Using Turbo Debugger 2.0

Version 1.0

Prepared by Pan Yan (panyan@nus.edu.sg)

1 What we can do with Turbo Debugger

Turbo Debugger (TD) is a powerful utility for debugging object codes generated by Borland's line of assemblers and compilers. With TD, we can easily carry out the following things on any object code:

1) Executing instructions step by step on source code level, tracing into sub-routine calls so as to see exactly the result of each single instruction.
2) Reading the content of any register or memory location on-the-fly in HEX or DEC format.
3) Setting break-points in the source code to stop the execution of the program when it reaches a certain point you set in the program, so that you can skip un-related instructions and dive directly into the problematic portion of your program.
4) Modify the content of registers (including flags) and memory. Thus you can continue debugging even though some values were already wrong.
5) Breaking the execution of the program at any time on-the-fly, so that you can easily find out where your program is hanging in. What is more, you will not have to re-boot in infinite loop situations, saving you a lot of time.

2 What we can NOT do with Turbo Debugger

We can not edit our source code in Turbo Debugger. Even though we can see the source code, we can not directly modify it inside TD. We have to prepare our source code (.asm file) in some editing software like windows notepad or MS Edit in DOS. Any plain text editor can be used to edit your source code but one with line-number display will be better because any assembling and linking error message will be referred to by line numbers.

Syntax errors can not be debugged with TD. Turbo Debugger works with linked .exe files. If you can not assemble or link your program, TD can not help you. You have to revise your code on your own to make it at least pass the assembling and linking steps.

3 How to use Turbo Debugger

To use turbo debugger, we have to first prepare the object code so that all the source information are included in the final executable. With a well prepared .exe file, we can easily debug it in the window-based integrated interface of TD.

3.1 Preparing the .exe file and starting TD

After modifying the assembly source code, save it to some .asm file. Then, use the following line to assemble it and get the corresponding .obj file.

```
tasm /zi filename.asm
```

The option /zi is to tell the assembler to save source code information in the .obj file. Then use the following line to link it into a .exe file.

```
tlink /v filename.obj
```

Here the /v option tells the linker to pass the source-code related info (symbol table) into the exe file. Now you will get a .exe file, which is ready to be used with TD.

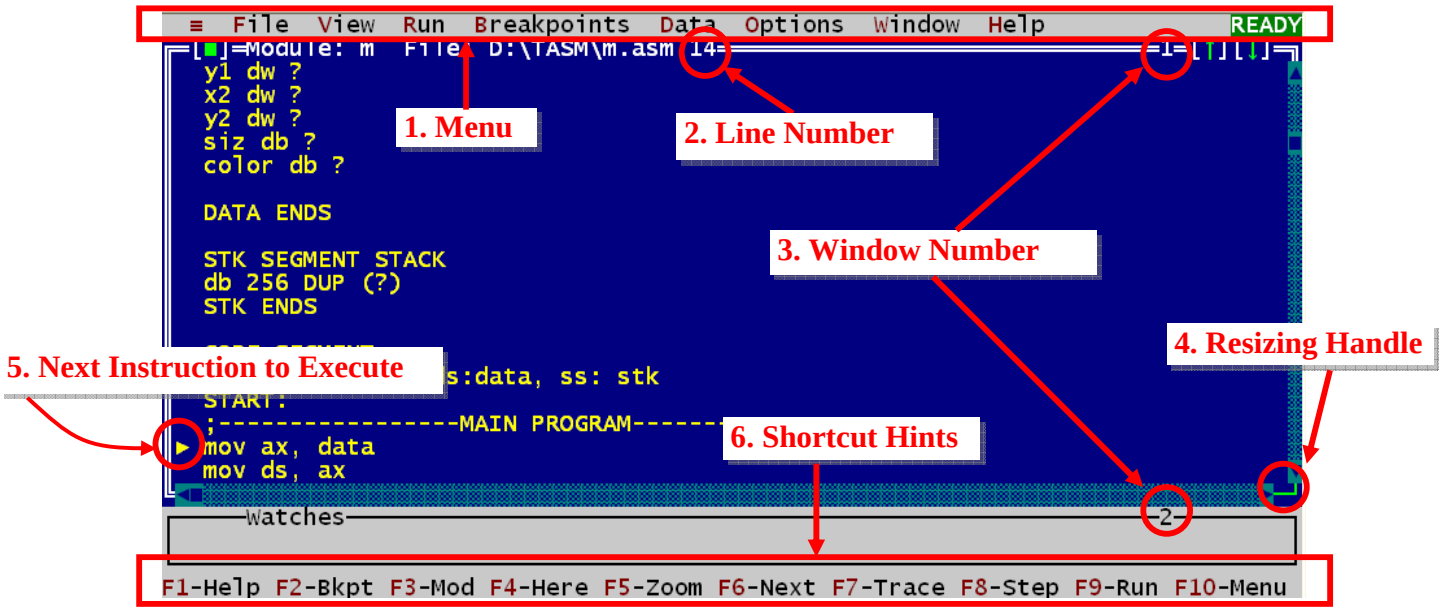
If you are interested, you can compare the .exe file with on that is assembled and linked without the /zi and /v options respectively. You will find the one got with the options is much larger in size. The difference is due to the additional debug info saved in the .exe file.

Now you can start TD with the following line:

```
td filename.exe
```

3.2 Getting started with the windows in TD

The Turbo Debugger is a window-based application. By default you will have a big blue window containing your source code and a small grey window at the bottom named “watches”, as shown below. If you see an error message saying “Program has no symbol table” and can’t find the source code window on the screen, make sure you have used the proper options when assembling and linking, as described in the previous sub-section.



1) Menu: At the top of the screen is the main menu. It’s quite similar to MS Windows menus in that you can use your mouse to click on them and active a

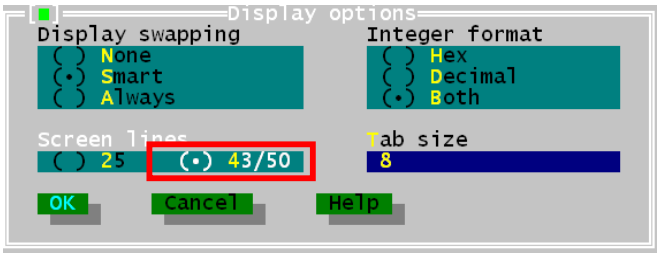
drop menu. Note there is one red letter in each menu name (F in File and B in Breakpoints). You can use short cut key **ALT+(red letter)** to activate each menu.

2) Line Number: There is a number behind the filename at the top of the source code window. This represents the position of your "cursor" (NOT the next instruction to execute). Thus you can browse through your source knowing exactly where you are. This is helpful especially when you go back to your editor to modify your code.
3) Window Number: As you might have noticed, TD has a multi-window interface. Each window has got a unique number. By using **ALT+ (number)**, you can active any window at any time. Activated window has a double-line border while inactivate window has a single-line border.
4) Resizing Handle: By clicking on this handle using your mouse, you can freely resize any active window. Alternatively, there are shortcuts: **Ctrl+F5** to enter "resize mode", when you will see the border of the current window turns green. Then use **Arrow Keys** to move the window and **Shift + Arrow Keys** to resize it. Finally, press **Enter** to exit the "resize mode".
5) Next Instruction to Execute: This triangle points to the next instruction to execute.
6) Shortcut Hints: This bottom line on the screen gives hints of currently available keyboard short cuts. Try pressing down **ALT** key and see what happens to this line.

3.3 Organizing the interface

It is important to re-organize the interface so as to make it most convenient for your debugging. The following steps are recommended.

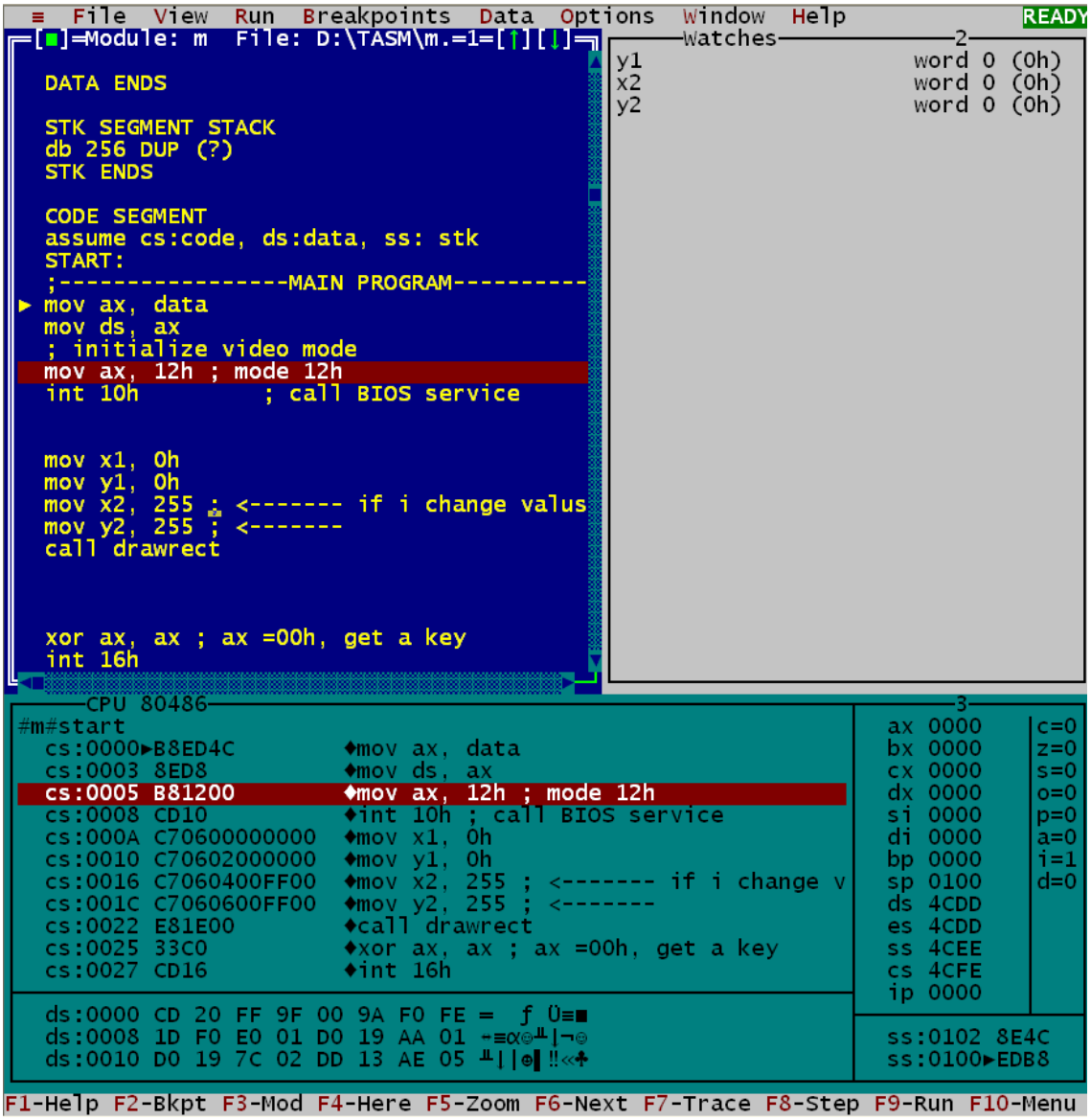
1) Change the screen size. It's not about buying a bigger monitor. You can choose to use smaller characters so that the screen can hold more lines. Press **Alt+O** to activate Options menu, then press **D** to open the Display Options dialog box. Now press **Alt+4** to select 43/50 screen lines. Press **Alt+K** to close this dialog box. You will see each character is relatively half the size and you can see more lines on the screen.



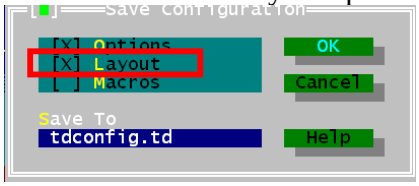
Menu Path: Options => Display Options => Screen Lines

2) Get the necessary windows. Some windows are important for debugging. The recommended windows are: CPU, Module (source code) and Watches. If any of them are not visible, you can get it from View menu. Then you can resize them to make them look good. The CPU window is very comprehensive, presenting the machine code (Code Segment), the memory (Data Segment), the stack (Stack segment), all the register values and the flags. You might like a screen like below:

Menu Path: View => CPU ; View => Module ; View => Watches



3) Saving the layout. It takes some effort to organize the interface. So it is reasonable that we want to save it so that every time TD will start like this. This can be done by pressing **Alt+O** to activate the Options menu, then press **S** to trigger the Save Options Dialog box. Now press **Alt+L** to select Layout option. Save the configuration in the tdconfig.td file in the directory where you execute TD from. Keep it on your floppy disk in case it is deleted when re-booting. When TD starts, it will automatically load tdconfig.td from the current directory. Otherwise, you can go to Options menu to "restore" from your specified configuration file.

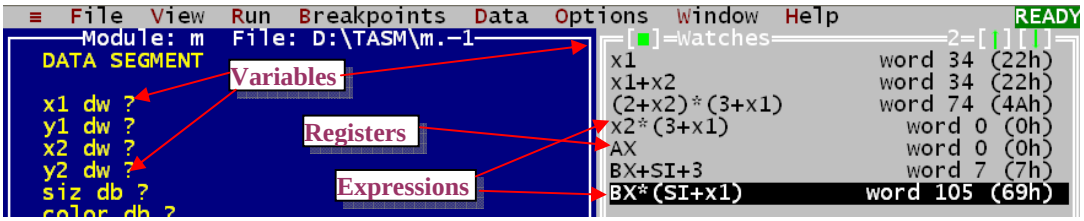


Menu Path: Options => Save Options => Layout

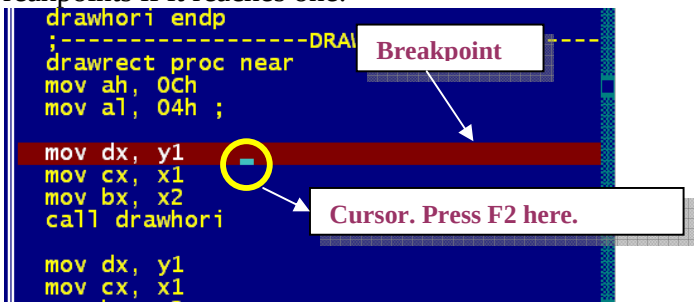
3.4 Debugging in TD

Now we are ready to start debugging.

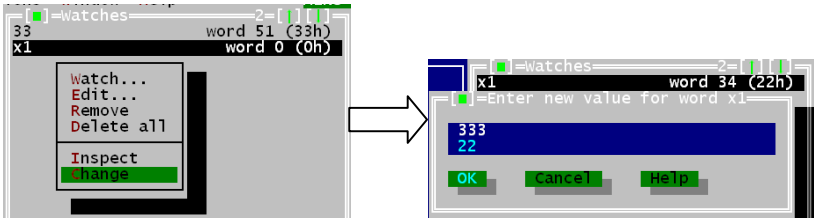
1) Monitoring values. With the CPU window, all the registers are already listed. What is more, the value of variables in the data segment can also be easily monitored. Switch to the watch window by pressing **ALT + 2** (or the window number of Watches window). Then, directly type the name of the variable you want to monitor. It will be listed with its content displayed in HEX or DEC. Note that you can also monitor Register values here. You can even enter an expression here to monitor.



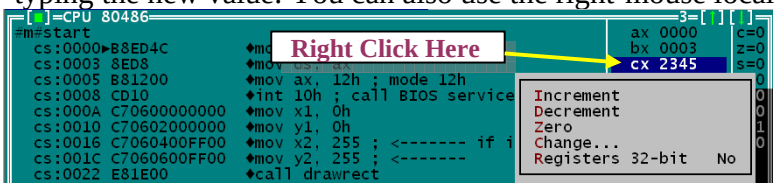
2) Single Stepping. Sometimes you might like to trace the execution of the program step by step. TD makes this as simple as a key-press. Thus you can see how each single instruction modifies the value of registers and variables in the Watches window.
   a. **F7**: Trace into. Pressing F7 will execute one more instruction pointed to by the little triangle we mentioned in section 3.2.
   b. **F8**: Step Over. For most instructions, F8 is the same as F7. The difference is for INT and CALL instructions. F8 will take such function calls as a single instruction and finish it in one step; while F7 will let you go into the body of the sub-routine and execute the instructions in step one by one.
   c. **Important**: On the source-code level, you can NOT trace into the body of any MACRO. Both **F7** and **F8** will take MACROs as one instruction. In this sense, you had better use subroutines over MACROs.
3) Breakpoints. Single stepping might not be sufficient in cases when you want to skip a large block of codes. Breakpoints fit in here well. First move the cursor by the Arrow Keys to the line of code where you want the program to stop, then press **F2**. You will see a red line highlighted. This red line indicates a breakpoint is set. Now you can run through the program by pressing **F9**. The program will stop by the breakpoints if it reaches one.



4) Modifying values. Another exciting feature of TD is that you can modify the value of registers and variables while the program is running.
   a. In the watches window, right click on the variable you want to modify. In the local menu, choose “Change”. Then you can type in the new value.



b. In the CPU window, you can easily modify the value of registers and flags by first selecting the register with a left mouse click and then directly typing the new value. You can also use the right-mouse local menu.



5) Screen Output. The integrated interface gives us comprehensive information about the execution of our program, but it also hides the normal output of our program. To see the screen output of our program, we can press **Alt+F5**. To come back to the integrated interface, simply press any key.

4 Exercises

With the above information, we can now set out to try some real debugging in the TD environment. Try debugging the sample.asm program. This is a program that counts the number of a certain letter in a long string. It also prints the final count on the screen after the counting. However, it is not working as given. Try the following things.

1) Assemble and link the given program sample.asm. Remember to use the proper options to make it friendly to TD.
2) Add watches to monitor variables and registers that interests you.
3) Follow the counting algorithm by single stepping, and try to find out the errors.
4) Try to look at the output screen when tracing the program.
5) Try to manually modify register values that is set wrongly by instructions in the program and continue debugging without modifying the code. (Optional)
6) After finishing debugging the counting algorithm, use a breakpoint to make the program stop at the label “FINISHED”. Then directly run to this point to continue debugging the result printing part.

5 Appendix – Short-cut Key List

Key	Function	Key	Function
Alt+NUM	Switch to a certain window	F7	Trace into
Alt+Letter	Activate a menu.	F8	Step Over
CTRL+F5	Enter resizing-mode	F9	Run through
ALT+F5	View the Output Screen	F4	Run to Cursor
F5	Maximize a window	F2	Set Break Point at Cursor

Sample.asm

```
; This program counts the number of of a specific letter in a string.
; There are logic errors. Try using TD to debug it.
; Copyright by Pan Yan, ECE Dept, NUS, 2005.

DATA SEGMENT
; Data Segment --- Variables
stri    DB 'I know not with what weapons World War III will be fought,'
        DB 'but World War IV will be fought with sticks and stones.$'
letter  DB 'w'
DATA ENDS

STK SEGMENT STACK
DB  1024 DUP (?)
TOS   LABEL   WORD
STK ENDS

CODE SEGMENT
ASSUME  CS:CODE, DS:DATA, SS:STK
start:  MOV AX, DATA
  MOV DS, AX
  MOV AX, STK
  MOV SS, AX
  MOV SP, OFFSET TOS

  XOR BX, BX
  XOR CX, CX
  MOV DL, letter

REP_1:  MOV AL, stri[BX]
  INC BX
  CMP AL, '$'       ; Check if the current letter is end-of-string sign.
  JE  FINISHED
  CALL  Print_Char  ; Print the character on the screen.
  MOV AL, stri[BX]
  CMP AL, DL        ; Check if the letter is the one we are to count.
  JNZ REP_1         ; If not, go on to the next letter.
  INC CX
  JMP REP_1

FINISHED:
  MOV AX,CX         ; Now the count is in CX. We are to print it on the screen.
  MOV BL, 10        ;We suppose the number has at most 2 digits.
  DIV BL
  ; Now AH keeps the remainder and AL keeps the quotient
  ADD AL, '0'       ; Convert the tens digit to a character
  CALL  Print_Char  ; Print the Tens Digit
  ADD AH, '0'       ; Convert the unit digit
  MOV AL, AH
  CALL  Print_Char  ; Print the unit digit

  MOV AH, 4ch     ; EXIT
  INT 21h

Print_Char PROC   ; This procedure prints the character in AL to the screen.
  MOV AH, 02h     ; Use INT 21h, 2
  MOV DL, AL      ; Print the character in AL.
  INT 21h
  RET
Print_Char ENDP

CODE ENDS
END start
```