DESCRIPTION OF ARITHMETIC OPERATIONS IN THE ILLIAC IV PROCESSING ELEMENT

TABLE OF CONTENTS

INTRODUCTION

This manual was written in order to provide a source of basic information as to how programmed arithmetic operations are implemented within an ILLIAC IV Processing Element. A description of the addition, multiplication, and division operations is included together with a summary of individual register and logic network functions.

Most of the enclosed material was derived from descriptive write-ups compiled by T. Economidis of Automation Technology Incorporated.

SECTION 1

DESCRIPTION OF THE PROCESSING ELEMENT

The Processing Element (PE) of the ILLIAC IV executes the data computations and local indexing for operand fetches. It contains the following elements:

1) Five 64-bit registers (A,B,C,R,S) to hold operands and results. 'A' serves as the accumulator, holding one of the operands in arithmetic operations and receiving the output of the adder at the conclusion of the operation. The 'B' register serves as the second operand register in arithmetic operations (with the exception of multiplication). The 'C' register is used in certain instructions to save carries from the adder. The 'R' register holds the divisor and multiplicand in those operations as well as serving as a data routing register. The 'S' register is a general storage register.

2) A fast parallel-adder (CPA) which functions as either a carry propogating adder using three levels of look-ahead with four bits in the first group, four groups in the second section, and four sections in the final level (producing a 64-bit sum in a single clock period), or as a carry save adder.

3) A set of multiplicand select gates (MSG) which generate multiples of the multiplicand from a decoding of the eight least significant bits of the multiplier during a given iteration.

4) A logic unit (LOG) for Boolean functions.

5) A four-level shift network called the barrel switch (BSW) which permits 32-bit or 64-bit words to be shifted left, right, end-off, or end-around in one clock period.

6) A 16-bit index register (RGX) and adder (ADA) for memory address modification and control.

7) A parallel logic network, called the Leading One Detector (LOD), which generates information as to the amount of shifting needed in a normalization operation and a binary number to be used for exponent correction.

8) An 8-bit mode register (RGM) to hold the results of tests and the results of tests and the PE ENABLE/DISABLE state information.

9) A non-propogating three-level adder network, called the Pseudoadder Tree (PAT), each level of which accepts an addend, an augend, and a carry-in. Output of the third level consists of a sum and a carry.

SECTION 2
ADDITION AND SUBTRACTION

2.1 INTRODUCTION

Since the process of adding (or subtracting) two floating point numbers is the longest and most complicated of all arithmetic operations performed by the ILLIAC IV, it seems a logical starting point for this discussion of the arithmetic unit. Only addition will be specifically discussed herein since subtraction is merely addition with a complemented operand.

Floating point addition requires that the two operands have equal exponents. If they are not the same, the mantissa of the smaller number must be shifted right by as many places as the exponents differ. This process is known as alignment.

After addition, the leading one bit of the sum's mantissa may not be in the most significant bit position. If normalization was specified in the programmed instruction, the mantissa must be shifted left to this position and the exponent decremented correspondingly.

In the 64-bit mode, the contents of the A register are added to the contents of the B register. In the 32-bit mode, the outer mantissa of A is added to outer mantissa of B and the inner mantissas summed likewise. The sum is placed in A in both cases.

As for implementation, the operands are initially loaded into registers A and B. The difference in exponents is calculated in the CPA, which transmits that information to the shift count register. This register affects control of the PE's shift unit, the four-level barrel switch, which outputs a corrected mantissa to the register (A or B) that formerly contained the smaller exponent.

The actual addition of mantissas is performed in the CPA and the resulting sum is routed to the "accumulator", the A register. The questions of overflow and underflow as well as implementation of the normalization and rounding options will be discussed in the remaining text of this section.

The hardware elements in this section are discussed in the "chronological" order in which they are employed for addition.

2.2 CARRY PROPOGATE ADDER (CPA)

Addition operations, such as the initial comparing of exponents and the later summing of mantissas, are performed in the Carry Propogate Adder (CPA). As noted in Section 1, this element also contains a carry look-ahead adder which determines whether, based on the inputs to the CPA, there will be a carry. If there is a carry, it is fed back into the CPA which produces the final sum.

In order to accommodate 64-bit data words, the adder is divided into sixteen 4-bit groups and four 4-group sections.

Addition of two bits can produce a bit generate (if the inputs are "1" and "1") or a bit transmit (if the inputs are "0" and "1" or "1" and "0"). Therefore, it follows that each group can similarly produce a group generate (a carry originating within the group) or a group transmit (a carry passed along by the group). Group transmit will only occur if all inputs to the group are either of the form 0, 1 or 1, 0. Logically, the generate/transmit concept can also be extended to sections.

Operation of the Carry Propogate Adder can be summarized as follows:

Each adder bit position produces a bit generate and a bit transmit which feed into group carry look-aheads four bits at a time. Each group look-ahead in turn produces a group generate and group transmit which feed into section carry look-aheads four groups at a time. Each section look-ahead produces a section generate and section transmit. All section generates and transmits feed into all other sections in order to create carry input to each section. The carry into each section is:

$$C_1 = G_4 + G_3 \cdot T_4 + G_2 \cdot T_3 \cdot T_4 + G_1 \cdot T_2 \cdot T_3 \cdot T_4$$
$$C_2 = G_1 + G_4 \cdot T_1 + G_3 \cdot T_4 \cdot T_1 + G_2 \cdot T_3 \cdot T_4 \cdot T_1$$
$$C_3 = G_2 + G_1 \cdot T_2 + G_4 \cdot T_1 \cdot T_2 + G_3 \cdot T_4 \cdot T_1 \cdot T_2$$
$$C_4 = G_3 + G_2 \cdot T_3 + G_1 \cdot T_2 \cdot T_3 + G_4 \cdot T_1 \cdot T_2 \cdot T_3$$

The group carries are produced by the above input carry to the section and by the group generates and transmits that precede it within the section. These group carries are fed into the group look-ahead, which in turn produces bit carries. These bit carries are a function of the input carry to the group and the bit generates and transmits that precede it within the group. The bit carries are then fed into the adder, which in conjunction with the original inputs to the particular bit position, produces the output sum.

The symmetry of the adder is more clearly noticeable in Figure 1.

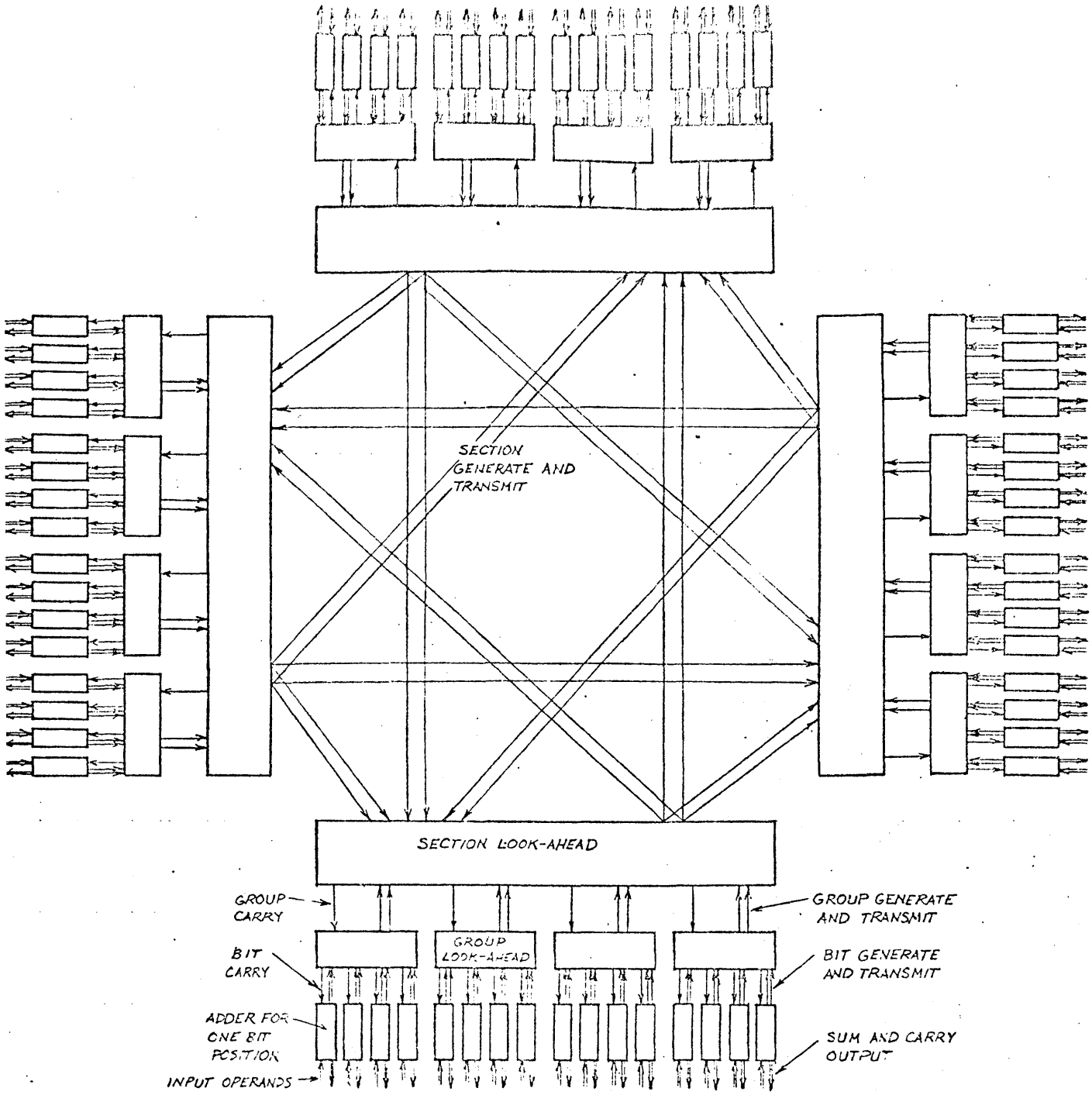


FIGURE 1

CARRY PROPOGATE ADDER BLOCK SCHEMATIC

2.3 SHIFT COUNTER (LOD #4)

After comparison of the two exponents, the CPA arrives at a number by which the mantissa of the smaller operand must be shifted right. The shifting itself is done in the Barrel Switch (which will be described in Section 2.4), but interpretation of this shift command into a form recognizable by the Barrel Shift control is done in a parallel logic network known as the Leading One Detector (LOD). Four sections of the network (LOD 1,2,3,5) are involved in normalization (Section 2.7). Section #4 of the network (LOD #4), however, is a 6-bit shift counter that decodes the CPA-derived shift amount into a set of proper shift signals for the Barrel Switch.

Before continuing, it should be noted that if the difference of exponents is greater than 47 in the 64-bit mode or 23 in the 32-bit mode, the mantissa of the smaller operand is forced to zero.

2.4 BARREL SWITCH (BSW)

The Barrel Switch is a shift network having four 64-bit levels that can shift a set of bits up to 63 positions, either left or right and either end-off or end-around, in one clock period. The first level of the Barrel Switch receives a parallel input of 64 bits from the Logic Unit (LOG). This level is essentially a buffer and has no shifting capability, but it can prohibit certain bits from entering the second level of the BSW. This feature is employed in implementing the "shift mantissa" instructions. The first level is also capable of swapping bytes within itself. This feature is used in the 32-bit mode division operations (Section 4).

The second level can shift in multiples of 16 bit positions, the third level in multiples of 4 bit positions, and the fourth level in multiples of 1 bit position. As previously stated, the amount of shifting (the multiple) for each level is controlled by decoded pairs of bits from the shift counter. Decoding of the two most significant bits results in multiples (0 to 3) of 16 to be applied to the second level and can affect shifts of 0, 16, 32, or 48 positions. The middle two bits control third level shifting and can result in a bit pattern being displaced by 0, 4, 8, or 12 positions from its second level position. Finally, the two least-significant bits of the shift counter are decoded and applied to the fourth level of the BSW to affect a shift of 0, 1, 2, or 3 bit positions.

For alignment, shifting is always to the right. If in taking the difference of the two exponents an end-around carry has been detected, the difference is indicated as positive and the true output of the shift counter is applied to the Barrel Switch controls. Otherwise the one's complement of the output of the shift counter is applied to the Barrel Switch controls.

It should be stated that left shift instructions are implemented by applying the above convention to the shift counter output but subtracting the shift amount from 64 before enabling BSW control.

2.4 BARREL SWITCH (BSW) (continued)

In order to better understand the implementation of a shift through the BSW hardware, consider a sample case where bit position 32 is to be shifted 27 positions to the right. The contents of bit 32 is input to level one of the BSW through the logic unit (LOG). It will then pass from the first level to bit 32 of the second level without any shifting having yet occurred. Level two receives a control signal from the shift counter to shift its contents once. Recall that a "1" multiple applied to the second level causes BSW control to shift level two right by 16 positions. Bit 32 is then shifted to bit position 48. Now bit position 48 is transferred to the third BSW level, where a shift-counter-originated "Multiple=2" signal enables BSW control to direct an 8-bit (2 x 4) shift operation. The contents of bit position 48 moves to bit 56. Now level three bit 56 is transferred to level four bit 56, a "multiple=3" signal enables BSW control to affect a 3-bit (3 x 1) shift, and bit 56 is shifted to bit position 59. The required 27-bit shift has now been completed. A schematic summary of the entire operation is given in Figure 2.

It should be kept in mind that although the example below focuses on displacement of only one bit, all bits entering levels 2, 3, and 4 are shifted by the same amount as that single bit.

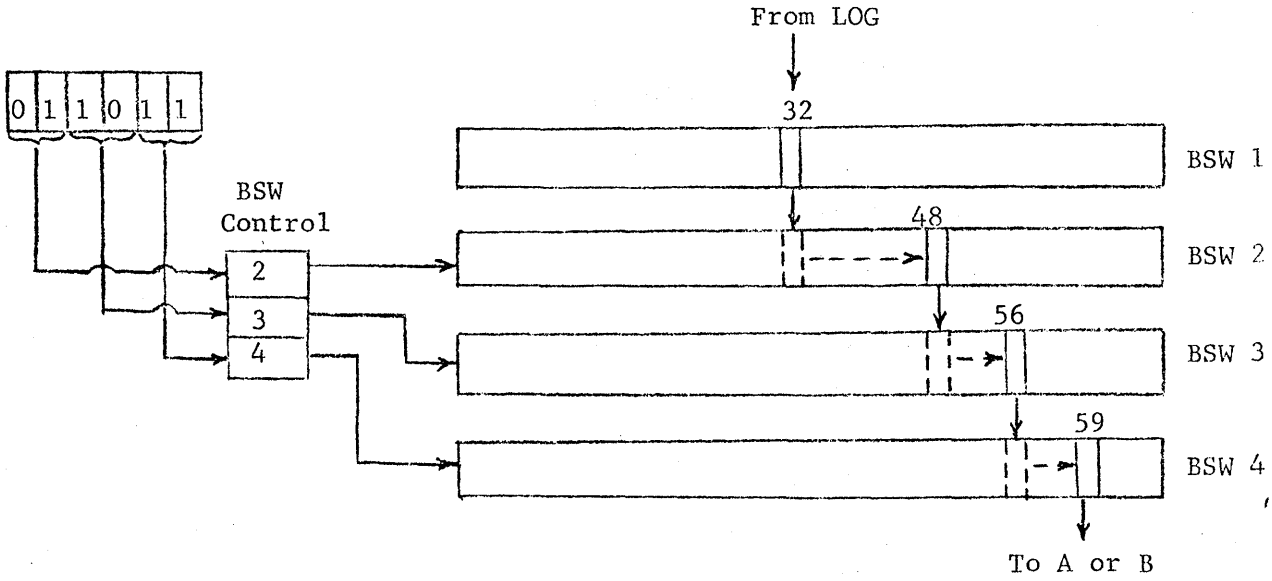


FIGURE 2. ILLUSTRATION OF BARREL SWITCH SHIFT

2.5 OVERFLOW

If two numbers of the same sign are added, the magnitude of the result might be greater than can be represented with the present exponent. This condition is called overflow.

Since floating point arithmetic is under consideration here, it becomes necessary to speak of both mantissa overflow (a carry into the exponent field) and exponent overflow (a carry into the sign bit).

Mantissa overflow can occur:

- o when the mantissas of addend and augend have the same sign
- o when rounding. If the signs of addend and augend are equal, the most significant shifted-off bit is added to the least significant bit of the sum.

Exponent overflow can occur:

- o when adding exponents of equal signs in order to find the difference for alignment
- o when mantissa overflow has occurred

Either kind of overflow will cause one of the F bits in the Mode Register (RGM) to be set. The correction of mantissa overflow is discussed in Section 2.7.

2.6 ROUNDING

Rounding is a programming option that is performed when the exponents of the two operands differ. Although the mantissa of the smaller operand has been right-shifted "end-off", use of the rounding option enables the most significant bit of that shifted-off bit string to be saved in a special latch (IMSB for 64-bit mode and 32-bit inner word, OMSB for 32-bit outer word).

If the operands agree in sign, the ISMB (OSMB) bit is added to the sum of the mantissas. If the operands differ in sign (a subtraction is taking place), the complement of ISMB (OSMB) is added to the final sum.

2.7 NORMALIZATION AND OVERFLOW CORRECTION

If normalization is requested in the programmed instruction, the first three sections of the Leading One Detector (LOD #1,2,3) are used to detect the position of the leading one in the mantissa. When one of these LOD's detects a leading one, it generates the controls required by the Barrel Switch in order to left-shift the mantissa's leading one to 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). LOD #1 and LOD #2 and LOD #3 control shifting in BSW levels 2, 3, and 4, respectively. They are enabled by LOD #5. If the mantissa in question is found to be already normalized as the result of the addition operation, LOD #5 disables LOD's 1, 2, and 3 and the Barrel Switch is set up for a zero shift.

In the case of mantissa overflow, LOD #5 disables LOD #1, 2 and enables LOD #3 for a one bit position end-off right shift.

Now that the mantissas have been properly manipulated, a discussion of exponent correction is required.

Exponent Correction in 64-Bit Mode

A fixed input of 00111111 is placed into the B register bit positions 0 to 7 and the contents of LOD #1,2,3 are enabled by LOD #5 into bit positions 10 to 15. Further, LOD #5 generates zeros into bits 8 and 9 in case of overflow or if the leading one is at bit position 16 and generates ones in any other case.

So, the exponent field of the B register is as follows for the 64-bit mode:

- o If overflow has occurred - 011 1111 0000 0001
- o If leading one is at bit position 16 - 011 1111 0000 0000
- o Normalization - 011 1111 11xx xxxx

Where the x's indicate normalization shift count.

Exponent Correction in 64-Bit Mode (continued)

The main criterion for the output of the B register into the CPA is the state of bits 8 and 9. If these bits are zero, then the complement of B register bits 1 to 7 is brought into the CPA, which means that the output of the B register bits 0-15 is:

- o If overflow has occurred - 0100 0000 0000 0001
- o If leading one is at bit position 16 - 0100 0000 0000 0000

If bits 8 and 9 are ones, the mantissa must be shifted to the left a certain number of places, which implies that this shift count must be subtracted from the exponent of the final sum. Since the true (uncomplemented) output of the B register is brought into the CPA for bits 8, 9 = 1, bit 1 is a zero and the exponent is negative.

Exponent Correction in 32-Bit Mode

For the 32-bit mode, the contents of LOD #1, 2, 3 are enabled by LOD #5 into bit positions 10 to 15 (for inner word) or 2 to 7 (for outer word). As for the 64-bit mode, LOD #5 generates a one in the exponent most significant bit (bit 1 or 9 for outer word or inner word, respectively) if overflow has occurred or if the mantissa requires no normalization.

So, the exponent field of the B register is as follows for the 32-bit mode:

- o If overflow has occurred, outer word - 100 0001 in bits 1-7
- o If overflow has occurred, inner word - 100 0001 in bits 9-15
- o If leading one is in bit position 40 (outer word) - 100 0000 in bits 1-7
- o If leading one is in bit position 16 (inner word) - 100 0000 in bits 9-15
- o Normalization - 0xx xxxx in bits 1-7 or 9-15

Where x's signify normalization shift count.

In the 32-bit mode the uncomplemented exponent is fed from the B register into the CPA. Observe that in the normalization case above, the leading zero in the exponent indicates how much the exponent of the sum must be reduced.

Exponent Underflow

If, in normalization, the value of the exponent is reduced below the minimum value that the register can accommodate, exponent underflow is said to have occurred. In such an instance, the entire contents of the A register is set to zero and the appropriate mode register F bit is set.

Normalization with Zero Mantissa

At this point, an interesting question arises. What will happen if the program calls for normalization and the mantissa of the sum is zero? It is apparent that in this case the LOD #1, 2, 3 won't detect a leading one and, therefore, the Barrel Switch will not be performing any mantissa shifting. Moreover, no correction bits will be inserted into the B register, which implies that the exponent will not be affected from that condition. Therefore, exponent underflow will not occur under any circumstances when attempting to normalize a zero mantissa.

In floating point arithmetic, a zero number is represented by a zero mantissa and the smallest exponent that the machine can hold. When the mantissa is zero, there is a signal called Zero Mantissa Level which inhibits the load clocks into the exponent field of the A register, while simultaneously the clear clocks are enabled to fill this field with zeros. Since the exponent is represented in excess code , zeros in the exponent field means that the number has the smallest possible exponent.

Normalization Reference Table

Table 1 indicates the mantissa shift amount and exponent correction for normalization in the 64-bit and 32-bit modes.

TABLE 1

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	0011111111111010	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	11101001

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					
OV2	-	0	0	0	1	- - - - - - - - - - - - - - - - -	0 1 0 0 0 0 0 1
40	0	24	32	8	0	0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0	0 1 0 0 0 0 0 0
41	0	24	32	4	3	0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0	0 0 1 1 1 1 1 1
42	0	24	32	4	2	0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0	0 0 1 1 1 1 1 0
43	0	24	32	4	1	0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1	0 0 1 1 1 1 0 1
44	0	24	32	4	0	0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0	0 0 1 1 1 1 0 0
45	0	24	32	0	3	0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1	0 0 1 1 1 0 1 1
46	0	24	32	0	2	0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0	0 0 1 1 1 0 1 0
47	0	24	32	0	1	0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1	0 0 1 1 1 0 0 1
48	0	24	32	0	0	0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0	0 0 1 1 1 0 0 0
49	0	24	16	12	3	0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1	0 0 1 1 0 1 1 1
50	0	24	16	12	2	0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0	0 0 1 1 0 1 1 0
511	0	24	16	12	1	0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1	0 0 1 1 0 1 0 1
52	0	24	16	12	0	0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0	0 0 1 1 0 1 0 0
53	0	24	16	8	3	0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1	0 0 1 1 0 0 1 1
54	0	24	16	8	2	0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0	0 0 1 1 0 0 1 0
55	0	24	16	8	1	0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1	0 0 1 1 0 0 0 1
56	0	24	16	8	0	0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0	0 0 1 1 0 0 0 0
57	0	24	16	4	3	0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1	0 0 1 0 1 1 1 1
58	0	24	16	4	2	0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0	0 0 1 0 1 1 1 0
59	0	24	16	4	1	0 0 1 1 1 1 1 1 1 1 0 0 1 1 0 1	0 0 1 0 1 1 0 1
60	0	24	16	4	0	0 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0	0 0 1 0 1 1 0 0
61	0	24	16	0	3	0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1	0 0 1 0 1 0 1 1
62	0	24	16	0	2	0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0	0 0 1 0 1 0 1 0
63	0	24	16	0	1	0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1	0 0 1 0 1 0 0 1

.8 TIMING OF ADDITION OPERATIONS

Following is a summary of the sequence of operations for addition in the 64-bit mode and 32-bit mode as a function of clock times:

Addition in 64-Bit Mode

T1 Take difference of exponents
T2 Save rounding bit
T3 Align mantissa having smaller exponent
T4 Add mantissas
T5 Complement, round, store overflow (if any)
T6 Complement, normalize, adjust exponent, determine sign
T7 Correct resultant exponent

Addition of 32-Bit Mode

T1 Take difference of inner word exponents
T2 Save rounding bit for inner word
T3 A) Take difference of outer word exponents
B) Align mantissa of inner word having smaller exponent
T4 Save rounding bit for outer word
T5 A) Align mantissa of outer word having smaller exponent
B) Add mantissas of inner word, store overflow (if any)
T6 Complement, round, store overflow (if any) of inner word
T7 A) Add mantissas of outer word, store overflow (if any)
B) Complement, normalize, adjust exponent of inner word
T8 Complement, round, adjust exponent of outer word
T9 Complement, normalize, adjust exponent of outer word
T10 Correct resultant exponent and sign of outer word

A detailed list of the sequence of operations occuring during each of the above clock times is given in Appendices A and B.

SECTION 3
MULTIPLICATION

3.1 INTRODUCTION

Multiplication is actually the addition of partial sums (to form the partial product), in which each partial sum is a multiple of the multiplicand. In floating point multiplication, the exponent of the product is the sum of the exponents of the operands.

Since multiplication time in the PE's is dependent on the fact that the control unit proceeds to the next operation only after all PE's in the quadrant have completed the multiplication, acceleration of the process is greatly dependent upon the reduction of the number of partial sums as well as the speed with which these sums are formed and the speed at which they are added.

As for implementation, the R register holds the multiplicand, the B register holds the multiplier, and the A register contains the partial product (and initially, the multiplicand). In the 64-bit mode, the A register will contain the 48 high-order bits of the final product and the 48 low-order bits will be found in the B register. In the 32-bit mode, if both E bits in the mode register are enabled, the A register will contain both 24-bit products while the B register will contain the 48-bit product of the outer word. If either or both E bits are disabled, the A register halfword will be properly restored and the B register will contain the 48-bit outer word product.

3.2 ADDITION OF EXPONENTS

In floating point multiplication, the exponents of the operands are added. The sign bit of the final exponent is determined by a carry (or absence of carry) out of the exponent field. Such a carry will exist if either a) both exponents are positive or, b) one exponent is positive and the other negative and their sum is a positive number.

The "carry" (0 or 1) is placed into a special circuit and brought back directly into the most significant bit position of the exponent. In that way, the exponent carry does not effect the sign of the mantissa.

3.3 RECORDING THE MULTIPLIER

It is established that binary multiplication with an n-bit multiplier conceptually takes n iterations. With multipliers of the order of 48 bits (64-bit mode mantissa) in the ILLIAC IV, a reduction in the number of iterations is essential to efficient operation. In the present design of the processing element, the number of iterations has been reduced to three for each of the words of the 32-bit mode and to six for the 64-bit mode.

Since the mantissa of a 32-bit word is comprised of 24 bits and that of a 64-bit word is 48 bits long, it is evident that one iteration must accomplish multiplication by 8 multiplier bits at a time. Such a feat is implemented as follows: In the first iteration, the contents of the B register (multiplier) mantissa are fed into the Logic Unit (LOG), which passes them in parallel to the Barrel Switch. Within the Barrle Switch, the mantissa is right-shifted end-off by 8 places. The shifted-off low-order 8 bits are routed to a network called the Multiplier Decoder Gates (MDG), where they are divided into 4 pairs. The 40 high-order mantissa bits are returned to the B register.

In subsequent iterations, the same shifting-decoding process is performed, but only the 40 low order bits from the B register are used. For each of those iterations, then, the Barrel Switch receives 40 bits every clock time and returns 32 bits to the B register.

Each pair of multiplier bits can be interpreted as follows:

- A bit pair 00 means that no addition of the multiplicand is required.
- A bit pair 01 means that the multiplicand must be added to the partial product.
- A bit pair 10 means that double the multiplicand (which is the multiplicand shifted left one position) must be added to the partial product.

3.3 RECODING THE MULTIPLIER (continued)

- A bit pair 11 means that the multiplicand must be multiplied by three before being added. This condition is implemented by subtracting the multiplicand from the partial product and then adding the multiplicand after shifting it left two positions (i.e., after multiplying it by four).

The above pairs, then, can be coded as multiples 0, 1, 2, and -1, respectively, with the latter implying an immediate subtraction and a carry into the next pair.

Considering the effects of a carry from a preceding pair, a "recoding" can be defined as follows:

PAIR	CARRY-IN	RECODED MULTIPLE	CARRY-OUT
00	0	0	0
01	0	1	0
10	0	2	0
11	0	-1	1
00	1	1	0
01	1	2	0
10	1	-1	1
11	1	0	1

The multiplier in the B register is partitioned as follows:

The 64-Bit Mode

Bit 63 does not participate in the bit-pairing process but is used to ~~check whether the initial loading of the multiplicand into the A register was correct (if it is one, the multiplicand remains in the A register).~~ gate, or not gate, the multiplicand into the A register as the initial partial product. The remaining 47 bits are grouped in pairs from the right and in six 4-pair sections as shown in Figure 3.

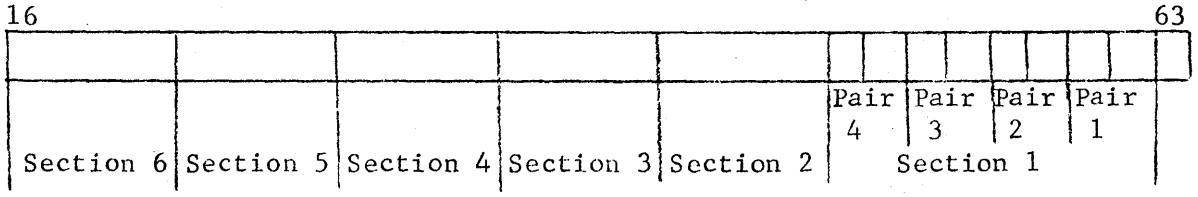


FIGURE 3. PARTITION OF 64-BIT MODE MULTIPLIER MANTISSA

3.3 RECODING THE MULTIPLIER (continued)

Since the mantissa contains 48 bits and bit 63 is not paired, it is clear that pair 4 of section 6 logically contains only one bit. However, to avoid having to worry about a possible carry into the exponent field, this "pair" should be thought of as an actual two-bit pair in which the leading bit is zero. So if bit 16 is a one and a carry occurs from Pair 3, Pair 4 will become 10 (an addition of twice the multiplicand to the partial product).

Each of the above sections require one iteration (one clock time) for the formation of the partial product.

The 32-Bit Mode

The partitioning process for multiplication in the 32-bit mode is similar to the process for 64-bit mode multiplication described above, except that bits 39 and 63 will be the initial multiplicand enables and outer and inner word mantissas will each be partitioned into three sections.

Application of Recoded Multiples

The actual recoding takes place in the MDG, where one multiple is generated for each of the four bit-pairs. These multiples are sent to the Multiplicand Select Gates (MSG), which apply each to the multiplicand from the R register. The MSG outputs four levels of shifted multiplicand, each level reflecting the result of applying one of the recoded multiples to the multiplicand. Clearly, each output level must consist of 49 bit positions in order to accomodate the "multiple = 2" case, where a 48-bit number is shifted left by one place.

The first three MSG output levels become input to the first three levels of the Pseudoadder Tree (Section 3.4), respectively. The fourth MSG output level, which is the result of applying the recoded bit-pair #4 to the multiplicand, is input to the Carry Propogate Adder.

3.4 PSEUDOADDER TREE (PAT)

Since multiplication is defined as successive additions of the multiplicand, it is evident that the use of a very fast adder is essential. However, instead of adding only two numbers to produce the sum, a new adder has been incorporated into the ILLIAC IV PE which adds three numbers (addend, augend, and carry-in) and produces not only a sum, but both a sum and a carry. This adder is called the Pseudoadder Tree (PAT).

The PAT has three levels and can accommodate 56 bits. Its high speed is due to the fact that there is no carry propogation since carry constitutes one of the three inputs to the next level of the tree.

The operations of each level of the PAT are as follows:

- In the first level of the Pseudoadder Tree, the 48-bit partial product from the A register is added to the first shifted multiplicand that is output from the MSG. The sum and carry output from this level is input to level two.
- The second level adds this first level input to the MSG result obtained by applying the second recoded bit-pair to the multiplicand. A new sum and carry is generated to serve as input to the third level.
- The third level of the PAT adds the MSG third output level to this input to produce a final sum and carry for the PAT.

At this point, six of the eight bits in a multiplier section have been applied to the multiplicand and the result has been added to the partial product. The resulting sum and carry of this operation comprise the output of the PAT. All that remains is to add this result to the output of the fourth MSG section, which is the multiplicand after it has been multiplied by the recoded bit-pair #4. This addition is performed in the Carry Propogate Adder. It can be observed that the CPA is actually behaving as a fourth level of the Pseudoadder.

3.4 PSEUDOADDER TREE (PAT) (continued)

The output of the CPA is the final 56-bit sum and carry of one multiplication iteration. The 48 high-order bits of the sum are loaded into the A register mantissa and the 8 low-order bits are loaded into the B register mantissa. Carry is fed into the C register for input to the first level of the Pseudoadder in the next iteration. Of course, in the initial iteration the C register contains all zeros and therefore contributes no carry to the PAT.

To aid understanding, the process described above is schematically represented in Figure 4.

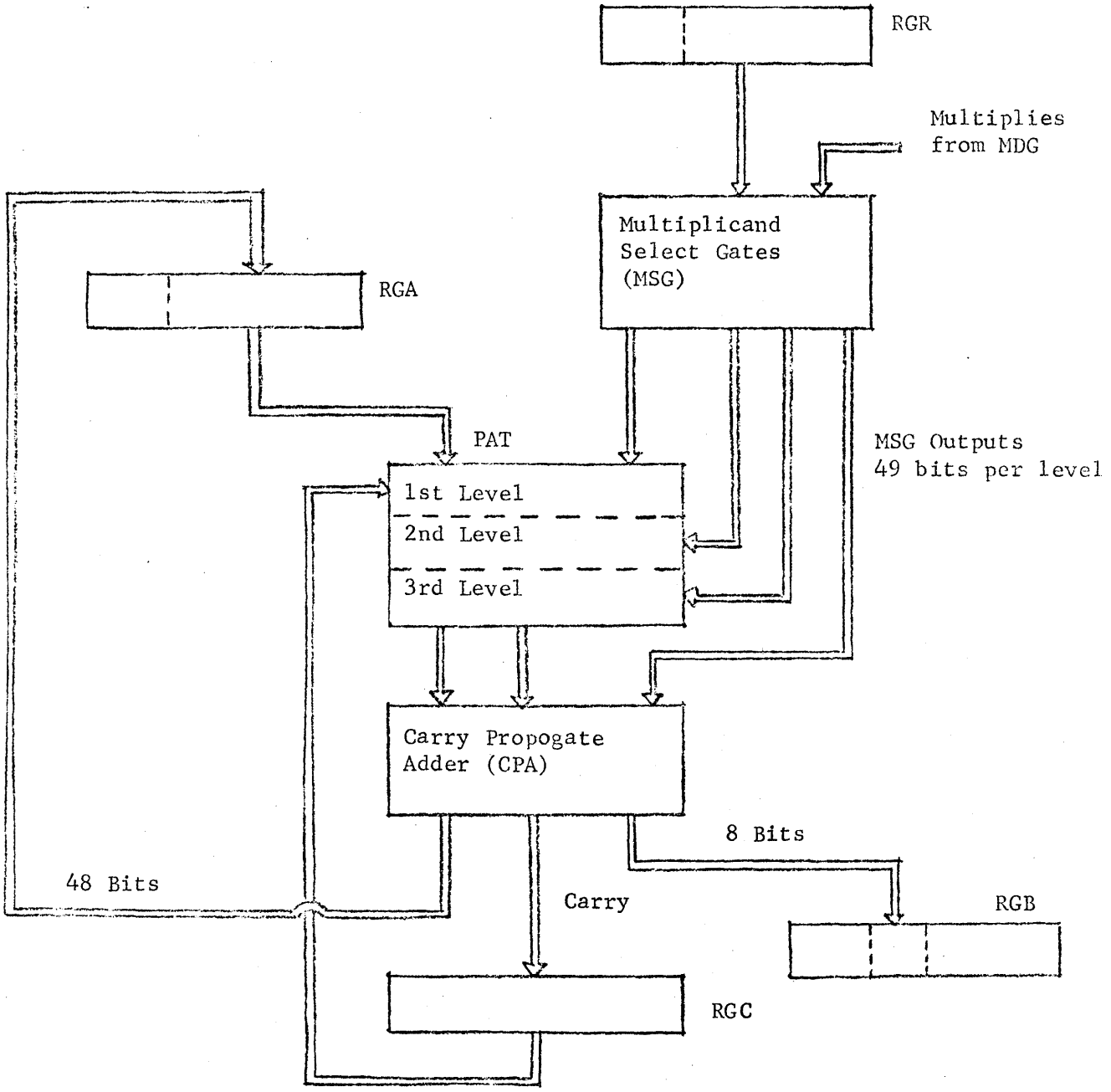


Figure 4. Schematic Representation of Multiplication

RGA - Partial Product
RGB - Multiplier
RGR - Multiplicand

3.5 TIMING OF MULTIPLICATION OPERATIONS IN 64-BIT MODE

Following is a summary of the sequence of operations for multiplication in the 64-bit mode as a function of clock times:

T1 A) Calculate Exponent
B) Recode 8 bits of multiplier mantissa for first iteration

T2 Recode 8 bits of multiplier mantissa for second iteration

T3 Recode 8 bits of multiplier mantissa for third iteration

T4 Recode 8 bits of multiplier mantissa for fourth iteration

T5 Recode 8 bits of multiplier mantissa for fifth iteration

T6 Recode 8 bits of multiplier mantissa for sixth iteration

T7 Sixth iteration

T8 Form the final product

T9 Normalize final product

A detailed list of the sequence of operations occuring during each of the above clock times is given in Appendix C.

SECTION 4

DIVISION

4.1 INTRODUCTION

There are several methods for performing the division operation. Among the most common are "restoring", "non-restoring", and "non-performing". In the restoring method, the divisor is successively subtracted from the high-order bits of the dividend; the result replaces the dividend. The quotient is increased for each successful subtraction, i.e., for each subtraction yielding a positive result. When a negative result of subtraction is obtained, the dividend is restored by adding the divisor to it. The dividend and the quotient are then shifted left one position and the process is repeated.

Non-restoring division is based on the observation that a subtraction yielding a negative result in the above method requires a restore operation (i.e., an addition of the divisor) followed by a subtraction of the divisor during the next iteration (with the divisor shifted one place to the right, or divided by two). The two operations "add present divisor" and "subtract one-half present divisor" can be combined to a single operation "add one-half of present divisor". From a hardware viewpoint, however, this process requires extra logic gates to pick up locally the TRUE or COMPLEMENT output of the divisor register and implies added cost and complexity of the processing element.

The ILLIAC IV employs an altered restoring method known as the non-performing method. Recall that in the restoring method, a one is entered in the quotient for a positive subtraction result and a zero is entered otherwise. This rule only applies to positive quotients, however. If the sign of the quotient is negative (i.e., if divisor and dividend have opposite signs), the opposite is performed (zero for positive subtraction result, one otherwise). In non-performing division, the sign of the result is considered. If this sign is the same as that of the present partial remainder, a one is entered in the quotient bit if the quotient is positive and a zero is entered if the quotient is negative. If the sign of the subtraction result differs from that of the present partial remainder, the result of the subtraction is ignored and the partial remainder is shifted left one place. A zero is then added to the quotient if the quotient is positive and a one is added to a negative quotient.

.1 INTRODUCTION (continued)

As for register allocation, the double-length dividend is initially in registers A and B. The normalized divisor is in the R register. Following each iteration, the one-bit quotient is loaded into the B register at position 63 and the remainder is retained in the A register. At the end of the required 48 iterations, the quotient is in the B register and the remainder is in A. The contents of these two registers are then swapped so that in the final form, the 48-bit quotient is in the A register.

.2 IMPLEMENTATION OF THE DIVISION OPERATION

As previously stated, the mantissa of the dividend is of a length double that of a single register's mantissa field. For the 64-bit mode, then, a dividend mantissa is 96 bits long. These bits initially occupy both the A and B registers with the 48 high-order bits located in the A register. The dividend need not be normalized. The divisor, located in the R register is 48 bits long and must be in normalized form before the division process is started.

Since the recursive division process requires subtraction of the divisor from the dividend, the one's complement of the divisor is taken into the CPA where a one is added to it to form the two's complement. The divisor, now in complemented form, is added to the dividend. If the subtraction is successful (i.e., if the result is positive), then the quotient bit is a one. Otherwise the quotient bit is a zero.

The system hardware puts a one in the quotient bit if either:

- o The "RSIGN" latch is set, or
- o The result of the subtraction of divisor from dividend causes a carry out of the most significant bit of the adder (positive result of subtraction).

That is, $Q = RSIGN_{i+1} + GC16$ GC16 = Carry Out of the MSB

The "RSIGN" latch can be thought of as a one-bit left extension of the A register. It is set when bit position 16 of either the A register or the CPA are one and the previous quotient bit was one.

That is: $RSIGN_{i+1} = A16 \cdot Q_i + CPA16 \cdot Q_i$

This specification is predictable since, given the fact that the divisor is always normalized, a successful subtraction requires a one in the most significant bit position (Bit 16) of the partial remainder.

The RSIGN latch is initialized to zero, so the quotient for the first iteration is entirely dependent on whether the subtraction of divisor from dividend produces an overflow. That is, on whether the result is positive.

4.2 IMPLEMENTATION OF THE DIVISION OPERATION (continued)

Subsequent iterations require consideration of the previous result. If $Q_i = 1$, the preceding subtraction must have been successful. Therefore, the remainder is taken through the CPA and loaded back into the A register after being shifted left end-off by one place.

If $Q_i = 0$, however, a negative remainder is indicated. Thus, the result of the subtraction (remainder) is ignored and the current dividend from the A register is passed through the PAT and, after being shifted to the left by one position, is re-entered in the A register.

It should be explained at this point that neither the CPA nor the PAT have any extensive shift capability built into them. Their ability to affect the above single position left-shifts is as follows: The PAT has three levels, each level receiving three inputs (sum, carry, and the shifted R register contents). In division, the C register is cleared in clock time T1 and therefore, no carry enters the first level. Further, the division process, unlike multiplication, does not access the MDG recoding network. So the contents of the A register constitute the only input to the PAT first level. The PAT has been designed so that in the absence of MDG and carry-in, the contents of the A register is directed to the PAT third level. This level is hard-wire connected back to the A register, but displaced one position to the left relative to the original position. That is, bit 63 becomes bit 62, 62 becomes 61, etc.

The CPA is constructed in the same way. The A register is hard-wired to the CPA, the output of which goes back to the A register, but displaced one position to the left.

Each time the A register mantissa contents are left-shifted, the B register mantissa contents are also left-shifted (but in the Barrel Switch instead of the PAT or CPA). Thus, B register bit 16 is transferred to A register bit 63 and the now-vacant B register bit 63 receives the quotient bit.

.2 IMPLEMENTATION OF THE DIVISION OPERATION (continued)

Following the 48th iteration, the dividend has undergone 47 left shifts and quotient Q_1 is in bit location 16 of the B register. The last subtraction has determined Q_{48}. If $Q_{48} = 1$, the subtraction result is shifted left and then loaded back into the A register to become the partial remainder for the 49th iteration of the recursive process. If $Q_{48} = 0$, the previous remainder is shifted to the left by one place and then returned to the A register.

In the next clock time, no execution takes place, but a test is made to determine whether the original dividend was greater or equal to the divisor. If bit position 16 of the B register contains a one ($Q_1 = 1$) the exponent in A is increased by one. If this bit is zero ($Q_1 = 0$), the exponent remains unchanged. The latter indicates that one more bit of precision can be added to the quotient.

At the end of the 49th iteration (clock time T53), the contents of Q_1 is interpreted as regards the quotient. If $Q_1 = 1$, the B register is blocked to prevent insertion of Q_{49}. If $Q_1 = 0$, however, the quotient is shifted left one position and Q_{49} is enabled into B register bit position 63.

Division is now complete and the mantissas of the A and B registers are interchanged, leaving the final quotient in the A register and the final remainder in the B register.

.3 SUBTRACTION OF EXPONENTS

In division, the exponent of the divisor is subtracted from the exponent of the dividend and the result is placed into the exponent part of the A register. At the beginning of the division process (clock time T1), the exponent of the divisor (R register) is passed through the Operand Select Gates (OSG) and is loaded into the exponent part of the B register. Then, at clock time T3 (T2 is used for mantissa adjustment), the subtraction of A and B register exponents is performed in CPA and the result is gated back into the A register.

Nothing further is done with the exponent until the 48th mantissa division iteration has been completed. At that time (T52), the exponent in A is incremented by one if the first quotient bit (Q_1) was a one. If this incrementation produces exponent overflow, the F (overflow) bit of the mode register is set.

.4 NORMALIZATION

The process of mantissa normalization and corresponding exponent correction for division is essentially the same as described in Section 2.7 for the addition operation.

If normalization is specified in the programmed 64-bit mode division instruction, the binary number 00 111 111 is placed into bit positions 0 to 7 of the B register (clock time T3) for exponent adjustment. At clock time T55, the leading one detector determines the amount of shifting that will be necessary for mantissa adjustment. The Barrel Switch shifts the A mantissa by this amount and the shift count is inserted into the exponent of the B register. At T56, the A and B register exponents are added in the Carry Propogate Adder with the sum transferred back into the A register exponent field. If exponent underflow occurred, the entire A register is set to zero.

A similar procedure is followed in 32-bit mode division. The exponent correction factors are entered into the B register at clock times T3 and T6 (for outer and inner word, respectively). The mantissas are normalized and exponents are adjusted in clock times T67 to T69.

.5 TIMING OF DIVISION OPERATIONS

Following is a summary of the sequence of operations for division in the 64-bit mode and 32-bit mode as a function of clock times:

Division in 64-Bit Mode

T1 Transfer exponent of R register into B register

T2 If rounding, transfer mantissa of R register into B register shifted end-off to the right by one

T3 Calculate exponent

T4 - T51 Form the quotient field

T52 Increment exponent of A register if $Q_1 = 1$

T53 Test Q_1 in order to determine use of Q_{49}

T54 Interchange mantissas of Register A and Register B

T55 } If normalizing, detect leading one of Register A mantissa and shift
T56 } accordingly.

T56 Adjust exponent of Register A, check for exponent underflow

Division in 32-Bit Mode

T1 Transfer exponents (inner and outer) of R register into B register

T2 If rounding, transfer outer mantissa of R register into B register shifted end-off to the right by one

T3 A) If rounding, transfer shifted inner mantissa of R register into B register as in T2
B) Calculate exponents

T4-T5 Interchange outer and inner mantissas of B register

T6 Interchange outer mantissas of A and B registers

T7-T30 Form the Quotient field for inner words

T31 Increment inner exponent of A register if $Q_1 = 1$

T32 Test Q_1 in order to determine use of Q_{25}

T33 Interchange inner mantissas of A and B registers

T34-T36 Interchange inner and outer mantissas of R register

T37 Check for non-normalized divisor in R register

T38-T61 Form the quotient field for inner words (recalling inner and outer words have been interchanged)

4.5 TIMING OF DIVISION OPERATIONS (continued)

T62 Increment outer exponent of A register if $Q_1 = 1$

T63 Test Q_1 in order to determine use of Q_{25}

T64 Interchange mantissas of A and B registers

T65-T66 Interchange inner and outer mantissas of B register

T66 Clear inner or outer exponent and mantissa of A register if exponent underflow

T67 If normalizing, detect leading one of Register A inner mantissa and shift accordingly

T68 A) If normalizing, detect leading one of register A outer mantissa and shift accordingly
B) Adjust inner exponent of Register A, check for exponent underflow

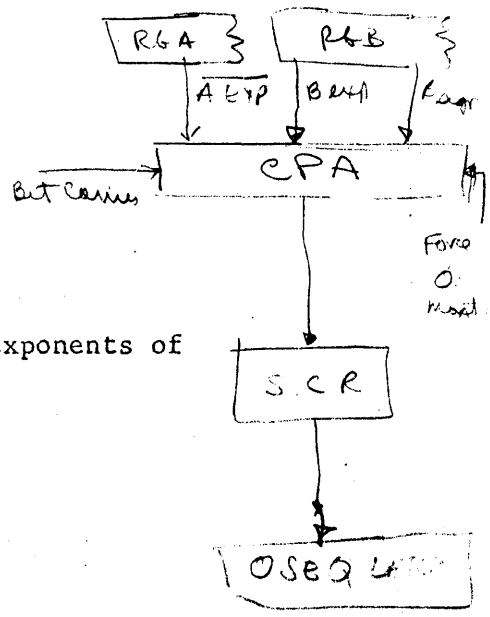
T69 Adjust outer exponent of Register A, check for exponent underflow

A detailed list of the sequence of operations occuring during each of the above clock times is given in Appendices D and E.

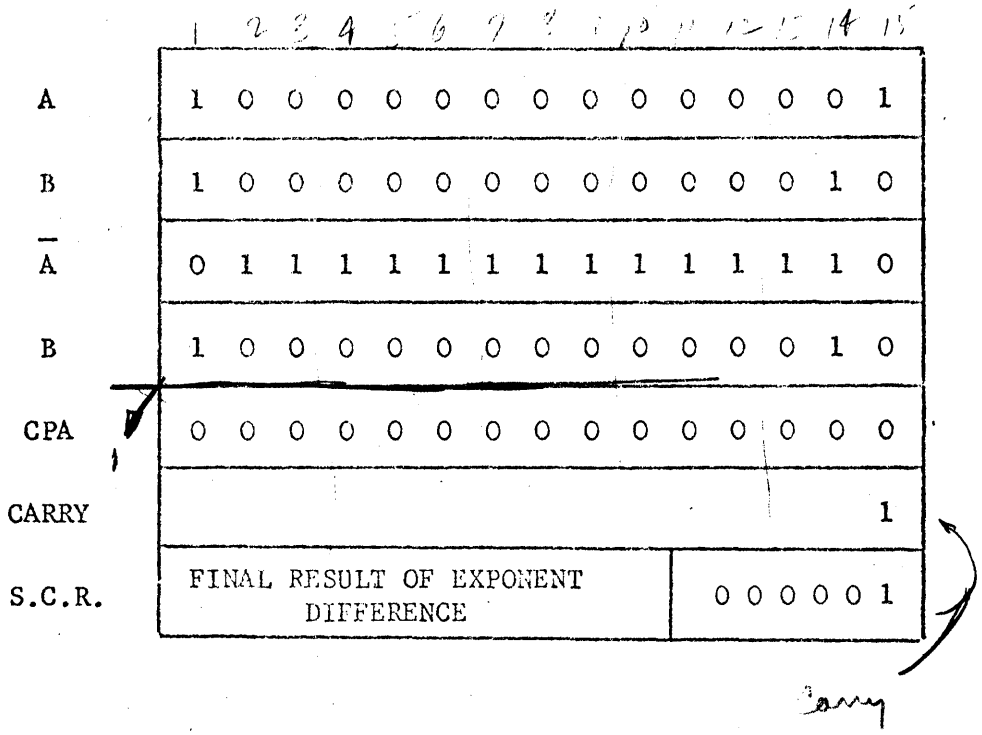
APPENDIX A: ADDITION IN 64 BIT MODE

T1 EXPONENT DIFFERENCE

1. - Enable true out of exponent of B
2. - Enable complement out of exponent of A
3. - Enable complement out of exponent of A into CPA
4. - Enable true out of exponent of B into CPA
5. - Enable the sign of B mantissa into CPA
6. - Enable the bit carries because of the addition of exponents of B & A into CPA
7. - Force zeroes into the mantissa part of CPA
8. - Clear and load clocks into S.C.R. (LOD #4)
9. - Put exponent part of CPA into SCR(8-15)
10. - Clear and load clocks into LOD
11. - Clear and load the latch for OSEQ



EXAMPLE OF TI OPERATIONS

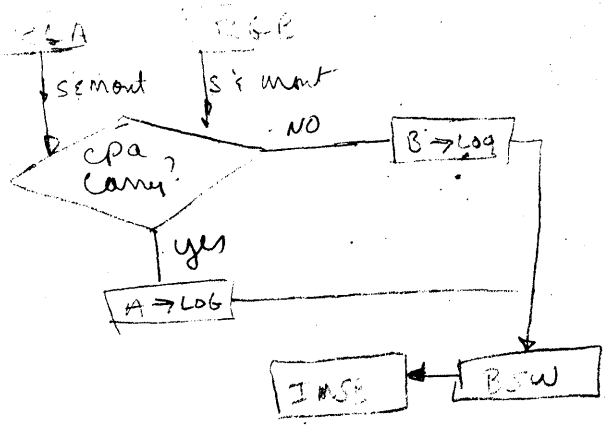


T2

ROUNDING (Optional)

1. - Enable true out of sign and mantissa of A
2. - Enable true out of sign and mantissa of B
3. - a) Enable A into LOG if there is (from exp. addition) a carry and therefore if exponent of A < exponent of B

   b) Enable B into LOG if there is no carry and therefore exponent A > exponent of B
4. - Enable the 64 shift counter from SCR
5. - Enable force shift left (control for end-off)
6. - Transfer LOG into BSW (bits 16-63)
7. - Load most significnat shifted-off bit into IMSB latch



T3 ALIGN MANTISSA HAVING SMALLER EXPONENT

1. - Enable complement of signs of A & B if V.....11..
2. - Enable true of signs of A & B
3. - Enable true of mantissas of A & B
4. - Load sign and mantissa of A into LOG if there is a carry
5. - Load sign and mantissa of B into LOG if there is no carry
6. - Enable 64 shift counter from S.C.R.
   a) True out of S.C.R. if thereis a carry
   b) Complement of SCR if there is no carry
7. - Transfer LOG into BSW (16-63 bits)
8. - a) Transfer the BSW into A (16-64 bits) if there is a carry
   b) Transfer the BSW into B (0-15, 16-63 bits) if there is no carry
9. - a) Load A mantissa if there is a carry
   b) Load B mantissa if there is no carry
10. - Enable exponent of B (1-15 bits)
11. - Restore sign of A (0 bit)
12. - Enable B exponent into the exponent part of A
13. - Clear and load the sign and exponent part of A.

T4

ADDITION OF MANTISSAS

1. - Enable true out of mantissa of B if mantissa signs equal
2. - Enable complement out of B if mantissa signs are unequal
3. - Enable true out of sign and mantissa 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 resulting from the addition of the mantissa into CPA.
8. - Transfer sign of mantissa of B into CPA
9. - Transfer the exponent of A into 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 to OV1

T5 COMPLEMENT, ROUND, STORE OVERFLOW

1. Enable the true of mantissa of A if WCMP latch is low.
2. Enable the complement of mantissa of A if WCMP latch is high.
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).
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 ROUND and add 1 to MSB of mantissa and WCMP latch is low.

   b) Force SGE = 0 and STE = 0 IF ROUND and subtract 1 from the MSB of mantissa and WCMP latch is low.
9. Set overflow OV1.
10. Enable the mantissa sign of B into CPA.
11. Enable the mantissa part of CPA into A.

T6 COMPLEMENT, NORMALIZE, ADJUST EXPONENT, DETERMINE SIGN

IF NORMALIZATION

1. Enable RGA (16-63) if WCMP latch = 0.
2. Enable compliment of RGA (16-63) if WCMP latch = 1.
3. Enable RGA (16-63) into LOD when normalizing.
4. Enable RGA (16-63) into LOG.
5. Enable LOG (16-63) into B SW.
6. Enable exponent correction bits into RGB (B-15) when normalizing.
7. Enable exponent correction bits into RGB (B-15) when normalizing.
8. Enable clear and load clocks into RGB (0-7).
9. Enable 00 111 111 into RGB (0-7) for exponent correction.
10. Load clocks for F.

IF NO NORMALIZATION

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 OVI.

T7 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 not a ONE.
4. Enable complement of RGB (1-7) if there is:
   a) Overflow or
   b) Bit 16 is a ONE.
5. Enable RGA (1-63) into CPA (65-79, 16-63).
6. Enable RGB (1-63) into CPA (65-79, 16-63).
7. Enable bit carries into CPA (16-63, 65-79).
8. Clear mantissa of RGA if there is exponent underflow or overflow (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).

T_1 DIFFERENCES OF EXPONENTS 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 (LOD #4)
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.

NUMERICAL EXAMPLE

Bit Position			
RGA	1 0 0 0 1 0 1 = $5)_{10}$	Decimal 5
RGB	1 0 0 0 0 0 1 = $1)_{10}$	
RGA	0 1 1 1 0 1 0	
CPA	1 1 1 1 0 1 1	
SCR	1 1 1 0 1 1	
SCR	0 0 0 1 0 0 = $4)_{10}$	Difference

T2 SAVE MSB TO BE SHIFTED OFF

FOR ROUNDING IN INNER WORD

1. Enable true out of m. sign and m. mant. of A (8,16-39)
2. Enable true out of m. sign and m. 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 counter from SCR
   a) SCR true out if there is a carry
   b) SCR comlement 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 we enable in LOG bits (0-39) but we effectively place the bits (0-39) of LOG into 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 a postion 64.

** Enabling is required so that the no-carry case can produce a right shift that has the appearance of a desired left shift.

T3 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 /// /// 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.

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.
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).

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

T4 SAVE M.S.P. 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 LOC 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.

T5 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.
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.

B. ADD MANTISSAS OF INNER WORD STORE OVERFLOW (OV_1) (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 CV.
3. Enable true out of inner mantissa of RGB if P--ISEQ--H.
4. Enable compliment 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 OVI (for inner word).
17. Enable true output of RGA inner sign.

T6 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 is low
2. - Enable compliment of inner mantissa of RGA (16-39) if inner CMP latch 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 zeroes 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)

T7 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 is high
2. - Enable complement out of RGB (40-63) if Outer Sign Equal latch 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 zeroes 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

B. - COMPLIMENT (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 is low
3. - Enable compliment out of RGA (16-39) if Inner Sign Equal latch 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 andload 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 LOP (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)

T8 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 of 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 (See Section 2.6)
10. - Enable to set OV2 (for outer word)
11. - Enable CPA (40-63) to RGA (40-63)

99 COMPLEMENT (if needed), NORMALIZE,
ADJUST EXPONENT OF
OUTER WORD

1. Clear and load clocks into RGA (40 - 63)
2. Enable true out of RGA (40 - 63) if OCMP latch is low
3. Enable complement out of RGA (40 - 63) if OCMP latch 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 to set 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 (72 - 79)
27. Enable CPA sum (72, 73 - 79) into RGA (8, 9 - 15)
28. Force a shift right ONE is overflow occurs
29. Inhibit section carries

T_{10} CORRECT RESULTANT
EXPONENT & SIGN
OF OUTER WORD

1. Enable RGA (0 - 7)
2. Enable true out of RGB (1 - 7) (for expon 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 there is FYEASNCO-T which conditionally clears OUTER word of RGA in exponent overflow or underflow or if there is underflow and P-ZML--HL
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 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)

APPENDIX C: MULTIPLICATION IN 64 BIT MODE

T1 FIND EXPONENT - RECODE 8 BITS

OF MULTIPLIER MANTISSA FOR FIRST ITERATION

1. Enable Registers A & B (0-63 bits)
2. Enable Register A into R (0-63 bits)
3. Enable Register R (16-63 bits)
4. Enable clear and load clocks into inner and outer word of register R

6. Enable clear and load clocks to inner and outer word VFLW latches
7. Enable exponent underflow and overflow into mode
8. Enable underflow into Register D contional on E and E_1
9. Inhibit clear clocks into Register D
10. Enable clear & load clocks into F
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 innter mantissa of Register A (16-39) if E_1=1
14. Enable clear clocks into outer mantissa of Register A (40-63)
15. Enable load clocks into outer sign and exponent of Register A (0-7) if E=1
16. Enable load clocks into innter 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) } For Unsigned Operand
24. Enable the complement output of the inner exponent of Register B (1-15) }
25. Enable Registers A & B (1-15) into CPA (65-79)
26. Enable the bit carries into CPA (64-79)
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 ofCPA into register A (8-15)
30. Restore the sign of Register A (0)

T2 FIRST ITERATION - RECODE 8 BITS

OF MULTIPLIER MANTISSA FOR SECOND ITERATION

1. Enable true out of RGA (16-23)
2. Enable true out of RGB (16-63)
3. Enable true out of RGR (16-23)
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 BSW (32-63) into RGB (32-63)
10. Enable PAT sum into RGA (16-39)
11. Enable WD4 (16-63) into CPA (16-63)
12. Enable WD4 (0-7) into CPA (64-71)
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. Enable clear clocks to RGA (16-63) if $E=E_1=1$
18. Enable load clocks to RGA (16-63) if $E=E_1=1$
19. Enable clear clocks to RGB (16-63)
20. Enable load clocks to RGB (16-63
21. Enable clear and load clocks to CPA carries
22. Select K function

T3 SECOND ITERATION - RECODE 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. Enable WD4 (0-7) into CPA (64-71)
16. Inhibit section carries
17. Enable load clocks to RGA (16-63) if $E=E_1=1$
18. Enable clear clocks to RGB (16-63)
19. Enable load clocks to RGB (16-63)
20. Enable clear and load clocks to CPA carries
21. Enable clear clocks to RGA (16-63) if $E=E_1=1$
22. Select K function

T4 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. Enagle true out of RGR (16-63)
4. Enable RGB (16-63) to LOG (16-63)
5. Enable LOG (16-63) into the BSW (16-63)
6. Fource Shift right end-off 8 positions to the BSW controls
7. Enable PAT sum and carry bits (16-71) to CPA
8. Enagle 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. Select K function
12. Enable the stored carry
13. Enable PAT sum and carry bits (16-71) into CPA
14. Enable WD 4 (0-7) into CPA (64-71)
15. Enable RGB (16-23) into CPA (72-79)
16. Enable RGC (65-72) into CPA (72-79)
17. Enable bit carries into CPA (72-79)
18. Inhibit section carries
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.

T5 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)
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
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 WD4 (0-7) 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. Select K function
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

T6 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)
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
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 WD4 (0-7) into CPA (64-71)
14. Enable RGB (16-23)
15. Enable RGC (65-72) into CPA (72-79)
16. Enable bit carries into CPA (72-79)
17. Inhibit section carries
18. Select K function
19. Enable clear clocks to BGA (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

T7 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)
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
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 WD4 (0-7) into CPA (72-79)
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. Select K function
19. Round (optional)
20. Enable clear clocks to RGA (16-63)
21. Enable load clocks to RGA (16-63)
22. Enable clear clocks to RGB (16-63)
23. Enable load clocks to RGB (16-63)
24. Enable clear and load clocks to CPA carries

T8 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)
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
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)
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 normalize and $E=E_1=1$
23. Enable load and clear clocks to LOD latches

T9 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 (L1) (64 - 71) to RGA (0 - 7)
4. Enable CPA sum (L1) (40 - 63) to RGA (40 - 63)
5. Enable CPA sum (72 - 79) to RGA (8 - 15)
6. Enable CPA sum (L1) (16 - 39) to RGA (16 - 63)
7. Clear RGA (0 - 63) on $\overline{\text{UFL}}$ or bit 16 is a ZERO ($\overline{\text{UFL}}$ + A16)
8. Load RGA (0 - 63) on $\overline{\text{UFL}}$; UF ; $\overline{\text{A16}}$; A17
9. Enable clear and load clocks to F bit mode register
10. Enable exponent underflow into mode register
11. Inhibit clear clocks to RGD
12. Enable RGA (1 - 63) into CPA (65 - 79, 16 - 63)
13. Enable RGB (1 - 7) into CPA (65 - 71)
14. Enable bit carries into CPA (64 - 79)
15. Inhibit section carries
16. Restore the sign of RGA (0)

APPENDIX D: DIVISION IN 64 BIT MODE

T_1 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 & COMPLEMENT out of RGR mantissa (16-63) *
4. Enable RGR (0-63) into OSG
5. Clear RGB exponent & sign (0-15)
6. Enable load clocks into RGB exponent & 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 we have to gate into OSG the COMPLEMENT form of RGR.

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

T_2 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, but since we enable only the mantissa of RGB, the exponent part of RGR is already in RGB from the previous clock time and need not be inserted again and therefore the mantissa of RGR shifted to the right by one is allowed to come into RGB.

T_3 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 (64-)
9. Compute sign of RGA
10. Clear exponent & 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 count 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 conditional on E, E_1
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 & sign of RGB (0-7)
27. Enable load clocks to OUTER exponent & sign of RGB (0-7)
28. Enable 00111111 into OUTER exponent & sign of RGB (0-7) for exponent correction during normalization
29. Initialize iteration counter for 47 times

$T_4 \longrightarrow T_{51}$ 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)
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 interations repeat all steps T_4 - T_{51}. If the counter has counted 47 iterations then go to T_{52}.
24. Increment iteration counter after the above testing.

* In step 3 above we have to enable WORD # 4 x 2 path through MSG 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 we allowed the load clocks of mantissa of RGB, we can say that in reality the mantissa part of RGB will pass through the Barrel Switch and will go back to RGB shifted left one.

T_{52} 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 COMPLEMENT out of sign an 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 $Q_1 = 1$ (RGB 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

T_{53} 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 & 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 916-63)
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 of 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 lead clocks to F bit
24. Enable clear and load clocks into mantissa of "B" register if bit 16 of "B" register is a ZERO ($Q_1 = 0$)

T_{54} INTERCHANGE MANTISSAS OF RGA & 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 that means that only the mantissa part of LOG is effectively used.

** From steps 10 & 13 above we conclude that: The mantissa of RGB is allowed to be transferred into the mantissa of RGA only if we normalize or the exponent underflow latch is low (contains ZERO) and do not normalize but 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 we do not normalize.

IF NORMALIZE ADJUST EXPONENT IN TWO CLOCK TIMES (T_{55}, T_{56})

T_{55} DETECT THE LEADING ONE OF MANTISSA OF RGA & 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 & 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.

T_{56} 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 if 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 modes register
16. Enable clear and load clocks into F bit

APPENDIX E: 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 keep in mind.

Since the recursive process was fully explained in 64 bit mode, and because almost 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, we will provide a summary of the actions being taken in each clock time and urge the reader to refer to the POSFILE for more detailed information.

CLOCK TIME	DESCRIPTION OF ACTIONS BEING TAKEN	REMARKS
T_1	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
T_2	The shifted to the right by 1 end off OUTER mantissa of RGR is transfered through the Barrel Switch into RGB	Only if rounding
T_3	The 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 the 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	
T_4	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	This is CU decision
T_5	INNER mantissa of RGB is placed into the OUTER mantissa of RGB OUTER mantissa of RGB is placed into the INNER mantissa of RGB	See table for Interchanging INNER & OUTER mantissas of RGB
T_6	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 (CUB) through OSG and Address Adder (This is like shifting left by 1 end-around)	This is CU decision
T_7-T_{30}	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 $Q_i = 0$)	
T_{31}	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

T_{32}

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 definitely 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

T_{33}

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

T_{34}	Transfer the INNER mantissa of "R" into the OUTER mantissa of "R" register	
T_{35}	Enable shift righ 16 end-around into shift count register from CDB through OSG and ADA	
T_{36}	Transfer the OUTER word of "R" register into the INNER word of "R" register	See Table 2
T_{37}	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
T_{38}-T_{61}	If the result of subtraction of INNER mantissa of "R" register from the INNER mantissa of "A" register ≥ 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 ≥ 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

		QUOTIENT					
$077)_8$	$077)_8$	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9

T_{62} Check bit 40 of "B" register. If it is a ONE that means 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

T_{63} 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 ≥ 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 execution 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 it has been previously explained ($X_o \geq Y$ case). — If ignore exponent

T_{64}

Transfer mantissa of "A" register into mantissa of "B" register through the Barrel Switch.
Transfer mantissa of "B" register into mantissa of "A" register through the CPA.
Enable shift left by 8 end around from CDB into shift count register through OSG and ADA.
Enable clear and load clocks to F bit. At this time the contents of "A" & "B" registers are as follows:

This is CU decision

"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

T_{65}

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 3. This is CU decision

T_{66}

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 3.

T_{67}	Enable TRUE of INNER mantissa of "A" register into Barrel Switch through LOG. Enable LOD to detect the leading ONE. Enable exponent adjustment into INNER exponent of "B" register. Enable Barrel Switch back to "A" register. At this time the contents of "A" & "B" registers are as follows:	Only if normalize -11- -11- -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

T_{68}	Enable OUTER mantissa of "A" register into Barrel Switch through LOG. Enable LOD to detect the leading ONE. Enable exponent adjustment into the OUTER exponent of "B" register. Enable Barrel Switch back to "A" register. Enable TRUE out of INNER exponent of "A" register and bring it into CPA. Enable adusted exponent out of INNER exponent of "B" register and bring it into CPA. 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. 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	Only if normalize -11- -11- -11- Only if normalize -11- -11- -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

T_{69} 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 1: 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"$_9$	T_3
LOG	$077)_8$	IN. EXP. of "R" REGISTER	"B"$_4$	"B"$_5$	"B"$_6$	"B"$_7$	"B"$_8$	"B"$_9$	
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"$_6$	T_4
"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"$_7$				
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"$_8$	T_5
"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"$_6$	

NOTE: 1) The shaded area indicates bytes which have not been enabled out of RGB and therefore at the end of clock time T_5 they are found unchanged in their location into RGB.

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

TABLE 2: Procedure for Interchanging

INNER & OUTER Mantissas of RGB

	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"$_9$ *	
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"$_5$	T_{34}
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								T_{35}
OSG	"R" IN. EXP.	"R"$_8$	"R"$_9$	"R" OUT EXP.	"R"$_7$				
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"$_8$	T_{36}
		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 content is enabled because the OSG gates are of negative logic and therfore in order to get a TRUE output from the OSG gates they must receive an input in COMPLEMENT form.

TABLE 3: 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						T_{65}
			R_7	R_8	R_9	R_{14}	R_5	R_6	
LOG	$077)_8$	$077)_8$	R_7	R_8	R_9	R_{14}	R_5	R_6	
BARREL SWITCH	R_9	$077)_8$	R_5	R_6	$077)_8$	R_{14}	R_7	R_8	
SHIFT LEFT by 8 EA	$077)_8$	R_5	R_6	$077)_8$	R_{14}	R_7	R_8	R_9	
"B" REGISTER	$077)_8$	R_5	R_6	$077)_8$	R_{14}	R_7	R_8	R_9	
LOG		R_5	R_6	$077)_8$	R_{14}				T_{66}
BARREL SWITCH	R_{14}	R_5	R_6	$077)_8$	R_{14}	R_5	R_6	$077)_8$	
SHIFT RIGHT by 16 EA	R_6	$077)_8$	R_{14}	R_5	R_6	$077)_8$	R_{14}	R_5	
		CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD	CLEAR & LOAD				
B	$077)_8$	$077)_8$	REMAINDER						
			R_{14}	R_5	R_6	R_7	R_8	R_9	