Chapter 4

Binary Data Representation and Binary Arithmetic

4.1 Binary Data Representation

Bit: smallest unit of information

yes / no, on / off, L / 0, 1 / 0, 5V / 0V

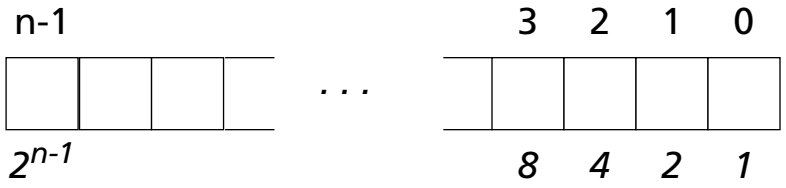
Byte: group of 8 bits

--> 2^8 = 256 different **states**

Word: the number of bits (word length) which can be processed by a computer in a single step (e.g., 32 or 64)

--> machine dependent

Representation:



- The word size in any given computer is fixed.

  Example: 16-bit word

  $\Rightarrow$ every word (memory location) can hold a 16-bit pattern, with each bit either 0 or 1.

How many distinct patterns are there in a 16-bit word?

- Each bit has 2 possible values: 0 or 1

  $\Rightarrow$ 1 bit has 2 distinct patterns: 0, 1

- With 2 bits, each position has 2 possible values: 00, 01, 10, 11

  $\Rightarrow 2^2 = 4$ distinct bit patterns

- With 3 bits, again each position can be either 0 or 1:

000	100
001	101
010	110
011	111

$\Rightarrow 2^3 = 8$ distinct bit patterns

- In general, for *n* bits (a word of length *n*) we have 2^n distinct bit patterns.

- NOTE: **What these bit patterns mean depends entirely on the context in which the patterns are used.**

4.2 Important Number Systems for Computers

4.2.1 Number System Basics

- Number systems, as we use them, consist of
  - a basic set Z of **digits** (or letters); example: $Z = \{0, 1, 2, \ldots, 9\}$
  - a **base** $B = |Z|$ (how many digits); example: $B = |Z| = 10$
- A **number** is a linear sequence of digits.
- The **value of a digit** at a specific position depends on its "assigned *meaning*" and on its position.
- The **value of a number** N is the sum of these values.

Number: $N_B = d_{n-1} d_{n-2} \ldots d_1 d_0$ with word length n

Value: $N_B = \sum_{i=0}^{n-1} d_i \cdot B^i = d_{n-1} B^{n-1} + d_{n-2} B^{n-2} + \ldots + d_1 B^1 + d_0 B^0$

4.2.2 Useful Number Systems for Computers

Name	Base	Digits
dual binary	2	0, 1
octal	8	0, 1, 2, 3, 4, 5, 6, 7
decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexadecimal sedecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

4.2.3 Decimal Number System

$Z = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}; B = 10$

- Each position to the **left** of a digit **increases** by a power of 10.
- Each position to the **right** of a digit **decreases** by a power of 10.

Example:

$$
\begin{aligned}
47692_{10} &= 2 \cdot 10^0 + \\
&\quad 9 \cdot 10^1 + \\
&\quad 6 \cdot 10^2 + \\
&\quad 7 \cdot 10^3 + \\
&\quad 4 \cdot 10^4 \\
&= 2 + 90 + 600 + 7000 + 40000
\end{aligned}
$$

4.2.4 Binary Number System

$Z = \{0, 1\}; B = 2$

- Each position to the **left** of a digit **increases** by a power of 2.
- Each position to the **right** of a digit **decreases** by a power of 2.

Example:

$$10111001_2 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 = 1 + 8 + 16 + 32 + 128 = 185_{10}$$

Counting in Binary

Decimal	Dual	Decimal	Dual
0	00 000	16	10 000
1	00 001	17	10 001
2	00 010	18	10 010
3	00 011	19	10 011
4	00 100	20	10 100
5	00 101	21	10 101
6	00 110	22	10 110
7	00 111	23	10 111
8	01 000	24	11 000
9	01 001	25	11 001
10	01 010	26	11 010
11	01 011	27	11 011
12	01 100	28	11 100
13	01 101	29	11 101
14	01 110	30	11 110
15	01 111	31	11 111

4.2.5 Octal Number System

$Z = \{0, 1, 2, 3, 4, 5, 6, 7\}; B = 8$

- Each position to the **left** of a digit **increases** by a power of 8.
- Each position to the **right** of a digit **decreases** by a power of 8.

Example:

$$
\begin{aligned}
12403_8 &= 3 \cdot 8^0 + \\
&\quad 0 \cdot 8^1 + \\
&\quad 4 \cdot 8^2 + \\
&\quad 2 \cdot 8^3 + \\
&\quad 1 \cdot 8^4 \\
&= 3 \cdot 1 + 0 \cdot 8 + 4 \cdot 64 + 2 \cdot 512 + 1 \cdot 4096 \\
&= 5379_{10}
\end{aligned}
$$

Counting in Octal

Decimal	Octal	Decimal	Octal
0	0 0	16	2 0
1	0 1	17	2 1
2	0 2	18	2 2
3	0 3	19	2 3
4	0 4	20	2 4
5	0 5	21	2 5
6	0 6	22	2 6
7	0 7	23	2 7
8	1 0	24	3 0
9	1 1	25	3 1
10	1 2	26	3 2
11	1 3	27	3 3
12	1 4	28	3 4
13	1 5	29	3 5
14	1 6	30	3 6
15	1 7	31	3 7

4.2.6 Hexadecimal Number System

$Z = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}; B = 16$

Each position to the **left** of a digit **increases** by a power of 16.

Each position to the **right** of a digit **decreases** by a power of 16.

Example:

$$
\begin{aligned}
FB40A_{16} &= 10 \cdot 16^0 + \\
&\quad 0 \cdot 16^1 + \\
&\quad 4 \cdot 16^2 + \\
&\quad 11 \cdot 16^3 + \\
&\quad 15 \cdot 16^4 \\
&= 10 \cdot 1 + 0 \cdot 16 + 4 \cdot 256 + 11 \cdot 4096 + 15 \cdot 65536 \\
&= 1{,}029{,}130_{10}
\end{aligned}
$$

Counting in Hexadecimal

Decimal	Hexadecimal	Decimal	Hexadecimal
0	0 0	16	1 0
1	0 1	17	1 1
2	0 2	18	1 2
3	0 3	19	1 3
4	0 4	20	1 4
5	0 5	21	1 5
6	0 6	22	1 6
7	0 7	23	1 7
8	0 8	24	1 8
9	0 9	25	1 9
10	0 A	26	1 A
11	0 B	27	1 B
12	0 C	28	1 C
13	0 D	29	1 D
14	0 E	30	1 E
15	0 F	31	1 F

4.2.7 Comparison of Number Systems

Decimal	Dual	Octal	Hexadecimal	Decimal	Dual	Octal	Hexadecimal
0	00 000	0 0	0 0	16	10 000	2 0	1 0
1	00 001	0 1	0 1	17	10 001	2 1	1 1
2	00 010	0 2	0 2	18	10 010	2 2	1 2
3	00 011	0 3	0 3	19	10 011	2 3	1 3
4	00 100	0 4	0 4	20	10 100	2 4	1 4
5	00 101	0 5	0 5	21	10 101	2 5	1 5
6	00 110	0 6	0 6	22	10 110	2 6	1 6
7	00 111	0 7	0 7	23	10 111	2 7	1 7
8	01 000	1 0	0 8	24	11 000	3 0	1 8
9	01 001	1 1	0 9	25	11 001	3 1	1 9
10	01 010	1 2	0 A	26	11 010	3 2	1 A
11	01 011	1 3	0 B	27	11 011	3 3	1 B
12	01 100	1 4	0 C	28	11 100	3 4	1 C
13	01 101	1 5	0 D	29	11 101	3 5	1 D
14	01 110	1 6	0 E	30	11 110	3 6	1 E
15	01 111	1 7	0 F	31	11 111	3 7	1 F

4.3 Powers of 2

N	2^N	N	2^N
0	1	17	131,072
1	2	18	262,144
2	4	19	524,288
3	8	20	1,048,576
4	16	21	2,097,152
5	32	22	4,194,304
6	64	23	8,388,608
7	128	24	16,777,216
8	256	25	33,554,432
9	512	26	67,108,864
10	1,024	27	134,217,728
11	2,048	28	268,435,456
12	4,096	29	536,870,912
13	8,192	30	1,073,741,824
14	16,384	31	2,147,483,648
15	32,768	32	4,294,967,296
16	65,536	33	8,589,934,592

4.4 Conversions Between Number Systems

4.4.1 Conversion of Natural Numbers

Base 10 $\Rightarrow$ Base *B* Conversion

We use the metod of **iterated division** to convert a number N_{10} of base 10 into a number N_B of base B.

1. Divide the number N_{10} by B: whole number $\overline{N}$ & remainder R.
   - Take $\overline{N}$ as the next number to be divided by B.
   - Keep R as the next left-most digit of the new number N_B.
2. If $\overline{N}$ is zero then STOP, else set $N_{10} = \overline{N}$ and go to step 1.

Example: $1020_{10} = ?_{8}$

Number to be converted: $N_{10} = 1020$; target base $B = 8$

Step 1: 1020 : 8 = 127 Remainder: **4** $127 \cdot 8 = 1016$

Step 2: 127 : 8 = 15 Remainder: **7** $15 \cdot 8 = 120$

Step 3: 15 : 8 = 1 Remainder: **7** $1 \cdot 8 = 8$

Step 4: 1 : 8 = 0 Remainder: **1**

$\Rightarrow 1020_{10} = 1774_{8}$

Base $B \Rightarrow$ Base 10 Conversion

We use the metod of **iterated multiplication** to convert a number N_B of base B into a number N_{10} of base 10.

We simply add the "contribution" of each digit.

Example: $1F2_{16} = ?_{10}$

Number to be converted: $N_{16} = 1F2$; target base $B = 10$

$$
\begin{aligned}
1F2_{16} &= 2 \cdot 16^0 + \\
&\quad 15 \cdot 16^1 + \\
&\quad 1 \cdot 16^2 \\
&= 2 + 240 + 256
\end{aligned}
$$

$$\Rightarrow 1F2_{16} = 498_{10}$$

Conversions between Numbers 2^n and 2^m

- **Conversion $2^n \Rightarrow 2^1$**

n = 3: octal $\Rightarrow$ dual:

Replace 1 octal digit by 3 binary digits.

Example: $3\ 5\ 7_8 = 011\ 101\ 111_2$

n = 4: hexadecimal $\Rightarrow$ dual

Replace 1 hexadecimal digit by 4 binary digits.

Example: $1\ F\ 2_{16} = 0001\ 1111\ 0010_2$

- **Conversion $2^1 \Rightarrow 2^n$**

n = 3: dual ⇒ octal:

Replace 3 binary digits by 1 octal digit.

Example: $101111010_2 = 101\ 111\ 010_2$

$= 5\ 7\ 2_8$

n = 4: hexadecimal ⇒ dual

Replace 4 binary digits by 1 hexadecimal digit.

Example: $110001111110010_2 = 11\ 0001\ 1111\ 0010_2$

$= 3\ 1\ F\ 2_{16}$

- **Conversion $2^n \Rightarrow 2^m$**

  $2^n \Rightarrow 2^1 \Rightarrow 2^m$

  Example: octal $\Rightarrow$ hexadecimal

$$26351_8 = 2\ \ 6\ \ 3\ \ 5\ \ 1_8$$

octal to binary: $= 010\ 110\ 011\ 101\ 001_2$

rearrange binary: $= 0010\ 1100\ 1110\ 1001_2$

$$= 2\ \ C\ \ E\ \ 9_{16}$$

4.4.2 Conversion of Rational Numbers

Note: An *exact* conversion of rational numbers $R_{10} \Rightarrow R_B$ or $R_B \Rightarrow R_{10}$ is not always possible.

- We want: $R_{10} = R_B + \varepsilon$, where ε is a "sufficiently" small number.

Base *B* $\Rightarrow$ Base 10 Conversion

Example: $0.11001_2 = ?_{10}$

$$0.11001 = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5}$$

$$= 1 \cdot 0.5 + 1 \cdot 0.25 + 0 \cdot 0.125 + 0 \cdot 0.0625 + 1 \cdot 0.03125$$

$$= 0.5 + 0.25 + 0.03125$$

$$= 0.78125_{10}$$

Base 10 ⇒ Base *B* Conversion

Example: $0.19_{10} = ?_2$ with $k = 9$ bit precision

Step i	N	Operation	R	$z_{(-i)}$
1	0.19	$0.19 \cdot 2 = 0.38$	0.38	0
2	0.38	$0.38 \cdot 2 = 0.76$	0.76	0
3	0.76	$0.76 \cdot 2 = 1.52$	0.52	1
4	0.52	$0.52 \cdot 2 = 1.04$	0.04	1
5	0.04	$0.04 \cdot 2 = 0.08$	0.08	0
6	0.08	$0.08 \cdot 2 = 0.16$	0.16	0
7	0.16	$0.16 \cdot 2 = 0.32$	0.32	0
8	0.32	$0.32 \cdot 2 = 0.64$	0.64	0
9	0.64	$0.64 \cdot 2 = 1.28$	0.28	1

$\Rightarrow 0.19_{10} = 0.001100001_2 + \varepsilon$

Multiplication!

4.5 Binary Logic

Logical AND ($x \wedge y$)

AND	0	1
0	0	0
1	0	1

Logical OR ($x \vee y$)

OR	0	1
0	0	1
1	1	1

Logical NOT ($\neg x$, negation)

x	$\neg x$
0	1
1	0

Logical XOR (exclusive OR)

XOR	0	1
0	0	1
1	1	0

Logical NAND (negated AND)

NAND	0	1
0	1	1
1	1	0

Examples:

AND

```
11001010 01000111 11110000
00110101 01110010 10101010
--------------------------
00000000 01000010 10100000
```

NAND

```
11001010 01000111 11110000
00110101 01110010 10101010
--------------------------
11111111 10111101 01011111
```

OR

```
11001010 01000111 11110000
00110101 01110010 10101010
--------------------------
11111111 01110111 11111010
```

XOR

```
11001010 01000111 11110000
00110101 01110010 10101010
--------------------------
11111111 00110101 01011010
```

4.6 Binary Arithmetic

Elementary Rules for addition, subtraction, and multiplication

	Operation	Result	Carry
Addition	0 + 0	0	0
	0 + 1	1	0
	1 + 0	1	0
	1 + 1	0	1
Subtraction	0 - 0	0	0
	0 - 1	1	1
	1 - 0	1	0
	1 - 1	0	0
Multiplication	0 · 0	0	0
	0 · 1	0	0
	1 · 0	0	0
	1 · 1	1	0

Examples:

Addition:

```
        1 0 1 0
+       1 0 1 1
-----------------
carry:  1   1
+     1   1
-----------------
      1 0 1 0 1
```

$$\begin{array}{r} 10_{10} \\ + \quad 11_{10} \\ \hline 21_{10} \end{array}$$

Subtraction:

```
        1 1 0 1
-       1 0 1 0
-----------------
carry:      1
-         1
-----------------
        0 0 1 1
```

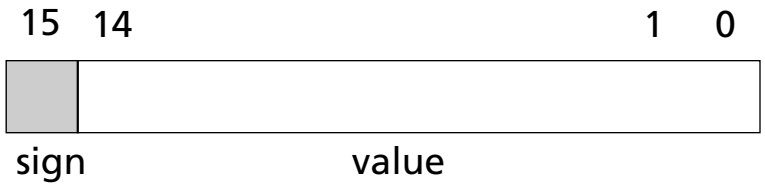
$$\begin{array}{r} 13_{10} \\ - \quad 10_{10} \\ \hline 3_{10} \end{array}$$

Multiplication:

```
                  1 1 1 · 1 0 1
-------------------------------
                  1  1  1
                     0  0  0
+                       1  1  1
-------------------------------
carry:                  1
+                    1
carry:               1
+                 1
carry:            1
+              1
-------------------------------
              1  0  0  0  1  1
```

4.7 Negative Numbers and Complements

Integer Numbers: A possible representation



Number = sign & value

- sign = 0 $\Rightarrow$ +, positive number
- sign = 1 $\Rightarrow$ -, negative number

Range of integer numbers for *n* bits: $-2^{n-1}+1 \ldots 2^{n-1}-1$

Example (1): *n = 3*

Sign	Value	Number
0	0 0	0
0	0 1	1
0	1 0	2
0	1 1	3
1	0 0	?
1	0 1	- 1
1	1 0	- 2
1	1 1	- 3

Negative zero?

Range: $-2^{3-1}+1 \ldots 2^{3-1}-1$

$= -2^{2}+1 \ldots 2^{2}-1$

$= -4+1 \ldots 4-1 = -3 \ldots +3$

4.7.1 Problems of Signed Number Representation

- Given 2 binary numbers: one positive, one negative

```
      0  0 1 1 0 0  (base 2)          12      (base 10)
+     1  0 1 1 1 1                   -15
 ------------------                 -----
      1  1 1 0 1 1  (= -27_10)        -3
```

- "1-1": 1 - 1 = 1 + (-1) = 0 ?

```
      0   0 0 1                        0 0 1
+     1   0 0 1                  -     0 0 1
 ---------------                  -------------
      1   0 1 0  (= -2_10)            0 0 0
```

Clearly this won´t work!

Solution: 1-Complement and 2-Complement

4.7.2 1-Complement ((*B-1*)-Complement)

We represent negative numbers by **negating** the **positive** representation of the number (and vice versa):

- any 1 is changed to a 0
- any 0 is changed to a 1

Example:

Binary	Decimal	One´s Complement	Decimal
0 1001	9	1 0110	- 9
1 1001	-6	0 0110	+ 6
0 0000	0	1 1111	-15
0 1111	15	1 0000	?

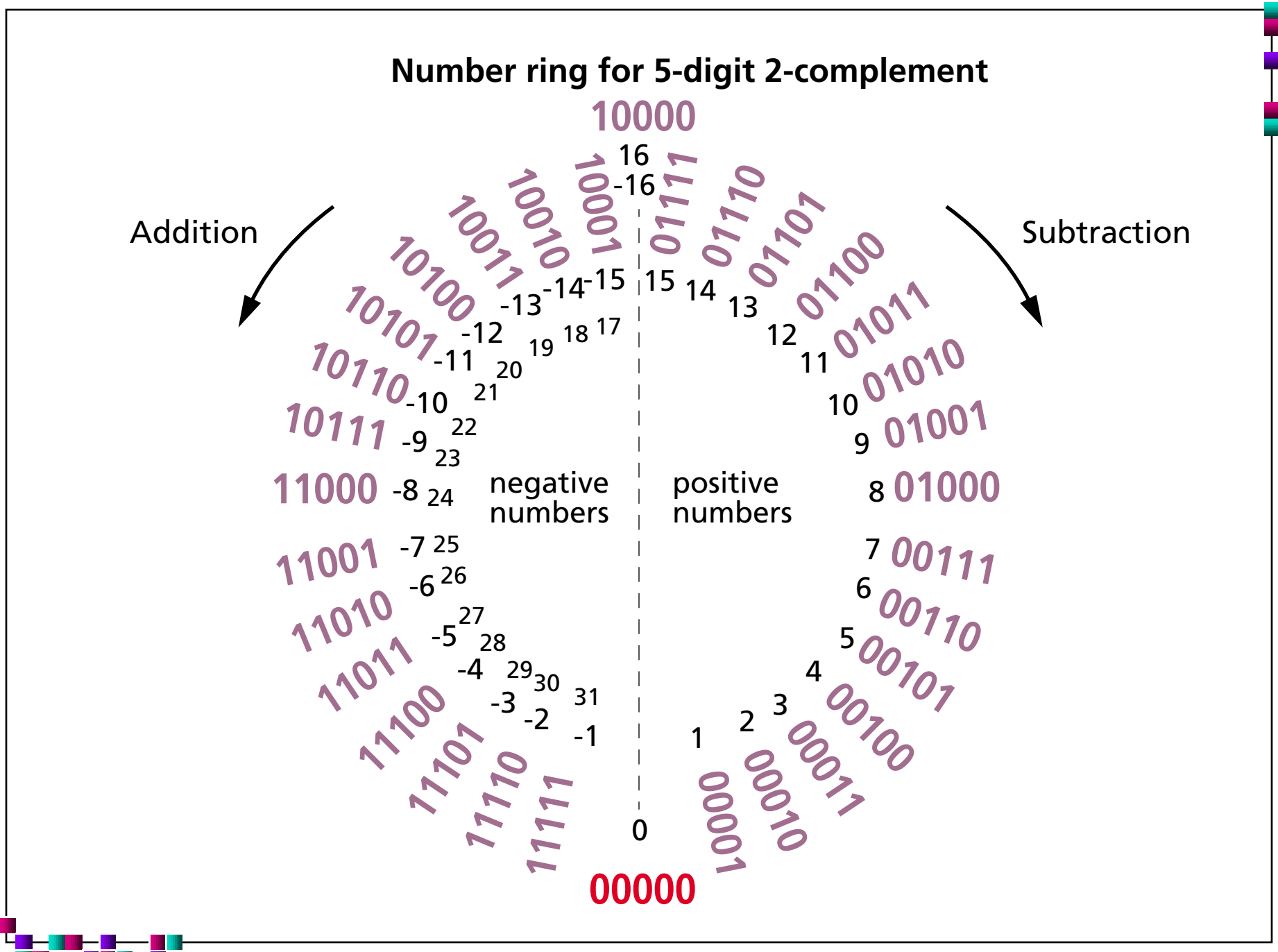
Problem!

4.7.3 2-Complement (*B*-Complement)

- Positive numbers are represented as usual.
- Negative numbers are created by
  - **negating the positive** value and
  - **adding one** (this avoids the negative zero).

Examples:

Binary	Decimal	Two´s Complement	Decimal
0 1001	9	1 0110 + 1 = 1 0111	-9
0 1101	3	1 0010 + 1 = 1 0011	-3
0 0000	0	1 1111 + 1 = 0 0000	0
0 1111	15	1 0000 + 1 = 1 0001	-15
1 1111	-1	¬(1 1111 - 1) = ¬(1 1110) = 0 0001	1
1 0110	-10	0 1010	10



With the 2-complement, subtractions can be performed as simple additions:

Example: 5 digit precision, base 10

	Example 1	*Example 2*
B = 10	$14_{10} - 7_{10}$	$9_{10} - 13_{10}$
Complement	7: 00111 $(-7)_{B-1}$: 11000 + 1 $(-7)_B$: 11001	13: 01101 $(-13)_{B-1}$: 10010 + 1 $(-13)_B$: 10011
Addition	14: 01110 $+(-7)_B$: + 11001 7: 1 00111	9: 01001 $+(-13)_B$: + 10011 -4: 11100

4.8 Floating Point Numbers

4.8.1 Mantissa and Exponent

Recall the general form of numbers expressed in scientific notation:

- 3.14159265358979323846264338327950288419 7
- 6.02×10^{23}

Printed out by a computer these numbers usually look like this:

- 3.1415e0
- 6.02e23

This representation saves space and reduces redundancy.

Hence, the usual format for a 32-bit real (floating point) number is:

4.8.2 Floating Point Arithmetics

Example: $0.740 \cdot 10^5 + 0.843 \cdot 10^3$

Operands	Compare Exponents	Adjust Exponents and Mantissae
$0.740 \cdot 10^5$ $0.843 \cdot 10^3$	$e_1 - e_2 = 2$	$0.740 \cdot 10^5$ $+ 0.008 \cdot 10^5$
		Result: $0.748 \cdot 10^5$

Addition:

- 1. Identify mantissae and exponents
- 2. Compare exponents
- 3. Adjust exponents if necessary
- 4. Adjust the shorter mantissa
- 5. Add mantissae
- 6. Normalize mantissa
- 7. Adjust exponent

Example: Multiplication

- 1. Multiply of mantissae
- 2. Normalize mantissae
- 3. Adjust exponents
- 4. Add exponents

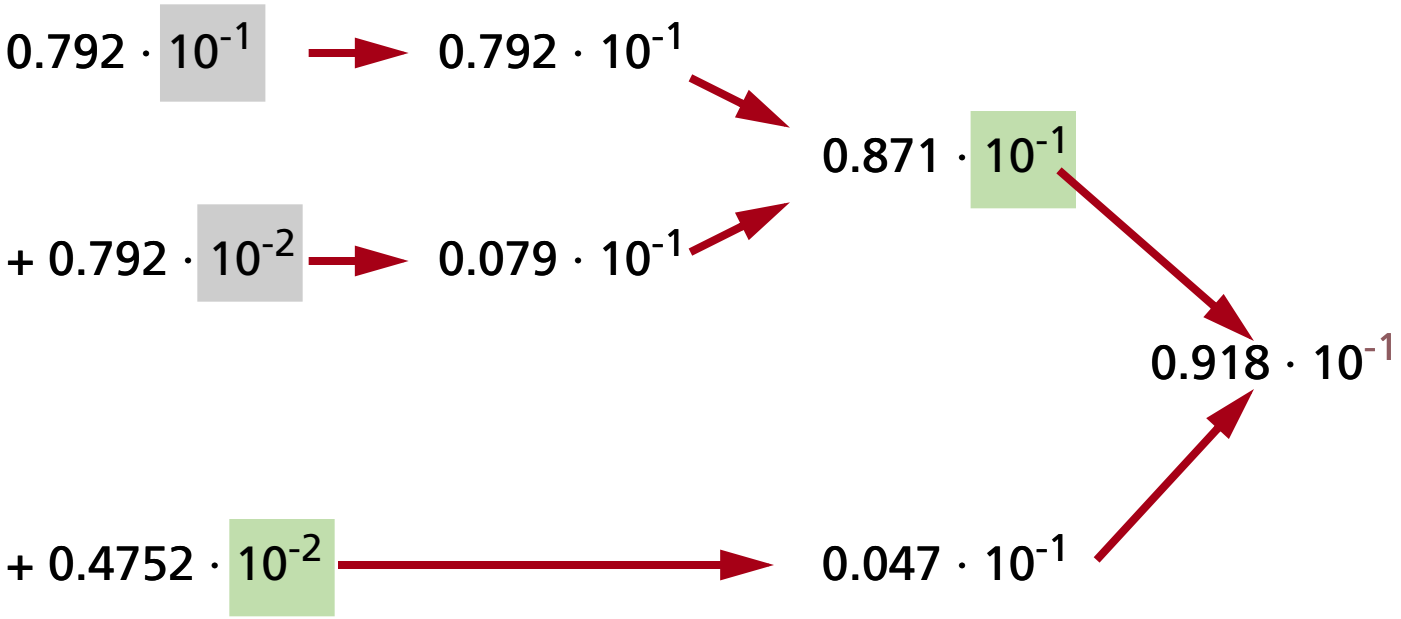
Example: $(0.792 \cdot 10^{5}) \cdot (0.116 \cdot 10^{-3})$

- **Step 1**: multiplication of the mantissae:

$0.792 \cdot 0.116 =$

$$\begin{array}{rl} & 0.792 \cdot 10^{-1} \\ + & 0.792 \cdot 10^{-2} \\ + & 4.752 \cdot 10^{-3} \end{array}$$

- **Step 1** (cont.): step-by-step multiplication of the mantissae



- **Step 2**: Add exponents

  $5 + (-3) + (-1) = 1$

- **Result**:

  $0.918 \cdot 10^1$ (with 3 digits of precision)in the mantissa)

4.8.3 Limited Precision

Note: Not all values can be represented!

Example:

- mantissa: 2 decimal digits
- exponent: 1 decimal digit

- Sample number: $74 \cdot 10^2 = 7400$

  What is the next higher value?

  $75 \cdot 10^2 = 7500$

  What about values $7400 < x < 7500$?

  $\Rightarrow$ **They cannot be represented!!!**

- Remedy, but not a perfect solution:

  Increase the number of mantissa digits.

4.9 Representation of Strings

4.9.1 Representation of Characters

Usually: single characters are represented by 1 byte

different standards: *ASCII*[1], EBCDIC[2]

Currently: internationalization with *Unicode*

a single character is represented by 2 bytes

1. ASCII: American Standard Code for Information Interchange
2. EBCDIC: Extended Binary Coded Decimal Interchange Code (derived from punch card standards)

ASCII Character Encoding

D	C	D	C	D	C	D	C	D	C	D	C	D	C	D	C
0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	‘	112	p
1	SCH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	“	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	´	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	S0	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	S1	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

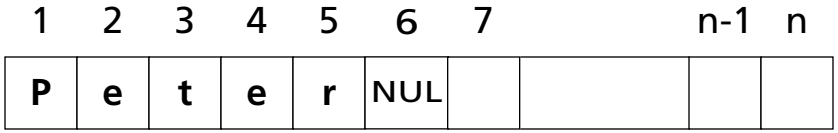
Compare the *Unicode* character encodings …

4.10 Representation of Strings

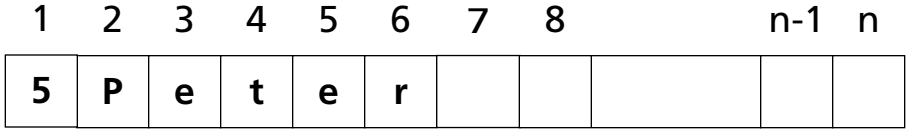
Strings are represented by sequences of characters (⇒ usually: byte sequence)

1. Fixed length strings (example: length = 7)

2. Variable length strings



3. Encoding with string length at the beginning



With *ASCII* charcter encoding **string comparisons** are possible:

´AA´	<	´AD´	because:	$101\ 101_8$ <	$101\ 104_8$
´AA´	<	´Aa´		$101\ 101_8$ <	$101\ 141_8$
´A1´	<	´AA´		$101\ 061_8$ <	$101\ 101_8$

Hence, sorting of strings becomes easy:

⇒ Compare their numerical representations!