TUTORIAL 3 BASE CONVERSION AND INTEGER REPRESENTATION

Overview

- **We will review the following concept in this tutorial:**
- **Base conversion**
  - Between base 2 (binary) and base 10 (decimal)
  - Between base 2 (binary) and base 16 (hexadecimal)
- **Representation of integers**
  - Signed magnitude representation
  - One’s complement
  - Two’s complement

The Decimal (Base 10) representation

- We are using the decimal (base 10) number system to represent numbers in our daily lives.

- For example when we say a year has $365_{(10)}$ days, this decimal number $365_{(10)}$ really means the following:

$$365_{(10)} = 300 + 60 + 5$$
$$365_{(10)} = 3 \times 100 + 6 \times 10 + 5 \times 1$$
$$365_{(10)} = 3 \times 10^2 + 6 \times 10^1 + 5 \times 10^0$$

- When we represent a fractional number in the decimal (base 10) format like in the following, it really means:

$$365.25_{(10)} = 3 \times 10^2 + 6 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

The General Base *r* Representation

- In general we can represent a number using any *base* (or radix) *r:*

$$\underbrace{(a_n \ldots\ a_5 a_4 a_3 a_2 a_1 a_0}_{\text{Integer part}}.\underbrace{a_{-1} a_{-2} a_{-3} a_{-4} \ldots a_{-m}}_{\text{Fractional part}})_r \qquad a_i < r$$

- For example: the following fractional number represented in base *r* is really:

$$(a_n \ldots\ a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} a_{-4} \ldots a_{-m})_r$$

$$= a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \cdots + a_0 \times r^0 + a_{-1} \times r^{-1} + \ldots + a_{-m} \times r^{-m}$$

Warm up exercise 1

- **Convert number from binary to decimal**
- Q1. $(111.01)_2=(?)_{10}$
- Solution
- $(111.01)_2=1\times2^0+1\times2^1+1\times2^2+0\times2^{-1}+1\times2^{-2}=(7.25)_{10}$

Warm up exercise 2

- **Convert number from decimal to binary**
- Q2. $(26)_{10}=(?)_2$
- Solution

Division	Quotient	Generated Remainder
$\frac{26}{2}$	13	0
$\frac{13}{2}$	6	1
$\frac{6}{2}$	3	0
$\frac{3}{2}$	1	1
$\frac{1}{2}$	0	1

Hence the converted binary number is 11010.

Warm up exercise 3

- **Convert number from decimal to hexadecimal**
- Q3. $(426)_{10}=(?)_{16}$
- Solution

Division	Quotient	Generated Remainder
$\frac{426}{16}$	26	10
$\frac{26}{16}$	1	10
$\frac{1}{16}$	0	1

Hence the converted hexadecimal number is 1AA.

Warm up exercise 4

- **Convert number from binary to hexadecimal**
- Q4. $(11010110)_2=(?)_{16}$
- Solution

1	1	0	1	0	1	1	0
D				6			

Hence the converted hexadecimal number is D6.

Different notation of base 16

- They are all representing a same hexadecimal number

$$2BC_{16} = 2BC_{hex} = 0x2BC$$

Extra exercises

- Convert $37_{(10)}$ to the binary format.
- 100101
- Convert $1034_{(10)}$ to the binary format.
- 10000001010
- Convert the positive integer $101001_{(2)}$ to the decimal format.
- 41
- Convert $10111001_{(2)}$ to the hexadecimal format.
- B9
- Convert $A7_{(16)}$ to the binary format.
- 1010 0111

Signed magnitude representation

- **Humans use a signed-magnitude system: we add + or - in front of a magnitude to indicate the sign.**
- **We could do this in binary as well, by adding an extra sign bit to the front of our numbers.**
  - A 0 sign bit represents a positive number.
  - A 1 sign bit represents a negative number.
- **Examples:**

$1101_2 = 13_{10}$ (a 4-bit unsigned number)
$0\,1101 = +13_{10}$ (a positive number in 5-bit signed magnitude)
$1\,1101 = -13_{10}$ (a negative number in 5-bit signed magnitude)

$0100_2 = 4_{10}$ (a 4-bit unsigned number)
$0\,0100 = +4_{10}$ (a positive number in 5-bit signed magnitude)
$1\,0100 = -4_{10}$ (a negative number in 5-bit signed magnitude)

Arithmetic with signed magnitude

- **Adding numbers is difficult, though. You can't do bit-by-bit addition directly.**
- **It’s based on comparing the signs of the augend and addend:**
  - If they have the same sign, add the magnitudes and keep that sign.
  - If they have different signs, then subtract the smaller magnitude from the larger one. The sign of the number with the larger magnitude is the sign of the result.
- **This method of subtraction would lead to a rather complex circuit.**
  - A decimal example

$$\begin{array}{rrrr} & +3 & 7 & 9 \\ + & -6 & 4 & 7 \\ \hline & -2 & 6 & 8 \end{array} \qquad \text{because} \qquad \begin{array}{rrrr} & 5 & 13 & 17 \\ & 6 & 4 & 7 \\ - & 3 & 7 & 9 \\ \hline & 2 & 6 & 8 \end{array}$$

One's complement representation

- A different approach, **one's complement**, negates numbers by complementing each bit of the number.
- We keep the sign bits: 0 for positive numbers, and 1 for negative. The sign bit is complemented along with the rest of the bits.
- Examples:

$1101_2 = 13_{10}$ (a 4-bit unsigned number)

0 1101 $= +13_{10}$ (a positive number in 5-bit one's complement)

1 0010 $= -13_{10}$ (a negative number in 5-bit one's complement)

$0100_2 = 4_{10}$ (a 4-bit unsigned number)

0 0100 $= +4_{10}$ (a positive number in 5-bit one's complement)

1 1011 $= -4_{10}$ (a negative number in 5-bit one's complement)

Arithmetic with one's complement

- **To add one's complement numbers:**
  - First do unsigned addition on the numbers, including the sign bits.
  - Then take the carry out and add it to the sum.
- **Examples:**

$$
\begin{array}{rrr}
 & 0111 & (+7) \\
+ & 1011 & +\ (-4) \\
\hline
1 & 0010 & \\
 & & \\
 & 0010 & \\
+ & 1 & \\
\hline
 & 0011 & (+3)
\end{array}
\qquad\Bigg|\qquad
\begin{array}{rrr}
 & 0011 & (+3) \\
+ & 0010 & +\ (+2) \\
\hline
0 & 0101 & \\
 & & \\
 & 0101 & \\
+ & 0 & \\
\hline
 & 0101 & (+5)
\end{array}
$$

- **This is simpler and more uniform than signed magnitude addition. Drawbacks of one's complement:**
  - Two representations of 0: 00000000 (+0) and 11111111 (-0)
  - Need to take care of the carry for addition.

Two's complement representation

- **Our final idea is two's complement. To negate a number, complement each bit (just as for ones' complement) and then add 1.**
  - Or, from LSB to MSB, don't negate any bit upto-and-including the least significant '1' bit, and then negate the rest.
- **Examples:**

	$1101_2 = 13_{10}$	(a 4-bit unsigned number)
0	$1101 = +13_{10}$	(a positive number in 5-bit two's complement)
1	$0010 = -13_{10}$	(a negative number in 5-bit *ones'* complement)
1	$0011 = -13_{10}$	(a negative number in 5-bit two's complement)
	$0100_2 = 4_{10}$	(a 4-bit unsigned number)
0	$0100 = +4_{10}$	(a positive number in 5-bit two's complement)
1	$1011 = -4_{10}$	(a negative number in 5-bit *ones'* complement)
1	$1100 = -4_{10}$	(a negative number in 5-bit two's complement)

Arithmetic with two's complement

- **Negating a two's complement number takes a bit of work, but addition is much easier than with the other two systems.**
- **To find A + B, you just have to:**
  - Do unsigned addition on A and B, including their sign bits.
  - Ignore any carry out.
- **For example, to find 0111 + 1100, or (+7) + (-4):**
  - First add 0111 + 1100 as unsigned numbers:

$$
\begin{array}{r}
0111 \\
+\ 1100 \\
\hline
10011
\end{array}
$$

  - Discard the carry out (1).
  - The answer is 0011 (+3).

Why does it work?

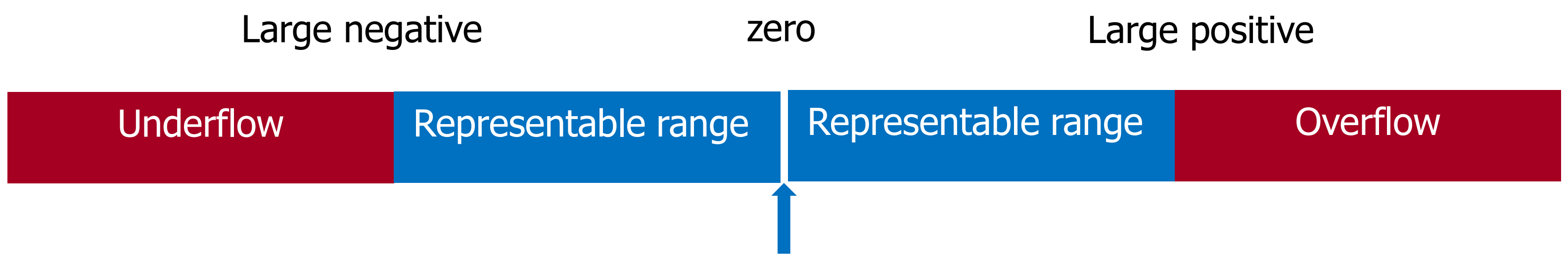
- For n-bit numbers, the negation of B in two's complement is $2^n - B$ (this is one of the alternative ways of negating a two's-complement number).

$$
\begin{aligned}
A - B &= A + (-B) \\
&= A + (2^n - B) \\
&= (A - B) + 2^n
\end{aligned}
$$

- If A ≥ B, then (A - B) is a positive number, and 2^n represents a carry out of 1. Discarding this carry out is equivalent to subtracting 2^n, which leaves us with the desired result (A - B).
- If A < B, then (A - B) is a negative number and we have 2^n - (B - A). This corresponds to the desired result in two's complement form.

Overflow and underflow of signed integer

- **Overflow (signed integer)**
  - The value is bigger than the largest integer that can be represented
- **Underflow (signed integer)**
  - The value is smaller than the smallest integer that can be represented



Exercises

- **Convert decimal number to 2's complement number on 6 bits**
- Q1. $(19)_{10} = (?)_2$
- Solution: 010011_2
- Q2. $(-32)_{10} = (?)_2$
- Solution: 100000_2
- Q3. $(32)_{10} = (?)_2$
- Solution: Cannot be represented (overflowed)