OR Operation

==========

Or Instruction:

==========

Operation: The OR instruction performs a bitwise OR operation between each pair of matching bits in two operands and stores the result in the destination operand.

```
OR reg, reg
OR reg, mem
OR reg, imm
OR mem, reg
OR mem, imm
```

Combinations above: Same as the AND instruction.

Operand Sizes: The operands can be 8, 16, 32, or 64 bits, and they must be the same size.

Truth Table: For each matching bit in the two operands: The output bit is 1 when at least one of the input bits is 1.

Example: x OR y, where x and y are bits.

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

Setting Bits: The OR instruction is useful when you need to set one or more bits in an operand without affecting other bits. This is a common technique in microcontroller and embedded systems programming to manipulate specific control bits in registers while preserving the rest of the configuration. For example, you can set bit 2 in the AL register with:

```
or AL, 00000100b ; Set bit 2, leave others unchanged
```

or

```
OR AL, 1 << 2
```

The OR instruction performs a bitwise OR operation on its two operands. The << operator shifts the number on its left by the number of bits specified by the number on its right.

In this case, the number 1 is shifted left by 2 bits, which results in the number 4. The OR instruction then ORs the AL register with the number 4, which sets the bit in position 2 of the AL register.

Here is an example of how to use the code above:

```
; Set the bit in position 2 of the control byte in the AL register.

mov al, 0b00111010 ; AL = 00111010
OR AL, 1 << 2 ; AL = 00111110

; The bit in position 2 of the AL register is now set.
```

You can use the same technique to set any bit in an operand, regardless of its position. To set bit n, simply OR the operand with the number 1 shifted left by n bits.

The instruction OR AL, 1 << 2 is a concise way of setting bit 2 (counting from the least significant bit) in the AL register without altering the other bits.

It uses the left shift (<<) operation to create the bitmask 00000100b, and then it performs a bitwise OR operation with the contents of AL.

Here's the breakdown:

1 << 2 shifts the binary value 00000001 two positions to the left, resulting in 00000100. This

creates a bitmask with only bit 2 set to 1. OR AL, 00000100b performs a bitwise OR operation between the contents of AL and the bitmask.

This sets bit 2 in AL to 1 while leaving the other bits unchanged. So, after executing this instruction, bit 2 in the AL register will be set to 1, and the rest of the bits will remain as they were.

Flags: The OR instruction always clears the Carry and Overflow flags. It modifies the Sign, Zero, and Parity flags based on the value assigned to the destination operand.

These instructions are essential for performing bitwise operations in assembly language, and they are often used for tasks like bit manipulation and flag setting/clearing. If you have any specific questions or would like further examples, feel free to ask!

```
SetX:
10000000000000000000000000000111

SetY:
10000010101000000000011101100011

Union (SetX OR SetY):
10000010101000000000011101100111
```

It's clear that the OR operation combines the two sets by preserving any bits that are set in either SetX or SetY.

In binary representation:

- SetX has bits set at positions 0, 1, and 31.
- SetY has bits set at positions 0, 5, 9, 14, 18, 23, 26, 30, and 31.

When you perform a bitwise OR between SetX and SetY, the resulting union has bits set at all the positions where at least one of SetX or SetY had a bit set. In this case, the union contains bits set at positions 0, 1, 5, 9, 14, 18, 23, 26, 30, and 31.

This operation can be visualized as a union operation in set theory, where you're combining the elements of two sets while eliminating duplicates.