THE 8085 TRAINING KIT WITH DISASSEMBLER CAPABILITY

Salih Fadıl
Department of Electrical Engineering
Osmangazi University
Batı Meşelik, 26480, Eskişehir, TURKEY
Tel: (222) 239 1074, E-mail: sfadil@ogu.edu.tr

Cem Polat
Department of Electrical Engineering
Osmangazi University
Batı Meşelik, 26480, Eskişehir, TURKEY

ABSTRACT
The 8085 training kit and its disassemblers are explained in this paper. The disassembler programs written for the training kit have ability to disassemble a given input file or a specified memory region of the kit. Disassembler programs are used to translate machine codes to source codes so that users can understand and analyze these codes easily. Disassembly process is also important when changing the program codes in the memory. When the program codes are changed, users can observe the changes in the source codes by using the written disassembler programs. The hardware and software of the training kit with new features are also explained.

INTRODUCTION
In the classical microprocessor training kits, machine hex codes of the user program are found from the instruction set of the microprocessor at first. Then, obtained hex codes are entered to the memory manually by using a hex keyboard. However, this procedure is not a practical method for loading user programs into the memory because it takes long time. Many errors may also occur in this process. Intel's SDK-85 is one of these training kits. It is a single board microcomputer designed with 8085 microprocessor for training purposes. SDK-85 system has extra abilities like single step execution and setting breakpoints to the program. The 8085 training kit has more abilities than these classical training kits. Since machine codes of the user programs are produced by the compiler of the computer program, there is no need to assemble the machine codes manually in the 8085 training kit. The abilities of the training kit are not limited with compiling, loading and running. The training kit also allows users to follow the execution of their programs. The users can view and change any memory content, registers and flags after pausing the execution of their programs [1].

1. SYSTEM HARDWARE
The 8085 training kit is composed of several boards that communicate each other through the system bus. These are microprocessor board, memory and system communication board, ADC/DAC board, parallel communication board, serial communication and timer board. The hardware of the system is expandable so that new boards can be added to the system. The system has two 32 Kb RAMs and one 32 Kb EPROM. The system operates in two modes which are DUMP and RUN modes. These modes are selected by using a switch. There is also a reset button available that resets the microprocessor. In the DUMP mode, the user programs or the subprogram is loaded into the memory of the system by the system program. The user programs are run in the RUN mode.

In decoding of the memory chips, "memory expansion technique" is used so that all of the memory area (64 Kb) can be allocated for the user programs. In order to accomplish this job, an additional address bit is created (A16) by using a D-type flip-flop. The memory maps in DUMP and RUN modes are shown in Table 1 and Table 2 respectively

A15	A16	Address Range	Selected Unit
0	x	0000H-7FFFH	EPROM
1	0	8000H-FFFFH	RAM 1
1	1	8000H-FFFFH	RAM 2

Table 1: Memory map in DUMP mode.

A15	A16	Address Range	Selected Unit
0	x	0000H-7FFFH	RAM 1
1	0	8000H-FFFFH	EPROM
1	1	8000H-FFFFH	RAM 2

Table 2: Memory map in RUN mode.

The training kit communicates with a personal computer by using the serial communication and timer board with RS232 serial communication protocol. The parallel communication board is used for parallel communication with external peripherals like the experiment board. The experiment board is an external board containing a DC motor, displays, relays and sensors. It is designed for making experiments by using these peripherals. The ADC/DAC board allows to process analog data. It is possible to record and play sound by using ADC/DAC board with its audio amplifiers [2].

2. SYSTEM SOFTWARE
The system program stored in EPROM with the computer program in the personal computer forms the operating system of the training kit.

2.1 The System Program
The system program is stored in the EPROM. The main purpose of the system program is to load user assembly programs into the RAM memory of the kit. The system program locates any program which is transmitted from the computer in Intel-hex format, to the specified RAM memory region. The computer

and the kit communicates with each other by using many control characters in their communication protocol. They do not transmit or receive data unless the other side is ready for operation. The system program also prepares the user programs that are loaded in the RAM memory to run. All these operations are performed in the DUMP mode.

The user assembly programs can be run either step by step under the control of the computer or independently from the computer. If the user assembly programs are run under the control of the computer, the microprocessor does not perform real time operation because its operation is paused or delayed in each step. Although running the programs in steps makes the programs easy to understand for the user, the operation speed is dramatically reduced in this mode. When the user programs are run under this operating mode, the communication between the system and the computer is performed by the subprogram in the RUN mode. The subprogram is loaded to the last 4 Kb of the RAM memory so that the user programs can not be loaded to this region. The subprogram can be loaded into the RAM memory from the subprogram file in the computer in Intel-hex format or it can be loaded from EPROM. Indeed, the subprogram can be loaded more fast if it is loaded from EPROM. The ability of loading subprogram from EPROM is a new feature of the updated version of the system program. The user programs can be also run independently from the computer. This operation mode is called as "direct load". This mode is essential if the user programs require real time operation.

The system program receives data from the computer in Intel-hex format. This type of data is sent in rows. Each row includes byte count, starting address, record type and sum check information with maximum 16 bytes of data. The data received in Intel-hex format is converted to the binary data by the system program before loading it to the memory.

In the updated version of the system program, sound messages can be played after a user program or the subprogram is loaded. If an error occurs while loading these programs, a sound message is also played. These sound records were recorded by using the ADC/DAC board and they were stored in EPROM. Playing the sound records can be enabled or disabled in the options menu in the computer program.

The used memory expansion technique causes some problems in the system program. In this technique, two memory locations in RAM1 and RAM2 have the same addresses and they are selected by A16 (See Table 1 and Table 2). So, there is also another memory location with the same address of the stack pointer. One possible problem occurs if the RAMs are changed inside a subroutine. In this case, the program counter is placed with the address located at the other memory location instead of the address of the stack pointer. So, the program can not return from subroutine to the main program and it fails. For that reason, in the previous version of the system program, subroutines that use the stack pointer were not used. In the updated version, this difficulty is overcame by selecting RAM2 where the stack pointer is placed, before subroutine calls and returns. In this version, many subroutines are used without any problem in order to perform the common operations [3].

2.2 The Computer Program

The computer program that communicates with the training kit is written in Visual Basic programming language. This program allows users to edit, compile, load and run their assembly programs.

The computer program has a main window including menus, a toolbar and editors. There are four editor windows available so that users can open and edit four different files at the same time. The instruction lines of the user program written into these editors are arranged and checked for the errors.

The file menu provides commands for creating new files, opening existing files, saving files, printing files, and exiting the computer program. The recent files are also contained in this menu.

The edit menu contains common edit commands which are undo, cut, copy, paste, select all, delete, find and replace commands. When find or replace commands are clicked, find window appears. By using this window, the words entered can be searched and replaced by other words in the active editor.

In the special operations menu, compile, add breakpoint and dump commands, load subprogram submenu and direct load option are included. The compile command compiles the user program in the active editor. The add breakpoint command calls break point window. The user can put and remove break points on desired lines of the assembly program in the active editor. The dump command dumps or loads the compiled user program into the memory of the training kit. The load subprogram submenu includes load subprogram from EPROM and load subprogram from file options. The direct load option enables or disables direct load operation.

The window menu contains window management commands which are tile vertical, tile horizontal and arrange. Break point window, list window and Intel-hex window can be also reached from this menu. The list window shows the list file of the assembly program in the active editor. The Intel-hex window shows the machine codes in Intel-hex code of the assembly program in the active editor. When the active editor is changed, the content of those windows are changed automatically.

The options menu allows to change the colors, fonts and other properties of the computer program. The general use window that can be accessed from this menu allows to change the number system (decimal, binary, hex) and representation of numbers (unsigned, ones complement, twos comple-

ment) which are used in displaying the contents of the registers in the memory and register window. The feature of playing sound messages can be enabled from this window. The calculator program of windows can be also accessed from this menu.

The help menu provides access to the help files. These help files include information about 8085 Assembly language, the hardware of the training kit and the computer program.

The operations performed by the commands in these menus can be also performed by the buttons in the toolbox and short cut keys.

The user files opened or created in the editors should be compiled before loading them to the training kit. The compiler of the computer program is a special 8085 compiler that converts user program written in Assembly language to Intel-hex format. This conversion is required because Intel-hex format is used in data transmission. After compilation, the compiled user program can be loaded to the memory of the training kit. But this operation should be done in DUMP mode. Then, RUN mode can be selected and the user program is run. If the direct load option is not selected from the special operations menu, the run time window and the memory and registers window appears. The loaded program that is run can be stopped, paused or restarted by using the buttons on the run time window. The user program is also placed in the run time window. The instruction currently executed is highlighted so that the order of the execution of the instructions can be followed. The user program that is run can be paused by the pause button and it can continue to run by the continue button. The execution of the user program can be restarted by the restart button. The stop button closes the run time window and stops the execution of the user program. The user program can be run step by step manually in order to control each step of the execution of the program. The execution of the program can be also animated. In this case, the steps of the program is executed automatically so that the operation is not paused in each step. The commands in the microprocessor control menu also performs these operations. The memory and registers window shows the registers and the content of the memory at the specified memory region when the program currently run is paused. This window has the memory request, plot graph and disassemble buttons. In the memory request process, the content of the memory region of the loaded program or any memory region can be received from the training kit. If the memory region of the loaded program is requested, the memory region between the start and end addresses of the loaded program is received and replaced to the memory grid located on the registers and memory window. The users can select another memory region by manually entering the start and end addresses. The content of the memory and registers can be also changed by just clicking over them. The graph of the memory content that is received can be plotted by clicking the plot graph button. The disassembler button in this window activates the memory disassembler. The disassemblers are explained in detail in the disassemblers section [4].

3. DISASSEMBLERS

The disassemblers are the programs that convert the machine codes of the microprocessors to the source codes or mnemonics. The purpose of disassembling machine codes is to make the machine codes meaningful for the user. In fact, machine codes are hard to understand and translating them to the corresponding instructions manually is not a practical method. Machine codes can be easily analyzed by using disassemblers. Also, if the source code of a program is lost, the source code can be easily generated by using a disassembler.

The disassemblers convert machine language files that were previously converted from source code files, to the source code files. Since all labels and variables are lost during compilation, the source codes generated in the disassembly process do not contain any original labels or variables. The disassemblers prepared for the training kit can generate labels by processing the label addresses. Although the actual names of the labels are lost, symbolic names with numbers (LABXXX) are given to the labels in these programs.

The disassemblers prepared for the training set were written in C programming language. C language is preferred because it is fast and powerful. These disassemblers are included in a single DLL (Dynamic Library Link) application (Dasm85.dll). DLLs can be called by any windows application and also they can be shared among multiple applications. By this way, the computer program written in Visual Basic can call and execute the functions of the disassemblers written in C. DLLs include several exportable functions which are also called as Application Programming Interface (API) functions. The disassembler APIs are interfaced and used by the computer program after they are declared for Visual Basic.

The disassemblers of the training kit are the file disassembler, the memory disassembler and the line disassembler. The memory and file disassemblers use the same function as they have common processes. In fact, they are the different operating modes of the dasm API function. Since, the operations performed in line disassembler is different, it has its own API function that is the line_dasm function.

3.1 The File Disassembler

The file disassembler is used to disassemble the binary files (*.bin) and the Intel-hex files (*.abs) to obtain their source code files. These files can be disassembled to the source code files if they are opened by the open file command in the file menu. When a file with binary file extension (*.bin) or a

file with an Intel-hex file extension (*.abs) is attempted to open, the disassembly program is executed by calling the dasm API function with the required parameters.

If an Intel-hex file is opened, the Intel-hex file is firstly translated to a temporary file (temp.bin) which is in binary format. This process is performed by a special algorithm. In this algorithm, the origin addresses are also selected and recorded. Then, the starting address is found. In the first pass, all the bytes in the temporary binary file is read and the labels are searched. While searching the labels, it is considered that they are only used with jump and call group instructions which are three byte branch instructions. It is also considered that the same label addresses should take the same labels. In the second pass, labels and origins are placed and the actual disassembly process is performed. A label is placed with comparing the label addresses with the recent address in a loop. If a label is not defined for the recent address, the program exits from the loop without placing a label. This process should be performed before disassembling an instruction because labels are placed before instructions in the source code. After label check, the recent address is also checked if it is the last address before an origin. If so, the corresponding origin address is placed with "ORG" directive before the next address. Then the first byte of the temporary file is read. The corresponding opcode for this byte is placed to the temporary source code file from the opcode table. Its size is also found from the opcode size table. Depending on the size of the instruction, the next byte or bytes are placed as operands. If it is a one byte instruction, no operand is placed. If the instruction is a three byte branch instruction, the corresponding label found in the first pass is placed after the opcode instead of the label address. The recent address is updated after placing an instruction. This process continues in a loop with generating new command lines until the end of the temporary binary file is reached. Then the program exits from the dasm API function and returns to the computer program after closing the opened files. The computer program just places the created source code file (temp.src) to the selected editor window. The list file created is placed to the list window and the Intel-hex file processed.is placed to the Intel-hex window in the same way.

The list file (temp.lst) of the input file is created with the same procedure but the line numbers, addresses and the hexadecimal codes of the instructions are also added to the lines before the instructions.

If a binary file is opened by the open file command in the computer program, the same procedure is applied to disassemble this file with a difference. The binary file is firstly translated to a temporary Intel-hex file (temp.abs) and then the translated Intel-hex file is disassembled with the same procedure. The binary file to Intel-hex file conversion is performed for some reasons. First of all, disassembling binary files including large spaces directly by disassembler is not practical because the source code file generated includes large numbers of "NOP" instructions in this case. Actually, the hexadecimal code of "NOP" instruction is 00h and it represents the space character at the same time. If a source code program starts with a high address, large spaces are generated in the binary file after it is compiled. If this file is directly disassembled, the disassembler can not distinguish "NOP" instructions from spaces. So, the output file size increases significantly and it takes longer time to process. In order to prevent this , it is considered that more than five "NOP" instructions can not be used consecutively in the source code files. So, if a binary file has more than five consecutive 00h codes it means that these are the space characters. Here, the number five is arbitrarily chosen and any small number is acceptable. This algorithm searches for the 00h codes and finds the spaces. After finding the spaces, addresses between the spaces are found and the Intel-hex file is generated after this operations. The binary codes are converted to hexadecimal numbers in rows when a binary file is translated to an Intel-hex file. The Intel-hex file should be created because Intel-hex codes of the binary file is required for the Intel-hex window of the computer program. If the disassembled file is a binary file, the binary file is also placed to the hex window. The binary file is viewed with its hexadecimal codes and the ASCII forms of that codes in the hex window.

3.2 The Memory Disassembler

The memory disassembler disassembles a specified region of the memory. It is activated when the disassemble button in the memory and registers window is clicked. A message button appears and asks the user to disassemble the memory region of the loaded program or any other memory region. If the memory region of the program is disassembled, the region of the memory between the start and end addresses of the loaded program is disassembled. Alternatively, the user can enter his own start and end addresses manually and the memory region between these addresses is disassembled.

After determining the memory region, the memory disassembler receives the contents of this memory region from the training kit. Then the program places the received data to the memory grid in the memory and registers window and saves this data to the temporary binary file. Actually, the memory disassembler does not disassemble the memory directly. It just disassembles the content of memory from this file. This file is also updated when the memory content is changed.

The memory disassembler disassembles the temporary binary file by calling the dasm API function with the required parameters. The same procedure is followed with the file disassembler except that the

temporary binary file is directly disassembled. Also, the starting address of the program is set to the address of the memory region in this case. For the memory disassembler, the starting address parameter can not be ignored because the starting address of the codes found in memory should be known.

After disassembling the specified memory region, the generated temporary source code file is placed to an unused editor. The user may edit and compile this file after exiting from the run mode. This file can be also loaded back to the training kit and it can be run again. By using the memory disassembler, the user has the ability to modify the loaded program in memory.

The limit of the memory request is 2000 bytes. The same limit is also valid for the memory disassembler as it shares the same data with the memory request. In fact, the size of the many user programs in the memory is less than 2000 bytes. Also the size of the generated file after disassemble process can reach more than ten times of that size. Indeed, the size of one instruction in a binary file is not more than 3 bytes but it corresponds to one line of the generated source code file. The text boxes of the editors are limited with 32 Kb of data. The disassemble process is limited with this size for that reason.

In order to view and edit the loaded programs in memory, the user should select the disassemble the memory region of the program choice form the message box. In this case, the program automatically selects the start and end addresses of the program region. If the start and end addresses are arbitrarily entered, the program can be clipped from the middle.

The memory region manually selected may not include the program codes of the loaded file. There may be old codes of the previously loaded programs or data in the specified region. The memory disassembler may not disassemble the old codes correctly because the exact start address of these codes should be known. In this case, the disassembler may start disassembling from an operand instead of an opcode on the entered start address and the codes are not disassembled correctly.

3.3 The Line Disassembler

The line disassembler generates one line with disassembling a single instruction. The line disassembler is activated if the memory content in the memory grid is changed when the program is paused

When a memory grid is changed, the computer program calls line_disasm API function with the required parameters. This function disassembles the old and new values of the changed memory location. The line_disasm function firstly finds the address of the instruction of the changed code. This process is achieved in a loop with computing the address of the instruction. After it is computed, the old and new codes are disassembled to their instructions. In order to indicate the changes in the operands, labels are not used in this function. This information is saved to a temporary file and the computer program gets this information from this file.

The computer program reads the corresponding instructions of the new and old codes with their addresses and hex codes and indicate them on the line disassembler window. The hex codes of the new and old codes and their instructions are in different colors so the user can notice the changes. The memory location is set with the new value. The program indicated at the grid of the run time window is also updated. This program is updated with disassembling the binary code of the changed program in the temporary binary file by the memory disassembler. The labels of the original program are changed with the generated labels as a result of the disassemble process. The user can also undo the changes by clicking the cancel button of the line disassembler window. In this case, the previous value of the code at the changed memory location is placed to the memory grid. The memory location and the temporary binary file are also updated.. The program at the run time window is rebuilt by disassembling the updated temporary binary file.

The line disassembler is operational for only the memory region of the loaded program. For the memory locations outside this region, the line disassembler is disabled. In this case, the disassembler may not distinguish operands from the opcodes or data from program codes so the process fails.

After changing the memory content of the loaded program, the user can continue to run the loaded program that was paused before changing the memory content [3].

4. CONCLUSION

The hardware and software of the 8085 training kit with the prepared disassemblers are explained in this paper. Both hardware and software of the training kit were tested many times and it is seen that they work properly. It is planned to enhance the abilities of the training kit with the addition of a logic analyzer. In order to accomplish this job, a logic analyzer board will be installed on a PC so that obtaining the timing diagrams of the 8085 microprocessor will be possible. It is also planned to add wireless serial communication ability between the personal computer and the training kit by adding radio frequency serial communication boards. The kit with its experiment board is currently used in our 8-bit microprocessor lab.

REFERENCES

[1] Gaonkar S. R., Microprocessor Architecture, Programming, and Applications with the 8085/8080A. *Macmillan Publishing Company, Inc. 1989, pp 621-631.*

[2] Kandemir, C. M., Özkır, T., Gürlek, H. G., 8085 eğitim seti tasarımı ve gerçekleşmesi, *August 1995, pp. 9-10.*

[3] Polat C., Disassembler programs for the 8085 training kit, *July 1999, pp. 4-5, 12-18.*

[4] Ergün U., Kızılkaya Z., 8085 Eğitim Setinin Seri Haberleşme Kullanan Monitör Programının Hazırlanması, *August, 1996, pp. 33-43.*