SYST 27198

CPU Architecture and Assembly

Dr. Rachel Jiang

Sheridan Institute of Technologies and Advanced Learning

Winter 2017

Acknowledgement

This notes is based on Professor Victor Ralevich's course Notes of

"**Structured Computer Organization**"

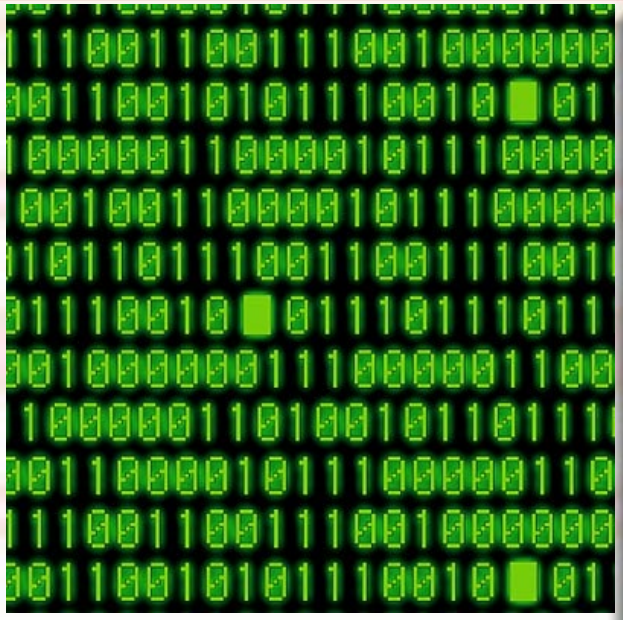
Updated by Rachel Jiang

Jan. 2017

Chapter Three

Data Representation and Storage

Topics

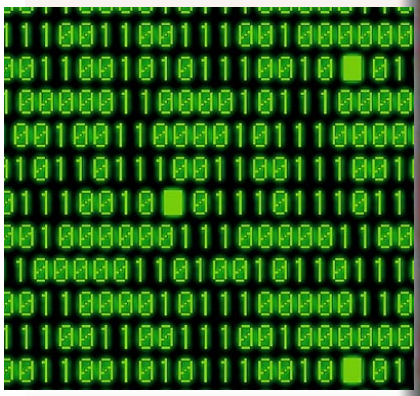


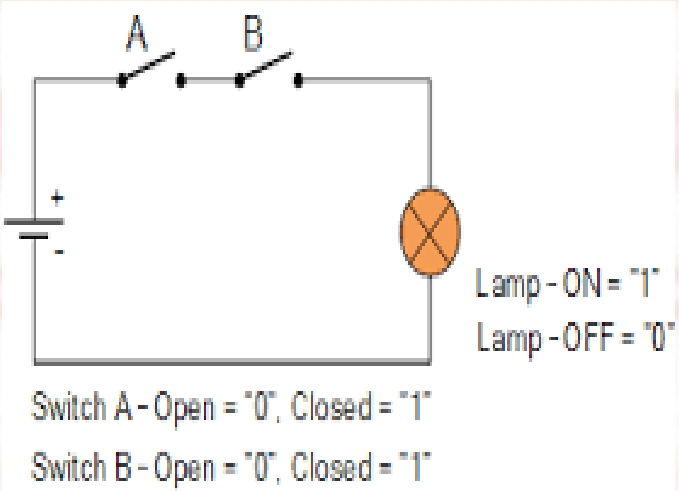
- Binary data representation
- Number system base conversion
  - Integers
  - Fraction numbers
- Representation of negative numbers
- Representing floating-point numbers
- Normalization
- Error, range, and precision
- Standards for floating point number computer storage
- Alphanumeric data

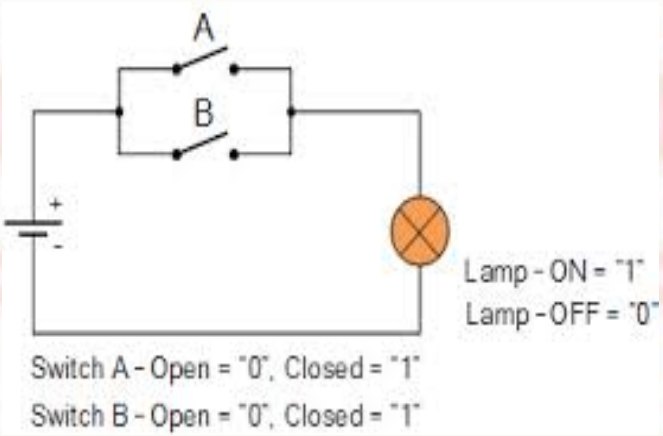
Binary Data Representation

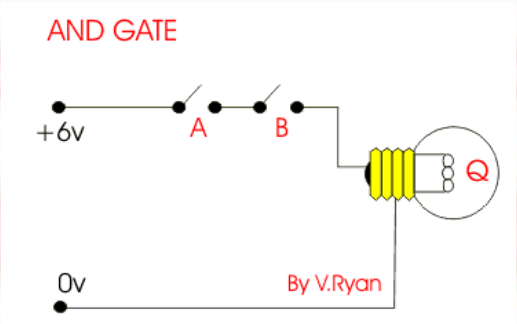
- Everything that is stored and processed in the CPU, and memory is always represented in binary form, i.e. as sequence of 0's and 1's.

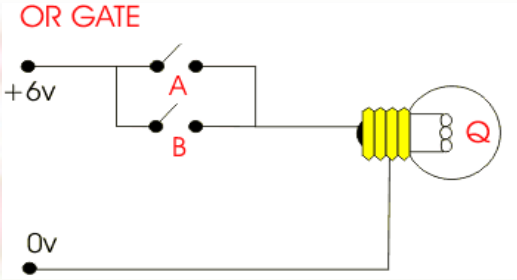
Data	Comment
Numeric data	integers, real numbers
Boolean constants	true, false
Instructions	programming language source code
Alphanumeric symbols	Unicode, ASCII, EBCDIC
Image (Bitmap)	GIF, PCX, TIFF, BMP
Outline graphics and fonts	PostScript, TrueType
Sound	WAV, AVI, MP3, MIDI
Page description	pdf, HTML, XML
Video	QuickTime, MPEG-2, RealVideo




A
B
+
-
Lamp - ON = "1"
Lamp - OFF = "0"
Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"


A
B
+
-
Lamp - ON = "1"
Lamp - OFF = "0"
Switch A - Open = "0", Closed = "1"
Switch B - Open = "0", Closed = "1"


AND GATE
+6v
A
B
Q
0v
By V.Ryan


OR GATE
+6v
A
B
Q
0v

Information Units

- bit - unit of information
- 1 nibble = 4 bits
- 1 Byte = 8 bits
- 1 KB = 2^{10} Bytes = 1024 Bytes
- 1 MB = 2^{10} KB = 1024 KB
  =1024*1024Bytes
- 1 GB = 2^{10} MB = 1024 MB
  =1024*1024*1024Bytes
- 1 TB = 2^{10} GB = 1024 GB
  =1024*1024*1024*1024Bytes

Number System Base Conversion

Positive Integers

Numeric system	Base	Digits
Binary	N = 2	0, 1
Ternary	N = 3	0, 1, 2
Octal	N = 8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	N = 10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	N = 16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
System with base n	N = n	0, 1, 2, 3, …, (n-1)

Binary representation: 1001 1100 1010 1110 1001110010101110_2

Octal representation: 116256 $0116256 = 116256_8$

Hexadecimal representation: 9CAE 9CAEh = 0x9CAE

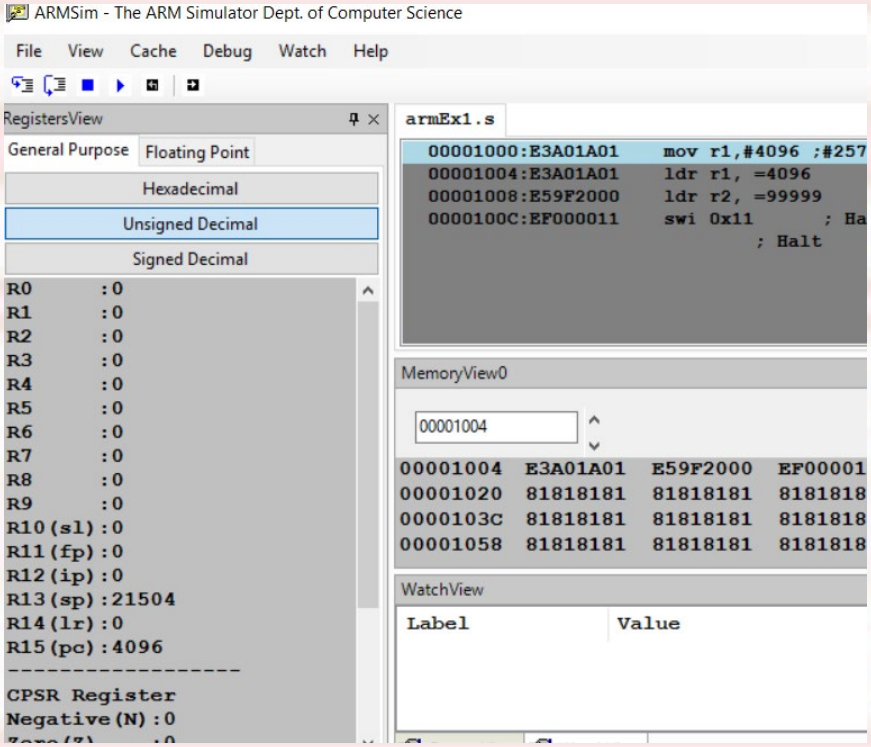
$1(2^{15}) + 0(2^{14}) + 0(2^{13}) + 1(2^{12}) + 1(2^{11}) + 1(2^{10}) + 0(2^9) + 0(2^8) + 1(2^7) + 0(2^6) + 1(2^5) + 0(2^4) + 1(2^3) + 1(2^2) + 1(2^1) + 0(2^0) = 40,110_{10}$

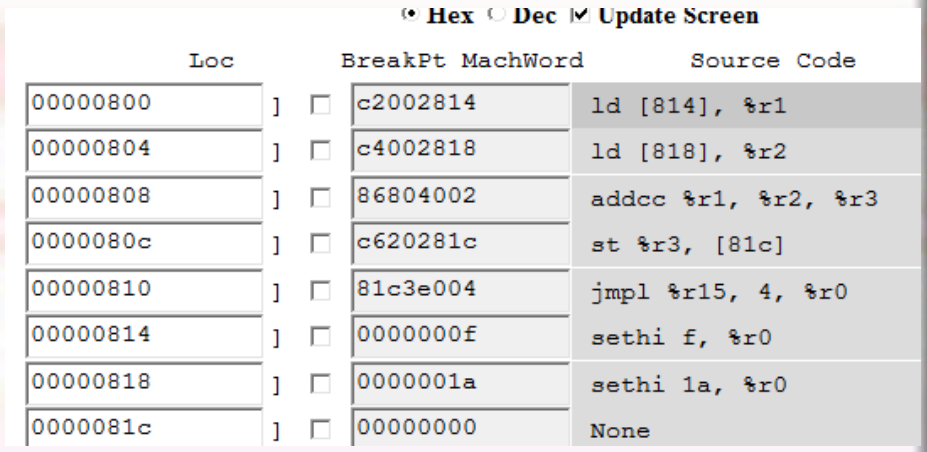
$9(16^3) + 12(16^2) + 10(16^1) + 14(16^0) = 40,110_{10}$

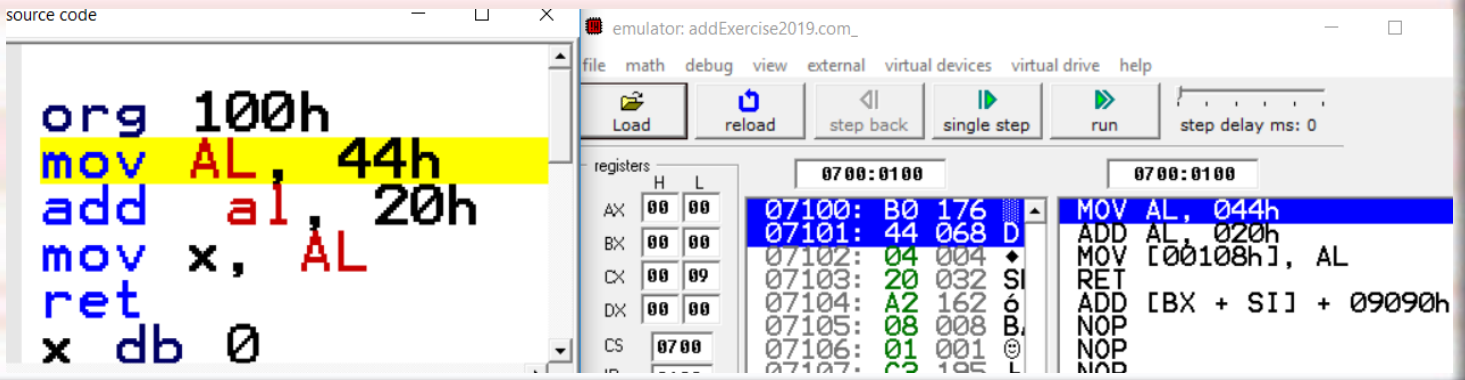
Hex decimal

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

Hex decimal


ARMSim - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help
RegistersView
General Purpose Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :0
R1 :0
R2 :0
R3 :0
R4 :0
R5 :0
R6 :0
R7 :0
R8 :0
R9 :0
R10 (sl) :0
R11 (fp) :0
R12 (ip) :0
R13 (sp) :21504
R14 (lr) :0
R15 (pc) :4096
CPSR Register
Negative (N) :0
armEx1.s
00001000:E3A01A01 mov r1,#4096 ;#257
00001004:E3A01A01 ldr r1, =4096
00001008:E59F2000 ldr r2, =99999
0000100C:EF000011 swi 0x11 ; Ha
; Halt
MemoryView0
00001004
00001004 E3A01A01 E59F2000 EF00001
00001020 81818181 81818181 8181818
0000103C 81818181 81818181 8181818
00001058 81818181 81818181 8181818
WatchView
Label Value


Hex Dec Update Screen
Loc BreakPt MachWord Source Code
00000800 c2002814 ld [814], %r1
00000804 c4002818 ld [818], %r2
00000808 86804002 addcc %r1, %r2, %r3
0000080c c620281c st %r3, [81c]
00000810 81c3e004 jmpl %r15, 4, %r0
00000814 0000000f sethi f, %r0
00000818 0000001a sethi 1a, %r0
0000081c 00000000 None


source code
org 100h
mov AL, 44h
add al, 20h
mov x, AL
ret
x db 0
emulator: addExercise2019.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
AX BX CX DX CS
0700:0100
07100: B0 176
07101: 44 068 D
07102: 04 004
07103: 20 032 SI
07104: A2 162 ó
07105: 08 008 B
07106: 01 001
MOV AL, 044h
ADD AL, 020h
MOV [00108h], AL
RET
ADD [BX + SI] + 09090h
NOP
NOP

- Representation of a number in a system with base **N** is given in the form:

$$M = a_k N^k + a_{k-1} N^{k-1} + \ldots + a_1 N^1 + a_0$$

with $0 \le a_j < N$ for every $0 \le j \le k$.

Alternative notation: $M = (a_k a_{k-1} \ldots a_1 a_0)_N$

- Conversion algorithm for whole numbers for obtaining coefficients $\mathbf{a_j}$ comes from the fact
  - $\mathbf{M_j = Int(M_{j-1}/N),\ 0 \le j \le k}$
  - $\mathbf{a_0 = M \bmod N}$

$$M = M_0 = a_0 + N(a_1 + N(a_2 + N(\ldots)) = a_0 + N \cdot M_1$$
$$M_1 = a_1 + N(a_2 + N(\ldots)) = a_1 + N \cdot M_2 = Int(M_0/N),$$
$$M_2 = a_2 + N(a_3 + N(\ldots)) = a_2 + N \cdot M_3 = Int(M_1/N), \ldots$$
$$\ldots\ldots\ldots$$
$$M_j = Int(M_{j-1}/N),\ 0 \le j \le k$$

$$a_0 = M \bmod N,$$

$$a_1 = Int(M/N) \bmod N = Int(M_0/N) \bmod N = M_1 \bmod N,$$
$$\ldots$$
$$a_j = M_j \bmod N,\ 0 \le j \le k.$$

Example

- ($2637):
- $A_0 = 2637 \bmod 10 = 7$
- $A_1 = \text{Int}(263/10) \bmod 10 = 3$
- $A_2 = \text{Int}(26/10) \bmod 10 = 6$
- $A_4 = \text{Int}(2/10) \bmod 10 = 2$
- $M_5 = \text{Int}(2/10) = 0$
  - *end*

$M = M_0 = a_0 + N(a_1 + N(a_2 + N(\ldots)) = a_0 + N \cdot M_1$
$M_1 = a_1 + N(a_2 + N(\ldots)) = a_1 + N \cdot M_2 = \text{Int}(M_0/N)$,
$M_2 = a_2 + N(a_3 + N(\ldots)) = a_2 + N \cdot M_3 = \text{Int}(M_1/N)$, …
……
$M_j = \text{Int}(M_{j-1}/N)$, $0 \le j \le k$

$a_0 = M \bmod N$,

$a_1 = \text{Int}(M/N) \bmod N = \text{Int}(M_0/N) \bmod N = M_1 \bmod N$,
…
$a_j = M_j \bmod N$, $0 \le j \le k$.

- **Example**:
  - Conversion of 1173_{10} into binary number

1173 divided by 2 = 586	Remainder	1
586 divided by 2 = 293	Remainder	0
293 divided by 2 = 146	Remainder	1
146 divided by 2 = 73	Remainder	0
73 divided by 2 = 36	Remainder	1
36 divided by 2 = 18	Remainder	0
18 divided by 2 = 9	Remainder	0
9 divided by 2 = 4	Remainder	1
4 divided by 2 = 2	Remainder	0
2 divided by 2 = 1	Remainder	0
1 divided by 2 = 0	Remainder	1

Read the remainders from the bottom to the top:

$10010010101_2 = 1173_{10}$

Data Type Range

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,80
unsigned long	8 bytes	0 to 18,446,745,073,709,551,616

More Data Type Example

int	32	-2147483648 to 2147483647
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
void	0	valueless

IEEE 754 single floating point number data range

Maximum:

3.402823 ×10000000000 0000000000 0000000000 00000000

Minimum:
0.0000000000 0000000000 0000000000 00000003402823

float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308

Inclass Exercise I

- Convert 169 (base 10) to
  - 1. base 2 notation
  - 2. base 3 notation
  - 3. base 5 notation
  - 4. base 8 notation
  - 5. base 16 notation

Inclass Exercise I (2)

- Convert 169 from the following base to decimal
  - 1. base 2 notation
  - 2. base 3 notation
  - 3. base 5 notation
  - 4. base 8 notation
  - 5. base 16 notation

Number System Base Conversion

- **Fraction numbers**
  - one base can be also transformed into any numeric system with some other base.
  - It may occur that number with finitely many "decimal" (fraction part) places in one base has infinitely many "decimals" in some other base (i.e convert 0.2 to base 2).

$$X = (0.x_{-1}x_{-2}x_{-3} \ldots x_{-k})_N = x_{-1}N^{-1} + x_{-2}N^{-2} + \ldots + x_{-k}N^{-k}$$

Example: $X = (0.3401)_{10} = 3 \cdot 10^{-1} + 4 \cdot 10^{-2} + 0 \cdot 10^{-3} + 1 \cdot 10^{-4}$

- # Transformation of number X given in numerical system with base N to a number in a system with base M:

$$X = X_0 = (0.x_{-1}x_{-2}x_{-3} \ldots x_{-k})_N = (0.a_{-1}a_{-2}a_{-3} \ldots a_{-k})_M$$

$$X_0 \cdot M = a_{-1} + (0.a_{-2}a_{-3} \ldots a_{-k})_M = a_{-1} + X_{-1}, \text{ and } a_{-1} = \text{Int}(X_0 \cdot M)$$

Consequently:

$$a_{-j} = \text{Int}(X_{-j+1} M), \quad 0 \le j \le k.$$

where X_{-j} is a fraction part of a product $X_{-j+1}M$, i.e.

$$X_{-j+1}M = a_{-j} + X_{-j}$$

Example

Conversion of a decimal number $X = (0.625)_{10}$ into the binary format $X = (0.a_{-1}a_{-2}a_{-3}\ldots)_2$.

$X = X_0 = (0.625)_{10}$,

$a_{-1} = \text{Int}(2 \cdot X) = \text{Int}(1.25) = 1, X_{-1} = (0.25)_{10}$,

$a_{-2} = \text{Int}(2 \cdot X_{-1}) = \text{Int}(0.5) = 0, X_{-2} = (0.5)_{10}$

$a_{-3} = \text{Int}(2 \cdot X_{-2}) = \text{Int}(1.0) = 1, X_{-j} = 0$ for $3 <= j$.

$X = (0.101)_2$

Example 2

- It is not always possible to convert a terminating base 10 fraction into an equivalent terminating base 2 fraction.

$X = (.2)_{10}$

$a_{-1} = \text{Int}(2 \cdot X) = \text{Int}(.4) = 0,\ X_{-1} = (0.4)_{10},$

$a_{-2} = \text{Int}(2 \cdot X_{-1}) = \text{Int}(0.8) = 0,\ X_{-2} = (0.8)_{10}$

$a_{-3} = \text{Int}(2 \cdot X_{-2}) = \text{Int}(1.6) = 1,\ X_{-3} = (0.6)_{10}$

$a_{-4} = \text{Int}(2 \cdot X_{-3}) = \text{Int}(1.2) = 1,\ X_{-4} = (0.2)_{10}$

$a_{-5} = \text{Int}(2 \cdot X_{-4}) = \text{Int}(0.4) = 0,\ X_{-4} = (0.4)_{10}$ .....

1173 divided by 2 = 586	Remainder	1
586 divided by 2 = 293	Remainder	0
293 divided by 2 = 146	Remainder	1
146 divided by 2 = 73	Remainder	0
73 divided by 2 = 36	Remainder	1
36 divided by 2 = 18	Remainder	0
18 divided by 2 = 9	Remainder	0
9 divided by 2 = 4	Remainder	1
4 divided by 2 = 2	Remainder	0
2 divided by 2 = 1	Remainder	0
1 divided by 2 = 0	Remainder	1

Read the remainders from the bottom to the top:

$10010010101_2 = 1173_{10}$

Example 3

- **More on Base Conversions**
  - Converting among power-of-2 bases:

$$1011_2 = (10_2)(11_2) = 23_4$$

$$23_4 = (2_4)(3_4) = (10_2)(11_2) = 1011_2$$

$$101010_2 = (101_2)(010_2) = 52_8$$

$$01101101_2 = (0110_2)(1101_2) = 6D_{16}$$

Inclass Exercise II

- **Fraction number base conversions:**
  - 1. Convert 3.14 to its base 2 notation
  - 2. convert 0.628 to its base 2 notation
  - 3. convert 0.101101 to base 10 notation

Representation of Negative Numbers

- Unsigned (Non-negative) Integers
  - Unsigned integer (nonnegative integer) for base (radix) N

$$M = (a_k a_{k-1} \ldots a_1 a_0)_N = a_k N^k + a_{k-1} N^{k-1} + \ldots + a_1 N^1 + a_0$$

For example, if N = 2, the binary representation is:

$$M = (a_k a_{k-1} \ldots a_1 a_0)_2 = a_k 2^k + a_{k-1} 2^{k-1} + \ldots + a_1 2^1 + a_0$$

Negative Integers

- There are several different representations of the negative integers. We'll cover
  - **signed-magnitude,**
  - **complementary representation**, and
  - **excess (biased)**
- All systems represent positive integers in **the same way** but they differ in the way they represent negative numbers.
- In all systems
  - we always use fixed predefined number of digits.

Signed Magnitude Method

- For binary numbers
  - the most significant bit is used to specify the sign:
    - 0 is positive
    - 1 is negative.
  - Sequence of *n* binary digits may be used to represent all integers within range

    $[-2^{n-1}+1, 2^{n-1}-1]$ and zero is represented in two different ways:

    100…0

    000…0.

Complementary representation

- The values depend on the word size of the number.

  complement = basis – value

- A fixed word size results in a range of some particular fixed size.
- It is possible to have a combination of numbers
  - adds to a result outside the range
  - This condition is called **overflow**

Two's Binary Complementary Representation

- The **two's complement system**
  - represents negative number –x in the following way:

    1. Find the **n-bit** base-2 representation of +x, and include all leading 0's,
    2. In result, replace 0's by 1's and 1's by 0's.
    3. Add 1 to the result of (2) and
    4. ignore any carry coming out of the most significant bit.

Example

- Number 29 in 8-bit representation is 00011101.
- In order to obtain binary form for -29_{10} using two's complement system, we switch 1's and 0's and get: 11100010_2.
- Than, finally, add 1. The two's complement method gives 11100011_2 as representation for -29_{10}.

- Most computers will use two's complement arithmetic.
- Computers provide a **carry flag**
  - Is used to correct for carries and borrows that occur
- **Carry and overflow can occur independently of each other**
  - The carry flag is set
    - when the result of an addition or subtraction exceeds the fixed number of bits allocated, without regard to sign.

Overflow

- Overflow occurs when
  - the addition of two positive numbers produces a negative result, or
  - the addition of two negative numbers produces a positive result.
  - Adding operands of unlike signs never produces an overflow.
    - discarding the carry out of the most significant bit during two's complement addition is a normal occurrence, and does not by itself indicate overflow.

Examples

- 4-bit binary representation arithmetic

$(+4_{10}) + (+2_{10})$

```
  0100
  0010
  0110  = (+6₁₀)    No overflow, no carry
```

$(+4_{10}) + (+6_{10})$

```
  0100
  0110
  1010  = (-6₁₀)    Overflow, no carry. The result is
                    incorrect
```

Examples

$(-4_{10}) + (-2_{10})$

```
  1100
  1110
 -----
 11010
```

No overflow, carry. Ignoring the carry the result is correct ($1010 = -6_{10}$)

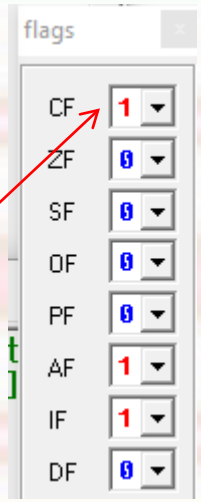
$(-4_{10}) + (-6_{10})$

```
  1100
  1010
 -----
 10110 = (+3₁₀)
```

Overflow, carry. Ignoring the carry, the result is incorrect

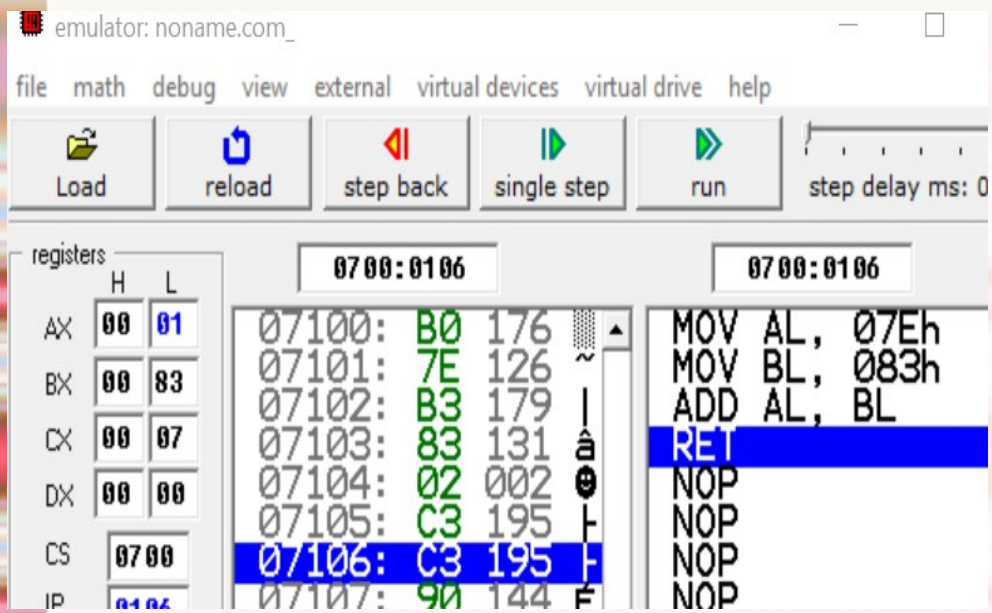
;carry example

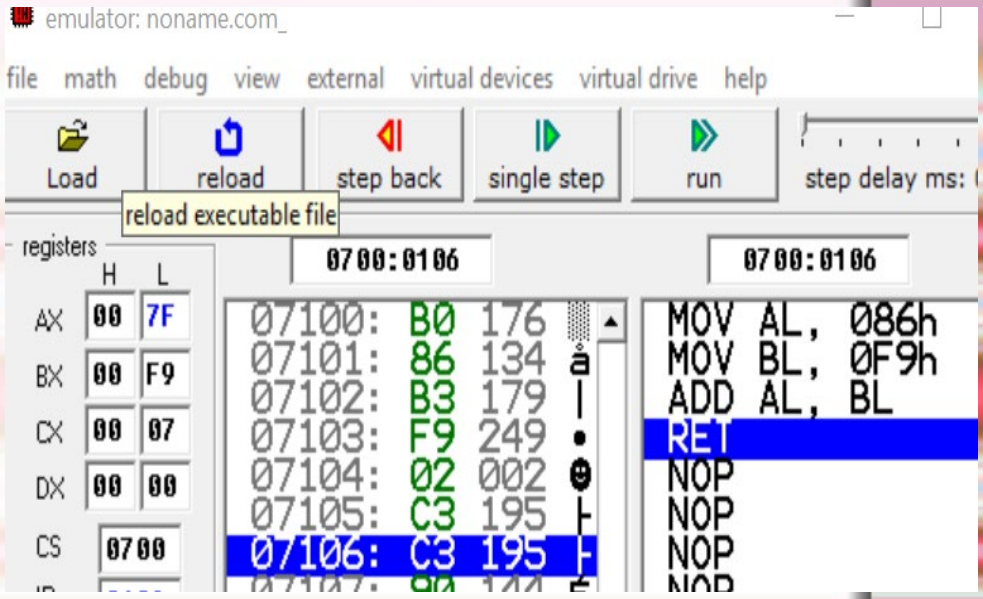
```
org 100h

MOV AL,126
MOV BL,131
add al,bl
ret
```



;overflow example

```
org 100h

MOV AL,-122
MOV BL, -7
add al,bl
ret
```





Ex

- Add the following signed numbers in binary and indicate if there is/are a carry or overflow, or both carry and overflow happened in the calculation:
  - 1. -123 + (-8)
  - 2. 123 + 8
  - 3. 123 + (-8)
  - 4. -123 + 8
  - 5. -85 + (-123)
  - 6. -1 + 69

Excess (Biased) Representation

- Excess (biased) representation of a number
  - obtained by adding a bias to the two's complement representation.
- Excess (or bias)
  - always chosen to be of such magnitude that all binary representations of the given length appear as being unsigned nonnegative numbers.
  - The actual value of the number is obtained by subtracting a bias form the number's binary representation.

Example

- Excess 127 "adds" 127 to the two's complement 8-bit format, ignoring any carry out of the most significant bit:
  - $+12_{10} = 10001100_2$ ($=(127+12)_2 = 139_2$),
  - $-12_{10} = 01110100_2$ ($=(127-12)_2 = 115_2$)
  - As a consequence there is only one representation for zero:
    - $+0 = 10000000_2$, $-0 = 10000000_2$.
    - Largest number is $+127_{10}$,
    - smallest number is -128_{10}, using an 8-bit representation.

BCD Representations
in Nine's and Ten's Complement

- Each binary coded decimal (BDC) digit is composed of 4 bits.

(a)	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 1	$(+301)_{10}$	Nine's and ten's complement
	$(0)_{10}$	$(3)_{10}$	$(0)_{10}$	$(1)_{10}$		
(b)	1 0 0 1	0 1 1 0	1 0 0 1	1 0 0 0	$(-301)_{10}$	Nine's complement
	$(9)_{10}$	$(6)_{10}$	$(9)_{10}$	$(8)_{10}$		
(c)	1 0 0 1	0 1 1 0	1 0 0 1	1 0 0 1	$(-301)_{10}$	Ten's complement
	$(9)_{10}$	$(6)_{10}$	$(9)_{10}$	$(9)_{10}$		

3 – Bit Signed Integer Representation

Decimal	Unsigned	Sign–Mag.	1's Comp.	2's Comp.	Excess 4
7	111	–	–	–	–
6	110	–	–	–	–
5	101	–	–	–	–
4	100	–	–	–	–
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
+0	000	000	000	000	100
-0	–	100	111	000	100
-1	–	101	110	111	011
-2	–	110	101	110	010
-3	–	111	100	101	001
-4	–	–	–	100	000

Inclass Exercise III

- Provide the binary representation -113 in 2's complement, 8-bit notation
  - Find out if overflow or carry occurs in the following signed number computations using Emu8086:
    - 127 + 3
    - 127 + (-3)
    - 64+5
    - (-124) +(-3)
    - (-64) + (-6)

Representing Floating Point Numbers

- Floating-point numbers
  - allow very large/small numbers to be represented using only a few digits, at the expense of precision.
  - The precision
    - primarily determined by the number of digits in the fraction (*significand*, integer and fractional parts),
  - the range
    - primarily determined by the number of digits in the exponent.

Scientific Notation

In science, we deal with some very **LARGE** numbers:

1 mole = 602000000000000000000000

In science, we deal with some very **SMALL** numbers:

Mass of an electron =
0.000000000000000000000000000000091 kg

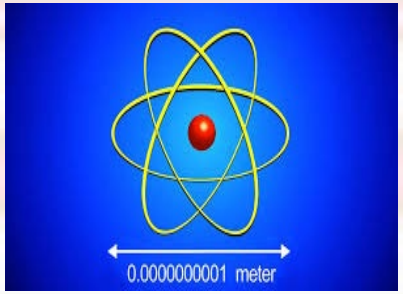


Table 2-2

SI Prefixes

Prefix	Symbol	Meaning	Multiplier (Numerical)	Multiplier (Exponential)
		Greater than 1		
tera	T	trillion	1 000 000 000 000	10^{12}
giga	G	billion	1 000 000 000	10^{9}
mega	M	million	1 000 000	10^{6}
*kilo	k	thousand	1 000	10^{3}
hecto	h	hundred	100	10^{2}
deka	da	ten	10	10^{1}
Unit	1			
		Less than 1		
*deci	d	tenth	0.1	10^{-1}
*centi	c	hundredth	0.01	10^{-2}
*milli	m	thousandth	0.001	10^{-3}
*micro	μ	millionth	0.000 001	10^{-6}
*nano	n	billionth	0.000 000 001	10^{-9}
pico	p	trillionth	0.000 000 000 001	10^{-12}
femto	f	quadrillionth	0.000 000 000 000 001	10^{-15}
atto	a	quintillionth	0.000 000 000 000 000 001	10^{-18}

The distance from Earth to Pluto

- At its most distant,
  - when the two bodies are on the opposite sides of the sun from one another
    - Pluto lies **4.67 billion miles (7.5 billion kilometers)** from Earth.
- At their closest,
  - the two are only **2.66 billion miles** (4.28 billion km) apart.
- Use scientific notation:
  - $0.75*10^{10}$KM and $0.428*10^{10}$KM
    - https://www.google.ca/?gws_rd=ssl#q=distance+from+earth+to+pluto

Exponential (scientific) notation

- 1. The sign of the number.
- 2. The magnitude of the number (mantissa).
- 3. The sign of the exponent.
- 4. The magnitude of the exponent.

- We need to know:
  - 1. The base of the exponent.
  - 2. The location of the radix point.

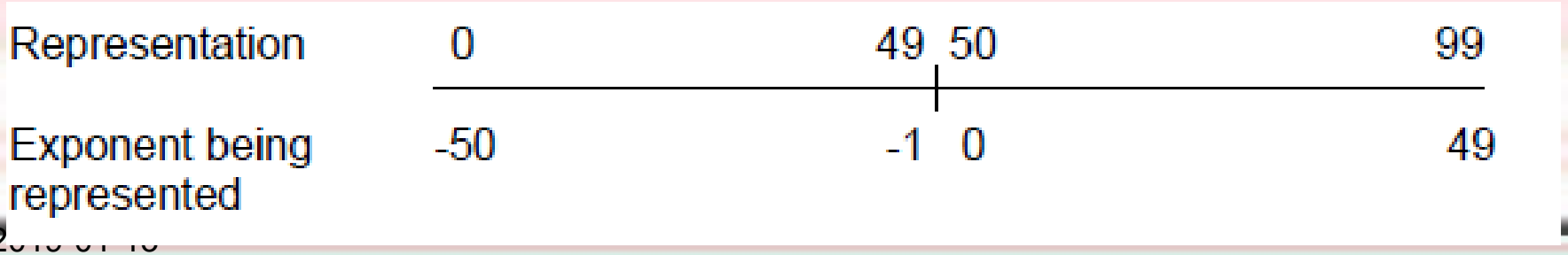
Example

- -0.35790×10^{-6}
- The floating point numbers might assign the digits as follows:
- SEEMMMMM
  - S – sign of mantissa
  - EE – two digits for exponent
  - MMMMM – five digits for mantissa
  - Any method for storing the sign and magnitude of integers could be used for the mantissa.

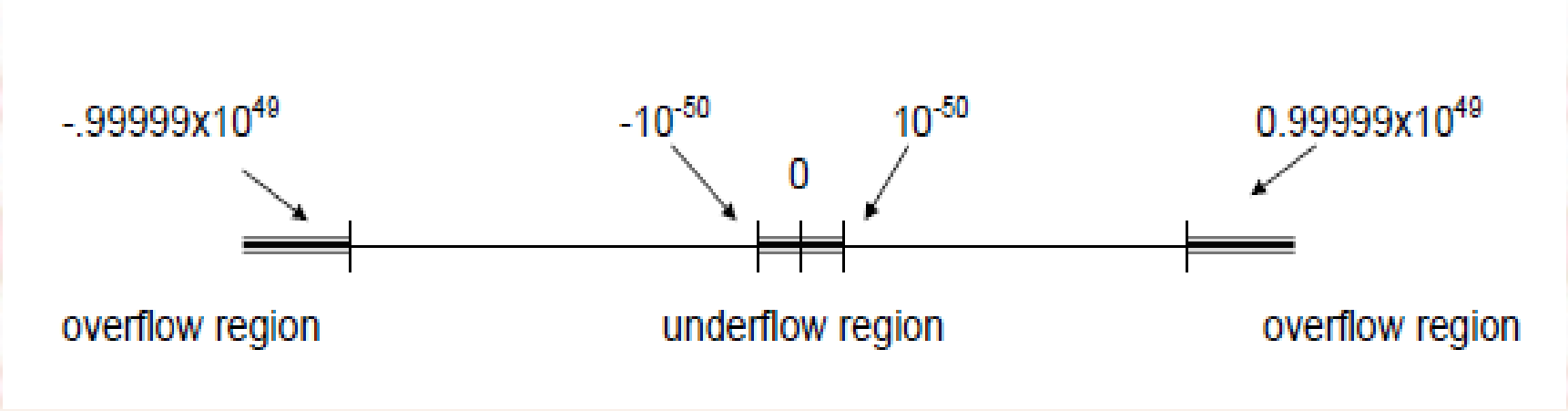
- There is no sign for the exponent
  - the **bias** *(offset)* is used.
- If we pick up a value somewhere in the middle of the possible values for the exponent and declare that value to correspond to the exponent 0
  - every value lower that that will be negative and those above will be positive.
  - What we've done is bias (offset), the value of exponent by the chosen amount.
- To convert exponent we add the bias *(offset)* to the exponent, and store it in that form. The stored form can be returned to usual exponential notation by subtracting the bias (offset).

Excess-50 representation

- This method of storing the exponent is known as **excess-N notation**
  - where N is the chosen midvalue (bias).
  - If the decimal point is located at the beginning of the five digits mantissa, excess-50 notation allows us a magnitude range of:
  - $0.00001 \times 10^{-50} <$ number $< 0.99999 \times 10^{+49}$
  - possible exponent values are from 10^{-50} to 10^{49}



- It is still possible to create an overflow
  - using a number of magnitude too large to be stored.
- It is also possible to have underflow
  - the number is a decimal fraction of magnitude too small to be stored.



Example

- If 0 represents +, and 1 represents -, offset 50, in system with base 10:

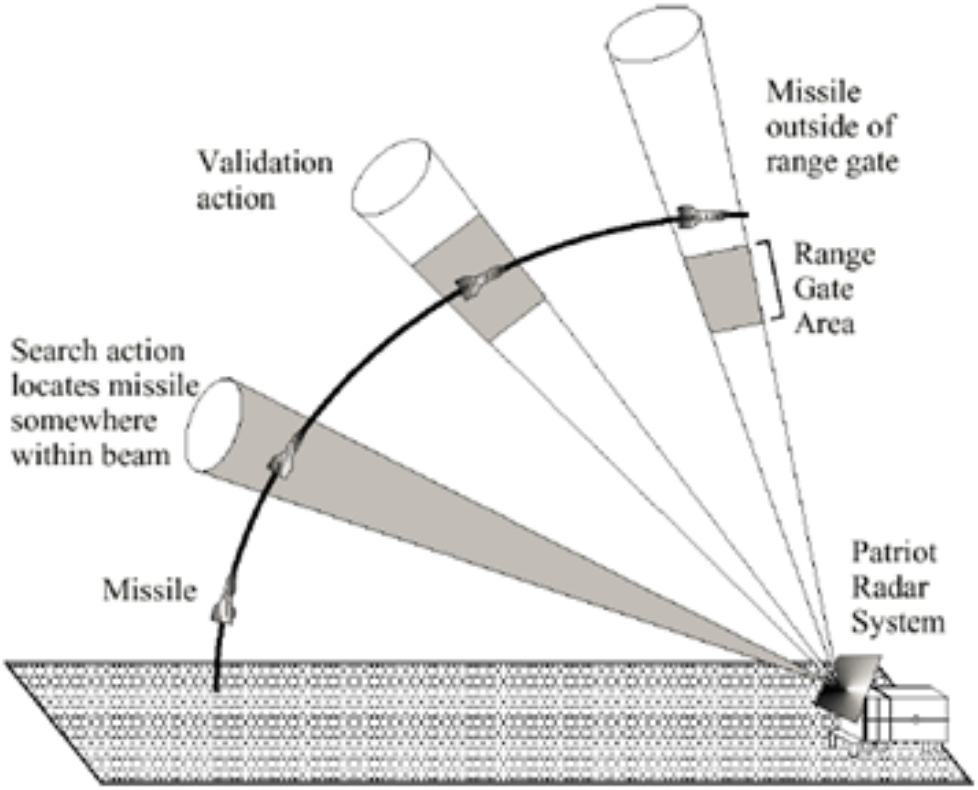
- 05324657 = $0.24657 \times 10^{3} = 246.57$
- 15555555 = $-0.55555 \times 10^{5} = -55555$
- 04925000 = $0.25000 \times 10^{-1} = 0.025000$

Normalization and formatting of floating point decimal numbers

- To maximize the precision for a given number of digits, numbers will be stored whenever possible with no leading zeros.
  - numbers are shifted left by increasing the exponent until leading zeros are eliminated when necessary.
- This process is called **normalization**.
- 0.MMMMM x 10^{EE}

Effect of Loss of Precision

According to the General Accounting Office of the U.S. Government, a loss of precision in converting 24-bit integers into 24-bit floating point numbers was responsible for the failure of a Patriot anti-missile battery.



Normalization algorithm

- 1. Provide an exponent of 0 for the number
  - if an exponent was not already specified as part of the number.
- 2. Shift the decimal point left or right by increasing or decreasing the **exponent**, respectively
  - until the decimal point is in the proper position.
- 3. Correct the precision
  - adding or discarding digits as necessary to meet the specification.
    - We discard or round any digits in excess of the specified precision by eliminating the least significant digits.
  - If the number has fewer than the specified number of digits, supply zeros at the end.

- 4. Convert the result into the desired format.
  - **Example**:
  - Convert 246.8035 into the standard format as described earlier (excess-50 and 5 digits normalized mantissa).
  - Step 1: 246.8035×10^0
  - Step 2: 0.2468035×10^3
  - Step 3: Drop the last two significant digits:
    - 0.24680×10^3
  - Step 4: 05324680
    - (0 is "+", 53 is excess-50 exponent value "3", magnitude of mantissa is 24680)
  - Example2: $1255 \times 10^{-3} \rightarrow$ 05112550

Exercise IV

- Provide the scientific notation in Exess-50 notation as: SEEMMMMM
  - Make sure apply normalization algorithm to maximize the precision.

- 1. 146.52437
- 2. 14.67
- 3. 0.001469823

Error, Range and Precision

- Represent 0.254 x10^3 in a normalized base 8 floating point format
  - a sign bit, followed by a 3-bit excess 4 exponent, followed by four base 8 digits.
    - *Step 1*: Convert to the target base.
    - .254 x 10^3 = 254_{10}. Using the remainder method
    - 254_{10}= 376 x 8^0:
      - 254/8 = 31 R 6
      - 31/8 = 3 R 7
      - 3/8 = 0 R 3

- Step 2: Normalize: $376 \times 8^0 = 0.376 \times 8^3$.
- Step 3: Fill in the bit fields, with a positive sign (sign bit = 0), an exponent of 3 + 4 = 7 (excess 4), and 4-digit fraction = .3760:
  - The result is: 073760 i.e. in a bit (binary) representation
  - 0 111 011 111 110 000
  - Let **b** be the base, **s** number of significant digits of the floating point number, **M** the largest exponent value, and **m** the smallest exponent value.
  - In the previous example, we have the base **b** = 8, the number of significant digits in the fraction **s** = 4, the largest exponent value **M** = 3, and the smallest exponent value **m** = -4.

- We will assume a bit pattern of 0 000 000 000 000 000 represents 0.
- Using **b**, **s**, **M**, and **m**, we would like to characterize this floating point representation:
  - the largest positive representable number,
  - the smallest (nonzero) positive representable number
  - the smallest gap between two successive numbers,
  - the largest gap between two successive numbers, and the total number of numbers that can be represented.

- Largest representable number:

$$b^{M}(1 - b^{-s}) = 8^{3}(1 - 8^{-4})$$

- Smallest positive representable number:

$$b^{m-1} = 8^{-4-1} = 8^{-5}$$

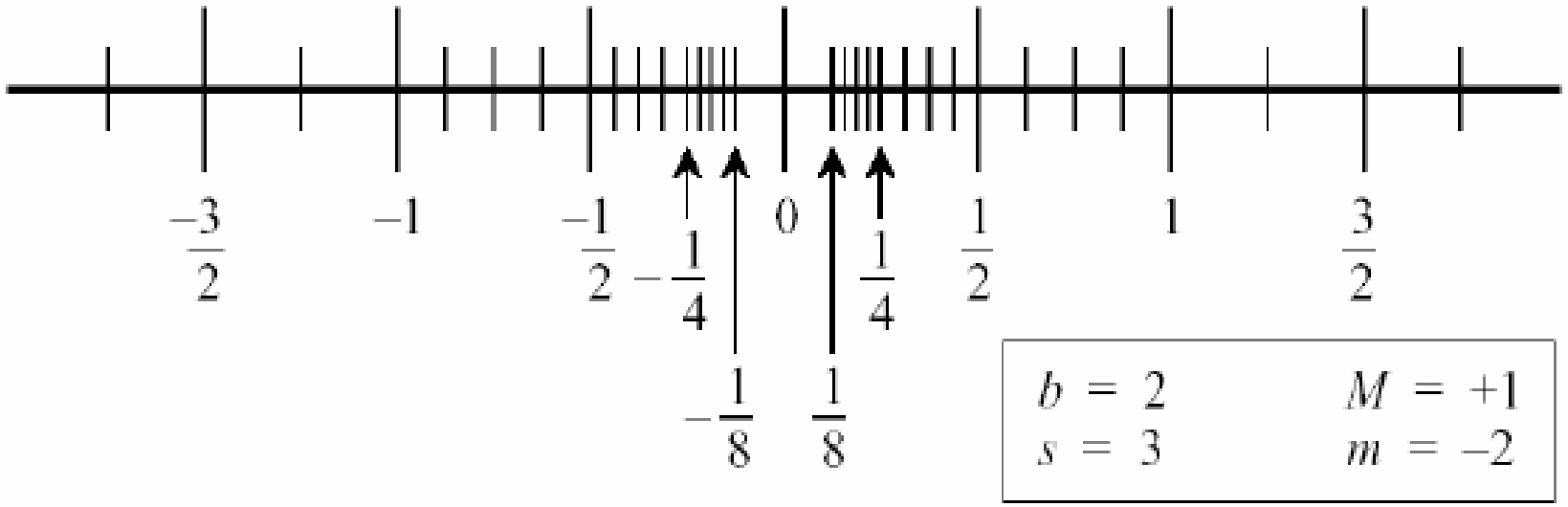
- Largest gap:

$$b^{M-s} = 8^{3-4} = 8^{-1}$$

- Smallest gap:

$$b^{m-s} = 8^{-4-4} = 8^{-8}$$

Example

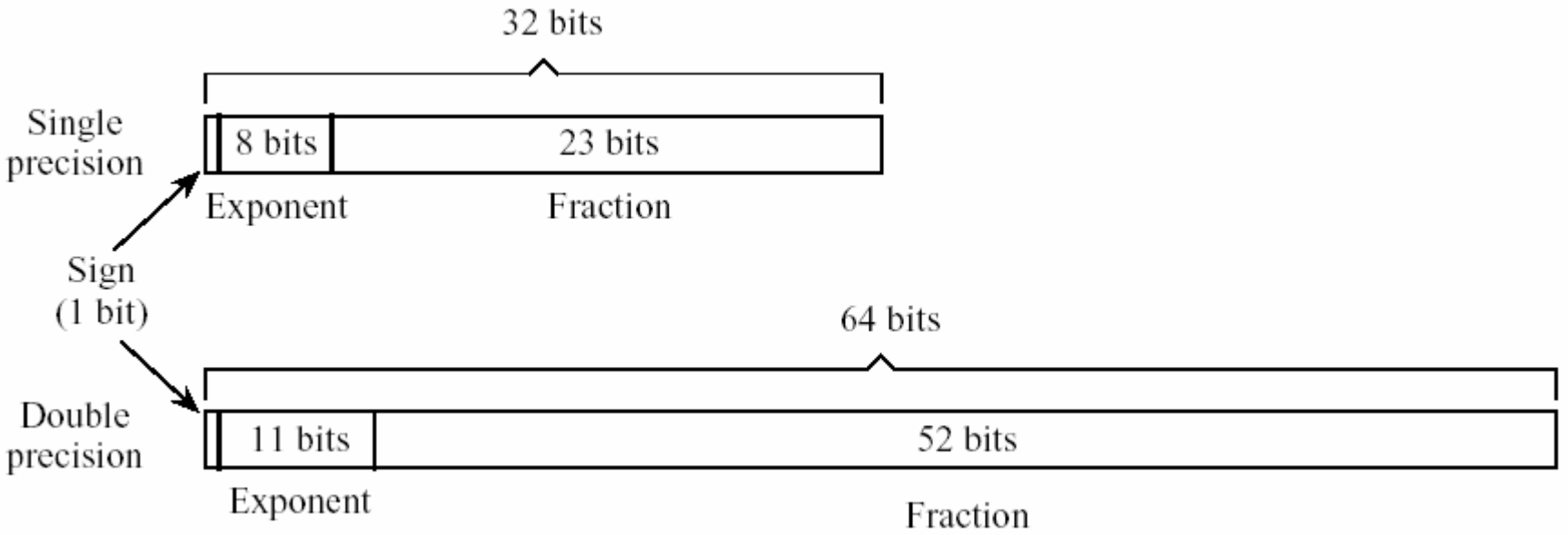


- Smallest positive number is $b^{m-1} = 2^{-3} = 1/8$
- Largest number is $b^{M}(1 - b^{-s}) = 2^{1}(1 - 2^{-3}) = 2(7/8) = 7/4$
- Smallest gap is $b^{m-s} = 2^{-2-3} = 2^{-5} = 1/32$
- Largest gap is $b^{M-s} = 2^{1-3} = 2^{-2} = 1/4$
- Number of representable numbers is 33. Prove this.

The IEEE 754 Floating Point Standard

- In 1985 IEEE developed the IEEE 754 floating-point standard for binary numbers.
  - Reason: To improve software portability and assure uniform accuracy of floating point calculations.
- Most personal computers conform to this standard
  - including IBM-PCs and the Apple Macintosh.
  - Sun UltraSparc systems
    - also support the standard, and include an additional 128-bit format.

- There are two primary formats in the IEEE 754 standard
  - single precision
  - double precision.
- The single precision format occupies 32 bits, and double precision format 64 bits.



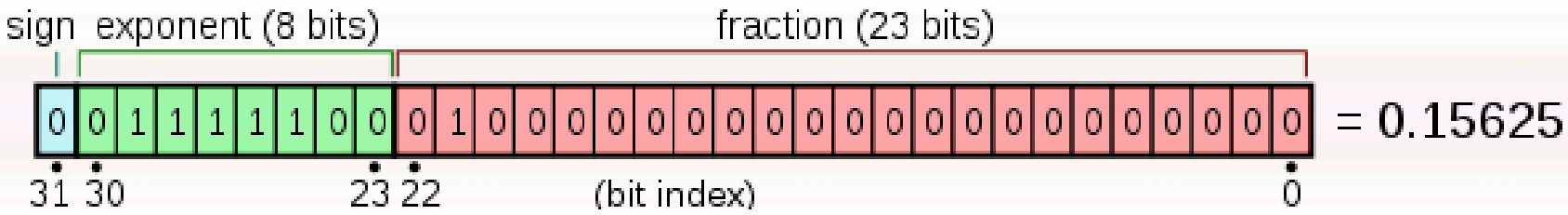
The IEEE-754 standard

- base 2 is a common way of storing floating point values (decimal fractions) in computers

$$(M)_2 * 2^N$$

- M (mantissa in binary) and
- N (exponent in binary) are stored separately as bit strings

Example


sign
exponent (8 bits)
fraction (23 bits)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
= 0.15625
31 30
23 22
(bit index)
0

example

- Conversion to Base 2 Floating Point
  - convert 1173.625 (base 10) to base 2
    - **First**, convert 1173_{10} to 10010010111_2

1173	divided by 2 is	586	1
586	divided by 2 is	293	1
293	divided by 2 is	146	1
146	divided by 2 is	73	0
73	divided by 2 is	36	1
36	divided by 2 is	18	0
18	divided by 2 is	9	0
9	divided by 2 is	4	1
4	divided by 2 is	2	0
2	divided by 2 is	1	0
1	divided by 2 is	0	1

example

- Secondly, convert 0.625_{10} to a fraction in base 2
  - Instead of dividing by 2, we multiply the fraction by 2 and record the resulting integer parts (from top to bottom)
  - Result: 0.625_{10} becomes 0.101_2

Decimal	*2	Integer Part	Fraction Part
0.625	1.25	1	0.25
0.25	0.5	0	0.5
0.5	1	1	0

example

$1173_{10} = 10010010111_2$

$1173.625 = 10010010111.101_2$

Converting to IEEE 754 single precision:

1. apply a normalization step:

$10010010111.101_2 = 1.0010010111101 * 2^{10}$

2. Find the binary representation of the exponent:

   we use excess -127 notation for representing exponent: $127+10=137 = 1000\ 1001_2$

3. put everything together:

0 1000 1001 0010010111101….

- The sign is in the leftmost position. It is only one bit: 0 for plus (+), and 1 for minus (-).
- For single precision, the 8-bit excess **127** exponent follows with 00000000 and 11111111 reserved for special cases
  - the exponential range is restricted from 2^{-126} to 2^{127}. The 23 bit base 2 fraction part (mantissa) follows.
- a ”hidden” bit 1 to the left of the binary point
  - which when taken together with the fraction part forms 24 bit significand of form 1.ffff… where ffff…is bit pattern stored in 23-bit fractional part.

IEEE-754 Standard: Single Precision

- The exponent field is 8-bit, so from 0 to 255
  - Special values: 0 means floating value is 0
  - 255 means it is infinity (Not a Number).
  - Exponent field stores the actual exponent plus offset of 127 (excess-127)
    - 127_{10} is 1111111_2
    - Therefore representation can be from 1 to 254 (because we exclude 0 and 255)
  - Actual value from -126 to 127.

Normalization for Single Precision

- How to normalize into $(1.x)_{binary} * 2^N$?
- Exercise A: $(1)_{10} = (1.0)_2$

  $= (1.0)_2 * 2^0$

- Exercise B: $(0.25)_{10} = (0.01)_2$

  $= (0.01)_2 * 2^0$

  Shift the dot to the right by 2 digits:

  $= (1.0)_2 * 2^{-2}$

Normalization for Single Precision

- Exercise C: $(0.375)_{10} = (0.011)_2$

$= (0.011)_2 * 2^0$

Shift the dot to the right by 2 digits:

$= (1.1)_2 * 2^{-2}$

Sign bit is 0 (for positive)

Exponent is -2, plus offset 127 is 125_{10}

Fraction bits (after the leading 1) are

10000000000000000000000

Normalization for Single Precision

- Exercise D: $(-12.375)_{10}$

  $= (-1100.011)_2 * 2^0$

  Shift the dot to the left by 3 digits:

  $= (-1.100011)_2 * 2^3$

  Sign bit is 1 (for negative)

  Exponent is 3, plus offset 127 is 130_{10}

  Fraction bits (after the leading 1) are

  10001100000000000000000

Normalization for Single Precision

- Exercise E: $(1100.011)_2$

  $= (1100.011)_2 * 2^0$

  Shift the dot to the left by 3 digits:

  $= (1.100011)_2 * 2^3$

  Sign bit is 0 (for positive)

  Exponent is 3, plus offset 127 is 130_{10}

  Fraction bits (after the leading 1) are

  10001100000000000000000

Normalization for Single Precision:
Full Example 1

- How to store 178.125_{10} in IEEE 754?
  - 178_{10} is 10110010 and 0.125_{10} is 0.001_2

    $178.125 = 10110010.001_2$
  - $10110010.001_2 = 1.0110010001 * 2^7$
    - Ignore/discard the '1' to the left of binary point
  - Need to offset the exponent (which is 7) by 127, therefore $7+127 = 134_{10}$ or 10000110_2

Result:

01000011001100100010000000000

IEEE 754 Single Precision:
Example 1

- **01000011001100100010000000000000**
  is stored in four bytes (using Little Endian!)

Byte	n	n+1	n+2	n+3
Binary	0000 0000	0010 0000	0011 0010	0100 0011
Hex	00	10	19	43

IEEE 754 Single Precision:
Full Example 2

- How to store 1.625_{10} in IEEE 754?
  - 1.625_{10} is $1.101_2 * 2^0$
  - Value is positive: so leading bit is 0
  - Need to offset the exponent (which is 0) by 127, therefore $0+127 = 127_{10}$ or 01111111_2
  - Ignore/discard the '1' to the left of binary point
  - Adding trailing zeros to 101, for a total of 23 digits in the mantissa field: 10100000000000000000000

Result: 00111111101000000000000000000000

IEEE 754 Single Precision:
Full Example 2

- **00111111 11010000 00000000 00000000**
  - is stored in four bytes (using Little Endian)

Byte	n	n+1	n+2	n+3
Binary	0000 0000	0000 0000	1101 0000	0011 1111
Hex	00	00	D0	3F

- Single precision format
  - allows approximately seven significant digits and an approximate value range of 10^{-38} to 10^{38}.
- For double precision
  - the 11-bit exponent is represented in excess **1023**, with 00000000000 and 11111111111 reserved for special cases.
  - The double precision format also uses a ”hidden” bit 1 to the left of binary point which supports 53-bit significand.
  - The double precision format supports approximately 15 significant decimal digits and a range of more than 10^{-300} to 10^{300}.
- For both formats, the number is normalized, unless denormalized numbers are supported.

- There are five basic types of numbers that can be represented:
  - **Nonzero normalized numbers**.
  - ”**Clean-zero**” represented by the reserved pattern 00000000 for single precision (or 0000000000 in case of double precision) in the exponent, and all zeros in the fraction (mantissa).
- The **sign** bit can be 0 or 1, and so there are two representations for zero.

- **Infinity** has representation in which
  - the exponent contains the reserved pattern 11111111 (or 11111111111 for double precision)
  - the fraction contains all zeros and the sign bit is 0 or 1.
  - Infinity is useful in handling overflow situations or in giving a valid interpretation to a nonzero number divided by zero.
- If zero is divided by zero or infinity is divided by infinity then the result is undefined and such situations are represented
  - by the **NaN** (not a number) format in which the exponent contains the reserved bit pattern 11111111 (or 11111111111 for double precision), fraction part is nonzero, and sign is 0 or 1.

Examples of IEEE 754 floating point numbers

	Value	Bit Pattern		
		Sign	Exponent	Fraction
(a)	$+1.101 \times 2^{5}$	0	1000 0100	101 0000 0000 0000 0000 0000
(b)	-1.01011×2^{-126}	1	0000 0001	010 1100 0000 0000 0000 0000
(c)	$+1.0 \times 2^{127}$	0	1111 1110	000 0000 0000 0000 0000 0000
(d)	$+0$	0	0000 0000	000 0000 0000 0000 0000 0000
(e)	-0	1	0000 0000	000 0000 0000 0000 0000 0000
(f)	$+\infty$	0	1111 1111	000 0000 0000 0000 0000 0000
(g)	$+2^{-128}$	0	0000 0000	010 0000 0000 0000 0000 0000
(h)	+NaN	0	1111 1111	011 0111 0000 0000 0000 0000
(i)	$+2^{-128}$	0	011 0111 1111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

- As with all normalized representations, there is a large gap between zero and the first representable number.
- The **denormalized**, ”dirty zero” representation
  - allows numbers in this gap to be represented.
- The sign can be 0 or 1,
- the exponent contains the reserved bit pattern 000…0 that represents -126 for single or -1022 for double precision, and the fraction contains the actual bit pattern for the actual magnitude of the number. There is no hidden 1 for this format.
  - Note that the denormalized representation is not the same as unnormalized representation.

- There are four **rounding modes** in the IEEE 754 standard
  - One mode rounds to 0,
  - another rounds towards +∞, and
  - other rounds towards -∞.
  - The default mode rounds to the nearest representable number.
  - Halfway cases round to the number whose low-order digit is even.
    - For example, 1.01101 rounds to 1.0110 and 1.01111 rounds to 1.1000

Inclass Exercise V

- 1. Convert 6.28 to IEEE 754 single precision format
- 2. Convert 314.628 to IEEE 754 single precision format

IEEE 754 single/double precision format

Maximum:

3.402823 ×10000000000 0000000000 0000000000 00000000

Minimum:
0.0000000000 0000000000 0000000000 00000003402823

float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308

Inclass Exercise V

- How to convert and store the following floating point value in IEEE 754 in four bytes with both Little Endian and Big Endian:
  - 1. 78.125_{10} ?
  - 2. 2.625_{10} ?
  - 3. 0.002625_{10} ?
  - 4. -78.125_{10}?
  - 5. -2.625_{10} ?
  - 6. -0.002625_{10} ?

Alphanumeric Data

- The data entered as characters, number digits, and punctuation is known as ***alphanumeric data***.
- **ASCII** (developed by ANSI) is defined as 7-bit code but also has 8-bit extensions (Latin-I)
- **EBCDIC** 8-bit code
- **UNICODE** 16-bit code. Includes ASCII as its subset. Unicode can represent 65,536 characters, and about 49,000 have already been defined.
  - An additional 6,400 characters are reserved for private use, leaving approximately 10,000 available for further expansion.

Unicode

- 00
  - FF Latin-I (ASCII)
- FF
  - 2000 General character alphabets: Latin, Cyrillic, Greek, Hebrew, Arabic, Thai, etc.
- 2000-3000
  - Symbols: punctuation, math, technical, geometric shapes, etc.
- 3000-33FF
  - Miscellaneous punctuations, symbols, and phonetics for Chinese, Japanese, and Korean
- 33FF-4E00
  - Unassigned
- 4E00-9FFF
  - Chinese, Japanese, Korean ideograms
- A000-AC00
  - Unassigned
- AC00-D7AF
  - Korean Hangui syllables
- D7AF–E000
  - Space for surrogates
- E000-F8FF
  - Private use
- F8FF-FFFF
  - Various special characters

ASCII Character Code

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

- ASCII is a 7-bit code, commonly stored in 8-bit bytes.
- • "D" is at 44_{16}. To convert upper case letters to lower case letters, add 20_{16}. Thus "d" is at $44_{16} + 20_{16} = 64_{16}$.
- • The character "5" at position 35_{16} is different than the number 5.

Unicode Character Code

0000	NUL	0020	SP	0040	@	0060	`	0080	Ctrl	00A0	NBS	00C0	À	00E0	à
0001	SOH	0021	!	0041	A	0061	a	0081	Ctrl	00A1	¡	00C1	Á	00E1	á
0002	STX	0022	"	0042	B	0062	b	0082	Ctrl	00A2	¢	00C2	Â	00E2	â
0003	ETX	0023	#	0043	C	0063	c	0083	Ctrl	00A3	£	00C3	Ã	00E3	ã
0004	EOT	0024	$	0044	D	0064	d	0084	Ctrl	00A4	¤	00C4	Ä	00E4	ä
0005	ENQ	0025	%	0045	E	0065	e	0085	Ctrl	00A5	¥	00C5	Å	00E5	å
0006	ACK	0026	&	0046	F	0066	f	0086	Ctrl	00A6	¦	00C6	Æ	00E6	æ
0007	BEL	0027	'	0047	G	0067	g	0087	Ctrl	00A7	§	00C7	Ç	00E7	ç
0008	BS	0028	(	0048	H	0068	h	0088	Ctrl	00A8	¨	00C8	È	00E8	è
0009	HT	0029	)	0049	I	0069	i	0089	Ctrl	00A9	©	00C9	É	00E9	é
000A	LF	002A	*	004A	J	006A	j	008A	Ctrl	00AA	ª	00CA	Ê	00EA	ê
000B	VT	002B	+	004B	K	006B	k	008B	Ctrl	00AB	«	00CB	Ë	00EB	ë
000C	FF	002C	´	004C	L	006C	l	008C	Ctrl	00AC	¬	00CC	Ì	00EC	ì
000D	CR	002D	-	004D	M	006D	m	008D	Ctrl	00AD	–	00CD	Í	00ED	í
000E	SO	002E	.	004E	N	006E	n	008E	Ctrl	00AE	®	00CE	Î	00EE	î
000F	SI	002F	/	004F	O	006F	o	008F	Ctrl	00AF	¯	00CF	Ï	00EF	ï
0010	DLE	0030	0	0050	P	0070	p	0090	Ctrl	00B0	°	00D0	Ð	00F0	¶
0011	DC1	0031	1	0051	Q	0071	q	0091	Ctrl	00B1	±	00D1	Ñ	00F1	ñ
0012	DC2	0032	2	0052	R	0072	r	0092	Ctrl	00B2	²	00D2	Ò	00F2	ò
0013	DC3	0033	3	0053	S	0073	s	0093	Ctrl	00B3	³	00D3	Ó	00F3	ó
0014	DC4	0034	4	0054	T	0074	t	0094	Ctrl	00B4	´	00D4	Ô	00F4	ô
0015	NAK	0035	5	0055	U	0075	u	0095	Ctrl	00B5	µ	00D5	Õ	00F5	õ
0016	SYN	0036	6	0056	V	0076	v	0096	Ctrl	00B6	¶	00D6	Ö	00F6	ö
0017	ETB	0037	7	0057	W	0077	w	0097	Ctrl	00B7	·	00D7	×	00F7	÷
0018	CAN	0038	8	0058	X	0078	x	0098	Ctrl	00B8	¸	00D8	Ø	00F8	ø
0019	EM	0039	9	0059	Y	0079	y	0099	Ctrl	00B9	¹	00D9	Ù	00F9	ù
001A	SUB	003A	:	005A	Z	007A	z	009A	Ctrl	00BA	º	00DA	Ú	00FA	ú
001B	ESC	003B	;	005B	[	007B	{	009B	Ctrl	00BB	»	00DB	Û	00FB	û
001C	FS	003C	<	005C	\	007C	|	009C	Ctrl	00BC	1/4	00DC	Ü	00FC	ü
001D	GS	003D	=	005D	]	007D	}	009D	Ctrl	00BD	1/2	00DD	Ý	00FD	Þ
001E	RS	003E	>	005E	^	007E	~	009E	Ctrl	00BE	3/4	00DE	ý	00FE	þ
001F	US	003F	?	005F	_	007F	DEL	009F	Ctrl	00BF	¿	00DF	ß	00FF	ÿ

NUL	Null	SOH	Start of heading	CAN	Cancel	SP	Space
STX	Start of text	EOT	End of transmission	EM	End of medium	DEL	Delete
ETX	End of text	DC1	Device control 1	SUB	Substitute	Ctrl	Control
ENQ	Enquiry	DC2	Device control 2	ESC	Escape	FF	Form feed
ACK	Acknowledge	DC3	Device control 3	FS	File separator	CR	Carriage return
BEL	Bell	DC4	Device control 4	GS	Group separator	SO	Shift out
BS	Backspace	NAK	Negative acknowledge	RS	Record separator	SI	Shift in
HT	Horizontal tab	NBS	Non-breaking space	US	Unit separator	DLE	Data link escape
LF	Line feed	ETB	End of transmission block	SYN	Synchronous idle	VT	Vertical tab

Questions?

- ???