4.10 Signed Integer / Fraction Multiplier

1 Introduction

Multiplication of "*Signed*" Binary Numbers

- A straightforward way for the multiplication of ***signed*** binary numbers:
  1. ***Complement*** the **multiplier** if ***negative***.
  2. ***Complement*** the **multiplicand** if ***negative***.
  3. ***Multiply*** the two ***positive*** binary numbers.
     ⇒ Multiplication of ***unsigned*** binary numbers
  4. ***Complement*** the **product** if it should be ***negative***.
  - Adv.: conceptually simple
  - Disadvs.: requires more hardware and computation time.

J.J. Shann 4-107

- signed FXP binary numbers
  - use 2's complement for negative numbers.
  - 2's complement of a
    - binary integer

$$N^* = \begin{cases} 0 & (\text{if } N \neq 0) \\ 2^n - N & (\text{otherwise}) \end{cases}$$

    - binary fraction

$$F^* = 2 - F$$

Ex. binary representation

Example:

- Represent signed binary fractions +5/8 and −5/8.

* The 2's complement of a binary fraction F:
$F^* = 2 - F$

<Ans.>

+ 5/8	0.101
− 5/8	1.011

$$\begin{array}{r} 10.000 \\ -\ 0.101 \\ \hline 1.011 \end{array}$$

Sign bit
0 for positive &1 for negative

[2] 2's Complement Multiplication

A. 2's Complement Multiplication

- 2's complement multiplication:
  - Use ***2's complement*** for ***negative*** numbers.
- Four cases considered when multiplying signed binary numbers:

Multiplicand	Multiplier
+	+
−	+
+	−
−	−

Case1: Both Multiplicand and Multiplier are "+" (+ × +)

- Standard binary multiplication is used.
- E.g.:

0.1 1 1	(+7/8)	←	Multiplicand
× 0.1 0 1	(+5/8)	←	Multiplier
(0. 0 0)0 1 1 1	(+7/64)	←	*Note*: The proper representation of the
(0.)0 1 1 1	(+7/16)	←	fractional partial products requires extension
0. 1 0 0 0 1 1	(+35/64)		of the sign bit past the binary point, as indicated in parentheses. (Such extension is not necessary in the hardware.)

Case 2: Multiplicand is "–" and Multiplier is "+" (– × +)

- ***Extend the sign bit of the multiplicand*** so that the partial products and final product will have the proper "–" sign.
- E.g.:

1.1 0 1	(−3/8)		
× 0.1 0 1	(+5/8)		
(1. 1 1)1 1 0 1	(−3/64)	←	*Note*: The extension of the
(1.)1 1 0 1	(−3/16)	←	sign bit provides proper representation of the negative products.
1. 1 1 0 0 0 1	(−15/64)		

Case 3: Multiplicand is "+" and Multiplier is "–" (+ × –)

- A negative fraction of the form 1.g has a numeric value –1 + 0.g:
  - The 2's-comp of a positive binary fraction F:

    $F^* = 2 - F = 1.g = 1 + 0.g$

    $\Rightarrow 2 - F = 1 + 0.g \quad \Rightarrow -F = -1 + 0.g$
  - E.g.:

    –5/8: 10.000 – 0.101 = 1.011

    1.011 = –1 + 0.011 = –0.101 = –5/8

  ⇒ For 1.g: treat .g as a positive fraction, but the sign bit as –1.

- For a negative fraction of the form 1.g, it has a numeric value –1 + 0.g.

  ⇒ For 1.g: treat .g as a positive fraction, but the sign bit as –1
- Multiplication procedure for Case 3: (+ × –)
  - Multiplication proceeds in the normal way as we multiply by each bit of the fraction and accumulate the partial products.
  - When we reach the ***negative sign bit of the multiplier***, **add** in the ***2's complement of the multiplicand*** instead of the multiplicand itself.

- E.g.:

0.1 0 1	(+5/8)		
× 1.1 0 1	(−3/8)		
(0.0 0)0 1 0 1	(+5/64)		
(0.)0 1 0 1	(+5/16)		
(0.)0 1 1 0 0 1			
1. 0 1 1	(−5/8)	←	*Note*: The 2's complement of the multiplicand is added at this point.
1. 1 1 0 0 0 1	(−15/64)		

Case 4: Both Multiplicand and Multiplier are "–" (– × –)

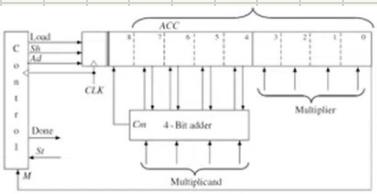
- Case 4 = Case 2 + Case 3
  - At each step, **extend the sign bit** of the ***partial product*** to preserve the proper negative sign.
  - At the final step, **add** in the ***2's complement of the multiplicand***, since the sign bit of the multiplier is negative.
- E.g.:

1.1 0 1	(−3/8)		
× 1.1 0 1	(−3/8)		
(1. 1 1) 1 1 0 1	(−3/64)	←	*Note*: Extend sign bit
(1.)1 1 0 1	(−3/16)		
1. 1 1 0 0 0 1			
0. 0 1 1	(+3/8)	←	Add the 2's complement of the multiplicand.
0. 0 0 1 0 0 1	(+9/64)		

Summary

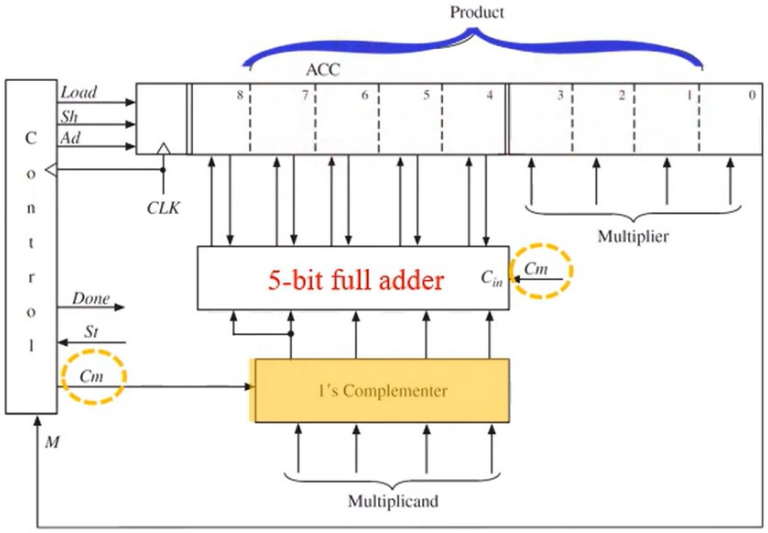
- For multiplying signed 2's complement binary fractions:
  - Procedure: the same as for multiplying positive binary fractions, except
    1. preserve the ***sign*** of the ***partial product*** at each step
    2. if the ***sign*** of the ***multiplier*** is ***negative***, complement the multiplicand before adding it in at the last step
  - Hardware: almost identical to that used for multiplication of positive numbers, except
    * attach a ***complementer*** for the multiplicand

Block Diagram (4×4 Signed Multiplier)



- Hardware required to multiply two 4-bit fractions (includes sign bit):

Multiplication of ***unsigned*** numbers (§4-8)



St: start signal
M: current multiplier bit
Load: load multiplier into the lower 4 bits of ACC and clear the upper 5 bits of ACC
Sh: shift signal, causes ACC to shift right one place w/ sign extension
Ad: add signal, transfers the adder outputs to the upper 5 bits of ACC by the next clock pulse

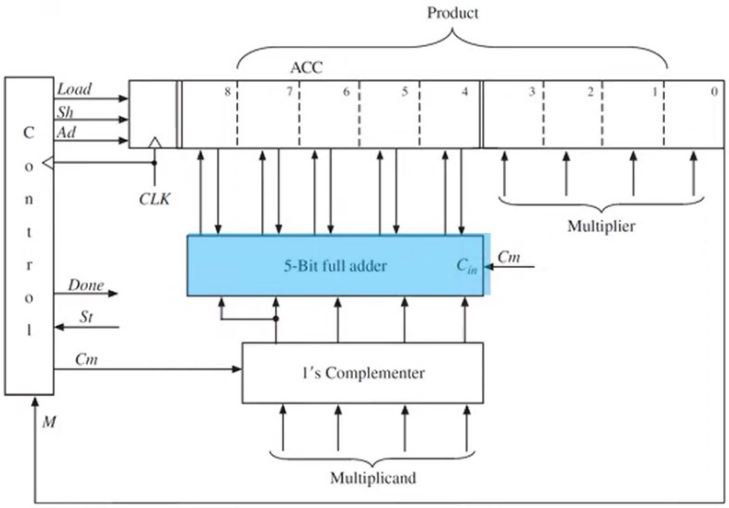
Cm: complement signal, causes the multiplicand (*Mcand*) to be complemented (1's complement) before it enters the adder inputs and the adder adds 1 plus the 1's complement of *Mcand*, i.e, 2's complement of *Mcand*, to the ACC.

* 4×4 signed multiplication ⇒ 7-bit product

1 sign bit + 3 bits

1 sign bit + (3 + 3) bits

St: start signal
M: current multiplier bit
Load: load multiplier into the lower 4 bits of ACC and clear the upper 5 bits of ACC
Sh: shift signal, causes ACC to shift right one place w/ sign extension
Ad: add signal, transfers the adder outputs to the upper 5 bits of ACC by the next clock pulse
Cm: complement signal, causes the multiplicand (*Mcand*) to be complemented (1's complement) before it enters the adder inputs and the adder adds 1 plus the 1's complement of *Mcand*, i.e, 2's complement of *Mcand*, to the ACC.



- a 5-bit adder:
  - is used so the ***sign*** of the sum is not lost due to a carry into the sign bit position
  - The carry out from the last bit of the adder is discarded, since it is a 2's complement addition.

State Graph

- State diagram for the control circuit: (w/o counter)

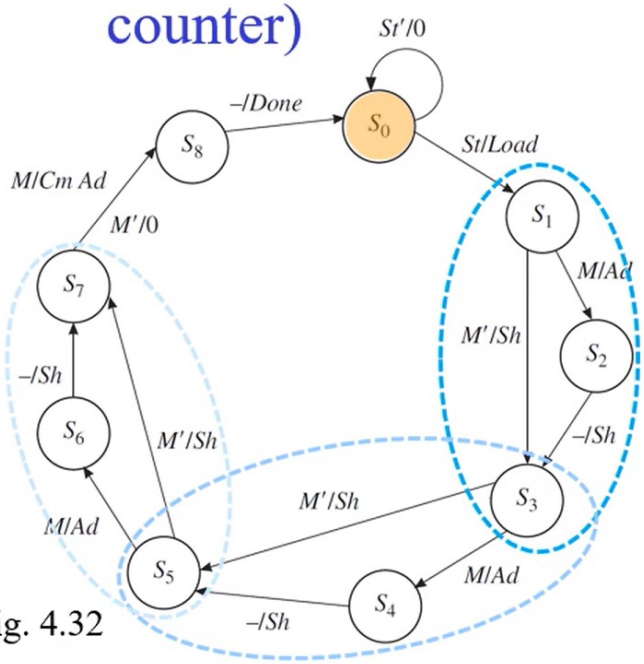
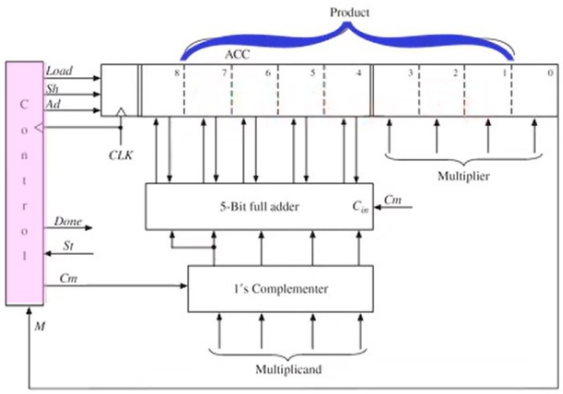


Fig. 4.32



* The ***add*** and ***shift*** operations are done at **two** separate clock times.

Sh: shift ACC right w/ sign extension
Cm: complement multiplicand and set C_{in} of the adder to 1