB (0-15).
8. Enable load clocks into sign and exponent of RGB (0-15).
9. Enable 001111111 corrections bits into OUTER sign and exponent of RGB (0-7).
10. Enable LOD into INNER sign and exponent of RGB (8-15).
11. Clear mantissa of RGA (16-63).
12. Enable load clocks into RGA (16-63).
13. Enable Barrel Switch into mantissa of RGA (16-63). At this time the leading one of mantissa is at bit position 16 of RGA.

CLOCK TIME T56

Adjust Exponent of RGA, Check for Exponent Underflow

1. Enable TRUE out of RGA sign and exponent (0-15).
2. Enable COMPLEMENT of corrections bits of RGB (1-7) if there is OVERFLOW and bit 16 is ONE.
3. Enable TRUE of correction bits of RGB (1-7) if there is no OVERFLOW and bit 16 = 0.
4. Enable the INNER sign and exponent of RGB (8-15).
5. Enable RGA into CPA (64-79).
6. Enable RGB into CPA (64-79).
7. Enable bit carries into CPA (64-79).
8. Restore sign of RGA (0).
9. Clear sign and exponent of RGA (0-15).
10. Enable load clocks into sign and exponent of RGA (0-15) if there is no exponent underflow and mantissa is not ZERO or exponent underflow latch is low.
11. Enable CPA (64-79) into RGB (0-15).
12. Clear mantissa of RGA (16-63) if there is an overflow or the latch for exponent underflow is HIGH.
13. Failure to mode register conditional upon whether there is exponent underflow or the exponent underflow latch is HIGH and the mantissa $\neq 0$.
14. Enable exponent underflow into mode register (decision of CU).
15. Inhibit clear clocks into mode register.
16. Enable clear and load clocks into F bit.

4. Division in 32-Bit Mode. In this mode $E = E_1 = 1$ and therefore both OUTER and INNER words are enabled. This means that the A register contents are not protected, which is something that the programmer should always have in mind.

Since the recursive process was fully explained in 64-bit mode, and because essentially the same steps are used for the 32-bit mode, with the exception that more clock times are required for the completion of the division, only a summary of the actions being taken in each clock time is provided and the reader is urged to refer to the POSFILE for more detailed information.

CLOCK TIME	DESCRIPTION OF ACTIONS BEING TAKEN	REMARKS
T1	Clear RGC to allow proper use of PAT	
	Transfer OUTER sign and exponent of RGR into RGB	
	Transfer INNER sign and exponent of RGR into RGB	
	Prepare the SCR for shifting right by 1 end off	If rounding
T2	Shifted to the right by 1 end off, OUTER mantissa of RGR is transfered through the Barrel Switch into RGB	Only if rounding
T3	Shifted to the right by 1 end off, INNER mantissa of RGR is transfered through the Barrel Switch into RGB	Only if rounding
	Subtract INNER exponent of RGB from the INNER exponent of RGA and put the result into RGA	If do not ignore exponent
	Subtract OUTER exponent of RGB from the OUTER exponent of RGA and put the result into RGA	
	Enable INNER and OUTER signs into sign logic and restore the sign into the sign of RGA	If do not ignore signs
	Clear R sign latch	
	Check exponent overflow and underflow and set F, F_1 bits	If do not ignore exponent

	Insert 0111111 into OUTER exponent of RGB for exponent correction during normalization Shift left by 8 end around enable into SCR	
T4	Enable shift right 16 end around into the shift count register from CDB through OSG and ADA Set F bit if bit 16 of R register is a ZERO INNER mantissa of RGB is placed into the OUTER mantissa of RGB	This is CU decision See Table 40 for Inter-changing INNER & OUTER mantissas or RGB
T5	OUTER mantissa of RGB is placed into the INNER mantissa of RGB	
T6	The OUTER mantissa of RGA is transfered into the OUTER mantissa of RGB The OUTER mantissa of RGB is transfered into the OUTER mantissa of RGA Insert 0111111 $(077)_8)$ into the INNER exponent of RGB (which contains the INNER exponent of R register) Initialize iteration counter to count up to 25 Enable shift right 63 end around into the shift count register from Common Data Bus (CDB) through OSG and Address Adder (This is like shifting left by 1 end around)	This is CU decision
T7-T30	If the result of subtraction of INNER mantissa of R register from the INNER mantissa of A register is ≥ 0 then this result is transferred through the CPA shifted left by 1 into RGA. If the result is < 0 then the mantissa of RGA is transferred through the PAT shifted left by 1 back to RGA. Shift mantissa of RGB through the Barrel Switch left by 1 end around to provide space for the quotient bit. Transfer the most significant bit of RGB into the least significant bit of RGA Transfer the quotient bit into bit 63 of RGB (if the difference is ≥ 0 $Q_i = 1$, if the difference is < 0 then $Q_i = 0$)	
T31	Check bit 40 of RGB which contains Q_1. If $Q_1 = 1$ then increase the INNER exponent of RGA by 1 Enable F_1 bit if an exponent overflow occurred.	If do not ignore exponent

T32

Check Q_1 quotient bit. If $Q_1 = 0$ shift OUTER mantissa of RGB through the Barrel Switch left by 1 end around to provide space for Q_{25} into bit 63 of RGB.

If the result of the subtraction of INNER mantissa of R register from the INNER mantissa of A register is ≥ 0 then this result is brought back to RGA through the CPA but not shifted at all and Q_{25} is equal to 1.

If the result < 0 then $Q_{25} = 0$ and the remainder is the mantissa of RGA used for the 25th execution of the recursive process.

Check bit 40 of RGB. If it is a ONE enable F_1 to indicate fault, because in this case the remainder is invalid.

If ignore exponent

T33

Transfer INNER mantissa of RGA into INNER mantissa of RGB through Barrel Switch.

Transfer INNER mantissa of RGB into INNER mantissa of RGA through CPA.

Enable shift left by 8 end around into the shift count register from CDB through OSG and ADA.

This is CU decision

At this time the contents of A and B registers are as follows:

A REGISTER

0 7	8 15	16		39	40		63
OUTER EXP. of A REG.	INNER EXP. of A REG.	A_7	A_8	A_9	B_7	B_8	B_9

B REGISTER

0 7	8 15	16		39	40		63
077_8	077_8	REMAINDER			QUOTIENT		
		R_4	R_5	R_6	Q_4	Q_5	Q_6

PREPARE FOR DIVISION OF OUTER MANTISSA OF A & B REGISTERS BY THE OUTER MANTISSA OF R REGISTER

T34	Transfer the INNER mantissa of R into the OUTER mantissa of R register	
T35	Enable shift right 16 end around into shift count register from CDB through OSG and ADA	
T36	Transfer the OUTER word of R register into the INNER word of R register	See Table 41
T37	Enable shift right 63 end around into shift count register from CDB through OSG and ADA. Initialize iteration counter to count up to 23. Set F bit if bit 16 of R register is a ZERO because the divisor is assumed to be normalized before the division begins.	This is CU decision
T38-T61	If the result of subtraction of INNER mantissa of R register from the INNER mantissa of A register $\geqslant 0$ then this result is transferred through the CPA (WD # 4 x 2) shifted by one to the left into A register. If this result < 0 then the mantissa of A register through the PAT, but shifted by one to the left. Shift mantissa of B register through the Barrel Switch left by one end around to provide space for the quotient bit. Transfer the most significant bit of B register into the least significant bit of A register. Transfer the quotient bit into bit 63 of B register which will be $Q_2 = 1$ if the result of subtration $\geqslant 0$ or $Q_1 = 0$ if the result is < 0. At the end of clock time T_{61} the contents of A and B registers are as follows:	Remember that the mantissas have been interchanged

A REGISTER

A OUTER EXPONENT	A INNER EXPONENT	REMAINDER					
		R_7	R_8	R_9	R_4	R_5	R_6

B REGISTER

077_8	077_8	QUOTIENT					
		Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

T62	Check bit 40 of B register. If it is a ONE Q_1 of the OUTER quotient is equal to ONE in which case increase the OUTER exponent of A register by 1. Enable F bit if an exponent overflow occurred.	If do not ignore exponent
T63	Check Q_1 of OUTER quotient field. If $Q_1 = 0$ shift the OUTER mantissa (Q_7, Q_8, Q_9) of B register through the Barrel Switch left by 1 end around to provide space for Q_{25} of OUTER quotient field. In this case transfer Q_{25} into bit 63 of B register.	Q_1 is located at bit position 40 of B register
	If the result of the subtraction of the OUTER mantissa of R register from the OUTER mantissa of A register is $\geqslant 0$ then this result is brought back to A register through the CPA (WORD # 4 x 2) but not shifted to the left as in the previous clock times. In this case $Q_{25} = 1$. If the result is < 0 then $Q_{25} = 0$ and the remainder is the mantissa of A register used for the 25th step of the recursive process.	
	If bit 40 of B register ($Q_1 = 1$) is a ONE then set F bit to indicate fault because since the exponent is ignored the remainder will be invalid as has been previously explained ($X_0 \geqslant Y$ case).	If ignore exponent

T64

Transfer mantissa of A register into mantissa of B register through the Barrel Switch.

Transfer mantissa of B register into mantissa of A regicter through the CPA.

Enable shift left by 8 end around from CDB into shift count register through OSG and ADA.

This is CU decision

Enable clear and load clocks to F bit. At this time the contents of A & B registers are as follows:

A REGISTER

A OUTER EXPONENT	A INNER EXPONENT	QUOTIENT					
		Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

B REGISTER

077_8	077_8	REMAINDER					
		R_7	R_8	R_9	R_4	R_5	R_6

T65

Transfer INNER mantissa of B register into OUTER mantissa of B register.

Enable shift right 16 end around into shift count register from CDB through OSG and ADA.

See Table 40 This is CU decision

T66

Complete the transfer of INNER mantissa of B register into the OUTER mantissa of B register.

Clear OUTER exponent and mantissa of A register if do not normalize and the exponent underflow latch for the OUTER word is high (ONE).

Clear INNER exponent and mantissa of A register if do not NORMALIZE and the exponent underflow latch for the INNER word is high (ONE).

See Table 40

T67	Enable TRUE of INNER mantissa of A register into Barrel Switch through LOG.	Only if normalize
	Enable LOD to detect the leading ONE.	-11-
	Enable exponent adjustment into INNER exponent of B register.	-11-
	Enable Barrel Switch back to A register. At this time the contents of A & B registers are as follows:	-11-

A REGISTER

A OUTER EXPONENT	A INNER EXPONENT	NORMALIZED			UNNORMALIZED		
		Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

B REGISTER

077_8	EXPONENT ADJUSTED	REMAINDER					
		R_4	R_5	R_6	R_7	R_8	R_9

T68	Enable OUTER mantissa of A register into Barrel Switch through LOG.	Only if normalize
	Enable LOD to detect the leading ONE.	-11-
	Enable exponent adjustment into the OUTER exponent of B register.	-11-
	Enable Barrel Switch back to A register.	-11-
	Enable TRUE out of INNER exponent of A register and bring it into CPA.	Only if normalize
	Enable adusted exponent out of INNER exponent of B register and bring it into CPA.	-11-
	Enable CPA into INNER exponent of A register if: There is no exponent underflow, the exponent underflow latch for the INNER exponent is low, the INNER mantissa of A register is not ZERO and normalization takes place.	-11-
	If exponent underflow of INNER exponent (Exp. UF_1) has occurred and the INNER mantissa is not ZERO then the mode register indicates failure provided that F_1 has been set on	-11-

underflow and normalization takes place.

At this time the contents of A and B registers are as follows:

A REGISTER

A OUTER EXPONENT	A INNER ADJUSTED EXPONENT	NORMALIZED					
		Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

B REGISTER

EXPONENT ADJUSTED	EXPONENT ADJUSTED	REMAINDER					
		R_4	R_5	R_6	R_7	R_8	R_9

T69 Enable TRUE out of OUTER exponent of A register and the adjusted exponent out of OUTER exponent of B register, and bring both into CPA. Only if normalize

Enable CPA into OUTER exponent of A register if: -11-

There is no exponent underflow, the exponent underflow latch for the OUTER exponent is low, the OUTER mantissa of A register is not ZERO and normalization takes place. Only if normalize

If exponent underflow of OUTER exponent (Exp. UF) has occurred and the OUTER mantissa of A register is not ZERO then the mode register indicates failure provided F bit has been set on underflow and normalization takes place. -11-

The final contents of A and B registers are as follows:

A REGISTER

A OUTER ADJUSTED EXPONENT	A INNER ADJUSTED EXPONENT	NORMALIZED					
		Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

B REGISTER

ADJUSTED EXPONENT	ADJUSTED EXPONENT	REMAINDER					
		R_4	R_5	R_6	R_7	R_8	R_9

Table 40. Procedure for Interchanging INNER & OUTER Mantissas of RGB

	BYTES								CLOCK TIME
	A	B	C			D			
	0/1	2/3	4	5	6	7	8	9	
B REGISTER	077_8	IN. EXP. of R REGISTER	B_4	B_5	B_6	B_7	B_8	B_9	T3
LOG	077_8	IN. EXP. of R REGISTER	B_4	B_5	B_6	B_7	B_8	B_9	T4
BARREL SWITCH	B_6	IN. EXP. of R REGISTER	B_8	B_9	077_8	B_7	B_4	B_5	
SHIFT BY 8 LEFT END AROUND	IN. EXP. of R REGIS.	B_8	B_9	077	B_7	B_4	B_5	B_6	
B REGISTER	IN. EXP. of R REGIS.	B_8	B_9	077_8	B_7	B_4	B_5	B_6	
LOG	▨	B_8	B_9	077_8	B_7	▨	▨	▨	T5
BARREL SWITCH	B_7	B_8	B_9	077_8	B_7	B_8	B_9	077_8	
SHIFT RIGHT BY 16 END AROUND	B_9	077_8	B_7	B_8	B_9	077_8	B_7	B_8	
B REGISTER	▨	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	▨	▨	▨	
B REGISTER	IN. EXP. of R REGIS.	077_8	B_7	B_8	B_9	B_4	B_5	B_6	

NOTES:

1) **The shaded area indicates bytes which have not been enabled out of RGB and therefore at the end of Clock Time T5 they are found unchanged in their location in RGB.**

2) **B stands for B register and the subscripts 4, 5, 6, etc., indicate 8 bit bytes as they have been defined in the description of the organization of the word format.**

Table 41. Procedure for Interchanging INNER & OUTER Mantissas of RGR

	BYTES								CLOCK TIME
	A	B	4	5	6	7	8	9	
R REGISTER	R OUT. EXP. *	R IN EXP. *	R_4 *	R_5 *	R_6 *	R_7 *	R_8 *	R_9 *	T34
OSG	R OUT. EXP. *	R IN. EXP. *	R_4 *	R_5 *	R_6 *	R_7 *	R_8 *	R_9 *	
LOG	R OUT EXP.	R IN. EXP.	R_4	R_5	R_6	R_7	R_8	R_9	
BARREL SWITCH	R_6	R IN. EXP.	R_8	R_9	R OUT. EXP.	R_7	R_4	R_5	
SHIFT LEFT by 8 EA	R IN. EXP.	R_8	R_9	R OUT. EXP.	R_7	R_4	R_5	R_6	
R REGISTER	R IN. EXP.	R_8	R_9	R OUT. EXP.	R_7	R_4	R_5	R_6	
	ENABLE SHIFT RIGHT 16 END AROUND INTO SHIFT COUNT REGISTER FROM CDB THROUGH OSG & ADA								T35
OSG	R IN. EXP.	R_8	R_9	R OUT. EXP.	R_7	▨	▨	▨	T36
LOG	R IN. EXP.	R_8	R_9	R OUT. EXP.	R_7	R_8	R_9	R OUT. EXP.	
BARREL SWITCH	R_7	R_8	R_9	R OUT. EXP.	R_7	R_8	R_9	R OUT. EXP.	
SHIFT RIGHT 6, 16 EA	R 9	R OUT. EXP.	R_7	R_8	R_9	R OUT. EXP.	R_7	R_8	
	▨	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	▨	▨	▨	
R REGISTER	R IN. EXP.	R OUT. EXP.	R_7	R_8	R_9	R_4 *	R_5 *	R_6 *	

* The complement OUT of RGR is enabled because the OSG gates use negative logic and therefore in order to get a TRUE ouput from the OSG gates they must receive an input in COMPLEMENT form.

Table 42. Procedure for Interchanging INNER & OUTER Remainder (RGB)

	BYTES								CLOCK TIME
	A	B	4	5	6	7	8	9	
B REGISTER	077_8	077_8	REMAINDER						T65
			R_7	R_8	R_9	R_4	R_5	R_6	
LOG	077_8	077_8	R_7	R_8	R_9	R_4	R_5	R_6	
BARREL SWITCH	R_9	077_8	R_5	R_6	077_8	R_4	R_7	R_8	
SHIFT LEFT by 8 EA	077_8	R_5	R_6	077_8	R_4	R_7	R_8	R_9	
B REGISTER	077_8	R_5	R_6	077_8	R_4	R_7	R_8	R_9	
LOG	▨	R_5	R_6	077_8	R_4	▨	▨	▨	T66
BARREL SWITCH	R_4	R_5	R_6	077_8	R_4	R_5	R_6	077_8	
SHIFT RIGHT by 16 EA	R_6	077_8	R_4	R_5	R_6	077_8	R_4	R_5	
	▨	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	▨	▨	▨	
B	077_8	077_8	REMAINDER						
			R_4	R_5	R_6	R_7	R_8	R_9	

ACKNOWLEDGMENT

The author wishes to express his thanks to Terry Gore and George Marmarou of Burroughs Corporation for their explanatory remarks on the operation of the PE Logic elements. The cooperation of Carl Semmelhaack of Burroughs in clarifying the implementation of certain PE Instructions is greatly appreciated. The author also wishes to express his gratitude to Morris Knapp of the University of Illinois and Theodore Edlin of NASA/AMES Research Center for their administrative assistance which helped to bring this manuscript into existence. Special thanks are extended to Mrs. Cynthia Economidis of the University of Wisconsin in Milwaukee for her constructive suggestions regarding the development of this document and to Mrs. Mildred Pape and Mrs. JoAnn Kuehl for their invaluable help during the typing of the manuscript.

BIBLIOGRAPHY

[1] D. L. Slotnick, "The Fastest Computer," IEEE Transactions on Computers, V.C-17, No. 8, August 1968, pp 746-757

[2] "ILLIAC IV Systems Characteristics and Programming Manual," Burroughs Corporation, 66000D, IL4-PM1, Revised May 16, 1972

[3] I. Flores, "The Logic of Computer Arithmetic," Englewood Cliffs, N.J.: Prentice-Hall, 1963

[4] R. L. Davis, "The ILLIAC IV Processing Elements," IEEE Transactions on Computers, V-C-18, No. 9, September 1969, pp 800-816

[5] T. Economidis, "Principles of Operation of The ILLIAC IV Memory Logic Unit," NASA/AMES Research Center, December 1, 1971

[6] T. Economidis, "The ILLIAC IV Processing Element Memory," NASA/AMES Research Center, March 7, 1972

[7] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, Vol. EL-13, pp 14-17, February 1964

[8] J. E. Robertson, "A New Class of Digital Division Methods," IEEE Transactions on Electronic Computers, Vol. EC-7, pp 218-222

[9] T. Economidis, "Power Distribution to Illiac IV Computer" NASA/AMES Research Center, November 29, 1973