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 # 2

Assembly Language Fundamentals

Objective:

To be familiar with Assembly Language Fundamentals.

Data Types:

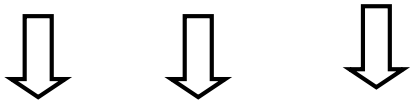
BYTE, SBYTE	**8-bit unsigned integer; 8-bit signed integer**
WORD, SWORD	16-bit unsigned & signed integer
DWORD, SDWORD	32-bit unsigned & signed integer
QWORD	64-bit integer
TBYTE	80-bit integer

Data Definition Statement:

A data definition statement sets aside storage in memory for a variable.

Syntax:

[*name*] *directive initializer* [, *initializer*] . . .



```
val1   BYTE    10
```

db

- All initializers become binary data in memory

We can define a single byte of storage; use multiple initializers, defining Strings.

End-of-line character sequence:

- 0Dh = carriage return
- 0Ah = line feed

```
str1 db "Enter your name:    ",0Dh,0Ah
     db "Enter your address: ",0
newLine db 0Dh,0Ah,0
```

Integer Constants

An *integer constant* (or integer literal) is made up of an optional leading sign, one or more digits and an optional suffix character (called a *radix)* indicating the number's base:

[{+ | - }] ***digits [radix]***

Common radix characters:

h – hexadecimal
d – decimal
b – binary
o – octal

Examples: 30d, 6Ah, 42, 1101b, 777o
Hexadecimal beginning with letter: 0A5h

- If no radix is given, the integer constant is decimal
- A hexadecimal beginning with a letter must have a leading 0

Integer Expressions

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

Character Constants

A *character constant* is a single character enclosed in either single or double quotes. The assembler converts it to the binary ASCII code matching the character. Examples are:

'A'
"x"

ASCII character = 1 byte.

String Constants

String constant is a string of characters enclosed in either single or double quotes:

'ABC'
"xyz"
'Say "Goodnight," Gracie'

Each character occupies a single byte.

Reserved Words

Assembly language has a list of words called *reserved words*. These have special meaning and can only be used in their correct context. Reserved words can be any of the following:

- Instruction mnemonics, such as MOV, ADD or MUL which correspond to built-in operations performed by Intel processors.

- Directives, which tell MASM how to assemble programs.
  - Used to declare code, data areas, select memory model, declare procedures, etc.
  - not case sensitive
  - For example .data, .code, proc.

- Attributes, which provide size and usage information for variables and operands. Examples are BYTE and WORD.

- Operators, used in constant express ions. For example (+, -, …)

- Predefined symbols. Such as **@data** which return constant integer values at assembly time.

Identifiers

- They are not case sensitive.
- The first character must be a letter (A..Z, a..z), underscore (_), @ , ?, or $. Subsequent characters may also be digits.
- An identifier **cannot** be the same as an assembler reserved word.

Comments

Comments can be specified in two ways:

- Single-line comments, beginning with a semicolon character (**;**). All characters following the semicolon on the same line are ignored by the assembler and may be used to comment the program.
- Block comments, beginning with the COMMENT directive and a user-specified symbol.

All subsequent lines of text are ignored by the assembler until the same user-specified symbol appears. For example:

COMMENT ! This line is a comment. This line is also a comment. !	COMMENT & This line is a comment. This line is also a comment. &

Using the DUP Operator:

Use DUP to allocate (create space for) an array or string.

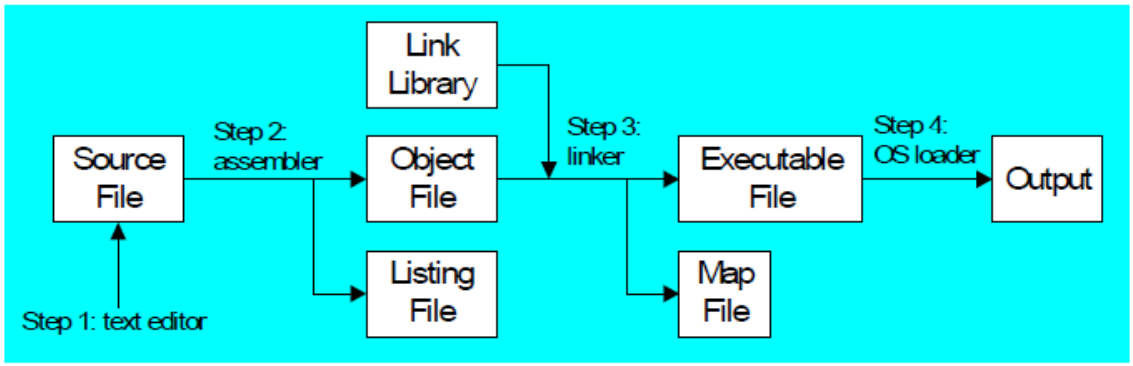
Syntax:
Counter DUP (argument)

Counter and argument must be constants or constant expressions.

```
var1 db 20 DUP(0)                ; 20 bytes, all equal to zero
var2 db 20 DUP(?)                ; 20 bytes, uninitialized
var3 db 4 DUP("STACK")     ; 20 bytes: "STACKSTACKSTACKSTACK"
var4 db 10,3 DUP(0),20     ; 5 bytes
```

Assemble-Link Execute Cycle:

The following diagram describes the steps from creating a source program through executing the compiled program:



Note: If the source code is modified, Steps 2 through 4 must be repeated.

.386:

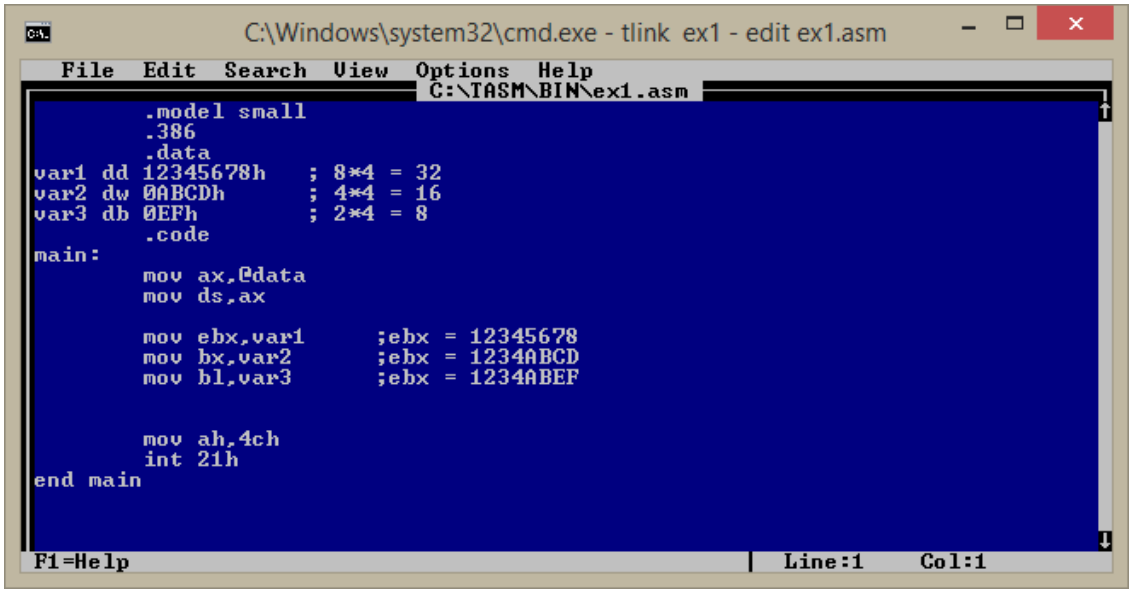
- Enables assembly of nonprivileged instructions for the 80386 processor; disables assembly of instructions introduced with later processors.
- The **.386** directive identifies the minimum CPU required for the program (Intel386).
- Used after the .model directive in our small model. In other models it is used before .model directive.

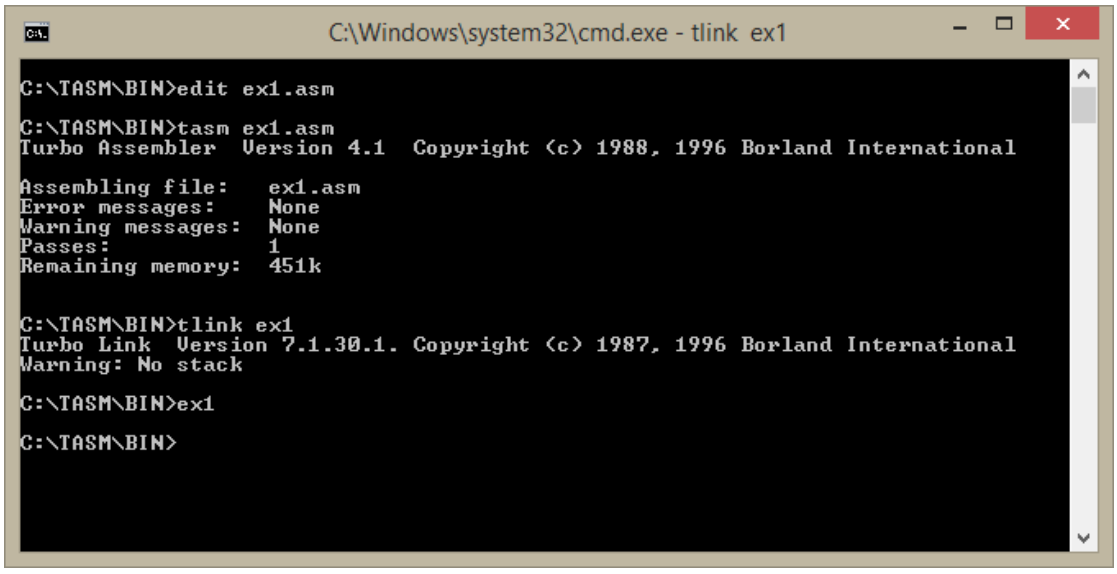
Lab work:

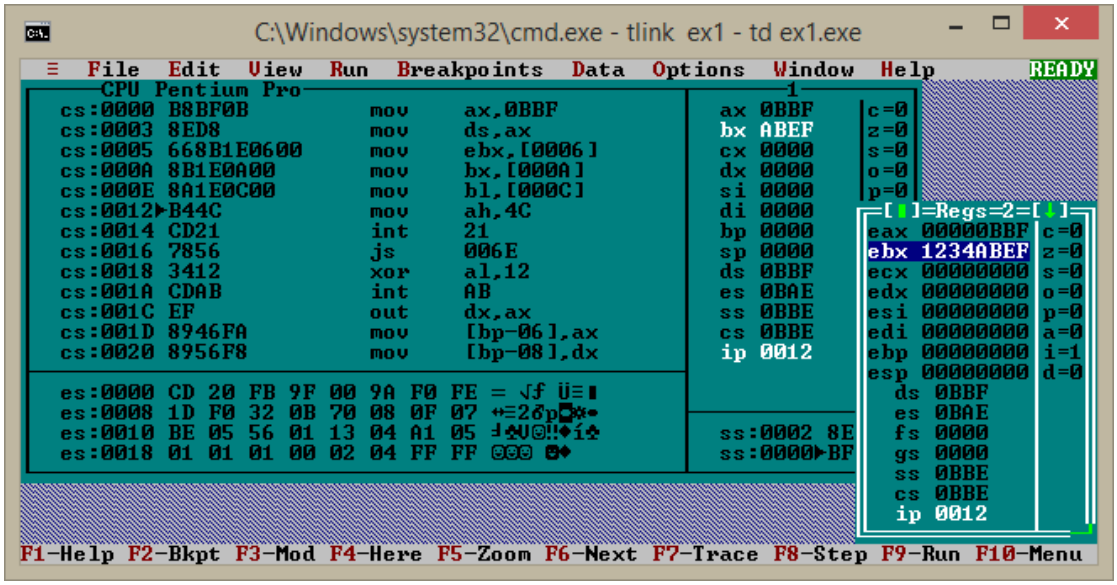
Excercise1:

Define the three variables using data definition statements to initialize ebx to the value 1234ABEF.

var1 = 12345678h var2 = 0ABCDh var3 = 0EFh


C:\Windows\system32\cmd.exe - tlink ex1 - edit ex1.asm
File Edit Search View Options Help
C:\TASM\BIN\ex1.asm
.model small
.386
.data
var1 dd 12345678h ; 8*4 = 32
var2 dw 0ABCDh ; 4*4 = 16
var3 db 0EFh ; 2*4 = 8
.code
main:
mov ax,@data
mov ds,ax
mov ebx,var1 ;ebx = 12345678
mov bx,var2 ;ebx = 1234ABCD
mov bl,var3 ;ebx = 1234ABEF
mov ah,4ch
int 21h
end main
F1=Help
Line:1 Col:1


C:\Windows\system32\cmd.exe - tlink ex1
C:\TASM\BIN>edit ex1.asm
C:\TASM\BIN>tasm ex1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International
Assembling file: ex1.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 451k
C:\TASM\BIN>tlink ex1
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
Warning: No stack
C:\TASM\BIN>ex1
C:\TASM\BIN>

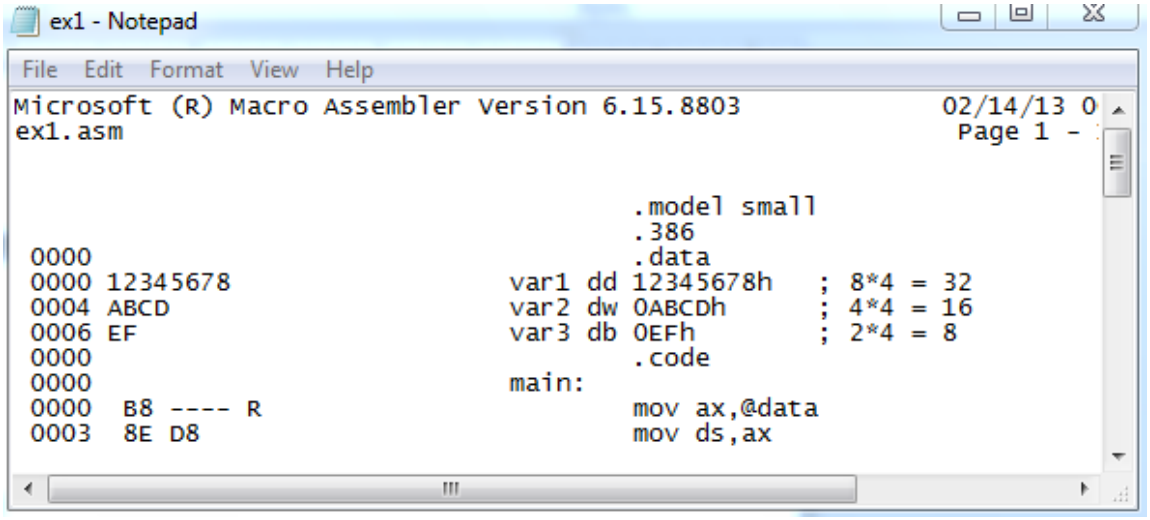

C:\Windows\system32\cmd.exe - tlink ex1 - td ex1.exe
File Edit View Run Breakpoints Data Options Window Help
READY
CPU Pentium Pro
cs:0000 B8BF0B mov ax,0BBF
cs:0003 8ED8 mov ds,ax
cs:0005 668B1E0600 mov ebx,[0006]
cs:000A 8B1E0A00 mov bx,[000A]
cs:000E 8A1E0C00 mov bl,[000C]
cs:0012 B44C mov ah,4C
cs:0014 CD21 int 21
cs:0016 7856 js 006E
cs:0018 3412 xor al,12
cs:001A CDAB int AB
cs:001C EF out dx,ax
cs:001D 8946FA mov [bp-06],ax
cs:0020 8956F8 mov [bp-08],dx
es:0000 CD 20 FB 9F 00 9A F0 FE
es:0008 1D F0 32 0B 70 08 0F 07
es:0010 BE 05 56 01 13 04 A1 05
es:0018 01 01 01 00 02 04 FF FF
ax 0BBF
bx ABEF
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 0BBF
es 0BAE
ss 0BBE
cs 0BBE
ip 0012
c=0
z=0
s=0
o=0
p=0
[]=Regs=2=[]
eax 00000BBF
ebx 1234ABEF
ecx 00000000
edx 00000000
esi 00000000
edi 00000000
ebp 00000000
esp 00000000
ds 0BBF
es 0BAE
fs 0000
gs 0000
ss 0BBE
cs 0BBE
ip 0012
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Listing File:

Use it to see how your program is compiled

Contains: source code, addresses, object code (machine language), symbols (variables, procedures)

Example: ex1.lst



Map File:

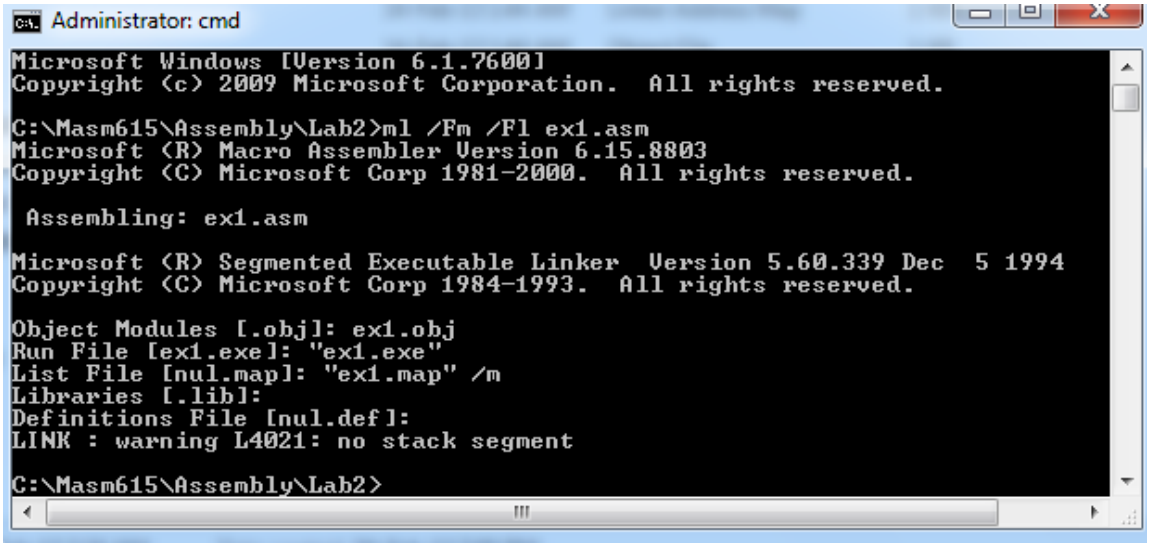
Contain Information about each program segment: starting address, ending address, size

To generate map file and list file do as follows:

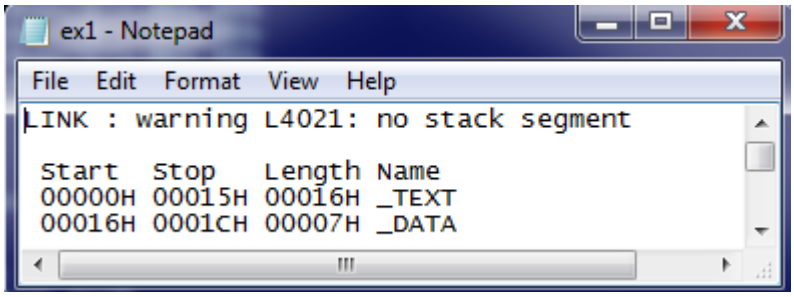
> ml /Fm /Fl ex1.asm

Note:

1. Ml.exe needs Link.exe to work. You can find the link file in C:\Masm615 folder.
2. /Fm[File] Generate Map.
3. /Fl[File] Generate Listing.



Example: ex1.map



Note:

1. Note that in our all previous programs, we have a warning that there is no stack segment, so from now we will define the stack segment in the next codes.

2. To define stack segments do as follows:

```
.STACK [[size]]
```

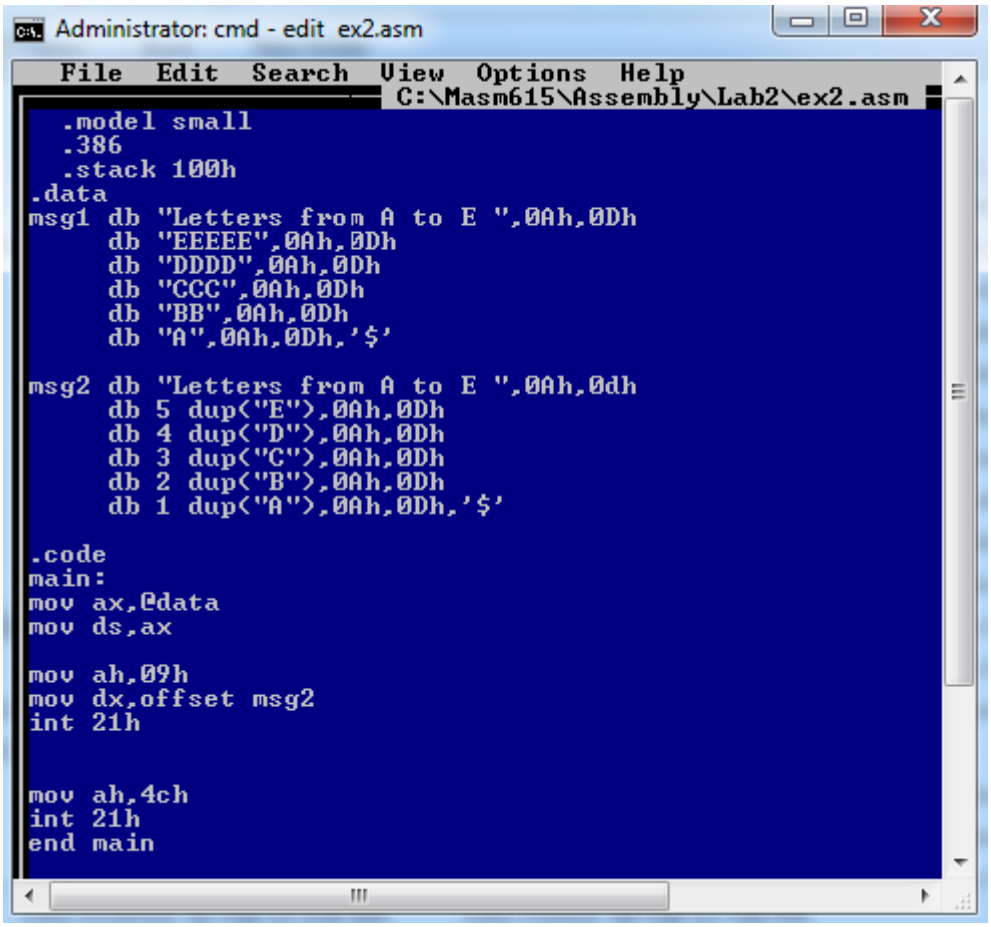
The optional size specifies the number of bytes for the stack (default 1,024)

Exercise2:

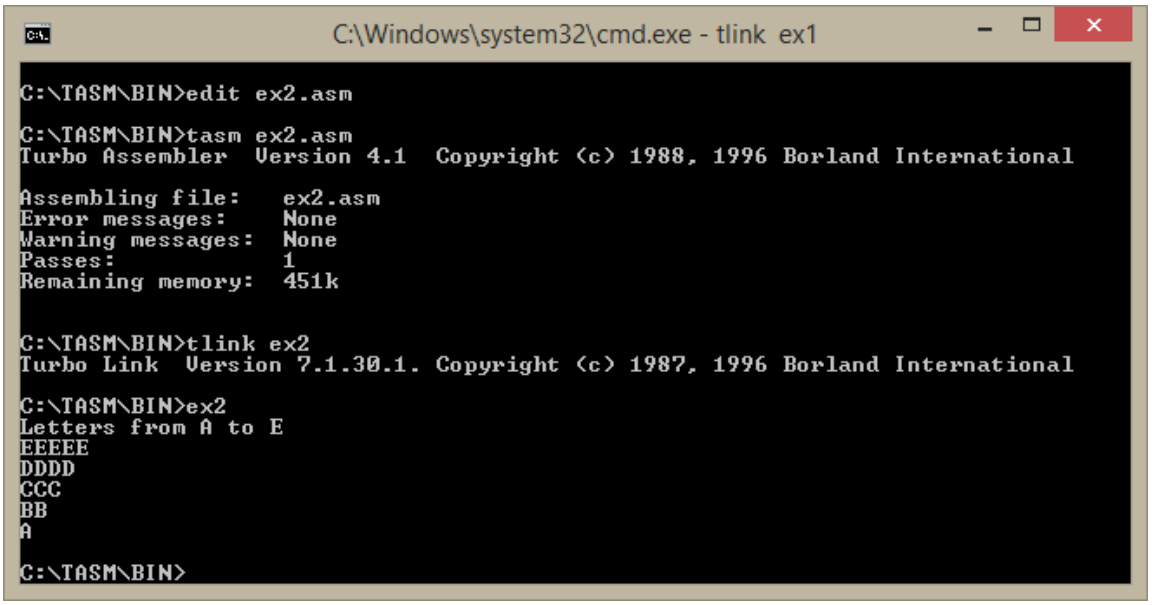
What is the output of the following Assembly language program?

1- Using msg1

2- Using msg2



Output: (The two messages will print the same output)



Homework:

1. Print the following output on the command window using the DUP Operator:

```
*
**
***
****
*****
```

2. Write your name on the console (using 0Ah = line feed) as the following:

```
Eihab
      Salah
            El-Radie
```

3. Write an assembly language program that moves in ebx value 12344321 and moves in dx value 4334 using the variables String1 and String2 stored in memory.

```
String1 dd 12343412h
String2 dd 43214321h
```