

# Basic Assembly

Load Effective Address

# Objectives

- We will study the LEA instruction.

# LEA

- Load Effective Address.
- LEA dest,[expr]
  - Calculates expr and stores the result inside dest.
  - **Doesn't actually access any memory.**
    - Only calculates the resulting address.
  - Can be used to calculate addresses, or any other calculations.
  - Doesn't change the flags register!
- dest has to be a register.

# LEA (Cont.)

- Examples:

- `lea eax, [eax+1]`
  - $eax \leftarrow eax + 1$
- `lea esi, [eax+2*edx]`
  - $esi \leftarrow eax + 2 \cdot edx$
- `lea di, [eax+2*edx+5]`
  - $di \leftarrow (eax + 2 \cdot edx + 5) \% 2^{16}$

- Invalid syntax:

- `lea [eax], [eax+1]`
- `lea ecx, edx`

# LEA (Cont.)

- Examples:

- `lea eax, [eax+1]`
  - $eax \leftarrow eax + 1$
- `lea esi, [eax+2*edx]`
  - $esi \leftarrow eax + 2 \cdot edx$
- `lea di, [eax+2*edx+5]`
  - $di \leftarrow (eax + 2 \cdot edx + 5) \% 2^{16}$

- Invalid syntax:

- ~~`lea [eax], [eax+1]`~~
- ~~`lea ecx, edx`~~

# LEA (Cont.)

- The expression in brackets is just like any expression in brackets. (Like in MOV,AND etc...)
- LEA is special, because **no access to memory is done**, although there are brackets.
- The expression can not be too complex.
  - $esi + 2 * ecx + 5$  is about as complex as it can get.

# Example 1

- LEA calculates addresses:

```
section '.data' readable writeable
    nums      dd 100h dup (12345678h)
    snums     dw 100h dup (0ababh)

section '.text' code readable executable

start:
    mov     esi,nums
    mov     edi,snums
    call    read_hex

    ; Get address of dword number eax:
    lea     edx,[esi + 4*eax]

    ; Get address of word number eax:
    lea     ebx,[edi + 2*eax]
```

```
section '.data' readable writeable
    nums      dd 100h dup (12345678h)
    snums     dw 100h dup (0ababh)

section '.text' code readable executable

start:
    mov     esi,nums
    mov     edi,snums
    call    read_hex

    ; Get dword number eax:
    mov     edx,[esi + 4*eax]

    ; Get word number eax:
    mov     ebx,[edi + 2*eax]
```

# Example 2

- Adding numbers from 1 to 100.
- LEA doesn't change the flags register:

```
mov     ecx,100
xor     edx,edx

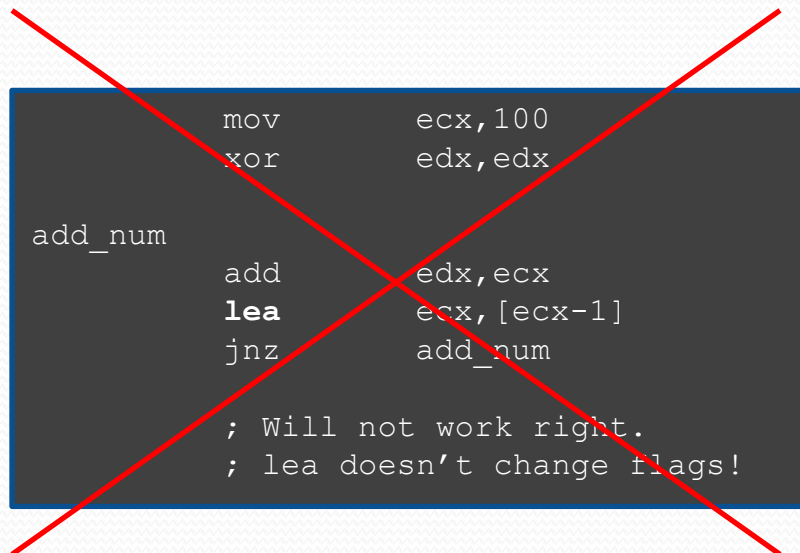
add_num
    add     edx,ecx
    lea     ecx,[ecx-1]
    jnz     add_num

; Will not work right.
; lea doesn't change flags!
```



# Example 2

- Adding numbers from 1 to 100.
- LEA doesn't change the flags register:



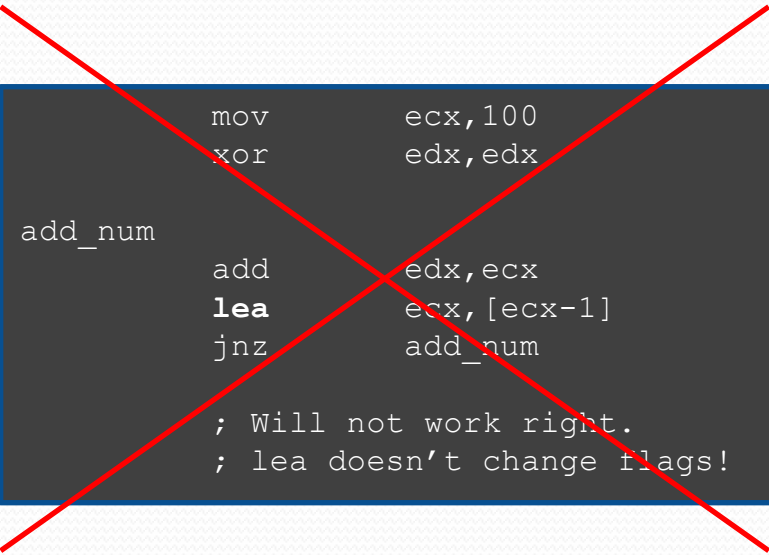
```
mov     ecx,100
xor     edx,edx

add_num
    add     edx,ecx
    lea    ecx,[ecx-1]
    jnz     add_num

; Will not work right.
; lea doesn't change flags!
```

## Example 2 (Cont.)

- Adding numbers from 1 to 100.
- LEA doesn't change the flags register:



```
mov     ecx,100
xor     edx,edx

add_num
    add     edx,ecx
    lea     ecx,[ecx-1]
    jnz     add_num

; Will not work right.
; lea doesn't change flags!
```

```
mov     ecx,100
xor     edx,edx

add_num
    add     edx,ecx
    lea     ecx,[ecx-1]
    test    ecx,ecx
    jnz     add_num

; edx = 1 + 2 + 3 + ... + 100
```

# Example 3

- LEA “saves” instructions:

```
mov     esi,ecx  
shl     edx,2  
add     esi,edx  
add     esi,5
```



```
lea esi,[ecx + 4*edx + 5]
```

# Summary

- The LEA instruction allows to calculate addresses easily.
- LEA doesn't actually access memory. It only calculates addresses.
- The LEA instruction doesn't change the flags register.