Basic Concepts

Base 2 integer encoding
- Suppose we want the computer to store a number 3. How do you think the computer will store it? Store it as “3” or other format?
  • Everything stored in a computer is encoded in a sequence of 0’s and 1’s.
  • Computers will store 3 in binary, 11, using base 2 integer encoding.
  • To represent 3 in binary, a computer needs at least 2 bits. Each “1” here is a bit.

- What is base 2?
  • Base 2 is just like base 10 except you have only two digits to work with on your odometer.
  • Let’s start counting: 0, 1, 2, 3, 4, 5, … (decimal) —> 0, 1, 10, 11, 100, 101, … (binary)
  • We use the following array to assist the transition from a binary to a decimal.

<table>
<thead>
<tr>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
</table>

- So, to calculate the decimal “1101” represent, we can use
  \[ 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13 \]
  • This also means that a computer needs at least 4 bits to store 13.

- Exercise:
  • 11010
    - \[ 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26 \]

- What is a bit?
  • Bit is a basic unit.
  • It’s the smallest piece of information a computer can work with.
  • A bit represents a logical state with one of two possible values, 0 and 1. Just like a light bulb, it can be either on or off.

- Another term used a lot in computer science is **Byte**.
  • A byte is 8 bits.
  • What is the smallest number a byte can represent? [0]
  • What is the largest number a byte can represent?
    \[ 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 2^8 - 1 = 255 \]
  • If use a byte to represent a number that needs less than 8 bits, the left bits are filled with 0s. E.g., 26 will be 00011010.
How do we convert a decimal to binary? Convert 126 into 1 byte binary.

Suppose we are going to use 1 byte to represent 126, then the binary is 01111110.

2’s Complement
- So far so good. But what if we want to store a negative number? [2’s complement]
- Representation for signed binary numbers

<table>
<thead>
<tr>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The value of the leftmost bit is \(-2^{(N-1)}\), if given the fixed size N.
- To represent -126 in binary, there are three steps:
  - Step1: Convert 126 to binary (like what we did above)
  - Step2: Invert each bit
    - 01111110 \(\rightarrow\) 10000001
  - Step3: Add one to the result of Step2
    - 10000001 + 1 = 10000010
- To verify:
  - \(1 \times -2^7 + 1 \times 2^1 = -128 + 2 = -126\)

- The largest negative value represented by 8 bits in 2’s complement is 1000 0000 = \(-2^7\)
- The largest positive value represented by 8 bits in 2’s complement is 0111 1111 = 127 = \(2^{7}-1\)
- The range represented by N bits in 2’s complement is \(-2^{(N-1)} \sim 2^{(N-1)}-1\)
Overflow

- Calculate 127 + 127 by using 8-bits 2'complement

  0111 1111  ←→ 127
  0111 1111  ←→ 127
  — — — — — — — — — —
  1111 1110  ←→ -2

  (usually the calculation result uses the same representation as the operands; int + int = int)

• The right answer should be 254, which is out of the range of 8-bits in 2’s complement.
• The error is caused by overflow.
• 2’s complement can be thought as a clock, half of it is positive values and the other is
  negative value. If a positive value exceeds the boundary, it actually enters in the the half for
  the negative values. So it generates an incorrect value.

- This is why the InfiniteLoop example we discussed in the first class generates a negative
  value and does not loop forever.