
Content

Unsigned Number

Signed Number


Other Numbers and Codes

Appendix

Decimal Numbers

$$\begin{aligned} 568.23 &= (5 \times 10^2) + (6 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2}) \\ &= (5 \times 100) + (6 \times 10) + (8 \times 1) + (2 \times 0.1) + (3 \times 0.01) \\ &= \mathbf{500} + \mathbf{60} + \mathbf{8} + \mathbf{0.2} + \mathbf{0.03} \end{aligned}$$

$$10^2 \ 10^1 \ 10^0 . 10^{-1} \ 10^{-2} \ 10^{-3} \dots$$

 **Decimal point**

- The value is the sum of digit multiplied by its weight
- The right-most bit is the LSB (least significant bit). The left-most bit is the MSB (most significant bit).

Counting in Binary

Decimal Number	Binary Number			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Binary Numbers

The Weighting Structure

$$2^{n-1} \dots 2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} \dots 2^{-n}$$

- With n bits one can count up to a number equal to $2^n - 1$
- 8-bit grouping is known as byte

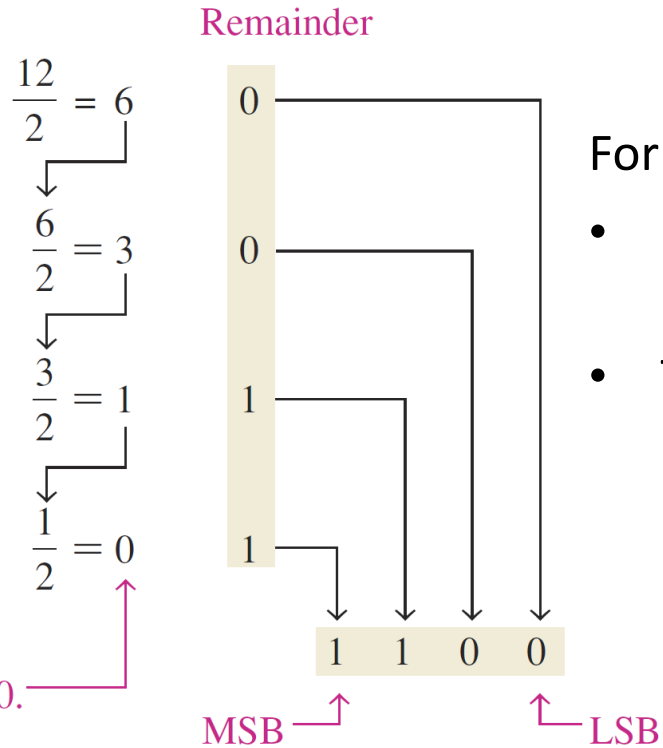
Binary-to-Decimal Conversion

$$(1011001)_2 = (?)_{10}$$

$$(101.110)_2 = (?)_{10}$$

Decimal-to-Binary Conversion: Integer

Division-by-2



For example, to convert 12,

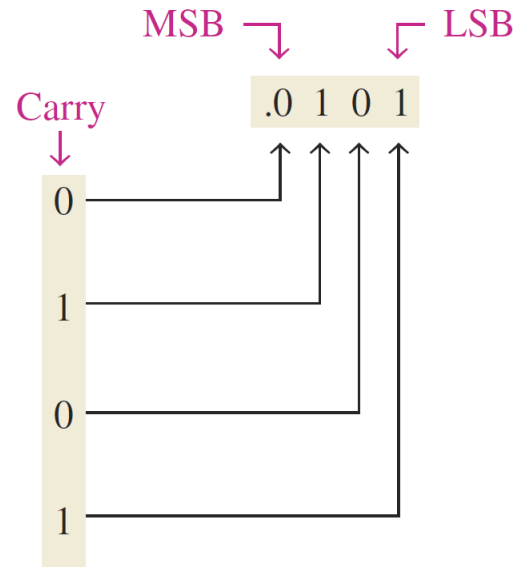
- Dividing 12 by 2, then divide each resulting quotient by 2 until a 0 quotient.
- The remainders generated by each division form the binary number. The first remainder is the LSB, and the last remainder is the MSB.

Decimal-to-Binary Conversion: Fractions

Multiplication-by-2

$$\begin{array}{l} 0.3125 \times 2 = 0.625 \\ \downarrow \\ 0.625 \times 2 = 1.25 \\ \downarrow \\ 0.25 \times 2 = 0.50 \\ \downarrow \\ 0.50 \times 2 = 1.00 \end{array}$$

Continue to the desired number of decimal places
or stop when the fractional part is all zeros.



$$(21.375)_{10} = (?)_2$$

For example, to convert 0.3125,

- Begin by multiplying 0.3125 by 2 and then multiplying each resulting fractional part of the product by 2 until the fractional product is zero.
- The carry digits produce the binary number. The first carry produced is the MSB, and the last carry is the LSB

Binary Addition & Subtraction

- Basic rules for addition

$0 + 0 = 0$	Sum of 0 with a carry of 0
$0 + 1 = 1$	Sum of 1 with a carry of 0
$1 + 0 = 1$	Sum of 1 with a carry of 0
$1 + 1 = 10$	Sum of 0 with a carry of 1

$$\begin{array}{r} 11 \\ + 11 \\ \hline \mathbf{110} \end{array} \quad \begin{array}{r} 3 \\ + 3 \\ \hline 6 \end{array}$$

- Basic rules for subtraction

$0 - 0 = 0$	
$1 - 1 = 0$	
$1 - 0 = 1$	
$10 - 1 = 1$	$0 - 1$ with a borrow of 1

$$\begin{array}{r} 101 \\ - 011 \\ \hline \mathbf{010} \end{array} \quad \begin{array}{r} 5 \\ - 3 \\ \hline 2 \end{array}$$

Binary Multiplication & Division

- Basic rules for multiplication

$$\begin{array}{l} 0 \times 0 = 0 \\ 0 \times 1 = 0 \\ 1 \times 0 = 0 \\ 1 \times 1 = 1 \end{array}$$

- Multiplication and division in binary follows the same procedure as that in decimal

$$\begin{array}{r} \text{Partial products } \left\{ \begin{array}{r} 11 \\ \times 11 \\ \hline 11 \\ +11 \\ \hline 1001 \end{array} \right. \end{array}$$

$$\begin{array}{r} 3 \\ \times 3 \\ \hline 9 \end{array}$$

$$\begin{array}{r} \text{Partial products } \left\{ \begin{array}{r} 111 \\ \times 101 \\ \hline 111 \\ 000 \\ +111 \\ \hline 100011 \end{array} \right. \end{array}$$

$$\begin{array}{r} 7 \\ \times 5 \\ \hline 35 \end{array}$$

$$\begin{array}{r} \mathbf{10} \quad 2 \\ 11 \overline{)110} \quad 3 \overline{)6} \\ \underline{11} \quad \underline{6} \\ 000 \quad 0 \end{array}$$

$$\begin{array}{r} \mathbf{11} \quad 3 \\ 10 \overline{)110} \quad 2 \overline{)6} \\ \underline{10} \quad \underline{6} \\ 10 \quad 0 \\ \underline{10} \\ 00 \end{array}$$

Content

Unsigned Number

Signed Number

Other Numbers and Codes

Appendix

Sign-Magnitude Form

- Signed integer can be represented by sign-magnitude, 1's complement, or 2's complement.
- The left-most bit is the sign bit and the remaining bits are the magnitude bits. A 0(1) sign bit indicates a positive(negative) number.

00011001
Sign bit ——— ↑ ↑ ——— Magnitude bits
+25

10011001
-25

1's Complement Form

- The 1's complement is obtained by changing all 1s to 0s and all 0s to 1s,
 - Positive numbers are same as the positive sign-magnitude numbers.
 - Negative numbers are the 1's complements of the corresponding positive numbers.
-
- +25 -> 00011001
 - -25 -> 11100110

2's Complement Form

- The 2's complement is obtained by adding 1 to the 1's complement
- Positive numbers are same as the positive sign-magnitude numbers.
- Negative numbers are the 2's complements of the corresponding positive numbers. e.g., -25 is expressed as the 2's complement of +25 (00011001) as 11100111

Write down the sign-magnitude, 1's and 2's complement of the following number

- +27
- -27
- -34
- -16

Think about it:

- Find the 2's complement of this number expressed in sign-magnitude form:
1001.1

2's Complement Form

- To go from the 2's complement form back to true binary, take the 1's complement of the 2's complement number and add 1 to the least significant bit.

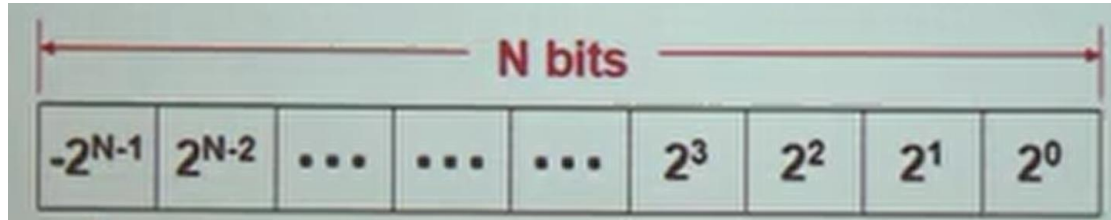
Think about it:


- Proof it is equivalent to “NOT(2's complement-1)”

Determine the decimal value of the number expressed in 2's complement

- 11111001

2's Complement to Decimal



111110001  $-2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 1$ How to get the value faster?

- With n bits, one can represent 2^n numbers. For 2's complement signed numbers, it ranges from -2^{n-1} to $(2^{n-1} - 1)$

Expand 2's Complement

Think about it:

Expand the following 2's complement to 7 bits

- 00101
- 101001

Addition

- We limit the discussion to 2's complement arithmetic since it enables performing $+-*/$ using only add and shift circuit unit.
- Add the two numbers and discard any final carry bit

- Both numbers positive

00000111	7
+ 00000100	+ 4
<u>00001011</u>	<u>11</u>

- Positive number with magnitude larger than negative number

00001111	15
+ 11111010	+ -6
<u>1 00001001</u>	<u>9</u>

Discard carry \longrightarrow 1

- Negative number with magnitude larger than positive number

00010000	16
+ 11101000	+ -24
<u>11111000</u>	<u>-8</u>

- Both numbers negative

11111011	-5
+ 11110111	+ -9
<u>1 11110010</u>	<u>-14</u>

Discard carry \longrightarrow 1

Overflow Condition

- When two numbers are added and the number of bits required to represent the sum exceeds the number of bits in the two numbers, an overflow results as indicated by an incorrect sign bit. An overflow can occur only when both numbers are positive or both numbers are negative. If the sign bit of the result is different than the sign bit of the numbers that are added, overflow is indicated.

	01111101	125
	+ 00111010	+ 58
	<hr/>	<hr/>
	10110111	183
Sign incorrect	↑	
Magnitude incorrect	↑	

Think about it:

- How to confidently perform the calculation like the last page?

Subtraction

- Subtracting +3 (subtrahend) from +8 (minuend) is equivalent to adding -3 to +8.
- The sign of a positive or negative binary number is changed by taking its 2's complement. (proof it)

	00001000	Minuend (+8)
	+ 11111101	2's complement of subtrahend (-3)
Discard carry →	1 00000101	Difference (+5)

- It's important to determine the number of bits in advance

+13	0	01101	+13	0	01101
+10	0	01010	-10	1	10110
+23	0	10111	+3	0	00011
-13	1	10011	-13	1	10011
+10	0	01010	-10	1	10110
-3	1	11101	-23	1	01001

+13	0	1101	+13	0	1101
+10	0	1010	-10	1	0110
+23	1	0111	+3	0	0011
-13	1	0011	-13	1	0011
+10	0	1010	-10	1	0110
-3	1	1101	-23	0	1001

Multiplication: the direct addition method

- Multiplication can be accomplished using addition
- Direct addition and partial products are two basic methods.
- When two binary numbers are multiplied, both numbers must be in true uncomplemented form
- In the direct addition method, one adds the multiplicand a number of times equal to the multiplier, e.g., $8 + 8 + 8 = 24$.
- Multiply the signed binary numbers 01001101 and 00000100

8	Multiplicand
$\times 3$	Multiplier
<hr/>	
24	Product

01001101	1st time
+ 01001101	2nd time
<hr/>	
10011010	Partial sum
+ 01001101	3rd time
<hr/>	
11100111	Partial sum
+ 01001101	4th time
<hr/>	
100110100	Product

Multiplication: the partial products method

- The partial products method is perhaps the more common one
1. If the signs are the same (different), the product is positive (negative).
 2. Change any negative number to true uncomplemented form. Generate the partial products and add all of them.
 3. If the sign bit that was determined in step 1 is negative, take the 2's complement of the product. If positive, leave the product in true form. Attach the sign bit to the product.

Multiplication: the partial products method

Multiply 01010011 and 11000101

- The sign bit will be negative
- Change the multiplier to the true uncomplemented form $11000101 \longrightarrow 00111011$

1010011	Multiplicand
$\times 0111011$	Multiplier
1010011	1st partial product
+ 1010011	2nd partial product
11111001	Sum of 1st and 2nd
+ 0000000	3rd partial product
011111001	Sum
+ 1010011	4th partial product
1110010001	Sum
+ 1010011	5th partial product
100011000001	Sum
+ 1010011	6th partial product
1001100100001	Sum
+ 0000000	7th partial product
1001100100001	Final product

- Finally, take the 2's complement of the product

Attach the sign bit \longrightarrow

$1001100100001 \longrightarrow 0110011011111$

1 0110011011111

Division

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

- The division operation in computers is accomplished using subtraction
 1. If the signs are the same (different), the quotient is positive (negative).
 2. Subtract the divisor from the dividend using 2's complement addition to get the first partial remainder and add 1 to the quotient. If this partial remainder is positive, repeat this process until a zero or negative result.
 3. Count the number of times that the divisor is subtracted, which gives the quotient

Division

Divide 01100100 by 00011001

01100100	Dividend
+ 11100111	2's complement of divisor
<hr/> 01001011	Positive 1st partial remainder

- Add 1 to quotient: $000 + 001 = 001$

01001011	1st partial remainder
+ 11100111	2's complement of divisor
<hr/> 00110010	Positive 2nd partial remainder

- Add 1 to quotient: $001 + 001 = 010$.

00110010	2nd partial remainder
+ 11100111	2's complement of divisor
<hr/> 00011001	Positive 3rd partial remainder

- Add 1 to quotient: $010 + 001 = 011$.

00011001	3rd partial remainder
+ 11100111	2's complement of divisor
<hr/> 00000000	Zero remainder

- Add 1 to quotient: $011 + 001 = 100$ (final quotient).



Content

Unsigned Number

Signed Number

Other Numbers and Codes

Appendix

Hexadecimal Numbers

- The hexadecimal number system has a base of sixteen. One can use “h” or subscript 16 to designate hexadecimal numbers.

Counting in Hexadecimal

..., E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2A, 2B, 2C, 2D, 2E, 2F, 30, 31, ...

Binary-to-Hexadecimal Conversion

1100101001010111
↓ ↓ ↓ ↓
C A 5 7 = CA57₁₆

Hexadecimal-to-Binary Conversion

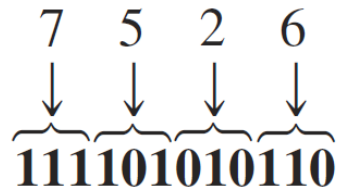
1 0 A 4
↓ ↓ ↓ ↓
1000010100100

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

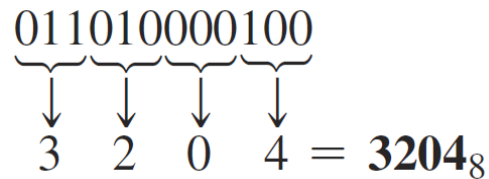
Octal Numbers

- The octal number system is composed of eight digits, 0, 1, 2, 3, 4, 5, 6, 7
- Counting octal number: 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, ...
- an “o”, “Q” or subscript 8 are used to indicate an octal number.

Octal-to-Binary Conversion



Binary-to-Octal Conversion



Binary Coded Decimal (BCD)

- BCD means that each decimal digit, 0 through 9, is represented by a binary code of four bits.
- The 8421 code is a type of BCD code. The designation 8421 indicates the binary weights of the four bits (2^3 , 2^2 , 2^1 , 2^0).

Decimal/BCD conversion.

Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

- 1010, 1011, 1100, 1101, 1110, and 1111 are invalid

2 4 6 9
↓ ↓ ↓ ↓
0010010001101001

1001010001110000
↓ ↓ ↓ ↓
9 4 7 0

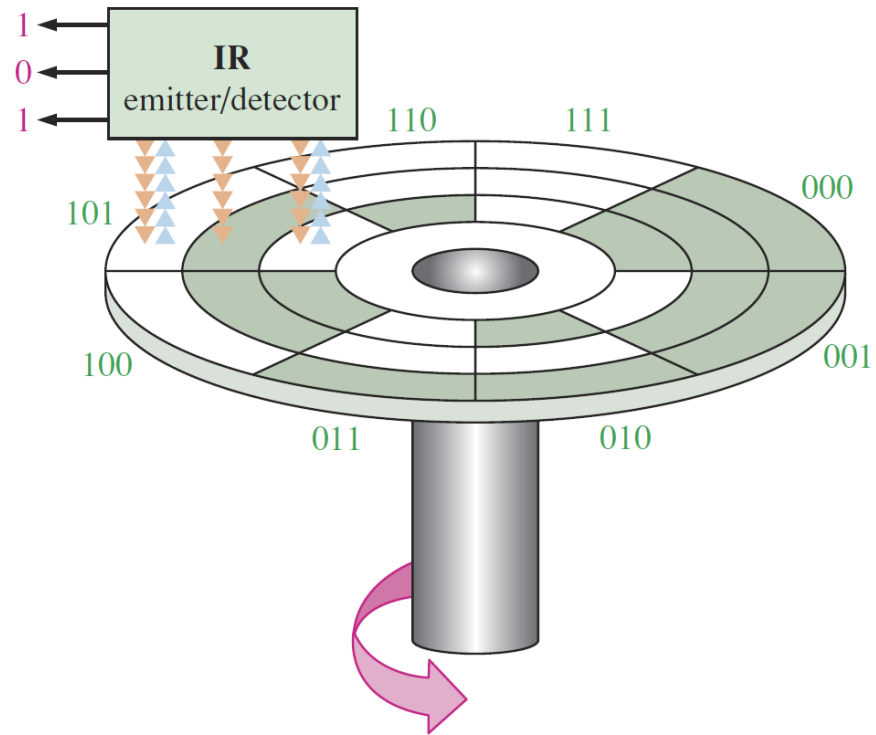
The Gray Code

- The Gray code is unweighted.
- An important feature is that it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications, such as shaft position encoders.

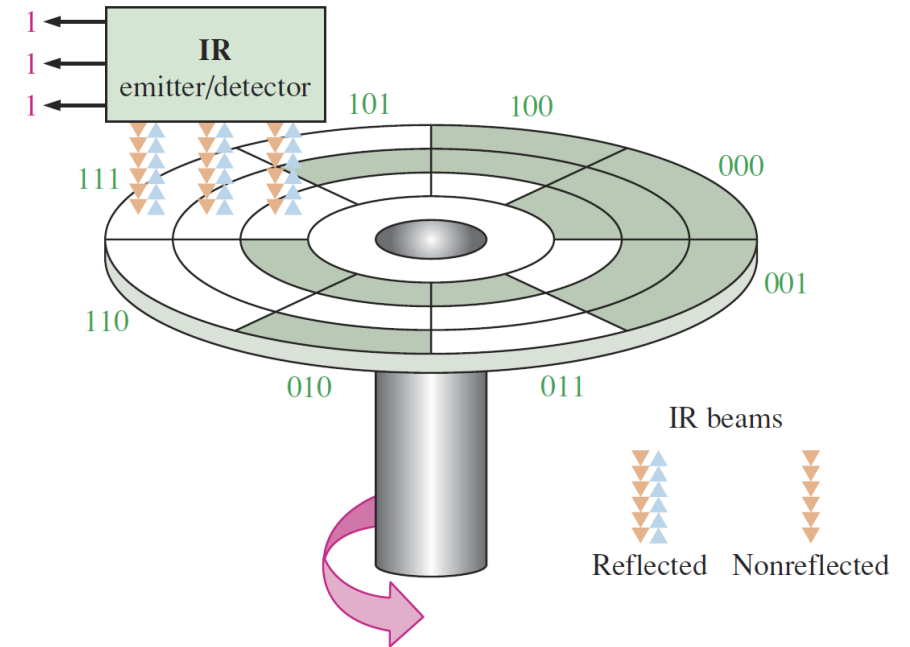
Four-bit Gray code.

Decimal	Binary	Gray Code	Decimal	Binary	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Gray Code Application



Binary code



Gray code

Alphanumeric Codes & ASCII & unicode

- Alphanumeric codes represent numbers, alphabetic and symbols
- ASCII (American Standard Code for Information Interchange) is a universally accepted alphanumeric code
- ASCII has 128 characters and symbols represented by a 7-bit binary code
- Unicode provides the ability to encode all of the characters used for the written languages of the world

Error Codes – Parity Method for Error Detection

- Any group of bits contain either an even or an odd number of 1s. An even (odd) parity bit makes the total number of 1s even (odd).
- A parity bit provides for the detection of a single bit error but cannot check for two errors

The BCD code with parity bits.

Even Parity		Odd Parity	
<i>P</i>	BCD	<i>P</i>	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Content

Unsigned Number

Signed Number

Other Numbers and Codes


[Appendix](#)

Reading materials

- Chapter 2 of Floyd book
- Chapter 1 of 阎石 book

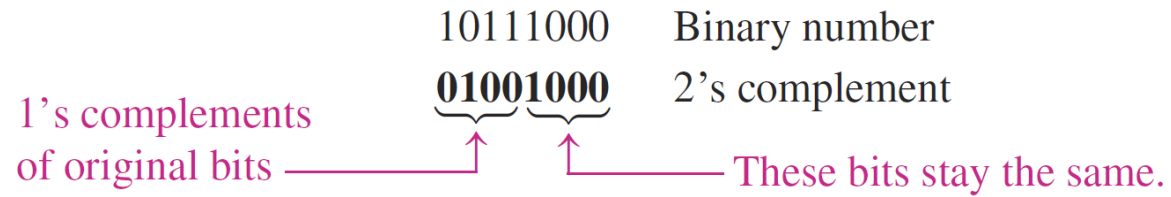
1's Complement to Decimal

- Decimal values of negative numbers are determined by assigning a negative value to the weight of the sign bit, summing all the weights where there are 1s, and adding 1 to the result.

11101000	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
	1	1	1	0	1	0	0	0		$-128+64+32+8+1=-23$

2's complement

- An alternative method to get 2's complement: (a) Start at the LSB and write the bits as they are up to and including the first 1. (b) Take the 1's complements of the remaining bits.

	10111000	Binary number
	01001000	2's complement
1's complements of original bits		These bits stay the same.

Floating-Point Numbers

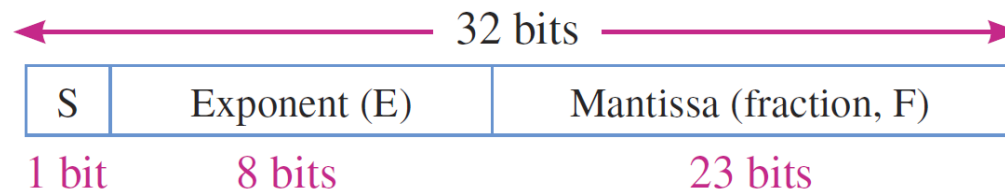
- A floating-point number consists of the mantissa, exponent plus a sign. The mantissa is between 0 and 1. The exponent represents the number of places that the decimal point is to be moved, e.g., the mantissa of 241,506,800 is .2415068 and the exponent is 9, written as

$$0.2415068 \times 10^9$$

Binary Floating-Point Numbers

- For binary floating-point numbers, the format is defined by ANSI/IEEE Standard 754-1985 in three forms: single-precision (32 bits), double-precision (64 bits), and extended-precision (80 bits). These all have the same basic formats except for the number of bits.

Single-Precision Floating-Point Binary Numbers



- The eight bits in the exponent represent a biased exponent, which is obtained by adding 127 to the actual exponent. The purpose of the bias is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$.

Binary Floating-Point Numbers

$$1011010010001 = 1.011010010001 \times 2^{12}$$

S	E	F
0	10001011	011010010001000000000000

- Assuming that this is a positive number, the sign bit is 0. The exponent, 12 is biased to $12 + 127 = 139$ (10001011). The fractional part (F) is .011010010001. Because there is always a 1 to the left of the binary point, it is not included in the mantissa.
- To evaluate a binary number in floating-point format

$$\text{Number} = (-1)^S (1 + F)(2^{E-127})$$

S	E	F
1	10010001	100011100010000000000000

$$\begin{aligned} \text{Number} &= (-1)^1 (1.10001110001)(2^{145-127}) \\ &= (-1)(1.10001110001)(2^{18}) = -1100011100010000000 \end{aligned}$$

- There are two exceptions: 0.0 is represented by all 0s, and infinity is represented by all 1s in the exponent and all 0s in the mantissa.

Hexadecimal Numbers

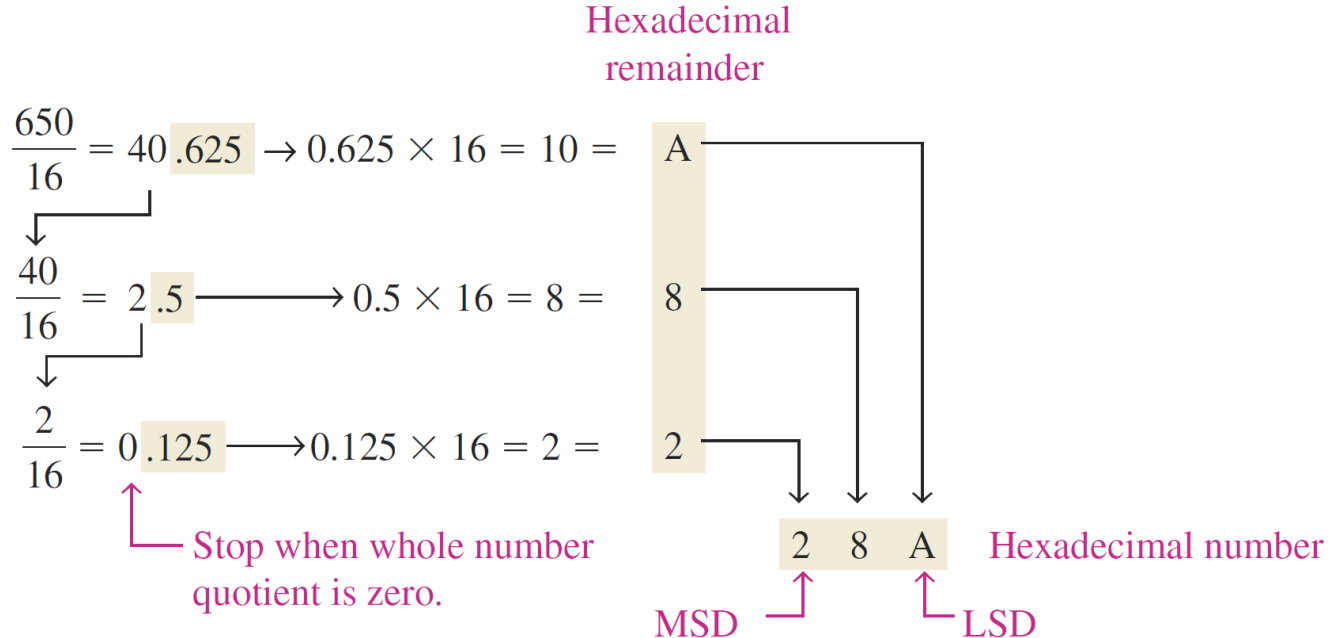
Hexadecimal-to-Decimal Conversion

$$\begin{array}{c} 1 \quad C \\ \downarrow \quad \downarrow \\ \underbrace{0001}_{1} \underbrace{1100}_{C} = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = \mathbf{28}_{10} \end{array}$$

$$\begin{aligned} \text{B2F8}_{16} &= (\text{B} \times 4096) + (2 \times 256) + (\text{F} \times 16) + (8 \times 1) \\ &= (11 \times 4096) + (2 \times 256) + (15 \times 16) + (8 \times 1) \\ &= 45,056 + 512 + 240 + 8 = \mathbf{45,816}_{10} \end{aligned}$$

Decimal-to-Hexadecimal Conversion

- Repeated division by 16 will produce the hexadecimal number



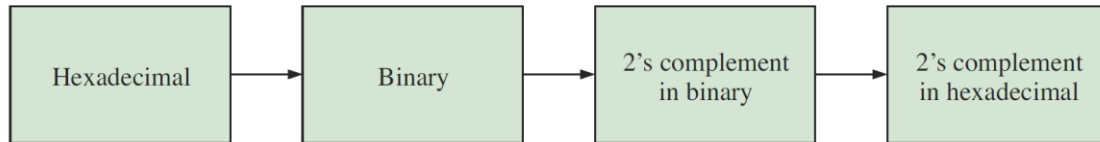
Hexadecimal Numbers

Hexadecimal Addition

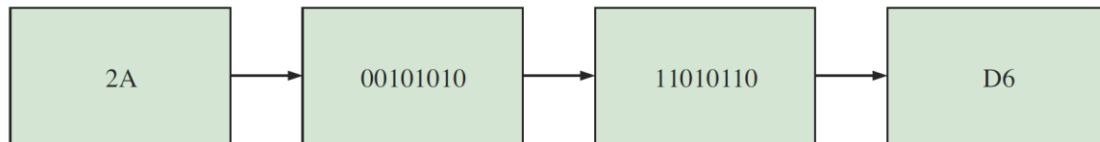
$$\begin{array}{r} 58_{16} \\ + 22_{16} \\ \hline 7A_{16} \end{array} \qquad \begin{array}{r} DF_{16} \\ + AC_{16} \\ \hline 18B_{16} \end{array}$$

Hexadecimal Subtraction

Method 1

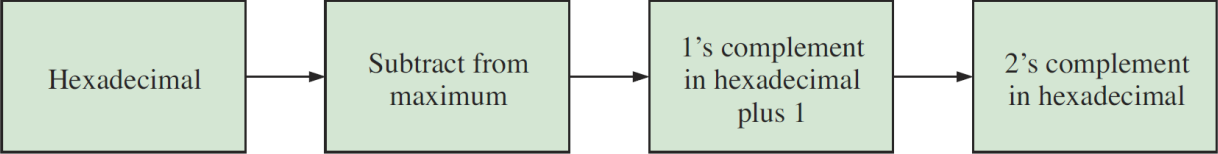


Example:

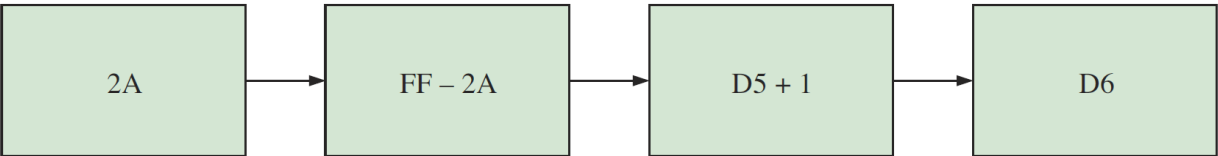


Hexadecimal Numbers

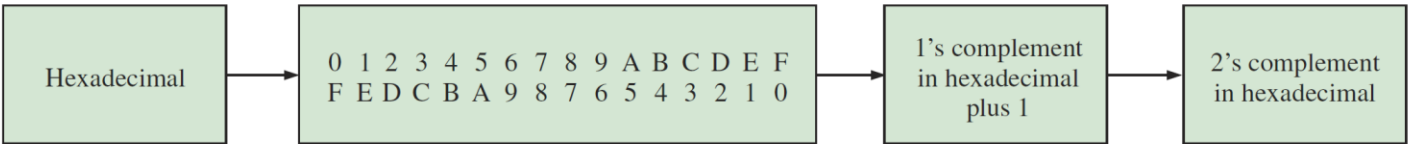
Method 2



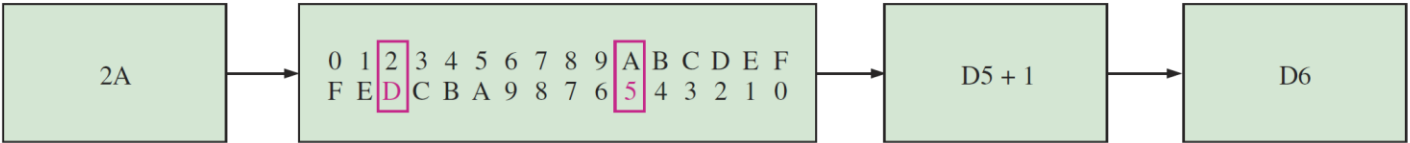
Example:



Method 2



Example:

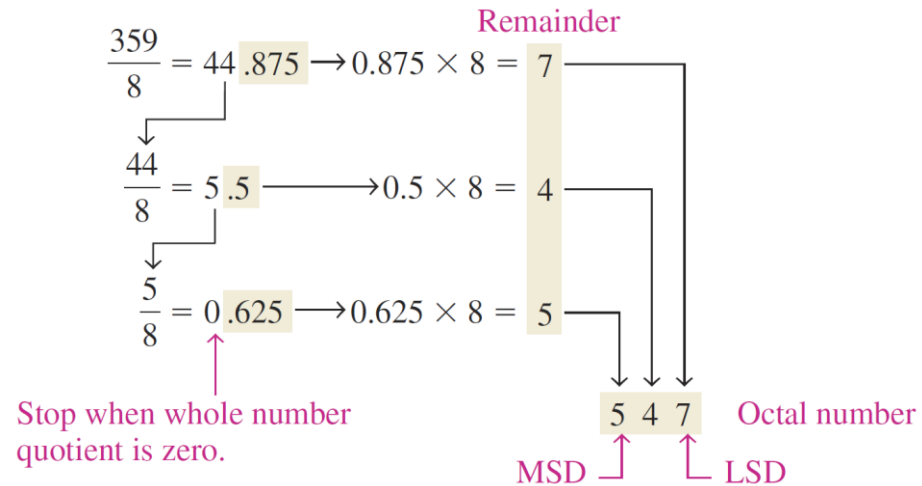


Octal Numbers

Octal-to-Decimal Conversion

$$\begin{aligned} 2374_8 &= (2 \times 8^3) + (3 \times 8^2) + (7 \times 8^1) + (4 \times 8^0) \\ &= (2 \times 512) + (3 \times 64) + (7 \times 8) + (4 \times 1) \\ &= 1024 + 192 + 56 + 4 = 1276_{10} \end{aligned}$$

Decimal-to-Octal Conversion



BCD Addition

- If a 4-bit sum is greater than 9, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

$$\begin{array}{r} 0001 \quad 0110 \\ + 0001 \quad 0101 \\ \hline 0010 \quad 1011 \end{array}$$

+ 0110

$$\begin{array}{r} \overline{0011} \quad \overline{0001} \\ \downarrow \quad \downarrow \\ 3 \quad 1 \end{array}$$

Right group is invalid (>9),
left group is valid.

Add 6 to invalid code. Add
carry, 0001, to next group.

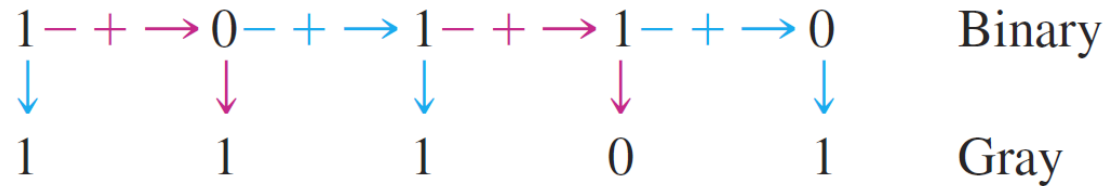
Valid BCD number

$$\begin{array}{r} 16 \\ + 15 \\ \hline 31 \end{array}$$

The Gray Code

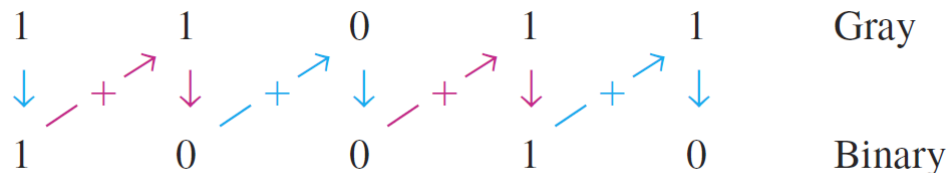
Binary-to-Gray Code Conversion

- The MSB remains the same.
- Going from left to right, add each adjacent pair of bits to get the next bit. Discard carries.



Gray-to-Binary Code Conversion

- The MSB remains the same.
- Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries



Error Codes – Cyclic Redundancy Check (CRC)

- CRC is a widely used code used for detecting one- and two-bit transmission errors.
- If it is properly designed, the CRC can also detect multiple errors for a number of bits in sequence (burst errors).
- In CRC, a certain number of check bits, called a checksum, are appended to the end of data bits. The transmitted data are tested by the receiver using the CRC.

Error Codes – Cyclic Redundancy Check (CRC)

- Select a fixed generator code; it can have fewer bits than the data bits.
- Append a number of 0s equal to the number of bits in the generator code to the data bits.
- Divide the data bits including the appended bits by the generator code bits using modulo-2.
- If the remainder is 0, the data and appended bits are sent as is.
- If the remainder is not 0, the appended bits are made equal to the remainder bits in order to get a 0 remainder.
- At the receiving end, the receiver divides the incoming appended data bit code by the generator code.
- If the remainder is 0, there is no error detected (it is possible in rare cases for multiple errors to cancel). If the remainder is not 0, an error has been detected in the transmission and a retransmission is requested by the receiver.

Error Codes – Cyclic Redundancy Check (CRC)

Data: 11010011

Generator code: 1010



- The appended data: $D' = 110100110000$
- Divide D' by the generator code
- Remainder = 0100
- Since the remainder is not 0, append the data with the remainder bits. Then divide by the generator code.
- Remainder = 0

$$\frac{D'}{G} = \frac{11010011\textcolor{blue}{0000}}{1010}$$

$$\begin{array}{r} 11010011\textcolor{blue}{0000} \\ \textcolor{violet}{1010} \downarrow \\ 1110 \\ \textcolor{violet}{1010} \downarrow \\ 1000 \\ \textcolor{violet}{1010} \downarrow \\ 1011 \\ \textcolor{violet}{1010} \downarrow \\ 1000 \\ \textcolor{violet}{1010} \downarrow \\ 100 \end{array}$$

$$\begin{array}{r} 11010011\textcolor{blue}{10100} \\ \textcolor{violet}{1010} \downarrow \\ 1110 \\ \textcolor{violet}{1010} \downarrow \\ 1000 \\ \textcolor{violet}{1010} \downarrow \\ 1011 \\ \textcolor{violet}{1010} \downarrow \\ 1010 \\ \textcolor{violet}{1010} \downarrow \\ 00 \\ 47 \end{array}$$

Hamming Code

- The Hamming code is used to detect and correct a single-bit error in a transmitted code. To accomplish this, four redundancy bits are introduced in a 7-bit group of data bits. These redundancy bits are interspersed at bit positions 2^n ($n = 0, 1, 2, 3$) within the original data bits. At the end of the transmission, the redundancy bits have to be removed from the data bits.