Week 2 | Number Representation

Decimal (Base 10)

① Value of a digit depends on its position

e.g. 333 → three
↓ 3-hundred
↘ thirty

☆ Each position represents a power of ten

② Digit Value says how many of that power of ten

☆ possible digits are 0 → 9

e.g. 345 → 5×10^0
↓ 3×10^2
↓ 4×10^1

Useful Terminology: Significance

① Digit position 0 : Least-significant Digit

② Highest digit position : Most-significant Digit

e.g. 2617

- 7 → least significant [contributes 7 ones]
- 2 → most significant [contributes 2 thousands]

Positional Notation

- Digits represent powers of the base / radix
  - Each digit D in radix r is restricted to $0 \le D < r$
- N-digit non-negative number with radix r:

☆ $D_{N-1} \cdots D_1 D_0 = D_{N-1} \times r^{N-1} + \cdots + D_1 \times r^1 + D_0 \times r^0$

☆☆ The value of a number is determined by digits' positions and bases

Binary (Base 2)

- Each position represents a power of two
- Digit value indicates how many of that power of two
  - Possible digits: 0 through 1

(In computing, a "bit" is a binary digit)

e.g. $1101_2 = (1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0)$

1×2^3 1×2^2 0×2^1 1×2^0

$= 8 + 4 + 0 + 1$

$= 13$

Octal (Base 8)

- Here, each position represents a power of eight

∴ possible digits: 0 through 7

e.g. $135_8 = 1\times8^2 + 3\times8^1 + 5\times8^0$

1×8^2 3×8^1 5×8^0

$= 93_{(10)}$

Hexadecimal (Base 16) → (often called "Hex")

- Each digit represents power of sixteen

☆ Hex digit: 0–9 is the same as decimal values

10 → A
11 → B
12 → C
13 → D
14 → E
15 → F

e.g. $3C2_{(16)} = 2\times16^0 + 12\times16^1 + 3\times16^2$

3×16^2 ↙ 12×16^1 ↓ 2×16^0 ↘

$= 9b2_{(10)}$

☆ How many unique numbers can be represented?

# Digits	Binary	Octal	Decimal	Hexadecimal
N	2^N	8^N	10^N	16^N

How we use Bases

1. Decimal is used in real life
2. Computers use binary only
3. Hex / Oct makes it easier for humans to deal with large numbers

---

Value vs. Number

- A value is a particular quantity
- A number is a way to represent a value

☆ To know the value represented by the number, you need to know the format

∴ Same "number" ≠ same values

$$110_{(10)} \neq 110_{(2)} \neq 110_{(16)}$$

Base Conversion

(I) From Decimal : Universal Method

- Divide by base, the remainders are the digits

e.g. $7139_{(10)} \rightarrow$ Hex

16 | 7139
16 | 446 R3
16 | 27 R14
16 | 1 R11
0 R1

$\rightarrow 1BE3_{(16)} = 7139_{(10)}$

★★★ (Remember to write remainders from bottom to top)

e.g. $40_{(10)} \rightarrow$ Binary

2 | 40
2 | 20 R0
2 | 10 R0
2 | 5 R0
2 | 2 R1
2 | 1 R0
0 R1

$\rightarrow 101000_{(2)} = 40_{(10)}$

☆☆ Faster Way to convert Decimal to Binary

- Use comparison and repeated subtraction instead of repeated division

e.g. $40_{(10)} \rightarrow$ Binary

$\therefore 2^5 = 32$ (Largest value of power of 2 less than 40)

$40 - 32 = 8 = 2^3$

	2^5			2^3			
64	32	16	8	4	2	1
0	1	0	1	0	0	0

Note: Only put 1 in these fields, all the others are all 0

e.g. $1073_{(10)} \rightarrow$ Binary

1024	512	256	128	64	32	16	8	4	2	1
1	0	0	0	0	1	1	0	0	0	1

$1073 - 1024 = 49$

$49 - 32 = 17$

$17 - 16 = 1$

Binary ⟷ Hex

- Directly substitute a hex digit for a 4-bit binary number

e.g. $0110\ 1101\ 0001\ 1111_{(2)} \rightarrow$ Hex

$6\ D\ 1\ F$

$\therefore 6D1F_{(16)}$

✡ Why does it work?

$\because 16 = 2^4$, so the range of each is $[0 \ldots 15]$ in decimal

---

Binary ⟷ Octal

- Directly substitute an octal digit for a 3-bit binary number

$(\because 2^3 = 8)$

e.g. $0\ 110\ 110\ 100\ 011\ 111_{(2)}$

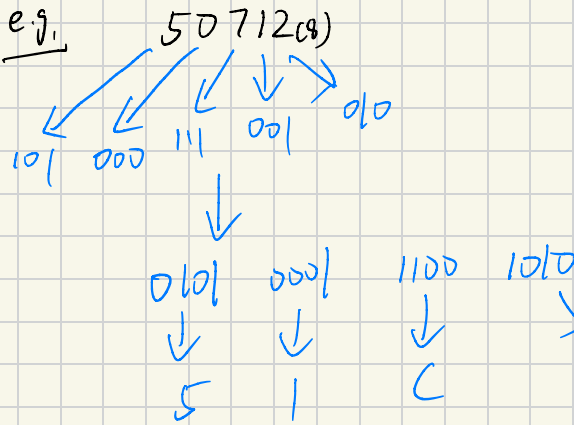
$0\ 6\ 6\ 4\ 3\ 7$

$\therefore 664378)$

Octal ↔ Hex

- use binary as an intermediate form
  - ↳ (substitute, regroup, then substitute again)

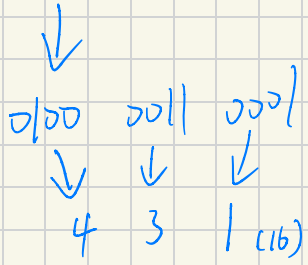
e.g.



Decimal → Octal / Decimal → Hex

- Apart from universal method, it may be <u>easier to use binary</u> as an intermediate form

e.g. $1073_{(10)} \rightarrow$ Hex



(Note: Never use <u>decimal</u> as a conversion intermediary)

Counting

Counting in Decimal

- Increment the least-significant digit until reaches "nine"
- A digit increments if all digits to its right are at their max values —— "nine"

☆ After all digits are at their max, the count wraps to zero

eg 9999 (max for a 4-digit decimal)

↓

0000

☆☆☆ Overflow

[The value we want to represent "overflowed" the number of digits can be used to represent it]

Counting in Another Radix

- The max digit value in radix R is equal to R-1

e.g. (base 10) → max value is 9
(base 2) → 1
(base 8) → 7
(base 16) → F

Arithmetic

Addition Methodology

Shortcut: If the column sum is in the same base as the numbers being added:

① Column sum one's digit is result for current col

② Column sum base's digit is carried to next col

☆WARNING: Does not work in Decimal

e.g.

$$\begin{array}{r} {\scriptstyle 1\ \ 1\ \ \ } \\ 0\ 1\ 1 \\ +\ 0\ 1\ 1 \\ \hline 1\ 1\ 0 \end{array}$$

$1+1 = 1|0_{(2)}$ (carry | sum)

$1+1+1 = 1|1_{(2)}$

$1+0+0 = 1_{(2)}$

e.g

$$\begin{array}{r} {}^{1}3\,7_{(8)} \\ +\ 1\,4_{(8)} \\ \hline 5\,3 \end{array}$$

$7+4=13_{(8)}$

$1+3+1=5_{(8)}$

---

e.g

$$\begin{array}{r} 1\,F\,9_{(16)} \\ +\ 1\,2_{(16)} \\ \hline 1\,0\,B \end{array}$$

$9+2=B_{(16)}$

$F+1=10_{(16)}$

Subtraction using Borrowing

- If necessary, borrow the base's value from one position higher

e.g

$$\begin{array}{r} {}^{2}\cancel{3}\,{}^{12}2_{(8)} \\ -\ 1\,4_{(8)} \\ \hline 1\,6 \end{array}$$

$12_{(8)}-4_{(8)}=6_{(8)}$

$2_{(8)}-1_{(8)}=1_{(8)}$

Multiply by Power of Base

- Multiplying by the base "shifts" digit positions because positions represent powers of base

e.g Binary multiplied by two:

$11 = 1 \times 2 + 1 \times 1 \longrightarrow 11 \times 2 = 1 \times 4 + 1 \times 2$

$\downarrow$

$110_{(2)}$

$\therefore$ ~~★★★★~~ Multiplying a base R number by R^n, shifts the number left by n positions

e.g $120_{(8)} \times 8^2 = 12000_{(8)}$

$11_{(2)} \times 2^4 = 110000_{(2)}$

$C_{(16)} \times 16^1 = C0_{(16)}$

Dividing by Power of Base

- same idea as multiplication, but shifts number to the right

e.g. $120_{(8)} \div 8^1 = 12_{(8)}$

$C000_{(16)} \div 16^3 = C_{(16)}$

$11000_{(2)} \div 2^4 = 1.1_{(2)}$

↓

"radix point"

Signed Numbers

Signed - Magnitude Representation

- In decimal, we show a number's sign using (+ or -) in front of its magnitude

☆ Similar approach in binary:

① The most-significant bit is the sign (1 is negative)

② remaining bits are magnitude

e.g.

sign	magnitude
0	010
+	2

sign	magnitude
1	010
−	2

Problems

① It complicates addition and subtraction

② It has a redundant representation of zero

A better representation

★ **2's – complement Representation**

- This makes math easier for computer!

① Positive numbers start with 0
   - The rest of the bits work the same as unsigned binary

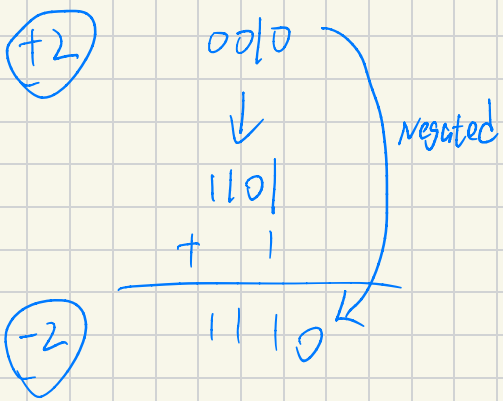
② Negative numbers start with 1
   - The interpretation is more complex

☆ Easiest to find a negative by negating the positive

↓

Complementing each bit and add 1

e.g



---

e.g

1011

↓

0100

\+ 1

0101

∴ 0101 is 5, from this, we know 1011 is -5

Double checking: 1011

\+ 0100

0000 ✓

Addition with 2's - complement

e.g

Decimal:

$$\begin{array}{r} -1 \\ +\ -1 \\ \hline -2 \end{array}$$

Binary

$$\begin{array}{r} +111 \\ +\ 111 \\ \hline 110 \end{array}$$

↙

$110 \rightarrow 011$

$$\begin{array}{r} +\ 1 \\ \hline 010 \end{array}$$

(+2 in decimal)

$\therefore$ 110 must be -2 in decimal

---

e.g

Decimal:

$$\begin{array}{r} -3 \\ +\ 5 \\ \hline 2 \end{array}$$

Binary:

$$\begin{array}{r} +11\,01 \\ +\ 01\,01 \\ \hline 00\,10 \end{array}$$

Subtraction With 2's - Complement

- Add the negation of the value we wish to subtract

e.g

Decimal:

$$\begin{array}{r} 3 \\ +\ -2 \\ \hline 1 \end{array}$$

Binary:

$$\begin{array}{r} 011 \\ +\ 110 \\ \hline 001 \end{array}$$

e.g. Decimal:

$$\begin{array}{r} 2 \\ -3 \\ \hline -1 \end{array}$$

Binary:

$$\begin{array}{r} 0\ 0\ 1\ 0 \\ +\ 1\ 1\ 0\ 1 \\ \hline 1\ 1\ 1\ 1 \end{array}$$

Operand / Result Bitwidths Match

- Operands and results must have the same number of bits for the math to work correctly

☆ Operands can be sign-extended if needed to have matching operands

Sign-Extension and Overflow

Changing the number of bits

- Same value can be represented using a different number of bits

① Zero-Extension:

- Insert zeros at most significant position

e.g. $110_{(2)} = 6_{(10)}$ / $000110_{(2)} = 6_{(10)}$

What about a 2's-complement number?

$010_{(2)} = +2_{(10)}$

$00010_{(2)} = +2_{(10)}$

But $10_{(2)} \neq 010_{(2)}$ in 2's-complement

Notice:

$010_{(2)} = +2_{(10)}$

$00010_{(2)} = +2_{(10)}$

$10_{(2)} = -3_{(10)}$

$110_{(2)} = -3_{(10)}$

$11110_{(2)} = -3_{(10)}$

☆ This is called sign-extension

- Replicate the sign bit as many times as needed

(Notes)

- The principles of sign extension can represent a value in a fewer bits (when possible)

e.g. $111101_{(2)} \rightarrow 101_{(2)}$

$000011_{(2)} \rightarrow 011_{(2)}$

Overflow

① For unsigned binary: If there is a carry out exceeds the limit of the space, it is overflow

② For signed binary: A carry-out does NOT mean overflow

For 2-bit complement binary, any addition > 3 or < -4 will be an overflow ☆☆☆

☆ A more efficient way to detect overflow is that:

If the number have the same sign, then the result's sign must also be the same