ECE 242
Digital Computers and Computing

Lecture 3. Representation of Numbers and Characters
Spring 2021

Won-Jae Yi

Data Representations

- Programs, data stored in the memory → B, N, HEX, DEC
- Two's complement
- Binary-coded Decimal (BCD)
- Floating-point numbers → non-integer value
- Text and ASCII code

Integer Value

0 1 0 1

1/3 2/7

0.333333 . . . .

Non-negative Integers

- A non-negative integer in base r

$$N_r = d_{m-1}r^{m-1} + d_{m-2}r^{m-2} + \cdots + d_1 r + d_0 = \sum_{i=0}^{m-1} d_i r^i$$

- $324_{10} = 3\times10^2 + 2\times10^1 + 4\times10^0$
- $5A_{16} = 5\times16^1 + 10\times16^0$
- $132_8 = 1\times8^2 + 3\times8^1 + 2\times8^0$

$5A_{16} = 0101\,1010_2$

$5A_{16} = 0101\,1010_2 = 132_8$

Number System	Base r	Digits
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Binary	2	0, 1

- Hexadecimal, decimal, octal, binary number systems

Non-negative Integers

- Largest m-digit positive integer for Base *r*:

Consider $N_r = \underbrace{(r-1)\,(r-1)\,(r-1)\,\cdots\,(r-1)}_{m}$

largest possible integer = $\boxed{r^m - 1}$

- Possible number of integer for Base *r* of m-digit long number:

$\boxed{0 \sim r^m - 1}$ r^m Numbers

- $324.14_{10} = 3 * 10^2 + 2 * 10^1 + 4 * 10^0 + 1 * 10^{-1} + 4 * 10^{-2}$

$\leftarrow 1001.01\overrightarrow{0}_2$ $0 \times 2^{-1} + 1 \times 2^{-2}$

Example

Example 3.2

The first example showed that the largest m-digit integer for unsigned integers has the value $r^m - 1$. Thus the largest 16-bit (binary) integer

$$1111\ 1111\ 1111\ 1111_2$$

has the value

$$2^{16} - 1 = 65{,}535$$

in decimal representation. The largest 16-bit fraction

$$.1111\ 1111\ 1111\ 1111_2$$

has the decimal value

$$2^{-16} \times 65{,}535 = 0.99998474.$$

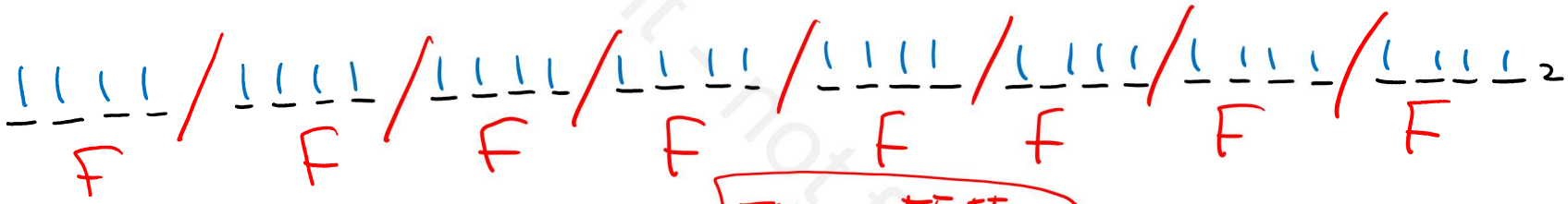
This is obtained by scaling the 16-bit fraction as

$$2^{-16} \times (2^{16} - 1) = 1 - 2^{-16}$$

and performing the arithmetic on a calculator with a sufficient number of decimal places.

Example

- Compute the largest integer representable in a 32-bit computer word



$4,294,967,295_{10}$ ≃ 32-bit word computer → possible max. address 2^{32} bytes ≃ 4 GB

Signed Numbers

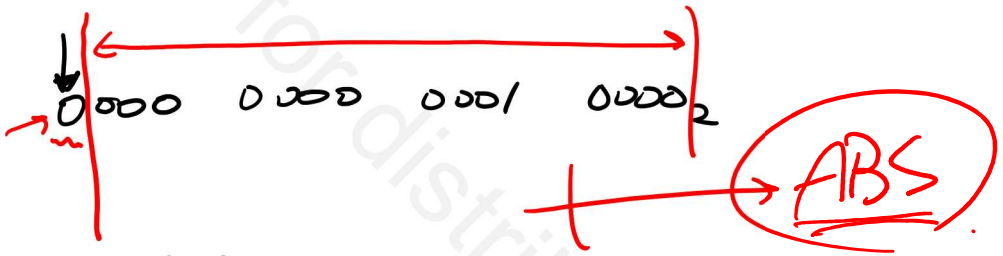
MSB LSB

- Sign-Magnitude Representation
  - MSB (most significant bit; leftmost) indicates the sign of the number
    - Sign bit = 0 for positive, r-1 for negative (where r is the Base r)

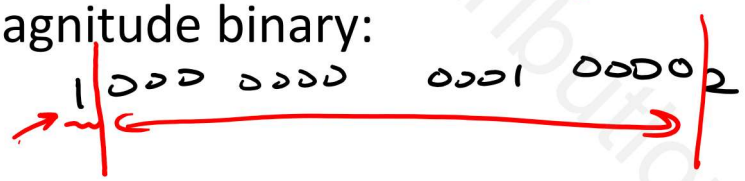
$$N_r = (d_{m-1}d_{m-2}\cdots d_1d_0)_r \qquad d_{m-1} = \begin{cases} 0 & \text{if } N_r \geq 0 \\ r-1 & \text{if } N_r < 0 \end{cases}$$

Assume 16bit wide data

- 16_{10} in binary: 0000 0000 0001 0000$_2$



- -16_{10} in sign-magnitude binary:



Signed Numbers

- Complement Representation
  - Most microprocessors' arithmetic instructions on negative numbers use a **complement** number system
  - Positive numbers have the same representation
  - Negative numbers in form of complement number
    - Radix-complement: $N'_r = r^m - N_r$ (2's, 10's complement)
    - Diminished radix-complement = $N'_r = r^m - N_r - 1$ (1's, 9's complement)
- Advantage over sign-magnitude notation
  - Sign digit doesn't have to be treated in a special way (addition/subtraction)
  - Simplifies the arithmetic circuits of the CPU

complement bits +1

complement bits
+0, -0

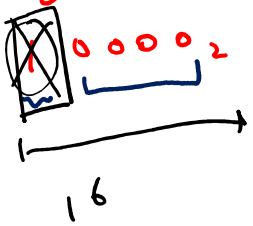
require extra HW or circuit to determine the sign bit.

Signed Numbers

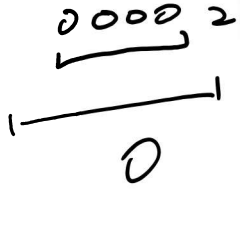
32bit
+ 32 bit
XXX

- In machine computations, *m*-digit number can only represent *m*-digit number.
- If any operation's result that requires more than m digits, then higher-order digit is ignored → **Overflow** (out of range)

e.g.) Calculation result was 16 for a 4bit wide ↕ 0 for a 4bit wide



Vs



16

0

↳ flag somewhere so it knows it's an overflow.

Value	One's	Two's	Value	Nine's	Ten's
7	0111	0111	4999	4999	4999
6	0110	0110	4998	4998	4998
5	0101	0101	•	•	•
4	0100	0100	•	•	•
3	0011	0011	•	•	•
2	0010	0010	0002	0002	0002
1	0001	0001	0001	0001	0001
0	0000	0000	0000	0000	0000
−0	1111	—	−0000	9999	—
−1	1110	1111	−0001	9998	9999
−2	1101	1110	−0002	9997	9998
−3	1100	1101	•	•	•
−4	1011	1100	•	•	•
−5	1010	1011	•	•	•
−6	1001	1010	•	•	•
−7	1000	1001	−4999	5000	5001
−8	—	1000	−5000	—	5000

Signed Numbers

$1110_2 \Rightarrow 0001+1$

$= 0010_2$

-2

- Radix complement of a number
  - Complement each digit, then add 1 to the result

4 digit system

- 2's complement of -2:

take $|-2| \Rightarrow 0010_2 \rightarrow 1101 + 1$

-2

$=$

1110_2

- 10's complement of -2:

take $|-2| \Rightarrow 2$

$9999 - 2 = 9997 + 1 = 9998$

-2

- 16's complement -2:

$FFFF - 2 = FFFD + 1 = FFFE_{16} = -2_{10}$.

Signed Numbers

- Calculate largest positive number for an *m*-digit radix-complement number in base *r*

$$\left(\frac{1}{2} \times r^m - 1\right)$$ A.

$m = 4$ $\quad 3$ $\quad 2^3 - 1 = 7$

2's complement max. value

most negative number

	most neg.	most pos.
Sign-magnitude	$-2^{m-1} + 1$	$2^{m-1} - 1$
1's comp.	$-2^{m-1} + 1$	"
2's comp.	-2^{m-1}	"

Representation Conversions

- 111110_2 to decimal:

$1\times2^5 + 1\times2^4 + \dots \Rightarrow 62_{10}$

- 0.502_8 to decimal:

$5\times8^{-1} + 0\times8^{-2} + 2\times8^{-3} \Rightarrow 0.62890625_{10}$

- 3964_{10} to octal:

8 | 3964
8 | 495 – 4
8 | 61 – 7
7 – 5

$\rightarrow 7574_8$

- 10110111.00101_2 to octal:

2 6 7 1 2 $\Rightarrow 267.12_8$

- 10110111.00101_2 to hexadecimal:

B 7 . 2 8 $\Rightarrow B7.28_{16}$

Binary-coded Decimal (BCD)

- Many microcomputers provide instructions to perform arithmetic operations on data considered on decimal numbers

BCD	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- 95_{10} to BCD: 9 → 1001, 5 → 0101
- 0001011100111001_2 to BCD: 1 7 3 9

Binary-coded Decimal (BCD)

- Packed BCD: 1 byte (8 bits) represent two BCD digits
- Unpacked BCD: 1 byte (8 bits) represent one BCD digits
- Example: 3475 in Packed BCD vs. Unpacked BCD

↳ 0011 0100 0111 0101 : Packed

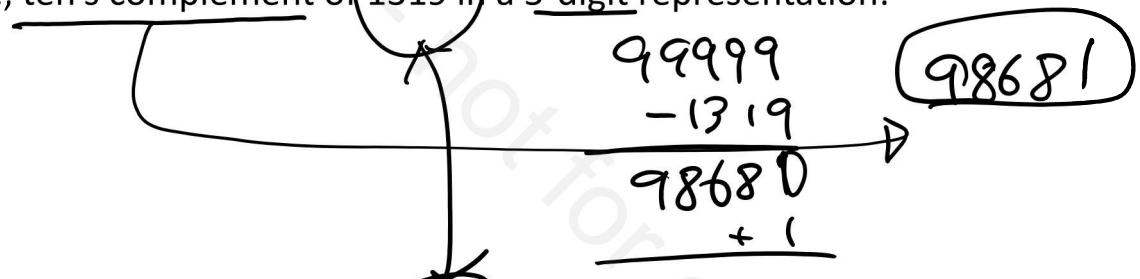
3 4 7 5

Unpacked: 0000 0011 0000 0100 0000 0111 0000 0101

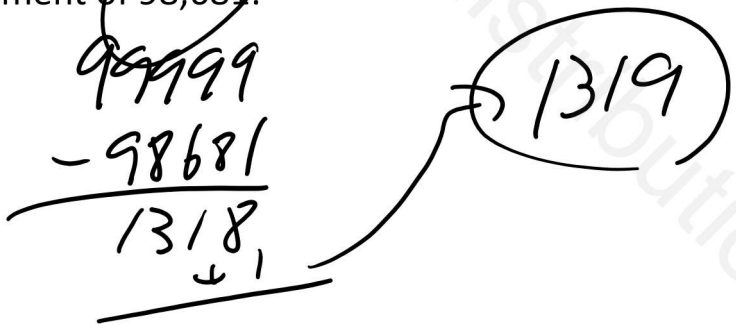
3 4 7 5

Binary-coded Decimal (BCD)

- 1010_2 to 1111_2 are not used in BCD → 10_{10} ~ 15_{10} require two digits!
- Signed BCD Integers utilize ten's complement of a decimal number.
- e.g., ten's complement of 1319 in a 5-digit representation:



- e.g., ten's complement of 98,681:



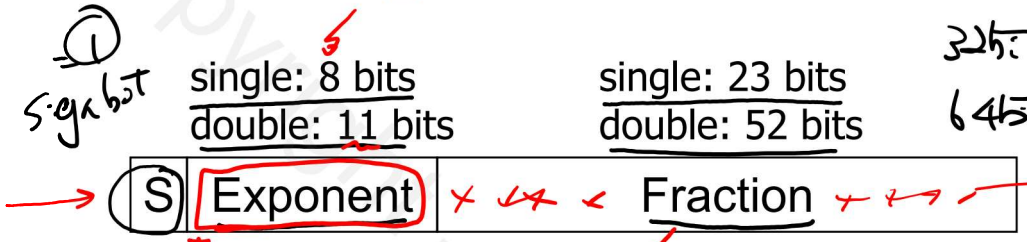
Floating-point Representations

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

Advantage of F.P. Representation -

1. Simplifies exchange of data that includes FP numbers.
2. Simplifies FP arithmetic algorithms to know that numbers will be always in this format
3. Increases accuracy of numbers that can be stored in word
   - ~~M68K : 16bits~~
   - MIPS : 32bits

IEEE Floating-Point Format



single: 8 bits
double: 11 bits

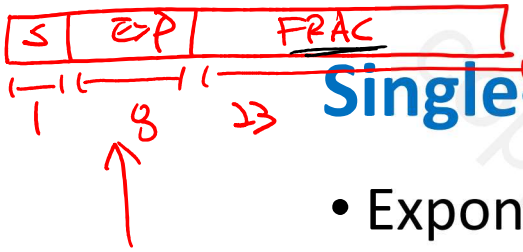
single: 23 bits
double: 52 bits

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: $1.0 \le |\text{significand}| < 2.0$
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the "1." restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1023

- overflow : exponent too large to be represented in exponent field

- underflow: exponent too small to be represented in exponent field. (negative exponent)



Single-Precision Range

- Exponents 00000000 and 11111111 reserved
- Smallest value (in terms of ABS)
  - Exponent: 00000001
    ⇒ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 ⇒ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value (in terms of ABS)
  - exponent: $11111110_2 = 254_{10}$
    ⇒ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 ⇒ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

- Exponents 0000…00 and 1111…11 reserved
- Smallest value
  - Exponent: 00000000001
    ⇒ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 ⇒ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    ⇒ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 ⇒ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Precision

- Relative precision
  - all fraction bits are significant
  - Single: approx. 2^{-23}
    - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 6$ decimal digits of precision
  - Double: approx. 2^{-52}
    - Equivalent to $52 \times \log_{10}2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Floating-Point Example

- Represent –0.75 = -0.11_2
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: –1 + 127 = 126 = 01111110_2
    - Double: –1 + 1023 = 1022 = 01111111110_2
- Single: 10111111010000…00 ⇒
- Double: 10111111111010000…00

$0.75 \times 2 = 1.5 \quad -1$

$0.5 \times 2 = 1.0 \quad -1$

$0 \times 2 = 0 \quad -0$

$-0.11_2 = (-1)^S \times (1 + \text{Fra}) \times 2^{(\text{exp}-\text{bias})}$

$= (-1)^1 \times (1.10\ldots) \times 2^{-1}$

$\rightarrow -1.1_2 \times 2^{-1}$

$-1 = \text{exp} - \text{bias}$

$= \text{exp} - 127$

$\therefore \text{exp} = 126$

Floating-Point Example

- What number is represented by the single-precision float

  1 10000001 01000…00

  - S = 1
  - Fraction = $01000…00_2$
  - Exponent = 10000001_2 = 129

- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

$(-1)^1 \times (1 + 0.0100\ldots0) \times 2^{(EXP - BIAS)}$

$= (-1)^1 \times (1.01_2) \times 2^2$

$0.01_2 = 0.25_{10}$

ASCII Code

- American Standard Code for Information Interchange Code
- Allowing human-readable input/output data to be manipulated internally by the computer
- Numbers, letters, special characters recognized by the assembler
- Encode/decode based on the ASCII Table

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

ASCII Code

Space included

- Convert "Motorola MC68000" to ASCII Code (in HEX)

4D 6F 74 6F 72 6F 6C 61 20 4D 43 36 38 30 30 30

6 8 0 0 0

+ 30 30 30 30 30

36 38 30 30 30

Ø .