Chapter 2
Binary arithmetic

Learning Objectives

- **Numerical issues and data formats.**
- **Fixed point.**
- **Fractional number.**
- **Floating point.**
- **Comparison of formats and dynamic ranges.**

Numerical Issues and Data Formats

Numerical Representation

Fixed point arithmetic:

- **N-bit (integer or fractional)**
- **Signed or unsigned.**

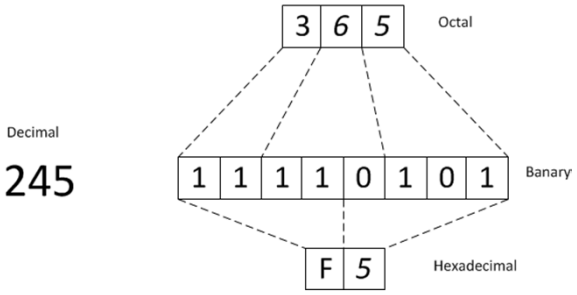
Floating point arithmetic (IEEE 754)

- **32-bit single precision.**
- **64-bit double precision.**

Reminder

Dec	Bin	Hex	Oct
0	0000	0	000
1	0001	1	001
2	0010	2	002
3	0011	3	003
4	0100	4	004
5	0101	5	005
6	0110	6	006
7	0111	7	007
8	1000	8	010
9	1001	9	011
10	1010	A	012
11	1011	B	013
12	1100	C	014
13	1101	D	015
14	1110	E	016
15	1111	F	017

Example



0xF5: 0x means Hexadecimal.

Fixed Point Arithmetic – Definition
Unsigned integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	0	
Unsigned integer numbers	0	0	0	0	0

Fixed Point Arithmetic – Definition
Unsigned integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	1	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1

Fixed Point Arithmetic – Definition
Unsigned integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	1	0	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2

Fixed Point Arithmetic – Definition
Unsigned integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	1	1	1	1	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	0	
Signed integer numbers	0	0	0	0	0

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	1	0	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

	-2^3	2^2	2^1	2^0	
Binary Number	0	1	1	1	**Decimal Equivalent**
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

	-2^3	2^2	2^1	2^0	
Binary Number	1	0	0	0	**Decimal Equivalent**
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	1	0	0	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8
	1	0	0	1	-7

Fixed Point Arithmetic – Definition
Signed integer numbers

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	1	1	1	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8
	1	0	0	1	-7
	1	0	1	0	-6
	1	0	1	1	-5
	1	1	0	0	-4
	1	1	0	1	-3
	1	1	1	0	-2
	1	1	1	1	-1

2’s complement

- Computers use this method to represent negative numbers.

In decimal we use the minus sign in front of a number to show that the number is negative. Eg – 5.

We the two’s complement as with the sign number we use the last digit to represent a negative number. However, this last bit does not represent not the sign but also this bit is weighted as shown in the previous slide with the signed integer.

Eg: The 2’s complement representation of -4 is:

For 4-bit = 1100

For 8-bit = 11111100

one’s complement

To obtain a one’s complement, all bits in a byte are inverted by changing each 1 to 0 and each 0 to 1.

Eg:

Original number	One’s complement number
1111	0000
1100	0011
1010	0101
0000	1111

Fixed Point Arithmetic - Problems

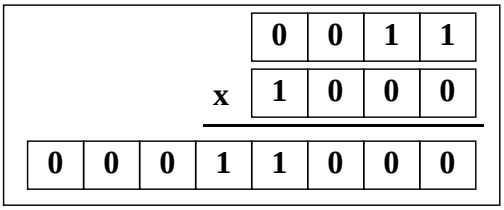
- **The following equation is the basis of many DSP algorithms (See Chapter 1):**

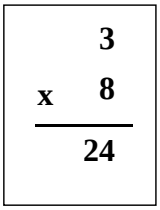
$$y(n) = \sum_{k=0}^{N-1} a(k)x(n-k)$$

- **Two problems arise when using signed and unsigned integers:**
  - **Multiplication overflow.**
  - **Addition overflow.**

Multiplication Overflow

- **16-bit x 16-bit = 32-bit**
- **Example: using 4-bit representation**

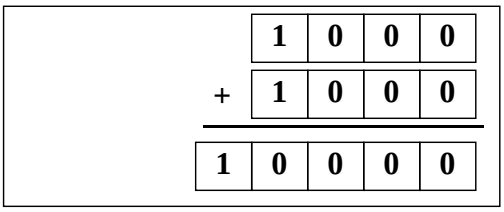


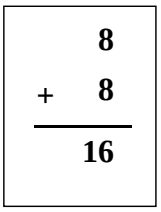


- **24 cannot be represented with 4-bits.**

Addition Overflow

- 32-bit + 32-bit = 33-bit
- Example: using 4-bit representation





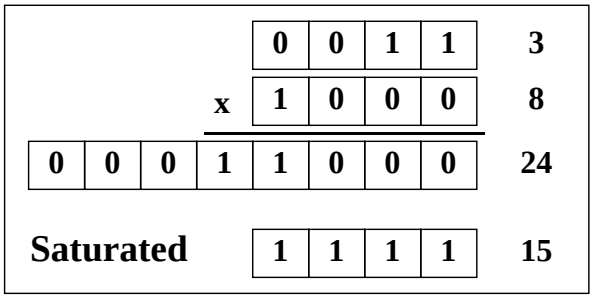
- 16 cannot be represented with 4-bits.

Fixed Point Arithmetic - Solution

- The solutions for reducing the overflow problem are:
  - Saturate the result.
  - Use double precision result.
  - Use fractional arithmetic.
  - Use floating point arithmetic.

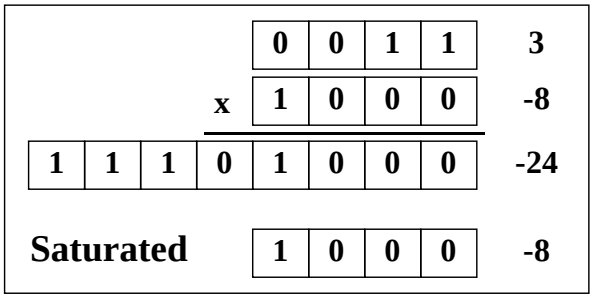
Solution - Saturate the result

- **Unsigned numbers:**
  - **If A x B ≤ 15 → result = A x B**
  - **If A x B > 15 → result = 15**



Solution - Saturate the result

- **Signed numbers:**
  - **If -8 ≤ A x B ≤ 7 → result = A x B**
  - **If A x B > 7 → result = 7**
  - **If A x B < -8 → result = -8**



Solution - Double precision result

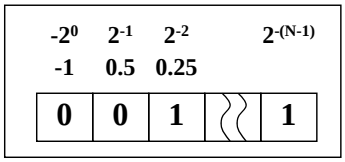
- For a 4-bit x 4-bit multiplication hold the result in an 8-bit location.
- Problems:
  - Uses more memory for storing data.
  - If the result is used in another multiplication the data needs to be represented into single precision format (e.g. prod = prod x sum).
  - Results need to be scaled down if it is to be sent to an A/D converter.

Solution - Fractional arithmetic

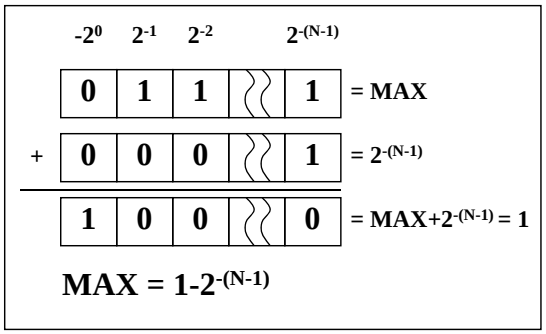
- If A and B are fractional then:
  - A x B < min(A, B)
  - i.e. The result is less than the operands hence it will never overflow.
- Examples:
  - 0.6 x 0.2 = 0.12 (0.12 < 0.6 and 0.12 < 0.2)
  - 0.9 x 0.9 = 0.81 (0.81 < 0.9)
  - 0.1 x 0.1 = 0.01 (0.01 < 0.1)

Fractional numbers

- **Definition:**

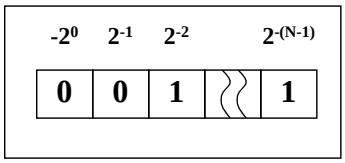


- **What is the largest number?**
- **Largest Number:**

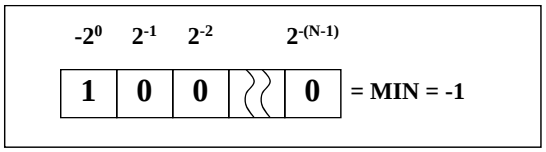


Fractional numbers

- **Definition:**

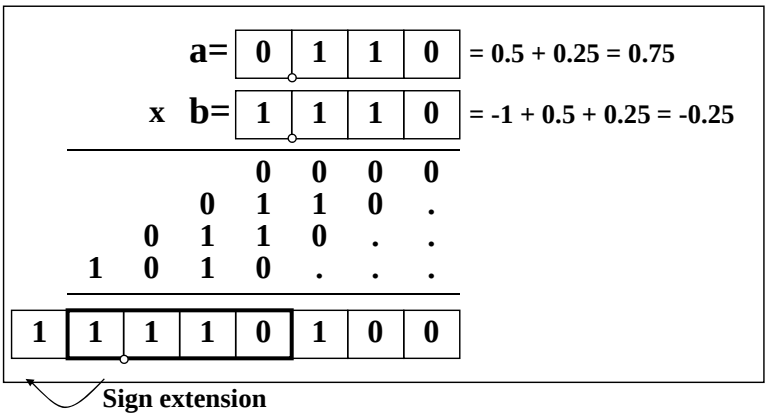


- **What is the smallest number?**
- **Smallest Number:**

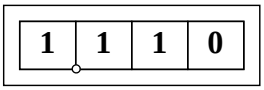


- **For 16-bit representation:**
  - **MAX = $1 - 2^{-15} = 0.999969$**
  - **MIN = -1**
  - **$-1 \leq x < 1$**

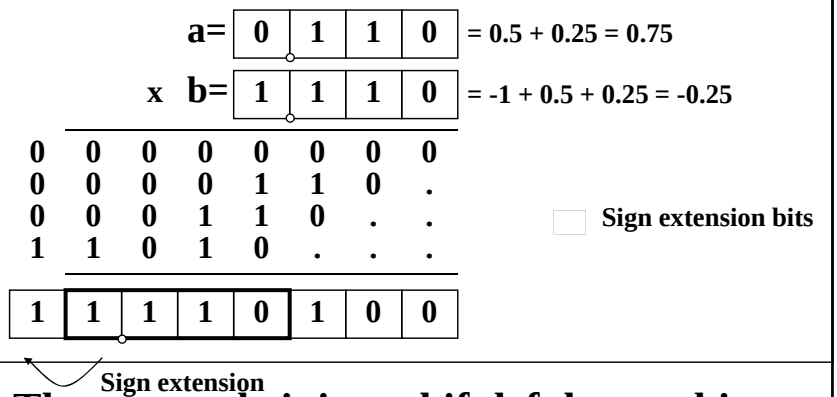
Fractional numbers - Sign Extension



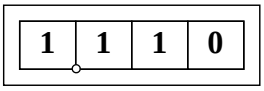
- To keep the same resolution as the operands we need to select these 4-bits:



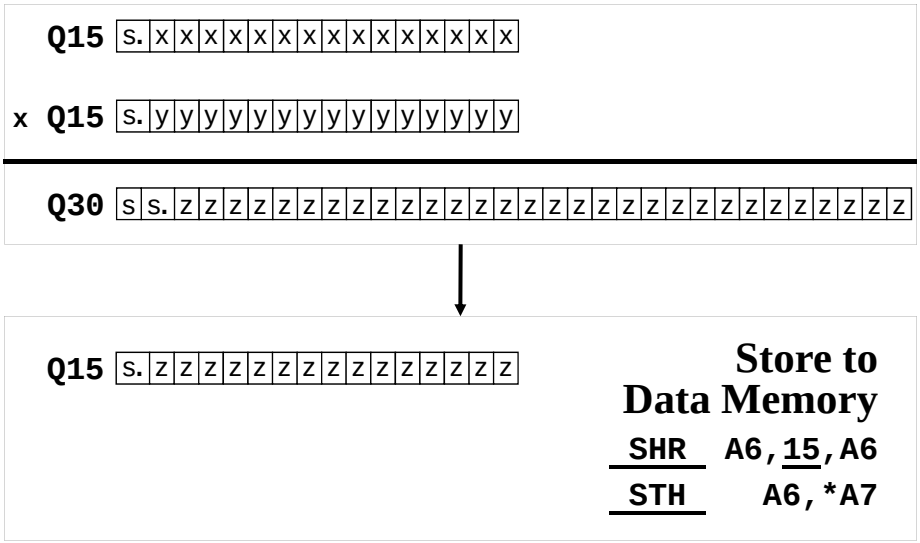
Fractional numbers - Sign Extension



- The way to do it is to shift left by one bit and store upper 4-bits or right shift by three and store the lower 4-bits:



15-bit * 15-bit Multiplication



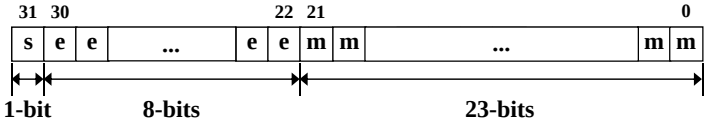
C Data Types (eg: for a DSP)

Type	Size	Representation
char, signed char	8 bits	ASCII
unsigned char	8 bits	ASCII
short	16 bits	2's complement
unsigned short	16 bits	binary
int, signed int	32 bits	2s complement
unsigned int	32 bits	binary
long, signed long	40 bits	2's complement
unsigned long	40 bits	binary
enum	32 bits	2's complement
float	32 bits	IEEE 32-bit
double	64 bits	IEEE 64-bit
long double	64 bits	IEEE 64-bit
pointers	32 bits	binary

Floating Point Arithmetic

Floating Point Arithmetic

- **The single precision format is as follows:**



s = sign bit

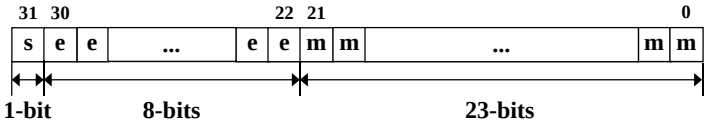
e = exponent (8-bit biased : -127)

m = mantissa (23-bit normalised fraction)

$$\text{value} = (-1)^{\text{sign}} * (1.\text{mantissa}) * 2^{(\text{exponent}-127)}$$

Floating Point Arithmetic

- **Example:**
  - **s =1**
  - **m = 0**
  - **e = 127**



value = (-1)^{sign} * (1.mantissa) * 2$^{(exponent-127)}$

value = (-1)* (1.0) * 2$^{(127-127)}$

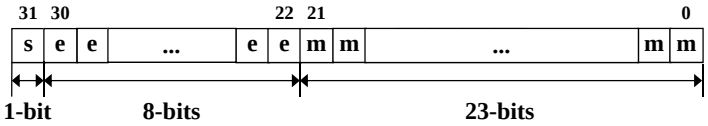
Value (decimal) = -1*1 * 1 = -1

Value (Binary)= 1 01111111 00000000000000000000000

Value=0xBF80 0000

Floating Point Arithmetic

- **Example: s =1**
  - **M= 0**
  - **E =127**



value = (-1)^{sign} * (1.mantissa) * 2$^{(exponent-127)}$

value = (-1)* (1.0) * 2$^{(127-127)}$

Value (decimal) = -1*1 * 1 = 1

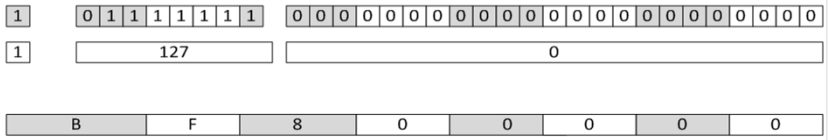
Value (Binary)= 1 01111111 00000000000000000000000

Value=0xBF80 0000

Example 1

Example 1:

Sign = 1
Exponent = 127
Mantissa = 0

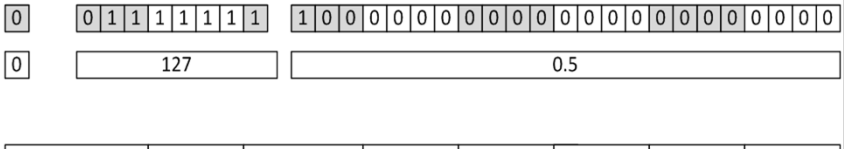


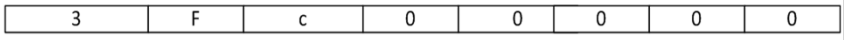
The Hexadecimal representation

$(-1)^1 * (1.0) * 2^{(127-127)} = -1$: The decimal equivalent

Example 2

Example 2:
Sign = 0
Exponent = 2^8 -1
Mantissa = 2^22





$(-1)^0 * (1.5) * 2^{(127-127)} = 1.5$

http://www.binaryconvert.com/index.html

Special Numbers for the 32-bit and 64-bit floating-point formats

Sign	Exponent (Hex)		Mantissa (Hex)	Description
	Single	Double Precision		
0	00	000	000000	Positive zero
1	00	000	000000	Negative zero
0	FF	3FF	000000	Positive infinity
1	FF	3FF	000000	Negative infinity
0	FF	3FF	≠ 000000	NaN - Not a Number

Chapter 2
Binary Arithmetic
- End -