# TASM TUTORIAL

## - Building Assembler programs

In order to be able to create a program, several tools are needed:

First an editor to create the source program. Second a compiler, which is nothing more than a program that "translates" the source program into an object program. And third, a linker that generates the executable program from the object program.

The editor can be any text editor at hand, and as a compiler we will use the TASM macro assembler from Borland, and as a linker we will use the Tlink program.

The extension used so that TASM recognizes the source programs in assembler is .ASM; once translated the source program, the TASM creates a file with the .OBJ extension, this file contains an "intermediate format" of the program, called like this because it is not executable yet but it is not a program in source language either anymore. The linker generates, from a .OBJ or a combination of several of these files, an executable program, whose extension usually is .EXE though it can also be .COM, depending of the form it was assembled.

## - Assembler Programming

To build assembler programs using TASM programs is a different program structure than from using debug program.

It's important to include the following assembler directives:

*.MODEL SMALL*
Assembler directive that defines the memory model to use in the program

*.CODE*
Assembler directive that defines the program instructions

*.STACK*
Assembler directive that reserves a memory space for program instructions in the stack.

*END*
Assembler directive that finishes the assembler program.

**First step**

```
; use ; to put comments in the assembler program
.MODEL SMALL ;memory model
.STACK ;memory space for program instructions in the stack
.CODE ;the following lines are program instructions
mov ah,1h ;moves the value 1h to register ah
mov cx,07h ;moves the value 07h to register cx
int 10h ;10h interruption
mov ah,4ch ;moves the value 4 ch to register ah
int 21h ;21h interruption
END ;finishes the program code
```

This assembler program changes the size of the computer cursor.

**Second step**

Save the file with the following name: examp1.asm Don't forget to save this in ASCII format.

**Third step**

Use the TASM program to build the object program.

Example:

```
C:\>tasm exam1.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland
International

Assembling file: exam1.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 471k
```

The TASM can only create programs in .OBJ format, which are not executable by themselves, but rather it is necessary to have a linker which generates the executable code.

**Fourth step**

Use the TLINK program to build the executable program example:

```
C:\>tlink exam1.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland
International

C:\>
```

Where exam1.obj is the name of the intermediate program, .OBJ. This generates a file directly with the name of the intermediate program and the .EXE extension.

**Fifth step**

Execute the executable program

```
C:\>exam1[enter]
```

Remember, this assembler program changes the size of the cursor.

**- Assembly process.**

**Segments**

The architecture of the x86 processors forces to the use of memory segments to manage the information, the size of these segments is of 64kb.

The reason of being of these segments is that, considering that the maximum size of a number that the processor can manage is given by a word of 16 bits or register, it would not be possible to access more than 65536 localities of memory using only one of these registers, but now, if the PC's memory is divided into groups or segments, each one of 65536 localities, and we use an address on an exclusive register to find each segment, and then we make each address of a specific slot with two registers, it is possible for us to access a quantity of 4294967296 bytes of memory, which is, in the present day, more memory than what we will see installed in a PC.

In order for the assembler to be able to manage the data, it is necessary that each piece of information or instruction be found in the area that corresponds to its respective segments. The assembler accesses this information taking into account the localization of the segment, given by the DS, ES, SS and CS registers and inside the register the address of the specified piece of information. It is because of this that when we create a program using the Debug on each line that we assemble, something like this appears:

```
1CB0:0102 MOV AX,BX
```

Where the first number, 1CB0, corresponds to the memory segment being used, the second one refers to the address inside this segment, and the instructions which will be stored from that address follow. The way to indicate to the assembler with which of the segments we will work with is with the .CODE, .DATA and .STACK directives.

The assembler adjusts the size of the segments taking as a base the number of bytes each assembled instruction needs, since it would be a waste of memory to use the whole segments. For example, if a program only needs 10kb to store data, the data segment will only be of 10kb and not the 64kb it can handle.

**Table of Symbols**

Each one of the parts on code line in assembler is known as token, for example on the code line:

```
MOV AX,Var
```

we have three tokens, the MOV instruction, the AX operator, and the VAR operator. What the assembler does to generate the OBJ code is to read each one of the tokens and look for it on an internal "equivalence" chart known as the reserved words chart, which is where all the mnemonic meanings we use as instructions are found.

Following this process, the assembler reads MOV, looks for it on its chart and identifies it as a processor instruction. Likewise it reads AX and recognizes it as a register of the processor, but when it looks for the Var token on the reserved words chart, it does not find it, so then it looks for it on the symbols chart which is a table where the names of the variables, constants and labels used in the program where their addresses on memory are included and the sort of data it contains, are found.

Sometimes the assembler comes on a token which is not defined on the program, therefore what it does in these cased is to pass a second time by the source program to verify all references to that symbol and place it on the symbols chart.There are symbols which the assembler will not find since they do not belong to that segment and the program does not know in what part of the memory it will find that segment, and at this time the linker comes into action, which will create the structure necessary for the loader so that the segment and the token be defined when the program is loaded and before it is executed.

**Another example**

**First step**
use any editor program to create the source file. Type the following lines:

```
;example11
.model small
.stack
.code
mov ah,2h ;moves the value 2h to register ah
mov dl,2ah ;moves de value 2ah to register dl
            ;(Its the asterisk value in ASCII format)
int 21h ;21h interruption
mov ah,4ch ;4ch function, goes to operating system
int 21h ;21h interruption
end ;finishes the program code
```

**Second step**

Save the file with the following name: exam2.asm Don't forget to save this in ASCII format.

**Third step**

Use the TASM program to build the object program.

```
C:\>tasm exam2.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland
International

Assembling file: exam2.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 471k
```

Fourth step

Use the TLINK program to build the executable program

```
C:\>tlink exam2.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland
International

C:\>
```

**Fifth step**

Execute the executable program

```
C:\>ejem11[enter]
*
C:\>
```

This assembler program shows the asterisk character on the computer screen.