www.ijifr.com | Volume 5 | Issue 1 | September 2017

International Journal of Informative & Futuristic Research | ISSN: 2347-1697

Implementation of Reverse Engineering for Assembler

Paper ID	IJIFR/V5/ E1/ 027	Page No.	8758- 8763	Subject Area	Information Technology
Key Words	Assembler, Reengineering, Software Systems				

1st	Anurag Goyal	M.Tech. Student , Department of Information Technology, Shri Shankaracharya Technical Campus, Bhilai-Chattisgarh
2nd	Madhuri Gupta	Assistant Professor , Department of Information Technology, Shri Shankaracharya Technical Campus, Bhilai-Chattisgarh

Abstract

As discussed in the paper [9], Reengineering of software systems is one of the key areas in the world of software systems and will be gaining more attention in the coming time. In the paper, we discussed about the reverse engineering for one of the crucial technology Assembler. We covered the problems associated with it and proposed few solutions. In this paper; we are going to analyze the solutions proposed by us in the paper [9]. There will be some great takeaways from the solutions, as well as few boundaries and new problems will be introduced. Of course, everything can be resolved with period of time and should be addressed.

I. INTRODUCTION

Assembly language is a low-level language for programming hardware like the CPU, integrated chips, and microcontrollers. An assembly language is simply a machine-specific and non-portable symbolic representation of the underlying machine code that can be interpreted by a machine. The symbols are called mnemonics, and are designed by the manufacturer of the hardware. An assembler is basically a utility that does the same job as a compiler does for high level languages – it translates the mnemonics into respective binary sequences.

ISSN: 2347-1697
International Journal of Informative & Futuristic Research (IJIFR)
Volume - 5, Issue -1, September 2017
Page No. : 8758- 8763

II. REVERSE ENGINEERING FOR ASSEMBLER

As we have seen in the above sections that Assembler is having mnemonics and symbols. So first of all, most common such words are collected and listed down. After collection of keywords, all those are need to be handled and algorithm must be drawn for reading and processing. A list of all the instructions is prepared and their equivalent pseudo code possible for both cases (as discussed in paper [9]) must be written. This list must also include all the possible formats of these instructions, as this is known that each instruction can be used with multiple operands. These operands can be different entities like variables, registers, constant value etc. So, we must be ready with the list of instructions, including different formats, keywords, variables, registers and how these are going to be read by the application and how that will be processed and what output should be generated. Once we are ready with such preparations we can start implementation of the solutions.

III. IMPLEMENTATION OF THE SOLUTIONS

Before we start with solutions separately, first we will see which will be common in both. So first thing we have to do is to prepare list. List of all instructions, keywords, and macros will be prepared and kept aside.Next we have to see, what characters our variables can be formed with. Since this is assembler, so it can have all alphabets, numbers, special characters etc. So our lex should be capable of handling it. So, a pattern needs to be prepared which could handle this.Another thing is that operands can be numbers only. So again, one pattern needs to be defined for numbers. But things don't end here. In assembler, one can have number with different data types, like hexadecimal, binary etc. So one has to handle that as well. First, we have to read this, which could be of format X'1A'. Means there will be data type associated with it. So read all these constants with their data types. Once it is read and passed to yacc, it must also have the expression defined to handle such data types. Once expressions are defined, and then we will have to see if we need to put it as it is or do we need conversion. It has to be processed as per individual's requirement.

Similar, instance can happen with strings as well. In assembler, characters can be defined and used differently and strings are defined and used differently. One has to see and precede accordingly.Once we are done with the listing part, we can move on to next part. Here we will have to define expressions and need to see which all instructions are covered under that expression. Definitely there will separate processing for different instruction.There are many instructions which are specific to assembler only and there is no such instruction or equivalent to that in any other programming language. These types of instructions are handled with utter importance which will allow us to bring some logical points out of the program logic.

Another major problem is having register as one of the operands. Registers are being used for data manipulation, address resolution, saving a return point or many more. So how to handle these are of great importance. For such purpose, few intermediate

ISSN: 2347-1697
International Journal of Informative & Futuristic Research (IJIFR)
Volume - 5, Issue -1, September 2017
Page No. : 8758- 8763

variables are being introduced during the processing. These variables help us to understand the program logic and derive the functionality of the program. But, since these are the variable inserted by us, so one has to be careful while making use of the application. There could be some more problem arise due to this.

Here, we will have to see, that for our proposed solutions, we need to have different processing. As, for line by line conversion, we will have some simple lines to be printed. But, in case of group conversion, we will have to store some data for some of the instructions and for rest we may have same print.So, in this manner we will have to be ready for the implementation of the solutions we proposed.

IV. PROPOSED SOLUTION

There were mainly two solutions proposed which are as follows:

Line by Line Conversion

In this type of conversion, I proposed that each line of the assembler code will be converted into the pseudo code. So, for each and every instruction in assembler, I have pseudo code written. Each instruction will be parsed and will get converted into the pseudo code. The biggest advantage of this is that it will have common format of one type of instruction. It will get standardize due to automation. There are many instructions which perform same type of operation on the operand but are treated with different flavours. So this will help us to bring such instruction under the same umbrella.

Group Conversion

In this type of conversion, a group of assembler code will be translated into the pseudo code. This pseudo code will then be used by the analyst to understand the program logic and derive the functionality of the system. Here, each instruction which will do some data manipulation will get printed, only if it deals with variables. Instructions doing data manipulations but have only register or variable as operands will not get printed.

This approach will be able to generate the precise information from the code and knowledge can be extract easily with this approach. This will be very useful who is from the non technical background.

Tools and Technologies

There are various tools and technologies which we are going to use. These are:

- Lex
- Bison/Yacc
- C

V. RESULT DISCUSSION

Whenever we do something, we do it to get some positive results out of it. The very first thing we sought is cost cutting. And further we would like to spend less time to complete anything and also we would like to do less work. So overall one would look for all the comforts he can draw out of it.

ISSN: 2347-1697
International Journal of Informative & Futuristic Research (IJIFR)
Volume - 5, Issue -1, September 2017
Page No. : 8758- 8763

There are no standard formula or set parameter based on which we can analyze the proposed solutions. But still based on the below lines we would try to analyze.

[10] The amount of time needed to understand assembly code does not grow linear. For example, one can understand a simple function with 10 lines of assembly relatively quickly (lets say 4-5 minutes). As the code grows the more complex it is to understand the code. One needs to catch how different functions work together, so the equation does not result in 8-10 minutes.

As one can made clear the amount of time vastly depends on the complexity of the code, but this is roughly the same for every standard application I would say.

Understanding 100,000 lines of C code takes several weeks (Guess! one did never try to understand 100,000 lines of C code before), depending on the time you're actually working. With this basis I guess it takes several months with smart analysis.

But when we try to use above solutions, we can get the output within few minutes. But this is only time to run the application and we need to include more time to it. There will time taken for the manual updating of the code which could be ready as the input to the tool. Once we get the output, time required for manual updation, rechecking also need to be added. But, with all these we can assure that it will take less time than any other approach.

Another important factor is the time required to do customization of application, as per users requirement. This could be the most time consuming among all the activities involved in the entire process. If the customization is done properly, then it would make it bliss. But, if not than it would become a disaster, which may not be recovered or could become an endless task. So this step could make this entire process either a success or a complete failure.

Though we do not have any set standard, we have tried to analyze it. It varies based on the code written, project to project, complexity involved etc.

VI. LIMITATIONS

Like any other solution or approach, these will also have some limitations. Some of the limitations could be as follows:

- Pre-processing of the original code may be required. This includes removal of comments from the code and all unnecessary parts should also be removed. In short only executable assembler code should be the input for the software.
- This may required many input files, like one with source code alone, another with data section alone, another could be the calls to programs and procedures.
- There are instructions like BAS, BAL, which divert the program flow through different paths. So the data updating done in such routines needs to be captured and at the same time the document should be as per code written not on the basis of actual program flow.
- There are many instructions, which are rare in use. Such instruction may not be covered as initial part. But may require being included when occurred.

ISSN: 2347-1697
International Journal of Informative & Futuristic Research (IJIFR)
Volume - 5, Issue -1, September 2017
Page No. : 8758- 8763

- Similarly, for any instruction there could be many formats. All the possible formats will only be covered only when application is used widely and mostly by all.
- Manual updating of the document will be required once it is received from the application. This could depend on the structure of the assembler program and how it is handled.

These are the few limitations which can be sought at this point of time. We will need to see if some more limitation is there. Also we will have to look for the solutions for these limitations. These should be removable with time.

VII. SUMMARY

In this paper, we have discussed the implementation of the various solutions possible for the reverse engineering approach for Assembler. The methods for implementation of the solutions are discussed. The overall development could be one time cost and can save a lot of time in the future projects.

Few things which are expected from the work are improvement in the process of reverse engineering for the Assembler. This should be helpful in reducing the efforts normally put by the engineers. Also the overall cost incurred and time devoted should be minimized. This should be able to give the high level design of an existing system which could be used for the new system design.

VIII. CONCLUSION

Anyone looking for reengineering for assembler codes can make use of one of the methods present above. This will be able to help them reduce the time to think of an approach and will give a kick start to proceed ahead. Most of the time wasted to decide on the approach will be saved and help them reduce the time and hard work. Once the model is ready, it will take minimum time to customize as per individuals requirement and can save lot of time and manpower.

IX. FUTURE WORK

These given methods are new in the field of reverse engineering for legacy systems having assembler as their base. The future work is to test these tools for suitability to fit on the basis of analysis of current and past data. These tools can be accepted as it is or improved or rejected. Once fit and fine these tools will help in reengineering the legacy software with optimal cost. This work will be beneficial to the both communities, the software managers and the software engineers.

Also the sustainability of the approach and product need to be tested. It has to be analysed by the various people and various types of code needs to be checked. It may be possible that it is not durable for all and may require some alteration. This could be less for some cases and it may be more for some cases.Since it may involve some amount of alteration, so cost also becomes one of the important factors. For few cases, there is

possibility that cost incurred for alteration is much more than the entire process. For such cases, it may not be useful and other alternatives need to be looked for.

X. REFERENCES

[1] E. Chikofsky and J.H.Cross, "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software Engineering journal, (Jan. 1990), pp 13-17.
[2] IEEE Std 1219-1998, In IEEE Standards Software Engineering, 1999 Edition, Volume Two, Process Standards, IEEE Press.
[3] Dr. Ashok Kumar, Bakhshsish Singh Gill, "Maintenance vs. Reengineering Software Systems", Global Journals Inc. (USA), Version 1.0 December 2011.
[4] R. K. Fjeldstad and W. T. Hamlen., ''Application Program Maintenance Study: Report to Our Respondents,'' *Proceedings GUIDE 48*, Philadelphia, PA, 1979. Tutorial on Software Maintenance, G. Parikh and N. Zvegintozov, editors, IEEE Computer Society, April 1983, IEEE Order No. EM453.
[5] Spencer Rugaber, "White Paper on Reverse Engineering", College of Computing, Georgia Institute of Technology.
[6] Lamb, John (June 2008). "Legacy systems continue to have a place in the enterprise". Computer Weekly.
[7] Jean-Marc DeBaud, "Using Executable Domain Models to Implement Legacy Software Re-Engineering", College of Computing, Georgia Institute of Technology.
[8] Barry W. Boehm, Software Engineering Economics, Prentice Hall, 1981.
[9] Anurag Goyal, Madhuri Gupta, "Reverse Engineering for Assembler", IJIFR.
[10] https://reverseengineering.stackexchange.com

XI. AUTHOR'S BIOGRAPHIES

Author Anurag Goyal was born on 9th Feb 1989 and belongs from Bhilai. He has completed his BE (IT) from Shri Shankaracharya College of Engineering and Technology, Bhilai in the year 2010 and currently pursuing MTech (IT) from Shri Shankaracharya Technical Campus, Bhilai. He has worked with Cognizant Technology Solutions, Chennai for around 3.5 years and has experience in legacy systems. He has interest in legacy systems modernization and do some research activities in that area.

Guide Ms. Madhuri Gupta has guided him preparing this research paper. She is currently pursuing PHd from Chhattisgarh Swami Vivekanand Technical University, Bhilai. She is currently Assistant Professor in IT Dept in Shri Shankaracharya Technical Campus, Bhilai and having an experience of around 7-8 years. She has interest in Cyber Crime and security topics.

TO CITE THIS PAPER

Goyal, A. , Gupta, M. (2017) :: "Implementation of Reverse Engineering for Assembler" ***International Journal of Informative & Futuristic Research (ISSN: 2347-1697*****), Vol. (5) No. (1), September 2017, pp. 8758- 8763, Paper ID: IJIFR/V5/E1/027. Available online through- http://www.ijifr.com/searchjournal.aspx**