# Chapter 5
# Data Representation

## Objectives

Upon completion of this chapter you will be able to:
- Describe the binary and hexadecimal number systems,
- Given a number in one of base 2, 10, or 16, convert that number to its equivalent in the other bases,
- Perform addition with binary and hexadecimal numbers, and
- Demonstrate how a character is stored internally using the `EBCDIC` or `ASCII` collating sequences.

## Introduction

We've done about as much assembler as we can without learning more about how data is represented internally. In this chapter we will discuss the binary and hexadecimal number systems. But first, we need to take a brief look at our decimal number system....

## A Closer Look at Our Decimal Number System

Time now for some very basic arithmetic review. This may take you back a while! Consider the number `1956`. We know this is a valid decimal number, but why? It is a valid decimal number because it is composed of valid digits for the decimal, or base 10, number system, specifically, 0 through 9 inclusive. That is, the base 10 number system consists of ten digits. Recall that 10 is not a digit, but is a valid decimal number consisting of the digits 1 and 0.

Consider again the number `1956`. We know this number to be one thousand nine hundred fifty six, but why? Let's look at this more closely. Recall:

```
1956
::::
::::... 6 x    1 =    6
:::.... 5 x   10 =   50
::..... 9 x  100 =  900
:...... 1 x 1000 = 1000
                   1956
```

Furthermore, recall:

$$\begin{aligned} 1 &= 10^0 \\ 10 &= 10^1 \\ 100 &= 10^2 \\ 1000 &= 10^3 \end{aligned}$$

And so we can rewrite the above as:

```
1956
::::
::::... 6 x  10^0 =    6
:::.... 5 x  10^1 =   50
::..... 9 x  10^2 =  900
:...... 1 x  10^3 = 1000
                    1956
```

So we see that for a base 10 number, each digit is multiplied by a successive power of ten.

What is the highest decimal number with four digits? Answer: `9999`. We know this because 9 is the highest digit allowed in the base 10 number system, and to get the highest decimal number with four digits, we simply use this digit in all four positions.

Similarly, the lowest decimal number with four digits is `0000`, because 0 is the lowest digit allowed in the base 10 number system, and to get the lowest decimal number with four digits, we simply use this digit in all four positions.

So in summary,

- The base 10 system has ten digits. They are 0 through 9 inclusive.
- The value of a number is found by multiplying each digit by a successive power of ten and summing these products.
- To get the highest number in a fixed number of digits, use the highest digit (9) in all positions. Similarly, to get the lowest number in a fixed number of digits, use the lowest digit (0) in all positions.

*These same principles will be used as we discuss other number systems!*

## The Binary Number System

The binary number system is also known as the base 2 number system. Just as the base 10 system has 10 digits, the base 2 system has 2 digits. They are 0 and 1. Just as 10 was not a valid digit in the base 10 system, 2 is not a valid digit in the base 2 system.

So we see that a binary number is one composed of zeros and ones only. `10110` is a valid binary number, but `10120` is not valid. Note that `10110` is *also* a valid decimal number. When the base is not obvious, we often use a subscript to indicate the base. For example, $10110_2$ vs. $10110_{10}$.

The binary number system is the system used by all computers to represent data. The reason for doing so is that a binary digit is easily represented electronically, such as by the presence or absence of current.

Each binary digit is referred to as a **bit**. Bit is an abbreviation for binary digit.

## Binary-to-Decimal Conversion

Using the same technique as was used with the base 10 numbering system, the value of a binary number can be found by multiplying each digit by a successive power of two and summing these products. For example,

```
10110
:::::
:::::... 0 x 2^0 = 0 x  1 =  0
::::.... 1 x 2^1 = 1 x  2 =  2
:::..... 1 x 2^2 = 1 x  4 =  4
::...... 0 x 2^3 = 0 x  8 =  0
:....... 1 x 2^4 = 1 x 16 = 16
                            22
```

So we see that $10110_2 = 22_{10}$

## You Try It...

1. $1101_2$ = _____10
2. $10010_2$ = _____10
3. $110111_2$ = _____10

## The EBCDIC and ASCII Collating Sequences

A **byte** is a collection of bits. There are usually 8 bits in a byte, an exception being some early teletype devices which had 6 or 7 bits in a byte. For our purposes, *we will always assume 8 bits per byte*.

A natural question then is, "What is the maximum decimal value representable by a single byte?" Recall from our discussion of the base 10 number system, to get the highest number in a fixed number of digits, use the highest digit in all positions. The highest digit in the base 2 number system is 1, so the highest base 2 number in 8 digits is `11111111`.

```
11111111
::::::::
::::::::... 1 x 2^0 = 1 x   1 =   1
:::::::.... 1 x 2^1 = 1 x   2 =   2
::::::..... 1 x 2^2 = 1 x   4 =   4
:::::...... 1 x 2^3 = 1 x   8 =   8
::::....... 1 x 2^4 = 1 x  16 =  16
:::........ 1 x 2^5 = 1 x  32 =  32
::......... 1 x 2^6 = 1 x  64 =  64
:.......... 1 x 2^7 = 1 x 128 = 128
                                255
```

So we see that the maximum decimal number representable by 8 bits or 1 byte is 255. Similarly, the lowest decimal number representable by 8 bits or 1 byte is 0 ($00000000_2$).

There are, therefore, a total of 256 possible values (0 through 255) representable by one byte.

This fact is important, but still does not answer the question, "How does a computer, which processes bits (which are ON or OFF only) represent data?" The answer to the question lies in the use of codes.

You may have played with secret codes when you were younger. For example, you may have used the following:

| A | B | C | D | E | F | G | H | I | ... | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 25 | 26 |

Using this code, if I need to send the word CABBAGE, I would send 3-1-2-2-1-7-5 instead, and no one would know what I was saying. (This is commonly referred to as a code, but is actually what is known as a substitution cipher. A code is really something different.)

In order to represent data, similar coding schemes are used to assign a decimal value (from 0 to 255 inclusive) to each character. There are two such coding schemes. The first, **EBCDIC**, or Extended Binary Coded Decimal Interchange Code, is the code used by IBM mainframes (and therefore PC/370). A portion of that code is shown here:

```
A..193  G..199  M..212  S..226  Y..232  4..244
B..194  H..200  N..213  T..227  Z..233  5..245
C..195  I..201  O..214  U..228  0..240  6..246
D..196  J..209  P..215  V..229  1..241  7..247
E..197  K..210  Q..216  W..230  2..242  8..248
F..198  L..211  R..217  X..231  3..243  9..249
```

**ASCII**, or American Standard Code for Information Interchange, is the code used by non-IBM mainframes, and all PCs. A portion of that code is shown here:

```
A..65   G..71   M..77   S..83   Y..89   4..52
B..66   H..72   N..78   T..84   Z..90   5..53
C..67   I..73   O..79   U..85   0..48   6..54
D..68   J..74   P..80   V..86   1..49   7..55
E..69   K..75   Q..81   W..87   2..50   8..56
F..70   L..76   R..82   X..88   3..51   9..57
```

These codes determine the **collating sequence**, or sort sequence, of the characters. Note that when using EBCDIC, letters come before numbers, while in ASCII, numbers come before letters. Similarly, when using EBCDIC, lower case letters come before upper case letters, while in ASCII, upper case letters come before lower case letters.

## Decimal-to-Binary Conversion

To convert a decimal number to its binary equivalent, subtract the highest power of two which is less than or equal to the number. Continue until the difference is equal to zero. Those powers of two which were subtracted will be 1's in the binary number. All other digits will be 0. For example:

```
  55
- 32      32 = 2^5 --> the fifth* digit is 1
  23
- 16      16 = 2^4 --> the fourth digit is 1
   7
-  4       4 = 2^2 --> the second digit is 1
   3
-  2       2 = 2^1 --> the first digit is 1
   1
-  1       1 = 2^0 --> the zeroth digit is 1
   0

Answer:   Binary number is 110111
```

Another example:

```
 200
-128     128 = 2^7 --> the seventh digit is 1
  72
- 64      64 = 2^6 --> the sixth digit is 1
   8
-  8       8 = 2^3 --> the third digit is 1
   0

Answer:   Binary number is 11001000
```

* *Note that the references to digit placement are relative to zero, that is fifth digit is actually the sixth digit from the right.*

## You Try It...

4. $20_{10}$ = _____$_2$
5. $87_{10}$ = _____$_2$
6. $146_{10}$ = _____$_2$

## The Hexadecimal Number System

Working with binary numbers can be very tedious: the numbers are long and are prone to errors. The hexadecimal, or base 16, number system is commonly used in place of binary numbers.

Just as the base 10 system has 10 digits, and the base 2 system has 2 digits, the base 16 system has 16 digits. They range in value from 0 to 15 inclusive. But 15 cannot be a digit, as it would be confused with a 1 and a 5. So to avoid such confusion, we use the letters A-F for the digits 10-15. The complete hexadecimal digits are as follows:

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

To be effective in assembler, you must commit this table to memory.

Note that each hexadecimal digit uses four bits. Therefore, two hexadecimal digits are necessary to indicate the value of a single byte (eight bits).

As before, there is the possibility of confusion over the base of a number. The number `2A7F` can only be a hexadecimal number, but `10110` is a valid binary, decimal, and hexadecimal number! When the base is not obvious, we use a subscript of 16 to indicate hexadecimal, for example, $10110_{16}$.

## Hexadecimal-to-Decimal Conversion

Using the same technique as was used with the base 10 and base 2 numbering systems, the value of a hexadecimal number can be found by multiplying each digit by a successive power of sixteen and summing these products. For example,

```
2A7F
::::
::::..... F x 16^0 = 15 x    1 =    15
:::..... 7 x 16^1 =  7 x   16 =   112
::..... A x 16^2 = 10 x  256 =  2560
:..... 2 x 16^3 =  2 x 4096 =  8192
                                10879
```

So we see that $2A7F_{16} = 10879_{10}$

---

## You Try It...

7. $6D_{16}$ = _____10
8. $A5_{16}$ = _____10
9. $2FE_{16}$ = _____10

## Decimal-to-Hexadecimal Conversion

The process of converting a decimal number to its hexadecimal equivalent is similar to the process of converting a decimal number to its binary equivalent: subtract the highest multiple of a power of 16 which is less than or equal to the number. Continue until the difference is equal to zero. Those multiples of powers of 16 which were subtracted become the digits in the hexadecimal number. All other digits will be 0. For example, to find the hexadecimal equivalent to 748:

```
 748
-512     512 =  2 x 16^2 --> the second digit is 2
 236
-224     224 = 14 x 16^1 --> the first digit is E (for 14)
  12
- 12      12 =  1 x 16^0 --> the zeroth digit is C (for 12)
   0

Answer:  Hexadecimal number is 2EC
```

The following table will simplify the conversion process:

| **third digit** | | **second digit** | | **first digit** | | **zeroth digit** | |
|---|---|---|---|---|---|---|---|
| hex | decimal | hex | decimal | hex | decimal | hex | decimal |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4,096 | 1 | 256 | 1 | 16 | 1 | 1 |
| 2 | 8,192 | 2 | 512 | 2 | 32 | 2 | 2 |
| 3 | 12,288 | 3 | 768 | 3 | 48 | 3 | 3 |
| 4 | 16,384 | 4 | 1,024 | 4 | 64 | 4 | 4 |
| 5 | 20,480 | 5 | 1,280 | 5 | 80 | 5 | 5 |
| 6 | 24,576 | 6 | 1,536 | 6 | 96 | 6 | 6 |
| 7 | 28,672 | 7 | 1,792 | 7 | 112 | 7 | 7 |
| 8 | 32,768 | 8 | 2,048 | 8 | 128 | 8 | 8 |
| 9 | 36,864 | 9 | 2,304 | 9 | 144 | 9 | 9 |
| A | 40,960 | A | 2,560 | A | 160 | A | 10 |
| B | 45,056 | B | 2,816 | B | 176 | B | 11 |
| C | 49,152 | C | 3,072 | C | 192 | C | 12 |
| D | 53,248 | D | 3,328 | D | 208 | D | 13 |
| E | 57,344 | E | 3,584 | E | 224 | E | 14 |
| F | 61,440 | F | 3,840 | F | 240 | F | 15 |

## You Try It...

10. $312_{10}$ = _____16
11. $708_{10}$ = _____16
12. $778_{10}$ = _____16

## Hexadecimal-to-Binary Conversion

You will need to be able to convert a hexadecimal number to its binary equivalent and vice-versa. This is *very* easy to do.

To convert a hexadecimal number to its binary equivalent, simply replace each hexadecimal digit with its four bit equivalent as per the above table. For example, to convert $5E_{16}$ to binary, replace 5 with 0101 and E with 1110 giving $01011110_2$.

## You Try It...

13. $4A_{16}$ = _____$_2$
14. $B17_{16}$ = _____$_2$

## Binary-to-Hexadecimal Conversion

To convert a binary number to its hexadecimal equivalent, simply divide the bits into groups of four. Start from the right, and add extra zeroes on the left if necessary so the total number of bits is a multiple of four. Then replace each group of four with a single hexadecimal digit. For example, to convert $10010010110111_2$ to hexadecimal, rewrite it as |10|0100|1011|0111|, add two leading zeroes giving |0010|0100|1011|0111|, then replace each four-bit string by its equivalent hexadecimal digit, giving $24B7_{16}$.

## You Try It...

15. $10111001001_2$ = _____16
16. $10011010100100_2$ = _____16

## Sample Program

We have seen that when the EBCDIC collating sequence is used, the letter A is equivalent to a decimal 193. This value (193) is also equal to $11000001_2$ or $C1_{16}$. This is shown in the following program:

```
A:\MIN>type data5a.mlc
         PRINT NOGEN
         START 0
BEGIN    BEGIN
         MVI   LETTER,W
         WTO   MESSAGE
         MVI   LETTER,X
         WTO   MESSAGE
         MVI   LETTER,Y
         WTO   MESSAGE
         MVI   LETTER,Z
         WTO   MESSAGE
```

```
RETURN
```

```
MESSAGE  DS    0CL24
         DC    CL22'DATA5A ... Letter is <'
LETTER   DC    CL1' '
         DC    CL1'>'
W        EQU   C'A'
X        EQU   X'C1'
Y        EQU   B'11000001'
Z        EQU   193
         END   BEGIN

A:\MIN>m370 data5a

A:\MIN>a370 data5a/lx

A:\MIN>l370 data5a/lx

A:\MIN>data5a
DATA5A ... Letter is <A>
DATA5A ... Letter is <A>
DATA5A ... Letter is <A>
DATA5A ... Letter is <A>
```

## Binary Arithmetic

Binary arithmetic (we look at addition only) is really quite simple. It's a lot like decimal addition, except you have to remember that, just as you "carry the one" every time you reach ten in decimal addition, you will "carry the one" every time you reach two in binary addition. For example:

```
1.      11010
      +  1011
         ----
         1     0 + 1 = 1 --> no overflow


          1
2.      11010
      +  1011
         ----
         01    1 + 1 = 2 --> 2 is not a valid digit,
                             enter 0 and carry the 1.
                             (Recall 2 10 = 10 2)
          1
3.      11010
      +  1011
         ----
         101   1 + 0 + 0 = 1 --> no overflow

        1 1
4.      11010
      +  1011
         ----
         0101  1 + 1 = 2 --> enter 0 and carry the 1.


        1 1
5.      11010
      +  1011
         ----
        100101  1 + 1 = 2 --> enter 0 and carry the 1.
```

The answer is $11010_2 + 1011_2 = 100101_2$. If you convert these values to decimal, you'll see this is equivalent to $26 + 11 = 37$.

**You Try It...**

17. $10101_2 + 1101_2 =$ _____$_2$
18. $11011_2 + 11011_2 =$ _____$_2$
19. $10011011_2 + 11110001_2 =$ _____$_2$

**Hexadecimal Arithmetic**

Hexadecimal arithmetic (addition only) isn't as easy as binary arithmetic, but it's not so bad once you get used to it. There will be occasions when you need to use hexadecimal arithmetic in BAL, such as when debugging your program. During such times you will probably want to use a calculator. Nevertheless, some understanding of simple hexadecimal arithmetic will improve your effectiveness as an assembler programmer.

As you might imagine, with hexadecimal addition you will "carry the one" every time you reach sixteen. Consider the following example:

```
1.          1
           ADE
        +   C7
           ---
            5   E + 7 = 14 + 7 = 21 = 16 + 5
                enter the 5 and carry the 1 (for 16)


2.         11
           ADE
        +   C7
           ---
           A5   1 + D + C = 1 + 13 + 12 = 26 = 16 + 10
                   enter A (for 10) and carry the 1
                   (for 16)
           11
3.         ADE
        +   C7
           ---
           BA5  1 + A = 1 + 10 = 11
                enter B (for 11)
```

The answer is $ADE_{16} + C7_{16} = BA5_{16}$. If you convert these values to decimal, you'll see this is equivalent to $2782 + 199 = 2981$.

**You Try It...**

20. $289_{16} + 3A5_{16} =$ _____$_{16}$
21. $4A8_{16} + DD7_{16} =$ _____$_{16}$
22. $32FE_{16} + 722_{16} =$ _____$_{16}$

## Exercises

1. True or false.

   T F a. The base 2 number system is also known as the binomial number system.
   T F b. The base 2 number system has two digits: 1 and 2.
   T F c. 1011 is a valid base 2, base 10, and base 16 number.
   T F d. A subscript may be used to indicate the base of a number when the base is unclear.
   T F e. The binary equivalent of the letter R is 11011001 when using the EBCDIC collating sequence.
   T F f. A bit consists of eight bytes.
   T F g. The hexadecimal equivalent for the decimal number 13 is C.
   T F h. A byte has 255 possible values.
   T F i. $2^6 = 64$.
   T F j. $2^0 = 16^0$.
   T F k. $2^1 = 16^1$.
   T F l. $1_2 = 1_{16}$.
   T F m. $11101111_2 = EF_{16}$.

2. Find the binary equivalent of 11, 23, 26, and 30. What can you conclude about the binary representation of any odd decimal number? Even decimal number?

3. Complete the following:

   a. $10111_2 =$ _____$_{10} =$ _____$_{16}$
   b. $11101111_2 =$ _____$_{10} =$ _____$_{16}$
   c. _____$_2 = 31_{10} =$ _____$_{16}$
   d. _____$_2 = 240_{10} =$ _____$_{16}$
   e. _____$_2=$ _____$_{10} = 83_{16}$
   f. _____$_2 =$ _____$_{10} = AE_{16}$

4. Complete the following:

   a. $11101_2 =$ _____$_{10} =$ _____$_{16}$
   b. $10101011_2 =$ _____$_{10} =$ _____$_{16}$
   c. _____$_2 = 48_{10} =$ _____$_{16}$
   d. _____$_2 = 221_{10} =$ _____$_{16}$
   e. _____$_2 =$ _____$_{10} = 7F_{16}$
   f. _____$_2 =$ _____$_{10} = BC_{16}$

## Exercises

5. Complete the following:

   a. $11010_2$ = _____$_{10}$ = _____$_{16}$
   b. $10100001_2$ = _____$_{10}$ = _____$_{16}$
   c. _____$_2$ = $67_{10}$ = _____$_{16}$
   d. _____$_2$ = $234_{10}$ = _____$_{16}$
   e. _____$_2$ = _____$_{10}$ = $ED_{16}$
   f. _____$_2$ = _____$_{10}$ = $6A_{16}$

6. Perform the following base 2 additions:

   a.
   ```
     10110011
   +   111001
   ```

   b.
   ```
     10001111
   +  1011101
   ```

   c.
   ```
     10011100
   + 10101110
   ```

   d.
   ```
     11010001
   + 11101101
   ```

   e. Find the decimal equivalent to each of the sums in parts a-d.

7. Perform the following base 16 additions:

   a.
   ```
     CA18
   +  797
   ```

   b.
   ```
     7EFA
   +  322
   ```

   c.
   ```
     2DB2
   +  693
   ```

   d.
   ```
     DEAD
   +  ACE
   ```

   e. Find the decimal equivalent to each of the sums in parts a-d.

8. This chapter includes a table showing the EBCDIC representation of all upper case letters. Take the decimal numbers for the letters A, B, R, S, and Z and find their binary equivalents. The decimal numbers for the *lower case* equivalents for these letters are 129, 130, 153, 162, and 169 respectively. Find the binary equivalents for these numbers as well. Use these binary numbers to answer the following question: In EBCDIC, what is the difference between an upper case letter and its lower case equivalent?

9. This chapter includes a table showing the EBCDIC representation of all numbers, 0 through 9 inclusive. Find the hexadecimal equivalents for each of these numbers. Use these hexadecimal numbers to answer the following question: In EBCDIC, what do all numbers have in common?