LABORATORY WORK NO. 3
TURBO DEBUGGER ENVIRONMENT

1. Object of laboratory

The object of this lab is to get used 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 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;
- mouse using;
- 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 windows you can see the disassembly instructions of the program. The title shows the processor on which the program runs. Lines source numbers and the labels appear in front of the one 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 colored), symbolizes the current instruction.
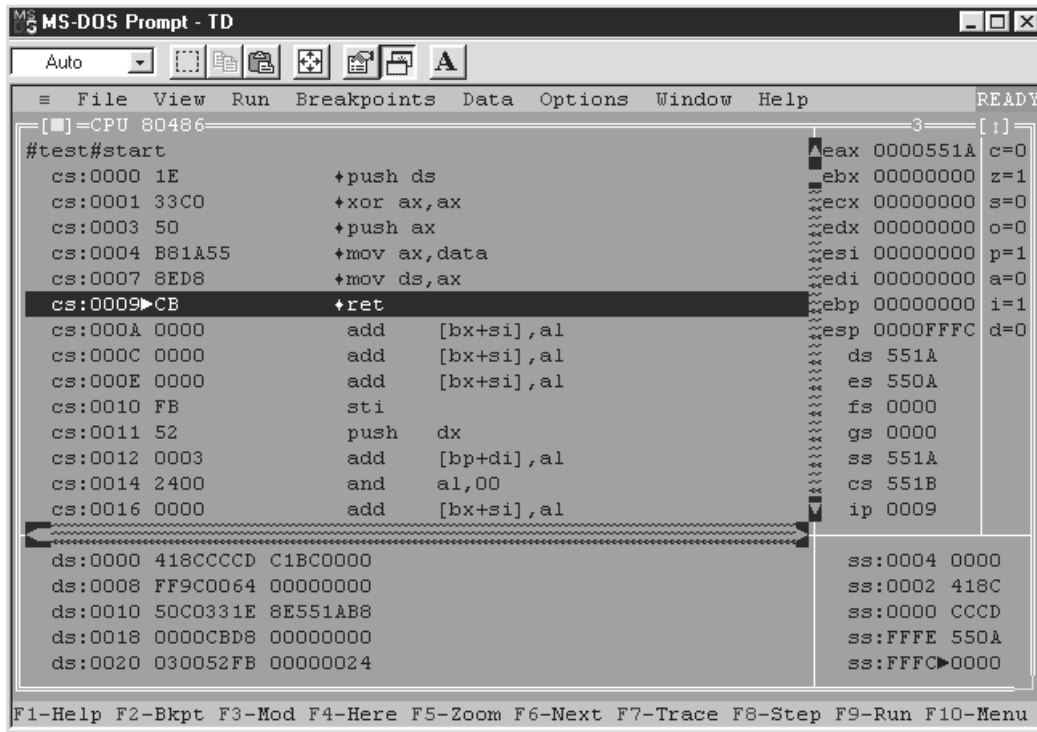
2.2.2. Register window

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 of displaying mode of registers in 32 bits format (extended registers EAX, BAX, inclusively segment registers FS and GS, etc.).

```
MS MS-DOS Prompt - TD                                          _ □ ×
 Auto        ▾   ┊┅┊ ▤▤  ⊞  ☝☝  A
  ≡   File   View   Run   Breakpoints   Data   Options   Window   Help        READY
 ┌[■]=CPU 80486════════════════════════════════════════════════3═══[↕]═
 #test#start                                        ▲eax 0000551A c=0
   cs:0000 1E              ↓push ds                  ■ebx 00000000 z=1
   cs:0001 33C0            ↓xor ax,ax                ≋ecx 00000000 s=0
   cs:0003 50              ↓push ax                  ≋edx 00000000 o=0
   cs:0004 B81A55          ↓mov ax,data              ≋esi 00000000 p=1
   cs:0007 8ED8            ↓mov ds,ax                ≋edi 00000000 a=0
   cs:0009▶CB              ↓ret                      ≋ebp 00000000 i=1
   cs:000A 0000            add    [bx+si],al         ≋esp 0000FFFC d=0
   cs:000C 0000            add    [bx+si],al         ≋    ds 551A
   cs:000E 0000            add    [bx+si],al         ≋    es 550A
   cs:0010 FB              sti                        ≋    fs 0000
   cs:0011 52              push   dx                  ≋    gs 0000
   cs:0012 0003            add    [bp+di],al          ≋    ss 551A
   cs:0014 2400            and    al,00               ≋    cs 551B
   cs:0016 0000            add    [bx+si],al          ▼    ip 0009
 ◄═══════════════════════════════════════════════════►
   ds:0000 418CCCCD C1BC0000                         ss:0004 0000
   ds:0008 FF9C0064 00000000                         ss:0002 418C
   ds:0010 50C0331E 8E551AB8                         ss:0000 CCCD
   ds:0018 0000CBD8 00000000                         ss:FFFE 550A
   ds:0020 030052FB 00000024                         ss:FFFC▶0000
 F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

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 the data 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 precise values (ex. 54F7:0008h), the positioning will be effected to 54F7 segment and 8h offset;
  - variable_name (ex. No1), in this case the compilation is done with information including for debugging.

- 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 dada 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 pair 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 it will be described the most common part of Turbo Debugger menus.

- "File" menu
  It has the following submenu:
    - "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
  It 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 indicators. 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

26

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
  It 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 wrong given).
- "Program reset" (Ctrl-F2) – reloads the current program. The command is used when re-running of a program is wanted.

- "Breakpoints" menu
  It 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
  It 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
  It 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

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

For the beginning, it will be written, using an editor, the following program, which only declares some variables in data segment, and in the code segment it contains only activation instructions of data segment. The program will be used especially for exemplifying the visualization mode of data in the memory. The name of the program will be "test.asm".

```
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
```

The program will be assembled with the command:

Tasm /zi test.asm

This command will generate object module test.obj (in case of successful compiling). If option /la (expanded listing) is being used, while assembling it will be obtained a .lst file which 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 instructions. Carrying on, the symbol table will be listed in detailed mode.

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

Next is the link-edit of the .obj module (or of many modules) for obtaining the executable file, with the instruction:

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 including of the information from the symbol table so that Turbo Debugger to list variable and label names instead of code addresses. This option cannot be used in .com program case. If obtaining a .com program would be wanted, in link-edit option /t should have been used (and the code must have been written to follow a .com program conditions, meaning to have one segment, to be loaded at address 100h, etc.).

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

For testing and debugging the program it will be used:

Td test.exe

This will launch the Turbo Debugger and the program test.exe will be automatically loaded (if the program wouldn't be given as a parameter, this could have been loaded from File - Open).

If the program has been compiled and link-edited with the options of debugging information inclusion, it would appear only the initial written text which can be debugged and ran by Turbo Debugger commands. For a better understanding of what is happening in the computer memory it is recommended the CPU window to be opened (View - CPU); it contains CPU windows, the registers, the flags, the stack and data segment. The instructions modification can also be made (or new instruction insertion), but it is not recommended, because there cannot be made a saving in .asm file, Turbo Debugger working on .exe file.

Now, tracing the program can begin. 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. For the returning to the beginning of the program, it is used Run – Program reset.

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

For data visualization directly in data segment, it is switched to data window and it positions on 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 the direct writing of the variable name (only in case in which options for debugging information have been included at compilation). 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.

Usual, data window is positioned on Byte status, meaning data is displayed byte by byte, and to the right it is given their ASCII representation. In tested program case, data segment starts with no1 = 17.6 value (in short IEEE format, so on 4 bytes), no2=-23.5 (same 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, it will be used the setting Display as – Float, and for viewing their internal representation it will be used Display as – Long, for viewing the memory content (17.6 is represented as 418CCCCD, and –23.5 is represented as C1BC0000). To view no3 and no4 it must be seen the memory content from address ds:0008h (so, offset 2*4 bytes as much as those 2 values in short IEEE format represent); the visualization is made with Display as Word and it can be noticed that 100 is represented in the memory as 0064h, and –100 as FF9Ch.

Carrying on, the program can be traced till end (here it must be taken good care, because most of the programs that "block" actually are not ending as programmer would like to, thing that can be easily seen in Turbo Debugger, meaning that instructions being ran are no longer part of the written code).