Prof. Dr. Armin Biere
Dr. Mathias Fleury

Freiburg, 09/12/2021

Computer Architecture
Exercise Sheet 4 (version 2022-4)

Exercise 1

In the lecture you have seen how an adder works on two's complement integers. What we have not discussed is how to efficiently check if an overflow occurs; i.e. if the result of an addition/substraction requires more bits than the bit width of the adder.

Your task in this exercise is to prove that it is sufficient to only check the carry-in and carry-out of the most significant bit to detect every possible case of an overflow!

(Taken from "Computer Organization and Design" by David A. Patterson, John L. Hennessy)

Exercise 2

In this exercise we are interested in expression the correction of circuits via polynomial. For exampe and is $a \wedge b = a \cdot b$.

a) Construct a two-bit adder based on half-adders.

b) Express the correctness of the adder by a polynomial relating inputs and ouputs both for unsigned equations.

c) Translate the adder to a system of polynomial equations.

d) By combining the equations, derive the equations from Question b. You might have to use that $x^2 = x$ for Boolean values.

e) Simplify $\neg(\neg x \wedge \neg x)$ using the polynomial expression.

To simplify the calculation, we use half-adders instead of gates, but the verification can also be done directly on gate level. One approach to do the testing consists in using Gröbner bases to reduce the target polynomial to prove correctness or build a counterexample.

Exercise 3

a) Add the numbers $0.110_2 (= \frac{1}{2} + \frac{1}{4} + \frac{0}{8})$ and 0.011_2 as floating point numbers (bfloats16).

b) Double the number 0.11011_2 (multiplication with 2) in IEEE standard (bfloats16).

c) Do more normalized or more non-normalized floating point numbers exist in this standard?

d) We consider an ALU with 4 bits of decimal precision (so 5 with the leading digit). Consider $x = 1.000\,08_{10}$. Calculate the result of $x \times x - 1$ as the RISC-V processor would do (so one addition and one multiplication); i.e., after each operation there is a rounding `rn` operation that removes bits that cannot be represented.

   Newer processors and GPUs have a "fused-multiply-addition" (FMA) that does the operation at once, rounding the result only once. Is there any difference in the result? Do understand why you need the unsafe `-ffast-math` option for the compiler to use those instructions?

   (The example is taken from `https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf`)

e) Using our imaginary bfloat8 (1 sign bit, 3 fraction bits, 4 exponent bits, exponent bias 7), calculate:

$$1.010_2 \cdot 2^7 - 1.111_2 \cdot 2^4$$
$$1.010_2 \cdot 2^{-1} - 1.011_2 \cdot 2^{-4}$$
$$1.010_2 \cdot 2^{-1} - 1.100_2 \cdot 2^{-4}$$

No submission is needed. The exercise sheet will be discussed on December 9th, 2021 in the online exercise session, 10:00 at:
`https://uni-freiburg.zoom.us/j/65775356475?pwd=dmUvei8ybDN4RFlmT1JUZnRtYlBGZz09`