Binary Arithmetic

Binary Addition

- basically the same as adding regular numbers, but digits can't exceed 1
    - i.e 1001 0101 + 0001 1100 = 1011 0001
- binary numbers store a limited amount of space, i.e, 8 bits can only store up to 256.

Adder Circuits

- A **Half Adder** can add two single-bits together
    - it can be made using an XOR gate and an AND gate, calculating the sum and the carry respectively
    - `HA` in block notation
- A **Full Adder** can add three bits, and is made by combining half adders

Signed Binary Representation

- Signed Binary attempts to indicate whether a binary number is positive or negative.
- **Signed Bit Representation**
    - uses the first bit in the sequence to indicate whether the number is positive or negative.
        - 0 - positive/zero
        - 1 - negative
    - This breaks arithmetic.
- **1s Complement Representation**
    - uses the first bit to indicate polarity as well, but also flips all bits in negative numbers
    - arithmetic is better, but not perfect
        - 0001 0101 + 0001 1100 = 1001
- **2s Complement Representation**
    - same as 1s Complement Representation, but also adds 1 to the flipped number
    - this works.

Binary Subtraction

- Actually the same as addition, just apply one 2s complement representation to any negative number and then add as usual using the adder circuits.

Overflow

- With unsigned integers, this happens when an operation can't be carried out with a certain amount of bits, and this can be detected fairly easily.
- With signed integers this is detected when adding two positive integers has a negative result, or vice versa

Shift and Rotation

- A **Logical Shift** is used to shift bit values, and it's exactly what it sounds like; the values are just shifted to the right or left.
    - i.e. a right logical shift of 1101 would be 0110
- An **Arithmetic Shift** is identical, but a right shift preserves the signatory leftmost bit.
    - it's used to multiply or divide by powers of two.
        - left to multiple, right to divide
    - it can also be used to multiply by arbitrary numbers, by first breaking said number into powers of two
        - i.e. to multiply by 13; break it down into 8 + 4 + 1 or 2^(3) + 2^(2) + 2^(1)
        - so shift left 3 times, then 2 times, then one more time.

Booth's Algorithm

- A fast method of multiplying integers.
    - `Q` - multiplier
    - `M` - multiplicand
    - `Q'` - stores low order bit
    - `A` - stores high order portion of result
- The number of step is equal to the number of bits in `M`
- For each step, compare `Q` to `Q'` and take steps based on the algorithm below, where the values are `Q,Q'`
    - 0,0 = Arithmetic shift
    - 0,1 = Add, Arithmetic shift
    - 1,0 = Subtract, Arithmetic shift
    - 1,1 = Arithmetic shift

A	**B**	**C**	**(B.C)**	**A + (B.C)**	**(A+B)**	**(A+C)**	**(A+B).(B+C)**
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1

1	1	1	1	1	1	1	1

Early Papers and Architectures:
John von Neumann's 1945 paper, "First Draft of a Report on the EDVAC", outlined an organization of logical elements, and Alan Turing's "Proposed Electronic Calculator for the Automatic Computing Engine" also referenced von Neumann's work. These papers are crucial for understanding the development of early computer architecture.