ADVANCED ARCHITECTURE

IEEE 754

Annalisa Massini

Lecture 7

2025-2026

References

Computer Architecture - A Quantitative Approach

Hennessy Patterson – *Fifth Edition*

Appendix J – *Computer arithmetic -* *David Goldberg*

REPRESENTATION

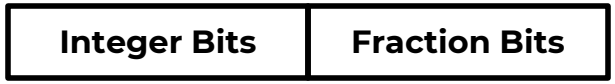
Numbers with Fractions

Two common notations:

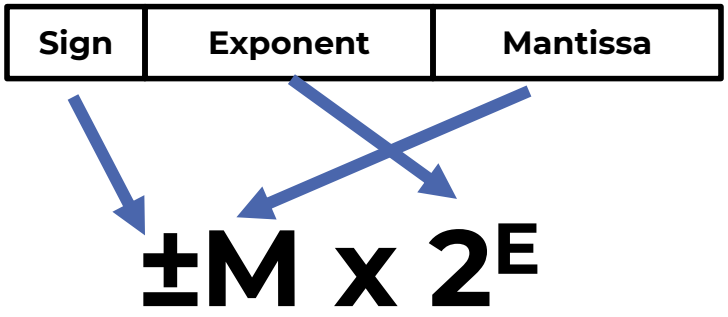
- Fixed-point: binary point fixed
  - Restricted range of values
  - Faster to compute, more power-efficient, easier to implement in hardware
- Floating-point: binary point floats to the right of the most significant 1
  - Larger range of values
  - More expensive in terms of performance and power

Numbers with Fractions

Fixed-Point



Floating-Point



Fixed-Point Numbers (Binary->Base 10)

- Using 4 integer bits and 4 fraction bits:

```
01101100
0110.1100
```

$$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6 + 0.75 = 6.75$$

- The number of integer and fraction bits must be agreed upon beforehand
- Binary point is implied

Fixed-Point Numbers (Base 10->Binary)

- 11.5_{10} to binary (4 integer bits and 4 fraction bits)
- Integer part: $11 = 2^3 + 2^1 + 2^0$ -> 1011
- Fractional part: $0.5 = 2^{-1}$ -> 1000
- $7.5_{(10)}$ = 10111000

Fixed-Point Numbers (Base 10->Binary)

Conversion of fractional part

1. Separate integer from fractional parts
2. Convert **integer** part by **divisions** by 2
3. **Multiply fractional part** *p by 2* to obtain a new value *2p*
   - If the new value $2p<1$ the next digit is 0
   - Otherwise, the next digit is 1 and the new fractional part is *p-1*
4. If $p \neq 0$, go back to point 3, otherwise we are done

Signed Fixed-Point Numbers

- Representations:
  - Sign/magnitude
  - Two's complement
- Example: Represent -7.5_{10} using 4 integer and 4 fraction bits
  - Sign/magnitude:

    11111000
  - Two's complement:

```
1. +7.5:              01111000
2. Invert bits:       10000111
3. Add 1 to lsb: +           1
                      --------
                      10001000
```

ATTENTION: 2's complement applied to the whole number (not only on the integer part)

FLOATING-POINT NUMBERS

Floating-Point Numbers

- Binary point floats to the right of the most significant 1
- Similar to decimal scientific notation

- For example, write 2730_{10} in scientific notation:

$$2730 = 2.73 \times 10^3$$

- In general, a number is written in scientific notation as:

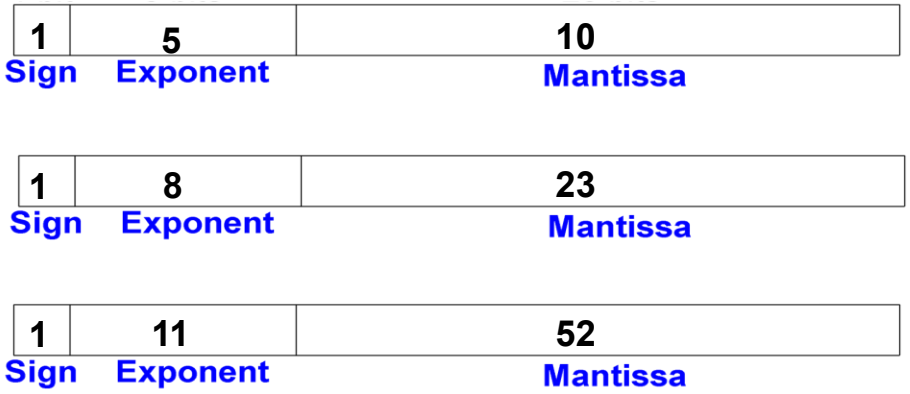
$$\pm M \times B^E$$

  - M = mantissa
  - B = base
  - E = exponent
  - In the example, M = 2.73, B = 10, and E = 3

Floating-Point Numbers

- How many bits for the exponent? How many for the mantissa?
- We describe the IEEE 754 floating-point standard starting
- We describe the representation in three steps, giving us the actual representation, passing through intermediate representations

- We have:
  - 16 bits for half precision
  - 32 bits for single precision
  - 64 bits for double precision



Example – single precision

1. Convert from base 10 to base 2 (don't reverse steps 1 & 2!):

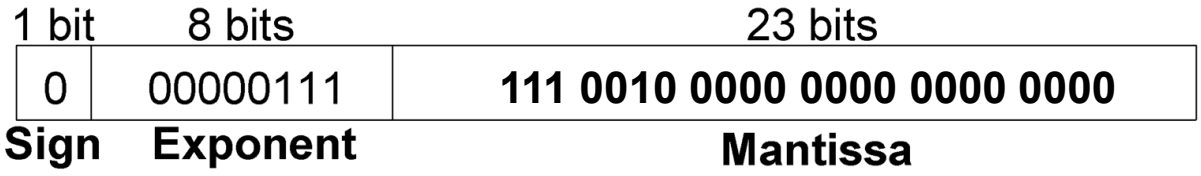
   $228_{10} = 11100100_2$

2. Write the number in "binary scientific notation":

   $11100100_2 = 1.11001_2 \times 2^7$

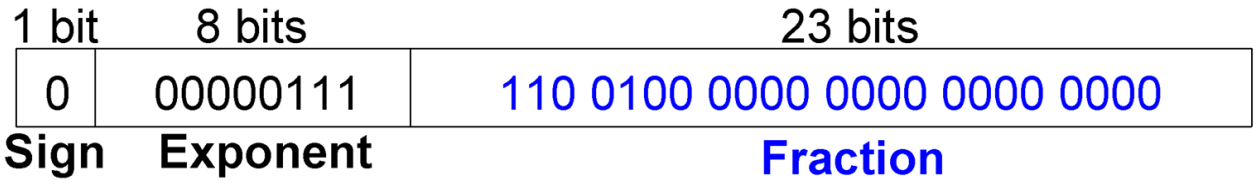
   Point floats to the right of the most significant 1

3. Fill in each field of the 32-bit floating point number:
   - The sign bit is positive (0)
   - The 8 exponent bits represent the value 7
   - The remaining 23 bits are the mantissa



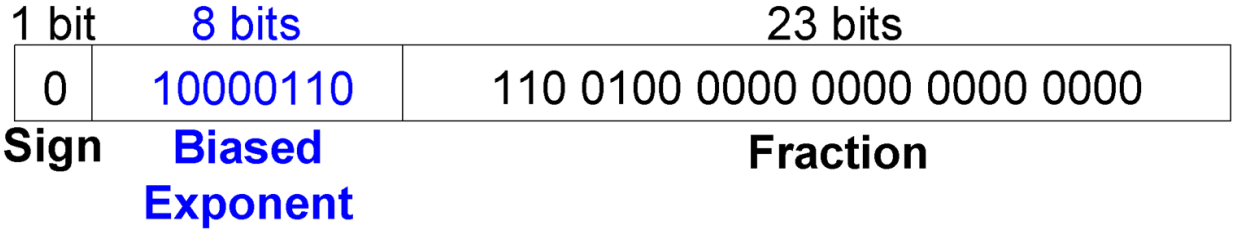
Example – single precision

- By definition, the first bit of the mantissa is always 1:
  - $228_{10} = 11100100_2 = 1.11001 \times 2^7$
- So, no need to store it: *implicit leading 1*
- Store just fraction bits in 23-bit field



Example – single precision

- Biased exponent = bias + exponent
  - Bias = 127 (01111111_2) = $2^{e-1} - 1$, where e is the exponent bits number
  - Bias is added to exponent 7, biased exponent is 127 + 7 = 134 = 10000110_2
  - Biased exponents 00000000 and 11111111 denote special cases (see later)
  - Range of biased exponent is: [00000001, 11111110]
  - The 2's complement exponent range [-126, 127] becomes [1, 254] when bias is added to the exponent
- The IEEE 754 32-bit floating-point representation of 228_{10}



in hexadecimal: 0x43640000

Biased exponent

Why biased exponent and not
2's complement or sign/magnitude?

Easier to compute which exponent is larger:

- If exponents stored as 2's complement: 010 > 101 ?
- If exponents stored as biased exponent, for example with bias=4: then 110 > 001 ?

Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

$$58.25_{10} = 111010.01_2$$

Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   $58.25_{10} = 111010.01_2$

2. Write in binary scientific notation:

   1.1101001×2^5

Example – single precision

Write -58.25_{10} in floating point (IEEE 754 single precision)

1. Convert decimal to binary:

   $58.25_{10} = 111010.01_2$

2. Write in binary scientific notation:

   1.1101001×2^5

3. Write biased exponent with 8 bits: (127 + 5) = 132 = 10000100_2

Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   $58.25_{10} = 111010.01_2$

2. Write in binary scientific notation:

   1.1101001×2^5

3. Write biased exponent with 8 bits: (127 + 5) = 132 = 10000100_2

4. Fill in fields:

   Sign bit: 1 (negative)
   8 bits exponent
   23 fraction bits: 110 1001 0000 0000 0000 0000

Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   $58.25_{10} = 111010.01_2$

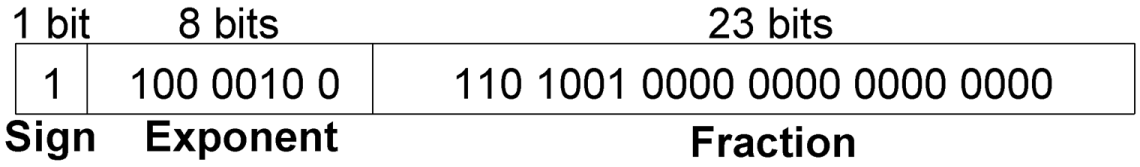
2. Write in binary scientific notation:

   1.1101001×2^5

3. Write biased exponent with 8 bits: (127 + 5) = 132 = 10000100_2

4. Fill in fields:

   Sign bit: 1 (negative)
   8 bits exponent
   23 fraction bits: 110 1001 0000 0000 0000 0000



Example – single precision

Write -58.25_{10} in floating point (IEEE 754)

1. Convert decimal to binary:

   $58.25_{10} = 111010.01_2$

2. Write in binary scientific notation:

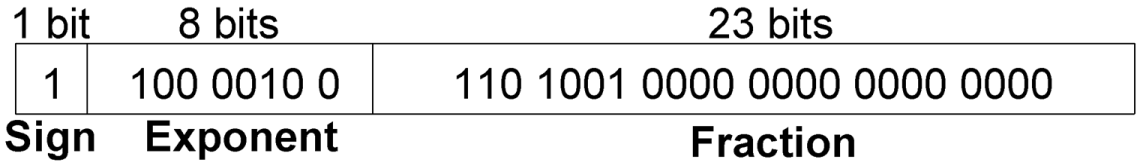
   1.1101001×2^5

3. Write biased exponent with 8 bits: (127 + 5) = 132 = 10000100_2

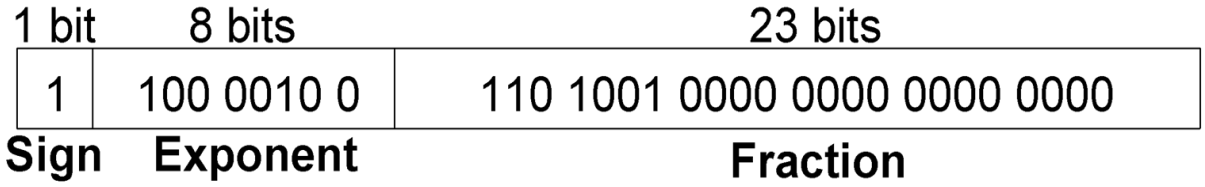
4. Fill in fields:

   Sign bit: 1 (negative)
   8 bits exponent
   23 fraction bits: 110 1001 0000 0000 0000 0000

$(-1)^S \times 2^{(E-127)} \times 1.F$



Conversion IEEE 754 floating point to decimal



$$(-1)^S \times 2^{(E-127)} \times 1.F =$$

$$= (-1)^1 \times 2^{(132-127)} \times 1.1101001 =$$

$$= -1 \times 2^5 \times 1.1101001 =$$

$$= -1 \times 11101001 \times 2^{-2} =$$

$$= -233 \times 0.25 =$$

$$= -58.25$$

IEEE 754 SPECIAL CASES

Special cases

- First bit of the mantissa is always 1:
  - $228_{10} = 11100100_2 = 1.11001 \times 2^7$
- So, no need to store it: *implicit leading 1*
- How to store 0?
- Can we represent ∞?
- Can we express irrational numbers?

IEEE 754 Floating-Point: Special Cases

Number	Sign	Exponent	Fraction
0	Any	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	Any	11111111	non-zero

«Not a Number» e.g.: √ -1

Small numbers

What is the smallest number we can represent, when having the implicit leading 1:

$2^{-126} \times 1.000...0 = 2^{-126}$

What can we do if we want to represent smaller numbers?

Denormalized numbers

How to represent very small numbers?
That is numbers very close to 0?

$$(-1)^s \times 2^{-126} \times 0.F, \text{ with } F \neq 0$$

We can represent up to:

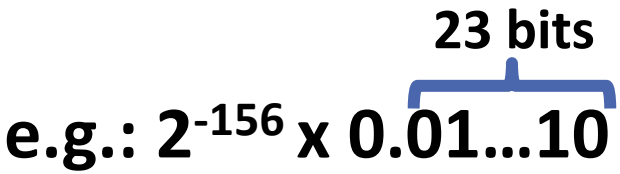
$$2^{-126} \times 0.0\ldots1 = 2^{-126} \times 2^{-23} = 2^{-149}$$

IEEE 754 Floating-Point: Special Cases

Number	Sign	Exponent	Fraction
0	Any	00000000	00000000000000000000000
∞	0	11111111	00000000000000000000000
- ∞	1	11111111	00000000000000000000000
NaN	Any	11111111	non-zero
Denormalized numbers	Any	00000000	F ≠ 0 $(-1)^s \times 2^{-126} \times 0.F$

Overflow

What if we want to represent even smaller numbers?



Overflow in IEEE 754 happens when we can not represent the exponent

Summary of IEEE 754

Precision	Num. Bits	Num. Sign Bits	Num. Exponent Bits	Num. Fraction Bits	Bias	Minimum Positive Normal Value	Maximum Value	Minimum Positive Subnormal Value
Half	16	1	5	10	15	2^{-14}	$(2-2^{-10}) \times 2^{15}$	2^{-24}
Single	32	1	8	23	127	2^{-126}	$(2-2^{-23}) \times 2^{127}$	2^{-149}
Double	64	1	11	52	1023	2^{-1022}	$(2-2^{-52}) \times 2^{1023}$	2^{-1074}

IEEE 754 ADDITION AND SUBTRACTION

Floating-Point Addition/Subtraction

- To add/subtract two floating point numbers, they must have the same exponent
- If they do not have the same exponent, the smallest is changed
- When the numbers have the same exponent, we can add/subtract the mantissas
- After the sum, we check if the result needs to be adjusted:
  - To have the implicit leading 1
  - And the correct exponent

Floating-Point Addition/Subtraction

1. Extract exponent and fraction bits
2. Prepend leading 1 to form mantissa
3. Compare exponents
4. Shift smaller mantissa if necessary
5. Add mantissas
6. Normalize mantissa and adjust exponent if necessary
7. Round result
8. Assemble exponent and fraction back into floating-point format

Example

Add the floating-point numbers A=1.5 and B=3.25 using half precision

$1.5 = 2^0+2^{-1} = 1.1 = 1.1 \times 2^0$

Exponent = 0+15 = 15 Fraction = .1

A = 0|01111|1000000000

$3.25 = 2^1+2^0+2^{-2} = 11.01 = 1.101 \times 2^1$

Exponent = 1+15 = 16 Fraction = .101

B = 0|10000|1010000000

Floating-Point Addition Example

1. Extract exponent and fraction bits
   For first number (A): E = 15, F = .1
   For second number (B): E = 16, F = .101

2. Prepend leading 1 to form mantissa
   A: 1.1
   B: 1.101

Floating-Point Addition Example

3. Compare exponents
15 – 16 = -1

4. Shift smaller mantissa if necessary
shift A's mantissa right by 1 mantissa: 1.1 becomes 1 = 0.11 (x 2^1)

5. Add mantissas

```
  0.11      (x 2^1)
+ 1.101     (x 2^1)
 10.011     (x 2^1)
```

Floating-Point Addition Example

6. Normalize mantissa and adjust exponent if necessary
   $10.011 \times 2^1 = 1.0011 \times 2^2$

7. Assemble exponent and fraction back into floating-point format
   $S = 0$, $E = 2 + 15 = 17 = 10001_2$, $F = 001100..$

R = A+B = 0|10001|0011000000

Exadecimal of R is: 0x44C0

How to subtract binary numbers (fractional)

Work on 2's complement and sum

Example A=4.25 and B=– 1.5

```
4.25 = 0100.01

 1.5  = 01.1
-1.5  = 10.0 +
            1 =
        -------
         10.1
```

Sign Extension

```
  0100.01 +
  1110.10 =
  ----------
 10010.11 =
= 0010.11 =
```

$= 2^{1}+2^{-1}+2^{-2} = 2.75$

IEEE 754 MULTIPLICATION

Fractional Binary Multiplication

- Multiply the two numbers as if they were unsigned numbers.
- Then, if the first number has *x* digits after the decimal point, and the second number has *y* digits after the decimal point, the result will have *x+y* digits after the decimal point

Example: multiply 1.111 x 1.1

```
 1111 x
 0011 =
-------
 1111 +
1111  =
-------
101101
```

The first number has 3 digits after decimal point, the second number has 1, so the product will have 3+1=4 digits after the decimal point

i.e., 1.111 x 1.1 = 10.1101

Floating-Point Multiplication

1. Sum the exponents and subtract the bias
2. Multiply the mantissas
3. Normalize (if needed) (remember to adjust the exponent)
4. Adjust the sign
5. Assemble the result

Floating-Point Multiplication - Example

Example: Compute N1xN2 – Half Precision

N1=0100011110000000 N2=1100101000000000

N1: Sign: Positive
Exponent: 10001
Fraction: 1110000000 -> Mantissa: 1.111

N2: Sign: Negative
Exponent: 10010
Fraction: 1000000000 -> Mantissa: 1.1

Floating-Point Multiplication - Example

- Sum the exponents and subtract the bias:
  `10001 + 10010 – 01111`

- By using 2's complement we have:
  `00010001 + 00010010 – 00001111 =`
  `00010001 + 00010010 + 11110001 = 10100`

- The biased exponent is 20, thus the actual exponent is 20-15=5

- Multiply the mantissas
  `1.111 x 1.1 = 10.1101`

Thus, the result would be 10.1101×2^5

Floating-Point Multiplication - Example

- Normalization (when needed): adjust the mantissa and exponent:

- We bring point to the right of most significant 1:

$$10.1101 \times 2^5 = 1.01101 \times 2^6$$

- Because the exponent we encoded corresponds to 5, we have to add +1 to it: $10100 + 1 = 10101$

Floating-Point Multiplication - Example

- Adjust the sign
- We multiplied a positive with a negative number, so the sign must be negative, that is 1

- Sign = `1`
- Biased exponent = `10101`
- Mantissa = `1.01101` -> Fraction = `0110100000`

IEEE 754 half-precision representation: `1|10101|0110100000`

Exadecimal is: 0xD5A0