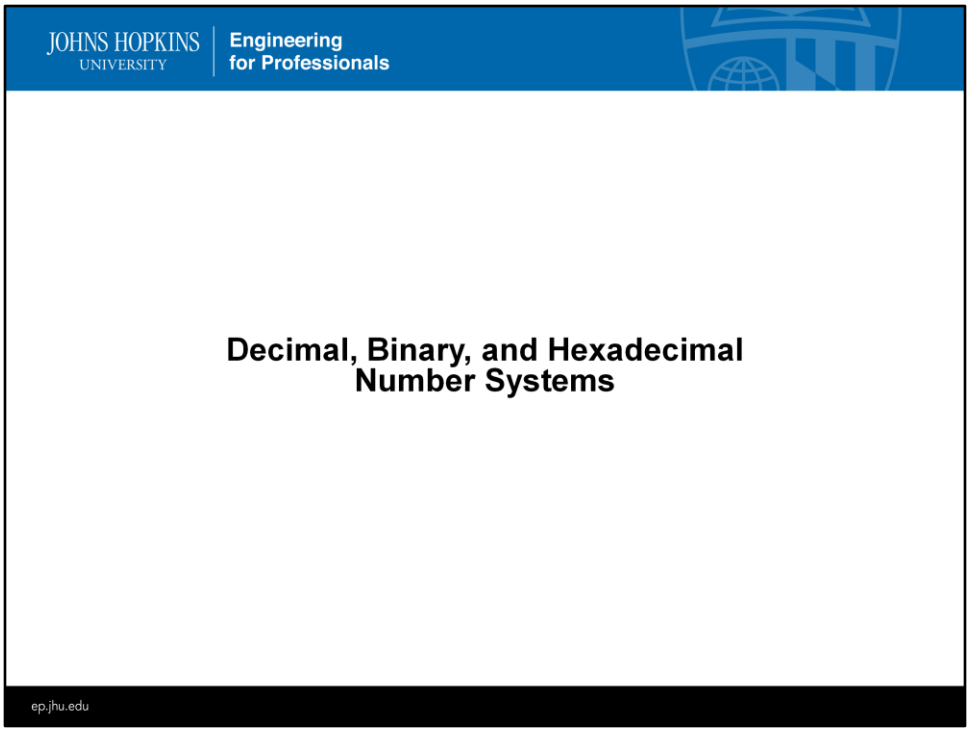


This presentation very briefly describes decimal, binary, and hexadecimal number systems and how to manually convert values between them. These knowledge and skill are useful in understanding and using Java bit-wise operators and debuggers.

So lets get to it!

Topics

Decimal number system

Binary number system

Hexadecimal (hex) number system

Converting values between decimal, binary, and hexadecimal

Negative integers

The topics of this presentation are the decimal number system (which you already know), binary number system, which is a base 2 number system, hexadecimal number system (AKA hex) which is a base 16 number system, and how to convert between values between them. We finish off with a short discussion on how negative integers are represented.

Decimal Number System

Base 10 system

- The number system you use every day
- Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

$10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$, etc.

You know all of this so lets move on….

The decimal number system is a base 10 number system. This is the number system you use every day. It has the 10 digits you know and you know the value of digit placements (1, 10, 100, etc.).

You know all of this but the concept of digit placement value will be used to help understand binary and hexadecimal number systems which we will go to now.

Binary Number System

Base 2

- Digits: 0, 1
- 1, 2, 4, 8, 16, etc.
- $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, etc.

Computer design based on binary system

Represented by a 0b or 0B prefix

Example:

- 10 = 0b1010 (0x1 + 1x2 + 0x4 + 1x8)

The binary number system is a base 2 system. There are only 2 symbols; 0 and 1 so each digit can have only two different values. The lowest-order digit can have only 2 different values, 0 and 1. Zero is zero and 1 is 1 so what happens with 2? Well in decimal if you have 9 in the lowest-order digit and you go to 10, you add 1 to the next higher order digit and set the lowest-order digit to 0 which give you 10. The same thing happens with binary except the value 2, which can not be represented in a single binary digit, adds 1 to the next higher digit and set the lowest-order digit to 0 which also gives you 10 but the 1 in the higher order digit is equal to 2 in the decimal system (and the hexadecimal system also which we will get to soon).

So in binary, the digits going from right to left represent the 1's place, then the 2's place, then the 4's place, etc. This takes some getting used to but with practice it becomes much easier.

Modern computer hardware design is based on binary values (switches on/off) making binary quite natural for computer systems (discussed in much more detail in your Computer Architecture course).

In Java, binary constants are represented by a 0b or 0B prefix. An example, of writing the binary constant 1010, or decimal 10, in Java is 0b1010. This binary represents 0x1 from the right-most digit . Then 1x2 from the second most-right digit, 0x4 from the third most-right digit, and finally 1x8 from the forth right-most digit. Add them together and you get 0x1 + 1x2 + 0x4 + 1x8 which equals 10 decimal.

Note just like in decimal, additional zeros to the left does not affect the value.

Hexadecimal Number System

Base 16

- Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- 1, 16, 256, 4096, 65,536, etc.
- $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4096$, etc.

Compact way of representing binary digits

- 1 hex digit = 4 binary digits
  - 10 = 0xA = 0b1010, 15 = 0xF = 0b1111

Represented by a 0x or 0X prefix

Example:

- 10 = 0xA, 100 = 0x64

The hexadecimal number system is a base 16 number system. It has 16 digit symbols. The normal decimal symbols 0 through 9 with the additional symbols A through F with A = 10 decimal, B = 11 decimal, through F = 15 decimal.

Since there are 16 digit symbols, each digit placement has a higher value than with decimal's 10 symbols. Going from right to left the first few digit values are 1, 16, 256, 4,098, 65,536 decimal respectively.

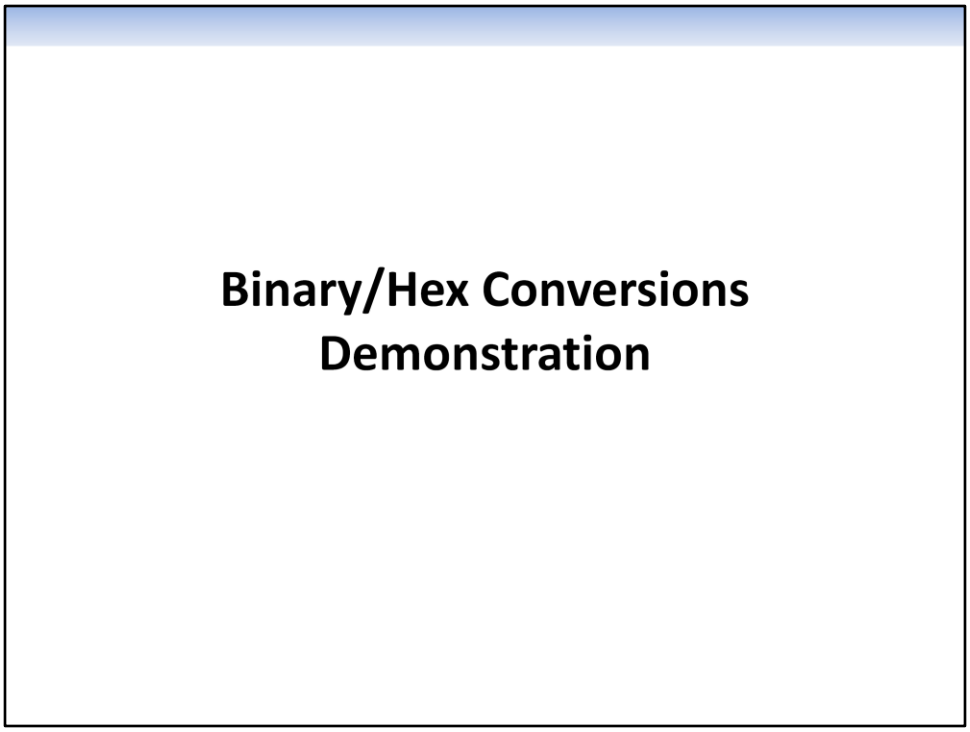
The main use of the hexadecimal number system is as a compact way of representing binary digits. One hexadecimal digit represents 4 binary digits. This makes sense $2^4 = 16 = 16^1$. So 10 decimal equals 1010 in binary which equals A in hexadecimal.

In Java, we use a 0x or 0X prefix for hex constants much like we use 0b or 0B for binary constants. Two examples shown here are 10 decimal is represented as 0xA in hex and 100 decimal equal to 0x64 in hex.

Decimal to Binary and Hex Conversions Demonstration

If you use binary and/or hex numbers in your program or are using a debugger to look at your variables at the bit-level, you will most likely need to know how to convert decimal to binary and/or hex.

Lets do some examples starting with decimal to binary and hex conversions then moving to binary to hex and hex to binary conversions.



Now on to our Binary/Hex Conversions Demonstration.

Java Code Binary/Hex Example

```
public class BinaryHexExamples
{
   public static void main(String[] args)
   {
      int dVar = 10;

      dVar = dVar + 0b1010; /* 0b1010 = 10 */
      System.out.println("dVar = " + dVar);
      dVar = dVar + 0x64; /* 0x64 = 100 */
      System.out.println("dVar = " + dVar);
      dVar = 10 + 0b1010 + 0xA; /* 10 + 10 + 10 */
      System.out.println("dVar = " + dVar);
   }
}
```

Results:

dVar = 20
dVar = 120
dVar = 30

Here are some examples of defining and using binary and hex constants in Java and using them in arithmetic calculations. Since these values are just different ways of representing numbers, you can mix the specification of these numeric constants in the same calculation. Lets look at this in a bit more detail.

Here we are adding the binary equivalent of 10 to the variable dVar. Note the 0b prefix.

Next we are printing the result of the above calculation.

In this line we are adding hex 64 which is decimal 100 to dVar. Note the 0x prefix for the hex constant.

Again we are printing out the results of dVar at this point.

In this line we are adding a decimal 10, a binary equivalent of decimal 10, and a hex equivalent of decimal 10 to the variable dVar. These constants are just different ways of representing the same number.

And again we are printing the results.

Note the results are displayed in decimal.

Negative Integers

- Computers use 2's complement format
  - Most significant bit (MSB) determines sign
    - 0 = positive, 1 = negative
  - Bit 31 for int, bit 63 for long.
- Changing positive to negative
  - Invert all bits than add 1
    - Example (8-bit) 0b0010 (2) -> 0b1101 + 0b1 = 0b1110 (-2)
      - 0b1110 (-2) -> 0b0001 + 0b1 = 0b0010 (2)
- Overflow – MSB changes when it should not
- Bit operations can change sign
  - Must be careful if you care about sign (later topic)

Negative integers in computers are represented in 2's complement format. In this format, the most significant bit (MSB) represents the sign of the integer. If the MSB is 0, the integer is positive. If the MSB is 1, the integer is negative.

Note the MSB is the sign no matter what the size of the integer-type. For example for an int it is bit 31. For long it is bit 63. For a byte it is bit 7.

To get the 2's complement (negative) of a positive number, first invert all the bits (change 0s to 1s and 1s to 0s) then add 1. The example here show how to convert the value 2 to -2 and back to 2. Note the inverting of the bits as the first step then adding 1. It works!

It is possible for calculations to result in a change in the MSB when it should not. For example adding two positive numbers or subtracting two negative numbers should produce a result with the same sign as the operands. However, if the numbers are big enough or small enough, the MSB could change as a result since the calculation resulted in a number which is beyond the range of the result's type. This does not produce an error (AKA exception).

Generally, bit operators do not care about the sign. They are working on each bit individually. So if you plan on using the results of a bit operation in a later numeric calculation, you must be careful about any sign changes. Bit operators are discussed in a separate presentation.

Another Binary/Hex Example

```
public class BinaryHexExamples
{
   public static void main(String[] args)
   {
   int dVar = 10;

   dVar = 0b10000000000000000000000000000000;
   System.out.println("dVar = " + dVar + " = 0x" +
      Integer.toHexString(dVar) + " = 0b" +
      Integer.toBinaryString(dVar));
   dVar = 0b01000000000000000000000000000000;
   System.out.println("dVar = " + dVar + " = 0x" +
      Integer.toHexString(dVar) + " = 0b" +
      Integer.toBinaryString(dVar));
   System.out.println("dVar = " + dVar + " = 0x" +
      Integer.toHexString(dVar) + " = 0b" +
      Integer.toBinaryString(dVar));
   }
}
```

Results:

```
dVar = -2147483648 = 0x80000000 = 0b10000000000000000000000000000000
dVar = 1073741824 = 0x40000000 = 0b01000000000000000000000000000000
```

Here are some examples of looking at decimal, binary, and hex values. Note how the numbers change from negative to positive and vise-versa by changing the value of the most significant bit.

The '+' operator in the println() method calls are string concatenation operations, not adding numbers. The "Integer.toHexString" and "Integer.toBinaryString" are just ways to print out the hex and binary representation of a number. Note for printing you have to add the 0x or 0b prefix if you want to display it in Java constant format.

Java Code Binary/Hex Example

```
public class BinaryHexExamples
{
public static void main(String[] args)
   {
   dVar = 0x7FFFFFFF;
   System.out.println("dVar = " + dVar + " = 0x" +
         Integer.toHexString(dVar) + " = 0b" +
         Integer.toBinaryString(dVar));
   dVar += 1;
   System.out.println("dVar = " + dVar + " = 0x" +
         Integer.toHexString(dVar) + " = 0b" +
         Integer.toBinaryString(dVar));
   }
}
```

Results:

dVar = 2147483647 = 0x7fffffff = 0b11111111111111111111111111111111

dVar = -2147483648 = 0x80000000 = 0b10000000000000000000000000000000

Here is an example of an arithmetic operation which changes the sign of the result from positive to negative when adding two positive numbers. The adding of one forced the MSB to change from a zero to a 1, making the result negative. Also notice no error is encountered.

References and Tutorials

Binary number system
(http://www.math.grin.edu/~rebelsky/Courses/152/97F/Readings/student-binary)

Hexadecimal number system
(http://homepage.smc.edu/morgan_david/cs40/hex-system.htm)

Here are a couple of Web resources which provide fairly easy explanations of binary and hexadecimal number systems. There are many more available.

That's it!

That's it!