Number and Data Representation

विद्याधनं सर्वधन प्रधानम्

Subhasis Bhattacharjee
Department of Computer Science and Engineering,
Indian Institute of Technology, Jammu

August 23, 2023

Outline

- Positive Integers
- Negative Integers
- Floating-Point Numbers
- Strings

- Computers do not understand natural human languages, nor programming languages
- They only understand the language of bits

Bit	0 or 1
Byte	8 bits
Word	4 bytes
kiloByte	1024 bytes
megaByte	10^6 bytes

Decimal Number System (Base 10)

Decimal number system

- 10 symbols $\rightarrow$ { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Uses the place value system
- Base is 10

Example: Decimal Number

$$5301 = 5 * 10^3 + 3 * 10^2 + 0 * 10^1 + 1 * 10^0$$

Binary Number System

Binary number system

- 2 symbols $\rightarrow \{0, 1\}$
- Base is 2
- Uses the place value system

Example: Binary Number (Equivalent decimal number =

$$110101 = 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

Examples

Number in decimal	Number in binary
5	101
100	1100100
500	111110100
1024	10000000000

Hexadecimal Number System

Hex number system

-
- 16 symbols $\rightarrow$ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Base is 16
- Uses the place value system

Example: Hex Number (Equivalent decimal number =

$110101 = 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

MSB and LSB

- **MSB (Most Significant Bit)** $\rightarrow$ The leftmost bit of a binary number. E.g., MSB of 1110 is 1
- **LSB (Least Significant Bit)** $\rightarrow$ The rightmost bit of a binary number. E.g., LSB of 1110 is 0

Storage of Data in Memory

- Data Types
  - char (1 byte), short (2 bytes), int (4 bytes), long int (8 bytes)
- How are multibyte variables stored in memory ?
  - Example : How is a 4 byte integer stored ?
  - Save the 4 bytes in consecutive locations
  - Little endian representation (used in ARM and x86) $\rightarrow$ The LSB is stored in the lowest location
  - Big endian representation (Sun Sparc, IBM PPC) $\rightarrow$ The MSB is stored in the lowest location

Storage of Data in Memory (Ex)

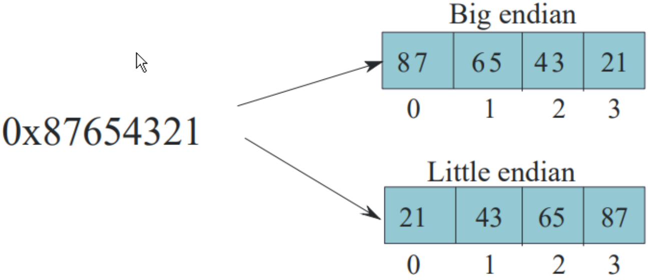
0x 12 34 56 78

	BE	LE
0x0000	12	78
0x0001	34	56
0x0002	56	34
0x0003	78	12

6 digit decimal number => 786549

B2	B1	B0
78	65	49
49	65	78

Little Endian vs Big Endian



- Note the order of the storage of bytes

Decimal to Binary

- Let N be a decimal number.
  - 1. Find the greatest number that is a power of 2 and when subtracted from N it produces a positive difference N1
  - 2. Put a 1 in the MSB
  - 3. Repeat Step 1, starting from N1 and finding difference N2 . Put a 1 in the corresponding bit. Stop when the difference is zero.

Decimal to Binary (Example)

Let N $= (717)_{10}$

$$\begin{aligned}
717-512 &= 205 = N_1 & \qquad 512 &= 2^9\\
205-128 &= 77 = N_2 & 128 &= 2^7\\
77-64 &= 13 = N_3 & 64 &= 2^6\\
13-8 &= 5 = N_4 & 8 &= 2^3\\
5-4 &= 1 = N_5 & 4 &= 2^2\\
1-1 &= 0 = N_6 & 1 &= 2^0
\end{aligned}$$

- $(717)_{10} = 2^9 + 2^7 + 2^6 + 2^3 + 2^2 + 2^0$
  - $= (1011001101)_2$

Hexadecimal and Octal Numbers

- Hexadecimal numbers
  - Base 16 numbers – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Start with 0x
- Octal Numbers
  - Base 8 numbers – 0,1,2,3,4,5,6,7
  - Start with 0 (it is o not 0)

Binary to Octal and Hexadecimal - Conversion

Binary to Octal

$8 = 2^3 \rightarrow$ every 3 binary bits convert 1 octal digit

Binary to Hexadecimal (*Hex in short*)

$16 = 2^4 \rightarrow$ every 4 binary bits convert 1 hex digit

Binary $\leftrightarrow$ Octal - Example 1 - TODO

Example: Binary $\leftrightarrow$ Octal for **Integer**

Example: Binary $\leftrightarrow$ Octal for **Fraction**

Binary ↔ Octal - Example 2 - TODO

Example: Binary ↔ Octal

Binary	011	010	101	000	.	111	101	011	100
Octal	3	2	5	0	.	7	5	3	4

Example: Octal ↔ Binary

Binary	011	010	101	000	.	111	101	011	100
Octal	3	2	5	0	.	7	5	3	4

Binary ↔ Hex

Binary	0110	1010	1000	.	1111	0101	1100
Hex	6	A	8	.	F	5	C

Convert Decimal to any base r

- Integer part: Divide by the base, keep track of remainder, and read-up
- e.g.$153_{10} = ?_8$, r = 8
  - 153 / 8 = 19 + 1/8 rem = 1 LSB
  - 19 / 8 = 2 + 3/8 rem = 3
  - 2 / 8 = 0 + 2/8 rem = 2 MSB
- $153_{10} = 231_8$

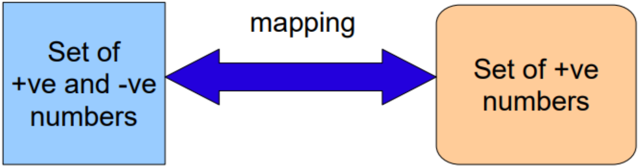
Convert Decimal to any base r (cont.)

- Fractional part: Multiply by the base, keep track of integer part, and read-down
- e.g.$0.78125_{10} = ?_{16}$, r = 16
  - 0.78125/16 = 12.5 integer = 12 = C MSB
  - 0.5 / 16 = 8.0 integer = 8 = 8 LSB
- $0.78125_{10} = 0.C8_{16}$

Representing Negative Integers

- Problem -> $2^6 = 64$ => N S
  - Assign a binary representation to a negative integer
  - Consider a negative integer, S
  - Let its binary representation be : xnxn-1. . . .x2 x1 (xi =0/1)
  - We can also expand it to represent an unsigned, +ve, number, N
  - If we interpret the binary sequence as :
    - An unsigned number, we get N
    - A signed number, we get S

- We need a mapping :
  - F : S $\rightarrow$ N (mapping function)
  - S $\rightarrow$ set of numbers (both positive and negative – signed)
  - N $\rightarrow$ set of positive numbers (unsigned)



Properties of the Mapping Function

- Preferably, needs to be a one to one mapping
- All the entries in the set, S, need to be mapped
- It should be easy to perform addition and subtraction operations on the representation of signed numbers

$$\text{SgnBit}(u) = \begin{cases} 1 \text{ , } u < 0 \\ 0 \text{ , } u >= 0 \end{cases}$$

Sign-Magnitude Base Representation

F(u) = SgnBit(u)*2_{n-1}+ |u|
Sign bit —> |u|

- Examples :
  - -5 in a 4 bit number system : 1101
  - 5 in a 4 bit number system : 0101
  - -3 in a 4 bit number system : 1011

Problems

- There are two representations for 0
  - 000000
  - 100000
- Addition and subtraction are difficult
- The most important takeaway point :
  - **Notion of the sign bit**

1's Complement Representation

F(u) = u, u >=0 OR (|u|) or (2^n -1 -|u|),u<0)

- Examples in a 4 bit number system
  - 3 $\rightarrow$ 0011 => if in 8 bit => 0-0000011
  - -3 $\rightarrow$ 1100 => if in 8 bit => 1-1111100
  - 5 $\rightarrow$ 0101
  - -5 $\rightarrow$ 1010

Problems

- Two representations for 0
  - 0000000
  - 1111111
- Easy to add +ve numbers
- Hard to add -ve numbers
- Point to note :
  - **The idea of a complement**

Example

$$\begin{array}{c} 0011 \\ 0010 \\ \hline 0101 \\ \hline \end{array}$$

3 + (-2) = 1

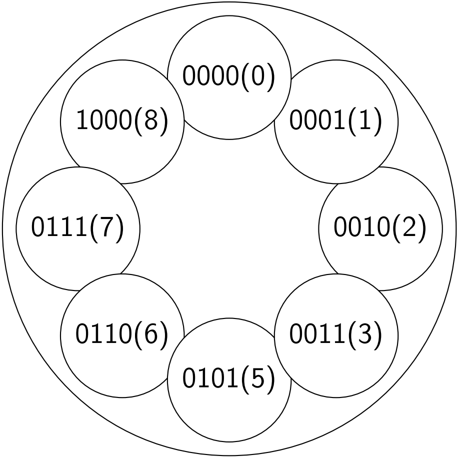
$$\begin{array}{c} 0011 \\ 1101 \\ \hline 0000 \\ \hline \end{array}$$

Bias Based Approach

$F(u) = u + bias$

- Consider a 4 bit number system with bias
- -3 $\rightarrow$ 0100
- 3 $\rightarrow$ 1010
- $F(u + v) = F(u) + F(v) - bias$
- Multiplication is difficult

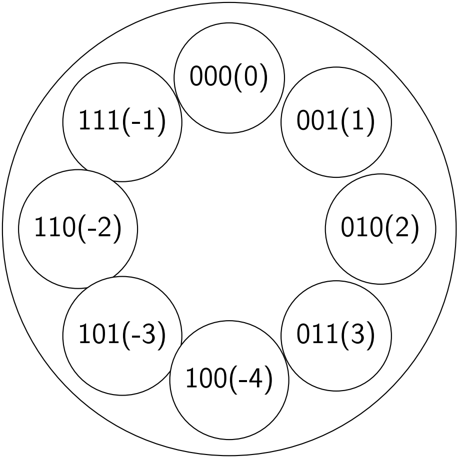
The Number Circle



Clockwise: increment

Anti-clockwise: decrement

Number Circle with Negative Numbers



————————break point

Using the Number Circle

- To add M to a number, N
  - locate N on the number circle
  - If M is +ve
    - Move M steps clockwise
  - Move M steps clockwise
  - If M is -ve
    - Move M steps anti-clockwise, or $2n - M$ steps clockwise
  - If we cross the break-point
    - We have an overflow
    - The number is too large/ too small to be represented

2's Complement Notation

$$F(u) = \begin{cases} u, 0 \le u \le 2^{n-1} - 1 \\ 2^n - |u|, -2^{n-1} \le u < 0 \end{cases}$$

- F(u) is the index of a point on the number circle. It varies from 0 to $2^{(n)}$ - 1
- Examples
- 4 $\rightarrow$ 0100
- -4 $\rightarrow$ 1100
- 5 $\rightarrow$ 0101
- -3 $\rightarrow$ 1101

Properties of the 2's Complement Notation

- Range of the number system :
  - -$2^{(n-1)}$ to $2^{(n-1)} - 1$
- There is a unique representation for 0 $\rightarrow$ 000000
- msb of F(u) is equal to SgnBit(u)
  - Refer to the number circle
  - For a +ve number, F(u) $< 2^{(n-1)}$. MSB $= 0$
  - For a -ve number, F(u) $>= 2^{(n-1)}$. MSB $= 1$

Properties - II

- Every number in the range [-$2^{(n-1)}$,$2^{(n-1)}$– 1]
  - Has a unique mapping
  - Unique point in the number circle
- F(-u) =$2^{(n)}$ – F(u)
  - Moving F(u) steps counter clock wise is the same as moving $2^{(n)}$ – F(u) steps clockwise from 0

Subtraction

- F(u-v) = F(u) + F(-v) = F(u) + $2^{(n)}$ - F(v)
- Subtraction is the same as addition
- Compute the 2's complement of F(v)

Computing the 2's Complement

- $2^{(n)} - u = 2^{(n)} - 1 - u + 1 = \tilde{}\ u + 1$
- ˜ u (1's complement)
- 1's complement of 0100

$$\begin{array}{c} 1111 \\ 0100 \\ \hline 1011 \end{array}$$

- 2's complement of 0100

- 1's complement of 0100 is 1011

$$\begin{array}{c} 1011 \\ 0001 \\ \hline 1100 \end{array}$$

Sign Extension

- Convert a n bit number to a m bit $2^{(s)}$ complement number ($m > n$)
- +ve
  - Add (m-n) 0s in the msb positions
  - Example, convert 0100 to 8 bits $\rightarrow$ 0000 0100
- -ve
  - $F(u) = 2^{(n)} - |u|$ (n bit number) system
  - Need to calculate $F'(u) = 2^{(m)} - |u|$

Sign Extension - II

- $2^{(m)} - u - (2^{(n)} - u)$
  $= 2^{(m)} - 2^{(n)}$
  $= 2^{(n)} + 2^{(n+1)} + \ldots + 2^{(m-1)}$
  $= 11110000$
  (11 =m-n) (110=n)

$F'(u) = F(u) + 2^{(m)} - 2^{(n)}$

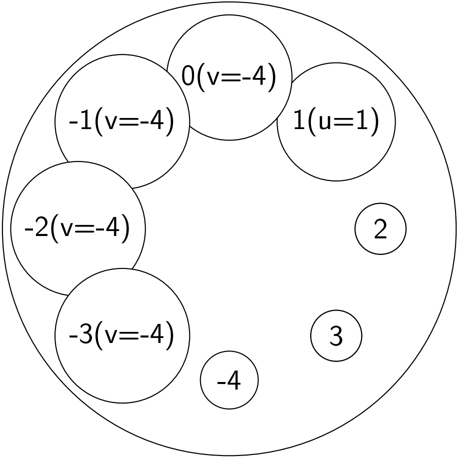
Sign Extension - III

- To convert a negative number :
  - Add (m-n) 1s in the msb positions
- In both cases, extend the sign bit by :
  - (m-n) positions

The Overflow Theorem

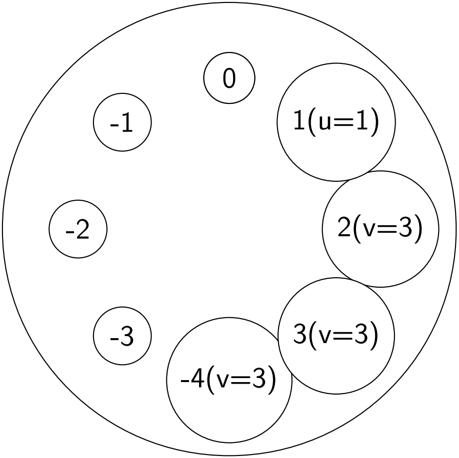
- Add : u + v
- If uv $<$ 0, there will never be an overflow
- Let us go back to the number circle
  - There is an overflow only when we cross the break-point
- If uv $=$ 0, one of the numbers is 0 (no overflow)
- If uv $>$ 0, an overflow is possible

Number Circle: uv < 0



————————break point

Number Circle: uv > 0



——————overflow

Conditions for an Overflow

- uv <= 0
  - Never
- uv > 0 (u and v have the same sign)
  - The sign of the result is different from the sign of u

Floating-Point Numbers

- What is a floating-point number ?
  - 2.356
  - 1.3e-10
  - -2.3e+5
- What is a fixed-point number ?
  - Number of digits after the decimal point is fixed
  - 3.29, -1.83

Generic Form for Positive Numbers

- Generic form of a number in base 10

$$A = \sum_{i=-n}^{n} x_i 10^i$$

- Example :
- $3.29 = 3 * 10^{(0)} + 2*10^{(-1)} + 9*10^{(-2)}$

Generic Form in Base 2

- Generic form of a number in base 2

$$A = \sum_{i=-n}^{n} x_i 2^i$$

Number	Expansion
0.375	$2^{(-2)} + 2^{(-3)}$
1	$2^{(0)}$
1.5	$2^{(0)} + 2^{(-1)}$
2.75	$2^{(1)} + 2^{(-1)} + 2^{(-2)}$
17.625	$2^{(4)} + 2^{(0)} + 2^{(-1)} + 2^{(-3)}$

Binary Representation

- Take the base 2 representation of a floating-point (FP) number
- Each coefficient is a binary digit

Number	Expansion	BinaryRepresentation
0.375	$2^{(-2)} + 2^{(-3)}$	0.011
1	$2^{(0)}$	1.0
1.5	$2^{(0)} + 2^{(-1)}$	1.1
2.75	$2^{(1)} + 2^{(-1)} + 2^{(-2)}$	10.11
17.625	$2^{(4)} + 2^{(0)} + 2^{(-1)} + 2^{(-3)}$	10001.101

Normalized Form

- Let us create a standard form of all floating point numbers
  A = $(-1)^{(s)}$ *P*$2^{(X)}$,(P =1+ M,0 <= M <1, X E Z)
- S $\rightarrow$ sign bit, P $\rightarrow$ significand,E=belongs to symbol
- M $\rightarrow$ mantissa, X $\rightarrow$ exponent, Z $\rightarrow$ set of Integers

Examples (in decimal)

- 1.3827 * 1e-23
  - Significand (P) = 1.3827
  - Mantissa (M) = 0.3827
  - Exponent (X) = -23
  - Sign (S) = 0
- -1.2*1e+5
  - P = 1.2 , M = 0.2
  - S = 1, X = 5

IEEE 754 Format)

- General Principles
  - The significand is of the form : 1.xxxxx
  - No need to waste 1 bit representing (1.) in the significand
  - We can just save the mantissa bits
  - Need to also store the sign bit (S), exponent (X)

IEEE 754 Format - II)

Sign(S)	Exponent(X)	Mantissa(M)
1	8	23

- sign bit – 0 (+ve), 1 (-ve)
- exponent, 8 bits
- mantissa, 23 bits

Representation of the Exponent

- Biased representation
  - bias = 127
  - E = X + bias
- Range of the exponent
  - 0 – 255 <—> -127 to +128
- Examples :
  - X = 0, E = 127
  - X = -23, E = 104
  - X = 30 , E = 157

Normal FP Numbers

- Have an exponent between -126 and +127
- Let us leave the exponents : -127, and +128 for special purposes.
  A = $(-1)^S$ *P*2 E-bias
  (P =1+ M,0 <= M <1, X belongs to Z,1<= E <= 254)
- What is the largest +ve normal FP number ?
- What is the smallest –ve normal FP number ?

Special Floating Point Numbers

Number	Expansion	BinaryRepresentation
255	0	if S=0
255	0	- if s=1
255	0	NAN (Not a number)
0	0	0
0	0	Denormal number

- NAN + x= NAN 1/0 = infinite
- 0/0 = NAN -1/0 = - infinite
- sin^{-1} (5) = NAN

Denormal Numbers

```
f = 2^(−126);
g = f/2;
if (g == 0)
print ("error");
```

- Should this code print "error" ?
- How to stop this behaviour ?

Denormal Numbers - II

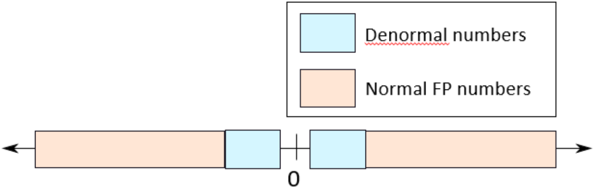
A = (-1) S *P*$2^{(-126)}$
(P = 0+ M,0 <= M <1)

- Significand is of the form : 0.xxxx
- E = 0, X = -126 (why not -127?)
- Smallest +ve normal number : $2^{(-126)}$
- Largest denormal number :
- 0.11...11 * $2^{(-126)} = (1 - 2^{(-23)})*2^{(-126)}$
  $=2^{(-126)}$ - $2^{(-149)}$

Example

- Find the ranges of denormal numbers.
  Answer
  - For positive denormal numbers, the range is $[2^{(-149)}, 2^{(-126)} - 2^{(-149)}]$
  - For negative denormal numbers, the range is $[-2^{(-149)}, -2^{(-126)} + 2^{(-149)}]$

Denormal Numbers in the Number Line



Extend the range of normal floating point numbers.

Double Precision Numbers

Field	Size(bits)
S	1
E	11
M	52

Field Size(bits)

- Approximate range of doubles
  - $\pm$ 2 1023 = $\pm$ 10308
  - This is a lot !!!

Floating Point Mathematics

A = $2^{(50)}$;
B = $2^{(10)}$;
C = (B+A)- A;

- C will be computed to be 0
    - There is no way of representing A+B in the IEEE 754 format
- A smart compiler can reorder the operations to increase precision
- Floating point math is approximate

Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- A decimal code: Decimal numbers (0..9) are coded using 4-bit distinct binary words
- Observe that the codes 1010 .. 1111 (decimal 10..15) are NOT represented (invalid BCD codes)

Binary-Coded Decimal (cont.)

- To code a number with n decimal digits, we need 4n bits in BCD e.g. $(365)10 = (0011\ 0110\ 0101)BCD$
- This is different to converting to binary, which is $(365)10 = (101101101)2$
- Clearly, BCD requires more bits. BUT, it is easier to understand/interpret

BCD Addition

- When 2 BCD codes are added:
  - If the binary sum is less than 1010 (9 in decimal), the corresponding BCD sum digit is correct
  - If the binary sum is equal or more than 1010, must add 0110 (6 in decimal) to the corresponding BCD sum digit in order to produce the correct carry into the digit to the left

BCD Addition (cont.)

- Example: Add 448 and 489 in BCD.

0100 0100 1000 (448 in BCD)
0100 1000 1001 (489 in BCD)
10001 (greater than 9, add 6)
1 0111 (carry 1 into middle digit)
1101 (greater than 9, add 6)
1001 1 0011 (carry 1 into leftmost digit)
1001 0011 0111 (BCD coding of 93710)

ASCII Character Set

- ASCII – American Standard Code for Information Interchange
- It has 128 characters
- First 32 characters (control operations)
  - backspace (8)
  - line feed (10)
  - escape (27)
- Each character is encoded using 7 bits

ASCII Character Set

Character	Code	Character	Code	Character	Code
a	97	A	65	0	48
b	98	B	66	1	49
c	99	C	67	2	50
d	100	D	68	3	51
e	101	E	69	4	52
f	102	F	70	5	53
g	103	G	71	6	54
h	104	H	72	7	55
i	105	I	73	8	56
j	106	J	74	9	57
k	107	K	75	!	33
l	108	L	76	#	35
m	109	M	77	$	36
n	110	N	78	%	37
o	111	O	79	&	38
p	112	P	80	(	40
q	113	Q	81	)	41
r	114	R	82	*	42
s	115	S	83	+	43
t	116	T	84	,	44
u	117	U	85	.	46
v	118	V	86	;	59
w	119	W	87	=	61
x	120	X	88	?	63
y	121	Y	89	@	64
z	122	Z	90	^	94

Unicode Format

- UTF-8 (Universal character set Transformation Format)
  - UTF-8 encodes 1,112,064 characters defined in the Unicode character set. It uses 1-6 bytes for this purpose.
  - UTF-8 is compatible with ASCII. The first 128 characters in UTF-8 correspond to the ASCII characters. When using ASCII characters, UTF-8 requires just one byte. It has a leading 0.
  - Most of the languages that use variants of the Roman script such as French, German, and Spanish require 2 bytes in UTF-8. Greek, Russian (Cyrillic), Hebrew, and Arabic, also require 2 bytes.

UTF-16 and 32

- Unicode is a standard across all browsers and operating systems
- UTF-8 has been superseded by UTF-16, and UTF-32
- UTF-16 uses 2 byte or 4 byte encodings (Java and Windows)
- UTF-32 uses 4 bytes for every character (rarely used)