13 Data Representation

In This Chapter

13.1 INTRODUCTION

Digital techniques have found their way into innumerable areas of technology, but the area of automatic digital computers is by far the most notable and most extensive. As you know, a computer is a system of hardware that performs arithmetic operations, manipulates data, and makes decisions.

In science, technology, business, and, in fact, most other fields of endeavour, we are constantly dealing with quantities; so are computers. Quantities are measured, monitored, recorded, manipulated arithmetically, observed, or in some other way utilized in most physical systems. In digital systems like computers, the quantities are represented by symbols called digits. Many number systems are in use in digital technology that represent the digits in various forms. The most common are the decimal, binary, octal, and hexadecimal systems. This chapter discusses these number systems and the physical representation of digits in computers.

13.2 DIGITAL NUMBER SYSTEMS

In digital representation, various number systems are used. The most common number systems used are *decimal*, *binary*, *octal*, and *hexadecimal* systems. Let us discuss these number systems briefly.

13.2.1 Decimal Number System

The *decimal system* is composed of 10 numerals or symbols (*Deca* means 10, that is why this system is called *decimal system*). These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ; using these symbols as *digits* of a number, we can express any quantity. The decimal system, also called the *base-10* system because it has 10 digits, has evolved naturally as a result of the fact that man has 10 fingers.

The decimal system is a *positional-value* system in which the value of a digit depends on its position. For example, consider the decimal number 729. We know that the digit 7 actually represents 7 *hundreds*, the 2 represents 2 *tens*, and the 9 represents 9 *units*. In essence, the 7 carries the most weight of three digits; it is referred to as the *most significant digit* (MSD). The 9 carries the least weight and is called the *least significant digit* (LSD).

Consider another example, 25.12. This number is actually equal to (2 *tens* plus 5 *units* plus 1 *tenths* plus 2 *hundredths*) *i.e.*, $2\times10+5\times1+1\times\frac{1}{10}+2\times\frac{1}{100}$. The decimal point is used to separate the integer and fractional parts of the number.

More rigorously, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in Fig. 13.1 where the number 2512.1971 is represented. The decimal point separates the positive powers of 10 from the negative powers. The number 2512.1971 is thus equal to

$$2\times10^{3}+5\times10^{2}+1\times10^{1}+2\times10^{0}+1\times10^{-1}+9\times10^{-2}+7\times10^{-3}+1\times10^{-4}$$

In general, any number is simply the sum of the products of each digit value and its positional value.

The squence of decimal numbers goes as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21.... See after 9, each successive number is a combination of two (or more) (unique) symbols of this system.

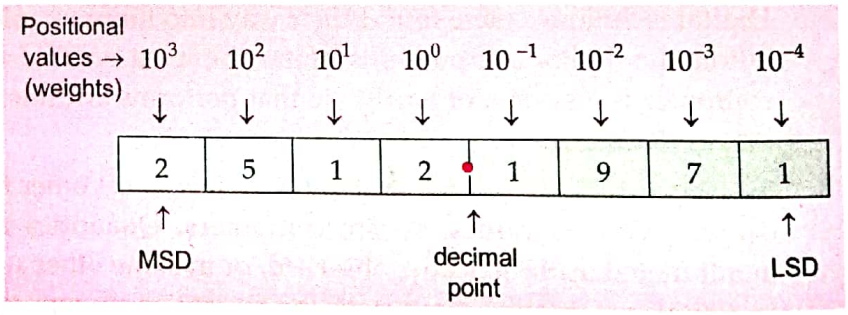


Figure 13.1 Positional values in decimal numbers.

13.2.2 Binary Number System

Unfortunately, the decimal number system does not lend itself to convenient implementation in digital systems. For example, it is very difficult to design electronic equipment so that it can work with 10 different voltage levels (each one representing one decimal character, 0 through 9). On the other hand, it is very easy to design simple, accurate electronic circuits that operate with only two voltage levels. For this reason, almost every digital system uses the binary number system (base 2) as the basic number system of its operations, although other systems are often used in conjunction with binary.

In the *binary system* there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems.

The binary system is also a positional-value system, wherein each binary digit has its own value or weight expressed as a power of 2. This is illustrated in Fig. 13.2.

Here, places to the left of the binary point (counterpart of the decimal point) are positive powers of 2 and places to the right are negative powers of 2. The number 1010.0101 is shown represented in the figure.

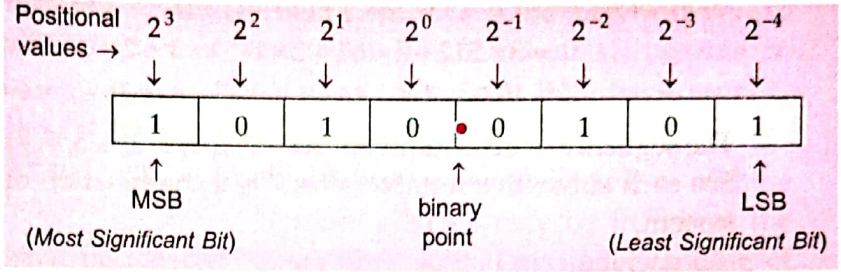


Figure 13.2 Positional values in binary numbers.

To find the decimal equivalent of above shown binary number, we simply take the sum of the products of each digit value (0 or 1) and its positional value :

$$1010.0101_2 = (1\times 2^3)+(0\times 2^2)+(1\times 2^1)+(0\times 2^0)+(0\times 2^{-1})+(1\times 2^{-2})+(0\times 2^{-3})+(1\times 2^{-4})$$
$$= 8+0+2+0+0+0.25+0+0.0625 = 10.3125_{10}$$

Notice in the preceding operation that subscripts (2 and 10) were used to indicate the base in which the particular number is expressed. This convention is used to avoid confusion whenever more than one number system is being employed.

In the binary system, the term *Binary digit* is often abbreviated to the term *bit*, which we'll use henceforth. As you see in Fig. 13.2, there are 4 bits to the left of the binary point, representing the integer part of the number, and 4 bits to the right of the binary point, representing the fractional part. The leftmost bit carries the largest weight and hence, is called the **most significant bit (MSB).** The rightmost bit carries the smallest weight, and hence called **least significant bit (LSB).**

The sequence of binary numbers goes as 00, 01, 10, 11, 100, 101, 110, 111, 1000, - - - - - -. The binary counting sequence has an important characteristic. The unit's bit (LSB) changes either from 0 to 1 or 1 to 0 with each count. The second bit (two's (2^1) position) stays at 0 for two counts, then at 1 for two counts, then at 0 for two counts, and so on. The third bit (four's (2^2) position) stays at 0 for four counts, then at 1 for four counts, and so on. The fourth bit (eight's (2^3) position) stays at 0 for eight counts, then at 1 for eight counts. If we wanted to count further we would add more places, and this pattern would continue with 0s and 1s alternating in groups of 2^{N-1}.

13.2.3 Octal Number System

The octal number system is very important in digital computer work. The octal number system has a base of *eight*, meaning that it has eight unique symbols : 0, 1, 2, 3, 4, 5, 6, and 7. Thus, each digit of an octal number can have any value from 0 to 7.

The octal system is also a positional value system, wherein each octal digit has its own value or weight expressed as a power of 8 (See Fig. 13.3). The places to the left of the octal

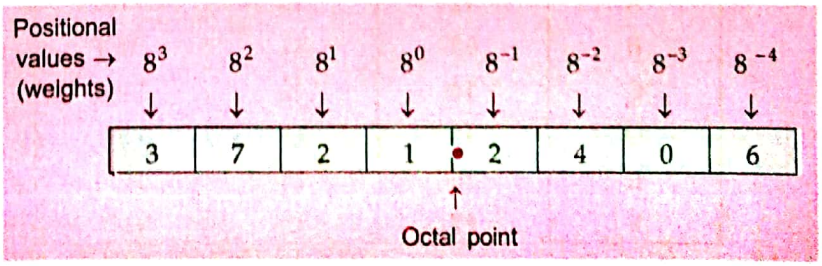


Figure 13.3 Positional values in Octal numbers.

point (counter-part of decimal point and binary point) are positive powers of 8 and places to the right are negative powers of 8. The number 3721.2406 is shown represented in the figure.

To find the decimal equivalent of above shown octal number, simply take the sum of products of each digit value and its positional value :

$$3721.2406_8 = (3\times 8^3)+(7\times 8^2)+(2\times 8^1)+(1\times 8^0)+(2\times 8^{-1})+(4\times 8^{-2})+(0\times 2^{-3})+(6\times 8^{-4})$$
$$= 3\times 512+7\times 64+2\times 8+1\times 1+2\times 0.125+4\times 0.015625+0+6\times 0.000244$$
$$= 1536+448+16+1+0.25+0.0625+0+0.001464 = 2001.313964_{10}$$

The sequence of octal numbers goes as 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22..... See each successive number after 7 is a combination of two or more unique symbols of octal system.

13.2.4 Hexadecimal Number System

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.

Just like above discussed systems, hexadecimal system is also a positional-value system, wherein each hexadecimal digit has its own value or weight expressed as a power of 16. (See Fig. 13.4). The digit positions in a hexadecimal number have weights as shown in Fig. 13.4.

Following table 13.1 shows the relationships between hexadecimal, octal, decimal and binary.

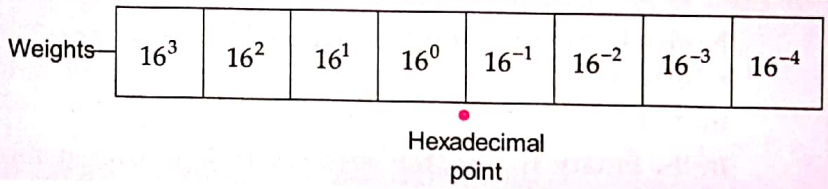


Figure 13.4 Positional values in hexadecimal numbers.

Table 13.1 *Relationship between Various Number Systems*

Hexadecimal	Octal	Decimal	Binary
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	10	8	1000
9	11	9	1001
A	12	10	1010
B	13	11	1011
C	14	12	1100
D	15	13	1101
E	16	14	1110
F	17	15	1111

Note that each hexadecimal digit represents a group of four binary digits. It is important to remember that *hex* (abbreviation for hexadecimal) digits A through F are equivalent to the decimal values 10 through 15.

13.3 NUMBER CONVERSIONS

The binary number system is the most important one in digital systems as it is very easy to implement in circuitry. The decimal system is important because it is universally used to represent quantities outside a digital system.

In addition to binary and decimal, octal and hexadecimal number systems find widespread application in digital systems. These number systems (octal and hexadecimal) provide an efficient means for representing large binary numbers. As we shall see, both these number systems have the advantage that they can be easily converted to and from binary.

Table 13.2 *Powers of 2*

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 487 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625

In a digital system, three or four of these number systems may be in use at the same time, so that an understanding of the system operation requires the ability to convert from one number system to another. This section discusses how to perform these conversions. So, let us discuss them one by one.

13.3.1 Decimal-to-Binary Conversion

There are *two* procedures for converting (integers) from decimal to binary.

The first method requires a table of powers of 2 (table 13.2). Because of this restriction, it is more useful for small numbers where these powers have been memorized. Starting with the decimal number to be evaluated, obtain the largest power of 2 from the table without exceeding the original number. Record this. Then subtract the table obtained number from the original number. Repeat the process for the remainder, and continue until the remainder is zero. Finally, add the binary numbers obtained from the table. The result is the answer.

EXAMPLE 13.1 *Convert 43_{10} to binary.*

Solution. From table 13.2, 32 is the largest number without exceeding 43.

$$32 = 100000$$

($2^5 = 32$ *i.e.*, put 1 at (5+1) th position and 0_s at all other positions. Thus $32_{10} = 100000_2$)

$$43 - 32 = 11$$

From the table, 8 (2^3) is the largest number without exceeding 11.

$$8 = 1000$$

($2^3 = 8$ *i.e.*, put 1 at $(3+1)^{th}$ position and 0_s at all other positions. Thus $8_{10} = 1000_2$)

$$11 - 8 = 3$$

From the table, 2 is the largest number without exceeding 3.

$$2 = 10$$

($2^1 = 2$, thus $2_{10} = 10_2$)

$$3 - 2 = 1$$

1 is the largest number without exceeding 1

$$1 = 1$$

Add all the binary numbers obtained *i.e.*,

$$\begin{array}{r} 100000 \\ 1000 \\ 10 \\ 1 \\ \hline 101011 \end{array}$$

Thus $43_{10} = 101011_2$

EXAMPLE 13.2 *Convert 200_{10} to binary.*

Solution. First largest number is 128.

Longest number

$$\begin{array}{ll} & 128 = 10000000 \\ 200 - 128 = 72 & 64 = \ 1000000 \\ 72 - 64 = 8 & 8 = \ \ \ 1000 \\ \hline & 200 = 11001000 \end{array}$$

Therefore, $200_{10} = 11001000_2$

The second method of converting decimal to binary is ***repeated-division*** method. In this method, the number is successively divided by 2 and its remainders recorded. The final binary result is obtained by assembling all the remainders, with the last remainder being the most significant bit (MSB).

EXAMPLE 13.3 *Convert 43_{10} to binary using repeated division method.*

Solution.

Repeated Division		Remainders	
2	43		
2	21	1	LSB
2	10	1	
2	5	0	Write in this order
2	2	1	
2	1	0	
	0	1	MSB

Reading the remainders from the bottom to the top,

$43_{10} = 101011_2$ (compare with result of example 13.1)

EXAMPLE 13.4 *Convert 200_{10} to binary using repeated division method.*

Solution.

		Remainders	
2	200		
2	100	0	LSB
2	50	0	
2	25	0	
2	12	1	
2	6	0	
2	3	0	
2	1	1	
	0	1	MSB

Reading the remainders from the bottom to the top, the result is : $200_{10} = 11001000_2$

On paper you may even compute the conversion as depicted through following example. (As you can see that this is just another way of representing the repeated division.)

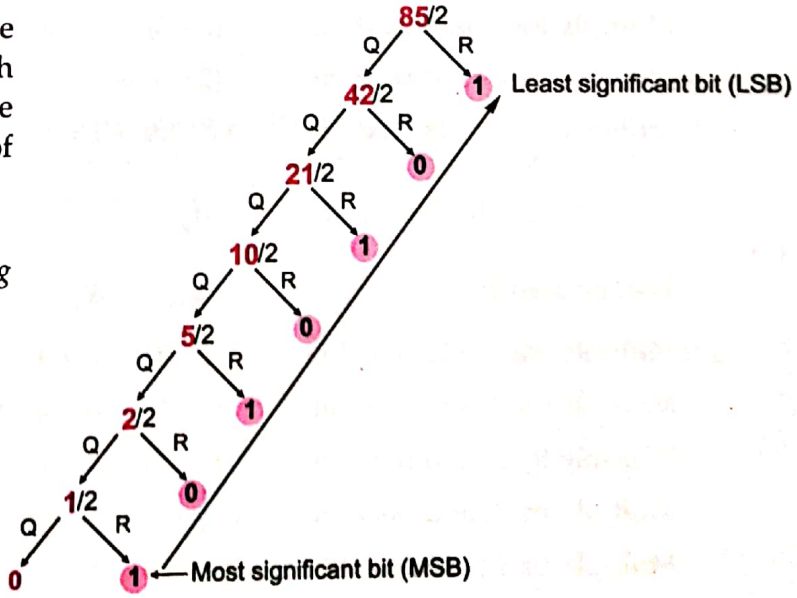
EXAMPLE 13.5 *Convert 85_{10} to binary using repeated division method.*

Solution. (See on the right)

Q – Quotient and

R – Remainder

$\therefore \quad 85_{10} = 1010101_2$



13.3.2 Binary-to-Decimal Conversion

The binary number system is a positional system where each binary digit (bit) carries a certain weight based on its position relative to the LSB. Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1. To illustrate, consider a binary number 11011_2

1	1	0	1	1	*(binary)*

Add positional weights for all 15 → $2^4 + 2^3 + 0 + 2^1 + 2^0 = 16 + 8 + 2 + 1$

$= 27_{10}$ *(decimal)*

Let's try another example with a greater number of bits *i.e.*, 10110101_2

1	0	1	1	0	1	0	1

Adding positional weights of all 1's → $2^7 + 0 + 2^5 + 2^4 + 0 + 2^2 + 0 + 2^0 = 181_{10}$

Note that the procedure is to find the weights (*i.e.*, powers of 2) for each bit position that contains a 1, and then to add them up. Also note that the *MSB has a weight of* 2^7 *even though it is the eighth bit; this is because the LSB is the first bit and has a weight of* 2^0.

The above method will always provide the correct decimal representation of a binary number. There is a second method, called the *dibble-dobble* method, that will also provide the solution. To use this method, start with the left-hand bit. Multiply this value by 2, and add the next bit to the right. Multiply the result value by 2, and add the next bit to the right. Stop when the binary point is reached.

To illustrate,

11011_2 (binary)

Copy down the leftmost bit :	1	
Multiply by 2, add next bit	$(2 \times 1) + 1 = 3$	
Multiply by 2, add next bit	$(2 \times 3) + 0 = 6$	
Multiply by 2, add next bit	$(2 \times 6) + 1 = 13$	
Multiply by 2, add next bit	$(2 \times 13) + 1 = 27$	$\therefore 11011_2 = 27_{10}$

Let us try another example, 110100_2

The leftmost bit :	1	
Multiply by 2, add next bit	$(2 \times 1) + 1 = 3$	
Multiply by 2, add next bit	$(2 \times 3) + 0 = 6$	
Multiply by 2, add next bit	$(2 \times 6) + 1 = 13$	
Multiply by 2, add next bit	$(2 \times 13) + 0 = 26$	
Multiply by 2, add next bit	$(2 \times 26) + 0 = 52$	$\therefore 110100_2 = 52_{10}$

13.3.3 Decimal-to-Octal Conversion

A decimal integer can be converted to octal by using the same repeated-division method that we used in the decimal-to-binary conversion, but with a division factor of 8 instead of 2. An example is shown below :

		Remainders
8	266	
8	33	2
8	4	1
	0	4 ↑

Reading up, $266_{10} = 412_8$

Note that the first remainder becomes the least significant digit (LSD) of the octal number, and the last remainder becomes the most significant digit (MSD).

13.3.4 Octal-to-Decimal Conversion

An octal number, then, can be easily converted to its decimal equivalent by multiplying each octal digit by its positional weight. For example,

$$372_8 = 3\times(8^2)+7\times(8^1)+2\times(8^0)$$
$$= 3\times 64 + 7\times 8 + 2\times 1 = 250_{10}$$

Another example :

$$24.6_8 = 2\times(8^1)+4\times(8^0)+6\times(8^{-1}) = 20.75_{10}$$

13.3.5 Octal-to-Binary Conversion

The primary advantage of the octal number system is the ease with which conversion can be made between binary and octal numbers. The conversion from octal to binary is performed by converting *each* octal digit to its 3-bit binary equivalent. The eight possible digits are converted as indicated in table 13.3.

Table 13.3 *Binary Equivalents of Octal Digits*

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

Using these conversions, any octal number is converted to binary by individually converting each digit. For example, we can convert 472_8 to binary using 3 bits for each octal digit as follows :

4	7	2
↓	↓	↓
100	111	010

Hence, octal 472 is equivalent to binary 100111010. As another example, consider converting 5431 to binary :

5	4	3	1
↓	↓	↓	↓
101	100	011	001

Thus, $5431_8 = 101100011001_2$.

The same process applies equally on fractions. For example,

$$3.1_8 = \begin{matrix} 3 & \bullet & 1 \\ \downarrow & & \downarrow \\ 011 & \bullet & 001 \end{matrix}$$

$$3.1_8 = 011.001_2$$

NOTE

The octal to binary conversion takes place using 3-bit substitution for each octal character.

13.3.6 Binary-to-Octal Conversion

Converting from binary integers to octal integers is simply the reverse of the foregoing process. The bits of the binary integer are grouped into groups of *three* bits starting at the LSB. Then each group is converted to its octal equivalent (table 13.3). To illustrate, consider the conversion of 100111010_2 to octal.

$$\underbrace{1\ 0\ 0}_{\downarrow\ 4}\quad \underbrace{1\ 1\ 1}_{\downarrow\ 7}\quad \underbrace{0\ 1\ 0}_{\downarrow\ 2_8}$$

Sometimes the binary number will not have even groups of 3 bits. For those cases, we can add one or two 0s to the left of MSB of the binary number to fill out the last group. This is illustrated below for the binary number 11010110.

0 added here to make it group of 3 bits

$$\underbrace{0\ 1\ 1}_{\downarrow\ 3}\quad \underbrace{0\ 1\ 0}_{\downarrow\ 2}\quad \underbrace{1\ 1\ 0}_{\downarrow\ 6_8}$$

Note that a 0 was placed to the left of the MSB in order to produce complete groups of 3.

The same process applies on fractions. But after the binary point, zeros are added to the right. For example

$$10110.0101_2 = \underline{0\ 1\ 0}\ \ \underline{1\ 1\ 0} \bullet \underline{0\ 1\ 0}\ \ \underline{1\ 0\ 0}$$

$$2 \qquad 6 \qquad\qquad 2 \qquad 4$$

2 zeros added here to make it a group of 3 bits

$$10110.0101_2 = 26.24_8$$

Note that, after the binary point, the groups of 3 bits are made starting from left-to-right. That is why, we added two zeros to make a group of three bits as the last group had only 1.

13.3.7 Decimal-to Hex Conversion

Recall that we did decimal-to-binary conversion using repeated division by 2, and decimal-to-octal using repeated division by 8. Likewise, decimal-to-hex conversion can be done using repeated division by 16. For example,

To convert 423_{10} to hex,

		Remainders
16	423	
16	26	7
16	1	A
	0	1

$(10_{10} = A_{16})$

Reading up, $423_{10} = 1A7_{16}$

Similarly, to convert 214_{10} to hex,

		Remainders
16	214	
16	13	6
	0	D

$(13_{10} = D_{16})$

Reading up, $214_{10} = D6_{16}$

Note that any remainders that are greater than 9 are represented by letters A through F.