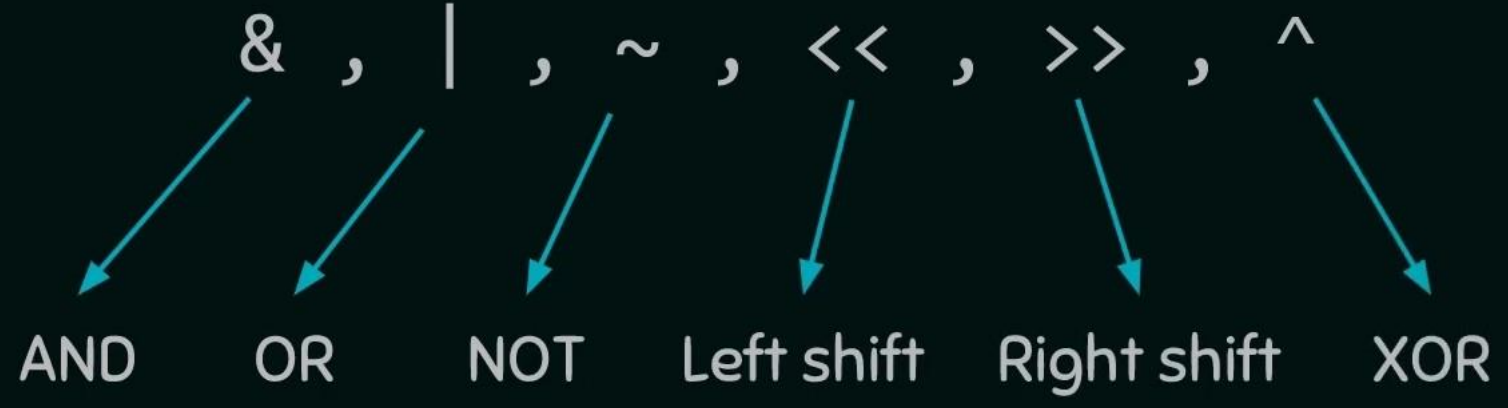
Introduction to bitwise operators

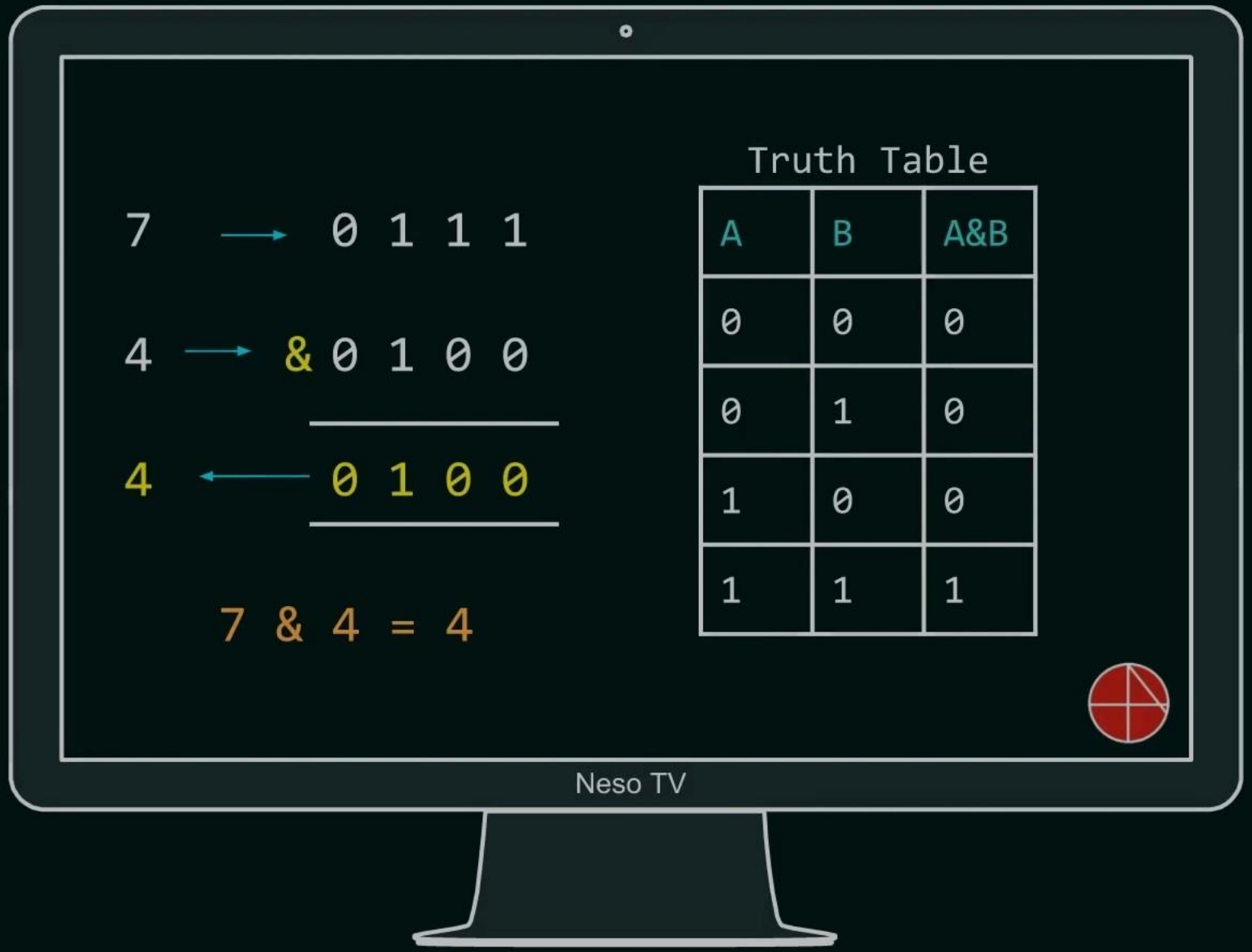
As name suggests – it does bitwise manipulation.

6 cool operators



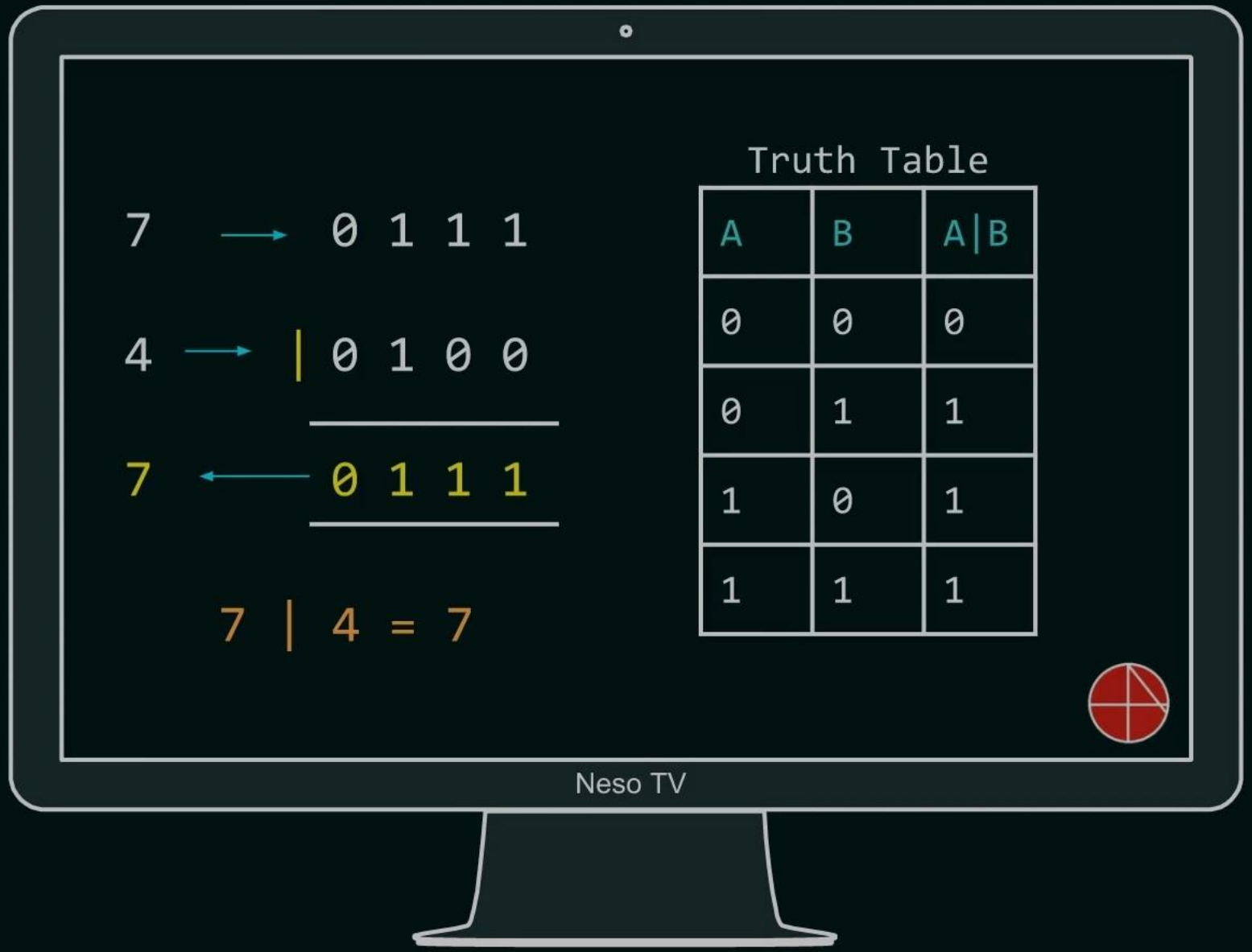
Bitwise AND (&) operator

- It takes two bits at a time and perform AND operation.
- AND (&) is binary operator. It takes two numbers and perform bitwise AND.
- Result of AND is 1 when both bits are 1



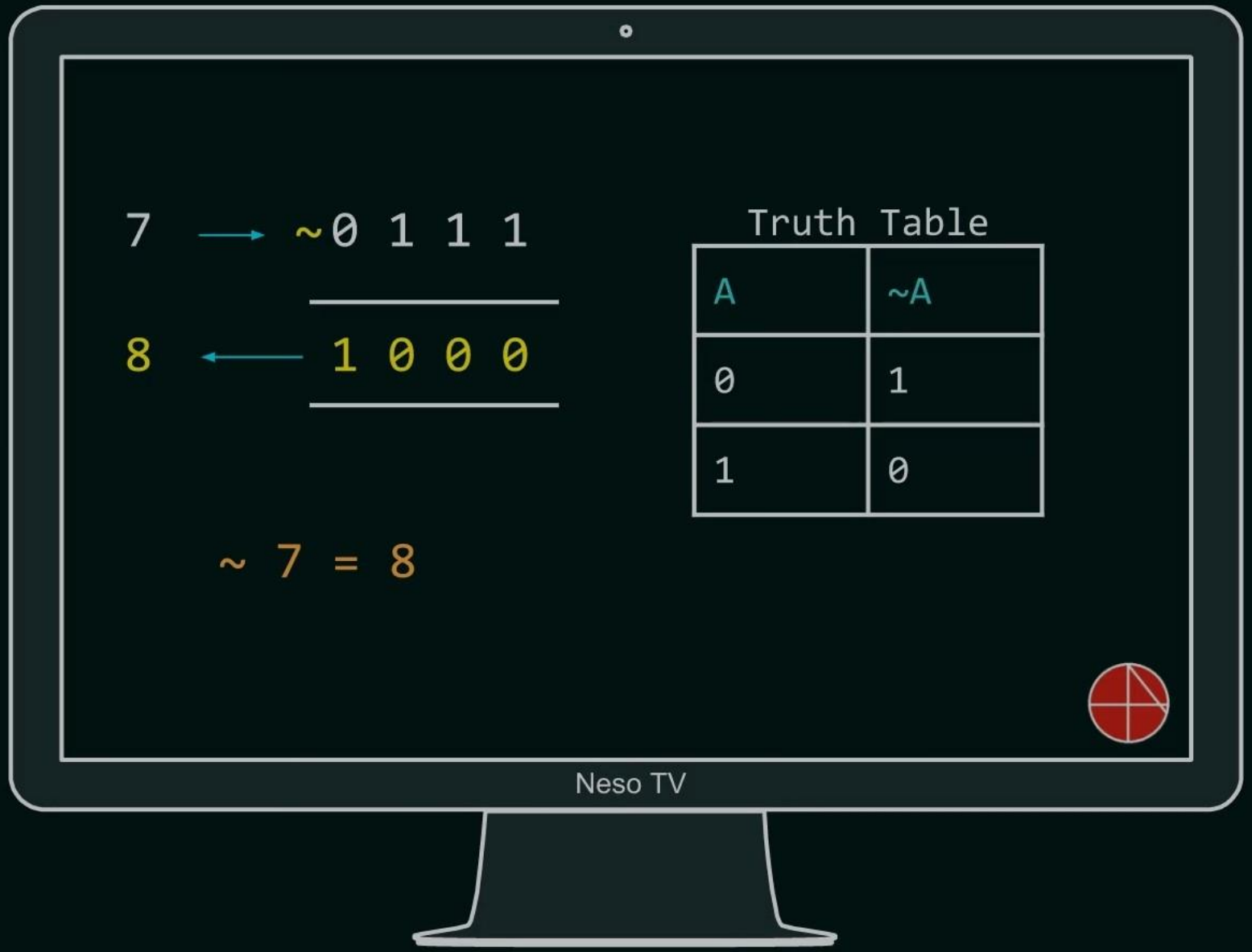
Bitwise OR (|) operator

- It takes two bits at a time and perform OR operation.
- OR (|) is binary operator. It takes two numbers and perform bitwise OR.
- Result of OR is 0 when both bits are 0



Bitwise NOT (~) operator

- NOT is a unary operator
- It s job is to complement each bit one by one.
- Result of NOT is 0 when bit is 1 and 1 when bit is 0



Difference between bitwise and logical operators

```c
#include <stdio.h>

int main() {
    char x = 1, y = 2; //x = 1(0000 0001), y = 2(0000 0010)
    if(x&y)                                  //1&2 = 0(0000 0000)
        printf("Result of x&y is 1");
    if(x&&y)                                 //1&&2 = TRUE && TRUE = TRUE = 1
        printf("Result of x&&y is 1");

    return 0;
}
```

Output:

```
Result of x&&y is 1
```

Left shift operator

```
First operand << Second operand
```

Whose bits get left shifted

Decides the number of places to shift the bits

Important points

1. When bits are shifted left then trailing positions are filled with zeros.


```c
#include <stdio.h>

int main() {
    char var = 3;  //Note: 3 in binary = 0000 0011
    printf("%d", var<<1);
    return 0;
}
```

How left shift works?

```
var << 1
```

Left shift by 1 position

```
var = 3

3 = 0000 0011

    0000 011_

    0000 0110   = 6
```

Trailing position filled with zero.

Important points

2 Left shifting is equivalent to multiplication by $2^{rightOperand}$

Example:

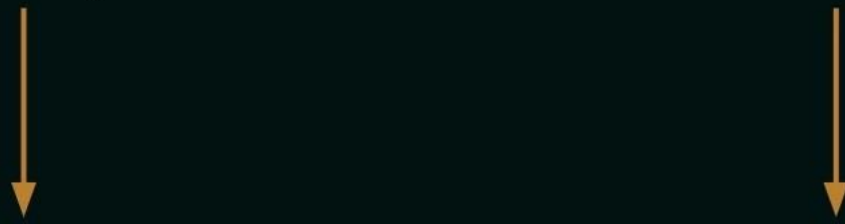
```
var = 3;

var << 1        Output: 6   [3 x 2^1]

var << 4        Output: 48  [3 x 2^4]
```

Right shift operator

```
First operand >> Second operand
```



Whose bits get right shifted

Decides number of places to shift the bits

Important points

1. When bits are shifted right then leading positions are filled with zeros.

```c
#include <stdio.h>

int main() {
    char var = 3;  //Note: 3 in binary = 0000 0011
    printf("%d", var>>1);
    return 0;
}
```

How right shift works?

```
var >> 1
```

Right shift by 1 position

```
var = 3

3 = 0000 0011

    _000 0001

    0000 0001   = 1
```

Leading position filled with zero.

Important points

2 Right shifting is equivalent to division by

$$2^{rightOperand}$$

Example:

```
var = 3;

var >> 1        Output: 1   [3 / 2^1]


var = 32;

var >> 4        Output: 2   [32 / 2^4]
```

Bitwise XOR operator

Better way to understand XOR operator

X - OR

Inclusive OR

- Either A is 1 or B is 1 or Both are 1, then the output is 1.
- Including BOTH

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive OR

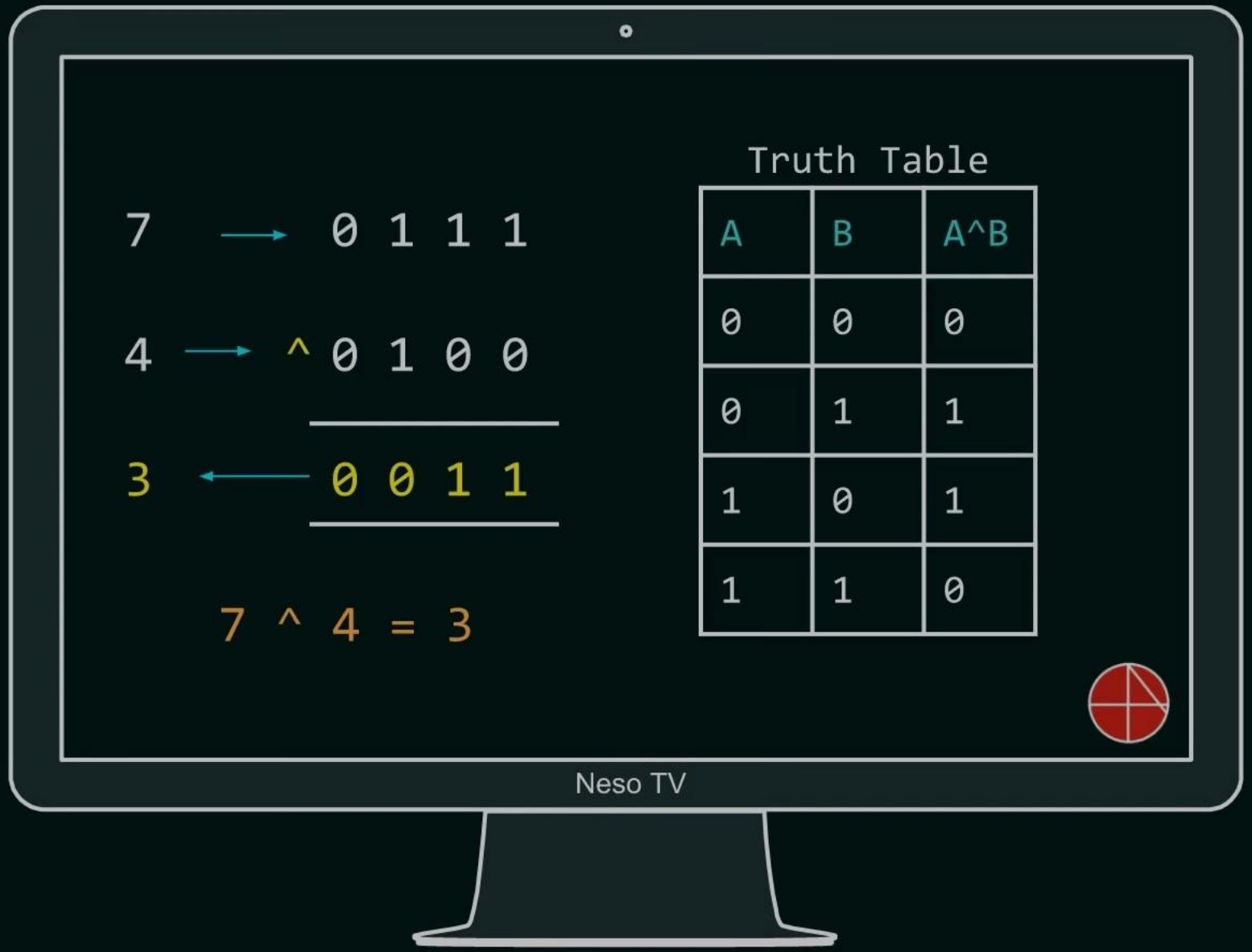
- Either A is 1 or B is 1 then the output is 1 but when both A and B are 1 then output is 0.
- Excluding BOTH

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Only difference

An example

- Bitwise XOR (^) is binary operator. It takes two numbers and perform bitwise XOR.
- Result of XOR is 1 when two bits are different otherwise the result is 0.



Homework problem

What is the output of the following program snippet?

```c
#include <stdio.h>

int main() {
    int a = 4, b = 3;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;

    printf("After XOR, a = %d and b = %d", a, b);
    return 0;
}
```

You can post your answer in the comment section below