Binary Arithmetic

2. Binary Arithmetic

2.1 Complement

2.2 Un-signed binary number

2.3 Signed binary number

2.4 Binary Arithmetic

- Addition
- Subtraction
- Multiplication
- Division

2.1 Complement

- 2 types of complement for any radix-r number system
  - For a n-digit number a
    - Its (r–1)'s complement number: $r^n - 1 - a$
    - Its r's complement number: $r^n - a$

Note: All systems are implemented with fixed digit (bit) length.

The length of complement is the same as the number itself.

- For decimal number $r = 10$
  - 9's complement of a : $b = 10^n - 1 - a$

Examples:

$a = 43$, $b = 100 - 1 - 43 = 56$ $(n = 2)$

$a = 7629$, $b = 10000 - 1 - 7629 = 2370$ $(n = 4)$

  - 10's complement of a : $c = 10^n - a$

Examples:

$a = 43$, $c = 100 - 43 = 57$ $(n = 2)$

$a = 7629$, $c = 10000 - 7629 = 2371$ $(n = 4)$

- For binary number r = 2
  - 1's complement of a : $b = 2^n - 1 - a$

Examples:

a = 1001

b = 10000 – 1 – 1001

= 0110 (n = 4, same bit length)

a = 1100101

b = 10000000 – 1 – 1100101

= 0011010 (n = 7, same bit length)

  - OR turn "1" to "0" and "0" to "1"

- For binary number r = 2

  - 2's complement of a : c = $2^n - a$
  - OR Its 1's complement plus "1", i.e. b **+ 1**

Examples:

a = 1001

c = 0110 + 1

= 0111 (*n* = 4, same bit length)

a = 1100101

c = 0011010 + 1

= 0011011 (*n* = 7, same bit length)

- A fast way to obtain the 2's complement

i. Starting from the LSB,

ii. Copy all digits including the first "1" encountered,

iii. Then complement the rest of the bits.

$\boldsymbol{a}$ = 1001, $\boldsymbol{c}$ = 0111

$\boldsymbol{a}$ = 1100101, $\boldsymbol{c}$ = 0011011

$\boldsymbol{a}$ = 101110010100 $\boldsymbol{c}$ = 010001101100

- **Property of r's complement of a n-digit number a.**

10's complement: $c = 10^n - a$

Sum of a and c: $a + c = a + 10^n - a = 10^n$

Examples:

$a = 43$, $c = 10^2 - 43 = 57$, $a + c = 100$ **(1=10²,n=2)**

$a = 7629$, $c = 10^2 - 7629 = 2371$, $a + c = 10000$ **(1=10⁴,n=4)**

For hardware system, c is also a n-digit number, and so as the result of $a + c$, therefore 1 is discarded from the result.

<u>**NOTE: 1 = 10^n**</u>, discard 1 $\Rightarrow a + c - 10^n \Rightarrow a + c = 0$

$n = 2$, $a + c = 00$ (*for 2-digit result*)

$n = 3$, $a + c = 000$ (*for 3-digit result*)

$n = 4$, $a + c = 0000$ (*for 4-digit result*)

$\Rightarrow$ "10's of a" $= c \equiv -a$ *for n-digit input and output system*

For a binary number a:

2's complement: $c = 2^n - a$

Sum of a and c: $a + c = a + 2^n - a = 2^n$

1001,	1100101	101110010100
+ 0111	+ 0011011	+ 010001101100
10000	10000000	1000000000000

NOTE: 1 = 2^n, discard 1 $\Rightarrow a + c - 2^n \Rightarrow a + c = 0$

$n = 2$, $a + c = 00$ (*for 2-bit result*)

$n = 3$, $a + c = 000$ (*for 3-bit result*)

$n = 4$, $a + c = 0000$ (*for 4-bit result*)

$\Rightarrow$ "2's of a" $= c \equiv -a$ *for n-bit input and output system*

Formal proof:

- For a n-bit binary number $\boldsymbol{a}$,
  it's 2's complement is $\boldsymbol{b} = 2^n - \boldsymbol{a}$
- $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{a} + 2^n - \boldsymbol{a} = 2^n$
- 2^n is the weight of the $(n+1)^{th}$ bit of the sum.
- If the $(n+1)^{th}$ bit is discarded for a n-digit system,
  then $\boldsymbol{a} + \boldsymbol{b} = \mathbf{0}$ and $\boldsymbol{b}$ can be treated as $-\boldsymbol{a}$.

2.2 Un-signed binary number

- For a ***n***-bit number, its value range is:
  - 0, 1, 2, … ,$2^{n} – 1$

$$a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \ldots + a_2 r^2 + a_1 r + a_0$$

$n = 2 \Rightarrow$ **0,1,2,3**

$n = 5 \Rightarrow$ **0,1,2,…, 30,31**

$n = 8 \Rightarrow$ **0,1,2,…, 254,255**

$n = 10 \Rightarrow$ **0,1,2,…, 1022,1023**

2.3 Signed number

2.3.1 signed magnitude representation

- The MSB is a sign bit (which does not carry any value)
  - 0 means positive "+"
  - 1 means negative "–"

S	Magnitude

Sign-magnitude format

Example:

4-bit system (+5 = 0101; –5= 1101)

Last 3 bits used for magnitude

- For a n-bit number, range value is:

$-(2^{n-1}-1), -2^{n-1}+2, \ldots, -2, -1, 0, +1, +2, \ldots, +2^{n-1}-2, +2^{n-1}-1$

- Problems of Sign-Magnitude representation
- Digital circuit cannot differentiate sign bit from number bit.

e.g. compute the sum of +5 and -3 using 4-bit system

+5 = 0101

–3 = 1011

Result = 10000 – **WRONG ANSWER!!**

Problem : The sign bit do not carry value, adding two sign bits **DOES NOT** make sense !!!

- Cannot perform arithmetic operation with signed-magnitude representation !!!
- NOT used in integer number system !!!
- Only used in floating-point system $(-1)^S$!!!

2.3.2 Signed number – 2’s complement format

- MSB is a sign bit with weight
  - 0 means positive “+” with weight of 0
  - 1 means negative “–” with weight of 2^{n-1}

S	Magnitude

Sign-magnitude format

- value: $S \times (-2^{n-1}) +$ magnitude

i.e.,

sign bit = 0: value of 0xx…x = 0 + xx…x

sign bit = 1: value of 1xx…x = -2^{n-1} + xx…x

Example:

$01001 \equiv 0+9 = +9$

$11110111 \equiv -2^{8-1}+119 = -9$

- Range value of 2's complement number

$$-2^{n-1}, -2^{n-1}+2, \ldots, -2, -1, 0, +1, +2, \ldots, +2^{n-1}-2, +2^{n-1}-1$$

$$-8 \le x[4] \le +7$$

$$-128 \le x[8] \le +127$$

$$-32768 \le x[16] \le +32767$$

- Sign extension – extend the bit length of a 2's complement number
- Repeatedly Duplicate the sign bit for the required bit length, i.e.

1001 ⇒ 11111001

111001010 ⇒ 1111111111111111001010

- Important for having the same bit length to perform arithmetic operation using digital circuit.

- Different number representation (3-bit example)

Unsigned	(Binary)	2's complement	(Binary)	Invert MSB
7	111	3	011	111
6	110	2	010	110
5	101	1	001	101
4	100	0	000	100
3	011	-1	111	011
2	010	-2	110	010
1	001	-3	101	001
0	000	-4	100	000

- Complementing the MSB of a 2's complement no turns it to unsigned representation, or vice versa.

- 2’s complement of unsigned number (3-bit example)

Unsigned	Binary	2’s complement (Binary)	Value	Sum
0	000	000	0	0
1	001	111	7	0
2	010	110	6	0
3	011	101	5	0
4	100	100	4	0
5	101	011	3	0
6	110	010	2	0
7	111	001	1	0
				3-bit result

- 2's complement of signed number (3-bit example)

signed	(Binary)	2's complement (Binary)	Value	Sum
3	011	101	-3	0
2	010	110	-2	0
1	001	111	-1	0
0	000	000	0	0
-1	111	001	1	0
-2	110	010	2	0
-3	101	011	3	0
-4	100	100	-4	0
				3-bit result

Signed number is represented in 2's complement form

Signed number also has its 2's complement

2.4 Binary Arithmetic

2.4.1 Binary number addition – unsigned number

9 + 5 = 14	12 + 7 = 19	
1 0 0 1	1 1 0 0	
+ 0 1 0 1	+ 0 1 1 1	
1 1 1 0	1 0 0 1 1	carry & sum

- For a *n*-bit number, its value range is:
  - 0, 1, 2, … ,$2^{n-1} – 1$, ⇒ 4-bit : 0,1,2, … , 15
- If result exceed *n*-bit, carry will be generated.
- For *n*-bit system, result is also interpreted with *n*-bit. Hence the result is **wrong** if **carry is produced**.

2.4.2 Binary number addition – 2's complement number

3 + 4 = 7

```
  0 0 1 1
+ 0 1 0 0
---------
  0 1 1 1
```

−7 + 3 = −4

```
  1 0 0 1
+ 0 0 1 1
---------
  1 1 0 0
```

1 = -8, 100 = 4

-8 + 4 = -4

−3 + (−4) = −7

```
  1 1 0 1
+ 1 1 0 0
---------
1 1 0 0 1
```

1 = -8, 001 = 1

-8 + 1 = -7

Discard the carry for 2's complement number addition.

2.4.3 Overflow – only for 2's complement number

- If the sum exceeds the bit length value range, overflow occurs.
- (+ve no) + (−ve no) $\Rightarrow$ overflow ***never*** occur
- (+ve no) + (+ve no) ***or*** (−ve no) + (−ve no) $\Rightarrow$ overflow ***may*** occur

```
   00110      + 6
+  00110      + 6
   01100      +12
```

same sign

```
   11010      - 6
+  11010      - 6
  110100      -12
```

same sign

overflow does not occur

```
   00110      + 6
+  01101      +13
  010011      +19
```

diff. sign

```
   11010      - 6
+  10011      -13
  101101      -19
```

diff. sign

overflow occurs

2.4.4 Binary number subtraction – Unsigned number

9 – 5 = 4

```
  1 0 0 1
- 0 1 0 1
---------
  0 1 0 0
```

6 – 11 = –5

```
    0 1 1 0
-   1 0 1 1
-----------
  1 0 0 1 1
```

Wrong result (borrower) !!

If the result is negative, use 2's complement to help

```
  6        0 1 1 0
-11      + 0 1 0 1   ← 2's complement of 1011
---      ---------
- 5        1 0 1 1
```

- Answer is the negative of the 2's complement of the sum
  i.e. 2's complement of 1011 ⇒ Answer = –0101 = –5

2.4.5 Binary number subtraction – 2's complement number

- Addition by means of using the 2's complement the subtrahend.

2 − 6 = −4	2 + (−6) = −4	
0 0 1 0	0 0 1 0	
− 0 1 1 0	+ 1 0 1 0	← 2's complement of 0110
1 1 0 0	1 1 0 0	

− 6 − 5 = −11	−6 + (−5) = −11	
1 0 1 0	1 0 1 0	
− 0 1 0 1	+ 1 0 1 1	← 2's complement of 0110
1 0 1 0 1	1 0 1 0 1	
Exceed the range value !!!	Result overflow !!!	

2.4.6 BCD number addition

- If the sum of two BCD digit addition greater than 9, add 0110 (6) to the digit sum.

```
    0 1 0 1   0 1 1 0          5 6
  + 0 0 1 1   0 1 1 1        +   3 7
  -----------------------    -------
    1 0 0 0   1 1 0 1          9 3
  + 0 0 0 0   0 1 1 0
  -----------------------
    1 0 0 1   0 0 1 1
       9         3
```

2.4.7 BCD number subtraction

- If the difference of two BCD digit subtraction greater than 9, subtract 0110 (6) from the digit difference.

```
    0 1 1 0   0 1 0 1          6 5
  - 0 0 0 1   1 0 0 1        -   1 9
  ---------------------      -------
    0 1 0 0   1 1 0 0          4 6
  - 0 0 0 0   0 1 1 0
  ---------------------
    0 1 0 0   0 1 1 0
       4         6
```

2.4.8 Binary multiplication

- Two one bit multiplication

A	B	A × B
0	0	0
0	1	0
1	0	0
1	1	1

- Two n-bit multiplication

```
        1 1 1 0
      × 1 0 1 1
      ---------
        1 1 1 0
      1 1 1 0
    0 0 0 0
+ 1 1 1 0
---------------
1 0 0 1 1 0 1 0
```

Results is in 2n-bit !!

2.4.9 Binary division

$$\frac{1000101001}{1101}$$

```
                1 0 1 0 1 0     ←  quotient
        ___________________
1 1 0 1 | 1 0 0 0 1 0 1 0 0 1
          1 1 0 1
         _________
            1 0 0 0 1
            1 1 0 1
            _________
              1 0 0 0 0
                1 1 0 1
                _______
                  1 1 1     ←  remainder
```

Answer with fractional part:

```
                  1 0 1 0 1 0.1 0 0 0 1 0 0 1 1 . . . .
1 1 0 1 | 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 . . . .
            1 1 0 1
              1 0 0 0 1
                1 1 0 1
                  1 0 0 0 0
                    1 1 0 1
                        1 1 1 0
                        1 1 0 1
                              1 0 0 0 0
                                1 1 0 1
                                    1 1 0 0 0
                                      1 1 0 1
                                      1 0 1 1 0
                                        1 1 0 1
                                        1 0 0 1 0 . . . .
                                            . . . . . . . .
```

Fractional part depends on precision requirement.

2.4.10 Floating-point addition

Example:

- Find the sum of 12_{10} and 1.25_{10} using the 32-bit floating-point format.

$12_{10} = 0.1100 \times 2^{4}$, $1.25_{10} = 0.101 \times 2^{1} = 0.000101 \times 2^{4}$.

	0	10000011	11000000000000000000000
+	0	10000011	00010100000000000000000
	0	10000011	11010100000000000000000

Ans = 0.110101×2^{4}

The exponents for the two number must be equal for operation.