(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 December 2002 (05.12.2002)

PCT

(10) International Publication Number
WO 02/097606 A1

(51) International Patent Classification[7]: G06F 7/52

(21) International Application Number: PCT/US02/16016

(22) International Filing Date: 22 May 2002 (22.05.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/293,173 25 May 2001 (25.05.2001) US
10/035,580 28 December 2001 (28.12.2001) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 4150 Network Circle, Santa Clara, CA 95054 (US).

(72) Inventor: STEELE, JR., Guy, L.; 9 Lantern Lane, Lexington, MA 02421 (US).

(74) Agents: GARRETT, ARTHUR, S. et al.; FINNEGAN, HENDERSON, FARABOW, GARRETT & DUNNER, L.L.P., 1300 I Street, N.W., Washingon, DC 20005-3315 (US).

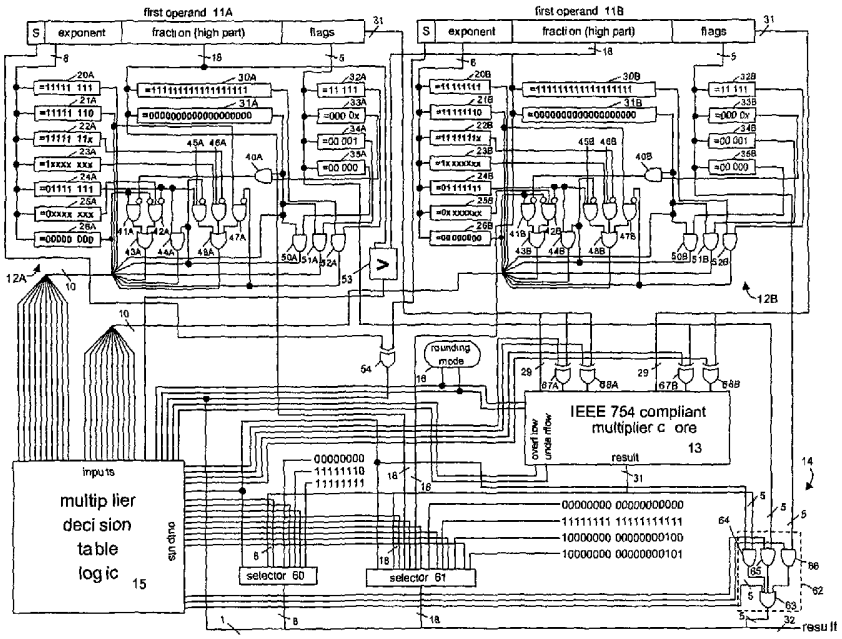
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: FLOATING POINT MULTIPLIER WITH EMBEDDED STATUS INFORMATION



(57) Abstract: A system for providing a floating point product comprises an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and data within the second floating point operand respectively. In addition, the system comprises a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the product of the first floating point operand and the second floating point operand. Additionally, the results circuit provides a resulting status embedded within the resulting floating point operand.

WO 02/097606 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

FLOATING POINT MULTIPLIER WITH EMBEDDED STATUS INFORMATION

DESCRIPTION OF THE INVENTION

Field of the Invention

[001] The invention relates generally to systems and methods for performing floating point operations, and more particularly to systems and methods for performing floating point multiplication with embedded status information associated with a floating point operand.

Background of the Invention

[002] Digital electronic devices, such as digital computers, calculators and other devices, perform arithmetic calculations on values in integer, or "fixed point," format, in fractional, or "floating point" format, or both. Institute of Electrical and Electronic Engineers (IEEE) Standard 754, (hereinafter "IEEE Std. 754" or "the Standard") published in 1985 and adopted by the American National Standards Institute (ANSI), defines several standard formats for expressing values in floating point format and a number of aspects regarding behavior of computation in connection therewith. In accordance with IEEE Std. 754, a representation in floating point format comprises a plurality of binary digits, or "bits," having the structure

[003] $$se_{msb}\cdots e_{lsb}f_{msb}\cdots f_{lsb}$$

where bit "s" is a sign bit indicating whether the entire value is positive or negative, bits "$e_{msb}\cdots e_{lsb}$" comprise an exponent field that represents the exponent "e" in unsigned binary biased format, and bits "$f_{msb}\cdots f_{lsb}$" comprise a fraction field that represents the fractional portion "f" in unsigned binary format ("msb" represents "most significant bit" and "lsb" represents "least significant bit"). The Standard defines two general formats. A "single" format comprises thirty-two bits while a "double" format comprises sixty-four bits. In the single format, there is one sign bit "s," eight bits "$e_7...e_0$" comprising the exponent field and twenty-three bits "$f_{22}...f_0$" comprising the fraction field. In the double format, there is one sign bit "s," eleven bits "$e_{10}...e_0$" comprising the exponent field and fifty-two bits "$f_{51}...f_0$" comprising the fraction field.

[004] As indicated above, the exponent field of the floating point representation "$e_{msb}\cdots e_{lsb}$" represents the exponent "E" in biased format. The biased format provides a mechanism by which the sign of the exponent is implicitly indicated. In particular, the bits "$e_{msb}\cdots e_{lsb}$" represent a binary encoded value "e" such that "e=E + bias." This allows the exponent E to extend from -126 to +127, in the eight-bit "single" format, and from -1022 to +1023 in the eleven-bit "double" format, and provides for relatively easy manipulation of the exponents in multiplication and division operations, in which the exponents are added and subtracted, respectively.

[005] IEEE Std. 754 provides for several different formats with both the single and double formats which are generally based on the bit patterns of the bits "$e_{msb}\cdots e_{lsb}$" comprising the exponent field and the bits "$f_{msb}\cdots f_{lsb}$" comprising the fraction field. If a number is represented such that all of the bits "$e_{msb}\cdots e_{lsb}$" of the exponent field are binary one's (*i.e.*, if the bits represent a binary-encoded value of "255" in the single format or "2047" in the double format) and all of the bits "$f_{msb}\cdots f_{lsb}$" of the fraction field are binary zeros, then the value of the number is positive or negative infinity, depending on the value of the sign bit "s." In particular, the value "v" is $v = (-1)^s \infty$, where "∞" represents the value "infinity." On the other hand, if all of the bits "$e_{msb}\cdots e_{lsb}$" of the exponent field are binary one's and if the bits "$f_{msb}\cdots f_{lsb}$" of the fraction field are not all zero's, then the value that is represented is deemed "not a number," which is abbreviated in the Standard by "NaN."

[006] If a number has an exponent field in which the bits "$e_{msb}\cdots e_{lsb}$" are neither all binary ones nor all binary zeros (*i.e.,* if the bits represent a binary-encoded value between 1 and 254 in the single format or between 1 and 2046 in the double format), the number is said to be a "normalized" format. For a number in the normalized format, the value represented by the number is $v=(-1)^s\ 2^{e\text{-}bias}(1.\ |f_{msb}\cdots f\cdots_{lsb})$, where "|" represents a concatenation operation. Effectively, in the normalized format, there is an implicit most significant digit having the value "one," so that the twenty-three digits in the fraction field of the single format, or the fifty-two digits in the fraction field of the double format, will effectively

represent a value having twenty-four digits or fifty-three digits of precision, respectively, where the value is less than two, but not less than one.

[007] On the other hand, if a number has an exponent field in which the bits "$e_{msb} \cdots e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are not all zero, the number is said to be a "de-normalized" format. For a number in the de-normalized format, the value represented by the number is $v = (-1)^s 2^{e-bias+1}(0.|f_{msb} \cdots f_{lsb})$. It will be appreciated that the range of values of numbers that can be expressed in the de-normalized format is disjoint from the range of values of numbers that can be expressed in the normalized format, for both the single and double formats. Finally, if a number has an exponent field in which the bits "$e_{msb} \cdots e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are all zero, the number has the value "zero". It will be appreciated that the value "zero" may be positive zero or negative zero, depending on the value of the sign bit.

[008] Generally, circuits or devices that perform floating point computations or operations (generally referred to as floating point units) conforming to IEEE Std. 754 are designed to generate a result in three steps:

[009] (a) In the first step, an approximation calculation step, an approximation to the absolutely accurate mathematical result (assuming that the input operands represent the specific mathematical values as described by IEEE Std. 754) is calculated that is sufficiently precise as to allow this accurate mathematical result to be summarized. The summarized result is usually represented by a sign bit, an exponent (typically represented using more bits than are used for an exponent in the standard floating-point format), and some number "N" of bits of the presumed result fraction, plus a guard bit and a sticky bit. The value of the exponent will be such that the value of the fraction generated in step (a) consists of a 1 before the binary point and a fraction after the binary point. The bits are commonly calculated so as to obtain the same result as the following conceptual procedure (which is impossible under some circumstances to carry out in practice): calculate the mathematical result to an infinite number of bits of precision in binary

scientific notation, and in such a way that there is no bit position in the significand such that all bits of lesser significance are 1-bits (this restriction avoids the ambiguity between, for example, 1.100000... and 1.011111... as representations of the value "one-and-one-half"); let the N most significant bits of the infinite significand be used as the intermediate result significand; let the next bit of the infinite significand be the guard bit; and let the sticky bit be 0 if and only if ALL remaining bits of the infinite significant are 0-bits (in other words, the sticky bit is the logical OR of all remaining bits of the infinite fraction after the guard bit).

[010] (b) In the second step, a rounding step, the guard bit, the sticky bit, perhaps the sign bit, and perhaps some of the bits of the presumed significand generated in step (a) are used to decide whether to alter the result of step (a). For conventional rounding modes defined by IEEE Std. 754, this is a decision as to whether to increase the magnitude of the number represented by the presumed exponent and fraction generated in step (a). Increasing the magnitude of the number is done by adding 1 to the significand in its least significant bit position, as if the significand were a binary integer. It will be appreciated that, if the significand is all 1-bits, then the magnitude of the number is "increased" by changing it to a high-order 1-bit followed by all 0-bits and adding 1 to the exponent.

[011] Regarding the rounding modes, it will be further appreciated that,

[012] (i) if the result is a positive number, and

[013] (a) if the decision is made to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up (*i.e.,* towards positive infinity), but

[014] (b) if the decision is made not to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down (*i.e.,* towards negative infinity); and

[015] (ii) if the result is a negative number, and

[016] (a) if the decision is made to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down, but

[017] (b) if the decision is made not to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up.

[018] (c) In the third step, a packaging step, a result is packaged into a standard floating-point format. This may involve substituting a special representation, such as the representation defined for infinity or NaN if an exceptional situation (such as overflow, underflow, or an invalid operation) was detected. Alternatively, this may involve removing the leading 1-bit (if any) of the fraction, because such leading 1-bits are implicit in the standard format. As another alternative, this may involve shifting the fraction in order to construct a denormalized number. As a specific example, it is assumed that this is the step that forces the result to be a NaN if any input operand is a NaN. In this step, the decision is also made as to whether the result should be an infinity. It will be appreciated that, if the result is to be a NaN or infinity from step (b), the original result will be discarded and an appropriate representation will be provided as the result.

[019] In addition in the packaging step, floating point status information is generated, which is stored in a floating point status register. The floating point status information generated for a particular floating point operation includes indications, for example, as to whether

[020] (i) a particular operand is invalid for the operation to be performed ("invalid operation");

[021] (ii) if the operation to be performed is division, the divisor is zero ("division-by-zero");

[022] (iii) an overflow occurred during the operation ("overflow");

[023] (iv) an underflow occurred during the operation ("underflow"); and

[024] (v) the rounded result of the operation is not exact ("inexact").

[025] These conditions are typically represented by flags that are stored in the floating point status register. The floating point status information can be used to dynamically control the operations in response to certain instructions, such as

conditional branch, conditional move, and conditional trap instructions that may be in the instruction stream subsequent to the floating point instruction. Also, the floating point status information may enable processing of a trap sequence, which will interrupt the normal flow of program execution. In addition, the floating point status information may be used to affect certain ones of the functional unit control signals that control the rounding mode. IEEE Std. 754 also provides for accumulating floating point status information from, for example, results generated for a series or plurality of floating point operations.

[026] IEEE Std. 754 has brought relative harmony and stability to floating-point computation and architectural design of floating-point units. Moreover, its design was based on some important principles, and rests on a sensible mathematical semantics that eases the job of programmers and numerical analysts. It also supports the implementation of interval arithmetic, which may prove to be preferable to simple scalar arithmetic for many tasks. Nevertheless, IEEE Std. 754 has some serious drawbacks, including:

[027] (i) Modes (*e.g.*, the rounding modes and traps enabled/disabled mode), flags (*e.g.*, flags representing the status information), and traps required to implement IEEE Std. 754 introduce implicit serialization issues. Implicit serialization is essentially the need for serial control of access (read/write) to and from globally used registers, such as a floating point status register. Under IEEE Std. 754, implicit serialization may arise between (1) different concurrent floating-point instructions and (2) between floating point instructions and the instructions that read and write the flags and modes. Furthermore, rounding modes may introduce implicit serialization because they are typically indicated as global state, although in some microprocessor architectures, the rounding mode is encoded as part of the instruction operation code, which will alleviate this problem to that extent. Thus, the potential for implicit serialization makes the Standard difficult to implement coherently and efficiently in today's superscalar and parallel processing architectures without loss of performance.

[028] (ii) The implicit side effects of a procedure that can change the flags or modes can make it very difficult for compilers to perform optimizations

on floating point code. As a result, compilers for most languages usually assume that every procedure call is an optimization barrier in order to be safe. This unfortunately may lead to further loss of performance.

[029] (iii) Global flags, such as those that signal certain modes, make it more difficult to do instruction scheduling where the best performance is provided by interleaving instructions of unrelated computations. Thus, instructions from regions of code governed by different flag settings or different flag detection requirements cannot easily be interleaved when they must share a single set of global flag bits.

[030] (iv) Furthermore, traps have been difficult to integrate efficiently into computing architectures and programming language designs for fine-grained control of algorithmic behavior.

[031] Thus, there is a need for a system that avoids such problems when performing floating point operations and, in particular, when performing floating point multiplication with embedded status information associated with a floating point operand.

SUMMARY OF THE INVENTION

[032] Consistent with the current invention, a floating point multiplier with embedded status information method and system are provided that avoid the problems associated with prior art floating point multiplier systems as discussed herein above.

[033] In one aspect, a system for providing a floating point product comprises an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. In addition, the system comprises a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[034] In another aspect, a method for providing a floating point product comprises determining a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. In addition, the method comprises asserting a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[035] In yet another aspect, a computer-readable medium on which is stored a set of instructions for providing a floating point product, which when executed perform stages comprising determining a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. The instruction set further comprises asserting a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[036] Both the foregoing general description and the following detailed description are exemplary and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[037] The accompanying drawings provide a further understanding of the invention and, together with the detailed description, explain the principles of the invention. In the drawings:

[038] FIG. 1 is a functional block diagram of an exemplary system for providing a floating point product consistent with an embodiment of the present invention;

[039] FIG. 2 illustrates exemplary formats for representations of floating point values generated by the system of FIG. 1 consistent with an embodiment of the present invention;

[040] FIG. 3 illustrates a table useful in understanding the operations of the exemplary system of FIG. 1 consistent with an embodiment of the present invention; and

[041] FIGs. 4A through 4C depict exemplary patterns of input and output signals received and generated by a multiplier decision table logic circuit used in the exemplary system of FIG. 1 consistent with an embodiment of the present invention.

DESCRIPTION OF THE EMBODIMENTS

[042] Reference will now be made to various embodiments according to this invention, examples of which are shown in the accompanying drawings and will be obvious from the description of the invention. In the drawings, the same reference numbers represent the same or similar elements in the different drawings whenever possible.

[043] FIG. 1 is a functional block diagram of an exemplary multiplier unit 10 constructed in accordance with an embodiment of the invention. Generally, the multiplier unit 10 receives two floating point operands and generates therefrom a result and, in some cases, floating point status information, with the floating point status information being encoded in and comprising part of the floating point representation of the result. Since the floating point status information comprises part of the floating point representation of the result, instead of being separate and apart from the result as in prior art multiplier units, the implicit serialization that is required by maintaining the floating point status information separate and apart from the result can be obviated.

[044] The multiplier unit 10 encodes the floating point status information in results that are generated in certain formats. This will be illustrated in connection with FIG. 2. FIG. 2 depicts exemplary formats of floating point operands that the multiplier unit 10 may receive and of results that it generates. With reference to the embodiment illustrated in FIG. 2, seven formats are depicted, including a zero format 70, an underflow format 71, a denormalized format 72, a normalized non-zero format 73, an overflow format 74, an infinity format 75 and a not-a-number (NaN) format 76. The zero format 70 is used to represent the values "zero," or,

more specifically, positive or negative zero, depending on the value of "s," the sign bit.

[045] The underflow format 71 provides a mechanism by which the multiplier unit 10 can indicate that the result of a computation is an underflow. In the underflow format, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary zero's, and the bits $f_{msb}...f_{lsb+1}$ of the fraction field, except for the least significant bit, are all binary zero's. The least significant bit f_{lsb} of the fraction field is a binary one.

[046] The denormalized format 72 and normalized non-zero format 73 are used to represent finite non-zero floating point values substantially along the lines of that described above in connection with IEEE Std. 754. In both formats 72 and 73, the sign bit "s" indicates whether the result is positive or negative. The bits $e_{msb}...e_{lsb}$ of the exponent field of the denormalized format 72 are all binary zero's, whereas the bits $e_{msb}...e_{lsb}$ of the exponent field of the normalized non-zero format 73 are mixed one's and zero's, except that the exponent field of the normalized non-zero format 73 will not have a pattern in which bits $e_{msb}...e_{lsb+1}$ are all binary ones and the least significant bit e_{lsb} zero and all of the bits $f_{msb}...f_{lsb}$ of the fraction field are all binary one's. In both formats 72 and 73, the bits $f_{msb}...f_{lsb}$ of the fraction field are not all binary zero's.

[047] The overflow format 74 provides a mechanism by which the multiplier unit 10 can indicate that the result of a computation is an overflow. In the overflow format 74, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb+1}$ of the exponent field are all binary ones, with the least significant bit e_{lsb} being zero. The bits $f_{msb}...f_{lsb}$ of the fraction field are all binary ones.

[048] The infinity format 75 provides a mechanism by which the multiplier unit 10 can indicate that the result is infinite. In the infinity format 75, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb}...f_{lsb+5}$ of the fraction field are all binary zero's. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags, which will be described below.

[049] The NaN format 76 provides a mechanism by which the multiplier unit 10 can indicate that the result is not a number. In the NaN format, the sign bit "s" can be any value, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb}...f_{lsb+5}$ of the fraction field are not all binary zero's. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags, which will be described below.

[050] As noted above, in values represented in the infinity format 75 and the NaN format 76, the five low order bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags. In the formats used with the multiplier unit 10, the five flags include the flags that are defined by IEEE Std. 754, including an invalid operation flag "n", an overflow flag "o", an underflow flag "u", a division-by-zero flag "z", and an inexact flag "x". For example, a value in the NaN format 76 in which both the overflow flag "o" and the division-by-zero flag "z" are set, indicates that the value represents a result of a computation that involved an overflow (this from the overflow flag "o"), as well as an attempt to divide by zero (this from the division-by-zero flag "z"). It should be noted that the flags provide the same status information as provided by, for example, information stored in a floating point status register in a prior art floating point unit. Because the information is provided as part of the result and stored therewith in registers in which the result is ultimately stored, multiple instructions can be contemporaneously executed. This is because the floating point status information that may be generated during execution of one instruction, when stored, will not over-write previously-stored floating point status information generated during execution of another instruction.

[051] In addition to including status information in the five low-order bits $f_{lsb+4}...f_{lsb}$ of the fraction field for values in the NaN format 76, other information can also be encoded in the next five low-order bits $f_{lsb+9}...f_{lsb+5}$. If the value in the NaN format 76 is the result of an operation, the other information indicates the operation and types of operands that gave rise to the result. In one embodiment, the other information is associated with binary encoded values (BEV) of those bits $f_{lsb+9}...f_{lsb+5}$ as follows:

Bit Pattern Of Result	BEV of $f_{lsb+9}...f_{lsb+5}$	Meaning
	0 or 1	no specific meaning
s 111111111 00000000000000010nouzx	2	infinity minus infinity
s 111111111 00000000000000011nouzx	3	OV minus OV
s 111111111 00000000000000100nouzx	4	zero times infinity
s 111111111 00000000000000101nouzx	5	UN times OV
	6 or 7	no specific meaning
s 111111111 00000000000001000nouzx	8	zero divided by zero
s 111111111 00000000000001001nouzx	9	infinity divided by infinity
s 111111111 00000000000001010nouzx	10	UN divided by UN
s 111111111 00000000000001011nouzx	11	OV divided by OV
s 111111111 00000000000001100nouzx	12	square root of less than zero
	13-16	no specific meaning
s 111111111 00000000000010001nouzx	17	remainder by zero
s 111111111 00000000000010010nouzx	18	remainder by UN
s 111111111 00000000000010011nouzx	19	remainder by OV
s 111111111 00000000000010100nouzx	20	remainder of infinity
s 111111111 00000000000010101nouzx	21	remainder of infinity by zero
s 111111111 00000000000010110nouzx	22	remainder of infinity by UN
s 111111111 00000000000010111nouzx	23	remainder of infinity by OV
s 111111111 00000000000011000nouzx	24	remainder of OV
s 111111111 00000000000011001nouzx	25	remainder of OV by zero
s 111111111 00000000000011010nouzx	26	remainder of OV by UN
s 111111111 00000000000011011nouzx	27	remainder of OV by OV
	28-31	no specific meaning

[052] In the following, it will be assumed that the formats represent thirty-two bit values; extension to, for example, sixty-four bit values or values represented in other numbers of bit will be readily apparent to those skilled in the art. Additionally, "OV" refers to an operand in the overflow format 74, "UN" refers to an operand in the underflow format 71 and "infinity" refers to an operand in the infinity format 75.

[053] In addition, it will be convenient in the following to have names for the finite nonzero numbers that are adjacent to +OV (a value in the overflow pattern with the sign bit "s" having the value "zero" indicating a positive value), -OV (a value in the overflow pattern with the sign bit "s" having the value "one," indicating a negative value), +UN (a value in the underflow pattern with the sign bit "s" having the value "zero,." indicating a positive value), and -UN (a value in the underflow pattern with the sign bit "s" having the value "one," indicating a negative value), as follows:

0 00000000 00000000000000000000010	+TINY
1 00000000 00000000000000000000010	-TINY
0 11111110 11111111111111111111110	+HUGE
1 11111110 11111111111111111111110	-HUGE

[054] Generally, +OV can be deemed to refer to "some (or any) value that is strictly between +HUGE and +∞" and +UN can be deemed to refer to "some (or any) value that is strictly between +0 and +TINY". Similarly, -OV can be deemed to refer to "some (or any) value that is strictly between -HUGE and -∞" and -UN can be deemed to refer to "some (or any) value that is strictly between -0 and -TINY."

[055] With this background, the structure and operation of the exemplary multiplier unit 10 will be described in connection with FIG. 1 and consistent with an embodiment of the invention. With reference to FIG. 1, the exemplary multiplier unit 10 includes two operand buffers 11A and 11B, respective operand analysis circuits 12A and 12B, a multiplier core 13, a result assembler 14 and a multiplier decision table logic circuit 15. The operand buffers 11A and 11B receive and store respective operands from, for example, a set of registers (not shown) in a conventional manner. The multiplier core 13 receives the operands from the operand buffers 11A and 11B, except as described below, and rounding mode information from, for example, a rounding mode store 16. The multiplier core 13 then generates a result in accordance with IEEE Std. 754. Multiplier core 13 is conventional and will not be described in detail herein.

[056] Each operand analysis circuit 12A, 12B analyzes the operand in the respective buffer 11A, 11B and generates signals providing information relating thereto, which signals are provided to the multiplier decision table logic circuit 15. The result assembler 14 receives information from a number of sources, including the operand buffers 11A and 11B, multiplier core 13 and several predetermined value stores as described below. Under control of control signals from the multiplier decision table logic circuit 15, the result assembler 14 assembles the result, which is provided on a result bus 17. The result bus 17, in turn, may deliver the result to any convenient destination, such as a register in a register set (not shown), for storage or other use.

[057] The system for providing a floating point product may comprise an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand, respectively. In one embodiment, the analyzer circuit includes buffers 11A, 11B and analysis circuits 12A, 12B. In addition, the system for providing a floating point product includes a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand. The results circuit may be implemented with a multiplier circuit (comprising the multiplier core 13), the multiplier decision logic table circuit 15, and result assembler 14.

[058] Those skilled in the art will appreciate that the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. It may also be provided using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, the invention may be practiced within a general purpose computer or in any other circuits or systems as are known by those skilled in the art.

[059] As noted above, each operand analysis circuit 12A, 12B analyzes the operand in the respective buffer 11A, 11B and generates signals providing information relating thereto. These signals are provided to the multiplier decision table logic circuit 15. In the exemplary embodiment, each operand analysis circuit 12A, 12B is implemented with a number of comparators, including:

[060] (i) a comparator 20A, 20B that generates an asserted signal if the bits $e_{msb} \ldots e_{lsb}$ of the exponent field of the operand in respective buffer 11A, 11B are all binary one's, which will be the case if the operand is in the infinity format 75 or the NaN format 76;

[061] (ii) a comparator 21A, 21B that generates an asserted signal if the bits $e_{msb}...e_{lsb+1}$ of the exponent field of the operand in the respective buffer 11A, 11B are all binary one's and the bit e_{lsb} is a binary zero, which will be the case if the operand is in the overflow format 74;

[062] (iii) a comparator 22A, 22B that generates an asserted signal if the bits $e_{msb}...e_{lsb+1}$ of the exponent field of the operand in respective buffer 11A, 11B are all binary one's and the bit e_{lsb} is either a binary one or a binary zero, which will be the case if the operand is in the overflow format 74, infinity format 75 or the NaN format 76;

[063] (iv) a comparator 23A, 23B that generates an asserted signal if the bit e_{msb} of the exponent field of the operand in respective buffer 11A, 11B is a binary one and respective bits $e_{msb-1}...e_{lsb}$ are either binary one or a binary zero, which may be the case if the operand is in the normalized non-zero format 73 and will be the case for the overflow format 74, infinity format 75 or NaN format 76;

[064] (v) a comparator 24A, 24B that generates an asserted signal if the bit e_{msb} of the exponent field of the operand in respective buffer 11A, 11B is a binary zero and bits $e_{msb-1}...e_{lsb}$ are all binary one's, which may be the case if the operand is in the normalized non-zero format 73 and will be the case if the operand has the value +1.0 or -1.0;

[065] (vi) a comparator 25A, 25B that generates an asserted signal if the bit e_{msb} of the exponent field of the operand in respective buffer 11A, 11B is a binary zero and respective bits $e_{msb-1}...e_{lsb}$ are either binary one or binary zero, which will be the case if the operand is in the zero format 70, underflow format 71, or denormalized format 72 and may be the case for normalized non-zero format 73;

[066] (vii) a comparator 26A, 26B that generates an asserted signal if the bit $e_{msb}...e_{lsb}$ of the exponent field of the operand in respective buffer 11A, 11B are all binary zero's, which will be the case if the operand is in the zero format 70, underflow format 71, or denormalized format 72;

[067] (viii) a comparator 30A, 30B that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in the respective buffer 11A, 11B

are all binary ones, which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or NaN format 76;

[068] (ix) a comparator 31A, 31B that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of fraction field of the operand in the respective buffer 11A, 11B are all binary zero's, which may be the case if the operand is in the zero format 70, underflow format 72, denormalized format 72, normalized non-zero format 73 or infinity format 75;

[069] (x) a comparator 32A, 32B that generates an asserted signal if the bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary one's, which may be the case if the operand is in the denormalized format 72 or normalized non-zero format 73 and which will be the case if the operand is in the overflow format 74, or if all of the flags "n," "o," "u," "z," and "x" are set in the infinity format 75 or NaN format 76;

[070] (xi) a comparator 33A, 33B that generates an asserted signal if the bits $f_{lsb+4}...f_{lsb+1}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary zero's and if the bit f_{lsb} of the fraction field is either a binary "zero" or "one," which will be the case if the operand is in the zero format 70 or underflow format 71 and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or if the flags "n," "o," "u," and "z" are clear and the flag "x" is either set or clear in the infinity format 75 or NaN format 76;

[071] (xii) a comparator 34A, 34B that generates an asserted signal if the bits $f_{lsb+4}...f_{lsb+1}$ of the fraction field of the operand in the respective buffer 11A, 11B are binary zero's and if the bit f_{lsb} of the fraction field is a binary "one," which will be the case if the operand is in the underflow format 71 and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or if the flags "n," "o," "u," and "z" are clear and the flag "x" is set in the infinity format 75 or NaN format 76; and

[072] (xiii) a comparator 35A, 35B that generates an asserted signal if all of the bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in the respective buffer 11A, 11B are binary zero's, which will be the case if the operand is in the zero format 70,

and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, overflow format 74, or if the flags "n," "o," "u," "z" and "x" are clear in the infinity format 75 or NaN format 76.

[073] In the illustrated embodiment, each exemplary operand analysis circuit 12A, 12B also includes combinatorial logic elements that receive selected ones of the signals from the comparators and generate characteristic signals to provide indications as to certain characteristics of the respective operand. In more detail, such combinational logic elements include:

[074] (xiv) an AND gate 50A, 50B, which will generate an a asserted signal if comparators 31A, 31B, and 35A, 35B are both generating asserted signals, which will be the case if the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A, 11B have the bit pattern 00000000000000000000;

[075] (xv) an AND gate 51A, 51B, which will generate an asserted signal if comparators 31A, 31B, and 34A, 34B, are both generating asserted signals, which will be the case if the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A, 11B have the bit pattern 00000000000000000001;

[076] (xvi) an AND gate 52A, 52B, which will generate an asserted signal if comparators 30A, 30B, and 32A, 32B are both generating asserted signals, which will be the case if the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A, 11B have the bit pattern 11111111111111111111;

[077] (xvii) an AND gate 40A, 40B that generates an asserted signal if the signals generated by both comparator 31A, 31B and comparator 33A, 33B are asserted, which will be the case if the respective operand is in the zero format 70 or underflow formal 71 and which may be the case if the operand is in the denormalized format 72, normalized non-zero format 73, or if the flags "n," "o," "u" and "z" are clear and the flag "x" is either set or clear in the infinity format 75; otherwise stated, AND gate 40A, 40B will generate an asserted signal if the value of the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer

11A, 11B has the bit pattern 00000000000000000000001 or the bit pattern 00000000000000000000000;

[078] (xviii) a NAND gate 41A, 41B that generates an asserted signal if the signal generated by comparator 26A, 26B is asserted and the signal generated by AND gate 40A and 40B is negated, which will be the case if the respective operand is in the denormalized format 72; otherwise stated, NAND gale 41A, 41B will generate an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B have the pattern 00000000 and a bit of the fraction field, other than the low order bit f_{lsb}, is a "one";

[079] (xix) a NAND gate 42A, 42B that generates an asserted signal if the signal generated by comparator 24A, 24B is negated and the signal generated by comparator 25A, 25B is asserted and the signal generated by comparator 26A, 26B is negated, which will be the case if the respective operand is in the normalized format 73 and its magnitude is less than 1.0; otherwise stated, NAND gate 42A, 42B will generate an asserted signal if the high-order bit e_{msb} of the exponent field of the operand in the respective operand buffer 11A, 11B is a "zero" but the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B do not have the bit pattern 00000000 or 01111111;

[080] (xx) an OR gate 43A, 43B that generates an asserted signal if either NAND gate 41A, 41B or NAND gate 42A, 42B is asserted, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B have the pattern 00000000 and a bit of the fraction field, other than the low order bit f_{lsb}, is a "one" (NAND gate 41A, 41B) or if the high-order bit e_{msb} of the exponent field is a "zero", and the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B do not have the bit pattern 00000000 or 01111111 (NAND gate 42A, 42B);

[081] (xxi) an AND gate 44A, 44B that generates an asserted signal if comparator 24A, 24B is generating an asserted signal and AND gate 50A, 50B is generating an asserted signal, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the same operand buffer 11A, 11B have the bit

pattern 01111111 and the bits $f_{msb}...f_{lsb}$ of the fraction field of the operand in the same operand buffer 11A, 11B have the bit pattern 00000000000000000000000;

[082] (xxii) a NAND gate 45A, 45B that generates an asserted signal if comparator 24A, 24B is generating an asserted signal and AND gate 50A, 50B is generating a negated signal, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B have the bit pattern 01111111 and at least one bit $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A, 11B is "one";

[083] (xxiii) a NAND gate 46A, 46B that generates an asserted signal if comparator 23A, 23B is generating an asserted signal and comparator 22A, 22B is generating a negated signal, which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B does not have either bit pattern 11111110 or 11111111, but the high-order bit e_{msb} of the exponent field of the operand in the respective operand buffer 11A, 11B is "one";

[084] (xxiv) a NAND gate 47A, 47B, which will generate an asserted signal if comparator 21A, 21B is generating an asserted signal and AND gate 52A, 52B is generating a negated signal which will be the case if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective operand buffer 11A, 11B have the bit pattern 11111110 and not every bit $f_{msb}...f_{lsb}$ of the fraction field of the operand in the respective operand buffer 11A, 11B is "one"; and

[085] (xxv) an OR gate 48A, 48B, which will generate an asserted signal if one or more of NAND gate 45A, 45B and NAND gate 46A, 46B and NAND gate 47A, 47B generates an asserted signal.

[086] In addition, the combinatorial logic in the illustrated embodiment includes a comparator 53 that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A represent a binary-encoded value that is larger than the binary-encoded value represented by bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B. The combinatorial logic also includes an XOR gate 54 that generates an asserted signal if the sign bits "s" of the operands in the operand buffers 11A and 11B have different values.

[087] Each exemplary operand analysis circuit 12A, 12B provides signals to the multiplier decision table logic 15 as shown in the following table:

(a) the signal generated by comparator 26A, 26B
(b) the signal generated by the comparator 21A, 21B
(c) the signal generated by the comparator 20A, 20B
(d) the signal generated by comparator 31A, 31B
(e) the signal generated by AND gate 50A, 50B
(f) the signal generated by AND gate 51A, 51B
(g) the signal generated by AND gate 52A, 52B
(h) the signal generated by the OR gate 43A, 43B
(i) the signal generated by the AND gate 44A, 44B
(j) the signal generated by the OR gate 48A, 48B

[088] In addition, the signal generated by comparator 53 is provided to the multiplier decision table logic 15, as are signals from rounding mode store 16 representative of the rounding mode and the signal generated by the XOR gate 54.

[089] In addition, the multiplier core 13 generates an overflow signal and an underflow signal, which are also provided to the multiplier decision table logic 15. The underflow signal is asserted if and only if the mathematical product should be non zero but the multiplier core produces a zero result (those skilled in the art will appreciate that this differs from the IEEE 754 definition of underflow). In the illustrated embodiment, the multiplier decision table logic 15 generates control signals for controlling the result assembler 14. In addition, control signals generated by the multiplier decision table logic 15 control sets of XOR gates 67A, 67B; 68A, 68B that control toggling of one or both of the two least significant bits f_{lsb}, f_{lsb+1} of the fraction field of the operands before they are provided to the multiplier core 13. As noted above, the result assembler 14 receives information from a number of sources, including the XOR gate 54, the operand buffers 11A and 11B, multiplier core 13 and several predetermined value stores as described below.

[090] Under control of control signals from the multiplier decision table logic circuit 15, the result assembler 14 assembles the appropriate results representing a product of the two operands, onto a result bus 17. In general, the result assembler

14 essentially assembles the result in four segments, including a sign segment that represents the sign bit of the result, an exponent segment that represents the exponent field of the result, a high-order fraction segment that represents the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result, and a low-order fraction segment that represents the five least significant bits $f_{lsb+4}...f_{lsb}$ of the result. It will be appreciated that the low-order fraction segment, in results in the infinity format 75 and NaN format 76, corresponds to the flags "n," "o," "u," "z", and "x". One or more of these segments will represent an embedded resulting status of the resulting floating point operand.

[091] In the illustrated embodiment, the result assembler 14 may include four elements, including a link from XOR gate 54, an exponent field selector 60, a high-order fraction field selector 61 and low-order fraction field combiner 62. The link from XOR gate 54 provides the sign of the result. As is conventional, the sign of the result is positive, in which case the sign bit would have the value "zero," if the sign bits of the operands in both operand buffers 11A, 11B are the same. On the other hand, the sign of the result is negative, in which case the sign bit would have the value "one," if the sign bits of the operands in the operand buffers 11A, 11B differ. It will be appreciated that the XOR gate 54 generates a signal of the appropriate value for the sign bit for the result, which signal is coupled onto the result bus 17.

[092] As noted above, the exemplary selector 60 couples exponent value signals representative of the exponent field of the result to the result bus 17. In this embodiment, the selector 60 receives four sets of exponent field value signals, namely, the signals from the multiplier core 13 associated with the exponent field as well as three sets of signals representative of three predetermined exponent field bit patterns, such as the patterns depicted in FIG. 1. It will be appreciated that these predetermined exponent field bit patterns correspond to the exponent fields associated with the zero format 70, underflow format 71, overflow format 74, infinity format 75 and NaN format 76. In addition, the selector 60 receives four exponent field control signals from the multiplier decision table logic 15. One of these control signals is associated with the signals from the multiplier core 13 that are associated

with the exponent field. The rest of these control signals are associated with each of the sets of exponent field value signals, respectively. In enabling the result assembler 14 to assemble the result, the multiplier decision table logic 15 will assert one of the four exponent field control signals. Additionally, the selector 60 will couple the set of exponent field value signals associated with the asserted exponent field control signal to the result bus 17 to provide the exponent field of the result.

[093] The exemplary selector 61 couples high-order fraction field signals representative of the high-order fraction field bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result to the result bus 17. In the illustrated embodiment, the selector 61 receives seven sets of high-order fraction field value signals, namely, the signals from the multiplier core 13 associated with the high-order fraction field, signals representative of bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in buffer 11B, as well as four sets of signals representative of four predetermined high-order fraction field bit patterns, such as the patterns depicted in FIG. 1. It will be appreciated that these predetermined high-order fraction field bit patterns correspond to the high-order fraction fields associated with the zero format 70, underflow format 71, overflow format 74, infinity format 75 and NaN format 76. In addition, the selector 61 receives seven high-order fraction field control signals from the multiplier decision table logic 15. One of these control signals is associated with the multiplier core 13. The rest of these control signals are associated with each of the sets of high-order fraction field value signals, respectively. It will be appreciated that the control signal associated with the multiplier core 13 is the same control signal that controls the selector 60. In enabling the result assembler 14 to assemble the result, the multiplier decision table logic will assert one of the seven high-order fraction field control signals, and the selector 61 will couple the set of high-order fraction field value signals associated with the asserted high-order fraction field control signal to the result bus 17 to provide bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result.

[094] Similarly, the exemplary combiner 62 couples low-order fraction field value signals representative of the low-order fraction field bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result to the result bus 17. The combiner 62 receives four sets of

low-order fraction field signals, namely, the signals from the multiplier core 13 associated with the low-order fraction field, signals representative of bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in buffer 11B, and one set of signals from the multiplier decision table logic 15. It will be appreciated that in the illustrated embodiment, the set of signals provided by the multiplier decision table logic 15 are used in controlling the condition of flags "n", "o", "u", "z", and "x" for those formats in which the low order bits $f_{lsb+4}...f_{lsb}$ represent flags. In addition, the sets of signals provided by the operands in buffers 11A and 11B may also represent the flags "n", "o", "u", "z", and "x". In addition, the combiner 62 receives three low-order fraction field control signals from the multiplier decision table logic 15. One control signal is associated with the sets of low-order fraction field value signals provided by the multiplier core and the two others are associated with the sets of signals provided by the buffers 11A and 11B.

[095] In enabling the result assembler 14 to assemble the result, the multiplier decision table logic 15 may provide signals representative of the low-order fraction field and negate all of the low-order fraction field control signals. When this occurs, the signals representative of the low order fraction field provided by the multiplier decision table logic 15 will be coupled to the result bus 13 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result.

[096] Alternatively, the multiplier decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert one of the three low-order fraction field control signals. When this occurs, the combiner 62 will couple the set of low-order fraction field value signals associated with the asserted low-order fraction field control signal to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result. As a further alternative, the multiplier decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert more than one of the three low-order fraction field control signals. As a result, the combiner 62 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result.

As yet another alternative, the multiplier decision table logic 15 may assert one or more of the low-order fraction field value signals provided thereby and assert one or more of the three low-order fraction field control signals. As a result, the combiner 62 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals and the low-order fraction field value signals provided by the multiplier decision table logic 15 to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result.

[097] In more detail, the exemplary combiner 62 in the illustrated embodiment comprises an OR circuit 63 and three AND circuits 64 through 66. (Each gate in the diagram actually represents five such gates, one for each bit position $f_{lsb+4}...f_{lsb}$, but for the sake of clarity and to avoid confusion, the diagram illustrates them as a single gate.) The AND circuits 64-66 receive the low-order fraction field value signals from the multiplier core 13 and operand buffers 11A and 11B respectively, as well as the respective low-order fraction field control signal. These AND circuits 64-66 perform a bit-wise AND operation to, if the respective low-order fraction field control signal is asserted, couple the low-order fraction field value signals to a respective input of OR circuit 63. The OR circuit 63, whose output is connected to the result bus 17, performs a bit-wise OR operation in connection with the signals that it receives from the AND circuits 64-66 and the low-order fraction field value signals provided by the multiplier decision table logic 15. If the multiplier decision table logic 15 negates all of the low-order fraction field control signals, the AND circuits 64-66 will block the low-order fraction field value signals that they receive, and the signals provided by the OR circuit 63 will conform to the low-order fraction field value signals provided by the multiplier decision table logic 15.

[098] On the other hand, if the multiplier decision table logic 15 asserts one or more of the low-order fraction field control signals, the AND circuits 64-66 that receive the asserted low-order fraction field control signal will couple the low-order fraction field value signals that they receive to the OR circuit 63 and the other AND gates will block the low-order fraction field signal that they receive. As will be described below, under some circumstances, the multiplier decision table logic 15 will assert two low-order fraction field control signals to enable two sets of low-order

fraction field value signals to be coupled to the OR circuit 63. In that case, the OR gate will perform a bit-wise OR operation in connection with signals representing respective bits of the low-order fraction field.

[099] Thus, multiplier decision table logic 15 will assert two low-order fraction signals if, for example, both operands in operand buffers 11A and 11B are in NaN format to enable the respective flags "n," "o," "u," and "x" to be ORed together. However, if the low-order fraction field value signals provided by the multiplier decision table logic 15 are negated, the low-order fraction field value signals provided by the OR circuit 63 will conform to the low-order fraction field signals provided by the AND circuit or circuits that receive the asserted low-order fraction field control signal.

[0100] As noted above, the multiplier decision table logic 15 generates control signals for controlling the selectors 60 and 61 and combiner 62 comprising the result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffers 11A and 11B before they are presented to the multiplier core 13. The control signals generated by the multiplier decision table logic 15 are such as to enable the result to be assembled in the desired format 70-76 having status information embedded within the result itself. Before proceeding further, it would be helpful to describe the results that are to be generated by the multiplier unit 10.

[0101] Generally, exemplary results generated by the multiplier unit 10 are described in the table depicted in FIG. 3. In that table, one skilled in the art will appreciate that "+P" or "+Q" means any finite positive representable value greater than "one," other than +OV (that is, a value in the overflow format 74 with the sign bit "s" being "zero"). "-P" or "-Q" means any finite negative representable value less than negative-one, other than -OV (that is, a value in the overflow format 74, with the sign bit being "one"). "+R" or "+S" means any positive non-zero representable value less than "one," other than +UN (that is, a value in the underflow format 71 with the sign bit "s" being "zero"). "-R" or "-S" means any negative non-zero representable value greater than negative-one, other than -UN (that is, a value in the underflow format 71, with the sign bit being "one"). Finally, those skilled in the

art will appreciate that "NaN" means any value whose exponent field is 11111111, other than one of the values represented by +∞ (that is, a value in the infinity format 75, with the sign bit "s" being "zero") and -∞ (that is, a value in the infinity format 75, with the sign bit "s" being "one").

[0102] Key to symbols in the table with exemplary results depicted in FIG. 3 are as follows:

[0103] (a) The result is +∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the two operands.

[0104] (b) The result is +∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand with 01001 (to indicate overflow and inexact).

[0105] (c) The result is +∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand.

[0106] (d) The result is +∞, with five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand with 00101 (to indicate underflow and inexact).

[0107] (e) For "round toward plus infinity", the result is +∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand. For "round toward minus infinity," the result is +0. For all other rounding modes, the result is a positive NaN value 0 11111111 10000000000000001001ouzx (to indicate "zero times infinity" with the invalid operation flag set), where ouzx are the four least significant bits $f_{lsb+3}...f_{lsb}$ of the fraction field of the infinite operand.

[0108] (f) For "round toward plus infinity," the result is -0. For "round toward minus infinity," the result is -∞, with five least significant bits $f_{lsb+4}...f_{lsb}$ of the

fraction field being equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand. For all other rounding modes, the result is a negative NaN value 1 11111111 10000000000000001001ouzx (to indicate "zero times infinity" with the invalid operation flag set), where ouzx are the four least significant bits $f_{lsb+3}...f_{lsb}$ of the fraction field of the infinite operand.

[0109] (g) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$, of the fraction field of the infinite operand with 00101 (to indicate underflow and inexact).

[0110] (h) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field being equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand.

[0111] (i) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand with 01001 (to indicate overflow and inexact).

[0112] (j) The result is -∞, with five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operands.

[0113] (k) The result is a copy of the NaN operand, except that its sign is reversed if the other operand is negative, and that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operands.

[0114] (l) For "round toward plus infinity," the result is the same as if -OV were replaced by -HUGE. For all other rounding modes, the result is +OV.

[0115] (m) For "round toward plus infinity," the result is +OV. For "round toward minus infinity," the result is +UN. For all other rounding modes, the result is the positive NaN value 0 11111111 10000000000000010111101 (to indicate "UN times OV" with the invalid operation, overflow, underflow, and inexact flags set).

[0116] (n) For "round toward plus infinity," the result is -UN. For "round toward minus infinity," the result is -OV. For all other rounding modes, the result is the negative NaN value 1 11111111 10000000000000010111101 (to indicate "UN times OV" with the invalid operation, overflow, underflow, and inexact flags set).

[0117] (o) For "round toward minus infinity," the result is the same as if -OV were replaced by -HUGE. For all other rounding modes, the result is -OV.

[0118] (p) The result is a copy of the NaN operand, except that its sign is reversed if the other operand is negative. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are ORed with 01001 (to indicate overflow and inexact).

[0119] (q) The result is as computed in accordance with IEEE Std. 754. However, if overflow occurs or if the rounding mode is "round toward plus infinity" and the mathematical product is greater than +HUGE, the result is +OV. Further, if underflow occurs and a computation in accordance with IEEE Std. 754 would result in the value +0 or if the rounding mode is "round toward minus infinity" and the mathematical product is less than +TINY, the result is +UN.

[0120] (r) For "round toward plus infinity," the result is the same as if -UN were replaced by -TINY. For all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0121] (s) For "round toward minus infinity," the result is the same as if +UN were replaced by +TINY. For all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0122] (t) The result is as computed in accordance with IEEE Std. 754. However, if overflow occurs or if the rounding mode is "round toward minus infinity" and the mathematical product is less than -HUGE, the result is -OV. Further, if underflow occurs and a computation in accordance with IEEE Std 754 would provide the result -0 or if the rounding mode is "round toward plus infinity" and the mathematical product is greater than -TINY, the result is -UN.

[0123] (u) The result is a copy of the NaN operand, except that its sign is reversed if the other operand is negative.

[0124] (v) For "round toward minus infinity," the result is the same as if +OV were replaced by +HUGE. For all other rounding modes, the result is -OV.

[0125] (w) For "round toward minus infinity," the result is the same as if -UN were replaced by -TINY. For all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0126] (x) The result is a copy of the NaN operand, except that its sign is reversed if the other operand is negative. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are ORed with 00101 (to indicate underflow and inexact).

[0127] (y) For "round toward plus infinity," the result is the same as if +UN were replaced by +TINY. For all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0128] (z) For "round toward minus infinity," the result is the same as if +OV were replaced by +HUGE. For all other rounding modes, the result is +OV.

[0129] (@) The result is a copy of the NaN operand that has the larger value in the fraction field, except that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operands and the sign bit of the result is 1, indicating a negative result, if and only if the sign bits of the two NaN operands differ.

[0130] Multiplication is commutative with multiplier unit 10 operating according to the table depicted in FIG. 3. This is true even with those cases where one or both operands are values in the NaN format 76.

[0131] As noted above, multiplier decision table logic 15 generates control signals for controlling the selectors 60, 61 and combiner 62 comprising the result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffers 11A and 11B before they are presented to the multiplier core 13. The particular signals that the multiplier decision table logic 15 will generate depends on the signals provided thereto by the operand buffers 11A and 11B representing the states of the respective sign bits, the operand analysis circuits 12A and 12B, comparator 40, rounding mode store 16, and the overflow and underflow signals from the multiplier core 13. In the illustrated

embodiment, the series of input signals received by the multiplier decision table logic 15 are as follows:

[0132] (a) a signal from comparator 26A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 00000000;

[0133] (b) a signal from comparator 21A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 11111110;

[0134] (c) a signal from comparator 20A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 11111111;

[0135] (d) a signal from the comparator 31A that is asserted if the operand in operand buffer 11A has a high order fraction field with all 0-bits;

[0136] (e) a signal from AND gate 50A that is asserted if the operand in operand buffer 11A has high- and low-order fraction fields with the collective bit pattern 00000000000000000000000;

[0137] (f) a signal from AND gate 51A that is asserted if the operand in operand buffer 11A has high- and low-order fraction fields with the collective bit pattern 00000000000000000000001;

[0138] (g) a signal from AND gate 52A that is asserted if the operand in operand buffer 11A has high- and low-order fraction fields with the collective bit pattern 11111111111111111111111;

[0139] (h) a signal from OR gate 43A that is asserted if any of the following signals are asserted:

[0140] (1) a signal from NAND gate 41A that is asserted if the exponent field of the operand in operand buffer 11A has a bit pattern 00000000 (which will be the case if the signal from comparator 26A is asserted) and the high- and low-order fraction field of the operand in operand buffer 11A has a bit pattern in which at least one bit other than the least significant bit, is "1" (which will be the case if the signal from AND gate 40A is negated); and

[0141] (2) a signal from NAND gate 42A that is asserted if the exponent field of the operand in operand buffer 11A does not have a bit pattern 01111111 (which will be the case if the signal from comparator 24A is negated) and also does not have the bit pattern 00000000 (which will be the case if the signal from

comparator 26A is negated), but the high-order bit e_{msb} of the exponent field in operand buffer 11A is "0" (which will be the case if the signal from comparator 25A is asserted);

[0142] (i) a signal from AND gate 44A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 01111111 (which will be the case if the signal from comparator 24A is asserted) and the bits of the fraction field of the operand in operand buffer 11A are all "0" (which, will be the case if the signal from AND gate 50A is asserted);

[0143] (j) a signal from OR gate 48A that is asserted if any of the following signals are asserted:

[0144] (1) a signal from NAND gate 45A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 01111111 (which will be the case if the signal from comparator 24A is asserted) and at least one bit of the fraction field of the operand in operand buffer 11A is "1" (which will be the case if the signal from AND gate 50A is negated);

[0145] (2) a signal from NAND gate 46A that asserted if the exponent field of the operand in operand buffer 11A does not have the bit pattern 11111110 or 11111111 but the high-order bit of the exponent field is 1 (which will be the case if the signal from comparator 23A is asserted and the signal from comparator 22A is negated); and

[0146] (3) a signal from NAND gate 47A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 11111110 (which will be the case if the signal from comparator 21A is asserted) and at least one bit of the fraction in the fraction field of the operand in operand buffer 11A is "0" (which will be the case if the signal from AND gate 52A is negated);

[0147] (k) a signal from comparator 26B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 00000000;

[0148] (l) a signal from comparator 21B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 11111110;

[0149] (m) a signal from comparator 20B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 11111111;

[0150] (n) a signal from the comparator 31B that is asserted if the operand in operand buffer 11B has a high-order fraction field with all 0-bits;

[0151] (o) a signal from AND gate 50B that is asserted if the operand in operand buffer 11B has high- and low-order fraction fields with the collective bit pattern 00000000000000000000000;

[0152] (p) a signal from AND gate 51 B that is asserted if the operand in operand buffer 11B has high- and low-order fraction fields with the collective bit pattern 00000000000000000000001;

[0153] (q) a signal from AND gate 52B that is asserted if the operand in operand buffer 11B has high- and low-order fraction fields with the collective bit pattern 11111111111111111111111;

[0154] (r) a signal from OR gate 43B that is asserted if any of the following signals are asserted:

[0155] (1) a signal from NAND gate 41B that is asserted if the exponent field of the operand in operand buffer 11B has a bit pattern 00000000 (which will be the case if the signal from comparator 26B is asserted) and the high- and low-order fraction field of the operand in operand buffer 11B has a bit pattern in which at least one bit, other than the least significant bit, is "1" (which will be the case if the signal from AND gate 40B is negated); and

[0156] (2) a signal from NAND gate 42B that is asserted if the exponent field of the operand in operand buffer 11B does not have a bit pattern 01111111 (which will be the cast if the signal from comparator 24B is negated) and also does not have the bit pattern 00000000 (which will be the case if the signal from comparator 26B is negated), but the high-order bit e_{msb} of the exponent field in operand buffer 11B is "0" (which will be the case if the signal from comparator 25B is asserted);

[0157] (s) a signal from AND gate 44B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 01111111 (which will be the case if the signal from comparator 24B is asserted) and the bits of the fraction field of the operand in operand buffer 11B are all "0" (which will be the case if the signal from AND gate 50B is asserted);

[0158] (t) a signal from OR gate 48B that is asserted if any of the following signals are asserted:

[0159] (1) a signal from NAND gate 45B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 01111111 (which will be the case if the signal from comparator 24B is asserted) and at least one bit of the fraction field of the operand in operand buffer 11B is "1" (which will be the case if the signal from AND gate 50B is negated);

[0160] (2) a signal from NAND gate 46B that asserted if the exponent field of the operand in operand buffer 11B does not have the bit pattern 11111110 or 11111111 but the high-order bit of the exponent field is 1 (which will be the case if the signal from comparator 23B is asserted and the signal from comparator 22B is negated); and

[0161] (3) a signal from NAND gate 47B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 11111110 (which will be the case if the signal from comparator 21B is asserted) and at least one bit of the fraction in the fraction field of the operand in operand buffer 11B is "0" (which will be the case if the signal from AND gate 52B is negated);

[0162] (u) a signal from comparator 53 that is asserted if the binary-encoded value of the bits comprising the high-order fraction field of the operand in operand buffer 11A is greater than the binary-encoded value of the bits comprising the high-order fraction field of the operand in operand buffer 11B;

[0163] (v) a signal from the rounding mode store 16 that is asserted if the rounding mode is either "round toward plus infinity" or "round toward minus infinity";

[0164] (w) a signal from the rounding mode store 16 that is asserted if the rounding mode is either "round toward zero" or "round toward minus infinity";

[0165] (x) a signal from XOR gate 54 that is asserted if operand signs are different;

[0166] (y) an "overflow" signal from the multiplier core 13; and

[0167] (z) an "underflow" signal from the multiplier core 13.

[0168] In response to these signals, the exemplary multiplier decision logic table 15 generates the following:

[0169] (1) a signal that, if asserted, enables the second least-significant bit of the operand in operand buffer 11A to be toggled before being presented to the multiplier core 13;

[0170] (2) a signal that, if asserted, enables the least significant bit of the operand in operand buffer 11A to be toggled before being presented to the multiplier core 13;

[0171] (3) a signal that, if asserted, enables the second-least significant bit of the operand in operand buffer 11B to be toggled before being presented to the multiplier core 13;

[0172] (4) a signal that, if asserted, enables the least significant bit of the operand in operand buffer 11B to be toggled before being presented to the multiplier core 13;

[0173] (5) a signal that, if asserted, enables the exponent field and the high-part of the fraction of the result to be provided by the multiplier core 13, and moreover, the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the output provided by the multiplier core 13 will contribute to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the result;

[0174] (6) a signal that, if asserted, will enable the exponent field of the result to have the bit pattern 00000000;

[0175] (7) a signal that, if asserted, the will enable the exponent field of the result to have the bit pattern 11111110;

[0176] (8) a signal that, if asserted, will enable the exponent field of the result to have the bit pattern 11111111;

[0177] (9) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the high-order portion of the fraction of the operand in operand buffer 11A;

[0178] (10) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the high-order portion of the fraction of the operand in operand buffer 11B;

[0179] (11) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 0000000000000000;

[0180] (12) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 1111111111111111;

[0181] (13) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 100000000000000100 (to represent a NaN value "zero times infinity");

[0182] (14) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 100000000000000101 (to represent a NaN value "underflow times overflow");

[0183] (15) a signal that, if asserted, will enable the low-order fraction field of the operand in output buffer 11A to contribute to the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result;

[0184] (16) a signal that, if asserted, will enable the low-order fraction field of the operand in output buffer 11B to contribute to the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result; and

[0185] (17)-(21) signals that always contribute to the five least-significant bits $f_{lsb4}...f_{lsb}$ of the fraction field of the result.

[0186] The specific patterns of output signals (1) through (21) generated by the exemplary multiplier decision table logic 15 in response to patterns of input signals (a) through (z) are depicted in FIGS. 4A through 4C. Generally, in FIGS. 4A-4C, each row represents conditions of the output signals (1) through (21) that are generated by the multiplier decision table logic 15 in response to one pattern of input signals (a) through (z). In each row, the indicia to the left of the asterisk (*) represent the pattern of input signals (a) through (z) and the indicia to the right of the asterisk represent the pattern of output signals (1) through (21) with a "1" indicating that the respective input or output signal is asserted, a "0" indicating that the respective input or output signal is negated and a "-" indicating that the respective input signal may be either negated or asserted. Each row is further annotated with an indication as to the respective format 70 through 76 of the operand in the respective operand buffers 11A and 11B and the format of the result.

[0187] Referring now to FIG. 4A, a discussion of the first row of input signal values and corresponding output signal values follows.

[0188] (A) for the three input patterns to the left of the asterisk:

[0189] (i) the first pattern "--10------" indicates that signal (c) is asserted, signal (d) is negated, and signals (a), (b), and (e) through (j) may be either asserted or negated, with the pattern indicating a value in the NaN format 76 ("[NaN]");

[0190] (ii) the second pattern "--10------" indicates that signal (m) is asserted, signal (n) is negated, and signals (k), (l) and (o) through (t) may be either asserted or negated, with the pattern indicating a value in the NaN format 76 ("[NaN]"); and

[0191] (iii) the third pattern "1 -- - --" indicates that the signal (u) is asserted, indicating that the binary-encoded value of the high-order bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is greater than the binary-encoded value of the high-order bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B, and signals (v) through (z) may be either asserted or negated; and

[0192] (B) for the six output patterns to the right of the asterisk:

[0193] (i) the pattern "0000" to the immediate right of the asterisk indicates that the signals provided to XOR gates 67A, 68A, 67B and 68B, which control the toggling of the low-order bits f_{lsb+1} and f_{lsb} of the fraction fields of the operands in operand buffers 11A and 11B before being presented to multiplier core 13, are all negated;

[0194] (ii) the next "0" indicates that the signal provided to selectors 60 and 61 and AND circuit 64, is negated thereby to ensure that the output from multiplier core 13 will not contribute the result;

[0195] (iii) the next pattern "001" indicates that the signal will be asserted that will enable selector 60 to couple signals representative of the pattern 11111111 to the result bus 17, and the signals associated with the other patterns 00000000 and 11111110 will be negated;

[0196] (iv) the next pattern "100000" indicates selector 61 is enabled to couple the signals associated with the bits $f_{msb}...f_{lsb+5}$ (comprising the

high-order fraction field of the operand in buffer 11A) to the result bus 17 as the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result;

[0197] (v) the next pattern "11" indicates that the signals provided by the multiplier decision table logic 15 will provide asserted signals to both AND circuits 65 and 66, enabling both AND gates to couple signals received thereby from both operand buffers 11A and 11B to the OR circuit 63; and

[0198] (vi) the last pattern "00000" indicates that the signals provided by the multiplier decision table logic 15 to the OR circuit 63 are all negated. With such a last pattern, the OR gate will perform a bit-wise OR operation in connection with those signals and the signals provided thereto by AND circuits 65 and 66. Further, the negated signal described in (B)(ii) provides that the signals provided by AND circuit 64 are also negated in which case the signals coupled by OR circuit 63 to result bus 17 will correspond to the OR of the bits $f_{lsb+4}...f_{lsb}$ from the fraction fields of the operands in operand buffers 11A and 11B.

[0199] On the right hand side of the first row in FIG. 4A, the legend "[NaN op1 f1|f2]" indicates that the result value is in the NaN format 76 with the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result corresponding to bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A and the bits $f_{lsb+4}...f_{lsb}$ of the result corresponding to the OR of the bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operands in both operand buffers 11A and 11B. It should be noted that this corresponds to the result represented by symbol (@) in the table depicted in FIG. 3.

[0200] In the context of the above discussion, the other rows of FIG. 4A and the rows in FIGS. 4B-4C will be apparent to those skilled in the art.

[0201] As described above, the sign of the result is provided directly by the XOR gate 54.

[0202] Multiplier decision table logic 15 may be implemented by many different circuit elements that will be apparent to those skilled in the art, including, but not limited to programmable logic arrays, ASIC circuits, general memory registers, other addressable memory storage devices or a combination thereof.

[0203] One of ordinary skill in the art will recognize that other formats and bit patterns could be used to represent the floating point operand formats without

departing from the principles of the present invention. One of ordinary skill in the art will also recognize that the floating point status information contained in the operands could easily be represented by other bit combinations (not shown) without departing from the principles of the present invention. For example, more or fewer bits could be used, a subset or superset of the exemplary status bits could be used, or the most significant bits of an operand (or some other subset of bits) could be used to indicate the floating point status information, instead of the least significant bits illustrated.

[0204] It will be appreciated that a system in accordance with an embodiment of the invention can be constructed in whole or in part from special purpose hardware or a general purpose computer system, or any combination thereof. Any portion of such a system may be controlled by a suitable program. Any program may in whole or in part comprise part of or be stored on the system in a conventional manner, or it may in whole or in part be provided in to the system over a network or other mechanism for transferring information in a conventional manner. In addition, it will be appreciated that the system may be operated and/or otherwise controlled by means of information provided by an operator using operator input elements (not shown) which may be connected directly to the system or which may transfer the information to the system over a network or other mechanism for transferring information in a conventional manner.

[0205] The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that various variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. It is the object of the appended claims to cover these and such other variations and modifications as come within the true spirit and scope of the invention.

[0206] Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.

CLAIMS

WHAT IS CLAIMED IS:

1. A system for providing a floating point product, comprising:

an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and data within the second floating point operand respectively; and

a results circuit coupled to the analyzer circuit and configured to assert a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

2. The system for providing a floating point product of claim 1, wherein the analyzer circuit further comprises:

a first operand buffer configured to store the first floating point operand;

a second operand buffer configured to store the second floating point operand;

a first operand analysis circuit coupled to the first operand buffer, the first operand analysis circuit configured to generate a first characteristic signal having information relating to the first status; and

a second operand analysis circuit coupled to the second operand buffer, the second operand analysis circuit configured to generate a second characteristic signal having information relating to the second status.

3. The system for providing a floating point product of claim 2, wherein the first status and the second status are determined without regard to memory storage external to the first operand buffer and the second operand buffer.

4. The system for providing a floating point product of claim 3, wherein the memory storage external to the first operand buffer and the second operand buffer is a floating point status register.

5. The system for providing a floating point product of claim 1, wherein the results circuit further comprises:

a multiplier circuit coupled to the analyzer circuit, the multiplier circuit configured to produce the product of the first floating point operand and the second floating point operand;

a multiplier logic circuit coupled to the analyzer circuit and configured to produce the resulting status based upon the first status and the second status; and

a result assembler coupled to the multiplier circuit and the multiplier logic circuit, the result assembler configured to assert the resulting floating point operand and embed the resulting status within the resulting floating point operand.

6. The system for providing a floating point product of claim 5, wherein the multiplier logic circuit is organized according to the structure of a decision table.

7. The system for providing a floating point product of claim 1, wherein the product of the first floating point operand and the second floating point operand is identical in all cases to the product that would be produced if the two operands were first swapped.

8. The system for providing a floating point product of claim 1, wherein the first status, the second status, and the resulting status are each one of the following: an invalid operation status, an overflow status, an underflow status, a division by zero status, an infinity status, and an inexact status.

9. The system for providing a floating point product of claim 8, wherein the overflow status represents one in a group of a +OV status and a -OV status.

10. The system for providing a floating point product of claim 8, wherein the overflow status is represented as a predetermined non-infinity numerical value.

11. The system for providing a floating point product of claim 8, wherein the underflow status represents one in a group of a +UN status and a -UN status.

12. The system for providing a floating point product of claim 8, wherein the underflow status is represented as a predetermined non-zero numerical value.

13. The system for providing a floating point product of claim 8, wherein the invalid status represents a not-a-number (NaN) status due to an invalid operation.

14. The system for providing a floating point product of claim 8, wherein the infinity status represents one in a group of a positive infinity status and a negative infinity status.

15. A method for providing a floating point product, comprising:

determining a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and data within the second floating point operand respectively; and

asserting a resulting floating point operand containing the product of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

16. The method for providing a floating point product of claim 15, wherein the determining stage further comprises:

storing the first floating point operand in a first operand buffer;

storing the second floating point operand in a second operand buffer;

generating a first characteristic signal representative of the first status; and

generating a second characteristic signal representative of the second status.

17. The method for providing a floating point product of claim 16, wherein the first characteristic signal and the second characteristic signal are generated

without regard to memory storage external to the first operand buffer and the second operand buffer.

18. The method for providing a floating point product of claim 17, wherein the memory storage external to the first operand buffer and the second operand buffer is a floating point status register.

19. The method for providing a floating point product of claim 15, wherein the asserting stage further comprises:

producing the product of the first floating point operand and the second floating point operand; and

asserting the resulting floating point operand having the resulting status embedded within the resulting floating point operand.

20. The method for providing a floating point product of claim 15, wherein the product of the first floating point operand and the second floating point operand is identical in all cases to the product that would be produced if the two operands were first swapped.

21. The method for providing a floating point product of claim 15, wherein the first status, the second status, and the resulting status are each one of the following: an invalid operation status, an overflow status, an underflow status, a division by zero status, an infinity status, and an inexact status.

22. The method for providing a floating point product of claim 21, wherein the overflow status represents one in a group of a +OV status and a -OV status.

23. The method for providing a floating point product of claim 22, wherein the overflow status is represented as a predetermined non-infinity numerical value.

24. The method for providing a floating point product of claim 21, wherein the underflow status represents one in a group of a +UN status and a -UN status.

25. The method for providing a floating point product of claim 24, wherein the underflow status is represented as a predetermined non-zero numerical value.

26. The method for providing a floating point product of claim 21, wherein the invalid status represents a not-a-number (NaN) status due to an invalid operation.

27. The method for providing a floating point product of claim 21, wherein the infinity status represents one in a group of a positive infinity status and a negative infinity status.

28. The method for providing a floating point product of any one of claims 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, or 27 wherein a set of instructions are stored on a computer readable media, which when executed, perform the method.

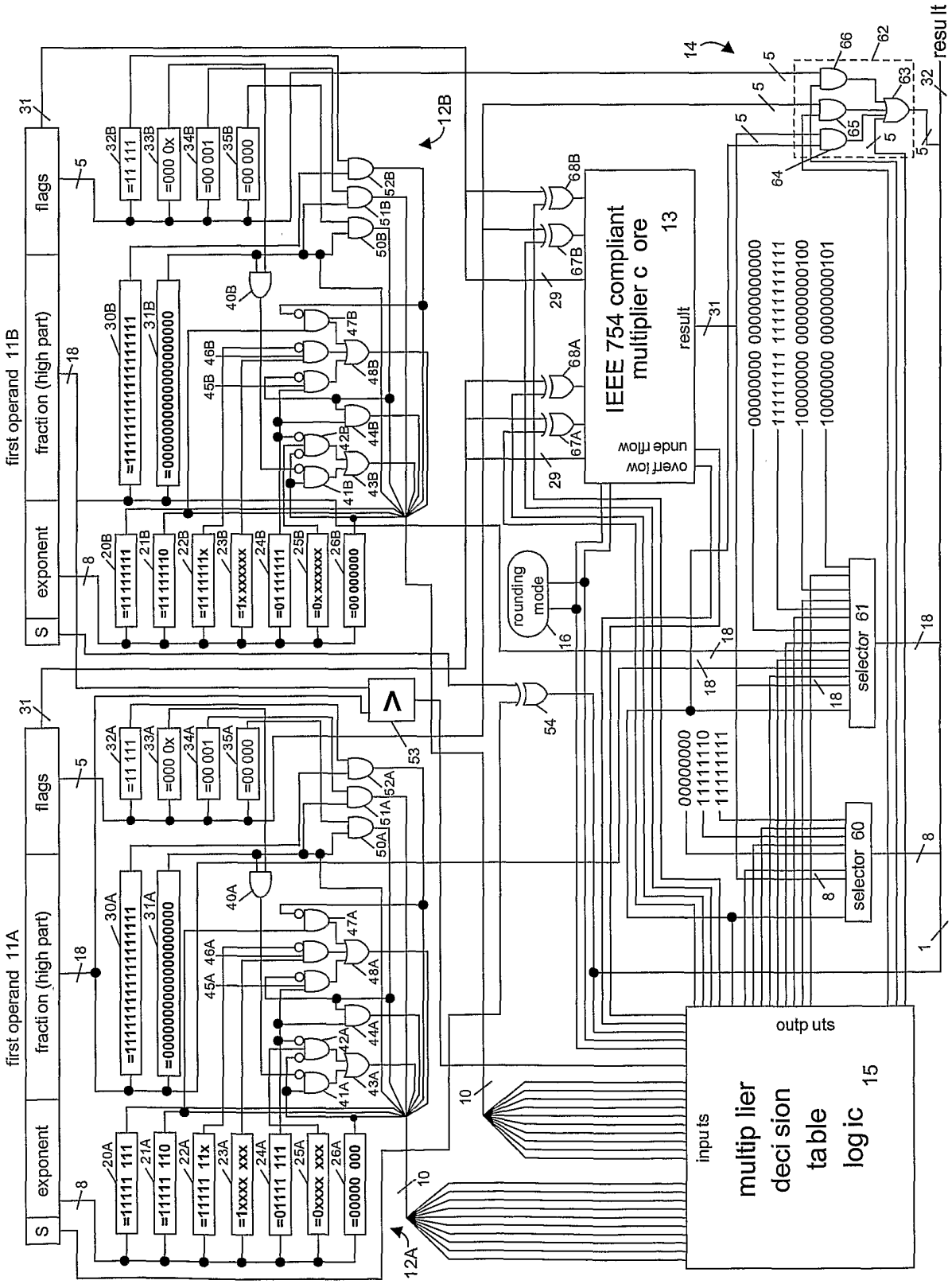

10
first operand 11B
S
exponent
fracti on (high part)
flags
31
18
8
5
12B
12A
14
IEEE 754 compliant multiplier c ore
13
result
overf low
unde rflow
rounding mode
16
multip lier deci sion ta ble log ic
15
inputs
outp uts
selector 60
selector 61
resu lt
32
00000000
11111110
11111111
00000000 00000000000
11111111 11111111111
10000000 00000000100
10000000 00000000101

Fig. 1

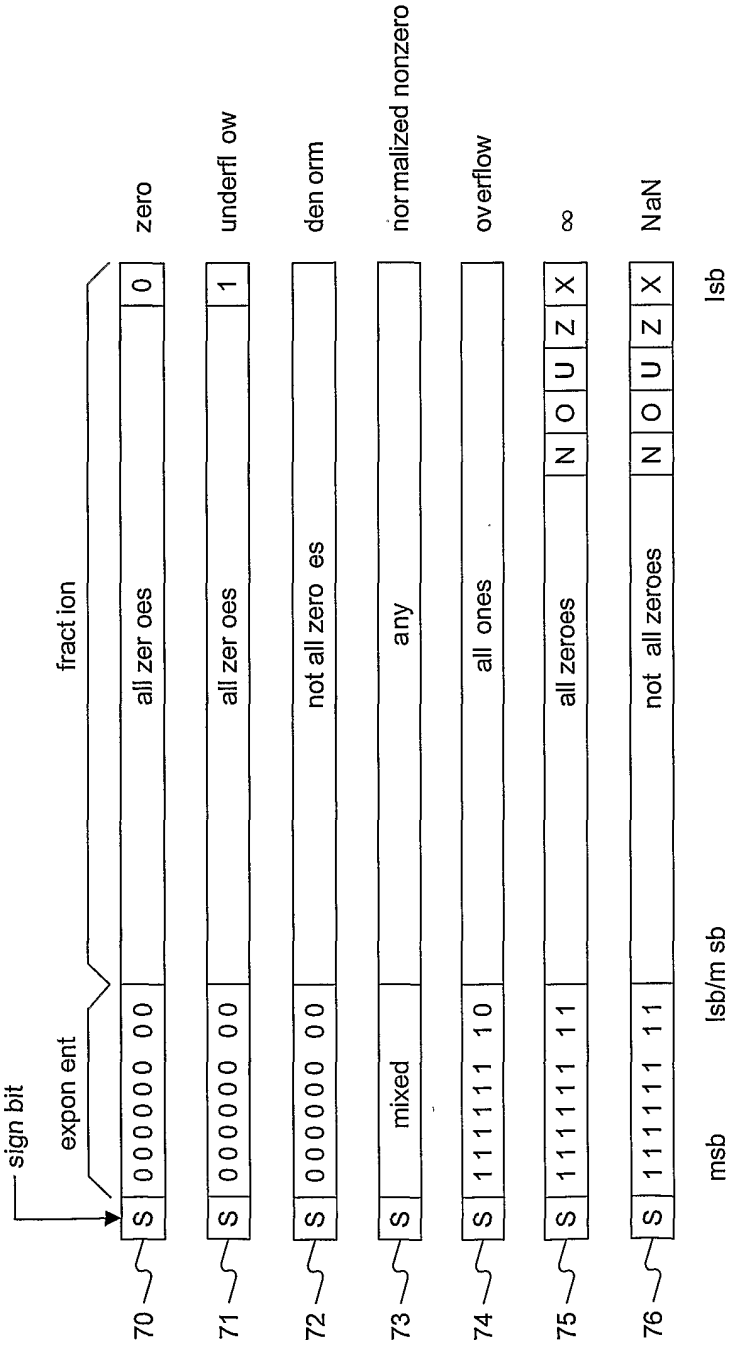

sign bit
expon ent
fract ion
msb
lsb/m sb
lsb
S 0 0 0 0 0 0 0 0 all zer oes 0
S 0 0 0 0 0 0 0 0 all zer oes 1
S 0 0 0 0 0 0 0 0 not all zero es
S mixed any
S 1 1 1 1 1 1 1 0 all ones
S 1 1 1 1 1 1 1 1 all zeroes N O U Z X
S 1 1 1 1 1 1 1 1 not all zeroes N O U Z X
70 zero
71 underfl ow
72 den orm
73 normalized nonzero
74 overflow
75 ∞
76 NaN

Fig. 2

*	-∞	-OV	-Q	-1	-S	-UN	-0	+0	+UN	+S	+1	+Q	+OV	-∞	NaN
-∞	(a)	(b)	(c)	(c)	(c)	(d)	(e)	(f)	(g)	(h)	(h)	(h)	(i)	(j)	(k)
-OV	(b)	+OV	+OV	+OV	(l)	(m)	+0	-0	(n)	(o)	-OV	-OV	-OV	(i)	(p)
-P	(c)	+OV	(q)	(q)	(q)	(r)	+0	-0	(s)	(t)	(t)	(t)	-OV	(h)	(u)
-1	(c)	+OV	(q)	(q)	(q)	+UN	+0	-0	-UN	(t)	(t)	(t)	-OV	(h)	(u)
-R	(c)	(l)	(q)	(q)	(q)	+UN	+0	-0	-UN	(t)	(t)	(t)	(v)	(h)	(u)
-UN	(d)	(m)	(r)	+UN	+UN	+UN	+0	-0	-UN	-UN	-UN	(w)	(n)	(g)	(x)
-0	(e)	+0	+0	+0	+0	+0	+0	-0	-0	-0	-0	-0	-0	(f)	(u)
+0	(f)	-0	-0	-0	-0	-0	-0	+0	+0	+0	+0	+0	+0	(e)	(u)
+UN	(g)	(n)	(s)	-UN	-UN	-UN	-0	+0	+UN	+UN	+UN	(y)	(m)	(d)	(x)
+R	(h)	(o)	(t)	(t)	(t)	-UN	-0	+0	+UN	(q)	(q)	(q)	(z)	(c)	(u)
+1	(h)	-OV	(t)	(t)	(t)	-UN	-0	+0	+UN	(q)	(q)	(q)	+OV	(c)	(u)
+P	(h)	-OV	(t)	(t)	(t)	(w)	-0	+0	(y)	(q)	(q)	(q)	+OV	(c)	(u)
+OV	(i)	-OV	-OV	-OV	(v)	(n)	-0	+0	(m)	(z)	+OV	+OV	+OV	(b)	(p)
+∞	(j)	(i)	(h)	(h)	(h)	(g)	(f)	(e)	(d)	(c)	(c)	(c)	(b)	(a)	(k)
NaN	(k)	(p)	(u)	(u)	(u)	(x)	(u)	(u)	(x)	(u)	(u)	(u)	(p)	(k)	(@)

Fig. 3

--10-----	[NaN]	1 -- - --	*	0000 0 001 100000 11	00000	[NaN op1 f1 | f2]
--10-----	[NaN]	0 -- - --	*	0000 0 001 010000 11	00000	[NaN op2 f1 | f2]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 11	00000	[NaN op1 f1 | f2]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	01001	[NaN op1 f1 | o x]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	00000	[NaN op1 f1]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	00000	[NaN op1 f1]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	00000	[NaN op1 f1]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	00101	[NaN op1 f1 | u x]
--10-----	[NaN]	- -- - --	*	0000 0 001 100000 10	00000	[NaN op1 f1]
--11-----	[Inf]	- -- - --	*	0000 0 001 010000 11	00000	[NaN op1 f1 | f2]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 11	00000	[Inf f1 | f2]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 10	00000	[Inf f1 | ox]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 10	00000	[Inf f1]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 10	00000	[Inf f1]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 10	00000	[Inf f1]
--11-----	[Inf]	- -- - --	*	0000 0 001 001000 10	00101	[Inf f1 | ux]
--11-----	[Inf]	- 0 - - --	*	0000 0 010 000010 10	10000	["0*Inf" f1 | n]
--11-----	[Inf]	- 1 0 0 --	*	0000 0 001 001000 10	00000	[Inf f1]
--11-----	[Inf]	- 1 0 1 --	*	0000 0 000 001000 00	00000	[0]
--11-----	[Inf]	- 1 1 0 --	*	0000 0 000 001000 00	00000	[0]
--11-----	[Inf]	- 1 1 1 --	*	0000 0 001 001000 10	00000	[Inf f1]
-1---1---	[NaN]	- -- - --	*	0000 0 001 010000 01	01001	[NaN op2 f1 | o x]
-1---1---	[Inf]	- -- - --	*	0000 0 001 001000 01	01001	[Inf f2 | ox]
-1---1---	[OV_]	- -- - --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[Q_]	- -- - --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[1_]	- -- - --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[S_]	- 0 - - --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[S_]	- 1 0 0 --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[S_]	- 1 0 1 --	*	0100 1 000 000000 00	00000	[HUGE *S]
-1---1---	[S_]	- 1 1 0 --	*	0100 1 000 000000 00	00000	[HUGE *S]
-1---1---	[S_]	- 1 1 1 --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[UN_]	- 0 - - --	*	0000 0 010 000001 00	11111	["UN* OV" noux]
-1---1---	[UN_]	- 1 0 0 --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[UN_]	- 1 0 1 --	*	0000 0 100 001000 00	00001	[UN]
-1---1---	[UN_]	- 1 1 0 --	*	0000 0 100 001000 00	00001	[UN]
-1---1---	[UN_]	- 1 1 1 --	*	0000 0 010 000100 00	11111	[OV]
-1---1---	[OV_]	- -- - --	*	0000 0 001 001000 00	00000	[0]

Fig. 4A

```
-------1  [P_]  --10-----  [NaN]  - --- --  *  0000 0 001 010000 01  00000  [NaN op2  f2]
-------1  [P_]  --11-----  [Inf]  - --- --  *  0000 0 001 001000 01  00000  [Inf f2]
-------1  [P_]  -1---1---  [OV_]  - --- --  *  0000 0 010 000100 11  11111  [OV]
-------1  [P_]  -------1--  [Q_]  - -- -0-  *  0000 1 000 000000 00  00000  [P*Q]
-------1  [P_]  -------1--  [Q_]  - -- -1-  *  0000 0 010 000100 00  11111  [OV]
-------1  [P_]  ------1---  [1_]  - --- --  *  0000 1 000 000000 00  00000  [P*1]
-------1  [P_]  ------1---  [S_]  - --- --  *  0000 1 000 000000 00  00000  [P*S]
-------1  [P_]  1---1-----  [UN_]  -0- ---  *  0000 0 100 001000 10  00001  [UN]
-------1  [P_]  1---1-----  [UN_]  -10 0--  *  0011 1 000 000000 00  00000  [P*TINY]
-------1  [P_]  1---1-----  [UN_]  -10 1--  *  0000 0 100 001000 00  00001  [UN]
-------1  [P_]  1---1-----  [UN_]  -11 0--  *  0000 0 100 001000 00  00001  [UN]
-------1  [P_]  1---1-----  [UN_]  -11 1--  *  0011 1 000 000000 00  00000  [P*TINY]
-------1  [P_]  1---1-----  [1_]  - --- --  *  0000 0 100 001000 00  00000  [0]

------1-  [1_]  --10-----  [NaN]  - --- --  *  0000 0 001 010000 01  00000  [NaN op2  f2]
------1-  [1_]  --11-----  [Inf]  - --- --  *  0000 0 001 001000 01  00000  [Inf f2]
------1-  [1_]  -1---1---  [OV_]  - --- --  *  0000 0 010 000100 00  11111  [OV]
------1-  [1_]  -------1--  [Q_]  - --- --  *  0000 1 000 000000 00  00000  [1*Q]
------1-  [1_]  ------1---  [1_]  - --- --  *  0000 1 000 000000 00  00000  [1*1]
------1-  [1_]  -------1--  [S_]  - --- --  *  0000 1 000 000000 00  00000  [1*S]
------1-  [1_]  ------1---  [UN_]  - --- --  *  0000 0 100 001000 10  00001  [UN]
------1-  [1_]  ------1---  [0_]  - --- --  *  0000 0 100 001000 00  00000  [0]

-----1--  [R_]  --10-----  [NaN]  - --- --  *  0000 0 001 010000 01  00000  [NaN op2  f2]
-----1--  [R_]  --11-----  [Inf]  - --- --  *  0000 0 001 001000 01  00000  [Inf f2]
-----1--  [R_]  -1---1---  [OV_]  -0- ---  *  0000 0 010 000100 00  11111  [OV]
-----1--  [R_]  -1---1---  [OV_]  -10 0--  *  0000 0 010 000100 00  11111  [OV]
-----1--  [R_]  -1---1---  [OV_]  -10 1--  *  0001 1 000 000000 00  00000  [R*HUGE]
-----1--  [R_]  -1---1---  [OV_]  -11 0--  *  0001 1 000 000000 00  00000  [R*HUGE]
-----1--  [R_]  -1---1---  [OV_]  -11 1--  *  0000 0 010 000100 00  11111  [OV]
-----1--  [R_]  ------1---  [Q_]  - --- --  *  0000 1 000 000000 00  00000  [R*Q]
-----1--  [R_]  ------1---  [1_]  - --- --  *  0000 1 000 000000 00  00000  [R*1]
-----1--  [R_]  ------1---  [S_]  - -- -0-  *  0000 1 000 000000 00  00000  [R*S]
-----1--  [R_]  ------1---  [S_]  - --- -1-  *  0000 0 100 001000 00  00001  [UN]
-----1--  [R_]  1---1-----  [UN_]  - --- --  *  0000 0 100 001000 00  00001  [UN]
-----1--  [R_]  1---1-----  [0_]  - --- --  *  0000 0 100 001000 00  00000  [0]
```

Fig. 4B

1---1----	[UN_]	--10------	[NaN]	- -- - --	*	0000 0 001 010000 01	00101	[NaN op2 f1 | u x]
1---1----	[UN_]	--11------	[Inf]	- -- - --	*	0000 0 001 001000 01	00101	[Inf f2 | ux]
1---1----	[UN_]	-1----1---	[OV_]	- 0- - --	*	0000 0 010 000001 00	11101	["UN* OV" noux]
1---1----	[UN_]	-1----1---	[OV_]	- 1 0 0 --	*	0000 0 010 000100 00	11111	[OV]
1---1----	[UN_]	-1----1---	[OV_]	- 1 0 1 --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	-1----1---	[OV_]	- 1 1 0 --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	-1----1---	[OV_]	- 1 1 1 --	*	0000 0 010 000100 00	11111	[OV]
1---1----	[UN_]	--------1	[Q__]	- 0- - --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	--------1	[Q__]	- 1 0 0 --	*	1100 1 000 000000 00	00000	[TINY*Q]
1---1----	[UN_]	--------1	[Q__]	- 1 0 1 --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	--------1	[Q__]	- 1 1 0 --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	--------1	[Q__]	- 1 1 1 --	*	1100 1 000 000000 00	00000	[TINY*Q]
1---1----	[UN_]	-------1-	[1__]	- -- - --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	-------1--	[S__]	- -- - --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	1----1----	[UN_]	- -- - --	*	0000 0 100 001000 00	00001	[UN]
1---1----	[UN_]	1---1-----	[0__]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	--10------	[NaN]	- -- - --	*	0000 0 001 010000 01	00000	[NaN op1 f2]
1---1-----	[0__]	--11------	[Inf]	- 0- - --	*	0000 0 001 000010 01	10000	["0*Inf" f2 | n]
1---1-----	[0__]	--11------	[Inf]	- 1 0 0 --	*	0000 0 001 001000 01	00000	[Inf f2]
1---1-----	[0__]	--11------	[Inf]	- 1 0 1 --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	--11------	[Inf]	- 1 1 0 --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	--11------	[Inf]	- 1 1 1 --	*	0000 0 001 001000 01	00000	[Inf f2]
1---1-----	[0__]	-1----1---	[OV_]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	--------1	[Q__]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	-------1-	[1__]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	-------1--	[S__]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	1----1----	[UN_]	- -- - --	*	0000 0 100 001000 00	00000	[0]
1---1-----	[0__]	1---1-----	[0__]	- -- - --	*	0000 0 100 001000 00	00000	[0]

Fig. 4C

INTERNATIONAL SEARCH REPORT

	ational Application No
	PCT/US 02/16016

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F7/52

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
WPI Data, EPO-Internal, INSPEC, IBM-TDB, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 748 516 A (GODDARD MICHAEL D ET AL) 5 May 1998 (1998-05-05) abstract; figures 2-5 column 4, line 39 -column 5, line 17 column 8, line 16 -column 14, line 51 tables 1-3	1-28
X	US 5 126 963 A (FUKASAWA HISAKO) 30 June 1992 (1992-06-30) abstract; figure 3 column 5, line 54 -column 6, line 37	1-5, 15-19

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 September 2002	01/10/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Pfab, S

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 02/16016

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5748516 A	05-05-1998	WO 9712316 A1	03-04-1997
US 5126963 A	30-06-1992	JP 2055530 C	23-05-1996
		JP 2306337 A	19-12-1990
		JP 7092739 B	09-10-1995