How to implement multibit shift in hardware?

Let us illustrate the process for a 4-bit number B = (b3, b2, b1, b0). You can extend the idea for 8-bit or 32-bit numbers in the future.

a) Consider logical right shift

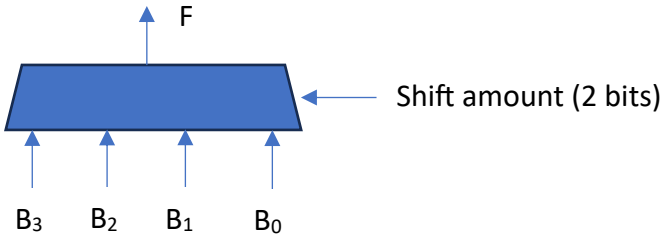
Shift by 0: B = (b3, b2, b1, b0) → B_0 = (b3, b2, b1, b0)

Shift by 1: B = (b3, b2, b1, b0) → B_1 = (0, b3, b2, b1)

Shift by 2: B = (b3, b2, b1, b0) → B_2 = (0, 0, b3, b2)

Shift by 3: B = (b3, b2, b1, b0) → B_3 = (0, 0, 0, b3)

Consider a multiplexer circuit as shown below.



Actually, there are four 4-to-1 multiplexers. To generate B_0, B_1, B_2 and B_3, you do not need any extra circuits; just some wiring. This circuit can generate shifted data by any desired amount.

b) Consider arithmetic right shift

Shift by 0: B = (b3, b2, b1, b0) → B_0 = (b3, b2, b1, b0)

Shift by 1: B = (b3, b2, b1, b0) → B_1 = (b3, b3, b2, b1)

Shift by 2: B = (b3, b2, b1, b0) → B_2 = (b3, b3, b3, b2)

Shift by 3: B = (b3, b2, b1, b0) → B_3 = (b3, b3, b3, b3)

This can again be implemented using multiplexers.

c) Consider shift left

Shift by 0: B = (b3, b2, b1, b0) → B_0 = (b3, b2, b1, b0)

Shift by 1: B = (b3, b2, b1, b0) → B_1 = (b2, b1, b0, 0)

Shift by 2: B = (b3, b2, b1, b0) → B_2 = (b1, b0, 0, 0)

Shift by 3: B = (b3, b2, b1, b0) → B_3 = (b0, 0, 0, 0)

This can again be implemented using multiplexers.

For 8-bit data, you will need eight 8-to-1 multiplexers. For 32-bit data, you will need 32 32-to-1 multiplexers.

How to count number of 1's in an 8-bit data word?

Let the data word be denoted as B = (b7, b6, b5, b4, b3, b2, b1, b0). We have to add all the bits of the number using a combinational circuit. We are not allowed to use a shift register of a counter with multiple clock pulses to do this.

We can do this using a tree of adders of progressively larger sizes – 1-bit adder (same as a full adder, can add three bits), 2-bit adder, and 3-bit adder. One possible schematic is shown below.