EE 213 Computer Organization and Assembly Language

Week # 5 Lecture # 14

17th Muharram ul Haram, 1440 A.H

28th September 2018

These slides contains materials taken from various sources. I fully acknowledge all copyrights.

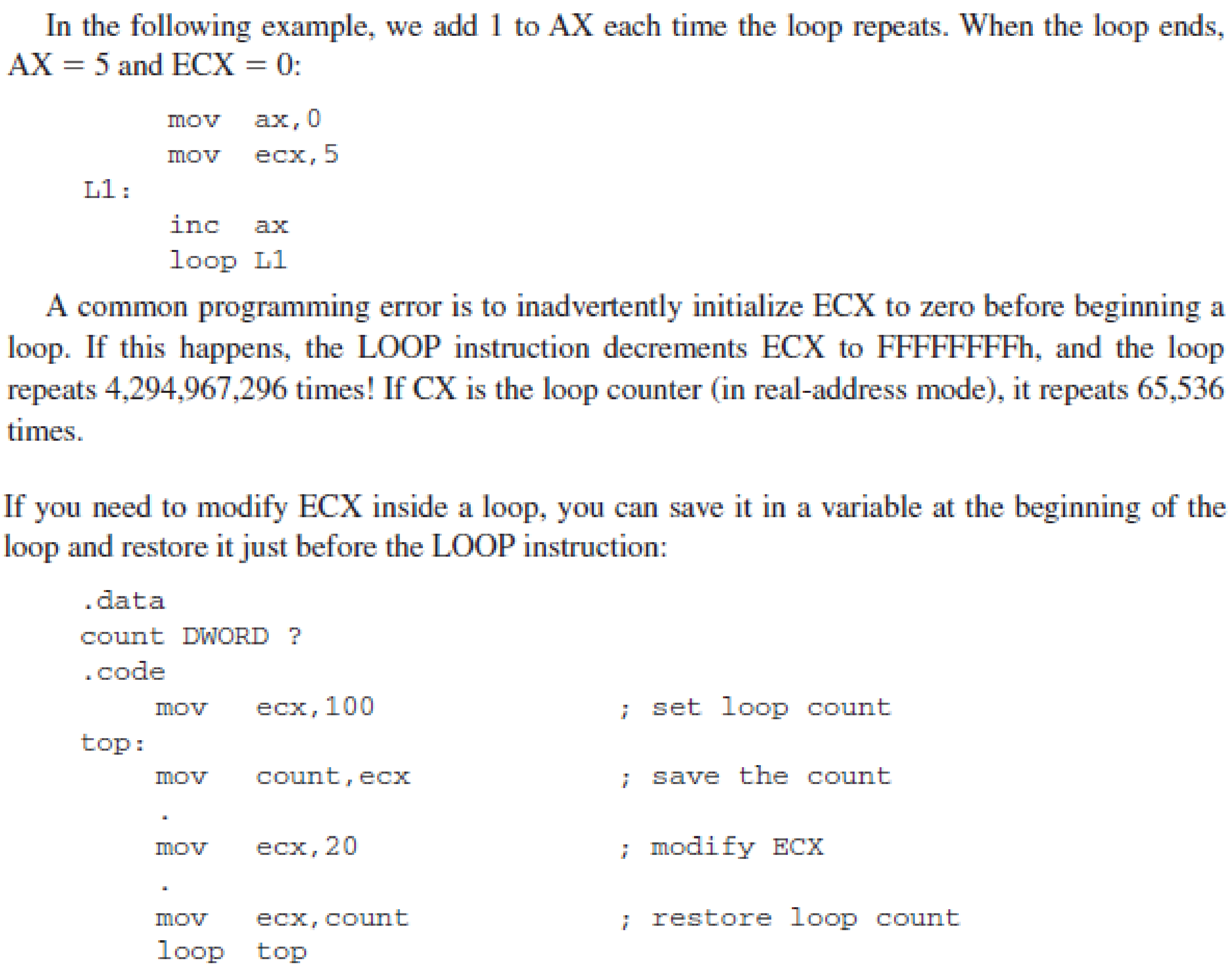
Minds open...

... Laptops closed

This presentation helps in delivering the lecture.
Take notes, interact and read text book to learn and gain knowledge.

Today’s Topics

- Loop Instruction example
- Quiz Solution
  - See separate item on slate



In the following example, we add 1 to AX each time the loop repeats. When the loop ends, AX = 5 and ECX = 0:

```
        mov   ax,0
        mov   ecx,5
L1:
        inc   ax
        loop L1
```

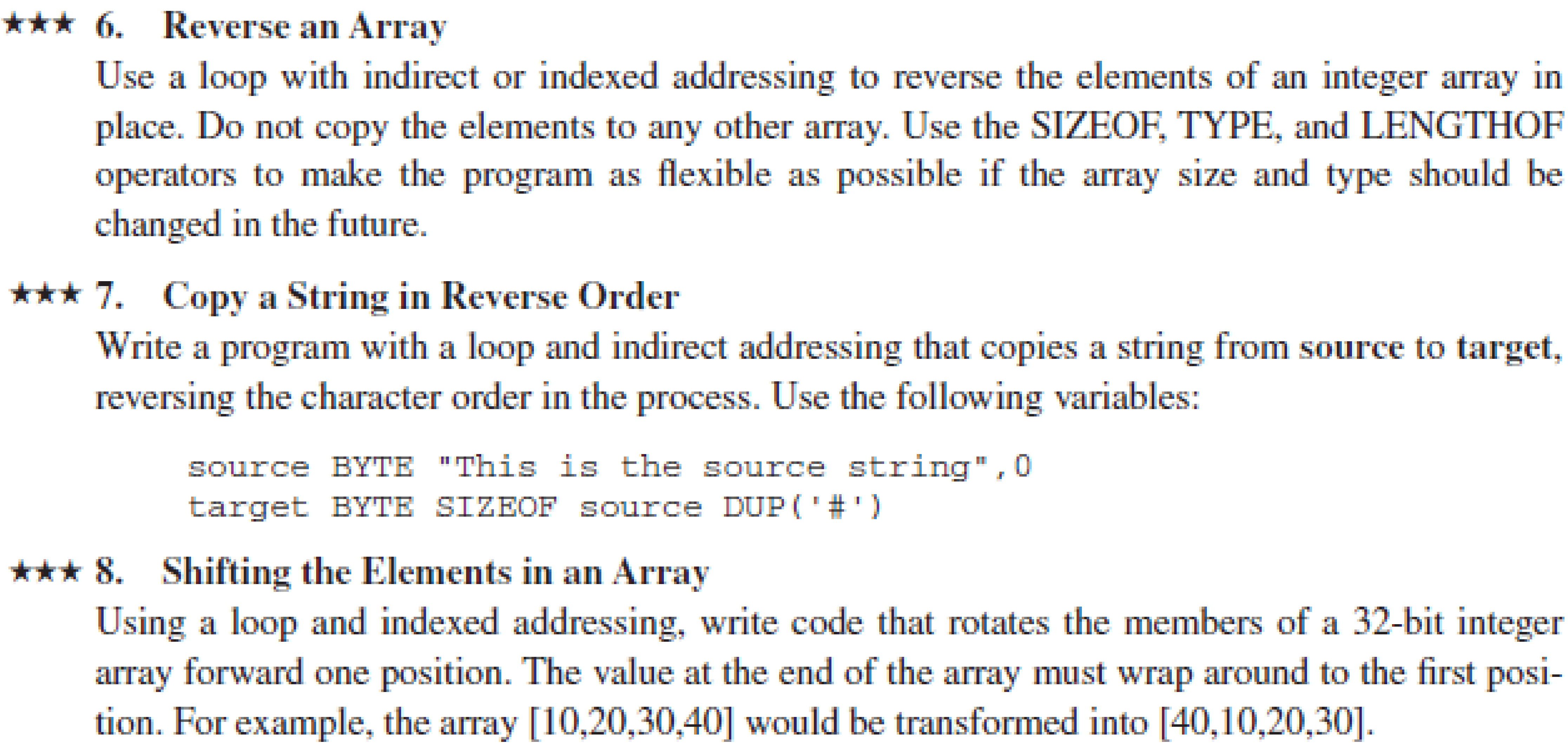
A common programming error is to inadvertently initialize ECX to zero before beginning a loop. If this happens, the LOOP instruction decrements ECX to FFFFFFFFh, and the loop repeats 4,294,967,296 times! If CX is the loop counter (in real-address mode), it repeats 65,536 times.

If you need to modify ECX inside a loop, you can save it in a variable at the beginning of the loop and restore it just before the LOOP instruction:

```
.data
count DWORD ?
.code
    mov   ecx,100                ; set loop count
top:
    mov   count,ecx              ; save the count
    .
    mov   ecx,20                 ; modify ECX
    .
    mov   ecx,count              ; restore loop count
    loop  top
```

Nested Loops When creating a loop inside another loop, special consideration must be given to the outer loop counter in ECX. You can save it in a variable:

```
.data
count DWORD ?
.code
    mov   ecx,100                     ; set outer loop count
L1:
    mov   count,ecx                   ; save outer loop count
    mov   ecx,20                      ; set inner loop count
L2:
    .
    .
    loop  L2                          ; repeat the inner loop
    mov   ecx,count                   ; restore outer loop count
    loop  L1                          ; repeat the outer loop
```



★★★ 6. Reverse an Array

Use a loop with indirect or indexed addressing to reverse the elements of an integer array in place. Do not copy the elements to any other array. Use the SIZEOF, TYPE, and LENGTHOF operators to make the program as flexible as possible if the array size and type should be changed in the future.

★★★ 7. Copy a String in Reverse Order

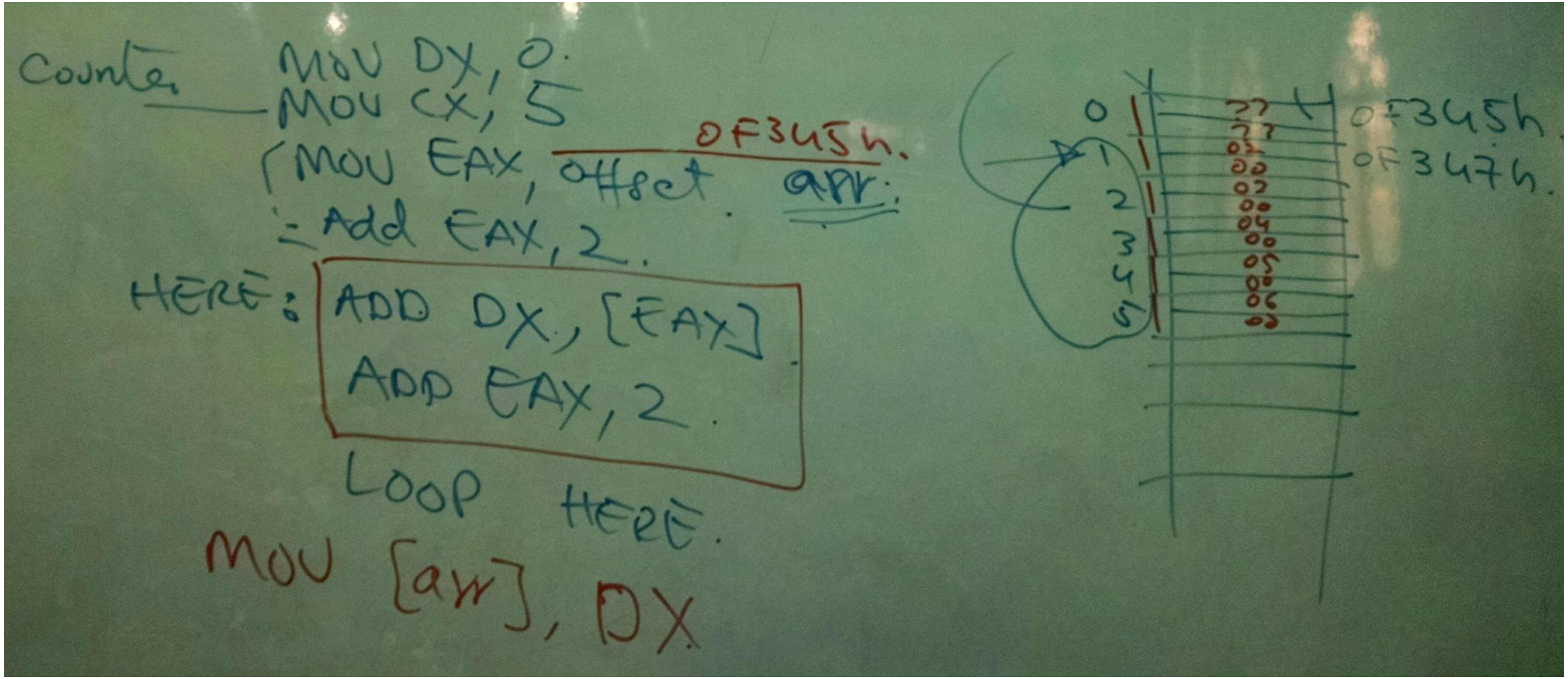
Write a program with a loop and indirect addressing that copies a string from **source** to **target**, reversing the character order in the process. Use the following variables:

```
source BYTE "This is the source string",0
target BYTE SIZEOF source DUP('#')
```

★★★ 8. Shifting the Elements in an Array

Using a loop and indexed addressing, write code that rotates the members of a 32-bit integer array forward one position. The value at the end of the array must wrap around to the first position. For example, the array [10,20,30,40] would be transformed into [40,10,20,30].

Suppose an integer array stores in its first element the sum of its elements. Write x86 assembly code snippet that adds element 1 to 5 of this array and places the sum in element 0. Assume each element of size 2 bytes and array starting from memory location 0F345H. ***Note: Drawing a memory map of this array help you understanding the problem more clearly. Only write assembly instructions, do not declare variables or write assembler directives.***



```
      MOV DX, 0.
      MOV CX, 5
      MOV EAX, offset arr.
      Add EAX, 2.
HERE: ADD DX, [EAX].
      ADD EAX, 2.
      LOOP HERE.
MOV [arr], DX
```