Digital Logic Circuits
Fundamentals II

Ch.2 Numbers, Operations and Codes

Outline

- Decimal Numbers, Binary Numbers
- Decimal-to-Binary Conversion
- Binary Arithmetic
- 1’s and 2’s Complements of Binary Numbers
- Signed Numbers, Hexadecimal, Octal Numbers
- Binary Coded Decimal (BCD)
- Digital Codes
- Error Detection & Correction Codes

Decimal Numbers

- The decimal number system has ten digits (0 through 9).
- The position of each digit in a weighted number system is assigned a weight based on the **base** (or **radix)** of the system.
- The base of decimal numbers is 10, because only ten symbols (0 ~ 9) are used to represent any number.
- The column weights of decimal numbers are powers of 10 that increase from right to left beginning with $10^0 = 1$:

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

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

- For fractional decimal numbers, the column weights are negative powers of 10 that decrease from left to right:
- Decimal numbers can be expressed as the sum of the products of each digit times the column value for that digit.

Ex: 9240 → $(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$

Binary Numbers

- For digital systems, the binary number system is used.
- Binary has a base of 2 and uses the digits 0 and 1 to represent quantities.

- The column weights of binary numbers are powers of 2 that increase from right to left beginning with $2^0 = 1$:

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

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

- For fractional binary numbers, the column weights are negative powers of 2 that decrease from left to right:

Binary to Decimal Conversion

- 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.

  - Ex) Convert the binary number 100101.01 to decimal.

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

2^5 2^4 2^3 2^2 2^1 2^0. 2^{-1} 2^{-2}

Binary Point

32	16	8	4	2	1	.	½	¼
1	0	0	1	0	1	.	0	1

32 +4 +1 +¼ = 37¼

Decimal to Binary Conversion

- 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.
  - Ex) Convert the decimal number 49 to binary.
    - **Sum-of-Weights Method**: Find the binary weights that add up to the decimal number.

2^6 2^5 2^4 2^3 2^2 2^1 2^0.

64 32 16 8 4 2 1

1 1 0 0 0 1 ← Binary number for decimal 49 = 32 + 16 + 1

MSB LSB

    - **Repeated Division by-2 Method**: Divide the decimal number by 2 until the quotient is 0. Remainders form the binary number.

1 1 0 0 0 1 ← remainder

0 1 3 6 12 24 49 2

Continue until the last quotient is 0 — Quotient — Decimal number — base

Decimal Fractions to Binary Conversion

- 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.
  - Ex) Convert the decimal fraction 0.188 to binary by repeatedly multiplying the fractional results by 2.
    - **Repeated Multiplication by-2 Method**:

0.188 x 2 = 0.376	carry = 0
0.376 x 2 = 0.752	carry = 0
0.752 x 2 = 1.504	carry = 1
0.504 x 2 = 1.008	carry = 1
0.008 x 2 = 0.016	carry = 0

MSB

To increase accuracy, continue until the fractional part is all zeros.

Answer = .00110 (for five significant digits)

MSB LSB

- **Sum-of-Weights Method**:

Binary Arithmetic

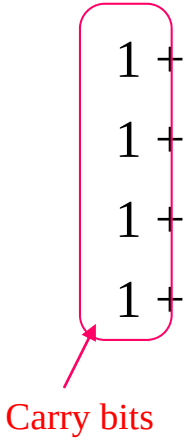
- **Binary Addition**

  - The rules for binary addition are as follows.

0 + 0 = 0	Sum = 0, carry = 0
0 + 1 = 1	Sum = 1, carry = 0
1 + 0 = 1	Sum = 1, carry = 0
1 + 1 = 10	Sum = 0, carry = 1

  - When there is a carry of 1, three bits are being added as follows.

1 + 0 + 0 = 01	Sum = 1, carry = 0
1 + 0 + 1 = 10	Sum = 0, carry = 1
1 + 1 + 0 = 10	Sum = 0, carry = 1
1 + 1 + 1 = 11	Sum = 1, carry = 1



• Ex) Add the binary numbers 00111 and 10101 and show the equivalent decimal addition.

```
  0 1 1 1
   00111      7
   10101     21
  -------   ----
   11100   = 28
```

• Binary Subtraction

- The rules for binary subtraction are as follows.

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

- Ex) Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

```
  1 1 1
  1 0 1 0 1     21
  0 0 1 1 1      7
-------------   ---
  0 1 1 1 0   = 14
```

- **Binary Multiplication**

  - The rules for multiplying bits are as follows.

    0 x 0 = 0
    0 x 1 = 0
    1 x 0 = 0
    1 x 1 = 1

- **Binary Division**

  110 / 11 = 10
  110 / 10 = 11

1’s Complement of a Binary Number

- **1’s Complement**

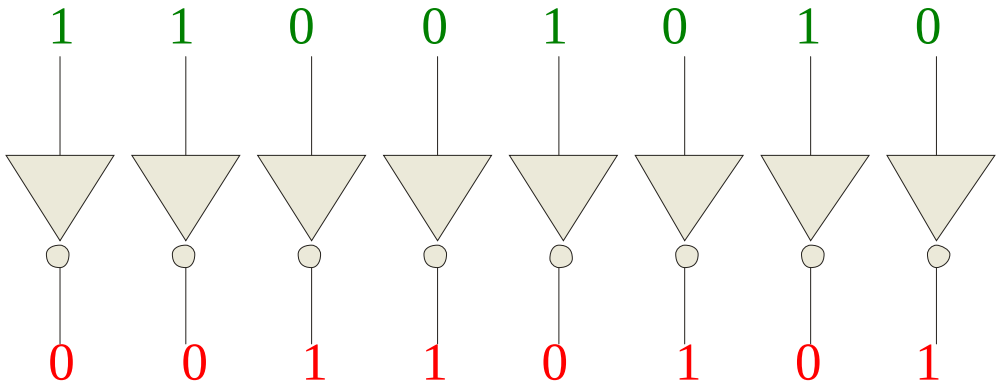
: The 1’s complement of a binary number is just the inverse of the digits.

To form the 1’s complement, change all 0’s to 1’s and all 1’s to 0’s.

- Ex) The 1’s complement of **11001010** is

00110101

- In digital circuits, the 1’s complement is formed by using inverters (NOT Gate):



- How to represent **negative** numbers in digital systems?

  →1's complement , 2's complement, and sign-magnitude can be used to represent negative numbers.

- The ones' complement form of a negative binary number is the bitwise NOT applied to it

- The range of signed numbers using ones' complement is represented by $-(2^{N-1}-1)$ to $(2^{N-1}-1)$ and +/−0.
- A conventional 8-bit byte is -127_{10} to $+127_{10}$ with zero being either 00000000 (+0) or 11111111 (−0).
- Ones' complement has two representations of 0:

  → 00000000 (+0) and 11111111 (−0).

8-bit 1's complement		
Binary value	1' complement interpretation	Unsigned interpretation
00000000	**0**	0
00000001	**1**	1
…	**…**	…
01111101	**125**	125
01111110	**126**	126
01111111	**127**	127
10000000	**−127**	128
10000001	**−126**	129
10000010	**−125**	130
…	**…**	…
11111110	**−1**	254
11111111	**−0**	255

2's Complement of a Binary Number

- **2's Complement**

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

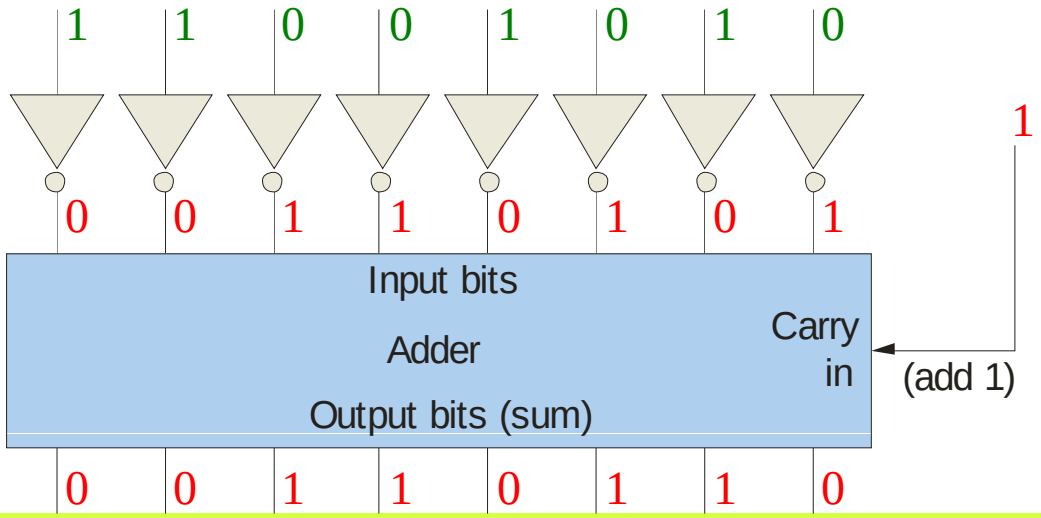
- Ex) Find the 2's complement of **11001010**.

Recall that the 1's complement of **11001010** is

00110101 (1's complement)

To form the 2's complement, add 1→ +1

00110110 (2's complement)



Computers form the 2's complement by inverting the bits and adding 1.

• The problems of multiple representations of 0 (+0 & -0) can be solved by using **2's complement**.

•In 2's-complement, there is only one zero (00000000).

•Negating a number (whether negative or positive) is done by inverting all the bits and then adding 1 to that result.

• The range of signed numbers using ones' complement is represented by $-(2^{N-1})$ to $(2^{N-1}-1)$ and 0.

• A conventional 8-bit byte is -128_{10} to $+127_{10}$ with zero 00000000 (0).

• Twos' complement has one representations of 0:

→ 00000000 (0)

8-bit 2's complement

Binary value	2's complement interpretation	Unsigned interpretation
00000000	**0**	0
00000001	**1**	1
...	**...**	...
01111110	**126**	126
01111111	**127**	127
10000000	**−128**	128
10000001	**−127**	129
10000010	**−126**	130
...	**...**	...
11111110	**−2**	254
11111111	**−1**	255

Signed Binary Numbers

- There are three forms in which signed integer (whole) numbers can be represented in binary:
  - **Sign-Magnitude**
  - **1's Complement**
  - **2's Complement** (→ most important)

1) Sign-Magnitude Form: When a signed binary number is represented in sign-magnitude, the left-most bit is the sign bit and the remaining bits are the magnitude bits.

- Ex) Positive number +25 is written using 8-bits as 00011001.

  - Sign Bit : The left-most bit in a signed binary number is the sign bit. It tells the number is positive or negative.

    0 sign bit indicates a positive number. 1 sign bit indicates a negative number.

- Ex) Negative number -25 is written using 8-bits as 10011001.

2) 1's Complement Form:

- Positive Numbers : represented the same way as the positive sign-magnitude numbers.
- Negative Numbers: represented as the1's complement of the corresponding positive numbers.

  - Ex) Negative number -25 is expressed as the 1's complement of +25 (00011001).

1's complement form of -25 ⟶ **11100110**

3) 2's Complement Form:

- Positive Numbers : represented the same way as the positive sign-magnitude numbers.
- Negative Numbers: represented as the 2's complement of the corresponding positive numbers.

- Ex) Negative number -25 is expressed as the 2's complement of +25 (00011001).

11100110
1

2's complement form of -25 → **11100111**

- **Computers use the 2's complement for negative integer numbers in all arithmetic operations.**

→ **Subtracting of a number = Adding the 2's complement of the number.**

Range of Signed Integer Numbers

- The number of different combinations of n bits = 2^n
- For 2's complement signed numbers, the range of values for n-bit numbers is as follows.
  - Range = $-(2^{n-1}) \sim (2^{n-1} - 1)$.

  Ex) n=4-bit → -8 to +7

Floating-Point Numbers

- The floating-point number (= real number) system (based on scientific notation) is capable of representing very large and very small numbers without an increase in the number of bits. It can also represent numbers that have both integer and fractional components.

수 = (가수) x (밑수)^{지수}

-3 = (-) 3 x 10^{-1}

sign, exponent, Fraction (mantissa)

- A floating-point number consists of three parts.
  - **Sign (S)**
  - **Exponent (E)** : represents the number of places that the decimal or binary point is to be moved.

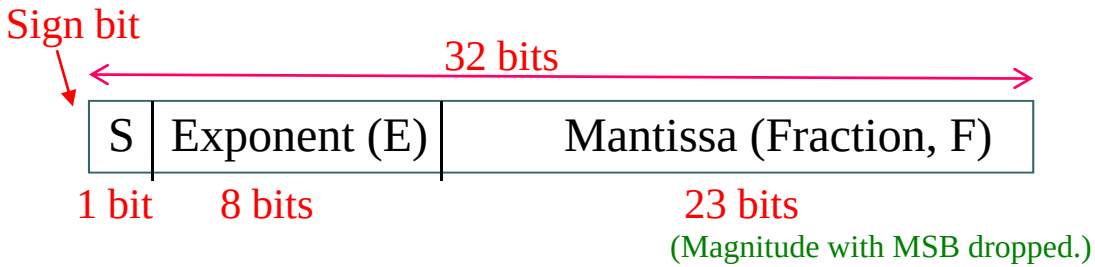
= (-) 30 x 10^{-2}

  - **Mantissa (Fraction, F)** : represents the magnitude of the number

- For binary floating-point numbers, the format is defined by ANSI/IEEE standard in three forms:
  - Single-precision : 32 bits
  - Double-precision : 64 bits
  - Extended-precision : 80 bits

Single-Precision Floating-Point Binary Numbers

- Format of a 32-bit single-precision number



Exponent (E) : The 8 bits represent a biased exponent, allowing a range of actual exponent values from -127 to +128.

How to get this E? → adding 127 to the actual exponent.

- Ex) Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

$3.248 \times 10^4 = 32480_{10} = 111\ 1110\ 1110\ 0000_2$

In scientific notation $= 1.11\ 1110\ 1110\ 0000 \times 2^{14}$

S = 0 because the number is positive.

$E = 14 + 127 = 141_{10} = 1000\ 1101_2$.

F = It is the next 23 bits after the first 1 is dropped.

The complete floating point number is →

0	1000 1101	11 1110 1110 0000 0000 0000 0

Arithmetic Operations with Signed Numbers

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

- **Addition**

; Just add the two signed numbers in 2's complement form.
Discard any final carries. The result is in signed form.

- Ex) Add two 8-bit signed numbers.

30 + 15	14 + (-17)	(-1) + (-8)
00011110 = +30	00001110 = +14	11111111 = –1
00001111 = +15	11101111 = –17	11111000 = –8
00101101 = +45	11111101 = –3	1 11110111 = –9

Discard carry

- **Addition of a + number and a larger – number or two negative numbers yields a negative number in 2's complement form**

• Overflow : Note that if the number of bits required for the answer is exceeded, overflow will occur. This occurs only if both numbers have the same sign. The overflow will be indicated by an incorrect sign bit.

• Ex)

- **Subtraction**

; Subtracting of a number = Adding the 2's complement of the number. Discard any final carries. The result is in signed form.

- Ex) Repeat the examples done previously, but subtract:

00011110 (+30)	00001110 (+14)	11111111 (–1)
– 00001111 –(+15)	– 11101111 –(–17)	– 11111000 –(–8)

→ 2's complement subtrahend and add:

00011110 = +30	00001110 = +14	11111111 = –1
11110001 = –15	00010001 = +17	00001000 = +8
~~1~~ 00001111 = +15	00011111 = +31	~~1~~ 00000111 = +7

Discard carry

Discard carry

- **To subtract two signed numbers, take the 2's complement of the subtrahend and add. Discard any final carry bit.**

- **Multiplication**

  Multiplication is equivalent to adding a number to itself a number of times equal to the multiplier

  - Ex) 5 (multiplicand) x 3 (multiplier) = 15 (product)

    → 0101 x 011

```
  0101    (1'st time)
 +0101    (2'nd time)
------
  1010
 +0101    (3'rd time)
------
  1111
```

```
  0101
   011
------
  0101
 0101
------
  1111
```

- **Division**

$$\frac{\text{Dividend}}{\text{Divisor}} = \text{quotient}$$

  - Ex) 15 / 5 = 3

	15	
–	5	1’st subtraction
	10	
–	5	2’nd subtraction
	5	
–	5	3’rd subtraction
	0	Zero remainder

→ The devisor was subtracted from the dividend 3 times. Therefore the quotient is 3.

Hexadecimal Numbers

- Hexadecimal uses 16 characters to represent numbers: the numbers 0 through 9 and the alphabetic characters A through F.

- Large binary number can easily be converted to hexadecimal by grouping 4 bits at a time and writing the equivalent hexadecimal character.

- Ex) Express $1001\ 0110\ 0000\ 1110_2$ in hexadecimal:

  Group the binary number by 4-bits starting from the right.

  $1001\ 0110\ 0000\ 1110_2$
  9 6 0 E
  → Thus the answer is 960E

Decimal	**Hexadecimal**	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

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

Column weights $\{$ 16^3 16^2 16^1 16^0 .
4096 256 16 1 .

- Ex) Express $1A2F_{16}$ in decimal.

Start by writing the column weights:

4096 256 16 1

1 A 2 F_{16}

$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$

Octal Numbers

- Octal uses 8 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.

- Ex) Express $1\ 001\ 011\ 000\ 001\ 110_2$ in octal:

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

  $1\ 001\ 011\ 000\ 001\ 110_2$ (binary)
  1 1 3 0 1 6 (octal)
  Thus, 113016_8

Decimal	**Octal**	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	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

• 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 .

• Ex) Express 3702_8 in decimal.

Start by writing the column weights:

512 64 8 1

3 7 0 2_8

$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$

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 8421 code is a type of BCD code.
- 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
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

- You can think of BCD in terms of column weights in groups of 4 bits. For an 8-bit BCD number, the column weights are: 80 40 20 10 8 4 2 1.

- Ex) What are the column weights for the BCD number 1000 0011 0101 1001?

Note that you could add the column weights where there is a 1 to obtain the decimal number. For this case:

$$8000 + 200 + 100 + 40 + 10 + 8 + 1 = 8359_{10}$$

Gray Code

- Gray code is unweighted and is not an arithmetic code.
- Gray code exhibits only a single bit change between one code word and the next in a sequence.
- The single bit change characteristic of the Gray code minimizes the change for error.

→ Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

Decimal	Binary	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

ASCII Code

- **ASCII** (American Standard Code for Information Interchange) is a universally accepted alphanumeric code used in most computers and other electronic equipment.
- Most computer keyboards are standardized with the ASCII.
- ASCII has 128 characters (numbers, letters) and symbols represented by a 7-bit binary code.
- The first 32 characters are control characters and are used for control purpose.

- **Extended ASCII**

; 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.

- ASCII example:

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	' '
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111 Binary	077 Oct	63 Dec	3F Hex	? Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F