Numbers in Computers

Review

- Last lecture, we saw
  - How to convert a number in binary to decimal
    - Simply adding up the exponents of 2 where the binary digit is 1
  - How to convert a number in decimal into a number in binary
    - Keep on dividing it by 2 until the quotient is 0. Then write down the remainders, last remainder first.
  - How to do addition and subtraction in binary

This Lecture

- We will deal with
  - Signed numbers
  - Numbers with fractions

Signed Numbers

- Two's complement
  - The negative of a two's complement is given by inverting each bit (0 to 1 or 1 to 0) and then adding 1.
  - If we are allowed to use only n bits, we ignore any carry beyond n bits (take only the lower n bits).

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

1	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1

2’s complement

- In any computer, if numbers are represented in n bits, the non-negative numbers are from 0000…00 to 0111…11, the negative numbers are from 1000…00 to 1111…11.
- The leading bit is called the ``sign bit.’’
- What is the representation of 0?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0_{ten}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1_{ten}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2_{ten}
…																									…							
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
…																									…							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	-3_{ten}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	-2_{ten}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1_{ten}

- The positive half from 0 to 2,147,483,647
- The negative half from -2,147,483,648 to -1

Question

- What is the range of numbers represented by 2's complement with 4 bits?

Question

- What is the range of numbers represented by 2's complement with 4 bits?
- The answer is [-8,7].
- This is because all numbers with leading bit being 1 is a negative number. So we have 8 of them. Then 0 is all 0. Then seven positive numbers.
- If numbers are represented in n bits, the non-negative numbers are from 0000…00 to 0111…11, the negative numbers are from 1000…00 to 1111…11.

Two's Complement Representation

Type (C)	Number of bits	Range (decimal)
char	8	-128 to 127
short	16	-32768 to 32767
int	32	-2,147,483,648 to 2,147,483,647
long long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
n+1 bits (in general)	n+1	-2^n to $2^n - 1$

Subtraction with 2’s Complement

- How about $39_{ten} + (-57_{ten})$?

Subtraction with 2's Complement

- First, what is (-57_{ten}) in binary in 8 bits?
  1. 00111001 (57_{ten} in binary)
  2. 11000110 (invert)
  3. 11000111 (add 1)
- Second, add them.

  00100111 (39_{ten} in binary)

  11000111 (-57_{ten} in binary)

  11101110 (-18_{ten} in binary)

Converting 2’s complement to decimal

- What is 11101110_{ten} in decimal if it represents a two’s complement number?

1. 11101110 (original)
2. 11101101 (after minus 1. Binary subtraction is just the inverse process of addition. Borrow if you need.)
3. 00010010 (after inversion)

Two's Complement Representation

- Sign extension
  - We often need to convert a number in n bits to a number represented with more than n bits
    - From char to int for example
  - This can be done by taking the most significant bit from the shorter one and replicating it to fill the new bits of the longer one
    - Existing bits are simply copied

Sign Extension Example

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
																								1	1	1	1	1	1	0	1		-3_{ten}
																1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1		-3_{ten}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1		-3_{ten}

- How about unsigned numbers?

Sign Extension Example: Unsigned

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
																								1	1	1	1	1	1	0	0		252_{ten}
																0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0		252_{ten}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0		252_{ten}

Unsigned and Signed Numbers

- Note that bit patterns themselves do not have inherent meaning
  - We also need to know the type of the bit patterns
  - For example, which of the following binary numbers is larger?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Unsigned and Signed Numbers

- Note that bit patterns themselves do not have inherent meaning
  - We also need to know the type of the bit patterns
  - For example, which one is larger?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

    - Unsigned numbers?
    - Signed numbers?

Numbers with Fractions

- So, done with negative numbers. Done with signed and unsigned integers.
- How about numbers with fractions?
- How to represent, say, 5.75_{ten} in binary forms?

Numbers with Fractions

- In general, to represent a real number in binary, you first find the binary representation of the integer part, then find the binary representation of the fraction part, then put a dot in between.

Numbers with fractions

- The integer part is 5_{ten} which is 101_{two}. How did you get it?

Numbers with Fractions

- The fraction is 0.75. Note that it is $2^{-1} + 2^{-2} = 0.5 + 0.25$, so

  $5.75_{ten} = 101.11_{two}$

How to get the fraction

- In general, what you do is kind of the reverse of getting the binary representation for the integer: divide the fraction first by 0.5 (2^{-1}), take the quotient as the first bit of the binary fraction, then divide the remainder by 0.25 (2^{-2}), take the quotient as the second bit of the binary fraction, then divide the remainder by 0.125 (2^{-3}),

How to get the fraction

- Take 0.1 as an example.
  - 0.1/0.5=0*0.5+0.1 –> bit 1 is 0.
  - 0.1/0.25 = 0*0.25+0.1 –> bit 2 is 0.
  - 0.1/0.125 = 0*0.125+0.1 –> bit 3 is 0.
  - 0.1/0.0625 = 1*0.0625+0.0375 –> bit 4 is 1.
  - 0.0375/0.03125 = 1*0.03125+0.00625 –> bit 5 is 1.
- And so on, until the you have used all the bits that hardware permits

Floating Point Numbers

- Recall scientific notation for decimal numbers
  - A number is represented by a significand (or mantissa) and an integer exponent $F * 10^E$
    - Where F is the significand, and E the exponent
  - Example:
    - $3.1415926 * 10^2$
    - Normalized if F is a single digit number

Floating Points in Binary

- Normalized binary scientific notation

$$1.xxxxxxxxxx_{two} \times 2^{yyyy}$$

  – For a fixed number of bits, we need to decide
    - How many bits for the significand (or fraction)
    - How many bits for the exponent
    - There is a trade-off between precision and range
      – More bits for significand increases precision while more bits for exponent increases the range

IEEE 754 Floating Point Standard

- Single precision
  - Represented by 32 bits

31	30 23	22 0
S	E	F
1 bit	8 bits	23 bits

  - Since the leading 1 bit in the significand in normalized binary numbers is always 1, it is not represented explicitly

Exponent

- If we represent exponents using two's complement, then it would not be intuitive as small numbers appear to be larger

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Biased Notation

- The most negative exponent will be represented as 00…00 and the most positive as 111…11
  - That is, we need to subtract the bias from the corresponding unassigned value
  - The value of an IEEE 754 single precision is

$$(-1)^{S} \times (1 + 0.\text{Fraction}) \times 2^{(\text{Exponent} - 127)}$$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent								fraction																						
1 bit	8 bits								23 bits																						

Example

$101.11_{two} = 2^2 + 2^0 + 2^{-1} + 2^{-2} = 5.75_{ten}$

The normalized binary number will be

$1.0111 \times 2^2 = 1.0111 \times 2^{(129-127)}$

So the exponent is 129_{ten} = 10000001

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

As a hexadecimal number, the representation is

0x40B80000

IEEE 754 Double Precision

- It uses 64 bits (two 32-bit words)
  - 1 bit for the sign
  - 11 bits for the exponent
  - 52 bits for the fraction
  - 1023 as the bias

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>s</td><td colspan="11">Exponent</td><td colspan="20">fraction</td></tr>
<tr><td>1 bit</td><td colspan="11">11 bits</td><td colspan="20">20 bits</td></tr>
<tr><td colspan="32">Fraction (continued)</td></tr>
<tr><td colspan="32">32 bits</td></tr>
</table>

Example (Double Precision)

$101.11_{two} = 2^2 + 2^0 + 2^{-1} + 2^{-2} = 5.75$

The normalized binary number will be

$1.0111 \times 2^2 = 1.0111 \times 2^{(1025-1023)}$

So the exponent is $1025_{ten} = 10000000001_{two}$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

As a hexadecimal number, the representation is

0x4017 0000 0000 0000

Special Cases

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	nonzero	0	nonzero	±denormalized number
1-254	anything	1-2046	anything	±floating-point number
255	0	2047	0	± infinity
255	nonzero	2047	nonzero	NaN (Not a number)

Floating Point Numbers

- How many different numbers can the single precision format represent? What is the largest number it can represent?

Ranges for IEEE 754 Single Precision

- Largest positive number

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

- Smallest positive number (floating point)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ranges for IEEE 754 Single Precision

- Largest positive number

$$(1 + 1 - 2^{-23}) \times 2^{(254 - 127)} = 2^{128} - 2^{104} = 3402823466 \quad 3852885981 \quad 1704183484 \quad 516925440$$

$$\approx 3.4028235 \times 10^{38}$$

- Smallest positive number (floating point)

$$(1 + 0.0) \times 2^{(1-127)} = 2^{-126} \approx 1.175494351 \times 10^{-38}$$

Ranges for IEEE 754 Double Precision

- Largest positive number

$$(1 + 1 - 2^{-52}) \times 2^{(2046 - 1023)} = 2^{1024} - 2^{971} \approx 1.7976931348623157 \times 10^{308}$$

- Smallest positive number (Floating-point number)

$$(1 + 0.0) \times 2^{(1-1023)} = 2^{-1022} \approx 2.2250738585 \times 10^{-308}$$

Comments on Overflow and Underflow

- Overflow (and underflow also for floating numbers) happens when a number is outside the range of a particular representation
  - For example, by using 8-bit two's complement representation, we can only represent a number between -128 and 127
    - If a number is smaller than -128, it will cause overflow
    - If a number is larger than 127, it will cause overflow also
  - Note that arithmetic operations can result in overflow