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).
<table>
<thead>
<tr>
<th></th>
<th>Floating-point system</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normalized</td>
<td>Unnormalized</td>
</tr>
<tr>
<td>A</td>
<td>(- (r^{m-f} - r^{-f}) \times b^{E_{max}})</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(- r^{m-f-1} \times b^{E_{min}})</td>
<td>(- r^{-f} \times b^{E_{min}})</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(r^{m-f-1} \times b^{E_{min}})</td>
<td>(r^{-f} \times b^{E_{min}})</td>
</tr>
<tr>
<td>E</td>
<td>((r^{m-f} - r^{-f}) \times b^{E_{max}})</td>
<td></td>
</tr>
</tbody>
</table>
### DISTRIBUTION FOR $b = 2$, $m = f = 4$, and $e = 2$

<table>
<thead>
<tr>
<th>Significand</th>
<th>$2^E$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.1000</td>
<td>1/2</td>
</tr>
<tr>
<td>0.1001</td>
<td>9/16</td>
</tr>
<tr>
<td>0.1010</td>
<td>10/16</td>
</tr>
<tr>
<td>0.1011</td>
<td>11/16</td>
</tr>
<tr>
<td>0.1100</td>
<td>12/16</td>
</tr>
<tr>
<td>0.1101</td>
<td>13/16</td>
</tr>
<tr>
<td>0.1110</td>
<td>14/16</td>
</tr>
<tr>
<td>0.1111</td>
<td>15/16</td>
</tr>
</tbody>
</table>
DISTRIBUTION FOR $b = 2$, $m = f = 3$, and $e = 3$

<table>
<thead>
<tr>
<th>Significand</th>
<th>$2^E$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.100</td>
<td>1/2</td>
</tr>
<tr>
<td>0.101</td>
<td>5/8</td>
</tr>
<tr>
<td>0.110</td>
<td>6/8</td>
</tr>
<tr>
<td>0.111</td>
<td>7/8</td>
</tr>
</tbody>
</table>
DISTRIBUTION FOR $b = 4$, $m = f = 4$, and $e = 2$

<table>
<thead>
<tr>
<th>Significand</th>
<th>$4^E$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.0100</td>
<td>1/4</td>
</tr>
<tr>
<td>0.0101</td>
<td>5/16</td>
</tr>
<tr>
<td>0.0110</td>
<td>6/16</td>
</tr>
<tr>
<td>0.0111</td>
<td>7/16</td>
</tr>
<tr>
<td>0.1000</td>
<td>1/2</td>
</tr>
<tr>
<td>0.1001</td>
<td>9/16</td>
</tr>
<tr>
<td>0.1010</td>
<td>10/16</td>
</tr>
<tr>
<td>0.1011</td>
<td>11/16</td>
</tr>
<tr>
<td>0.1100</td>
<td>12/16</td>
</tr>
<tr>
<td>0.1101</td>
<td>13/16</td>
</tr>
<tr>
<td>0.1110</td>
<td>14/16</td>
</tr>
<tr>
<td>0.1111</td>
<td>15/16</td>
</tr>
</tbody>
</table>
DISTRIBUTION OF FLPT NUMBERS

(a) \( b=2, f=4, e=2 \)

\[
E: \begin{array}{cccc}
1 & 2 & 4 & 8 \\
\end{array}
\]

\[
0 \quad 1/2 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\]

(b) \( b=2, f=3, e=3 \)

\[
E: \begin{array}{cccc}
1 & 2 & 4 & 8 \\
\end{array}
\]

\[
0 \quad 1/2 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \quad 20 \quad 24 \quad 28 \quad 32 \quad 40 \quad 48 \quad 56 \quad 64 \quad 80 \quad 96 \quad 112
\]

(c) \( b=4, f=4, e=2 \)

\[
E: \begin{array}{cccc}
1 & 4 & 16 & 64 \\
\end{array}
\]

\[
0 \quad 1/4 \quad 1/2 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad ..., \quad 16 \quad 20 \quad 24 \quad ..., 60
\]

Figure 8.2: EXAMPLES OF DISTRIBUTIONS OF FLOATING-POINT NUMBERS.
SIGNIFICAND: SM with HIDDEN BIT

EXPONENT: BIASED \( E_R = E + B \), \( \min E_R = 0 \) \( \Rightarrow \) \( B = -E_{\text{min}} \)

Symmetric range \( -B \leq E \leq B \) \( \Rightarrow \) \( 0 \leq E_R \leq 2B \leq 2^e - 1 \)

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
• Exact results (inf. precision): \(x, y,\) etc.

• FLPT number representing \(x\) is \(R_{\text{mode}}(x)\) with rounding mode \(\text{mode}\)

• Basic relations:

  1. If \(x \leq y\) then \(R_{\text{mode}}(x) \leq R_{\text{mode}}(y)\)

  2. If \(x\) is a FLPT number then \(R_{\text{mode}}(x) = x\)

  3. If \(F_1\) and \(F_2\) are two consecutive FLPT numbers then for \(F_1 \leq x \leq F_2\) 
     \(x\) is either \(F_1\) or \(F_2\)

Figure 8.3: Relation between \(x, R_{\text{mode}}(x)\), and floating-point numbers \(F_1\) and \(F_2\).
ROUNDING MODES

- Round to nearest (tie to even). $R_{\text{near}}(x)$ is the floating-point number that is closest to $x$.

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

- Round toward zero (truncate). $R_{\text{trunc}}(x)$ is the closest to 0 among $F1$ and $F2$.

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

- Round toward plus infinity. $R_{\text{pinf}}(x)$ is the largest among $F1$ and $F2$

$$R_{\text{pinf}}(x) = F2$$

- Round toward minus infinity. $R_{\text{minf}}(x)$ is the smallest among $F1$ and $F2$

$$R_{\text{minf}}(x) = F1$$
ROUNDING ERRORS

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

\[ ABRE = Rmode(x) - x \]

so that

\[ MABRE = \max_x(|ABRE|) \]

2. The average bias (\( RB \))

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

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 (\( RRE \))

\[ RRE = \frac{Rmode(x) - x}{x} \]
• $x$ 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:

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

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

$$R_{\text{near}}(x) = \left(\lfloor (M_x + \frac{r^{-f}}{2})r^{-f} \rfloor \right) r^{-f}$$

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

\[
\begin{array}{c}
1.100100011101 \\
+ 1 \hfill \\
\hline \\
1.10010010
\end{array}
\]
The absolute error is

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

The maximum absolute error occurs when \( M_d = 1/2 \)

\[ MABRE[R_{\text{near}}] = \frac{r^{-f}}{2} \times b^{E_{\text{max}}} \]

unbiased round to nearest

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

For this mode

\[ RB[R_{\text{near}}] = 0 \]
ROUND TOWARD ZERO (TRUNCATION)

- rounded significand is obtained by discarding $M_d$.

$$R_{zero}(x) = ([M \times r^f])r^{-f} = M_f$$

- The absolute error

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

and

$$MABRE[R_{zero}] \approx r^{-f} \times b^{E_{max}}$$

- Absolute error always negative, the average bias is significant

$$AB[R_{zero}] \approx -\frac{1}{2}r^{-f}$$
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:

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

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

- The addition of \( r^{-f} \) can produce a significand overflow
Figure 8.4: ROUNDDING 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
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 $v = \text{NAN}$ (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
  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)
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 ± 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

\[
M_z^* = \begin{cases} 
  (M_x^* \pm (M_y^*) \times (b^{E_y-E_x})) \times b^{E_x} & \text{if } E_x \geq E_y \\
  ((M_x^*) \times (b^{E_x-E_y}) \pm M_y^*) \times b^{E_y} & \text{if } E_x < E_y 
\end{cases}
\]

\[
E_z = \max(E_x, E_y)
\]

\( E_x - E_y = 4 \)

\[
\begin{align*}
M_x & = 1.xxxxxxxxxxx \\
M_y(2^{(E_y-E_x)}) & = 0.0001yyyyyyyyyyyy \\
\text{-----------------} & \\
z & .zzzzzzzzzzzzzzzz
\end{align*}
\]

2. Normalize significand and update exponent.

3. Round, normalize and adjust exponent.

4. Set flags for special cases.

Digital Arithmetic - Ercegovac/Lang 2003
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):
   
   \[
   \begin{array}{ccc}
   \text{Floating-point op.} & \text{Signs of operands} & \text{Effective operation (EOP)} \\
   \text{ADD} & \text{equal} & \text{add} \\
   \text{ADD} & \text{different} & \text{subtract} \\
   \text{SUB} & \text{equal} & \text{subtract} \\
   \text{SUB} & \text{different} & \text{add} \\
   \end{array}
   \]
4. Normalization of result. Three situations can occur:

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

\[
\begin{align*}
1.10011111 & \\
0.00101011 & \\
\text{ADD} & \quad \text{----------}
\end{align*}
\]

\[
1.11001010
\]

(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

\[
\begin{align*}
1.10011111 & \\
0.0110110 & \\
\text{ADD} & \quad \text{----------}
\end{align*}
\]

\[
10.0000101
\]

\[
\text{NORM} \quad 1.00000101
\]
(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.

\[
\begin{align*}
1.10011111.1001010 & \quad \text{SUB} \quad \underline{---------} \\
0.0000101 & \\
\text{NORM} & = 1.0100000
\end{align*}
\]

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 ± infinity), exponent underflow (special value gradual underflow), inexact, and the special value zero.
Figure 8.5: BASIC IMPLEMENTATION OF FLOWING-POINT ADDITION.

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

exponent overflow/underflow, zero, inexact, NaN
• 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
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
• 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
1. Effective addition:

- Result either normalized or produces an overflow
- Normalization: a 1-bit right shift (if overflow); no left shift required
- \( f + 1 \) 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

\[
\begin{align*}
1.0101110 \\
0.00010101010 \\
\text{ADD} & \quad \text{---------} \\
1.01110001 & \quad T=\text{OR}(010)=1
\end{align*}
\]
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.

\[ f + 2 \] fractional bits of result required

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

\[ f + 3 \] bits required in subtraction (GRT)
Example: After alignment

1.0000011
0.000011011001

SUB ----------------

During alignment compute \( T = \text{OR}(001) = 1 \) resulting in

1.0000011
0.0000110111

SUB -------------

0.1111100001

NORM 1.1111000010
(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

\[
\begin{align*}
1.0000011 & \quad 0.11111001 \\
\text{SUB} & \quad \underline{---------} \\
0.00001101 & \quad 0.00001101 \\
\text{NORM} & \quad 1.10100000
\end{align*}
\]
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.xxxxxxxxxxxxx
  ----f----

• During normalization sticky bit recomputed ( OR of the previous T and the previous R)
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 -> \( \text{rnd} = 1 \) |
| L 1 0 0 0 0 0 | G=1, R=0, T=1 -> \( \text{rnd} = 1 \) (tie case) |
| L 0 x x x x x | G=0 \( \text{rnd} = 0 \) |
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 ± 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 \((E = 0 \text{ and } F \neq 0)\): no hidden 1
  - Set operand of addition to \(E = 1\) and \(0.F\)

• Zero operand \((E = 0 \text{ and } F = 0)\): 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 \(E = 0\)
Figure 8.6: FLPT ADDITION: Critical Path.
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.

Digital Arithmetic - Ercegovac/Lang 2003
8 – Floating-Point Arithmetic
Figure 8.8: DOUBLE-PATH IMPLEMENTATION OF FLOATING-POINT ADDITION.
Figure 8.9: Dependence graph for double-path scheme.
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

   $$M^*_z = M^*_x \times M^*_y$$
   $$E_z = E_x + E_y$$

2. Normalize $M^*_z$ and update exponent
3. Round
4. Determine exception flags and special values
Figure 8.11: BASIC IMPLEMENTATION OF FLOATING-POINT MULTIPLICATION.
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

$$S_z = S_x \oplus S_y$$
4. Normalization: \( 1 \leq M_x, M_y < 2 \), the result in range \([1, 4)\)

Output of multiplier module \( P \):

Bit position: \((-1)0.123\ldots(m-2)(m-1) m (m+1)\ldots(2m-2)\)

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

\[ L = P[m-1], \quad G = P[m], \quad T = \text{OR}(P[m+1], \ldots, P[2m-2]) \]

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

\[ L = P[m-2], \quad G = P[m-1], \quad T = \text{OR}(P[m], \ldots, P[2m-2]) \]
5. Rounding: four rounding modes with guard bit (G) and sticky bit (T)

- Round to nearest

\[ \text{rnd} = G(T) + G(T)'L = G(T + L) \]

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

  \[ \text{rnd} = \text{sgn}'(G + T) \]

  negative infinity

  \[ \text{rnd} = \text{sgn}(G + T) \]
• Overflow: exponent too large;
detected after exponent update;
overflow flag set; result value is ±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 ± infinity;
• zero result set
• Inexact: result inexact if, after normalization, $G \times T = 1$

• 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

control signals, handling of exponents and exceptions not shown

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

\[
\begin{array}{l}
S & s & s & s & s & s & s & s \\
C & c & c & c & c & c & c & c \\
-1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
z & z & z & z & z & z & z & z \\
t & t & t & t & t & t & t & t
\end{array}
\]

\[
z_i = (s_i \oplus c_i)'
\]

\[
t_i = s_{i+1} \cdot c_{i+1}
\]  \hspace{1cm} (8.3)

Compute

\[
w_i = z_i \oplus t_i
\]  \hspace{1cm} (8.4)

Sticky bit is

\[
T = NAND(w_i)
\]  \hspace{1cm} (8.5)
Product $P[-1:2m-2]$

```
-------(m+2)------ ----- (m-2)------
xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx
```

$c_m$

$c_m$ 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.
Bit position: \((-1)(0).123\ldots(m-2)(m-1)m\)

0 \hspace{1em} 1 \hspace{1em} \ldots \hspace{1em} x \hspace{1em} x \hspace{1em} x \hspace{1em} c_m \hspace{1em} 1

(a)

Bit position: \((-1)(0).123\ldots(m-2)(m-1)m\)

1 \hspace{1em} x \hspace{1em} \ldots \hspace{1em} x \hspace{1em} x \hspace{1em} x \hspace{1em} 1 \hspace{1em} c_m

(b)

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

\[ c_m \ c'_m \leftrightarrow (c_m+1)2^{-m} \]

Get P0 and P1 = P0 + 2^{-m}:

\[ \text{P0 ovf} \ x \ x \ x \ x \ x \ x \ x \]

After selection:

\[ \text{P} \quad 1. \ x \ x \ x \ \ldots \ x \ L \]

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:
   \[
   T^* = T * P1[m] * P0[-1]
   \]

   adjustment of the least-significant bit
   \[
   L = P[m - 1](P[m] + T^*)
   \]
## REMOVING $c_m$ FROM CRITICAL PATH

<table>
<thead>
<tr>
<th>carry + sum in pos. $m$</th>
<th>range of $\Sigma$ before pre-add</th>
<th>range of $c_{m-1}$</th>
<th>pre-add $1?$</th>
<th>range of $\Sigma$ after pre-add</th>
<th>range of $c_{m-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[1,3]</td>
<td>[0,1]</td>
<td>NO</td>
<td>[1,3]</td>
<td>[0,1]</td>
</tr>
<tr>
<td>1</td>
<td>[2,4]</td>
<td>[1,2]</td>
<td>YES</td>
<td>[0,2]</td>
<td>[0,1]</td>
</tr>
<tr>
<td>2</td>
<td>[3,5]</td>
<td>[1,2]</td>
<td>YES</td>
<td>[1,3]</td>
<td>[0,1]</td>
</tr>
</tbody>
</table>
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.
--- m --- 2m ---

Product \(x \times y\):
00xx.xxxxx...xxxxxxxxxxx

Addend:
1.xxxxxxxxxxxxx

|--- m-1+4 ---|

(a)

--- 2m ---

Product \(x \times y\):
xx.xxxxx...xxxxxxxxxxx

Addend:
01xxxxxxxxxxxxxx

Shift distance:
|--- 2m-2+1 ---|

(b)

Figure 8.19: Position of addends using bidirectional shift: (a) Maximum left shift. (b) Maximum right shift.
• Position addend $M_w m + 3$ bits to the left of the product
• shift right by the distance

$$d = E_x + E_y - E_w + m + 3$$ (8.6)

• for biased exponent performed as

$$d = E^B_x + E^B_y - E^B_w - B + m + 3$$ (8.7)

• No shift performed for $d \leq 0$ and the maximum shift is $3m + 1$
Initial position:

\[ \begin{array}{cccccccc}
& & & & & & 2m & \\
& & & & & & \text{--- m-1+4 ---} & \\
& & & & & & \text{sticky region} & \\
& & & & & \hline
\end{array} \]

Product x\(y\):

\[00\ldots xxxxxxxxx\]

Addend:

\[1.xxxxxxxxxxxxx\]

\[\text{--- m-1+4 ---}\]

\[\text{--- sticky region ---}\]

(a)

Alignment when Exy = Ew:

\[ \begin{array}{cccccccc}
& & & & & & 2m & \\
& & & & & & \text{--- m-1+4 ---} & \\
& & & & & & \text{sticky region} & \\
& & & & & \hline
\end{array} \]

Product x\(y\):

\[00xx.\ldots xxxxxxxxx\]

Addend:

\[1.xxxxxxxxxxxxx\]

\[-\text{sticky}\]

Shift distance: \[\text{--- m-1+4 ---}\]

(b)
Alignment when Exy - Ew = k:

\[ \begin{array} {c c c c}
\text{Product } x*y: & 00xx.xxxxxx & \ldots & xxxxxxxxxx \\
\text{Addend:} & 1xxxxxxxxxxxxxxxx \\
\text{Shift distance:} & |----- m+3 -------|----- k ----|
\end{array} \]  

(c)

Alignment when Exy - Ew >= 2m-1:

\[ \begin{array} {c c c c}
\text{Product } x*y: & 00xx.xxxxxx & \ldots & xxxxxxxxxx \\
\text{Addend:} & 01xxxxxxxxxxxxxxxxx \\
\text{Shift distance:} & |----- m+3 -------|----- 2m-1 -------|
\end{array} \]  

(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.xxxxxxxxxxxxxxxxxxxxxx

After shift: 1.xxxxxxxxxxxxxLGRT

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

\[ q = \frac{x}{d} \]  

(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

\[ M^*_q = \frac{M^*_x}{M^*_d} \]

\[ E_q = E_x - E_d \]  

(8.9)

2. Normalize \( M^*_q \) and update exponent

3. Round

4. Determine exception flags and special values
control signals, special cases not shown

Figure 8.23: Basic implementation of floating-point division.