Design and Development of Advanced Cross Assembler for 8085 Microprocessor

Shashank Bansal
HMR Institute of Technology and Management, Hamidpur, GGSIP University, ND- 36, India

Avneesh Mittal
Bhaskaracharya College of Applied Sciences, University of Delhi, Dwarka, ND-75, India

Vijay Sharma
National Physical Laboratory, Dr. K. S. Krishnan Road, New Delhi- 12, India.

O.P. Sharma
Kirorimal College, University of Delhi, Delhi-110007, India

T.K. Saxena
National Physical Laboratory, Dr. K.S. Krishnan Road, ND – 12, India

ABSTRACT

This paper describes the development of 8085 cross assembler for students working in their microprocessor lab. The software has been written in Visual BASIC 5.0 and is user friendly. The cross assembler converts any 8085 assembly program to the corresponding operational code and saves it in user defined binary file used for burning the EPROM chip. Minimization of the conversion time is the basic aim achieved in the present software which has been successfully tested on many sample programs and systems.

Keywords: 8085, Assembler, Cross Assembler, Microprocessor.

1. INTRODUCTION

Assembler is a software program that takes an assembly program segment of mnemonics, the source language, and translates it into an equivalent binary file program, the target machine language, which can thus be used to burn the EPROM chip for dedicated application. A simple assembler means that one can develop a program in one platform and run it on the same platform. Cross assembler defines that one can develop a program in one platform run it on other platform. A cross-assembler is just like any other assembler except that it runs on CPU other than the one for which it assembles code. Cross-assemblers are useful as one can use available CPU with memory, disk drives, a text editor, an operating system, and all sorts of hard-to-build or expensive facilities to develop code for another target CPU at different place.

The desired task of a cross assembler has been divided in to four basic passes [1, 2]. The First pass is the *preprocessor*, Second pass is the heart of cross assembler called *lexical analyzer* (parser, symbol table, code generation), the third pass is the, *intermediate code generation*, improves the quality of the generated intermediate language. The Fourth pass of the cross assembler is the op*codes generation*, which is the form of binary executable code.

The assembler/ cross assembler performs more or less isomorphic translation (one-to-one mapping) from mnemonic statements into machine instructions and data. This is in contrast with high-level languages, in which a single statement generally results in many machine instructions.

The basic processor of 8085 is taken here for simplicity, which is a complete 8 bit parallel central processor. An 8085 [3] instruction/ mnemonic consist of operational code and operand in five different addressing modes. It has 256 different instruction codes (including 10 new instructions) comprising 74 different operations (operational code) and their operand combinations. Programming language Visual BASIC 5.0 has been used for developing the 8085 cross assembler because of its powerful string manipulation capabilities and GUI.

2. DEVELOPED SOFTWARE

The cross assembler software has been programmed in the Visual BASIC 5.0 [4, 5]. The developed software is user friendly. It reads the instruction set as well as its corresponding binary code from a predefined *8085.txt* file in the system in use. On running the program it opens a window as shown in Fig 1 and allows the user to browse the existing assembly file or to enter the mnemonic code in the run time also as shown in Fig 2(a). It then allows the user to save the entered code in a user defined *.asm* file, as shown in Fig 2(b). The program is insensitive to the character case. On the press of the *assemble* command the file is assembled into the desired binary *.bin* format as described in the flow chart of Fig 3.

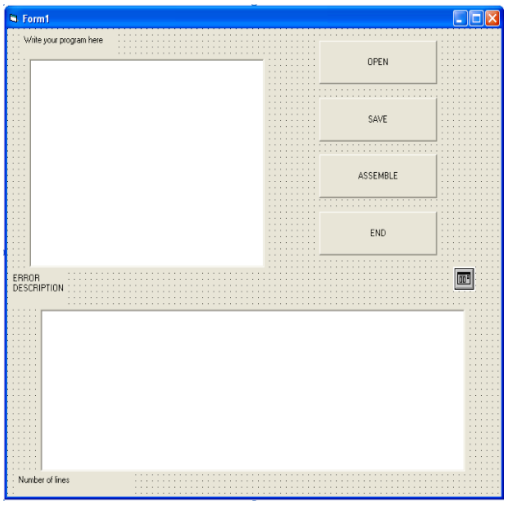

Fig 1: Initial form of the 8085 Cross Assembler developed

The file conversion starts by removing any unwanted characters from the entered mnemonic like comma, space, tabs or blank lines. The resultant file is stored in a *.tmp* file. Thereafter it starts the parser. The parser breaks the assembly line into the parts, *tokens*, to be used by the lexical analyzer as well as lexical grammar. The immediate data, the direct or indirect addresses are also separated and verified. The immediate data is assumed to start from the symbol #. In case

if there is any mismatch with the lexical grammar then it gives the error. The labels or pseudo directives are separated from the entered file. They are separately verified or matched for the proper syntax/ functioning by using a symbol table. If there is no error then it generates the operational code. In case there is a mismatch between the labels it will give error in the label

The speed enhancement has been done in this step by reducing the comparison time with the standard mnemonics table. A two-dimensional table of [26×11] has been generated with the first column containing the starting character from A-Z. Correspondingly other column contains the different possible mnemonics of that character. The operand part of the instruction is compared by finite automata algorithm by generating a tree for an instruction.

An error is generated if any of the mismatch, syntax, and label errors is found. Machine codes are then generated. The generated codes can then be used to burn the EEPROM or directly load in the laboratory kit. User can easily end the compilation by selecting END option available

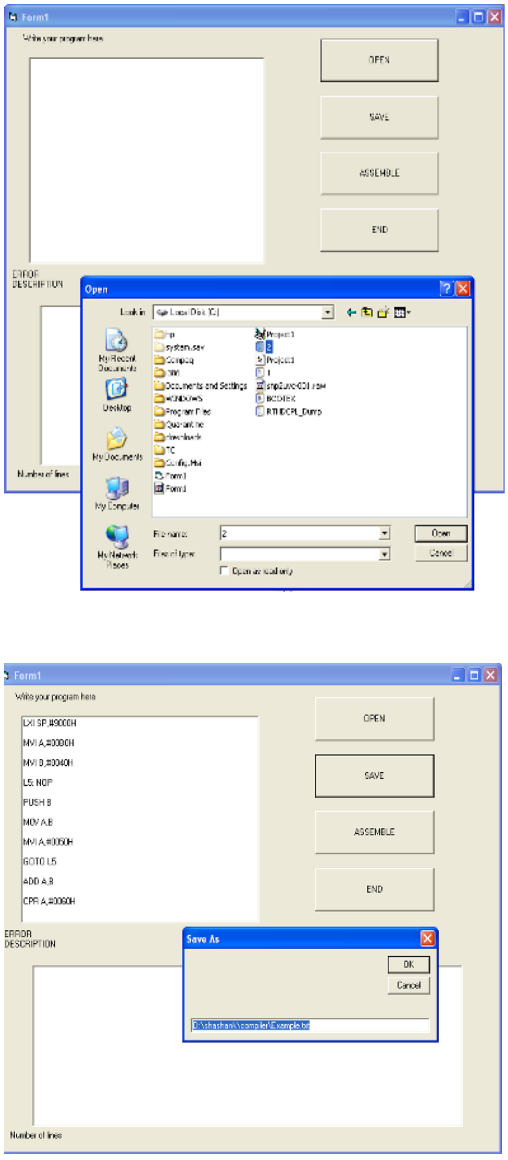

Fig 2 (A, B): Steps of the developed cross assembler to open and save the entered code.

3.RESULTS

The developed software was successfully used to convert various assembly programs to their opcodes saved in the user defined files. One of the examples is shown in Fig.4 (A) & (B). Fig 4A shows some of the errors in the entered assembly program segment. After using the above messages a successful compilation was performed as shown in the Fig 4B. After successful compilation the program generates the binary code and saves it in a desired *.Bin* file.

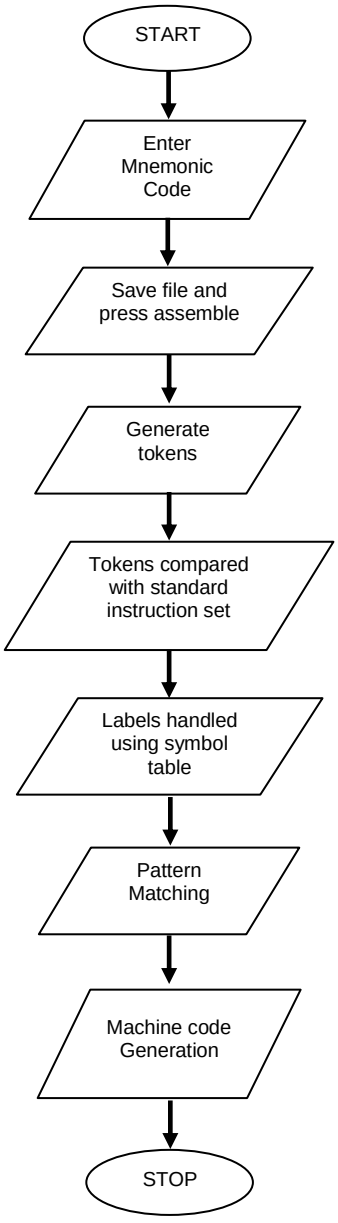

Fig 3: Flow Chart of the developed Software of 8085 Cross Assembler

4. CONCLUSION

The algorithm used in the above software is very general and can be used to design cross assembler software from Intel, Atmel, Zilog, Motorola and other processors and controllers. The developed software is a cost effective and fast way to

generate the *.bin* file. It also gives the computer students an inside knowledge of Parser, lexical analyzer, handling of the instruction set having direct and indirect addressing modes, label handling etc. used in designing of the cross compiler/ assembler.

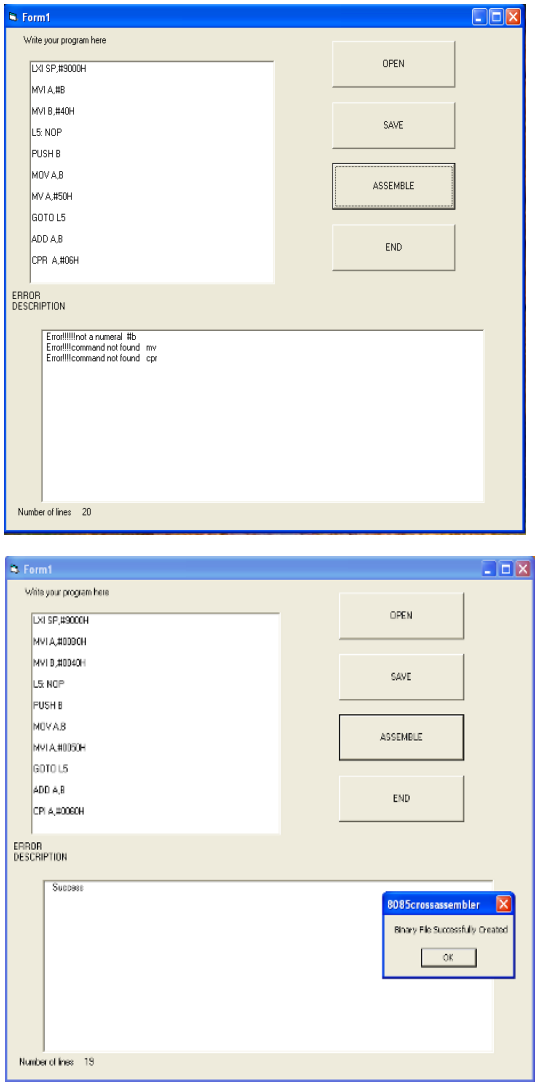

Fig 4: Results (A) Errors message generated by the developed assembler
(B) Successful Compilation after using the above messages of the developed assembler

5. REFERENCES

[1] Allen I. Holub, "Complier Design in C", Prentice Hall of India Pvt. Ltd., 2007.

[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, " Compliers: Principles Techniques and Tools", Pearson Education, 2001.

[3] Ramesh S. Gaonkar, "Microprocessor Architecture, Programming, and Applications with the 8085/8085A", Wiley Eastern Limited.

[4] Gary Cornel, "Visual Basic 5 from Ground Up", TMH Publications

[5] Brian Siler and Jeff Spotts, "Using Visual Basic 6", Que Prentice Hall of India Pvt. Ltd., 2002.