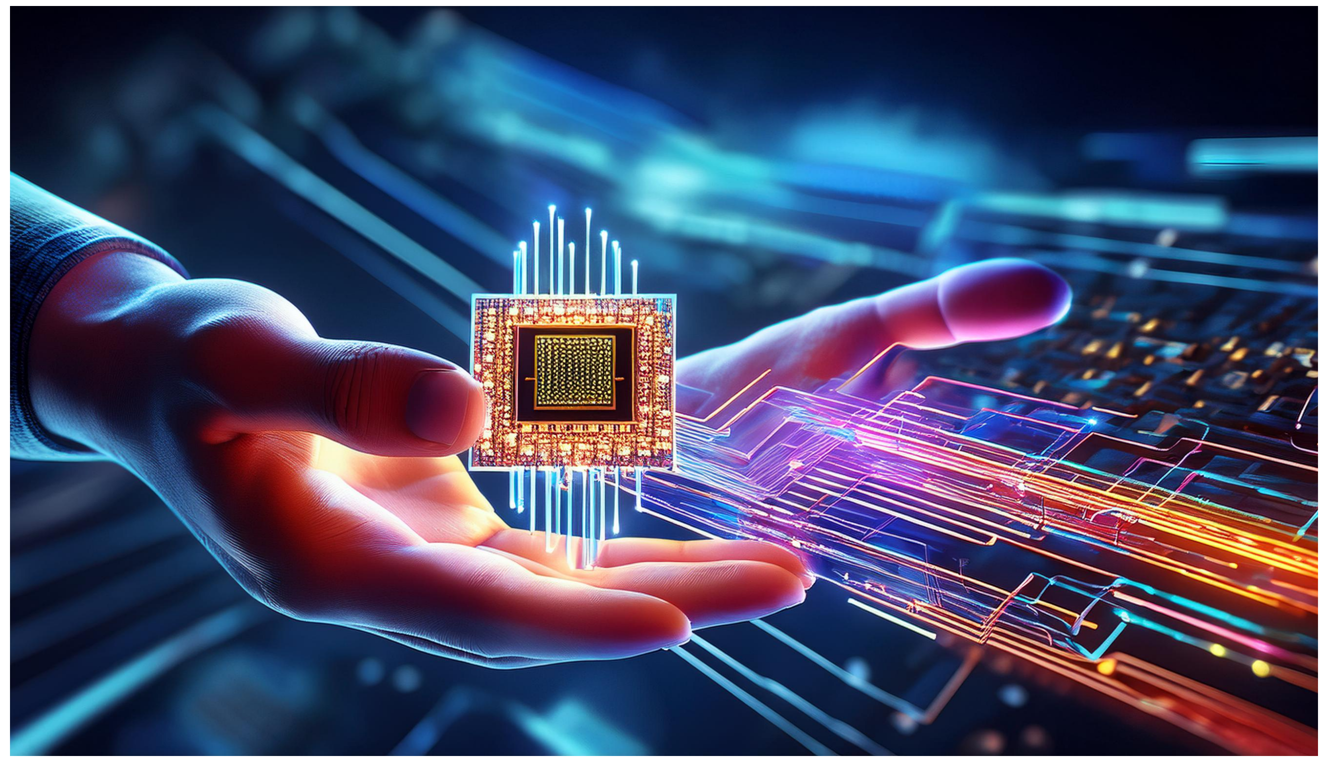
13.09.24 17:56

CSCI 250
Introduction to Computer Organisation
Lecture 1: Beyond Integer Arithmetics II



Jetic Gū

2024 Fall Semester (S3)

Overview

- Focus: Course Introduction
- Architecture: Logical Circuits
- Core Ideas:
  1. Float Numbers
  2. Float addition and subtractions
  3. Float multiplications and division

Float Numbers

Why float?

- In a computer, binary **integers** (signed or unsigned) have fixed positions for the radix point (like decimal point for decimal numbers), and therefore are **fixed-point numbers**
- If you want fractional numbers, computers uses **Float-point numbers** instead, where the radix point can be moved

Review

Standard scientific notation

- For decimal numbers, the **standard scientific notation** is an example for float point numbers in mathematics
- $12345.6789 \text{ -> } 1.23456789 \times 10^4$

This exponent represents the position of the radix point

This is called the factor

Review

Float (Binary)

- For **binary float** numbers, we can represent values using **standard scientific notation** as well
- $20.625 = 16 + 4 + 0.5 + 0.125 = 2^4 + 2^2 + 2^{-1} + 2^{-3} = (10100.101)_2$
- $(10100.101)_2 = (1.0100101)_2 \times (10000)_2$
  $= (1.0100101)_2 \times 2^4$
  $= (1.0100101)_2 \times 2^{(100)_2}$
- We call $(1.0100101)_2$ the **mantissa**, and $(100)_2$ here the **exponent**

Concept

P1
Float Numbers

Computer Float Numbers

$$\underbrace{(1.0100101)_2}_{\text{Mantissa}} \times 2^{\overbrace{(100)_2}^{\text{Exponent}}}$$

- In 1985, IEEE standardised binary float representations as IEEE 754
- IEEE 754 provides specifications for 16bit, 32bit, 64bit, 80bit, and 128bit float numbers. They are all **signed**. They have different number of bits for the **Mantissa** and **Exponent**
- First, let's take a look at **16bit**, or **half precision float**

Concept

16bit Float Numbers

$$\underbrace{(1.0100101)_2}_{\text{Mantissa}} \times 2^{\overbrace{(100)_2}^{\text{Exponent}}}$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M
0	1	0	0	1	1	**0**	**1**	**0**	**0**	**1**	**0**	**1**	0	0	0
	Exponent					Mantissa									

- This is the partition of a 16bit single precision float number
  - The first bit is the **sign bit**, 1 for negative, 0 for 0 or positive
  - The next 5 bits are the **exponent**
  - The last 10bits are the **mantissa/significand/magnitude**

1. significand is not a typo

Concept

16bit Float Numbers

$$\underbrace{(1.0100101)_2}_{\text{Mantissa}} \times 2^{\overbrace{(100)_2}^{\text{Exponent}}}$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M
0	1	0	0	1	1	**0**	**1**	**0**	**0**	**1**	**0**	**1**	0	0	0
	Exponent					Mantissa									

- The exponent
    - The exponent is an integer, if it's non-zero the float number is **Normal**, an offset of -15 is applied
    - If it's 0, the float number is **subnormal**, the offset is then -14
    - E.g. $(00100)_2$ = 4, this is **Normal** as an exponent it's equal to -11 after the offset
      $(10011)_2$ = 16 + 3 = 19, this is **Normal** as an exponent it's equal to 4 after the offset

Concept

16bit Float Numbers Exponent

Binary Exponent	Decimal Equivalent	Normal/Subnormal	After offset	Actual value of entire float
00000	0	Subnormal	-14	Mantissa * 2^{-14}
00001	1	Normal	-14	Mantissa * 2^{-14}
01111	15	Normal	0	Mantissa * 2^{0}
10000	16	Normal	1	Mantissa * 2^{1}
10100	20	Normal	5	Mantissa * 2^{5}
11111	31	Normal	16	Infinity

Technical

P1
Float Numbers

16bit Float Numbers

$$\underbrace{(1.0100101)_2}_{\text{Mantissa}} \times 2^{\overbrace{(100)_2}^{\text{Exponent}}}$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M
0	1	0	0	1	1	**0**	**1**	**0**	**0**	**1**	**0**	**1**	0	0	0
	Exponent					Mantissa									
						2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}

- The mantissa
  - The mantissa is a binary number, but purely in fractions
  - The most significant bit is always on the left, representing 2^{-1}
  - Normal float: mantissa value is offset by 1, so 0100101000 represents 1.0100101
    Subnormal: mantissa value won't have offset, so 0101010101 represents 0.0101010101

Concept

P1
Float Numbers

16bit Float Numbers (Subnormal)

Binary	Hex	Decimal Value	Notes
0 00000 0000000000	0000	0	
0 00000 0000000001	0001	$2^{-14} \times (\frac{1}{1024})$	smallest positive subnormal number
0 00000 1111111111	03ff	$2^{-14} \times (\frac{1023}{1024})$	largest subnormal number
1 00000 0000000000	8000	0	

Technical

P1
Float Numbers

16bit Float Numbers (Normal)

Binary	Hex	Decimal Value	Notes
0 00001 0000000000	0400	$2^{-14} \times (1 + \frac{0}{1024})$	smallest positive normal number
0 01101 0101010101	3555	$2^{-2} \times (1 + \frac{341}{1024})$	nearest value to 1/3
0 01110 1111111111	3bff	$2^{-1} \times (1 + \frac{1023}{1024})$	largest number less than one
0 01111 0000000000	3c00	$2^{0} \times (1 + \frac{0}{1024}) = 1$	one
0 01111 0000000001	3c01	$2^{0} \times (1 + \frac{1}{1024})$	smallest number larger than one
0 11110 1111111111	7bff	$2^{15} \times (1 + \frac{1023}{1024})$	largest normal number
0 11111 0000000000	7c00	∞	infinity
1 10000 0000000000	c000	$-2^{1} \times (1 + \frac{0}{1024}) = -2$	
1 11111 0000000000	fc00	$-\infty$	negative infinity

Technical

P1
Float Numbers

Float numbers in CSCI250

- I can't remember all of the details of float numbers! What should I do?
  - In CSCI250, we will NOT quiz/exam you on stuff like:
    how many digits are there for 16bit exponent/mantissa?
    what's the exponent offset for normal/subnormal numbers?
    how do you distinguish normal/subnormal float numbers?
  - Here an example:

Concept

Float numbers in CSCI250

- In IEEE 754, assume 32bit (single precision) normal number, where the exponent offset is -127 for normal numbers, 8bit for exponent, 23bit for mantissa, what is the following number's equivalent in single precision float?

    - 17.5

    - Solution:
      $17.5 = (10001.1)_2 = (1.00011)_2 \times 2^4$
      Exponent: $4 + 127 = 131 = (100)_2 + (0111\ 1111)_2 = (1000\ 0011)_2$
      =
      Mantissa: $(1 + 0.00011)_2$ -> 0001 1000 0000 0000 0000 000
      Combined: 0 1000 0011 0001 1000 0000 0000 0000 000

Exercise

Issues with Float numbers

- Precision issue
  - Float numbers provide approximations to actual values with limited precision
  - With computers, sometimes this causes issues
    - e.g. 1/3 * 3 doesn't necessarily equal to 1, ALL programming languages will have the same issue but some goes around it (to a limited degree) by rounding it up
    - Here's an example in python[2]: 0.1 + 0.1 + 0.1 == 0.3

Concept

1. https://www.h-schmidt.net/FloatConverter/IEEE754.html
2. https://docs.python.org/3/tutorial/floatingpoint.html

Exercise 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M
1	1	0	0	0	1	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**

- Exponent: 10001, this is a **Normal** float
  $(10001)_2 = 17$, after offset: $17 - 15 = 2$

- Mantissa: 0101000000, since this number is **Normal**, mantissa is 1.0101
  $(1.0101)_2 = 1 + 2^{-2} + 2^{-4} = 1 + 0.25 + 0.0625 = 1.3125$

- Finally, sign bit is 1, so negative number
  Final value (Bin): $-1 \times (1.0101)_2 \times (10)_2^2 = (-101.01)_2$
  Final value (Dec): $-1 \times 1.3125 \times 2^2 = -5.25$

Exercise

Exercise 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M
0	1	0	0	1	1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

- Exponent: 10011, this is a **Normal** float
  $(10011)_2 = 19$, after offset: $19 - 15 = 4$

- Mantissa: 1010000000, since this number is **Normal**, mantissa is 1.101
  $(1.101)_2 = 1 + 2^{-1} + 2^{-3} = 1 + 0.5 + 0.125 = 1.625$

- Finally, sign bit is 0, so positive number
  Final value (Bin): $(1.101)_2 \times (10)_2^4 = (11010)_2$
  Final value (Dec): $1.625 \times 2^4 = 26$

Exercise

Float Number Arithmetics

Float Number Addition and Subtraction

- In order to perform addition and subtraction between float numbers, it is important to understand how this can be done mathematically
- Say, we have two float numbers $X = X_m \times 2^{X_e}, Y = Y_m \times 2^{Y_e}$
- How can we perform addition and subtraction?

Concept

P2
Float Arithmetic

Float Number Addition and Subtraction

$$X = X_m \times 2^{X_e} \qquad Y = Y_m \times 2^{Y_e}$$

- Case 1: $X_e = Y_e$
  - When the exponents are equal, the mantissas can be directly added/subbed
  - ! Normal vs Subnormal
    - Subnormal float: mantissas can be directly added
    - Normal float: there may be an offset

Technical

Float Number Addition and Subtraction

$$X = X_m \times 2^{X_e} \qquad Y = Y_m \times 2^{Y_e}$$

What if $X_e < Y_e$?

- Case 2: $X_e > Y_e$
  - When the exponents are not equal, we cannot add/sub the mantissas directly
  - We can right shift Y_m, so exponents align

$$\begin{aligned} Y &= Y_m \times 2^{Y_e} \\ &= (Y_m >> 1) \times 2^{Y_e+1} \\ &= (Y_m >> 2) \times 2^{Y_e+2} = \ldots \\ &= (Y_m >> (X_e - Y_e)) \times 2^{X_e} \end{aligned}$$

Technical

Exercise

$$X = (1.1010)_2 \times 2^{(101)_2} \qquad Y = (1.0011)_2 \times 2^{(11)_2}$$

- Simulate float point addition of $X + Y$
  - Step 1: equalise the exponent, $(101)_2 - (11)_2 = 2$:
    $Y = ((1.0011)_2 >> 2) \times 2^{(11)_2 + 2} = (0.010011)_2 \times 2^{(101)_2}$
  - Step 2: perform addition
    $(1.1010)_2 + (0.010011)_2 = (1.111011)_2$
  - Step 3: adjust exponent if needed
    $X + Y = (1.111011)_2 \times 2^{(101)_2}$

Exercise

Float
Multiplication and Division

- Say, we have two float numbers $X = X_m \times 2^{X_e}$, $Y = Y_m \times 2^{Y_e}$
- How can we perform multiplication and division?

Concept

P2
Float Arithmetic

Float Multiplication

$$X = X_m \times 2^{X_e} \qquad Y = Y_m \times 2^{Y_e}$$

- Simulate float point multiplication: XY
  - $XY = X_m \times 2^{X_e} \times Y_m \times 2^{Y_e}$
  - $= X_m Y_m \times 2^{X_e} \times 2^{Y_e}$
  - $= X_m Y_m \times 2^{X_e + Y_e}$

Is $X_m Y_m$ difficult?
Can it be accomplished using unsigned int multiplier?

Exercise

LAB 1 Part 1
A Python Exercise

P3
LAB 1

LAB 1 Part 1
A Python Exercise

- You will define a Python class to support a custom float standard
- This class will need to support addition and subtraction
- This class will need to be able to convert binary float to decimal
- Download the template to modify: jetic.org/dl/myfloat.py
- Example tester: jetic.org/dl/myfloat_test.py

Concept

P3
LAB 1

LAB 1 Part 1
A Python Exercise

- Assume the number of digits for Exponent is e, the offset is $-2^{e-1}+1$
- We'll only test your code with normal numbers
- Do not modify the filename
- Do not change the interface of the class
- Do not modify or add print expressions
- Due next Sunday midnight

Concept