## Hexadecimal – Base 16

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
<td>1E</td>
<td>1F</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
<td>2B</td>
<td>2C</td>
<td>2D</td>
<td>2E</td>
<td>2F</td>
<td>30</td>
</tr>
</tbody>
</table>
Octal – Base 8

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>
Conversion from base 10 → base 16 → base 2 and back → base 10

Example 1: $674_{10}$

\[
\begin{array}{c|c}
 16 & 674 \\
- 64 & 42 \\
\hline
 34 & 2 \\
\end{array}
\]

\[
\begin{array}{c|c}
 16 & 42 \\
- 32 & 32 \\
\hline
 10 & 0 \\
\end{array}
\]

Remainder 2 Remainder A Remaider 2

Read Remainders backwards

Therefore, $674_{10} = 2A2_{16} = 0010 1010 0010$
Example 1 Cont’d

$2A2_{16}$ to base 10:
$= 2 \times 16^0 + A \times 16^1 + 2 \times 16^2$
$= 674_{10}$

$0010\ 1010\ 0010_2$ to base 10:
$= 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3$
$+ 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 1 \times 2^7$
$+ 0 \times 2^8 + 1 \times 2^9 + 0 \times 2^{10} + 0 \times 2^{11}$
$= 674_{10}$
Conversion from base 10 to 2

This method known as “repeated division by 2.”

Example 2: Convert $37_{10}$ to base 2.

Stop dividing when the answer is 0. The remainders are read in reverse order to form the answer.

Therefore, $37_{10} = 100101 = 25_{16}$ (Note: don’t add 00 in front to make it 8-bits)
Example 2 Cont’d

Note: $37_{10} = 7 \times 10^0 + 3 \times 10^1 = 7 + 30 = 37$

Therefore,

$$100101_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 0 \times 2^7$$

$$= 1 + 0 + 4 + 0 + 0 + 32 + 0 + 0$$

$$= 37_{10}$$
Example 3

Convert $110.111_2$ to the decimal system

Solution:

$$110.111_2 = (1 \times 2^{-3}) + (1 \times 2^{-2}) + (1 \times 2^{-1})$$
$$+ (0 \times 2^0) + (1 \times 2^1) + (1 \times 2^2)$$
$$= 0.125 + 0.25 + 0.5 + 2 + 4$$
$$= 6.875_{10}$$
Example 4

Convert $2A2.5_{16}$ to the decimal system

Solution:

$2A2.5 = (5 \times 16^{-1}) + (2 \times 16^{0}) + (A \times 16^{1})$
+ $(2 \times 16^{2})$

$= 0.3125 + 2 + 160 + 512$

$= 674.3125$
Example 5

Convert $0.75_{10}$ to Binary

Solution:

$0.75 \times 2 = 1.50 \Rightarrow 1$

$0.50 \times 2 = 1.00 \Rightarrow 1$

Therefore, $0.75_{10} = 0.11_2$
Example 6

Convert $0.3017_{10}$ to Binary

Solution:

$0.3017 \times 2 = 0.6034 \rightarrow 0$

$0.6034 \times 2 = 1.2068 \rightarrow 1$

$0.2068 \times 2 = 0.4136 \rightarrow 0$

$0.4136 \times 2 = 0.8272 \rightarrow 0$

Therefore, $0.3017_{10} = 0.0100 \ldots$
The circuitry required for computers to add binary numbers is relatively simple. However, since circuitry to subtract binary numbers is very difficult to design and build, another subtraction method would be desirable. The method is subtraction by addition of the 1’s or 2’s complements of the number. Subtraction by the addition of 1’s complement requires much more hardware to implement.

Most computers on the market perform subtraction by the addition of 2’s complement. Therefore we will concentrate only on subtraction by 2’s complement.

Binary number = 1011 1001
1’s complement = 0100 0110
2’s complement = 0100 0111

When subtraction is performed the answer can be positive or negative, depending upon the relative value of the minuend and subtrahend. We must, therefore, in some way account for the sign of the answer.
The most significant bit (MSB) is used as the sign bit. If the MSB is a ‘1’, the number represented by the remaining 7 bits is assumed to be negative and is represented by its 2’s complement. If the MSB is a ‘0’, the number represented in the remaining seven bits is positive and is equal to the binary equivalent of these seven bits.

Example 1: $17_{10} - 6_{10} = (11_{16} - 6_{16})$

$0000\ 0110 = 6_{16}$
$1111\ 1001 = 1$’s complement of $6_{16}$
$1111\ 1010 = 2$’s complement of $6_{16}$

$0001\ 0001 = 11_{16}$
$1111\ 1010 = 6_{16}$ in 2’s complement

-------------
$1\ 0000\ 1011$

$1\ 0000\ 1011 = +\ 0B_{16} = +\ 11_{10}$
Example 2: \[ 6_{10} - 17_{10} = (6_{16} - 11_{16}) \]

0001 0001 = 11_{16}
1110 1110 = 1’s complement of 11_{16}
1110 1111 = 2’s complement of 11_{16}

0000 0110 = 6_{16}
1110 1111 = 11_{16} in 2’s complement

\[ \begin{array}{c}
0000 1010 \\
\hline
1111 0101
\end{array} \]

1111 0101 = - (0000 1010 + 1) = - 0000 1011 = - 0B_{16} or - 11_{10}

Note: C and N bits will be set. C is set to 1 because the absolute value of subtraction (17) is larger than the absolute value of the minuend (6)
Since bit 7 serves as the sign bit. A ‘0’ in bit 7 indicates a positive number. Thus the maximum positive number an 8-bit processor can represent in 2’s complement is 0111 1111₂ or 7F₁₆ or 127₁₀.

Likewise a ‘1’ in bit 7 indicates a negative number. The minimum negative number that an 8-bit processor can represent in 2’s complement is 80₁₆ (1000 0000). In binary 2’s complement of 1000 0000 = – (0111 1111 + 1) = - (1000 0000) = - 80₁₆ = - 128₁₀

It is possible to (add) or subtract two numbers and get invalid answer, especially when the answer is out of the range of –128 ↔ +127

This situation is referred to as 2’s complement overflow. If a subtraction (or addition) is performed and 2’s complement overflow occurs the V bit will be set to a ‘1’ otherwise it will be cleared.
Example 3: \(-85_{10} - 90_{10} = -175_{10}\)

\[85_{10} = 55_{16} = 0101 \ 0101_2\]
\[1010 \ 1010 = 1\text{'s comp. of } 55_{16}\]
\[1010 \ 1011 = 2\text{'s comp. of } 55_{16}\]

\[90_{10} = 5A_{16} = 0101 \ 1010_2\]
\[1010 \ 0101 = 1\text{'s comp. of } 5A_{16}\]
\[1010 \ 0110 = 2\text{'s comp. of } 5A_{16}\]

\[1010 \ 1011 = (2\text{'s comp. of } 55_{16})\]
\[1010 \ 0110 = (2\text{'s comp. of } 5A_{16})\]

\[1 \ 0101 \ 0001 = +51_{16} = +81_{10}\]

V bit will be set.
C bit will be set.
Example 4: \(4D_{16} + 66_{16}\)

\[
\begin{align*}
01001101 & \quad \text{positive} \\
01100110 & \quad \text{positive} \\
\hline
10110011 & \quad \text{negative}
\end{align*}
\]

V bit is set.
Logic Gates
### “AND” Function

#### Symbol

![Symbol diagram](image)

#### Equation

\[ C = A \cdot B \]

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Output C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### “OR” Function

#### Input Table

<table>
<thead>
<tr>
<th>Input</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Symbol

```
+        
A -- B --> C
```

#### Equation

\[ C = A + B \]
### “Exclusive OR” Function (XOR)

**Symbol**

\[ A \oplus B \]

**Equation**

\[ C = A \oplus B \]

<table>
<thead>
<tr>
<th>Input ( A )</th>
<th>Input ( B )</th>
<th>Output ( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
“Not” Function (Inverter)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Symbol

Equation

\[ C = \overline{A} \]
### “NOR” Function

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Output C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### “NAND” Function

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Output C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
QUESTIONS???