Formal Verification of Machine Code
Anthony Fox, Mike Gordon and Magnus Myreen

Instruction Sets

Microprocessors are ubiquitous — they are used to control servers, laptops, tablets, phones, TVs, transportation and a vast range of other digital devices. The behaviour of microprocessors is controlled by low-level software or machine code. An instruction set is a defined collection of machine code instructions, as implemented by a class of processors. Families of instruction sets include: ARM, x86, Power, MIPS and SPARC.

Instruction set architectures (ISAs) are often extremely complex — consisting of hundreds of low-level instructions, each altering a processor’s registers and memory in a wide variety of different ways.

FormalSpecification and Verification

There are applications where software assurance extremely important. Errors in software can have significant repercussions, with a single bug having the potential to cause huge corporate and/or personal loss. Using mathematical models, it is possible to verify that software will always behave as required. Our work involves formally specifying the semantics of the machine code instruction set architecture and using this as the basis for formally verifying the correctness of machine code programs.

HOL4: Interactive Theorem Proving

Interactive theorem provers are software tools that provide assistance in constructing formal proofs. The HOL4 proof assistant derives from Robin Milner’s LCF theorem prover, which was initially developed in the 1970s. The logical foundation of HOL4 is Higher-Order Logic. HOL4 provides an excellent framework in which to write tools for the formal verification of machine code programs.

L3: ISA Specification

The L3 language has been designed to ease the task of constructing ISA models in theorem provers. In particular, L3 acts as an authoring tool for HOL4 ISA specifications. We have L3 written specifications for the ARM and x86-64 ISAs. Advanced tools have been developed in HOL4 for working with these ISA models — these include a decompiler and web interface for exploring the semantics of ARM instructions.

L3 specification for the ARM instructions BL and BLX

Disassembly of section .text:

```
00000000 <hypotenuse>:
  0: e92d4008 push {r3, lr}
  4: ee607a00 vmul.f32 s15, s1, s1
  8: ee407a00 vmla.f32 s15, s0, s8
  c: efe70ae7 vcvt.f64.f32 d16, s15
  10: efe80be0 vsqrt.f64 
  14: eef406b0 vcmp.f64 d16, d16
  18: eef1a10 veq APSP_Rzcv, fpcr
  1c: 0000002e breq 2c <hypotenuse+0x2c> 
  20: eeb70ae7 vcvt.f64.f32 d0, s15
  24: ebfffe bl 0 <sqrt> 
  28: eef80b40 vmov.f64 d16, d0 
  2c: eeb70be0 vcvt.f32.f64 s0, d16
  30: e8bd0008 pop {r3, pc}
```

ARM machine code — as produced by the GCC compiler

Raspberry Pi™ comes with an ARM11 processor

Intel® Manuals

ARM instruction evaluator

```
Disassembly of section .text:
00000000 <hypotenuse>:
  0: e92d4008 push {r3, lr}
  4: ee607a00 vmul.f32 s15, s1, s1
  8: ee407a00 vmla.f32 s15, s0, s8
  c: efe70ae7 vcvt.f64.f32 d16, s15
  10: efe80be0 vsqrt.f64 
  14: eef406b0 vcmp.f64 d16, d16
  18: eef1a10 veq APSP_Rzcv, fpcr
  1c: 0000002e breq 2c <hypotenuse+0x2c> 
  20: eeb70ae7 vcvt.f64.f32 d0, s15
  24: ebfffe bl 0 <sqrt> 
  28: eef80b40 vmov.f64 d16, d0 
  2c: eeb70be0 vcvt.f32.f64 s0, d16
  30: e8bd0008 pop {r3, pc}
```

Web interface