OBJECT ORIENTED PROGRAMMING IN JAVA

Binary and Hexadecimal Numeration and Logical Operations

TABLE OF CONTENTS

1.1 Binary numeration, whole numbers
1.2 Binary fractions
1.3 Adding binary numerals
1.4 Hexadecimal
1.5 Logical operations

PURPOSE

A single bit is a piece of information that may have one of two states, on or off, high or low, 1 or 0. While a single bit may store only two patterns, a group of bits may store many more patterns. For example, eight bits, or a byte, may store one of 256 different bit patterns. The digits 0 and 1 are also the two digits used in the binary, or base 2, numeration system. By studying binary numeration, we can more easily talk about how information, such as numbers, letters, pictures and sounds, is stored in a computer. Additionally, the hexadecimal, or base 16, numeration system gives us a way of referring to a four – bit pattern with a single hexadecimal digit.

If you continue in computer science these topics will be more thoroughly discussed in a hardware course. In computer engineering, these are basic topics in circuits.

TO PREPARE

- Read Wu: Chapter 0 pages 2 - 7
- Read through this laboratory session

TO COMPLETE LAB 01

- Work in groups of two people. If there is an odd number of people in the lab, there may be a group of three people. You are expected to be on time so that appropriate groups may be formed.
- Each person in the group should keep his/her own answers to the questions. To get the most of this shared experience, you should individually answer a question and then compare the answer with your partner. If one of you does not understand a concept, the other should explain until the concept or process is understood. If you can explain, you truly do understand. If you are unsure of your answers, check with the lab tutor.

When you have finished this lab, see the lab tutor, who will give you an open – note, twenty – point quiz. This will be your grade for Lab 01
1.1 BINARY NUMERATION – WHOLE NUMBERS

In both class and the text book, you have seen that binary numerals use the digits 0 and 1 and are based on grouping in groups of two. A place value diagram for a binary numeral using eight digits is

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^7</td>
<td>2^6</td>
<td>2^5</td>
<td>2^4</td>
<td>2^3</td>
<td>2^2</td>
<td>2^1</td>
<td>2^0</td>
</tr>
</tbody>
</table>

Exercise 1.1 Count from 0 to 15 in binary, record the binary numerals

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Binary to Decimal:

Given a binary numeral, to find the decimal numeral, add the place values that contain a one.

\[1101 = \frac{1}{8} \frac{1}{4} \frac{1}{2} \frac{1}{1} = 8 + 4 + 1 = 13\]

\[11000011 = \frac{1}{128} \frac{1}{64} \frac{1}{32} \frac{1}{16} \frac{1}{8} \frac{1}{4} \frac{1}{2} \frac{1}{1} = 128 + 64 + 2 + 1 = 195\]

Decimal to Binary:

Given a decimal numeral, to determine the binary numeral, place a one in the place values that sum to the number, fill in the remaining places with a zero.

\[9 = \frac{1}{8} \frac{1}{4} \frac{1}{2} \frac{1}{1} = 8 + 1 \rightarrow 1001\]

\[140 = \frac{1}{128} \frac{1}{64} \frac{1}{32} \frac{1}{16} \frac{1}{8} \frac{1}{4} \frac{1}{2} \frac{1}{1} = 128 + 8 + 4 \rightarrow 10001100\]
Exercise 1.2 Complete the charts.

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>101101</td>
<td>17</td>
</tr>
<tr>
<td>10110</td>
<td>37</td>
</tr>
<tr>
<td>11111</td>
<td>127</td>
</tr>
<tr>
<td>1001001</td>
<td>198</td>
</tr>
</tbody>
</table>

Exercise 1.3 Answer these questions.

A. The largest value that can be stored in 4 bits is \( \underbrace{2^n} \) = \( \underbrace{10} \)
   The next number is \( \underbrace{2^n+1} \) = \( \underbrace{10} \)

B. The largest value that can be stored in 6 bits is \( \underbrace{2^n} \) = \( \underbrace{10} \)
   The next number is \( \underbrace{2^n+1} \) = \( \underbrace{10} \)

C. The largest value that can be stored in 8 bits is \( \underbrace{2^n} \) = \( \underbrace{10} \)
   The next number is \( \underbrace{2^n+1} \) = \( \underbrace{10} \)

Algorithms

An algorithm is a step by step process for solving a problem. For example, a recipe for baking a chocolate case is an algorithm. In this first semester of computer science, you will learn how to write a computer program, a set of instructions, or algorithm, that the computer follows in order to successfully complete a task.

An algorithm for converting a decimal whole number to binary involves division by two. The decimal numeration system groups in tens; the binary numeration system groups in twos. Read this example that uses the algorithm before the algorithm is actually stated.

The process of grouping by twos is accomplished by dividing by two. To convert the decimal numeral 13 to a binary numeral, repeatedly divide by 2.

1. \( \begin{array}{c} 13 \div 2 = 6 \text{ (groups of two)} \text{ with a remainder of 1} \\ 6 \div 2 = 3 \text{ (groups of four)} \text{ with a remainder of 0} \text{ (groups of two)} \\ 3 \div 2 = 1 \text{ (group of eight)} \text{ with a remainder of 1} \text{ (group of four)} \\ 1 \div 2 = 0 \text{ (groups of sixteen)} \text{ with a remainder of 1} \text{ (group of eight)} \end{array} \)

Notice that this process can be described with instructions that are repeated many times until some condition is met.

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
8 & 4 & 2 & 1
\end{array}
\]
The instructions that describe this process use this terminology from arithmetic:

\[
\begin{array}{c}
\text{quotient} \quad 6 \\
\text{divisor} \quad 2 \overline{13} \\
\text{dividend} \\
\text{remainder} \\
\text{12} \\
\text{1}
\end{array}
\]

The algorithm is written as English sentences and describes the process on the previous page. Instructions 1 and 2 are executed only once. Instructions a, b, c and d are executed repeatedly as long as the condition in 3 is true. We call this a loop.

**The algorithm to convert a whole decimal numeral into a binary numeral**

1. The dividend is the decimal numeral to be converted.
2. Initially, \( n \) is 0. (\( n \) is the exponent of 2, \( 2^n \) gives the place value.)
3. Repeat instructions a – d while the dividend is not equal to 0
   a. Divide the dividend by 2, finding a quotient and a remainder
   b. In the binary numeral, the digit at place value \( 2^n \) is the remainder
   c. Increase the value of \( n \) by one.
   d. The new value for the dividend is the quotient

**Example:** Convert \( 13_{10} \) to a binary numeral.

\[
\begin{array}{c|c|c|c|c}
\text{START} & \text{n = 3} & \text{n = 2} & \text{n = 1} & \text{n = 0} \\
\hline
\text{STOP} & 0 & 1 & 3 & 6 \\
\hline
2 \bmod 13 & 2 \bmod 3 & 2 \bmod 6 & 2 \bmod 13 \\
\hline
0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

**Exercise 1.4** Follow the algorithm to convert the indicated decimal numerals to binary. Clearly show your work.

A. Convert the decimal numeral 18 to binary
B. Convert the decimal numeral 51 to binary

1.2 BINARY FRACTIONS

In the decimal numeration system, the whole portion of a number is separated from the fractional portion with a decimal point. To the right of the point, the place values continue the pattern established to the left of the decimal point.

\[
\begin{array}{cccccccc}
100 & 10 & 1 & 1/10 & 1/100 & 1/1000 & 1/10000 \\
10^2 & 10^1 & 10^0 & 10^{-1} & 10^{-2} & 10^{-3} & 10^{-4}
\end{array}
\]

Similarly, in the binary numeration system, the whole portion of a number is separated from the fractional portion with a binary point.

\[
\begin{array}{cccccccc}
4 & 2 & 1 & 1/2 & 1/4 & 1/8 & 1/16 \\
2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4}
\end{array}
\]

**Binary fraction to decimal fraction**

This process is the same that is used with whole numbers.

0.1011₂ represents \( 1/2 + 1/8 + 1/16 = 11/16 \) or 0.6875

101.10₁₂ represents \( 4 + 1 + 1/2 + 1/8 = 5 5/8 \) or 5.625

**Exercise 1.5** Complete the chart, converting the binary numerals to decimal numerals.

<table>
<thead>
<tr>
<th>binary</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>10.01</td>
<td></td>
</tr>
<tr>
<td>1011.1001</td>
<td></td>
</tr>
</tbody>
</table>
Decimal fraction to binary fraction

In the decimal numeration system, some fractions such as 3/4, 7/8, and 2/5 can be expressed as terminating decimals 0.75, 0.875 and 0.4 respectively. However, other fractions, such as 1/6, 1/3 and 1/7, can only be expressed as non-terminating, repeating decimals.

\[
\begin{align*}
1/6 &= 0.1666 \\
1/3 &= 0.333 \\
1/7 &= 0.142857
\end{align*}
\]

This is also true when expressing fractions in base 2. The fractions 3/4 and 7/8 are expressed in binary as 0.11 and 0.111 respectively. But, though 2/5 is a terminating decimal fraction, it is a non-terminating, repeating binary fraction 0.01100110.

The algorithm in Section 1.1 converts a whole decimal numeral to a whole binary numeral by repeatedly dividing by 2. In contrast, the algorithm to convert a fractional decimal numeral to a fractional binary numeral uses repeated multiplication by 2.

The instructions that describe the process use this terminology from arithmetic:

- multiplicand
- multiplier
- product

Note that when converting a mixed numeral such as 6 2/5, the whole portion is converted with the first algorithm, and the fractional portion is converted using the second algorithm.

The algorithm to convert a decimal fraction into a binary fraction

1. The multiplicand is the fractional portion of the decimal numeral to be converted.
2. \( n = -1 \) (\( n \) is the exponent of 2, \( 2^n \) gives the place value)
3. Repeat instructions a – d while the multiplicand is not equal to 0 or more digits are needed to determine the repeated pattern
   a. Multiply the multiplicand by 2, finding a product
   b. The whole number portion of the product is the binary digit that belongs at place value \( 2^n \)
   c. The new value for the multiplicand is the fractional portion of the product
   d. Decrease the value of \( n \) by one.

Example Convert the decimal numeral 0.4 to binary.

```

\[
\begin{align*}
\text{START} & \quad 0.4 & \quad 0.8 & \quad 0.6 & \quad 0.2 \\
\times 2 & \quad 0.8 & \quad 1.6 & \quad 1.2 & \quad 0.4 \\
\text{Answer} & \quad 0.110
\end{align*}
\]

DONE The new multiplicand is 0.4, which is a repeat multiplicand. Therefore, the pattern is established.

Answer \( 0.4 = 0.0110_2 \)
Exercise 1.6 Follow the algorithm to convert the indicated decimal numerals to binary. Clearly show your work.
A. Using any method to convert the decimal numeral 3.25 to a terminating binary numeral.

B. Using the algorithm, convert the decimal numeral 0.7 to a repeating binary numeral.

1.3 ADDING BINARY NUMERALS
When adding decimal numerals there are 100 combinations of sums, assuming order is important. Since there are only two binary digits, there are only four possible addition combinations. These combinations and their sums are

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
+ 0 & + 1 & + 0 & + 1 \\
0 & 1 & 1 & 10 \rightarrow 2
\end{array}
\]

With these addition facts and your knowledge of how to add decimal numerals, you should be able to add using binary numerals. Add the columns from right to left. If the result is more than one digit record the 1’s digit and carry the remaining digit to the next column to the left. In the example, the related decimal addition problem is also recorded for verification purposes.

Example

\[
\begin{array}{cccc}
1 & 1 \\
1 0 1 0 1 \\
+ 1 1 1 0 0 \\
1 1 0 0 0 1
\end{array}
\]

carry bit

21
+ 28
49
**Exercise 1.7** Complete the following addition problems. Next to each problem, rewrite it using decimal numerals for verification.

A. \[ 10 + 10 \]  
B. \[ 101 + 11 \]  
C. \[ 1010 + 1111 \]  
D. \[ 1011.101 + 110.001 \]

### 1.4 HEXADECIMAL NUMERATION

Because information in the computer is based on the binary numeration system, it is easy for a human to make an error when transposing long bit streams such as 001011100011100110100111. If instead, each group of four bits is assigned a single symbol, errors are less likely to be made. A group of four bits can be arranged in \(2^4 = 16\) different bit patterns. Hexadecimal numeration, or base 16, uses sixteen different digits, just as a base 2 uses two digits and base 10 uses ten digits. By universal agreement the 16 digits are 0 – 9, and the first six letters of the alphabet, either a – f or A – F. The digit A represents ten, and F represents fifteen.

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

**Binary to hexadecimal**

Therefore, the bit pattern 001011100011100110101011 separated into 4 – bit groups 0010 1110 0011 1001 1010 1011 can be written as 2E39AB.
Similarly the six-bit binary numeral 110101 can be converted to binary by viewing it as

\[ 11 \ 0101 \text{ or } 0011 \ 0101, \text{ which is } 35_{16} \]

**Hexadecimal to binary**

To convert \( 5B_{16} \) to binary, write each hexadecimal digit as a four-bit binary numeral.

\[ 5B_{16} = 0101 \ 1011 \text{ If leading zeros are not needed, this is } 1011011_2 \]

**Hexadecimal to decimal**

As with all place value numeration systems, a place value chart for hexadecimal numerals is

\[
\begin{array}{cccccc}
256 & 16 & 1 & & & \\
16^2 & 16^1 & 16^0 & & & \\
\end{array}
\]

| \( 34_{16} \) | \( 3 \cdot 16 + 4 \cdot 1 = 52 \) | \( AC_{16} \) | \( 10 \cdot 16 + 12 \cdot 1 = 172 \) | \( F5_{16} \) | \( 15 \cdot 16 + 5 \cdot 1 = 245 \) |

**Counting**

To count in hexadecimal, you do not need to do any conversions. Just follow the same pattern that you follow when counting in base 10 and base 2.

In base 10, the largest two-digit number is 99. 99 is followed by 100

In base 2, the largest two-digit number is 11. 11 is followed by 100

In base 16, the largest two-digit number is FF. FF is followed by 100

In base 16, counting from \( 95_{16} \):

\[ 96 \ 97 \ 98 \ 99 \ 9A \ 9B \ 9C \ 9D \ 9E \ 9F \ A0 \ A1 \ A2 \ A3 \ A4 \ A5 \ A6 \ A7 \ A8 \ A9 \ AA \ AB \ AC \ AD \ AE \ AF \ B0 \]

**Exercise 1.8** Answer the following questions.

A. Rewrite the byte \( 11001001 \) in hex:

B. Rewrite the hex numeral \( 6D \) in binary:

C. For each of the hexadecimal numerals, write the next numeral. This is a counting question.

\[
\begin{array}{|c|c|}
\hline
N_{16} & \text{next } N_{16} \\
\hline
F & \text{next } N_{16} \\
C2 & \text{next } N_{16} \\
CB & \text{next } N_{16} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
N_{16} & \text{next } N_{16} \\
\hline
CF & \text{next } N_{16} \\
5F & \text{next } N_{16} \\
FFF & \text{next } N_{16} \\
\hline
\end{array}
\]
D. Convert the base 16 numerals to base 2 and base 10.

a. \(43_{16} = \quad \quad \quad = \quad \quad \quad 10\)

b. \(BD_{16} = \quad \quad \quad = \quad \quad \quad 10\)

E. Convert base 10 numerals to base 16 and base 2.

a. \(11011100_2 = \quad \quad \quad 16 = \quad \quad \quad 10\)

b. \(11101_2 = \quad \quad \quad 16 = \quad \quad \quad 10\)

F. Convert the base 10 numerals to base 16 and base 2.

c. \(45_{10} = \quad \quad \quad 16 = \quad \quad \quad 2\)

d. \(165_{10} = \quad \quad \quad 16 = \quad \quad \quad 2\)

1.5 LOGICAL OPERATIONS

Given two bits as operands, there are logical operations that result in a third bit. The operations AND, OR, XOR, and NOT are called Boolean operations after the nineteenth century logician George Boole. NOT is a unary operator, meaning that it is applied to a single operand that has one of two values, either true or false. The other operations are applied to two operands and are, therefore, binary operands. For the following examples, the operands are two statements P and Q, which are either true or false. Suppose

P: December 25 is Christmas Day    -    is true
Q: February 13 is Valentine’s Day  -    is false

AND

The statement

December 25 is Christmas Day AND February 13 is Valentine’s Day

is false. That is, P AND Q is false. In fact, P AND Q is true only if both P and Q are true.

Let the binary digit 0 represent false and 1 represent true.

When using binary digits, the rule for computing P AND Q translates into

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND 1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>False</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>True</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
**OR**

Now consider the logical operation OR. The statement

\[
\text{December 25 is Christmas Day } \text{ OR } \text{February 13 is Valentine's Day}
\]

is true. That is, \( P \text{ OR } Q \) is true. In fact, \( P \text{ OR } Q \) is only false if both \( P \) and \( Q \) are false.

When using binary digits, this translates into:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**XOR**

The XOR operation is read as exclusive or. The statement \( P \text{ XOR } Q \) is true if exactly one of \( P \) and \( Q \) is true. If both \( P \) and \( Q \) are true, \( P \text{ XOR } Q \) is false.

For example, the statement.

\[
\text{December 25 is Christmas Day } \text{ XOR } \text{February 13 is Valentine's Day}
\]

is true. But, the statement

\[
\text{December 25 is Christmas Day } \text{ XOR } \text{February 14 is Valentine's Day}
\]

is false.

When using binary digits, this translates into:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**NOT**

There is one more logical operation that is applied to only one Boolean operand and that is NOT. NOT negates the value of an expression. That is,

\[
\text{NOT (December 25 is Christmas Day)} \text{ or rather December 25 is not Christmas Day}
\]

is false. That is, if \( P \) is true, \( \text{NOT } P \) is false. And, if \( P \) is false, \( \text{NOT } P \) is true. When using binary digits, this translates into:

\[
\text{NOT } 0 \rightarrow 1 \quad \text{NOT } 1 \rightarrow 0
\]

Each of these operations is implemented in a computer with a device called a gate. An AND, OR, or XOR gate has two inputs and outputs one value, a NOT gate has one input and outputs one value. The circuitry of a computer is built using these gates as the lowest level building blocks.
Exercise 1.9

A. What is the result when AND is applied to these bytes?

\[ \begin{array}{c}
10101010 \\
\text{AND} \\
11110000
\end{array} \]

**Generalize:** If \( x \) is a bit, that is, either 0 or 1, then \( x \text{ AND } 0 \) is ______ and \( x \text{ AND } 1 \) is ______

B. What is the result when OR is applied to these bytes?

\[ \begin{array}{c}
10101010 \\
\text{OR} \\
11110000
\end{array} \]

**Generalize:** If \( x \) is a bit, that is, either 0 or 1, then \( x \text{ OR } 0 \) is ______ and \( x \text{ OR } 1 \) is ______

C. What is the result when XOR is applied to these bytes?

\[ \begin{array}{c}
10101010 \\
\text{XOR} \\
11110000
\end{array} \]

**Generalize:** If \( x \) is a bit, that is, either 0 or 1, then \( x \text{ XOR } 0 \) is ______ and \( x \text{ XOR } 1 \) is ______