Binary, Octal, and Hexadecimal Number Systems

CS 10A – NUMBER BASES

Allen Zhao - SRJC CS 10A

Introduction to Bases

- There are multiple ways to represent numbers in written form when using the traditional Arabic numerals.
- Our traditional number system runs in base 10 (decimal).
  - 10 different symbols (0-9), each representing one number
  - Each place to the left represents an additional power to the base
    - 1s place, 10s place, 100s place, 1000s place, etc.
- Arabic numerals are designed for base 10. Easy to read.
  - i.e. $96 = 9*10^1 + 6*10^0$, $258 = 2*10^2 + 5*10^1 + 8*10^0$
- However, there are advantages to changing the base

Binary

- At some point we realized that designing faster, more powerful, and more accurate computers would be much easier if everything was represented with several on/off switches instead of controlling arbitrary analog signals.
- So, to represent these switches, all computers run in Base 2, otherwise known as binary.
  - Binary only uses 0 and 1.
  - 0 translates into off or false, 1 translates into on or true.
  - Each place in binary is known as a bit.
  - Left-most bit is called most-significant bit (MSB)
  - Right-most bit is called least-significant bit (LSB)

Octal

- It can be tedious representing large numbers using binary, so other base systems are used to condense binary notation. One of them is Base 8, known as octal.
- A single number in base 8 (0-7) is easily converted into three binary bits ($2^3 = 8$)

Octal	Binary	Octal	Binary
7	111	3	011
6	110	2	010
5	101	1	001
4	100	0	000

Hexadecimal

- Octal is actually not that popular since it's easily mistaken for base 10. Instead, we prefer base 16, or hexadecimal.
- $2^4 = 16$, so a single hexadecimal (hex for shorthand) place takes place of 4 binary bits.
- Since the base value exceeds 9, we used letters a-f to represent the values 10-15, respectively.

Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
f (15)	1111	b (11)	1011	7	0111	3	0011
e (14)	1110	a (10)	1010	6	0110	2	0010
d (13)	1101	9	1001	5	0101	1	0001
c (12)	1100	8	1000	4	0100	0	0000

Application of Bases – Conversion to Base 10

For a given base x, you can only use numbers with the range z_n = [0, x-1]

$$y_x = z_n \dots z_7\ z_6\ z_5\ z_4\ z_3\ z_2\ z_1\ z_0$$

$$y_x = \_\ \dots\ \_\ \ \_\ \ \_\ \ \_\ \ \ \_\ \ \_\ \ \_\ \ \_$$

For each respective n^{th} position, the value represented by that number is $z_n * x^n$

Conversion from base x to base 10 uses the formula below

$$y_{10} = \sum_{n=0}^{\infty} z_n * x^n$$

Application of Bases – Range of Values

- Regardless of the base you're using, all of them follow the same rules with regard to calculating their range.
- Determining the absolute maximum is as follows:
  - max = $base^{places} - 1$ // -1 is used to account for 0
- Determining the range is as follows:
  - Minimum is 0 when ignoring negatives, but you must always remember to actually count zero!
  - Possible values = $base^{places}$ // Also known as Range
- i.e. In base 10, a 4 digit number can have 10^4 = 10000 different possible values, ranging [0, 9999].
- i.e. In binary, a 4 bit number can have 2^4 = 16 different possible values, ranging from [0, 15].

Base Notation in C/C++

- Binary, octal, and hexadecimal can all be used in C/C++
- Use int variables for all three types, outputs in decimal
- To denote the difference between the three in code,
  - Binary values lead with 0b (i.e. 0b111, which is 7)
  - Octal values lead with 0 (i.e. 0111, which is 73)
  - Hex values lead with 0x (i.e. 0x111, which is 273)
    - Hex is especially fun because programmers often use it to write words into their code (i.e. 0xf00d)
- Note that not all compilers will support these other bases, but most of the modern ones should.

Base Notation in C/C++

Program

```
int ex_bi = 0b1101;
int ex_oct = 042;
int ex_hex = 0xb00;
int main()
{
        cout << “Binary: ” << ex_bi
              << “\nOctal: ” << ex_oct
              << “\nHex: ” << ex_hex << endl;
        return 0;
}
```

Console

```
➢ ./a.exe
Binary: 13
Octal: 34
Hex: 2816
```

Application Notes

- Any kind of arithmetic that you do in base 10 can also be done in other bases. The same rules apply.
- In the realm of software, using different bases doesn't have too many applications.
- In the realm of hardware (firmware development), this knowledge is critical since bit strings are exactly how you need to communicate with ICs (integrated circuits).

Adding Numbers in Binary

```
  0b1100    →     12
+ 0b0011    →   +  3
-------------   -------
  0b1111    →     15
```

```
   1 1 1               1     ← carry over
  0b1111    →         15
+ 0b0110    →       +  6
-------------       ------
 0b10101    →         21     ← +1 bit
```

Adding numbers in binary follows the same rules of base 10 addition you learned in elementary school: adding and carrying over, except you're only using 1 and 0. You should get the same result. These rules apply to other bases too.

In binary, instead of carrying over a 1 to the next place when the total is >= 10, you carry over when the total is >= 2. Be mindful of memory limitations (see Variables in Memory slides) or else your math could turn out awry.