Bitwise Operators and Bitboards

A brief introduction

Maxim Rebguns

November 2023

About Me

Maxim Rebguns

- Computer scientist
  - Favorite languages: C and Python
  - Worked with different algorithms, embedded systems, web development, game development
- Avid Linux user
- Theater kid

Binary data

Binary as a number system

- Just like with decimal numbers, but instead of 0–9, we only have 0 and 1.
- We use place values, just like with base-10 (decimal) numbers.

Representing data in binary

- Integers: place value, two's complement
- Real numbers: floating-point, fixed-point, logarithmic
- Characters: ASCII, UTF-8
- Pointers? Arrays? Structures? Classes?

Getting rid of the abstraction

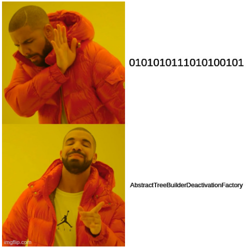

Figure 1: Average programmer

- It's all just bits under the hood

Bitwise operators

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems
- Cryptography

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems
- Cryptography
- Compression

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems
- Cryptography
- Compression
- Graphics

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems
- Cryptography
- Compression
- Graphics
- Speed

Purpose

Why would we need to manipulate individual bits?

- Representing true or false data
- Embedded systems
- Cryptography
- Compression
- Graphics
- Speed
- Certain data structures

Bitwise AND (binary)

A	B	?
0	0	0
0	1	0
1	0	0
1	1	1

Performs an AND operation on each bit:

```
  011010100
& 101001101
  ---------
  001000100
```

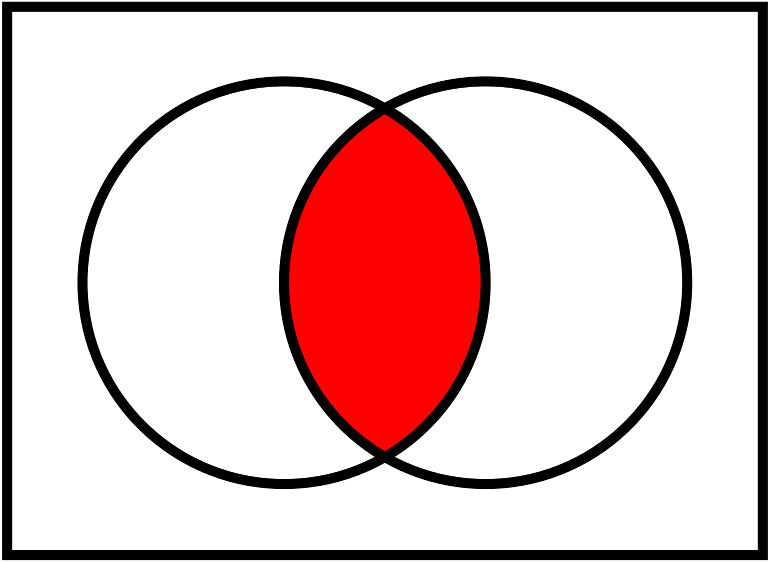

Figure 2: We are intersecting our two inputs

Bitwise OR (binary)

A	B	?
0	0	0
0	1	1
1	0	1
1	1	1

Performs an OR operation on each bit:

```
  011010100
| 101001101
  ---------
  111011101
```

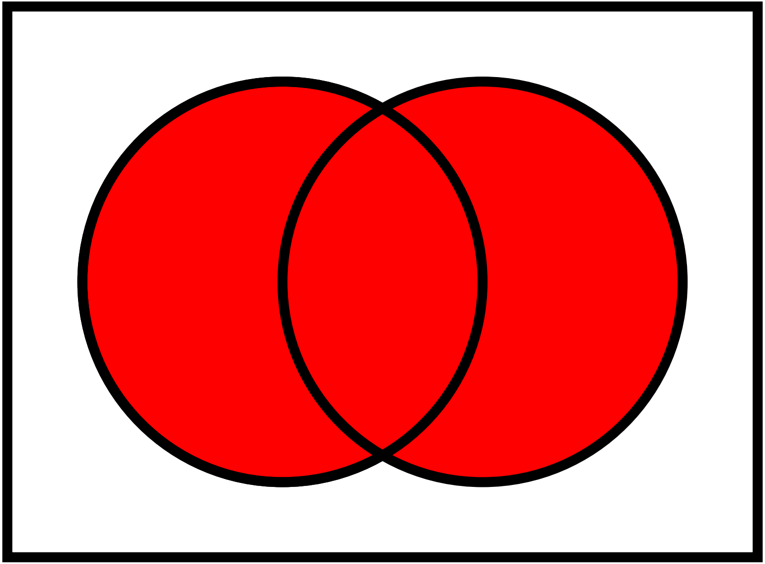

Figure 3: We are unioning our two inputs

Bitwise XOR (binary)

A	B	?
0	0	0
0	1	1
1	0	1
1	1	0

Performs an XOR operation on each bit:

```
  011010100
^ 101001101
  ---------
  110011001
```

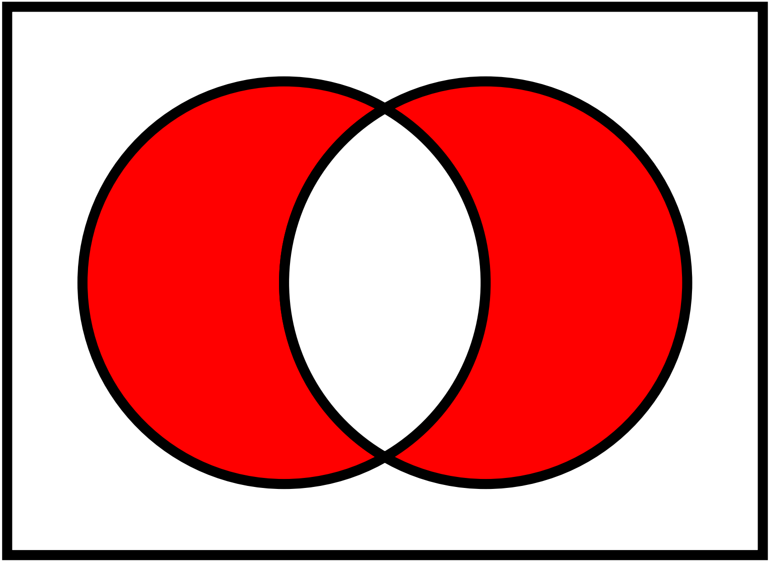

Figure 4: We are taking the symmetric difference of our two inputs

Bitwise NOT (unary)

A	?
0	1
1	0

Performs a NOT operation on each bit:

```
~ 011010100
  ---------
  100101011
```

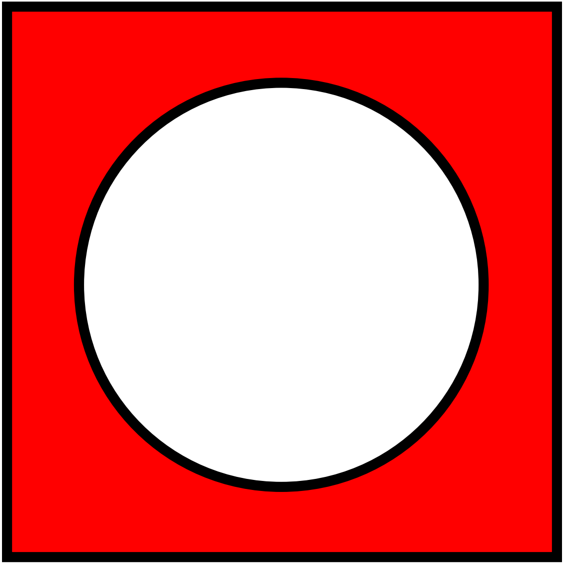

Figure 5: We are negating our one input

Left/Right Shift (binary)

Allows you to shift all bits of a number to the left or right by another number.

```
0010110 >> 10 becomes 0000101
0010110 << 10 becomes 1011000
```

- Note that 10 in binary means 2 in decimal.
- Typically we represent the shift amount in decimal for easier understanding.

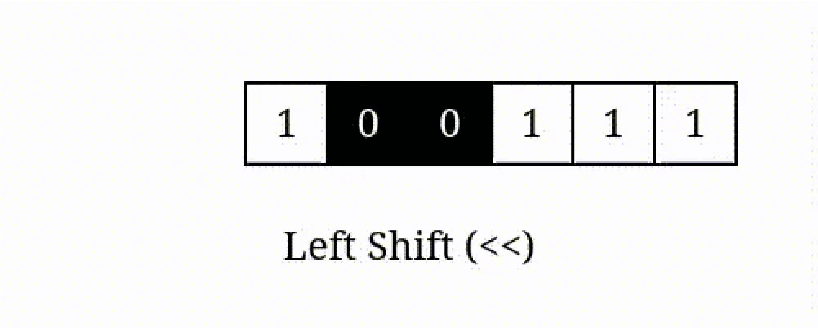

Figure 6: Left shift. A right shift is the same but in the other direction

Bitboards

Intro to bitboards

Let's look at an application of bitwise operators that is often used for representing grids in games: **bitboards**

- A way to represent a grid of binary numbers in a single integer.
- Highly compact.
- Allows for boolean operations using bitwise operators.
- Very useful for gridded board games.

The intuitive method

Goal: Create a grid representing the tic-tac-toe board.

o	x	x
	x	
o	o	

```
typedef enum { MOVE_EMPTY, MOVE_X, MOVE_O } move;
move board[][] = {
    {MOVE_O, MOVE_X, MOVE_X},
    {MOVE_EMPTY, MOVE_X, MOVE_EMPTY},
    {MOVE_O, MOVE_O, MOVE_EMPTY}
};
```

Issues with this method

- Each spot is an integer, which takes at *least* 16×9 bits!

Issues with this method

- Each spot is an integer, which takes at *least* 16×9 bits!
  - We could use a 255 bit `char`, but 252 of those bits would still be wasted.

Issues with this method

- Each spot is an integer, which takes at *least* 16×9 bits!
  - We could use a 255 bit `char`, but 252 of those bits would still be wasted.
- Searching for things in the array would be done by expensive loops.

Issues with this method

- Each spot is an integer, which takes at *least* 16×9 bits!
  - We could use a 255 bit `char`, but 252 of those bits would still be wasted.
- Searching for things in the array would be done by expensive loops.
- Representing possible wins is painful.

Issues with this method

- Each spot is an integer, which takes at *least* 16×9 bits!
  - We could use a 255 bit `char`, but 252 of those bits would still be wasted.
- Searching for things in the array would be done by expensive loops.
- Representing possible wins is painful.
- What if we were doing chess? How would we simulate the range of moves of pieces without contrived loops?

Enter bitboards

What if we represented the board as two binary numbers, one for each side?

1 o	2 x	3 x
4 x	5	6
7 o	8 o	9

```
typedef uint16_t board;

//                        123456789
board x_positions = 0b011100000;
board o_positions = 0b100000110;
```

Things you can do with bitboards

- Get an intuitive understanding: https://tearth.dev/bitboard-viewer/
- Get a bitboard representing all taken positions:

```
board taken_positions = x_positions | o_positions;
```

- Check if a player's move is valid:

```
board valid_positions = ~taken_positions;
bool is_valid = (move & valid_positions) != 0;
```

More bitboard tricks

- There are 8 ways to win in tic-tac-toe. You can represent these 8 board positions as bitboards, and then AND them with the player's positions to see if they won:

```
bool has_won = ((board & l_vert_win) == l_vert_win)
                || (board & m_vert_win) == m_vert_win)
                || ...);
```

- Bitwise operators allow you to manipulate bits efficiently, which is what makes this a great methods for complex games like chess.

Conclusion

- There are a truckload of ways to represent data with binary.
- Bitboards are one of them.
- Bitwise operators allow for the manipulation of individual bits of data.
- This is extremely fast and broadly applicable.

Thanks!

Credits

- Binary Representation [pdf]
- Wikipedia: Floating-pint arithmetic
- StackOverflow: Real world use cases of bitwise operators
- Wikipedia: Bitwise operation
- Wikipedia: Bitwise operators in C
- Chess Programming Wiki: Bitboards

About