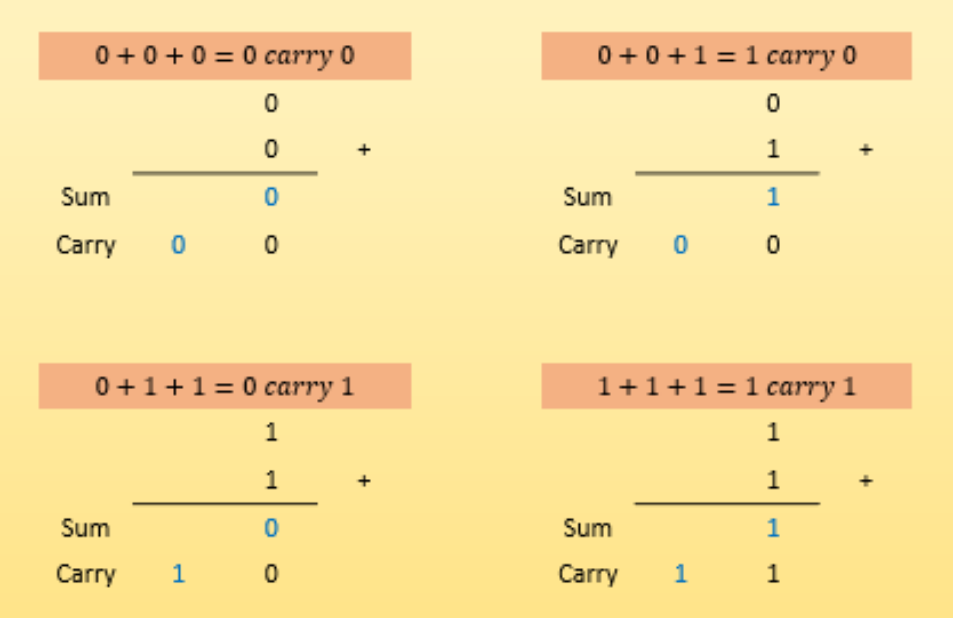
# 4.5.4 Binary Number Systems

## Adding Binary Numbers

To add binary numbers together we need to know how to add individual bits.

As binary is a base 2 number system, we have to carry a bit to the next column every time the sum gets to 2.

This carry bit will be used in the next bit's calculation.




**Example: Adding the unsigned binary numbers 10110011 and 01100110**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | |
| | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | + |
| Sum | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | |
| Carry | | | | | 1 | 1 | | | |

## Multiplying Binary Numbers

**Example: Multiplying the unsigned binary numbers 1101 and 1011.**

The following calculations show how each 1 bit in the multiplier creates a shifted partial product before the results are added.

| | | | | |
|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | |
| | 1 | 0 | 1 | 1 | × |

- We multiply the first binary number by the rightmost 1 bit of the second binary number.

| | | | | |
|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | |
| | 1 | 0 | 1 | 1 | × |
| | 1 | 1 | 0 | 1 | |

- Next we multiply the first binary number by the next 1 bit from the right of the second binary number. As this bit is 1 column over we are going to pad the answer with a 0 on the right side.

| | | | | | |
|---|---|---|---|---|---|
| | | 1 | 1 | 0 | 1 | |
| | | 1 | 0 | 1 | 1 | × |
| | | 1 | 1 | 0 | 1 | |
| | 1 | 1 | 0 | 1 | 0 | |

- Next we multiply the first binary number by the next 1 bit to the right. As this bit is 3 columns over we are going to pad the answer with 3 0s on the right side.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 1 | 0 | 1 | |
| | | | | 1 | 0 | 1 | 1 | × |
| | | | | 1 | 1 | 0 | 1 | |
| | | | 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | | |

- Lastly we add these results together.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 1 | 0 | 1 | |
| | | | | 1 | 0 | 1 | 1 | × |
| | | | | 1 | 1 | 0 | 1 | |
| | | | 1 | 1 | 0 | 1 | 0 | |
| | 1 | 1 | 0 | 1 | 0 | 0 | 0 | + |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |
| 1 | 1 | 1 | 1 | | | | | |

$$= \mathbf{10001111}$$

Decimal check: $1101_2 = 13$ and $1011_2 = 11$, so $13 \times 11 = 143$. Therefore, $10001111_2 = 143$.

## Signed and Unsigned Binary

So far we have been using **unsigned binary**, which works for positive numbers.

The range of numbers that can be represented using **n** bits is between **0 and $2^n$ - 1**.

- With 8 bits using unsigned binary, any number between 0 and 255 ($2^8$ - 1) can be represented.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 000 | 2 | 010 | 4 | 100 | 6 | 110 |
| 1 | 001 | 3 | 011 | 5 | 101 | 7 | 111 |

3 bits can represent any number between 0 and 7 ($2^3$ - 1).

We also need a way to represent **signed binary numbers**, including both negative and positive values.

One possible coding method for representing signed binary numbers is called **two's complement**.

## Two's Complement

In two's complement format, the most significant bit has a negative place value. For example, in an 8-bit number, the leftmost bit represents -128 instead of 128.

The 4-bit binary number 1001 in two's complement.

| -8 | 4 | 2 | 1 |
|---|---|---|---|
| 1 | 0 | 0 | 1 |

1001 in two's complement is $-8 + 1 = -7$

The 4-bit binary number 11001010 in two's complement.

| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

11001010 in two's complement is $-128 + 64 + 8 + 2 = -54$

## Subtraction using Two's Complement

This section shows how subtraction can be treated as addition by converting the value being subtracted into its two's complement form.

$$34 - 22 \text{ is equivalent to } 34 + (-22)$$

**The method to convert from a positive number to its' negative equivalent**

- Start from the right
- Keep the bits the same up to and including the first 1
- Flip every other bit

| | | | | | | |
|---|---|---|---|---|---|---|
| **22** | 0 | 1 | 0 | 1 | 1 | 0 |
| **−22** | 1 | 0 | 1 | 0 | 1 | 0 |

**Example: 01010011 – 00010110**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | − |

- The second binary number needs to be converted from its positive form to its negative form.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | + |

- We can then add the binary numbers. Any carry bits that extend beyond the leftmost bit are discarded.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | + |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | |
| ~~1~~ | 1 | | | | 1 | | | |

## Two's Complement Range

The range of numbers a $n$-bit binary number in two's complement can represent is between $-2^n$ and $2^n - 1$.

**Example: An 8-bit binary number in two's complement can range from $-128$ to $127$:**

| Decimal | Binary | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| -128 ($-2^8$) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Decimal | Binary | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 127 ($2^8$-1) | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## Fixed Point

Fixed point is used to represent fractional values when the position of the binary point stays fixed.

Bits to the left of the point represent whole numbers and bits to the right represent fractions.

**Example: Fixed point binary number $1100 \cdot 1010$**

| $2^3$ | $2^2$ | $2^1$ | $2^0$ | | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | · | 1 | 0 | 1 | 0 |

$$8 + 4 + \frac{1}{2} + \frac{1}{8}$$

$$= 12\frac{5}{8}$$

## Floating Point

Floating point is used to represent a wider range of fractional values by storing a mantissa and an exponent.

The point is floating as it can be moved left or right by increasing or decreasing the exponent.

**Example: A 12-bit floating point binary number with an 8-bit mantissa and a 4-bit exponent, both in two's complement.**

0.1001110 0011

Mantissa:

| $-1$ | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | $\frac{1}{64}$ | $\frac{1}{128}$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

The mantissa is equal to:

$$\frac{1}{2}+\frac{1}{16}+\frac{1}{32}+\frac{1}{64}=\frac{39}{64}$$

Exponent:

| $-8$ | $4$ | $2$ | $1$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |

The exponent is equal to: $2+1=3$

$$\frac{39}{64}\times 2^3=\frac{39}{8}$$

## Errors

The range of numbers that fixed point and floating point binary can represent depends on the number of bits being used and how those bits are distributed.

Some decimals cannot be represented accurately in these formats at all.

**Example: The decimal $0.1$**

- $0.1$ in binary is $0.0001100110011001100110011\ldots$ with the pattern of $0011$ repeating indefinitely.
- Using fixed point form with 1 bit before the point and 7 bits after the point the closest representation possible would be $0.0001100$
- This is 0.1015625 in decimal.
- This is a **rounding error**.

The **absolute error** is the difference between the numerical value that was intended to be stored and the actual numerical value stored.

$$\boldsymbol{absolute\ error = |actual\ value\ - intended\ value|}$$

**Example: Absolute error**

We wanted to store the decimal $0.1$ but the closest we could store in the given format was 0.1015625 in decimal.

The absolute error is 0.1015625 – 0.1 = 0.0015625

The **relative error** demonstrates the proportion of the absolute error to the intended value.

$$\boldsymbol{relative\ error = \frac{absolute\ error}{intended\ value} \times 100}$$

**Example: Relative error**

$$\frac{0.0015625}{0.1} \times 100 = 1.5625\%$$

## Fixed Point: Range and precision

The range of a fixed point binary number depends on the position of the binary point.

- The more digits to the left of the binary point the greater the range.

| 64 | 32 | 16 | 8 | 4 | 2 | 1 | . | 1/2 |
|---|---|---|---|---|---|---|---|---|

- Conversely, the precision of the fixed point number increases the greater the number of digits to the right of the binary point.

| 1 | . | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 | 1/64 | 1/128 |
|---|---|---|---|---|---|---|---|---|

Given a fixed point number with a given number of bits there is a trade-off between the range and precision of the numbers that can be represented when distributing the bits over the left and right side of the binary point.

## Floating Point: Range and precision

The range and precision of a floating point binary number depend on the number of bits dedicated to the mantissa and exponent.

- The more bits that are dedicated to the exponent, the greater the range.

| Mantissa | | | |
|---|---|---|---|
| $-1$ | $1/2$ | $1/4$ | $1/8$ |

| Exponent | | | | | | | |
|---|---|---|---|---|---|---|---|
| $-128$ | $64$ | $32$ | $16$ | $8$ | $4$ | $2$ | $1$ |

- The more bits that are dedicated to the mantissa the greater the **precision** of the floating point number.

| Mantissa | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $-1$ | $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{8}$ | $\frac{1}{16}$ | $\frac{1}{32}$ | $\frac{1}{64}$ | $\frac{1}{128}$ | $\frac{1}{256}$ | $\frac{1}{512}$ |

| Exponent | |
|---|---|
| $-2$ | $1$ |

Given a floating point number with a given number of bits there is a trade-off between the precision and range of the numbers that can be represented when distributing the bits over the mantissa and exponent.

- To increase the precision, we can take a bit from the exponent to the mantissa at the cost of reducing the range.

- To increase the range, we can take a bit from the mantissa to the exponent at the cost of reducing the precision.