Technical University of Lodz
Institute of Electronics

# Algorithms and Data Structures

## 5. Computer Memory

Łódź 2013

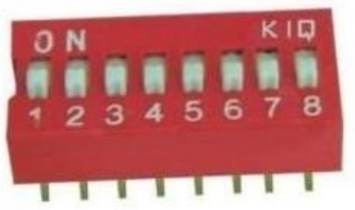
# Bits and Bytes

Computer memory can be categorized according to access speed and size

| Type | Access speed | Proximity to CPU | Size | Volatile |
|---|---|---|---|---|
| Register | Fastest | Inside | 10’s | Y |
| Caches | Very fast | Adjacent | MB | Y |
| RAM | Fast | Near | GB | Y |
| Hard disk | Slow | Far | TB | N |

- Computer memory can be thought of as a series of electronic on/off switches. Each on/off switch is called a **bit**. A group of 8 bits is called a **byte (B)**.
- Bits are interpreted as numbers by thinking of „off” as 0 and „on” as 1, for example:

| off | on | on | off | on | off | off | off |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |



## Decimal and Binary Numbers

The numbers that we most often use every day are **decimal** or **base ten**.

| 2 | 1 | 7 | 4 |
|---|---|---|---|
| 1000's | 100's | 10's | 1's |
| $10^3$ | $10^2$ | $10^1$ | $10^0$ |

**= 2*1000 + 1*100 + 7*10 + 4*1**

**Binary** or **base two** numbers use powers of two instead the powers of ten.

| 1 | 0 | 1 | 1 |
|---|---|---|---|
| 8's | 4's | 2's | 1's |
| $2^3$ | $2^2$ | $2^1$ | $2^0$ |

**= 1*8 + 0*4 + 1*2 + 1*1 = 11**

```
>>> bin(11)
>>> int(0b1011)
>>> bin(2174)   #Very long string!
```

## Hexadecimal Numbers

- **Hexadecimal** numbers are **base sixteen;** they use powers of 16 and the following digits:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **A** | **B** | **C** | **D** | **E** | **F** |

- Every four bits can be thought of as a single hexadecimal „digit”, since 4 bits can hold values between 0 an 15.

```
>>> hex(15)
>>> hex(255)
>>> hex(2**32)
>>> hex(-1024)
```

- **Unsigned** integers are stored in memory as binary numbers explained above.
- **Signed** integers are usually stored using **two's complement** representation whose details goes beyond this course.

## Memory Sizes

- Memory sizes are given in terms of bytes, with a prefix to indicate the scale. Common prefixes are borrowed from the metric system:

| Prefix | Value |
|---|---|
| kilo- | $10^3$ = 1000 = 1 thousand |
| mega- | $10^6$ = 1,000,000 = 1 million |
| giga- | $10^6$ = 1,000,000,000 = 1 billion |
| tera- | $10^9$ = 1,000,000,000,000 = 1 trillion |

- Because computer memory is organized in bits, powers of 2 are used to specify the memory size.

- These are close to metric equivalents but are not exactly the same.

| Prefix | Value |
|---|---|
| kilo- | $2^{10}$ = 1024 |
| mega- | $2^{20}$ = 1,048,576 |
| giga- | $2^{30}$ = 1,073,741,824 |
| tera- | $2^{40}$ = 1,099,511,627,776 |

## Floats

Predict what this code will do when it runs:

```
>>> x = 0
>>> while x != 1:
>>>     print (x)
>>>      x += 0.1
```

Then run it. Are you surprised? Is 0.1 the same as 1/10?

```
>>> x = 0.1
>>> print x
>>> print „%.32f” % (x)
```

Computers use **base two** representation and 0.1 has an infinitely repeated binary pattern, 1/10 = 0.0001100110011…

Similarly, 1/3 can not be stored exactly in a **base ten** system, 1/3 = 0.333333…

# Floats

Fractional values are converted to binary the same way as integers, except they use negative powers of 2, e.g.

| 0. | 1 | 0 | 1 | 1 |
|---|---|---|---|---|
| | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ |

= 1*1/2 + 0 *1/4 + 1*1/8 + 1*1/16 = 11/16

Unlike 1/16, the number 1/10 can not be expressed as a finite sum of negative powers of 2: 1/10 = 0.0001100110011…

Thus 1/10 can not be stored **exactly** as a floating point value in a binary computer.

**Conclusion: Use inequalities with floats.**

Because so many floating point values can not be stored exactly in memory, testing floats using either **==** or **!=** is risky. Use inequalities (**<, <=, >, >=**) whenever possible.

## Exercises

5.1. How to multiply a binary number by 2 without using any addition or multiplication? Multiply by 8? Divide by 2?

5.2. Give the largest binary value that can be stored in one unsigned byte along with its decimal and hexadecimal equivalents.

5.3. Give the largest binary value that can be stored in two unsigned byte along with its decimal and hexadecimal equivalents.

5.4. Determine the largest unsigned integer that can be stored in a 32-bit machine.

5.5. Determine the largest unsigned integer that can be stored in a 64-bit machine.

## Exercises

5.6. Write a short program **binhex.py** that prints a table of binary and hexadecimal values for the decimal integers 1 through 36. Hint: Use ***%ns,*** where *n* is the number of digits, for formatting the printed numbers.

5.7. Convert the following decimal values to binary. Indicate which can or cannot be stored precisely as floats.
a) 0.25, b) 0.375, c) 5.125, d) 10.4375, e) 0.3, f) 0.5

5.8. Convert the following binary values to decimal
a) 0.1, b) 0.0101, c) 0.111, d) 10100.01, e) 111.1011, f) 1000.10001

5.9. Fix the while loop code at slide 6 so that it terminates as apparently intended.

## Summary

1) Binary numbers are base two and work the same as decimal numbers except that they use powers of 2 instead of powers of 10.
2) Hexadecimal numbers are base 16. They provide shorter number representation – each 4 bits are replaced by a single symbol {0-9,A-F}.
3) If memory size is specified using powers of two, the indexes kilo-, mega- giga-, tera- and so on give different numbers than in the case of decimal specification.
4) Computers use base two representation of fractions, so many floating point numbers, such as 1/100, 10.5 are stored with errors.
5) Operations == and != should not be used with floating point variables; instead, inequalities are apropriate for floats.

## Literature

Brian Heinold, Introduction to Programming Using Python, Mount St. Mary's University, 2012 (*http://faculty.msmary.edu/heinold/python.html*).

Brad Dayley, Python Phrasebook: Essential Code and Commands, SAMS Publishing, 2007 (dostępne też tłumaczenie: B. Dayley, Python. Rozmówki, Helion, 2007).

Mark J. Johnson, A Concise Introduction to Programming in Python, CRC Press, 2012.