EE-2003
Computer Organization & Assembly Language

INSTRUCTOR

Engr. Aashir Mahboob
Lecturer, Department of Computer Science
FAST NUCES (Karachi)
aashir.mahboob@nu.edu.pk

JMP AND LOOP INSTRUCTIONS

- **By default, the CPU loads and executes programs sequentially.However, control may be transferred to a new location in the program.**

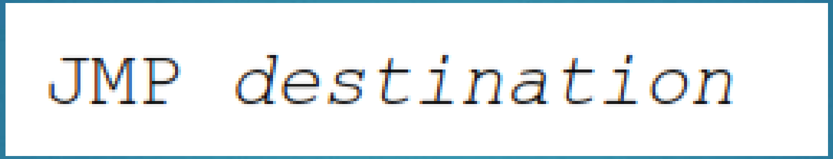
- **A transfer of control, or branch, is a way of altering the order in which statements are executed, there are two basic types:**

- **1. Unconditional Transfer:** No condition is involved, control is transferred to a new location in all cases.

- **2. Conditional Transfer:** The program branches if a certain condition is true (based on status of flags).

JMP Instruction

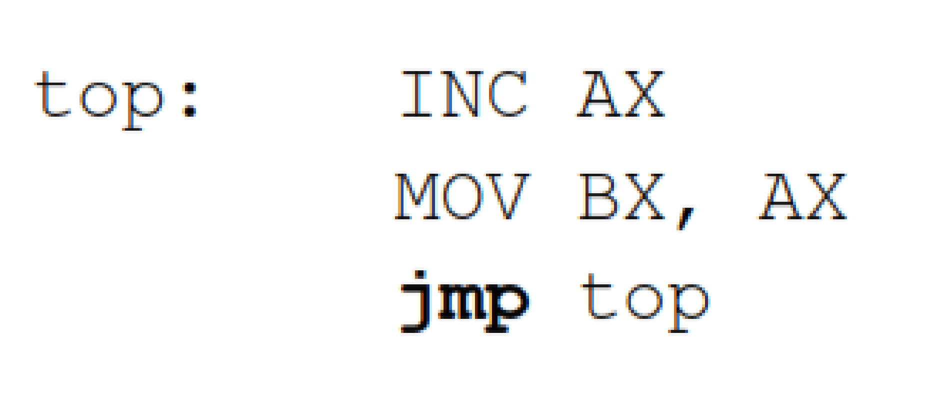
- The JMP instruction causes an unconditional transfer to a destination, identified by a code label.



```
JMP destination
```

- Offset of destination is moved into the instruction pointer, causing execution to continue at the new location.

- Logic: EIP ← Label.

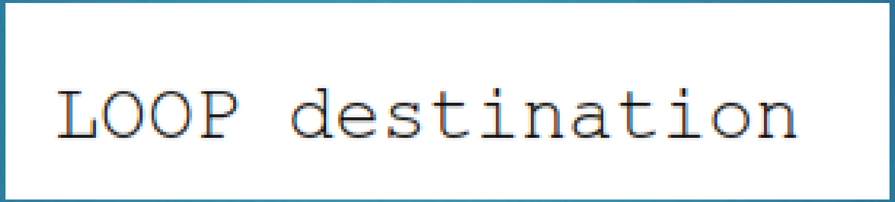


```
top:      INC AX
          MOV BX, AX
          jmp top
```

LOOP Instruction

- **The LOOP instruction, formally known as Loop According to ECX Counter, repeats a block of statements a specific number of times.**

- **ECX is automatically used as a counter and is decremented each time the loop repeats.**



```
LOOP destination
```

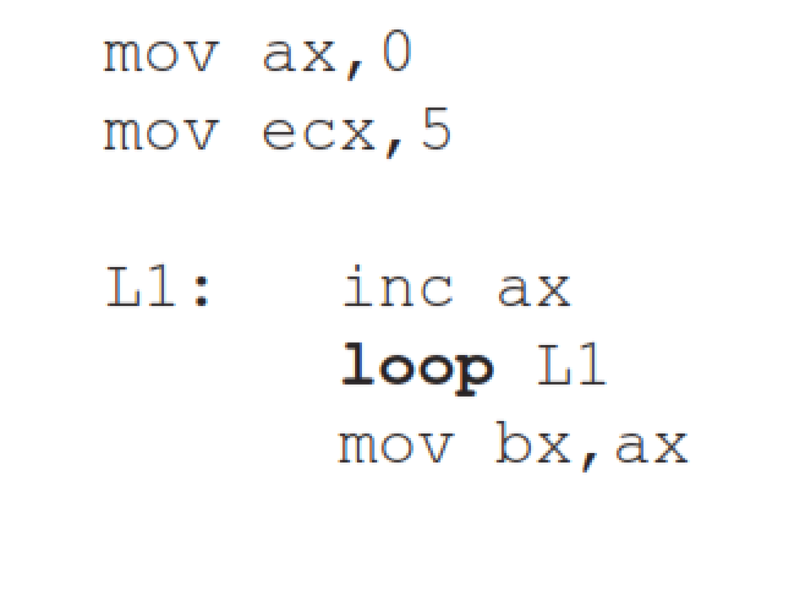
- **Logic:**
- **•ECX ←ECX –1**
- **•if ECX != 0 , jump to *target***

LOOP Instruction

- The execution of the LOOP instruction involves two steps:

- 1. First, it subtracts 1 from ECX.

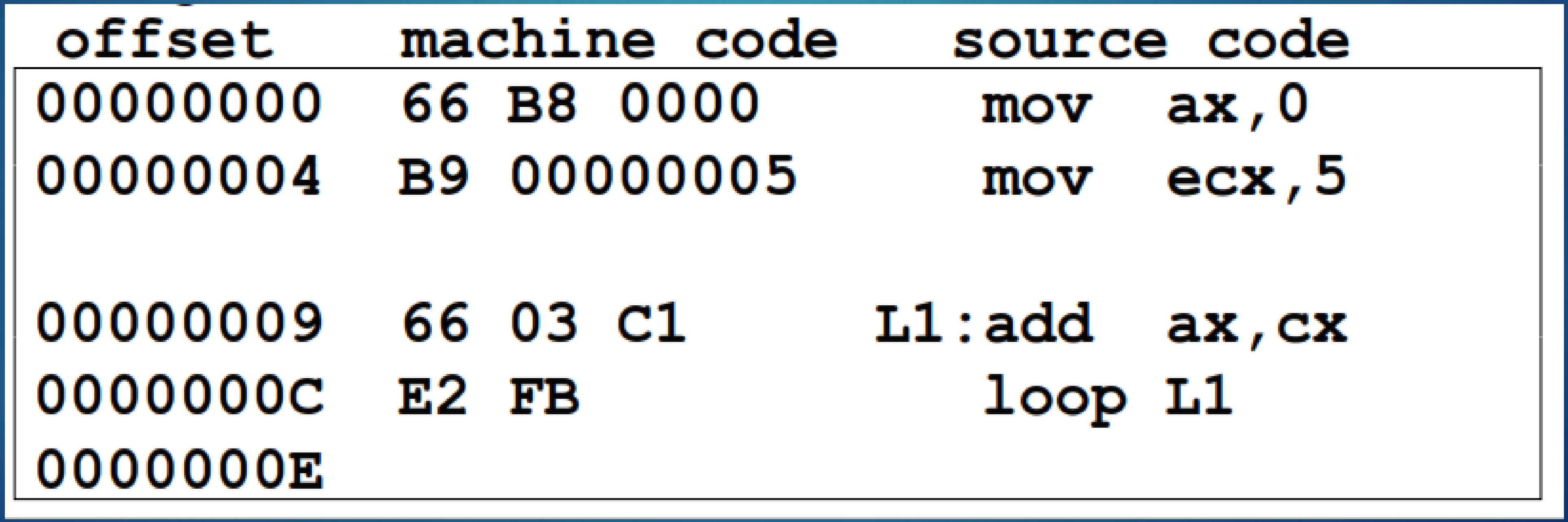
- 2. Next, it compares ECX to zero. If ECX is not equal to zero, a jump is taken to the label identified by *destination*. *Otherwise, no jump* takes place, and control passes to the instruction following the loop.



```
mov ax,0
mov ecx,5

L1:    inc ax
       loop L1
       mov bx,ax
```

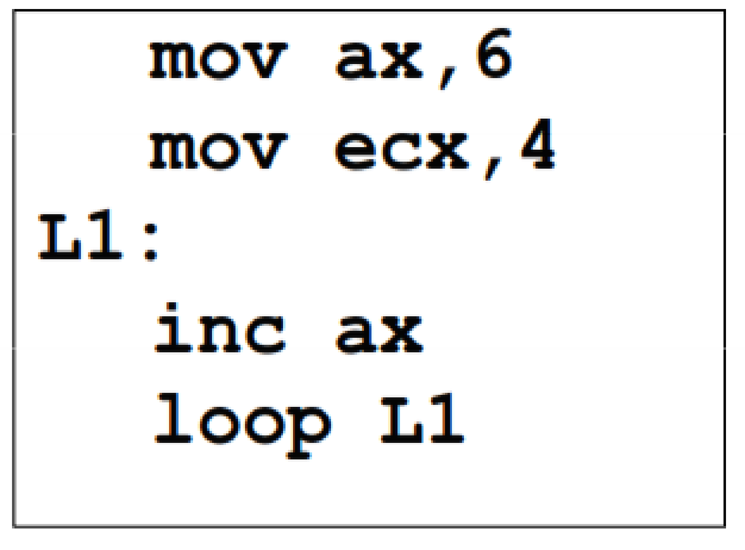
LOOP Instruction

- •The assembler calculates the distance in bytes between the current location and the offset of the target label. It is called the relative offset.
- The relative offset is added to EIP .



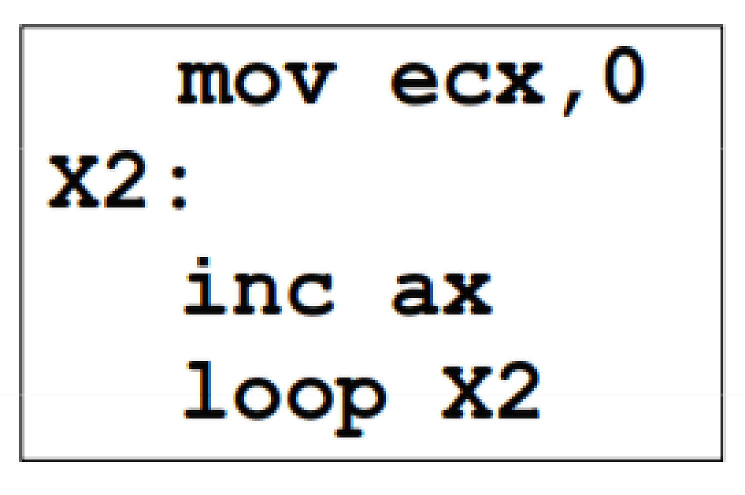
```
 offset     machine code      source code
00000000   66 B8 0000           mov   ax,0
00000004   B9 00000005          mov   ecx,5

00000009   66 03 C1          L1:add   ax,cx
0000000C   E2 FB                loop L1
0000000E
```

EXERCISE



```
   mov ax,6
   mov ecx,4
L1:
   inc ax
   loop L1
```

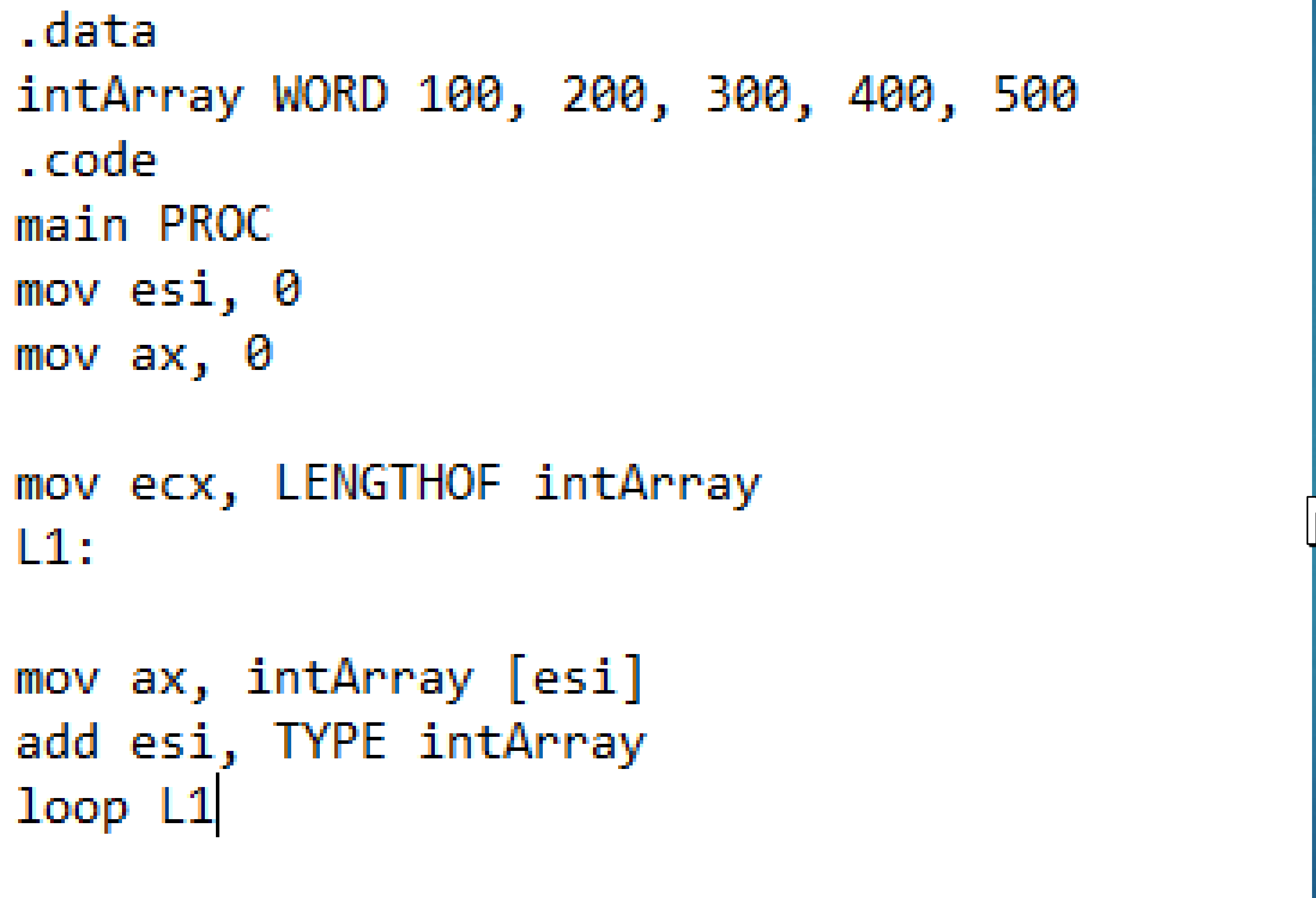
What will be the final value of AX?



```
   mov ecx,0
X2:
   inc ax
   loop X2
```

How many times will the loop execute?

EXAMPLE



```
.data
intArray WORD 100, 200, 300, 400, 500
.code
main PROC
mov esi, 0
mov ax, 0

mov ecx, LENGTHOF intArray
L1:

mov ax, intArray [esi]
add esi, TYPE intArray
loop L1
```

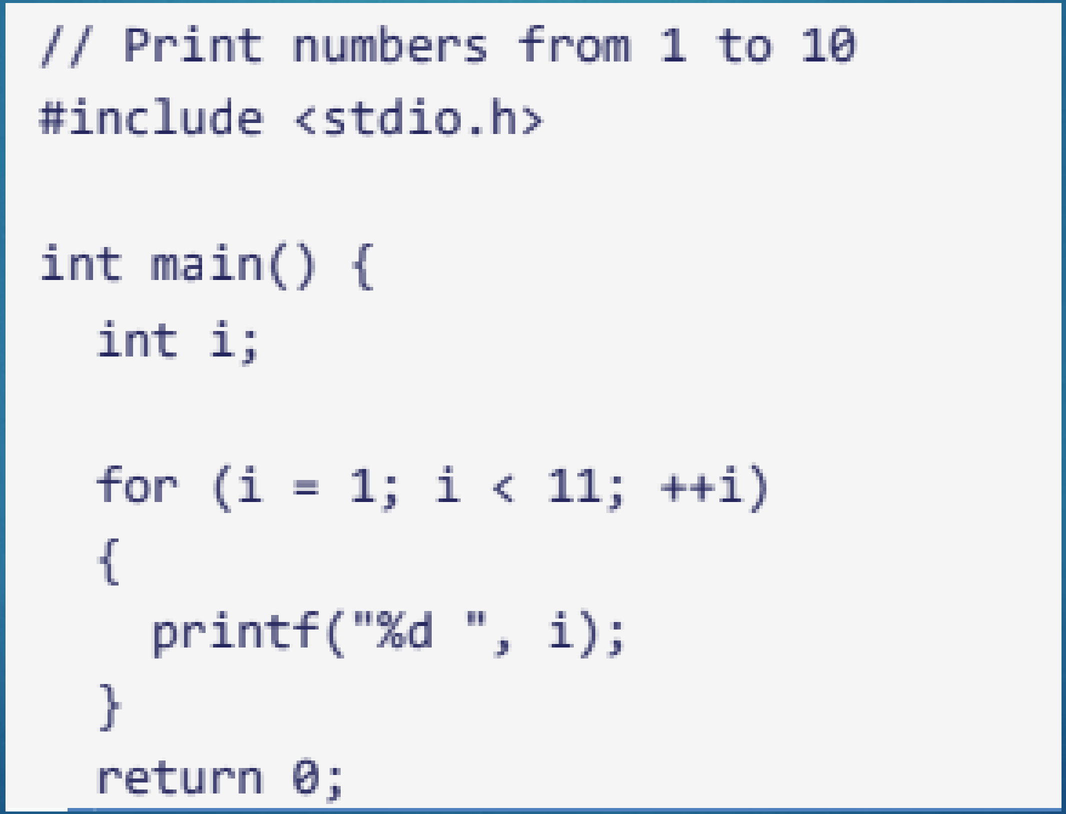
Final value of AX?

Final value of ESI?

No: of times loop executed?

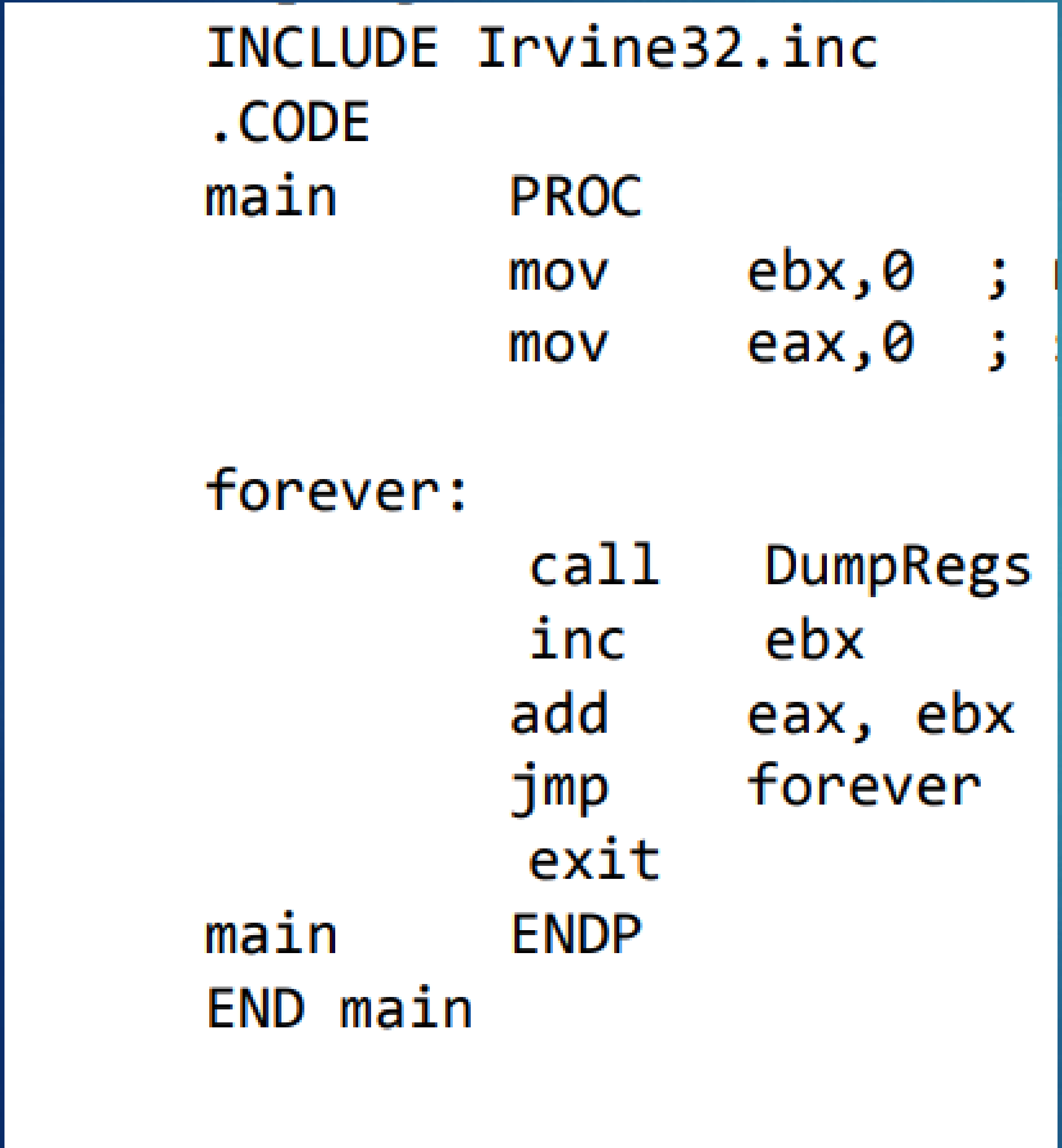
EXERCISE

- Write a program to print the numbers from 1 till N(your desire higher range) and move value in eax instead of using print statement. The following is the helping program of numbers to print from 1 to 10.



```
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
```

EXERCISE

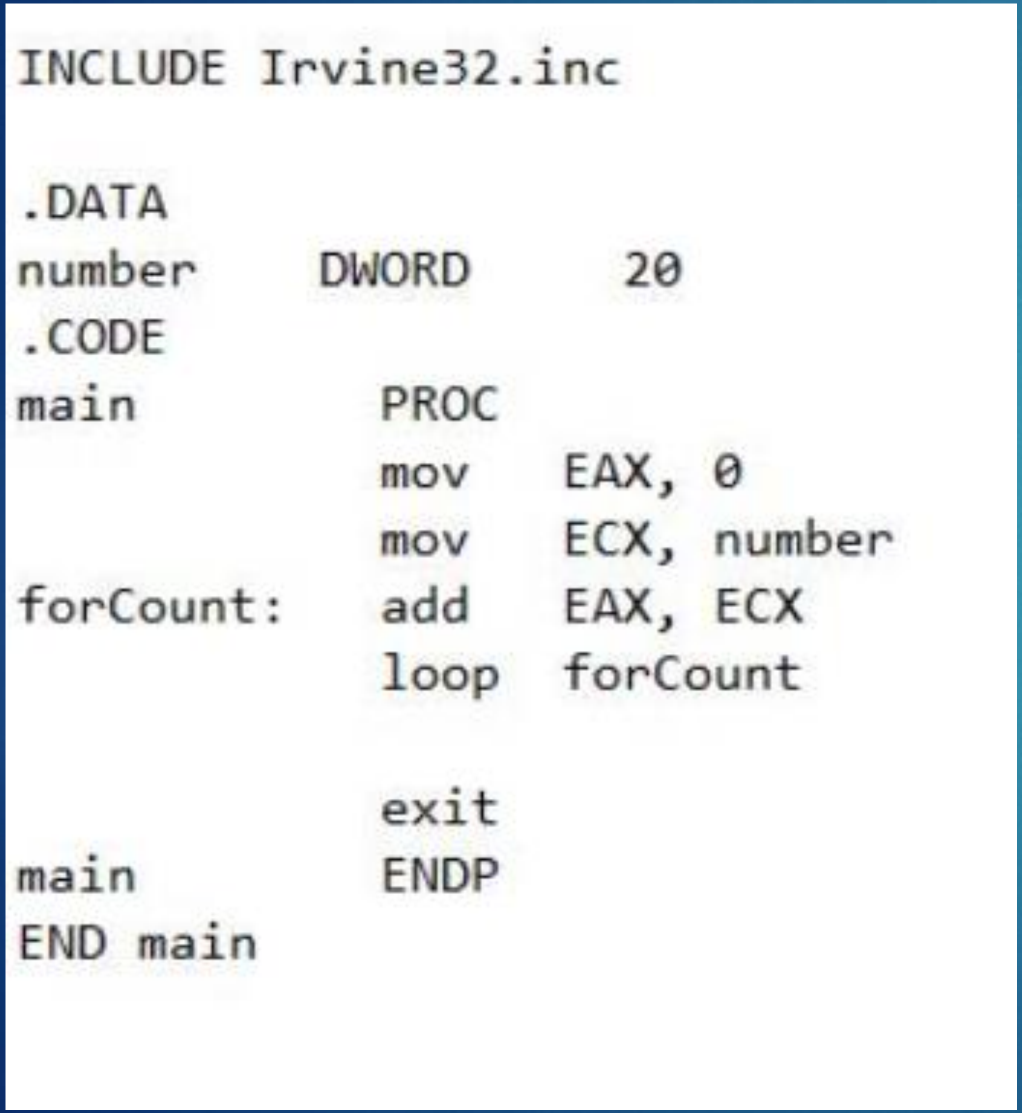


```
INCLUDE Irvine32.inc
.CODE
main      PROC
          mov     ebx,0   ;
          mov     eax,0   ;

forever:
           call    DumpRegs
           inc     ebx
          add     eax, ebx
          jmp     forever
           exit
main      ENDP
END main
```

Final value of EAX?

No: of times loop executed?

EXERCISE



```
INCLUDE Irvine32.inc

.DATA
number     DWORD     20
.CODE
main            PROC
                mov     EAX, 0
                mov     ECX, number
forCount:       add     EAX, ECX
                loop    forCount

                exit
main            ENDP
END main
```

Final value of EAX?

Final value of ECX?

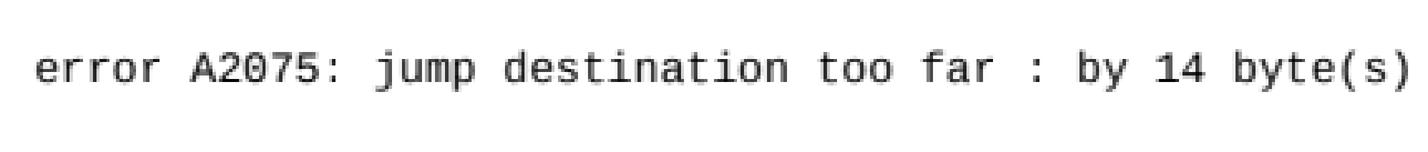
No: of times loop executed?

Programming Errors with Loop

- 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.

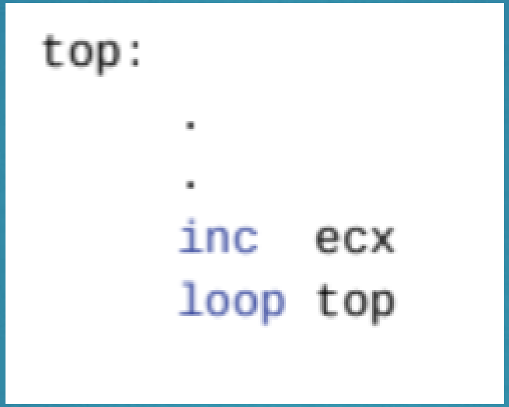
- Occasionally, you might create a loop that is large enough to exceed the allowed relative jump range of the LOOP instruction. Following is an example of an error message generated by MASM because the target label of a LOOP instruction was too far away:



```
error A2075: jump destination too far : by 14 byte(s)
```

Programming Errors with Loop

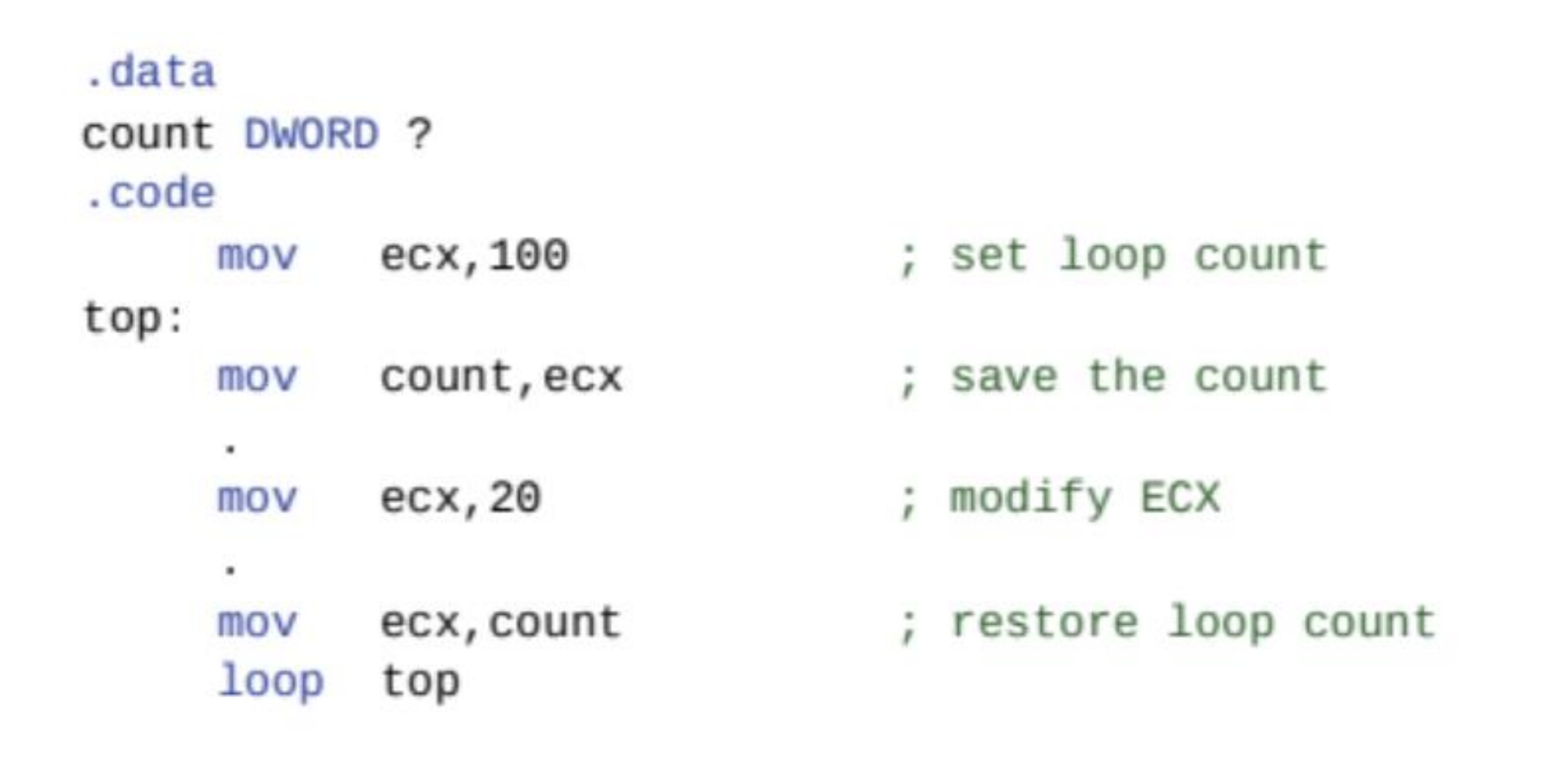
- Rarely should you explicitly modify ECX inside a loop. If you do, the LOOP instruction may not work as expected. In the following example, ECX is incremented within the loop. It never reaches zero, so the loop never stops:



```
top:
        .
        .
        inc   ecx
        loop  top
```

- 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:

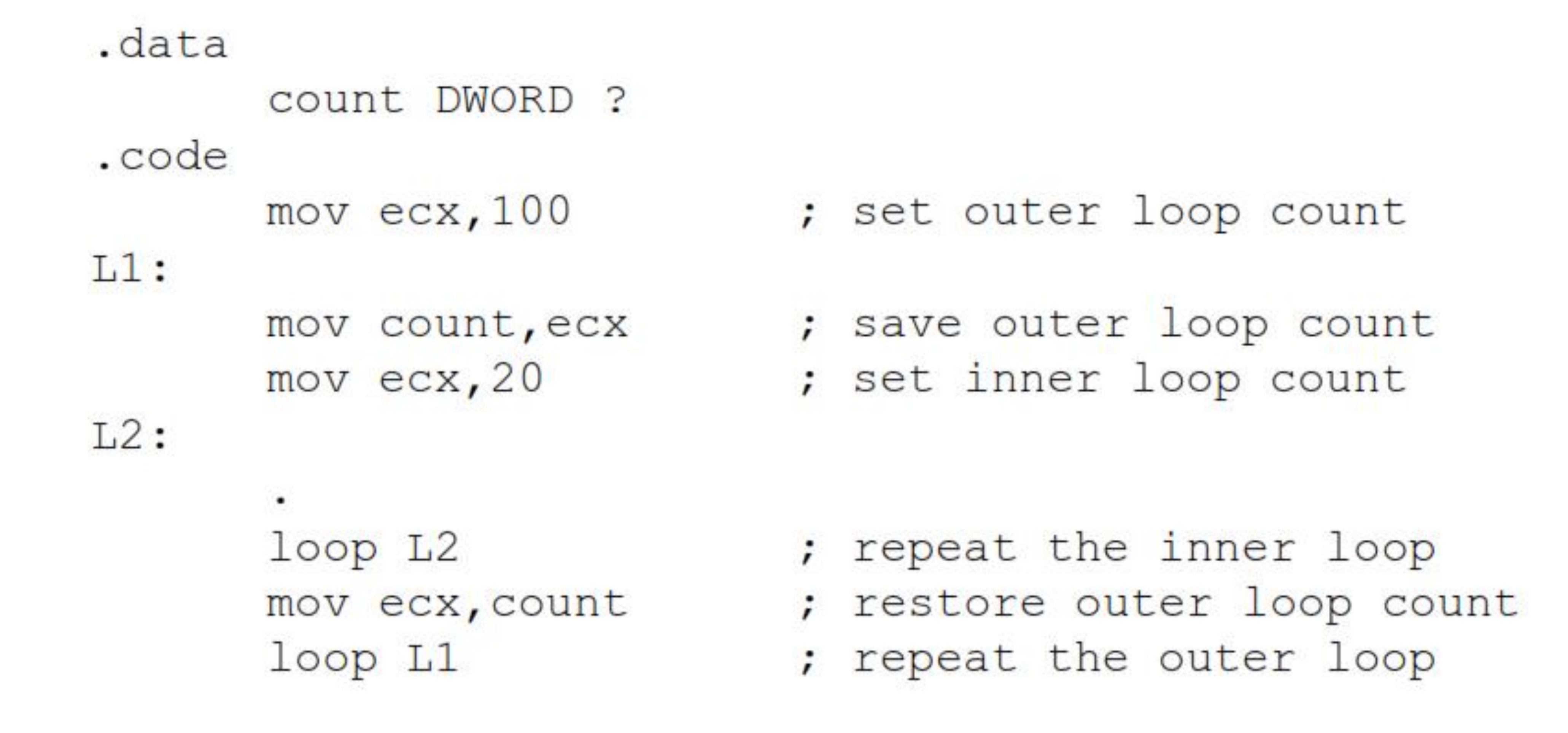
Programming Errors with Loop



```
.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
```

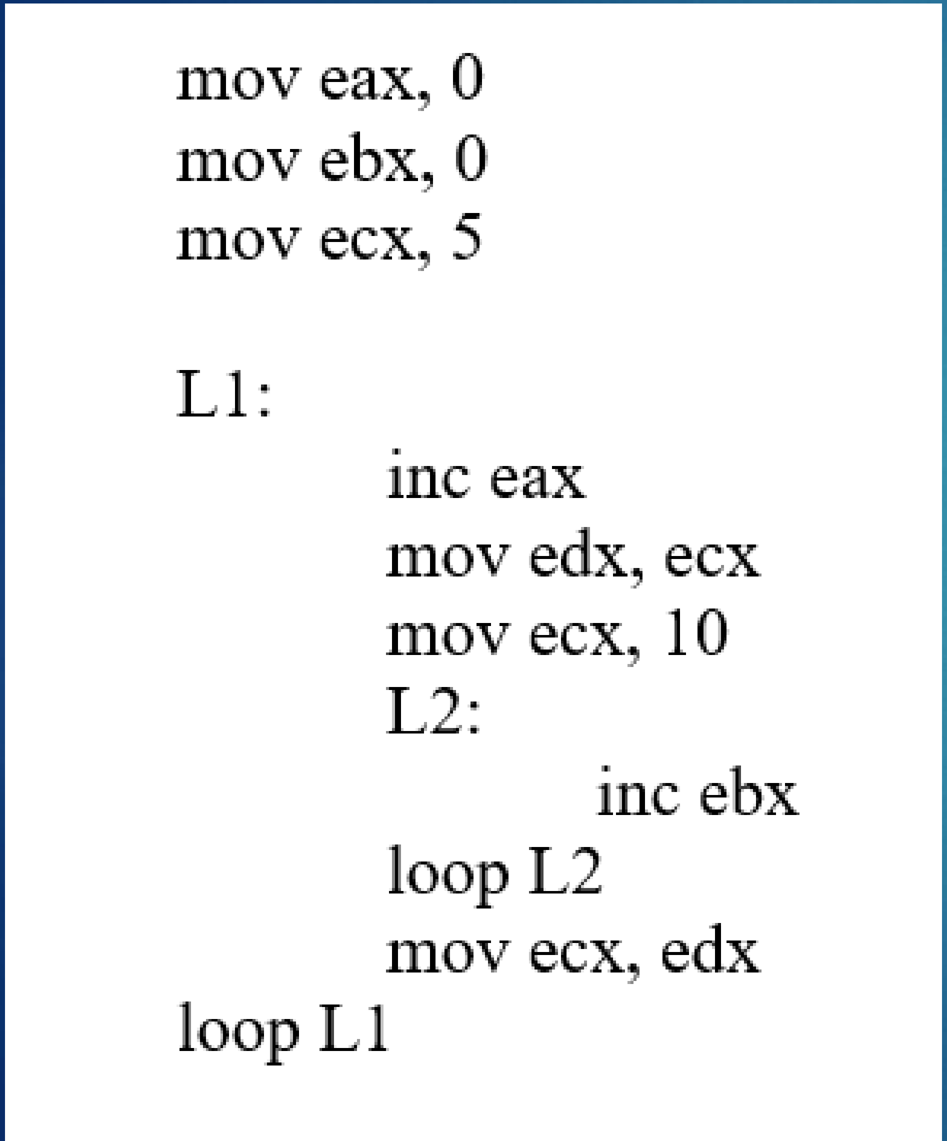
NESTED LOOPS

- 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
```

EXAMPLE



```
mov eax, 0
mov ebx, 0
mov ecx, 5

L1:
        inc eax
        mov edx, ecx
        mov ecx, 10
        L2:
                inc ebx
        loop L2
        mov ecx, edx
loop L1
```

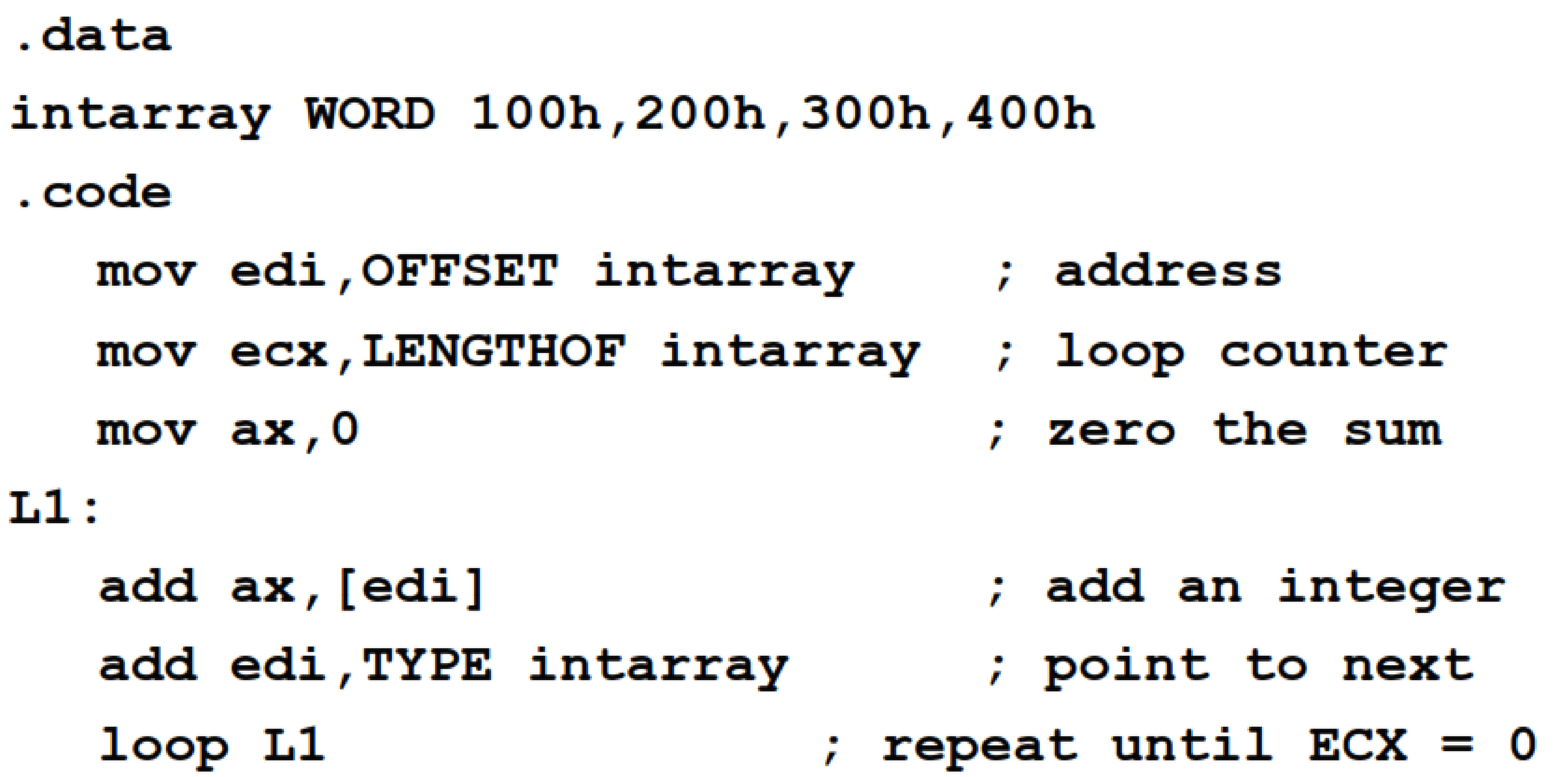
How many times L1 will execute?

How many times L2 will execute?

EXERCISE

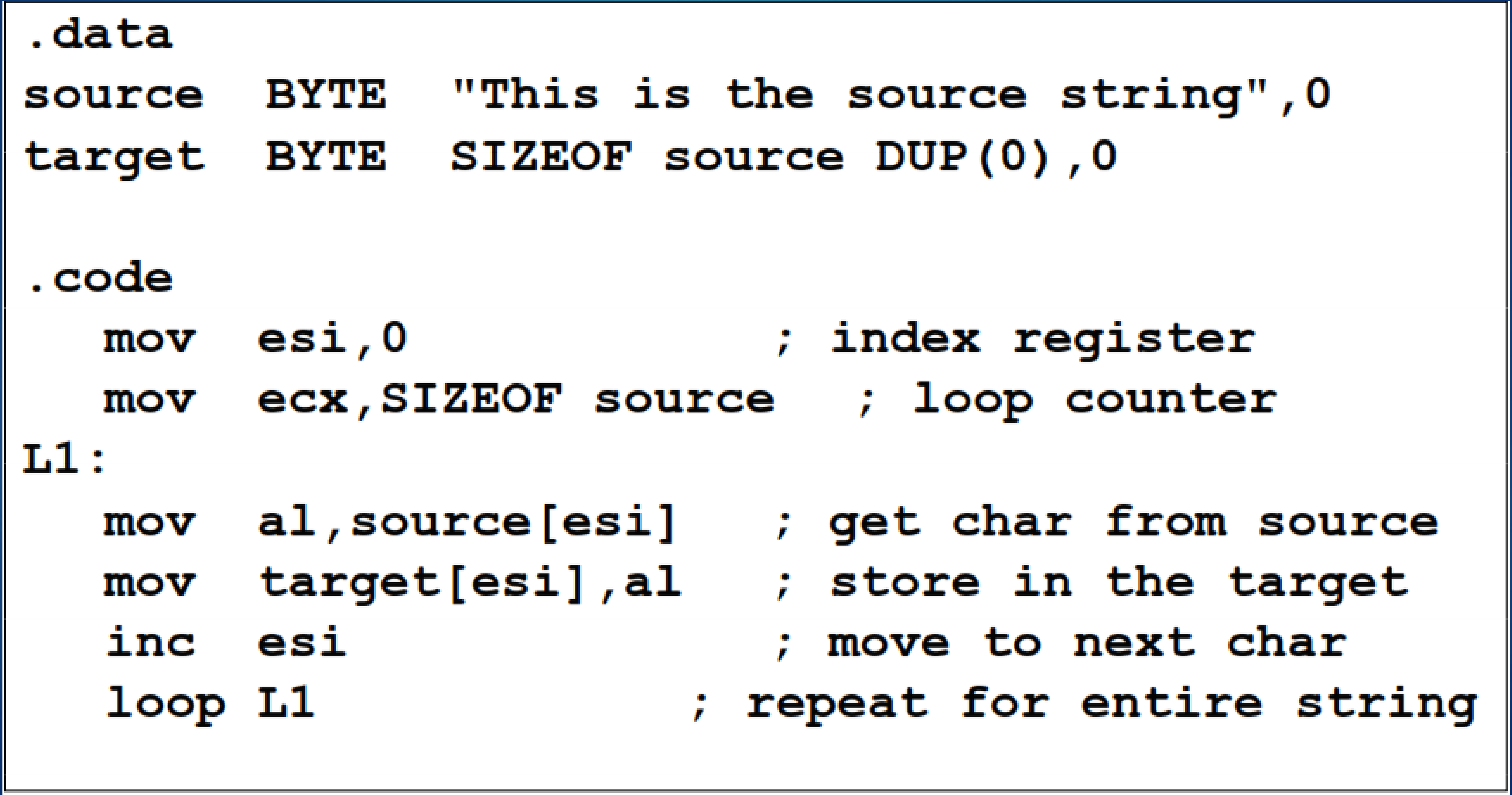
- Write a code to add 16- bit integers Stored in an Pre-defined Array.
- LOOP Instructions must be Used.
- Also use OFFSET, LENGTHOF and TYPE Operators.
- Array must contain 4- unsigned intergers.

SOLUTION



```
.data
intarray WORD 100h,200h,300h,400h
.code
   mov edi,OFFSET intarray      ; address
   mov ecx,LENGTHOF intarray    ; loop counter
   mov ax,0                     ; zero the sum
L1:
   add ax,[edi]                 ; add an integer
   add edi,TYPE intarray        ; point to next
   loop L1                  ; repeat until ECX = 0
```

EXERCISE (Copying a String)



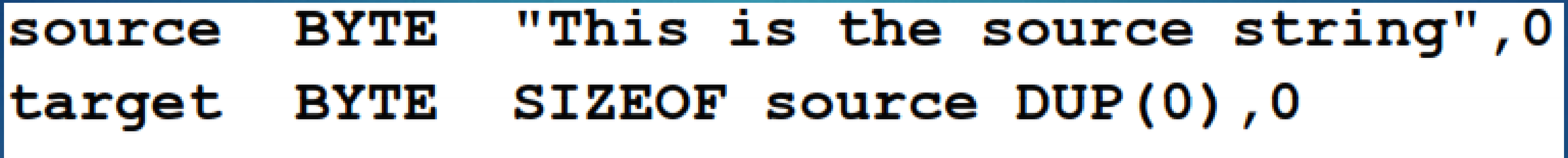
```
.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0),0

.code
   mov  esi,0            ; index register
   mov  ecx,SIZEOF source   ; loop counter
L1:
   mov  al,source[esi]   ; get char from source
   mov  target[esi],al   ; store in the target
   inc  esi              ; move to next char
   loop L1            ; repeat for entire string
```

EXERCISE

- Write a loop that iterates through a doubleword array and calculates the sum of its elements using a scale factor with indexed addressing.

EXERCISE

- 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(0),0
```