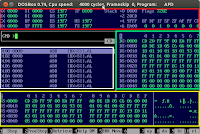
8086 Assembly Debugging with AFD - Advance Full Screen Debug - PART 2

http://wetolearn.blogspot.com/2013/10/8086-assembly-debugging-with-afd-advance.html

In the [last tutorial](http://wetolearn.blogspot.com/2013/09/8086-assembly-debugging-with-afd.html) we have covered some basic features of AFD  to get you started . Lets look at some advance features that will make debugging 16 bit assembly programs less painful.

[](http://4.bp.blogspot.com/-HR7DrUIcnx0/Ujir3mkEl5I/AAAAAAAAAIU/g44-MpaP-Iw/s1600/AFD.png)This image is from last tutorial in which we divided the AFD display into four regions.  
1- Registers  
2- Disassembly  
3 & 4 - Memory Window 1 & 2

The features are explained with examples ( where necessary ) below.

**Navigation**

The very first thing that I will write about is how to change content of these regions without typing a command in CMD.  
To do that, you can simply navigate through these regions using F7 , F8 , F9 and F10 keys  
The functions of these keys are as follow

* F7   => Move To Region Above
* F8   => Move To Region Below
* F9   => Move To Region on Left
* F10 => Move To Region on Right

After you are in any of these regions you can use arrows key to go to specific position and then change content respectively.

**Changing Code on the fly**

Lets suppose, while debugging in AFD, you find a bug and according to you it can be resolved by editing just 2 lines but you will have to assemble the asm file again and repeat all the steps again to check the effect of the changes and if you require changing that code too, you will have to go through those steps again and again.

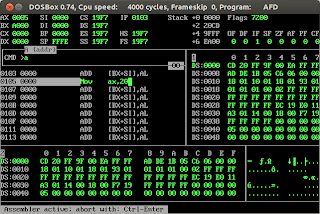
Here comes another feature of AFD that let you assemble/change the code on the fly.

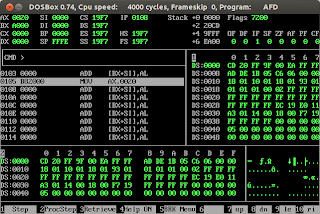
Remember, by using four navigation keys ( discussed above) we could change content of any region except the disassembly region. So for changing code we will have to type the follwing command in cmd.

CMD> a F123

here "**F123**" is address of the line where you want to start changing code. You can then make multiple changes ( pressing ENTER with each change for it to take effect ) and press CTRL + ENTER when complete.

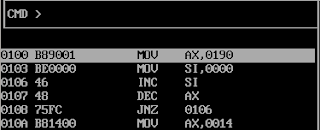
Note that the address part is optional. You can simply type "a" and it will move cursor to the line that was highlighted . This has been shown in the images below

[](http://3.bp.blogspot.com/-zbmuCfWftT0/UmGxlCHIpmI/AAAAAAAAAKc/bO2lXnrVW6E/s1600/mov+ax20.png)

[](http://1.bp.blogspot.com/-pW77nugiDcU/UmGxlGOFLEI/AAAAAAAAAKY/hmCdgJpgrH8/s1600/mov+ax20+effect.png)

**Continue Execution**

Uptil Now we have learned only two keys F1 and F2 for step by step execution of the the instructions in the debugger. Now, Look at the the code in the image below

[](http://1.bp.blogspot.com/-JoEwJ5lbhIQ/UmG5dAIsCxI/AAAAAAAAAK4/0ygIG5Y_Eic/s1600/loop.png)

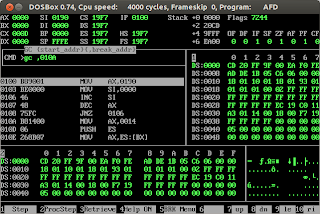
This codes sets **AX to 400 ( Hex: 0190 )** and **SI to 0** . Then it runs a loop ( 0106 - 0108 ) where it **decrements AX** and check its value to decide if loop continues or not. So this loop runs exactly 400 times meaning that we will have to press **F1/ F2 key 400 times** because of that loop.  
  
To save you from this, AFD has another command that executes instructions until it reaches a specific address. This command is called "go continue" command and it is one of the most frequently used commands in AFD.  
  
For the scenario we discussed above, we will use it like this

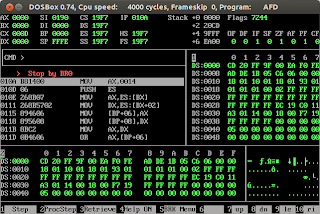
CMD> gc 0100,010A

here "0100" is starting address ( from where you want to start the execution ) and 010A is where the **break will occur**.   
In our scenario we can also write it as

CMD> gc ,010A

It will start executing from current instruction ( highlighted line ) and break at 010A .   
**Note** the comma before the address 010A that tells that it is ending\_address.    
  
The images below show the usage of gc.

[](http://3.bp.blogspot.com/-At48AwjUbeo/UmHAUpCwh6I/AAAAAAAAALM/nnvcEMKy8hs/s1600/gc.png)

[](http://1.bp.blogspot.com/-1-JtGJKgJ84/UmHAURLYKRI/AAAAAAAAALI/mzbjbCX0qMY/s1600/after+gc.png)

You can also enter only "gc". It will make program run until it terminates or hits a conditional break point ( we haven't covered breakpoints yet - We can ignore it for now ).

**Using "gc" without specifying addresses is very useful when detecting INFINITE LOOPS in the program. You can enter this command and when the program keeps on running ( i.e. it has hit an infinite loop ) press CTRL + ESC to stop the execution. You can then diagnose the cause of that endless loop by analysing the values of registers, memory content and further debugging the program .**

**Memory Operations:**

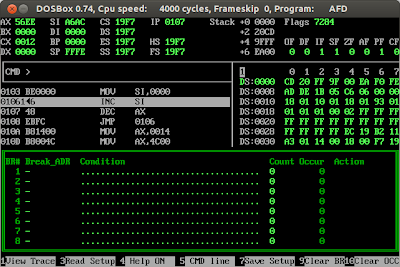
Some of the very useful commands that we can use to manipulate the strings or other content in the memory, while debugging, are as follows.

1. **c address1, address2 , len**
   * compares content starting from address1 with the content starting from address 2  
     having length equal to third parameter 'len'
2. **co source\_address, destination\_address , number\_of\_bytes**
   * copies content starting from source address to memory area starting from destination   
     address. The third parameter is total number of bytes transfered
3. **p address,string**
   * copies a string to the memory area starting from "address"
4. **f address, repeat\_count, string**
   * Copies/Fill a string ( third parameter ) to the memory area starting from address ( first parameter ). Second parameter describe how many copies of the strings will be copied  
     to the address ( it should be at least 1 to copy the string ).  
     example:  f  0ff4, 1 , "ABCDabcd123"
5. **s <address><,string>**
   * Searches a string in the memory starting from specified address but the address parameter is optional. We will have to give  at least  string once. After that we can only enter "s" to search that string repeatedly.  
     example:
     + s 0100,"ABCABC"
     + s  ,"123abc"  
       note the comma when address field is empty

**BreakPoints:**

The last, but not least, feature that I am going to cover is to set breakpoints in the code. Those break points may or may not be conditional ( i.e. to break execution when a specific condition is met ).

when you press "F5" a new menu pops up. This has been shown in the image below

[](http://1.bp.blogspot.com/-fd1v2JwTRXQ/UmHcuvTqdqI/AAAAAAAAALg/uQzT7v_8gmo/s1600/brWindow.png)

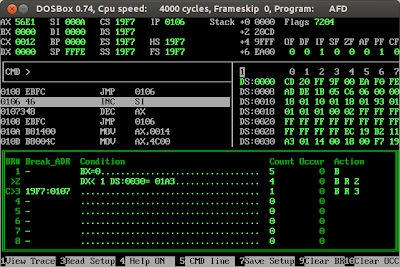
This menu contains 4 configurable options

1. **Break Address** :  can be set to "-", <any address> or empty.
2. **Condition** : 8 possible conditions can be given here each separated by spaces. Examples of conditions are as follows
   * AX>=5 BX<4
   * CF=1
   * DS:[SI]= /W ES:FF04  
     Occur field will increase by 1 , Whenever all the conditions of a break point are true.
3. **Count**: It specifies the number of times the condition  must be true before performing the specified action
4. **Action** : Can be a combination of the following
   * Break ( written as B ) - breaks the execution
   * Restart ( written as R <number>) - reset the Occur field of the break point specified by the *number.*
   * Jmp ( written as J address ) - take a jump to specified *address*  
     There are other actions also but they are not commonly used so I am skipping them.

The break point will only be activated if its *Break\_Addr*field does not have "-" as a value.  
If this field is empty the *condition* will be checked after each step.  
Whereas if any address is specified the condition will only be evaluated when we hit that address ( i.e. current instruction is on that address ).  
  
The *Occur* fieldis very important. Whenever the condition field is evaluated as true it increases *Occur* count by 1*.* When *Occur* becomes equal to *Count*, the *Action* is performed.   
After the action is performed we need to clear the Occur field ( that reactivates the breakpoint).We can do that by pressing "F10" while we are in BreakPoints' Menu. ( To return to instruction execution we have to press F5 again ).

Note: Another reason why we use *Restart Action* is that it automatically re-activates the breakpoint by reseting its occurance count.

The three breakpoints shown in the image below will help you understand the usage.

[](http://2.bp.blogspot.com/-sH-6q0TpGcE/UmHm6F9ObVI/AAAAAAAAALw/yKE2ifjYcaQ/s1600/brssss.png)

1. **First break point** is not activated as Break\_ADR field contains "-".
2. **Second break point** has Break\_ADR field empty it means the condition in this breakpoint will be checked after each step ( execution of instruction ). **The condition is DX< 1 and DS:0030= 01A3, note that 01A3 is in little endian format** . So on each instruction this condition will be checked and if it is true the *Occur count* will increase and when *occur* reaches 4 ( i.e. equals to *Count field* ) the *Action* will be performed.  
   If we look at the action it is the combination of  Break and Restart. So Action will be performed as follows
   1. Break the execution ( displays message : Stop by br2 and stop execution )
   2. Reset the Count of Break # 2 ( i.e. reset its own *count* ). So this breakpoint is set again. If "R 2" was not specified then this break point would have never been able to stop execution.
3. **Third break point** is very useful. It has no condition i.e. it will always be true. But it has an address meaning that whenever we reach the instruction on address "0107" it will evaluate the condition. As condition is always true it will increase the occurrence with 1 and as the *Count* is equal to 1 the action will be taken by AFD  i.e. **Breaking the execution and reseting breakpoint again.**  
   **In a nutshell**, whenever we reach the specified address the execution will break. It is just like setting a breakpoint on a line number.