FLOATING-POINT ARITHMETIC

- Floating-point representation and dynamic range
- Normalized/unnormalized formats
- Values represented and their distribution
- Choice of base
- Representation of significand and of exponent
- Rounding modes and error analysis
- IEEE Standard 754
- Algorithms and implementations: addition/subtraction, multiplication and division

VALUES REPRESENTED IN FLPT SYSTEM

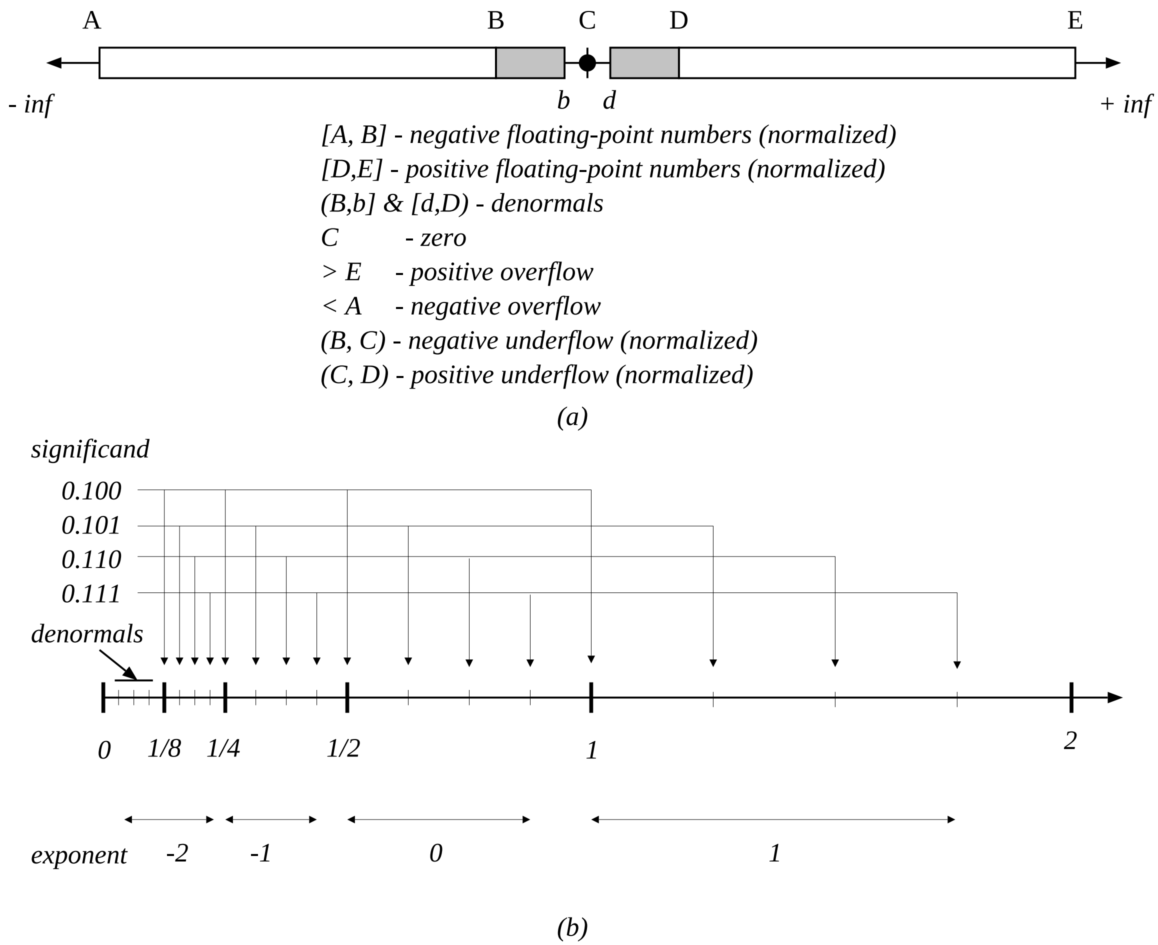


Figure 8.1: a) Regions in floating-point representation. b) Example for $m = f = 3$, $r = 2$, and $-2 \leq E \leq 1$ (only positive region).

	Floating-point system	
	Normalized	Unnormalized
A	$-(r^{m-f} - r^{-f}) \times b^{Emax}$	
B	$-r^{m-f-1} \times b^{Emin}$	$-r^{-f} \times b^{Emin}$
C	0	
D	$r^{m-f-1} \times b^{Emin}$	$r^{-f} \times b^{Emin}$
E	$(r^{m-f} - r^{-f}) \times b^{Emax}$	

DISTRIBUTION FOR $b = 2$, $m = f = 4$, and $e = 2$

Significand	2^E			
	1	2	4	8
0.1000	1/2	1	2	4
0.1001	9/16	9/8	9/4	9/2
0.1010	10/16	10/8	10/4	5
0.1011	11/16	11/8	11/4	11/2
0.1100	12/16	12/8	3	6
0.1101	13/16	13/8	13/4	13/2
0.1110	14/16	14/8	14/4	7
0.1111	15/16	15/8	15/4	15/2

DISTRIBUTION FOR $b = 2$, $m = f = 3$, and $e = 3$

Significand	2^E							
	1	2	4	8	16	32	64	128
0.100	1/2	1	2	4	8	16	32	64
0.101	5/8	5/4	5/2	5	10	20	40	80
0.110	6/8	3/2	3	6	12	24	48	96
0.111	7/8	7/4	7/2	7	14	28	56	112

DISTRIBUTION FOR $b = 4$, $m = f = 4$, and $e = 2$

Significand	4^E			
	1	4	16	64
0.0100	1/4	1	4	16
0.0101	5/16	5/4	5	20
0.0110	6/16	6/4	6	24
0.0111	7/16	7/4	7	28
0.1000	1/2	2	8	32
0.1001	9/16	9/4	9	36
0.1010	10/16	10/4	10	40
0.1011	11/16	11/4	11	44
0.1100	12/16	3	12	48
0.1101	13/16	13/4	13	52
0.1110	14/16	14/4	14	56
0.1111	15/16	15/4	15	60

DISTRIBUTION OF FLPT NUMBERS

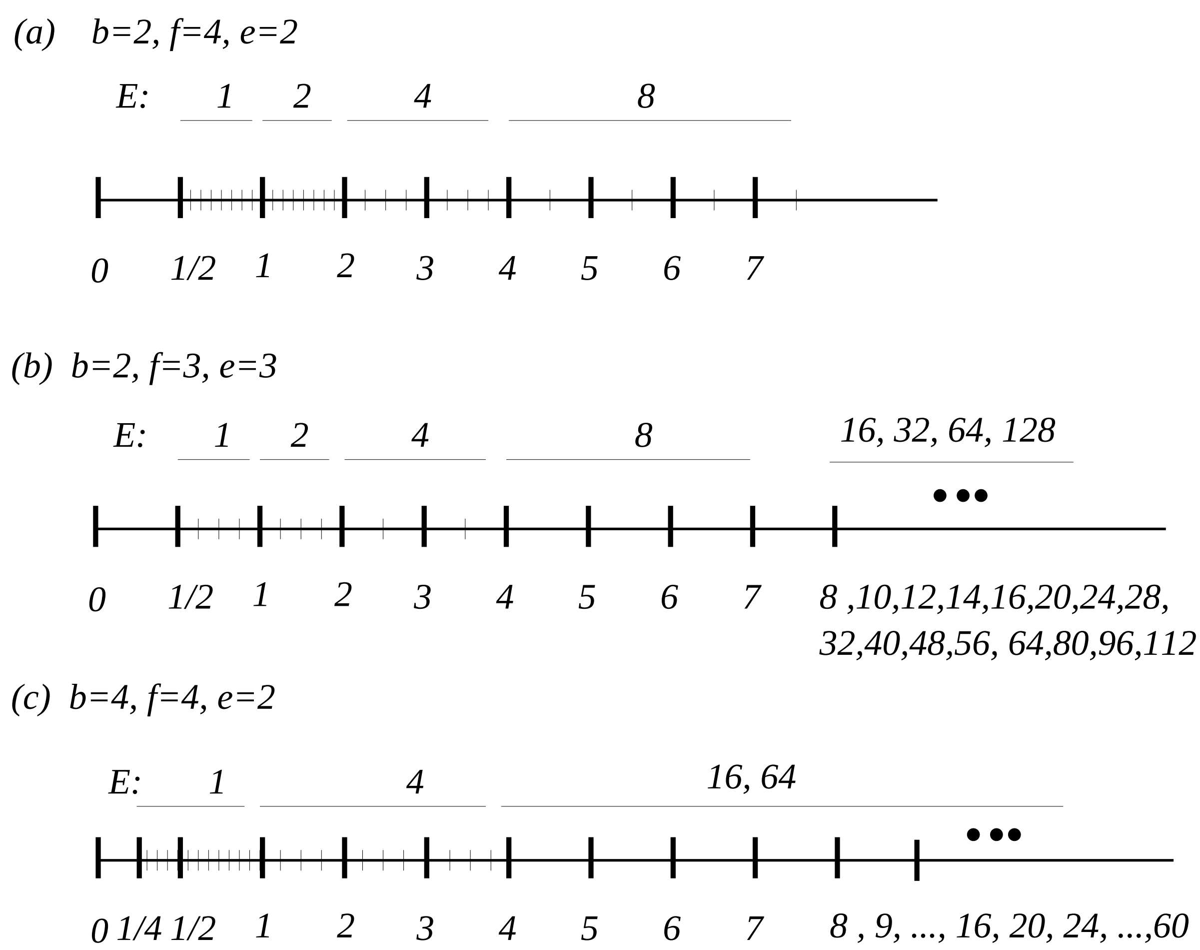


Figure 8.2: EXAMPLES OF DISTRIBUTIONS OF FLOATING-POINT NUMBERS.

REPRESENTATION OF SIGNIFICAND AND EXPONENT

- SIGNIFICAND: SM with HIDDEN BIT

- EXPONENT: BIASED $E_R = E + B$, min$E_R = 0 \quad \Rightarrow \; B = -E_{min}$
- Symmetric range $-B \leq E \leq B \quad \Rightarrow \; 0 \leq E_R \leq 2B \leq 2^e - 1$
- for 8-bit exponent: $B = 127$, $-127 \leq E \leq 128$, $0 \leq E_R \leq 255$
- $E_R = 255$ not used

- SIMPLIFIES COMPARISON OF FLOATING-POINT NUMBERS (same as in fixed-point)

- MINIMUM EXPONENT REPRESENTED BY 0 SO THAT FLOATING-POINT VALUE 0: ALL ZEROS
  (0 sign, 0 exponent, 0 significand)

SPECIAL VALUES AND EXCEPTIONS

- Special values - not representable in the FLPT system
  - NAN (Not A Number)
  - Infinity (pos, neg)
  - allow computation in presence of special values
- Exceptions: result produced not representable - set a flag
  - Exponent overflow
  - Underflow

ROUNDOFF MODES AND ERROR ANALYSIS

- Exact results (inf. precision): x, y, etc.
- FLPT number representing x is $R_{mode}(x)$ with rounding mode $mode$
- Basic relations:

1. If $x \leq y$ then $R_{mode}(x) \leq R_{mode}(y)$
2. If x is a FLPT number then $R_{mode}(x) = x$
3. If $F1$ and $F2$ are two consecutive FLPT numbers then for $F1 \leq x \leq F2$ x is either $F1$ or $F2$

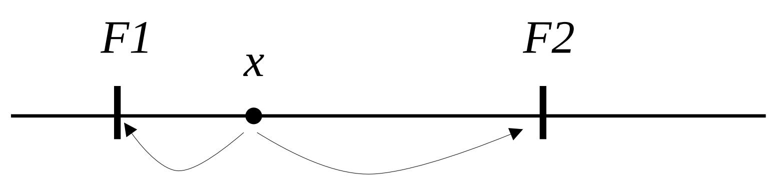


Figure 8.3: Relation between x, $Rmode(x)$, and floating-point numbers $F1$ and $F2$.

ROUNDING MODES

- Round to nearest (tie to even). $Rnear(x)$ is the floating-point number that is closest to x.

$$Rnear(x) = \begin{cases} F1 & \text{if } |x - F1| < |x - F2| \\ F2 & \text{if } |x - F1| > |x - F2| \\ even(F1, F2) & \text{if } |x - F1| = |x - F2| \end{cases}$$

- Round toward zero (truncate). $Rtrunc(x)$ is the closest to 0 among $F1$ and $F2$.

$$Rtrunc(x) = \begin{cases} F1 & \text{if } x \geq 0 \\ F2 & \text{if } x < 0 \end{cases}$$

- Round toward plus infinity. $Rpinf(x)$ is the largest among $F1$ and $F2$

$$Rpinf(x) = F2$$

- Round toward minus infinity. $Rninf(x)$ is the smallest among $F1$ and $F2$

$$Rninf(x) = F1$$

ROUNDING ERRORS

1. The (maximum) absolute representation error ABRE (MABRE))

$$ABRE = Rmode(x) - x$$

so that

$$MABRE = max_x(|ABRE|)$$

2. The average bias (RB)

$$RB = \lim_{t \to \infty} \frac{\Sigma_{M \in \{M_{m+t}\}}(Rmode(M) - M)}{\#M}$$

where $\{M_{m+t}\}$ is the set of all significands with $m + t$ bits, and $\#M$ is the number of significands in the set.

3. The relative representation error (RRE)

$$RRE = \frac{Rmode(x) - x}{x}$$

ERRORS AND IMPLEMENTATION CHARACTERISTICS OF R-MODES

- x described $exactly$ by the triple $(S_x,\ E_x,\ M_x)$
- M_x normalized but having infinite precision
- M_x decomposed into two components M_f and M_d:

$$M_x = M_f + M_d \times r^{-f}$$

- M_f has precision of significand in the FLPT system
- M_d represents the rest, $0 \leq M_d < 1$

ROUNDING TO NEAREST - UNBIASED, TIE TO EVEN

- Value represented - closest possible to the exact value
- The smallest absolute error - the default mode of the IEEE Standard
- Round to nearest specification:

$$Rnear(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d \geq 1/2 \\ M_f & \text{if } M_d < 1/2 \end{cases}$$

- The addition of r^{-f} can produce significand overflow
- Equivalently

$$Rnear(x) = (\lfloor (M_x + \frac{r^{-f}}{2}) r^f \rfloor) r^{-f}$$

- Example: The exact value 1.100100011101 is rounded to nearest with 8-bit precision

```
  1.100100011101
+          1
  --------------
  1.10010010
```

ROUND TO NEAREST (cont.)

- The absolute error is

$$ABRE[Rnear] = \begin{cases} -M_d r^{-f} \times b^E & \text{if } M_d < 1/2 \\ (1 - M_d) r^{-f} \times b^E & \text{if } M_d \geq 1/2 \end{cases}$$

- The maximum absolute error occurs when $M_d = 1/2$

$$MABRE[Rnear] = \frac{r^{-f}}{2} \times b^{Emax}$$

- **unbiased round to nearest**

$$Rnear(x) = \begin{cases} M_f & \text{if } M_d < 1/2 \\ M_f + r^{-f} & \text{if } M_d > 1/2 \\ M_f & \text{if } M_d = 1/2 \text{ and } M_f = \text{even} \\ M_f + r^{-f} & \text{if } M_d = 1/2 \text{ and } M_f = \text{odd} \end{cases}$$

- For this mode

$$RB[Rnear] = 0$$

ROUND TOWARD ZERO (TRUNCATION)

- rounded significand is obtained by discarding M_d.

$$Rzero(x) = (\lfloor M \times r^f \rfloor) r^{-f} = M_f$$

- The absolute error

$$ABRE[Rzero] = -M_d r^{-f} \times b^E$$

and

$$MABRE[Rzero] \approx r^{-f} \times b^{Emax}$$

- Absolute error always negative, the average bias is significant

$$AB[Rzero] \approx -\frac{1}{2} r^{-f}$$

ROUND TOWARD PLUS/MINUS INFINITY

- These two directed modes useful for interval arithmetic (operands and the result of an operation are intervals)
- This permits the monitoring of the accuracy of the result
- Specs:

$$Rpinf(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d > 0 \text{ and } S = 0 \\ M_f & \text{if } M_d = 0 \text{ or } S = 1 \end{cases}$$

$$Rninf(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d > 0 \text{ and } S = 1 \\ M_f & \text{if } M_d = 0 \text{ or } S = 0 \end{cases}$$

- The addition of r^{-f} can produce a significand overflow

ILLUSTRATIONS OF ROUNDING MODES

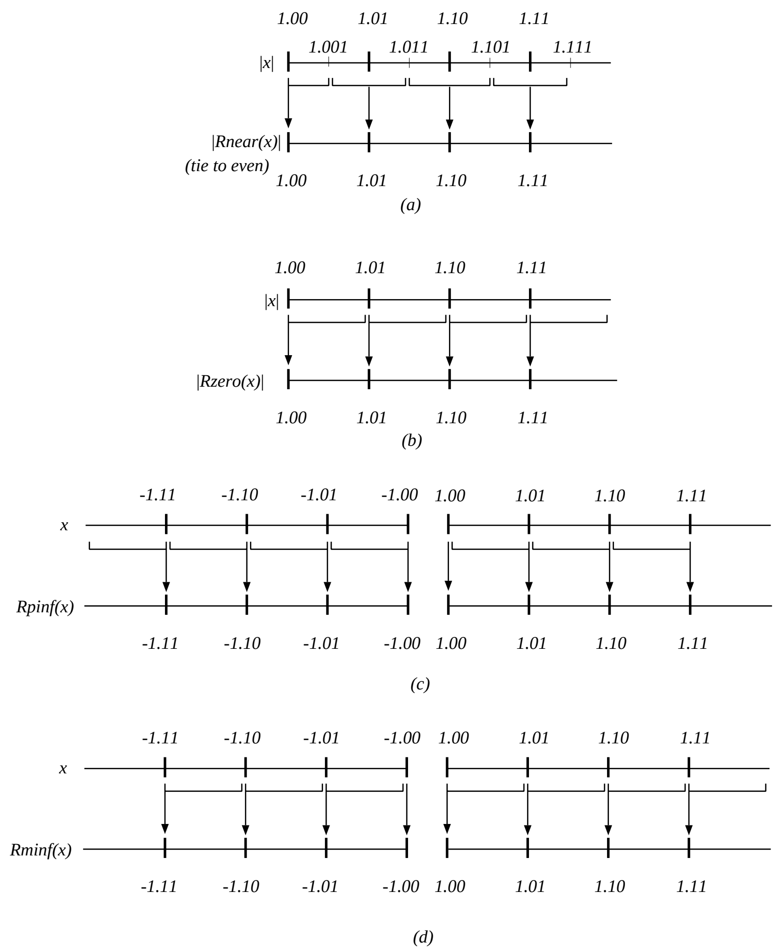


Figure 8.4: ROUNDING TO (a) NEAREST, TIE TO EVEN. (b) ZERO. (c) PLUS INFINITY. (d) MINUS INFINITY.

IEEE FLOATING-POINT STANDARD 754

- Minimizes anomalies
- Enhances portability
- Enhances numerical quality
- Allows different implementations

REPRESENTATION AND FORMATS

1. The significand in SM representation:
   - *Sign* S. One bit. $S = 1$ if negative.
   - *Magnitude* (also called the significand). Represented in radix 2 with one integer bit. That is, the normalized significand is represented by

     $$1.F$$

     where F of f bits (depending on the format) is called the **fraction** and the most-significant 1 is the **hidden bit**.
     The range of the (normalized) significand

     $$1 \leq 1.F \leq 2 - 2^{-f}$$

2. *Exponent*. Base 2 and biased representation; the exponent field e, depending of the format; biased with bias $B = 2^{e-1} - 1$.

SPECIAL VALUES

- The representation of floating-point zero: $E = 0$ and $F = 0$. The sign S differentiates between positive and negative zero.
- The representation $E = 0$ and $F \neq 0$ used for denormals; in this case the floating-point value represented is

$$v = (-1)^S 2^{-(B-1)}(0.F)$$

- The maximum exponent representation $(E = 2^e - 1)$ represents not-a-number (NAN) for $F \neq 0$ and plus and minus infinity for $F = 0$.

BASIC AND EXTENDED FORMATS

- The basic format allows representation in single and double precision

1. Basic: single (32 bits) and double (64 bits)

   - single: S(1),E(8),F(23)

     (a) If $1 \leq E \leq 254$, then $v = (-1)^S 2^{E-127}(1.F)$ (normalized fp number)

     (b) If $E = 255$ and $F \neq 0$, then $v = NAN$ (not a number)

     (c) If $E = 255$ and $F = 0$, then $v = (-1)^S \infty$ (plus and minus infinity)

     (d) If $E = 0$ and $F \neq 0$, then $v = (-1)^S 2^{-126}(0.F)$ (denormal, gradual underflow)

     (e) If $E = 0$ and $F = 0$, then $v = (-1)^S 0$ (positive and negative zero)

   - double: S(1) E(11) F(52)

     – Similar representation to single, replacing 255 by 2047, etc.

2. Extended: single (at least 43=1+11+31) and double (at least 79=1+15+63)

ROUNDING, OPERATIONS, AND EXCEPTIONS

- Rounding

  Default Mode:

  round to nearest, to even when tie

  Directed modes:

  round toward plus infinity

  round toward minus infinity

  round toward 0 (truncate)

- Operations

  Numerical:

  Add, Sub, Mult, Div, Square root, Rem

  Conversions

  Floating to integer

  Binary to decimal (integer)

  Binary to decimal (floating)

cont.

Miscellaneous

Change formats

Compare and set condition code

- Exceptions: By default set a flag and the computation continues

Overflow (when rounded value too large to be represented). Result is set to $\pm$ infinity.

Underflow (when rounded value too small to be represented)

Division by zero

Inexact result (result is not an exact floating-point number). Infinite precision result different than floating-point number.

Invalid. This flag is set when a NAN result is produced.

FLOATING-POINT ADDITION/SUBTRACTION

- x and y - normalized operands represented by (S_x, M_x, E_x) and (S_y, M_y, E_y)

1. Add/subtract significand and set exponent

$$
\begin{aligned}
M_z^* &= \begin{cases} (M_x^* \pm (M_y^*) \times (b^{E_y - E_x})) \times b^{E_x} & \text{if } E_x \geq E_y \\ ((M_x^*) \times (b^{E_x - E_y}) \pm M_y^*) \times b^{E_y} & \text{if } E_x < E_y \end{cases} \\
E_z &= max(E_x, E_y)
\end{aligned}
$$

```
Ex - Ey = 4

Mx                1.xxxxxxxxxxx
My(2^(Ey-Ex))     0.0001yyyyyyyyyyy
                  -----------------
                  z.zzzzzzzzzzzzzzz
```

2. Normalize significand and update exponent.
3. Round, normalize and adjust exponent.
4. Set flags for special cases.

BASIC ALGORITHM

1. **Subtract exponents** ($d = E_x - E_y$).
2. **Align significands**
   - Shift right d positions the significand of the operand with the smallest exponent.
   - Select as exponent of the result the largest exponent.
3. **Add (Subtract) significands and produce sign of result**. The effective operation (add or subtract):

Floating-point op.	Signs of operands	Effective operation (EOP)
ADD	equal	add
ADD	different	subtract
SUB	equal	subtract
SUB	different	add

cont.

4. **Normalization of result**. Three situations can occur:

(a) The result already normalized: no action is needed

```
         1.10011111
         0.00101011
ADD      ----------
         1.11001010
```

(b) Effective operation addition: there might be an overflow of the significand. The normalization consists in

- Shift right the significand one position
- Increment by one the exponent

```
        1.1001111
        0.0110110
ADD     ---------
       10.0000101
NORM    1.00000101
```

cont.

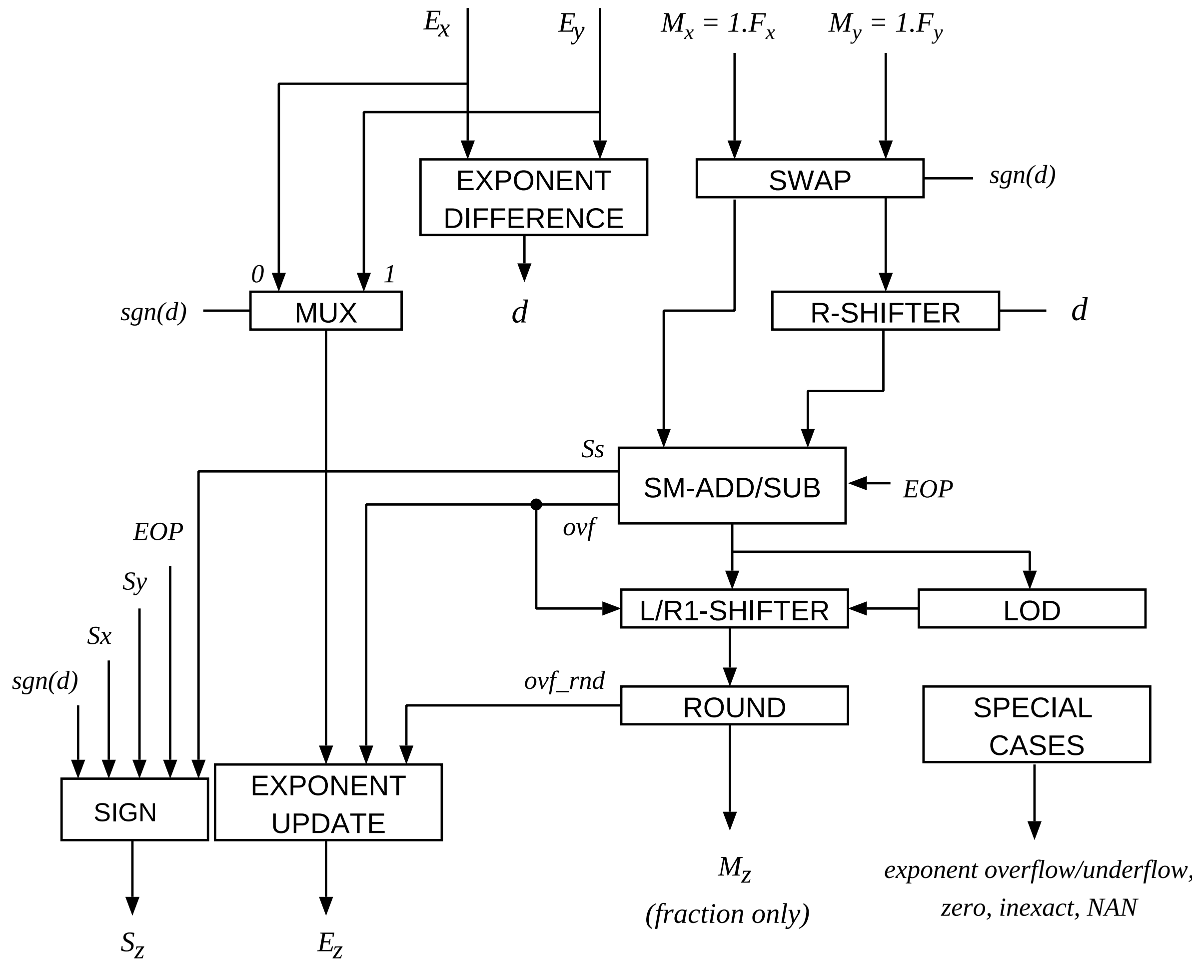
(c) Effective operation subtraction: the result might have leading zeros. Normalize:

- Shift left the significand by a number of positions corresponding to the number of leading zeros.
- Decrement the exponent by the number of leading zeros.

```
         1.1001111
         1.1001010
  SUB    ---------
         0.0000101
 NORM    1.0100000
```

5. **Round**. According to the specified mode. Might require an addition. If overflow occurs, normalize by a right shift and increment the exponent.

6. **Determine exception flags and special values** : exponent overflow (special value $\pm$ infinity), exponent underflow (special value gradual underflow), inexact, and the special value zero.



EOP: effective operation
R-SHIFTER: variable right shifter
L/R1-SHIFTER: variable left/one pos. right shifter
LOD: Leading One Detector

Figure 8.5: BASIC IMPLEMENTATION OF FLOATING-POINT ADDITION.

COMMENTS ON BASIC IMPLEMENTATION

- Significand normalized and in SM
- Base of exponent is 2

1. One alignment shifter: swap the significands according to the sign of the exponent difference.
2. The adder: SM adder. Complicated - several options can be used:
   (a) Use a two's complement adder
   (b) Use a ones' complement adder
   (c) Use a two's complement adder; complement the smallest operand so that the result is positive and no complementation is required.
   To determine the smallest operand, two cases:
   - The exponents are different: the operand with smallest exponent shifted right and complemented
   - The exponents are the same: compare the significands in parallel with the alignment

COMMENTS (cont.)

3. The normalization step requires:
   - The detection of the position of the leading 1 uses LOD (Leading-One-Detector)
   - A shift performed by the shifter:
     - no shift
     - right shift of one position, or
     - left shift of up to m positions
4. The rounding step uses several guard bits

GUARD BITS AND ROUNDING

- Keep all $2m$ bits? No, a few additional bits sufficient: **guard bits**
- How many?
- For rounding toward zero (truncation): f fractional bits
- For rounding to nearest: one additional bit is required ($f + 1$ fractional bits).

  For unbiased rounding to even: necessary to know when the rest of the bits are all zero
- For rounding toward infinity: necessary to know when all the bits to be discarded are zero

EFFECTIVE ADDITION/SUBTRACTION CASES

1. Effective addition:
   - Result either normalized or produces an overflow
   - Normalization: a 1-bit right shift (if overflow); no left shift required
   - $\Rightarrow f+1$ fractional bits of the result required (R)
   - Determine whether all the discarded bits are zero: *sticky bit* T, corresponds to the OR of the discarded bits

```
           1.0101110
           0.00010101010
ADD        -------------
           1.01110001   T=OR(010)=1
```

cont.

2. Effective subtraction. Two sub-cases:

   (a) The difference of exponents d is larger than 1.

   - the smallest operand is aligned so that there are more than one leading zeros
   - the result is either normalized or, if not normalized, has only one leading zero
   - the normalization is performed by a left shift of one position, in addition to the bit for rounding to nearest, another bit is required in the result of the addition.

   $\Rightarrow f+2$ fractional bits of result required

   - During the subtraction, a borrow produced when sticky $= 1$

   $\Rightarrow f+3$ bits required in subtraction (GRT)

cont.

Example: After alignment

```
          1.0000011
          0.000011011001
 SUB      ----------------
```

During alignment compute T=OR(001)=1 resulting in

```
          1.0000011
          0.0000110111
   SUB    -------------
          0.1111100001
   NORM   1.1111000010
```

cont.

(b) The difference of exponents is either 0 or 1.

- Result might have more than one leading zeros
- Left shift of up to m positions required
- Since alignment shift only of zero or one position, at most one non-zero bit is shifted in during the normalization

$\Rightarrow$ only one additional bit required

```
        1.0000011
        0.11111001
SUB     ----------
        0.00001101
NORM    1.10100000
```

SUMMARY OF GUARD BITS

- in all cases three additional bits sufficient:
  guard (G), round(R), and sticky (T)
- After normalization guard bits labeled as follows:

```
         LGRT
1.xxxxxxxxxxxx
  ----f----
```

- During normalization sticky bit recomputed (OR of the previous T and the previous R)

ROUND TO NEAREST

- Round up (add rnd to position L)
  - If $G = 1$ and R and T are not both zero, $rnd = G(R + T)$
  - If $G = 1$ and $R = T = 0$ then $rnd = G(R + T)'L$ – tie case

Combining both cases,

$$rnd = G(R + T) + G(R + T)'L = G(L + R + T)$$

```
L 1 1 0 1 1 1     G=1, R=1, T = 1 -> rnd = 1

L 1 0 0 0 0 0     G=1, R=0, T=1  -> rnd = 1 (tie case)

L 0 x x x x x     G=0   rnd = 0
```

DIRECTED ROUNDINGS

- Round toward zero: after normalization, truncate at bit L
- Round toward infinity:

  Positive infinity

$$rnd = sgn'(G + R + T)$$

  Negative infinity

$$rnd = sgn(G + R + T)$$

EXCEPTIONS AND SPECIAL VALUES

- Overflow:
  - detected by an exponent $E \geq 255$
  - set overflow flag, set result to $\pm$ infinity
- Underflow:
  - detected when during the left shift the exponent $E = 1$ and the significand not normalized
  - set underflow flag, set result exponent to $E = 0$
  - fraction left unnormalized (denormal, gradual underflow)
- Zero: the significand of the result of addition is 0

  The result is $E = 0$ and $F = 0$
- Inexact:
  - detected before rounding: the result is inexact if $G + R + T = 1$
  - set inexact flag
- NAN: if one (or both) operand is a NAN, the result set to NAN.

DENORMAL AND ZERO OPERANDS AND/OR RESULT

- Operand(s):
  - Operand a denormal number ($E = 0$ and $F \neq 0$): no hidden 1
  - Set operand of addition to $E = 1$ and $0.F$
- Zero operand ($E = 0$ and $F = 0$): treated as a denormal number
- Result:
  - detected during left shift: partially updated exponent $E = 1$ and significand not normalized
  - If resulting significand is not 0 then it is a denormal,
    if it is 0 then the result is zero
    exponent set to $E = 0$

CRITICAL PATH

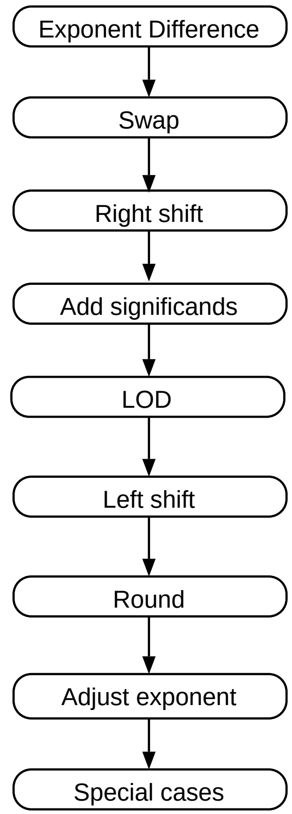
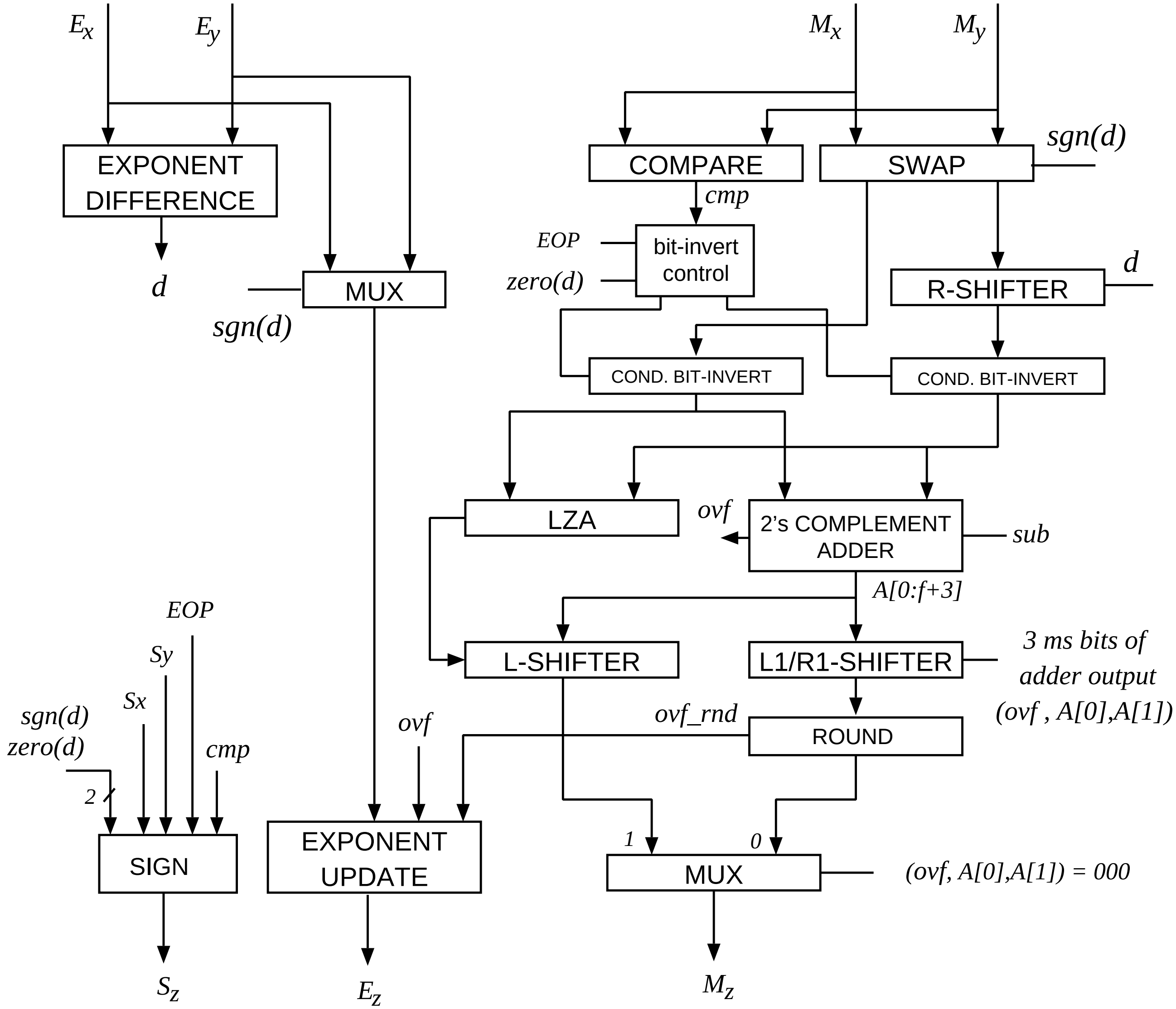


Figure 8.6: FLPT ADDITION: Critical Path.



(handling of special cases not shown)

EOP: effective operation
R-SHIFTER: variable right shifter
L-SHIFTER: variable left shifter
L1/R1-SHIFTER: one position left/right shifter

Figure 8.7: IMPROVED SINGLE-PATH FLOATING-POINT ADDITION.

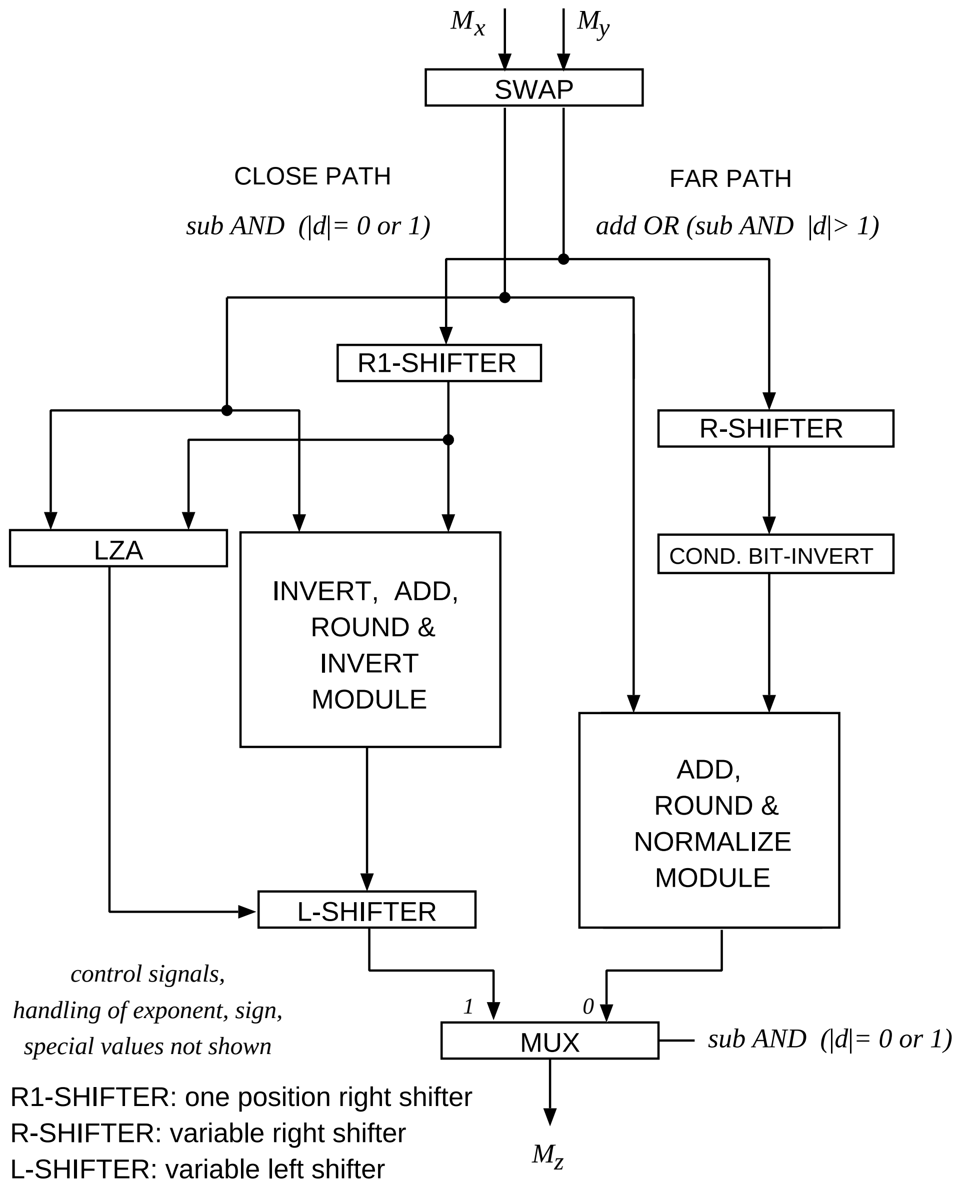


Figure 8.8: DOUBLE-PATH IMPLEMENTATION OF FLOATING-POINT ADDITION.

DEPENDENCE GRAPH FOR DOUBLE-PATH SCHEME

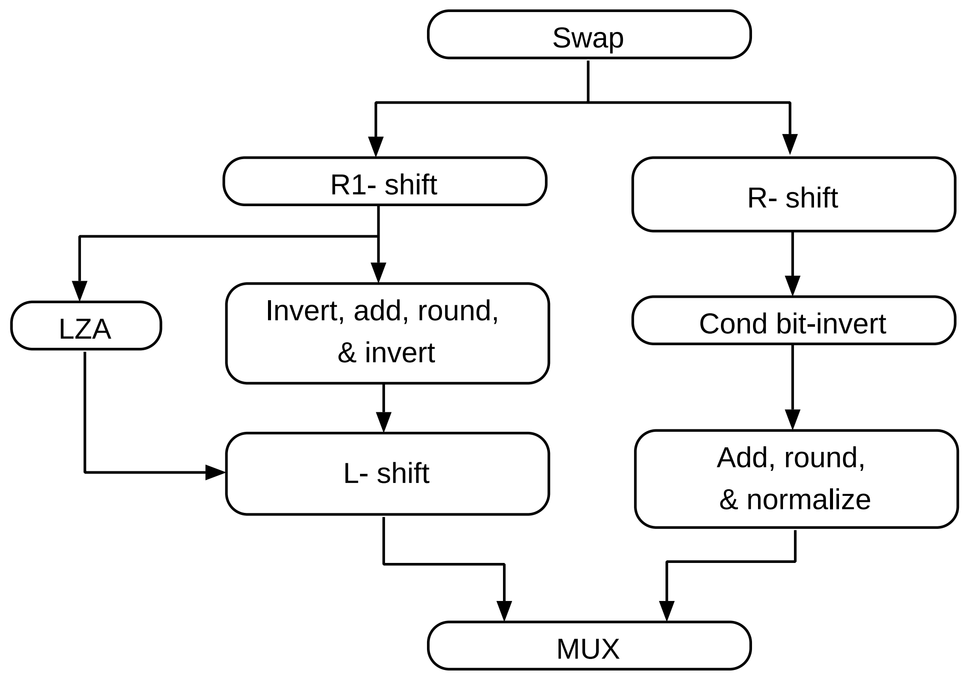


Figure 8.9: Dependence graph for double-path scheme.

PIPELINED IMPLEMENTATION

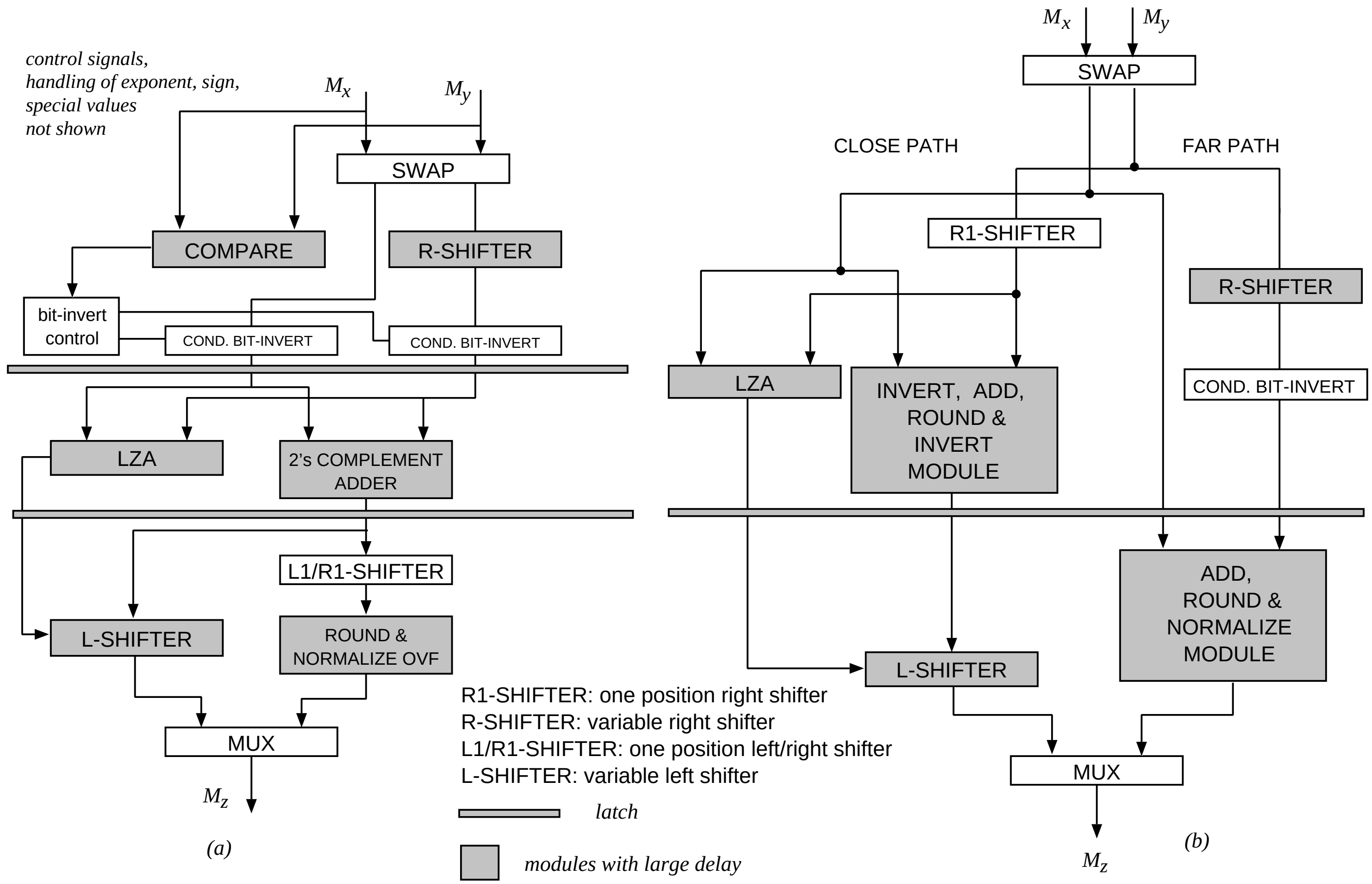


Figure 8.10: PIPELINED IMPLEMENTATIONS: (a) SINGLE-PATH SCHEME. (b) DOUBLE-PATH SCHEME.

FLPT MULTIPLICATION

- x and y - normalized operands represented by (S_x, M_x, E_x) and (S_y, M_y, E_y)

  1. Multiply significands, add exponents, and determine sign

  $$\begin{aligned} M_z^* &= M_x^* \times M_y^* \\ E_z &= E_x + E_y \end{aligned}$$

  2. Normalize M_z^* and update exponent
  3. Round
  4. Determine exception flags and special values

BASIC IMPLEMENTATION

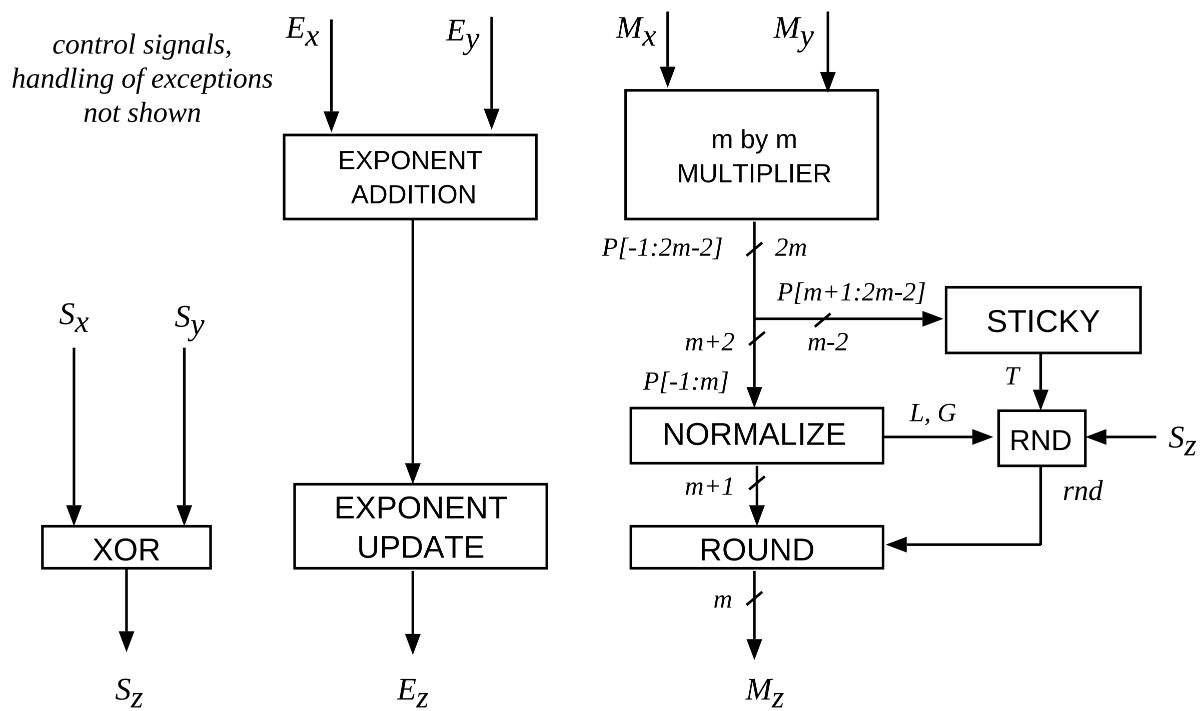


Figure 8.11: BASIC IMPLEMENTATION OF FLOATING-POINT MULTIPLICATION.

COMMENTS ON IMPLEMENTATION

1. Multiplication of magnitudes
   - produces magnitude P of $2m$ bits - only m bits in result: one guard bit and the sticky bit

```
Output of multiplier module P:
Bit position: (-1)0.123...(m-2)(m-1) m (m+1)...(2m-2)
```

2. Exponent of result

$$E_z = E_x + E_y - B$$

3. Sign of result

$$S_z = S_x \oplus S_y$$

cont.

4. Normalization: $1 \leq M_x, M_y < 2$, the result in range $[1,4)$

```
Output of multiplier module P:

Bit position: (-1)0.123...(m-2)(m-1) m (m+1)...(2m-2)

If P[-1]=0, P is normalized:

   L = P[m-1], G = P[m], T = OR(P[m+1],...,P[2m-2])

If P[-1] = 1, normalize P by shifting right one position

   L = P[m-2], G = P[m-1], T = OR(P[m],...,P[2m-2])
```

cont.

5. Rounding: four rounding modes with guard bit (G) and sticky bit (T)

- Round to nearest

$$rnd = G(T) + G(T)'L = G(T + L)$$

with G and T the two bits following L AFTER the normalization.

- Round toward zero

Result after normalization truncated at bit L

- Round toward infinity

positive infinity add

$$rnd = sgn'(G + T)$$

negative infinity

$$rnd = sgn(G + T)$$

EXCEPTIONS AND SPECIAL VALUES, ETC.

- Overflow: exponent too large;
  detected after exponent update;
  overflow flag set; result value is $\pm$infinity
- Underflow: resulting exponent too small;
  underflow flag set; exponent set to $E = 0$
  significand shifted right to represent a denormal
- Zero: when one of the operands has value 0 and the other is not $\pm$ infinity;
  - zero result set

EXCEPTIONS, SPECIAL VALUES, ETC. (cont.)

- Inexact: result inexact if, after normalization, $G + T = 1$
- NAN: result NAN if one (or both) of the operands is a NAN or if one of the operands is a 0 and the other $\pm$ infinity
- Denormals: result denormal if one or both operands are denormal;

  left shift necessary;

  if exponent underflow, right shift (gradual underflow); set E=0

ALTERNATIVE IMPLEMENTATION

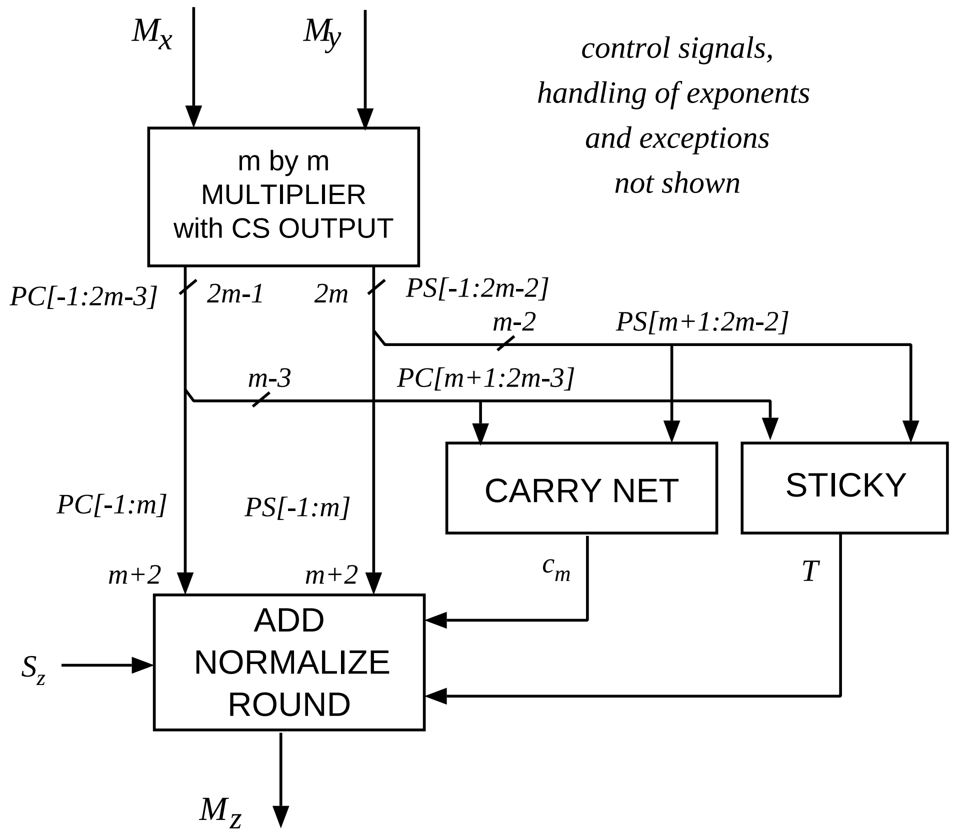


Figure 8.12: ALTERNATIVE IMPLEMENTATION.

REDUCING THE LATENCY

- Compute MS half ($+$ guard bit) in conventional form using c_m;
  c_m in the critical path
- Determine sticky from the operands;
  needs detector of trailing zeros, adder, and comparator

- Determine sticky from CS form of the LS half

```
S       s s s s s s s s
C       c c c c c c c c
-1      1 1 1 1 1 1 1 1
        ----------------
        z z z z z z z z
        t t t t t t t
```

$$\begin{aligned} z_i &= (s_i \oplus c_i)' \\ t_i &= s_{i+1} + c_{i+1} \end{aligned} \tag{8.3}$$

Compute

$$w_i = z_i \oplus t_i \tag{8.4}$$

Sticky bit is

$$T = NAND(w_i) \tag{8.5}$$

CARRY FROM LS PART

Product P[-1:2m-2]

```
-------(m+2)----- -----(m-2)-----
xxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx xxxxxxxxxxxxxx
                c_m

    c_m is the carry produced by
    the least-significant m-2 bits of product P
    and added in position m.
```

Figure 8.13: ADDING CARRY FROM THE LEAST SIGNIFICANT HALF.

ROUNDING POSITION

```
Bit position: (-1)(0).123...(m-2)(m-1) m
                0  1 .xxx...  x    x   x
                                       c_m
                                       1
                  (a)

Bit position: (-1)(0).123...(m-2)(m-1) m
                1  x .xxx...  x    x   x
                                   1   c_m
                  (b)
```

Figure 8.14: ROUNDING POSITION: (a) NORMALIZED PRODUCT. (b) UNNORMALIZED PRODUCT.

ADDING CARRY AND ROUNDING

- Product in CS form - normalized?
- Combine final addition and rounding. Select the correct result.
- $PM = PS + PC$ – the MS of the product up to position m
- Compute

$$P0 = PM + (c_m + 1) \times 2^{-m}$$

and

$$P1 = PM + (c_m + 2) \times 2^{-m}$$

and then select

$$P = \begin{cases} P0 & \text{if } P0[-1] = 0 \\ 2^{-1}P1 & \text{if } P0[-1] = 1 \end{cases}$$

```
      (-1) 0. 1 2 3 ... (m-2)(m-1)(m)
PS      x   x  x x x        x    x   x
PC      x   x  x x x        x    x   x
                                 c_m c'_m <=> (c_m+1)2^(-m)
    ---------------------------------
PS*     x   x  x x x        x    x   x
PC*     x   x  x x x        x    x

Get P0 and P1 = P0 + 2^{-m}:
PS*     x   x  x x x        x    x   x
PC*     x   x  x x x        x    x   0
   -----------------------------------
P0     ovf  x  x x x        x    x   x
P1      x   x  x x x        x    x   x

After selection:
P           1. x x x ...    x    L
```

Figure 8.15: ADDING CARRY c_m AND ROUNDING.

IMPLEMENTATION

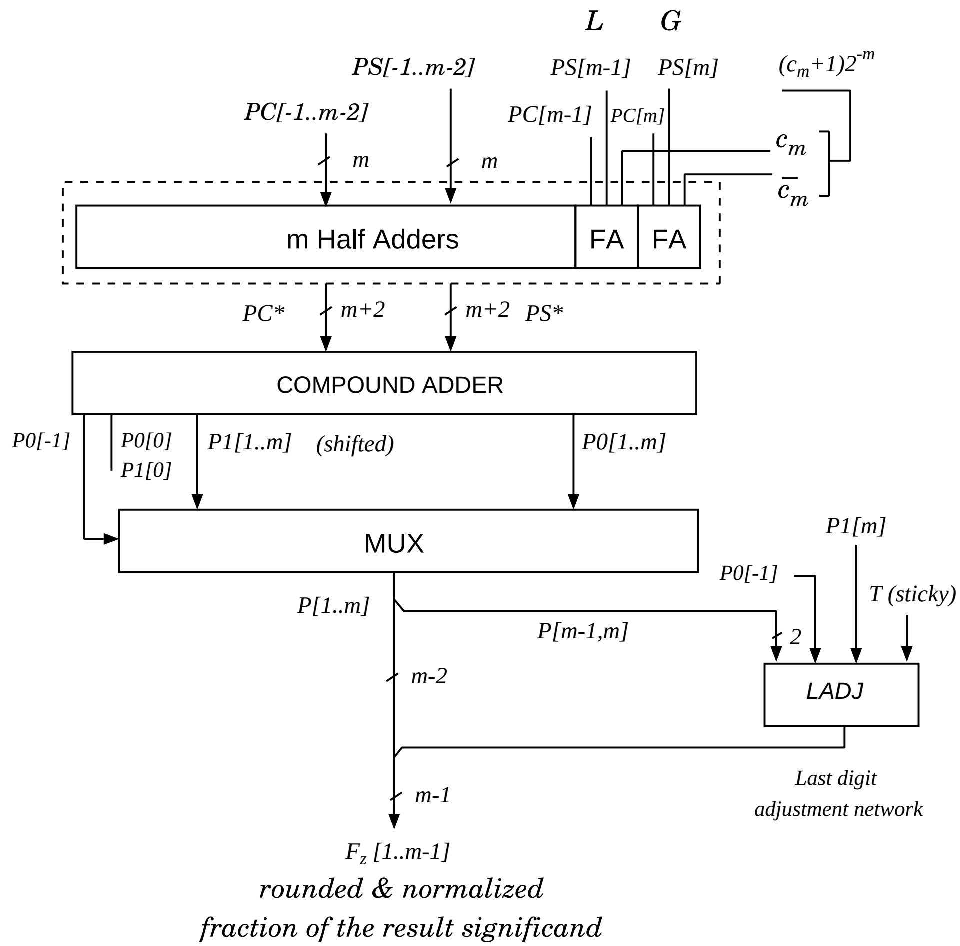


Figure 8.16: ADDING CARRY, NORMALIZATION, AND ROUNDING IMPLEMENTATION

IMPLEMENTATION (cont.)

1. A row of HAs and FAs to add $(c_m + 1)2^{-m}$ to $PS[-1, m]$ and $PC[-1, m]$.
2. A compound adder that produces the sum $P0$ and the sum plus 1 ($P1$).
3. A multiplexer which selects $P0$ or the normalized (shifted) $P1$ depending whether $P0$ does not overflow or overflows
4. A module $LADJ$ which determines the least-significant bit of the significand.

   sticky bit update:

$$T^* = T + P1[m] \cdot P0[-1] \quad \text{update sticky bit}$$

   adjustment of the least-significant bit

$$L = P[m-1](P[m] + T^*)$$

REMOVING c_m FROM CRITICAL PATH

carry+sum in pos. m	range of Σ before pre-add	range of c_{m-1}	pre-add 1?	range of Σ after pre-add	range of c_{m-1}
0	[1,3]	[0,1]	NO	[1,3]	[0,1]
1	[2,4]	[1,2]	YES	[0,2]	[0,1]
2	[3,5]	[1,2]	YES	[1,3]	[0,1]

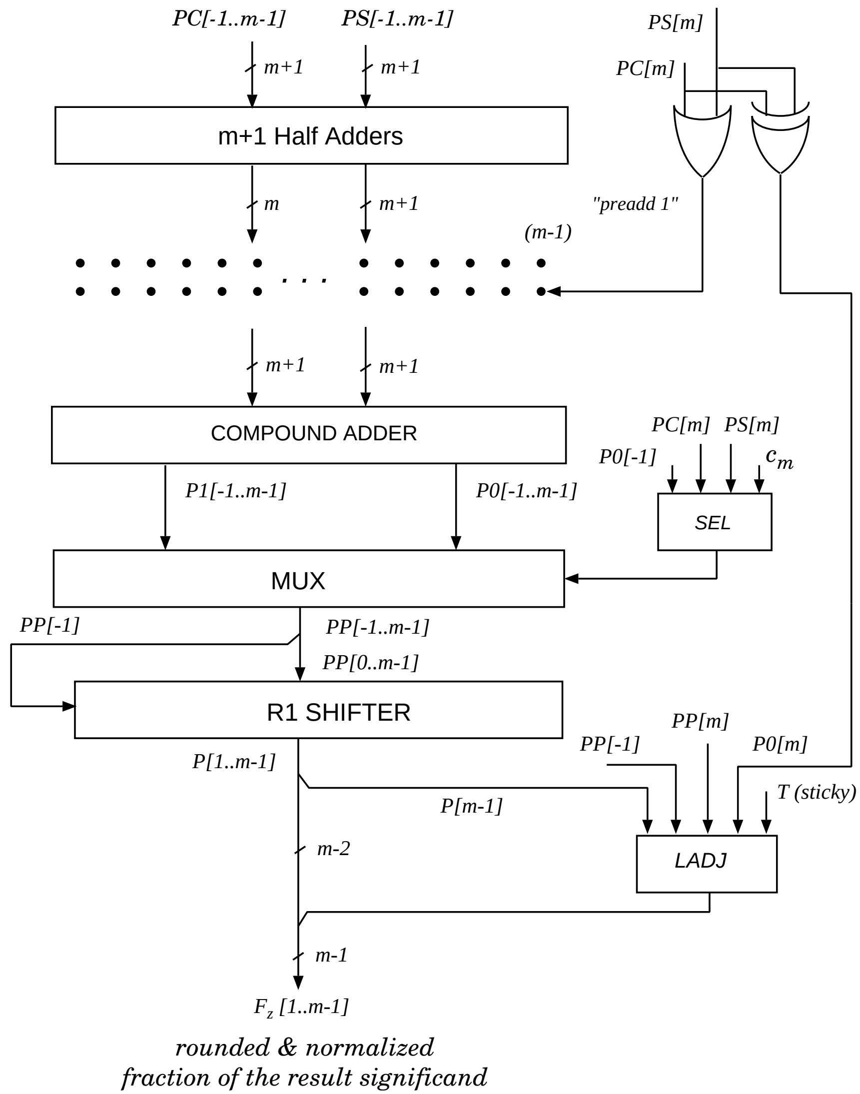


Figure 8.17: Adding carry, normalization, and rounding implementation with carry out of critical path.

MULTIPLY-ADD FUSED (MAF)

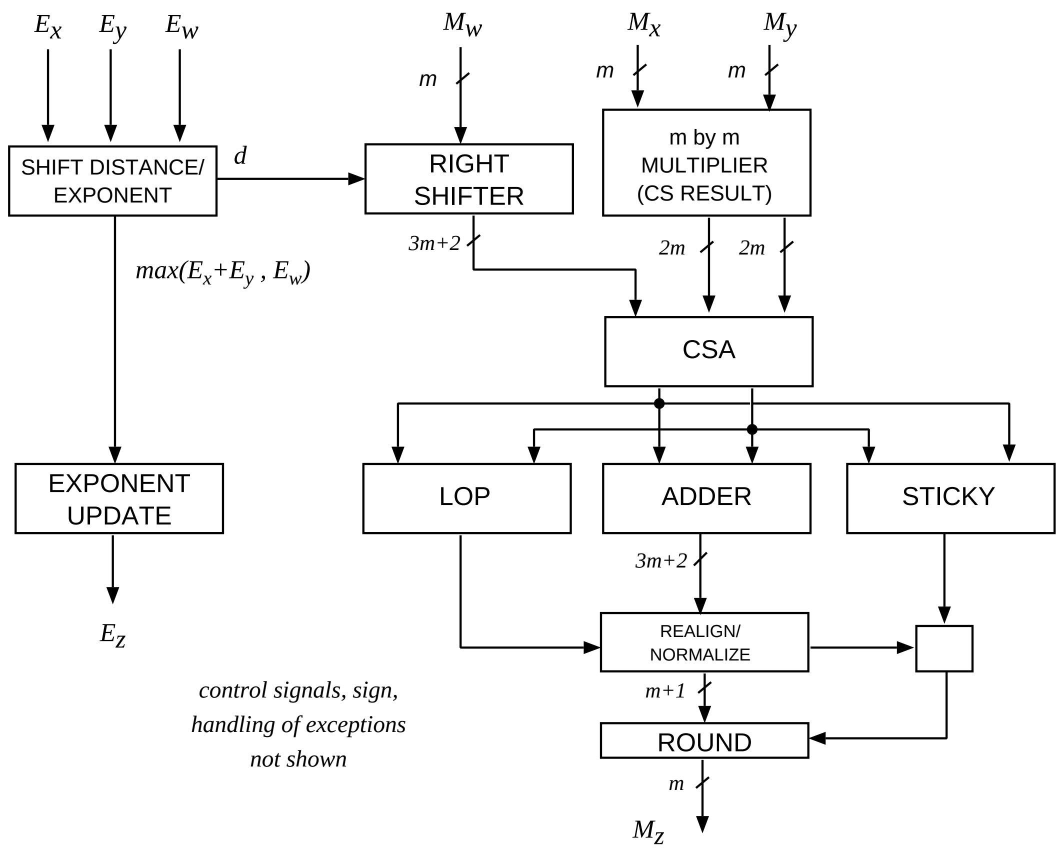


Figure 8.18: BASIC IMPLEMENTATION OF MAF OPERATION.

MAF (cont.)

```
                 ------ m ------  -------- 2m ---------
Product x*y:                    00xx.xxxxx...xxxxxxxxxx
Addend:          1.xxxxxxxxxxxxx
                  |--- m-1+4 -------|
                         (a)

                                  -------- 2m ---------
Product x*y:                      xx.xxxxx...xxxxxxxxxx
Addend:                                                  01xxxxxxxxxxxxx
Shift distance:                     |--- 2m-2+1 -------|

                          (b)
```

Figure 8.19: Position of addends using bidirectional shift: (a) Maximum left shift. (b) Maximum right shift.

AVOIDING BIDIRECTIONAL SHIFTER

- Position addend M_w $m+3$ bits to the left of the product
- shift right by the distance

$$d = E_x + E_y - E_w + m + 3 \tag{8.6}$$

- for biased exponent performed as

$$d = E_x^B + E_y^B - E_w^B - B + m + 3 \tag{8.7}$$

- No shift performed for $d \leq 0$ and the maximum shift is $3m+1$

```
Initial position:
               ------ m ------  -------- 2m ------------
Product x*y:                  00xxxxxxxx....xxxxxxxxxxxx
Addend:        1.xxxxxxxxxxxxx
                |--- m-1+4 -------|
                                |--- sticky ------------|
                                     region
                         (a)

Alignment when Exy = Ew:
               ------ m ------  -------- 2m --------------
Product x*y:                  00xx.xxxxx....xxxxxxxxxxxxxxx
Addend:                          1.xxxxxxxxxxxxx
                                                  |-sticky|
Shift distance: |--- m-1+4 -------|

                         (b)
```

```
Alignment when Exy - Ew = k:
               ------ m ------  -------- 2m --------
Product x*y:                   00xx.xxxxx....xxxxxxxx
Addend:                                        1xxxxxxxxxxxxxx
Shift distance: |----- m+3 -------|----- k ----|

                                (c)

Alignment when Exy - Ew >= 2m-1:
               ------ m ------  -------- 2m --------
Product x*y:                   00xx.xxxxx....xxxxxxxx
Addend:                                             01xxxxxxxxxxxxxx
Shift distance: |----- m+3 -------|----- 2m-1 -------|

                                (d)
```

Figure 8.20: Alignment with right shifter.

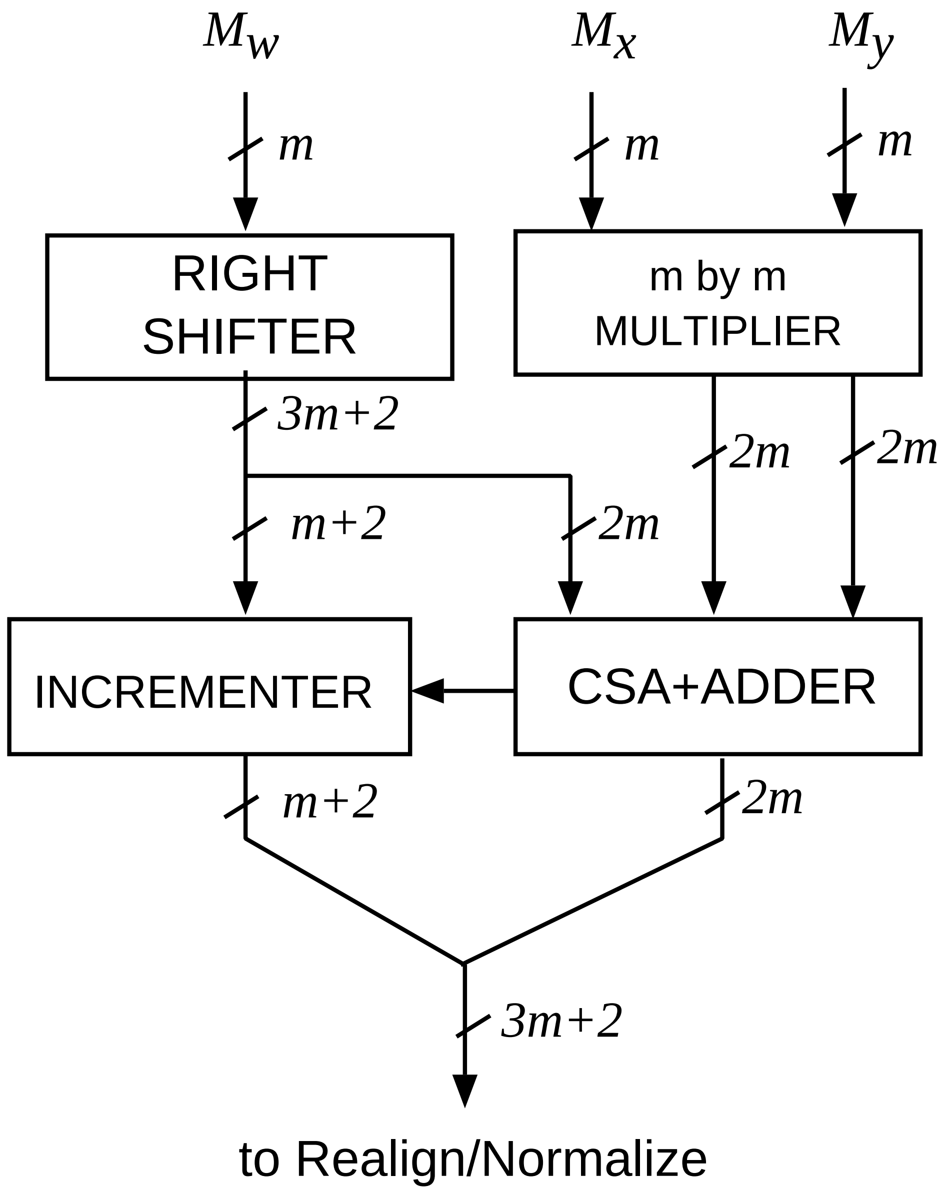


Figure 8.21: Implementation of MAF adder.

LEFT SHIFTING OF ADDER OUTPUT

```
Adder output       |----- m+2 -----|--------- 2m ----------|

Before shift        000000000000000001.xxxxxxxxxxxxxxxxxxx

After shift         1.xxxxxxxxxxxxLGRT
```

Figure 8.22: Left shifting of the adder output.

PIPELINED MAF

- MAF unit usually pipelined.
- Three-stage pipeline:
  - Stage 1 implements the multiplication, alignment and 3-2 carry-save addition;
  - Stage 2 performs 2-1 addition and predicts the leading one in the sum;
  - Stage 3 performs normalization and rounding

FLOATING-POINT DIVISION

- Operands: x and d represented by $(M_x^*,\ E_x)$ and $(M_d^*,\ E_d)$, with M_x^* and M_d^* signed and normalized. The result

$$q = x/d \tag{8.8}$$

represented by $(M_q^*,\ E_q)$, with M_q also signed and normalized.

- The high-level description of the floating-point division algorithm

1. Divide significands and subtract exponents

$$\begin{aligned} M_q^* &= M_x^*/M_d^* \\ E_q &= E_x - E_d \end{aligned} \tag{8.9}$$

2. Normalize M_q^* and update exponent
3. Round
4. Determine exception flags and special values

IMPLEMENTATION

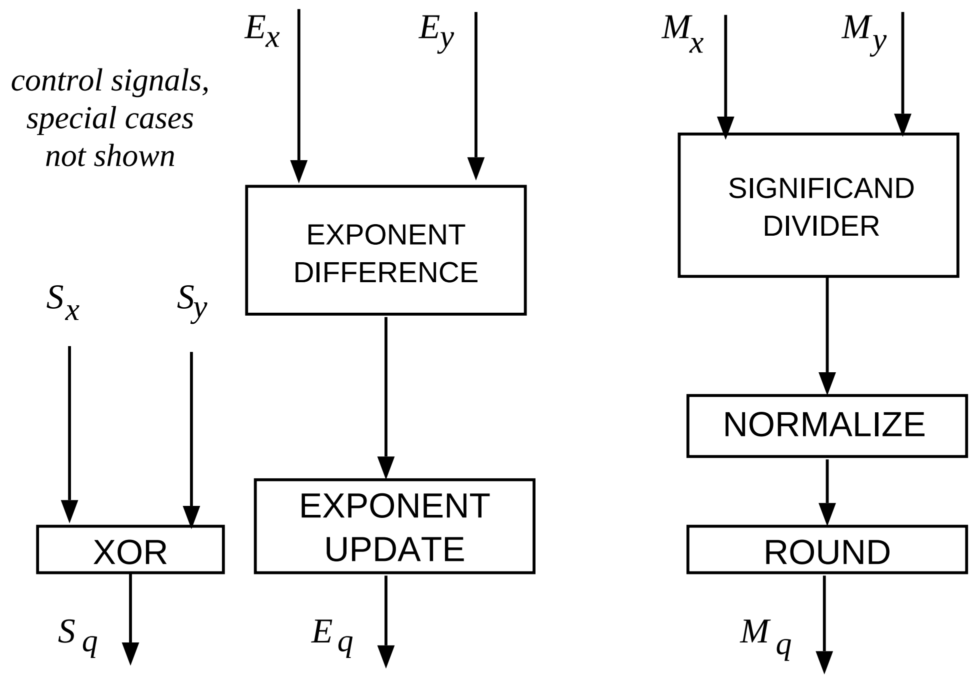


Figure 8.23: Basic implementation of floating-point division.