Week 2: Bit masking

Bit masks uses Boolean operations to access individual bits from binary data

- Used in 'low level' programming
  - device drivers and other hardware configuration/communication
  - data packet encoding/decoding
  - low level graphics
- We shall use Java to illustrate the bitwise Boolean operators which behave like the logical NOT, AND, OR, and XOR which you have already met
- In Java, ! is the bitwise NOT operator
- In Java, | (a pipe) is the bitwise OR operator
- In Java, & (an ampersand) is the bitwise AND operator
- In Java, ^ is the bitwise XOR operator

Bitwise operators in java

A Boolean variable in Java is one that has two states:

- true
- false

In Java Boolean operations can be written as:
Boolean x, a=true;
x = !a; // x is 'false'
– where ! is the NOT operator (see next slide)

It can also be written numerically as follows:
int x, a=1;
x = ~a; // x is 0
– where ~ is the COMPLEMENT operator (bitwise NOT)

We will show only bitwise operators from now on.

Boolean bitwise operators

- Boolean Bitwise Operator: NOT
  – NOT returns the opposite value
    • i.e. True becomes False, False becomes True

NOT	-
0	1
1	0

(written as /x or -x or sometimes !x or)

Java: (Note 0x denotes a hexadecimal number follows)
int x, a=1; // a = 0x01, 0000 0001_2
x = ~a; // x = 0xFE, 1111 1110_2 = 254

Bitwise AND operator

AND is true when ALL inputs are true – written as: x.y or x∧y or xy

AND	.
0 . 0	0
0 . 1	0
1 . 0	0
1 . 1	1

Java:

```
int x,  a = 0x18,  b = 0x11;
x = a & b;          // a = 0001 1000₂
                    // b = 0001 0001₂
                    // x = 0001 0000₂ = 0x10 = 16
```

Bitwise OR operator

OR is true when ANY input is true – written as: x+y or x∨y

OR	+
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	1

Java:

```
int x,  a = 0x18,  b = 0x11;
x = a | b;          // a = 0001 1000₂
                    // b = 0001 0001₂
                    // x = 0001 1001₂ = 0x19 = 25
```

Bitwise XOR operator

XOR (exclusive OR) is similar to OR, except it is true when only one, not both, input is true – written as $x \oplus y$

XOR	⊕
0 . 0	0
0 . 1	1
1 . 0	1
1 . 1	0

Java:

```
int x,  a = 0x18,  b = 0x11;
x = a ^ b;          // a = 0001 1000₂
                    // b = 0001 0001₂
                    // x = 0000 1001₂ = 0x09 = 9
```

Bit masking (set)

To set (i.e. make 1) a bit in a byte, OR it with a mask of all 0's except the bit to set
– e.g. set B4 of 10001101

```
 10000101
 00001000 ← turn ON bits where there is a '1'
 -------- Boolean OR
 10001101
```

- In Java:

```
short data = 0x85; //In binary, 1000 1101
short bitmask = 0x08; //In binary, 0000 1000
short result =  (short) (data | bitmask);
System.out.println(String.format("%x", result));
```

Bit masking (reset)

To reset (i.e. make 0) a bit in a byte, AND it with a mask of all 1's except the bit to reset
– e.g. reset the LSB of 10001101

```
 10001101
  11111110 ← turn OFF bits where there is a '0'
 -------- Boolean AND
 10001100
```

- In Java:

```
short data = 0x8D; //In binary, 1000 1101
short bitmask = 0x01; // In binary, 0000 0001
short result =  (short) (data &   ! bitmask);
System.out.println(String.format("%x", result));
```

Bit masking (flip)

To flip (i.e. 1→0,0→1) a bit in a byte, XOR it with a mask of all 0's except the bit(s) to flip
– e.g. flip B4 and the LSB of 10001101
10000101
00001001 ← complement bits where there is a '1'
---------- Boolean XOR
10001100

- In Java:

```
short data = 0x85; // In binary, 1000 0101
short bitmask = 0x08 | 0x01; // 0000 1000 | 0000 0001
short result =  (short) (data ^ bitmask);
System.out.println(String.format("%x", result));
```

Example of Setting Groups of Bits Using the OR Function

You saw earlier, the OR function's output is true (one) if any of its inputs is true (one). If you "OR" a bit with a value known to be 1, the result is always going to be a 1, no matter what the other value is. If you "OR" with a 0, the original value (1 or 0) is not changed.

Suppose we have the 12-bit binary input number: xxxxxxx xxxx , where X is 1 or 0, and we want to set the middle six bits.

Input	x	x	x	x	x	x	x	x	x	x	x	x
Mask	0	0	0	1	1	1	1	1	1	0	0	0
Result of OR Operation	x	x	x	1	1	1	1	1	1	x	x	x

Example of Clearing (Resetting) Groups of Bits Using the AND Function

If you AND a bit with 0, it will clear/unset it to 0 regardless of what the bit was before, while ANDing with 1 will leave the bit unchanged. To reset the middle six bits:

Input	X	X	X	X	X	X	X	X	X	X	X	X
Mask	1	1	1	0	0	0	0	0	0	1	1	1
Result of And Operation	X	X	X	0	0	0	0	0	0	X	X	X

Inverting(Flipping) Groups of Bits Using the XOR Function

We can also look at this "unsetting" function a different way: ANDing with a bit mask means that you "keep" the bits where the mask is a one and "remove" the bits where it is a zero.

A way to remember XOR is "one or the other but not both".

If you XOR with a 1, the input value is flipped, while XORing with a 0 causes the input to be unchanged.

To Invert(flip) the middle six bits:

Input	x	x	x	x	x	x	x	x	x	x	x	x
Mask	0	0	0	1	1	1	1	1	1	0	0	0
Result of XOR Operation	x	x	x	$\bar{x}$	$\bar{x}$	$\bar{x}$	$\bar{x}$	$\bar{x}$	$\bar{x}$	x	x	x