2.3. Rounding & normalization rules:
– sticky bits: g, r, s

$$X_M + Y_{M_{cd}} = Z_M$$

Consider Z_M: Cout

$Z_M = Z_m\, Z_{m-1} . Z_{m-2}\, Z_{m-3} \ldots Z_1\, Z_0 \,|\, g\ r\ s$

Normalization:

$Z_{M_n} = 1 \,|\, Z_{m-2_n} \,|\, Z_{m-3_n} \,|\, \ldots \,|\, Z_{1_n} \,|\, Z_{0_n} \,|\,$ R | S — rounding ("fractional part")

Normalization cases:

1) $Z_m = 1 \Rightarrow$ 1-bit RShift: $1_0\, Z_{m-1}\, Z_{m-2} \ldots Z_2\, Z_1 \,|\, Z_0 \,|\, (g$ OR r OR $s)$; Z_E++
2) $Z_m=0, Z_{m-1}=1$ — Z_M is already normalized: $1_0\, Z_{m-2}\, Z_{m-3} \ldots Z_1\, Z_0 \,|\, g \,|\, (r$ OR $s)$;
3) $Z_m=0, Z_{m-1}=0, Z_{m-2}=1$ — 1-bit LShift: $1_0\, Z_{m-3}\, Z_{m-4} \ldots Z_0\, g \,|\, r \,|\, s$; Z_E– –
4) $Z_m=Z_{m-1}=Z_{m-2}=0, Z_{m-3}=1$: $1_0\, Z_{m-4}\, Z_{m-5} \ldots g\, 0 \,|\, 0 \,|\, 0$; Z_E–=2

! if Z_M requires a LShift by 2 or more bits then after appending g, Z_M is completed with bits of 0's additionally R=S=0

Normalization is followed by rounding: use R, S
– using R, S ≡ "fractional part of 2 bits"

①.0001 → 2
1.xy

Rounding mode	$Z_{M_n} \geq 0$	$Z_{M_n} \leq 0$
to 0	(discard R, S)	(discard R, S)
towards −∞	(discard R, S)	if (R OR S) then $Z_{M_n} - 1$
towards +∞	if (R OR S) then $Z_{M_n}+1$	
to nearest even	if (R AND (S OR Z_{0n})) then $Z_{M_n}+1$	if (R AND (S OR Z_{0n})) then $Z_{M_n}-1$

! $Z_M + 1$ can generate Cout ⇒ post-normalization

2.4. Fp. addition/subtraction with rounding
– use a ~~IEEE 754~~ format scaled down: $\frac{9}{2^4} = \frac{9}{16}$
  – 1 bit: sign
  – 3 bits: exponent field, e=3, bias $= 2^{e-1}-1=3$
  – 4 bits: significand → 3 bits for fractional part of significand

$X = 0.5625_{(10)} = 0.1001 \times 2^0 = 1.001 \times 2^{-1}$
$Y = -3.75_{(10)} = -11.11 \times 2^0 = -1.111 \times 2^1$

Packed operands: $X_E = -1 + bias = -1+3 = 2$; $Y_E = +1 + bias = 1+3 = 4$

X: | 0 | 0 1 0 | 0 0 1 | Y: | 1 | 1 0 0 | 1 1 1 |

F.p. addition with rounding algorithm

Step 1: unpack operands: → adding hidden bit
→ check for exceptions (0, ±∞, NaN, ...)

X: | 0 | 0 1 0 | 1 0 0 1 | Y: | 1 | 1 0 0 | 1 1 1 1 |
(X_E, X_M; Y_E, Y_M)

Step 2: Compute exponent difference $d = X_E - Y_E$
→ if $d<0$ ⇒ SWAP $X \leftrightarrow Y$ ⇒ $Z_E = Y_E$
($X_E < Y_E$ ⇒ $|X| < |Y|$)
→ if $d \geq 0$ ⇒ $Z_E = X_E$

$d = 010 - 100 = 2-4 = -2 < 0$
⇒ SWAP $X \leftrightarrow Y$
$Z_E = Y_E = 4$.

X: | 1 | 1 0 0 | 1 1 1 1 | Y: | 0 | 0 1 0 | 1 0 0 1 |

Step 3: if sign(X) ≠ sign(Y) → Complement Y_M of 2
sign(X) = 1, sign(Y) = 0 } complement Y_M of 2

Y: | 0 | 0 1 0 | 1 0 0 1 |
$Y_{Mc2} = 0.111$

Step 4: Align of Y_M: RShift by |d|
→ if Y_M was complemented of 2 in Step 3, then at RShift introduce bits of 1s in the msb Y_M, instead of 0s
→ preserve sticky bits: g, r, s

$Y_{Mc2} = 0.111$ $d=-2$
$Y_{Mc2d} = 1.101 \,|\, 1\ 1\ 0$ (g r s)

Step 5: Add the 2 significands: $Z_M = X_M + Y_{Mcd}$
→ if sign(X) = sign(Y)
→ if Cout is generated from msb of Z_M: PRESERVE IT
→ if sign(X) ≠ sign(Y)
→ if Cout is NOT generated: Z_M is negative
⇒ Complement Z_M of 2
→ if Cout is generated: Z_M is positive
⇒ IGNORE Cout

```
  X_M    =   1.111        +
  Y_Mc2d =   1.101 | 1 1 0
  Z_M    = 1 1.100 | 1 1 0   (cout, >0)
```

Step 6: Pre-normalization: according to the rules in section 2.3 ⇒ determine Z_{M_n}, update Z_E
→ exception checking:
– if $Z_E == Z_{E_{MAX}}$ $(2^3-2=6)$ AND Z_M requires 1-bit RShift
⇒ OVERFLOW
– if $Z_E == Z_{E_{min}}$ (1) AND Z_M requires 1 or more bit LShift
⇒ UNDERFLOW

$Z_M = \text{(cout)}\ 0\ 1.100 \,|\, 1\ 1\ 0$ (g r s)
Rule 2) (already normalized)
$Z_{M_n} = 1.100$ $Z_E = 4$
Z_{0n} ⇒ Z_{M_n} is "even"

Step 7: Calculate R S: according to the same rules in 2.3.
Rule 2): $R = g = 1$
$S = r$ OR $s = 1$ OR $0 = 1$

Step 8: Round the Z_{M_n}: according to the rules in 2.3.
– if rounding generates Cout ⇒ post-normalization: 1-bit RShift
Z_E++ !! check exceptions (like in Step 6)

Consider round to "nearest even" mode
if (R AND (S OR Z_{0n})) then $Z_{M_n}+1$
R = 1
S = 1
Z_{0n} = 0
1 AND (1 OR 0) = 1 ⇒

```
Z_Mn  = 1.100 +
            1
Z_M*  = 1.101
```

no carry out ⇒ no post-normalization

Step 9: Calculate sign of result
– if sign(X) == sign(Y) ⇒ sign(Z) = sign(X)
– if sign(X) != sign(Y) ⇒ initial signs of X, Y (before the SWAP)

SWAP (Step 2)	Complement of 2 (Step 5)	sign(X)	sign(Y)	sign(Z)
YES		+	−	−
YES		−	+	+
NO	YES	+	−	−
NO	YES	−	+	+
NO	NO	+	−	+
NO	NO	−	+	−

SWAP (Step 2) YES: sign(Z) = sign(Y) = − (initial sign)

Step 10: Pack the result
sign(Z) = −; $Z_E = 4$; $Z_M = 1.101$

Z: | 1 | 1 0 0 | 1 0 1 | (Z_E; fractional part of Z_M)

Verify: $X = 0.5625$ $Y = -3.75$
$Z = X + Y = -3.1875$ (infinite precision)

Unpack Z:
Z: | 1 | 1 0 0 | 1 1 0 1 | (Z_E, Z_M)

$Z = (-1)^{sign} \times 2^{Z_E - bias} \times Z_M = (-1) \times 2^{4-3} \times 1.101 = -2 \times 1.101 = -11.01_{(2)} = -3.25$

Chapter III Functional Analysis & Synthesis of Binary Multiplication Devices

3.1. Multiplication methods
Y = multiplicand   X = multiplier
P = X * Y   X = 11, Y = 12

a) paper & pencil

```
        1100 ---- Y
        1011 = x3x2x1x0 --- X
        1100 --- x0·Y·2^0
       1100  --- x1·Y·2^1
      0000   --- x2·Y·2^2
     1100    --- x3·Y·2^3
    10000100 = P = Σ(i=0..3) xi·Y·2^i = 132
```

HW investment: – 2, 4-bit registers for X, Y
– multi-operand adder (CSA) ← latency ↑↑
– multiplicand gating

b) keep fix the partial products

```
        1100 ---------- Y
        1011 = x3x2x1x0 --- X
    00000000 ---------- P0 := 0
        1100 ---------- x0·Y·2^0      |+
    00001100 ---------- P1 := P0 + x0·Y·2^0
       1100  ---------- x1·Y·2^1      |+
    00100100 ---------- P2 := P1 + x1·Y·2^1
      0000   ---------- x2·Y·2^2      |+
    00100100 ---------- P3 := P2 + x2·Y·2^2
     1100    ---------- x3·Y·2^3      |+
    10000100 ---------- P4 := P3 + x3·Y·2^3 = P
```

Iteration step: $P_{i+1} := P_i + x_i \cdot Y \cdot 2^i$, $i \geq 0$, $P_0 := 0$
HW investment: → 2, 4-bit registers for X, Y
→ 8-bit register for result / P_i
→ 8-bit adder
→ LShift for $x_i \cdot Y \cdot 2^i$