Bit and Bitwise Operations

Bit, Numerals Systems and Bitwise Operations

01

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#TECH-FUND

0|1

Bit

What is a bit?

Bit

- Unit **used in computing**
- Unit of **information**
- Have only one of **two values** – either a **0** or **1**
- Anything with **two separate states** can store 1 bit
  - Logical values (True/False)
  - Algebraic signs (+/-)
  - Activation States (On/Off)

SoftUni Foundation

5

101_b

0x8

Numerals Systems

Decimal, Binary and Hexadecimal

Numeral Systems

- System for **expressing numbers**
- Different systems represent **real** and **integer numbers**
- Each system has a **base** (e.g. 2, 10, 16)

Decimal	Binary	Hexadecimal
30	111110	1E
45	101101	2D
60	111100	3C

Decimal Numbers

- Decimal numbers (**base 10**)
  - Represented using 10 numerals
    - **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**
  - Each position represents a **power of 10**

$$
\begin{aligned}
401 &= 4*10^2 + 0*10^1 + 1*10^0 = \\
&= 4*100 + 0*10 + 1*1 = \\
&= 400 + 0 + 1 = 401
\end{aligned}
$$

Binary Numeral System

- The **binary system** is used in computation
- Binary numbers (**base 2**)
  - Represented by **sequence of 0** or **1**

$$5 \rightarrow 101_b$$

  - Each position represents a **power of 2**

$$101_b = 1*2^2 + 0*2^1 + 1*2^0 = 4 + 0 + 1 = 5$$

$$1010_b = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 8 + 0 + 2 + 0 = 10$$

Binary and Decimal Conversion

- Binary to Decimal

$$1011_b = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$$
$$= 1*8 + 0*4 + 1*2 + 1*1 =$$
$$= 8 + 0 + 2 + 1 =$$
$$= 11$$

- Decimal to Binary

```
11 / 2 = 5 (1)
5 / 2 = 2 (1)
2 / 2 = 1 (0)
1 / 2 = 0 (1)
```

Problem: Binary Digits Count

- You are given a positive integer number **N** and one binary digit **B** (0 or 1)
- Write a program that finds the number of **B** digits in **N**

```
20
0
```
→
```
3
```

```
15
1
```
→
```
4
```

```
10
0
```
→
```
2
```

```
23
1
```
→
```
4
```

Solution: Binary Digits Count

SoftUni Foundation

- Read the input from the user – **n** and **b**
- Convert the number in binary numeral system
- Count the **b** digits in **n**
- Print the count

Hexadecimal Numbers

SoftUni Foundation

- Hexadecimal Numbers (**base 16**)
  - Represented using **16 literals**
    - **0, 1, 2, …9, A, B, C, D, E and F**
- Usually **prefixed with 0x** (0x8) in computer science
- Each position represents a **power of 16**

$$
\begin{aligned}
9786_{hex} &= 9*16^3 + 7*16^2 + 8*16^1 + 6*16^0 = \\
&= 9*4096 + 7*256 + 8*16 + 6*1 = \\
&= 36864 + 1792 + 128 + 6 = 38790
\end{aligned}
$$

Hexadecimal Conversions

- Hexadecimal to Decimal

```
1F4hex = 1*16² + 15*16¹ + 4*16⁰ =
       = 1*256 + 15*16 + 4*1 =
       = 256 + 240 + 4 =
       = 500
```

- Decimal to Hexadecimal

```
500 / 16 = 31 (4)
31 / 16 = 1 (F)
1 / 16 = 0 (1)
```

Hexadecimal Conversions (2)

- The conversion from **binary** to **hexadecimal** (and back) is straightforward
  - Each hex digit corresponds to a **sequence of 4 binary digits**

```
A2E3F = 1010 0010 1110 0011 1111
A = 1010
2 = 0010
E = 1110
3 = 0011
F = 1111
```

1010 0010 1110 0011 1111 = A2E3F

$1010_b = 10_{dec} = A_{hex}$

$0010_b = 2_{dec} = 2_{hex}$

$1110_b = 14_{dec} = E_{hex}$

$0011_b = 3_{dec} = 3_{hex}$

$1111_b = 15_{dec} = F_{hex}$

Storing Information

Integer and Floating-Point Numbers and text

Representing Integers

- Integer numbers are sequence of bits
- The sign is determined by the **Most Significant Bit** (**MSB**)
- Leading **0** means **positive number**
- Leading **1** means **negative number**
- Example (8 bit numbers)

```
0XXXXXXX_b > 0 //00010010_b = 18
00000000_b = 0
1XXXXXXX_b < 0 //10010010_b = -110
```

Representation of Integer Numbers

- Positive **8-bit** numbers have the format **0XXXXXXX**
  - The value is the decimal value of their last **7 bits** (**XXXXXXX**)
- Negative **8-bit** numbers have the format **1YYYYYYY**
  - The value is **128**(2^7) minus the decimal value of **YYYYYYY**

-2^7

$10010010_b = -(2^7 - 10010_b) = -(128 - 18) = -110$

Positive and Negative Integers

- The largest 8-bit integer is:

```
127 = (2⁷ – 1) = 01111111b
```

- The smallest negative 8-bit integer is:

```
-128 = (-2⁷) = 10000000b
```

- The largest 32-bit integer is:

```
2147483647 = (2³¹ – 1) = 0111…1111b
```

- The smallest negative 32-bit integer is:

```
-2147483648 = (-2³¹) = 1000…0000b
```

2^7

2^{31}

Representing Real Numbers

- **IEEE 754** - **technical standard** for floating-point computation
- Addressed **many problems** found in the **floating point implementations**
- The standard defines:
  - Arithmetic Formats – sets of binary and decimal floating-point data
  - Exchange formats – encoding (bit sequences)
  - Rounding Rules
  - Operations – arithmetic and other operations
  - Exceptions – such as division by zero

Storing Floating-Point Numbers

- Sequence of bits
- Consists of **sign bit**, **exponent** and **mantissa**

$$\overbrace{\underbrace{1}_{\text{Sign}}\ \underbrace{10000011}_{\text{Exponent}}.\underbrace{01010010100000000000000}_{\text{Mantissa}}}^{\text{32 Bits}}$$

- Errors in **calculations** and **precision**
  - Cannot be represented as a **sum of powers of the number 2**

Representing Text

- System that uses **binary numbers** (**0** and **1**) to represent chars
  - Letters, numerals, etc.
- In the **ASCII** each character consists of **8 bits**
- In the **Unicode** encoding each character consists of **16 bits**

Binary	Decimal	Character
01000001	65	A
01000010	66	B

A

Sequence of Characters

- **String** is the text representation in the programming
  - **String** is an **array of characters**

0	1	2	3	4
H	E	L	L	O

  - Characters in the string can be:
    - 8 bit (ASCII)
    - 16 bit (UTF-16)

Bitwise Operations

Bitwise Operators and Bit Shifts

Bitwise Operators

- Bitwise operator `~` turns all `0` to `1` and all `1` to `0`
  - Like `!` for boolean expressions but bit by bit
- The operators `|`, `&` and `^` behave like `||`, `&&` and `^` for boolean expressions but bit by bit
- Behavior of the operators `|`, `&` and `^`:

<table>
<tr><th>Operator</th><th>|</th><th>|</th><th>|</th><th>&</th><th>&</th><th>&</th><th>^</th><th>^</th><th>^</th></tr>
<tr><td>Operand</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>Operand2</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>Result</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
</table>

Bitwise Operators Examples

SoftUni Foundation

- Bitwise **NOT (~)**

```
5       //0101
~5      //1010
```

- Bitwise **AND (&)**

```
5       //0101
3       //0011
5 & 3   //0001
```

- Bitwise **OR (|)**

```
5       //0101
3       //0011
5 | 3   //0111
```

- Bitwise **XOR(^)**

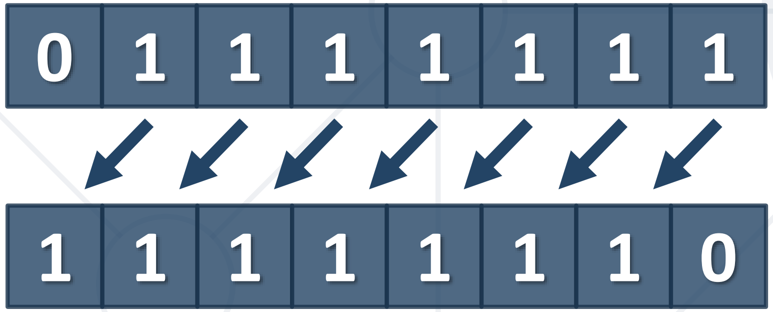
```
5       //0101
3       //0011
5 ^ 3   //0110
```

Bit Shifts

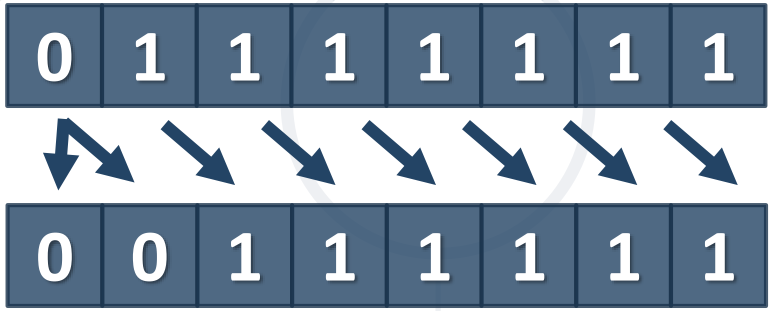
- Bits are moved (**shifted**) to the **left** or **right**
- Registers in a computer have **fixed width**
  - The bits that fall outside the number are **lost** and **replaced with 0**
- **Left** and **Right** Shifts
- **Logical** and **Arithmetic** Shifts

Arithmetic Shift

- Bits that are shifted out of either end are discarded
- Left Arithmetic Shift (<< operator)

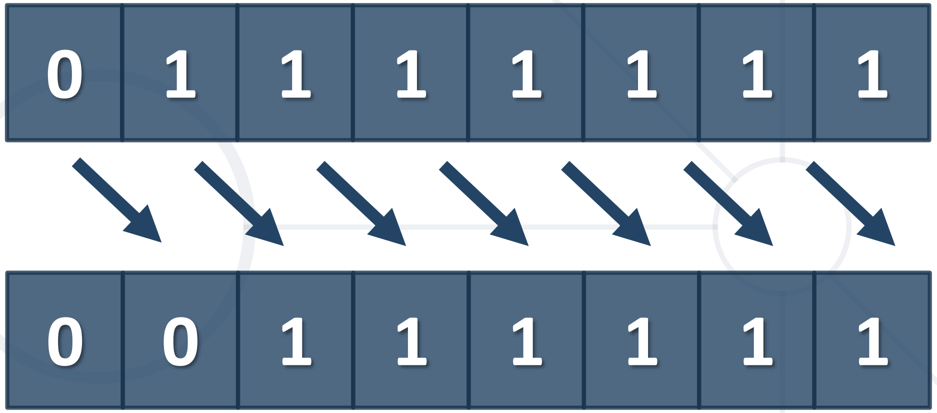


- Right Arithmetic Shift (>> operator)



Logical Shift

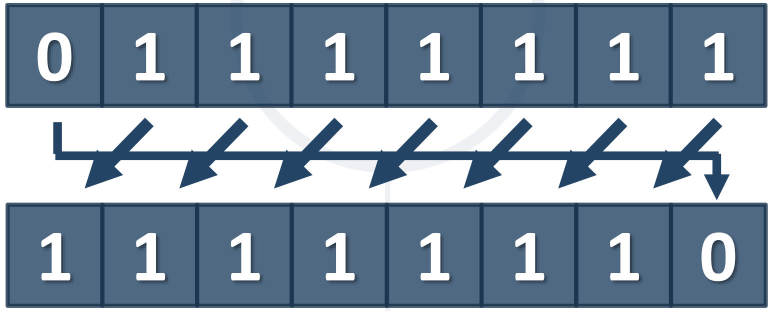
- Arithmetic and Logical Shifts are **very similar**
- The main difference is in the **right-shift**
  - Logical Right Shift **inserts 0** in the **MSB**, **instead of copying the sign bit**
  - Arithmetic Right Shift is ideal for **unsigned binary numbers**
  - Logical Right Shift is ideal for **signed binary numbers**



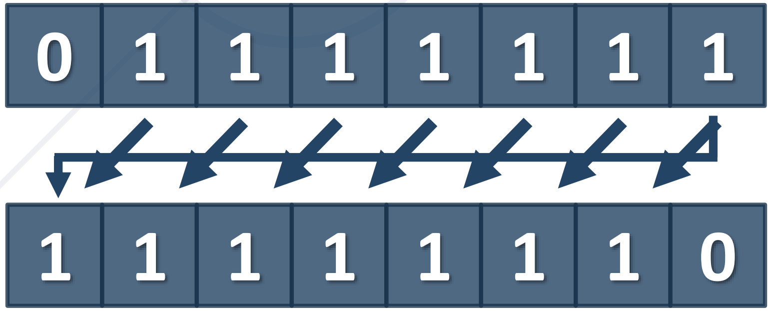
Circular Shift

- Bits are "rotated" as if
  the **left** and **right ends** were **joined**
- Operation is useful if it is necessary
  to **retain all existing bits**
- It is frequently used in **digital cryptography**

Left Circular Shift



Right Circular Shift



Simple Bitwise Operations

- How to get the bit at position **p** from a number **n**

```
p = 5                 //00000101
n = 125               //01111101
125 >> p              //00000011 = 3
3 & 1                 //1
```

- How to set the bit at position **p** to **0** or **1**

```
p = 5                 //00000101
n = 125               //01111101
mask = ~(1 << p)      //00100000
result = n & mask     //01011101
```

```
p = 5                 //00000101
n = 125               //01111101
mask = 1 << p         //00100000
result = n | mask     //01111101
```

Problem: First Bit

- Write a program that prints the bit at position 1 of a number

51	1	24	0
13	0	2	1

- Solution:

```
p = 1               //00000001
n = 51              //01111101
n = n >> p          //00011001 = 25
n & 1               //1
```

Live Exercises

Summary

- Computers store information using **bits**
- Representing data in different **numeral systems**
- Modifying bits using **bitwise operators** and **simple masks**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

XSsoftware

SBTech we know sports

telenor

SoftwareGroup doing it right

NETPEAK

SmartIT

Postbank Решения за твоето утре

SUPER HOSTING ®.BG

INDEAVR Serving the high achievers

INFRAGISTICS®

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License