

Binary Representation of Numbers

Binary Representation of Numbers (1/2)

► Any integer greater than 1 can serve as a base for a number system. In computer science, base 2 notation, or binary notation, is of special importance because the signals used in modern electronics are always in one of only two states. (The Latin root bi means “two.”)



Example 2.5.1 – *Binary Notation for Integers from 1 to 9*

- ▶ Derive the binary notation for the integers from 1 to 9.

$$1_{10} = 1 \cdot 2^0 = 1_2$$

$$2_{10} = 1 \cdot 2^1 + 0 \cdot 2^0 = 10_2$$

$$3_{10} = 1 \cdot 2^1 + 1 \cdot 2^0 = 11_2$$

$$4_{10} = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 100_2$$

$$5_{10} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 101_2$$

$$6_{10} = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 110_2$$

$$7_{10} = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 111_2$$

$$8_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 1000_2$$

$$9_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1001_2$$

Example 2.5.1 – Solution

Binary Representation of Numbers (2/2)

A list of powers of 2 is useful for doing binary-to-decimal and decimal-to-binary conversions.

Power of 2	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Form	1024	512	256	128	64	32	16	8	4	2	1

Powers of 2

Table 2.5.1

Example 2.5.2 – Converting a Binary to a Decimal Number

Represent 110101_2 in decimal notation.

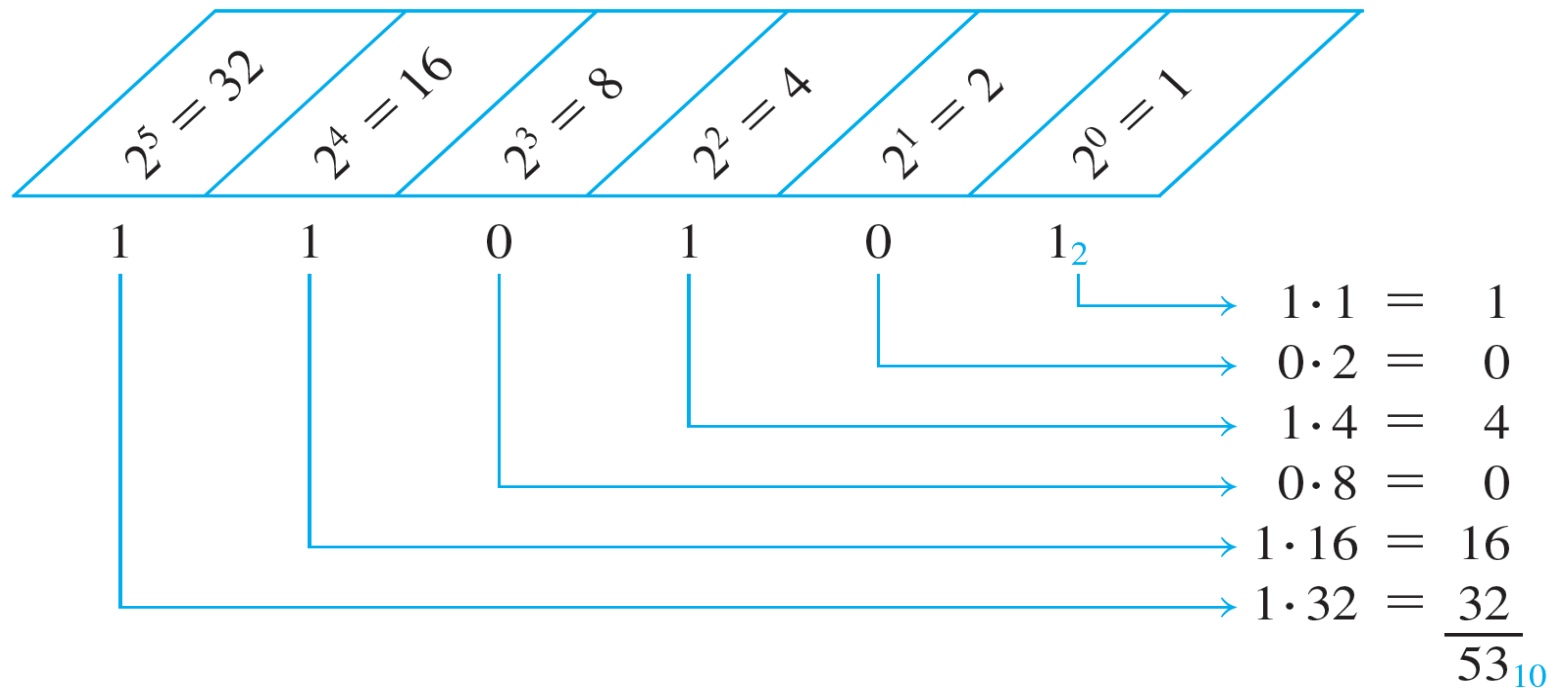
Example 2.5.2 – Solution

$$110101_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$= 32 + 16 + 4 + 1$$

$$= 53_{10}$$

► A



Example 2.5.3 – *Converting a Decimal to a Binary Number*

- Represent 209 in binary notation.

Example 2.5.3 – Solution (1/3)

► Use Table 2.5.1 to write 209 as a sum of powers of 2, starting with the highest power of 2 that is less than 209 and continuing to lower powers.

Power of 2	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Form	1024	512	256	128	64	32	16	8	4	2	1

► Powers of 2

► Table 2.5.1

► Since 209 is between 128 and 256, the highest power of 2 that is less than 209 is 128. Hence

$$209_{10} = 128 + \text{a smaller number.}$$

Example 2.5.3 – Solution (2/3)_{continued}

► Now $209 - 128 = 81$, and 81 is between 64 and 128, so the highest power of 2 that is less than 81 is 64. Hence

$$209_{10} = 128 + 64 + \text{a smaller number.}$$

► Continuing in this way, you obtain

$$209_{10} = 128 + 64 + 16 + 1$$

$$= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.$$

Example 2.5.3 – Solution (3/3)_{continued}

► For each power of 2 that occurs in the sum, there is a 1 in the corresponding position of the binary number. For each power of 2 that is missing from the sum, there is a 0 in the corresponding position of the binary number. Thus

$$209_{10} = 11010001_2$$

Binary Addition and Subtraction

Binary Addition and Subtraction (1/1)

► The computational methods of binary arithmetic are analogous to those of decimal arithmetic. In binary arithmetic the number 2 (which equals 10 in binary notation) plays a role similar to that of the number 10 in decimal arithmetic.

Example 2.5.4 – *Addition in Binary Notation*

- ▶ Add the binary values 1101 and 111 using binary notation.

Example 2.5.4 – Solution (1/2)

► Because $2_{10} = 10_2$ and $1_{10} = 1_2$, ► the translation of $1_{10} + 1_{10} = 2_{10}$ ► to binary notation is

$$\begin{array}{r} 1_2 \\ + 1_2 \\ \hline 10_2 \end{array}$$

► It follows that adding two 1's together results in a carry of 1 when binary notation is used. Adding three 1's together also results in a carry of 1 since

$3_{10} = 11_2$ ► (“one one base two”).

$$\begin{array}{r} 1_2 \\ + 1_2 \\ + 1_2 \\ \hline 11_2 \end{array}$$

Example 2.5.4 – Solution (2/2)

Thus, the addition can be performed as follows:

$$\begin{array}{rcccccc} & & \overset{1}{1} & \overset{1}{1} & \overset{1}{0} & 1_2 & \leftarrow \text{carry row} \\ + & & & 1 & 1 & 1_2 & \\ \hline 1 & 0 & 1 & 0 & 0_2 & & \end{array}$$

Example 2.5.5 – Subtraction in Binary Notation

Subtract 1011_2 from 11000_2 using binary notation.

Example 2.5.5 – Solution (1/2)

- ▶ In decimal subtraction the fact that $10_{10} - 1_{10} = 9_{10}$ is used to borrow across several columns.
- ▶ For example, consider the following:

$$\begin{array}{r} 99 \\ 100_{10} \\ - 58_{10} \\ \hline 942_{10} \end{array}$$

← borrow row

Example 2.5.5 – Solution (2/2)_{continued}

- ▶ In binary subtraction it may also be necessary to borrow across more than one column. But when you borrow a 1_2
- ▶ from 10_2 , what remains is 1_2 .

$$\begin{array}{r} 10_2 \\ - 1_2 \\ \hline 1_2 \end{array}$$

- ▶ Thus, the subtraction can be performed as follows:

$$\begin{array}{rcccccc} & & 0 & 1 & 1 & & \\ & & 1 & 1 & 1 & & \leftarrow \text{borrow row} \\ 1 & 1 & 0 & 0 & 0 & 0_2 & \\ - & 1 & 0 & 1 & 1 & 1_2 & \\ \hline & 1 & 1 & 0 & 1 & 1_2 & \end{array}$$

Circuits for Computer Addition

Circuits for Computer Addition

(1/9)

- Consider the question of designing a circuit to produce the sum of two binary digits P and Q . Both P and Q can be either 0 or 1. And the following facts are known:

$$\begin{array}{l} 1_2 + 1_2 = 10_2, \\ 1_2 + 0_2 = 1_2 = 01_2, \\ 0_2 + 1_2 = 1_2 = 01_2, \\ 0_2 + 0_2 = 0_2 = 00_2. \end{array}$$

- It follows that the circuit must have two outputs—one for the left binary digit (this is called the **carry**) and one for the right binary digit (this is called the **sum**).

Circuits for Computer Addition (2/9)

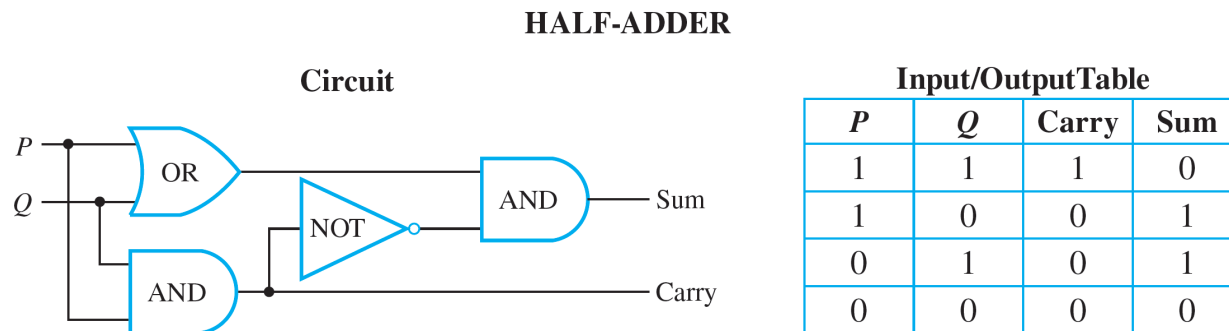
► The carry output is 1 if both P and Q are 1; it is 0 otherwise. Thus the carry can be produced using the AND-gate circuit that corresponds to the Boolean expression $P \wedge Q$.

► The sum output is 1 if either P or Q, but not both, is 1. The sum can, therefore, be produced using a circuit that corresponds to the Boolean expression for exclusive or:

$$(P \vee Q) \wedge \sim(P \wedge Q).$$

Circuits for Computer Addition (3/9)

Hence, a circuit to add two binary digits P and Q can be constructed as in Figure 2.5.1. This circuit is called a **half-adder**.



Circuit to Add $P + Q$, Where P and Q Are Binary Digits

Figure 2.5.1

Circuits for Computer Addition (4/9)

► Now consider the question of how to construct a circuit to add two binary integers, each with more than one digit. Because the addition of two binary digits may result in a carry to the next column to the left, it may be necessary to add three binary digits at certain points.

► In the following example, the sum in the right column is the sum of two binary digits, and, because of the carry, the sum in the left column is the sum of three binary digits.

$$\begin{array}{r} \\ \\ + \\ \hline 1 \end{array} \quad \begin{array}{l} \leftarrow \text{carry row} \\ 1_2 \\ 1_2 \end{array}$$

Circuits for Computer Addition (5/9)

► Thus, in order to construct a circuit that will add multidigit binary numbers, it is necessary to incorporate a circuit that will compute the sum of three binary digits. Such a circuit is called a full-adder.

► Consider a general addition of three binary digits P , Q , and R that results in a carry (or left-most digit) C and a sum (or right-most digit) S .

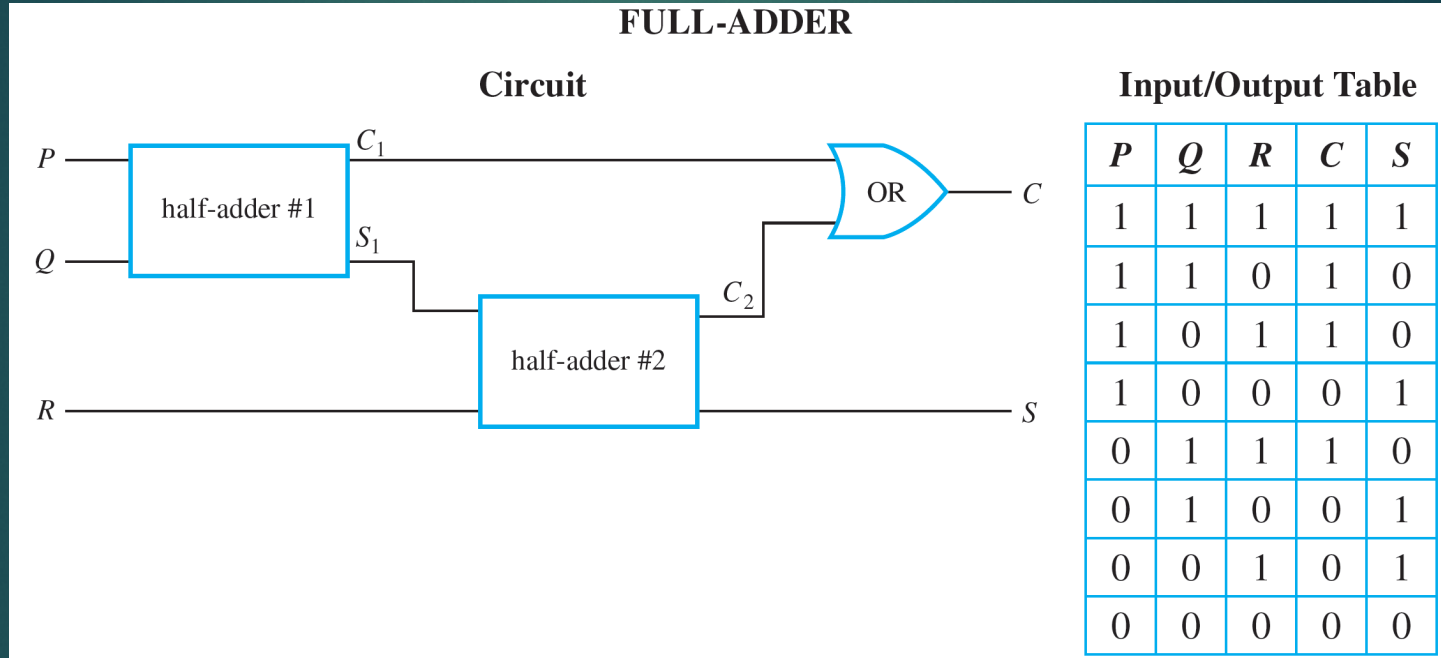
$$\begin{array}{r} P \\ + Q \\ + R \\ \hline CS \end{array}$$

Circuits for Computer Addition (6/9)

► The operation of the full-adder is based on the fact that addition is a binary operation: Only two numbers can be added at one time. Thus P is first added to Q and then the result is added to R. For instance, consider the following addition:

$$\begin{array}{r} + 1_2 \\ + 0_2 \\ + 1_2 \\ \hline 10_2 \end{array} \quad \left. \begin{array}{l} 1_2 + 0_2 = 01_2 \\ 1_2 + 1_2 = 10_2 \end{array} \right\}$$

Circuits for Computer Addition (7/9)



► Circuit to Add $P + Q + R$, Where P , Q , and R Are Binary Digits

► **Figure 2.5.2**

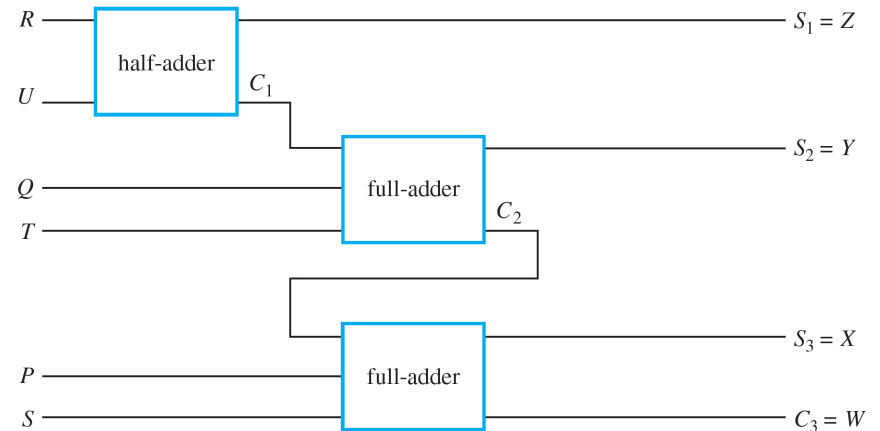
Circuits for Computer Addition (8/9)

► Two full-adders and one half-adder can be used together to build a circuit that will add two three-digit binary numbers PQR and STU to obtain the sum WXYZ.



Circuits for Computer Addition (9/9)

► This is illustrated in Figure 2.5.3. Such a circuit is called a parallel adder. Parallel adders can be constructed to add binary numbers of any finite length.



A Parallel Adder to Add PQR and STU to Obtain $WXYZ$

Figure 2.5.3

Two's Complements and the Computer Representation of Signed Integers

Two's Complements and the Computer Representation of Signed Integers (1/9)

► Typically a fixed number of bits is used to represent integers on a computer. One way to do this is to select a particular bit, normally the left-most, to indicate the sign of the integer, and to use the remaining bits for its absolute value in binary notation.

► The problem with this approach is that the procedures for adding the resulting numbers are somewhat complicated and the representation of 0 is not unique. A more common approach is to use “two's complements,” which makes it possible to add integers quite easily and results in a unique representation for 0.

Two's Complements and the Computer Representation of Signed Integers (2/9)

► Bit lengths of 64 and (sometimes) 32 are most often used in practice, but, for simplicity and because the principles are the same for all bit lengths, this discussion will focus on a bit length of 8.

Definition

The 8-bit two's complement for an integer a between -128 and 127 is the 8-bit binary representation for
$$\begin{cases} a & \text{if } a \geq 0 \\ 2^8 - |a| & \text{if } a < 0. \end{cases}$$

Two's Complements and the Computer Representation of Signed Integers (3/9)

Thus the 8-bit representation for a nonnegative integer is the same as its 8-bit binary representation. As a concrete example for the negative integer -46 , observe that

$$(2^8 - |-46|)_{10} = (256 - 46)_{10} = 210_{10} = (128 + 64 + 16 + 2)_{10} = 11010010_2,$$

and so the 8-bit two's complement for -46 is 11010010 .

Two's Complements and the Computer Representation of Signed Integers (4/9)

► For negative integers, however, there is a more convenient way to compute two's complements, which involves less arithmetic than applying the definition directly.

The 8-Bit Two's Complement for a Negative Integer

The 8-bit two's complement for a negative integer a that is at least -128 can be obtained as follows:

- Write the 8-bit binary representation for $|a|$.
- Switch all the 1's to 0's and all the 0's to 1's. (This is called flipping, or complementing, the bits.)
- Add 1 in binary notation.

Example 2.5.6 – *Finding a Two's Complement*

- ▶ Use the method described above to find the 8-bit two's complement for -46 .

Example 2.5.6 – Solution

► Write the 8-bit binary representation for $|-46| (=46)$, all the 1's to 0's and all the 0's to 1's, and then add 1.

$$|-46|_{10} = 46_{10} = (32 + 8 + 4 + 2)_{10}$$

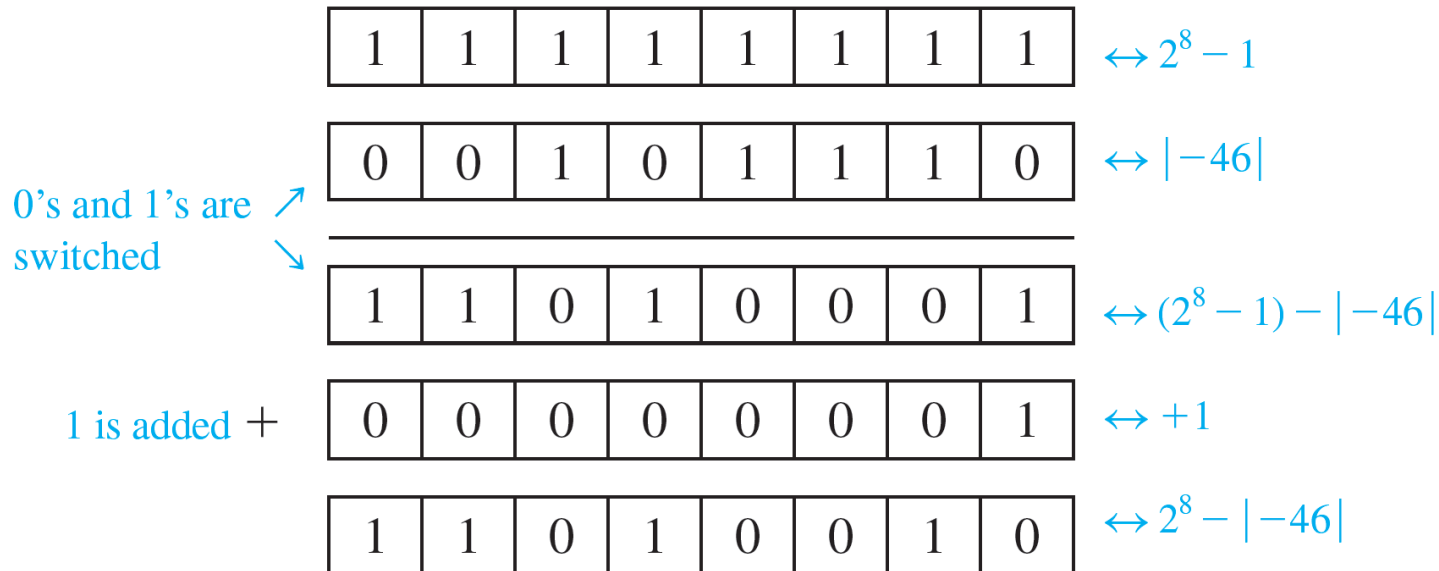
$$= 00101110_2 \xrightarrow{\text{flip the bits}} 11010001 \xrightarrow{\text{add 1}} 11010010.$$

► Note that this is the same result as was obtained directly from the definition.

Two's Complements and the Computer Representation of Signed Integers (5/9)

► The fact that the method for finding 8-bit two's complements works in general depends on the following facts:

1. The binary representation of $2^8 - 1$ is 11111111_2 .
2. Subtracting an 8-bit binary number a from 11111111_2 switches all the 1's to 0's and all the 0's to 1's.
3. $2^8 - |a| = [(2^8 - 1) - |a|] + 1$ for any number a .



Two's Complements and the Computer Representation of Signed Integers (6/9)

HERE IS HOW THE FACTS ARE USED WHEN $A = -46$:

Two's Complements and the Computer Representation of Signed Integers (7/9)

Because 127 is the largest integer represented in the 8-bit two's complement system and because $127_{10} = 01111111_2$,

all the 8-bit two's complements for nonnegative integers have a leading bit of 0.


Moreover, because the bits are switched, the leading bit for all the negative integers is 1.

Two's Complements and the Computer Representation of Signed Integers (8/9)

Table 2.5.2 illustrates the 8-bit two's complement representations for the integers from -128 through 127 .

Integer	8-Bit Two's Complement	Decimal Form of Two's Complement for Negative Integers
127	01111111	
126	01111110	
\vdots	\vdots	
2	00000010	
1	00000001	
0	00000000	
-1	11111111	$2^8 - 1$
-2	11111110	$2^8 - 2$
-3	11111101	$2^8 - 3$
\vdots	\vdots	\vdots
-127	10000001	$2^8 - 127$
-128	10000000	$2^8 - 128$

Table 2.5.2



To find the decimal representation of the negative integer with a given 8-bit two's complement:

- Apply the two's complement procedure to the given two's complement.
- Write the decimal equivalent of the result.

Two's Complements and the Computer Representation of Signed Integers (9/9)

Example 2.5.7 – Finding a Number with a Given Two's Complement

WHAT IS THE DECIMAL REPRESENTATION FOR THE INTEGER
WITH TWO'S COMPLEMENT 10101001?

Example 2.5.7 – Solution

Since the left-most digit is 1, the integer is negative. Applying the two's complement procedure gives the following result:

$$\begin{aligned} 1010\ 1001 &\xrightarrow{\text{flip the bits}} 0101\ 0110 \xrightarrow{\text{add 1}} 0101\ 0111_2 \\ &= (64 + 16 + 4 + 2 + 1)_{10} = 87_{10} = |-87|_{10}. \end{aligned}$$

So the answer is -87 . You can check its correctness by deriving the two's complement of -87 directly from the definition:

$$(2^8 - |-87|)_{10} = (256 - 87)_{10} = 169_{10} = (128 + 32 + 8 + 1)_{10} = 10101001_2.$$



Addition and Subtraction with Integers in Two's Complement Form

Addition and Subtraction with Integers in Two's Complement Form (1/4)

- ▶ The main advantage of a two's complement representation for integers is that the same computer circuits used to add nonnegative integers in binary notation can be used for both additions and subtractions of integers in a two's complement system of numeration.
- ▶ First note that because of the algebraic identity
- ▶
$$a - b = a + (-b) \text{ for all real numbers,}$$
- ▶ any subtraction problem can be changed into an addition one.

Addition and Subtraction with Integers in Two's Complement Form (2/4)

- ▶ For example, suppose you want to compute $78 - 46$. This equals $78 + (-46)$, which should give an answer of 32.
- ▶ To see what happens when you add the numbers in their two's complement forms, observe that the 8-bit two's complement for 78 is the same as the ordinary binary representation for 78, which is 01001110 because $78 = 64 + 8 + 4 + 2$, and, as previously shown, the 8-bit two's complement for -46 is 11010010.

Addition and Subtraction with Integers in Two's Complement Form (3/4)

Adding the numbers using binary addition gives the following:

	0	1	0	0	1	1	1	0	$\leftrightarrow 78$
+	1	1	0	1	0	0	1	0	$\leftrightarrow -46$
<hr/>									
1	0	0	1	0	0	0	0	0	$\leftrightarrow 32?$

The result has a carry bit of 1 in the ninth, or 2^8 th, position, you discard it, you obtain 00100000, which is the correct answer in 8-bit two's complement form because, since

$$32 = 2^8,$$

$$32_{10} = 00100000_2.$$

Addition and Subtraction with Integers in Two's Complement Form (4/4)

General Procedure for Using 8-Bit Two's Complements to Add Two Integers

To add two integers in the range -128 through 127 whose sum is also in the range -128 through 127 :

- Convert both integers to their 8-bit two's complement representations.
- Add the resulting integers using ordinary binary addition, discarding any carry bit of 1 that may occur in the 2^8 th position.
- Convert the result back to decimal form.

Hexadecimal Notation

Hexadecimal Notation (1/5)

- ▶ **Hexadecimal notation** is even more compact than decimal notation, and it is much easier to convert back and forth between hexadecimal and binary notation than it is between binary and decimal notation.
- ▶ The word *hexadecimal* comes from the Greek root *hex-*, meaning “six,” and the Latin root *deci-*, meaning “ten.”
- ▶ Hence *hexadecimal* refers to “sixteen,” and hexadecimal notation is also called **base 16 notation**.

Hexadecimal Notation (2/5)

Hexadecimal notation is based on the fact that any integer can be uniquely expressed as a sum of numbers of the form

$$d \cdot 16^n,$$

where each n is a nonnegative integer and each d is one of the integers from 0 to 15.

Hexadecimal Notation (3/5)

► The 16 hexadecimal digits are shown in Table 2.5.3, together with their decimal equivalents and, for future reference, their 4-bit binary equivalents.

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

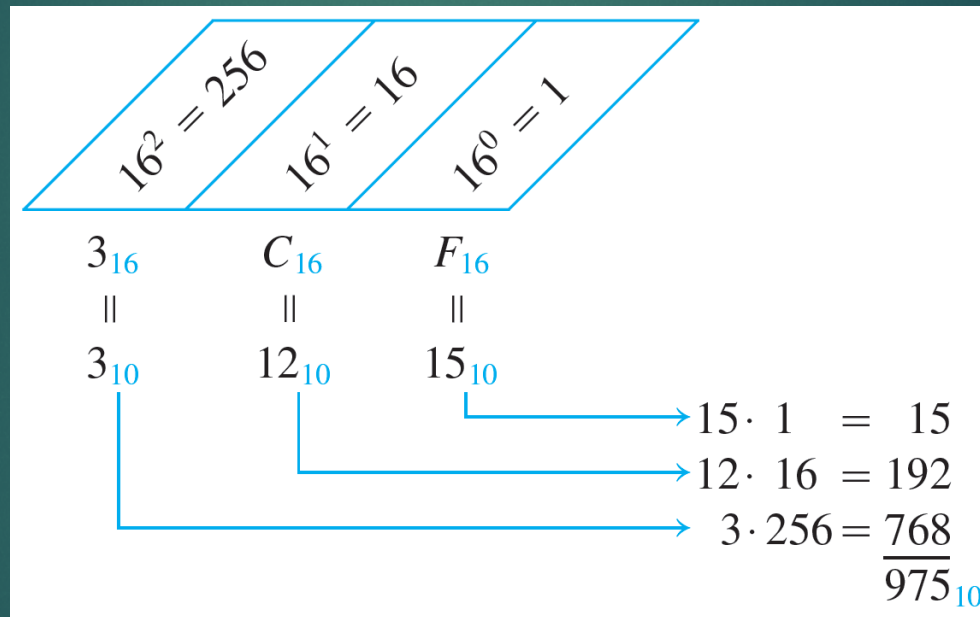
► Table 2.5.3

Example 2.5.8 – Converting from Hexadecimal to Decimal Notation

Convert $3CF_{16}$ to decimal notation.

Example 2.5.8 – Solution

► A schema similar to the one introduced in Example 2.5.2 can be used here.



So $3CF_{16} = 975_{10}$.



To convert an integer from hexadecimal to binary notation:

- Write each hexadecimal digit of the integer in 4-bit binary notation.
- Juxtapose the results.

Hexadecimal Notation (4/5)

Example 2.5.9 – Converting from Hexadecimal to Binary Notation

Convert **B09F**₁₆ to binary notation.

Example 2.5.9 – Solution

$B_{16} = 11_{10} = 1011_2$, $0_{16} = 0_{10} = 0000_2$, $9_{16} = 9_{10} = 1001_2$, and
 $F_{16} = 15_{10} = 1111_2$. Consequently,

B	0	9	F
↕	↕	↕	↕
1011	0000	1001	1111

and the answer is 1011000010011111_2 .

Hexadecimal Notation (5/5)

► To convert integers written in binary notation into hexadecimal notation, reverse the steps of the previous procedure. Note that the commonly used computer representation for integers uses 32 bits. When these numbers are written in hexadecimal notation only eight characters are needed.

To convert an integer from binary to hexadecimal notation:

- Group the digits of the binary number into sets of four, starting from the right and adding leading zeros as needed.
- Convert the binary numbers in each set of four into hexadecimal digits. Juxtapose those hexadecimal digits.

Example 2.5.10 – Converting from Binary to Hexadecimal Notation

Convert 100110110101001_2 to hexadecimal notation.

Example 2.5.10 – Solution (1/2)

► First group the binary digits in sets of four, working from right to left and adding leading 0's if necessary.

► 0100 1101 1010 1001.

► Convert each group of four binary digits into a hexadecimal digit.

0100	1101	1010	1001
↕	↕	↕	↕
4	D	A	9

Example 2.5.10 – Solution (2/2)

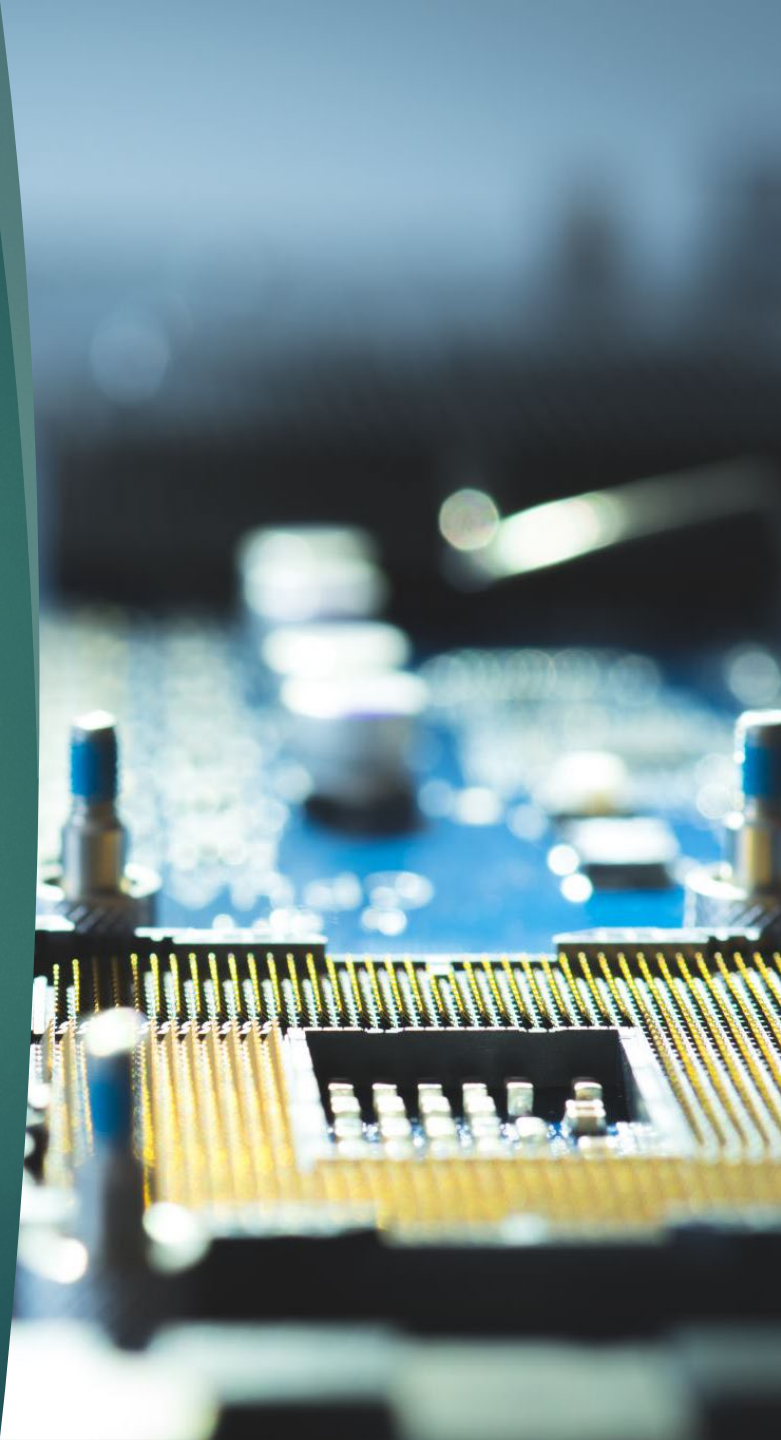
continued

Then juxtapose the hexadecimal digits. $4DA9_{16}$

Example 2.5.11 – Reading a Memory Dump

► The smallest addressable memory unit on most computers is one byte, or eight bits. In some debugging operations a dump is made of memory contents; that is, the contents of each memory location are displayed or printed out in order. To save space and make the output easier on the eye, the hexadecimal versions of the memory contents are given, rather than the binary versions. Suppose, for example, that a segment of the memory dump looks like

► A3 BB 59 2E.



Definition:

- "In little endian byte order, the least significant byte (LSB) is stored at the lowest memory address, and the significance of bytes increases with the memory address."

Key Points:

- **Least Significant Byte (LSB) First:** The smallest value byte is placed in the lowest address.
- **Memory Addressing:** As you move to higher memory addresses, you encounter more significant bytes.
- **Usage:** Common in x86 architectures and many microcontrollers.

Little Endian

Little Endian

- ▶ Therefore, for a 32-bit integer number: A3 BB 59 2E in memory
 - ▶ Is really the number 2E59BBA3
-
- ▶ A 16bit integer number: 02 B4
 - ▶ Is really B402