CS 240 Lab 3
Basic Digital Circuits

- **Review of Two's Complement and Overflow**
- **Exclusive Or**
- **Adder**
- **Bit Puzzles**

Two's Complement and Overflow

Given n bits, the range of binary values which can be represented using

Unsigned representation: $0 \rightarrow 2^{n} - 1$

Signed representation: $-2^{n-1} \rightarrow 2^{n-1} - 1$, MSB is used for sign

Two's Complement (signed representation):

Most significant /leftmost bit (0/positive, 1/negative)

Example: given a fixed number of 4 bits:

1000_2 is negative.
0111_2 is positive.

Overflow

Given a fixed number of n available bits:

Overflow occurs if a value cannot fit in n bits.

Example: given 4 bits:

The largest negative value we can represent is -8_{10} (1000_2)
The largest positive value we can represent is $+7_{10}$ (0111_2)

Overflow in Addition

When adding two numbers with the same sign which each can be represented with n bits, the result may cause an overflow (not fit in n bits).

An overflow occurs when adding if:

- Two positive numbers added together yield a negative result, or
- Two negative numbers added together yield a positive result, or
- The Cin and Cout bits to the most significant pair of bits being added are not the same.

An overflow cannot result if a positive and negative number are added.

Example: given 4 bits:

$$\begin{array}{r} 0111_2 \\ \underline{+\ 0001_2} \\ 1000_2 \end{array}$$

1000_2 overflow NOTE: there is not a carry-out!

In two's complement representation, a carry-out does not indicate an overflow, as it does in unsigned representation.

Example: given 4 bits,

$$\begin{array}{r} 1001_2\ (-7_{10}) \\ \underline{+\ 1111_2\ (-1_{10})} \\ 1\ 1000_2\ (-8_{10}) \end{array}$$

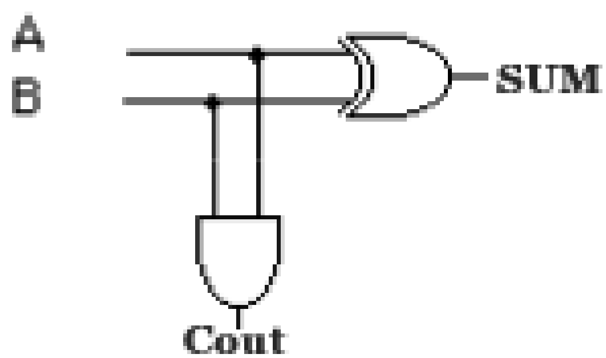
$1\ 1000_2\ (-8_{10})$ no overflow, even though there is a carry-out

Exclusive Or
Useful for comparisons

A **parity bit** is an extra bit of information which is sent when data is transmitted, to check for errors in transmission. For a given set of bits, the number of bits whose value is 1 is counted. The parity bit is an extra bit which is also sent with the original data. The party bit is set to 0 or 1 to make the total number of 1 bits even.

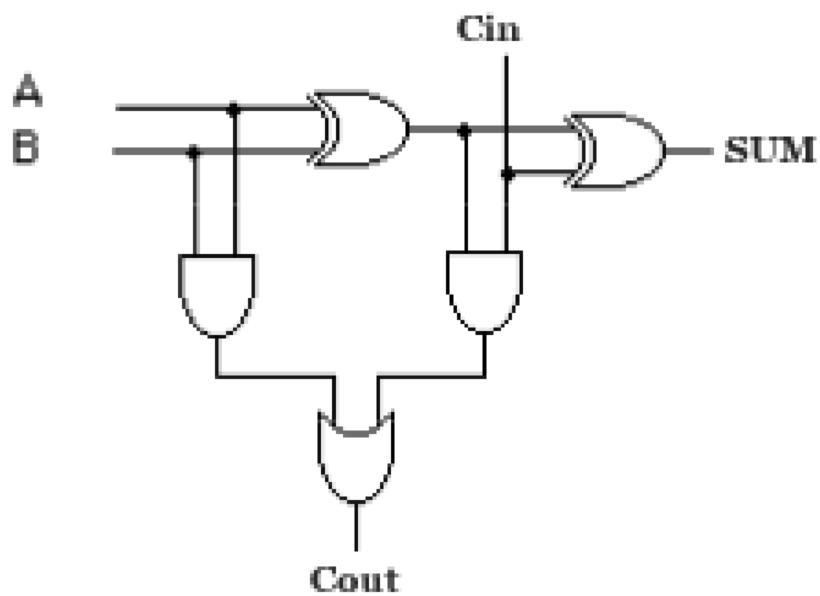
A	**B**	**C**	$\mathbf{P_{even}}$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Half-Adder – adds two one-bit values



A	B	Sum	Cout
0	0		
0	1		
1	0		
1	1		

Full Adder – incorporates a carry-in



A	**B**	**Cin**	**Sum**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sum = A \oplus B \oplus Cin$

$Cout = AB + (A \oplus B)Cin$

n-bit adder = n 1-bit adders

Carry-out of each adder = Carry-in of the adder for next two most significant bits being added

Bit Puzzles

Example:

```
/* isPower2 returns 1 is x is a power of 2, and 0 otherwise
   isPower2(5) = 0, isPower2(8) = 1, isPower2(0) = 0

    No negative value is a power of 2

    Legal operations: ! ~ & ^ | + << >>

    Max operations: 20

    Rating: 4
*/

int isPower2(int x) {
  return 2
}
```

You must write C code to return the correct value for a given input

Constants ,must not be larger than 0xFF (decimal 256)

You may not use conditionals or loops

Tips

Although integers are 32-bit values in this program, assume a smaller number of bits in your handwritten examples to make your binary numbers easier to work with

Handwrite some specific binary values and manipulate them with boolean operators.

Here are some simple manipulations and tips which may help you find a solution:

- Complement the number
- Add and/or subtract 1
- Mask (bitwise AND with a mask value to isolate bits)
- Shift left and then right again (or vice versa)
- Use Exclusive OR to compare values
- Bitwise OR a general solution with a special case (such as 0)
- !(0) = 1, but !(any other number) = 0