Bitwise operations

Binary shifts

What are binary shifts?

- A binary shift is an operation that **moves all the bits in a binary number left or right** by a certain number of positions
- Often used in programming and computer systems for **fast multiplication or division by powers of 2**, or for manipulating individual bits
- There are three main types of binary shift:
  - **Logical**
  - **Arithmetic**
  - **Cyclic**

Logical

- **Moves bits left or right** and **fills the gap with 0s**
- Used for **unsigned binary numbers** or raw bit manipulation

Direction	What happens
Left	All bits shift left, a 0 fills the rightmost bit
Right	All bits shift right, a 0 fills the leftmost bit

Left

- The following number is shifted by **two** places to the left

Step	128	64	32	16	8	4	2	1
Original	0	0	0	0	1	1	1	0
Left shift by 2	0	0	1	1	1	0	0	0

- **Original number:** 0000 1110 = **14**
- **Left shift (2) result:** 0011 1000 = **56**
- Each left shift has **doubled** the number:
  - **Original value** = **14**
  - **Left shift 1** - Doubled the number to 28

Right

Left shift 2 - Doubled the number to 56

- The following number is shifted by **three** places to the right

Step	128	64	32	16	8	4	2	1
Original	1	1	0	0	1	0	0	0
Right shift by 3	0	0	0	1	1	0	0	1

- **Original number:** 1100 1000 = **200**
- **Right shift (3) result:** 0001 1001 = **25**

- Each right shift has **halved** the number:
  - **Original value** = **200**
  - **Right shift 1** - Halved the number to 100
  - **Right shift 2** - Halved the number to 50
  - **Right shift 3** - Halved the number to 25

Arithmetic

- **Moves bits**, but **preserves the sign bit** (used for signed binary numbers in two's complement)

Direction	What happens
Left	Same as logical left (shift left, 0 in)
Right	Shift right, **copy the sign bit (MSB)** into the new leftmost bit

Right

- The following number is shifted by **three** places to the right

Step	128	64	32	16	8	4	2	1
Original	1	1	1	0	1	0	0	0
Right shift by 3	1	1	1	1	1	1	0	1

- **Original number**: 1110 1000 = **−24**
- **Right shift (3) result**: 1111 1101 = **−3**
- Each arithmetic right shift **divides the number by 2**, rounding **towards negative infinity** (preserving the sign bit)
  - **Original value** = **−24**
  - **Right shift 1** → 1111 0100 = **−12**
  - **Right shift 2** → 1111 1010 = **−6**
  - **Right shift 3** → 1111 1101 = **−3**

Cyclic

- **Rotates** the bits around, nothing is lost, no 0s added
- The bit that falls off one end is **reused** at the other end

Direction	What happens
Left	Leftmost bit moves to the rightmost position
Right	Rightmost bit moves to the leftmost position

- The following number is shifted by **three** places to the left and right

Left and Right

Step	128	64	32	16	8	4	2	1
Original	1	0	1	1	0	0	0	1
Cyclic left (×3)	1	0	0	0	1	1	0	1
Cyclic right (×3)	0	0	1	1	0	**1**	1	0

- The 3 leftmost bits 101 wrap around to the right side
- The 3 rightmost bits 001 wrap around to the left side
- **Original**: 1011 0001 = **177**
- **Cyclic left (3)**: 1000 1101 = **141**
- **Cyclic right (3)**: 0011 0110 = **54**

Shift type	Left shift	Right shift	Used for
Logical Shift	Shift bits left, fill with 0	Shift bits right, fill with 0	Unsigned numbers, raw bits
Arithmetic Shift	Same as logical left	Shift bits right, **preserve sign bit**	Signed numbers (two's complement)
Cyclic Shift	Rotate bits left (no loss)	Rotate bits right (no loss)	Bit rotation, encryption, checksums

- Cyclic shifts **do not** preserve the sign bit and are not equivalent to multiplying or dividing by powers of two

Device control & bit masking

What is device control?

- In computer systems and embedded devices, **each bit** in a byte can be used to represent the **state of a device or feature** (e.g. turning a light on, checking if a sensor is active, or flagging an error)
- You can **control or monitor** these bits using **bitwise operations** and **bit masking**

What is bit masking?

- Bit masking uses binary patterns (masks) to **test**, **set**, **clear**, or **toggle** specific bits **without affecting the others**

Bitwise AND operation (&)

- Used to **test** if a specific bit is set to 1
- Only returns 1 when **both** the binary value **and** the mask have 1 in the same position
- Useful for **checking** the status of a device

Description	128	64	32	16	8	4	2	1
Binary	1	0	1	1	1	0	0	1
Mask	0	0	1	1	0	0	0	0
Result	0	0	1	1	0	0	0	0

- Check if a **heater (Bit 5)** and **light (Bit 4)** are both ON
- Apply a mask with 1s at those positions
- If the result is **not 0**, the devices are **ON**

Bitwise OR operation (|)

- Used to **set** specific bits to 1
- Returns 1 if **either** the binary value **or** the mask has 1
- This is useful to **turn on a device** (set a bit to 1) without changing the others

Description	128	64	32	16	8	4	2	1
Binary	1	1	0	0	1	0	1	0
Mask	0	1	1	1	0	0	0	0
Result	1	1	1	1	1	0	1	0

- Turn ON the **alarm system (Bit 6)** and **door lock (Bit 5)** by ORing with a mask that has 1s at those positions

Bitwise XOR operation (^)

- Used to **toggle** (flip) specific bits
- Returns 1 if the bits are **different**
- Can be used for **switching** a device from ON to OFF or vice versa

Description	128	64	32	16	8	4	2	1
Binary	1	0	1	0	1	0	1	0
Mask	0	0	1	1	0	0	0	0
Result	1	0	0	1	1	0	1	0

- Toggle the **fan (Bit 5)** and **heating system (Bit 4)** with a mask that has 1s in those positions

Summary

Operation	Symbol	Purpose	Example Use
Bitwise AND	&	Test a bit	Check if a sensor is active
Bitwise OR	|	Set a bit	Turn ON a device
Bitwise XOR	^	Toggle a bit	Flip device state (ON ↔ OFF)