XOR Operation

==========

XOr Instruction:

==========

The XOR (exclusive OR) instruction performs a boolean exclusive-OR operation between corresponding bits in two operands and stores the result in the destination operand.

The XOR operation follows these rules:

If both bits are the same (both 0 or both 1), the result is 0. If the bits are different (one 0 and one 1), the result is 1. A bit XORed with 0 retains its value, and a bit XORed with 1 toggles (complements) its value.

XOR is reversible, meaning applying it twice to the same operand reverts it to the original value.

Here's a truth table for **XOR (x ⊕ y):**

XOR Operation

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

The XOR instruction performs a bitwise exclusive OR operation on its two operands.

```
xor destination_operand, source_operand
```

The XOR instruction always clears the Overflow and Carry flags. It modifies the Sign, Zero, and Parity flags according to the value assigned to the destination operand.

Parity checking is a method used to determine whether a binary number has even or odd parity. It counts the number of 1 bits in the number, and if the count is even, it's considered even parity, and if it's odd, it's considered odd parity.

In x86 processors, the **Parity flag (PF)** is set when the lowest byte (8 bits) of the destination operand of a bitwise or arithmetic operation has even parity. If the lowest byte has odd parity, the

PF is cleared.

Here's an example of how to check parity without changing the value of a byte in assembly:

```
mov al, 10110101b ; 5 bits = odd parity
xor al, 0         ; Parity flag clear (odd)

mov al, 11001100b ; 4 bits = even parity
xor al, 0         ; Parity flag set (even)
```

The XOR instruction can be used to toggle (invert) bits, check the parity of a number, and perform other bitwise operations. Here are some examples of how to use the XOR instruction:

```
mov ax, 64C1h ; 0110 0100 1100 0001
xor ah, al    ; Parity flag set (even)
```

To calculate parity for 32-bit values, you can XOR all the bytes together, like this:

```
B0 XOR B1 XOR B2 XOR B3
```