Pipelined Packet-Forwarding Floating Point: I. Foundations and a Rounder *

David W. Matula
Dept. of Computer Science and Engineering
Southern Methodist University
Dallas, Texas
matula@seas.smu.edu

Asger Munk Nielsen
Dept. of Mathematics and Computer Science
Odense University, Denmark
asger@imada.ou.dk

Abstract

This paper presents the foundations for a packet forwarding floating point format and the design of a rounder ensuring compatibility between packet forwarding format and the standard binary IEEE 754 floating point format. The packet forwarding format and related addition and multiplication algorithms described in this series propose a new ALU pipeline paradigm for handling data hazards in pipelined floating point operations. The execution phases for the adder and multiplier packet forwarding pipelines are illustrated by a proposed implementation having four stages. The latter two stages in each pipeline employ the rounder described herein.

The stages of the execution phase are intended to map to logic designs with only some fifteen logic levels per stage, allowing stages to be mapped to reasonably short cycles. The packet forwarding format provides for input and output in packet format with only two cycle effective latency between cooperating adder and multiplier pipelines. The designs we propose cut the effective latency in half and reduce the stall cycles by a factor of three compared to conventional forwarding pipelines processing data dependent operations. The speedup is realized with preservation of IEEE 754 binary floating point compatibility.

1 Introduction and Summary

The foundations for a temporally partitioned floating point representation $a = (-1)^s 2^e (f + c2^{p-1})$ are developed in support of a packet forwarding pipeline paradigm. The pipeline is designed to accept the principal part packet f and carry-round packet c of the significand as inputs in successive execution stages of the pipeline, and similarly to generate the output significand packets f' and c' in successive stages. Herein we utilize f and c to denote digit string packets and $f = ||f||$ and $c = ||c||$ to denote their values. This provides for discussing the redundancy in the digit string representation. Without loss of generality, s and e are used for values and packets as they are assumed uniquely encoded.

The packet forwarding floating point format and cooperating packet forwarding multiplier and adder pipelines constitute a proposed new ALU paradigm for handling data hazards in pipelined floating point implementations.

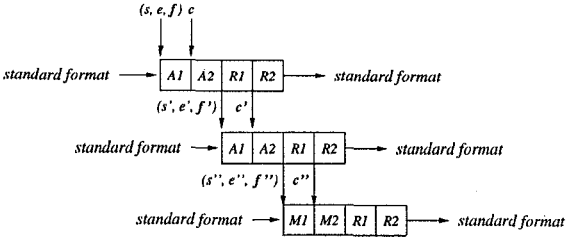

Figure 1. Cooperating packet forwarding pipeline operation.

The cooperating adder and multiplier pipes illustrated in Fig.1, employ a four stage execution sequence with the latter two stages of each being the rounder phase. Both pipelines accept one operand in a standard format at the start of stage one. Both pipelines accept the packet forwarded operand in packets at the start of stages one and two, and output the result in packets after stages two and three. The output of stage four is the identical valued result in the standard binary floating point IEEE 754 format for retirement to a register. The example of Figure 1 reduces effective latency from four to two stages (cycles). Recent research [9] has documented the high frequency of dependent multiplies and additions in floating point intensive applications. Reducing the effective latency is the best way to reduce stalls and allow improved super scaler performance.

*This work was supported in part by a grant from Cyrix Corporation and by the Texas Advanced Technology Program Grant 003613013, and grant no. 5.21.08.02 from the Danish Research Council.

In the papers of this series we investigate stages of rounder, adder [2] and multiplier pipeline designs where each stage can be implemented in a single short cycle of no more than fifteen logic levels. The two stage (cycle) rounder is common to both adder and multiplier pipelines.

Rounding plays a central role in the IEEE 754 standard for floating point arithmetic [1]. Standardized rounding procedures can make numerical software predictable as well as capable of providing accurate and reliable approximations and bounds. Although relatively simple in conception, floating point rounding is surprisingly involved. It is not uncommon that rounding accounts for half the latency in pipelined floating point units for addition and multiplication. Our proposed rounding algorithm deviates from previous algorithms [4, 5, 6], by accepting redundant input *and* producing redundant as well as standard rounded output. Our algorithm employs novel circuitry for computing the normalization and sticky digits needed for rounding. The procedure allows that the principal part packet can be forwarded to another functional unit before rounding, with a follow-up carry-round packet forwarded after just one rounding operation stage. The rounding procedure in general has a latency of two stages to produce the IEEE 754 standard format output.

In Section 2 we explore floating point factorizations to provide the foundation for alternative compatible floating point representations. Our principal result in Section 2 is the establishment of prenormalized floating point factorizations utilizing borrow-save and carry-save encodings, which can host standard binary IEEE 754 compatible numeric results.

Section 3 defines standard and packet forwarding operand formats of double extended precision, and illustrates the packet forwarding paradigm with these particular operands.

Section 4 provides a high level design of the two cycle rounder. The main result in Section 4 , is an efficient algorithm and circuitry for determining the positive, negative or zero sense, termed the *signed sticky digit*, of a borrow-save encoded number. This circuit is the key to determining both normalization and rounding information for the floating point results at less cost than the usual 2-1 compression initiating normalization and rounding computations.

The logic for determining the carry-round packet coercing compatible output values for the packet forwarding and IEEE 754 standard outputs is described in Section 5. The principal result there is the formulation of the rounding table, giving the carry-round packet as a function of the guard, round, normalization and sticky digits and the rounding mode. The remainder of Section 5 describes the second rounding stage (cycle). That stage is divorced from the packet forwarding computation and is the only stage of the full adder (or multiplier) pipeline where a time and area consuming 2-1 compressor is needed.

2 Floating Point Factorization, Normalization and Number Systems

A *binary number* is a rational with a binary rational factorization $a = i2^j$, with i, j integers. A *binary floating point number* is a binary number with a *floating point factorization (fp-factorization)*:

$$a = (-1)^s 2^e \, \|d_m d_{m-1} \cdots d_l\| , \tag{1}$$

comprising sign, scale and significand factors where

- $s \in \{0, 1\}$ is a *sign bit* determining the *sign factor* $(-1)^s$,
- e is an integer *exponent* determining the *scale factor* 2^e,
- $\boldsymbol{f} = d_m d_{m-1} \cdots d_l$ is a digit string, termed the *significand*, determining a binary *significand factor*

$$f = \|\boldsymbol{f}\| = \|d_m d_{m-1} \cdots d_l\| = \sum_{l}^{m} d_i 2^i \tag{2}$$

according to a prescribed fixed point format.

Prescribing the significand format determines the fixed point range and granularity of the significand factor. A significand factor f is termed *normalized* when it is in the half open binade $1 \leq f < 2$. More particularly,

$$f \in F(p) = \{1 + \frac{i}{2^{p-1}} \mid 0 \leq i \leq 2^{p-1} - 1\} \tag{3}$$

is a *p-bit factor* with *precision p*, *range* $[1, 2 - 2^{-(p-1)}]$ and *granularity* $2^{-(p-1)}$.

Regarding the significand digit string format itself, a *standard p-bit significand* $\boldsymbol{f} = b_0.b_1 b_2 \cdots b_{p-1}$, has digits (bits) $b_i \in \{0, 1\}$ for $0 \leq i \leq p - 1$, with $\boldsymbol{f}$ *normalized* when $b_0 = 1$ and *denormalized* when $b_0 = 0$.

Observation 2.1 *Every binary number $a \neq 0$ has a unique fp-factorization with a normalized significand factor (the normalized fp-factorization):*

$$a = (-1)^s 2^e f \, , \, 1 \leq f < 2 \tag{4}$$

where $e = \lfloor log_2 |a| \rfloor$. Every binary number $a = i2^j$ with $1 \leq |i| \leq 2^p - 1$ has a unique fp-factorization with a p-bit factor f (the standard p-bit fp-factorization):

$$a = (-1)^s 2^e f = (-1)^s 2^e \, \|1.b_1 b_2 \cdots b_{p-1}\| , \tag{5}$$

where the p-bit factor f is given by a standard normalized p-bit significand $\boldsymbol{f} = 1.b_1 b_2 \cdots b_{p-1}$, and each bit $b_i \in \{0, 1\}$ is uniquely determined.

For any given exponent e and precision $p \geq 2$, $FP(e,p)$ denotes the set of binary floating point numbers of the positive $[2^e, 2^{e+1})$ and negative $(-2^{e+1}, -2^e]$ binades having a standard p-bit fp-factorization,

$$FP(e,p) = \{(-1)^s 2^e \| 1.b_1 b_2 \cdots b_{p-1} \| \mid s, b_i \in \{0,1\}\}. \quad (6)$$

Equivalently, $FP(e,p)$ is defined by the binary rational factorizations,

$$FP(e,p) = \left\{ i 2^{e+1-p} \mid 2^{p-1} \leq |i| < 2^p - 1 \right\}. \quad (7)$$

It is emphasized that a number of the set $FP(e,p)$ may be termed a *standard binary floating point number of precision p and exponent e* independent of the actual representation of the number. This is particularly useful for discussing compatibility when the floating point number may be given in an unnormalized format that might include carry-save or borrow-save digit encoding.

It is well known that addition (same sign) and multiplication of non zero floating point numbers produce a result with a floating point factorization $r = (-1)^s 2^e f$ with $1 \leq f < 4$ where e can be determined from the exponents of the unique normalized floating point factorizations of the arguments. A similar result holds for subtraction when the argument exponents differ by at least two. Such resulting factorizations can describe the internal output of an adder or multiplier input to a rounder. The result may be passed still having a redundant digit significand. The factorization of the subsequent rounded result $r' = (-1)^s 2^e f'$ (without exponent adjustment) can be described by allowing $1 \leq f' \leq 4$. Our approach is to extend and formalize these existing "two-binade limited" internal formats to characterize the packet forwarding format. Our purpose is to support fused arithmetic operations with output of a rounded addition or multiplication immediately input to a consuming successor addition or multiplication. We shall coerce compatibility without need for intermediate compression to the standard unique factorization format.

A significand factor f is termed *prenormalized* when it is in the two binade closed interval $1 \leq f \leq 4$.

Observation 2.2 *Let $e = \lfloor log_2 |a| \rfloor$ for any binary number $a \neq 0$. Then a has two or three fp-factorizations with prenormalized significand factors (prenormalized fp-factorizations):*

- *for $e = log_2 |a|$, there are 3 fp-factorizations:*

$$a = (-1)^s 2^e \cdot 1 = (-1)^s 2^{e-1} \cdot 2 = (-1)^s 2^{e-2} \cdot 4,$$

- *for $e < log_2 |a|$, there are 2 prenormalized fp-factorizations:*

$$a = \begin{cases} (-1)^s 2^e f & \text{with } 1 < f < 2, \\ (-1)^s 2^{e-1} f & \text{with } 2 < f < 4. \end{cases}$$

The set of *p-digit prenormalized* significand factors is given by

$$F_{pre}(p) = \left\{ f \mid f = 1 + \frac{i}{2^{p-1}} \text{ or } f = 2 + \frac{i}{2^{p-2}} \right\}, \quad (8)$$

for $0 \leq i \leq 2^{p-1}$. Note that the p-digit prenormalized significands have granularity $1/2^{p-1}$ in the binade $[1,2]$ doubling to $2/2^{p-1}$ in the binade $[2,4]$. This step in granularity allows that the set of binary numbers with prenormalized fp-factorizations with exponent e and a p-digit prenormalized significand can be readily associated with standard binary floating point numbers of precision p and the appropriate exponents.

Observation 2.3 *For any e and $p \geq 2$,*

$$\{a \mid a = (-1)^s 2^e f, s \in \{0,1\}, f \in F_{pre}(p)\} \quad (9)$$
$$= FP(e,p) \cup FP(e+1,p) \cup \left\{-2^{e+2}, 2^{e+2}\right\}. \quad (10)$$

The specification of the set of p-digit prenormalized significands $F_{pre}(p)$ is introduced to allow fp-factorizations where the significand can be represented employing conventional carry-save or borrow-save representations, with only the final digit needing revision.

Lemma 2.4 *The p-digit prenormalized significand factors can be obtained from redundant binary significands with a restricted low order digit as follows:*

- *Borrow-save (or signed bit) form:*

$$F_{pre}(p) = \left\{ \left\| 1 b_0 . b_1 b_2 \cdots b_{p-2} b^*_{p-1} \right\| \right\}$$

with $b_0 \in \{0,1\}$, $b_i \in \{-1,0,1\}$ for $1 \leq i \leq p-2$ and

$$b^*_{p-1} \in \begin{cases} \{-2,0,2\} & \text{for } \|1 b_0 . b_1 b_2 \cdots b_{p-2}\| > 2 \\ \{-2,-1,0,2\} & \text{for } \|1 b_0 . b_1 b_2 \cdots b_{p-2}\| = 2 \\ \{-2,-1,0,1,2\} & \text{for } \|1 b_0 . b_1 b_2 \cdots b_{p-2}\| < 2 \end{cases}$$

- *Carry-save form:*

$$F_{pre}(p) = \left\{ \left\| d_0 . d_1 d_2 \cdots d_{p-2} d^*_{p-1} \right\| \right\}$$

with $d_0 \in \{1,2\}$, $d_i \in \{0,1,2\}$ for $1 \leq i \leq p-2$ and

$$d^*_{p-1} \in \begin{cases} \{0,2,4\} & \text{for } \|d_0 . d_1 d_2 \cdots d_{p-2}\| > 2 \\ \{0,1,2,4\} & \text{for } \|d_0 . d_1 d_2 \cdots d_{p-2}\| = 2 \\ \{0,1,2,3,4\} & \text{for } \|d_0 . d_1 d_2 \cdots d_{p-2}\| < 2 \end{cases}$$

Proof: Consider the borrow-save form and let

$$f_p = \|1 b_0 . b_1 b_2 \cdots b_{p-2}\|$$

denote the p-signed bit *principal part* contribution to the significand factor, where $b_0 \in \{0,1\}$, $b_i \in \{-1,0,1\}$ for $1 \leq i \leq p-2$. The range of values for f_p is the sequence $1+1\cdot 2^{-(p-2)}, 1+2\cdot 2^{-(p-2)}, \ldots, 1+(2^{p-1}+2^{p-2}-1)2^{-(p-2)} = 4-2^{-(p-2)}$, with granularity $2^{-(p-2)}$ throughout. For $f_p > 2$, the values of $f_p + b^*_{p-1}2^{-(p-1)}$ with $b^*_{p-1} \in \{-2,0,2\}$ augments the sequence of f_p values only by $4 - 2^{-(p-2)} + 2\cdot 2^{-(p-1)} = 4$, yielding the required values of F_{pre} over the binade $[2,4]$. For $f_p < 2$, the values of $f_p + b^*_{p-1}2^{-(p-1)}$ with $b^*_{p-1} \in \{-2,-1,0,1,2\}$ effectively extends the precision by one position to the right compensating for the loss of the leading digit and covering the required values of F_{pre} over the binade $[1,2]$ with granularity $2^{-(p-1)}$ throughout. For $f_p = 2$, the values of $f_p + b^*_{p-1}2^{-(p-1)}$ with $b^*_{p-1} \in \{-2,-1,0,1,2\}$ cover precisely the values of F_{pre} within the interval $[2-2^{-(p-2)}, 2+2^{(p-2)}]$ where the granularity of spacing changes as the binade boundary is crossed. The argument is similar for the carry-save form. □

Let the *signed sticky digit* $s(\boldsymbol{f}) \in \{-1,0,1\}$ for a borrow-save digit string $\boldsymbol{f} = b_0 b_1 \cdots b_n$, $b_i \in \{-1,0,1\}$ be given by

$$S(b_0 b_1 \cdots b_n) = signum(||b_0 b_1 \cdots b_n||). \qquad (12)$$

Corollary 2.5 *For the borrow-save prenormalized significand $1b_0.b_1b_2\cdots b_{p-2}b^*_{p-1}$ with b^*_{p-1} having an extended range as specified in Lemma 2.4, determination of which extended range applies for b^*_{p-1}, is equivalent to finding the signed sticky digit $S(b_0.b_1b_2\cdots b_{p-2})$.*

In Section 4 we show that the signed sticky digit can be computed more efficiently than by 2-1 compression of the string $b_0.b_1b_2\cdots b_{p-2}$ to a 2's complement or standard signed binary string.

A binary normalized *floating point number system* with *exponent range* $[E_{min}, E_{max}]$ and precision $p \geq 2$ is denoted:

$$FP(E_{min}, E_{max}, p) = \cup_{e=E_{min}}^{E_{max}} FP(e,p). \qquad (13)$$

The specification employed in (6) and (13) for $FP(e,p)$ follows the IEEE standard 754 [1] for normalized floating point numeric values. Thus the normalized binary IEEE 754 values are given for the various precision levels by: $FP(-126, 127, 24)$ for single, $FP(-1022, 1023, 53)$ for double and $FP(E_{min} \leq -16382, E_{max} \geq 16383, p \geq 64)$ for double extended precision.

In view of Observation 2.3, compatibility with the normalized IEEE754 values may be achieved using prenormalized significands. E.g. for double precision:

$$FP(-1022, 1023, 53) = \{(-1)^s 2^e f\},$$

where $s \in \{0,1\}$, $-1022 \leq e \leq 1022$ and $f \in F_{pre}(53)$, $|f| \neq 2^{1024}$.

The IEEE 754 standard also prescribes certain denormal values. In practice, implementations such as the various $x86$ architectures use normalized number systems of greater exponent range and treat denormals, underflow and overflow by appropriate special logic. Similar techniques may be used based on prenormalized significands. It follows that borrow-save and carry-save significands may be employed for compatible IEEE standard arithmetic results without conversion to standard binary format providing the final digit rules of Lemma 2.4 are observed.

3 Packet Forwarding Pipelines

The packet forwarding pipeline paradigm we propose accepts one input in a "standard" binary format derived from the factorization (5) and the IEEE 754 standard [1]. The other operand is in a "packet forwarding" format using a borrow-save p-digit prenormalized significand defined with reference to Lemma 2.4.

Specifically a *standard* double extended IEEE 754 *floating point operand* with unique factorization

$$a = (-1)^{s_1} 2^{e_1} f_1 \qquad (14)$$

as specified in [1], consists of (see Fig. 2 (a)):

- s_1 is a sign bit,
- e_1 is encoded in an exponent field of 15 or more bits,
- $f_1 = 1.a_1a_2\cdots a_{p-1}$ with $a_i \in \{0,1\}$ is a normalized binary significand with at least 64 bits of precision (henceforth $p = 64$ is employed).

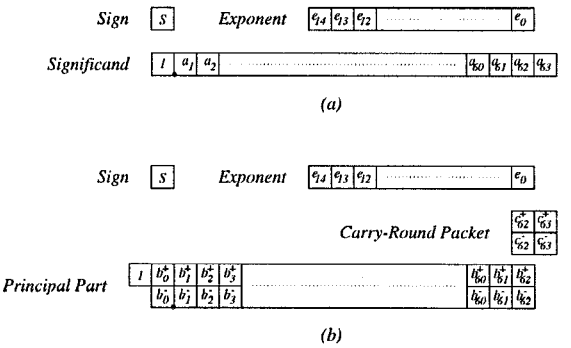

Figure 2. Floating point operand formats. (a) Standard IEEE 754 operand format. (b) Packet forwarding operand format.

Our *packet forwarding floating point operands* (see Fig. 2 (b)) have a non unique factorization

$$b = (-1)^{s_2} 2^{e_2} (f_2 + c2^{-63}) \qquad (15)$$

with the same sign and exponent format as a standard double extended precision operand. The significand $(f_2 + c2^{-63})$ in (15) is kept in a temporally staggered format by partitioning it into two packets (see Lemma 2.4):

- $\boldsymbol{f_2} = 1b_0.b_1b_2 \cdots b_{62}$ with $b_i \in \{-1, 0, 1\}$, is a 64 digit borrow-save encoded digit string termed the *principal part packet.*
- $\boldsymbol{c} = c_{62}c_{63}$, is given by a two digit borrow-save digit string with $c = ||c_{62}c_{63}|| \in \{-2, -1, 0, 1, 2\}$ termed the *carry-round packet.*

The principal part is a redundant leading part of the IEEE compliant result of a floating point operation, that is *prenormalized* such that it belongs to the two binade range $(1, 4)$, in particular $[1 + 2^{-62}, 4 - 2^{-62}]$. The carry-round term $c2^{-63}$ has the range $[-2^{-62}, 2^{62}]$, with the significand range then being $[1, 4]$.

The packet format has the packets $(s, e, \boldsymbol{f}, \boldsymbol{c})$ ordered for input to a multiple execution stage pipeline in sequential order with output of packets for forwarding occurring at sequential stages. The *packet forwarding pipeline paradigm* has one standard operand input and one packet forwarding operand input. Since standard double extended operands, by wired recoding [3], constitute a subclass of the more general packet forwarding operands, both inputs can be standard operands.

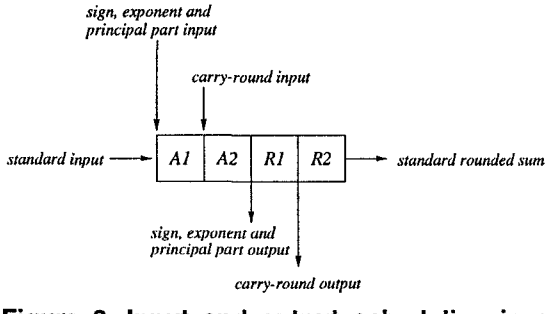

Figure 3. Input and output scheduling in a packet forwarding pipeline.

In Figure 3 we illustrate a four stage (cycle) execution pipeline with the first two stages being the operation and the last two the rounding. The pipeline accepts one operand in standard format $(s_1, e_1, \boldsymbol{f_1})$ and the first three packets $(s_2, e_2, \boldsymbol{f_2})$ of the second operand utilizing the packet format, at the start of cycle one. The carry round packet $(\boldsymbol{c_2})$ is input at the start of the second cycle. After the second cycle the three leading packets $(s_3, e_3, \boldsymbol{f_3})$ of the output in packet format are available for forwarding. After the third cycle the carry round packet (c_3) is forwarded, and in the fourth cycle the result in standard format $(s_4, e_4, \boldsymbol{f_4})$ is available for retirement to a register. The rounder must assure compatibility in coercing the two factorizations to have the same value, i.e.

$$(-1)^{s_3} 2^{e_3} (f_3 + c_3 2^{-63}) = (-1)^{s_4} 2^{e_4} f_4. \qquad (16)$$

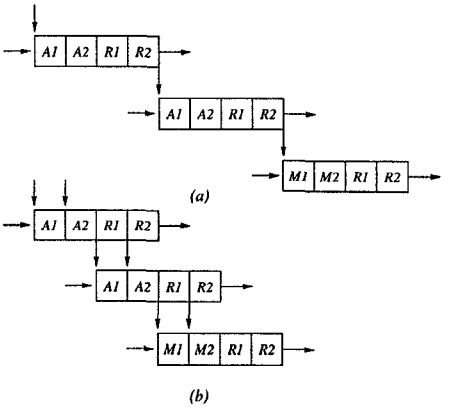

Figure 4. (a) Conventional forwarding operation in the case of data hazards with 4 cycle latency and 3 stalls per pipe, and (b) comparable packet forwarding operation with 2 cycle effective latency and one stall per pipe.

In Figure 4 (a) we illustrate execution of successive dependent floating point operations exhibiting conventional pipeline *data hazards.* With the pipelined packet forwarding paradigm in Figure 4 (b), a dependent instruction can be issued after 2 clock cycles rather than 4, thus the latency is cut in half and the number of stall cycles is reduced by a factor of three.

4 Signed Sticky Digits and a Rounding Unit

In the following we discuss the rounding algorithm, assuming that the input to the rounder comes from a multiplier compression tree in redundant form or from the addition algorithm in our accompanying paper [2].

The significand data-path of the rounding unit is depicted in Fig. 5. A 129 digit borrow-save uncompressed result is taken as the input. The leading 64 bits are termed the *principal part packet.* This packet can be forwarded to a cooperating pipeline or fed back as the forwarding input of the same pipeline for a successive operation.

In the first stage of the rounder two signed sticky digits are computed based on upper and lower parts of the full

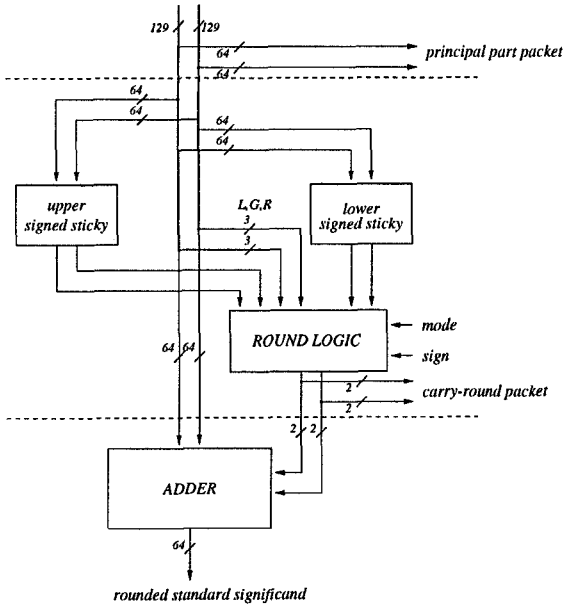

Figure 5. Significand data-path for the two cycle rounder (the dashed lines represents pipeline cuts).

Signed Sticky	Interval of fraction	Encoding (s, m)
-1	$(-1, 0)$	$(1, 1)$
0	$[0, 0]$	$(0, 0)$ or $(1, 0)$
1	$(0, 1)$	$(0, 1)$

Table 1. Interpretation of a signed sticky digit

double width input. These digits are used along with other information needed for rounding, as entries for a table that determines the *carry-round packet* that is forwarded. The final stage computes the standard format result.

Traditionally the *sticky bit* is computed on the lower part of the compressed output immediately to the right of the round bit, and it is defined as zero if the fraction is equal to zero, otherwise it is one. Here we shall compute upper and lower sticky digits each with range $\{-1, 0, 1\}$. The lower sticky replaces the ordinary sticky bit, and the high order sticky is needed for normalization according to Lemma 2.4 and Corollary 2.5. The interpretation and encoding of a signed sticky digit is given in Table 1. Note that the sticky is encoded in a sign-magnitude format for convenience of computation by the following algorithm.

In what follows we describe how to compute the (s, m) pair from a $2n$ digit borrow-save encoded significand:

$$x = 0.x_1 x_2 \ldots x_{2n}, \tag{17}$$

$x_i = x_i^+ - x_i^- \in \{-1, 0, 1\}$ and $x_i^+, x_i^- \in \{0, 1\}$, by a divide and conquer algorithm. Let us assume that we have computed the signed sticky of the upper and lower halves of x, and let them be denoted by (s_h, m_h) respectively (s_l, m_l). Then the signed sticky of the full $2n$ digit fraction is given by

$$m = m_h \vee m_l, \text{ and } s = \begin{cases} s_h & \text{if } m_h = 1 \\ s_l & \text{otherwise} \end{cases}. \tag{18}$$

For the base case of a single borrow-save encoded digit

$$s_i = x_i^- \text{ and } m_i = x_i^+ \oplus x_i^-. \tag{19}$$

To implement (18) and (19) we derive a simple muxing circuit (see Fig. 6 and [8]).

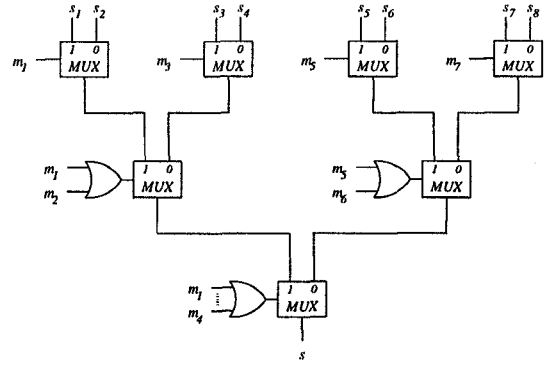

Figure 6. Signed sticky computation.

Since the mux selection signals can be set up early, the sticky information can flow through the tree at high speed. In this manner the signed sticky digit computation is considerably faster that if done by 2-1 compression of the redundant number followed by a conventional sticky computation, i.e. a tree of **and** gates. Since only the high order signed sticky is needed to determine the normalization digit, the 2-1 compression of the leading digits always employed to determine standard output is avoided when only the compatible packet forwarding format is needed.

5 Rounding Logic

The first stage of the rounder computes the *carry-round packet*. The rounding position (i.e. the position where a unit is to be added or subtracted as decided by the rounding logic) is defined as the position of the *guard digit* (G), i.e. position 63, if the unrounded full precision result is strictly less than two (see Fig. 7 (a)). In the case of significand overflow ($2 \leq$ full precision result < 4) the rounding position is one place to the left at the position of the L bit, i.e. position 62, (see Fig. 7 (b)).

Since the upper (S_u) and lower (S_l) sticky digits are computed based on the principle part from positions 0 down

to 63 and respectively from positions 65 and down (see Fig. 7 (c)), the range of the significand factor and the rounding position can be deduced from the sticky digits and the round digit R at position 64, as shown in Table 2.

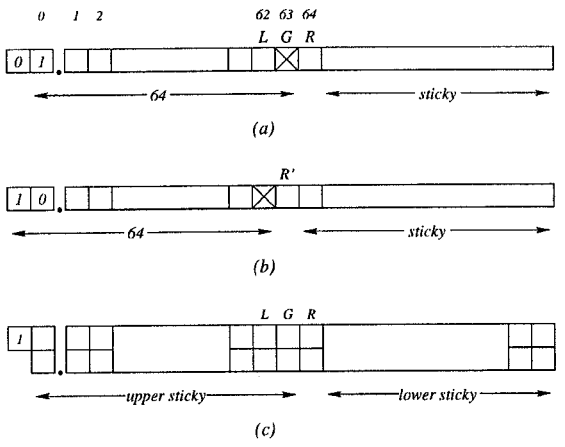

Figure 7.

S_u	R	S_l	range	round pos.
-1	x	x	$(1,2)$	G
0	-1	x	$(1,2)$	G
0	0	-1	$(1,2)$	G
0	0	0	$[2,2]$	L
0	0	1	$(2,4)$	L
0	1	x	$(2,4)$	L
1	x	x	$(2,4)$	L

Table 2. Range and rounding position.

The IEEE floating point standard specifies four basic rounding modes. These four modes can be reduced to three *atomic* rounding modes depending on the sign of the number (see Table 3). The rounding operation in the atomic modes can be computed without examining the sign.

IEEE rounding mode	positive	negative
round up	RI	RZ
round down	RZ	RI
round to zero	RZ	RZ
round to nearest even	RNe	RNe

Table 3. Conversion of IEEE rounding modes into 3 atomic modes: Round to infinity (RI), to zero (RZ) and to nearest even (RNe).

Based on the high order part value f up to the round digit, the round digit (R) and the low order sticky digit (S_l), the rounding ranges can be deduced as depicted in Fig. 8.

From the range we deduce what to add at the rounding position (i.e. -1, 0 or 1) as summarized in Table 4. If the round position is at the guard digit, we shall include this digit in the carry-round packet, since the principle part contains all the digits to the left of the guard digit, i.e.:

$$c = G + \text{Table4}(G, R, S_l, mode). \tag{20}$$

If on the other hand the rounding position is at the L digit, we shall treat L as the least significand digit, make the guard digit into round digit, and incorporate R into the low order sticky digit, i.e.

$$c = 2 \cdot \text{Table4}(L, G, R \wedge S_l, mode). \tag{21}$$

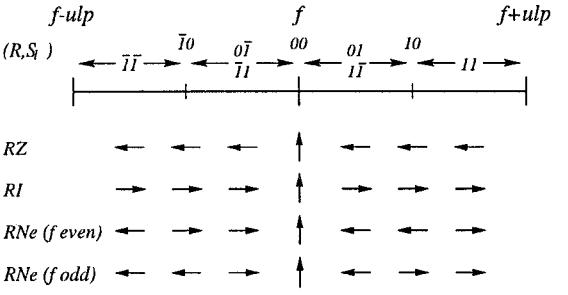

Figure 8. Rounding ranges

R	S_l	RZ	RI	RNe ($G=0$)	RNe ($G\neq 0$)
-1	-1	-1	0	-1	-1
-1	0	-1	0	0	-1
-1	1	-1	0	0	0
0	-1	-1	0	0	0
0	0	0	0	0	0
0	1	0	1	0	0
1	-1	0	1	0	0
1	0	0	1	0	1
1	1	0	1	1	1

Table 4. Simple rounding table (fraction assumed positive and rounding position at L)

Rounding logic can be implemented based on Tables 3–4 and equations (20) and (21). This straight forward implementation would require three consecutive table lookup steps followed by computation of the carry-round packet.

To speed up computation of the carry-round packet we may compute a packet for all possible sticky digit combinations, in parallel with the sticky digit computation. When the actual sticky digits are available the correct packet is selected using a multiplexor. Each one of the nine regions of the composite rounding Table 5, should be implemented

S_l			−1			0			1		
S_u	G	R	RZ	RI	RNe	RZ	RI	RNe	RZ	RI	RNe
-1	-1	-1	-2	-1	-2	-2	-1	-2	-2	-1	-1
	-1	0	-2	-1	-1	-1	-1	-1	-1	0	-1
	-1	1	-1	0	-1	-1	0	0	-1	0	0
	0	-1	-1	0	-1	-1	0	0	-1	0	0
	0	0	-1	0	0	0	0	0	0	1	0
	0	1	0	1	0	0	1	0	0	1	1
	1	-1	0	1	0	0	1	0	0	1	1
	1	0	0	1	1	1	1	1	1	2	1
	1	1	1	2	1	1	2	2	1	2	2
0	x	-1	-1	0	-1	-1	0	0	-1	0	0
	x	0	-1	0	0	0	0	0	0	2	0
	x	1	0	2	0	0	2	0	0	2	0
1	-1	-1	-2	0	-2	-2	0	-2	-2	0	-2
	-1	0	-2	0	-2	-2	0	$-2^{\dagger}$	-2	0	0
	-1	1	-2	0	0	-2	0	0	-2	0	0
	0	-1	-2	0	0	-2	0	0	-2	0	0
	0	0	-2	0	0	0	0	0	0	2	0
	0	1	0	2	0	0	2	0	0	2	0
	1	-1	0	2	0	0	2	0	0	2	0
	1	0	0	2	0	0	2	$2^{\dagger}$	0	2	2
	1	1	0	2	2	0	2	2	0	2	2

Table 5. Carry round Packets (entries marked with † should be 0 if $L = 0$).

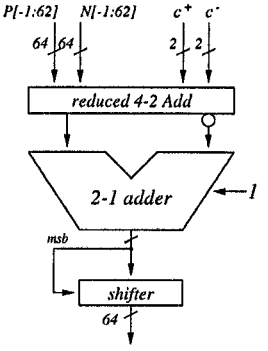

Figure 9. Final Addition.

separately using PLA's to minimize die area by exploiting redundancy in the table.

At the end of the first rounding stage the packet forwarding result output is complete. Now let us consider the final stage of the rounder and coincidently of the adder and multiplier pipes. The carry-round packet (c) is added to the principal part using a redundant 4-2 compressor (i.e. a borrow-save adder). Since the carry-round packet is only a two digit integer, the adder logic can be reduced considerably. The borrow-save encoded sum is sent to 2-1 adder for compression into a standard binary IEEE compliant representation (see Fig, 9). The final adder in effect subtracts the negative digits from the positive digits of the borrow-save encoded input, by forming the 2's complement of the negative digits. If the compressed result is greater than or equal to two, as signaled by the most significant bit, the significand is normalized by shifting it one position to the right. Since the adder does not need to produce two results, as with algorithms employing prediction [4, 5, 6], this part of the implementation can be less restrictively chosen from the large base of standard addition algorithms [7], such that a good tradeoff between latency and area is attainable.

In a floating point stack architecture such as the x86, results that are forwarded are consumed and need not be sent through the second rounding stage. Since floating point intensive computation has the majority of results forwarded [9], the adder and multiplier pipes could each have their own first rounding stage and share the second stage with little extra delay, saving the cost of a second 2-1 adder.

References

[1] "IEEE standard for Binary Floating-Point Arithmetic"; ANSI/IEEE std 754-1985, New York, The Inst. of Electrical and Electronics Engineers, Inc, Aug. 1985.

[2] Nielsen, A. M., Matula, D. W., Even, G., Lyu, C. N.: "Pipelined Packet-Forwarding Floating Point: II. An Adder"; in these proceedings

[3] Daumas, M., Matula, D. W.: "Recoders for Partial Compression and Rounding"; Technical report RR97–01, Ecole Normale Superieure de Lyon, LIP, available at http://www.ens-lyon.fr/LIP.

[4] Yu, R., Zyner, G.: "167 MHz Radix-4 Floating Point Multiplier"; In Proc. of the 9th IEEE Symp. on Computer Arithmetic, 1989, 149–154.

[5] Santoro, M., Bewick, G., Horowitz, M.: "Rounding Algorithms for IEEE multipliers"; In Proc. of the 12th IEEE Symp. on Computer Arithmetic, 1995, 176–183.

[6] Quach, N., Takagi, N., Flynn, M.: "On Fast IEEE Rounding"; Stanford Technical Report: CSL-TR-91-459, available at http://umunhum.stanford.edu/main.html.

[7] Omondi, A. R.: "Computer Arithmetic Systems - Algorithms, Architecture and Implementations"; Prentice Hall, ISBN 0-13-334301-4, 13–101.

[8] Lyu, C.-N.: "Micro-Architecture of a Pipelined Floating-Point Execution Unit"; PhD thesis, SMU, Dallas, Texas, Dec. 1995.

[9] Oberman, S. F., Flynn, M. J.: "Design Issues in Division and Other Floating-Point Operations"; *IEEE Transactions on Computers*, vol. 46, no. 2, February 1997, pp. 154–161.