A Formal Semantics of SIMD Instruction Sets

Pierre-Evariste Dagand

Do you know what the intrinsics

```
__m128i _mm_ternarylogic_epi64 (__m128i a, __m128i b, __m128i c, int imm8)
```

does on a CPU supporting the AVX512 instruction set? Intel[1] describe it as a

```
Bitwise ternary logic that provides the capability to implement any
three-operand binary function; the specific binary function is
specified by value in imm8. For each bit in each packed 64-bit
integer, the corresponding bit from src, a, and b are used to form a 3
bit index into imm8, and the value at that bit in imm8 is written to
the corresponding bit in dst using writemask k at 64-bit granularity
(64-bit elements are copied from src when the corresponding mask bit
is not set).
```

and then kindly provides the following pseudo-code

```
FOR j := 0 to 1
  i := j*64
  IF k[j]
    FOR h := 0 to 63
      index[2:0] := (src[i+h] << 2) OR (a[i+h] << 1) OR b[i+h]
      dst[i+h] := imm8[index[2:0]]
    ENDFOR
  ELSE
    dst[i+63:i] := src[i+63:i]
  FI
ENDFOR
dst[MAX:128] := 0
```

If you ever have written formal specifications and machine-checked models, you are likely to feel disappointed by the above specification. For instance, from the above description, do you think that it would be possible, given a 128-bits register, to use this instruction to perform a logical `or` on the first 64 bits of the register and a logical `and` on the last 64 bits of the register *in a single instruction*?

[1] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX_512&text=tern&expand=5834,5836

As part of the Usuba[2] project, we produce efficient cryptographical code, exploiting various SIMD (Single Instruction, Multiple Data) instruction sets such as SSE, AVX, AVX2 or AVX512 on Intel platforms. Aside from producing high-performance code, Usuba also aims at being a trustworthy compiler: we would like to prove that the generated code follows the semantics prescribed by the input program. Sadly, in most cases, we can barely make sense of the informal specification provided by Intel: it takes several minutes of intense concentration to check that an instruction does what its name suggest, leaving us at a loss for the most intimidating instructions such as `_mm512_maskz_4dpwssd_epi32`.

Currently, Usuba relies on a C compiler (which can be any of GCC, Clang or ICC) to produce vectorized code from a C program using the Intel intrinsics[3]. This means that the C compiler is part of our trusted computing base. As a result, it becomes nearly impossible to prove the correctness of Usuba down to assembly: we are stuck hoping that the C compiler is correct.

To address these issues, we would like to add a Jasmin[4] back-end to Usuba. Jasmin is a portable assembly language that has been proved correct in Coq. Jasmin would allow us to obtain an end-to-end correctness proof, from Usuba down to vectorized assembly. However, to date, Jasmin only supports a subset of the SSE instruction set.

This project offers to:

- give a Coq semantics to a significant subset of the SSE and AVX instruction sets, using as many generic programming tricks as possible to keep the model as concise as possible;
- develop an automated testing infrastructure, allowing us to exercise the formal semantics against the actual implementations of the instruction set;
- extend the Jasmin language with the instructions supported by our model, updating its correctness proof in the process.

[2] https://github.com/DadaIsCrazy/usuba
[3] https://software.intel.com/sites/landingpage/IntrinsicsGuide
[4] https://github.com/jasmin-lang