Burroughs

BSP

FLOATING POINT ARITHMETIC

BSP

BURROUGHS SCIENTIFIC PROCESSOR

FLOATING POINT ARITHMETIC

CONTENTS

1. INTRODUCTION

One of the most important features of any computer is its arithmetic. This document discusses the implementation of floating point arithmetic in the Burroughs Scientific Processor (BSP). Data representation in both the BSP memory and arithmetic element is described, as are the arithmetic algorithms used in the BSP. Of particular interest are the techniques used for error checking in the arithmetic element and for rounding in both the scalar processor and the parallel processor. The BSP arithmetic operations, including instructions and cycle operations, are described in detail in Appendix A, and the accuracy of arithmetic operations is discussed in Appendix B.

2. DATA REPRESENTATION IN MEMORY

The representation of data in the memory of the Burroughs Scientific Processor (BSP) is as follows:

1. Single precision floating point word format,
2. Integer word format,
3. Double precision real floating point word format.

SINGLE PRECISION FLOATING POINT WORD FORMAT

A single precision floating point number, X, is represented by an ordered pair of numbers, E (exponent) and m (mantissa), such that:

$$X = 2^{E} * m$$

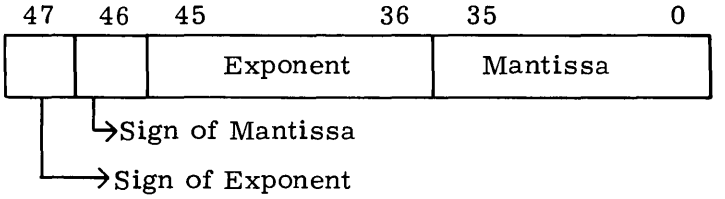
where E is an integer and m satisfies the condition:

$$-1/2 \leq m < -1 \quad \text{or} \quad 1/2 \leq m < 1 \text{ or } m = 0.$$

In order to meet this condition, that is, $1/2 \leq |m| < 1$, the last step in every floating point operation is the normalization of the mantissa (removal of leading zeroes). The layout of the floating point word in memory is indicated below. The bits are numbered from right to left. The least significant bit is numbered 0; the most significant bit of the mantissa is bit 35; the least significant of the exponent is bit 36. The most significant bit of the exponent is bit 45. Bit 46 is the sign bit of the mantissa; bit 47 is the sign bit of the exponent. Every group of consecutive bits is

called a field, and is denoted by W[x:y]; W is the name of the data unit x is the address of the left-most bit, and y is the length of the field. Thus, a data word of the data unit, A, is defined as A[47:48], and its null indicator is defined as A[48:1].

Using field notation X[leading bit:numbers of bits], the mantissa is represented by X[35:36], the exponent by X[45:10], the sign bit of the mantissa by X[46:1].



The range of representable numbers in single precision is as follows:

1. For positive X:

$$2^{-1023} * 1/2 \leq X \leq 2^{1023} * (1-2^{-36}), \text{ where}$$

$$2^{-1023} * 1/2 \cong 10^{-308.25}$$

2. For negative X:

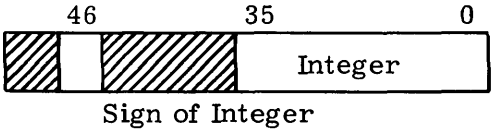
$$-2^{-1023} * 1/2 \geq X \geq -2^{1023} * (1-2^{-36}), \text{ where}$$

$$2^{-1023} * (1-2^{-36}) \cong 10^{307.95}.$$

INTEGER WORD FORMAT

An integer, I, is defined by its absolute value m (I) and by its sign bit S (I). Field I[35:36] contains the magnitude, and the sign bit is in field[46:1]. The unused bits of the data word are set to 0. The range of integer values is symmetric about zero.

$$-2^{36} + 1 \leq I \leq 2^{36} - 1$$



DOUBLE PRECISION REAL FLOATING POINT WORD FORMAT

A double precision floating point number X is represented by two single precision numbers FIRST (X) and SECOND (X). Both of these numbers are normalized. The mantissa sign bits must be the same. Due to normalization, the relationship between exponents is:

EXPONENT (SECOND (X)) $\leq$ EXPONENT (FIRST (X)) -36

SECOND (X) = 0 is a valid word.

Thus, a double precision floating point word is the sum D(X) = FIRST (X) + SECOND (X). The BSP double precision format is different from the B 7800 double precision format. In the B 7800, the exponents and the mantissas of the two single precision words are concatenated, and the two words form one entity.

The range of a double precision number X is given as follows:

1. For positive X:

$$2^{-1023} * 1/2 \leq X \leq 2^{1023} * (1-2^{-36}) + 2^{987} * (1-2^{-36})$$

2. For negative X:

$$-\left[(2^{1023} * (1-2^{-36}) + 2^{987} * (1-2^{-36})\right] \leq X \leq -2^{-1023} * (1/2)$$

The double precision format of the BSP, specifically the fact that the exponent of the second word is less than or equal to the exponent of the first word minus 36, can lead to an underflow condition in the second word, while there is no underflow in the first word. For example:

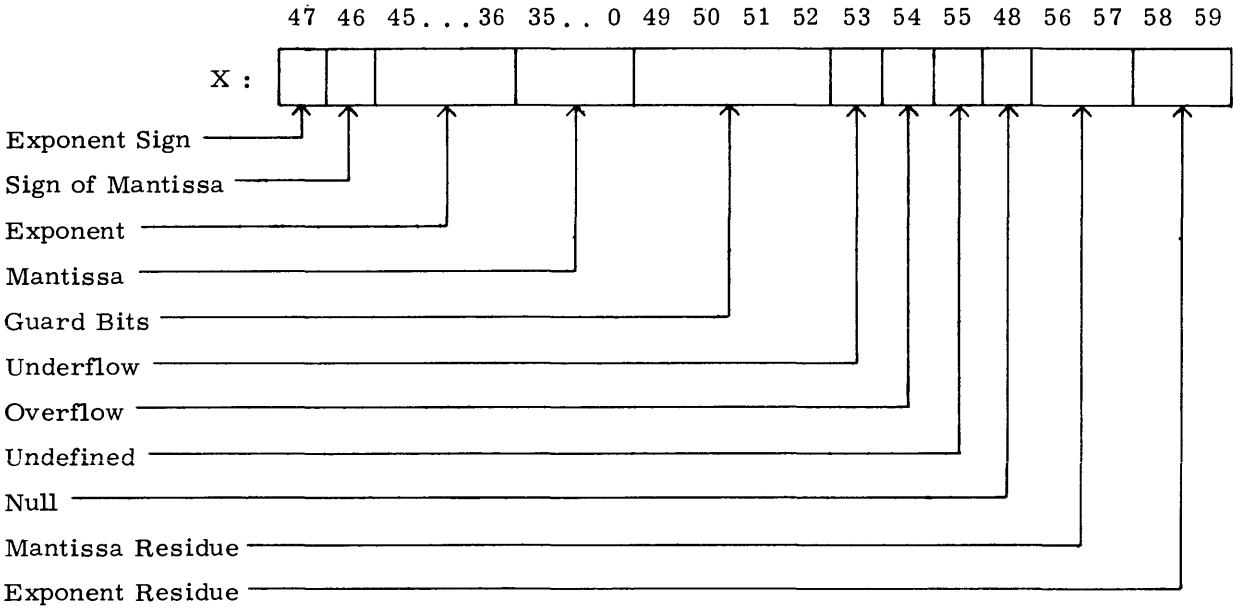
$$\left[2^{-951} * (1 - 2^{-36}) + 2^{-1023} * (1 - 2^{-36})\right] * 1/2$$

The multiplication of the number in brackets by 1/2 will lead to underflow in the second word. However, the underflow condition can be disabled.

3. DATA REPRESENTATION IN THE ARITHMETIC ELEMENTS

BASIC DATA REPRESENTATION

The basic BSP data word consists of 48 binary digits (bits). To this word are appended 12 other bits to form a data unit: a one-bit null indicator, three error condition bits (underflow, overflow, and undefined), four guard bits, two modulo 3 residue code bits for the mantissa, and two modulo 3 residue code bits for the exponent. For input and output purposes, only 49 bits are used: the 48-bit data word, and the null indicator. The positions or addresses of binary digits within the data unit are designated by the decimal numbers 0 to 59. Within the data word, the bits are numbered 0 to 47. The data format in the arithmetic unit is given in the following diagram:



REPRESENTATION OF ZERO

Integer, single precision, and double precision zero are represented with all bits set to zero. So called dirty zeroes are eliminated by hardware action. Operations which may result in dirty zeroes, for example: (0) * (-5) or N-N are tested. When a zero mantissa is found, sign bits and exponent are also set to zero.

COMPLEX NUMBER SINGLE PRECISION)

Complex numbers are implemented implicitly. Two contiguous, single precision floating point numbers represent a complex number Z, where the first number is the real part of Z and the second number is the imaginary part of Z. No double precision complex numbers are provided. Single precision complex operations are implemented by software using real arithmetic hardware operators.

4. HARDWARE ERROR CHECKING

Error checking in the arithmetic element is done in three ways:

1. Errors which result in overflow, underflow, or undefined operations within arithmetic elements.

2. Errors within the arithmetic elements are checked by a modulo 3 residue code. Exponent and mantissa are checked separately (two bits each). A modulo 3 residue code is limited; it cannot detect errors which are multiples of three. A modulo 3 error results in an interrupt. Errors are reported for logging and to the parallel memory control to disable parallel memory write.

3. Errors in data transmission. A Hamming Code generator computes seven parity bits of a Hamming Code over the 48 bits of data. This code is a single error correction/double error detection code (SEC/DED), which protects the parallel memory and the data transmission of the alignment networks. Input data from the arithmetic elements are encoded by the alignment networks; input data from the control processor and from file memory are already encoded.

Detected errors are logged in the maintenance log. The Hamming Code will report bit error, arithmetic element numbers and error type. There are three classes of errors:

1. Tolerable error (e. g., Hamming Code single bit error),

2. Fatal error (e. g., memory address error during write cycle)

3. Retryable error (e. g., double bit Hamming Code error, that is, conditional, overflow, underflow, undefined).

These three errors are reported as interrupts to the Master Control Program (MCP). It is up to the MCP to determine the appropriate action.

Fatal and retryable errors will inhibit a write cycle. Depending on error type, a retry or a system shutdown as initiated.

5. ARITHMETIC ALGORITHMS

IMPLEMENTATION OF RECIPROCATION AND SQUARE ROOT

The BSP implements reciprocation and the square root by iterative procedures. Both algorithms are applications of the Newton-Raphson method that utilize multiplications and additions only. Both algorithms are partially hardware-implemented.

The Newton-Raphson procedure has quadratic convergence, and the accuracy which can be achieved depends on the choice of the starting value and on the number of iterations.

In order to obtain the machine accuracy of 36 bits in reciprocation and in the square root, it was required that the necessary accuracy is to be obtained in three iterations.

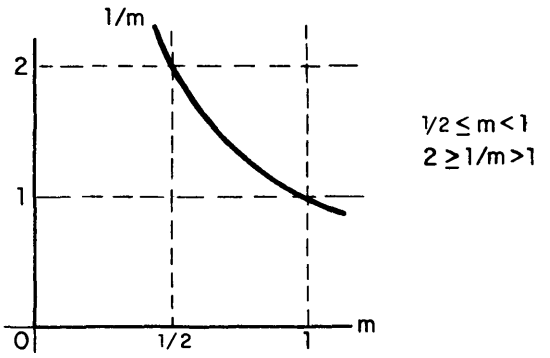
RECIPROCATION

To obtain the reciprocal of a floating point number A, Newton's method is used to solve the equation:

$$F(X) = 1/X - A = 0$$

The first step is to find the initial approximation X_0. This is done via a table lookup from a ROM. Recall that the internal representation of A is as (E . m), where E is the exponent (base 2) and m is the normalized mantissa $1/2 \leq m < 1$.

Thus, $1/A = 2^{-E} * 1/m$ and reciprocation is reduced to the computation of $1/m$ in the interval $(1/2, 1)$. The function $1/m$ in the interval $(1/2, 1)$ is shown below:



The range of the function of $1/m$ in the interval $(1/2, 1)$ is $(2, 1)$. Since it is required that division and the square root operations be accurate to 36 bits within three iterations, the starting value for the Newton-Raphson iteration has to be at least five bits, assuming quadratic convergence. However, since the slope of the curve is not constant, one has also to consider the maximum slope of the curve. The maximum slope of $1/m$ is $-1/m^2$. At $m = 1/2$, the slope is -4. Therefore, an additional two bits are required to achieve the required accuracy in the neighborhood of $m = 1/2$. For practical reasons, a ROM of eight bits is used, which corresponds to a table of 256 entries.

When X_0 has been determined from ROM, X_1 and X_2 are formed using the recursion:

$$X_{n+1} = X_n * (2 - A * X_n).$$

To get the iteration started, seven bits are taken from ROM. The first iteration is accurate to 13(14) bits , the second iteration yields 25(26) bits. These 25 bits are truncated to 19 bits; the last iteration results in 38 bits. The last two bits are used as guard bits. In the present application, the algorithm approaches the true value always from below, and hence, the rounding is not unbiased. The rounding error is less or equal to 2^{-36}. If $A = 0$, the result is undefined; if $A < 2^{-1023}$, the overflow flag is set.

Double precision reciprocation requires an additional Newton-Raphson iteration. All operations are executed in double precision. The double precision reciprocation is accurate to 70 bits.

DIVISION

If A and B are single precision floating point numbers, A/B is found in two steps:

1. 1/B is found by reciprocation
2. R = A * (1/B).

The maximum error in floating point division is:

$$(1/A)\ (1 + E_1) * B\ (1 * E_1) \approx A/B\ (1 + E_1 + E_1)$$

where:

$$\left|E_1\right| \leq 2^{-36} \text{ and}$$

$$\left|E_1 + E_1\right| < 2 * 2^{-36}.$$

If A and B are double precision floating point numbers, A/B is found in two steps:

1. 1/B is found by reciprocation
2. R = A * (1/B).

Double precision division is accurate to 70 bits.

SQUARE ROOT

If A is a single precision floating point number, the reciprocal square root $1/\sqrt{A}$ is found by solving the equation:

$$F(X) = 1/X^2 - A = 0.$$

The representation of A is $2^E * m$. Therefore, $1/\sqrt{A} = 1/\sqrt{2^E * m} = 1/\sqrt{2^E} * 1/\sqrt{m}$. If the exponent, E, is odd, 2^E is multiplied by two to make it even. The mantissa is multiplied by 1/2. The range of the mantissa is thus changed to $1/4 \leq m < 1$.

As in division, the first iteration for X_0, $1/\sqrt{m}$, is read from a ROM. Subsequent iterations are obtained by solving:

$$X_{n+1} = X_n * (3-X^2 * A)/2.$$

Seven bits suffice to start the iteration. The first iteration yields 13(14) bits, the second iteration generates 25(26) bits which are truncated to 19. In the last step, 38 bits are generated: 36 bits plus 2 bits for rounding. The rounding is biased, because the true value of $1/\sqrt{X}$ is approached from below. The error in the reciprocal square root is estimated as $E_1 = 2 * 2^{-36}$. The square root itself requires an additional multiplication. The total error in the square root is, therefore, $E \leq 2 * 2^{-36}$.

The double precision reciprocal square root requires an additional iteration. In the last iteration, all operations are executed in double precision.

The square root of a single precision number is found in two steps:

1. $Y = 1/\sqrt{A}$

2. $R = A * Y$.

To compute the double precision square root, steps 1 and 2 above are done with a double precision reciprocal square root computed first, followed by a double precision multiplication.

6. ROUNDING AND NORMALIZATION

In the control processor, rounding is done after each binary floating point operation as well as after reciprocation and after reciprocal square root.

In the parallel processor, rounding is done at the conclusion of each binary floating point operation and after reciprocation and reciprocal square roots. This type of rounding applies also to the execution of templates such as triads, tetrads, etc. The rounding operations correspond to the rounding operations in a sequential machine although in some instances the sequence of operations may be changed.

ROUNDING - SINGLE PRECISION

The rounding rules in arithmetic operations -- addition, subtraction, multiplication, reciprocation and reciprocal square root are:

1. If the discarded portion of the binary fraction is less than half of the least significant bit, leave the least significant bit unchanged.

2. If the discarded portion is greater than half of the least significant bit, add one to the least significant bit with full carry propagation.

3. If the discarded portion is exactly half of the least significant bit, set the least significant bit.

The number of bits used for rounding in different operations differs, depending upon the operation. Addition and subtraction use four rounding bits; multiplication uses 18 bits; reciprocation and reciprocal square root use two bits.

The following examples demonstrate the rounding rules for addition, using a mantissa of four bits and two guard bits. Rounding for the other rounding operations follows the same rules, except that the number of guard bits varies as stated previously.

Example 1a: 1.0 + 15/16 = 31/16

$$\left.\begin{array}{l} 2^1 * (.1000) \\ +\,2^0 * (.1111) \end{array}\right\} = \begin{array}{l} 2^1 * (.1000) \\ +\,2^1 * (.0111)\ 1 \\ \hline 2^1 * (.1111)\ 1 \\ \text{after rounding} \quad 2^1 * (.1111) \end{array}$$

Example 1b: 9/8 + 11/16 = 29/16

$$\left.\begin{array}{l} 2^1 * (.1001) \\ +\,2^0 * (.1011) \end{array}\right\} = \begin{array}{l} 2^1 * (.1001) \\ +\,2^1 * (.0101)\ 1 \\ \hline 2^1 * (.1110)\ 1 \\ \text{after rounding} \quad 2^1 * (.1111) \end{array}$$

Example 2: 3/16 + 15/16 = 45/16

$$\left.\begin{array}{l} 2^1 * (.1111) \\ +\,2^0 * (.1111) \end{array}\right\} = \begin{array}{l} 2^1 * (.1111) \\ +\,2^1 * (.0111)\ 1 \\ \hline 2^1 * (1.0110)\ 1 \\ 2^2 * (.1011)\ 01 \\ \text{after rounding} \quad 2^2 * (.1011) \end{array}$$

Example 3: 40/16 + 15/16 = 55/16

$$\begin{array}{l} 2^2 * (.1010) \\ +\,2^0 * (.1111) \end{array} = \begin{array}{l} 2^2 * (.1010) \\ +\,2^2 * (.0011)\ 11 \\ \hline 2^2 * (.1101)\ 11 \\ \text{after rounding} \quad 2^2 * (.1110) \end{array}$$

ROUNDING - DOUBLE PRECISION

Double precision operations are not rounded: there are no guard bits.

NORMALIZATION

All real numbers are stored in normalized form, that is, the leading bit of the mantissa is always a one.

$$1/2 \leq \text{mantissa} < 1$$

In double precision, both words are normalized.

APPENDIX A

ARITHMETIC OPERATIONS

Instruction	Mnemonic	Time in Clocks
Single Precision, Floating Point Family		
Add	ADD	2
Subtract	SUB	2
Multiply	MUL	2
Reciprocate	RECIP	6
Divide	DIV	8
Reciprocal Square Root	SQRTR	10
Square Root	SQRT	12
Square	SQR	2
Extract Exponent	EXTX	2
Insert Exponent	INSX	2
Binary Scale to Left	BSCL	2 *
Binary Scale to Right	BSCR	2 *
Normalize	NORM	2 *
Truncated Add	TADD	2
Truncated Subtract	TSUB	2
Truncated Multiply	TMUL	2
Maximum	MAX	2 *
Minimum	MIN	2 *
Absolute Maximum	AMAX	2
Absolute Minimum	AMIN	2

*These operations can be preformed in one clock if they are incorporated in the appropriate template.

Instruction	Mnemonic	Time in Clocks
Extended Precision Family		
(Single Precision Operand, Double Precision Result)		
Extended Add	EADD	5
Extended Subtract	ESUB	5
Extended Multiply	EMUL	4
Double Precision Family		
Double Precision Add	DPADD	8
Double Precision Subtract	DPSUB	8
Double Precision Multiply	DPMUL	11
Double Precision Reciprocate	DPREC	16
Double Precision Divide	DPDIV	27
Double Precision Reciprocal Square Root	DPSQRR	21
Double Precision Square Root	DPSQR	32
Double Precision Round to Single Precision	SNGL	2
Integer Family		
Float Integer	FLOAT	2*
Integer Add	IADD	2*
Integer Subtract	ISUB	2*
Integer Multiply	IMUL	3
Integer Divide	IDIV	15
Type Transfer Operations		
Integer with Truncation	FIXT	2*
Integer with Rounding	FIXR	2*
Integer with Floor	FIXF	2
Integer with Ceiling	FIXC	2
Integer to Floating Point	FLOAT	2*
Normalize	NORM	2*
Double Precision to Single Precision	SNGL	2

*These operations can be performed in one clock if they are incorporated in the appropriate template.

Cycle Operations

ADD/SUBTRACT — Single Precision

1. $A \pm B \rightarrow R_G$
2. NORM $(R_G) \rightarrow Z$ (Round)

MULTIPLY — Single Precision

1. $A * LSH(B) \rightarrow PP$ LSH(B) = Least Significant Half (B)
2. $A * MSH(B) + PP \rightarrow Z$ MSH(B) = Most Significant Half (A)

RECIP (A) — Single Precision

1. $2 - (A * PROM(A)) \rightarrow P_1$ Table look-up (7 bits)
2. $P_1 * PROM(A) \rightarrow R_1$
3. $2 - (A * R_1) \rightarrow P_2$
4. $P_2 * R_1 \rightarrow R_2$
5. $2 - (A * R_2) \rightarrow P_3$
6. $P_3 * R_2 \rightarrow Z$ (Round)

DIV — Single Precision

1-6. RECIP (B) $\rightarrow$ C (Round)

7-8. A MULT B $\rightarrow$ Z (Round)

SQRTR (A) — Single Precision

1. A shift if exp. odd $\rightarrow A^1$
2. $A^1 * PROM(A^1) \rightarrow P_1$
3. $3 - P_1 * PROM(A^1) \rightarrow P_2$
4. $P_2 * PROM(A^1) \rightarrow R_1$

5. $A^1 * R_1 \longrightarrow P_3$

6. $3\text{-}P_3 * R_1 \longrightarrow P_4$

7. $P_4 * R_1 \longrightarrow R_2$

8. $A^1 * R_2 \longrightarrow P_5$

9. $3\text{-}P_5 * R_2 \longrightarrow P_6$

10. $P_6 * R_2 \longrightarrow Z$

DPSQRR — Double Precision Reciprocal Square Root

1-10. $SQRTR(A_1) \rightarrow B_1$ & A_1 shift if exp odd $\rightarrow A_1^1$

11. $A_1 + A_2 \longrightarrow A_2^1$ (A_2^1 has A_1 exponent)

12. $A_2^1 \rightarrow A_2^1$ (pass to R_0)

13. A_2^1 shift if exp odd $\rightarrow A_2^{11}$

14-16. $B_1 * (A_1^1 + A_2^{11})$

17. $\rightarrow C_1 + C_2$

18-20. $3\text{-}B_1 * (C_1 + C_2)$

21. $\rightarrow D_1 + D_2$

22-24. $B_1 * (D_1 + D_2)$

25. $\longrightarrow Z_1 + E_2$

26. $Z_1 \longrightarrow$ output

27. NORM $(E_2) \longrightarrow Z_2$

DPMUL Double Precision MULT

1. $A_1 * B_1$
2. $\longrightarrow D_1 + D_2$
3. $A_1 * B_2$
4. $\longrightarrow E_{1G}$
5. $A_2 * B_1$
6. $\longrightarrow F_{1G}$
7. $F_{1G} + E_{1G} \longrightarrow G_{1G}$
8. $G_{1G} + D_2 \longrightarrow H_{1G}$
9. $H_{1G} + D_1 \longrightarrow Z_1 + F_2$
10. $H_{1G} + D_1 \longrightarrow Z_1 + C_2$
11. $NORM(C_2) \longrightarrow Z_2$

DPREC Double Precision Reciprocation

1.-6. RECIP(A_1) $\rightarrow B_1$ (No Rounding)

7. $A_2 * B_1$

8. $\rightarrow C_{1G}$

9. $2-A_1 * B_1$

10. $\rightarrow D_1 + D_2$

11. $-C_{1G} + D_2 \rightarrow E_{1G}$

12. $E_{1G} * B_1$

13. $\rightarrow F_{1G}$

14. $B_1 + F_{1G} \rightarrow Z_1 + B_2$

15. $B_1 + F_{1G} \rightarrow Z_1 + B_2$

16. NORM(B_2) $\rightarrow Z_2$

SQRT (A) Single Precision

1-10. SQRTR(A) $\rightarrow$ C

11-12. C MULT A $\rightarrow$ Z

EADD/ESUB Extended Add/Sub

1. $A \pm B \rightarrow P_1 + R_1$
2. $NORM(P_1) \rightarrow P_3$
3. $P_3 + R_1 \rightarrow Z_1 + P_4$
4. $P_3 + R_1 \rightarrow Z_1 + P_4$
5. $NORM(P_4) \rightarrow Z_2$

EMUL Extended Multiply

1. $A * LSH(B) \rightarrow PP$
2. $A * MSH(B) + PP \rightarrow Z_1 + P_2$
3. $Z_1 \rightarrow Z_1$
4. $NORM(P_2) \rightarrow Z_2$

DPADD/DPSUB Double Precision ADD/SUB

2. $A_1 \pm B_1 \rightarrow D_1 + D_2$
2. $\pm B_2 + D_2 \rightarrow E_{1G}$
3. $E_{1G} + A_2 \rightarrow F_{1G}$
4. $F_{1G} + D_1 \rightarrow G_1 + G_2$
5. $NORM(G_1) \rightarrow P_1$
6. $P_1 + G_2 \rightarrow C_1 + C_2$
7. $NORM(C_1) \rightarrow Z_1$
8. $NORM(C_2) \rightarrow Z_2$

APPENDIX B

ERROR ESTIMATES FOR ARITHMETIC OPERATIONS

		Floating Point	
	Operation	Single Precision	Double Precision
Addition/Subtraction	$A \pm B$	$(A \pm B)(1 + E_1)$	$(A \pm B)(1 + 4E_2)$
Multiplication	$A * B$	$(A * B)(1 + E_1)$	$(A * B)(1 + 4E_2)$
Reciprocation	$1/A$	$1/A\ (1 + E_1)$	$1/A\ (1 + 4E_2)$
Division	B/A	$B/A\ (1 + E_1)(1 + E_1)$	$1/A\ (1 + 4E_2)(1 + 4E_2)$
Reciprocal Square Root	$1/\sqrt{A}$	$1/\sqrt{A}\ (1 + E_1)$	$1/\sqrt{A}\ (1 + 4E_2)$
Square Root	$A/\sqrt{A}$	$A/\sqrt{A}\ (1 + E_1)(1 + E_1)$	$A/\sqrt{A}\ (1 + 4E_2)(1 + 4E_2)$

where:

$\left|E_1\right| \leq 2^{-36}$ for single precision instructions,

$\left|E_2\right| \leq 2^{-72}$ for double precision instructions.