Computer Numbering Systems

There are three different numbering systems used with computers: decimal, binary and hexadecimal. The decimal numbering system is the normal human numbering system we use every day. The binary numbering system is the system used by computers since computers can only represent 2 different states (on or off/ 1 or 0). The hexadecimal numbering system is a shorthand method for representing large/long binary numbers since humans have difficulty with binary.

All three numbering systems have different bases and universe of digits. The universe of digits is the set of characters which represent a valid digit in the numbering system. In the decimal numbering system the base is 10 and the universe of digits is 0 – 9. For example, the token *&1* does not represent a valid decimal number because the character *&* is not within the universe of digits for decimal numbers. Note they are ordered from 0 to 9 as well (e.g. 0<1<2<3<4….<9).

Humans are information specialists. We are so good that we aren't even aware we are applying an algorithm every time we work with numbers. As a basis, we are going to continually review what we know about the decimal number system and apply it to Binary as we use the same algorithms and thinking regardless of the numbering system's base (i.e. decimal is base 10, binary is base 2 and Hexadecimal is base 16).

Binary Introduction

Why Binary?

Computers need to use binary because they can only detect 2 states, on and off, just like a light switch. This is represented with a positive voltage (on) or no voltage (off). (It is too difficult and expensive to design a computer to detect 10 different or varying voltages.)

Learning Binary – Relax

The best advice I can give is don't try to hard. Humans are information processing specialists. Our brains know the algorithms so well that our brains have developed short cuts. Our thinking is so advanced we don't even know what we know. So just relax as this can be quite easy if rely on our knowledge of the decimal number system rather than our rote usage of it. I recommend you read each sentence/paragraph and pause (Take a few breaths and reread the passage to reflect upon the knowledge presented.)

Binary terminology

Binary is a number system based on switches called bits. (**B**inary Dig**its**) First, some terminology:

Each bit is switch.
Four bits is called a nibble.
Eight bits is called a byte.

If the switch is on, the value of the switch is **1**. If the switch off, the value of the switch is **0**.

Why people find binary "different"

There are few things about binary numbers that make them different to the plain vanilla decimal (base10) numbers that surround us every day.

1. Binary only uses two of the digits that we are used to - **0** and **1**
2. We need to learn to read them right to left (like decimal the smallest or least significant bit is on the right)

So, step 1 to binary success is to accept point 1 and just accept point 2!

Counting in Decimal

Let us revisit how we count in decimal. We begin with 0 and add 1 to 0 which gives us 1. To find the next number we add the integer 1 to it or 1 + 1 = 2. We continue this until we run out of numbers (e.g. 9 + 1 = ?) when we find we have to carry a one to the next positional place (e.g. 9 + 1 = 10). If we analyze this number we find that we have 1 x 10 + 0 x 1's. We now see that we have place values associated with the powers of 10 (decimal means 10). Our counting algorithm is:

```
Counting Algorithm

    Start:

      Begin with 0

      Add 1 to present number to obtain the next or successive number

      If the present number is the last number in the sequence (Decimal = 9)
              ➔We start over at 0 carrying or adding the one to the next higher
              decimal place

      Goto Start:
```

Decimal Positional Table

10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
10000000	1000000	100000	10000	1000	100	10	1

Counting in Binary

We use the same algorithm as above for decimal however we are now in base 2 and can only use 0 and 1. Note the table of binary powers and positional values below.

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

```
Start:
    Begin with 0
    Add 1 to present number to obtain the next or successive number
    If the present number is 1 (we only have 0 and 1)
            ➔We start over at 0 carrying the one to the next decimal place
    Goto Start:
```

0	
1	(0+1 ➔ 1)
10	(There is no 2 so we carry the 1 and begin with 0 in the 1’s place)
11	(10 + 1 ➔ 11)
100	(11 + 1 ➔ record a 0 in the 1’s place and carry the 1 to the 2’s place noting this results in a 0 and causes 1 to be carried and added to the 4’s place… this is analogous to decimal 99 + 1 where 1 + 9 is 10 => carry the 1 to the 10’s place, 1 + 9 is 10 carry the 1 to the 100’s place)
101	(100 + 1 ➔ 101)
110	(101 + 1 ➔ record a 0 and carry the 1)
111	(110 + 1 ➔ 111)
1000	(111 + 1 results in the 1 being carried 3 times)

Determining a Decimal Number's Value

To start, let's review what we know (or have forgotten) in a familiar system – decimal.

Everybody knows what value of decimal 975 is. We don't even think about it as (9 x 100) + (7 x 10) + (5 x 1) ➔ 975. Some of you probably just added 5 with out multiplying it by 1 skipping that portion of the algorithm as we consider this common sense (e.g. (9 x 100) + (7 x 10) + 5 ➔ 975. We know that we multiply the place holding values (e.g. 9, 7 and 5) by their positional values (e.g. 100, 10 and 1) which are powers of 10 (i.e. decimal)

What if I asked you to create an algorithm to separate out the individual digits of 975? Some of you would look at the powers of 10 and their positional values. We would find the Largest Common Denominator (LCD) and divide the number by this LCD to extract the individual values. We would then apply the same algorithm to the remainder.

10^3	10^2	10^1	10^0
1000s	**100s**	**10s**	**1s**

Let's see:
There are no 1000s in 975
There are 9 100s in 975
There are 7 10s in 975
There are 5 1's in 975

How did we do this? (This is algorithm development)

We found the Largest Common Denominator (LCD) in base 10.
Divided the number by this LCD
Saved the remainder and used it as the new number.
975/100 ➔ 9 remainder 75 (LCD is 100)
75/10 ➔ 7 remainder of 5 (LCD is 10)
5/1 ➔ 5 (LCD is 1)

With a number like 907 we do the following to strip out the individual digits
907/100 ➔ 9 remainder 07 (LCD is 100)
07/10 ➔ 0 remainder of 7 (LCD is 10) but me make sure we record the 0
7/1 ➔ 7 (LCD is 1)

Understanding binary's numeric value in base 10

In these examples we will work with 8-bit binary (1 byte). You can work with larger or smaller units but the byte is the most commonly used one.

OK, let's take a look at a byte of binary

01101101

OK, so the first thing to remember is to read the byte from right to left. Let's make things easier by creating a table:

Switch On = 1 Off = 0	0	1	1	0	1	1	0	1
Bit Position	8	7	6	5	4	3	2	1
Bit Value in decimal	128	64	32	16	8	4	2	1
Positional Powers	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Bit position is simply the position of the bit from the right - the bit furthest to the right is bit 1 and the bit on the left is number 8.

Decimal value of a bit is dependant on the position of the bit in the byte. The decimal value of the first bit is 1, the second 2 and so on. The eighth bit has a decimal value of 128. These correspond to powers of 2. (In decimal we use powers of 10 e.g. 1's place, 10's place, 100's place, 1000,s place… etc.) Converting a byte into decimal is simply adding up the decimal values of each on bit in the byte.

Highlight all the on bits and ignore the off bits:

Switch On = 1 Off = 0	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**
Bit Position	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Bit Value in decimal	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
01101101 =		**64 +**	**32 +**		**8 +**	**4 +**		**1**

Bit 8 =	0
Bit 7 =	64
Bit 6 =	32
Bit 5 =	0
Bit 4 =	8
Bit 3 =	4
Bit 2 =	0
Bit 1 =	1

Take out the zeros and add together all the decimal values of the on bits to the the value of the byte in decimal => 01101101 = 64 + 32 + 8 + 4 + 1 = 109 … Easy!

Let's try another example.

10111001

First, create a table (do this until you are comfortable working with binary numbers).

Switch On = 1 Off = 0	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
Bit Position	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Bit Value in decimal	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Now highlight all the on bits and ignore the off bits:

Switch On = 1 Off = 0	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
Bit Position	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Bit Value in decimal	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
10111001 =	**128 +**		**32 +**	**16 +**	**8 +**			**1**

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

Take out the zeros and add together all the decimal values of the on bits to the the value of the byte in decimal => 10111001 = 128 + 32 + 16 + 8 + 1 = 185

Binary to Decimal (in bytes)

1. **Write out positional powers of 2 in a table (e.g. 128, 64, 32, 16, 8, 4, 2, 1)**
2. **Fill in the table above with 1's & 0's**
3. **Add the positional values wherever there is a 1**

Here are a few for you to practice on! (i.e. what are the decimal values of the following?)

1. 11110111
2. 00010010
3. 10101010
4. 11111111
5. 11001100

Solutions – Highlight from here on and change the font to black. The font is presently white thus invisible. To do this hold down the shift key and scroll down the page using the down arrow key, then go up to Format and Font and change the color.

Food for thought – You are a system administrator and you suspect two of your employees are involved with corporate espionage. You are asked by the CEO to read their emails. Would you have caught this simple method of hiding messages?

Decimal to Binary Conversion

There are 2 different algorithms to convert from decimal to binary. The easier one in my opinion is by inspection. Please review the segment above on **Determining a Decimal Number's Value** above as we will use the same approach or algorithm.

Let's look at a simple example – Decimal 19
(Could also be represented as 0d 19 => 0d before the 19 means number is in decimal)

I have copied the binary positional chart below for convenience along with an extra row to hold the binary equivalent of decimal 19 (i.e. 0d 19) Also by convention we will represent in 8 bits or 1 byte.

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

6. By inspection we see the LCD of 19 is 16 therefore I put a one in the 16s place and zeros in the 128, 64 and 32 places.
7. 19 – 16 = 3 thus I continue looking for the next LCD for the number 3 assigning zeros to positions greater than 3.
8. I find the LCD for 3 is 2 and therefore write down a 1
9. 3 – 2 = 1 thus I continue looking for the next LCD for the number 1 which is obviously one and we are done.

Let's check our work, we have 0b 00010011
(preceding 00010011 with 0b means it is in binary)

```
    0 x 128 =   0
+   0 x 64 =    0
+   0 x 32 =    0
+   1 x 16 =   16
+   0 x 8 =     0
+   0 x 4 =     0
+   1 x 2 =     2
+   1 x 2 =     1
```

Another detailed example is given in Appendix A

Binary to Decimal (in bytes)

1. **Write out positional powers of 2 in a table (e.g. 128, 64, 32, 16, 8, 4, 2, 1)**
2. **Fill in the table above with 1's & 0's**
3. **Add the positional values wherever there is a 1**

Hexadecimal

Basically, the only reason we have hexadecimal is that humans are terrible using binary (long strings of 1s and 0s). Hexadecimal provides a quick and convenient way to express binary numbers as they are represented on computers (i.e. bytes). Hexadecimal has 16 digits. The decimal system only has 10 digits thus we must use 6 other characters for the additional Hexadecimal digits. We use A, B, C, C, E, F to represent decimal 10, 11, 12, 13, 14, 15 respectively. See the correspondence in the table below.

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

Also, compare the positional tables of Binary, Hexadecimal and Decimal

Binary Positional Table

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

Hexadecimal Positional Table
(note 2^4 **=** 16^1 and 2^8 **=** 16^2 and …)

16^3	16^2	16^1	16^0
4096	256	16	1

Decimal Positional Table

10^3	10^2	10^1	10^0
1000	100	10	1

Binary & Hexadecimal Conversion

Once I have either Binary or Hexadecimal I create a table to convert every 4 bits into their equivalent Hex digit or vice versa (i.e. 1 hex digit into 4 bits).

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example 01100101 (from above)

1. Separate each byte into 2 groups of 4 bits

    ➔ 0110 0101 (also represented as 0b 0110 0101 which indicates it is binary)

2. Find the corresponding hex digit in table above for each 4 bits.

    0110 ➔ 6
    0101 ➔ 5
    ➔ Hex 65 (also represented as 0x65)

Let's check/verify our Hex 65 result. Our Algorithm is:

To convert each numbering system to decimal, each bit/hex digit is multiplied by its positional value (an increasing power of the base). The power of the base increases by one for every digit moved to the left.
Our number is:

Power of 16	16^2	16^1	16^0
Positional Value	256	16	1
Our number	0	6	5

Thus we have:

```
  0 x 256     ➔ 0
+ 6 x 16      ➔ 96
+ 5 x 1       ➔ 5
                101 decimal
```

Now we have 3 numbering systems and need a way to disambiguate them. We do this by preceding the number with 0d for decimal, 0b of binary and 0x for hexadecimal

Convert the following to Binary and Hexadecimal

> **0d 45**
>
> **0d 131**
>
> **0d 257 (Note that we always represent binary/hexadecimal numbers in a full Byte.**

Convert the following to Decimal

> **0x 37**
>
> **0x AB**
>
> **0b 10111011**

Solutions are whited out above (Highlight the area and change the font color)

Appendix A
Decimal to Binary Conversion

Algorithm Decimal to Another Base by Inspection from above

Start:

1. ***Find*** *the* ***largest common denominator(LCD)*** *in desired base for* ***decimal number*** *(Use the Positional Tables below as necessary)*

2. Note ***place 0s*** *in* ***result positions where number cannot be divided (positional place is not LCD)***

3. ***Divide*** *this* ***decimal number*** *by the* ***largest common denominator*** *and* ***record result*** *in corresponding position*

4. ***Save remainder***

5. ***Take remainder*** *and* ***use*** *as new* ***decimal number***

*6****. If*** *new* ***decimal number is not 0***

Then Goto Start

Binary Positional Table

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

Hexadecimal Positional Table

16^3	16^2	16^1	16^0
4096	256	16	1

Decimal Positional Table

10^3	10^2	10^1	10^0
1000	100	10	1

Convert the following to Binary and Hexadecimal

0d 101 (0d means decimal 101 – (One Hundred One))

I created a table to help me with the powers of 2 as well as keep track of my progress and data storage. I will step through the conversion of 0d 101 using the ➔ to indicate deduced result of each step.

Positions	128	64	32	16	8	4	2	1
Result								
Remainder								
Num to Subtract								

*1. **Find** the **largest common denominator** in desired base for **decimal number** (or **remainder**)*

I compare the value of 101 with the powers of 2

➔ Largest common denominator for 101 is 64 since 128 > 101 > 64

2. Place 0s in result positions where number cannot be divided

101 cannot be divided (Integer Division) by 128 so I place a 0 in the 128 bit position

Positions	128	64	32	16	8	4	2	1
Result	0							
Remainder								
Num to Subtract								

3. Divide this decimal number by the largest common denominator and record result moving left to right

101 / 64 ➔ result is 1 in 64 place

Positions	128	64	32	16	8	4	2	1
Result	0	1						
Remainder		37						
Num to Subtract		64						

4. Subtract this number from the decimal number (if necessary) and record remainder (in binary this is already done)

101 – 64 ➔ 37

5. Take remainder and use as new decimal number

Use 37 as new decimal number

*6**. If** new **decimal number is not 0***

Then Goto Start (1)

My new decimal number is now 37 and I am here:

Positions	128	64	32	16	8	4	2	1
Result	0	1						
Remainder		37						
Num to Subtract		64						

*1. **Find** the **largest common denominator** in desired base for **decimal number** (or **remainder**)*

➔ 32 since 37 > 32

*2. Note **place 0s** in **result positions where number cannot be divided***

➔ not necessary

*3. **Divide** this **decimal number** by the **largest common denominator** and **record result** in corresponding position*

37 / 32 ➔ 1 with remainder of 5

Positions	128	64	32	16	8	4	2	1
Result	0	1	1					
Remainder		37	5					
Num to Subtract		64	32					

*4. **Subtract** this **number** (positional value) **from** the **decimal number** and **record remainder** (in binary this is already done as the remainder is this number)*

➔ Already done in Binary

*5. **Take remainder** and **use** as new **decimal number***

new decimal number is 5

*6**. If** new **decimal number is not 0***

Then Goto Start

➔ it is not 0 , it is 5 goto start

From previous page I Start with Decimal number is 5

Positions	128	64	32	16	8	4	2	1
Result	0	1	1					
Remainder		37	5					
Num to Subtract		64	32					

*1. **Find** the **largest common denominator** in desired base for **decimal number** (or **remainder**) (Use the Positional Tables below as necessary)*

➔ 4 since 16 > 8 > 5 > 4

*2. Note **place 0s** in **result positions where number cannot be divided***

➔ 0s placed in 16s and 8s place

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0			
Remainder		37	5					
Num to Subtract		64	32					

*3. **Divide** this **decimal number** by the **largest common denominator** and **record result** in corresponding position*

5 / 4 ➔ 1 with remainder 1

4. **Subtract** this **number** (positional value) **from** the **decimal number** and **record remainder** (in binary this is already done as the remainder is this number)

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0	1		
Remainder		37	5			1		
Num to Subtract		64	32			4		

*5. **Take remainder** and **use** as new **decimal number***

➔ 1

*6**. If** new **decimal number is not 0***
Then Goto Start

Decimal number is 1 thus not 0 so Goto Start

Continuing on - we Start with decimal number = 1

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0	1		
Remainder		37	5			1		
Num to Subtract		64	32			4		

*1. **Find** the **largest common denominator** in desired base for **decimal number** (or **remainder**) (Use the Positional Tables below as necessary)*

➔ 1 because 2 > 1

*2. Note **place 0s** in **result positions where number cannot be divided***

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0	1	0	
Remainder		37	5			1		
Num to Subtract		64	32			4		

*3. **Divide** this **decimal number** by the **largest common denominator** and **record result** in corresponding position*

1 / 1 = 1 with remainder = 0

*4. **Subtract** this **number** (positional value) **from** the **decimal number** and **record remainder** (in binary this is already done as the remainder is this number)*

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0	1	0	1
Remainder		37	5			1		0
Num to Subtract		64	32			4		

*5. **Take remainder** and **use** as new **decimal number***

new decimal number is 0

*6**. If** new **decimal number is not 0***
Then Goto Start

It is 0 thus we are done.

Let’s check our work and using the result:

Positions	128	64	32	16	8	4	2	1
Result	0	1	1	0	0	1	0	1
Remainder		37	5			1		0
Num to Subtract		64	32			4		

Our Algorithm is:
To convert each numbering system to decimal, each bit/hex digit is multiplied by its positional value (an increasing power of the base). The power of the base increases by one for every digit moved to the left.

By inspection I see the binary number 01100101 I use the table above to determine the decimal value.

0 x 128	➔ 0
+1 x 64	➔ 64
+1 x 32	➔ 32
+0 x 16	➔ 0
+0 x 8	➔ 0
+1 x 4	➔ 4
+0 x 2	➔ 0
+1 x 1	➔ 1

Now binary is easy and by inspection I only look at the positions that have the value 1 since 0 times anything is 0 thus: 64 + 32 + 4 + 1

Binary & Hexadecimal

Once I have either Binary or Hexadecimal I create a table to convert every 4 bits into their equivalent Hex digit or vice versa (i.e. 1 hex digit into 4 bits)

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Example 01100101 (from above)

1. Separate number into 2 groups of 4 bits

   ➔ 0110 0101 (also represented as 0b0110 0101)

2. Find the corresponding hex digit in table above for each 4 bits.

   0110 ➔ 6
   0101 ➔ 5
   ➔ Hex 65 (also represented as 0x65)

Lastly, let’s check/verify our Hex 65 result. Our Algorithm is:

To convert each numbering system to decimal, each bit/hex digit is multiplied by its positional value (an increasing power of the base). The power of the base increases by one for every digit moved to the left.

Our number is:

Power of 16	16^2	16^1	16^0
Positional Value	256	16	1
Our number	0	6	5

Thus we have:

$$\begin{array}{lll} & 0 \times 256 & \Rightarrow 0 \\ + & 6 \times 16 & \Rightarrow 96 \\ \underline{+} & \underline{5 \times 1} & \underline{\Rightarrow 5} \\ & & 101 \text{ decimal} \end{array}$$

Conversions Algorithms in a Nutshell

Decimal to Binary (in bytes – max value 255)

1. **Write out positional powers of 2 in a table (e.g. 128, 64, 32, 16, 8, 4, 2, 1)**
2. **Find the largest common denominator (LCD moving left to right)**
3. **Write a 1 in the LCD**
4. **Subtract the LCD and continue moving L - > R (step 2) repeating until done**

Binary to Decimal (in bytes)

4. **Write out positional powers of 2 in a table (e.g. 128, 64, 32, 16, 8, 4, 2, 1)**
5. **Fill in the table above with 1's & 0's**
6. **Add the positional values wherever there is a 1**

Binary to Hexadecimal and vice versa

1. **Create a binary hexadecimal/binary conversion table by:**
    a. **Count to 15 in binary**
    b. **Count to 15 in hexadecimal alongside the binary**
2. **Convert every 4 bits into hexadecimal or every hexadecimal number into 4 bits usint the**