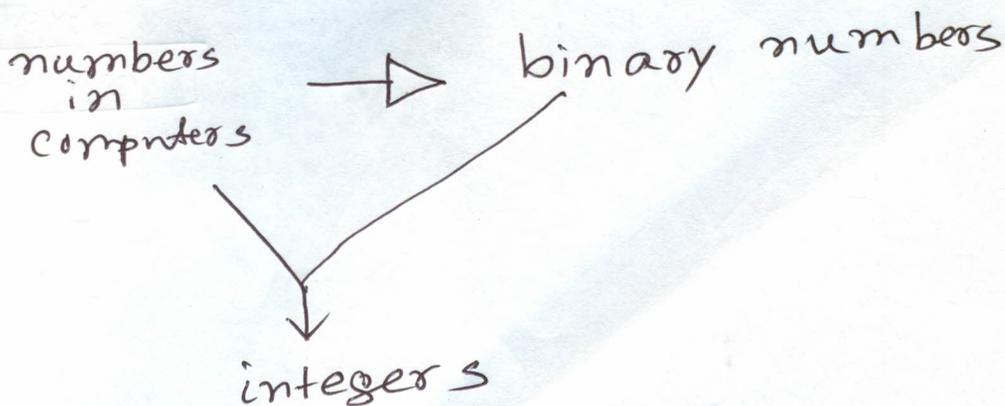


Chapter 3: Arithmetic for Computers

We already know,



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

-	(6)	(6)							
0	0	0	0	0	0	0	0	0	
+	0	0	0	0	0	1	1	1	
=	0	0	0	0	1	1	0	1	⇒ 13

(Carries)

(1)(1)(0)

(1)(1)(0)

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

Let 8-bit,

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

- to do this using addition, we will have to convert $(+6)$ to its two's complement representation of (-6)

$$\begin{array}{r} +6 \rightarrow 0000\ 0110 \\ -6 \rightarrow 1111\ 1010 \end{array}$$

Shortcut to two's complement,

find the first one from right, before that change nothing, the first ~~of~~ ^{don't} _{change} most 1 as well, after that flip everything.

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

* 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 happens during overflow

↳ sign bit is set with the value rather than 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