Hi 👋

🖐 I'm Etienne

⚛️ React trainer & dev & consultant

le reacteur.

@LeReacteurIO

A Binary adder written with TypeScript types only

Binary what ? 🤨

An adder is a digital circuit that performs addition of numbers.

Yes, we are just trying to add two numbers...

👆 But...

...Using TypeScript only !

Note: **TypeScript** actually mean **TypeScript type system** here.

Numbers

```
// JavaScript has numbers...
const num = 3;
```

```
// ... TypeScript too
type Num = 3;
let three: Num = 3; // ok
three = 4;
// ^ Type '4' is not assignable to type '3' - ts(2322)
```

The + operator

```
// JavaScript has a + operator...
const result = 3 + 4;
```

```
// ...but TypeScript does not !
type Result = 3 + 4;
//              ^ Error: ';' expected - ts(1005)
```

Our Goal 🥅

```
type Result = Add<120, 42>;
```

To be the same as:

```
type Result = 162;
```

Easy right ? 🙄

First try: Brut-force 💪

Yay \o/

```
);
    'Result' is declared but never used. ts(6196)
    type Result = 3
type Result = Add<1, 2>;
// It works
```

But...

To add numbers from 0 to X

We need to register X^2 cases

For numbers up to **100**...

...that's **10 000** lines

We can do better !

Binary to the rescue !

Processor don't have a + operator either !

they use electric flow and logic gates like **OR**, **AND** and **XOR**

TypeScript has logic using **ternary** and **extends**

We can do the same !

The plan 🗺️

1. Convert decimal type to a binary representation

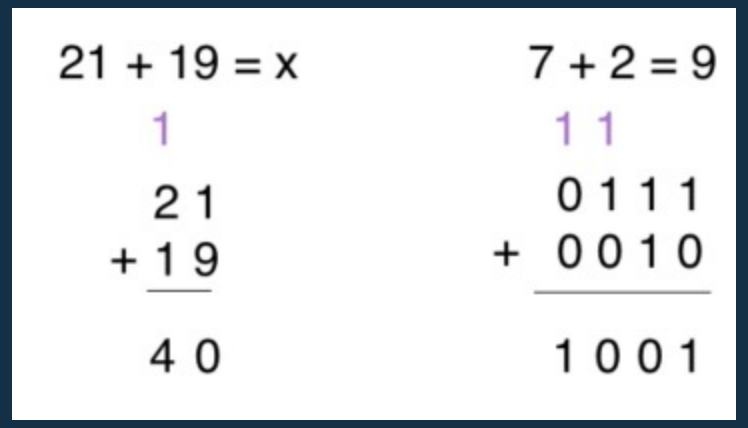
2. Use logic to compute the addition

3. Convert back to decimal type

Binary representation 🤔

```
// we can use Tuple to replesent a binary value
type Bit = 0 | 1;
type Byte = [Bit, Bit, Bit, Bit];
```

Binary addition



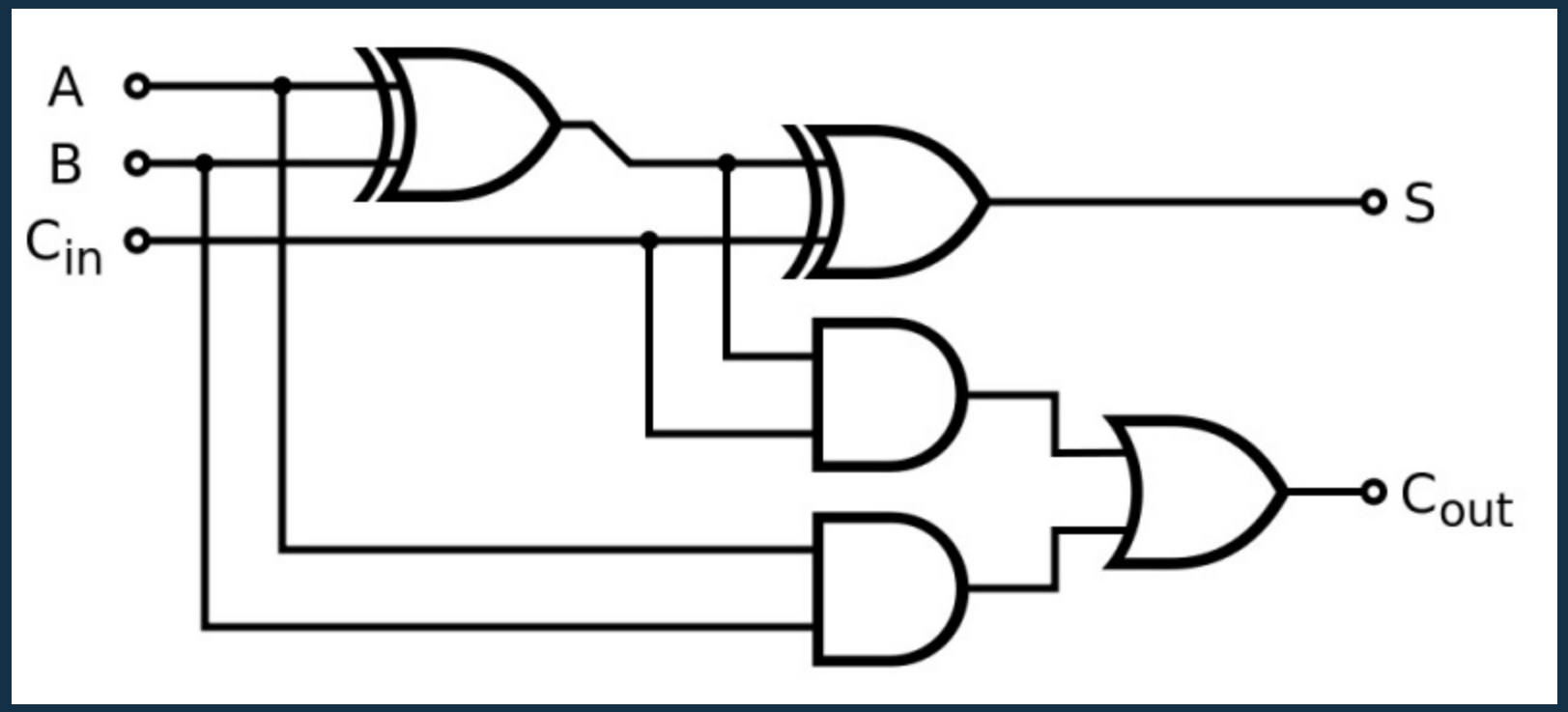
For each column we take

1. The digit from the first number
2. The digit from the second number
3. The carry from the previous column

...and we compute:

1. The sum
2. The carry of the next column

Sum & Carry with Logic



Logic gates

```
import { Bit } from "./types";
```

Sum & Carry

Binary Adder

```
import { Byte } from "./types";
```

Yay \o/

```
'Result' is declared but never used. ts(6196)
type Result = [1, 0, 0, 1]
type Result = AddBinary<[0, 1, 1, 1], [0, 0, 1, 0]>;
```

Convert to binary

```
// prettier-ignore
```

Don't write borring stuff !

```
// prettier-ignore
const range = num => Array(num).fill(null).map((v, i) => i);
const split = arr => {
  if (arr.length === 2) {
    return arr;
  }
  return [split(arr.slice(0, arr.length / 2)), split(arr.slice(-arr.length / 2))];
};
const result = range(Math.pow(2, 4));
const splitted = split(result);
console.log(JSON.stringify(splitted));

// [[[[0,1],[2,3]],[[4,5],[6,7]]],[[[8,9],[10,11]],[[12,13],[14,15]]]]
```

Convert to decimal 🙃

```
import { Byte } from "./types";
import { Decimal, ToBinary } from "./05-to-bin";

// prettier-ignore
export type ToDecimal<T extends Byte | "overflow"> = ({
  [K in Decimal]: ToBinary<K> extends T ? K : never
})[Decimal];
```

Mixing everything together

```
import { Decimal, ToBinary } from "./05-to-bin";
import { ToDecimal } from "./07-to-deci";
import { AddBinary } from "./04-binary-adder";

export type Add<A extends Decimal, B extends Decimal> = ToDecimal<
  AddBinary<ToBinary<A>, ToBinary<B>>
>;

type Result = Add<7, 2>;
```

Yay \o/

```
'Result' is declared but never used. ts(6196)
type Result = 9
```

```
// Re
// Tr                                                    to
type Result = Add<7, 2>;
```

Full code

```
type Bit = 0 | 1;

type Byte = [Bit, Bit, Bit, Bit];

type DecimalTree = [
  [[[0, 1], [2, 3]], [[4, 5], [6, 7]]],
  [[[8, 9], [10, 11]], [[12, 13], [14, 15]]]
];

type Decimal = DecimalTree[any][any][any][any];

type ToBinary<T extends Decimal> = [
  T extends DecimalTree[0][any][any][any] ? 0 : 1,
  T extends DecimalTree[any][0][any][any] ? 0 : 1,
  T extends DecimalTree[any][any][0][any] ? 0 : 1,
  T extends DecimalTree[any][any][any][0] ? 0 : 1
];

export type ToDecimal<T extends Byte | "overflow"> = ({
  [K in Decimal]: ToBinary<K> extends T ? K : never
})[Decimal];

type And<A extends Bit, B extends Bit> = B extends 1 ? (A extends 1 ? 1 : 0) : 0;

type Or<A extends Bit, B extends Bit> = B extends 0 ? (A extends 0 ? 0 : 1) : 1;

type Xor<A extends Bit, B extends Bit> = A extends 0
  ? (B extends 0 ? 0 : 1)
```

Now let's scale up to 8 bit !

```
// prettier-ignore
type DecimalTree = [
  [[[[[0, 1], [2, 3]], [[4, 5], [6, 7]]], [[[8, 9], [10, 11]], [[12, 13], [14, 15]]]], [[[16, 17],
  [18, 19], [20, 21], [22, 23], [24, 25], [26, 27], [28, 29], [30, 31], [32, 33],
  [34, 35], [36, 37], [38, 39], [40, 41], [42, 43], [44, 45], [46, 47], [48, 49], [50, 51],
  [52, 53], [54, 55], [56, 57], [58, 59], [60, 61], [62, 63], [64, 65],
  [66, 67], [68, 69], [70, 71], [72, 73], [74, 75], [76, 77], [78, 79], [80, 81],
  [82, 83], [84, 85], [86, 87], [88, 89], [90, 91], [92, 93], [94, 95], [96, 97],
  [98, 99], [100, 101], [102, 103], [104, 105], [106, 107], [108, 109], [110, 111],
  [[[112, 113], [114, 115]], [[116, 117], [118, 119]]], [[[[120, 121], [122, 123]], [[124, 125],
  [126, 127]]], [[[[128, 129], [130, 131], [132, 133], [134, 135]], [[136, 137],
  [138, 139]], [[140, 141], [142, 143]]], [[[[144, 145], [146, 147]], [[148, 149], [150, 151]]],
  [[152, 153], [154, 155]], [[156, 157], [158, 159]], [[160, 161], [162, 163]], [164, 165],
  [166, 167]]], [[[[168, 169], [170, 171]], [[172, 173], [174, 175]]], [[[176, 177], [178, 179]],
  [180, 181], [182, 183]], [[184, 185], [186, 187], [188, 189], [190, 191]], [[[192, 193],
  [194, 195]], [196, 197], [198, 199]], [[200, 201], [202, 203]], [204, 205], [206, 207]]],
  [[[208, 209], [210, 211]], [[212, 213], [214, 215], [216, 217], [218, 219]], [[220, 221],
  [222, 223]]]], [[[[224, 225], [226, 227]], [228, 229], [230, 231]], [[232, 233], [234, 235],
  [236, 237], [238, 239]]], [[[240, 241], [242, 243], [244, 245], [246, 247]], [[248, 249],
  [250, 251]], [252, 253], [254, 255]]]];

// prettier-ignore
type Decimal = DecimalTree[any][any][any][any][any][any][any][any];

// prettier-ignore
type ToBinary<T extends Decimal> = [
  T extends DecimalTree[0][any][any][any][any][any][any][any] ? 0 : 1,
  T extends DecimalTree[any][0][any][any][any][any][any][any] ? 0 : 1,
  T extends DecimalTree[any][any][0][any][any][any][any][any] ? 0 : 1.
```

Yay \o/

```
'Result' is declared but never used. ts(6196)
// Re
// Tr type Result = 162
type Result = Add<120, 42>;
```

TS doesn't like that 🤭

```
Type instantiation is excessively deep and possibly infinite. ts(2589)
Quick Fix...    Peek Problem
  AddBinary<ToBinary<A>, ToBinary<B>>
>;
```

Can we do better ?

Yes ! 10 bit 🤩

```
type Result = 871
'Result' is declared but never used. ts(6196)
Quick Fix...
// Re
// Tr                                                    see its type
type Result = Add<553, 318>;
```

takes ~3s to compute the type ⏳

11 bit ? 🤯

Yep 🤪

```
type Result = 1797
```

```
type Result = Add<1200, 597>;
```

More than 30s to compute types 🥵

Only works with TS 3.3 😬

Is this useful ? 🤔

Nope ¯_(ツ)_/¯

https://ts-binary-adder.etienne.tech

Questions ?

PS: I'm on twitter @Etienne_dot_js