Computer Architecture Homework 2

110062219

April 15, 2023

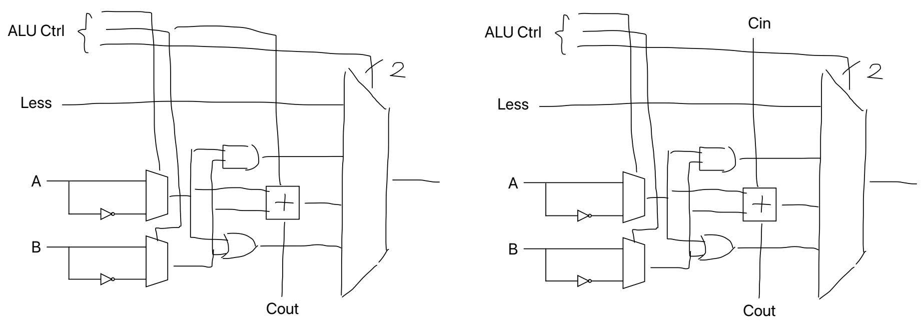
Contents

1 Unsigned ALU

There's few difference for and, or and add operations between signed and unsigned numbers. The only one is that the overflow of the addition occurred if and only if the carry out of MSB is true.

As for subtraction, the definition of 2's complement is still applied. When subtracting an integer s from another integer m, we add its 2's complement, $2^n - s$. Since $m - s \geq 0 \iff m + (2^n - s) \geq 2^n$, the difference is non-negative and $m \geq s$ if and only if the addition occurred overflow!

Please note that for 0 we should take its 2's complement by adding 1 to $2^n - 1$. As a consequence, overflow also occurs.



(a) Leftmost one

(b) Others

Figure 1: 1-bit unsigned ALU

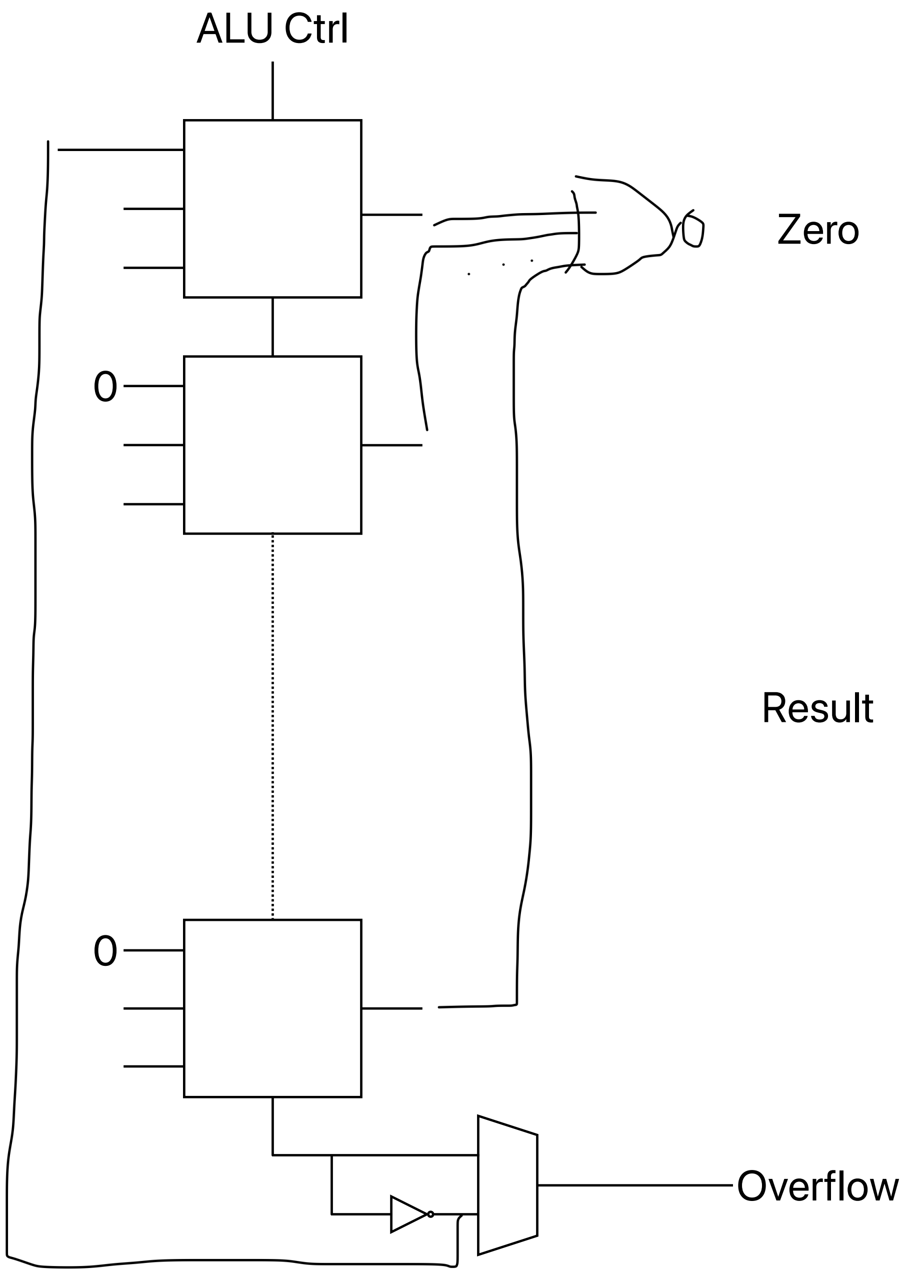


Figure 2: 64-bit unsigned ALU

2 Multiplications

2.1 Ver. 1

Table 1: Multiplication Ver. 1

Multiplicand	Multiplier	Product
0000 0111	1101	0000 0111
0000 1110	0110	0000 0111
0001 1100	0011	0010 0011
0011 1000	0001	0101 1011
0111 0000	0000	0101 1011

2.2 Ver. 2

Table 2: Multiplication Ver. 2

Multiplicand	Product / Multiplier
0111	0111 1101
0111	00111 110
0111	100011 11
0111	1011011 1
0111	01011011

The 4 underlined bits in each row are the rightmost 4 bits in the next row. By adding the left 3 bits of the product and the multiplicand, we could obtain the left half of the next product.

3 Divisions

3.1 Ver. 1

Table 3: Division Ver. 1

Remainder	Divisor	Quotient
0000 0111	0110 0000	0000
0000 0111	0011 0000	0000
0000 0111	0001 1000	0000
0000 0111	0000 1100	0000
0000 0001	0000 0110	0001

3.2 Ver. 2

Table 4: Division Ver. 2

Remainder / Quotient	Divisor
$\underline{0000}$111 0	0110
$\underline{0001}$11 00	0110
$\underline{0011}$1 000	0110
$\underline{0111}$ 0000	0110
0001 0001	0110

Initially, the remainder is shifted left by 1 bit. In each row, we check whether the leftmost 4 bits of remainder are greater than or equal to the divisor. If true, the former would be subtracted by the latter and we append 1 to the quotient. Otherwise, we append 0 to the quotient.

4 IEEE 754 Single Precision

4.1 Two Floating-point Numbers

$$X = 0.3125 = \frac{3125}{10^4} = \frac{5^5}{2^4 \times 5^4} = \frac{5}{2^4} = 5 \times 2^{-4} = (4+1) \times 2^{-4}$$

$$Y = -15.98765 = -\frac{1598765}{10^5} = -\frac{511 \times 5^5}{2^5 \times 5^5} = -(\sum_{i=0}^{8} 2^i) \times 2^{-5}$$

And $-2 + 127 = 125_{10} = 01111101_2$, $3 + 127 = 130_{10} = 10000010_2$.

Table 5: Bit Representations

	Sign	Exponent	Fraction
X	0	01111101	01000000000000000000000
Y	1	10000010	11111110000000000000000

4.2 Their Sum

1. Align exponents. So X equals `0 10000010 00001010000000000000000`.
2. Add fractions. We get `11110101000000000000000`.
3. Normalised result. Not necessary in this case.
4. Round. Not necessary, too.

Hence we have $X + Y =$ `1 10000010 11110101000000000000000`.

4.3 Their Product

1. Add exponents. We get $-2 + 3 = 1$.
2. Multiply fractions. $1.01_2 \times 1.1111111_2 = 10.0111111011_2$.
3. Normalised result. The exponent increases by 1.
4. Round. Not necessary in this case.
5. Determine the sign. It's negative.

Hence we have $X \times Y =$ `1 10000001 00111111011000000000000`.

5 *Quarter* Precision

5.1 Largest Negative "Normalised" Number a_0

$$a_0 = -1 \times 1 \times 2^{-2}$$

5.2 Smallest Negative "Denormalised" Number a_1 & 2nd Smallest Negative "Denormalised" Number a_2

$$a_1 = -1 \times (0.1111)_2 \times 2^{-2}$$

$$a_2 = -1 \times (0.1110)_2 \times 2^{-2}$$

5.3 Difference Between a_0 & a_1, a_1 & a_2

$$|a_2 - a_1| = |a_1 - a_0| = (0.0001)_2 \times 2^{-2} = 2^{-6}$$

Denormalised numbers are distributed evenly between the largest negative normalised number and the smallest positive normalised number.

5.4 Convert `0x5C`

Since `0x5C = 0b 0 101 1100`, it represent $(1.1100)_2 \times 2^2 = 4 + 2 + 1 = 7$.

5.5 Approximation U for -5.7

$$5.7 = 2^2 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} + \cdots \approx (1.0110\ 110)_2 \times 2^2$$

Since guard bit, round bit are both 1, we take $U = (1.0111)_2 \times 2^2 = 5.75$. So the error is 0.05.

6 Compare `floats`

The function would return true if and only if any of the following conditions is satasified:

- Both x, y are 0.
- $x \geq 0 \wedge y \leq 0$
- $x \geq 0 \wedge y \geq 0 \wedge |x| \geq |y|$
- $x \leq 0 \wedge y \leq 0 \wedge |x| \leq |y|$

These above cover all of the cases that $x \geq y$. Hence the claim is correct.