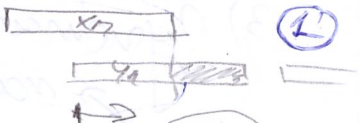
CA C10



3.3. Floating point addition without rounding

- while alligning keeps all the bits of the RShifted mantissa

$X \pm Y = (X_m \pm Y_m \cdot 2^{Y_e - X_e}) \cdot 2^{X_e}$ if $X_e \geq Y_e$

m+2 mantissa
1 sign
1 hidden (significant bit)
m fraction part of significand

adder/subtracter 1

declare registers A[(m+1):0], M[(m+1):0], E_1[(e-1):0], E_2[(e-1):0], E[(e-1):0], A_COUT, ERROR;

declare bus INBUS[(m+e+1):0], OUTBUS[(m+e+1):0];

```
BEGIN:         A_COUT:=0, ERROR:=0,
INPUT:         E1:=INBUS(Xe), A:=INBUS(Xm);
               E2:=INBUS(Ye), M:=INBUS(Ym);
COMPARE:       E:=E1-E2;
ALIGN:         if E<0 then A:=RShift(A), E:=E+1, goto ALIGN;
               if E>0 then M:=RShift(M), E:=E-1, goto ALIGN;
ADD/SUBTRACT:  A:=A±M, E:=max(E1,E2);
OVERFLOW:      if A_COUT==1 then begin
                   if E==E_MAX then goto ERROR,
                   A:=RShift(A), E:=E+1, goto END; ← unconditional jump
               end
ZERO:          if A==0 then E:=0, goto END; ← conditional jump
NORMALIZE:     if isNormalized(A)==1 then goto END,
UNDERFLOW:     if E>Emin then
                   A:=LShift(A), E:=E-1, goto NORMALIZE;
ERROR:         ERROR:=1;
END:
```

Pseudolanguage:

1) declare register → name, width
   - concatenation: A_COUT . A   {A_COUT, A} in Verilog
2) declare bus → name, width   2m+2e+4 bits
   - unified IOBUS m+e+2 bit

3) Synchronous execution
- non-conflicting operations: executed concurrently, separated by , (same clk cycle)
- sequential operations: executed sequentially separated by ; (different clk cycles)

4) := assign operator
- hardwired operations: max, RShift, LShift, isNormalized

5) flow control
- unconditional jump: go to OUS
- conditional jump: if A=0 then E:=0, go to OUS

6) simultaneous read/write to registers/buses:
$A[7]:=M[7] \text{ xor } \boxed{Q[0]}, \boxed{Q[0]}:=0;$

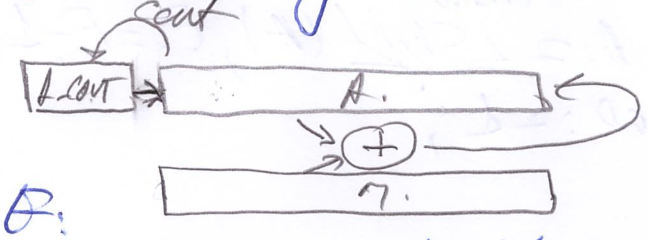
Comments regarding the f.p. algorithm.

A) operands are on m+e+2 bits
- 1 sign
- e exponent
- m+1 significand
  - 1 hidden
  - m fractional part.

m+2 of tradition

- received in packed format from the INBUS

B) registers A and M
- store sign bit + significand (≡ mantissa).
- RShift capabilities for alignment.
- A has LShift capabilities for normalization.
- A is extended by the A_COUT bit (flag)



C) register E:
- has ++ / -- capabilities
- loaded with the maximum of E_1, E_2.

D) flag ERROR
- indicates exceptions: OVERFLOW, UNDERFLOW

3.4. Rounding and normalization rules for f.p. addition. ②

Consider $X_M = \underbrace{1}_{\text{hidden bit}} . \overset{\text{weight } 2^{-1}}{x_{m-2}} \overset{2^{-2}}{x_{m-3}} \dots x_i \dots x_1 \overset{2^{-m+1}}{x_0}$ $\quad X_E \geq Y_E$

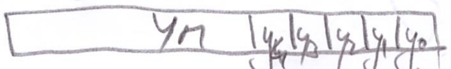
$Y_M = 1 . y_{m-2} y_{m-3} \dots y_i \dots y_1 y_0$

alligmment of Y_M by RShifting with d bits $\quad d = |X_E - Y_E|$

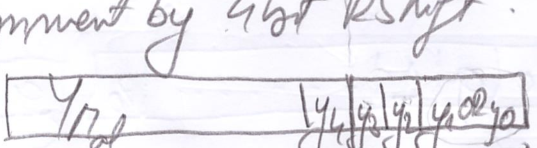
Y_M's alignment
- → with infinite precission: keep all bits of Y_M including bits with weight $< 2^{-m+1}$
- → with finite precision: keep only 3 bits out of all bits having weight $< 2^{-m+1}$

the 3 bits: the sticky bits

- g: guard bit, weight of 2^{-m}
  guards against loss of precision after Y_M's allign.
- r: round bit, weight of 2^{-m-1}
  used for result rounding
- s: sticky bit, weight of 2^{-m-2}
  - obtained as a logic OR of all other less significant bits that were RShifted out of Y_M, except g and r

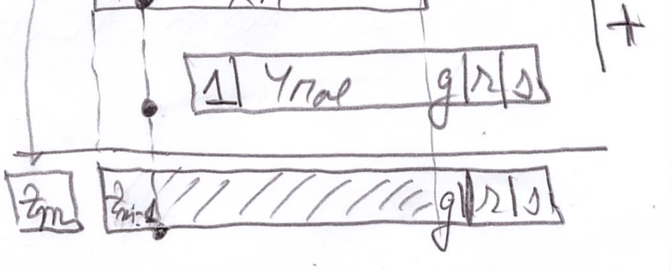


Y_M allignment by 4bit RShift



$Y_{Mal} = Y_M$ after allignment

$\rightarrow Z_M = X_M + Y_{Mal}$



Consider z_r to be.

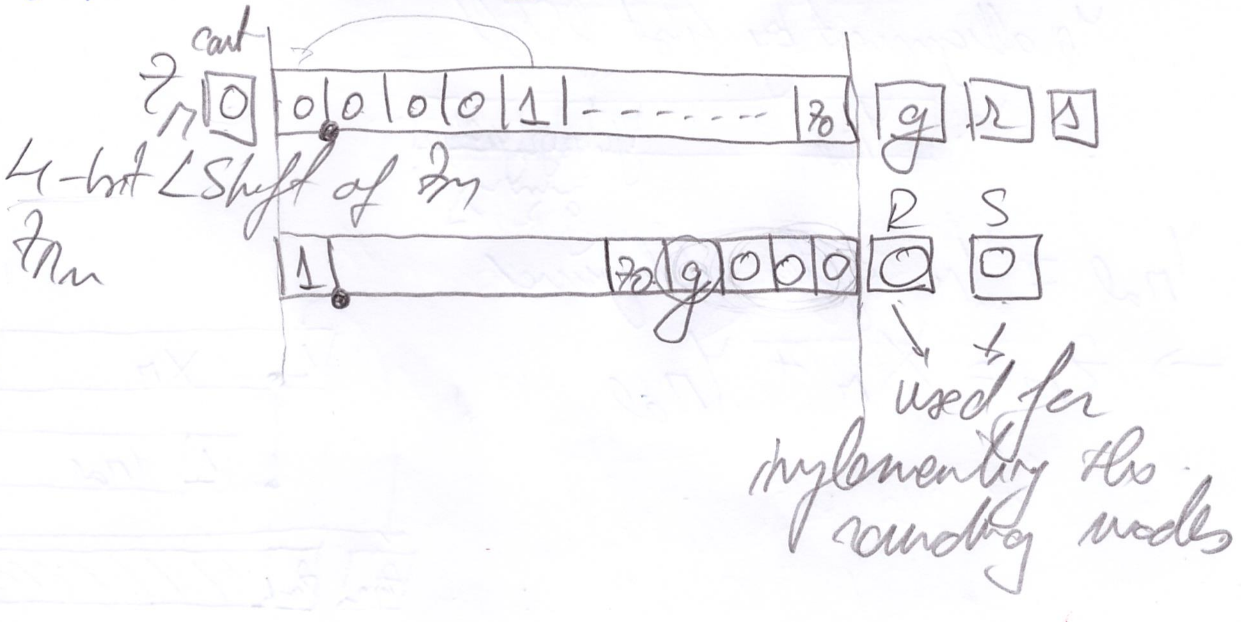
$$z_r = z_m z_{m-1} \bullet z_{m-2} z_{m-3} \dots z_1 z_0 \mid g \; r \; s$$

Normalisation of $z_r \rightarrow z_{r_n}$

only need 2 bits for implementing the 4 rounding modes

$z_{r_n} =$	$1.$	z_{m-2_n}	z_{m-3_n}	---	z_{1_n}	z_{0_n}	R	S
Cases. 1) $z_m = 1$ $\Rightarrow$ 1-bit RShift of z_r (z_E++;)	$1.$	z_{m-1}	z_{m-2}	----	z_2	z_1	z_0	(g OR r OR s)
2) $z_m=0, z_{m-1}=1$ $\Rightarrow$ z_r is already normalised	$1.$	z_{m-2}	z_{m-3}	—	z_1	z_0	g	(r OR s)
3) $z_m=0, z_{m-1}=0, z_{m-2}=1$ $\Rightarrow$ 1-bit LShift of z_r (z_E--;)	$1.$	z_{m-3}	z_{m-2}	—	z_0	g	r	s
4) $z_m=0, z_{m-1}=0, z_{m-2}=0, z_{m-3}=1$ $\Rightarrow$ 2-bit LShift of z_r (z_E -=2;)	$1.$	z_{m-4}	z_{m-5}	---	g	0	0	0

- If a 2bit or more bits LShift is required for z_r's normalisation
  - append g bit to z_r, after z_0
  - Complete all remaining positions with 0s
  - set $R = S = 0$



carry
z_r
0 0 0 0 0 1 ------- z_0 g r s
4-bit LShift of z_r
R S
z_{r_n}
1 z_0 g 0 0 0 0 0
used for implementing the rounding modes

Rounding of z_m: uses R and S bits.
- use 2 "fractional" bits   "X" $= x_{m-1}x_{m-2}\ldots x_1x_0.RS$
to 0 discard RS

Rounding mode	$z_m > 0$	$z_m < 0$
to 0	(discard R and S)	(discards R and S)
towards $-\infty$	(discard R and S)	if (R or S) then $z_m - 1$
towards $+\infty$	if (R or S) then $z_m + 1$	(discard R and S)
to nearest even	if (R and (S or z_{0_m})) then $z_m + 1$	if (R and (S or z_{0m})) then $z_m - 1$

$-1.00005 \rightarrow -2$

$+7.0000001 \rightarrow +8$

$$z_m = \underbrace{1.z_{m-2_m}\ldots z_{1_m}z_{0_m}}\ \Big|\ \underbrace{RS}$$

round to nearest even for positives:

a) if fractional part $< \frac{1}{2}$ $\Rightarrow$ discard R and S
$4.49 \rightarrow 4$

b) if fractional part $= \frac{1}{2}$ and the integer part is an even no $\Rightarrow$ discard R and S
$4.5 \rightarrow 4$

c) if fractional part $= \frac{1}{2}$ and the integer part is an odd no $\Rightarrow +1$ to int. part
$3.5 \rightarrow 4$

d) if fractional part $> \frac{1}{2} \Rightarrow +1$ to integer part
$4.55 \rightarrow 5$

in c) and d) $\equiv$ add 1 unit $\rightarrow$ R, S
using 2 bits of fr. part:

when to +1 to z_m: $\rightarrow R = S = 1$ (.75)
$\rightarrow R = 1, S = 0$ (.5) and $z_{0_m} = 1$

R	S	fr. part
~~0~~	~~0~~	~~(.0)~~
~~0~~	~~1~~	~~(.25)~~
1	0	(.5)
1	1	(.75)

(+1) if integer part is odd.
(+1)

condition for +1 to z_{rn} when rounding to nearest even.

$$\underbrace{R \cdot S}_{(1\quad 1 = 0.75)} + \underbrace{R \cdot \overline{S} \cdot z_{0m}}_{(1\quad 0 = 0.5)} = R \cdot (S + z_{0m})$$

! if rounding generates a carry out
=> post normalisation.