relipmoC – i386 Assembly code to C code translator

Introduction

Translation from assembly code to high-level language (HLL) code is of importance in the maintenance of legacy code, as well as in the areas of program understanding, porting and recovery of code. The translation process aims not to get the original source program but a program which is identical to the original program with respect to the output generated. The translation process is part of a larger process which converts machine code to HLL code. A program which converts machine code to HLL code is called a ***decompiler.*** We can write a decompiler by first converting the machine code to assembly code and then by making use of the assembly to C (asm2C) translator to get the HLL code.

Why is it needed?

In the context of fixing the year 2000 bug, the Gartner Group estimated that many organizations are missing 3% to 5% of their source code portfolios. This means that a medium-sized information systems organization with a software portfolio of 30 to 50 million lines of code could easily be missing a million lines or more. Further, some large organizations have thousands of lines of code written in assembly code and cannot benefit from state of the art object-oriented techniques unless the assembly code is reengineered into some HLL.
Also, asm2C translators could be used to “crack” virus codes.
These facts point to the need for research into ways of translating machine and assembly code to HLL code, based on sound compiler technology.

Is it possible?

In essence, the problem of asm2C translation becomes one of program comprehension; being able to understand existing code generated by assembler and compiler tools, and to apply techniques which will “undo” what the assembler/compiler has done. This transformation converts low-level assembly code into a higher level of abstraction that resembles today's HLL programs.

Consider that the processor, by executing the machine language program, can successfully figure out what to do with any input, to create the correct output. An assembly code program is a set of instructions, plus a set of data bytes. Each assembly code instruction has semantics - you can define precisely what each instruction does. A HLL is more abstract than an assembly code program - details (such as individual instructions, registers, and so on) are abstracted away. In a way, asm2C translation is the **judicious deleting of unneeded information**. A series of transformations can remove the machine specific aspects of the program semantics. Machine code features such as individual registers disappear with these transformations. Finally, the intermediate representation can be structured, replacing jump instructions with loops and conditionals.

However, fully automated translation of asm2C is not possible. What this means is that automatic (no expert intervention) translation cannot be achieved for all possible programs that are ever written. Further, even if a certain degree of success is achieved, the automatically generated program will lack meaningful comments. Also missing are meaningful variable and function names as these are not normally stored in the assembly code (except when stored for debugging purposes).

The techniques used

The main types of analyses are:

Data flow analysis to recover HLL expressions and statements (other than control-transfer statements), actual parameters, and function return values, and to remove hardware references from the code, such as registers ,condition codes and stack references;
Control flow analysis to recover control flow information, such as loops and conditional statements, as well as their nesting level and;
Type analysis to recover high-level type information for variables, formal and actual parameter types, and function return types.

Assumptions

We assume that,

- the assembly language program used for translation was obtained by compiling a C program,
- the instruction set used(in the assembly language program) is that of Intel 80386 processor,
- the compiler used for compiling is gcc 3.2.2

Limitations

Some factors which hinder the asm2C translation process are:

- User-defined data types like **struct**s, **typedef**s, **union**s and bit-fields add more to the confusion of the asm2C translator. Though it is easy to use structures while writing the code, it is almost impossible to figure out if a variable is a part of a structure or is it a basic type on its own, by looking at the compiled output.
- Use of processor-specific instructions/optimizations.

Decompilers are not necessarily evil - but they do pose an ethical dilemma for many software developers. Decompilers offer great potential for legitimate purposes, but can also be used to steal the source code of competitors, or by hackers to determine weaknesses in the design of software. But just don't blame the decompiler - its the programmer who uses it for intellectual property theft, or the hacker that decompiles the software to find security holes that is at fault!