

Notes – Chapter 1

Binary Numbers and Codes

• Number Systems

A number system contains the number of digits named as its base.

If the base is n , then the digits are $0, \dots, n - 1$.

For example,

| Base # | Name | Digits |
|---------|--------------------|---------------------------------|
| Base 10 | <i>Decimal</i> | 0,1,2,3,4,5,6,7,8,9 |
| Base 2 | <i>Binary</i> | 0,1 |
| Base 8 | <i>Octal</i> | 0,1,2,3,4,5,6,7 |
| Base 16 | <i>Hexadecimal</i> | 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F |

Note : “New” digits (A,B,C,D,E,F) must be “invented” for use in a **hexadecimal** system. Other symbols could be used, but these appeal logically and are the convention used throughout this course.

Problems :

Given a number and its base, write the next 5 numbers in that base.

$(97)_{10}$ _____, _____, _____, _____, _____

$(775)_8$ _____, _____, _____, _____, _____

$(1011)_2$ _____, _____, _____, _____, _____

$(9FD)_{16}$ _____, _____, _____, _____, _____

Note that the largest digit in another base acts just like a 9 in Base 10.

● **Converting from another Base to the Decimal**

Follow these steps when converting from any other base to decimal *base 10* .

1. Write the column value for each number above each digit.
2. Multiply each digit by its column value (*multiply using base 10*).
3. Sum them up.

Example — Base 5 to Decimal (*Base 10*)

$$342_5 = (?)_{10}$$

$$\begin{array}{r} 342_5 = \quad 2 \times 5^0 \Rightarrow \quad 2 \times 1 = \quad 2 \\ \quad \quad 4 \times 5^1 \Rightarrow \quad 4 \times 5 = \quad 20 \\ \quad \quad 3 \times 5^2 \Rightarrow \quad 3 \times 25 = \quad 75 \\ \hline \quad \quad \quad \quad \quad \quad \quad 97_{10} \end{array}$$

Exercise — Base 8 (*octal*) **to Decimal** (*Base 10*)

$$7241_8 = (?)_{10}$$

Exercise — Base 16 (*hexadecimal*) **to Decimal** (*Base 10*)

$$4E3_{16} = (?)_{10}$$

Exercise — Base 2 (*binary*) **to Decimal** (*Base 10*)

$$01101001_2 = (?)_{10}$$

• **Converting from Decimal to Another Base**

There are 2 methods for converting from decimal to other bases

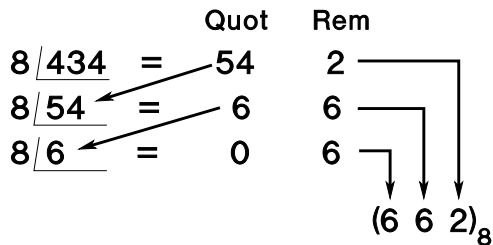
1. The Division Method, and
2. The Subtraction Method.

Example — using the Division Method

METHOD: Divide by the base. The remainder becomes the least significant digit. Then take the quotient and repeat, until the quotient becomes zero.

$$434_{10} = (?)_8$$

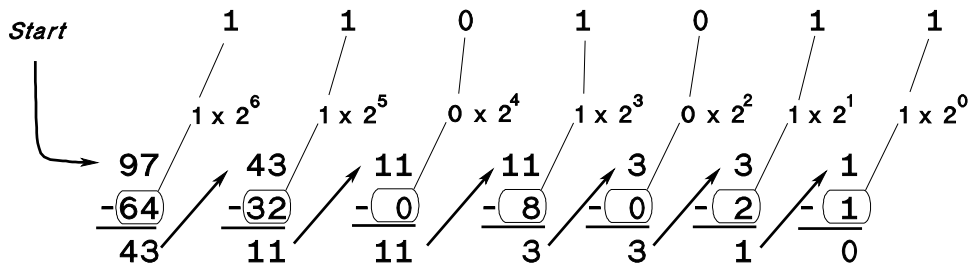
$$289_{10} = (?)_{16}$$



Example — using the Subtraction Method. Most useful for relatively small numbers. The method to use when computing “in your head.”

METHOD: Find the largest b^n ($b =$ base) that is *smaller* than the number. Find the largest *multiple* of that value that is smaller than the number. Subtract this value from the number. Repeat the process with the powers $n - 1, n - 2, \dots, 0$.

$$97_{10} = (?)_2$$



$$41_{10} = (?)_8$$

- **Converting from a non-decimal Base to a non-decimal Base**

The key to this type of conversion is to use Base 10 (*Decimal*) as the “bridge”.

$$Number_{given\ base} \rightarrow Equivalent_{Base\ 10} \rightarrow Equivalent_{desired\ base}$$

That is, you should convert your given number to base 10, then use that result to convert to the desired base.

Example — Convert $(1AF)_{16}$ to Base 5.

Step 1: Convert $1AF_{16}$ to Decimal

$$\begin{array}{r} F \times 16^0 \Rightarrow 15 \times 1 = 15 \\ A \times 16^1 \Rightarrow 10 \times 16 = 160 \\ 1 \times 16^2 \Rightarrow 1 \times 256 = \underline{256} \\ \hline (431)_{10} \end{array}$$

Step 2: Convert 431_{10} to Base 5

| | Quot | Rem |
|-----------------------|------|-----|
| $5 \overline{)431} =$ | 86 | 1 |
| $5 \overline{)86} =$ | 17 | 1 |
| $5 \overline{)17} =$ | 3 | 2 |
| $5 \overline{)3} =$ | 0 | 3 |

$(3\ 2\ 1\ 1)_5$

Exercise — Convert $(1632)_7$ to Base 12.

- **Simplified Conversions between Binary, Octal and Hexadecimal**

Because **Binary (Base 2)**, **Octal (Base 8)** and **Hexadecimal (Base 16)** are all powers of 2 ($2 = 2^1$, $8 = 2^3$, $16 = 2^4$), there is a process to easily convert between them.

Conversion to Octal from Binary

Group binary digits by 3 from the decimal point outward (both left and right). View each group of 3 digits of binary numbers and convert to octal for each group; Pad with zeros for groups less than 3.

Example - 11001100.11 binary to octal

011 001 100 . 110
~~~~ ~~~~ ~~~~~ ~~~~~  
3 1 4 . 6

**Conversion to Hexadecimal from Binary**

Group binary digits by 4 from the decimal point outward (both left and right). View each group of 4 digits of binary numbers and convert to hexadecimal for each group; Pad with zeros for groups less than 4.

**Exercise** - 0111 0101 1100.101 binary to hexadecimal

**Conversion between Hexadecimal and Octal**

**Rule :** When converting between Hexadecimal and Octal (*either Hexadecimal  $\rightarrow$  Octal OR Octal  $\rightarrow$  Hexadecimal*) **first convert to Binary, then convert to the target base.**

In other words use Binary as the “bridge” between these 2 bases.

**Exercise** -  $(7512)_8$  to  $(?)_{16}$

- **Converting Fractional Portions of Decimal (*Base 10*) Numbers to Other Bases**

**Process:** Multiply the decimal portion by the “new” base number. The integer is retained for the new number. Continue the process until zero is reached or the desired number of digits is attained.

**Example - Convert  $(0.75)_{10}$  to Binary**

$$\begin{array}{r}
 0.75 \times 2 = \underline{1.50} \quad 1 \\
 \swarrow \quad \searrow \\
 0.50 \times 2 = \underline{1.00} \quad 1 \\
 \swarrow \quad \searrow \\
 \text{Done} \\
 \text{Ans: } .11_2
 \end{array}$$

**Convert  $(0.65)_{10}$  to Hexadecimal**

$$\begin{array}{r}
 0.65 \times 16 = \underline{10.40} \quad A \\
 \swarrow \quad \searrow \\
 0.40 \times 16 = \underline{6.40} \quad 6 \\
 \swarrow \quad \searrow \\
 0.40 \times 16 = \underline{6.40} \quad 6 \\
 \text{Repeats} \\
 \text{Ans: } .A6\bar{6}_{16}
 \end{array}$$

**Exercise - Convert  $(0.312)_{10}$  to  $(?)_8$**

- **A little “bit” and a lot (*of bits*)**

A **bit** is a binary digit. ( $0, 1$ )

Large quantities of bits (and bytes) have names you should become familiar with. Here are a few names along with values.

| Greek Prefix | Abbreviation | Exponential Form | Actual Value  | Approximate Value |
|--------------|--------------|------------------|---------------|-------------------|
| <b>Kilo</b>  | <b>K</b>     | $2^{10}$         | 1,024         | 1,000             |
| <b>Mega</b>  | <b>M</b>     | $2^{20}$         | 1,048,576     | 1,000,000         |
| <b>Giga</b>  | <b>G</b>     | $2^{30}$         | 1,073,741,824 | 1,000,000,000     |

**An interesting fact** - Disk drive manufacturers use the “rounded” (approximate) values when specifying disk capacities, because they sound more impressive! That is why that new 1 Gigabyte drive actually ends up containing only 931.3 mB!

- Adding in Other Bases

**Example**

Octal:

$$\begin{array}{r}
 7\ 1\ 3\ 5 \\
 +\ 5\ 6\ 2\ 3 \\
 \hline
 14\ 7\ 6\ 0
 \end{array}$$

$\swarrow$   $12_{10} = 14_8$        $\swarrow$   $8_{10} = 10_8!$

$\nearrow$  *carry*

**Exercise**

Hex

$$\begin{array}{r}
 A\ 4\ F\ 1 \\
 +\ 3\ 8\ 1\ A \\
 \hline
 \end{array}$$

- Subtracting in Other Bases

The method shown here is not the method used by a computer. It is shown here so that you will have a “new method” to use to check your answers. The method you will be accountable for is the one explained in the next section.

**Example**

Octal:

$$\begin{array}{r}
 6\ 1\ 3\ 5 \\
 \cancel{7}\ 1\ 3\ 5 \\
 -\ 5\ 6\ 2\ 3 \\
 \hline
 1\ 3\ 1\ 2
 \end{array}$$

$\nearrow$  *borrow*  $9_{10}$

**Exercise**

Hex

$$\begin{array}{r}
 D\ A\ 1\ 7 \\
 -\ 7\ 4\ 3\ A \\
 \hline
 \end{array}$$

## • Signed Binary Numbers

How can we represent a **negative integer**? Since a computer stores only 0's and 1's, it cannot place a “-” sign on a number. Somehow the concept of negative must be expressed using only 0's and 1's. There are several ways to represent negative numbers; we will consider three:

1. **Signed Magnitude**,
2. **Signed-1's Complement**, and
3. **Signed-2's Complement**,

In each of these representations, the left-most bit is called the **Sign Bit**. A '0' in the Sign Bit indicates that the number is **Positive**. A '1' in the Sign Bit indicates that the number is **Negative**. When you look at the machine's “raw ” representation of the number (which sometimes must be done when using assembly or machine language), it is important to be able to correctly interpret the value of the number.

**In all three systems, the representations for *positive* numbers are the same. However, the three systems vary in the way they represent negative numbers.**

**NOTE:** The following sections discuss only how to use the complement systems. For a more complete (mathematical) treatment of how they work, see the Appendix on Complement Arithmetic.

## • Signed Magnitude :

We are already familiar with signed magnitude, since this is the system that our “regular” arithmetic system uses (except, of course, here we represent the values in base 2 rather than base 10). A positive number is represented by taking the unsigned binary representation of the number, and adding a leading zero (the sign bit). A negative number is represented by taking the unsigned binary (absolute value) representation of the number, and adding a leading one.

**Example.** Find the signed magnitude representations for  $+27_{10}$  and  $-47_{10}$ , using 8 bits.

Ans:

$+27$  is positive, so the signed magnitude representation is:

$$+27_{10} = 16 + 8 + 2 + 1 = \boxed{0}001\ 1011$$

$-47$  is negative, so take the positive representation for 45, then add a 1 sign bit:

$$-(47_{10}) = -(32 + 8 + 4 + 2 + 1) = \boxed{1}010\ 1111$$

**Example.** Take the negative of 1101 0010.

Ans:

$$-(11010010) = \boxed{0}010\ 1101$$

The addition of 2 numbers in signed magnitude form follows the same rules of ordinary arithmetic. While this system is very natural for us to use, since we have practiced it since early grade school, the rules for performing arithmetic are fairly complicated, and require some decisions to be made in the process. The rules for addition are:

- If both numbers are *positive*, then add the two numbers, and append a sign bit of zero (for positive) to the answer.
- If both numbers are *negative*, then add the absolute values of the two numbers, and append a 1 (for negative) to the answer.
- If the numbers are *opposite* in sign, subtract the smaller absolute value number from the larger absolute value number, and then append the sign of the larger absolute value number to the answer.

For subtraction, change the sign of the second number (i.e., “take the negative” of the second number), and then follow the same rules as for addition.

This method is complicated for a computer, because it requires the comparison of the two numbers first before deciding what rule to follow, AND it involves the use of two different procedures - ADD and SUBTRACT. Surprisingly, the next two number systems avoid these problems!

## ● One’s Complement:

In the *1’s Complement system*, positive numbers are represented **the same as before**, i.e., just like unsigned (and signed magnitude), with a leading zero (to represent a positive sign). However, negative numbers are formed by taking the positive (absolute value) of the number, and **switching all the 1’s to 0’s, and all the 0’s to 1’s**. This process, which is called “**taking the 1’s complement**” of a number, is equivalent to “**taking the negative**” of the number. The sign bit is treated no differently than any of the other bits.

Example. Find the 1’s complement representations for  $+27_{10}$  and  $-47_{10}$ , using 8 bits.

Ans:

$+27$  is positive, so the 1’s complement representation is the same as the unsigned value:

$$+27_{10} = 16 + 8 + 2 + 1 = 0001\ 1011$$

$-47$  is negative, so start with the positive representation for 47, then take the 1’s complement (“take the negative”):

$$-(47_{10}) = -(32 + 8 + 4 + 2 + 1) = -(0010\ 1111) = 1101\ 0000$$

Example. Take the 1’s complement (i.e., take the negative) of 1101 0010.

Ans: Switch all the bits:

$$-(1101\ 0010) = 0010\ 1101$$

**Exercise.** Find the 1's complement representation for  $+87_{10}$  using 8 bits.

**Exercise.** Find the 1's complement representation for  $-39_{10}$  using 8 bits.

**Exercise.** Find the 1's complement representation for  $-57_{10}$  using 8 bits.

The rules for addition and subtraction in 1's complement are simpler than for signed magnitude, at least for the computer, since no decisions need be made before performing the operations. **The rules are the same, regardless of whether the numbers are positive or negative!** The rules for addition are:

- Add the two numbers (including the sign bits).
- If the carry out of the sign bit is a 1, add it to the answer. (End-Around Carry).

To subtract, take the complement of (i.e., "take the negative of") the second number, then add (using the rules above).

**Example.** Perform the following *1's complement* arithmetic. Also show each problem in base 10.

$$\begin{array}{r}
 0001\ 1011 \\
 \hline
 +0010\ 1110 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1011 \\
 \hline
 +0011\ 0101 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 0100\ 1101 \\
 \hline
 -0101\ 0100 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1101\ 0110 \\
 \hline
 -0001\ 1011 \\
 \hline
 \end{array}$$

**Ans:** For the addition problems, we simply add:

$$\begin{array}{r}
 0001\ 1011 = +27 \\
 \hline
 +0010\ 1110 = +46 \\
 \hline
 0100\ 1001 = +73
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1011 = -(0001\ 0100) = -20 \\
 \hline
 +0011\ 0101 = \qquad\qquad\qquad +53 \\
 \hline
 0010\ 0000 \\
 \qquad\qquad\qquad +1 \quad \text{End around carry} \\
 \hline
 0010\ 0001 = +33
 \end{array}$$

For subtraction, we take the complement (of the second number) and add:

$$\begin{array}{r}
 0100\ 1101 = +77 \Rightarrow 0100\ 1101 = \qquad\qquad\qquad +77 \\
 \hline
 -0101\ 0100 = -(+84) \Rightarrow \underline{+1010\ 1011} = \qquad\qquad\qquad \underline{-84} \\
 \hline
 1111\ 1000 = -(0000\ 0111) = -7
 \end{array}$$

$$\begin{array}{r}
1101\ 0110 \\
\underline{-0001\ 1011} = \underline{-(0001\ 1011)} = -(+27) \Rightarrow \underline{+1110\ 0100} = \underline{-27} \\
\hline
1011\ 1010 \\
\quad \quad \quad \underline{+1} \text{ End around carry} \\
1011\ 1011 = -(0100\ 0100) = -68
\end{array}
\Rightarrow \begin{array}{r}
1101\ 0110 = -(0010\ 1001) = -41 \\
+1110\ 0100 = \underline{-27} \\
1011\ 1010 \\
\quad \quad \quad \underline{+1} \text{ End around carry} \\
1011\ 1011 = -(0100\ 0100) = -68
\end{array}$$

The advantage of the 1's complement system is that the arithmetic operations are the same, regardless of the signs of the numbers. Thus, no comparisons or decisions need be made before performing the operations. Also, note that only one operation, addition, is ever needed to compute the answer.

A problem with 1's complement arithmetic involves the value zero:

**Example.** Find the 1's complement of 0000 0000.

Ans:

$$-(0000\ 0000) = 1111\ 1111$$

There are representations for both positive and negative zero! This makes it awkward to compare numbers with zero, since two comparisons must be performed. The 2's complement system solves this problem.

**Exercise.** Perform the following *1's complement* arithmetic. Also show each problem in base 10.

$$\begin{array}{r}
0000\ 1011 \\
\underline{+0110\ 0110}
\end{array}
\qquad
\begin{array}{r}
1101\ 0001 \\
\underline{+0010\ 1101}
\end{array}
\qquad
\begin{array}{r}
1111\ 0001 \\
\underline{-1101\ 0100}
\end{array}
\qquad
\begin{array}{r}
1110\ 1011 \\
\underline{-0011\ 0111}
\end{array}$$

Ans:

- **Two's Complement:**

In *2's complement*, positive numbers are represented the same as unsigned numbers with a leading zero. Negative numbers are formed by taking the 1's complement ("switching all the bits") **and adding one**. This process is called "taking the 2's complement," and is equivalent to "taking the negative" of a number.

**Example.** Find the 2's complement representations for  $+27_{10}$  and  $-47_{10}$ , using 8 bits.

**Ans:**

$+27$  is positive, so the 2's complement representation is simply:

$$+27_{10} = 16 + 8 + 2 + 1 = 00011011$$

$-47$  is negative, so take the positive representation for 47, then take the 2's complement ("take the negative"):

$$-(47_{10}) = -(32 + 8 + 4 + 2 + 1) = -(00101111) = 11010000 + 1 = 11010001$$

**Example.** Take the 2's complement (i.e., take the negative) of 11010010.

**Ans:** Switch all the bits and add one:

$$-(11010010) = 00101101 + 1 = 00101110$$

**Example.** Take the 2's complement (i.e., take the negative) of 00000000.

**Ans:** Remember that we throw away any carries out of the sign bit!

$$-(00000000) = 11111111 + 1 = 00000000$$

**NOTE:** There is a single representation for zero!

**Exercise.** Find the 2's complement representation for  $+37_{10}$  using 8 bits.

**Exercise.** Find the 2's complement representation for  $-41_{10}$  using 8 bits.

**Exercise.** Find the 2's complement representation for  $-108_{10}$  using 8 bits.

The rules for addition and subtraction in 2's complement are even simpler than for 1's complement, since there is no end around carry. For addition, add the two numbers (**Ignore** any carries out of the sign bit). To subtract, take the 2's complement of (i.e., "take the negative of") the second number, then add.

**Example.** Perform the following *2's complement* arithmetic. Also show each problem in base 10.

$$\begin{array}{r}
 0001\ 1011 \\
 +0010\ 1110 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1011 \\
 +0011\ 0101 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 0100\ 1101 \\
 -0101\ 0100 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1101\ 0110 \\
 -0001\ 1011 \\
 \hline
 \end{array}$$

Ans: For the addition problems, simply add (ignoring carries out):

$$\begin{array}{r}
 0001\ 1011 = +27 \\
 +0010\ 1110 = +46 \\
 \hline
 0100\ 1001 = +73
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1011 = -(0001\ 0101) = -21 \\
 +0011\ 0101 = \\
 \hline
 0010\ 0000 = +32
 \end{array}$$

For subtraction, take the complement and add:

$$\begin{array}{r}
 0100\ 1101 = +77 \Rightarrow 0100\ 1101 = +77 \\
 -0101\ 0100 = -(+84) \Rightarrow +1010\ 1100 = -84 \\
 \hline
 1111\ 1001 = -(0000\ 0111) = -7
 \end{array}$$
  

$$\begin{array}{r}
 1101\ 0110 \Rightarrow 1101\ 0110 = -(0010\ 1010) = -42 \\
 -0001\ 1011 = -(0001\ 1011) = -(+27) \Rightarrow +1110\ 0101 = -27 \\
 \hline
 1011\ 1011 = -(0100\ 0101) = -69
 \end{array}$$

The 2's complement system trades off a slightly more difficult "complementing" process (you must complement **and** add one) for a slightly simpler addition process (no end around carry) and, even more importantly, a consistent representation for zero, when compared with 1's complement. For the latter reason, 2's complement is the representation used for integers in virtually ALL computers today.

**Also Note:** *2's complement* binary numbers and *unsigned* binary numbers are added using the same rules – only the interpretation of the numbers is different! Therefore, computers need only one common hardware circuit to handle both types of arithmetic!

**Exercise.** Perform the following *2's complement* arithmetic. Also show each problem in base 10.

$$\begin{array}{r}
 0010\ 1010 \\
 +0011\ 0110 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1101 \\
 +0110\ 1101 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1010\ 1001 \\
 -1101\ 0100 \\
 \hline
 \end{array}
 \qquad
 \begin{array}{r}
 1110\ 1011 \\
 -0001\ 0010 \\
 \hline
 \end{array}$$

Ans:

**Example.** Find the 8 bit and 16 bit 2's complement representations for +5 and -5.

Ans:

For +5, the representations are the same as for the 8 and 16 bit unsigned values:

$$+5 = (4 + 1) = 00000101_{8bit} = 0000000000000101_{16bit}$$

For -5, switch all the bits and add 1 for each representation:

$$-5 = 11111011_{8bit} = 1111111111111011_{16bit}$$

**Note:** To express a 2's complement value in a larger number of bits, the value is **sign-extended** – that is, the sign bit is replicated to make up the required number of bits.

● **Overflow**

*Overflow* occurs whenever a value is too large to be represented in the specified number of bits. In signed representations, this occurs when there are not enough bits to include a sign bit:

**Example.** Express the value +142<sub>10</sub> as an eight bit 2's complement number.

Ans:

$$+142_{10} = (128 + 8 + 4 + 2 + 1) = 1000\ 1111$$

But no room for a sign in 8 bits!

While this value could be represented as an unsigned value in 8 bits (i.e., no sign bit), there are not enough bits available to represent this value in 2's complement (or any other signed representation). For unsigned numbers of  $n$  bits, the values that can be represented range from  $0 \rightarrow (2^n - 1)$ . For the signed representations, since one bit must be used for the sign, the range is  $-(2^{n-1} - 1) \rightarrow +(2^{n-1} - 1)$ . (The low end for 2's complement is actually  $-2^{n-1}$ .)

Overflow also manifests itself when doing complement arithmetic. There are two ways to detect overflow when doing 1's and 2's complement arithmetic (they are both mathematically equivalent):

1. If the addition of two positive numbers yields a negative result, or vice versa.
2. If there is a carry out of the sign bit, but no carry in, or versa. (No carries, or a carry in AND a carry out are OK!)

The first case is most likely what we would observe during overflow, while the second case is the way a computer detects overflow.

**Example.** Perform the following 2's complement arithmetic. Also show each problem in base 10.

Ans:

|                                 |                                                                   |
|---------------------------------|-------------------------------------------------------------------|
| 0101 1011 = +91                 | 1011 1101 $\Rightarrow$ 1011 1101 = -67                           |
| <u>+0110 1110</u> = <u>+110</u> | <u>-0101 0100</u> $\Rightarrow$ <u>+1010 0100</u> = <u>-(+84)</u> |
| 1100 1001 = -55 ???             | 0010 0000      0110 0001      +97 ???                             |
| (Carry in, no carry out)        | (Carry out, no carry in)                                          |

● **Comparing the Three Signed Representations**

In all three signed representations, a positive number is expressed in the same way – the same as the unsigned value, with at least one zero appended on the front. For example, the value  $9_{10}$  would be expressed as 0000 1001 in eight bits on all three systems. However, the representations for negative values are different in each system; the only common characteristic is that the left-most (sign) bit is a one. For example,  $-9_{10}$  in each system would look like:

*Signed-mag:* 1000 1001    *1's comp:* 1111 0110    *2's comp:* 1111 0111

Reconstruct TABLE 1-3 (p. 15) in the space provided. Understand why each integer is represented the way it is.

| Decimal | 2's Complement | 1's Complement | Signed Magnitude |
|---------|----------------|----------------|------------------|
| +7      | 0111           | 0111           | 0111             |
| +6      |                |                |                  |
| +5      |                |                |                  |
| +4      |                |                |                  |
| +3      |                |                |                  |
| +2      |                |                |                  |
| +1      |                |                |                  |
| +0      |                |                |                  |
| -0      |                |                |                  |
| -1      |                |                |                  |
| -2      |                |                |                  |
| -3      |                |                |                  |
| -4      |                |                |                  |
| -5      |                |                |                  |
| -6      |                |                |                  |
| -7      |                |                |                  |
| -8      |                |                |                  |

For each of the binary patterns below, show the represented value in decimal for each of the signed representations:

| Binary value     | Signed-mag | 1's Complement | 2's Complement |
|------------------|------------|----------------|----------------|
| $(10\ 0010)_2$   |            |                |                |
| $(0111\ 0111)_2$ |            |                |                |
| $(1110\ 0010)_2$ |            |                |                |
| $(1001\ 1011)_2$ |            |                |                |

### • Summary of Rules for Complement Arithmetic

#### For 1's Complement:

- Positive numbers: Represented the **SAME AS** an unsigned value (with at least one leading zero).
- Negative numbers: Computed from the positive value by “switching all the bits”
- Addition: 1. Simply ADD THE NUMBERS. It makes no difference that either or both of the numbers is negative!  
2. If there was a carry out of the sign bit, add 1 to the result (end-around carry).
- Subtraction: Take the 1's complement of the second number, then add (as above).

#### For 2's Complement:

- Positive numbers: Represented the **SAME AS** an unsigned value (with at least one leading zero).
- Negative numbers: Computed from the positive value by “switching all the bits” and adding one.
- Addition: Simply ADD THE NUMBERS. It makes no difference that either or both of the numbers is negative!
- Subtraction: Take the 2's complement of the second number, then add.

### • Floating Point Numbers

We will not be using Floating Point (**real** or **float**) Numbers in this class. However, for completeness, their representation is included here. Floating Point numbers are stored as a *mantissa* (fraction) and *exponent* - the exact number of bits used for each part determines the maximum and minimum values, and the accuracy (number of digits) that can be stored. While virtually all computers use 2's complement for storing ints, each computer uses its own unique floating point format. An attempt to standardize the way that computers store floating point values resulted in the **IEEE Format**, whose bit layout and resulting maximum and minimum values are shown on the next page. Many of the more recent computer designs now use this format, and some of the older machines have been “retrofitted” with this format, making it easier to transfer data between different machines.

|          |                 |                            |
|----------|-----------------|----------------------------|
| <b>S</b> | <i>Exponent</i> | <i>Mantissa (fraction)</i> |
|----------|-----------------|----------------------------|

**S - Sign Bit (0 for positive, 1 for negative)**

*exp* - **exponent, base 2, 8 bits for single, 11 bits for double**

*mantissa* - **fraction, base 2, radix pt assumed to be in front of number, 23 bits for single, 52 bits for double**

*Range* - **single,  $\pm 2^{\pm 127}$ , ( $\approx \pm 10^{\pm 38}$ ), double,  $\pm 2^{\pm 1023}$ , ( $\approx \pm 10^{\pm 308}$ )**

*Accuracy* - **single, 24 bits (more than 7 decimal digits)**

- **double, 53 bits (almost 16 decimal digits)**

- **Decimal Codes**

There are a number of decimal codes in use. You are already familiar with the binary, octal, decimal and hexadecimal systems. The binary system is the most natural system for a computer. Humans are used to a decimal system. For this reason, the Binary Coded Decimal (BCD) system was invented. Some computers store BCD numbers. Each digit in a BCD number is represented by 4 bits. Copy TABLE 1-4 (p. 18) in the space below.

It is important to realize that BCD numbers are actually *decimal* numbers and *not* binary numbers, even though they are represented in bits. There are other decimal codes, but this is the only one that will be highlighted here.

- **Alphanumeric (character) Codes**

There are a number of different codes for representing characters. Two of the most prominent are listed below :

- ASCII - *American Standard Code for Information Interchange*
- EBCDIC - *Extended Binary Coded Decimal Interchange Code*

# ASCII Character Set

|   | MSD                                          |                                                |                              |                       | LSD                 |                     |                      |                                  |
|---|----------------------------------------------|------------------------------------------------|------------------------------|-----------------------|---------------------|---------------------|----------------------|----------------------------------|
|   | 0x0                                          | 0x1                                            | 0x2                          | 0x3                   | 0x4                 | 0x5                 | 0x6                  | 0x7                              |
| 0 | <b>NUL</b><br>Null Char<br>000 0             | <b>DLE</b><br>Data Link Escape<br>020 16       | <b>SP</b><br>Space<br>040 32 | <b>0</b><br>060 48    | <b>@</b><br>0100 64 | <b>P</b><br>0120 80 | <b>,</b><br>0140 96  | <b>p</b><br>0160 112             |
| 1 | <b>SOH</b><br>Start of Header<br>001 1       | <b>DC1</b><br>Device Control 1<br>021 17       | <b>!</b><br>041 33           | <b>1</b><br>061 49    | <b>A</b><br>0101 65 | <b>Q</b><br>0121 81 | <b>a</b><br>0141 97  | <b>q</b><br>0161 113             |
| 2 | <b>STX</b><br>Start of Transmission<br>002 2 | <b>DC2</b><br>Device Control 2<br>022 18       | <b>"</b><br>042 34           | <b>2</b><br>062 50    | <b>B</b><br>0102 66 | <b>R</b><br>0122 82 | <b>b</b><br>0142 98  | <b>r</b><br>0162 114             |
| 3 | <b>ETX</b><br>End of Transmission<br>003 3   | <b>DC3</b><br>Device Control 3<br>023 19       | <b>#</b><br>043 35           | <b>3</b><br>063 51    | <b>C</b><br>0103 67 | <b>S</b><br>0123 83 | <b>c</b><br>0143 99  | <b>s</b><br>0163 115             |
| 4 | <b>EOT</b><br>End of Tape<br>004 4           | <b>DC4</b><br>Device Control 4<br>024 20       | <b>\$</b><br>044 36          | <b>4</b><br>064 52    | <b>D</b><br>0104 68 | <b>T</b><br>0124 84 | <b>d</b><br>0144 100 | <b>t</b><br>0164 116             |
| 5 | <b>ENQ</b><br>Enquire<br>005 5               | <b>NAK</b><br>Negative Acknowledge<br>025 21   | <b>%</b><br>045 37           | <b>5</b><br>065 53    | <b>E</b><br>0105 69 | <b>U</b><br>0125 85 | <b>e</b><br>0145 101 | <b>u</b><br>0165 117             |
| 6 | <b>ACK</b><br>Acknowledge<br>006 6           | <b>SYN</b><br>Synchronize<br>026 22            | <b>&amp;</b><br>046 38       | <b>6</b><br>066 54    | <b>F</b><br>0106 70 | <b>V</b><br>0126 86 | <b>f</b><br>0146 102 | <b>v</b><br>0166 118             |
| 7 | <b>BEL</b><br>Bell<br>007 7                  | <b>ETB</b><br>End Transmission Block<br>027 23 | <b>'</b><br>047 39           | <b>7</b><br>067 55    | <b>G</b><br>0107 71 | <b>W</b><br>0127 87 | <b>g</b><br>0147 103 | <b>w</b><br>0167 119             |
| 8 | <b>BS</b><br>Backspace<br>010 8              | <b>CAN</b><br>Cancel<br>030 24                 | <b>(</b><br>050 40           | <b>8</b><br>070 56    | <b>H</b><br>0110 72 | <b>X</b><br>0130 88 | <b>h</b><br>0150 104 | <b>x</b><br>0170 120             |
| 9 | <b>HT</b><br>Horizontal Tab<br>011 9         | <b>EM</b><br>End of Medium<br>031 25           | <b>)</b><br>051 41           | <b>9</b><br>071 57    | <b>I</b><br>0111 73 | <b>Y</b><br>0131 89 | <b>i</b><br>0151 105 | <b>y</b><br>0171 121             |
| A | <b>LF</b><br>Line Feed<br>012 10             | <b>SUB</b><br>Substitute<br>032 26             | <b>*</b><br>052 42           | <b>:</b><br>072 58    | <b>J</b><br>0112 74 | <b>Z</b><br>0132 90 | <b>j</b><br>0152 106 | <b>z</b><br>0172 122             |
| B | <b>VT</b><br>Vertical Tab<br>013 11          | <b>ESC</b><br>Escape<br>033 27                 | <b>+</b><br>053 43           | <b>;</b><br>073 59    | <b>K</b><br>0113 75 | <b>[</b><br>0133 91 | <b>k</b><br>0153 107 | <b>{</b><br>0173 123             |
| C | <b>FF</b><br>Form Feed<br>014 12             | <b>FS</b><br>File Separator<br>034 28          | <b>,</b><br>054 44           | <b>&lt;</b><br>074 60 | <b>L</b><br>0114 76 | <b>\</b><br>0134 92 | <b>l</b><br>0154 108 | <b> </b><br>0174 124             |
| D | <b>CR</b><br>Carriage Return<br>015 13       | <b>GS</b><br>Group Separator<br>035 29         | <b>-</b><br>055 45           | <b>=</b><br>075 61    | <b>M</b><br>0115 77 | <b>]</b><br>0135 93 | <b>m</b><br>0155 109 | <b>}</b><br>0175 125             |
| E | <b>SO</b><br>Shift Out<br>016 14             | <b>RS</b><br>Record Separator<br>036 30        | <b>.</b><br>056 46           | <b>&gt;</b><br>076 62 | <b>N</b><br>0116 78 | <b>^</b><br>0136 94 | <b>n</b><br>0156 110 | <b>~</b><br>0176 126             |
| F | <b>SI</b><br>Shift In<br>017 15              | <b>US</b><br>Unit Separator<br>037 31          | <b>/</b><br>057 47           | <b>?</b><br>077 63    | <b>O</b><br>0117 79 | <b>_</b><br>0137 95 | <b>o</b><br>0157 111 | <b>DEL</b><br>Rubout<br>0177 127 |