Binary Numbers

1	1	0	1	1	0	0	0
1×2^7	1×2^6	0×2^5	1×2^4	1×2^3	0×2^2	0×2^1	0×2^0
= 128	= 64	= 0	= 16	= 8	= 0	= 0	= 0

128 + 64 + 0 + 16 + 8 + 0 + 0 + 0

= 216

Bit Strings as Sets

1	1	0	1	1	0	0	0

- a bit string can also be interpreted as a set
  - associate each position with an element of the set
  - treat 1 as “in” and 0 as “not in”
- integer types in Java
  - `byte` – 8 bits
  - `short` – 16 bits
  - `int` – 32 bits
  - `long` – 64 bits

Bitwise Operators

Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	op1 | op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	op1 << op2
>>	Bitwise Shift Right	op1 >> op2

Bitwise AND

Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Binary Representation of 2 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Binary Representation of a&b = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Bitwise OR

Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Binary Representation of 2 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Binary Representation of a|b = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Bitwise Operators

Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	op1 | op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	op1 << op2
>>	Bitwise Shift Right	op1 >> op2

Bitwise One's Complement

Binary Representation of 10 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Binary Representation of ~a = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1

Shift Operators

Operators	Description	Use
&	Bitwise AND	op1 & op2
|	Bitwise OR	op1 | op2
^	Bitwise Exclusive OR	op1 ^ op2
~	Bitwise Complement	~op
<<	Bitwise Shift Left	op1 << op2
>>	Bitwise Shift Right	op1 >> op2

Binary Representation of 8 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Binary Representation of a<<b = 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

Binary Representation of 8 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Binary Representation of a>>b = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Set Operations

- bitwise | corresponds to set union
- bitwise & corresponds to set intersection
- to test whether *x* is in *A* –
  `( A & x ) != 0`
- adding an element – $A \cup \{ x \}$
  `A | x`
- removing an element – $A \setminus \{ x \}$
  `A & ~x`
- generating { *x* } where *x* is the element in position *pos*
  `1 << pos`

Hexadecimal

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

$1010\,1011\,0111_2$ = 0xAB7

1. Suppose that the numbers x and y represent the sets A and B. Show that the set $A \setminus B$ is represented by $x \,\&\, (\tilde{}\, y)$.

2. Write each of the following binary numbers in hexadecimal:
a) 10110110_2 **b)** 10_2 **c)** 111100001111_2 **d)** 101001_2

3. Write each of the following hexadecimal numbers in binary:
a) 0x123 **b)** 0xFADE **c)** 0x137F **d)** 0xFF11

4. Give the value of each of the following expressions as a hexadecimal number:
a) 0x73 | 0x56A **b)** ~0x3FF0A2FF
c) (0x44 | 0x95) & 0xE7 **d)** 0x5C35A7 & 0xFF00
e) 0x5C35A7 & ~0xFF00 **f)** ~(0x1234 & 0x4321)

Applications

```
String source = e.getSource().getClass().getName();

int mod = e.getModifiers();


if ((mod & ActionEvent.ALT_MASK) > 0) {


}

if ((mod & ActionEvent.SHIFT_MASK) > 0) {


}

if ((mod & ActionEvent.META_MASK) > 0) {


}

if ((mod & ActionEvent.CTRL_MASK) > 0) {


}
```

event handling in Java Swing


```
function printInformation (): void {
    const info: any = {
        firstName: 'Hendrik',
        lastName: 'Erz',
        affiliation: 'Linköping University',
        occupation: 'PhD Student'
    }

    console.log(`${info.firstName} ${info.lastName}, ${info.occupation} (${info.affiliation})`)
}
```

```
const FIRST_NAME = 1
const LAST_NAME = 2
const OCCUPATION = 4
const AFFILIATION = 8
```

```
function printInformation (which: number): void {
    const info: any = {
        firstName: 'Hendrik',
        lastName: 'Erz',
        occupation: 'PhD Student',
        affiliation: 'Linköping University'
    }

    if (which & FIRST_NAME) {
        console.log(info.firstName)
    }

    if (which & LAST_NAME) {
        console.log(info.lastName)
    }

    if (which & OCCUPATION) {
        console.log(info.occuption)
    }

    if (which & AFFILIATION) {
        console.log(info.affiliation)
    }
}
```

```
printInformation(FIRST_NAME | LAST_NAME)

printInformation(OCCUPATION | LAST_NAME)
```

```
const SHORT_OCC = LAST_NAME | OCCUPATION

// ... snip ...

if (which & SHORT_OCC === SHORT_OCC) {
    console.log(`${info.lastName}, ${info.occupation}`)
    return
}
```

https://www.hendrik-erz.de/post/bitwise-flags-are-beautiful-and-heres-why

Applications

- Linux file permissions

```
$ ls -l

drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- File type: `-`
- Permission settings: `rw-r--r--`
- Extended attributes: dot (`.`)
- User owner: `root`
- Group owner: `root`

- `rw-` user
- `r--` group
- `r--` other

```
$ chmod ug+rwx example.txt
$ chmod o+r example2.txt
```

`chmod o-rwx example.txt`

`chmod 644 example.txt`

octal

- r (read): 4
- w (write): 2
- x (execute): 1

https://www.redhat.com/sysadmin/linux-file-permissions-explained

Applications of Bitwise Operations

- extracting RGB color components
  - RGB color is stored as a 32-bit integer

```
rgb = 0xff8000;
b = rgb & 0xff;
g = (rgb >> 8) & 0xff ; // g contains 80
r = (rgb >> 16) & 0xff ; //r contains ff
```

- top 8 bits are used for transparency (alpha)

9. Java, C, and C++ each have a boolean data type that has the values *true* and *false*. The usual logical and, or, and not operators on boolean values are represented by the operators &&, ||, and !. C and C++ allow integer values to be used in places where boolean values are expected. In this case, the integer zero represents the boolean value *false* while any non-zero integer represents the boolean value *true*. This means that if x and y are integers, then both $x \,\&\, y$ and $x \,\&\&\, y$ are valid expressions, and both can be considered to represent boolean values. Do the expressions $x \,\&\, y$ and $x \,\&\&\, y$ always represent the same boolean value, for any integers x and y? Do the expressions $x \,|\, y$ and $x \,||\, y$ always represent the same boolean values? Explain your answers.