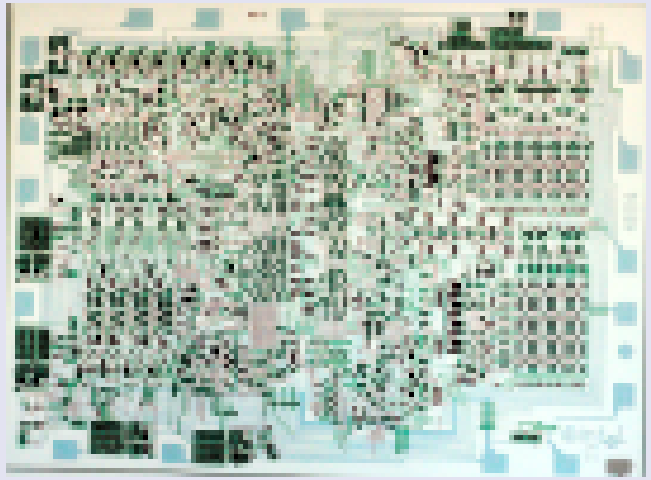

# Final Review: Binary Math

CSIS11: **Computer Architecture and Organization**

Chapter 2 in Patt and Patel: Introduction to Computing Systems...

# Binary Math Notes

- AI Issues
- Unsigned vs. Signed Math

# AI

- Be very careful if you are using AI to generate test questions and answers in binary math
- found it to be about 50% correct.
- frequently forgets the carry bit halfway through a calculation
- After spending 20 minutes double checking my work, I asked AI to re-calculate and it admitted its mistake.

# Signed Integers

## Sign Bit Behavior:

**Overflow**: Result incorrectly changes from positive to negative
**Underflow**: Result incorrectly changes from negative to positive

## Underflow Condition

An *underflow* is the result of a 2's complement operation which exceeds the range of -128 - 127 on the "*low*" or negative side.

## Additional notes:

An important point about binary addition and subtraction: for basic arithmetic operations, you do not need to know whether the numbers are signed or unsigned. Here's why:

1. Addition and subtraction operations are identical for both signed and unsigned binary numbers. The actual bit manipulation process remains the same regardless of how we interpret the numbers.
2. The only difference lies in how we interpret the results, not in how we perform the calculations.
3. This is because binary addition and subtraction (using 2's complement) work the same way for both signed and unsigned integers.

## When does it matter?

The only time you would need to know if the numbers are signed or unsigned would be:

- When interpreting the final result's meaning
- When checking for overflow conditions

## Two's Complement Carry

In 2's complement representation, a final carry bit is discarded because it signifies that the result of the addition is within the representable range for the given number of bits.

# Addition and Subtraction Problems

1. Add the following binary numbers:

```
   01101101
+  00110010
```

1. 01101101 + 00110010 = 10011111
   (109 + 50) = 159

2. Subtract the following binary numbers:

```
   10110011
-  01001101
```

2. 10110011 - 01001101 = 01100110

   (Subtraction performed by adding 2's complement of 01001101, which is 10110011)

   179 - 77 = 102

3. Add the following 2's complement numbers:

```
   11001100
+  10110101
```

3. 11001100 + 10110101 = 10000001
   (In 2's complement: -52 + -75 = -127)

4. Subtract the following 2's complement numbers:

```
   01110111
-  10001100
```

4. 01110111 - 10001100 = 11101011
   (In 2's complement: 119 - (-116) = 119 + 116 = 235, which wraps to -21 due to overflow)
   Remember, as these are 2's complement numbers, to determine the value of the second number, one needs to take the 2's complement of it, which is 01110100 or 116 and it is negative, thus -116

5. Add the following 2's complement numbers and indicate if overflow occurs:

```
   01111010
+  00111101
```

5. 01111010 + 00111101 = 10110111

   (In 2's complement: 122 + 61 = 183)

   Yes, overflow occurs (two positive numbers yielding a negative result)

6. Subtract the following 2's complement numbers and indicate if underflow occurs:

```
   10000101
-  00001110
```

6. 10000101 - 00001110 = 01110111

   (In 2's complement: -123 - 14 = -137)

   Yes, underflow occurs (result should be more negative than can be represented)

7. Add the following 2's complement numbers:

```
   10110010
+  11001001
```

7. 10110010 + 11001001 = 01111011

   (In 2's complement: -78 + -55 = -133, which wraps to 123 due to underflow)

8. Perform the following subtraction of 2's complement numbers:

```
   00111100
-  01010101
```

8. 00111100 - 01010101 = 11100111
   (In 2's complement: 60 - 85 = -25)

9. Add the following binary numbers:

```
   11110000
+  00001111
```

9. 11110000 + 00001111 = 11111111 or 255 (assume unsigned integer)

10. Subtract the following 2's complement numbers:

```
   01100100
-  11110010
```

10. 01100100 - 11110010 = 01110010

    (In 2's complement: 100 - (-14) = 100 + 14 = 114)