TABLE OF CONTENT – HANDOUT - by Sigrid Zuňiga

BINARY AND HEXADECIMAL

Introduction to Binary Numbers

Starting scenario:

You are setting up a network and have to get licenses for various software. You just received the figures of the budget from the accounting dept. = $9,979 total, and heard that you need 219 licenses. The price of each license is at least $40. You have to instantly calculate if you have enough and how much money you have available to spend *per* license. Your computer is currently down and you have no calculator available to you, what do you need to do?

Use estimation: $9,979 → converts to: $10,000
219 → converts to: 200
$50 per license.

What have we done? Rounding to **Place Values**:
Look at the place values: **9** 9 7 9
1000s 100s 10s 1st place

We rounded here to the *1000*s place: estimated between 10,000 and 9,000 (difference is 1,000)

Look at the **place values**: **2** 1 9
100s 10s 1st place

We rounded to the 100s place: between 200=300: 219 is closer to 200, thus round down to 200.

A Review of the Decimal System – Positional Weight

Anthropologists think that counting on 10 fingers resulted in the decimal system.
The base is 10, ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

You count using the decimal number system, you get the next number by adding. How do we count past 9:

```
 9
1 +
10
```

← You must write a 0 and carry a 1 to the next column

The decimal system is a **positional** system: The **place value of a digit** is determined by its position. I.e.: The digits 1 by itself are worth 1, but the digit 1 followed by a 0 is worth 10:

10^{th} place	1^{st} place	
	0	= 1
1	0	= 10

We have used the decimal system for so long that when we see a number like "**123**", we don't think about the value **123;** rather you generate mental image how many items this value represents. However, in reality, *the number **123** represents*:

$1*10^2$	$2*10^1$	$2*10^0$	
100 +	20 +	3	=**123**

So: Each digit appearing to the left of the decimal point represents a value between zero and nine times an increasing power of 10:
Another example: **6019 =**

($6x10^3$)	**($0x10^2$)**	**($1x10^{1)}$**	**($9x10^{0)}$**	**=**
(6 x 1000) +	(0 x 100) +	(1 x 10) +	(9 x 1)	=
6000 +	0 +	10 +	9	= **6019_{10}**

When we use the decimal system, we can also say: We are using **base 10 (written as $\#_{10}$)**. Besides base 10, we can also use other bases, i.e. base 2 (written as $\#_2$), base 3 ($\#_3$), base 4 … base16. We will discuss **base 2** (= binary system) and **base 16** (hexadecimal) (written as $\#_{16}$) in more detail. Choosing a base is a matter of choice. Before going into the base the computer uses let's just look at some base options:

Some different base options and their available number options															
Base **2**	0	1													
Base 3	0	1	2												
Base 5	0	1	2	3	4										
Base **8**	0	1	2	3	4	5	6	7							
Base **10**	0	1	2	3	4	5	6	7	8	9					
Base 12	0	1	2	3	4	5	6	7	8	9	A(=10)	B(=11)			
Base **16**	0	1	2	3	4	5	6	7	8	9	A(=10)	B(=11)	C(=12)	D(=13)	E(=14)	F(=15)

Digital Systems

How does the computer manipulate numbers and letters? It makes use of electrical circuits, so we need some type of conversion. People use symbols to write as 2, G, 8, etc. A computer is an electronic device, so it cannot write down the data it works with. A computer needs electrical signals to represent data: How it represents data depends on whether it is an **analog** device or a **digital** device.

- **Analog** device: Operates continuously varying data, i.e. a *dimmer* switch, *cassette* tape, records
- **Digital** device: Uses separate digits: <u>on</u> and <u>off</u> are the only possibilities.

With digital electronic circuits, information is first converted into a group of pulses. This code consists of a series of voltages. The early digital systems constructed in the 1950s made use of a decimal code that used *ten levels of voltages*, with each of these voltages corresponding to one of the ten digits in the decimal number system (0=0 Volt, 1 = 1 V, 2=2 V, 3 = 3 V, up to 9 = 9 V). The circuits that had to manage these decimal codes, however, were very complex and had to sense the difference between all ten voltage levels. This complexity led to inaccuracy since some circuits would periodically confuse one voltage level for a different voltage level. *The solution to this problem of circuit complexity and inaccuracy was solved by adopting **a <u>two-state</u>** system instead of a **<u>ten-state</u>** system.*
All information or data is encoded in HIGH and LOW voltages, in which the HIGH voltages are called "**1s**" (ones) and the LOW voltages are called "**0s**" (zeros). The computer represents values using two voltage levels (0v and +5V) and employs the **binary** numbering system. The binary system = **base 2**.

Bits, Bytes and the Binary System

Each number or letter is represented by a series of electrical signals. Each 1 or 0 that represents data is referred to as a **bit**. In most computers, a string (series) of 8 bits is called a **byte** and represents a **character**: number, letter, or symbol.

The binary numbering system works just like the decimal numbering system, with two exceptions: in the **binary** number system, there are only two digits: 0 and 1 (rather than 0-9) and binary uses powers of two instead of powers of ten. The numeral 2 cannot be used in the binary system, so when you run out of options after writing 1, instead of writing 2 you would carry over the 1 and write "1" in front of the 0: =10.

To understand how the binary system works, think about the **decimal** system first. Counting with the **binary number** system is similar to counting with the decimal system, but you can only use two digits, 1 and 0. You begin counting with 1. To get to the next number, you must add one. However, in the binary system 1 + 1 cannot equal 2 because there is no 2 digit. So, just as in the decimal system, when you run out of digits, you carry a 1 to the next column: 1

1 +

10

So:

2nd place	1st place	
	0	= 1
1	0	= 2

To Summarize:

The binary number system is positional, but it only uses 2 digits, 1 and 0.
*In binary, the digit 1 by itself is worth **one**, the digit 1 followed by a 0 is worth **two**, not **ten** as in the decimal system.*

Look again at the table:

Some different base options and their available number options												
Base **2**	0	1	**10**	**11**	100	101	110	111	1000	1001	1010	1011 etc.
Base 10	0	1	2	3	4	5	6	7	8	9	**10**	**11** etc.

Convert from binary to decimals:

The second difference with the decimal system is that binary uses the powers of *two* instead of the powers of *ten*. For each "1" in the binary string, add in 2^n where "n" is the zero-based position of the binary digit. We will use the example **10001100_2**

1. We need to first set up a table of binary place values, starting with the number 1, which we put in the right most column. Since 8 bits make up one byte, we use 8-place values: As mentioned above, the place values are the powers of 2.
2. Using pencil and paper, create the following table:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
↑multiply							
Add ——→							

3. Let's use our table by converting the **binary number 10001100 in a decimal number**:

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1	
1	**0**	**0**	**0**	**1**	**1**	**0**	**0**	
128 +	0	0	0	8 +	4 +	0	0	**=140**

4. We write the binary digit in each column: We multiply the place value by each binary Digit: this produces a decimal value for each column:

×	×	×	×	×	×	×	×	
0	0	0	0	0	0	0	0 = 00000000	Binary
0	0	0	0	0	0	0	0 = 0	Decimal

In the binary number system the place values are powers of 2:

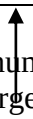
Power of 2:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Place value:	128	64	32	16	8	4	2	1

5. The final step: Find the sum of the decimal values for each column. (=140).
6. Now practice for yourself, using this method the binary number: **10101110** (=174).

Conversion from decimal into binary numbers:

To convert decimal to binary is slightly more difficult. You must find those powers of two which, when added together, produce the decimal result. The easiest method is to work from the larger power of two down to 2^0**. Consider the decimal value 73:**

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1



- Write the number 73
- Find the largest binary place value that you can subtract from the decimal number, in this case 64

Perform the subtraction: 73
64 -
9 remainder.

Place a 1 in the 64 column

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1
	1						

Remainder is 8: What is the largest place value you can subtract from the remainder? = 8: Place a 1 in column 8: (Another remainder left: 9-8 = 1)

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1
	1			**1**			

The remainder of 1 can only be placed in the first column: Now there are no remainders left.

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1
	1			**1**			**1**

- In the rest of the columns: add 0 in each column.
- In the bottom row multiply 64 x 1 = 64; 8 x 1 = 8; 1 x 1 = 1
- Add: 64 + 8 + 1 = 73 to check your answer.

2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1= 2	2^0= 1	
0	**1**	**0**	**0**	**1**	**0**	**0**	**1**	
0	64	0	0	8	4	9	1	**=73**

Summarized steps:

- Find the largest place value that will fit into a decimal number
- Subtract that place value from the decimal number
- Put a 1 in the column for that place value
- Repeat the process until there is no remainder.

Try out different numbers: i.e. decimal value **1359** to practice.

- 2^{10} =1024, 2^{11}=2048. So 1024 is the largest power of two less than 1359. Subtract 1024 from 1359 and begin the binary value on the left with a "1" digit. Binary = "1", decimal result is 1359 – 1024 = 335
- The next lower power of two (2^9 = 512) is greater than the result from above, so add a "0" to the end of the binary string. Binary = "10", decimal result is still 335
- The next lower power of two is 256 (2^8). Subtract this from 335 and add a "1" digit to the end of the binary number. Binary ="101". Decimal result is 79.
- 128 (2^7) is greater than 79, so tack a "0" to the end of the binary string. Binary = "1010", decimal result remains 79.
- The next lower power of two (2^6 = 64) is less than 79, so subtract 64 and append a "1" to the end of the binary string. Binary = "10101", Decimal result is 15.
- 15 is less than the next power of two (2^5 = 32) so simply add a "0" to the end of the binary string. Binary = "1010100", Decimal result is still 15
- 16 (2^4) is greater than the remainder so far, so append a "0" to the end of the binary string. Binary = "1010100" , decimal result is 15
- 2^3 (eight) is less than 15, so stick another "1" digit on the end of the binary string. Binary = "10101001", decimal result is 7.
- 2^2 is less than seven, so subtract four from seven and append another one to the binary string. Binary = "101010011", decimal result is 3.
- 2^1 is less than 3, so append a one to the end of the binary string and subtract 2 from the decimal value. Binary = "1010100111", Decimal result is now 1.
- Finally, the decimal result is one, which is 2^0, so add a final "1" to the end of the binary string. The final binary result is "**10101001111**"

2^{10}=1024	2^9=512	2^8=256	2^7= 128	2^6= 64	2^5= 32	2^4 = 16	2^3= 8	2^2= 4	2^1=2	2^0=1
1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**

From Binary (base 2) to Hexadecimal (Base 16)

In the United States, most people separate every three digits with a comma to make larger numbers easier to read. For example, 1,023,435,208 is much easier to read and comprehend than 1023435208. A similar convention is adopted for binary numbers. We can separate each group of four binary bits with a space. For example, the binary value 1010111110110010 will be written as 1010 1111 1011 0010

Computers generally work with specific numbers of bits: Common collections on the 80x86 microprocessor are:

1. **Single bits** = smallest unit of data on a binary computer, capable of representing only two values, zero and one
2. **Nibbles** = groups of four bits. With a nibble, we can represent up to 16 distinct values.
3. **Bytes** = group of eight bits and it is the smallest addressable data item. Bits in a byte are normally numbered from zero to seven. One byte represents a character, i.e. "A"
4. **Words** = groups of 16 bits. The bits in a word start from zero on up to fifteen. A word exactly contains two bytes.

The Hexadecimal Number System

A big problem with the binary system is verbosity. To represent the value 1359_{10} (see example above) requires 11 binary digits. The decimal version requires only 4 digits, and thus represents numbers much more compactly than does the binary numbering system. The hexadecimal (hex) system is used as a sort of shorthand. Furthermore, although we can convert between decimal and binary, this is not an easy task. The hexadecimal (base 16) numbering system solves this problem. Hexadecimal numbers offer two features: they are very compact, and it is simple to convert them to binary and vice versa.

Each hexadecimal digit can represent one of sixteen values between 0 and 15_{10}: **0-9** and beyond 9, hex makes use of the letters **A**$_{16}$(=10_{10}), **B**$_{16}$(=11_{10}), **C**$_{16}$(=12_{10}), **D**$_{16}$(=13_{10}), **E**$_{16}$=(14_{10}) and **F**$_{16}$(=15_{10}). Since the base of a hex number is 16, each hex digit to the left of the hex point represents some value times a successive power of 16:

Convert Hexadecimal Numbers to Decimal Numbers

For example, the number **1234**$_{16}$ =

$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$	
(1 * 4096)	(2 * 256)	(3 * 16)	(4 * 1)	=
4096 +	**512 +**	**48 +**	**4**	**= 4660$_{10}$**

Convert decimal numbers to hexadecimal numbers: 425_{10}

- The largest power of sixteen (256) is subtracted once from 425 and therefore a 1 is placed in the 256 column, leaving a remainder of 169.
- The next largest power of 16 is 16 which can be subtracted ten times from 169. Therefore the hexadecimal equivalent of ten, which is A, is placed in the sixteen's column, leaving a remainder of 9.
- Since nine 1s can be subtracted from the remainder of 9, the units column is advanced nine times, giving us our final hexadecimal result of **1A9**

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$	
(1 * 256)	(10 * 16)	(9 * 1)	=
1	**A**	**9**	**= 1A9**

Another example:
Convert decimal **4525** to its hexadecimal equivalent:

=4525 – 4096 = 429, 429 – 256 = 173,
173 – 16 – 16 – 16 – 16 –16 –16 – 16 – 16 –1 6 – 16 = 13,
13 – 1 –1 –1 –1 –1 –1 –1 – 1 – 1 – 1 – 1 – 1 – 1 = 0

4096	256	16	1
1	**1**	**A**	**D**

Convert between binary numbers and hexadecimal numbers:

As mentioned earlier, hexadecimal is used as a short-hand for representing large groups of binary digits. To illustrate this we can represent a 16-bit binary word in four hexadecimal digits. A four bit binary word can have any value from 0_{10} (0000_2) to 15_{10} (1111_2). To convert from hexadecimal to binary, we simply do the opposite. Since each hexadecimal digit represents four binary digits, a 4 digit hexadecimal number will convert to a 16 bit binary word.

Convert hexadecimal 2BF9 to its binary equivalent:

Use the table below for your conversion. The table provides all information to convert any hexadecimal number into a binary number and vice versa.

Hexadecimal Number	2	B	F	9
Binary equivalent	0010	1011	1111	1001

Convert binary 110011100001 to its hexadecimal equivalent:

Binary number	1100	1110	0001
Hexadecimal equivalent	C	E	1

Table for Binary/Hex/Octal conversion

Binary	Hexadecimal	Binary	Octal
0000	0	000	0
0001	1	001	1
0010	2	010	2
0011	3	011	3
0100	4	100	4
0101	5	101	5
0110	6	110	6
0111	7	111	7
1000	8		
1001	9		
1010	A		
1011	B		
1100	C		
1101	D		
1110	E		
1111	F		