CS2100 – L03 – Data Representation & Number Systems

Week 1 + 2

3.1 – Number Systems

3.2 – Data Representation

3.3 – Integer Representations

- Sign and Magnitude
- Complement
- Excess

3.4 – Floating Point Representation

- IEEE 754 Representation

3.5 – Character Representation

- ASCII Code
- Unicode

3.1 — Number Systems

Decimal to Binary Conversion

① Repeated division-by-2 (round numbers : use for > base-2)

② Sum of weights : easy for base-2, hard for others

Method 1 - Repeated division-by-2

Whole/round numbers

- Use successive division by 2 until the **quotient** is 0:
- The ***remainders*** form the answer:
  - The first remainder as the *least significant bit (LSB)*
  - The last as the *most significant bit (MSB)*.

2	43	
2	21 rem 1	← LSB
2	10 rem 1	
2	5 rem 0	
2	2 rem 1	
2	1 rem 0	
	0 rem 1	← MSB

$(43)_{10} = (101011)_2$

Fractions

- Use repeated multiplication by 2
- → Until the fractional product is 0
- The carried digits (carries) produce the answer
  - The first carry as the MSB, and the last as the LSB

	Carry	
0.3125×2=0.625	0	←MSB
0.625×2=1.25	1	
0.25×2=0.50	0	
0.5×2=1.00	1	←LSB

$(0.3125)_{10} = (.0101)_2$

Method 2 - Sum of weights

- $213_{10} = 1101\ 0101_2$

85 21

128	64	32	16	8	4	2	1
1	1	0	1	0	1	0	1

$$213 - 128 = 85 \qquad 85 - 64 = 21 \qquad \ldots$$

Base - K to base - J conversion

- Use decimal system as bridge
- Base - K → base - 10 → base - J

Shortcuts - base - 2 → base - 8

Binary	Octal
1010111010	1272_8
1111101100	1754_8
1011000001	1301_8
1000111110	1076_8
1001000101	1105_8
1101101010	1552_8

every 3 digits

Shortcuts - base-2 → base-16

Binary	Hexadecimal
1010111010	$2BA_{16}$
1111101100	$3EC_{16}$
1011000001	$2C1_{16}$
1000111110	$13E_{16}$
1001000101	245_{16}
1101101010	$36A_{16}$

every 4 digits

Shortcuts - base-8 to base-16

- base-8 ⟶ base-2 (bridge) ⟶ base-16

MUST KNOW BY HEART!!!

① Binary translation of 0 to 7

② Hexadecimal translation of 0 to 15

③ Decimal translation of A, B, C, D, E, F

④ Powers of two up to 1024 (i.e. 2^{10})

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

Decimal	Binary	Hexademical
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

3.2 — Data Representation

Common storage units

- Bit (Binary Digit) = a single '0' or '1'
- Byte = 8 Bits
  - Usually the smallest accessible unit
- Word = 4 bytes(32 bits) / 8 bytes(64 bits)
  - Platform dependent
  - Double word = 2 x Word, Halfword = word / 2

not very common

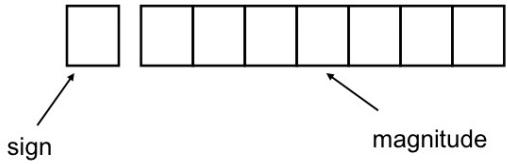
Common data types

Data type	32-bit processor	64-bit processor
int	4 bytes (32 bits)	8 bytes (64 bits)
float	4 bytes (32 bits)	8 bytes (64 bits)
double	8 bytes (64 bits)	16 bytes (128 bits)
char	1 byte (8 bit)	

3.3 — Integer Representations

Sign-and-magnitude (S&M)

- Eg: a 1-bit sign and 7-bit magnitude format



- Largest value: $01111111 = +127_{10}$
- Smallest value: $11111111 = -127_{10}$
- Zeroes: $00000000 = +0_{10}$, $10000000 = -0_{10}$
- Range: -127_{10} to $+127_{10}$ (8 bits)

- Question:
  - For an *n*-bit sign-and-magnitude representation, what is the range of values that can be represented?

$-(2^{n-1}-1)$ to $(2^{n-1}-1)$

- To negate a number, just invert the sign bit.

- Examples:
  - How to negate 00100001_{sm} (decimal 33)?
    Answer: 10100001_{sm} (decimal -33)
  - How to negate 10000101_{sm} (decimal -5)?
    Answer: 00000101_{sm} (decimal +5)

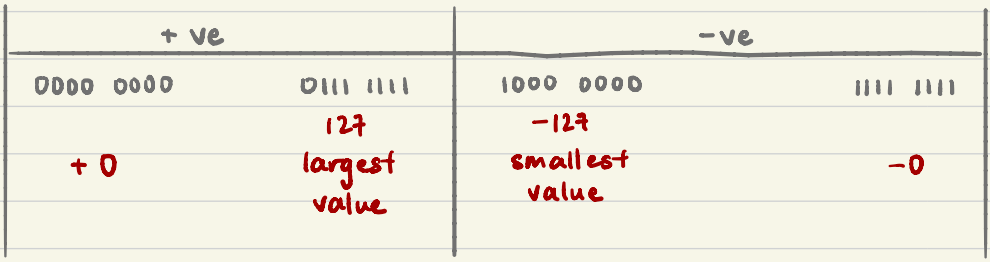
- **Question**: Sign-and-Magnitude seems intuitive, is there any drawback?

1s - complement

Binary	n-bit 1s-Complement Representation
X (i.e. positive value)	X (no change)
-X (negative value)	$2^n - X - 1$

- Question:
  - How do you tell from 1s that it is a +ve / -ve value?
  - How do we get the 1s easily for –ve value?

Look at MSB.
+ve : 0
-ve : 1
→ flip bits



- Conclusion: For n-bit 1s-Complement:

Largest Value	$2^{n-1} - 1$
Smallest Value	$-(2^{n-1} - 1)$
Zeroes	+0 -0

1s-Complement: Addition/Subtraction

- Algorithm for addition, A + B:
  1. Perform binary addition on the two numbers.
  2. If there is a carry out of the MSB, add 1 to the result.
  3. Check for overflow.
     - Overflow occurs if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:
  A – B = A + (-B)
  1. Take 1s-complement of B.
  2. Add the 1s-complement of B to A.

Overflow

- Signed numbers are of a **fixed range**
- If the result of addition/subtraction goes beyond this range, an **overflow occurs**
- Overflow can be easily detected:
  - **positive** add **positive** → **negative**
  - **negative** add **negative** → **positive**

2s - complement

Binary	n-bit 2s-Complement Representation
X (i.e. positive value)	X (no change)
-X (negative value)	$2^n - X$

- Question:
  - How do you tell from 2s that it is a +ve / -ve value?
  - How do we get the 2s easily for –ve value?

[L03 - AY2021S1]

↳ rightmost digit → copy until first 1 → flip rest

↳ Same as Is

+ ve		- ve	
0000 0000	0111 1111	1000 0000	1111 1111
	127	-128	
+ 0	largest value	smallest value	-1

- Conclusion: For n-bit 2s-Complement:

Largest Value	$2^{n-1} - 1$
Smallest Value	-2^{n-1}
Zero	0

2s-Complement: Addition/Subtraction

- Algorithm for addition, A + B:
  1. Perform binary addition on the two numbers.
  2. Ignore the carry out of the MSB.
  3. Check for overflow. Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B.

- Algorithm for subtraction, A – B:

  A – B = A + (-B)
  1. Take 2s-complement of B.
  2. Add the 2s-complement of B to A.

Excess

- Excess-K: add K to number, then represent as binary
  - K → bias / offset

- Example:

Value	4-bit Excess-8 Representation
7	7 + 8 = 15 → 1111
-8	-8 + 8 = 0 → 0000
0	0 + 8 = 8 → 1000

Relative magnitude maintained ⇒ easy comparison of magnitude

Excess Representation: 3-bit Excess-4

Decimal	Excess-4
-4	-4+4 = 0 → 000
-3	-3+4 = 1 → 001
-2	-2+4 = 2 → 010
-1	-1+4 = 3 → 011
0	0+4 = 4 → 100
1	1+4 = 5 → 101
2	2+4 = 6 → 110
3	3+4 = 7 → 111

- Question:
  - Can we use 3-bit **Excess-2**?
  - How about 3-bit **Excess-7**?

  } Yes. It's just that range will be different.

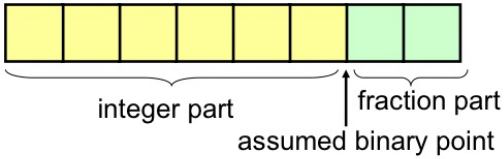
  - If we want roughly even range for +ve and –ve numbers, what's the best bias to use?

2^{n-1}

3.4 – Floating Point Representation

Fixed point representation

- In general, the binary point may be assumed to be at any pre-fixed location.
  - Example: Two fractional bits are assumed as shown below.



  - If 2s complement is used, we can represent values like:

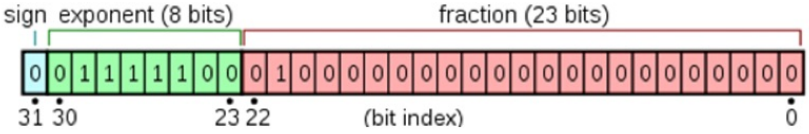
$011010.11_{2s} = 26.75_{10}$

$111110.11_{2s} = -000001.01_2 = -1.25_{10}$

IEEE 754

most common, other formats exist

- For single-precision floating point number on a 32-bit platform:



  - **sign bit**: 0 = +ve, 1 = -ve
  - **exponent**: Excess-127 → for easy magnitude comparison
  - **fraction**: ***normalized*** to 1.X and take X only
    - Also known as **mantissa**

3.5 – Character Representation

ASCII code

- Character includes:
  - Printable: ‘A’…’Z’, ‘a’…’z’, ‘0’…’9’, ‘ ’, ‘@’, ‘#’ ….
  - Unprintable: NULL, bell, tab, return, …. (control)
- Originally defined as a 7-bit sequence
  - 0 to 127: 128 characters are represented
  - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
    ↳ ASCII
- Subsequently extended to 8-bit
  - the extended range 128 to 255 can have platform dependent encoding

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

↳ row then column

- Examples:
  - Space ' ' = $32_{10} = 20_{16}$
  - ‘A’ = $65_{10} = 41_{16}$; 'a' = $97_{10} = 61_{16}$
  - '0' = $48_{10} = 30_{16}$; '9' = $57_{10} = 39_{16}$

Unicode

- Unicode replaced ASCII code to be the most commonly used representation for text

- Computing industry standard
  - maintained by Unicode Consortium
  - latest version: 8.0 (June 2015)
    - 120,000 characters across 129 scripts + symbol sets
    - Need 3-bytes per character

- For backward compatibility (among other reasons), a number of **encoding** schemes are proposed:
  - UTF-8, UTF-16, UTF-32, etc.…

essentially ASCII