Digital Logic & Circuit Design

Number Systems & Codes

Prepared by:
Nilesh Patidar and Shiraz Husain

Number Systems & Codes

- If base or radix of a number system is ‘r’, then the numbers present in that number system are ranging from zero to r-1.
- The total numbers present in that number system is ‘r’.
- So, we will get various number systems, by choosing the values of radix as greater than or equal to two.
- The following number systems are the most commonly used.
  - Decimal Number system (0,1,2,3,4….9) – Base 10
  - Binary Number system (0,1) – Base 2
  - Octal Number system (0,1,2,3……7) – Base 8
  - Hexadecimal Number system (0,1,2,3…..F) – Base 16

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

Decimal Number System

- The **base** or radix of Decimal number system is **10**.
- So, the numbers ranging from 0 to 9 are used in this number system.
- The part of the number that lies to the left of the **decimal point** is known as integer part.
- Similarly, the part of the number that lies to the right of the decimal point is known as fractional part.
- In this number system, the successive positions to the left of the decimal point having weights of 10^0, 10^1, 10^2, 10^3 and so on.
- Similarly, the successive positions to the right of the decimal point having weights of 10^{-1}, 10^{-2}, 10^{-3} and so on. That means, each position has specific weight, which is **power of base 10**

Example-

- Consider the **decimal number 1358.246**. Integer part of this number is 1358 and fractional part of this number is 0.246.
- The digits 1, 3, 5 and 8 have weights of 10^3, 10^2, 10^1 and 10^0 respectively.
- Similarly, the digits 2, 4 and 6 have weights of 10^{-1}, 10^{-2} and 10^{-3} respectively.
- **Mathematically**, we can write it as

$$1358.246 = (1 \times 10^3) + (3 \times 10^2) + (5 \times 10^1) + (8 \times 10^0) + (2 \times 10^{-1}) + (4 \times 10^{-2}) + (6 \times 10^{-3})$$

Binary Number System

- The **base** or radix of this number system is **2**.
- So, the numbers 0 and 1 are used in this number system.
- The part of the number, which lies to the left of the **binary point** is known as integer part.
- Similarly, the part of the number, which lies to the right of the binary point is known as fractional part.
- In this number system, the successive positions to the left of the binary point having weights of 2^0, 2^1, 2^2, 2^3 and so on. Similarly, the successive positions to the right of the binary point having weights of 2^{-1}, 2^{-2}, 2^{-3} and so on. That means, each position has specific weight, which is **power of base 2**.

Example

- Consider the **binary number 1101.011**.
- Integer part of this number is 1101 and fractional part of this number is 0.011.
- The digits 1, 0, 1 and 1 of integer part have weights of 2^0, 2^1, 2^2, 2^3 respectively.
- Similarly, the digits 0, 1 and 1 of fractional part have weights of 2^{-1}, 2^{-2}, 2^{-3} respectively.
- **Mathematically**, we can write it as

$$1101.011 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$$

Octal Number System

- The **base** or radix of octal number system is **8**.
- So, the numbers ranging from 0 to 7 are used in this number system.
- The part of the number that lies to the left of the **octal point** is known as integer part. Similarly, the part of the number that lies to the right of the octal point is known as fractional part.
- In this number system, the successive positions to the left of the octal point having weights of 8^0, 8^1, 8^2, 8^3 and so on.
- Similarly, the successive positions to the right of the octal point having weights of 8^{-1}, 8^{-2}, 8^{-3} and so on. That means, each position has specific weight, which is **power of base 8**.

Example

- Consider the **octal number 1457.236**.
- Integer part of this number is 1457 and fractional part of this number is 0.236.
- The digits 7, 5, 4 and 1 have weights of 8^0, 8^1, 8^2 and 8^3 respectively. Similarly, the digits 2, 3 and 6 have weights of 8^{-1}, 8^{-2}, 8^{-3} respectively.
- **Mathematically**, we can write it as

$$1457.236 = (1 \times 8^3) + (4 \times 8^2) + (5 \times 8^1) + (7 \times 8^0) + (2 \times 8^{-1}) + (3 \times 8^{-2}) + (6 \times 8^{-3})$$

Hexadecimal Number System

- The **base** or radix of Hexa-decimal number system is **16**.
- So, the numbers ranging from 0 to 9 and the letters from A to F are used in this number system.
- The decimal equivalent of Hexa-decimal digits from A to F are 10 to 15.
- The part of the number, which lies to the left of the **hexadecimal point** is known as integer part.
- Similarly, the part of the number, which lies to the right of the Hexa-decimal point is known as fractional part.
- In this number system, the successive positions to the left of the Hexa-decimal point having weights of 16^0, 16^1, 16^2, 16^3 and so on. Similarly, the successive positions to the right of the Hexa-decimal point having weights of 16^{-1}, 16^{-2}, 16^{-3} and so on. That means, each position has specific weight, which is **power of base 16**.

Examples

- Consider the **Hexa-decimal number 1A05.2C4**.
- Integer part of this number is 1A05 and fractional part of this number is 0.2C4.
- The digits 5, 0, A and 1 have weights of 16^0, 16^1, 16^2 and 16^3 respectively. Similarly, the digits 2, C and 4 have weights of 16^{-1}, 16^{-2} and 16^{-3} respectively.
- **Mathematically**, we can write it as

$$1A05.2C4 = (1 \times 16^3) + (10 \times 16^2) + (0 \times 16^1) + (5 \times 16^0) + (2 \times 16^{-1}) + (12 \times 16^{-2}) + (4 \times 16^{-3})$$

Decimal to Binary

Decimal to Octal

Decimal to Hexadecimal

16 x 1 = 16
16 x 2 = 32
16 x 3 = 48
16 x 4 = 64
16 x 5 = 80
16 x 6 = 96
16 x 7 = 112
16 x 8 = 128
16 x 9 = 144
16 x 10 = 160

Binary to octal

Binary to Decimal

Binary to Hexadecimal

Octal to Decimal

Octal to Binary

Octal to Hexadecimal

Hexadecimal to Binary

Hexadecimal to Octal

Hexadecimal to Decimal

Classification of Complements

- For each radix-r system, there are following two types of compliments:
  - The radix complement
  - The diminished radix complement
- The radix complement is referred to as the r's complement and the diminished radix complement is referred to as (r-1)'s complement.
- Let us consider the binary system with base r=2. Hence, the two types of complements for the binary system are 2's complement and 1's complement.
- Similary for octal system we have 8's and 7's complement.
- For decimal system we have 9's and 10's complement.

1’s Complement

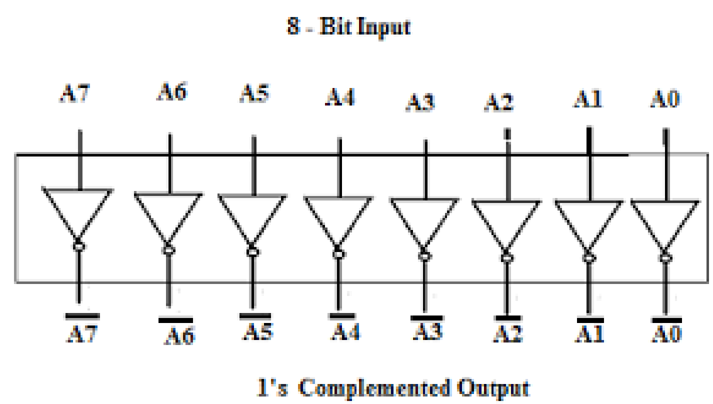
- The 1’s complement of a number is the number that results when we complement each bit.
- The 1’s complement of a number is used to represent negative numbers.
- The 1’s complement can be easily achieved using inverters

To represent -34 in 1’s complement form

+ 34 = 0 0 1 0 0 0 1 0

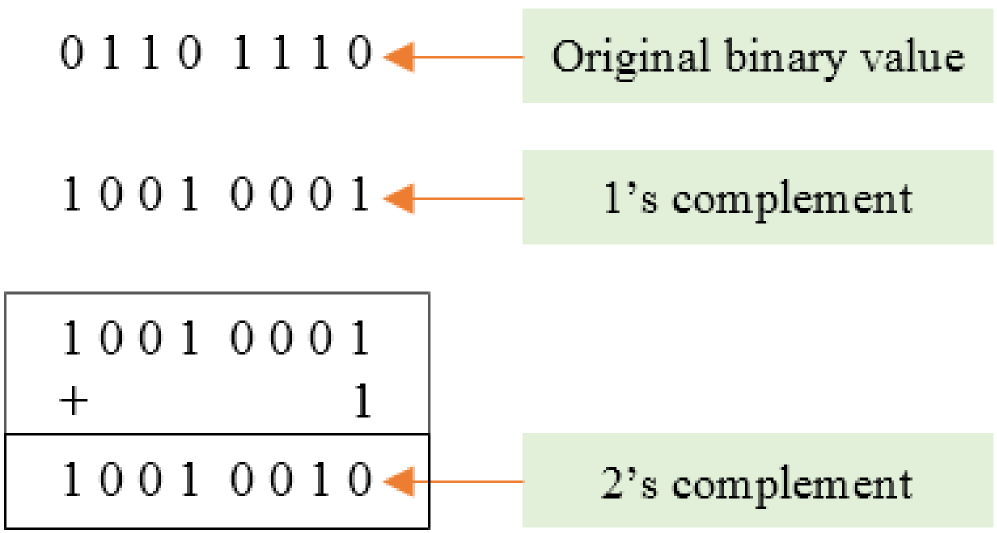
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

- 34 = 1 1 0 1 1 1 0 1 (1’s complement of + 34)



2'S Complement

- The 2's complement of a binary number is obtained by adding 1 to 1's complement of that number.
- Therefore, 2's complement = 1's complement + 1
- 2's complement of -110 is



Signed Binary Numbers

- To represent negative integers, we need a notation for negative values.
- It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.
- The convention is to make the sign bit 0 for positive and 1 for negative.
- Three methods are the sign/magnitude representation , the 1's
- complement and the 2's complement method of representation.
- Example: to represent the signed number (-9)

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Binary Addition

- Add $(25)_{10}$ and $(14)_{10}$

$$\begin{array}{r} 25 \\ 14 \\ \hline 39 \end{array} \qquad \begin{aligned} (25)_{10} &= 0\;0011001 \\ (14)_{10} &= 0\;0001110 \end{aligned}$$

$$\begin{array}{r} 0\;0\;0\;1\;1\;0\;0\;1 \\ 0\;0\;0\;0\;1\;1\;1\;0 \\ \hline \boxed{0}\;\underbrace{\boxed{0}\;0\;1\;0\;0\;1\;1\;1}_{39} \end{array} \quad \text{(Actual magnitude)}$$

Carry

MSB (+ve)

*No role of 1's and 2's Complement in Addition of positive numbers

1's Complement

- Add $(25)_{10}$ and $(-14)_{10}$

25	$(25)_{10}$ =	0 0011001
- 14	$(14)_{10}$ =	0 0001110
11	$(-14)_{10}$ =	1 1110001

```
    0 0 0 1 1 0 0 1
    1 1 1 1 0 0 0 1
  -----------------
  1 0 0 0 0 1 0 1 0    (Result in 1's Comp. form)
                + 1    (Add end-around carry)
  -----------------
    0 0 0 0 1 0 1 1    (Actual magnitude)
```

$\underbrace{0\,0\,0\,0\,1\,0\,1\,1}_{11}$

Carry

MSB (+ve)

1's Complement

- Add $(-25)_{10}$ and $(14)_{10}$

14	$(14)_{10}$	=	0 0001110
- 25	$(25)_{10}$	=	0 0011001
- 11	$(-25)_{10}$	=	1 1100110

```
        0 0 0 0 1 1 1 0
        1 1 1 0 0 1 1 0
      ------------------
Carry 0 1 1 1 1 0 1 0 0   (Result in 1's Comp. form)
        0 0 0 0 1 0 1 1   (Actual magnitude)
              11
MSB
(-ve)
```

1's Complement

- Add $(-25)_{10}$ and $(-14)_{10}$

-14	$(14)_{10}$	=	0 0001110
- 25	$(25)_{10}$	=	0 0011001
- 39	$(-25)_{10}$	=	1 1100110
	$(-14)_{10}$	=	1 1110001

```
   1 1 1 0 0 1 1 0
   1 1 1 1 0 0 0 1
 -----------------
 1 1 1 0 1 0 1 1 1     (Result in 1's Comp. form)
               + 1     (Add end around carry
   1 1 0 1 1 0 0 0     (Actual magnitude)
 -----------------
   0 0 1 0 0 1 1 1
```

$\underbrace{0\ 0\ 1\ 0\ 0\ 1\ 1\ 1}_{39}$

Carry

MSB
(-ve)

2’s Complement

- Add $(-18)_{10}$ with $(30)_{10}$

30	$(30)_{10}$	=	0 0011110	
- 18	$(18)_{10}$	=	0 0010010	
12	$(-18)_{10}$	=	1 1101101	(1’s Comp)
			+1	
			1 1101110	(2’s comp)

$$
\begin{array}{r}
0\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \\
+\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0 \\
\hline
\boxed{1}\ \underbrace{\boxed{0}\ 0\ 0\ 0\ 1\ 1\ 0\ 0}_{12}
\end{array}
$$

(Actual magnitude)

Discard Carry

MSB (+ve)

2's Complement

- Add $(18)_{10}$ with $(-30)_{10}$

18	$(18)_{10}$	=	0 0010010	
- 30	$(30)_{10}$	=	0 0011110	
- 12	$(-30)_{10}$	=	1 1100001	(1's Comp)
			+1	
	$(-30)_{10}$	=	1 1100010	(2's comp)

	0 0 0 1 0 0 1 0	
	+ 1 1 1 0 0 0 1 0	
Discard Carry	$\boxed{0}\ \boxed{1}$ 1 1 1 0 1 0 0	(result in 2's comp. form)
MSB (-ve)	0 0 0 0 1 0 1 1	(1's complement form of result)
	+ 1	
	0 0 0 0 1 1 0 0	(Actual magnitude) = $(12)_{10}$

2's Complement

$(18)_{10}$ = 00010010

$(-18)_{10}$ = 11101101 (1's Comp)

+1

11101110 (2's Comp)

- Add $(-18)_{10}$ with $(-30)_{10}$

$$\begin{array}{r} -18 \\ -30 \\ \hline -48 \end{array}$$

$(30)_{10}$ = 0 0011110

$(-30)_{10}$ = 1 1100001 (1's Comp)

+1

$(-30)_{10}$ = 1 1100010 (2's comp)

$$\begin{array}{r} 1\,1\,1\,0\,1\,1\,1\,0 \\ +\,1\,1\,1\,0\,0\,0\,1\,0 \\ \hline \boxed{1}\,\boxed{1}\,1\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,1\,1 \\ +\,1 \\ \hline 0\,0\,1\,1\,0\,0\,0\,0 \end{array}$$

Discard Carry

MSB (-ve)

(result in 2's comp. form)

(1's complement form of result)

(2's comp Actual magnitude) = $(48)_{10}$

1's Complement representation

- **Range**: Using n bits, the range of numbers that can be represented is from $-(2^{n-1} - 1)$ to $+2^{n-1} - 1$.
- **Advantages**:
  - Still relatively simple to represent the numbers,
  - Simpler Add/Subtract circuit design (subtracting a number from another involves complementing the subtracted and then adding it to the other number).
- **Disadvantages**:
  - Has the problem of double representing the 0 (-0 and +0),
  - It may require two addition operations as if there is a carry out it has to be added to get the correct result
- **Overflow**: occurs when the result is out of range

2’s Complement representation

- Advantages:
  - No double representation of 0 (the 2's complement of 0 is still 0),
  - Simplest Add/Subtract circuit design (subtracting a number from another involves 2's complementing the subtracted and then adding it to the other number),
  - Add/Subtract operations is done in one-step, the end carry is only examined to determine if an overflow has occurred otherwise it is discarded.
  - The end result is already represented in 2's complement (only if there is no overflow).
- That is why this is the most preferred method for signed-number representations in computers.

2’s Complement representation

- Disadvantages:
  - The unsymmetrical range, which is not a serious problem,
  - It is slightly more complex to obtain the 2's complement (it involves complementing and adding 1). However this can be accomplished very easily during the Add/Subtract operations (by making the first carry in 1 and complementing the subtrahend).
- Overflow: occurs if the result is out of range

9's Complement

When subtrahend is smaller than the minuend

General Subtraction

$$
\begin{array}{r}
841 \\
-\ 329 \\
\hline
512
\end{array}
$$

Subtraction using 9's Complement

$$
\begin{array}{rl}
841 & \\
+\ 670 & \leftarrow \text{(9's Complement of 329)} \\
\hline
1\,511 & \\
+1 & \\
\hline
512 &
\end{array}
$$

9's Complement

When subtrahend is greater than the minuend

General Subtraction

$$
\begin{array}{r}
841 \\
-\ 983 \\
\hline
-\ 142
\end{array}
$$

Subtraction using 9's Complement

$$
\begin{array}{rl}
841 & \\
+016 & \leftarrow \text{(9's Complement)} \\
\hline
857 & \text{(No carry indicates -ve value)} \\
\downarrow & \\
-142 & \text{(9's Complement of result)}
\end{array}
$$

9’s Complement

- Subtract the following numbers using 9’s complement method

  1) 745.81 – 436.62

  2) 436.62 – 745.81

10’s Complement (examples)

When subtrahend is smaller than the minuend

General Subtraction	Subtraction using 10’s Complement
$$\begin{array}{r} 325 \\ -\,641 \\ \hline -\,316 \end{array}$$	$$\begin{array}{rl} 325 & \\ +\,359 & \leftarrow \text{(10's Complement of 641)} \\ \hline 684 & \leftarrow \text{(No carry indicate negative -ve value)} \\ \hline -\,316 & \leftarrow \text{(10's Complement of result)} \end{array}$$

10’s Complement

When subtrahend is smaller than the minuend

General Subtraction	Subtraction using 10’s Complement
$$\begin{array}{r} 821 \\ -\ 413 \\ \hline 408 \end{array}$$	$$\begin{array}{rl} 821 & \\ +\ 586 & \text{(10's Complement of 413)} \\ \hline 1\,408 & \text{(Ignore the carry)} \\ \hline 408 & \end{array}$$

10's Complement (examples)

- Subtract the using 10's complement method

2928.54 – 416.73

Step-1: 9's complement of 0416.73 is obtained by subtracting each digit from 9

$$
\begin{array}{rcccccc}
 & 9 & 9 & 9 & 9 & . & 9 & 9 \\
- & 0 & 4 & 1 & 6 & . & 7 & 3 \\
\hline
 & 9 & 5 & 8 & 3 & . & 2 & 6
\end{array}
$$

Step-2: Now add 1 to the 9's complement to obtain the 10's complement :
9583.26 + 0.01 = 9583.27

Step-3: Now Add this 10's complement of B to A

$$
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & & & 1 & \\
 & 2 & 9 & 2 & 8 & . & 5 & 4 \\
+ & 9 & 5 & 8 & 3 & . & 2 & 7 \\
\hline
1 & 2 & 5 & 1 & 1 & . & 8 & 1
\end{array}
$$

10's Complement (examples)

- Subtract the following numbers using 10's complement method

  2) 416.73 – 2928.54

Binary Codes

- Usually, Digital data is represented, stored and transmitted as groups of binary digits (bits). This group of bits is called as binary code.
- It represents numbers and letters of the alphabet, special characters and control functions.
- They are classified broadly as numeric and alphanumeric codes.
- Numeric codes are used to represent numbers
- Alphanumeric codes represent alphabetic letters and numerals.
- In these codes, a numeral is treated simply as another symbol rather than a number or numeric value.

Classification of Binary codes

- Weighted codes
- Non weighted codes
- Reflective codes
- Sequential codes
- Alphanumeric codes
- Error Detecting and Correcting codes

Weighted Binary codes.

- Weighted binary codes are those which obey the positional weight principle.
- Each position of a number represents a specific weight.
- For example:
  - 8421, 2421, 3321, 5211

Decimal	Binary Code	BCD (8421)	5421	2421
0	0000	0000	0000	0000
1	0001	0001	0001	0001
2	0010	0010	0010	0010
3	0011	0011	0011	0011
4	0100	0100	0100	0100
5	0101	0101	1000	1011
6	0110	0110	1001	1100
7	0111	0111	1010	1101
8	1000	1000	1011	1110
9	1001	1001	1100	1111

Non-weighted codes.

- Non-weighted codes are not assigned with any weight to each digit position.
- Each digit position within the number is not assigned any fixed value.
- For example:
  - Excess – 3
  - Gray code :
    - One bit different in consecutive numbers
    - It is also a Cyclic code

Decimal	Gray code	XS-3
0	0000	0011
1	0001	0100
2	0011	0101
3	0010	0110
4	0110	0111
5	0111	1000
6	0101	1001
7	0100	1010
8	1100	1011
9	1101	1100

Reflective Codes

- A code is said to be reflective when the code for 9 is the complement for the code for 0, 8 for 1,7 for 2,6 for 3 and 5 for 4.
- For example: 2421, 5211, Excess-3 are reflective codes
- 8421 is not reflective code.
- Reflectivity is desirable in a code when the nine's complement is needed like in nine's complement subtraction.

Decimal digit	*Excess-3*			
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Sequential Codes

- In sequential codes, each succeeding code is one binary number greater than its preceding code.
- This greatly aids mathematical manipulation of data. The 8421 and excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Alphanumeric codes

- Alphanumeric codes are designed to represent numbers as well as alphabetic characters.
- Some of these codes can also represent some symbols and instructions.
- For example:
  - ASCII, stands for American Standard Code for Information Interchange.
  - EBCDIC, Extended Binary Coded Decimal Interchange Code.
  - Hollerith Code.

ASCII code

- Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort.
  - 26 alphabets with capital and small letters
  - Numbers from 0 to 9
  - Punctuation marks and other symbols.
- It's a 7-bit code. It represents 2^7=128 symbols.
- ASCII code for N = $(4E)_H = (1001110)_2$

ASCII Code

		$b_6b_5b_4$ (column)							
$b_3b_2b_1b_0$	Row (hex)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	–	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

ASCII Code (Extended)

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

EBCDIC Code

- Extended binary coded decimal interchange **code** (**EBCDIC**) is an 8-bit binary **code** for numeric and alphanumeric characters.
- This encoding was developed in 1963 and 1964.
- It was developed and used by IBM.
- It is a coding representation in which symbols, letters and numbers are presented in binary language.
- EBCDIC was developed to enhance the existing capabilities of binary-coded decimal (BCD) code.

EBCDIC Code

ACK	Acknowledge
BEL	Bell
BS	Backspace
CAN	Cancel
DEL	Delete
ENQ	Enquiry
EOT	End of Transmission
ETX	End Text
FS	Form Separator
HT	Horizontal Tab
LF	Line Feed
NUL	Null
VT	Vertical Tab
STX	Start Text

2nd hex digit

1st hex digit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-									0
1	SOH	DC1	SOS				/		a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z	`	I	R	Z	9
A	SMM	CC	SM		¢ CENT	!		:								
B	VT	CU1	CU2	CU3		$	.	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	|	–	?	"								

Binary Coded Decimal (BCD)

- The BCD code is the 8,4,2,1 code.
- 8, 4, 2, and 1 are weights
- BCD is a weighted code
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, but only encodes the first ten values from 0 to 9.
- Example: 1001 (9) = 1000 (8) + 0001 (1)
- How many “invalid” code words are there?
- What are the “invalid” code words?

Warning: Conversion or Coding?

- Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a BINARY CODE.
- $13_{10} = 1101_2$ (This is conversion)
- 13 $\Leftrightarrow$ 0001 0011 (This is coding)

BCD Arithmetic

Given a BCD code, we use binary arithmetic to add the digits:

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**

Note that the result is MORE THAN 9, so must be represented by two digits!

To correct the digit, subtract 10 by adding 6 modulo 16.

8	**1000**	**Eight**
+5	**+0101**	**Plus 5**
13	**1101**	**is 13 (> 9)**
	+0110	**so add 6**
	carry = 1 0011	**leaving 3 + carry**
	0001 | 0011	**Final answer (two digits)**

If the digit sum is > 9, add one to the next significant digit

BCD Addition Example

Add 2905 and 1897 using BCD addition

BCD code for 2905 :	0010	1001	0000	0101
BCD code for 1897 :	0001	1000	1001	0111
Addition :	0011	10001	1001	1100
If Invalid BCD then add 6 :		0110		0110
Addition :	0011	10111	1001	10010
Remaining bits except carry :	0011	0111	1001	0010
Carry :	1		1	
Addition :	0100	0111	1010	0010
If Invalid BCD then add 6 :			0110	
Addition :	0100	0111	10000	0010
Remaining bits except carry :	0100	0111	0000	0010
Carry :		1		
Addition :	0100	1000	0000	0010
BCD value :	4	8	0	2

Codes for detecting and correcting errors

- An error in a digital system is the corruption of data from its correct value to some other value.
- i.e., a change of some bits from 0 to 1 or vice versa.
- During the processing or transmission of digital data a noise may change some bits from 0 to 1 or vice versa.
- A short duration noise can affect only a single bit causes a single-bit error.
- A long duration noise can affect two or more bits causes a multi-bit error.

Codes for detecting and correcting errors

- Error-detecting codes normally add extra information to the data.
- In general, error-detecting codes contains redundant code.
- That is a code that uses n-bit strings need not contain 2^n valid code words.
- An error-detecting code has the property that corrupting or garbling a code word will likely produce a bit string that is not a code word.
- Thus errors in a bit string can be detected by a simple rule - if it is not a code word it contains an error.

Parity check

- One of the most common ways to achieve error detection is by means of a parity bit.
- A parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd or even.
- If an odd parity is adopted, the P bit is chosen such that the total number of 1's is odd.

Information Bits	*Even-parity Code*	*Odd-parity Code*
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

Error-detecting Codes

p: parity bit; even parity used in given codes

Distance between codewords: no. of bits they differ in

Minimum distance of a code: smallest no. of bits in which any two code words differ

Minimum distance of given single error-detecting codes = 2

Decimal digit	*Even-parity BCD*					*2-out-of-5*				
	8	4	2	1	p	0	1	2	4	7
0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	1	1	1	0	0	0
2	0	0	1	0	1	1	0	1	0	0
3	0	0	1	1	0	0	1	1	0	0
4	0	1	0	0	1	1	0	0	1	0
5	0	1	0	1	0	0	1	0	1	0
6	0	1	1	0	0	0	0	1	1	0
7	0	1	1	1	1	1	0	0	0	1
8	1	0	0	0	1	0	1	0	0	1
9	1	0	0	1	0	0	0	1	0	1

Hamming Code

- Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver.
- It can correct single bit error.
- Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the following formula:

  $2^r \geq m + r + 1$

  where, r = redundant bit, m = data bit

- Suppose the number of data bits is 7, then the number of redundant bits can be calculated using: = $2^4 \geq 7 + 4 + 1$, Thus, the number of redundant bits= 4

Parity bits for 4 bit dataword

- Position of redundant/parity bits for four bit binary code

$\underline{4+3} = 7$
$m+r$ Code

P8

D7	D6	D5	P4	D3	P2	P1
1	0	1	1	0	1	1

2^0

- Assignment of Parity bits
  - P1: 1,3,5,7
  - P2: 2,3,6,7
  - P4: 4,5,6,7

P8:

2^0 — 1
2^1 — 2
2^2 — 4
2^3 — 8

P	P4	P2	P1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Parity bits for 7 bit dataword

- Position of redundant/parity bits for 7 bit data word

D11	D10	D9	P8	D7	D6	D5	P4	D3	P2	P1
1	0	1	0	1	0	0	1	1	1	0
7	6	5		4	3	2		1		

- Assignment of Parity bits
  - P1: 1,3,5,7,9,11 $\xrightarrow{2^0}$ 0 1 0 1 1 1
  - P2: 2,3,6,7,10,11 $\xrightarrow{2^1}$ 1 1 0 1 0 1
  - P4: 4,5,6,7 $\xrightarrow{2^2}$ 1 0 0 1
  - P8: 8,9,10,11 ✓ 2^3

1011001

$m = 7$
$r = 4$

10101001110

P	P8	P4	P2	P1
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

Hamming Code (Examples)

Even Parity

- Generate hamming code for given data word = 1011

$m = 4$

$r = 3$

D7	D6	D5	P4	D3	P2	P1
1	0	1	0	1	0	1

$P1 = 1, 3, 5, 7 = 1\ 1\ 1\ 1 =$

$P2 = 2, 3, 6, 7 = 0\ 1\ 0\ 1$

$P4 = 4, 5, 6, 7 = 0\ 1\ 0\ 1$

1010101 $\longrightarrow$

Hamming Code (Examples)

- Assume that an even parity hamming code is transmitted H = 1010101 and that 1000101 is received. The receiver does not know what was transmitted. Determine bit location where error has occurred using received code.
- Assignment of Parity bits
  - P1: 1,3,5,7
  - P2: 2,3,6,7
  - P4: 4,5,6,7

Hamming Code (Examples)

- Generate Hamming code for data-word = 11011010

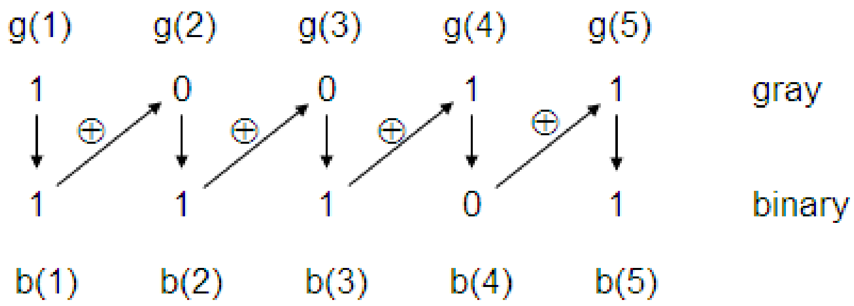
Binary to Gray Conversion

b(1)	b(2)	b(3)	b(4)	b(5)	
1	1	1	0	1	binary
1	0	0	1	1	gray
g(1)	g(2)	g(3)	g(4)	g(5)	

- Convert $(10111)_2$ to $(\qquad)_{Gray}$

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

Gray to Binary Conversion



- Convert $(11101)_{Gray}$ to $(\qquad)_2$

4 bit Gray Code	4 bit Binary Code
A B C D	B_4 B_3 B_2 B_1
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 1	0 0 1 0
0 0 1 0	0 0 1 1
0 1 1 0	0 1 0 0
0 1 1 1	0 1 0 1
0 1 0 1	0 1 1 0
0 1 0 0	0 1 1 1
1 1 0 0	1 0 0 0
1 1 0 1	1 0 0 1
1 1 1 1	1 0 1 0
1 1 1 0	1 0 1 1
1 0 1 0	1 1 0 0
1 0 1 1	1 1 0 1
1 0 0 1	1 1 1 0
1 0 0 0	1 1 1 1

Find the base?

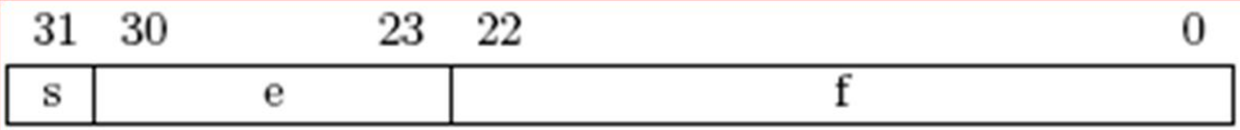
- $(193)_x = (623)_8$

- $(225)_x = (341)_8$

Complete the table

Decimal	Binary	BCD	Excess-3	Gray Code
5				
8				
14				

Floating Point representation



s sign bit - 0 = positive, 1 = negative
e biased exponent (8-bits) = true exponent + 7F (127 decimal). The values 00 and FF have special meaning (see text).
f fraction - the first 23-bits after the 1. in the significand.

- Example: $3.92 \times 10^2 \rightarrow (s)1.M \times 2^E$
- $3.92 \times 10^2 = 392_{10} = 110001000_2$ → Denormalized
- 1.10001000×2^8 → Normalized
- Here s=0, M=10001000, E' =E+127 = 8+127=135_{10}=10000111
- Binary →

0	10000111	10001000000000000000000

- Hex → 43C40000

How would 23.85 be stored?

- First, it is positive so the sign bit is 0.
- Next, the true exponent is 4, so the biased exponent is 7F+4 = 83_{16}.
- Finally, the fraction is 01111101100110011001100 (remember the leading one is hidden).

$$\boxed{\underline{0}\,100\;0001\;1\,\underline{011\;1110\;1100\;1100\;1100\;1100_2} = 41\mathrm{BECCCC}_{16}}$$

- -23.85 be represented? Just change the sign bit: C1 BE CC CD. Do not take the two's complement!

Thank You

Any Questions