Working with bytes

Reading a binary file

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  // Open the file 'mydata.dat'
  ifstream input;
  input.open("mydata.dat", std::ios::binary);

  // Read one byte from the file
  unsigned char mybyte=0;
  input.read((char *)&mybyte, 1);

  // Close the file
  input.close();
  return 0;
}
```

Open a file in binary mode

A good type for containing a byte = unsigned char

Read one byte from the file. Please have a look on the prototype of the function read:

`istream& read (char* s, streamsize n);`

Extracts n characters from the stream and stores them in the array pointed to by s.

Displaying a byte at the screen

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

using namespace std;

int main()
{
  // Defining a value
  unsigned char value = 64;

  // Displaying the value as a ASCII code
  cout << value << endl;

  // Displaying the value as an integer
  cout << (int)value << endl;

  // Displaying the value as an bits
  cout << bitset<8>(value) << endl;

  // End
  return 0;
}
```

Alternative: specifying the value in terms of bits

```
unsigned char value = 0b01000000;
```

By default, a char is considered as a ASCII code. Display: **@**

Static conversion into integer. Display: **64**

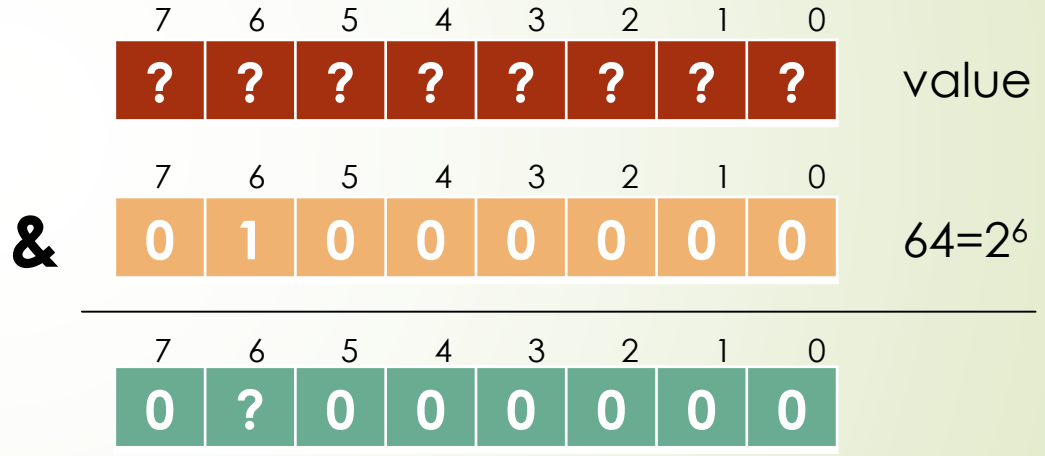
Static conversion into a set of bits. Display: **01000000**

Working with bits

Getting a bit from a byte

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
 // Define a value
  unsigned char value = 222;

  // Get the 6th bit of value
  bool mybit = false;
  mybit = (value&64)!=0;

  // Display the bit a the screen
  cout << (int) mybit << endl;
  return 0;
}
```

For extracting the n^{th} bit from a number, the best way is to do a LOGICAL AND with 2^n.

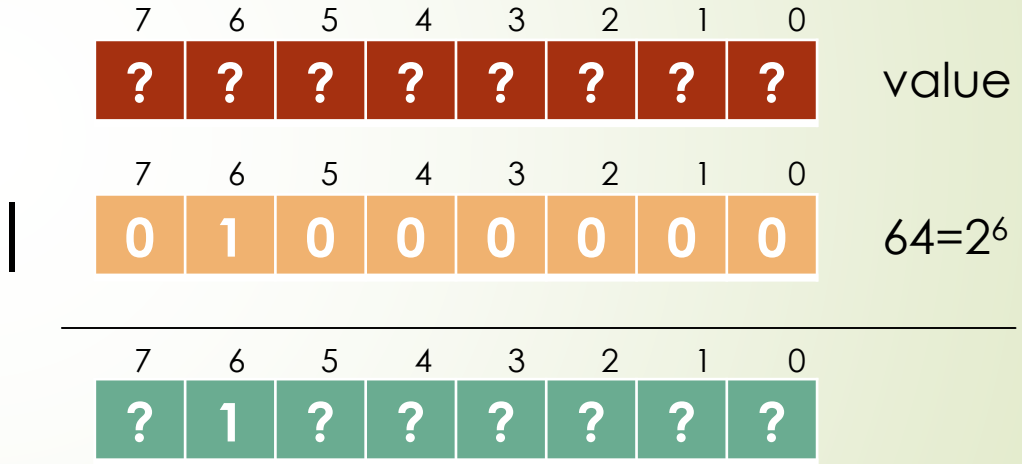


If the 6th bit is equal to 0, then the result is equal to 0.
Else the result is not equal to 0.

Setting a bit to 1 in a byte

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
 // Define a value
  unsigned char value = 222;
  cout << (int) value << endl;

  // Set the 6th bit to 1
  value = value | 64;

  // Display the modified value
  cout << (int) value << endl;
  return 0;
}
```

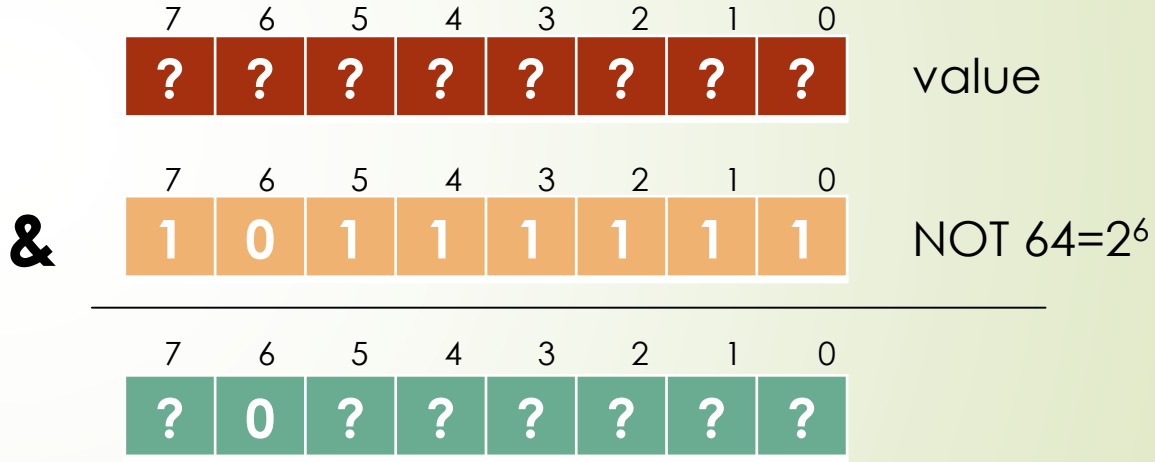
For setting the n^{th} bit from a number, the best way is to do a LOGICAL OR with 2^n.



Setting a bit to 0 in a byte

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
 // Define a value
  unsigned char value = 222;
  cout << (int) value << endl;

  // Set the 6th bit to 1
  value = value & (~64);

  // Display the modified value
  cout << (int) value << endl;
  return 0;
}
```

For setting the n^{th} bit from a number, the best way is to do a LOGICAL AND with NOT 2^n.



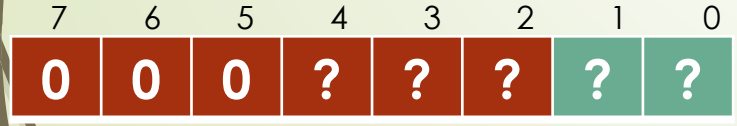
Extracting bits from a byte

We assume the following byte `value`, and we would like to extract 2 given bits (in green).

Step 1:

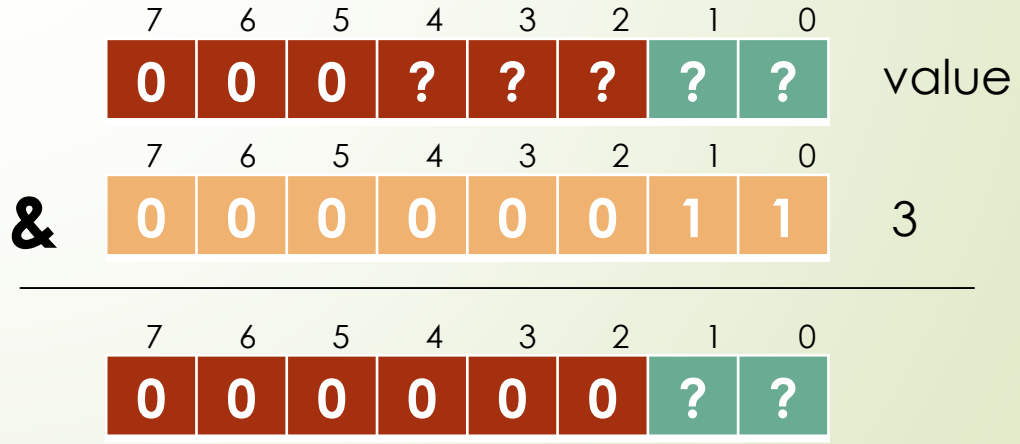
Shift the bits of `value`, 4 times on the right

value = value >> 4;



Step 2:

Set the non-green bits to 0.



Extracting bits from a byte

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
 // Define a value
  unsigned char value = 222;

  // Get the 2 bits
  unsigned char value2 = (value>>4)&3;

  // Display the value of the 2 bits
  cout << (int) value2 << endl;
  return 0;
}
```

Code corresponding to the previous slide.