Chapter 3: Arithmetic for Computers

We already know,

numbers in computers $\Rightarrow$ binary numbers

$\downarrow$

integers

how about?

- fractions or, real numbers?
- arithmetic operations of numbers?
- what happen if an operation creates a number bigger than can be represented?
- how does hardware really multiply or divide numbers?

Addition: $(6)_{10} + (7)_{10}$

let 8-bit

(1)(1)(0) (Carries)

```
    0000 0110
  + 0000 0111
  -----------
  = 0000 1101  => 13
```

⊛ Subtraction: $(+7)_{10} - (+6)_{10}$

Let 8-bit,

$$
\begin{array}{rr}
 & 0000\ 0111 \\
- & 0000\ 0110 \\
\hline
= & 0000\ 0001
\end{array}
\Rightarrow (1)_{10}
$$

- to do this using addition, we will have to convert (+6) to it's twos complement representation of (−6)

$+6 \rightarrow 0000\ 0110$

$-6 \rightarrow 1111\ 1010$

Shortcut to two's complement.

find the first one from right, before that change nothing, don't change the first ~~l~~ rightmost 1 aswell, after that flip everything.

$$
\begin{array}{rr}
 & 0000\ 0111 \\
+ & 1111\ 1010 \\
\hline
= & 0000\ 0001
\end{array}
\Rightarrow (1)_{10}
$$

⊛ Overflow:

when result from an operation cannot be represented with available hardware, in this case a 32- bit word.

Adding two operands with different signs overflow cannot occur.

- as sum $\leq$ one of the operands.

similarly, during subtraction, if the signs of the operands are same, overflow cannot occur.

What really happen during overflow

↳ sign bit is set with the value rather that the sign.

So, if the sign bit is wrong, overflow occurs.

(+) two (+) numbers $\Rightarrow$ sign (−) $\Rightarrow$ overflow

(+) two (−) numbers $\Rightarrow$ sign (+) $\Rightarrow$ overflow

(−) (+) − (−) $\Rightarrow$ sign (−) $\Rightarrow$ overflow

(−) (−) − (+) $\Rightarrow$ sign (+) $\Rightarrow$ overflow