ARM DECOMPILER

25.10.2017

Gyanesh Anand (2016039)
Aman Roy (2016011)
Suyash Singh (2016105)

Project Plan

Overview

ARM Decompiler

Decompilation is the reverse process of compilation so before understanding what is decompilation , one needs to know what is compilation . Compilation is the process of transforming the high level code i.e. the code in programming language like java or C to assembly language (i.e. the low level language) . This assembly language is designed in such a way that all the abstractions provided by the high level language is taken to the most trivial level, we deal with registers and flags in the assembly language and hence the level of abstraction decreases. A decompiler on the other hand is used to decompile a low level language back to high level language.

Motivation

Compiler's purpose is very clear but why would one need a decompiler is not that obvious. The major purpose of decompiler is to give the source code of any low level language code. Hence there may be a case where one might end up with 100 or 300 lines of code in machine language but the source code is lost, in such a case it is a tedious job to debug 300 lines of low level language and that's where decompiler comes into picture. A decompiler will give a fair idea of logic behind the machine code and will output a pseudo code which will help in understanding the logic behind the code and thereby making it easier for user to debug the code. Other than this decompilers are used in malware analysis because in that case the source code is not present and hence the executable malware file is run through a decompiler to find the target of attack.

Aim

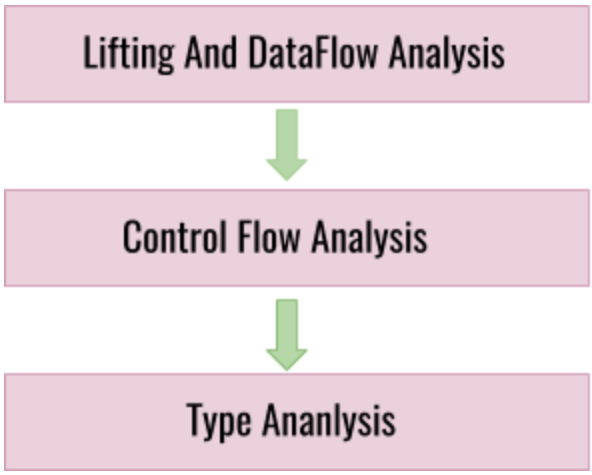
Our project aims to decompile a Low Level code written in ARM to a pseudo high level language code.

Methodology

1. Lifting and DataFlow Analysis.
    - Recognising distinct variables.
    - Detaching Variables from Registers and Addresses.
    - Recover Expressions.
    - Function Return Value.
    - Arguments.

2. Control Flow Analysis (CFA) (Determining control Flow Structures)
    - If - Else
    - For - Else
    - While and Do-While

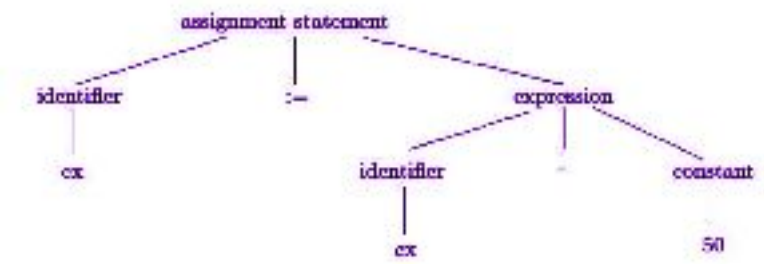
3. Type Analysis
    - Recover Type of Variables
    - Functions and Other pieces of Data.



Steps To Decompiler

1. Syntax Analysis :

- The parser or Syntax analyzer is required to group the programs according to



grammar of the source code of machine language.
- The expression sub cx, 50 is mathematically equivalent to cx:= cx-50 whose parse tree is shown above.
- The main problem while analysing syntax is to decide what is data and what is instruction.
- Also according to the nature of machine language , the parse tree is always hierarchical and has at least two layers of hierarchy.

2. Semantic Analysis :

- The semantic analysis phase checks the source program for the semantic meaning of groups of instructions, gathers type information, and propagates this type across the subroutine.
- In this phase , we deploy various techniques to ensure that the above decoded piece of code makes a good sense and can be used for further operations.
- In order to check for the semantic meaning of a group of instructions we check for sequence of instructions which form a logical entity aka Idioms.
- In many cases , although the instruction is syntactically correct, it is not semantically correct for the machine language we are decompiling, and thus an error needs to be reported.

3. Intermediate Code Generation :

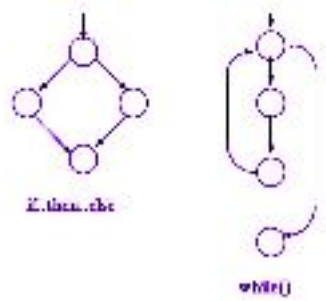
- In this stage an explicit representation of source program is provided to the decompiler so that it can analyse the program.
- This representation must be easy to generate and should be a suitable representation for target language.
- For this three-address representation is used in which an instruction can have at most three expressions.
- So from this representation not only the the mathematical operators of machine language can be operated upon but also we can operate upon more complex expressions which are combinations of operators.

4. Control Flow Graph Generation :

- In this stage we will make a graph of all the subroutines and how control is flowing from one branch to the other branch.
- It will be used to determine the high level control structure of the program.
- Also it is used to remove all the intermediate jumps which were generated due to offset limitations of some other conditional jump in machine language.

5. Data Flow Analysis :

- The data flow analysis phase attempts to improve the intermediate code, so that high-level language expressions can be found.
- The use of temporary registers and condition flags is eliminated during this analysis, as these concepts are not available in high-level languages
- The first set of instructions makes use of registers, stack variables and constants; expressions are in terms of identifiers

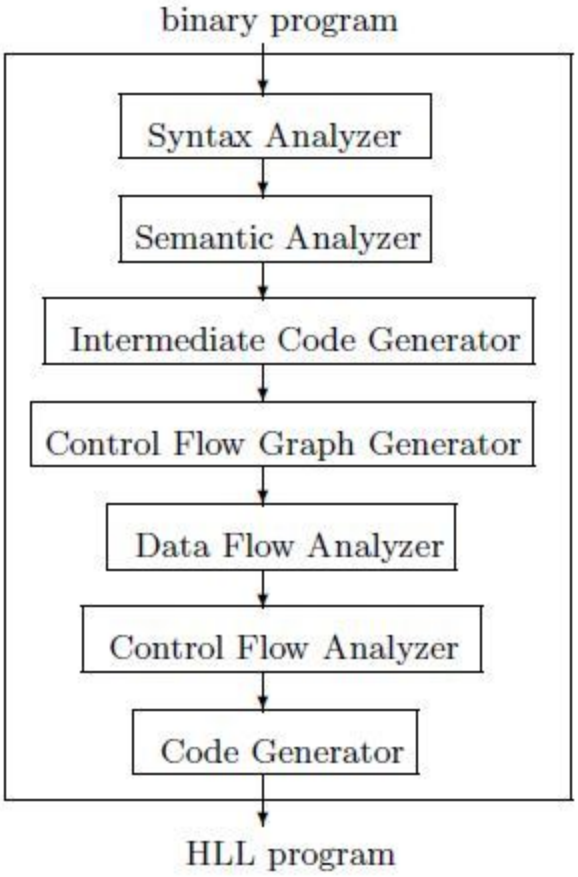


6. Control Flow Analysis :

- In this phase we model to create the flow graph of each subroutine from into a generic set of high level language constructs.
- Also this step will ensure that we our generic set do contains control instruction available in higher programming languages like looping statements , branching statements etc.

7. Code Generation :

- The final phase of the decompiler is to produce some high level programming language from control flow graph and intermediate code for each subroutine.
- Variables name are selected for local stack , argument , and register - register identifiers.
- Similarly subroutines names are also selected.
- All the control structures and intermediate instructions are translated into high level language statement.



Timeline

I. Project Research : 20th - 30th October

II. Implementation Phase : 1st - 10th November

1. Syntax : 3rd November, 20:59
2. Semantic : 3rd November, 22:59
3. Intermediate Code Generator : 3rd November, 23:59
4. Control Flow Graph Generator : 8th November, 02:59
5. Data Flow Analyzer : 8th November, 02:59
6. Control Flow Analyzer : 11th November, 02:59
7. Code Generation : 11th November, 02:59

III. Testing Phase : 11th - 15th November

==========

==========