6 コンピュータの $1+1$

旅の寄り道ついでにコンピュータが行う計算について述べておこう。繰り返しになるけれど、コンピュータが行う基本的な論理演算を示す。

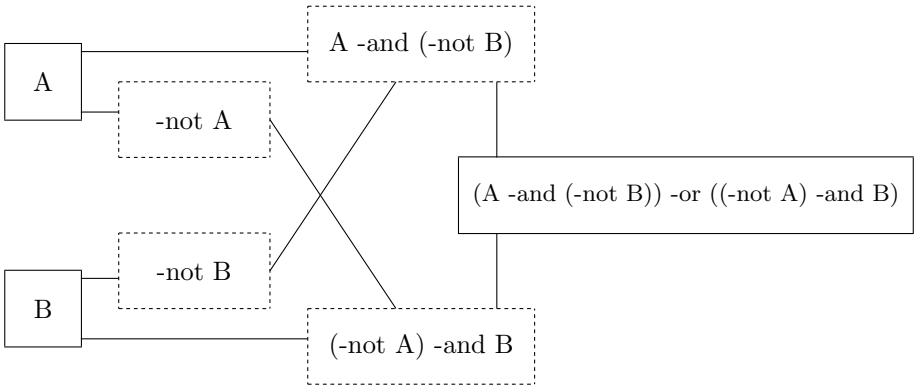
-and	0	1
0	0	0
1	0	1

-or	0	1
0	0	1
1	1	1

桁の繰り上がりを無視することにして、`1 -or 1` が `0` になれば`-or` 演算子は $+$ の代用にできる。桁の繰り上がりは、`0, 1` を格納する入れ物をつなげれば済むので、計算の本質とは関係ないからだ。

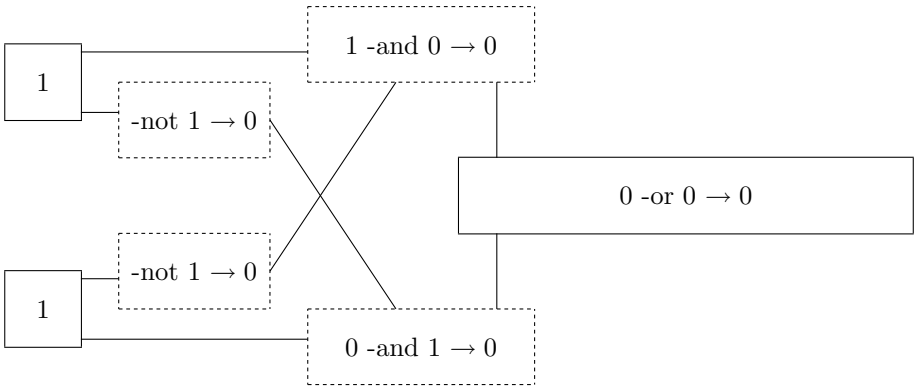
どうしたら、そんなことが実現できるのだろう。その考察のために演算子をもうひとつ紹介しておく。`-not` 演算子である。`-not` 演算を日常の言葉で言えば**否定**である。`-not A` とは A の否定を意味する。否定と言っても、いまは 0 と 1 しか考えてないので、`-not 1 = 0, -not 0 = 1` である。

そこで唐突ながら次のような仕組みを考えてみよう。



図は、左側の A と B に 0 か 1 が与えられたとき、いくつかの論理演算をして最終的に右端の結果が出ることを表している。たとえば A に与えられた値からは (-not A) が生成され、それと B の間で論理演算`-and` が処理される。同時に B に与えられた値からも (-not B) が生成され、それと A の間で論理演算`-and` が処理され、それら 2 つの間で`-or` 演算が施されて、ひとつの結果が得られるのである。

具体例として、A に 1、B に 1 が入力された場合を、順を追って確かめよう。



確かに私たちが望んでいる結果になった。これ以外について

$$0 \text{ と } 0 \to 0, \quad 0 \text{ と } 1 \to 1, \quad 1 \text{ と } 0 \to 1$$

であることは、君たちに確認してもらうことにして先へ行こう。

実際にコンピュータはこの仕組みで計算をしている。-not と-and の処理を組み合わせているので NAND 型（Negative AND 型）と呼ばれる。もっとも、仕組みがそっくり回路に置き換わるわけではないので、図がそのまま NAND 型の回路になるのではないけれど。

実は、**PowerShell** には NAND 型の回路と同じことをする演算子が備わっている。`-xor` 演算子である。もう一度、図を見てもらいたい。処理が 1 カ所でクロスしてから`-or` 演算をしているので、`-xor` は理にかなった名称だろう[*1]。では、実際の`-xor` 演算子の振る舞いを **PowerShell** で見ておこう。間違いないでしょ。

[ps script]

```
PS C:\Users\Yours > @"
>> 0 -xor 0
>> 0 -xor 1
>> 1 -xor 0
>> 1 -xor 1
>> "@ > test.ps1
>>
PS C:\Users\Yours > ./test
>> False
>> True
>> True
>> False
```

[*1] 実際は exclusive or（排他的論理和）の略。

しかしながら`-xor`演算子で $+$ の代用はできても、この体系では $1, 2, 3, \ldots$ と続く数を構成できていない。$\emptyset$ と集合の概念からなる数の体系は、$1, 2, 3, \ldots$ と続く数を構成できたものの、$+$ にあたる演算には不具合が生じてしまった。一方で、いま考察した `False`, `True` からなる数の体系は、$+$ にあたる演算を導入できるものの、私たちが使うような数を構成できない。帯に短したすきに長し、のように中途半端である。数を構成し、そこに $+$ 演算を導入するという道のりは、思ったより険しい道のりのようである。