FIT1047 Tutorial 2

Topics

- 2's complement, floating point
- Boolean logic, logic gates, logisim

Instructions

- The tasks are supposed to be done in groups. In some tasks, it might be useful to have different roles for people in the group.

Task 1: 2's complement

1.a Assume you have 8 bits to represent binary numbers. What decimal value does the binary number **10110101** have in the different notations?

(i) **unsigned binary number**

In unsigned binary number, all the bits represent the number.

$10110101_2 = 1\times2^7+0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0 = 181_{10}$

(ii) **sign-magnitude notation**

In this notation, the left bit represents the sign of the number: 0 for positive and 1 for negative. Thus, the number is negative. The actual value of the number is then just the binary representation of the remaining 7 bits:

$10100101_2 = -(0\times2^6+1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0 = -53_{10}$

(iii) **1's complement**

Let N = given number 10110101_2.

The 1 in the most significant bit indicates that N is a negative number. Thus, the number was derived by flipping all bits of the positive number with the same magnitude. 1's complement of N will give this corresponding positive number (i.e. flipping all 1s to 0s and 0s to 1s).

1's complement of N is 01011010.
$01011010_2 = 0\times2^7+1\times2^6+0\times2^5+0\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0 = 74_{10}$
Thus, the actual value is -74.

(iv) **2's complement**

Let N = given number 10110101_2.

The 1 in the most significant bit indicates that N is a negative number. Calculating 2's complement of N backwards (which is the same operation as the 2's complement) will give the corresponding positive number (i.e. flipping all bits and then adding one bit, just discarding any carry bit).

2's complement of N is $01001010 + 1 = 01001011$.

$01011011_2 = 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 75_{10}$
Thus, the actual value is -75.

1.b Create a table with all possible 4-bit binary values in one column and the decimal they represent in the different notations.

4-bit binary	unsigned int	sign-magnitude	1's complement	2's complement
0000	0	+0	+0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

1.c Assume you want to use a bit-wise approach to add 2 n-bit 2's complement numbers x+y=z.
You need to check for overflow.
Use the following notation:
$x = x_n x_{n-1} x_{n-2} ... x_1$
$y = y_n y_{n-1} y_{n-2} ... y_1$
$z = z_n z_{n-1} z_{n-2} ... z_1$
You can use a bit-wise + operator with one bit result and one bit for the carry:
$0 + 0 = 0$ with carry bit 0
$0 + 1 = 1$ with carry bit 0
$1 + 0 = 1$ with carry bit 0
$1 + 1 = 0$ with carry bit 1
For $i = 1, .., n$ the carry bit is denoted by c_i.

(i) For the actual addition, what do you need to do in steps $i = 1, ..., n$?

For each step add x_i, y_i and c_{i-1} to get c_i and z_i. For $i = 1$ you need to assume that $c_0 = 0$ and c_n is just discarded.

(ii) In which steps do you need to check for overflow conditions? What do you need to check?

- If $x_n = y_n = 0$ (both are positive) we need to check in step n if $z_n = 0$. If $z_n = 1$, we have an overflow.
- IF $x_n = y_n = 1$ (both are negative) we need to check in step $n - 1$ if $c_{n-1} = 1$. If $c_{n-1} = 0$, we have an overflow.

1.d In a (rather small) system, there is a floating point representation with one sign-bit, a 3-bit exponent and a 4-bit significand. Assume that the significand is normalized without any assumed bits (i.e. the significand always starts with "1."). The exponent uses 2's complement and exponents with all zeros and all 1s are allowed.

What is the smallest positive number and what is the largest positive number that can be stored on this system? Lets look at the exponent in 2's complement:

000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Since the significand is normalized without any assumed bits, the most significant bit of the significand must be non-zero (i.e. 1 in binary). Normalized means, that there is only one possible representation for each number and the description defines this number to start with a 1 before the devimal point.
Carefully looking at the description, and at the previous tasks above, you might have realised that there is no information on the notation of the significand. As floating point numbers usually reserve a bit for the sign, we can just use binary numbers.

Smallest positive (non-zero) value = $1.000_2 \times 2^{-4}$

Largest positive value = $1.111_2 \times 2^{+3}$

Task 2: Boolean logic, logic gates, logisim

2.a
- Download *logisim evolution* from Moodle
- Start logisim by clicking on the file or executing java -jar logisim-evolution.jar

2.b Make yourself familiar with logisim by doing a few very easy tasks:

- place one AND gate, 2 inputs and one output
- connect them
- poke the inputs to try simulation
- do the same with OR and NOT
- try some combinations of gates

2c In addition to AND, OR and NOT, there are a number of other Boolean logic operators. Examples are **XOR**, the exclusive or, and **NAND**, a negated AND. Their truth tables are shown in the tables above.

Build XOR and NAND in logisim only using AND, OR, and NOT gates. Use the simulation to test your result.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(a) XOR

A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

(b) NAND

2d (Additional advanced task:) Use logisim and only AND, OR and NOT to build a circuit for the following function:

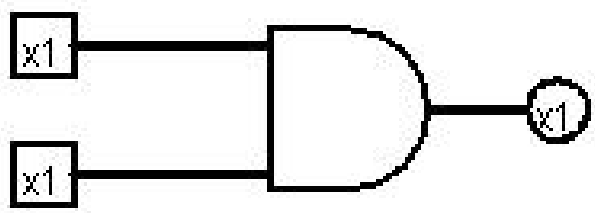
$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC$

How many gates do you need? Simplify the function first, e.g. by using k-maps.

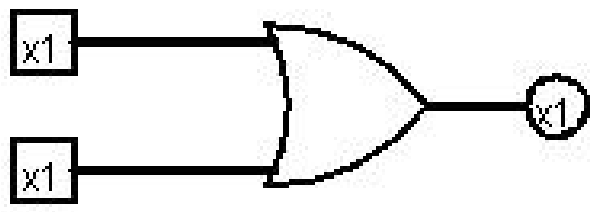
One possible solution would need 6 gates:

$= C(\bar{A} + B) + A\bar{B}$

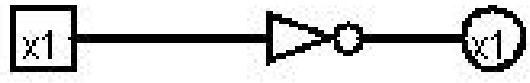
AND Gate



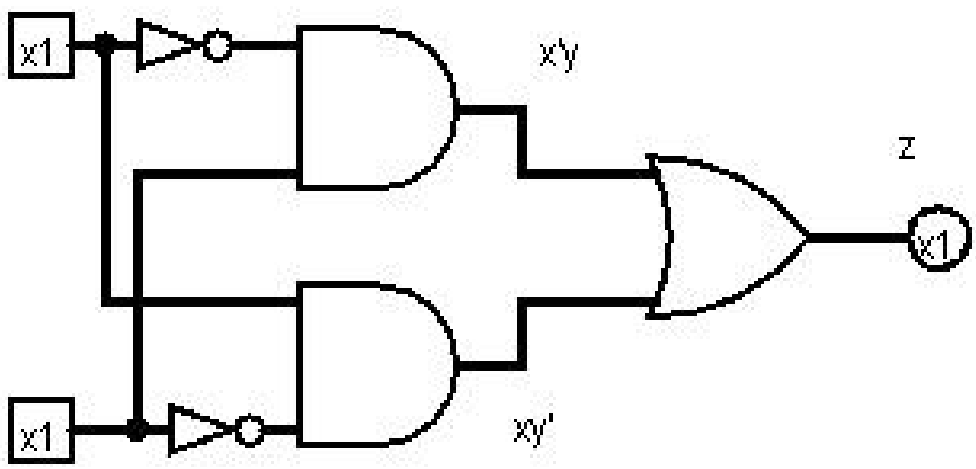
OR Gate



NOT Gate



x XOR y = x AND y' OR x' AND y



x NAND y = NOT (x AND y)