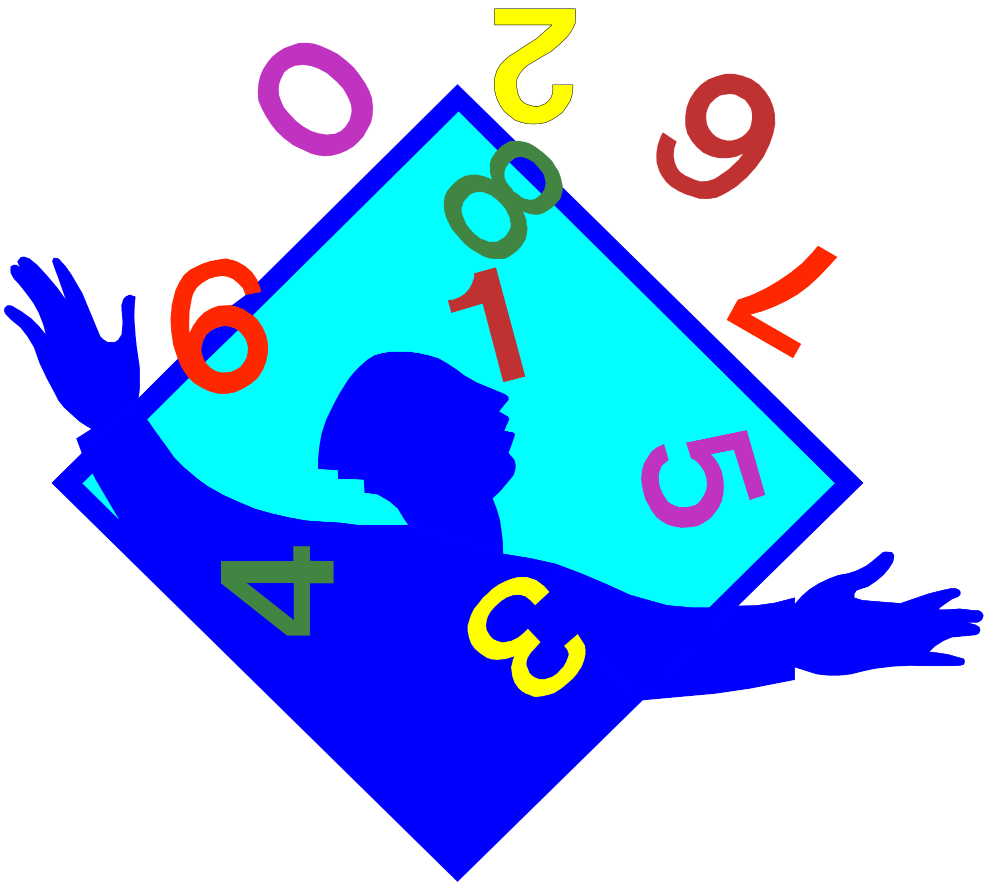
NUMBER SYSTEMS TUTORIAL



Courtesy of: thevbprogrammer.com

Number Systems Concepts

The study of number systems is useful to the student of computing due to the fact that number systems other than the familiar **decimal (base 10) number system** are used in the computer field.

Digital computers internally use the **binary (base 2) number system** to represent data and perform arithmetic calculations. The binary number system is very efficient for computers, but not for humans. Representing even relatively small numbers with the binary system requires working with long strings of ones and zeroes.

The **hexadecimal (base 16) number system** (often called "hex" for short) provides us with a shorthand method of working with binary numbers. One digit in hex corresponds to four binary digits (bits), so the internal representation of one byte can be represented either by eight binary digits or two hexadecimal digits. Less commonly used is the **octal (base 8)** number system, where one digit in octal corresponds to three binary digits (bits).

In the event that a computer user (programmer, operator, end user, etc.) needs to examine a display of the internal representation of computer data (such a display is called a "dump"), viewing the data in a "shorthand" representation (such as hex or octal) is less tedious than viewing the data in binary representation. The binary, hexadecimal , and octal number systems will be looked at in the following pages.

The decimal number system that we are all familiar with is a <u>positional</u> number system. The actual number of symbols used in a positional number system depends on its <u>base</u> (also called the <u>radix</u>). The <u>highest</u> numerical symbol always has a value of <u>one less</u> than the base. The decimal number system has a base of 10, so the numeral with the highest value is 9; the octal number system has a base of 8, so the numeral with the highest value is 7, the binary number system has a base of 2, so the numeral with the highest value is 1, etc.

Any number can be represented by arranging symbols in specific positions. You know that in the decimal number system, the successive positions to the left of the decimal point represent units (ones), tens, hundreds, thousands, etc. Put another way, <u>each position represents a specific power of base 10</u>. For example, the decimal number 1,275 (written $1{,}275_{10}$)* can be expanded as follows:

1 2 7 5_{10}

5	x	10^0	=	5	x	1	=	5
7	x	10^1	=	7	x	10	=	70
2	x	10^2	=	2	x	100	=	200
1	x	10^3	=	1	x	1000	=	1000

								1275_{10}

Remember the mathematical rule that $\mathbf{n^0 = 1}$, or any number raised to the zero power is equal to 1.

Here is another example of an expanded decimal number:

1 0 4 0 6_{10}

6	x	10^0	=	6	x	1	=	6
0	x	10^1	=	0	x	10	=	0
4	x	10^2	=	4	x	100	=	400
0	x	10^3	=	0	x	1000	=	0
1	x	10^4	=	1	x	10000	=	10000

								10406_{10}

* When doing number system problems, it is helpful to use a subscript to indicate the base of the number being worked with. Thus, the subscript "10" in 1275_{10} indicates that we are working with the number 1275 in <u>base 10</u>.

TRY THIS: *Expand the following decimal number:*

5 1 3 0_{10}

The Binary Number System

The same principles of positional number systems we applied to the decimal number system can be applied to the binary number system. However, the base of the binary number system is two, so each position of the binary number represents a successive power of two. From right to left, the successive positions of the binary number are weighted 1, 2, 4, 8, 16, 32, 64, etc. A list of the first several powers of 2 follows:

$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	$2^5 = 32$
$2^6 = 64$	$2^7 = 128$	$2^8 = 256$	$2^9 = 512$	$2^{10} = 1024$	$2^{11} = 2048$

For reference, the following table shows the decimal numbers 0 through 31 with their binary equivalents:

Decimal	Binary	Decimal	Binary
0	0	16	10000
1	1	17	10001
2	10	18	10010
3	11	19	10011
4	100	20	10100
5	101	21	10101
6	110	22	10110
7	111	23	10111
8	1000	24	11000
9	1001	25	11001
10	1010	26	11010
11	1011	27	11011
12	1100	28	11100
13	1101	29	11101
14	1110	30	11110
15	1111	31	11111

Converting a Binary Number to a Decimal Number

To determine the value of a binary number (1001_2, for example), we can expand the number using the positional weights as follows:

1 0 0 1_2

1	x	2^0	=	1	x	1	=	1
0	x	2^1	=	0	x	2	=	0
0	x	2^2	=	0	x	4	=	0
1	x	2^3	=	1	x	8	=	8

								9_{10}

Here's another example to determine the value of the binary number 1101010_2:

1 1 0 1 0 1 0_2

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

								106_{10}

TRY THIS: *Convert the following binary numbers to their decimal equivalents:*

(a) **1 1 0 0 1 1 0_2**

(b) **1 1 1 1 1 0 0 1_2**

<u>Converting a Decimal Number to a Binary Number</u>

To convert a decimal number to its binary equivalent, the <u>remainder method</u> can be used. (This method can be used to convert a decimal number into <u>any</u> other base.) The remainder method involves the following four steps:

(1) Divide the decimal number by the base (in the case of binary, divide by 2).

(2) Indicate the remainder to the right.

(3) Continue dividing into each quotient (and indicating the remainder) until the divide operation produces a zero quotient.

(4) The base 2 number is the numeric remainder reading from the last division to the first (if you start at the bottom, the answer will read from top to bottom).

Example 1: Convert the decimal number 99_{10} to its binary equivalent:

	$2\overset{0}{\overline{)1}}$	**1**	(7)	Divide 2 into 1. The quotient is 0 with a remainder of 1, as indicated. Since the quotient is 0, stop here.
	$2\overset{1}{\overline{)3}}$	**1**	(6)	Divide 2 into 3. The quotient is 1 with a remainder of 1, as indicated.
	$2\overset{3}{\overline{)6}}$	**0**	(5)	Divide 2 into 6. The quotient is 3 with a remainder of 0, as indicated.
	$2\overset{6}{\overline{)12}}$	**0**	(4)	Divide 2 into 12. The quotient is 6 with a remainder of 0, as indicated.
	$2\overset{12}{\overline{)24}}$	**0**	(3)	Divide 2 into 24. The quotient is 12 with a remainder of 0, as indicated
	$2\overset{24}{\overline{)49}}$	**1**	(2)	Divide 2 into 49 (the quotient from the previous division). The quotient is 24 with a remainder of 1, indicated on the right.
START HERE ⇒	$2\overset{49}{\overline{)99}}$	**1**	(1)	Divide 2 into 99. The quotient is 49 with a remainder of 1; indicate the 1 on the right.

The answer, reading the remainders from top to bottom, is **1100011**, so $\mathbf{99_{10} = 1100011_2}$.

Example 2: Convert the decimal number 13_{10} to its binary equivalent:

	$\begin{array}{r} 0 \\ 2\,\overline{)\,1} \end{array}$	**1**	(4)	Divide 2 into 1. The quotient is 0 with a remainder of 1, as indicated.
	$\begin{array}{r} 1 \\ 2\,\overline{)\,3} \end{array}$	**1**	(3)	Divide 2 into 3. The quotient is 1 with a remainder of 1, as indicated.
	$\begin{array}{r} 3 \\ 2\,\overline{)\,6} \end{array}$	**0**	(2)	Divide 2 into 6. The quotient is 3 with a remainder of 0, indicated on the right.
START HERE $\Rightarrow$	$\begin{array}{r} 6 \\ 2\,\overline{)\,13} \end{array}$	**1**	(1)	Divide 2 into 13. The quotient is 6 with a remainder of 1; indicate the 1 on the right.

The answer, reading the remainders from top to bottom, is **1101**, so $\mathbf{13_{10} = 1101_2}$.

<u>**TRY THIS:**</u> *Convert the following decimal numbers to their binary equivalents:*

(a) $\mathbf{49_{10}}$ (b) $\mathbf{21_{10}}$

Binary Addition

Adding two binary numbers together is easy, keeping in mind the following four addition rules:

(1)	**0**	+	**0**	=	**0**
(2)	**0**	+	**1**	=	**1**
(3)	**1**	+	**0**	=	**1**
(4)	**1**	+	**1**	=	**10**

Note in the last example that it was necessary to "carry the 1". After the first two binary counting numbers, 0 and 1, all of the binary digits are used up. In the decimal system, we used up all the digits after the tenth counting number, 9. The same method is used in both systems to come up with the next number: place a zero in the "ones" position and start over again with one in the next position on the left. In the decimal system, this gives ten, or 10. In binary, it gives 10_2, which is read "one-zero, base two."

Consider the following binary addition problems and note where it is necessary to carry the 1:

$$\begin{array}{rccc} & 1 & 1 & 0 \\ + & 0 & 0 & 1 \\ \hline & \mathbf{1} & \mathbf{1} & \mathbf{1} \end{array} \qquad \begin{array}{rcc} & 1 & 1 \\ + & 1 & 0 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{1} \end{array} \qquad \begin{array}{rccc} & 1 & 0 & 0 \\ + & 1 & 0 & 1 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array} \qquad \begin{array}{rcc} & {}^{1} & \\ & 1 & 1 \\ + & 0 & 1 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{0} \end{array} \qquad \begin{array}{rcccc} & {}^{1} & {}^{1} & & \\ & 1 & 0 & 1 & 0 \\ + & 0 & 1 & 1 & 1 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array} \qquad \begin{array}{rccccc} & {}^{1} & {}^{1} & {}^{1} & {}^{1} & \\ & 1 & 1 & 1 & 1 & 1 \\ + & 0 & 1 & 0 & 1 & 1 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{array}$$

TRY THIS: *Perform the following binary additions:*

$$\text{(a)} \begin{array}{rcccc} & 1 & 0 & 0 & 1 \\ + & 1 & 1 & 0 & 0 \\ \hline \end{array} \qquad \text{(b)} \begin{array}{rcccc} & 1 & 1 & 1 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline \end{array} \qquad \text{(c)} \begin{array}{rccccc} & 1 & 0 & 1 & 0 & 1 \\ + & 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} \qquad \text{(d)} \begin{array}{rcccccc} & 1 & 1 & 0 & 1 & 1 & 0 \\ + & 1 & 1 & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Subtraction Using Complements

Subtraction in any number system can be accomplished through the use of **complements**. A complement is a number that is used to represent the negative of a given number.

When two numbers are to be subtracted, the subtrahend* can either be subtracted directly from the minuend (as we are used to doing in decimal subtraction) or, the complement of the subtrahend can be added to the minuend to obtain the difference. When the latter method is used, the addition will produce a high-order (leftmost) one in the result (a "carry"), which must be dropped. This is how the computer performs subtraction: it is very efficient for the computer to use the same "add" circuitry to do both addition and subtraction; thus, when the computer "subtracts", it is really adding the complement of the subtrahend to the minuend.

* In mathematical terminology, the factors of a subtraction problem are named as follows: **Minuend - Subtrahend = Difference**.

To understand complements, consider a mechanical register, such as a car mileage indicator, being rotated backwards. A five-digit register approaching and passing through zero would read as follows:

00005
00004
00003
00002
00001
00000
99999
99998
99997
etc.

It should be clear that the number 99998 corresponds to -2. Furthermore, if we add

$$\begin{array}{r} 00005 \\ \underline{+\ 99998} \\ 1\ 00003 \end{array}$$

and ignore the carry to the left, we have effectively formed the operation of subtraction: 5 - 2 = 3.

The number 99998 is called the *ten's complement* of 2. The ten's complement of any decimal number may be formed by subtracting each digit of the number from 9, then adding 1 to the least significant digit of the number formed.

In the example above, subtraction with the use of complements was accomplished as follows:

(1) We were dealing with a five-digit subtrahend that had a value of 00002. First, each digit of the subtrahend was subtracted from 9 (this preliminary value is called the *nine's complement* of the subtrahend):

$$\begin{array}{r} 9 \\ -\ \underline{0} \\ 9 \end{array} \quad \begin{array}{r} 9 \\ -\ \underline{0} \\ 9 \end{array} \quad \begin{array}{r} 9 \\ -\ \underline{0} \\ 9 \end{array} \quad \begin{array}{r} 9 \\ -\ \underline{0} \\ 9 \end{array} \quad \begin{array}{r} 9 \\ -\ \underline{2} \\ 7 \end{array}$$

(2) Next, 1 was added to the nine's complement of the subtrahend (99997) giving the ten's complement of subtrahend (99998):

$$\begin{array}{r} 9\ 9\ 9\ 9\ 7 \\ \underline{+\ 1} \\ 9\ 9\ 9\ 9\ 8 \end{array}$$

(3) The ten's complement of the subtrahend was added to the minuend giving 100003. The leading (carried) 1 was dropped, effectively performing the subtraction of 00005 - 00002 = 00003.

$$\begin{array}{r} 0\ 0\ 0\ 0\ 5 \\ \underline{+\ 9\ 9\ 9\ 9\ 8} \\ \cancel{1}\ 0\ 0\ 0\ 0\ 3 \end{array}$$

The answer can be checked by making sure that 2 + 3 = 5.

Another example: Still sticking with the familiar decimal system, subtract **4589 - 322**, using complements ("eyeballing" it tells us we should get **4267** as the difference).

(1) First, we'll compute the four digit nine's complement of the subtrahend 0322 (we must add the leading zero in front of the subtrahend to make it the same size as the minuend):

$$\begin{array}{r} 9 \\ -\ 0 \\ \hline 9 \end{array} \quad \begin{array}{r} 9 \\ -\ 3 \\ \hline 6 \end{array} \quad \begin{array}{r} 9 \\ -\ 2 \\ \hline 7 \end{array} \quad \begin{array}{r} 9 \\ -\ 2 \\ \hline 7 \end{array}$$

(2) Add 1 to the nine's complement of the subtrahend (9677) giving the ten's complement of subtrahend (9678):

$$\begin{array}{r} 9\ 6\ 7\ 7 \\ +\ 1 \\ \hline 9\ 6\ 7\ 8 \end{array}$$

(3) Add the ten's complement of the subtrahend to the minuend giving 14267. Drop the leading 1, effectively performing the subtraction of 4589 - 0322 = 4267.

$$\begin{array}{r} 4\ 5\ 8\ 9 \\ +\ 9\ 6\ 7\ 8 \\ \hline \cancel{1}\ 4\ 2\ 6\ 7 \end{array}$$

The answer can be checked by making sure that 322 + 4267 = 4589.

<u>TRY THIS:</u> *Solve the following subtraction problems using the complement method:*

(a) **5086 - 2993 =**

(b) **8391 - 255 =**

<u>Binary Subtraction</u>

We will use the complement method to perform subtraction in binary and in the sections on octal and hexadecimal that follow. As mentioned in the previous section, the use of complemented binary numbers makes it possible for the computer to add or subtract numbers using only circuitry for addition - the computer performs the subtraction of A - B by adding A + (two's complement of B) and then dropping the carried 1.

The steps for subtracting two binary numbers are as follows:

(1) Compute the <u>one's complement</u> of the subtrahend by subtracting each digit of the subtrahend by 1. A shortcut for doing this is to simply reverse each digit of the subtrahend - the 1's become 0's and the 0's become 1's.

(2) Add 1 to the one's complement of the subtrahend to get the <u>two's complement</u> of the subtrahend.

(3) Add the two's complement of the subtrahend to the minuend and drop the high-order 1. This is your difference.

Example 1: Compute $\mathbf{11010101_2 - 1001011_2}$

(1) Compute the one's complement of $\mathbf{1001011_2}$ by subtracting each digit from 1 (note that a leading zero was added to the 7-digit subtrahend to make it the same size as the 8-digit minuend):

$$\begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ -\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array}$$

(Note that the one's complement of the subtrahend causes each of the original digits to be reversed.)

(2) Add 1 to the one's complement of the subtrahend, giving the two's complement of the subtrahend:

$$\begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ & & & & & & + & 1 \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

(3) Add the two's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{ccccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & & {\scriptstyle 1} & \\ & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ + & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \cancel{1} & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

So $\mathbf{11010101_2 - 1001011_2 = 10001010_2}$.

The answer can be checked by making sure that $\mathbf{1001011_2 + 10001010_2} = \mathbf{11010101_2}$ **.**

Example 2: Compute $\mathbf{11111011_2 - 11000001_2}$

(1) Come up with the one's complement of the subtrahend, this time using the shortcut of reversing the digits:

Original number: 1 1 0 0 0 0 0 1

One's complement: 0 0 1 1 1 1 1 0

(2) Add 1 to the one's complement of the subtrahend, giving the two's complement of the subtrahend (the leading zeroes of the one's complement can be dropped):

$$\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 0 \\ & & & & + & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

(3) Add the two's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{ccccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ & & + & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \cancel{1} & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{array}$$

So $\mathbf{11111011_2 - 11000001_2 = 111010_2}$.

The answer can be checked by making sure that $\mathbf{11000001_2 + 111010_2} = \mathbf{11111011_2}$.

<u>TRY THIS:</u> *Solve the following binary subtraction problems using the complement method:*

(a) $\mathbf{11001101_2 - 10101010_2 =}$

(b) $\mathbf{100100_2 - 11101_2 =}$

The Octal Number System

The same principles of positional number systems we applied to the decimal and binary number systems can be applied to the octal number system. However, the base of the octal number system is eight, so each position of the octal number represents a successive power of eight. From right to left, the successive positions of the octal number are weighted 1, 8, 64, 512, etc. A list of the first several powers of 8 follows:

$8^0 = 1$ $8^1 = 8$ $8^2 = 64$ $8^3 = 512$ $8^4 = 4096$ $8^5 = 32768$

For reference, the following table shows the decimal numbers 0 through 31 with their octal equivalents:

Decimal	Octal	Decimal	Octal
0	0	16	20
1	1	17	21
2	2	18	22
3	3	19	23
4	4	20	24
5	5	21	25
6	6	22	26
7	7	23	27
8	10	24	30
9	11	25	31
10	12	26	32
11	13	27	33
12	14	28	34
13	15	29	35
14	16	30	36
15	17	31	37

Converting an Octal Number to a Decimal Number

To determine the value of an octal number (367_8, for example), we can expand the number using the positional weights as follows:

3 6 7$_8$

$$\begin{array}{ccccccccr} 7 & \text{x} & 8^0 & = & 7 & \text{x} & 1 & = & 7 \\ 6 & \text{x} & 8^1 & = & 6 & \text{x} & 8 & = & 48 \\ 3 & \text{x} & 8^2 & = & 3 & \text{x} & 64 & = & 192 \\ & & & & & & & & \text{------} \\ & & & & & & & & \mathbf{247}_{10} \end{array}$$

Here's another example to determine the value of the octal number 1601_8:

1 6 0 1$_8$

$$\begin{array}{ccccccccr} 1 & \text{x} & 8^0 & = & 1 & \text{x} & 1 & = & 1 \\ 0 & \text{x} & 8^1 & = & 0 & \text{x} & 8 & = & 0 \\ 6 & \text{x} & 8^2 & = & 6 & \text{x} & 64 & = & 384 \\ 1 & \text{x} & 8^3 & = & 1 & \text{x} & 512 & = & 512 \\ & & & & & & & & \text{------} \\ & & & & & & & & \mathbf{897}_{10} \end{array}$$

TRY THIS: *Convert the following octal numbers to their decimal equivalents:*

(a) **5 3 6_8**

(b) **1 1 6 3_8**

Converting a Decimal Number to an Octal Number

To convert a decimal number to its octal equivalent, the remainder method (the same method used in converting a decimal number to its binary equivalent) can be used. To review, the remainder method involves the following four steps:

(1) Divide the decimal number by the base (in the case of octal, divide by 8).
(2) Indicate the remainder to the right.
(3) Continue dividing into each quotient (and indicating the remainder) until the divide operation produces a zero quotient.
(4) The base 8 number is the numeric remainder reading from the last division to the first (if you start at the bottom, the answer will read from top to bottom).

Example 1: Convert the decimal number 465_{10} to its octal equivalent:

	$\begin{array}{r} 0 \\ 8\overline{)7} \end{array}$	**7**	(3)	Divide 8 into 7. The quotient is 0 with a remainder of 7, as indicated. Since the quotient is 0, stop here.
	$\begin{array}{r} 7 \\ 8\overline{)58} \end{array}$	**2**	(2)	Divide 8 into 58 (the quotient from the previous division). The quotient is 7 with a remainder of 2, indicated on the right.
START HERE $\Rightarrow$	$\begin{array}{r} 58 \\ 8\overline{)465} \end{array}$	**1**	(1)	Divide 8 into 465. The quotient is 58 with a remainder of 1; indicate the 1 on the right.

The answer, reading the remainders from top to bottom, is **721**, so $\mathbf{465_{10} = 721_8}$.

Example 2: Convert the decimal number 2548_{10} to its octal equivalent:

	$\begin{array}{r} 0 \\ 8\,\overline{)\,4} \end{array}$	**4**	(4)	Divide 8 into 4. The quotient is 0 with a remainder of 4, as indicated. Since the quotient is 0, stop here.
	$\begin{array}{r} 4 \\ 8\,\overline{)\,39} \end{array}$	**7**	(3)	Divide 8 into 39. The quotient is 4 with a remainder of 7, indicated on the right.
	$\begin{array}{r} 39 \\ 8\,\overline{)\,318} \end{array}$	**6**	(2)	Divide 8 into 318 (the quotient from the previous division). The quotient is 39 with a remainder of 6, indicated on the right.
START HERE $\Rightarrow$	$\begin{array}{r} 318 \\ 8\,\overline{)\,2548} \end{array}$	**4**	(1)	Divide 8 into 2548. The quotient is 318 with a remainder of 4; indicate the 4 on the right.

The answer, reading the remainders from top to bottom, is **4764**, so $\mathbf{2548_{10} = 4764_8}$.

TRY THIS: *Convert the following decimal numbers to their octal equivalents:*

(a) $\mathbf{3002_{10}}$

(b) $\mathbf{6512_{10}}$

Octal Addition

Octal addition is performed just like decimal addition, except that if a column of two addends produces a sum greater than 7, you must subtract 8 from the result, put down that result, and carry the 1. Remember that there are no such digits as "8" and "9" in the octal system, and that $\mathbf{8_{10} = 10_8}$, $\mathbf{9_{10} = 11_8}$, etc.

Example 1: Add $543_8 + 121_8$ (no carry required):

$$
\begin{array}{cccc}
 & 5 & 4 & 3 \\
+ & 1 & 2 & 1 \\
\hline
 & 6 & 6 & 4
\end{array}
$$

Example 2: Add $7652_8 + 4574_8$ (carries required):

$$
\begin{array}{cccccc}
 & & {}^{1} & {}^{1} & & \\
 & & 7 & 6 & 5 & 2 \\
+ & & 4 & 5 & 7 & 4 \\
\hline
 & & \scriptstyle 12 - 8 = 4 & \scriptstyle 12 - 8 = 4 & \scriptstyle 12 - 8 = 4 & \\
 & \mathbf{1} & \mathbf{4} & \mathbf{4} & \mathbf{4} & \mathbf{6}
\end{array}
$$

TRY THIS: *Perform the following octal additions:*

(a)
$$
\begin{array}{ccccc}
 & 5 & 4 & 3 & 0 \\
+ & 3 & 2 & 4 & 1 \\
\hline
\end{array}
$$

(b)
$$
\begin{array}{ccccc}
 & 6 & 4 & 0 & 5 \\
+ & 1 & 2 & 3 & 4 \\
\hline
\end{array}
$$

Octal Subtraction

We will use the complement method to perform octal subtraction. The steps for subtracting two octal numbers are as follows:

(1) Compute the <u>seven's complement</u> of the subtrahend by subtracting each digit of the subtrahend by 7.

(2) Add 1 to the seven's complement of the subtrahend to get the <u>eight's complement</u> of the subtrahend.

(3) Add the eight's complement of the subtrahend to the minuend and drop the high-order 1. This is your difference.

Example 1: Compute $\mathbf{7526_8 - 3142_8}$

(1) Compute the seven's complement of $\mathbf{3142_8}$ by subtracting each digit from 7:

$$\begin{array}{r}7\\-\;3\\\hline 4\end{array}\qquad\begin{array}{r}7\\-\;1\\\hline 6\end{array}\qquad\begin{array}{r}7\\-\;4\\\hline 3\end{array}\qquad\begin{array}{r}7\\-\;2\\\hline 5\end{array}$$

(2) Add 1 to the seven's complement of the subtrahend, giving the eight's complement of the subtrahend:

$$\begin{array}{cccc}4&6&3&5\\&&+&1\\\hline 4&6&3&6\end{array}$$

(3) Add the eight's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{ccccc}&{\scriptstyle 1}&&{\scriptstyle 1}&\\&7&5&2&6\\+&4&6&3&6\\\hline&{\scriptstyle 12-8=4}&{\scriptstyle 11-8=3}&&{\scriptstyle 12-8=4}\\\cancel{\mathbf{1}}&\mathbf{4}&\mathbf{3}&\mathbf{6}&\mathbf{4}\end{array}$$

So $\mathbf{7526_8 - 3142_8 = 4364_8}$

The answer can be checked by making sure that $\mathbf{3142_8 + 4364_8} = \mathbf{7526_8}$ **.**

Example 2: Compute $\mathbf{545_8 - 14_8}$

(1) Compute the seven's complement of $\mathbf{14_8}$ (putting in a leading zero to make it a three-digit number) by subtracting each digit from 7:

$$\begin{array}{r} 7 \\ -\ \underline{0} \\ 7 \end{array} \qquad \begin{array}{r} 7 \\ -\ \underline{1} \\ 6 \end{array} \qquad \begin{array}{r} 7 \\ -\ \underline{4} \\ 3 \end{array}$$

(2) Add 1 to the seven's complement of the subtrahend, giving the eight's complement of the subtrahend:

$$\begin{array}{ccc} 7 & 6 & 3 \\ & + & 1 \\ \hline 7 & 6 & 4 \end{array}$$

(3) Add the eight's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{cccc} & {}^{1} & {}^{1} & \\ & 5 & 4 & 5 \\ + & 7 & 6 & 4 \\ \hline & \scriptstyle 13 - 8 = 5 & \scriptstyle 11 - 8 = 3 & \scriptstyle 9 - 8 = 1 \\ \mathbf{\cancel{1}} & \mathbf{5} & \mathbf{3} & \mathbf{1} \end{array}$$

So $\mathbf{545_8 - 14_8 = 531_8}$

The answer can be checked by making sure that $\mathbf{14_8 + 531_8 = 545_8}$ **.**

<u>TRY THIS:</u> *Solve the following octal subtraction problems using the complement method:*

(a) $\mathbf{6776_8 - 4337_8 =}$

(b) $\mathbf{5434_8 - 3556_8 =}$

The Hexadecimal Number System

The **hexadecimal (base 16) number system** is a positional number system as are the decimal number system and the binary number system. Recall that in any positional number system, regardless of the base, the highest numerical symbol always has a value of one less than the base. Furthermore, one and only one symbol must ever be used to represent a value in any position of the number.

For number systems with a base of 10 or less, a combination of Arabic numerals can be used to represent any value in that number system. The decimal number system uses the Arabic numerals 0 through 9; the binary number system uses the Arabic numerals 0 and 1; the octal number system uses the Arabic numerals 0 through 7; and any other number system with a base less than 10 would use the Arabic numerals from 0 to one less than the base of that number system.

However, if the base of the number system is greater than 10, more than 10 symbols are needed to represent all of the possible positional values in that number system. The hexadecimal number system uses not only the Arabic numerals 0 through 9, but also uses the letters A, B, C, D, E, and F to represent the equivalent of 10_{10} through 15_{10}, respectively.

For reference, the following table shows the decimal numbers 0 through 31 with their hexadecimal equivalents:

Decimal	Hexadecimal	Decimal	Hexadecimal
0	0	16	10
1	1	17	11
2	2	18	12
3	3	19	13
4	4	20	14
5	5	21	15
6	6	22	16
7	7	23	17
8	8	24	18
9	9	25	19
10	A	26	1A
11	B	27	1B
12	C	28	1C
13	D	29	1D
14	E	30	1E
15	F	31	1F

The same principles of positional number systems we applied to the decimal, binary, and octal number systems can be applied to the hexadecimal number system. However, the base of the hexadecimal number system is 16, so each position of the hexadecimal number represents a successive power of 16. From right to left, the successive positions of the hexadecimal number are weighted 1, 16, 256, 4096, 65536, etc.:

$16^0 = 1$ $16^1 = 16$ $16^2 = 256$ $16^3 = 4096$ $16^4 = 65536$

Converting a Hexadecimal Number to a Decimal Number

We can use the same method that we used to convert binary numbers and octal numbers to decimal numbers to convert a hexadecimal number to a decimal number, keeping in mind that we are now dealing with base 16. From right to left, we multiply each digit of the hexadecimal number by the value of 16 raised to successive powers, starting with the zero power, then sum the results of the multiplications. Remember that if one of the digits of the hexadecimal number happens to be a letter A through F, then the corresponding value of 10 through 15 must be used in the multiplication.

Example 1: Convert the hexadecimal number $20B3_{16}$ to its decimal equivalent.

2 0 B 3_{16}

3	x	16^0 =	3	x	1	=	3
11	x	16^1 =	11	x	16	=	176
0	x	16^2 =	0	x	256	=	0
2	x	16^3 =	2	x	4096	=	8192

							8371 $_{10}$

Example 2: Convert the hexadecimal number $12AE5_{16}$ to its decimal equivalent.

1 2 A E 5_{16}

5	x	16^0 =	5	x	1	=	5
14	x	16^1 =	14	x	16	=	224
10	x	16^2 =	10	x	256	=	2560
2	x	16^3 =	2	x	4096	=	8192
1	x	16^4 =	1	x	65536	=	65536

							76517 $_{10}$

TRY THIS. *Convert the following hexadecimal numbers to their decimal equivalents:*

(a) **2 4 3 F_{16}**

(b) **B E E F_{16}**

<u>Converting a Decimal Number to a Hexadecimal Number</u>

To convert a decimal number to its hexadecimal equivalent, the <u>remainder method</u> (the same method used in converting a decimal number to its binary equivalent) can be used. To review, the remainder method involves the following four steps:

(1) Divide the decimal number by the base (in the case of hexadecimal, divide by 16).
(2) Indicate the remainder to the right. If the remainder is between 10 and 15, indicate the corresponding hex digit A through F.
(3) Continue dividing into each quotient (and indicating the remainder) until the divide operation produces a zero quotient.
(4) The base 16 number is the numeric remainder reading from the last division to the first (if you start at the bottom, the answer will read from top to bottom).

Example 1: Convert 9263_{10} to its hexadecimal equivalent:

	$16\,\overline{)\,2}$ = 0	**2**	(4)	Divide 16 into 2. The quotient is 0 with a remainder of 2, as indicated. Since the quotient is 0, stop here.
	$16\,\overline{)\,36}$ = 2	**4**	(3)	Divide 16 into 36. The quotient is 2 with a remainder of 4, indicated on the right.
	$16\,\overline{)\,578}$ = 36	**2**	(2)	Divide 16 into 578 (the quotient from the previous division). The quotient is 36 with a remainder of 2, indicated on the right.
START HERE $\Rightarrow$	$16\,\overline{)\,9263}$ = 578	**F**	(1)	Divide 16 into 9263. The quotient is 578 with a remainder of 15; so indicate the hex equivalent, "F", on the right.

The answer, reading the remainders from top to bottom, is **242F**, so $\mathbf{9263_{10} = 242F_{16}}$.

Example 2: Convert 4259_{10} to its hexadecimal equivalent:

	$\begin{array}{r} 0 \\ 16\overline{)\,1} \end{array}$	**1**	(4)	Divide 16 into 1. The quotient is 0 with a remainder of 1, as indicated. Since the quotient is 0, stop here.
	$\begin{array}{r} 1 \\ 16\overline{)\,16} \end{array}$	**0**	(3)	Divide 16 into 16. The quotient is 1 with a remainder of 0, indicated on the right.
	$\begin{array}{r} 16 \\ 16\overline{)\,266} \end{array}$	**A**	(2)	Divide 16 into 266 (the quotient from the previous division). The quotient is 16 with a remainder of 10, so the hex equivalent "A" is indicated on the right.
START HERE ⇒	$\begin{array}{r} 266 \\ 16\overline{)\,4259} \end{array}$	**3**	(1)	Divide 16 into 4259. The quotient is 266 with a remainder of 3; so indicate 3 on the right.

The answer, reading the remainders from top to bottom, is **10A3**, so $\mathbf{4259_{10} = 10A3_{16}}$.

<u>**TRY THIS:**</u> *Convert the following decimal numbers to their hexadecimal equivalents:*

(a) $\mathbf{69498_{10}}$

(b) $\mathbf{114267_{10}}$

<u>Hexadecimal Addition</u>

One consideration is that if the result of an addition is between 10 and 15, the corresponding letter A through F must be written in the result:

	1	**9**	**5**
+	**3**	**1**	**9**
	4	**A**	**E**

In the example above, 5 + 9 = 14, so an "E" was written in that position; 9 + 1 = 10, so an "A" was written in that position.

A second consideration is that if either of the addends contains a letter A through F, convert the letter to its decimal equivalent (either by memory or by writing it down) and then proceed with the addition:

	3	**A** 10	**2**
+	**4**	**1**	**C** 12
	7	**B**	**E**

A third consideration is that if the result of an addition is greater than 15, you must subtract 16 from the result of that addition, put down the difference of that subtraction for that position, and carry a 1 over to the next position, as shown below:

		1	
	D 13	**E** 14	**B** 11
+	**1**	**0**	**E** 14
			11 + 14 = 25 25 - 16 = 9
	E	**F**	**9**

In the example above, when B_{16} (11_{10}) was added to E_{16} (14_{10}), the result was 25_{10}. Since 25_{10} is greater than 15_{10}, we subtracted 16_{10} from the 25_{10} to get 9_{10}. We put the 9 down and carried the 1 over to the next position.

Here is another example with carries:

	1		1	
	8	**F** 15	**9**	**7**
+	**D** 13	**5**	**4**	**C** 12
	1 + 8+ 13 = 22 22- 16 = 6	15 + 5 = 20 20 - 16 = 4		7 + 12 = 19 19 - 16 = 3
1	**6**	**4**	**E**	**3**

<u>TRY THIS:</u> *Perform the following hexadecimal additions:*

(a)

	B	**E**	**D**
+	**2**	**A**	**9**

(b)

	D	**E**	**A**	**D**
+	**B**	**E**	**E**	**F**

<u>Hexadecimal Subtraction</u>

We will use the complement method to perform hexadecimal subtraction. The steps for subtracting two hexadecimal numbers are as follows:

(1) Compute the <u>15's complement</u> of the subtrahend by subtracting each digit of the subtrahend by 15.

(2) Add 1 to the 15's complement of the subtrahend to get the <u>16's complement</u> of the subtrahend.

(3) Add the 16's complement of the subtrahend to the minuend and drop the high-order 1. This is your difference.

Example 1: Compute $\mathbf{ABED_{16}}$ **-** $\mathbf{1FAD_{16}}$

(1) Compute the 15's complement of $\mathbf{1FAD_{16}}$ by subtracting each digit from 15:

$$\begin{array}{rrrr} 15 & 15 & 15 & 15 \\ -\ \underline{\ \ 1} & -\ \underline{\ \ F} & -\ \underline{\ \ A} & -\ \underline{\ \ D} \\ E & 0 & 5 & 2 \end{array}$$

(2) Add 1 to the 15's complement of the subtrahend, giving the 16's complement of the subtrahend:

$$\begin{array}{cccc} E & 0 & 5 & 2 \\ & & + & 1 \\ \hline E & 0 & 5 & 3 \end{array}$$

(3) Add the 16's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{ccccc} {}^{1} & & {}^{1} & {}^{1} & \\ & A & B & E & D \\ + & E & 0 & 5 & 3 \\ \hline & {\scriptstyle 24 - 16 = 8} & & {\scriptstyle 20 - 16 = 4} & {\scriptstyle 16 - 16 = 0} \\ \cancel{\mathbf{1}} & \mathbf{8} & \mathbf{C} & \mathbf{4} & \mathbf{0} \end{array}$$

So $\mathbf{ABED_{16} - 1FAD_{16} = 8C40_{16}}$

The answer can be checked by making sure that $\mathbf{1FAD_{16} + 8C40_{16} = ABED_{16}}$.

Example 2: Compute $\mathbf{FEED_{16} - DAF3_{16}}$

(1) Compute the 15's complement of $\mathbf{DAF3_{16}}$ by subtracting each digit from 15:

$$\begin{array}{r} 15 \\ -\ \underline{\ \ D} \\ 2 \end{array} \qquad \begin{array}{r} 15 \\ -\ \underline{\ \ A} \\ 5 \end{array} \qquad \begin{array}{r} 15 \\ -\ \underline{\ \ F} \\ 0 \end{array} \qquad \begin{array}{r} 15 \\ -\ \underline{\ \ 3} \\ C \end{array}$$

(2) Add 1 to the 15's complement of the subtrahend, giving the 16's complement of the subtrahend:

$$\begin{array}{cccc} 2 & 5 & 0 & C \\ & & + & 1 \\ \hline 2 & 5 & 0 & D \end{array}$$

(3) Add the 16's complement of the subtrahend to the minuend and drop the high-order 1, giving the difference:

$$\begin{array}{ccccc} {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & \\ & F & E & E & D \\ + & 2 & 5 & 0 & D \\ \hline & {\scriptstyle 18 - 16 = 2} & {\scriptstyle 19 - 16 = 3} & & {\scriptstyle 26 - 16 = 10} \\ \cancel{\mathbf{1}} & \mathbf{2} & \mathbf{3} & \mathbf{F} & \mathbf{A} \end{array}$$

So $\mathbf{FEED_{16} - DAF3_{16} = 23FA_{16}}$

The answer can be checked by making sure that $\mathbf{DAF3_{16} + 23FA_{16} = FEED_{16}}$.

TRY THIS: *Solve the following hexadecimal subtraction problems using the complement method:*

(a) $\mathbf{98AE_{16} - 1FEE_{16} =}$

(b) $\mathbf{B6A1_{16} - 8B12_{16} =}$

Converting Binary-to-Hexadecimal or Hexadecimal-to-Binary

Converting a binary number to its hexadecimal equivalent or vice-versa is a simple matter. Four binary digits are equivalent to one hexadecimal digit, as shown in the table below:

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

To convert from binary to hexadecimal, divide the binary number into groups of 4 digits **starting on the right of the binary number.** If the leftmost group has less than 4 bits, put in the necessary number of leading zeroes on the left. For each group of four bits, write the corresponding single hex digit.

Example 1: $\mathbf{1101001101110111_2 = ?_{16}}$

Answer:

Bin:	**1101**	**0011**	**0111**	**0111**
Hex:	**D**	**3**	**7**	**7**

Example 2: $\mathbf{101101111_2 = ?_{16}}$

Answer:

Bin:	**0001**	**0110**	**1111**
Hex:	**1**	**6**	**F**

To convert from hexadecimal to binary, write the corresponding group of four binary digits for each hex digit.

Example 1: $\mathbf{1BE9_{16} = ?_2}$

Answer:

Hex:	**1**	**B**	**E**	**9**
Bin:	**0001**	**1011**	**1110**	**1001**

Example 2: $\mathbf{B0A_{16} = ?_2}$

Answer:

Hex:	**B**	**0**	**A**
Bin:	**1011**	**0000**	**1010**

Converting Binary-to-Octal or Octal-to-Binary

Converting a binary number to its octal equivalent or vice-versa is a simple matter. Three binary digits are equivalent to one octal digit, as shown in the table below:

Binary	**Octal**
000	**0**
001	**1**
010	**2**
011	**3**
100	**4**
101	**5**
110	**6**
111	**7**

To convert from binary to octal, divide the binary number into groups of 3 digits **starting on the right of the binary number.** If the leftmost group has less than 3 bits, put in the necessary number of leading zeroes on the left. For each group of three bits, write the corresponding single octal digit.

Example 1:	**1101 001101110111**$_2$ **= ?**$_8$						
Answer:	**Bin:**	**001**	**101**	**001**	**101**	**110**	**111**
	Oct:	**1**	**5**	**1**	**5**	**6**	**7**

Example 2:	**101101111**$_2$ **= ?**$_8$			
Answer:	**Bin:**	**101**	**101**	**111**
	Oct:	**5**	**5**	**7**

To convert from octal to binary, write the corresponding group of three binary digits for each octal digit.

Example 1:	**1764**$_8$ **= ?**$_2$				
Answer:	**Oct:**	**1**	**7**	**6**	**4**
	Bin:	**001**	**111**	**110**	**100**

Example 2:	**731**$_8$ **= ?**$_2$			
Answer:	**Oct:**	**7**	**3**	**1**
	Bin:	**111**	**011**	**001**

Computer Character Sets and Data Representation

Each character is stored in the computer as a **byte**. Since a byte consists of eight bits, there are 2^8, or 256 possible combinations of bits within a byte, numbered from 0 to 255. There are two commonly used character sets that determine which particular pattern of bits will represent which character: **ASCII** (pronounced "as-key", stands for **American Standard Code for Information Interchange**) is used on most minicomputers and PCs, and **EBCDIC** (pronounced "eb-suh-dick", stands for **Extended Binary Coded Decimal Interchange Code**) is used on IBM mainframes.

The ASCII Character Set
(Characters 32 through 127)

Shown below are characters 32 through 127 of the ASCII character set, which encompass the most commonly displayed characters (letters, numbers, and special characters). Characters 0 through 31 are used primarily as "control characters" (characters that control the way hardware devices, such as modems, printers, and keyboards work) - for example, character number 12 is the "form feed" character, which when sent to a printer, causes the printer to start a new page. Characters 128 through 255 are other special characters, such as symbols for foreign currency, Greek letters, and "box-drawing" characters that, for example, are used to make dialog boxes in DOS-text based (non-GUI) applications such as MS-DOS EDIT and QBASIC.

Decimal	Hex	Char
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Decimal	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Decimal	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

Data and instructions both "look" the same to the computer - they are both represented as strings of bits. The way a particular pattern of bits is treated by the computer depends on the context in which the string of bits is being used. For example, the bit pattern 000000001 (hex 01) can be interpreted by the computer in any of three ways: when it is interpreted as a machine language instruction, it causes the contents of two registers to be added together; when it is interpreted as a control code, it signifies a "start of heading" which precedes text in a data transmission; and when it is interpreted as a character (on IBM PCs), it shows up as a "happy face".

And in addition to differentiating between instructions and data, there are different data types, or formats, which the computer treats in specific ways. In the ASCII character chart on the previous page, when the computer is using the bit patterns in a data "character" context, character 65 (hex 41 or binary 01000001) is treated as a capital "A". Likewise, when a data item such a zip code or phone number is stored, although it consists only of numeric digits, no arithmetic will be performed with that data item, so it is also suitable for being stored in "character" format. So a data item containing the zip code "90210" would be stored as (in hex) 3930323130.

The computer cannot perform arithmetic on numeric quantities that are stored in character format. For example, if you wanted to add the number 125, the computer could not add it if it was stored as (hex) 313235. It would have to be stored as (or converted to) a numeric format that the computer can work with - either "integer" format or "floating point" format.

In the pages that follow, we will look at how the computer stores various data items and how we can look at that internal representation via a memory "dump".

The first order of business is to create some sample data. The following QBASIC program causes a file consisting of one record (consisting of fields of different data types) to be written to a file called "TESTFILE.DAT" and stored on disk:

```
TYPE MyRecord
    MyName   AS STRING * 16
    PosInt   AS INTEGER
    NegInt   AS INTEGER
    PosSing  AS SINGLE
    NegSing  AS SINGLE
END TYPE

DIM TestRecord AS MyRecord

OPEN "TESTFILE.DAT" FOR RANDOM AS #1 LEN = 28
TestRecord.MyName = "HARRY P. DODSON"
TestRecord.PosInt = 25
TestRecord.NegInt = -2
TestRecord.PosSing = 6.75
TestRecord.NegSing = -4.5
PUT #1, , TestRecord
CLOSE #1
END
```

This code here sets up a record structure with fields defined in various formats.

Opens the DOS file into which the record data will be stored.

Fills the fields of the record with test data.

Writes a record to the file, closes the file, and ends the program.

The QBASIC program listed above defines a record 28 bytes long, with the fields mapped out as follows:

A field called **MyName**, defined as **STRING * 16**. A STRING data type stores data in "character" format, using the ASCII characters as shown on the chart a couple of pages back. A field defined as STRING * *n* defines a character field *n* bytes long (16 bytes in this case). This then defines the first 16 bytes of the 28 byte record.

The next two fields are **INTEGER** fields, called **PosInt** and **NegInt**. A QBASIC INTEGER field takes up two bytes of storage, so these two fields define the next 4 bytes of the record. Only integers, or whole numbers, can be stored in INTEGER type fields. INTEGER fields store values in "signed binary" format, where the high-order (leftmost) bit designates the sign of the number: a "zero" high-order bit signifies a positive number, a "one" high-order bit signifies a negative number.

The bits of a positive integer field are arranged as you might expect: the value 5 would be stored as 00000000 00000101 (or 00 05 in hex). But a negative value is stored in two's complement notation - so the value -5 would be stored as 11111111 11111011 (or FF FB in hex). One more twist: the PC stores integer fields with the bytes arranged from right to left - NOT left to right as you might expect - so the 5 in the example above would actually show up on a dump as 05 00 in hex, and the -5 would show up as FB FF in hex.

The last two fields are defined as **SINGLE** fields, called **PosSing** and **NegSing**. A QBASIC SINGLE field takes up four bytes of storage, so these last two fields occupy the last eight bytes of the 28 byte record. SINGLE fields store numeric data in "floating point" format, which permits "real" numbers (numbers that can have digits after the decimal point) to be stored. Floating point format is the most complex data type to understand; it will be explained in the context of the data dump shown in the next section. Floating point fields are also stored with its bytes arranged from right to left.

QBASIC has two other data types not used in this example. They are **LONG** and **DOUBLE**. LONG is a four-byte integer counterpart to the two-byte INTEGER data type, and DOUBLE is an eight-byte floating point counterpart to the four-byte SINGLE data type.

After the sample QBASIC program was executed, a file called TESTFILE.DAT was created and placed in the default DOS directory. The file contained 28 bytes, for the storage of one record written out by the program. The contents of this file (or any file) can be "dumped" by the DOS **DEBUG** program. Below is a screen shot of that DOS session (the characters in **bold** represent the internal binary representation of the 28 bytes of the file, in hex):

```
F:\CLC\CP110>debug testfile.dat
-d
2F24:0100  48 41 52 52 59 20 50 2E-20 44 4F 44 53 4F 4E 20   HARRY P. DODSON
2F24:0110  19 00 FE FF 00 00 D8 40-00 00 90 C0 34 00 13 2F   .......@....4../
2F24:0120  00 DB D2 D3 E0 03 F0 8E-DA 8B C7 16 C2 B6 01 16   ................
2F24:0130  C0 16 F8 8E C2 AC 8A D0-00 00 4E AD 8B C8 46 8A   ..........N...F.
2F24:0140  C2 24 FE 3C B0 75 05 AC-F3 AA A0 0A EB 06 3C B2   .$.<.u........<.
2F24:0150  75 6D 6D 13 A8 01 50 14-74 B1 BE 32 01 8D 8B 1E   umm...P.t..2....
2F24:0160  8E FC 12 A8 33 D2 29 E3-13 8B C2 03 C3 69 02 00   ....3.)......i..
2F24:0170  0B F8 83 FF FF 74 11 26-01 1D E2 F3 81 00 94 FA   .....t.&........
-q

F:\CLC\CP110>
```

The first line of the screen shot shows that the DEBUG program was initiated from the DOS prompt (the filename TESTFILE.DAT was supplied to the DEBUG command). When DEBUG runs, all you see is a "hyphen" prompt. At the hyphen, any one of a number of one-character commands can be given. On the second line of the screen shot, you see that the command "d" (for "dump") was given. This caused a section of memory to be dumped (as shown on the next several lines of the screen shot). The dump command caused the 28-byte file to be loaded into memory. The data from that file, along with whatever other "junk" was occupying the subsequent bytes of memory was displayed. After the section of memory was dumped, the hyphen prompt returned, where the "q" (for "quit") command was given. This caused the DEBUG program to end, causing the DOS prompt to return.

The format of the DEBUG dump display is as follows: on the left of each line, the memory address (in hex) of the displayed data is given (in the screen shot above, the addresses are 2F24:0100, 2F24:0110, etc.). The main part of the line is the data being dumped, in hex format (16 bytes per line). The rightmost portion of each line is the ASCII character representation of the data being dumped; non-printable characters show up as dots (periods).

The entire first line of the dumped data shows the hex representation of the content of the 16-byte field **MyName** (into which the QBASIC program had placed "HARRY P. DODSON"). You should see that the hex 48 corresponds to the letter "H", 41 corresponds to "A", 52 corresponds to "R, and so on.

On the second line of the dumped data, the first two hex bytes are **19 00**. This represents the value 25 that the QBASIC program placed in the INTEGER field **PosInt**. Recall that integer fields are stored with their bytes arranged from right to left, so we should read these two bytes as **00 19** - and you know that $\mathbf{19_{16} = 25_{10}}$.

The next two hex bytes on the second line of dumped data are **FE FF**. This represents the value -2 that the QBASIC program placed in the INTEGER field **NegInt**. Read as **FF FE**, you can see that this represents the two's complement of 2 (recall that negative integers are stored in two's complement notation).

The next four bytes on the second line of dumped data are **00 00 D8 40**. This represents the value 6.75 that the QBASIC program placed in the SINGLE floating-point field **PosSing**. Recall that floating point fields, like integer fields, store their bytes from right to left, so we should read this as **40 D8 00 00**. As mentioned earlier, floating point is the most complex of the data types, so it requires a bit of a learning curve to understand. The following is a partial definition of the format from a computer manual:

> Floating-point numbers use the IEEE (Institute of Electrical and Electronic Engineers, Inc.) format. Values with a float type have 4 bytes, consisting of a sign bit, an 8-bit excess-127 binary exponent, and a 23-bit mantissa. The mantissa represents a number between 1.0 and 2.0. Since the high-order bit of the mantissa is always 1, it is not stored in the number.

In order to analyze what we see in the dump (i.e., "How do you get 6.75 from the hex bytes 40 D8 00 00?"), we must understand and apply the information above. Before we do that, we must understand the concept of <u>fractional binary numbers</u>, or binary numbers that have a <u>binary point</u>.

In the earlier parts of this document, we examined only "whole" binary numbers. We learned that each digit of the binary number represents a specific power of base 2 (from right to left, 2^0, 2^1, 2^2, etc.) When you have a binary number with a binary point (same principle as a decimal point in a base 10 number), such as 100.101, the digits to the right of the point are weighted as follows (from left to right): 2^{-1} = 1/2 = .5, 2^{-2} = 1/4 = .25, 2^{-3} = 1/8 = .125, etc. So $\mathbf{100.101_2}$ = 4 + 0 + 0 + .5 + 0 + .125 = $\mathbf{4.625_{10}}$.

Getting back to our number in the dump (40 D8 00 00), it will be necessary to expand this hex notation to binary to see how this represents the value 6.75.

4	**0**	**D**	**8**	**0**	**0**	**0**	**0**
0100	**0000**	**1101**	**1000**	**0000**	**0000**	**0000**	**0000**

The leftmost bit of this number is the <u>sign bit</u> - it is zero, so this means it is a positive number.

The next 8 bits, as stated in the IEEE definition of this format, is an "8-bit excess-127 binary exponent". These are the eight bits shown shaded in the figure above. "Excess-127" means that the value 127 must be subtracted from the value of these eight bits (the 127 is sometimes called a "bias quantity"). If you take the value of $\mathbf{10000001_2}$, you get the value $\mathbf{129_{10}}$. Subtract 127 from 129 and you get **2**. We'll see what to do with the 2 shortly.

Following the 8-bit exponent, as stated in the IEEE definition, we have a "23-bit mantissa". This is shown shaded below:

4	**0**	**D**	**8**	**0**	**0**	**0**	**0**
0100	**0000**	**1101**	**1000**	**0000**	**0000**	**0000**	**0000**

The mantissa represents the magnitude of the number being worked with. There is always an implied binary point in front of the stored mantissa, so the shaded portion above represents $\mathbf{.10110000000000000000000_2}$. Just as in a decimal number, we can drop insignificant trailing zeroes on the right of the point, so we get $\mathbf{.1011_2}$. The IEEE definition states "since the high-order bit of the mantissa is always 1, it is not stored in the number." This means we must tack on a leading one in front of this binary number, giving $\mathbf{1.1011_2}$.

The "2" we got from the previous step (when we were working with the exponent portion) <u>tells us where to move the binary point</u>: <u>*two*</u> *places to the right*. This gives us a final binary value of $\mathbf{110.11_2}$. Converting this number to decimal by applying the binary weights, we get 4 + 2 + 0 + .5 + .25 = **6.75**.

The next four bytes on the second line of dumped data (the last four bytes of the 28-byte record) are **00 00 90 C0**, which we should read as **C0 90 00 00**. This represents the value -4.5 that the QBASIC program placed in the SINGLE floating-point field **NegSing**. The hex code is analyzed in the following steps:

(1) Convert the hex code to binary:

C	**0**	**9**	**0**	**0**	**0**	**0**	**0**
1100	**0000**	**1001**	**0000**	**0000**	**0000**	**0000**	**0000**

The leftmost bit is 1, so we know that this is a negative number.

(2) Isolate the exponent portion (the next eight bits):

C	**0**	**9**	**0**	**0**	**0**	**0**	**0**
1100	**0000**	**1001**	**0000**	**0000**	**0000**	**0000**	**0000**

The decimal equivalent of these eight bits is 129. Subtract 127 from that to get 2.

(3) Isolate the mantissa:

C	**0**	**9**	**0**	**0**	**0**	**0**	**0**
1100	**0000**	**1001**	**0000**	**0000**	**0000**	**0000**	**0000**

Place **1.** in front of the mantissa and drop the trailing zeroes to get **1.001**.

(4) Move the binary point the number of places dictated by the result from step 2, which was 2. This gives us **100.1**.

(5) Convert the result from 4 to its decimal equivalent: $\mathbf{100.1_2}$ = 4 + 0 + 0 + .5 = $\mathbf{4.5_{10}}$.

The EBCDIC Character Set
(Selected Characters)

Shown below is a table of selected characters from the 255 character EBCDIC character set, used on IBM mainframes. The letter and number characters are found in the upper half of the range (128 through 255):

Decimal	Hex	Characters
64	40	blank space
129 thru 137	81 thru 89	lowercase "a" thru "i"
145 thru 153	91 thru 99	lowercase "j" thru "r"
162 thru 169	A2 thru A9	lowercase "s" thru "z"
193 thru 201	C1 thru C9	uppercase "A" thru "I"
209 thru 217	D1 thru D9	uppercase "J" thru "R"
226 thru 233	E2 thru E9	uppercase "S" thru "Z"
240 thru 249	F0 thru F9	digits 0 thru 9

A test to look at how the computer stores various data items and how we can look at that internal representation via a memory "dump" was performed on an IBM mainframe.

To create the sample data, the following COBOL program was compiled and executed, causing a file consisting of one record (consisting of fields of different data types) to be written to disk:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DUMPDATA.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTPUT-FILE
      ASSIGN TO UT-S-OUTPUT.

DATA DIVISION.
FILE SECTION.
FD  OUTPUT-FILE
    BLOCK CONTAINS 0 RECORDS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS OUTPUT-RECORD.
01  OUTPUT-RECORD     PIC X(80).

WORKING-STORAGE SECTION.
01  WS-OUTPUT-RECORD.
    05  O-NAME            PIC X(05)           VALUE 'HARRY'.
    05  FILLER            PIC X(03)           VALUE SPACES.
    05  O-POS-BINARY      PIC S9(4)    COMP   VALUE +25.
    05  FILLER            PIC X(02)           VALUE SPACES.
    05  O-NEG-BINARY      PIC S9(4)    COMP   VALUE -5.
    05  FILLER            PIC X(02)           VALUE SPACES.
    05  O-POS-PACKED      PIC S9(5)V99 COMP-3 VALUE +12345.67.
    05  O-NEG-PACKED      PIC S9(5)V99 COMP-3 VALUE -76543.21.

PROCEDURE DIVISION.
    OPEN OUTPUT OUTPUT-FILE.
    WRITE OUTPUT-RECORD FROM WS-OUTPUT-RECORD.
    CLOSE OUTPUT-FILE.
    STOP RUN.
```

The IBM mainframe supports the following data formats:

Character Stores data in "character" format, using the EBCDIC characters as shown on the chart on the previous page. In COBOL, a field defined as **PIC X(*n*)** defines a character field *n* bytes long.

Binary Similar to "integer" format on the PC, this data type stores signed numbers in binary format, with negative numbers represented in two's complement notation. A binary field can be either 2, 4, or 8 bytes long. Unlike the PC, fields of this type are not restricted to integer values, although that is what they are commonly used for. In COBOL, the word **COMP** in a data definition signifies a binary field, and **PIC S9(*n*)** determines its size. If *n* is 1 through 4, a 2-byte binary field is defined; if *n* is 5 through 9, a 4-byte binary field is defined; and if *n* is 10 through 18, an 8-byte binary field is defined.

Packed Decimal This format is native to IBM mainframes, NOT PCs. It stores numeric data as two decimal (hex) digits per byte, with the sign indicated as the rightmost hex digit of the rightmost byte. A positive value has a hex value of C (binary 1100); a negative value has a hex value of D (binary 1101). For example, a positive 123.45 would be stored as (in hex) **12 34 5C**; a negative 123.45 would be stored as (in hex) **12 34 5D**. There is no internal representation of the decimal point, this must be defined by the program that is processing that data. The length of a packed decimal field can vary from 1 to 8 bytes (allowing up to 15 total digits). In COBOL, the word **COMP-3** in a data definition signifies a packed-decimal field, and its **PIC** clause determines its size.

Floating Point Stores data in floating point format similar to the PC, but does not follow the IEEE format. Not used in this example.

Note: All formats store their bytes left to right (numeric formats are not stored right to left like they are on the PC).

Without delving further into COBOL syntax, trust that the following data definitions in the sample COBOL program cause the indicated data items to be stored in the test record:

This data definition	Stores
`O-NAME       PIC X(05)              VALUE 'HARRY'.`	The characters "HARRY" in a 5-byte character field.
`FILLER       PIC X(03)              VALUE SPACES.`	Blank spaces in a 3-byte character field.
`O-POS-BINARY PIC S9(4)    COMP      VALUE +25.`	The value 25 in a 2-byte binary field.
`FILLER       PIC X(02)              VALUE SPACES.`	Blank spaces in a 2-byte character field.
`O-NEG-BINARY PIC S9(4)    COMP      VALUE -5.`	The value -5 in a 2-byte binary field.
`FILLER       PIC X(02)              VALUE SPACES.`	Blank spaces in a 2-byte character field.
`O-POS-PACKED PIC S9(5)V99 COMP-3 VALUE +12345.67.`	The value 12345.67 in a 4-byte packed decimal field.
`O-NEG-PACKED PIC S9(5)V99 COMP-3 VALUE -76543.21.`	The value -76543.21 in a 4-byte packed-decimal field.

This data defines a total of 24 bytes, but was actually written to an 80-byte record, causing 56 trailing blank spaces to be written at the end of the record.

The IBM mainframe utility program IDCAMS was used to produce a dump of this file. The output is shown below:

```
IDCAMS  SYSTEM SERVICES                                          TIME: 16:22:43        12/09/97     PAGE   2
LISTING OF DATA SET -SYS97343.T162200.RF103.BDG0AMS.TEMP
RECORD SEQUENCE NUMBER - 1
000000 C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040 *HARRY    ..  ..   ...@....     *
000020 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 *                               *
000040 40404040 40404040 40404040 40404040                                     *                               *

IDC0005I NUMBER OF RECORDS PROCESSED WAS 1
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
```

The format of the IDCAMS output is similar to that of the DOS DEBUG command, although you get 32 bytes worth of dumped data per line (instead of DEBUG's 16 bytes). The actual dumped data from the 80-byte file is highlighted in **bold** above. All the items of interest appear in the first line of the dumped data, reproduced below:

C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The first eight bytes (shown shaded below) is the EBCDIC character representation of the word "HARRY" followed by three blank spaces:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The next two bytes shows the binary field into which the value 25 was stored:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

Two bytes of blanks:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The 2-byte binary field where -5 is stored in two's complement format:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

Two more bytes of blanks:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The 4-byte packed-decimal field where 12345.67 is stored:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The 4-byte packed-decimal field where -76543.21 is stored:
C8C1D9D9 E8404040 00194040 FFFB4040 1234567C 7654321D 40404040 40404040

The rest of the record is all blank spaces.

ANSWERS
TO THE
"TRY THIS"
EXERCISES

<u>TRY THIS</u>: *Expand the following decimal number:*

5 1 3 0_{10}

Answer: (Expand as on previous page)

<u>TRY THIS:</u> *Convert the following binary numbers to their decimal equivalents:*

(a) **1 1 0 0 1 1 0_2**

Answer: 102_{10}

(b) **1 1 1 1 1 0 0 1_2**

Answer: 249_{10}

<u>TRY THIS:</u> *Convert the following decimal numbers to their binary equivalents:*

(a) **49_{10}** (b) **21_{10}**

Answers: (a) 11001_2 (b) 10101_2

<u>TRY THIS:</u> *Perform the following binary additions:*

(a)
$$\begin{array}{r} 1\ 0\ 0\ 1 \\ +\ \underline{1\ 1\ 0\ 0} \end{array}$$

(b)
$$\begin{array}{r} 1\ 1\ 1\ 0 \\ +\ \underline{1\ 1\ 0\ 1} \end{array}$$

(c)
$$\begin{array}{r} 1\ 0\ 1\ 0\ 1 \\ +\ \underline{0\ 0\ 1\ 1\ 1} \end{array}$$

(d)
$$\begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0 \\ +\ \underline{1\ 1\ 1\ 0\ 1\ 1} \end{array}$$

Answers:
(a) 10101 (b) 11011 (c) 11100 (d) 1110001

<u>TRY THIS:</u> *Solve the following subtraction problems using the complement method:*

(a) **5086 - 2993 =**

Answer: 2093_{10}

(b) **8391 - 255 =**

Answer: 8136_{10}

TRY THIS: *Solve the following binary subtraction problems using the complement method:*

(a) $\mathbf{11001101_2 - 10101010_2 =}$

(b) $\mathbf{100100_2 - 11101_2 =}$

Answers: (a) 100011_2
(b) 111_2

TRY THIS: *Convert the following octal numbers to their decimal equivalents:*

(a) **5 3 $\mathbf{6_8}$**

Answer: 350_{10}

(b) **1 1 6 $\mathbf{3_8}$**

Answer: 627_{10}

TRY THIS: *Convert the following decimal numbers to their octal equivalents:*

(a) $\mathbf{3002_{10}}$

Answer: 5762_8

(b) $\mathbf{6512_{10}}$

Answer: 14560_8

TRY THIS: *Perform the following octal additions:*

(a)

	5	4	3	0
+	3	2	4	1

Answer: 10671_8

(b)

	6	4	0	5
+	1	2	3	4

Answer: 7641_8

TRY THIS: *Solve the following octal subtraction problems using the complement method:*

(a) $\mathbf{6776_8 - 4337_8 =}$

Answer: 2437_8

(b) $\mathbf{5434_8 - 3556_8 =}$

Answer: 1656_8

TRY THIS. *Convert the following hexadecimal numbers to their decimal equivalents:*

(a) **2 4 3 F_{16}**

Answer: 9279_{10}

(b) **B E E F_{16}**

Answer: 48879_{10}

TRY THIS: *Convert the following decimal numbers to their hexadecimal equivalents:*

(a) **69498_{10}**

Answer: $10F7A_{16}$

(b) **114267_{10}**

Answer: $1BE5B_{16}$

TRY THIS: *Perform the following hexadecimal additions:*

(a)

	B	**E**	**D**
+	**2**	**A**	**9**

Answer: $E96_{16}$

(b)

	D	**E**	**A**	**D**
+	**B**	**E**	**E**	**F**

Answer: $19D9C_{16}$

TRY THIS: *Solve the following hexadecimal subtraction problems using the complement method:*

(a) **$98AE_{16}$ - $1FEE_{16}$ =**

Answer: $78C0_{16}$

(b) **$B6A1_{16}$ - $8B12_{16}$ =**

Answer: $2B8F_{16}$

TAKE-HOME QUIZ

Name: _________________ Date: _________________

DIRECTIONS: Perform the operations indicated below. Show all work neatly on separate sheet(s) of paper. Write the final answers in the spaces provided.

Questions 1-30 are worth 3 points each.

Convert the following binary numbers to their decimal equivalents:

(1) 10010110_2 = $\_\_\_\_\_\_\_\__{10}$

(2) 1001111_2 = $\_\_\_\_\_\_\_\__{10}$

(3) 10000001_2 = $\_\_\_\_\_\_\_\__{10}$

Find the following binary sums:

(4) $1010_2 + 101_2$ = $\_\_\_\_\_\_\_\_\_\__{2}$

(5) $1111_2 + 1_2$ = $\_\_\_\_\_\_\_\_\_\__{2}$

Find the following binary differences:

(6) $1010_2 - 111_2$ = $\_\_\_\_\_\_\_\_\_\__{2}$

(7) $11011_2 - 1110_2$ = $\_\_\_\_\_\_\_\_\_\__{2}$

Convert the following decimal numbers to their binary equivalents:

(8) 255_{10} = $\_\_\_\_\_\_\_\_\_\_\_\_\_\__{2}$

(9) 89_{10} = $\_\_\_\_\_\_\_\_\_\_\_\_\_\__{2}$

(10) 166_{10} = $\_\_\_\_\_\_\_\_\_\_\_\_\_\__{2}$

Convert the following hex numbers to their decimal equivalents:

(11) $C0A8_{16}$ = $\_\_\_\_\_\_\_\_\_\_\_\__{10}$

(12) $FACE_{16}$ = $\_\_\_\_\_\_\_\_\_\_\_\__{10}$

(13) $64F0_{16}$ = $\_\_\_\_\_\_\_\_\_\_\_\__{10}$

Find the following hexadecimal sums:

(14) $CAB_{16} + BED_{16}$ = $\_\_\_\_\_\_\_\__{16}$

(15) $3FF_{16} + 1_{16}$ = $\_\_\_\_\_\_\_\__{16}$

Find the following hexadecimal differences:

(16) $FADE_{16}$ - BAD_{16} = ________$_{16}$

(17) $ACE9_{16}$ - $9ACE_{16}$ = ________$_{16}$

Convert the following decimal numbers to their hex equivalents:

(18) 69000_{10} = ________$_{16}$

(19) 1998_{10} = ________$_{16}$

(20) 32768_{10} = ________$_{16}$

Convert the following octal numbers to their decimal equivalents:

(21) 332_8 = ________$_{10}$

(22) 6240_8 = ________$_{10}$

(23) 5566_8 = ________$_{10}$

Find the following octal sums:

(24) 765_8+ 123_8 = ________$_8$

(25) 631_8 + 267_8 = ________$_8$

Find the following octal differences:

(26) 700_8 - 16_8 = ________$_8$

(27) 750_8 – 270_8 = ________$_8$

Convert the following decimal numbers to their octal equivalents:

(28) 6700_{10} = ________$_8$

(29) 1001_{10} = ________$_8$

(30) 254_{10} = ________$_8$

Questions 31-40 are worth 1 point each.

Convert the following hex numbers to their binary equivalents:

(31) $1FB_{16}$ = ________$_2$

(32) ABC_{16} = ________$_2$

(33) $101F_{16}$ = ________$_2$

Convert the following binary numbers to their hex equivalents:

(34) 110110010_2 = ________$_{16}$

(35) 1101011001110_2 = ________$_{16}$

(36) 11000010111100_2 = ________$_{16}$

Convert the following binary numbers to their octal equivalents:

(37) 1101011001110_2 = ________$_8$

(38) 11000010111100_2 = ________$_8$

Convert the following octal numbers to their binary equivalents:

(39) 472_8 = ______________________$_2$

(40) 613_8 = ______________________$_2$

Extra Credit:
Let's say you wrote a QBASIC program that stored the following values in the indicated field types. Write the sequence of bytes that would show up for each field in a DEBUG dump:

<u>Field type</u>	<u>Value</u>	<u>Hex bytes</u>
STRING * 5	"HELLO"	______________________________
INTEGER	45	______________________________
INTEGER	-18	______________________________
SINGLE	9.25	______________________________

TAKE-HOME QUIZ ANSWERS

Number Systems Take-Home Quiz Answer Key

1. 150
2. 79
3. 129
4. 1111
5. 10000
6. 11
7. 1101
8. 1111 1111
9. 101 1001
10. 1010 0110
11. 49320
12. 64206
13. 25840
14. 1898
15. 400
16. EF31
17. 121B
18. 10D88
19. 7CE
20. 8000
21. 218
22. 3232
23. 2934
24. 1110
25. 1120
26. 662
27. 460
28. 15054
29. 1751
30. 376
31. 0001 1111 1011
32. 1010 1011 1100
33. 0001 0000 0001 1111
34. 1B2
35. 1ACE
36. 30BC
37. 15316
38. 30274
39. 100 111 010
40. 110 001 011

EXTRA CREDIT

1. 48 45 4C 4C 4F
2. 2D 00
3. EE FF
4. 00 00 14 41