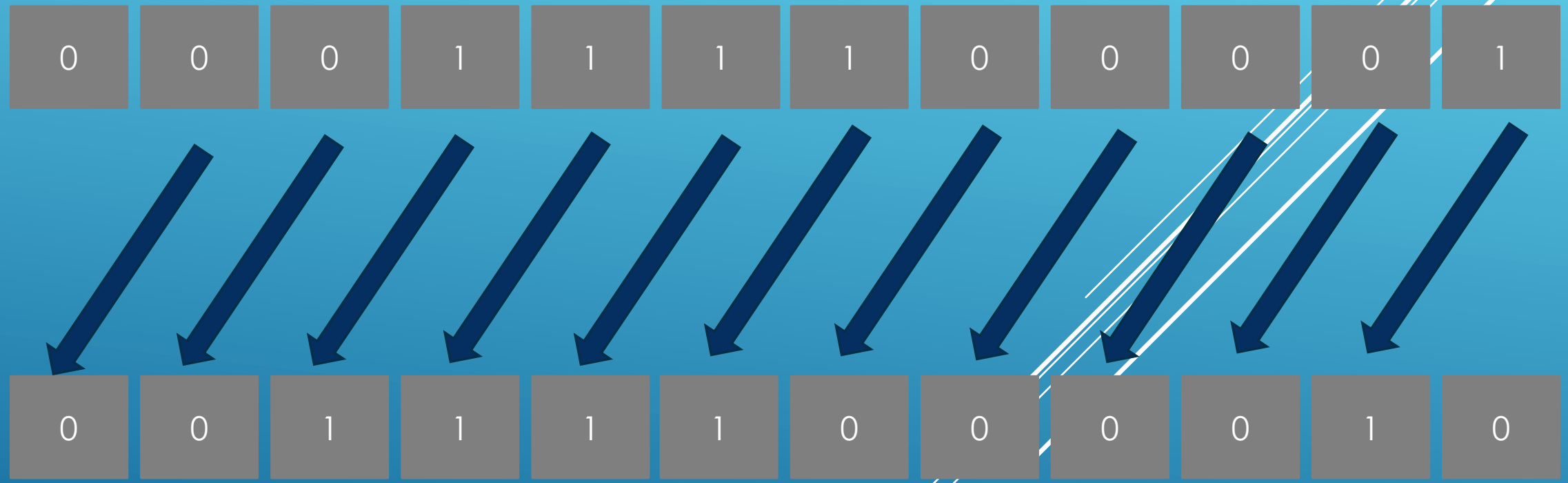
Slides

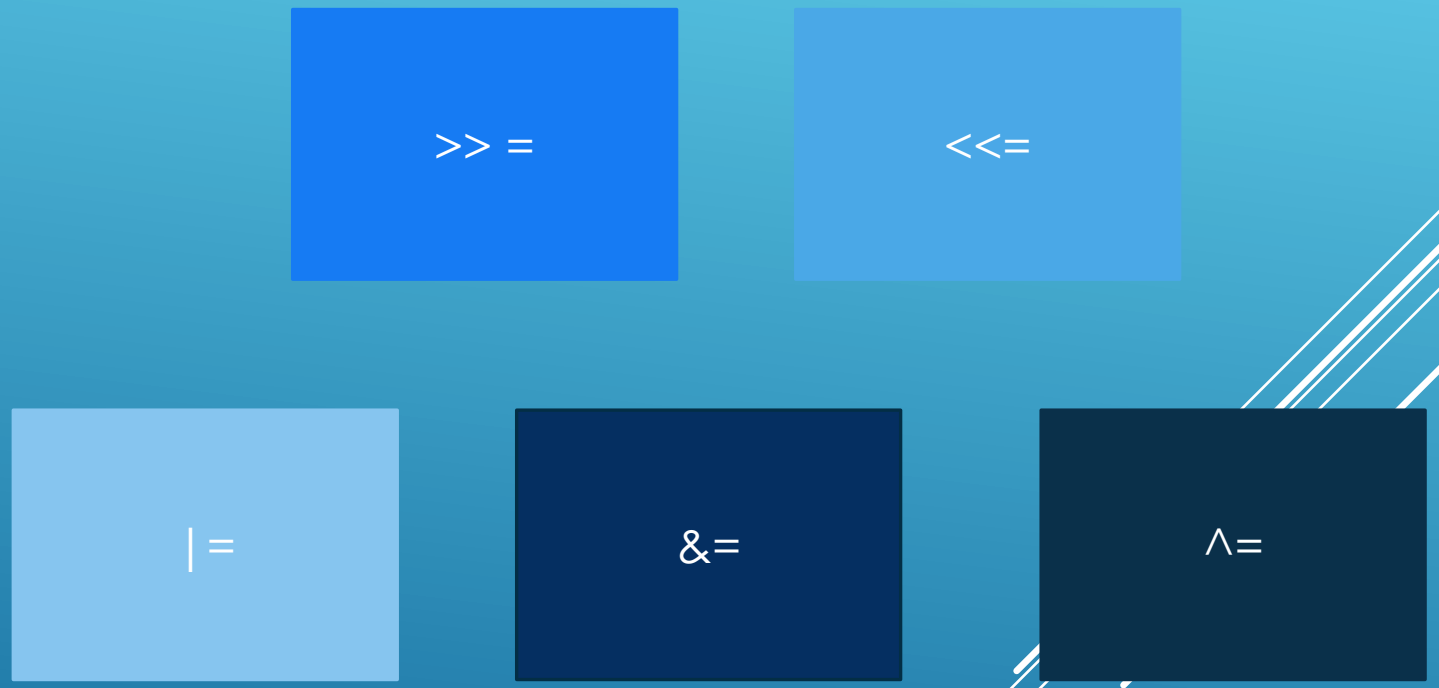
Section :Bitwise Operators

Slide intentionally left empty

Bitwise Operators : Introduction


0 0 0 1 1 1 1 0 0 0 0 1
0 0 1 1 1 1 0 0 0 0 1 0


AND
OR
XOR
NOT



Setting bit position 0

Slide intentionally left empty

Printing Integers in Binary

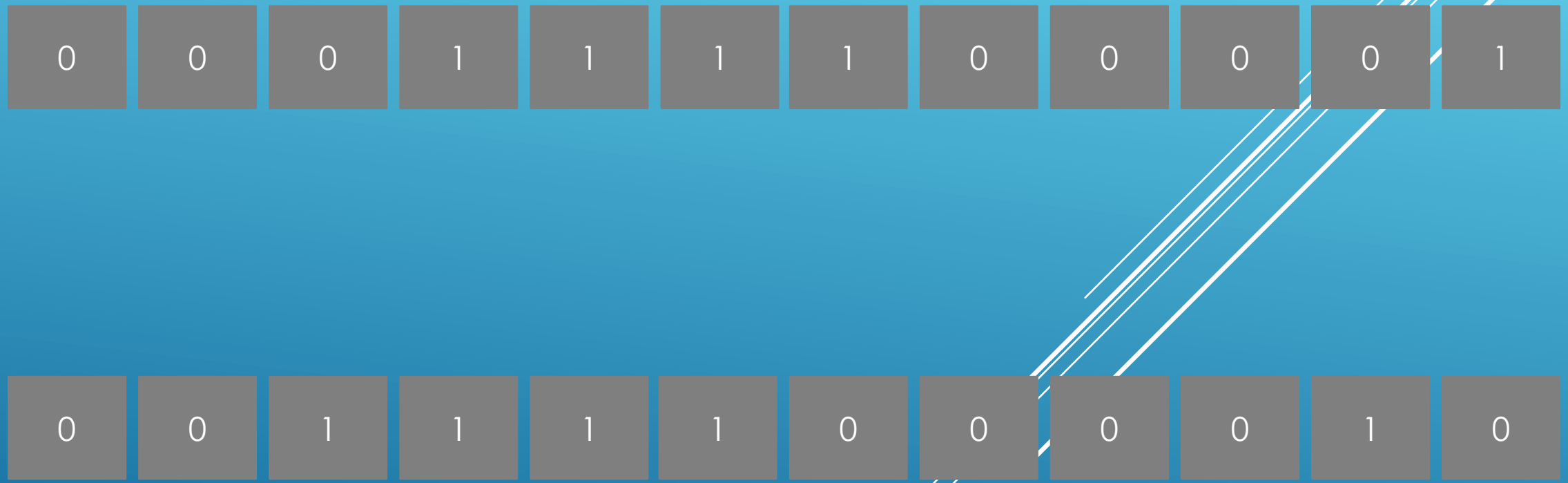
```cpp
#include <iostream>
#include <bitset>

int main(){

    unsigned short int data {0b11111110};
    std::cout << "data (dec) : " << std::dec << data << std::endl;
    std::cout << "data (hex) : " << std::hex << std::showbase << data << std::endl;
    std::cout << "data (bin) : " << std::bitset<16>(data) << std::endl;
}
```

Slide intentionally left empty

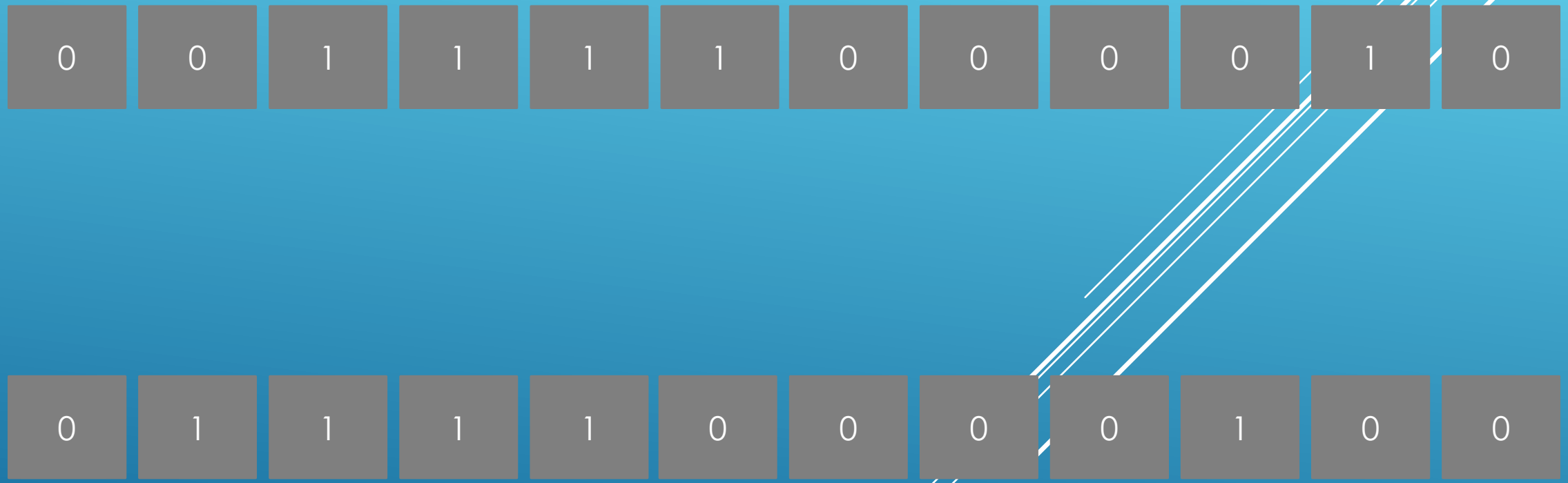
Shift Operators


0 0 0 1 1 1 1 0 0 0 0 1

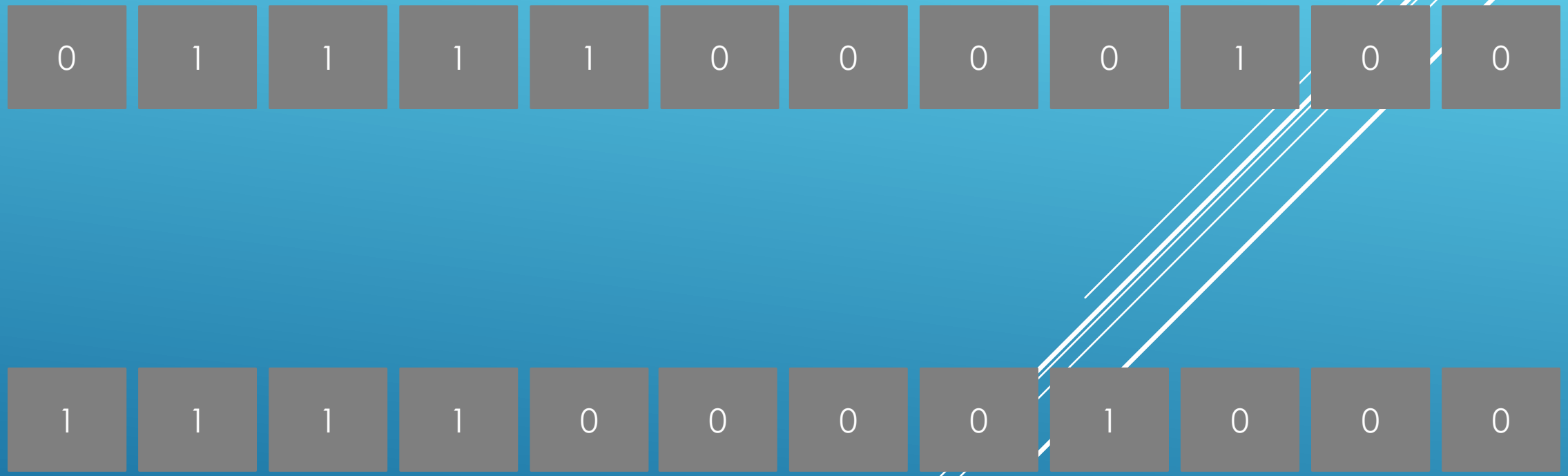
Shift left (<<) one bit position


0 0 0 1 1 1 1 0 0 0 0 1
0 0 1 1 1 1 0 0 0 0 1 0

Shift left (<<) one bit position

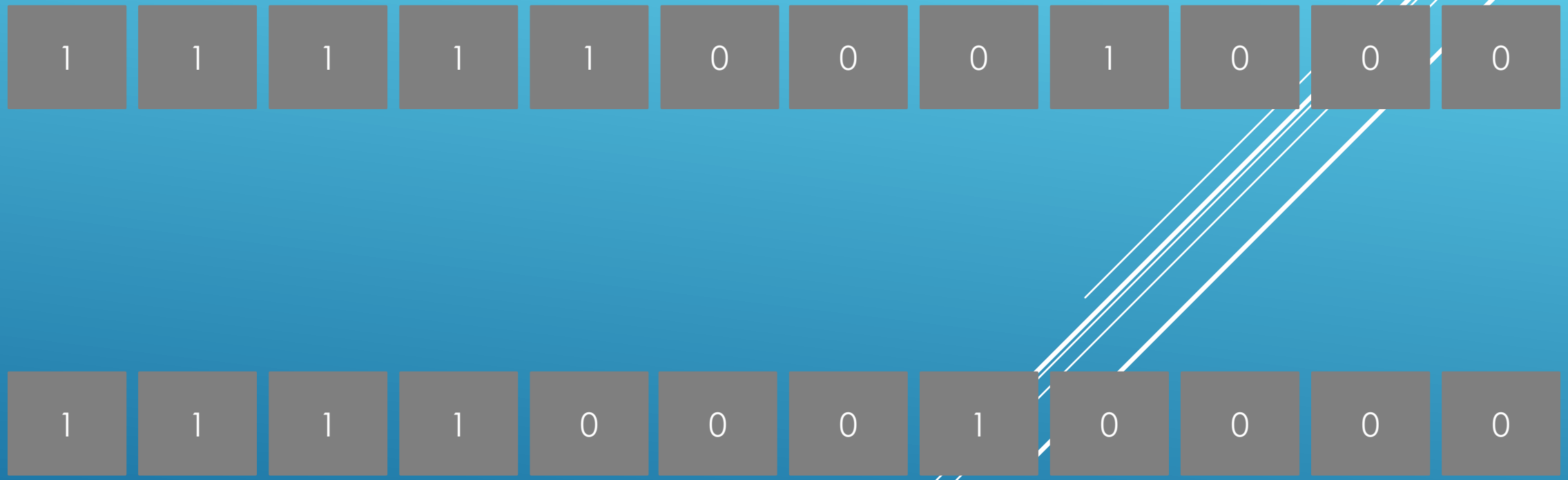

0 0 1 1 1 1 0 0 0 0 1 0
0 1 1 1 1 0 0 0 0 1 0 0

Shift left (<<) one bit position


0 1 1 1 1 0 0 0 0 1 0 0
1 1 1 1 0 0 0 0 1 0 0 0

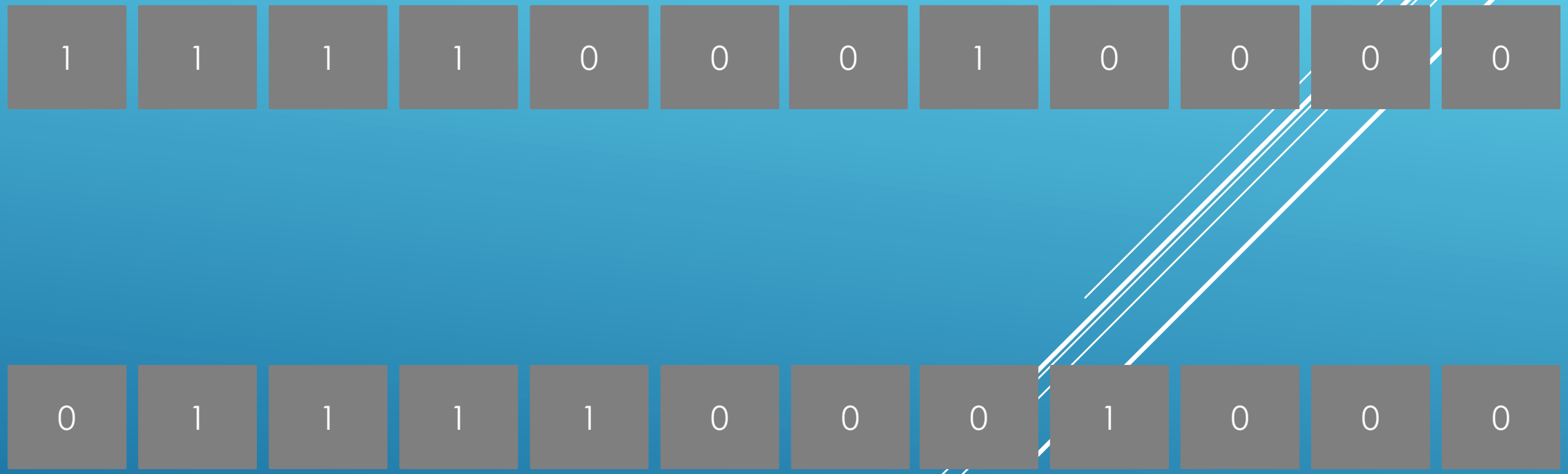
What happens when we shift left by 1 position

Shift left (<<) one bit position


1 1 1 1 1 0 0 0 1 0 0 0
1 1 1 1 0 0 0 1 0 0 0 0

Shift right (>>) and see if you get the 1 back

Shift right (>>) and see if you get the 1 back



If data is lost as a result of you shifting bits left(<<) or right(>>), you can't get the data back just by doing the reverse operation. You've just lost the data permanently!

Bit shifting is only supported for integral types like int, char,...

Bit shifting in C++

```
unsigned short int value {0xff0u};

std::cout << "value : " << std::bitset<16>(value)
<< ", dec : " << value << std::endl;

std::cout << std::endl;
std::cout << "Shifting right >>>>> " << std::endl;

//shift right 1 bit position
value = static_cast<unsigned short int>(value >> 1);
std::cout << "value : " << std::bitset<16>(value)
<< ", dec : " << value << " [After shift 1 bit position right] " << std::endl;
```

Shift left

```
value = static_cast<unsigned short int>(value << 1);
std::cout << "value : " << std::bitset<16>(value)
<< ", dec : " << value << " [After shift 1 bit position left] " << std::endl;
```

Shift several bits in one go

```
std::cout << std::endl;
std::cout << "shift left 4 bit positions:" << std::endl;
value = static_cast<unsigned short int>(value << 4);
std::cout << "value : " << std::bitset<16>(value)
<< ", dec : " << value << " [After shift 4 bit positions left] " << std::endl;
```

The golden rule of bit shifting

```
Shifting right divides by 2^n
Shifting left multiplies by 2^n
This rule breaks if you throw off 1's either to the right or the left.
```

Shift operators together with stream operators

```
std::cout << "value : " << (value_for_output >> 2) << std::endl;
```

Slide intentionally left empty

Bitwise Logical Operators

AND(&)

OR(|)

XOR(^)

NOT(~)

a	b	a&b	a | b	~a	a^b
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Bitwise AND

```
const int COLUMN_WIDTH {20};

unsigned char value1 {0x3}; // 0000 0011
unsigned char value2 {0x5}; // 0000 0101

std::cout << std::setw(COLUMN_WIDTH) << "value1 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(value1) << std::endl;

std::cout << std::setw(COLUMN_WIDTH) << "value2 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(value2) << std::endl;

//AND
std::cout << std::endl;
std::cout << "Bitwise AND :  " << std::endl;
std::cout << std::setw(COLUMN_WIDTH) << "value1 & value2 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(value1 & value2) << std::endl;
std::cout << std::endl;
```

Bitwise OR

```
const int COLUMN_WIDTH {20};

unsigned char value1 {0x3}; // 0000 0011
unsigned char value2 {0x5}; // 0000 0101

//OR
std::cout << std::endl;
std::cout << "Bitwise OR :   " << std::endl;
std::cout << std::setw(COLUMN_WIDTH) << "value1 | value2 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(value1 | value2) << std::endl;
std::cout << std::endl;
```

Bitwise NOT

```
const int COLUMN_WIDTH {20};

unsigned char value1 {0x3}; // 0000 0011
unsigned char value2 {0x5}; // 0000 0101

//NOT
std::cout << std::endl;
std::cout << "Bitwise NOT " << std::endl;

std::cout << std::setw(COLUMN_WIDTH) << "~value1 : "
    << std::setw(COLUMN_WIDTH) <<  std::bitset<8>(~value1) << std::endl;

std::cout << std::setw(COLUMN_WIDTH) << "~value2 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(~value2) << std::endl;

std::cout << std::setw(COLUMN_WIDTH) << "~01011001 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(~0b01011001) << std::endl;//Using bin literal

std::cout << std::setw(COLUMN_WIDTH) << "~01011001 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(~0x59) << std::endl;//Using hex literal
std::cout << std::endl;
```

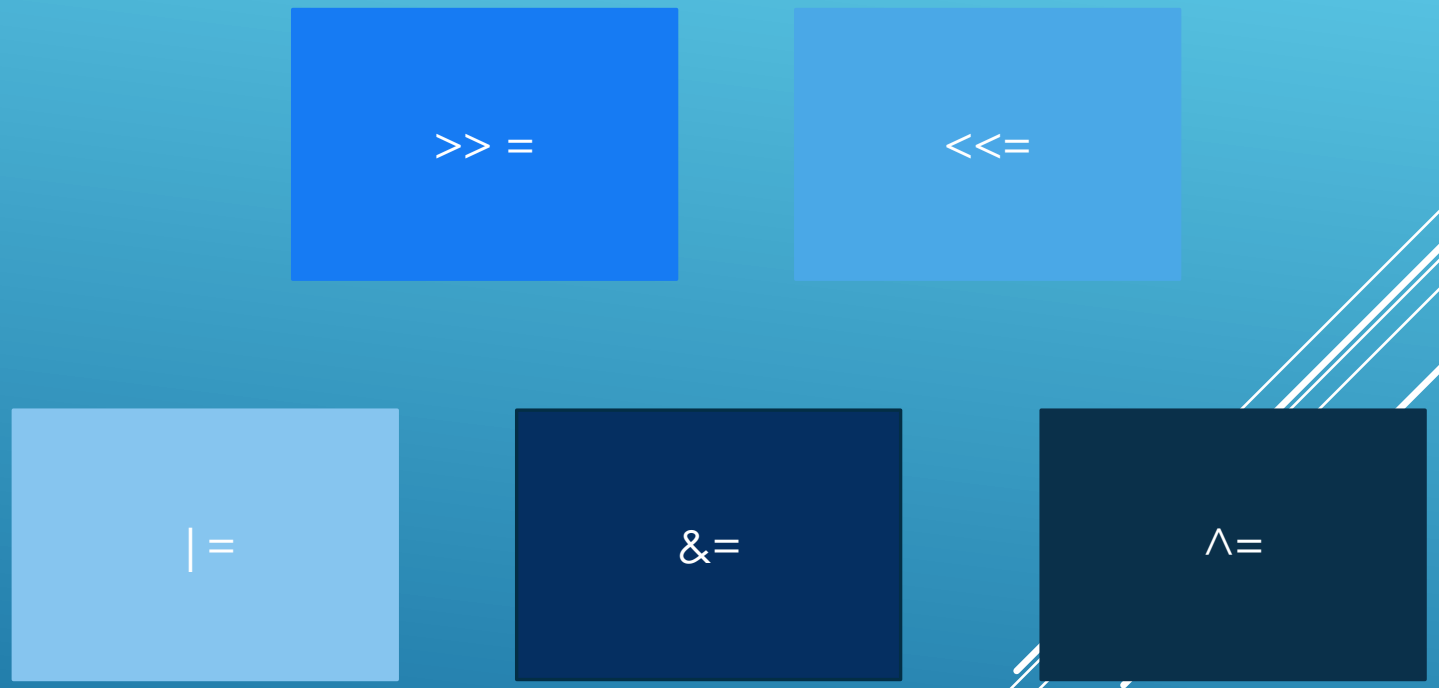
Bitwise XOR

```
const int COLUMN_WIDTH {20};

unsigned char value1 {0x3}; // 0000 0011
unsigned char value2 {0x5}; // 0000 0101

//XOR
std::cout << std::endl;
std::cout << "Bitwise XOR :  " << std::endl;
std::cout << std::setw(COLUMN_WIDTH) << "value1 ^ value2 : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(value1 ^ value2) << std::endl;
std::cout << std::endl;
```

Slide intentionally left empty

Compound Bitwise Assignment Operators

They work on the variable and put the result back in the same variable



Compound <<

```
const int COLUMN_WIDTH {20};

std::cout << std::endl;
std::cout << "Compound bitwise assignment operators" << std::endl;

unsigned char sandbox_var{0b00110100};

//Print out initial value
std::cout << std::endl;
std::cout << "Initial value :  " << std::endl;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;

//Compound left shift
std::cout << std::endl;
std::cout << "Shift left 2 bit positions in place :  " << std::endl;
sandbox_var <<= 2;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;
```

Compound >>

```
//Compound right shift
std::cout << std::endl;
std::cout << "Shift right 4 bit positions in place :  " << std::endl;
sandbox_var >>= 4;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;
```

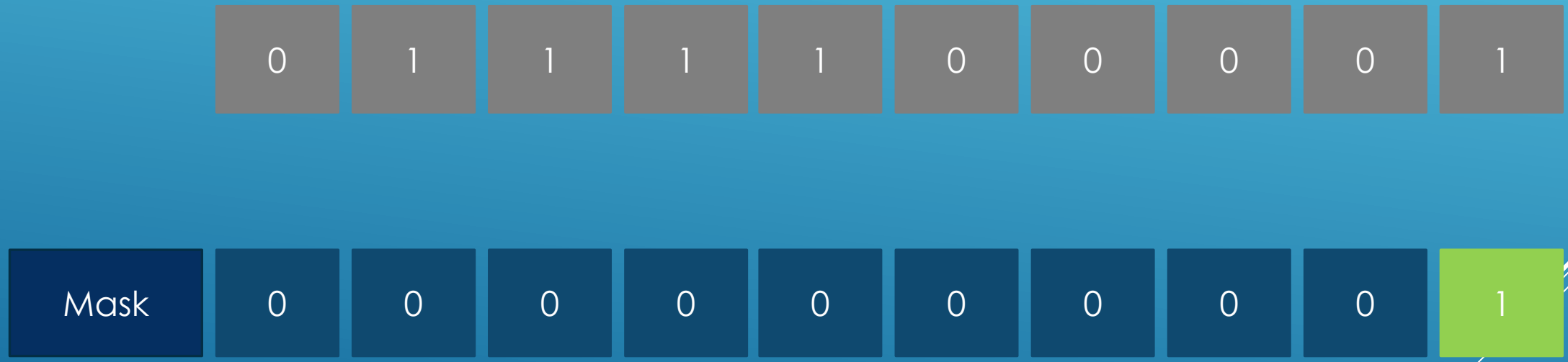
Compound Bitwise OR,AND and XOR

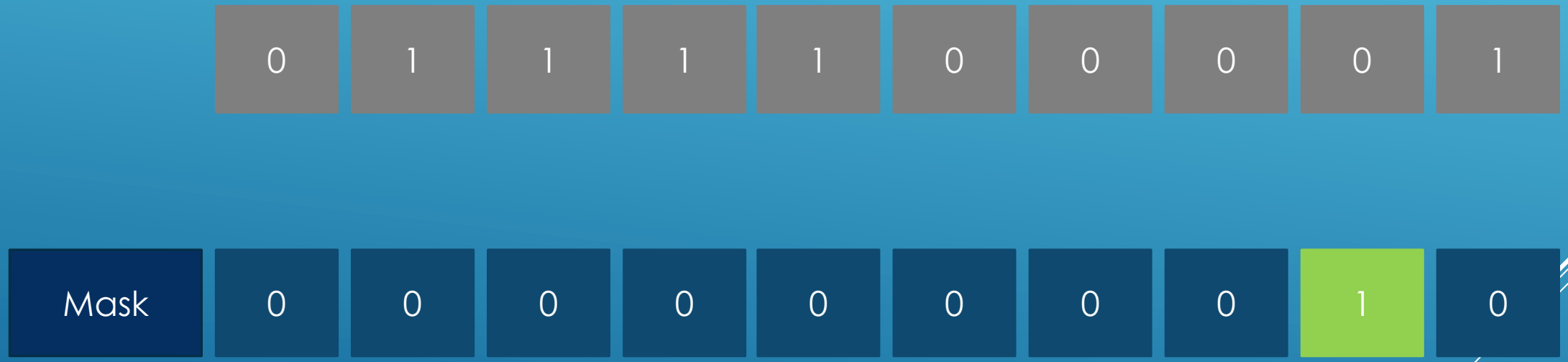
```
//Compound OR with 0000 0010 to have all lower 4 bits turned on
std::cout << std::endl;
std::cout << "Compound OR with 0000 0010 :  " << std::endl;
sandbox_var |= 0b00000010;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;

//Compound AND with 0000 1100 to turn off the 2 lowest bits
std::cout << std::endl;
std::cout << "Compound AND with 0000 1100 :  " << std::endl;
sandbox_var &= 0b00001100;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;

//XOR with 00000011 to turn on the 4 lowest bits again
    std::cout << std::endl;
std::cout << "Compound XOR with 0000 0011 :  " << std::endl;
sandbox_var ^= 0b00000011;
std::cout << std::setw(COLUMN_WIDTH) << "sandbox_var : "
    << std::setw(COLUMN_WIDTH) << std::bitset<8>(sandbox_var) << std::endl;
std::cout << std::endl;
```

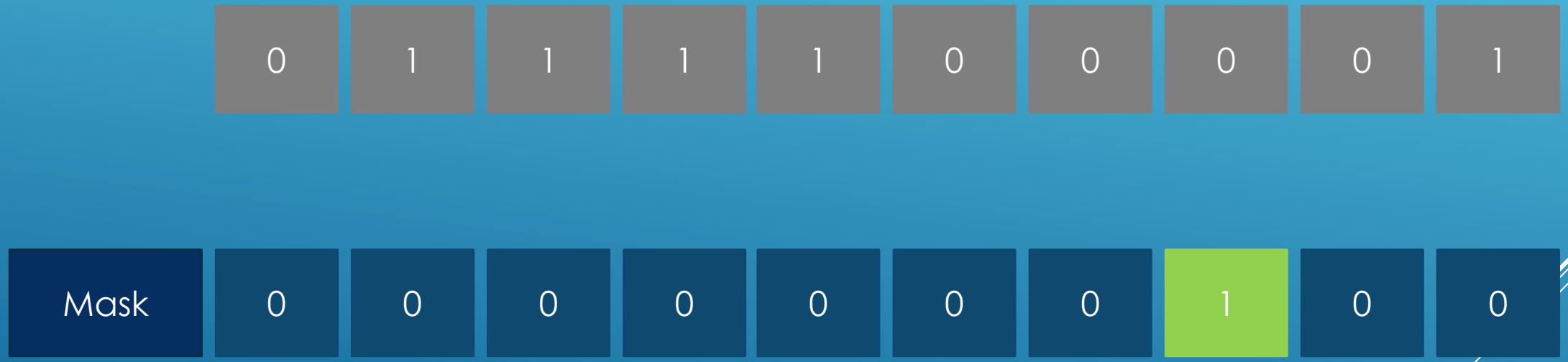
Slide intentionally left empty

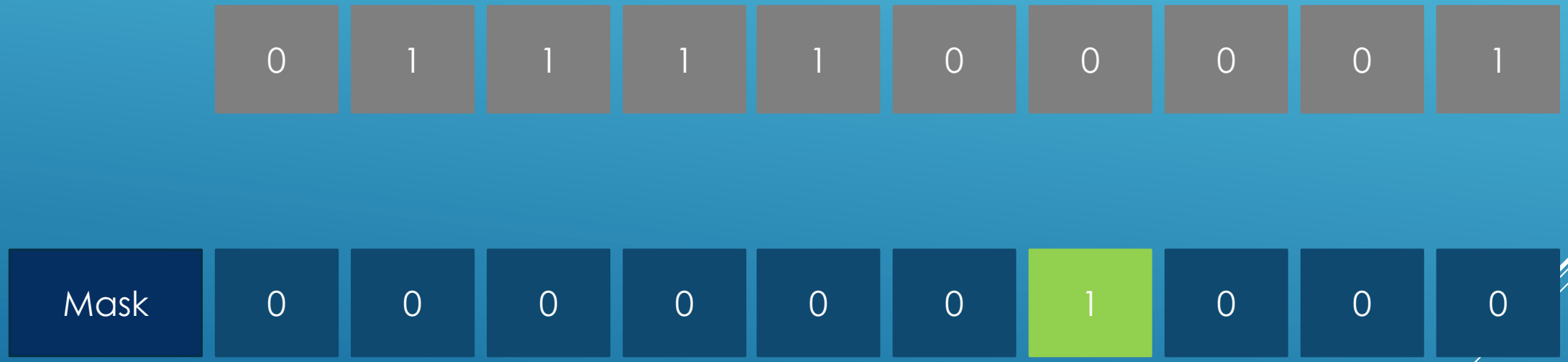
Masks

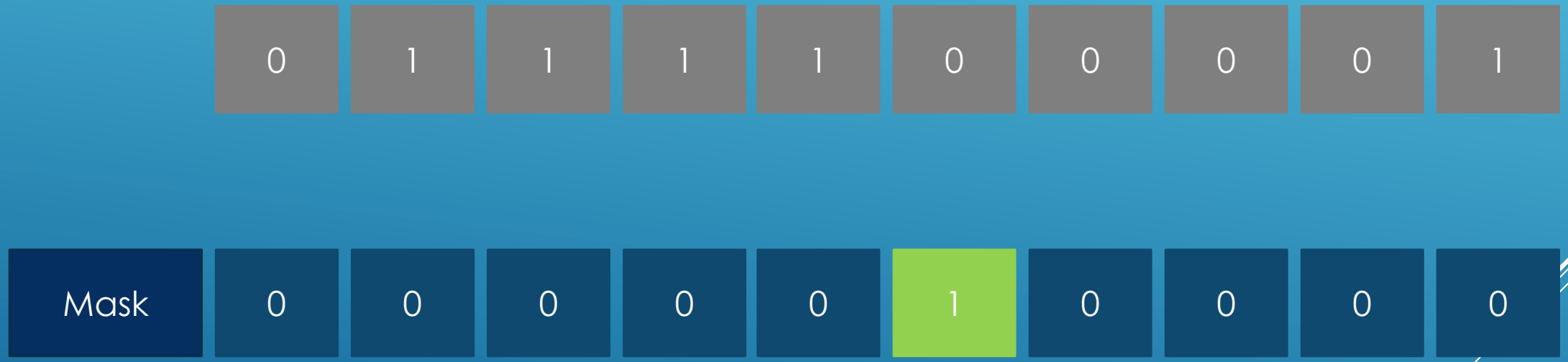

0 1 1 1 1 0 0 0 0 1

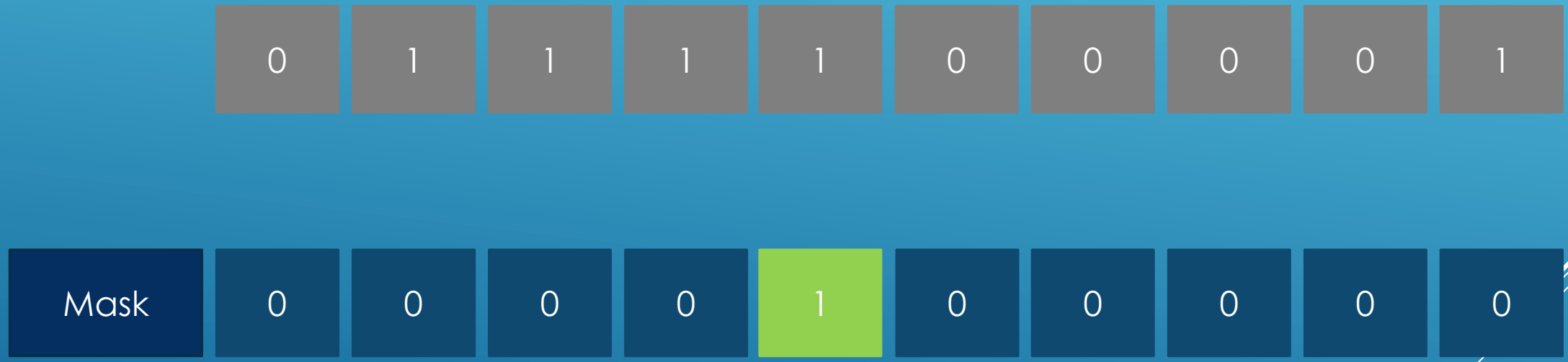

0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 0 0 0 0 0 1

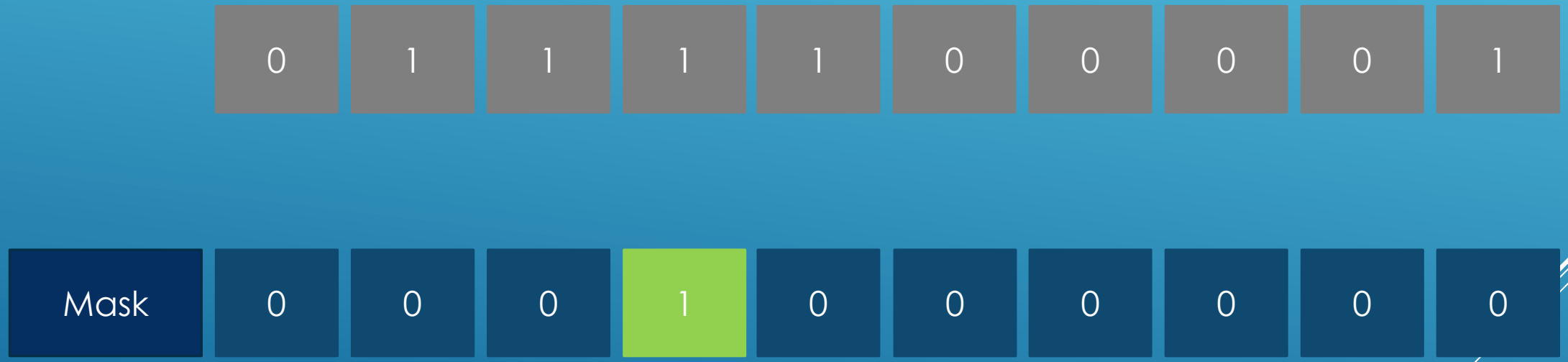

0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 0 0 0 0 1 0

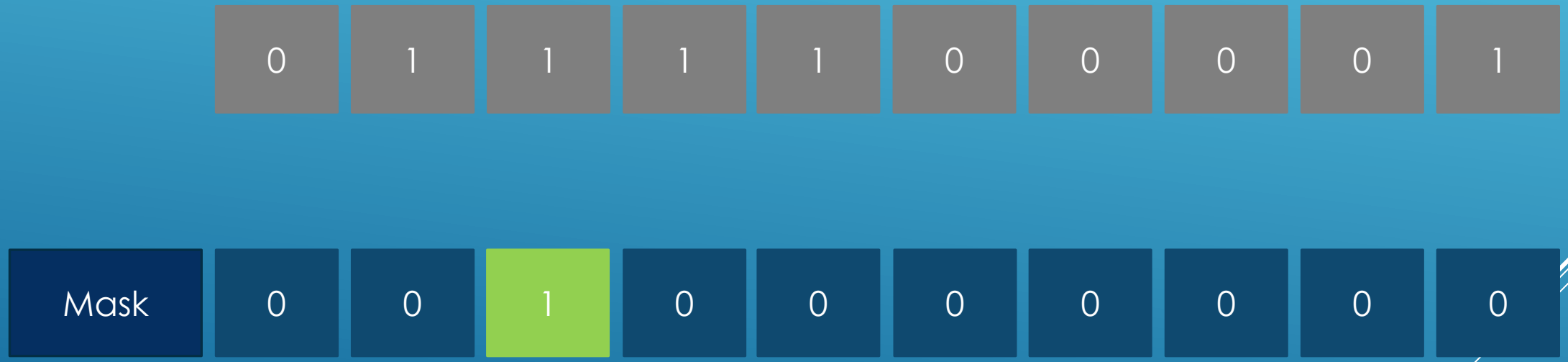

0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 0 0 0 1 0 0

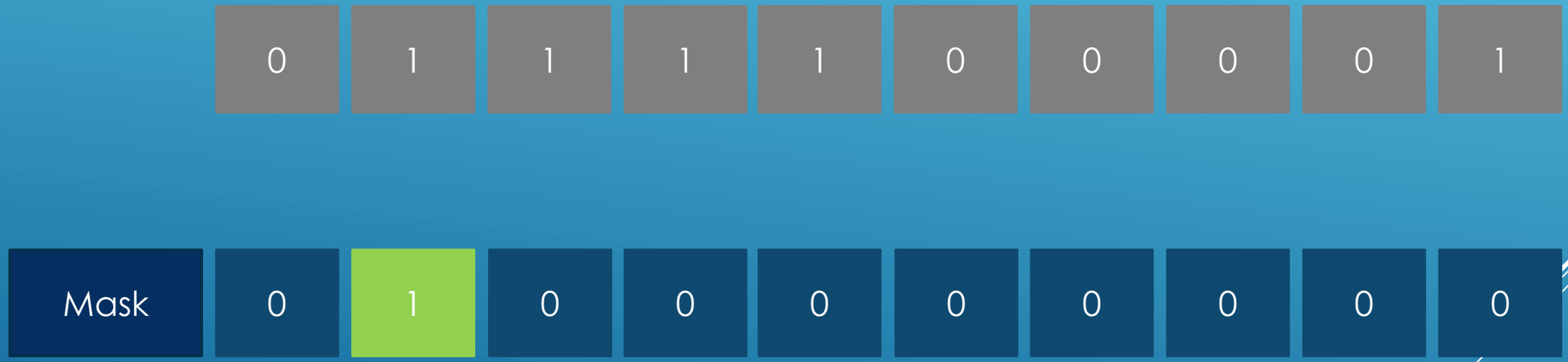

0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 0 0 1 0 0 0

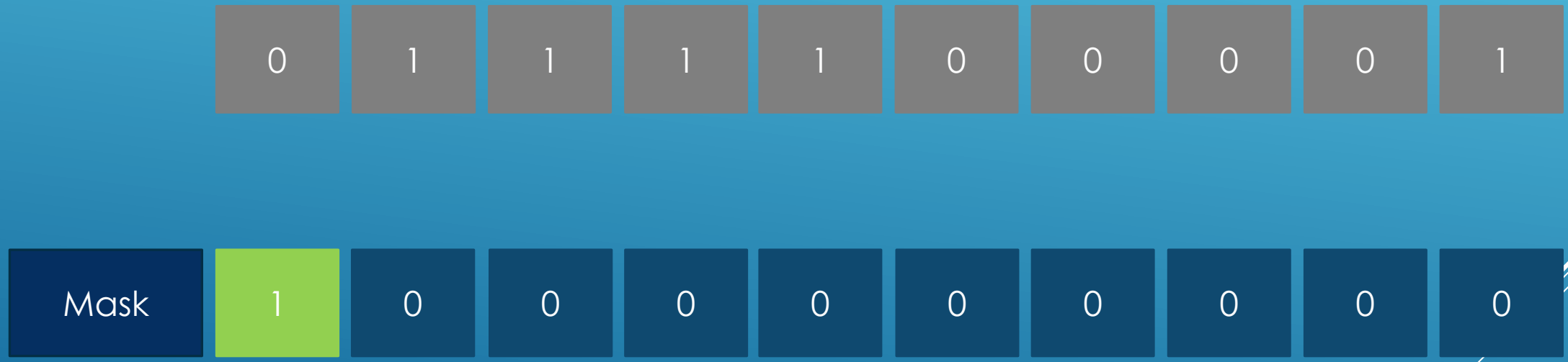

0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 0 1 0 0 0 0


0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 0 1 0 0 0 0 0


0 1 1 1 1 0 0 0 0 1
Mask 0 0 0 1 0 0 0 0 0 0


0 1 1 1 1 0 0 0 0 1
Mask 0 0 1 0 0 0 0 0 0 0


0 1 1 1 1 0 0 0 0 1
Mask 0 1 0 0 0 0 0 0 0 0


0 1 1 1 1 0 0 0 0 1
Mask 1 0 0 0 0 0 0 0 0 0

Bit masks

```
//Highlight position for bit of interest with a 1
//Mask other positions with 0

const unsigned char mask_bit_0 {0b00000001} ;//Bit0
const unsigned char mask_bit_1 {0b00000010} ;//Bit1
const unsigned char mask_bit_2 {0b00000100} ;//Bit2
const unsigned char mask_bit_3 {0b00001000} ;//Bit3
const unsigned char mask_bit_4 {0b00010000} ;//Bit4
const unsigned char mask_bit_5 {0b00100000} ;//Bit5
const unsigned char mask_bit_6 {0b01000000} ;//Bit6
const unsigned char mask_bit_7 {0b10000000} ;//Bit7
```

- Set bit position(s)
- Reset Bit position(s)
- Check bit position(s)
- Toggle bit position(s)

Setting bit position 0

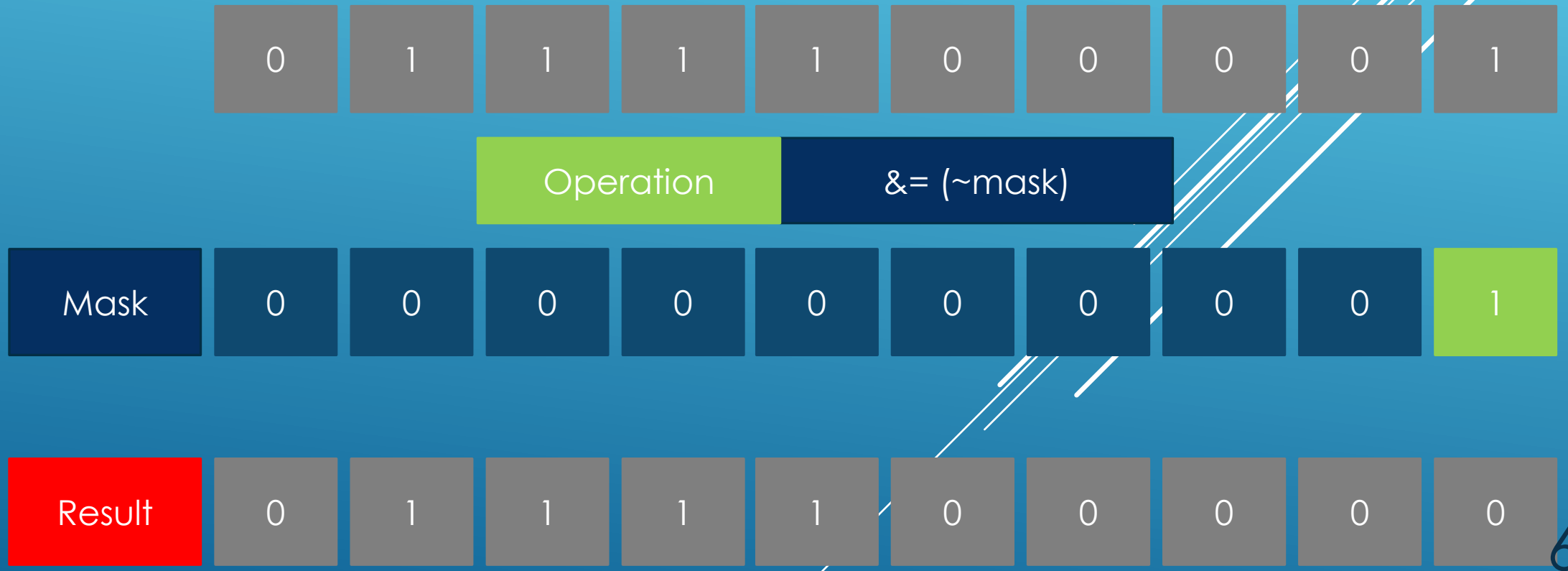
Setting bit position 1

Setting bit position 2

Setting some bits in code

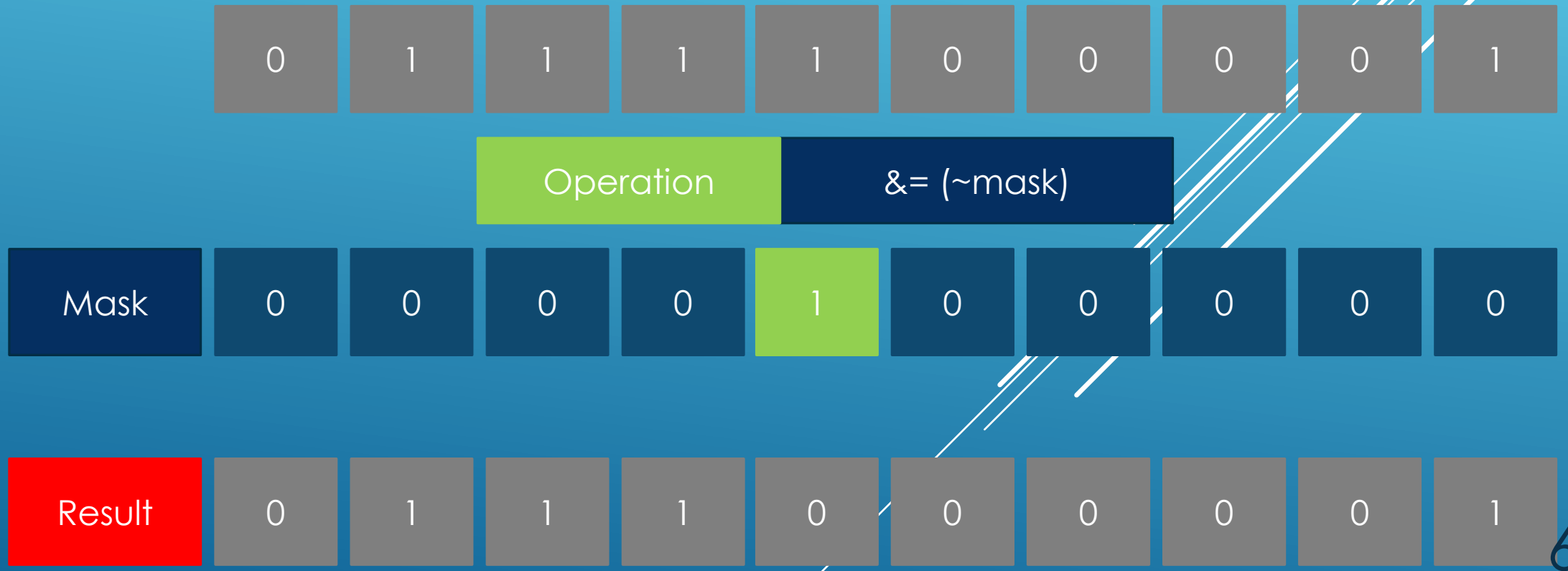
```
unsigned char var {0b00000000};// Starts off all bits off

std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//SETTING BITS
//Setting : |= with mask of the bit

//Set bit 1
std::cout << "Setting bit in position 1" << std::endl;
var |= mask_bit_1;
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//Set bit 5
std::cout << "Setting bit in position 5" << std::endl;
var |= mask_bit_5;
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;
```

Resetting bit position 0



Resetting bit position 1

Resetting bit position 1



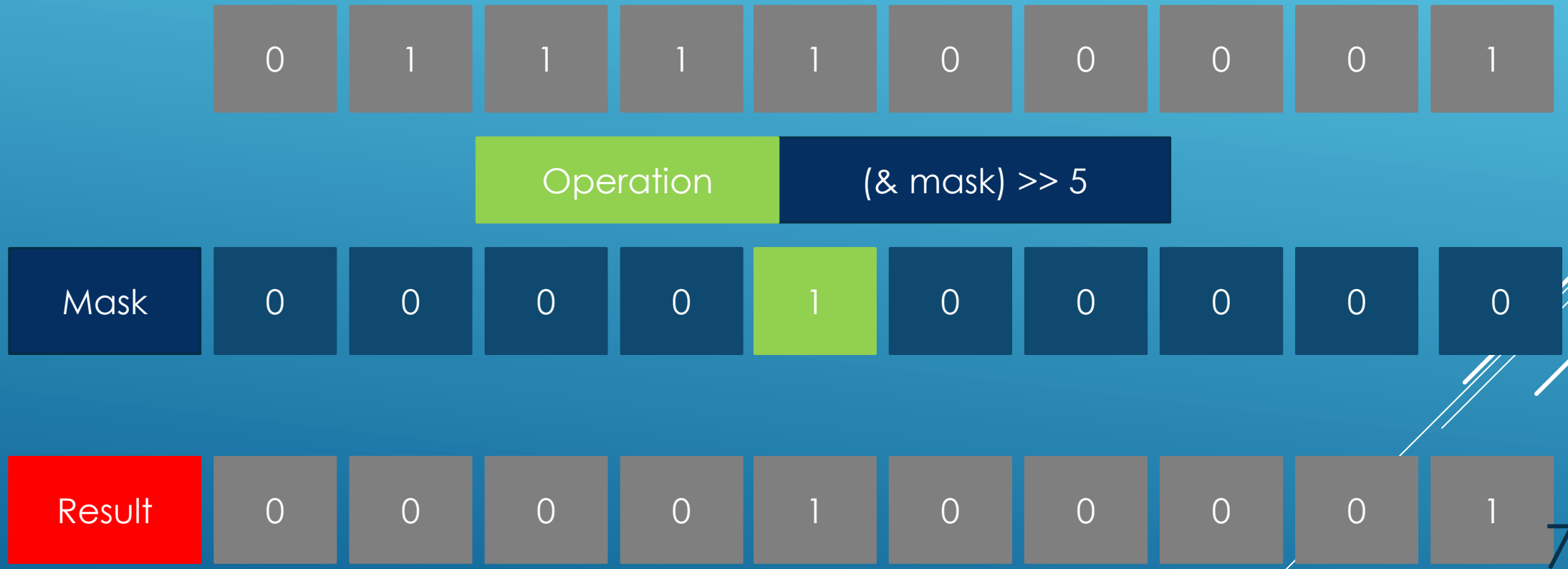
Resetting some bits in code

```
//RESETTING BITS : set to 0
//Resetting : &= (~mask)

//Reset bit 1
std::cout << "Resetting bit in position 1" << std::endl;
var &= (~mask_bit_1);
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//Reset bit 5
std::cout << "Resetting bit in position 1" << std::endl;
var &= (~mask_bit_5);
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;
```

Masking multiple bits at once

```
//Set all bits
std::cout << "Setting all bits" << std::endl;
var |= ( mask_bit_0 | mask_bit_1 | mask_bit_2 | mask_bit_3 |
         mask_bit_4 | mask_bit_5 | mask_bit_6 | mask_bit_7);
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//Reset bits at pos 0,2,4,6
std::cout << "Reset bits at pos 0,2,4,6" << std::endl;
var &= ~(mask_bit_0 | mask_bit_2 | mask_bit_4 | mask_bit_6);
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;
```

Checking bit position 5



Check state of a bit in code

```
//Check state of a bit
std::cout << std::endl;
std::cout << "Checking the state of each bit position (on/off)" << std::endl;
std::cout << "bit0 is " << ((var & mask_bit_0) >> 0 )<< std::endl;
std::cout << "bit1 is " << ((var & mask_bit_1) >> 1 ) << std::endl;
std::cout << "bit2 is " << ((var & mask_bit_2) >> 2 ) << std::endl;
std::cout << "bit3 is " << ((var & mask_bit_3) >> 3 ) << std::endl;
std::cout << "bit4 is " << ((var & mask_bit_4) >> 4 ) << std::endl;
std::cout << "bit5 is " << ((var & mask_bit_5) >> 5 ) << std::endl;

std::cout << "bit6 is " << ((var & mask_bit_6) >> 6 ) << std::endl;
std::cout << "bit6 is " << static_cast<bool>(var & mask_bit_6) << std::endl;

std::cout << "bit7 is " << ((var & mask_bit_7) >> 7 ) << std::endl;
std::cout << "bit7 is " << static_cast<bool>(var & mask_bit_7) << std::endl;
```

Toggling bit position 0

Toggle bits in code

```
//Toggle bits
//Toggle : var ^ mask

//Toggle bit 0
std::cout << std::endl;
std::cout << "Toggle bit 0" << std::endl;
var ^= mask_bit_0;
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//Toggle bit7
std::cout << "Toggle bit 7" << std::endl;
var ^= mask_bit_7;
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;

//Toggle multiple bits in one go : the 4 higher bits
std::cout << "Toggle multiple bits in one go : the 4 higher bits" << std::endl;
var ^= (mask_bit_7 | mask_bit_6 | mask_bit_5 | mask_bit_4);
std::cout << std::setw(COLUMN_WIDTH) <<  "var : "
     << std::setw(COLUMN_WIDTH) << std::bitset<8>(var) << std::endl;
```

Slide intentionally left empty

Mask Demo : Packing function parameters in a variable

```
void use_options_v0 (bool flag0, bool flag1, bool flag2, bool flag3,
      bool flag4, bool flag5, bool flag6, bool flag7){

    std::cout << "Flag0 is : " << flag0 << ", do something with it." <<  std::endl;
    std::cout << "Flag1 is : " << flag1 << ", do something with it."<<  std::endl;
    std::cout << "Flag2 is : " << flag2 << ", do something with it." <<  std::endl;
    std::cout << "Flag3 is : " << flag3 << ", do something with it."<<  std::endl;
    std::cout << "Flag4 is : " << flag4 << ", do something with it."<<  std::endl;
    std::cout << "Flag5 is : " << flag5 << ", do something with it."<<  std::endl;
    std::cout << "Flag6 is : " << flag6 << ", do something with it."<<  std::endl;
    std::cout << "Flag7 is : " << flag7 << ", do something with it."<<  std::endl;
}
```

```
use_options_v0(0,0,1,1,1,0,1,0);
```

```
void use_options_v1(unsigned char flags){

    std::cout << "bit0 is " << ((flags & mask_bit_0) >> 0 ) << ", do something with it!"<< std::endl;
    std::cout << "bit1 is " << ((flags & mask_bit_1) >> 1 ) <<", do something with it!"<< std::endl;
    std::cout << "bit2 is " << ((flags & mask_bit_2) >> 2 ) <<", do something with it!"<< std::endl;
    std::cout << "bit3 is " << ((flags & mask_bit_3) >> 3 ) <<", do something with it!"<< std::endl;
    std::cout << "bit4 is " << ((flags & mask_bit_4) >> 4 ) <<", do something with it!"<< std::endl;
    std::cout << "bit5 is " << ((flags & mask_bit_5) >> 5 ) <<", do something with it!"<< std::endl;
    std::cout << "bit6 is " << ((flags & mask_bit_6) >> 6 ) <<", do something with it!"<< std::endl;
    std::cout << "bit7 is " << ((flags & mask_bit_7) >> 7 ) <<", do something with it!"<< std::endl;
}
```

```
use_options_v1(mask_bit_2 | mask_bit_3 | mask_bit_4 | mask_bit_6);
```

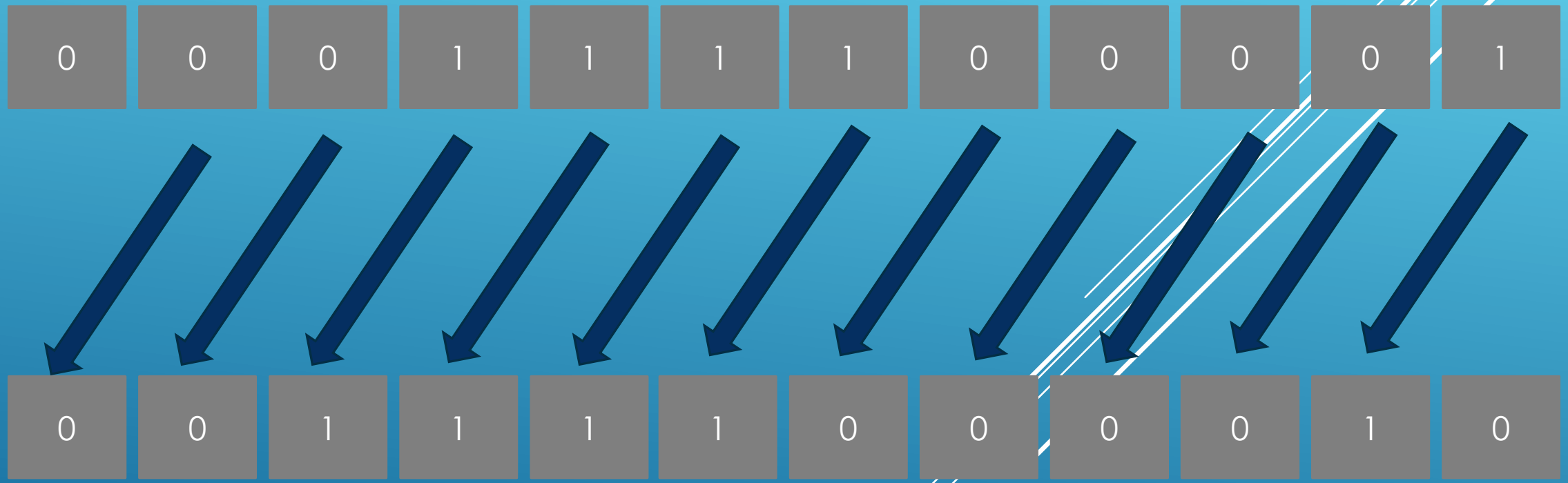
Slide intentionally left empty

Packing Color Information

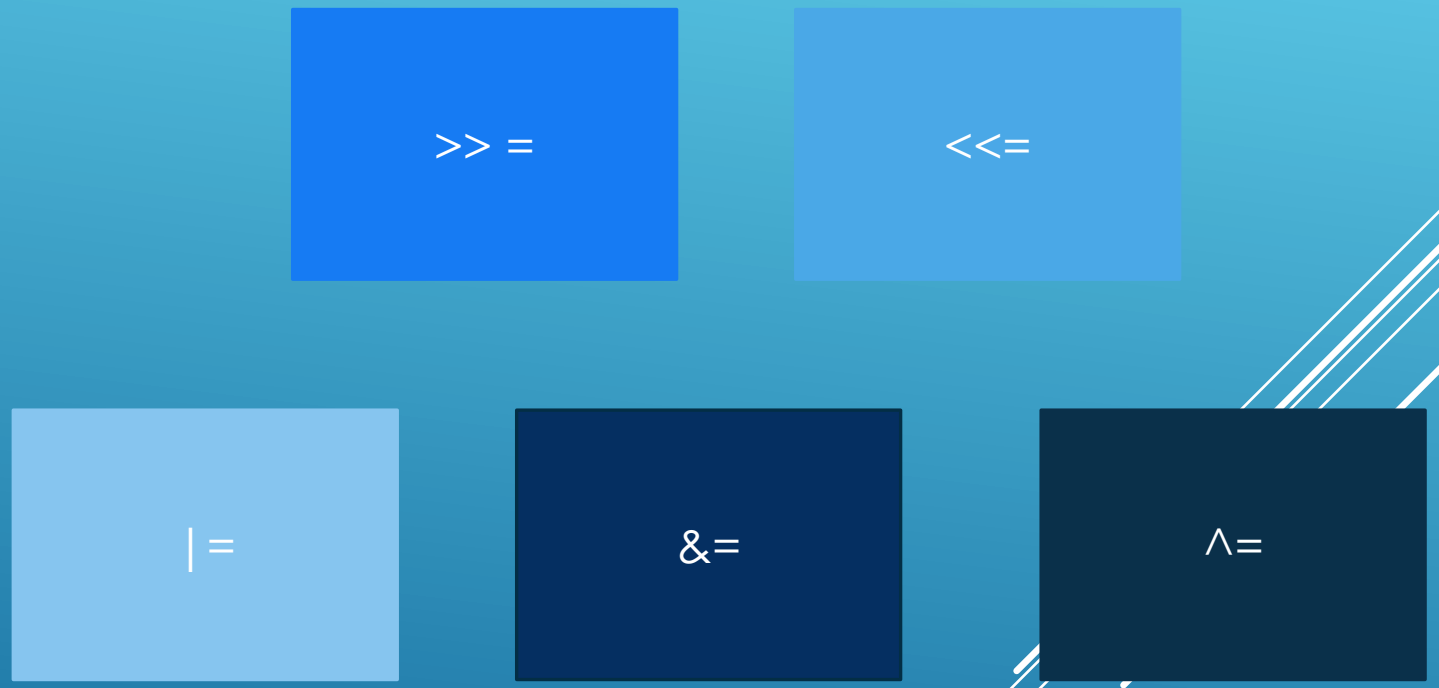
```cpp
const unsigned int red_mask {0xFF000000};
const unsigned int green_mask {0x00FF0000};
const unsigned int blue_mask {0x0000FF00};
const unsigned int alpha_mask {0x000000FF};


unsigned int my_color {0xAABCDE00};

//We shift to make sure the color byte of interest is in the
// lower index byte position so that we can interpret that as an integer,
// which will be between 0 and 255.

//Set some format options
std::cout << std::hex << std::showbase << std::endl;

std::cout << "Red is : " <<  ((my_color & red_mask) >> 24) << std::endl;
std::cout << "Green is : " <<  ((my_color & green_mask) >> 16) << std::endl;
std::cout << "Blue is : " <<  ((my_color & blue_mask) >> 8) << std::endl;
std::cout << "Alpha is : " <<  ((my_color & alpha_mask) >> 0) << std::endl;
```

Slide intentionally left empty

Bitwise Operators : Summary


0 0 0 1 1 1 1 0 0 0 0 1
0 0 1 1 1 1 0 0 0 0 1 0


AND
OR
XOR
NOT



Setting bit position 0

- Set bit position(s)
- Reset Bit position(s)
- Check bit position(s)
- Toggle bit position(s)