LAB WORK NO. 3
TURBO DEBUGGER ENVIRONMENT

1. Objective of the lab work

The purpose of this lab is to be able to debug programs written in assembly language and general executables, using a debugging tool.

2. Theoretical considerations

2.1 Turbo Debugger Environment

Turbo Debugger environment allows the testing and tracing of any executable program (.exe or .com) and allows:

- displaying memory and register content;
- their modification;
- step by step program execution;
- program execution till encountering a breakpoint;
- instruction insert in assembly language;
- memory area dump and disassembly.

To launch the Debugger type:

td [options] [program_name [arguments]]

Parameters between square brackets are optional. “Program_name” parameter represents the program to debug. If there is no extension we suppose it is .exe. “Arguments” parameter represents the arguments (input parameters) of the program to debug. Turbo Debugger options must be placed in front of the name of the program to debug.

If no option is given, program name or argument, Turbo Debugger will load without any program and with default options.

Examples:

td –r prog1 a

will run the Debugger with –r option (remote debugging), will load “prog1” program with parameter “a”.

td prog2 –x

will start Turbo Debugger and will open “prog2” program with parameter “-x”.

Some of the most important options are:

- the possibility of launching the environment with a configuration file;
- different ways of display refresh;
- the possibility of process switching depending on “id”;
- recording the keys pressed;
- remote debugging;
- using the mouse;
- program debugging in Windows.

To find more detailed information about the option for launching the Turbo Debugger, use the “td /?” command, or consult the Help page from Turbo Debugger which refers to the command line options.

2.2. Turbo Debugger windows

2.2.1 Code window

In the code window you can see the disassembled instructions of the program. The title shows the processor on which the program runs. Line source numbers and the labels appear in front of the lines on which they will be used.

On the left side we can see the memory address of the instruction (ex. CS:0100, meaning at the address resulted from the Code Segment (CS) and the offset 0100h). It follows a hexadecimal code of variable length (representing the machine code in hexadecimal) of the instruction of which the mnemonic is on the next column.

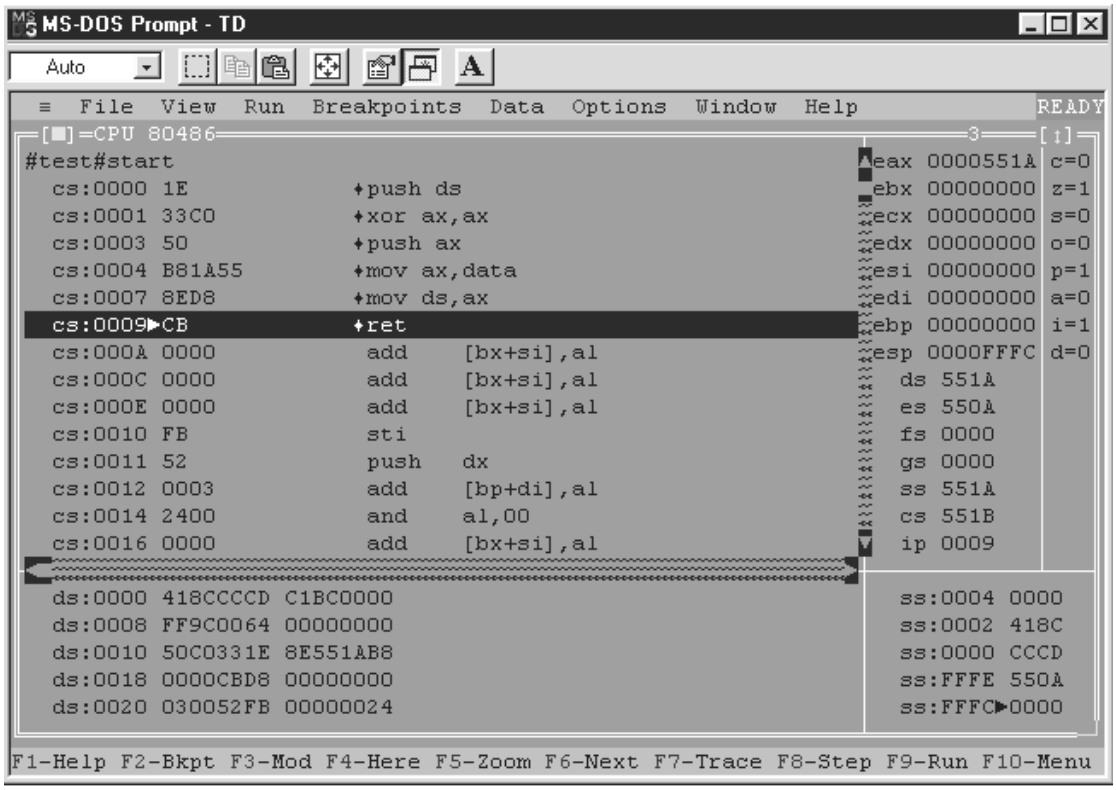
A distinguished sigh (an arrow), placed between the instruction address and its code (in case of an active CPU window, that line is coloured), and symbolizes the current instruction, the next instruction to be executed.

2.2.2. Register window

This window shows the processor registers. Their content is displayed in hexadecimal, in word size (2 bytes). The window local menu contains the following commands:

- Increment – allows adding the value 1 to the marked register content.

- Decrement – allows subtracting the value 1 from the marked register content.
- Zero – to set the register value on 0.
- Change – modifying the value of selected register.
- Registers 32-bit – allows changing the display mode of registers in 32 bits format (extended registers EAX, BAX, including segment registers FS and GS, etc.).



2.2.3. Flag window

This window shows the flag status (0 or 1). Every flag is indicated by a significant letter (C – Carry, Z – Zero, S – Sign, O – Overflow, P-Parity, A – BCD Carry (auxiliary carry), I – Interrupt, D - Direction).

The local menu of this window has only one command: Toggle – it switches the flags values 0 or 1 (to activate the command press “space” or “enter”).

2.2.4. Data window

This window shows, in hexadecimal, a part of the memory area from a segment. Window content displays the address (as "segment:offset") and the effective memory content.

Local menu of this window contains the following commands:

- GoTo – it sets the address on which the data display will begin in the data window. This address may be given in different ways:
  - offset (ex. 0100h), in this case current segment is taken by default;
  - segment:offset (ex. DS:0000h, or DS:0100h), and the positioning is effected to the absolute address given by segment and offset;
  - segment:offset with explicit values (ex. 54F7:0008h), the positioning will be effected to 54F7 segment and 8h offset;
  - variable_name (ex. No1), in case the assembly was done with debug information included.

- Search – searches a byte sequence in the memory.
- Next – searches the next appearance of the byte sequence given in Search command.
- Change – allows memory content modification from the current location by introducing a sequence of bytes.
- Follow – allows the positioning in the data window to a new address based on the number of bytes from the address specified by the current position (ex. code Near, Far, data offset, segment:offset).
- Previous – positions the window to the initial address, before modifying with GoTo or Follow.
- Display As – permits the choosing of the displaying mode of data in data window and has the options: Byte (1 byte), Word (2 bytes), Long (4 bytes), Comp (8 bytes), Float (real number, 4 bytes), Real (real number, 6 bytes), Double (real number, 8 bytes), Extended (10 bytes).
- Block – allows operating with memory blocks.

2.2.5. Stack window

Stack window shows the current content of the top of stack (last few elements) indicated by registers SS:SP.

Local menu of this window has the following commands:

- GoTo – allows the modification of the address from which the display is made in the window.
- Origin – it effects the return to the base address (SS:SP).
- Follow – allows setting the address from which the display is done as the value of the word selected from the stack. The command is useful when it is needed the following of the stack pointer content.
- Previous – positions the stack window to the initial address, before the modification with GoTo or Follow.
- Change – allows the modification of the stack content from the current location by inserting a new value (word).

2.3 Turbo Debugger Menus

In this part we will describe the most common part of Turbo Debugger menus.

- "File" menu has the following submenus:
  - "Open" – opens a dialog box for loading an executable file for debugging.
  - "Change dir" – allows the changing of current working directory (from where the file loading is made).
  - "Get info" – offers information about the loaded program and conventional memory status and EMS.
  - "DOS shell" – restores the control to the operating system, without closing Turbo Debugger (can go back using the command "exit").
  - "Resident" – allows quitting Turbo Debugger, with the possibility to have it resident in the memory (to activate it, use the command Ctrl-Break).
  - "Symbol load" – loads a symbol table specified by the user.
  - "Table relocate"- allows the relocation of the symbol table.
  - "Quit" – exits Turbo Debugger environment.

- "View" menu has the following submenu:
  - "Breakpoints" – allows the viewing of the breakpoints and their characteristics from the active program being debugged.
  - "Stack" – allows the viewing of stack window.
  - "Log" – opens log window (which has also a submenu).
  - "Watches" – opens a window for variable watch – the value of the variables you want to follow. With the local submenu, variables can be added, edited, deleted, as well as their content.
  - "Variables" – opens the visualization window of the variables defined in the program. The window contains a submenu to offer more operations on the variables.
  - "Module" – allows the selection for visualization of one of the program modules loaded from a list.
  - "File" – loads a file for visualization (the window can be used for viewing the source file, being the initial file of the executable file to debug).
  - "CPU" – opens the CPU window, referred in "Turbo Debugger Windows".
  - "Dump" – opens data window for displaying memory content (see "Turbo Debugger Windows" chapter). The window contains many commands in the local submenu.
  - "Registers" – opens the window for displaying and modifying register content (see "Turbo Debugger Windows" chapter). The window contains many commands in the local submenu.
  - "Numeric processor" – opens the math co-processor window, displaying the internal stack, as well as the flags. This window is used for debugging the programs that are using co-processor dedicated instructions.
  - "Execution history window" – opens the window that shows the last instructions executed by the central unit. Previous executed instructions are stored only when running with Trace command (from Run menu). If the tracing is made in a module visualization window, option "Full history" must be set on "yes". Local submenu has the following commands:
    - "Inspect" – opens a module visualization window in which it can be seen the source code of the instruction selected from those being executed.
    - "Reverse executed" – executes backwards the instructions from the current one to the selected one. Excepting the I/O

instructions, the status is the same as the one before the execution of the instruction up to which the return is made.
    - "Full history" – it is a switch that allows a slower and more complete way, or faster and less complete way of reverse execution.
  - "Hierarchy" – opens a window useful for debugging C++ or Pascal program that contain objects.
  - "Windows messages" – opens a message window when debugging programs under Windows.
  - "Another" – allows opening of another module, memory or file window (as described above).

- "Run" menu has the following submenu:
  - "Run" (F9) – runs the program till meeting a breakpoint, till breaking the program by the user with break keys, or till the end of the program. If the program is stopped with break keys (usually Ctrl-Break) there can be examined registers and program status.
  - "Goto cursor" (F4) – runs the program till reaching the selected line from the source code (CPU window or module visualization window).
  - "Trace into" (F7) – executes one instruction or code line (it is used the most in program debugging).
  - "Step over" (F8) – executes one instruction or code line as "Trace into" command, with the specification that procedure calls are being executed in one step, so it is not entering in procedures with the debugging.
  - "Execute to" (Alt-F9) – runs the program and is stopping to a specified location within the program. The user will be asked to insert an address to which the program to stop.
  - "Until Return" (Alt-F8) – runs the program being debugged till the current procedure or function is finished (to the first "return"). The command is used when accidentally "Trace into" is used instead of "Step over" and it is entered by mistake into a procedure, or in case of a procedure debugging and it is wanted the execution of the rest of it without stopping.
  - "Animate" – similar to "Trace into", being repeated. Instructions are executed continuous till key pressing. The debugger changes his status to notice the execution changes. The user is being asked for the instruction execution rate.

- "Back trace" – remakes the status by backwards execution of the last executed instruction (undo).
- "Instruction trace" – executes one machine instruction. The command is used to trace a break call in CPU window, for tracing a function into a module, which does not contain debugging information.
- "Run Arguments" – allows arguments changing from the command line of the program being debugged. The command is used when a program is debugged and it needs one or more input parameters, not given (or erroneously given).
- "Program reset" (Ctrl-F2) – reloads the current program. The command is used when re-running of a program is wanted.

- "Breakpoints" menu has the following submenu:
  - "Toggle" (F2) – marks (on/off) a breakpoint on current instruction; in this point the program will stop at every run.
  - "At…" (Alt-F2) – marks a breakpoint to a specified address.
  - "Change Memory Global" – sets a breakpoint, which will change the value of a memory area.
  - "Expression True Global" – sets a breakpoint that will take action when an inserted expression becomes true.
  - "Hardware breakpoint" – sets a hardware breakpoint by his detailed specification in the afferent dialog box.
  - "Delete all" – deletes all declared breakpoints.

- "Data" menu has the following submenu:
  - "Inspect" – allows the inspection of some variable or references inserted in memory at request in the dialog box.
  - "Evaluate/Modify" – evaluates an arbitrary expression, allows variable names as well as formulas, and displays the result in decimal and hexadecimal.
  - "Add Watch" – adds an expression or a variable in the variable watch window.
  - "Function return" – allows the inspection of the value that will be returned by the current function.

- "Options" menu has the following submenu:
  - "Language" – allows the specification of the way Turbo Debugger interprets the expressions user inserted.
  - "Macros" – creates, modifies and deletes macros assigned to certain keys (ex. command sequences mostly used).
  - "Display Options" – opens a dialog box for setting the display mode on the display, how to display numbers, as well as the way in which the display refresh is made.
  - "Path for Source" – allows setting of the way where Turbo Debugger searches the source files that compose the program.
  - "Save Options" – opens a dialog box for selecting the configuration part to be saved, as well as the configuration file.
  - "Restore Options" – allows configuration loading from a configuration file previously saved with "Save Options" command.

- "Window" and "Help" menus are similar to any other application, so they won't be described in this laboratory.

3. Lab tasks

A short program will be edited, assembled, link-edited and debugged, using the most used commands and functions from Turbo Debugger menu.

For the beginning, the following program will be edited, using an editor like Notepad. The program only declares some variables in the data segment. The code segment contains only the instructions needed to activate the data segment. The program will be used especially for exemplifying the visualization mode of data in memory. The name of the program will be "test.asm".

```
;------------------------------------------------------------ COPY FROM HERE
DATA SEGMENT PARA PUBLIC 'DATA'
NO1  DD 17.6
NO2  DD -23.5
NO3  DW 100
NO4  DW -100
DATA ENDS

CODE SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CODE, DS:DATA
START     PROC FAR
          PUSH DS
          XOR  AX,AX
          PUSH AX
          MOV  AX,DATA
          MOV  DS,AX
;OTHER PROGRAM INSTRUCTIONS
          RET
START     ENDP
CODE ENDS
END  START
;------------------------------------------------------------COPY UNTILL HERE
```

The program will be assembled with the command:

Tasm /zi test.asm

This command will generate the object module test.obj (in case of successful assembling). If /la option (expanded listing) is used, an .lst file will be obtained. This text file will contain information about line numbers, relative address on which the instructions are assembled, machine code

resulted after the assembly, as well as the initial form of the instructions. At the end the symbol table will be listed in detail.

For additional details referring Turbo Assembler parameters run “tasm /?” command, or simpler, without parameters, only “tasm”.

Next is step is to link-edit the .obj module, only one for this simple example, but there could be several modules. To obtain the executable file, write the command:

Tlink /v test.obj

As a result of this command, if no errors occur, the file test.exe will be generated. Option /v is used for easier debugging of the program and signifies the inclusion of the information from the symbol table so that Turbo Debugger will list and use variables and labels instead of memory addresses. This option cannot be used in a .com program. To obtain a .com program, the option /t must be used (and the code must have been written to meet .com program conditions, e.g. to have only one segment, that starts at address 100h, etc.).

For additional details referring Turbo Link parameters, use “tlink /?” or “tlink” command.

For testing and debugging the program the command line is:

Td test.exe

This will launch the Turbo Debugger and the program test.exe will be automatically loaded. Without a parameter you could load your program from the file menu.

If the program has been assembled and link-edited with the debug options, the initial source code will show up. This can be debugged and ran by Turbo Debugger commands. For a better understanding of what is happening in the computers memory it is recommended to open the CPU window (View - CPU); it contains CPU windows, the registers, the flags, the stack and the data segment. You may modify ore insert new instructions, but it is not recommended. You can not save your work in an .asm file.

Now, tracing the program may commence. The most used commands will be: step by step run: Run – Trace (F7) which executes one instruction at every step, or establishing breakpoints: Breakpoints – Toggle and then running with Run – Run, or positioning the cursor on a specified

instruction and using Run – Go to cursor. To return to the beginning of the program, use Run – Program reset.

While tracing the program, at every step the registers, flags, stack, and data area values can be inspected or even modified as needed.

For data visualization directly in data segment, switch to the data window and position to the beginning address of the data segment – Go to (DS:0000h) command, the address can be also written directly using DS value (ex. 551A:0000h). Another way to position is writing variable name (only in case of debugging information is included at assembly). The variable values can be also seen in Data menu: Data – Inspect or Data – Evaluate/Modify (function that can be used later as a simple decimal – hexadecimal conversion tool).

Another possibility is the window Data – Add watch, which monitors the specified variable values.

Usually, data window is positioned on Byte status, meaning data is displayed byte by byte, and on the right their ASCII representation is shown. For our program, the data segment starts with no1 = 17.6 value (in short IEEE format, so on 4 bytes), no2=-23.5 (same format as no1) and it is continued with no3=100 (on 2 bytes in C2) and no4=-100 (2 bytes in C2).

To view no1 and no2 values, you will set the Display as – Float, to view their representation you will use Display as – Byte, the memory content (17.6 is represented as 418CCCCD, and –23.5 is represented as C1BC0000).

To view no3 and no4 you should display memory content from address ds:0008h (so, offset 2*4 bytes as much as those 2 values in short IEEE format representation); the visualization is done with Display as Word. You may notice that 100 is represented in memory as 0064h, and –100 as FF9Ch.

The program can be traced till end. Take care, some programs may block or end unexpected, not as the programmer wanted. You may see strange instructions executed, this is the result of the earlier mentioned error.