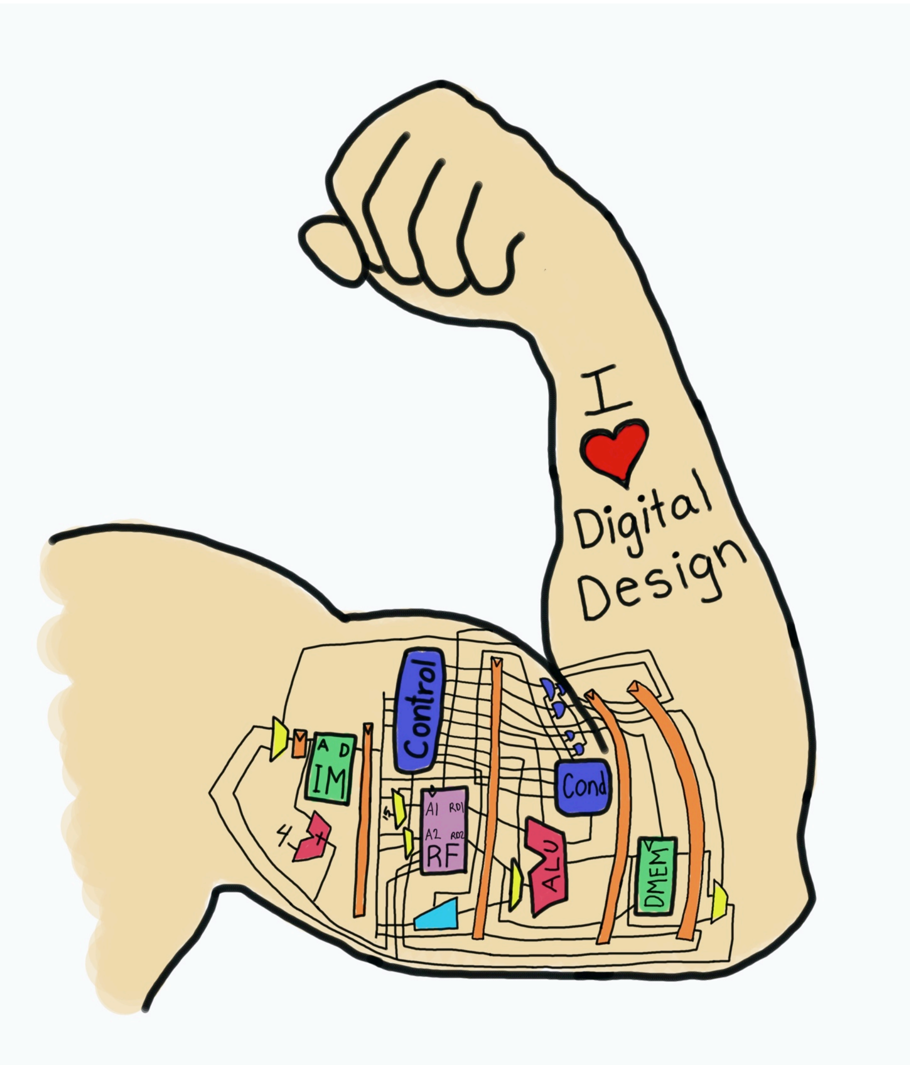
E85 Digital Design & Computer Engineering



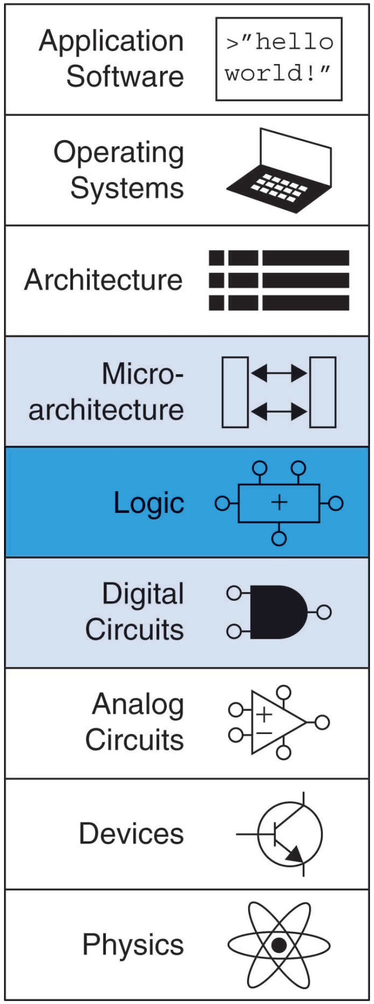
Lecture 9:

Shift, Mult, Div
Fixed & Floating Point

HARVEY MUDD COLLEGE

Lecture 9

- **Shifters**
- **Multipliers**
- **Dividers**
- **Fixed Point Numbers**
- **Floating Point Numbers**



Shifters

Logical shifter: shifts value to left or right and fills empty spaces with 0's

- Ex: 11001 >> 2 =
- Ex: 11001 << 2 =

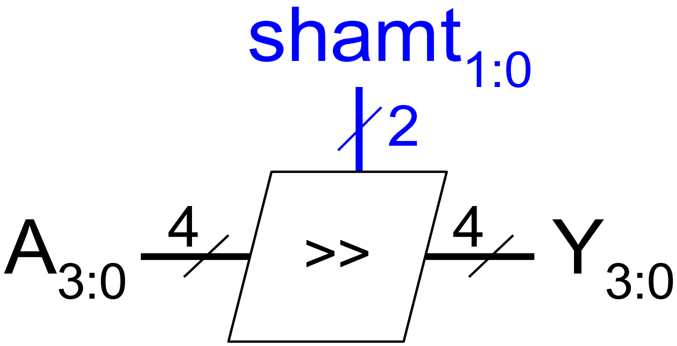
Arithmetic shifter: same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb)

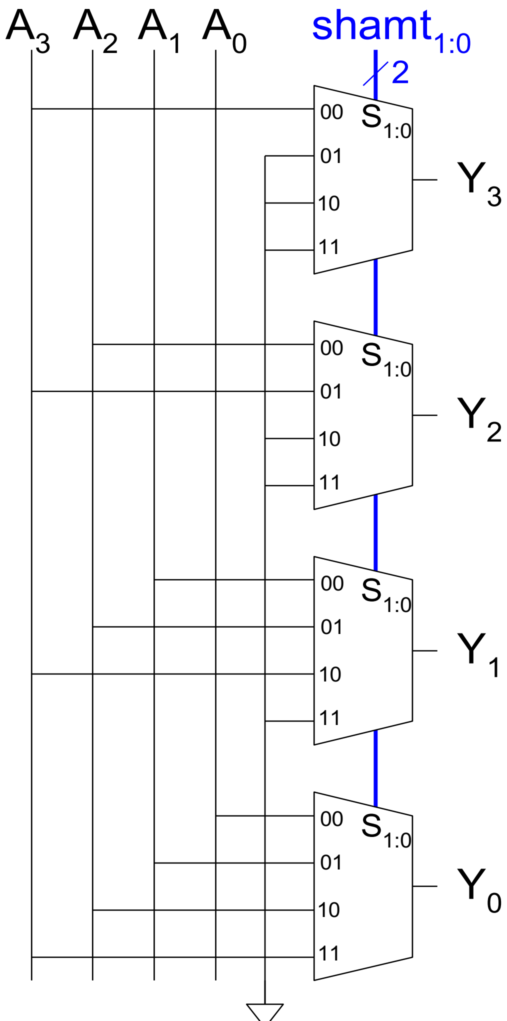
- Ex: 11001 >>> 2 =
- Ex: 11001 <<< 2 =

Rotator: rotates bits in a circle, such that bits shifted off one end are shifted into the other end

- Ex: 11001 ROR 2 =
- Ex: 11001 ROL 2 =

Shifter Design





Shifters as Multipliers, Dividers

- $A << N = A \times 2^N$
  - **Example:** 00001 << 2 = 00100 ($1 \times 2^2 = 4$)
  - **Example:** 11101 << 2 = 10100 ($-3 \times 2^2 = -12$)
- $A >>> N = A \div 2^N$
  - **Example:** 01000 >>> 2 = 00010 ($8 \div 2^2 = 2$)
  - **Example:** 10000 >>> 2 = 11100 ($-16 \div 2^2 = -4$)

Multipliers

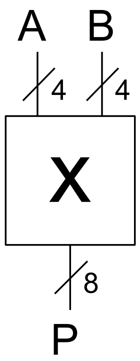
- **Partial products** formed by multiplying a single digit of the multiplier with multiplicand
- **Shifted** partial products **summed** to form result

Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial	0101
+ 920	products	0101
9660		0101
		+ 0000
	result	0100011

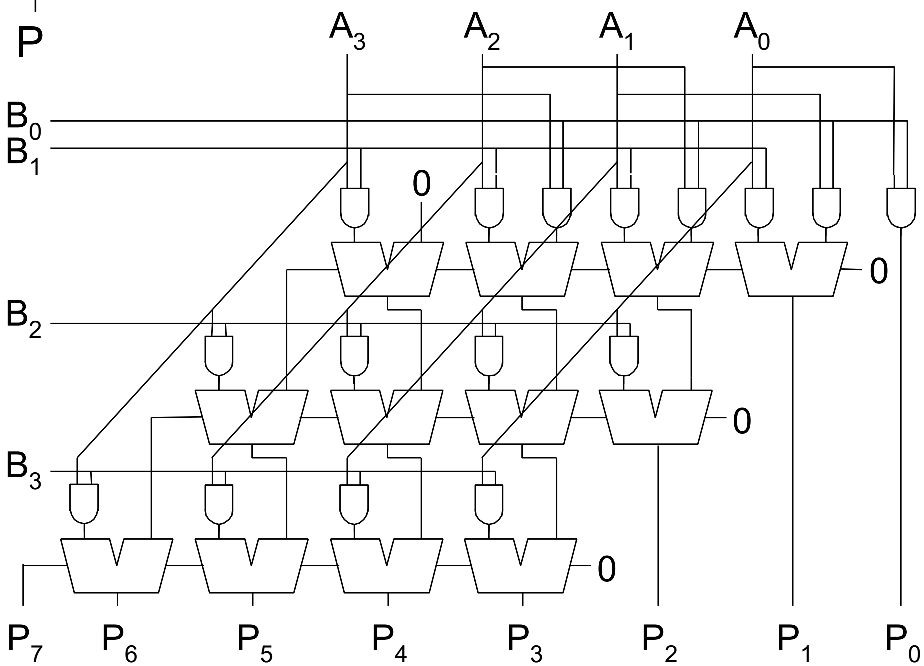
230 x 42 = 9660

5 x 7 = 35

4 x 4 Multiplier



				A_3	A_2	A_1	A_0
			x	B_3	B_2	B_1	B_0
				A_3B_0	A_2B_0	A_1B_0	A_0B_0
			A_3B_1	A_2B_1	A_1B_1	A_0B_1	
		A_3B_2	A_2B_2	A_1B_2	A_0B_2		
+	A_3B_3	A_2B_3	A_1B_3	A_0B_3			
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0



Dividers

$$A/B = Q + R/B$$

Decimal Example: **2584/15 = 172 R4**

Dividers

A/B = Q + R/B

Decimal Example: 2584/15 = 172 R4

Long-Hand:

```
       172  R4
    ________
15 | 2584
    -15
    ----
     108
    -105
    -----
       34
      -30
      ---
        4
```

Dividers

$$A/B = Q + R/B$$

Decimal Example: **2584/15 = 172 R4**

Long-Hand:

$$
\begin{array}{r}
172 \ \text{R4} \\
15\,\overline{)\,2584} \\
\underline{-15} \\
108 \\
\underline{-105} \\
34 \\
\underline{-30} \\
4
\end{array}
$$

Long-Hand Revisited:

$$
\begin{array}{r}
0002 \\
\underline{-\ \ 15} \\
-13
\end{array}
\qquad \underset{3}{\underline{0}}\ \underset{2}{\underline{}}\ \underset{1}{\underline{}}\ \underset{0}{\underline{}}
$$

$$
\begin{array}{r}
0025 \\
\underline{-\ \ 15} \\
10
\end{array}
\qquad \underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{}}\ \underset{0}{\underline{}}
$$

$$
\begin{array}{r}
0108 \\
\underline{-\ 105} \\
3
\end{array}
\qquad \underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{7}}\ \underset{0}{\underline{}}
$$

$$
\begin{array}{r}
0034 \\
\underline{-\ \ 30} \\
4
\end{array}
\qquad \underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{7}}\ \underset{0}{\underline{2}}
$$

Dividers

A/B = Q + R/B

Decimal: 2584/15 = 172 R4   **Binary: 1101/0010 = 0110 R1**

$\begin{array}{r} 0002 \\ -\ \ 15 \\ \hline -13 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{\ \ }}\ \underset{1}{\underline{\ \ }}\ \underset{0}{\underline{\ \ }}$	$\begin{array}{r} 0001 \\ -0010 \\ \hline 1111 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{\ \ }}\ \underset{1}{\underline{\ \ }}\ \underset{0}{\underline{\ \ }}$
$\begin{array}{r} 0025 \\ -\ \ 15 \\ \hline 10 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{\ \ }}\ \underset{0}{\underline{\ \ }}$	$\begin{array}{r} 0011 \\ -0010 \\ \hline 0001 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{\ \ }}\ \underset{0}{\underline{\ \ }}$
$\begin{array}{r} 0108 \\ -\ 105 \\ \hline 3 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{7}}\ \underset{0}{\underline{\ \ }}$	$\begin{array}{r} 0010 \\ -0010 \\ \hline 0000 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{1}}\ \underset{0}{\underline{\ \ }}$
$\begin{array}{r} 0034 \\ -\ \ 30 \\ \hline 4 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{7}}\ \underset{0}{\underline{2}}$	$\begin{array}{r} 0001 \\ -0010 \\ \hline 1111 \end{array}$	$\underset{3}{\underline{0}}\ \underset{2}{\underline{1}}\ \underset{1}{\underline{1}}\ \underset{0}{\underline{0}}$ R1

Divider Algorithm

$A/B = Q + R/B$

Binary: 1101/10 = 0110 R1

$R' = 0$

for i = N-1 to 0

  $R = \{R' << 1, A_i\}$

  $D = R - B$

  if $D < 0$, $Q_i = 0$; $R' = R$

  else $Q_i = 1$; $R' = D$

$R = R'$

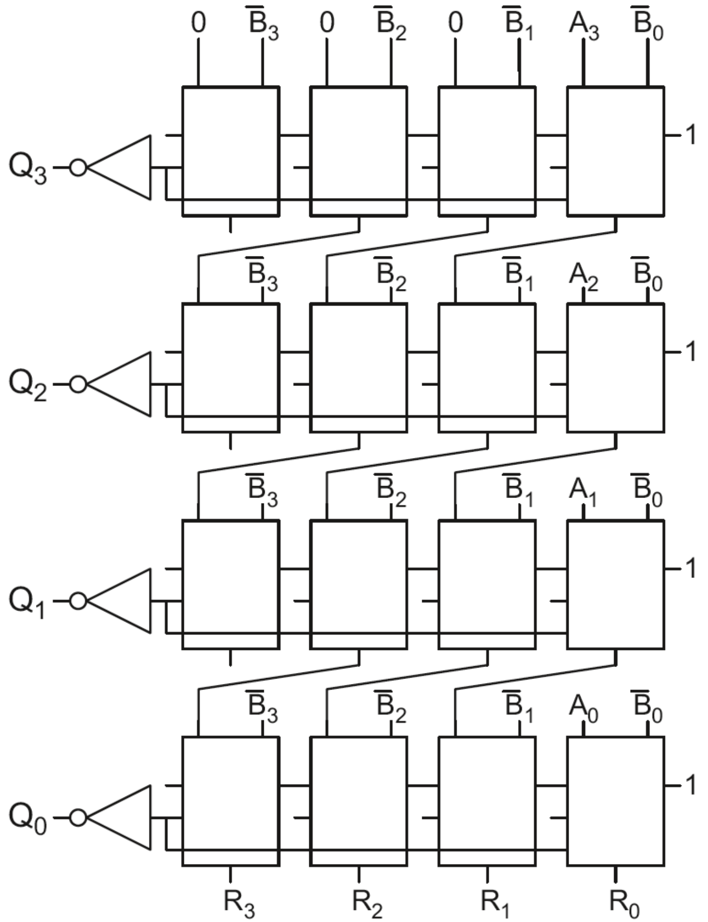
$$\begin{array}{r} 0001 \\ -\,0010 \\ \hline 1111 \end{array} \qquad \underset{3}{0}\ \underset{2}{\_}\ \underset{1}{\_}\ \underset{0}{\_}$$

$$\begin{array}{r} 0011 \\ -\,0010 \\ \hline 0001 \end{array} \qquad \underset{3}{0}\ \underset{2}{1}\ \underset{1}{\_}\ \underset{0}{\_}$$

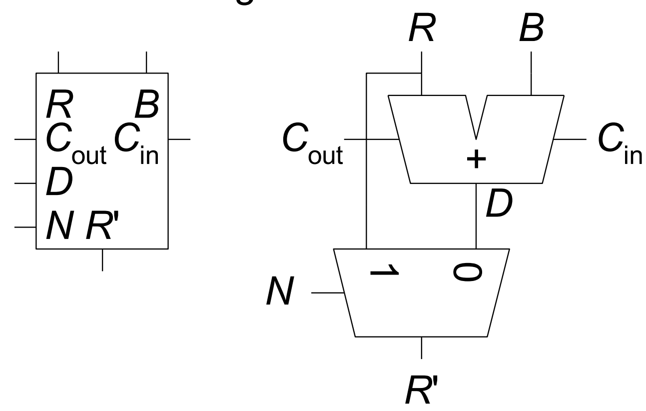
$$\begin{array}{r} 0010 \\ -\,0010 \\ \hline 0000 \end{array} \qquad \underset{3}{0}\ \underset{2}{1}\ \underset{1}{1}\ \underset{0}{\_}$$

$$\begin{array}{r} 0001 \\ -\,0010 \\ \hline 1111 \end{array} \qquad \underset{3}{0}\ \underset{2}{1}\ \underset{1}{1}\ \underset{0}{0}\ \ \text{R1}$$

4 x 4 Divider



Legend



Division: $A/B = Q + R/B$

$R' = 0$

for i = N-1 to 0

  $R = \{R' \ll 1, A_i\}$

  $D = R - B$

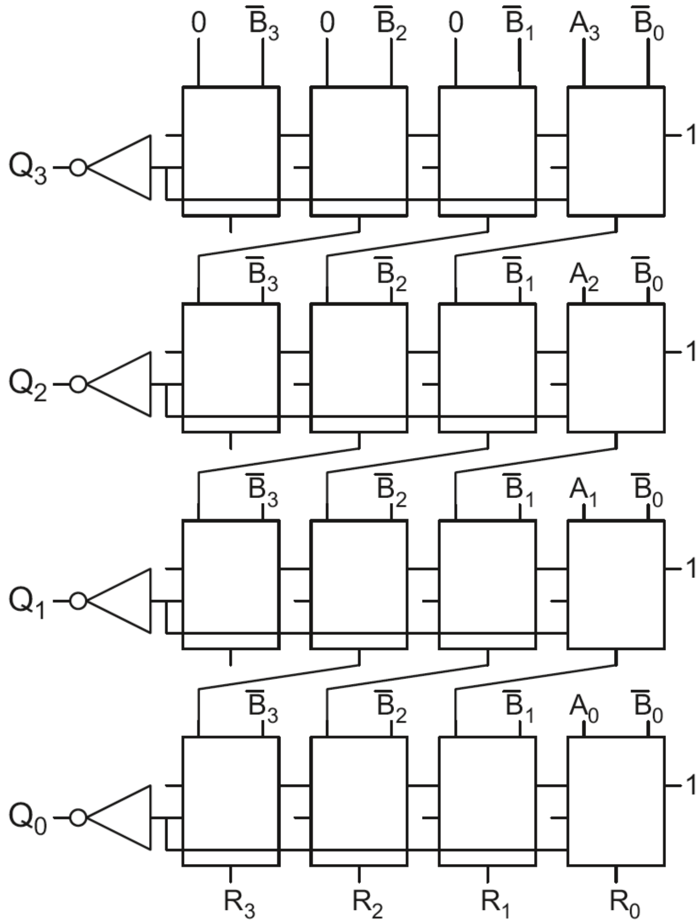
  if $D < 0$, $Q_i=0$, $R'=R$

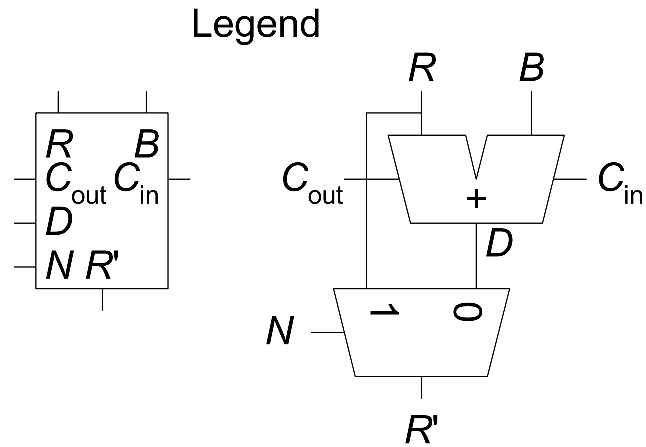
  else $Q_i=1$, $R'=D$

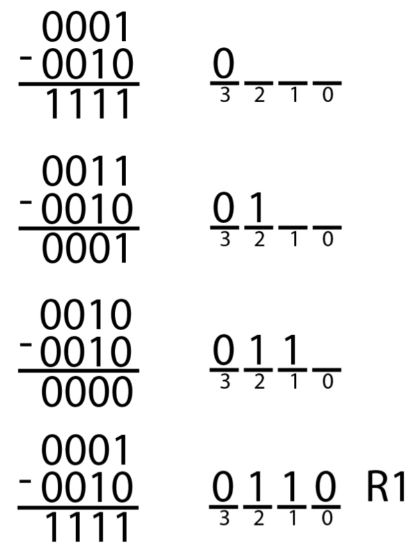
$R=R'$

Each row computes one iteration of the division algorithm.

4 x 4 Divider







Each row computes one iteration of the division algorithm.

Number Systems

Numbers we can represent using binary representations

- **Positive numbers**
  - Unsigned binary
- **Negative numbers**
  - Two's complement
  - Sign/magnitude numbers

What about **fractions**?

Numbers with Fractions

Two common notations:

- **Fixed-point:** binary point fixed
- **Floating-point:** binary point floats to the right of the most significant 1

Fixed-Point Numbers

- 6.75 using 4 integer bits and 4 fraction bits:

$$01101100$$

$$0110.1100$$

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$$

  - Binary point is implied
  - The number of integer and fraction bits must be agreed upon beforehand

Fixed-Point Number Example

- Represent 7.5_{10} using 4 integer bits and 4 fraction bits.

Signed Fixed-Point Numbers

- **Representations:**
  - Sign/magnitude
  - Two's complement
- **Example:** Represent -7.5_{10} using 4 integer and 4 fraction bits
  - **Sign/magnitude:**
  - **Two's complement:**

    1. +7.5:
    2. Invert bits:
    3. Add 1 to lsb: + 1

ELSEVIER

Floating-Point Numbers

- Binary point floats to the right of the most significant 1
- Similar to decimal scientific notation

- For example, write 273_{10} in scientific notation:

$$\mathbf{273 = 2.73 \times 10^2}$$

- In general, a number is written in scientific notation as:

$$\pm \mathbf{M} \times \mathbf{B^E}$$

  - **M** = mantissa
  - **B** = base
  - **E** = exponent
  - In the example, M = 2.73, B = 10, and E = 2

Floating-Point Numbers

- **Example:** represent the value 228_{10} using a 32-bit floating point representation

We show three versions – final version is called the **IEEE 754 floating-point standard**

Floating-Point Representation 1

1. Convert decimal to binary

   $228_{10} = 11100100_2$

2. Write the number in "binary scientific notation":

   $11100100_2 = 1.11001_2 \times 2^7$

3. Fill in each field of the 32-bit floating point number:
   - The sign bit is positive (0)
   - The 8 exponent bits represent the value 7
   - The remaining 23 bits are the mantissa

1 bit	8 bits	23 bits
0	00000111	11 1001 0000 0000 0000 0000
Sign	**Exponent**	**Mantissa**

Floating-Point Representation 2

- First bit of the mantissa is always 1:
  - $228_{10} = 11100100_2 = \mathbf{1.11001} \times 2^7$
- So, no need to store it: *implicit leading 1*
- Store just fraction bits in 23-bit field

1 bit	8 bits	23 bits
0	00000111	110 0100 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

Floating-Point Representation 3

- *Biased exponent*: bias = 127 (01111111_2)
  - Biased exponent = bias + exponent
  - Exponent of 7 is stored as:

$$127 + 7 = 134 = 0x10000110_2$$

- The **IEEE 754 32-bit floating-point representation** of 228_{10}

1 bit	8 bits	23 bits
0	10000110	110 0100 0000 0000 0000 0000
Sign	**Biased Exponent**	**Fraction**

in hexadecimal: **0x43640000**

Floating-Point Example

Write -58.25_{10} in floating point (IEEE 754)

Floating-Point Example

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   58.25_{10} = **111010.01_2**

2. Write in binary scientific notation:

   1.1101001×2^5

3. Fill in fields:

   Sign bit: 1 (negative)
   8 exponent bits: (127 + 5) = 132 = **10000100_2**
   23 fraction bits: **110 1001 0000 0000 0000 0000**

1 bit	8 bits	23 bits
1	100 0010 0	110 1001 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

in hexadecimal: **0xC2690000**

Floating-Point: Special Cases

Number	Sign	Exponent	Fraction
0	X	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	X	11111111	non-zero

Floating-Point Precision

- **Single-Precision:**
  - 32-bit
  - 1 sign bit, 8 exponent bits, 23 fraction bits
  - bias = 127

- **Double-Precision:**
  - 64-bit
  - 1 sign bit, 11 exponent bits, 52 fraction bits
  - bias = 1023

Floating-Point: Rounding

- **Overflow:** number too large to be represented
- **Underflow:** number too small to be represented
- **Rounding modes:**
  - Down
  - Up
  - Toward zero
  - To nearest
- **Example:** round 1.100101 (1.578125) to only 3 fraction bits
  - Down: 1.100
  - Up: 1.101
  - Toward zero: 1.100
  - To nearest: 1.101 (1.625 is closer to 1.578125 than 1.5 is)

Floating-Point Addition

1. Extract exponent and fraction bits
2. Prepend leading 1 to form mantissa
3. Compare exponents
4. Shift smaller mantissa if necessary
5. Add mantissas
6. Normalize mantissa and adjust exponent if necessary
7. Round result
8. Assemble exponent and fraction back into floating-point format

Floating-Point Addition Example

Add the following floating-point numbers:

0x3FC00000

0x40500000

Floating-Point Addition Example

1. **Extract exponent and fraction bits**

1 bit	8 bits	23 bits
0	01111111	100 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

1 bit	8 bits	23 bits
0	10000000	101 0000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

For first number (N1): S = 0, E = 127, F = .1

For second number (N2): S = 0, E = 128, F = .101

2. **Prepend leading 1 to form mantissa**

N1: 1.1

N2: 1.101

Floating-Point Addition Example

3. Compare exponents

127 – 128 = -1, so shift N1 right by 1 bit

4. Shift smaller mantissa if necessary

shift N1's mantissa: 1.1 >> 1 = 0.11 $(\times 2^1)$

5. Add mantissas

$$
\begin{array}{rl}
 0.11 & \times 2^1 \\
+\ 1.101 & \times 2^1 \\
\hline
10.011 & \times 2^1
\end{array}
$$

Floating Point Addition Example

6. **Normalize mantissa and adjust exponent if necessary**

   $10.011 \times 2^1 = 1.0011 \times 2^2$

7. **Round result**

   No need (fits in 23 bits)

8. **Assemble exponent and fraction back into floating-point format**

   $S = 0$, $E = 2 + 127 = 129 = 10000001_2$, $F = 001100..$

1 bit	8 bits	23 bits
0	10000001	001 1000 0000 0000 0000 0000
Sign	**Exponent**	**Fraction**

in hexadecimal: **0x40980000**