

---

## Digital Logic

---

• Number Systems	4
• Boolean Algebra & Logic Simplification	31
• Logic Gates	60
• Combinational Logic Analysis	75
• Functions of Combinational Logic	96
• Sequential Logic	125
• Design of sequential circuits	163
• Appendix 1	
➤ Properties of Logical operators	222
• Appendix 2	
➤ Half & full Subtractor	231
• Appendix 3	
➤ Implementation of HA & FA using NAND gates	233
• Appendix 4	
➤ Implementation of Boolean functions using NAND & NOR gates	235
• Appendix 5	
➤ Positive & Negative Logic system	237

# Number Systems

---

## DECIMAL NUMBERS

The position of each digit in a weighted number system is assigned a weight based on the **base** or **radix** of the system. The **radix** of decimal numbers is **ten**, because only ten symbols (0 through 9) are used to represent any number.

The column weights of decimal numbers are **powers of ten** that increase from right to left beginning with  $10^0=1$ :

$$\dots 10^5 \ 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0.$$

For **fractional** decimal numbers, the column weights are **negative powers** of ten that decrease from left to right:

$$10^2 \ 10^1 \ 10^0. \ 10^{-1} \ 10^{-2} \ 10^{-3} \ 10^{-4} \dots$$

Decimal numbers can be expressed as the **sum of the products** of each digit times the column value for that digit. Thus, the number **9240** can be expressed as:

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$9 \times 1,000 + 2 \times 100 + 4 \times 10 + 0 \times 1$$

**Example:** Express the number **480.52** as the sum of values of each digit.

**Solution:**  $480.52 = (4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})$

## BINARY NUMBERS

For digital systems, the binary number system is used. Binary has a **radix** of **two** and uses the digits **0** and **1** to represent quantities. The column weights of binary numbers are **powers of two** that increase from right to left beginning with  $2^0=1$ :

$$\dots 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0.$$

For **fractional** binary numbers, the column weights are **negative powers** of two that decrease from left to right:

$$2^2 \ 2^1 \ 2^0. \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \dots$$

## BINARY-TO-DECIMAL CONVERSIONS

The decimal equivalent of a binary number can be determined by adding the column values of all of the bits that are **1** and discarding all of the bits that are 0.

**Example:** Convert the binary number **100101.01** to decimal.

**Solution:** Start by writing the column weights; then add the weights that correspond to each **1** in the number.

$$\begin{array}{r}
 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \cdot 2^{-1} \ 2^{-2} \\
 32 \ 16 \ 8 \ 4 \ 2 \ 1 \cdot \frac{1}{2} \ \frac{1}{4} \\
 1 \ 0 \ 0 \ 1 \ 0 \ 1 \cdot 0 \ 1 \\
 32 \ +4 \ +1 \ +\frac{1}{4} \ = \ 37\frac{1}{4}
 \end{array}$$

### DECIMAL-TO-BINARY CONVERSIONS

You can convert a decimal whole number to binary by reversing the procedure. Write the decimal weight of each column and place **1**'s in the columns that sum to the decimal number.

**Example:** Convert the decimal number **49** to binary.

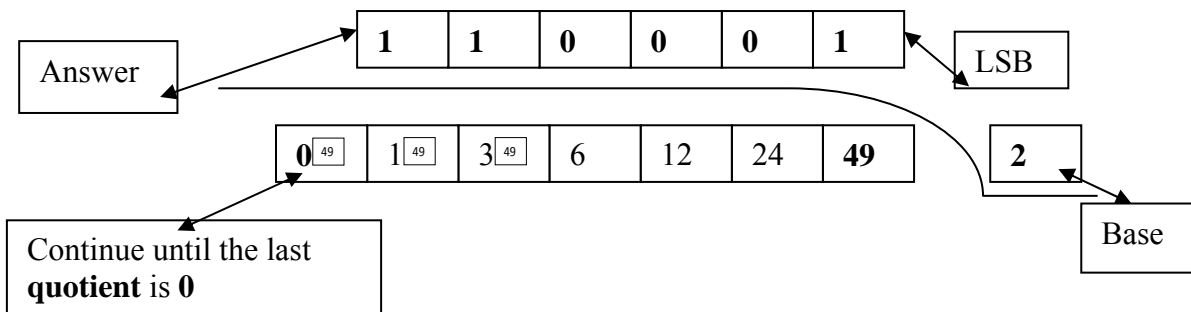
**Solution:** Write down column weights until the last number is **larger** than the one you want to convert.

$$\begin{array}{r}
 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1 \\
 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1
 \end{array}$$

Another way convert decimal to any other base by **repeatedly dividing by the base**. For binary, repeatedly divide by **2**:

**Example:** Convert the decimal number **49** to binary by repeatedly dividing by **2**.

**Solution:** Begin by dividing **49** by **2**, then divides each resulting **quotient** by **2** until there is a **0** quotient. The **remainder** generated by each of division form the **binary number** (Put quotients to the left and remainders on top).



### CONVERTING DECIMAL FRACTIONS TO BINARY

You can convert a decimal fraction to binary by repeatedly multiplying the fractional results of successive multiplications by **2**. The carries form the binary number.

**Example:** Convert the decimal fraction **0.188** to binary by repeatedly **multiplying** the fractional results by **2**.

**Solution:**

$0.188 \times 2 = 0.376$	<b>carry = 0</b>	<b>MSB</b>
$0.376 \times 2 = 0.752$	<b>carry = 0</b>	
$0.752 \times 2 = 1.504$	<b>carry = 1</b>	
$0.504 \times 2 = 1.008$	<b>carry = 1</b>	
$0.008 \times 2 = 0.016$	<b>carry = 0</b>	<b>LSB</b>
<b>Answer = .00110</b> (for five significant digits)		↓

## BINARY ARITHMETIC

### BINARY ADDITION

The **rules** for binary addition are:

$$\begin{array}{ll}
 0 + 0 = 0 & \text{Sum} = 0, \text{ carry} = 0 \\
 0 + 1 = 1 & \text{Sum} = 1, \text{ carry} = 0 \\
 1 + 0 = 1 & \text{Sum} = 1, \text{ carry} = 0 \\
 1 + 1 = 10 & \text{Sum} = 0, \text{ carry} = 1
 \end{array}$$

When an input (carry = 1) due to a previous result, the **rule** is:

$$1 + 1 + 1 = 11 \quad \text{Sum} = 1, \text{ carry} = 1$$

**Example:** Add the binary numbers **00111** and **10101** and show the equivalent decimal addition

**Solution:**

	111	
<b>00111</b>	<b>7</b>	
<u><b>10101</b></u>	<u><b>21</b></u>	
<b>11100</b>	<b>28</b>	

### BINARY SUBTRACTION

The **rules** for binary subtraction are:

$$\begin{array}{l}
 0 - 0 = 0 \\
 1 - 1 = 0 \\
 1 - 0 = 1 \\
 10 - 1 = 1 \text{ with a borrow of } 1
 \end{array}$$

**Example:** Subtract the binary number **00111** from **10101** and show the equivalent decimal subtraction.

**Solution:**

	111	
<b>10101</b>	<b>21</b>	
<u><b>00111</b></u>	<u><b>7</b></u>	
<b>01110</b>	<b>14</b>	

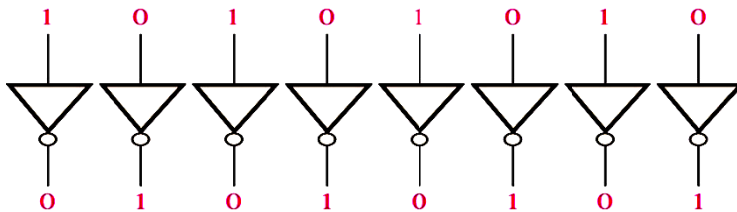
## 1'S AND 2'S COMPLEMENTS OF BINARY NUMBERS

### 1'S COMPLEMENT

The 1's complement of a binary number is just the inverse of the digits. To form the 1's complement, change all **0's** to **1's** and all **1's** to **0's**.

For example, the 1's complement of **11001010** is 00110101

In digital circuits, the 1's complement is formed by using inverters:



### 2'S COMPLEMENT:

The 2's complement of a binary number is found by adding **1** to the **LSB** of the 1's complement.

Recall that the 1's complement of **11001010** is  
**00110101** (1's complement)

To form the 2's complement, add 1:

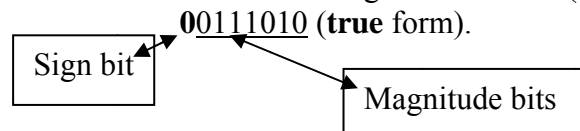
$$\begin{array}{r} +1 \\ \hline 00110101 \\ \hline 00110110 \end{array} \text{ (2's complement)}$$

## SIGNED BINARY NUMBERS

There are several ways to represent signed binary numbers. In all cases, the **MSB** in a signed number is the **sign bit**, that tells you if the number is **positive** or **negative**.

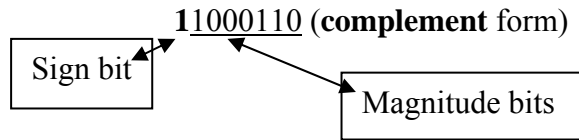
Computers use a **modified 2's complement** for signed numbers. **Positive** numbers are stored in **true** form (with a 0 for the sign bit) and **negative** numbers are stored in **complement** form (with a 1 for the sign bit).

For example, the **positive** number **58** is written using **8-bits** in **true (un-complemented)** as:



**Negative** numbers are written as the 2's complement of the corresponding positive number.

The negative number -58 is written as:



An easy way to read a signed number that uses this notation is to assign the **sign bit** a column **weight** of **-128** (for an **8-bit** number). Then add the column weights for the **1**'s.

**Example:** Assuming that the sign bit = -128, show that **11000110** = **-58** as a **2**'s complement signed number.

Solution: Column weights

-128	64	32	16	8	4	2	1.
<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>-128</b>	<b>+64</b>	<b>+4</b>	<b>+2</b>				<b>= -58</b>

## ARITHMETIC OPERATIONS WITH SIGNED NUMBERS

### ADDITION

Using the **signed** number notation with **negative** numbers in **2**'s complement form simplifies addition and subtraction of signed numbers.

**Rules** for addition: Add the two **signed** numbers. **Discard** any **final carries**. The result is in **signed** form.

**Examples:**

$$\begin{array}{r} 00011110 = +30 \\ 00001111 = +15 \\ \hline 00101101 = 45 \end{array}$$

$$\begin{array}{r} 00001110 = +14 \\ 11101111 = -17 \\ \hline 11111101 = -3 \end{array}$$

$$\begin{array}{r} 11111111 = -1 \\ 11111000 = -8 \\ \hline 1\ 11110111 = -9 \end{array}$$

Discard Final

### 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 have the same sign.

**Two examples are:**

$$\begin{array}{r} 01000000 = +128 \\ 01000001 = +129 \\ \hline 10000001 = -126 \end{array}$$

Discard

$$\begin{array}{r} 10000001 = -127 \\ 10000001 = -127 \\ \hline 1\ 00000010 = -2 \end{array}$$

Wrong! The answer is incorrect and the sign bit has changed

## SUBTRACTION

**Rules** for subtraction: 2's complement the subtrahend and **add** the numbers. **Discard** any **final carries**. The result is in **signed** form.

**Examples:**

$$\begin{array}{r} 00011110 = +30 \\ 11110001 = -15 \\ \hline 1\ 00001111 = +15 \end{array}$$

↙ ↘

Discard Final

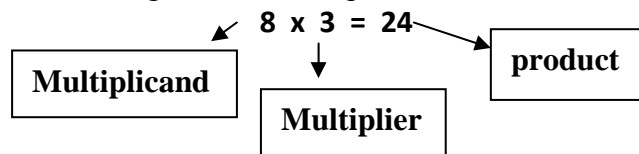
$$\begin{array}{r} 11111111 = -1 \\ 00001000 = +8 \\ \hline 1\ 00000111 = +7 \end{array}$$

↙ ↘

Discard Final

## MULTIPLICATION

The numbers in a multiplication are the **multiplicand**, the **multiplier**, and the **product**. These are illustrated in the following decimal multiplication:



The **multiplication** operation in most computers is accomplished using **addition** in two different ways: **direct addition** and **partial products**.

In direct addition method, you add the multiplicand a number of times equal to the multiplier. The disadvantage of this approach is that it becomes very lengthy if the multiplier is a large number. For example, to multiply **350 x 75**, you must add 350 to itself **75** times.

The **partial product** method is more common, the multiplicand is multiplied by each multiplier digit begins with the least significant digit, and the result is called a **partial product**. Each successive partial product is moved (**shifted**) one place to the left, and when all partial products have been produced, they are **added** to get the **final product**.

The **rules** for partial product are:

- Determine if the **sign** of the **multiplicand** and **multiplier** are the same or different:  
If the signs are the same, the product is positive  
If the signs are **different**, the product is **negative**
- Change any **negative** number to **true** form (uncomplemented).
- Starting with the LSB of the multiplier, generate partial products, shift each successive partial product one bit to the left, and add each successive partial product to the sum of the previous partial products to get the **final product**.

- If the sign bit that was determined in step 1 is negative, take the 2' complement of the product. If positive, leave the product in true form
- Attach the sign bit to the product.

**Example:** Multiply the **signed** binary numbers: **01010011** (multiplicand) and **11000101** (multiplier).

**Solution:**

Since the two numbers have different signs, the sign bit of the product will be negative (**1**).

Take the **2' complement** of the multiplier to put it in **true** form **11000101 --> 00111011**  
The multiplication proceeds as follows. Notice that only the **magnitude** bits are used

	1010011	<b>Multiplicand</b>
<b>x</b>	<b>0111011</b>	<b>Multiplier</b>
	1010011	1 <sup>st</sup> partial product
<b>+</b>	1010011	2 <sup>nd</sup> partial product
	11111001	<b>Sum of 1<sup>st</sup> and 2<sup>nd</sup></b>
<b>+</b>	<b>0000000</b>	3 <sup>rd</sup> partial product
	011111001	<b>Sum</b>
<b>+</b>	1010011	4 <sup>th</sup> partial product
	1110010001	<b>Sum</b>
<b>+</b>	1010011	5 <sup>th</sup> partial product
	100011000001	<b>Sum</b>
<b>+</b>	1010011	6 <sup>th</sup> partial product
	1001100100001	<b>Sum</b>
<b>+</b>	<b>0000000</b>	7 <sup>th</sup> partial product
	<b>1001100100001</b>	<b>Final (Sum) product</b>

1. Since the final product should be **negative**, take the **2' complement** of the final product  
1001100100001      0110011011111
2. Attach the sign bit → **10110011011111**

## DIVISION

The numbers in a division are the **dividend**, the **divisor**, and the **quotient**. These are illustrated in the following standard format:



**dividend / divisor = quotient**

The **division** operation in computers is accomplished using **subtraction**. Since subtraction is done with an **adder**, division can also be accomplished with **adder**.

The result of a division is called the **quotient**; the quotient is the number of times that the divisor will go into the dividend. This means that the divisor can be **subtracted** from the dividend a number of times equal to the **quotient**, as illustrated by dividing **21** by **7** using subtraction operations:

$$21 - 7 = 14 - 7 = 7 - 7 = 0$$

In this simple example, the divisor was subtracted from the dividend **3** times before a **remainder** of **0** was obtained. Therefore, the quotient is **3**.

When two binary numbers are divided, both numbers must be in **true** (uncomplemented) form. The basic **rules** in a division process are as follows:

1. Determine if the **sign** of the **dividend** and **divisor** are the same or different:
  - If the signs are **the same**, the **quotient** is **positive**
  - If the sign are **different**, the **quotient** is **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 the partial **remainder** is **positive**, go to step to step **3**. If the partial **remainder** is zero or negative, the division is complete.
3. Subtract the divisor from the partial remainder and add **1** to the quotient. If the result is positive, repeat for the next partial remainder. If the result is zero or negative, the division is complete.
4. Continue to **subtract** the divisor from the dividend and the partial remainder until there is a **zero** or **negative** result.
5. Count the number of times that the divisor is subtracted and you have the quotient.

Example: Divide 01100100 by 00011001

Solution:

1. The sign of both numbers are positive, so the quotient will be positive. The quotient is initially zero: **00000000**.
2. Subtract the divisor from the dividend using 2's complement addition (remember that the final carries are discarded).

01100100	dividend
+ 11100111	2's complement of divisor
<hr style="width: 100%; border: 0.5px solid black;"/>	

01001011      positive 1<sup>st</sup> partial remainder

Add 1 to quotient: 00000000 + 00000001 = 00000001

3. Subtract the divisor from the 1<sup>st</sup> partial remainder using 2's complement addition.

01001011      1<sup>st</sup> partial remainder  
 + 11100111      2's complement of divisor  
 -----  
 00110010      positive 2<sup>nd</sup> partial remainder

Add 1 to quotient: 00000001 + 00000001 = 00000010

4. Subtract the divisor from the 2<sup>nd</sup> partial remainder using 2's complement addition.

00110010      2<sup>st</sup> partial remainder  
 + 11100111      2's complement of divisor  
 -----  
 00011001      positive 3<sup>rd</sup> partial remainder

Add 1 to quotient: 00000010 + 00000001 = 00000011

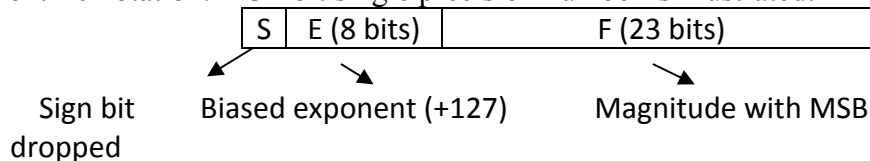
5. Subtract the divisor from the 3<sup>rd</sup> partial remainder using 2's complement addition.

00011001      3<sup>rd</sup> partial remainder  
 + 11100111      2's complement of divisor  
 -----  
 00000000      zero remainder

Add 1 to quotient: 00000011 + 00000001 = 00000100 (final quotient), the process is complete.

## FLOATING POINT NUMBERS

Floating point notation is capable of representing **very large** or **small** numbers by using a form of scientific notation. A 32-bit single precision number is illustrated:



**Example:** Express the speed of light,  $c$ , in single precision floating point notation.  
 $(c = 0.2998 \times 10^9)$

**Solution:**

**Binary:**  $c = 0001\ 0001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000_2$

**Scientific notation:**  $c = 1.001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000 \times 2^{28}$

**Floating point notation:**

0	10011011	001 1101 1110 1001 0101 1100
---	----------	------------------------------

**S** = 0 because the number is positive.

**E** =  $28 + 127 = 155_{10} = 1001\ 1011_2$ .

**F** = the next 23 bits after the first 1 is dropped

## HEXADECIMAL NUMBERS

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

Column weights

$$\begin{array}{cccc} 16^3 & 16^2 & 16^1 & 16^0 \\ 4096 & 256 & 16 & 1 \end{array}$$

**Example:** Express  $1A2F_{16}$  in decimal.

**Solution:** Start by writing the column weights:

$$\begin{array}{cccc} 4096 & 256 & 16 & 1 \\ 1 & A & 2 & F \\ 1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10} \end{array}$$

**Or Binary form:**

$$\begin{array}{cccc} 1 & A & 2 & F \\ 0001 & 1010 & 0010 & 1111 \end{array}$$

**Weights**

$$\dots 4096 \ 2048 \ 1024 \ 512 \ 256 \ 128 \ 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1$$

Decimal	Hexa	Binary
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

**Result**  $4096+2048+512+32+8+4+2+1 = 6703_{10}$

## HEXADECIMAL ARITHMETIC

### HEXADECIMAL ADDITION

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Hexadecimal Addition Table

#### Examples

$$\begin{array}{r}
 \text{Carry} \\
 \text{Addend} \quad 3B2 \\
 \text{Augend} \quad + 41C \\
 \hline
 \text{Sum}
 \end{array}
 \qquad
 \begin{array}{r}
 0 \\
 3B2 \\
 + 41C \\
 \hline
 E
 \end{array}
 \qquad
 \begin{array}{r}
 00 \\
 3B2 \\
 + 41C \\
 \hline
 CE
 \end{array}
 \qquad
 \begin{array}{r}
 00 \\
 3B2 \\
 + 41C \\
 \hline
 7CE
 \end{array}$$

$$\begin{array}{r}
 \text{Carry} \\
 \text{Addend} \quad A27 \\
 \text{Augend} \quad + C3B \\
 \hline
 \text{Sum}
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 A27 \\
 + C3B \\
 \hline
 2
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 A27 \\
 + C3B \\
 \hline
 62
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 A27 \\
 + C3B \\
 \hline
 662
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 A27 \\
 + C3B \\
 \hline
 1662
 \end{array}$$

## HEXADECIMAL SUBTRACTION

Uses the same principle of "borrowing" that decimal and binary subtraction uses.

### Example

Borrow				
Minuend	6E	6E	6E	
Subtrahend	- 29	- 29	-29	
Difference	<u>        </u>	<u>        </u>	<u>        </u>	
		5	45	
Minuend	AC3	AC3	AC3	AC3
Subtrahend	- 604	- 604	- 604	- 604
Difference	<u>        </u>	<u>        </u>	<u>        </u>	<u>        </u>
		F	BF	4BF

## OCTAL NUMBERS

Octal uses eight characters the numbers 0 through 7 to represent numbers. There is no 8 or 9 character in octal. Binary number can easily be converted to octal by grouping **3 bits** at a time and writing the equivalent octal character for each group.

**Example:** Express  $1\ 001\ 011\ 000\ 001\ 110_2$  in octal

### Solution:

Group the binary number by **3-bits** starting from the right.

Thus,  $1\ 001\ 011\ 000\ 001\ 110_2 = 113016_8$

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

**Octal** is also a **weighted** number system. The column weights are powers of **8**, which increase from right to left.

Column weights  $8^3\ 8^2\ 8^1\ 8^0$   
**512 64 8 1**

**Example:** Express  $3702_8$  in decimal

### Solution:

Start by writing the column weights:

$$\begin{array}{r}
 512\ 64\ 8\ 1 \\
 3\ 7\ 0\ 2 \\
 3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}
 \end{array}$$

## OCTAL ARITHMETIC

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	9
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Octal Addition Table

### Examples

Carry		1		
Addend	127	127	127	127
Augend	+ 42	+ 42	+ 42	+ 42
	<u>        </u>	<u>        </u>	<u>        </u>	<u>        </u>
Sum		1	71	171

Carry		1	11	11	1
Addend	1777	1777	1777	1777	1777
Augend	+ 777	+ 777	+ 777	+ 777	+ 777
	<u>        </u>	<u>        </u>	<u>        </u>	<u>        </u>	<u>        </u>
Sum		6	76	776	2776

## OCTAL SUBTRACTION

This is performed exactly like binary and decimal subtraction with the borrowing technique. Whenever the subtrahend is larger than the minuend, a 1 is borrowed from the next column.

### Example

Minuend	124	124	124
Subtrahend	- 63	- 63	- 63
	<u>        </u>	<u>        </u>	<u>        </u>
Sum		1	41

## GENERAL "B'S COMPLEMENT" NUMBERS FOR BASE B AND BASE 2

Everything flows in general the same as above except for the changes shown in the table below. In this table, "b" in column 2 represents a general base b number. Column 3 represents binary numbers.

Row 2 gives the representation of the number -1 (with leading digits equal to b-1, b the base).

Row 3 gives the equation for complementing each of the digits (in each case, subtract the digit from b-1, b the base)

Row 4 gives examples of positive numbers (with leading zeros extending leftward to infinity).

Row 5 shows the complement of the number in row 4 (and shows that the sum of the two equals zero)

	<b>Base 10's</b> <b>(base - 1) = 10 - 1 = 9</b>	<b>Base b</b> <b>(base - 1) = (b - 1) = B</b>	<b>Base 2</b> <b>(base - 1) = (2 - 1) = 1</b>
<b>Number = -1</b>	...9...9999999999	...B...BBBBBBBBBB	...1...1111111111
Complement $d_k$ of digit $c_k$	$d_k = 9 - c_k$	$d_k = (b - 1) - c_k$ $= B - c_k$	$d_k = 1 - c_k$ If $c_k = 1, d_k = 0$ If $c_k = 0, d_k = 1$
Example of 8 digit positive number (leading 0 required)	00034857	For base 5, as an example, 00340240	01011010
Example of 8 digit negative number (leading (b-1) required) with magnitude equal to positive number in row above	00034857 -> 99965143  $\begin{array}{r} 00034857 \\ + \underline{99965143} \\ \hline 00000000 \end{array}$	00340240 -> 44104210  $\begin{array}{r} 00340240 \\ + \underline{44104210} \\ \hline 00000000 \end{array}$	01011010 -> 10100110  $\begin{array}{r} 01011010 \\ + \underline{10100110} \\ \hline 00000000 \end{array}$

## BINARY CODED DECIMAL (BCD)

Binary coded decimal (**BCD**) is a weighted code that is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.

The table illustrates the difference between straight binary and BCD. **BCD** represents each decimal digit with a **4-bit** code. Notice that the codes 1010 through 1111 are not used in BCD.

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

### GRAY CODE

Gray code is an **unweighted** code that has a **single bit change** between one code word and the next in a sequence. Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

A shaft encoder is a typical application. Three IR emitter/detectors are used to encode the position of the shaft. The encoder on the left uses binary and can have three bits change together, creating a potential error. The encoder on the right uses gray code and only 1-bit changes, eliminating potential errors.

Decimal	Binary	Gray
0	000	0000
1	001	0001
2	010	0011
3	011	0010
4	100	0110
5	101	0111
6	110	0101
7	0111	0100

### **Binary-to-Gray Conversion**

The algorithm to convert a binary value to its corresponding standard Gray code value is as follows:

1. Retain the MSB.
2. From left to right, add each adjacent pair of binary code bits to get the next Gray code bit, discarding the carry.



The following example shows the conversion of binary number  $(10110)_2$  to its corresponding standard Gray code value,  $(11101)_{\text{Gray}}$ .

$$\begin{array}{|c|} \hline 1 \ 0 \ 1 \ 1 \ 0 \ \text{Binary} \\ \hline \downarrow \\ \hline 1 \qquad \qquad \text{Gray} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline \underline{1} + \underline{0} \ 1 \ 1 \ 0 \ \text{Binary} \\ \hline \downarrow \\ \hline 1 \ \mathbf{1} \qquad \qquad \text{Gray} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline 1 \ \underline{0} + \underline{1} \ 1 \ 0 \ \text{Binary} \\ \hline \downarrow \\ \hline 1 \ 1 \ \mathbf{1} \qquad \qquad \text{Gray} \\ \hline \end{array}$$
  

$$\begin{array}{|c|} \hline 1 \ 0 \ \underline{1} + \underline{1} \ 0 \ \text{Binary} \\ \hline \downarrow \\ \hline 1 \ 1 \ 1 \ \mathbf{0} \qquad \text{Gray} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline 1 \ 0 \ 1 \ \underline{1} + \underline{0} \ \text{Binary} \\ \hline \downarrow \\ \hline 1 \ 1 \ 1 \ 0 \ \mathbf{1} \ \text{Gray} \\ \hline \end{array}$$

### Gray-to-Binary Conversion

The algorithm to convert a standard Gray code value to its corresponding binary value is as follows:

1. Retain the MSB.
2. From left to right, add each binary code bit generated to the Gray code bit in the next position, discarding the carry.

The following example shows the conversion of the standard Gray code value  $(11011)_{\text{Gray}}$  to its corresponding binary value,  $(10010)_2$ .

$$\begin{array}{|c|} \hline 1 \ 1 \ 0 \ 1 \ 1 \ \text{Gray} \\ \hline \downarrow \\ \hline 1 \qquad \qquad \text{Binary} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline 1 \ \underline{1} \ 0 \ 1 \ 1 \ \text{Gray} \\ \hline + \downarrow \\ \hline \underline{1} \ \mathbf{0} \qquad \qquad \text{Binary} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline 1 \ 1 \ \underline{0} \ 1 \ 1 \ \text{Gray} \\ \hline + \downarrow \\ \hline 1 \ \underline{0} \ \mathbf{0} \qquad \qquad \text{Binary} \\ \hline \end{array}$$
  

$$\begin{array}{|c|} \hline 1 \ 1 \ 0 \ \underline{1} \ 1 \ \text{Gray} \\ \hline + \downarrow \\ \hline 1 \ 0 \ \underline{0} \ \mathbf{1} \ \text{Binary} \\ \hline \end{array}
 \quad
 \begin{array}{|c|} \hline 1 \ 1 \ 0 \ 1 \ \underline{1} \ \text{Gray} \\ \hline + \downarrow \\ \hline 1 \ 0 \ 0 \ \underline{1} \ \mathbf{0} \ \text{Binary} \\ \hline \end{array}$$

### EXCESS-3 CODE

The *Excess-3 code* uses a bias value of three. It means we have to add 3(0011) to the corresponding BCD code. Hence the codes for the ten digits 0, 1, ..., 9 are 0011, 0100, ..., 1100 respectively. The decimal number 294 would be represented as 0101 1100 0111.

### 84-2-1 CODE

The *84-2-1 code* uses the weights of 8, 4, -2 and -1 in the coding. The decimal number 294 would be represented as 0110 1111 0100.

### 2421 CODE

The *2421 code* uses the weights of 2, 4, 2 and 1 in the coding. According to the weights, certain digits may have alternative codes. For instance, the digit 3 could be represented as 0011 or 1001. However, we pick the former in the standard 2421 coding, so that the codes for the first five digits 0 – 4 begin with 0, whereas the codes for the last five digits 5 – 9 begin with 1. The decimal number 294 would be represented as 0010 1111 0100.

### ASCII CODE

**ASCII** is a code for alphanumeric characters and control characters. In its original form, **ASCII** encoded **128** characters and symbols using **7-bits**. The first **32** characters are control characters, that are based on obsolete teletype requirements, so these characters are generally assigned to other functions in modern usage.

In 1981, IBM introduced extended **ASCII**, which is an **8-bit** code and increased the character set to **256**. Other extended sets (such as **Unicode**) have been introduced to handle characters in languages other than English.

### ERROR DETECTION

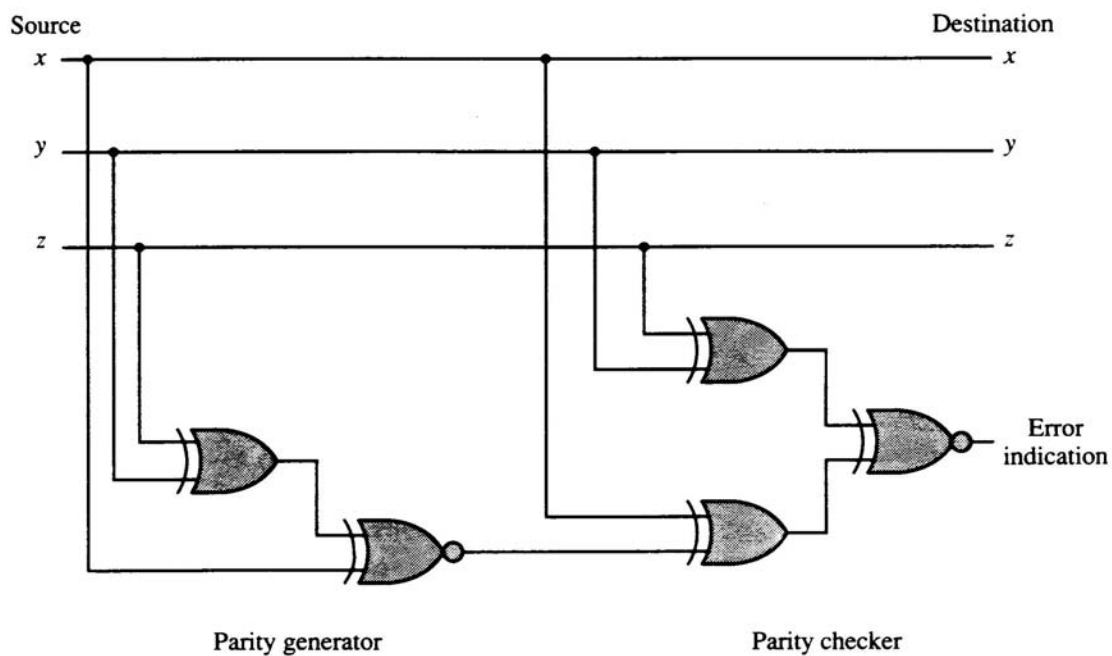
- Transmitted binary information is subject to noise that could change bits 1 to 0 and vice versa
- An *error detection code* is a binary code that detects digital errors during transmission
- The detected errors cannot be corrected, but can prompt the data to be retransmitted
- The most common error detection code used is the *parity bit*
- A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even

**TABLE 3-7** Parity Bit Generation

Message <i>xyz</i>	<i>P(odd)</i>	<i>P(even)</i>
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

- The P(odd) bit is chosen to make the sum of 1's in all four bits odd
- The even-parity scheme has the disadvantage of having a bit combination of all 0's
- Procedure during transmission:
  - At the sending end, the message is applied to a *parity generator*
  - The message, including the parity bit, is transmitted
  - At the receiving end, all the incoming bits are applied to a *parity checker*
  - Any odd number of errors are detected
- Parity generators and checkers are constructed with XOR gates (odd function)
- An odd function generates 1 iff an odd number of input variables are 1

**Figure 3-3** Error detection with odd parity bit.



## Quick Review Questions

- 2-1. Convert the binary number 1011011 to its decimal equivalent.  
 a. 5                      b. 63                      c. 91                      d. 92                      e. 139
- 2-2. What is the weight of the digit '3' in the base-7 number 12345?  
 a. 3                      b. 7                      c. 14                      d. 21                      e. 49
- 2-3. Which of the following has the largest value?  
 a.  $(110)_{10}$               b.  $(10011011)_2$       c.  $(1111)_5$               d.  $(9A)_{16}$               e.  $(222)_8$
- 2-4. If  $(321)_4 = (57)_{10}$ , what is the decimal equivalent of  $(321000000)_4$ ?  
 a.  $57 \times 10^4$               b.  $57 \times 10^6$               c.  $57 \times 4^4$               d.  $57 \times 4^6$               e.  $57^4$
- 2-5. Convert each of the following decimal numbers to binary (base two) with at most eight digits in the fractional part, rounded to eight places.  
 a. 2000                      b. 0.875                      c. 0.07                      d. 12.345
- 2-6. Convert each of the decimal numbers in Question 2-5 above to septimal (base seven) with at most six digits in the fractional part, rounded to six places.
- 2-7. Convert each of the decimal numbers in Question 2-5 above to octal (base eight) with at most four digits in the fractional part, rounded to four places.
- 2-8. Convert each of the decimal numbers in Question 2-5 above to hexadecimal (base sixteen) with at most two digits in the fractional part, rounded to two places.
- 2-9. Which of the following octal values is equivalent to the binary number  $(110001)_2$ ?  
 a.  $(15)_8$                       b.  $(31)_8$                       c.  $(33)_8$                       d.  $(49)_8$                       e.  $(61)_8$
- 2-10. Convert the binary number  $(1001101)_2$  to  
 a. quaternary              b. octal                      c. decimal                      d. hexadecimal
- 2-11. What is  $(1011)_2 \times (101)_2$ ?  
 a.  $(10000)_2$               b.  $(110111)_2$               c.  $(111111)_2$               d.  $(111011)_2$               e.  $(101101)_2$
- 2-12. Perform the following operations on binary numbers.  
 a.  $(10111110)_2 + (10001101)_2$   
 b.  $(11010010)_2 - (01101101)_2$   
 c.  $(11100101)_2 - (00101110)_2$
- 2-13. In a 6-bit 2's complement binary number system, what is the decimal value represented by  $(100100)_{2s}$ ?  
 a. -4                      b. 36                      c. -36                      d. -27                      e. -28

- 2-14. In a 6-bit 1's complement binary number system, what is the decimal value represented by  $(010100)_{1s}$ ?  
a. -11      b. 43      c. -43      d. 20      e. -20
- 2-15. What is the range of values that can be represented in a 5-bit 2's complement binary system?  
a. 0 to 31      b. -8 to 7      c. -8 to 8      d. -15 to 15      e. -16 to 15
- 2-16. In a 4-bit 2's complement scheme, what is the result of this operation:  $(1011)_{2s} + (1001)_{2s}$ ?  
a. 4      b. 5      c. 20      d. -12      e. overflow
- 2-17. Assuming a 6-bit 2's complement system performs the following subtraction operations by converting it into addition operations:  
a.  $(011010)_{2s} - (010000)_{2s}$   
b.  $(011010)_{2s} - (001101)_{2s}$   
c.  $(000011)_{2s} - (010000)_{2s}$
- 2-18. Assuming a 6-bit 1's complement system, perform the following subtraction operations by converting it into addition operations:  
a.  $(011111)_{1s} - (010101)_{1s}$   
b.  $(001010)_{1s} - (101101)_{1s}$   
c.  $(100000)_{1s} - (010011)_{1s}$
- 2-19. Which of the following values cannot be represented accurately in the 8-bit sign-and-magnitude fixed-point number format shown in Figure 2.4?  
a. 4      b. -29.5      c. 20.2      d. -3.75      e. 12.25
- 2-20. What does 1 110 1001 represent in this floating-point number scheme: 1-bit sign, 3-bit normalized mantissa, followed by 4-bit 2's complement exponent?  
a.  $0.125 \times 2^9$       b.  $-0.125 \times 2^9$       c.  $-0.75 \times 2^{-1}$       d.  $-0.75 \times 2^{-6}$       e.  $-0.75 \times 2^{-7}$
- 2-21. How to represent  $(246)_{10}$  in the following system/code?  
a. 10-bit binary      b. BCD      c. Excess-3      d. 2421 code      e. 84-2-1 code
- 2-22. The decimal number 573 is represented as 1111 0110 1011 in an unknown self-complementing code. Find the code for the decimal number 642.
- 2-23. Convert  $(101011)_2$  to its corresponding Gray code value.  
a.  $(101011)_{\text{Gray}}$       b.  $(010100)_{\text{Gray}}$       c.  $(110010)_{\text{Gray}}$       d.  $(111110)_{\text{Gray}}$       e.  $(43)_{\text{Gray}}$
- 2-24. Convert  $(101011)_{\text{Gray}}$  to its corresponding binary value.  
a.  $(101011)_2$       b.  $(010100)_2$       c.  $(110010)_2$       d.  $(111110)_2$       e.  $(010101)_2$

## Answers to Quick Review Questions

2-1. (c)      2-2. (e)      2-3. (c)      2-4. (d)

2-5. (a)  $(2000)_{10} = (11111010000)_2$       (b)  $(0.875)_{10} = (0.111)_2$   
 (c)  $(0.07)_{10} = (0.00010010)_2$       (d)  $(12.345)_{10} = (1100.01011000)_2$

2-6. (a)  $(2000)_{10} = (5555)_7$       (b)  $(0.875)_{10} = (0.606061)_7$   
 (c)  $(0.07)_{10} = (0.033003)_7$  or  $(0.033004)_7$   
 (d)  $(12.345)_{10} = (15.226223)_7$

2-7. (a)  $(2000)_{10} = (3720)_8$       (b)  $(0.875)_{10} = (0.7)_8$   
 (c)  $(0.07)_{10} = (0.0437)_8$       (d)  $(12.345)_{10} = (14.2605)_8$

2-8. (a)  $(2000)_{10} = (7D0)_{16}$       (b)  $(0.875)_{10} = (0.E)_{16}$   
 (c)  $(0.07)_{10} = (0.12)_{16}$       (d)  $(12.345)_{10} = (C.58)_{16}$

2-9. (e)

2-10. (a)  $(1031)_4$       (b)  $(115)_8$       (c)  $(77)_{10}$       (d)  $(4D)_{16}$

2-11. (b)

2-12. (a)  $(101001011)_2$       (b)  $(01100101)_2$       (c)  $(10110111)_2$

2-13. (e)      2-14. (d)      2-15. (e)      2-16. (e)

2-17. (a)  $(001010)_{2s} = (10)_{10}$       (b)  $(001101)_{2s} = (13)_{10}$       (c)  $(110011)_{2s} = -(13)_{10}$

2-18. (a)  $(001010)_{1s} = (10)_{10}$       (b)  $(011100)_{1s} = (28)_{10}$       (c) overflow

2-19. (c)      2-20. (e)

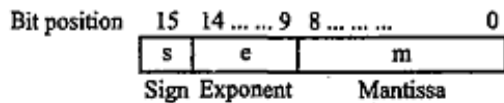
2-21. (a)  $(0011110110)_2$       (b)  $(0010\ 0100\ 0110)_{BCD}$       (c)  $(0101\ 0111\ 1001)_{\text{Excess-3}}$   
 (d)  $(0010\ 0100\ 1100)_{2421}$       (e)  $(0110\ 0100\ 1010)_{84-2-1}$

2-22.  $642 = 0100\ 0000\ 1001$

2-23. (d)      2-24. (c)

Q.1 GATE 2003

The following is a scheme for floating point number representation using 16 bits.



Let  $s$ ,  $e$ , and  $m$  be the numbers represented in binary in the sign, exponent, and mantissa fields respectively. Then the floating point number represented is

$$\begin{cases} (-1)^s(1 + m \times 2^{-9})2^{e-15}, & \text{if the exponent } \neq 11111 \\ 0 & \text{otherwise} \end{cases}$$

What is the maximum difference between two successive real numbers representable in this system?

- (a)  $2^{-40}$                       (b)  $2^{-9}$   
 (c)  $2^{22}$                         (d)  $2^{31}$

Q.2 GATE 2004

If  $73_x$  (in base- $x$  number system) is equal to  $54_y$  (in base- $y$  number system), the possible values of  $x$  and  $y$  are

- (a) 8, 16                      (b) 10, 12  
 (c) 9, 13                      (d) 8, 11

Q.3 GATE 2004

Let  $A = 1111\ 1010$  and  $B = 0000\ 1010$  be two 8-bit 2's complement numbers. Their product in 2's complement is

- (a) 1100 0100                (b) 1001 1100  
 (c) 1010 0101                (d) 1101 0101

Q.4 GATE 2005

The hexadecimal representation of  $657_8$  is

- (a) 1AF                        (b) D78  
 (c) D71                        (d) 32F

Q.5 GATE 2006

We consider the addition of two 2's complement numbers  $b_{n-1} b_{n-2} \dots b_0$  and  $a_{n-1} a_{n-2} \dots a_0$ . A binary adder for adding unsigned binary numbers is used to add the two numbers. The sum is denoted by  $c_{n-1} c_{n-2} \dots c_0$  and the carryout by  $c_{out}$ . Which one of the following options correctly identifies the overflow condition?

- (a)  $c_{out} \overline{(a_{n-1} \oplus b_{n-1})}$   
 (b)  $a_{n-1} b_{n-1} \overline{c_{n-1}} + \overline{a_{n-1} b_{n-1}} c_{n-1}$   
 (c)  $c_{out} \oplus c_{n-1}$   
 (d)  $a_{n-1} \oplus b_{n-1} \oplus c_{n-1}$

Q.6 GATE 2006

Consider numbers represented in 4-bit gray code. Let  $h_3 h_2 h_1 h_0$  be the gray code representation of a number  $n$  and let  $g_3 g_2 g_1 g_0$  be the gray code of  $(n + 1)$  (modulo 16) value of the number. Which one of the following functions is correct?

- (a)  $g_0(h_3 h_2 h_1 h_0) = \Sigma(1, 2, 3, 6, 10, 13, 14, 15)$   
 (b)  $g_1(h_3 h_2 h_1 h_0) = \Sigma(4, 9, 10, 11, 12, 13, 14, 15)$   
 (c)  $g_2(h_3 h_2 h_1 h_0) = \Sigma(2, 4, 5, 6, 7, 12, 13, 15)$   
 (d)  $g_3(h_3 h_2 h_1 h_0) = \Sigma(0, 1, 6, 7, 10, 11, 12, 13)$

Q.7 GATE 2009

$(1217)_8$  is equivalent to

- (a)  $(1217)_{16}$       (b)  $(028F)_{16}$   
 (c)  $(2297)_{10}$       (d)  $(0B17)_{16}$



Q.8 GATE 2010

P is a 16-bit signed integer. The 2's complement representation of P is  $(F87B)_{16}$ . The 2's complement representation of  $8*P$  is

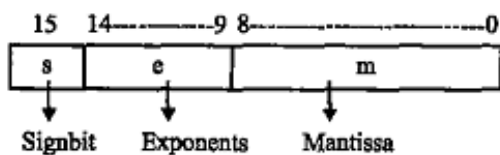
- (a)  $(C3D8)_{16}$     (b)  $(187B)_{16}$   
 (c)  $(F878)_{16}$     (d)  $(987B)_{16}$

### Answers Key

Q.1 C    Q.2 D    Q.3 A    Q.4 A    Q.5 C    Q.6 C    Q.7 B  
 Q.8 A

### Solutions

Q.1  
 (c)



Then the floating point number represented

$$\begin{cases} (-1)^s (1 + m \times 2^{-9}) 2^{e-31}, & \text{if the exponent} \\ 0 \text{ otherwise} & \text{Max} \neq 111111 = 63 \\ & \therefore \text{min} = 111110 = 62 \end{cases}$$

Difference = (max value - min value)

$\therefore$  Maximum value

$$(-1)^s [1 + (m + 1) \times 2^{-9}] \times 2^{e-31}$$

$\therefore e = 62$

Minimum value

$$(-1)^s [1 + m \times 2^{-9}] \times 2^{e-31}$$

$\therefore$  Difference

$$\begin{aligned} &= (-1)^s [1 + (m + 1) \times 2^{-9}] \times 2^{62-31} - (-1)^s \\ &\quad [1 + m \times 2^{-9}] \times 2^{62-31} \\ &= [1 + (m + 1) \times 2^{-9}] 2^{31} - [1 + m \times 2^{-9}] 2^{31} \\ &= 2^{31} \times 2^{-9} \\ &= 2^{22} \end{aligned}$$

Q.2

(d)

$$(73)_x \text{ if } x = 8$$

$$\begin{aligned}(73)_8 &= 7 \times 8^1 + 3 \times 8^0 \\ &= 56 + 3 = 59\end{aligned}$$

$$(54)_y \text{ if } y = 11$$

$$\begin{aligned}(54)_{11} &= 5 \times 11^1 + 4 \times 11^0 \\ &= 55 + 4 = 59\end{aligned}$$

Q.3

(a)

$$A = 11111010$$

$$\begin{aligned}&= -2^7 \times 1 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 \\ &\quad + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 \\ &= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^4 + 2^3 + 2^1 \\ &= -128 + 64 + 32 + 16 + 8 + 2 \\ &= -128 + 122 \\ &= -6\end{aligned}$$

$$B = 00001010$$

$$\begin{aligned}&= 2^3 \times 1 + 2^1 \times 1 \\ &= 8 + 2 = 10\end{aligned}$$

$$A \cdot B = (-6) \times 10$$

$$= -60$$

So the value of AB in 2's complement representation

$$AB = 11000100$$

Q.4

(a)

$$\begin{aligned}657_{(8)} &= 6 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 \\ &= 6 \times 64 + 5 \times 8 + 7 \times 1 \\ &= 384 + 40 + 7 \\ &= 431\end{aligned}$$

$$\begin{array}{r|l} 16 & 431 \quad 15 \\ 16 & \underline{26} \quad 10 \\ & \underline{\quad} \quad 1 \end{array}$$

$$11015$$

$$1AF$$

Q.5

(c)

$$B = b_{n-1} b_{n-2} \dots b_0$$

$$A = a_{n-1} a_{n-2} \dots a_0$$

$$S = c_{n-1} c_{n-2} \dots c_0$$

The over flow condition V

$$V = b'_{n-1} a'_{n-1} c_{n-2} + b_{n-1} a_{n-1} c'_{n-2}$$

 $c_{n-1}$  is  $c_{out}$  so

$$V = c_{out} \oplus c_{n-1}$$

So the over flow condition is

$$c_{out} \oplus c_{n-1}$$

Q.6

(c)

$h_3 h_2 h_1 h_0$  is the gray code representation of number  $n$ ,  $g_3 g_2 g_1 g_0$  is the gray code representation of  $(n+1)$  (modulo 16)

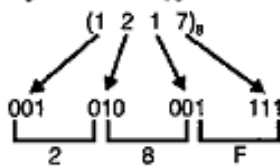
$(n+1)$  is 5 bit in decimal its range is 0 to 31.

So  $g_1(h_3 h_2 h_1 h_0) = \sum(4, 9, 10, 11, 12, 13, 14, 15)$  is the correct function for  $(n+1)$  mod 16

Q.7

(b)

$$(1217)_8 = (028F)_{16}$$



Q.8

(a)

2's complement representation of P is (F87B)<sub>16</sub>

$$\text{So } (F87B)_{16} = (1111100001111011)_2$$

So 2's complement of P means =  $\bar{P} + 1$

$$\text{So } \bar{P} + 1 = 1111\ 1000\ 0111\ 1011$$

$$\bar{P} = 1111\ 1000\ 0111\ 1010$$

$$\text{So } P = 0000\ 0111\ 1000\ 0101$$

$$\text{So } P = 1 \times 2^0 + 1 \times 2^2 + 1 \times 2^7 + 1 \times 2^8 + 1 \times 2^9 + 1 \times 2^{10}$$

$$= 1 + 4 + 128 + 256 + 512 + 1024 = 1925$$

$$\text{So } 8 \times P = 8 \times 1925 = 15400$$

Convert the into binary

2	15400	0
2	7700	0
2	3850	0
2	1925	1
2	962	0
2	481	1
2	240	0
2	120	0
2	60	0
2	30	0
2	15	1
2	7	1
2	3	1
	1	

$$\text{So } \underline{0011110000101000}$$

$$\text{So } 8 \times P = 0011\ 1100\ 0010\ 1000$$

So 1's complement of

$$8 \times P = 1100\ 0011\ 1101\ 0111$$

2's complement of  $8 \times P$

$$= \underline{1100001111011000}$$

$$= (C3\ D8)_{16}$$