MS-DOS in-built Assembler Lab: DEBUG

Presentation

The document you are looking at has the primordial function of introducing you to assembly language programming and it has been thought for those people who have never worked with this language.

The tutorial is completely focused towards the computers that function with processors of the x86 family of Intel and considering that the language bases its functioning on the internal resources of the processor the described examples are not compatible with any other architecture.

The information was structured in units in order to allow easy access to each of the topics and facilitate the following of the tutorial.

In the introductory section some of the elemental concepts regarding computer systems are mentioned along with the concepts of the assembly language itself and continues with the tutorial itself.

Why learn assembler language?

The first reason to work with assembler is that it provides the opportunity of knowing more the operation of your PC which allows the development of software in a more consistent manner.
The second reason is the total control of the PC which you can have with the use of the assembler.
Another reason is that the assembly programs are quicker, smaller, and have larger capacities than ones created with other languages.
Lastly the assembler allows an ideal optimization in programs be it on their size or on their execution.

Basic description of a computer system

This section has the purpose of giving a brief outline of the main components of a computer system at a basic level which will allow the user a greater understanding of the concepts which will be dealt with throughout the tutorial.
We call computer system to the complete configuration of a computer including the peripheral units and the system programming which make it a useful and functional machine for a determined task.

Central Processor

This part is also known as central processing unit or CPU which in turn is made by the control unit and the arithmetic and logic unit. Its functions consist in reading and writing the contents of the memory cells to forward data between memory cells and special registers and decode and execute the instructions of a program. The processor has a series of memory cells which are used very often and thus are part of the CPU. These cells are known with the name of registers. A processor may have one or two dozen of these registers. The arithmetic and logic unit of the CPU realizes the operations related with numeric and symbolic calculations. Typically these units only have capacity of performing very elemental operations such as: the addition and subtraction of two whole numbers whole number multiplication and division handling of the registers' bits and the comparison of the content of two registers. Personal computers can be classified by what is known as word size this is the quantity of bits which the processor can handle at a time.

Central Memory

It is a group of cells now being fabricated with semi-conductors used for general processes such as the execution of programs and the storage of information for the operations.
Each one of these cells may contain a numeric value and they have the property of being addressable this is that they can distinguish one from another by means of a unique number or an address for each cell.

The generic name of these memories is Random Access Memory or RAM. The main disadvantage of this type of memory is that the integrated circuits lose the information they have stored when the electricity flow is interrupted. This was the reason for the creation of memories whose information is not lost when the system is turned off. These memories receive the name of Read Only Memory or ROM.

Input and Output Units

In order for a computer to be useful to us it is necessary that the processor communicates with the exterior through interfaces which allow the input and output of information from the processor and the memory. Through the use of these communications it is possible to introduce information to be processed and to later visualize the processed data.
Some of the most common input units are keyboards and mice. The most common output units are screens and printers.

Auxiliary Memory Units

Since the central memory of a computer is costly and considering today's applications it is also very limited. Thus the need to create practical and economical information storage systems arises. Besides, the central memory loses its content when the machine is turned off therefore making it inconvenient for the permanent storage of data.
These and other inconveniences give place for the creation of peripheral units of memory which receive the name of auxiliary or secondary memory. The most common are the tapes and magnetic discs.

Assembler language Basic concepts

Information Units

In order for the PC to process information it is necessary that this information be in special cells called registers. The registers are groups of 8 or 16 flip-flops.
A flip-flop is a device capable of storing two levels of voltage a low one regularly 0.5 volts and another one commonly of 5 volts. The low level of energy in the flip-flop is interpreted as 0 and the high level as 1. These states are usually known as bits and are the smallest information unit in a computer.
A group of 16 bits is known as word; a word can be divided in groups of 8 bits called bytes and the groups of 4 bits are called nibbles.

Numeric systems

The numeric system we use daily is the decimal system but this system is not convenient for machines since the information is handled codified in the shape of on or off bits; this way of codifying takes us to the necessity of knowing the positional calculation which will allow us to express a number in any base where we need it.

Converting binary numbers to decimals

When working with assembly language we come on the necessity of converting numbers from the binary system which is used by computers to the decimal system used by people.
The binary system is based on only two conditions or states be it on (1) or off (0) thus its base is two.
For the conversion we can use the positional value formula. E.g. if we have the binary number of 10011 we take each digit from right to left and multiply it by the base elevated to the new position they are:

Binary: 1 1 0 0 1
Decimal: 1*2^0 + 1*2^1 + 0*2^2 + 0*2^3 + 1*2^4 = 1 + 2 + 0 + 0 + 16 = 19 decimal.

The ^ character is used in computation as an exponent symbol and the * character is used to represent multiplication.

Converting decimal numbers to binary

There are several methods to convert decimal numbers to binary; only one will be analyzed here.
Naturally a conversion with a scientific calculator is much easier but one cannot always count with one so it is convenient to at least know one formula to do it.
The method that will be explained uses the successive division of two keeping the residue as a binary digit and the result as the next number to divide.

Let us take for example the decimal number of 43.

43/2=21 and its residue is 1
21/2=10 and its residue is 1
10/2=5 and its residue is 0
5/2=2 and its residue is 1
2/2=1 and its residue is 0
1/2=0 and its residue is 1

Building the number from the bottom we get that the binary result is: 101011

Hexadecimal system

On the hexadecimal base we have 16 digits which go from 0 to 9 and from the letter A to the F these letters represent the numbers from 10 to 15. Thus we count 0 1 2 3 4 5 6 7 8 9 A B C D E and F.

The conversion between binary and hexadecimal numbers is easy. The first thing to do is a conversion of a binary number to a hexadecimal is to divide it in groups of 4 bits beginning from the right to the left. In case the last group the one most to the left is under 4 bits the missing places are filled with zeros.

For example the binary number of 101011, we divide it in 4 bits groups and we are left with: 10;1011

Filling the last group with zeros (the one from the left): 0010;1011

Afterwards we take each group as an independent number and we consider its decimal value:

0010=2;1011=11

But since we cannot represent this hexadecimal number as 211 because it would be an error we have to substitute all the values greater than 9 by their respective representation in hexadecimal with which we obtain:

2Bh where the h represents the hexadecimal base.

In order to convert a hexadecimal number to binary it is only necessary to invert the steps: the first hexadecimal digit is taken and converted to binary and then the second and so on.

Data representation methods in a computer

ASCII code

ASCII is an acronym of American Standard Code for Information Interchange. This code assigns the letters of the alphabet decimal digits from 0 to 9 and some additional symbols a binary number of 7 bits putting the 8th bit in its off state or 0. This way each letter digit or special character occupies one byte in the computer memory.

We can observe that this method of data representation is very inefficient on the numeric aspect since in binary format one byte is not enough to represent numbers from 0 to 255 but on the other hand with the ASCII code one byte may represent only one digit. Due to this inefficiency the ASCII code is mainly used in the memory to represent text.

BCD Method

BCD is an acronym of Binary Coded Decimal. In this notation groups of 4 bits are used to represent each decimal digit from 0 to 9. With this method we can represent two digits per byte of information.

Even when this method is much more practical for number representation in the memory compared to the ASCII code it still less practical than the binary since with the BCD method we can only represent digits from 0 to 99. On the other hand in binary format we can represent all digits from 0 to 255.

This format is mainly used to represent very large numbers in mercantile applications since it facilitates operations avoiding mistakes.

Floating point representation

This representation is based on scientific notation this is to represent a number in two parts: its base and its exponent.

As an example the number 1234000 can be represented as 1.123*10^6 in this last notation the exponent indicates to us the number of spaces that the decimal point must be moved to the right to obtain the original result.

In case the exponent was negative it would be indicating to us the number of spaces that the decimal point must be moved to the left to obtain the original result.

Using Debug program

Program creation process

For the creation of a program it is necessary to follow five steps:

1. Design of the algorithm stage the problem to be solved is established and the best solution is proposed creating schematic diagrams used for the better solution proposal.
2. Coding the algorithm consists in writing the program in some programming language; assembly language in this specific case taking as a base the proposed solution on the prior step.
3. Translation to machine language is the creation of the object program in other words the written program as a sequence of zeros and ones that can be interpreted by the processor.
4. Test the program after the translation the program into machine language executes the program in the computer machine.
5. The last stage is the elimination of detected faults on the program on the test stage. The correction of a fault normally requires the repetition of all the steps from the first or second.

CPU Registers

The CPU has 4 internal registers each one of 16 bits. The first four AX BX CX and DX are general use registers and can also be used as 8 bit registers if used in such a way it is necessary to refer to them for example as: AH and AL which are the high and low bytes of the AX register. This nomenclature is also applicable to the BX CX and DX registers.

The registers known by their specific names:

AX	Accumulator
BX	Base register
CX	Counting register
DX	Data register
DS	Data segment register
ES	Extra segment register
SS	Stack segment register
CS	Code segment register
BP	Base pointers register
SI	Source index register
DI	Destiny index register
SP	Stack pointer register
IP	Next instruction pointer register
F	Flag register

Debug program

To create a program in assembler two options exist the first one is to use the MASM or Macro Assembler of Microsoft and the second one is to use the debugger - on this first section we will use this last one since it is found in any PC with the MS-DOS which makes it available to any user who has access to a machine with these characteristics.

Debug can only create files with a .COM extension and because of the characteristics of these kinds of programs they cannot be larger that 64 kB and they also must start with displacement offset or 0100H memory direction inside the specific segment.

Debug provides a set of commands that lets you perform a number of useful operations:
A - Assemble symbolic instructions into machine code
D - Display the contents of an area of memory
E - Enter data into memory beginning at a specific location
G - Run the executable program in memory
N - Name a program
P - Proceed or execute a set of related instructions
Q - Quit the debug program
R - Display the contents of one or more registers
T - Trace the contents of one instruction
U - Unassembled machine code into symbolic code
W - Write a program onto disk

It is possible to visualize the values of the internal registers of the CPU using the Debug program. To begin working with Debug type the following prompt in your computer:

```
C:/>Debug [Enter]
```

On the next line a dash will appear this is the indicator of Debug at this moment the instructions of Debug can be introduced using the following command:

```
-r[Enter]

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0D62 ES=0D62 SS=0D62 CS=0D62 IP=0100 NV EI PL NZ NA PO NC
0D62:0100 2E CS:0D62:0101 803ED3DF00 CMP BYTE PTR [DFD3] 00 CS:DFD3=03
```

All the contents of the internal registers of the CPU are displayed; an alternative of viewing them is to use the "r" command using as a parameter the name of the register whose value we want to se. E.g. :

```
-rbx
BX 0000
:
```

This instruction will only display the content of the BX register and the Debug indicator changes from - to ":"

When the prompt is like this it is possible to change the value of the register which was seen by typing the new value and [Enter] or the old value can be left by pressing [Enter] without typing any other value.

Assembler structure

In assembly language code lines have two parts the first one is the name of the instruction which is to be executed and the second one are the parameters of the command. For example:

```
add ah, bh
```

Here "add" is the command to be executed in this case an addition and "ah" as well as "bh" are the parameters. For example:

```
mov al, 25
```

In the above example we are using the instruction mov it means move the value 25 to al register.
The name of the instructions in this language is made of two three or four letters. These instructions are also called mnemonic names or operation codes since they represent a function the processor will perform. Sometimes instructions are used as follows:

```
add al, [170]
```

The brackets in the second parameter indicate to us that we are going to work with the content of the memory location 170 and not with the 170 value; this is known as direct addressing.

Creating basic assembler program

The first step is to initiate the Debug this step only consists of typing debug [Enter] at the prompt.

To assemble a program on the Debug the "a" (assemble) command is used; when this command is used the address where you want the assembling to begin can be given as a parameter if the parameter is omitted the assembling will be initiated at the locality specified by CS:IP usually 0100h which is the locality where programs with .COM extension must be initiated. And it will be the place we will use since only Debug can create this specific type of programs.

Even though at this moment it is not necessary to give the "a" command a parameter it is recommendable to do so to avoid problems once the CS:IP registers are used therefore we type:

```
a 100 [enter]
mov ax, 0002 [enter]
mov bx, 0004 [enter]
add ax, bx [enter]
nop [enter] [enter]
```

What does the program do? move the value 0002 to the ax register move the value 0004 to the bx register add the contents of the ax and bx registers the instruction no operation to finish the program.

In the debug program; the following appears on the screen:

```
C:\>debug
-a 100
0D62:0100 mov ax, 0002
0D62:0103 mov bx, 0004
0D62:0106 add ax, bx
0D62:0108 nop
0D62:0109
```

Type the command "t" (trace) to execute each instruction of this program example:

```
-t [enter]
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000 DS=0D62 ES=0D62 SS=0D62
CS=0D62 IP=0103 NV EI PL NZ NA PO NC 0D62:0103 BB0400 MOV BX 0004
```

You see that the value 2 move to AX register. Type the command "t" (trace) again and you see the second instruction is executed.

```
-t [enter]
AX=0002 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000 DS=0D62 ES=0D62 SS=0D62
CS=0D62 IP=0106 NV EI PL NZ NA PO NC 0D62:0106 01D8 ADD AX BX
```

Type the command "t" (trace) to see the instruction add is executed you will see the follow lines:

```
-t [enter]
AX=0006 BX=0004 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000 DS=0D62 ES=0D62 SS=0D62
CS=0D62 IP=0108 NV EI PL NZ NA PE NC 0D62:0108 90 NOP
```

The possibility that the registers contain different values exists but AX and BX must be the same since they are the ones we just modified.

To exit Debug use the "q" (quit) command.

Storing and loading the programs

It would not seem practical to type an entire program each time it is needed and to avoid this it is possible to store a program on the disk with the enormous advantage that by being already assembled it will not be necessary to run Debug again to execute it.

The steps to save a program that it is already stored on memory are:

Obtain the length of the program subtracting the final address from the initial address naturally in hexadecimal system.
Give the program a name and extension.
Put the length of the program on the CX register.
Tell Debug to write the program on the disk.

By using as an example the following program we will have a clearer idea of how to take these steps:

When the program is finally assembled it would look like this:

```
0C1B:0100 mov ax, 0002
0C1B:0103 mov bx, 0004
0C1B:0106 add ax, bx
0C1B:0108 int 20
0C1B:010A
```

To obtain the length of a program the "h" command is used since it will show us the addition and subtraction of two numbers in hexadecimal. To obtain the length of ours we give it as parameters the value of our program's final address (10A) and the program's initial address (100). The first result the command shows us is the addition of the parameters and the second is the subtraction.

```
-h 10a, 100
020a 000a
```

The "n" command allows us to name the program.

```
-n test.com
```

The "rcx" command allows us to change the content of the CX register to the value we obtained from the size of the file with "h" in this case 000a since the result of the subtraction of the final address from the initial address.

```
-rcx
CX 0000
:000a
```

Lastly the "w" command writes our program on the disk indicating how many bytes it wrote.

```
-w
Writing 000A bytes
```

To save an already loaded file two steps are necessary:

Give the name of the file to be loaded. Load it using the "l" (load) command.

To obtain the correct result of the following steps it is necessary that the above program be already created.

Inside Debug we write the following:

```
-n test.com
-l
-u 100 109
0C3D:0100 B80200      MOV AX, 0002
0C3D:0103 BB0400      MOV BX, 0004
0C3D:0106 01D8        ADD AX, BX
0C3D:0108 CD20        INT 20
```

The last u command is used to verify that the program was loaded on memory. What it does is that it disassembles the code and shows it disassembled. The parameters indicate to Debug from where and to where to disassemble.

Debug always loads the programs on memory on the address 100H otherwise indicated.