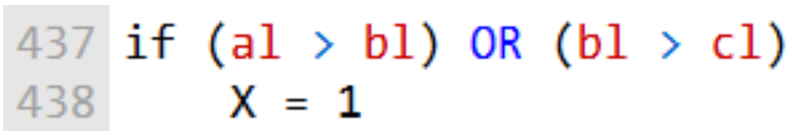
Short-Circuit evaluation(OR)

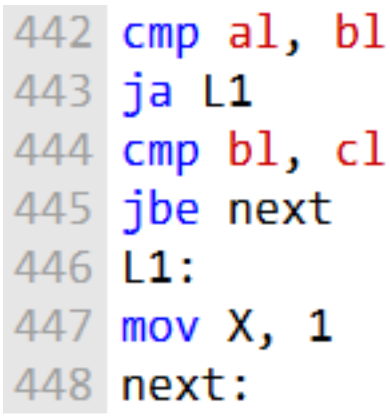
With short-circuit evaluation, the second operand of an OR expression is only evaluated if the first operand is false.

This is because if the first operand is true, then the overall expression must be true, regardless of the value of the second operand.

The following assembly language code implements short-circuit evaluation for the OR operator:



```
if (al > bl) OR (bl > cl)
    X = 1
```



```
cmp al, bl
ja L1
cmp bl, cl
jbe next
L1:
mov X, 1
next:
```

This code first compares the values of the registers al and bl. If al is greater than bl, then the second operand of the OR expression is not evaluated, and the program jumps to the L1 label.

Otherwise, the program compares the values of the registers bl and cl. If bl is less than or equal to cl, then the program jumps to the next label. Otherwise, the program stores the value 1 in the register X and jumps to the next label.

The next label is used to exit the code, regardless of whether the OR expression evaluated to true or false.

The following table shows the difference between the two code examples:

Instruction	Description
cmp al, bl	Compare the values in registers al and bl.
ja L1	Jump to label L1 if al is greater than bl.
cmp bl, cl	Compare the values in registers bl and cl.
jbe next	Jump to the next label if bl is less than or equal to cl.
L1:	Label.
mov X, 1	Store the value 1 in the register X.
next:	Label.

The first code example is more efficient because it uses a ja instruction instead of a jbe instruction. The ja instruction can be implemented as a single machine instruction, while the jbe instruction may require multiple machine instructions.

In practice, the compiler will typically generate the most efficient code possible, regardless of whether the programmer uses a ja instruction or a jbe instruction.

However, it is important for programmers to understand how short-circuit evaluation is implemented in assembly language so that they can write efficient code.

As you mentioned, there are multiple ways to implement a compound expression containing OR operators in assembly language. For example, the following code is also functionally equivalent to the previous example:

```
cmp al, bl
je L1
cmp bl, cl
je L1
mov X, 1
next:
L1:
```

This code first compares the values of the registers al and bl. If al is equal to bl, then the program jumps to the L1 label. Otherwise, the program compares the values of the registers bl and cl.

If bl is equal to cl, then the program jumps to the L1 label. Otherwise, the program stores the value 1 in the register X and jumps to the next label.

The L1 label is used to indicate that the overall expression is true. The next label is used to exit the code, regardless of whether the OR expression evaluated to true or false.

Ultimately, the best way to implement a compound expression containing OR operators in assembly language will depend on the specific needs of the program.