Vx86: x86 Assembler Simulated in C Powered by Automated Theorem Proving

Stefan Maus\textsuperscript{1}, Michal Moskal\textsuperscript{2}, and Wolfram Schulte\textsuperscript{3}

\textsuperscript{1} Universität Freiburg, Freiburg, Germany
\textsuperscript{2} European Microsoft Innovation Center, Aachen, Germany
\textsuperscript{3} Microsoft Research, Redmond, WA, USA

Abstract. Vx86 is the first static analyzer for sequential Intel x86 assembler code using automated deductive verification. It proves the correctness of assembler code against function contracts, which are expressed in terms of pre-, post-, and frame conditions using first-order predicates. Vx86 takes the annotated assembler code, translates it into C code simulating the processor, and then uses an existing C verifier to either prove the correctness of the assembler program or find errors in it. First experiments on applying Vx86 on the Windows Hypervisor code base are encouraging. Vx86 verified the Windows Hypervisor’s memory safety, arithmetic safety, call safety and interrupt safety.

1 Introduction

The correctness of operating systems is critical for the security and reliability of any computer system. However, debugging and testing operating systems is very difficult: kernel operations are hard to monitor, and algorithms are highly optimized and often concurrent. These factors suggest that one should verify operating systems. In fact, there are currently several projects that try to do just that \cite{21,15,10,19}. However, such projects still leave us far from a practical methodology for verifying real systems.

One gap is in the verification targets. Existing verification projects often use idealistic sequential code written in clean programming languages. In contrast, modern system code is typically multithreaded, racy, written in C and assembler. Assembler is used (1) to access special instructions that are not available in C (like CPUID, which returns some important properties of the processor), and (2) to improve the performance of critical algorithms like interrupt dispatch, context switch, clearing pages, etc. While several verifiers for C exist \cite{11,14,17}, we think that it is imperative to verify the assembler portion of a verified operating system as well.

To address this gap, we developed an automatic static analysis tool, called Vx86, targeted towards the verification of the Windows Hypervisor \cite{45}. Vx86 proves correctness of Intel x86 assembler code with AMD virtualization extensions against procedure contracts and loop invariants. It does so by building on top of other tools. First, Vx86 translates annotated assembler code to annotated C code. The C translation makes the machine model explicit and provides a meaning for the instructions by simulating the instructions on the machine...
state. The resulting C code is then passed to VCC, Microsoft’s Verifying C Compiler [16]. VCC translates the annotated C programs into BoogiePL [9], an intermediate language for verification. Boogie [1] then generates logical verification conditions for the translated C code and passes them on to the automatic first-order theorem prover Z3 [8] to either prove the correctness of the translated assembler program or find errors in it.

We found that the simulation approach is a very good fit for our assembler verification effort. There are two reasons for that, one is technical and the other one is social. The technical reason is that C and assembler are in fact very closely related: both use arbitrary pointer arithmetic, both have a very weak type system (albeit a bit stronger in the case of C). So C verifiers, which are good enough for verifying low level OS code, should be good enough to deal with assembler code as well. Mapping assembler code to C thus obviates the need of implementing a full-blown assembler verifier. The social reason is that the users of the assembler verifier are likely to also use the C verifier for other parts of the code, therefore they can get familiar with only one tool instead of two.

This paper presents the design and use of Vx86. Our contributions are

- the development of a translator from annotated assembler code to C (see Subsection 3.1).
- the development of a semantics of x86 assembler with virtualization extensions by providing a simulator in C (see Subsection 3.2).
- the development of correctness criteria for assembler code (see Subsection 4.2).
- the application of the resulting verifier on the Windows Hypervisor code base (approximately 4,000 lines of assembler code) (see Subsection 4.3).

Section 2 introduces the challenges in assembler verification; furthermore it provides some background on VCC. Sections 5 and 6 discuss related work and conclude.

2 Background

2.1 Running Example: SetZero

We will explain the inner workings of Vx86 with the SetZero assembler code (see Figure 1 on the following page). It is literally taken from the Windows Hypervisor code base; it sets a memory block of 4096 bytes to zero.

This code is written in assembler, because it is optimized for branch prediction, cache lines and pipelines of the processor, something that the Microsoft C compiler cannot achieve.

2.2 Challenges in Assembler Verification

Verifying assembler code is challenging.

- Almost all assembler languages, including Microsoft’s x86 assembler, are un-typed; however, most of the automatic verification tools use type information to help with the problems of aliasing, framing, etc.