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.

Formal Specification 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 mathematic models, it is possible to verify that software will always behave as required. Our work involves formally specifying the *semantics* of instruction set architectures and using this as the basis for formally verifying the correctness of machine code programs.

HOL4: Interactive Theorem Proving

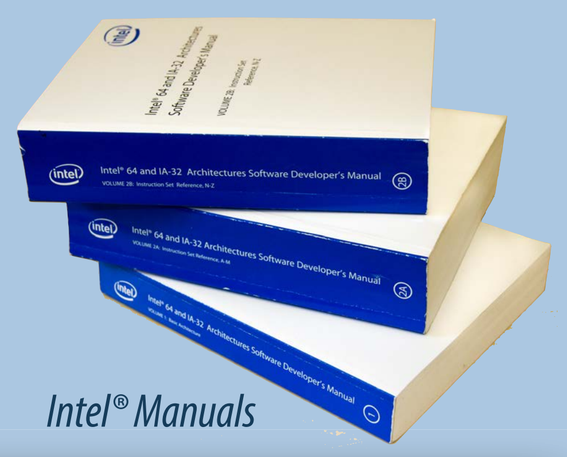
Interactive theorem provers are software tools that provide assistance in constructing formal proofs. The HOL4 proof assistant derives from Robin Milner's LCF theorem prover, which was initially developed in the 1970s. The logical foundation of HOL4 is Higher-Order Logic. HOL4 provides a excellent framework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

```
Disassembly of section .text:

00000000 <hypotenuse>:
   0: e92d4008    push  {r3, lr}
   4: ee607aa0    vmul.f32 s15, s1, s1
   8: ee407a00    vmla.f32 s15, s0, s0
   c: eef70ae7    vcvt.f64.f32   d16, s15
  10: eef10be0    vsqrt.f64   d16, d16
  14: eef40b60    vcmp.f64 d16, d16
  18: eef1fa10    vmrs  APSR_nzcv, fpscr
  1c: 0a000002    beq   2c <hypotenuse+0x2c>
  20: eeb70ae7    vcvt.f64.f32   d0, s15
  24: ebfffffe    bl 0 <sqrt>
  28: eef00b40    vmov.f64 d16, d0
  2c: eeb70be0    vcvt.f32.f64   s0, d16
  30: e8bd8008    pop   {r3, pc}
```

ARM machine code — as produced by the GCC compiler

Raspberry Pi® comes with an ARM11 processor



Intel® Manuals

L3: ISA Specification

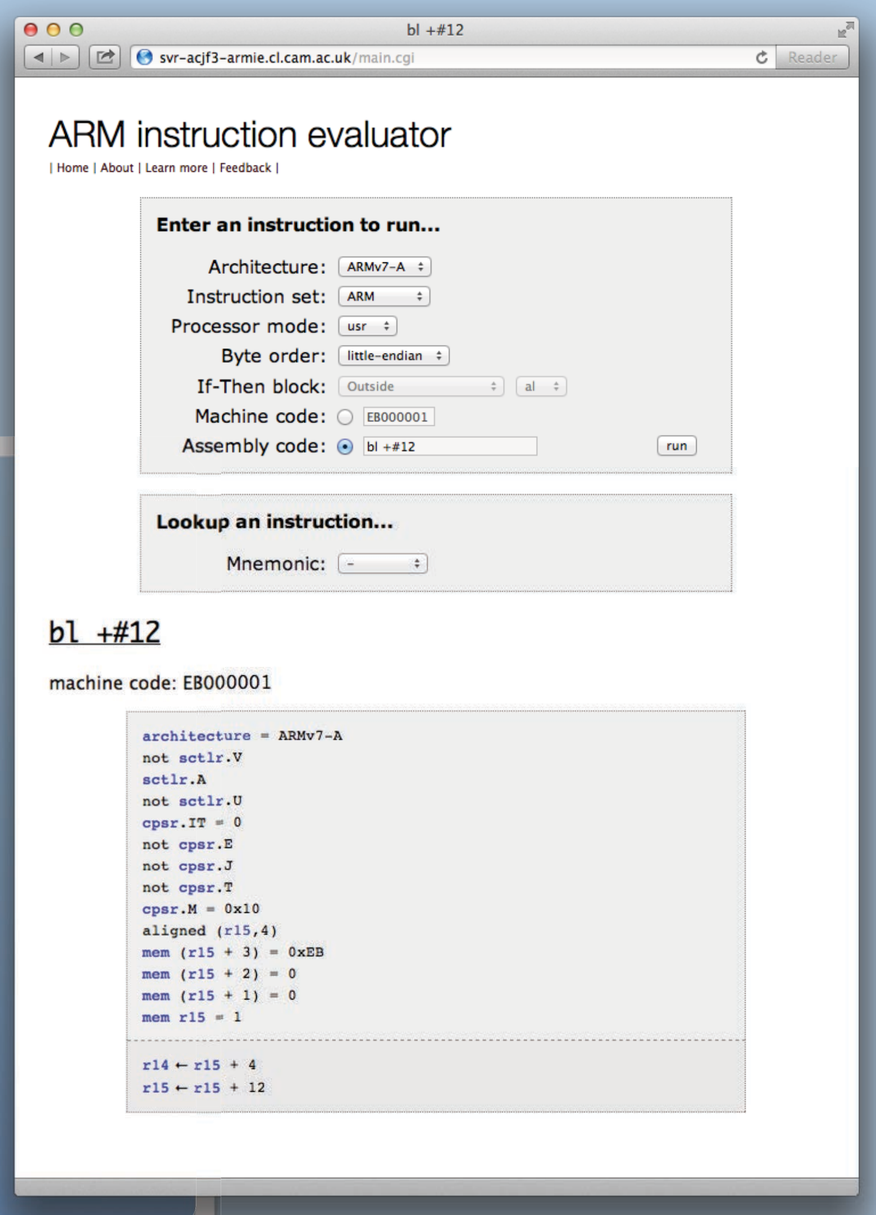
The L3 language has been designed to ease the task of constructing ISA models in theorem provers. In particular, L3 acts as an authoring tool for HOL4 ISA specifications. We have L3 written specifications for the ARM and x86-64 ISAs. Advanced tools have been developed in HOL4 for working with these ISA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

```
-- BL<c>  <label>
-- BLX<c> <label>
define Branch > BranchLinkExchangeImmediate
   ( targetInstrSet :: InstrSet,
     imm32          :: bits(32) )
   =
{  if CurrentInstrSet() == InstrSet_ARM then
      LR <- PC - 4
   else
      LR <- PC<31:1> : '1';
   targetAddress =
     if targetInstrSet == InstrSet_ARM
        then Align (PC, 4) + imm32
     else PC + imm32;
   SelectInstrSet (targetInstrSet);
   BranchWritePC (targetAddress)
}
```

L3 specification for the ARM instructions BL and BLX

```
|- SPEC (STATE arm_proj,NEXT_REL $= NextStateARM,arm_instr,$=)
     (cond (Aligned (pc,4) ∧ Aligned (r13,4)) *
      (arm_exception NoException * arm_undefined und *
       arm_CurrentCondition cond * arm_Encoding enc * arm_CPSR_J F *
       arm_CPSR_E F * arm_CPSR_T F * arm_CPSR_M 16w *
       arm_REG RName_SPusr r13 * arm_REG RName_3usr r3 *
       arm_REG RName_LRusr r14 * arm_REG RName_PC pc *
       arm_MEM (r13 - 8w) b0 * arm_MEM (r13 - 8w + 1w) b1 *
       arm_MEM (r13 - 8w + 2w) b2 * arm_MEM (r13 - 8w + 3w) b3 *
       arm_MEM (r13 - 8w + 4w) b4 * arm_MEM (r13 - 8w + 4w + 1w) b5 *
       arm_MEM (r13 - 8w + 4w + 2w) b6 *
       arm_MEM (r13 - 8w + 4w + 3w) b7)) {(pc,0xE92D4008w)}
     (arm_exception NoException * arm_undefined F *
      arm_CurrentCondition 14w * arm_Encoding Encoding_ARM *
      arm_CPSR_J F * arm_CPSR_E F * arm_CPSR_T F * arm_CPSR_M 16w *
      arm_REG RName_SPusr (r13 - 8w) * arm_REG RName_3usr r3 *
      arm_REG RName_LRusr r14 * arm_REG RName_PC (pc + 4w) *
      arm_MEM (r13 - 8w) ((7 >< 0) r3) *
      arm_MEM (r13 - 8w + 1w) ((15 >< 8) r3) *
      arm_MEM (r13 - 8w + 2w) ((23 >< 16) r3) *
      arm_MEM (r13 - 8w + 3w) ((31 >< 24) r3) *
      arm_MEM (r13 - 8w + 4w) ((7 >< 0) r14) *
      arm_MEM (r13 - 8w + 4w + 1w) ((15 >< 8) r14) *
      arm_MEM (r13 - 8w + 4w + 2w) ((23 >< 16) r14) *
      arm_MEM (r13 - 8w + 4w + 3w) ((31 >< 24) r14)):
thm
```

HOL4 theorem for the semantics of the machine code instruction 0xE92D4008



Web interface

UNIVERSITY OF CAMBRIDGE
Computer Laboratory