CEG 3156: Computer Systems Design (Winter 2024)
Prof. Rami Abielmona
Assignment #1: Arithmetic Circuits

January 11, 2024

Objective and Due Date

The objective of this assignment is to comprehend arithmetic circuits and operations in modern computer architectures. The due date of this assignment will be **Monday January 29, 2024 at 20:00**, to be submitted on Brightspace.

Question I

This question deals with adder/subtractor arithmetic circuitries. Parts b and c are questions B.26 and B.27 in your textbook.

Part a

Design a circuit for the overflow detection block shown in Figure B.5.10 of our textbook (or bottom right Figure of slide 23 of lecture 4). Write the RTL VHDL model of your circuit.

Part b

Rewrite the equations on page B-734 for a carry-lookahead logic for a 16-bit adder using a new notation. First use the names for the CarryIn signals of the individual bits of the adder. That is, use c4, c8, c12, ... instead of C1, C2, C3,... Also, let $\mathrm{P}_{i,j}$ mean a propagate signal for bits i to j, and $\mathrm{G}_{i,j}$ mean a generate signal for bits i to j. For example, the equation

$$\mathrm{C2} = \mathrm{G1} + (\mathrm{P1} \cdot \mathrm{G0}) + (\mathrm{P1} \cdot \mathrm{P0} \cdot \mathrm{c0})$$

can be rewritten as

$$\mathrm{c8} = \mathrm{G}_{7,4} + (\mathrm{P}_{7,4} \cdot \mathrm{G}_{3,0}) + (\mathrm{P}_{7,4} \cdot \mathrm{P}_{3,0} \cdot \mathrm{c0})$$

This more general notation is useful in creating wider adders.

Part c

Write the equations for the carry-lookahead logic for a *64-bit* adder using the new notation from the previous part (b) and using 16-bit adders as building blocks. Include a drawing similar to Figure B.6.3 in your solution.

Part d

Design a 1-bit full-subtractor, showing the truth table, optimization scheme (e.g. K-map) and the resulting logic implementations of the difference and borrow outputs. Your inputs should be a minuend, a subtrahend and the previous borrow.

Question II

This question deals with multiplication circuitries. For part b of this question, you might want to look at the implementation of **nxn array multipliers**.

Part a

Prove that the multiplication of two n-digit numbers in base B gives a product of no more than $2n$ digits.

Part b

Design a 4-bit by 4-bit multiplier, using full adders and AND gates only. You can show the logical implementation of a full adder circuit first, then abstract it away in your multiplier circuit, by including a Full Adder (FA) box. Test your multiplier by showing it works on the multiplication of 11x5.

Question III

This question deals with division algorithms.

Part a

The division algorithm in slide 17 of lecture 5 (Advanced Arithmetic Circuitries) is called *restoring division*, since each time the result of subtracting the divisor from the dividend is negative you must add the divisor back into the dividend to restore the original value. Recall that shift left is the same as multiplying by two. Let's look at the value of the left half of the Remainder again, starting with step 3b of the divide algorithm and then going to step 2:

$$(\text{Remainder} + \text{Divisor}) \text{ x } 2 - \text{Divisor}$$

This value is created from restoring the Remainder by adding the Divisor, shiting the sum left, and then subtracting the Divisor. Simplifying the result we get

Remainder x 2 + Divisor x 2 - Divisor = Remainder x 2 + Divisor

Based on this observation, write a *nonrestoring division* algorithm using the notation of the flowchart on slide 17 that does not add the Divisor to the Remainder in step 3b. Show that your algorithm works by dividing $(0000\ 0111)_2$ by $(0010)_2$.

Part b

Derive the floating-point algorithm for division as we did for addition and multiplication on pages 250 through 258. First divide $(1.110)_{10}$ x 10^{10} by $(1.100)_{10}$ x 10^{-5}, showing the same steps that we did in the example starting on page 253. Then derive the floating-point division algorithm using a format similar to the multiplication algorithm in Figure 3.17 on page 258.

Question IV

This question deals with floating-point arithmetic. Assume that for our floating-point representation here, we have a ceg3156-precision as shown in figure 1.

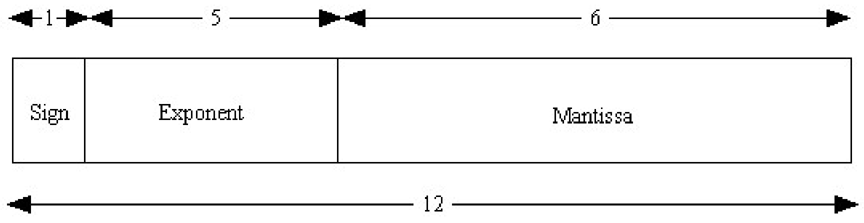

Figure 1: Ceg3156-precision floating-point number representation

Here are a couple of notes to keep in mind while working with ceg3156-precision:

Sign 1-bit wide, and used to denote the sign of the number (0 signifies + and 1 signifies -);

Exponent 5-bit signed exponent in excess-15 representation;

Mantissa 6-bit fractional component.

Part a

Show the IEEE-754 binary representation for the floating-point number $(25.125)_{10}$ in ceg3156-precision.

Part b

Show the IEEE-754 binary representation for the floating-point number $(-1023.6666...)_{10}$ in ceg3156-precision.

Part c

What are the smallest and largest numbers representable in ceg3156-precision format ?

Part d

Add, subtract, multiply and divide the following two operands:
A = 1 00010 011001
B = 0 01110 111000

in ceg3156-precision floating-point representation, and represent their results, both in binary and decimal notation.

Bonus Question

Prove that if two 2's complement numbers are added, the overflow bit is the XOR of the carry-in and carry-out of the most significant bit.