CGT 456

Access Modifiers

Logic

Access Modifiers

- private

- protected

- public

Access Modifiers: private

- A class's **private** variables and methods are not directly accessible to the class's clients. They are not accessible outside the class.

Access Modifiers: protected

- Using **protected** access offers an intermediate level of access between *public* and *private*.
- A base class's **protected** members can be accessed by members of that base class *and* by members of it's derived classes.

Access Modifiers: public

- The primary purpose of **public** methods is to present to the class's clients a view of the services the class provides (the class's public interface).

- Clients of the class need not be concerned with how the class accomplishes its tasks.

- **public** members are accessible wherever the application has a reference to an object of that class or one of its derived classes.

Access Modifiers (cont.)

○ Note that members of a class – for instance, methods and instance variables – do not need to be explicitly declared **private**.

○ If a class member is not declared with and access modifier, it has **private** access by default.

Logic

- ALU
  - Arithmetic Logic Unit
    - The brawn of the computer, the device that performs the arithmetic operations like addition and subtraction or logical operations like AND and OR.
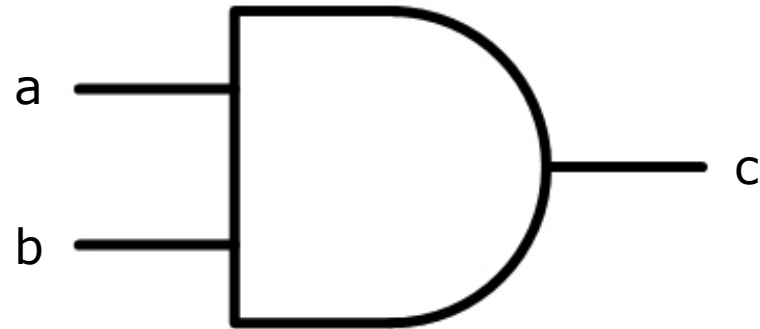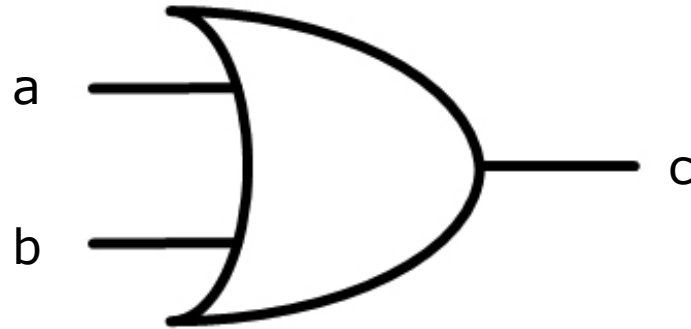
Logic

- Objectives
  - Calculate the decimal (base 10) numeric value of an 8-bit binary number (base 2).
  - Learn to both **Logical AND** two binary numbers together, as well as, **Logical OR** two binary numbers together.
  - Learn to **Bitwise AND**, as well as, **Bitwise OR** two 8-bit binary numbers together.
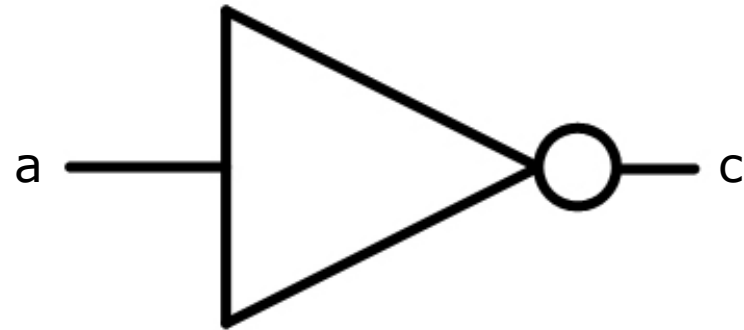 - Learn to use AND and OR gates.

AND Gate (c = a * b)

a	b	c=a*b
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate (c=a+b)

a	b	c=a+b
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate (c=ā)

a	c=ā
0	1
1	0

Binary Numbers

- Binary numbers are made up of 0 and 1.
- An example of a binary number would look like: 10010111
  - This is an example of an 8-bit binary number.
  - A 16-bit binary number would look like: 1001001011011001

How do we calculate it?

128	64	32	16	8	4	2	1

Binary numbers

- **Binary Number   Decimal Value**
- 10000000   =   128
- 10000001   =   129
- 00000000   =   0
- 00000001   =   1
- 00000010   =   2
- 00000011   =   3
- 00000100   =   4
- 00000101   =   5
- 11111111   =   255

- Thus, 0 to 255 offers 256 values within an 8-bit binary number.