LEA vs MOV

----------

1)

LEA means Load Effective Address

* MOV means Load Value

In short, LEA loads a pointer to the item you're addressing whereas MOV loads the actual value at that address.

The purpose of LEA is to allow one to perform a non-trivial address calculation and store the result [for later usage]

LEA ax, [BP+SI+5] ; Compute address of value

MOV ax, [BP+SI+5] ; Load value at that address

Where there are just constants involved, MOV (through the assembler's constant calculations) can sometimes appear to overlap with the simplest cases of usage of LEA. Its useful if you have a multi-part calculation with multiple base addresses etc.

Given the following code:

L1 db "word", 0

mov al, [L1]

mov eax, L1

What do the brackets ([L1]) represent?

[L1] means the memory contents at address L1. After running mov al, [L1] here, The al register will receive the byte at address L1 (the letter 'w').

Operands of this type, such as [ebp], are called [memory operands](http://www.imada.sdu.dk/Courses/DM18/Litteratur/IntelnATT.htm).

I see that none tells about the caveat in following this as a rigid rule - if brackets, then dereference, except when it's the lea instruction.

lea is an exception to the above rule. Say we've

mov eax, [ebp - 4]

The value of ebp is subtracted by 4 and the brackets indicate that the resulting value is taken as an address and the value residing at that address is stored in eax. However, in lea's case, the brackets wouldn't mean that:

lea eax, [ebp - 4]

The value of ebp is subtracted by 4 and the resulting value is stored in eax. This instruction would just calculate the address and store the calculated value in the destination register. See [this post](https://stackoverflow.com/a/1699778/183120) for further details.

2)

Nasm syntax:

mov eax, var == lea eax, [var] ; i.e. mov r32, imm32

lea eax, [var+16] == mov eax, var+16

lea eax, [eax\*4] == shl eax, 2 ; but without setting flags

3)

-fPIE

-fPIC