

Lecture 16

EE 421 / CS 425

Digital System Design

Fall 2024

Shahid Masud

Topics

- Examples: Booth / Radix 4 Multiplication
- Binary Divider Operation
- Binary Divider Circuit
- STG of Divider
- Floating Point Representation (if time permits)
- Floating Point Multiplier – design and operation

Bit-Pair Encoding

Modified Booth Encoding

Radix-4 Encoding

m_{i+1}	m_i	m_{i-1}	Code	BRC_{i+1}	BRC_i	Value	Status	Multiply Actions
0	0	0	0	0	0	0	String of 0s	Shift by 2
0	0	1	1	0	1	+1	End of string of 1s	Add, Shift by 2
0	1	0	2	0	1	+1	Single 1	Add, Shift by 2
0	1	1	3	1	0	+2	End of string of 1s	Shift by 1, Add, Shift by 1
1	0	0	4	1	0	-2	Begin of string of 1s	Shift by 1, Subtract, Shift by 1
1	0	1	5	0	1	-1	Single 0	Subtract, Shift by 2
1	1	0	6	0	1	-1	Begin of string of 1s	Subtract, Shift by 2
1	1	1	7	0	0	0	Midstring of 1s	Shift by 2

Bit-Pair / Radix-4 Recoding of -65_{10}

$-65_{10} =$



Imaginary '0' if LSB=1

$+65 = (01000001)$

2's Complement

$-65 = (10111111)$

m_{i+1}	m_i	m_{i-1}	BRC_{i+1}	BRC_i	Value
0	0	0	0	0	0
0	0	1	0	1	+1
0	1	0	0	1	+1
0	1	1	1	0	+2
1	0	0	1	0	-2
1	0	1	0	1	-1
1	1	0	0	1	-1
1	1	1	0	0	0

$-65_{10} =$



Question of Bit-Pair/Radix-4 Encoding

Express -75_{10} in Radix-4 Encoded format using 8 bits to express the given number

m_{i+1}	m_i	m_{i-1}	BRC_{i+1}	BRC_i	Value
0	0	0	0	0	0
0	0	1	0	1	+1
0	1	0	0	1	+1
0	1	1	1	0	+2
1	0	0	1	0	-2
1	0	1	0	1	-1
1	1	0	0	1	-1
1	1	1	0	0	0

$$+75_{10} = (64+8+2+1) = (0100\ 1011)_2$$

Thus 2's Complement
 $= (1011\ 0101)_2 = -75$

1 0 1 1 0 1 0 1[0]

2; coded 01

2; coded 01

6; coded 0 -1

5; coded 0 -1

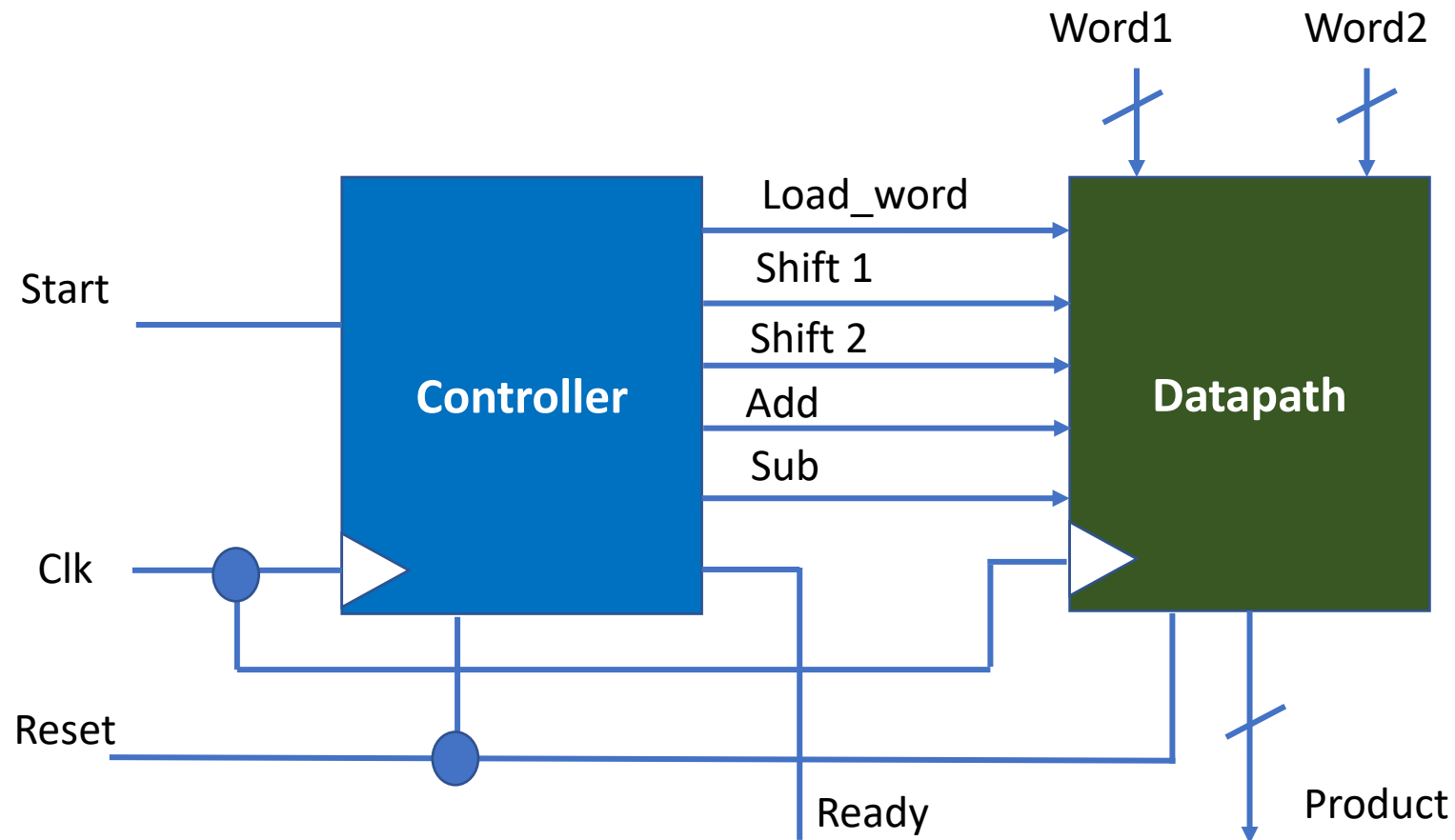
Radix 4 Encoded = 0 -1 0 -1 0 1 0 1

Radix 4 Encoded = 0 1 0 1 0 1

Radix 4 Coding for Multiplication

m_{i+1}	m_i	m_{i-1}	Code	Multiply Actions
0	0	0	0	Shift Left by 2
0	0	1	1	Add Multiplicand, Shift Left by 2
0	1	0	2	Add Multiplicand, Shift Left by 2
0	1	1	3	Shift by 1, Add Multiplicand, Shift by 1
1	0	0	4	Shift by 1, Subtract Multiplicand, Shift by 1
1	0	1	5	Subtract Multiplicand, Shift Left by 2
1	1	0	6	Subtract Multiplicand, Shift Left by 2
1	1	1	7	Shift Left by 2

Data Path Architecture of a Radix 4 Sequential Multiplier



Radix 4 Multiplication – Example 1

Imagine Zero bit if LSB = 1

Show Radix 4 Encoded multiplication of 8 x 9, using 8 bits for both numbers

8 = 0000 1000

9 = 0000 1001

Convert 9 = 0000 1001 to Radix 4 Encoded bits

9 = 0 0 0 0 1 0 0 1 [0]

RECODED

010 → 01

100 → -1 0

001 → 01

000 → 00

8 = Multiplicand

X 9 = Recoded Multiplier

										0	0	0	0	1	0	0	0
										0	0	0	1	-1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	1	1	1	1	1	1	1	1	1	0	0	0	X	X	X	X
0	0	0	0	0	0	0	0	0	1	0	0	0	X	X	X	X	X
													X	X	X	X	X
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

0 1 = Add Multiplicand, Shl2

-1 0 = Shl 1, Sub, Shl1

0 1 = Add, Shl2

0 0 = Only Shl2, No op

Answer = (0100 1000) = +(64 + 8) = +72₁₀

Radix 4 Multiplication – Example 2

Show Radix 4 Encoded multiplication of **68 x -19**, using 8 bits for both numbers

68 = 0100 0100
And 2's Compl is
-68 = 1011 1100

19 = 0001 0011
And 2's Compl is
-19 = 1110 1101

Convert -19 = 1110 1101 to Radix 4 Encoded bits

-19 = 1 1 1 0 1 1 0 1 [0]

Imagine Zero

RECODED
010 → 01
110 → 0-1
110 → 0-1
111 → 00

68 = Multiplicand

X -19 = **Recoded** Multiplier

Result

										0	1	0	0	0	1	0	0
										0	0	0	-1	0	-1	0	1
	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	X	X	
1	1	1	1	1	1	0	1	1	1	1	0	0	X	X	X	X	
													X	X	X	X	X
1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	0	0	

0 1 = Add Multiplicand, Shl2

0 -1 = Sub, Shl2

0 -1 = Sub, Shl2

0 0 = Only Shl2, No op

Take 2's Complement of Result = -(0101 0000 1100) = -(50C) Hex = -(1292)₁₀

Radix 4 Multiplication – Example 3

Show Radix 4 Encoded multiplication of **76 x 55**, using 8 bits for both numbers

76 = 0100 1100
And 2's Compl is
-76 = 1011 0100

55 = 0011 0111
And 2's Compl is
-55 = 1100 1001

Convert 55 = 0011 0111 to Radix 4 Encoded bits

55 = 0 0 1 1 0 1 1 1 [0]

Imagine Zero

RECODED
110 → 0-1
011 → 10
110 → 0-1
001 → 01

76 = Multiplicand

X 55 = Recoded Multiplier

Partial Sum

Partial Sum

Result

										0	1	0	0	1	1	0	0
										0	1	0	-1	1	0	0	-1
	1	1	1	1	1	1	1	1	1	0	1	1	0	1	0	0	
	0	0	0	0	0	0	0	1	0	0	1	1	0	0	X	X	X
Partial Sum	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0
	1	1	1	1	1	1	0	1	1	0	1	0	0	X	X	X	X
Partial Sum	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	0
	0	0	0	0	1	0	0	1	1	0	0	X	X	X	X	X	X
Result	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0

0 -1 = Sub, Shl2

1 0 = Shl1, Add, Shl1

0 -1 = Sub, Shl2

0 1 = Add, Shl2

Answer = 0001 0000 0101 0100 = (4+16+64+4096) = (4180)₁₀

Question?

Perform the following multiplication using Radix 4 Encoding.

Multiplicand = 38, Multiplier = 23 (bits allocated?)

How many Adds and Shifts are required in this multiplication?

How does this compare to a simple binary array multiplier?

Division Operation in Decimal Numbers

Division of $274 \div 13$

				2	1	Quotient
Divisor	1	3	2	7	4	Dividend
		-	-2	6		
				1	4	
			-	1	3	
		Remainder		1	Rem	

Division Operation in Decimal Numbers

Division of $299 \div 15$

			1	9	Quotient	
Divisor	1	5	2	9	9	Dividend
		-	2	8	5	
		Remainder	1	4	Rem	

Decimal Division – another example

		1	0	0	4	Quotient
Divisor	8	8	0	3	5	Dividend
	-	8				
		0	0	3	5	
	-		3	2		
	Remainder				3	

Division of $274 \div 13$

Divisor

Dividend

Remainder

Division Operation in Binary – Example 2

Division of $299 \div 15$

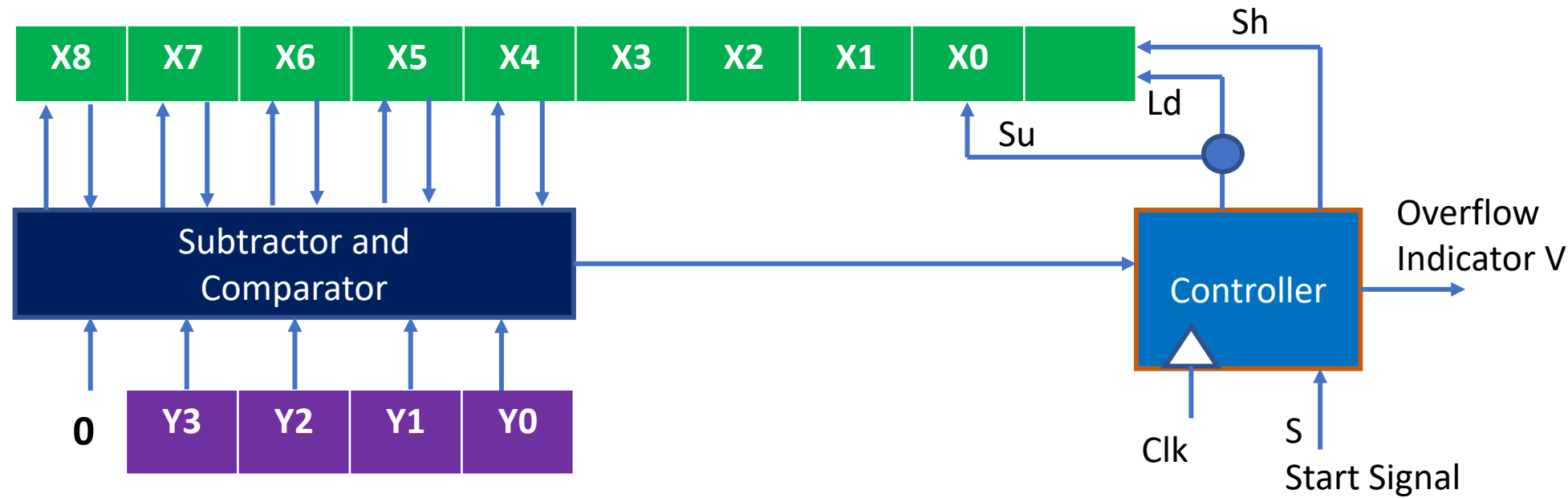
			1	9	
1	5	2	9	9	
	-	2	8	5	
			1	4	Rem

Divisor														Quotient			
1	1	1	1	1	0	0	1	0	1	0	1	1			Dividend		
				-	1	1	1	1									
					0	0	1	1	1	0	1						
				-	0	0	1	1	1	1	1						
					0	0	1	1	1	0	1						
				-					1	1	1	1					
									</								

Division Operation in Binary – Example 3

								Quotient			
Divisor	1	0	1	1	1	0	0	1	0	0	1
				-	1	0	1	1			
					0	1	1	1	0		
				-		1	0	1	1		
					0	0	1	1	1	1	
				-			1	0	1	1	
					Remainder		0	1	0	0	Re

Block Diagram of Sequential Binary Divider



Dividend Register

0	1	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

Divisor Register

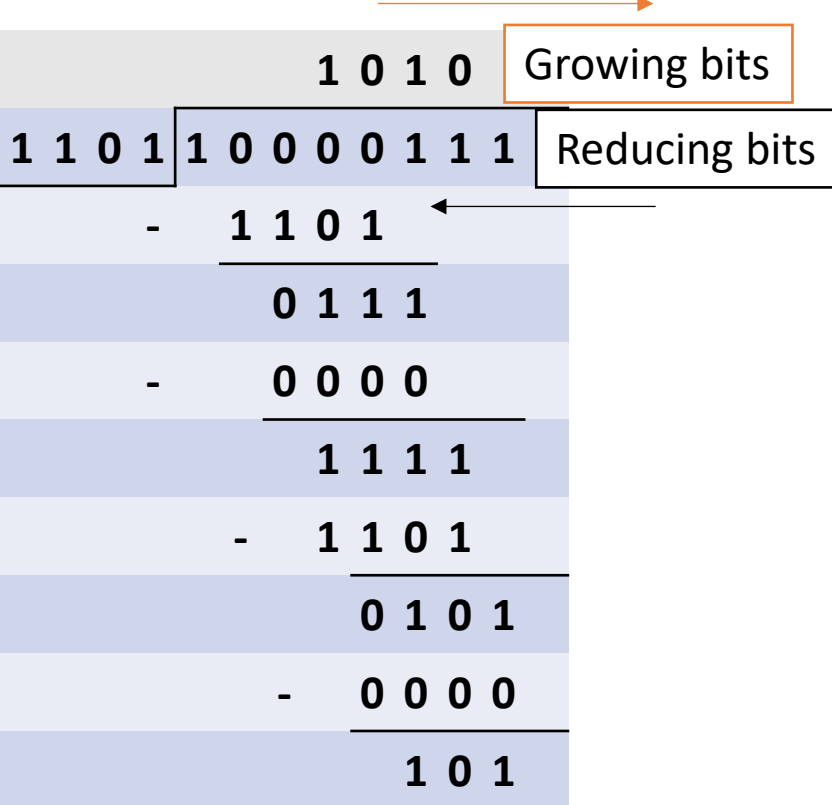
1	1	0	1
---	---	---	---

Overflow V = As a result of a division operation, if the quotient requires more bits than are available for storing quotient

Operation of Sequential Binary Divider

Show binary division

135 ÷ 13



Dividend

1 0 0 0 0 1 1 1 0

1 1 0 1 Divisor

Dividing line between Dividend and Quotient

After the shift, the right most position in dividend register is 'empty'

Subtraction is now carried out. The first quotient digit of 1 is stored in the unused portion of the dividend register

0 0 0 1 1 1 1 1 1

First quotient digit

Next we shift the dividend one place to the left

0 0 1 1 1 1 1 1 0

1 1 0 1

Since subtraction yields negative result so we shift dividend to the left again, and the second quotient bit remains 0

0 1 1 1 1 1 1 0 0

1 1 0 1

Subtraction is now carried out, the third quotient digit of 1 is stored in the unused portion of the dividend register

0 0 0 1 0 1 1 0 1

Third quotient digit

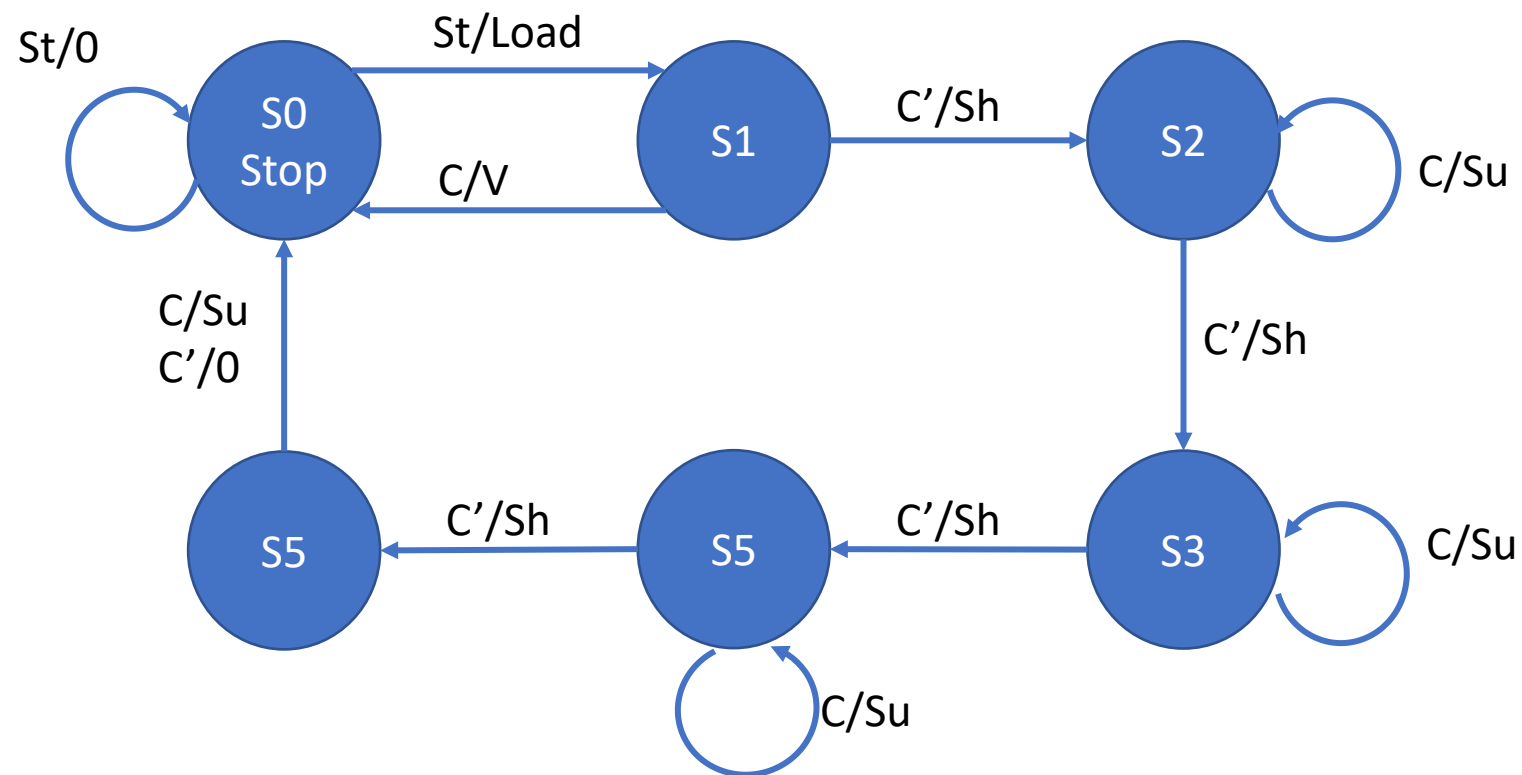
A final shift is done and fourth quotient bit is set to 0

0 0 1 0 1 1 0 1 0

Quotient

Remainder

STG of a Binary Divider



Su = Subtract Signal

C = Comparator Output

**If divisor is greater than 5 leftmost dividend bits (as per given number),
then C=0; otherwise C=1**

Whenever C=1, then subtract signal is generated and quotient bit is set to 1

**Whenever C=0, then subtraction cannot occur without a negative result so a
Shift signal Sh is generated**

Floating Point Arithmetic – Digital Design

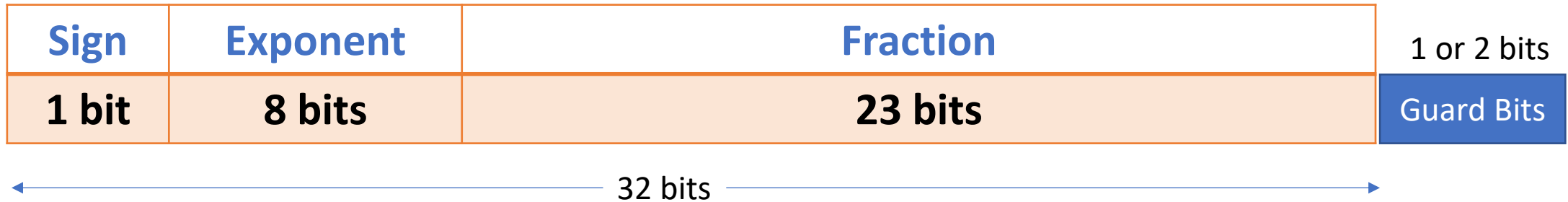
$$N = (-1)^S \times (1+F) \times 2^E$$

E.g. $91.820734 \times 10^{-34}$

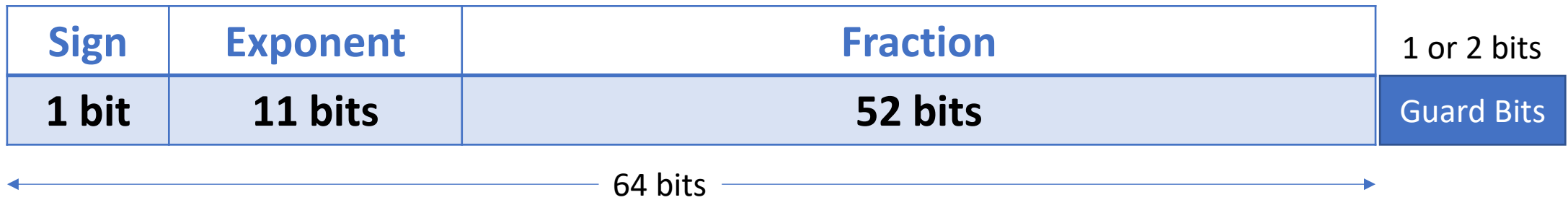
- ❖ A signed-magnitude system for the fractional part and a biased notation for the exponent
- ❖ Three subfields
 - ❖ Sign S
 - ❖ Fraction F (or Significand or Mantissa)
 - ❖ Exponent E
- ❖ Sign bit is 0 for positive numbers, 1 for negative numbers
- ❖ Fractions always start from **1**.xxxx, hence the integer **1** is not written (register has xxxx)
- ❖ Exponent is biased by +127 (add 127 to whatever is in register bits)
- ❖ Normalize: Express numbers in the standard format by shifting of bits and adding / subtracting from Exponent register

IEEE 754 Floating Point Representation

Single Precision IEEE 754



Double Precision IEEE 754



Examples of Floating Point Representation

$-(13.45)_{10}$
 $= (1101.01\ 1100\ 1100\ 1100\ \dots)^2$; this is un-normalized
 $= (1.10101\ 1100\ 1100\ 1100\ 1100\ 1) \times 2^3$; normalized
 Fraction part is 10101 1100 1100 1100 1
 Biased Exponent is $3+127 = 130$
 Sign = 1

5.0345
 $= 101.0000\ 1000\ 1101\ 0100\ 1111\ 110$; this is un-normalized
 $= 1.01\ 0000\ 1000\ 1101\ 0100\ 1111\ 110 \times 2^2$; normalized
 Biased Exponent = $2+127 = 129 = (1000\ 0001)_2$
 Fraction = 01 0000 1000 1101 0100 1111 110
 Sign = 0

Floating Point Multiplication

Consider two floating point numbers:
 $(F_1 \times 2^{E1})$ and $(F_2 \times 2^{E2})$

The product of these two numbers is:
 $= (F_1 \times 2^{E1}) \times (F_2 \times 2^{E2})$
 $= (F_1 \times F_2) \times 2^{(E1+E2)}$
 $= F \times 2^E$

Sign of result depends on Sign of the two numbers

Floating Point Multiplication Steps

1. Normalize the two numbers if not done already
2. The exponents of the Multiplier (E1) and the multiplicand (E2) bits are added and the base value is subtracted from the added result. The subtracted result is put in the exponential field of the result block $\rightarrow E1+E2-\text{bias}$
3. Multiply the two fractions (or significands)
4. S1, the signed bit of the multiplicand is XOR'd with the multiplier signed bit of S2. The result is put into the resultant sign bit.
5. The mantissa of the Multiplier (M1) and multiplicand (M2) are multiplied and the result is placed in the resultant field of the mantissa (truncate/round the result for 24 bits) $\rightarrow M1 * M2$
6. If the product is 0, adjust the proper representation of answer to 0
7. If the product fraction is too big, normalize by shifting it right and incrementing the exponent
8. If the product fraction is too small, normalize by shifting left and decrementing the exponent
9. Round to appropriate number of bits. If rounding results in loss of normalization, then first normalize and then do the rounding
10. If an exponent underflow (below -127) or overflow (above +127) occurs then generate an error condition

Division Examples

- Try using 2's Complement Add instead of Sub in Division operations