

Assembly Language LAB

Islamic University – Gaza

Engineering Faculty

Department of Computer Engineering

2013

ECOM 2125: Assembly Language LAB

Created by: Eng. Ahmed M. Ayash

Modified and Presented By: Eihab S. El-Radie



Lab # 4

Loop Instruction

Objective:

- To know more about Assembly language, such as how to repeat a block of statements using Loop Instructions.

❖ Loop Instruction

The Loop instruction provides a simple way to repeat a block of statements a specific number of times. **ECX** is automatically used as a counter and is decremented each time the loop repeats.

Syntax:

Loop *target*

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, if ECX equals zero, no jump takes place and control passes to the instruction following the loop.

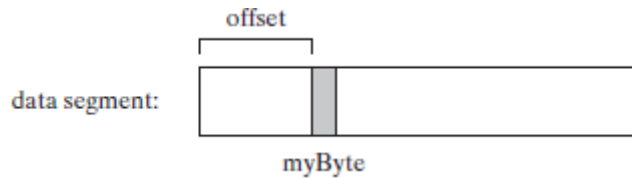
✚ Nested Loop:

If you need to code a loop within a loop, you must save the outer loop counter's ECX value.

```
.data
count dd ?
.code
    mov ecx,value          ; set outer loop count
L1:
    mov count,ecx         ; save outer loop count
    mov ecx,New_value     ; set inner loop count
L2:
    (inner loop code)
    loop L2              ; repeat the inner loop
    mov ecx,count        ; restore outer loop count
    loop L1              ; repeat the outer loop
```

❖ OFFSET Operator

The OFFSET operator returns the offset of a data label. The offset represents the distance, in bytes, of the label from the beginning of the data segment. To illustrate, the following figure shows a variable named **myByte** inside the data segment.



For example:

```
.DATA
bVal BYTE ? ; Assume bVal is at 00404000h
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?
.CODE
mov esi, OFFSET bVal ; ESI = 00404000h
mov esi, OFFSET wVal ; ESI = 00404001h
mov esi, OFFSET dVal ; ESI = 00404003h
mov esi, OFFSET dVal2 ; ESI = 00404007h
```

❖ PTR Operator

You can use the PTR operator to override the declared size of an operand. This is only necessary when you're trying to access the variable using a size attribute that's different from the one used to declare the variable.

Suppose, for example, that you would like to move the lower 16 bits of a doubleword variable named **myDouble** into AX. The assembler will not permit the following move because the operand sizes do not match. But the WORD PTR operator makes it possible to move the low-order word (5678h) to AX:

```
.data
myDouble DWORD 12345678h
.code
mov ax,myDouble ; error
mov ax,WORD PTR myDouble ; ax=5678h
mov ax,WORD PTR [myDouble+2] ; ax=1234h
```

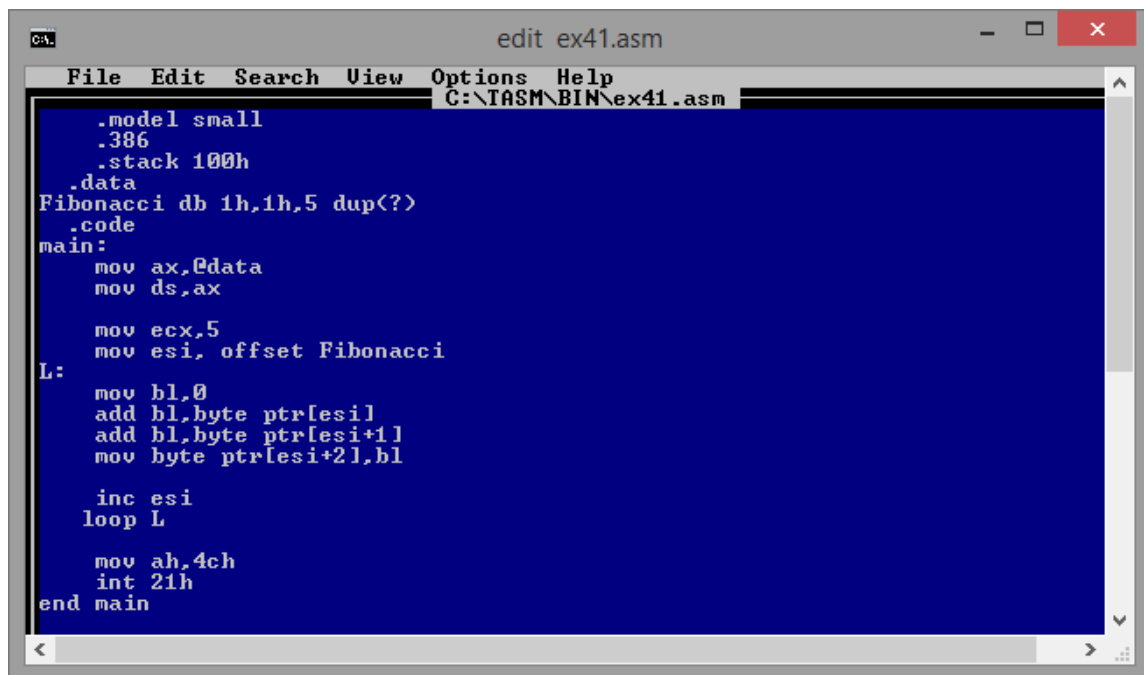
doubleword	word	byte	offset	
12345678	5678	78	0000	myDouble
		56	0001	myDouble + 1
	1234	34	0002	myDouble + 2
		12	0003	myDouble + 3

Lab work:

Excercise1:

Write a program that uses a loop to calculate the first seven values in the Fibonacci number sequence { 1,1,2,3,5,8,13 } where The Rule is $F_n = F_{n-1} + F_{n-2}$. The Fibonacci sequence is referenced in the memory by the byte memory array called Fibonacci save the remaining five elements in the same array.

Fabonacci db 1h,1h,5 dup(?)

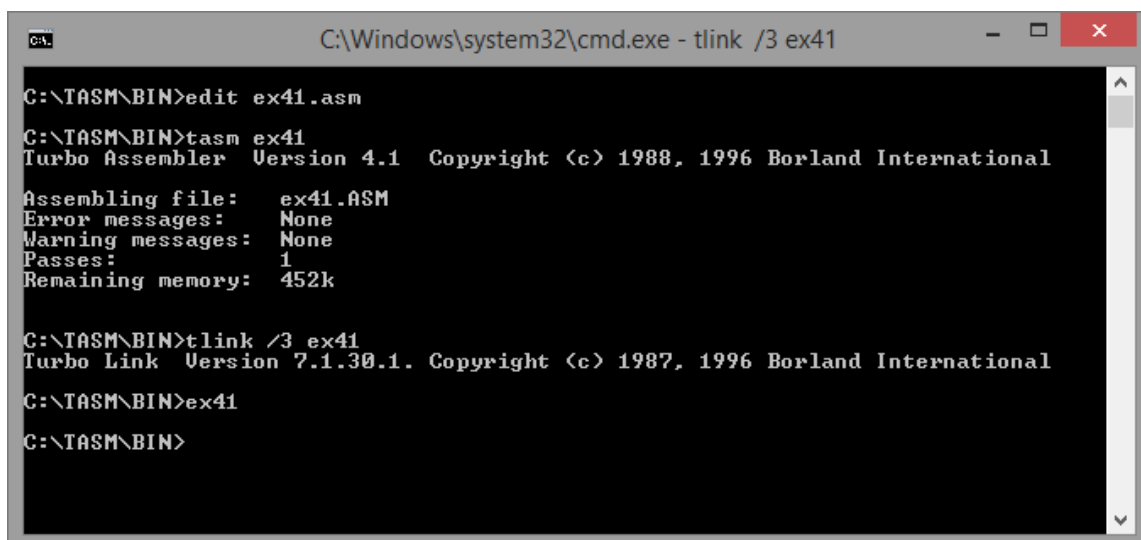


```
edit ex41.asm
File Edit Search View Options Help
C:\TASM\BIN\ex41.asm
.model small
.386
.stack 100h
.data
Fibonacci db 1h,1h,5 dup(?)
.code
main:
mov ax,@data
mov ds,ax

mov ecx,5
mov esi, offset Fibonacci
L:
mov bl,0
add bl,byte ptr[esi]
add bl,byte ptr[esi+1]
mov byte ptr[esi+2],bl

inc esi
loop L

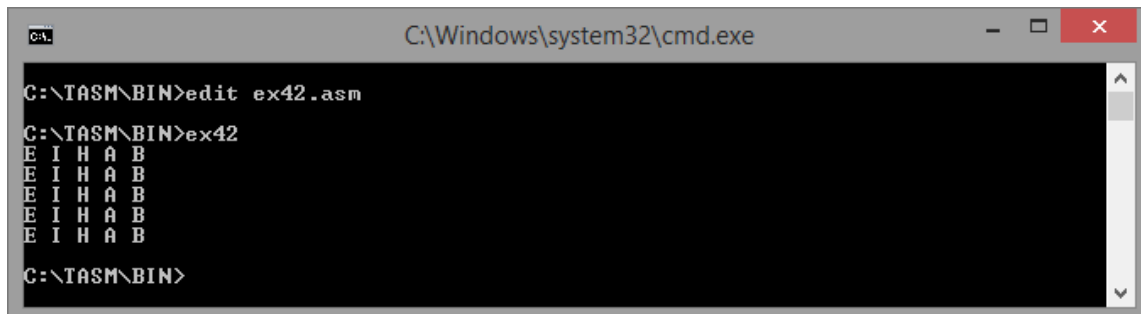
mov ah,4ch
int 21h
end main
```



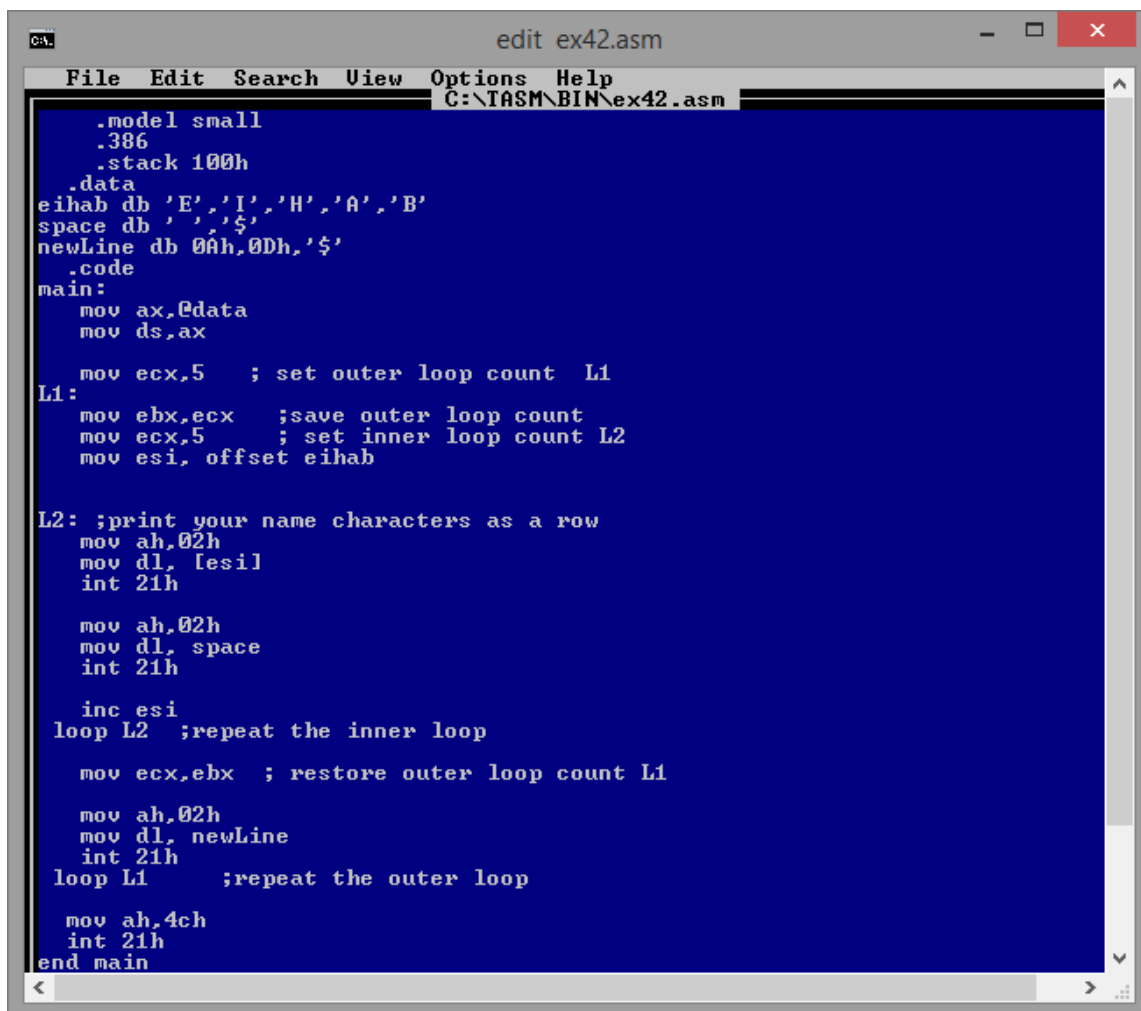
```
C:\Windows\system32\cmd.exe - tlink /3 ex41
C:\TASM\BIN>edit ex41.asm
C:\TASM\BIN>tasm ex41
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International
Assembling file: ex41.ASM
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 452k
C:\TASM\BIN>tlink /3 ex41
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
C:\TASM\BIN>ex41
C:\TASM\BIN>
```

Excercise2:

Write an assembly code that prints the characters of **your name**, 5 times on the screen. Each character followed by space and each full name from 1 to 5 is separated by new line, as the following:



```
C:\Windows\system32\cmd.exe
C:\TASM\BIN>edit ex42.asm
C:\TASM\BIN>ex42
E I H A B
E I H A B
E I H A B
E I H A B
E I H A B
C:\TASM\BIN>
```



```
File Edit Search View Options Help
C:\TASM\BIN\ex42.asm
.model small
.386
.stack 100h
.data
eihab db 'E','I','H','A','B'
space db ' ','$'
newLine db 0Ah,0Dh,'$'
.code
main:
mov ax,@data
mov ds,ax

mov ecx,5 ; set outer loop count L1
L1:
mov ebx,ecx ;save outer loop count
mov ecx,5 ; set inner loop count L2
mov esi, offset eihab

L2: ;print your name characters as a row
mov ah,02h
mov dl, [esi]
int 21h

mov ah,02h
mov dl, space
int 21h

inc esi
loop L2 ;repeat the inner loop

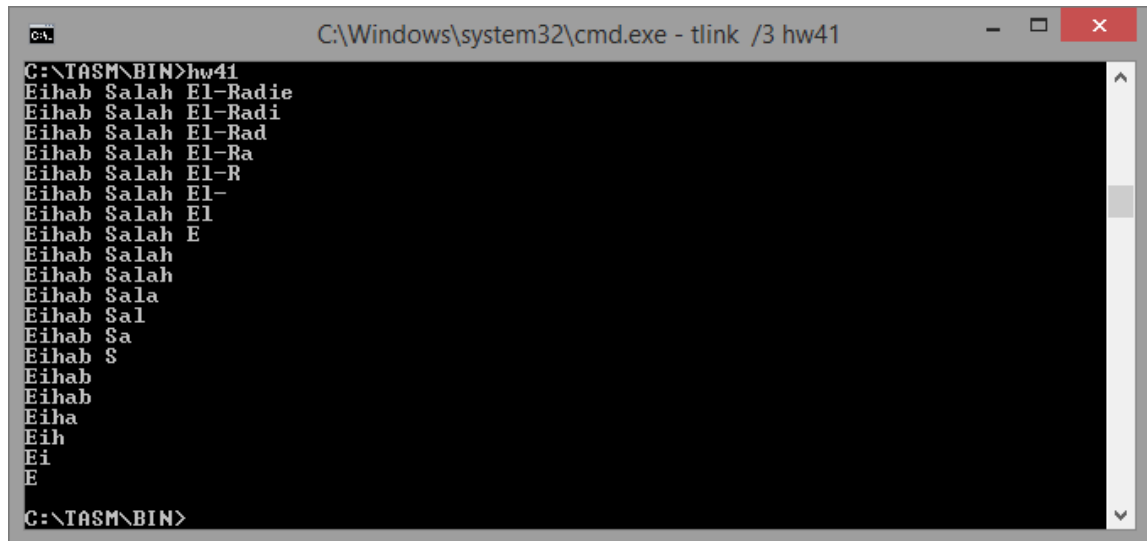
mov ecx,ebx ; restore outer loop count L1

mov ah,02h
mov dl, newLine
int 21h
loop L1 ;repeat the outer loop

mov ah,4ch
int 21h
end main
```

Homework:

1. Write an assembly language program using the Loop instruction to print all letters of **your full name** decreasingly as follows:



```
C:\Windows\system32\cmd.exe - tlink /3 hw41
C:\TASM\BIN>hw41
Eihab Salah El-Radie
Eihab Salah El-Radi
Eihab Salah El-Rad
Eihab Salah El-Ra
Eihab Salah El-R
Eihab Salah El-
Eihab Salah El
Eihab Salah E
Eihab Salah
Eihab Salah
Eihab Sala
Eihab Sal
Eihab Sa
Eihab S
Eihab
Eihab
Eiha
Eih
Ei
E
C:\TASM\BIN>
```

2. Using loop, Use the array odd to find the square of the numbers between 1 and 5 and put the square of each number in the array square:

<pre>odd db 1,3,5,7,9 square db 5 dup(?)</pre>
--

3. Rewrite Exercise1 (Fibonacci number sequence) with just 2 add instructions and without PTR Operator.
4. Rewrite Exercise1 (Fibonacci number sequence) with just 2 add instructions and without PTR and OFFSET Operators.