Chapter 3

Data Representation

Topics:

- Fixed point numbers
- Floating point numbers
- Characters

Motivation

Computers are able to

- Perform arithmetic operations,
- Process character strings (text processing),
- Display graphics and movies, play sound

Software algorithms are formulated at high levels and operate on abstract objects. However, in the end all objects have to be represented in the machine using sequences of bits.

Number Systems

Definition

A *number system* can be defined as a triple $S = (b, Z, \delta)$ with the following properties:

- $b \geq 2$ is called the ***basis*** of the number; b is a natural number.
- Z is a set consisting of b *digits.*
- $\delta : Z \rightarrow \{0, 1, ..., b-1\}$ is a transformation assigning a natural number from the range $[0$ to $b-1]$ to each digit of Z.

Number systems

Examples

- **Decimal system** $b = 10$, $Z = \{0,1,2,3,4,5,6,7,8,9\}$
- **Binary system** $b = 2$, $Z = \{0,1\}$
- **Octal system** $b = 8$, $Z = \{0,1,2,3,4,5,6,7\}$
- **Hexadecimal system**:

  $b = 16$ $Z = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

  $A \rightarrow 10$, $B \rightarrow 11$, $C \rightarrow 12$, ...

Fixed point numbers

Definition:

A *fixed point number* can be defined as

a finite sequence of digits of a number system to the basis b and with the set of digits Z.

It consists of n digits left of the decimal point ($n > 0$) and $k \geq 0$ digits right of the decimal point.

The value $<d>$ of a non-negative fixed point number

$$d = d_{n-1}\, d_{n-2} \ldots\, d_1 d_0 \quad d_{-1} \ldots\, d_{-k}$$

with $d_i \in Z$ is defined as

$$< d >= \sum_{i=-k}^{n-1} b^i \cdot \delta(d_i)$$

Fixed point numbers

Notation

Digits left and right of the decimal point are separated by a point or a comma:

$$d = d_{n-1}\, d_{n-2} \ldots.\; d_1 d_0 \,.\, d_{-1} \ldots.\; d_{-k}$$

To clarify which number system is used the basis of the number system sometimes is appended as an index to the sequence of digits.

Example: 0110_2

Fixed point numbers

Examples

Given: $n = 2$, $k = 2$, $d =$ 1010

$b = 2$	$<d> = 2.5_{10}$
$b = 8$	$<d> = 8.125_{10}$
$b = 10$	$<d> = 10.1_{10}$
$b = 16$	$<d> = 16.0625_{10}$

All computers use the binary number system. In the following we will always assume: $b = 2$.

Decimal to Binary Conversion

Dividing digits left of decimal point by *b* yields ***d_0***:

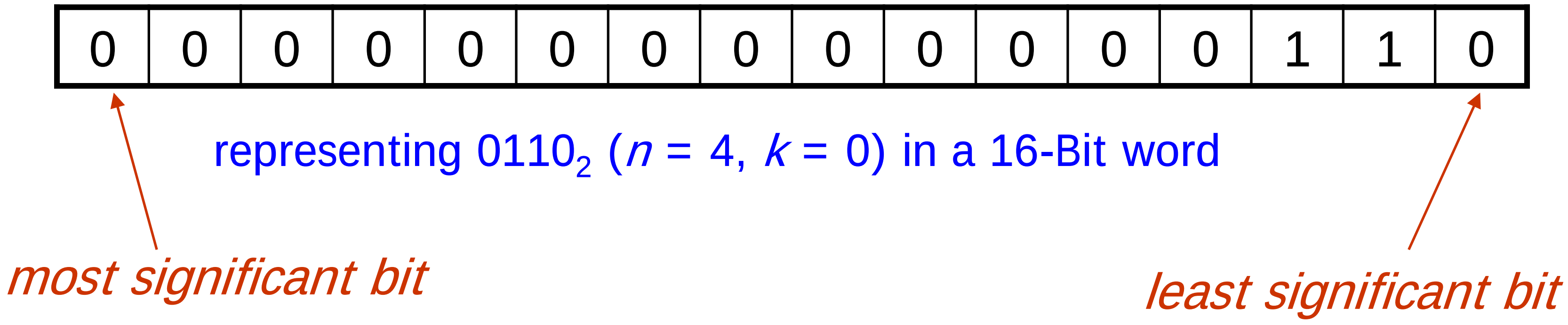
$$< d > = \sum_{i=0}^{n-1} d_i \cdot b^i \quad \Longrightarrow \quad \frac{< d >}{b} = \sum_{i=0}^{n-2} d_{i+1} \cdot b^i \quad REM : d_0$$

Multiplying digits right of decimal point by *b* yields ***d_{-1}***:

$$< d > = \sum_{i=-k}^{-1} d_i \cdot b^i \quad \Longrightarrow \quad < d > \cdot b = d_{-1} + \sum_{i=-k}^{-2} d_{-i} \cdot b^{i+1}$$

Fixed point numbers

- Number of digits available for representing numbers is restricted by the word width of the data path (e.g. 16, 32 or 64)



representing 0110_2 (n = 4, k = 0) in a 16-Bit word

- If the result of an operation does not fit in the given number of digits an *overflow* occurs
  $\Rightarrow$ detected by HW, handled by SW

Signed fixed point numbers

There are four methods commonly used to represent signed numbers:

- **Sign and magnitude**

  $<d_{n-1}, \ldots, d_0, d_{-1}, \ldots, d_{-k}> \; = (-1)^{d_{n-1}} \sum_{i=-k,\ldots,n-2} d_i 2^i$

- **One‘s complement**

  $<d_{n-1}, \ldots, d_0, d_{-1}, \ldots, d_{-k}> \; = \sum_{i=-k,\ldots,n-2} d_i 2^i - d_{n-1} (2^{n-1} - 2^{-k})$

- **Two‘s complement**

  $<d_{n-1}, \ldots, d_0, d_{-1}, \ldots, d_{-k}> \; = \sum_{i=-k,\ldots,n-2} d_i 2^i - d_{n-1} 2^{n-1}$

- **Bias**

  $<d_{n-1}, \ldots, d_0, d_{-1}, \ldots, d_{-k}> \; = \sum_{i=-k,\ldots,n-1} d_i 2^i - bias$

  typically: $bias = 2^{n+k-1}$ or $2^{n+k-1} - 1$

Sign and Magnitude

$$<d_{n-1},\ldots,d_0,d_{-1},\ldots,d_{-k}> \;=\; (-1)^{d_{n-1}} \sum_{i=-k,\ldots,n-2} d_i 2^i$$

Example: $n = 3$, $k = 0$

a	000	001	010	011	100	101	110	111
<a>	0	1	2	3	0	-1	-2	-3

Properties:

- range of numbers is symmetric:

Smallest number:	$-(2^{n-1} - 2^{-k})$
Largest number:	$2^{n-1} - 2^{-k}$

- Two possibilities for representing zero: **000** and **100** (for $n = 3$, $k = 0$).

One‘s complement

$$<d_{n-1},\ldots,d_0,d_{-1},\ldots,d_{-k}> = \sum_{i=-k,\ldots,n-2} d_i 2^i - d_{n-1}(2^{n-1}-2^{-k})$$

Example: $n = 3$, $k = 0$

a	000	001	010	011	100	101	110	111
<a>	0	1	2	3	-3	-2	-1	-0

Properties:

- range of numbers is symmetric:

Smallest number:	$-(2^{n-1} - 2^{-k})$
Largest number:	$2^{n-1} - 2^{-k}$

- Two possibilities for representing zero: **000** and **111** (for $n = 3$, $k = 0$).
- “adjacent numbers” have the same distance 2^{-k}.

Let *a* be a fixed point number and let *a’* be another fixed point number being derived from *a* by complementing all digits ($0 \rightarrow 1$, $1 \rightarrow 0$). Then assuming one’s complement representation $<a'> = -<a>$ holds.

Two‘s complement

$$<d_{n-1},\ldots,d_0,d_{-1},\ldots,d_{-k}> \;=\; \sum_{i=-k,\ldots,n-2} d_i 2^i - d_{n-1} 2^{n-1}$$

Example: $n = 3$, $k = 0$

a	000	001	010	011	100	101	110	111
<a>	0	1	2	3	-4	-3	-2	-1

Properties:

- Range of numbers is not symmetric:
  - Smallest number: $-\ 2^{n-1}$
  - Largest number: $2^{n-1}-2^{-k}$
- Unique representation
- “Adjacent numbers” have the same distance 2^{-k}.

Benefit of two‘s complement:

Simple HW for performing addition/subtraction of signed numbers

$\Rightarrow$ commonly used for representing signed integer numbers

Let *a* be a fixed point number and let *a’* be another fixed point number being derived from *a* by complementing all digits ($0 \rightarrow 1$, $1 \rightarrow 0$). Then assuming two’s complement representation $<a'> + 2^{-k} = - <a>$ holds.

Biased representation

$$<d_{n-1},\ldots,d_0,d_{-1},\ldots,d_{-k}> = \sum_{i=-k,\ldots,n-1} d_i 2^i - \text{bias}$$

Example: $n = 3$, $k = 0$, bias $= 2^{n-1}-1$

a	000	001	010	011	100	101	110	111
<a>	-3	-2	-1	0	1	2	3	4

Properties:

- Range of numbers is not symmetric:
  - Smallest number: $-\,bias$
  - Largest number: $2^n - 2^{-k} - bias$
- Unique representation
- "Adjacent numbers" have the same distance 2^{-k}.

Benefit of biased representation:

Sorting can be done in the same way as for unsigned numbers

Multiplying unsigned fixed point numbers

$$a = \sum_{i=-k}^{n-1} a_i \cdot 2^i \qquad b = \sum_{j=-k}^{n-1} b_j \cdot 2^j \qquad a \cdot b = \sum_{i=-k}^{n-1} \sum_{j=-k}^{n-1} a_i \cdot b_j \cdot 2^{i+j}$$

Example: $n = 4$, $k = 0$

a_3	a_2	a_1	a_0	*	b_3	b_2	b_1	b_0
					a_3b_0	a_2b_0	a_1b_0	a_0b_0
+				a_3b_1	a_2b_1	a_1b_1	a_0b_1	
+			a_3b_2	a_2b_2	a_1b_2	a_0b_2		
+		a_3b_3	a_2b_3	a_1b_3	a_0b_3			
	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

Multiplying two's complement numbers

1. Consider sign bits; If operands *a* and *b* are negative negate them by creating two's complement

2. Multiply |a| and |b|

3. Determine sign of result: $\text{sign}_{\text{result}} = \text{sign}_a \oplus \text{sign}_b$

4. Negate result if necessary

Limitations of Fixed Point numbers

Consider two's complement numbers with *n* digits left of the decimal point and *k* digits right of the decimal point.

- Operations are not closed!

  The result of $2^{n-1}+2^{n-1}$ can not be represented although 2^{n-1} can be represented

- Even if the final result of an arithmetic operation can be represented an intermediate result may be out of range.

  $\Rightarrow$ **Associative law and distributive law do not hold!**

  Example: $n = 4$, MAX = 7, MIN = -8

  $7 + (4 – 6) = 7 - 2 = 5$ OK! but: $(7 + 4) – 6 = \mathbf{11} - 6$

  $\uparrow$ out of range!

- Very large and very small numbers cannot be represented!

Floating point numbers

- Location of decimal point not fixed!
- Two components: mantissa and exponent:

$$\textbf{Mantissa x } 2^{\text{Exponent}}$$

- Given a particular number of digits floating point numbers cover a larger range of numbers than fixed point numbers

IEEE 754 Standard:

- IEEE single precision: 32 bits

31	30 29 28 27 26 25 24 23	22 21 . . . 3 2 1 0
S	Exponent E	Mantissa M

- IEEE double precision: 64 bits

63	62 61 . . . 53 52	51 50 . . . 3 2 1 0
S	Exponent E	Mantissa M

$$(-1)^S \times M \times 2^E$$

Normalized floating point representation

Note:

In general the floating point representation of a particular number is **not** unique!

Example: $0.111 \times 2^3 = 0.0111 \times 2^4$

Definition

A floating point number (S,M,E) is ***normalized***, if $1.0 \leq M < 2.0$ holds, e.g.: $M = 1.m_{-1} \ldots m_{-k}$

$$\boxed{\text{The 1 on the left side of the decimal point is not stored } (\rightarrow \text{„hidden bit“})}$$

31	30 29 28 27 26 25 24 23	22 21 . . . 3 2 1 0
S	Exponent E	Mantisse m
		$m_{-1}\ m_{-2}$ ... ... m_{-23}

The value of the mantissa of a normalized floating point number results to:

$$M = 1 + \sum_{i=-1,\ldots,-k} m_i 2^i.$$

$\Rightarrow$ Zero has to be represented in a special way!

Floating point numbers - IEEE 754 standard

31	30 29 28 27 26 25 24 23	22 21 . . . 3 2 1 0
S	Exponent E	Mantisse m

$e_7\ e_6 \quad \ldots \quad e_0$

- IEEE 754: exponent represented as *biased* fixed point number
- For *n* -bit exponent $BIAS = 2^{n-1}-1$,

  $\Rightarrow$ single precision BIAS = 127

  $\Rightarrow$ double precision BIAS = 1023.
- Value of exponent:

$$E = \sum_{i=0,\ldots,n-1} e_i 2^i - BIAS$$

Special cases defined by the IEEE 754 Standard

The biased exponents 0 and 2^n-1 are used to represent special values:

Exponent 2^n-1:

All mantissa bits are 0:	represents $\pm\infty$.
Some Mantissa bits are 1:	represents ***NaN*** (not a number)

Exponent 0:

“hidden bit” of mantissa is not used; the value of the mantissa then is

$$M = \sum_{i=-k}^{-1} m_i \cdot 2^{i+1}$$

$\Rightarrow$ Allows for representation of *denormalized numbers* being smaller than the smallest normalized number

$\Rightarrow$ Allows for representation of 0: all bits of mantissa and exponent are 0

Floating point numbers

	single precision	double precision
#Sign bit	1	1
#Bits of exponent	8	11
#Bits of mantissa (without hidden Bit)	23	52
Total #Bits	32	64
Bias	127	1023
Range of exponent	-126 to 127	-1022 to 1023
Smallest normalized number		
Largest normalized number	?	
Smallest denormalized number		
Largest denormalized number		

Addition of floating point numbers

Algorithm:

- Align the smaller exponent to the larger exponent
- Add mantissas
- Normalize, round (if necessary)

Example

$$
\begin{aligned}
+(1.000)_2\times 2^{-1} + -(1.110)_2\times 2^{-2} &= +(1.000)_2\times 2^{-1} + -(0.111)_2\times 2^{-1} \\
&= +(0.001)_2\times 2^{-1} \\
&= +(1.000)_2\times 2^{-4}
\end{aligned}
$$

Multiplying floating point numbers

Algorithm:

- Multiply signs
- Multiply mantissas
- Add exponents (*and subtract Bias!*)
- Normalize, round (if necessary)

Example:

$+(1.000)_2 \times 2^{-1} \times -(1.110)_2 \times 2^{-2}$

Multiply signs: $0 \oplus 1 = 1$

Multiply mantissas: $(1.000)_2 \times (1.110)_2 = (1.110)_2$

Add exponents: $-1 + (-2) = -3$

Result: $-(1.110)_2 \times 2^{-3}$

Representing characters

code in hexadecimal representation

character

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	’
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	‘	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

(7-Bit) code for representing characters in computers

American Standard Code for Information Interchange (ASCII)

binary: 101 1011