DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004

Chapter 8: Solutions to Exercises

– with contributions by Fabrizio Lamberti –

Exercise 8.1

- *Fixed-point representation*

  Planck's constant:

  $$6.63 \times 10^{-27} \rightarrow 0.\underbrace{00000000000000000000000000663}_{29}$$

  Avogadro's number:

  $$6.02 \times 10^{23} \rightarrow \underbrace{602000000000000000000000}_{24}.0$$

  To represent the approximation of Planck's constant 6.63×10^{-27}, 29 radix-10 fractional digit are needed, while representing the approximation of Avogadro's number 6.02×10^{23} requires 24 integer digits. In conclusion, to represent the approximations of both Planck's constant and Avogadro's number in a fixed-point number format, $29 + 54 = 53$ radix-10 digits are needed.

- *Floating-point representation*

  In the considered radix-10 base-10 biased representation for the exponent (such that $E_{biased} = E + 50$), the exponent of both Planck's constant 6.63×10^{-27} and Avogadro's number 6.02×10^{23} can be represented using 2 digits, since $-27+50 = 23$ and $23+50 = 73$. To represent the significands, 3 radix-10 digits are needed. Therefore, to represent the approximations of both Planck's constant and Avogadro's number in a floating-point radix-10 base-10 number format, $3 + 2 = 5$ digits are needed.

Exercise 8.4

Since in a normalized representation the most significant digit of the significand is always different from zero, if we assume a floating point representation with f digits for the significand and e digits for the exponent, the number of values for the first digit of the significand depends on the base that is being considered. For instance, the first four bits (one hexadecimal digit) have 8 values for radix 2, 12 values for radix 4 and 15 values for radix 16. The values that can be represented using the remaining $f-4$ digits of the significand and e digits of the exponent remain unchanged for different bases. Therefore we have

(a) *System A has base* 16 *and system B has base* 2

Since the number of normalized significands for system A is $15 \times 2^{f-4}$ and the number of normalized significands for system B is $8 \times 2^{f-4}$, the ratio between the number of floating-point numbers that are represented by systems A and B is $\frac{15}{8}$.

(b) *System A has base* 16 *and system B has base* 4

Since the number of normalized significands for system A is $15 \times 2^{f-4}$ and the number of normalized significands for system B is $12 \times 2^{f-4}$, the ratio between the number of floating-point numbers that are represented by systems A and B is $\frac{15}{12}$.

Exercise 8.7

In a normalized base-64 floating-point representation, the number of values that can be represented with the first digit is limited to 63. Therefore the number of different significands that can be represented with 48-bit significands is $63 \times 2^{48-6} = 63 \times 2^{42}$.

Exercise 8.10

Notice that for rounding toward zero only f fractional bits are required. For rounding to nearest, one additional bit is required to take into account all discarded bits (since the sticky bit T is not provided, we assume $T = 0$ for ties). For rounding toward plus infinity it is necessary to know the sign as well as when all the bits to be discarded are zero.

s	exp	fraction	guard	round mode
0	00011111	1111111111111	1	
0	00100000	0000000000000		RNE
0	00011111	1111111111111		RNO
0	00011111	1111111111111		RZ
0	00100000	0000000000000		RPINF

s	exp	fraction	guard	round mode
0	11111110	1111111111111	1	
0	11111111	0000000000000		RNE
0	11111110	1111111111111		RNO
0	11111110	1111111111111		RZ
0	11111111	0000000000000		RPINF

s	exp	fraction	guard	round mode
1	11111110	1111111111111	1	
1	11111111	0000000000000		RNE
1	11111110	1111111111111		RNO
1	11111110	1111111111111		RZ
1	11111110	1111111111111		RPINF

Exercise 8.12

Hex-vector	Value
00000000	0.0
80000000	-0.0
A73FF801	$(1.01111111111100000000001)_2 \times 2^{-51}$
A6800000	-1.0×2^{48}
7F7FFFFF	$\left(2 - 2^{-23}\right) \times 2^{127}$
00800000	1.0×2^{-126}
7F800000	$+\infty$
FF800000	$-\infty$
7FC00000	NAN

Exercise 8.16

	Operation	X	Y
A	Add	000110001001111000	000110011100011101
B	Add	000110001001111000	100110011100011101
C	Sub	000110001001111000	000110001001110111
D	Sub	011111110111100011	111111100001010101

(A) *EOP is ADD*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	00110011
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.01001111000
$OUTPUT_{SM-ADD/SUB}$	1.11011101100
$OUTPUT_{L/R1-SHIFTER}$	1.11011101100
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.110111011 00110011
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.110111011 00110011
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.110111011 00110011
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.110111011 00110011
$OUTPUT_{SIGN}$	0

(B) *EOP is SUB*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	00110011
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.01001111000
$OUTPUT_{SM-ADD/SUB}$	1.00111111100
$OUTPUT_{L/R1-SHIFTER}$	1.00111111100
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.001111111 00110011
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.001111111 00110011
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.001111111 00110011
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.001111111 00110011
$OUTPUT_{SIGN}$	1

(C) *EOP is SUB*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	0
$OUTPUT_{MUX}$	00110001
$INPUT_{R-SHIFTER}$	1.001110111
$OUTPUT_{R-SHIFTER}$	1.001110111
$OUTPUT_{SM-ADD/SUB}$	0.000000111
$OUTPUT_{L/R1-SHIFTER}$	1.110000000
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.110000000 00101010
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.110000000 00101010
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.110000000 00101010
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.110000000 00101010
$OUTPUT_{SIGN}$	0

(D) *EOP is ADD*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	11111110
$INPUT_{R-SHIFTER}$	1.001010101
$OUTPUT_{R-SHIFTER}$	0.01001010101
$OUTPUT_{SM-ADD/SUB}$	10.00111100001
$OUTPUT_{L/R1-SHIFTER}$	1.000111100001
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.000111100 11111110
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.000111100 11111110
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.000111101 11111110
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.000111100 11111110
$OUTPUT_{SIGN}$	0

Exercise 8.20

(a) *Determine the delay of the floating-point adder in Fig. 8.5 for single and double precision*

Module	Delay for Single precision	Delay for Double precision
Exponent difference	1.4 ns	1.7 ns
Swap (incl. buffer for control)	0.5 ns	0.5 ns
Right shift	1.0 ns	1.2 ns
Add significands (s+m)	2.5 ns	2.8 ns
LOD	1.5 ns	1.8 ns
Left shift (includes buffer)	1.7 ns	2 ns
Round	1.0 ns	1.2 ns
Right shift (one pos., incl. buf.)	0.5 ns	0.5 ns
Special cases	0.8 ns	0.8 ns
Delay	10.9 ns	12.5 ns

(b) *Pipeline the floating-point adder (for single and double precision) for a clock rate of 200 Mhz (stage delay should not be larger than 80% of the clock cycle)*

Since a clock rate of $200Mhz$ correspond to a clock cycle of 5 ns, stage delay should not be larger that 4 ns. The floating-point adder for single precision could be pipelined as follows (3 stages):

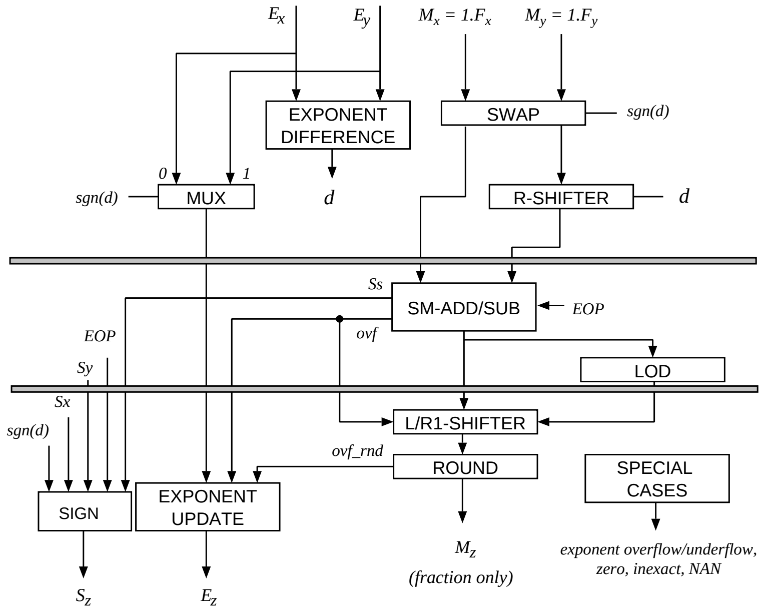

Figure E8.2: Pipelined implementation of the floating-point adder in Figure 8.5 for single precision.

The floating-point adder for double precision could be pipelined as follows (4 stages):

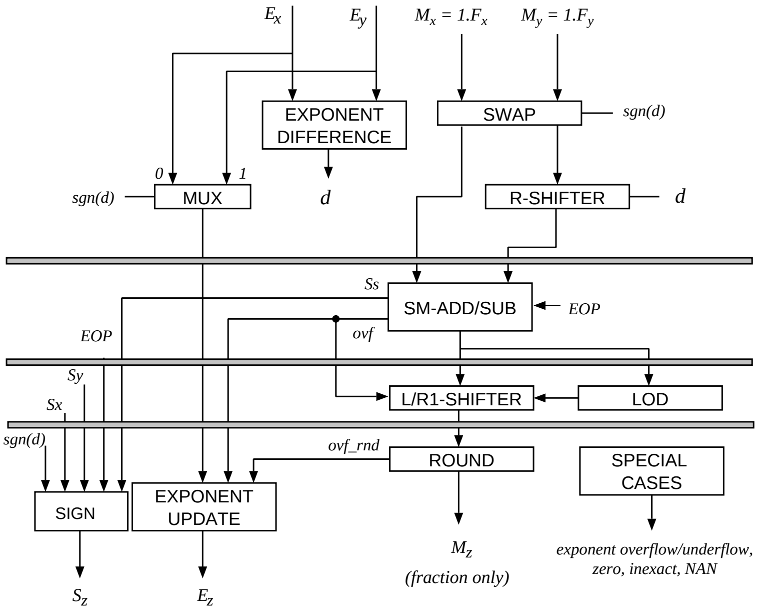

Figure E8.3: Pipelined implementation of the floating-point adder in Figure 8.5 for double precision.

Exercise 8.23

	Operation	X	Y
A	Add	000110001001111000	001001100100011101
B	Sub	000110001001111000	101001100100011101
C	Sub	000110001001111000	000110001001110111
D	Sub	011111110111100011	111111100001010101

(A) *EOP is ADD*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	27
$OUTPUT_{MUX}$	01001100
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.000000000 001
$OUTPUT_{COND.BIT\ INVERT}$	0.000000000 001
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	1.100011101 001 1.100011101 1.100011101
$OUTPUT_{MUX}$	1.100011101
$OUTPUT_{EXPONENT\ UPDATE}$	01001100
$OUTPUT_{SIGN}$	0

(B) *EOP is SUB*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	27
$OUTPUT_{MUX}$	01001100
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.000000000 001
$OUTPUT_{COND.BIT\ INVERT}$	1.111111111 001
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	1.100011101 001 1.100011101 1.100011101
$OUTPUT_{MUX}$	1.100011101
$OUTPUT_{EXPONENT\ UPDATE}$	01001100
$OUTPUT_{SIGN}$	1

(C) *EOP is SUB*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	0
$OUTPUT_{MUX}$	00110001
$INPUT_{R1-SHIFTER}$	1.001110111
$OUTPUT_{R1-SHIFTER}$	1.001110111
$INPUT_{R-SHIFTER}$	
$OUTPUT_{R-SHIFTER}$	
$OUTPUT_{COND.BIT\ INVERT}$	
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	0.000000001
$OUTPUT_{L-SHIFTER}$	1.000000000
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	
$OUTPUT_{MUX}$	1.000000000
$OUTPUT_{EXPONENT\ UPDATE}$	00101000
$OUTPUT_{SIGN}$	0

(D) *EOP is ADD*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	11111110
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001010101
$OUTPUT_{R-SHIFTER}$	0.010010101 010
$OUTPUT_{COND.BIT\ INVERT}$	0.010010101
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	10.001111000 10.001111000 1.000111100 $E_z = 255 \rightarrow M_z = 0$ (overflow)
$OUTPUT_{MUX}$	1.000111100 $E_z = 255 \rightarrow M_z = 0$ (overflow)
$OUTPUT_{EXPONENT\ UPDATE}$	11111111
$OUTPUT_{SIGN}$	0

Exercise 8.25

Operation	X	Y
A	001010101010110011	101111111101110011
B	110011110101110010	111000111011111100

(A) $S_z = 1$

$OUTPUT_{EXP.\ BIASED\ ADDITION}$: 01010101

$OUTPUT_{m\ by\ m\ MULTIPLIER}$: $P[-1, 2m-2]$ = 10.010101000000110001

$P[-1] = 1 \Rightarrow$ normalize by shifting right by one (exponent must be incremented by one).

$OUTPUT_{NORMALIZE}$: 1.00101010 0 01

L GT

Rounding: RNE (round down), RZ (round down), RPINF(round down), RMINF(round up).

$OUTPUT_{EXPONENT\ UPDATE}$: 01010110

(B) $S_z = 0$

$OUTPUT_{EXP.\ BIASED\ ADDITION}$: 11100110

$OUTPUT_{m\ by\ m\ MULTIPLIER}$: $P[-1, 2m-2]$ = 10.100100100000111000

$P[-1] = 1 \Rightarrow$ normalize by shifting right by one (exponent must be incremented by one).

$OUTPUT_{NORMALIZE}$: 1.01001001 0 01

L GT

Rounding: RNE (round down), RZ (round down), RPINF(round up), RMINF(round down).

$OUTPUT_{EXPONENT\ UPDATE}$: 11100111

Exercise 8.29

Operation	X	Y
A	001010101010111000	001010101010010000
B	010000000001100000	001010101011000000

(A) Performing the computation of the multiplication using the basic implementation, the output of m *by* m $MULTIPLIER$ block: is $P[-1, 2m-2]$ = 01.101111011110000000. Since $P[-1] = 0$, $T = 1$. To determine the value of the sticky bit directly from the operands of the multiplier we have to compute the sum of the number of trailing zeros of X and Y (that is, 3 + 4 = 7). Since no normalization is required, we can say that not all the discarded bits are zeros and, as a consequence, $T = 1$, as expected. If we want to compute the value of the sticky bit using the carry-save representation of the second half of the product, we need

$PC[-1 : 2m-3]$ = 00.10111100000000000 and

$PS[-1 : 2m-2]$ = 01.000000011110000000.

$PC[m+1 : 2m-3]$ = 0000000 and

$PS[m+1 : 2m-2]$ = 10000000.

```
10000000 s
00000000 c
01111111 z
0000000  t
0111111  w
```

Therefore, $T = NAND(w_i) = 1$ as expected.

(B) Performing the computation of the multiplication using the basic implementation, the output of *m by m* $MULTIPLIER$ block: is $P[-1, 2m-2]$ =01.101000100000000000. Since $P[-1] = 0$, $T = 0$. To determine the value of the sticky bit directly from the operands of the multiplier we have to compute the sum of the number of trailing zeros of X and Y (that is, 5 + 6 = 11). Since no normalization is required, we can say that all the discarded bits are zeros and, as a consequence, $T = 0$, as expected. If we want to compute the value of the sticky bit using the carry-save reresentation of the second half of the product, we need

$PC[-1 : 2m - 3] =$ 00.01001000000000000 and

$PS[-1 : 2m - 2] =$ 01010110100000000000.

$PC[m + 1 : 2m - 3] =$ 0000000 and

$PS[m + 1 : 2m - 2] =$ 00000000.

```
00000000 s
00000000 c
11111111 z
0000000  t
1111111  w
```

Therefore, $T = NAND(w_i) = 0$ as expected.

Exercise 8.31

(a) *Round to zero*

For rounding to zero, the result is simply truncated to m bits and no additional operation is required.

(b) *Round to plus infinity*

$$R_{pinf} = \begin{cases} M_f + r^{-f} & \text{if} \quad M_d > 0 \text{ and } S = 0 \\ M_f & \text{if} \quad M_d = 0 \text{ or } S = 1 \end{cases}$$

In this case, a 1 should be added to position R (bit m) if $S = 0$ (where S is the sign of the result) and $M_d > 0$ (that is if the sticky bit $T = 1$). However, the result can be either normalized or unnormalized, while the rounding if performed before knowing whether the result is normalized. Therefore, the following quantities have to be calculated:

$$\begin{array}{rcl} P0 & = & PM + \big(c_m + \bar{S} \cdot T\big) \times 2^{-m} \\ P1 & = & PM + \big(c_m + \bar{S} \cdot T + 1\big) \times 2^{-m} \end{array}$$

up to position L (bit $m-1$).

The rounded result is obtained by selecting

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

that is if there is no overflow, select $P0$, while if there is overflow, select $P1$, shift right and truncate at resulting bit L.

Proof

In all cases cm needs to be added to position R (bit m). In case there is no overflow the result is truncated at position L. In the following cases a 1 needs to be added to position L:

- $\bar{S} \cdot T = 1$
- $\bar{S} \cdot T = 0$ and bit of sum in position $R = 1$

Both cases are accounted for by adding $\bar{S} \cdot T + 1$ in position R. In case there is overflow the result is truncated at bit $L-1$ and shifted one bit right. Before shifting a 1 needs to be added to position $L-1$ in the following situations:

- $\bar{S} \cdot T = 1$
- $\bar{S} \cdot T = 0$ and bit of sum in position L or $R = 1$

All cases are accounted for by adding $\bar{S} \cdot T + 1$ in position R and selection $P0+1$ in case of overflow. This is because if $\bar{S} \cdot T = 1$ adding 2 to position R corresponds to adding 1 to position L, so selection $P0 + 1$ corresponds to adding 2 to position L or 1 to $L-1$. On the contrary, if $\bar{S} \cdot T = 0$, if $R = 1$ then when 1 is added to R there is a carry to position L, so 1 is added to L, while if $R = 0$ and $L = 1$ then adding 1 to $P0$ produces a carry to bit $L-1$ so that $P0 + 1$ truncated to bit $l-1$ corresponds to adding 1 to bit $L-1$.

The implementation consists of an array of HAs and FAs, which adds 1 to 3 to position R (that is, add $\overline{\big(c_m \oplus \bar{S} \cdot T\big)}$ to bit R and $\big(c_m + \bar{S} \cdot T\big)$ to bit L), a compound adder producing P0 and $P0 + 1$, The complete process then requires a row of HAs and FAs, a compound adder that computes the sum $P0$ and the sum plus 1 and a multiplexer which selects $P0$ or the normalized (shifted) $P1$ depending whether $P0$ overflows or not.

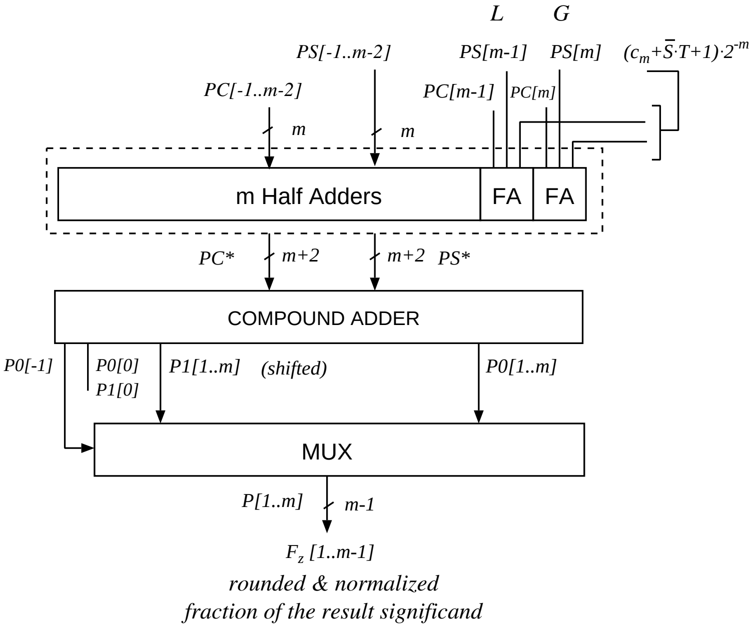

Figure E8.6: Alternative implementation modified to perform round to plus infinity.

(c) *Round to minus infinity*

$$R_{minf} = \begin{cases} M_f + r^{-f} & \text{if} \quad M_d > 0 \text{ and } S = 1 \\ M_f & \text{if} \quad M_d = 0 \text{ or } S = 0 \end{cases}$$

The algorithm for rounding to minus infinity is therefore the same used for rounding to plus infinity, except that $\overline{S}$ should be substituted with S.

Exercise 8.34

X	Y	W
0010101010101110011	1011111111101110011	1100111101011110010

Output of the m by m $MULTIPLIER\ CS$:

PS 01.101000001101101001
PC 00.101100110000000000

Computing $d = -42 + 0 - 31 + m + 3$ we get $d = -60$ (since $m = 10$). Therefore no right shift is needed and the output of the $RIGHT\ SHIFTER$ block is 1.101110010.

```
PS                               01.101000001101101001
PC                               00.101100110000000000
Addend          1.101110010 00
----------------------------------------------------
S               1.101110010 00 01 000100111101101001
C               0.000000000 00 01 010000000000000000
Adder output    1.101110010 00 10 010100111101101001
                          L GR T
```

The output of the adder does not require any realignement/normalization left shift since it is already normalized (leading 1 in the left most position).

Rounding mode	
RNE	Round down
RZ	Round down
RPINF	Round down
RMINF	Round up

The output of the $EXPONENT\ UPDATE$ block is $max(E_x + E_y, E_w) = E_w$. Finally, the result is negative ($S_z = 1$).

Exercise 8.38

	X	Y
A	001010101011010011	101111111110110011
B	110011110001011010	111000111101011101

(A) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 1 \\ E_q &= E_x - E_d + 127 = 01010101 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.011010011}{1.110110011} = \frac{0.1011010011}{0.1110110011}$$

The last conversion is necessary in order to be able to use the quotient-digit selection function of the implementation presented in Section 5.3.1. Since $n = 10$, the number of iterations to be performed is $n+2 = 12$. The initialization is as follows:

$$\text{scaled residual } 2w[0] = 2(x/2) = x,\ q_{computed} = q/2$$

2WS[0]	000.1011010011		
2WC[0]	000.0000000001	$\hat{y}$[0]=0.5	$q_1 = 1$
$-q_1d$	111.0001001100		
2WS[1]	111.0100111100		
2WC[1]	000.0100000100	$\hat{y}$[1]=-1/2	$q_2 = 0$
$-q_2d$	000.0000000000		
2WS[2]	110.0001110000		
2WC[2]	001.0000010000	$\hat{y}$[2]=-1	$q_3 = -1$
$-q_3d$	000.1110110011		
2WS[3]	111.1110100110		
2WC[3]	000.0011000001	$\hat{y}$[3]=0	$q_4 = 1$
$-q_4d$	111.0001001100		
2WS[4]	001.1001010110		
2WC[4]	100.1100010000	$\hat{y}$[4]=-1	$q_5 = -1$
$-q_5d$	000.1110110011		
2WS[5]	011.0111101010		
2WC[5]	011.0001001000	$\hat{y}$[5]=-2	$q_6 = -1$
$-q_6d$	000.1110110011		
2WS[6]	001.0000100010		
2WC[6]	101.1110101000	$\hat{y}$[6]=-1	$q_7 = -1$
$-q_7d$	000.1110110011		
2WS[7]	000.0001110010		
2WC[7]	111.1010001000	$\hat{y}$[7]=-1/2	$q_8 = 0$
$-q_8d$	000.0000000000		
2WS[8]	111.0111110100		
2WC[8]	000.0000000000	$\hat{y}$[8]=-1/2	$q_9 = 0$
$-q_9d$	000.0000000000		
2WS[9]	110.1111101000		
2WC[9]	000.0000000000	$\hat{y}$[9]=-3/2	$q_{10} = -1$
$-q_{10}d$	000.1110110011		
2WS[10]	100.0010110110		
2WC[10]	011.1010000000	$\hat{y}$[10]=-1/2	$q_{11} = 0$
$-q_{11}d$	000.0000000000		
WS[11]	111.1000110110		
WC[11]	000.0100000000	$\hat{y}$[11]=-1/2	$q_{12} = 0$

Since the last residual is negative, the last bit has to be corrected, therefore $q_{12} = -1$. The computed result is then, which however has to be shifted left 1 position since the computed result is $q/2$. The significand before normalization and rounding is then M_q=0.11000011011.

After normalization (M_q=1.1000011011 and E_q=01010100) the result has $f+1$ fractional bits. For round-to-nearest, $2^{-(f+1)}$ has to be added to the result; therefore the rounded significand is

```
1.1000011011 +
0.0000000001
------------
1.1000011100
```

The final result expressed in the IEEE Standard format is

$$Q = 0|01010100|1000011100$$

(B) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 0 \\ E_q &= E_x - E_d + 127 = 11010110 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.001011010}{1.101011101} = \frac{0.1001011010}{0.1101011101}$$

The last conversion is necessary in order to be able to use the quotient-digit selection function of the implementation presented in Section 5.3.1. Since $n = 10$, the number of iterations to be performed is $n+2 = 12$. The initialization is as follows:

$$\text{scaled residual } 2w[0] = 2(x/2) = x, \; q_{computed} = q/2$$

2WS[0]	000.1001011010		
2WC[0]	000.0000000001	$\hat{y}[0]$=0.5	$q_1 = 1$
$-q_1d$	111.0010100010		
2WS[1]	111.0111110010		
2WC[1]	000.0000001000	$\hat{y}[1]$=-1/2	$q_2 = 0$
$-q_2d$	000.0000000000		
2WS[2]	110.1111110100		
2WC[2]	000.0000000000	$\hat{y}[2]$=-1	$q_3 = -1$
$-q_3d$	000.1101011101		
2WS[3]	100.0101010010		
2WC[3]	011.0101010000	$\hat{y}[3]$=-1/2	$q_4 = 0$
$-q_4d$	000.0000000000		
2WS[4]	110.0000000100		
2WC[4]	001.0101000000	$\hat{y}[4]$=-1/2	$q_5 = 0$
$-q_5d$	000.0000000000		
2WS[5]	110.1010001000		
2WC[5]	000.0000000000	$\hat{y}[5]$=-1	$q_6 = -1$
$-q_6d$	000.1101011101		
2WS[6]	100.1110101010		
2WC[6]	010.0000100000	$\hat{y}[6]$=-1	$q_7 = -1$
$-q_7d$	000.1101011101		
2WS[7]	100.0110101110		
2WC[7]	011.0010100000	$\hat{y}[7]$=-1/2	$q_8 = 0$
$-q_8d$	000.0000000000		
2WS[8]	110.1000011100		
2WC[8]	000.1010000000	$\hat{y}[8]$=-1/2	$q_9 = 0$
$-q_9d$	000.0000000000		
2WS[9]	100.0100111000		
2WC[9]	010.0000000000	$\hat{y}[9]$=-1	$q_{10} = -1$
$-q_{10}d$	000.1101011101		
2WS[10]	101.0011001010		
2WC[10]	001.0001100000	$\hat{y}[10]$=-1	$q_{11} = -1$
$-q_{11}d$	000.1101011101		
WS[11] =	100.1111110111		
WC[11] =	010.0010010000	$\hat{y}[11]$=-1	$q_{12} = -1$

The computed result is then

$$q = .10\bar{1}00\bar{1}\bar{1}00\bar{1}\bar{1}\bar{1} = .010110011001$$

which has to be corrected by subtracting one in the last position since the last residual is negative and thus

$$q = .010110011000$$

Moreover, the result has to be shifted left 1 position since the computed result is $q/2$. The significand before normalization and rounding is then $M_q = 0.10110011000$. After normalization ($M_q = 1.0110011000$ and $E_q = 11010101$) the result has $f+1$ fractional bits. For round-to-nearest,$2^{-(f+1)}$ has to be added to the result; therefore the rounded significand is

```
1.0110011000 +
0.0000000001
------------
1.0110011001
```

The final result expressed in the IEEE Standard format is

$$Q = 0|11010101|011001100$$

Exercise 8.41

	X	Y
A	001010101011010011	101111111110110011
B	110011110001011010	111000111101011101

(A) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 1 \\ E_q &= E_x - E_d + 127 = 10011111 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.011010011}{1.110110011}$$

The method requires the calculation of an initial approximation of the reciprocal of the divisor (of 4 bits in this case), which can be obtained, for instance, by means of a lookup table. The initial approximation is 0.1000. The number of iterations to be performed is then

$$m = \left\lceil \log_2\left(\frac{n}{k}\right) \right\rceil = \left\lceil \log_2\left(\frac{9}{4}\right) \right\rceil = 2$$

Since this algorithm is not self-correcting, all multiplications are performed using a 16 bits multiplier. The algorithm is as follows (assuming that multiplications are performed using a floating-point multiplier with rounding to nearest):

1. $P[0] = 0.1000$ (initial approximation of $1/d$)
2. $d[0] = d \times P[0] = 1.110110011000000 \times 2^{-1}$
   $R[0] = x \times P[0] = 1.011010011000000 \times 2^{-1}$
3. $P[1] = 2 - d[0] = 1.000100110100000 \times 2^{0}$
   $d[1] = d[0] \times P[1] = 1.111111010001101 \times 2^{-1}$
   $R[1] = R[0] \times P[1] = 1.100001001010111 \times 2^{-1}$
4. $P[2] = 2 - d[1] = 1.000000010111010 \times 2^{0}$
   $R[2] = R[1] \times P[2] = 1.100001101110001 \times 2^{-1}$

The final q, rounded to the final number of bit, is then 1.100001110. The final result expressed in the IEEE Standard format is

$$Q = 0|01010100|100001110$$

(B) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 0 \\ E_q &= E_x - E_d + 127 = 01111100 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.001011010}{1.101011101}$$

The method requires the calculation of an initial approximation of the reciprocal of the divisor (of 4 bits in this case), which can be obtained, for instance, by means of a lookup table. The initial approximation is 0.1001. The number of iterations to be performed is then

$$m = \left\lceil \log_2 \left(\frac{n}{k} \right) \right\rceil = \left\lceil \log_2 \left(\frac{9}{4} \right) \right\rceil = 2$$

Since this algorithm is not self-correcting, all multiplications are performed using a 16 bits multiplier. he algorithm is as follows (assuming that multiplications are performed using a floating-point multiplier with rounding to nearest):

1. $P[0] = 0.1001$ (initial approximation of $1/d$)
2. $d[0] = d \times P[0] = 1.111001000101000 \times 2^{-1}$
   $R[0] = x \times P[0] = 1.010100101010000 \times 2^{-1}$
3. $P[1] = 2 - d[0] = 1.000011011101100 \times 2^0$
   $d[1] = d[0] \times P[1] = 1.111111101000000 \times 2^{-1}$
   $R[1] = R[0] \times P[1] = 1.011001001111000 \times 2^{-1}$
4. $P[2] = 2 - d[1] = 1.000000001100000 \times 2^0$
   $R[2] = R[1] \times P[2] = 1.011001011111110 \times 2^{-1}$

The final q, rounded to the final number of bit, is then 1.011001100. The final result expressed in the IEEE Standard format is

$$Q = 0|11010101|011001100$$

Exercise 8.44

Round to nearest is performed by adding $2^{-(\mathrm{f}+1)}$ and truncating to f bit. Overflow can occur if $q + 2^{-(f+1)} \geq 2$.

Since the normalized significand is in the range $1 \leq 1.F \leq 2 - 2^{-f}$, the quotient is comprised in the range $\frac{1}{2-2^{-f}} \leq q \leq \frac{2-2^{-f}}{1}$.

Therefore we obtain $q \leq 2 - 2^{-f} \Rightarrow q + 2^{-(f+1)} \leq 2 - 2^{-f} + 2^{-(f+1)} = 2 - 2^{-(f+1)} < 2$. Since $q + 2^{-(f+1)} < 2$, the overflow condition is never satisfied.