3 コンピュータ内での情報の表現

3–1 2進法による情報の表し方

情報の最小単位:

現在のほとんどの計算機はフリップフロップ，すなわち

2つの安定した状態(e.g.電圧の高低,磁化の向き)を持つ素子

を基本要素にして構成されている。

[この様な素子の構成の容易さや動作の安定性などの理由による。]

フリップフロップを用いた情報表現:

数値や文字,プログラムなどのデータはフリップフロップを複数個組み合わせて表される。 多くの計算機では、通常、

数値は32ビット, (bit;BInary digiTの略で、フリップフロップ何個分であるかを表す記憶容量の単位)

英数字は8ビット,

漢字は16(または24)ビット

で表される。 例えば 、0100 0001 という列で “A” という文字を表す。

2進法による情報の表現:

フリップフロップの持つ２つの安定した状態が物理的にどんなものであるかは、記憶素子について議論する場合以外は重要でない。

⇨ 以下では、フリップフロップの**2つの安定状態を 0 と 1 で表す**。

⇨ 全てのデータは 0 と 1 の列によって表される。
この表記法を**2進法**という。

（以下では、
10進法と区別するために
0 と 1 の列は例えば B’10101101111’ と表す。）

8進法と16進法:

2進法ではデータを表すのに字数が長くなり、
我々人間にとって不便である(i.e.分かりにくく間違え易い)。

⇨ 0 と 1 の列を**3~ 4ビット毎に区切って**
それぞれの区画を1文字で表す方がコンパクトで分かり易い。

⇨ 次の左側の表に従って3ビットの列の各々を 0~ 7 の8文字で代用する表記法を8進法,
右側の表に従って4ビットの列の各々を 0~ 9,A~ F の16文字で代用する表記法を16進法という。

2進法	8進法
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

2進法	16進法
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

例えば 、B'101011010111' は

	(この授業)	C言語
8進法では 101 011 010 111 と見て 5327, ···	O'5327'	05327
16進法では 1010 1101 0111 と見て AD7 ···	X'AD7	0xAD7

と表される。

2進法	8進法
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

2進法	16進法
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

3–2 コンピュータ内での文字データの表現

計算機を利用する際、

ビット列 (i.e. 0 と 1 の列) を入力してビット列の出力を得る

というのでは不便である。

文字列を入力して文字列の出力を得ることが出来ないといけない。

⇨ 6~ 8ビットを組み合わせて英数字や記号1文字を表す方式が
いくつか定められている。

(英数字何文字を記憶できるかを表す容量の単位としてはバイト (byte) が用いられる。

通常、1バイト=8ビット である。[JISでも、....。])

次に、一般的な文字符号体系を3つ示す。

- EBCDIC符号体系 :
  IBM社が1964年に定義したもので、IBMの汎用計算機 (とその互換機) で採用された。具体的なコード表は省略。

- **ASCII7ビット符号体系**(American Standard Code for ...) :

下位4ビット	上位3ビット 0	1	2	3	4	5	6	7
0	NUL	DLE	空白	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

機能文字 ... 機能文字

文字 ”K” の場合は、
上位3ビットが O”4” で、
下位4ビットが X”B”
なので、内部では
B”100 1011”
と表される。

補足：

機能文字 (functional character) は書式の変更や伝送データの開始・終了などの制御を行うための文字で、制御文字 (control —) とも言う。それぞれの名称／意味は次の通り。

NUL…空白 (null)
SOH…ヘディング開始 (start of heading)
STX…テキスト開始 (start of text)
ETX…テキスト終了 (end of text)
EOT…伝送終了 (end of transmission)
ENQ…問合せ (enquiry)
ACK…肯定応答 (acknowledge)
BEL…ベル (bell)
BS …後退 (backspace)
HT …水平タブ (horizontal tabulation)
LF …改行 (line feed)
VT …垂直タブ (vertical tabulation)
FF …書式送り (form feed)
CR …復帰 (carriage return)
SO …シフトアウト (shift out)
SI …シフトイン (shift in)

DLE …伝送制御拡張 (data link escape)
DC1 …装置制御 1(device control 1)
DC2 …装置制御 2(device control 2)
DC3 …装置制御 3(device control 3)
DC4 …装置制御 4(device control 4)
NAK…否定応答 (negative acknowledge)
SYN …同期信号 (synchronous idle)
ETB …伝送ブロック終結 (end of transmission block)
CAN…取消 (cancel)
EM …媒体終端 (end of medium)
SUB …置換文字 (substitute character)
ESC …拡張 (escape)
FS …ファイル分離文字 (file separator)
GS …グループ分離文字 (group separator)
RS …レコード分離文字 (record separator)
US …ユニット分離文字 (unit separator)
DEL …抹消 (delete)

- **JIS8ビット符号体系**(Japanese Industrial Standard):

下位4ビット＼上位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	空白	0	@	P	`	p	未定義			ー	タ	ミ	未定義	
1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
2	STX	DC2	"	2	B	R	b	r			「	イ	ツ	メ		
3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
4	EOT	DC4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BEL	ETB	’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	LF	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	-	=	M	]	m	}			ュ	ス	ヘ	゛		
E	SO	RS	.	>	N	^	n	~			ョ	セ	ホ	゜		
F	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ			

X’5C’,X’7E’以外はASCIIと同じ

最後に、日本語符号体系としては次の4つがある。

- **JIS** 漢字符号体系：
  JIS漢字符号は**2バイト**/16ビットのビット列で構成され、登録可能な漢字数2^{16}=65536の内第1水準漢字(平仮名,片仮名,英数字, 特殊文字も含む)として2965字, 第2水準漢字として3390字, 補助漢字として5810字, 計12156字が定められている。

  例えば 「ＪＩＳ漢字コード」という文字列は
  X‘234A 2349 2353 3441 3B7A 2533 213C 2549’という16バイトのビット列で表される。

  JIS 8ビット符号と漢字符号を混ぜて使うために、
  　漢字符号の開始を表すコード,
  　8ビット符号の開始を表すコード
  を用いる方法が定められている。

  JIS符号体系を用いる利点は 最上位ビットを使っていないということである。通信の際は最上位ビットが失われることが多いので、漢字を含む文書の送信はJISコードで行うのがよい。

- **MS漢字コード**体系：
  シフトJISコード系とも呼ばれるが、JISでなく...。
  Escシーケンスを用いずに漢字符号と8ビット符号を混在させられる様に、定義されたコード体系であり、長い間パソコン/Windowsの世界では事実上の標準になっていた。

  JIS 8ビット符号で未定義の X'81'~ X'9F', X'E0'~ X'FC' のそれぞれにもう1バイト繋げることにより**2バイトで**1つの漢字を表す。

- **EUC**(Extended Unix Code)コード体系：
  UNIX-JISコード,または**UJIS**とも呼ばれている。
  日本語UNIXシステム諮問委員会の検討の結果を基に1986年にAT&T社が定めたコード体系であり、長い間(2007年頃まで?) **UNIX/Linux**における標準コードとして用いられていた。

  2~ 3バイトで漢字1文字を表す。

- **Unicode体系：**
  世界各国の文字体系を共通化するために、Unicodeコンソーシアムが提唱したもので、1993年にはISO の標準に、1995年にはJIS規格となった。

  Unicodeでは、漢字, アルファベット, 中国語, ハングル語, などの文字をひとまとめに取り扱おうとする。

  日本語に関しては従来の第1水準, 第2水準, 補助漢字のコードがUnicodeの中でもほぼそのまま使われているが、意味を無視して、外見がほぼ同じ中国の漢字と日本の漢字に同じコードを割り当てるといったことも行なわれているため、賛同しない人もいた。幾つかの符号化方式があるが、その内
  UTF-8は1~ 4バイトにエンコードしてASCIIに上位互換する方式で、**最も一般的に利用**されている。また、
  UTF-16は最初の65536文字を16ビットで表し, それ以外を32ビットで表すもので、**WindowsではXP以降**で採用されている。
  UTF-32では各々の文字を32ビットで表す。

3–3 2進法による非負整数の表現

10進法では 0~ 9 の数字列 $d_n d_{n-1} \cdots d_1 d_0$ によって

$$d_n \times 10^n + d_{n-1} \times 10^{n-1} + \cdots + d_1 \times 10^1 + d_0 \times 10^0$$

という非負整数を表す。例えば

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

である。

これに対して、

2進法では 0~ 1 の数字列 $a_n a_{n-1} \cdots a_1 a_0$ によって

$$a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_1 \times 2^1 + a_0 \times 2^0$$

という非負整数を表す。 例えば、2進法の 11010 は(10進)非負整数

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 26$$

を表す。

10進→2進変換の実際的方法:

非負整数 x が2進法で $B'a_n a_{n-1} \cdots a_1 a_0'$ と表される場合、

$$x = a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_1 \times 2^1 + a_0 \times 2^0$$

であるから各 a_i は x と i を用いて

$$a_i = \mathrm{mod}\left(\left\lfloor \frac{x}{2^i} \right\rfloor, 2\right)$$

但し $\lfloor\ \rfloor$ は小数点以下切り捨てを,

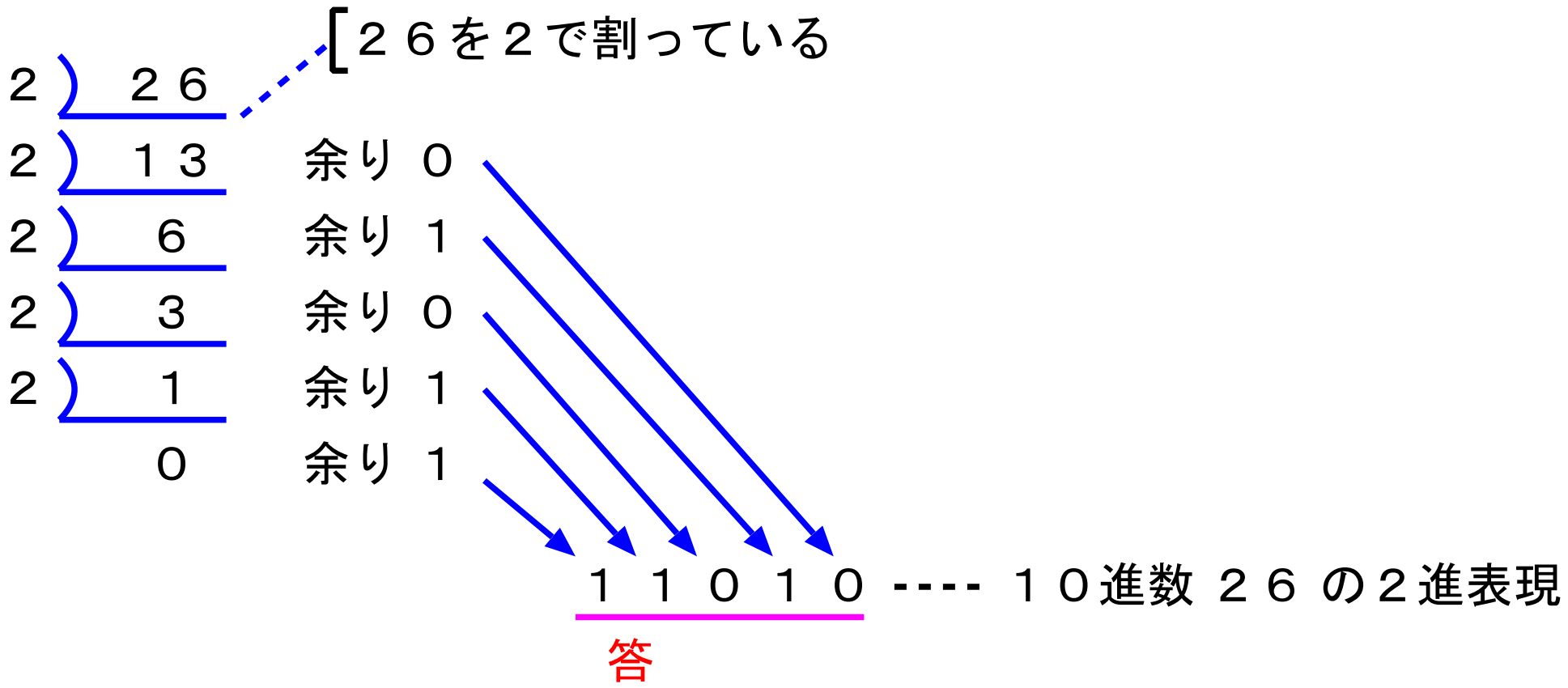
mod は第1引数を第2引数で割ったときの余りを表す

と表せる。

この事実を用いると、

(10進) 非負整数が与えられた時その2進表現は

次の様にして求めることができる。


２６を２で割っている
２ ） ２６
２ ） １３ 余り ０
２ ） ６ 余り １
２ ） ３ 余り ０
２ ） １ 余り １
０ 余り １
１ １ ０ １ ０ ---- １０進数 ２６ の２進表現
答

2進法における加減乗除:

加減乗除は10進法の場合と全く同じ様に行える。

$$\begin{array}{r} 11011 \\ +\ 101111 \\ \hline 1001010 \end{array} \qquad \begin{array}{r} 101111 \\ -\ 11011 \\ \hline 10100 \end{array}$$

$$\begin{array}{r} 11011 \\ \times\ \ 1011 \\ \hline 11011 \\ 11011 \\ 11011 \\ \hline 100101001 \end{array}$$

$$\begin{array}{r} 101 \\ \hline 101\overline{)\ 11010} \\ 101 \\ \hline 110 \\ 101 \\ \hline 1 \end{array}$$

3–4 コンピュータ内での整数データの表現

もし仮に、

10進数 26 (2進で 11010)は　5　ビットで，そして一般に
非負整数 m　は $\lfloor \log_2 m \rfloor + 1$ ビットで

表そうとすると、計算機内のどこからどこまでが1つの整数を表すかの識別が大変で取扱いが不便である。

$\lfloor\ \rfloor$ は小数部を切り捨てる関数

⇨ 「ある決められたビット数で数値を表す」という様な標準化が必要。

- 数値データを計算機内部で記憶する際の容量の標準単位としては、普通、語 (word) が用いられる。
- 語は何バイトかを集めて構成されるが、1語が何バイトに相当するかは計算機の機種によって様々。

補足：
語は単に数値データを記憶する時だけの基本単位ではなく、機械語プログラムを記憶する時の基本単位でもある。

非負整数データの内部表現:

2進法による非負整数の表現がそのまま利用される。

例えば　2進法で

　　10進数 26 は11010 と表される

から、8ビットで数値データを表す場合

　　10進数 26 は計算機内部で 00011010 と表される。

一般に　n ビットで非負整数を表す場合は

0 ~ $2^n - 1$ の間の整数が表現可能である。

整数データの内部表現:

表す整数データが非負の場合には

「非負整数データの内部表現」と同じビット列で表す。

例えば　10進数 26 は8ビットではやはり 00011010 と表される。

しかし、負数の表し方については次の3種類の方法が考えられた。

- 符号と絶対値による方法
- 2の補数による方法
- 1の補数による方法

実際には、
これら3種類の内2の補数による方法がほとんど全ての計算機で採用されている。

以下、整数データを一般に n ビットで記憶する場合を考える。

- 符号と絶対値による方法：

  最上位の　1ビット　で数値の符号を，　(0$\Longrightarrow$正，　1$\Longrightarrow$負)
  残りの　n−1 ビット　で数値の絶対値を

  表す方法で、我々人間の表現法と本質的には同じものである。

  例えば、10進数 26 は2進法で 11010 と表されるから、
  n=8の場合 計算機内部で
  整数　26　は 00011010,
  整数　−26　は 10011010
  と表される。

  ⇨ この方法では

  ◇ $-(2^{n-1}-1)$ ~ $2^{n-1}-1$ の間の整数を表現可能、

  ◇ $a_{n-1}a_{n-2}\cdots a_1a_0$ なるビット列によって
  $(-1)^{a_{n-1}}\times(a_{n-2}\times 2^{n-2}+a_{n-3}\times 2^{n-3}+\cdots+a_1\times 2^1+a_0\times 2^0)$
  すなわち
  $(-1)^{a_{n-1}}\times \sum_{i=0}^{n-2} a_i\ 2^i$
  という整数を表す。

- **2の補数による方法：** $\cdots$ (ほとんど全ての計算機で採用)

表す整数 x が $0 \sim 2^{n-1}-1$ の間の非負整数なら

x の2進表現

(の左側に必要なだけ 0 を埋めてnビットに拡張したもの) を,

そして x が $-2^{n-1} \sim -1$ の間の負整数なら

x の**2の補数** 2^n+x の2進(非負整数)表現を

内部表現とする。

例えば　n=8 の場合、-26 の2の補数 2^8-26 は2進法で

$$\begin{aligned} 2^8-26 &= (11111111+1)-00011010 \\ &= (11111111-00011010)+1 \\ &= 11100101+1 \\ &= 11100110 \end{aligned}$$

と計算できるから、計算機内部で

整数　26　は 00011010,

整数 -26 は 11100110　　$\cdots 2^8-26$ の2進(非負整数)表現

と表される。

ビット列 $a_{n-1}a_{n-2}\cdots a_1a_0$ の表す値は？

$$\begin{cases} 0\leq x\leq 2^{n-1}-1 \text{の時:} & \Longrightarrow x<2^{n-1} \quad \Longrightarrow \text{最上位ビットは } 0、\\ -2^{n-1}\leq x\leq -1 \text{の時:} & \Longrightarrow 2^n+x\geq 2^{n-1} \quad \Longrightarrow \text{最上位ビットは } 1 \end{cases}$$

それゆえ、逆にビット列 $a_{n-1}a_{n-2}\cdots a_1a_0$ が与えられた時、このビット列の表す値は、

$a_{n-1}=0$ の時

$$\begin{aligned} & a_{n-1}\times 2^{n-1}+a_{n-2}\times 2^{n-2}+\cdots+a_1\times 2^1+a_0\times 2^0 \\ =\ & a_{n-2}\times 2^{n-2}+\cdots+a_1\times 2^1+a_0\times 2^0 \end{aligned}$$

$a_{n-1}=1$ の時

$$\begin{aligned} & -2^n+(a_{n-1}\times 2^{n-1}+a_{n-2}\times 2^{n-2}+\cdots+a_1\times 2^1+a_0\times 2^0) \\ =\ & -2^n+(1\times 2^{n-1}+a_{n-2}\times 2^{n-2}+\cdots+a_1\times 2^1+a_0\times 2^0) \\ =\ & -1\times 2^{n-1}+(a_{n-2}\times 2^{n-2}+\cdots+a_1\times 2^1+a_0\times 2^0) \end{aligned}$$

すなわち

$$-a_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2} a_i\ 2^i$$

という整数を表す。

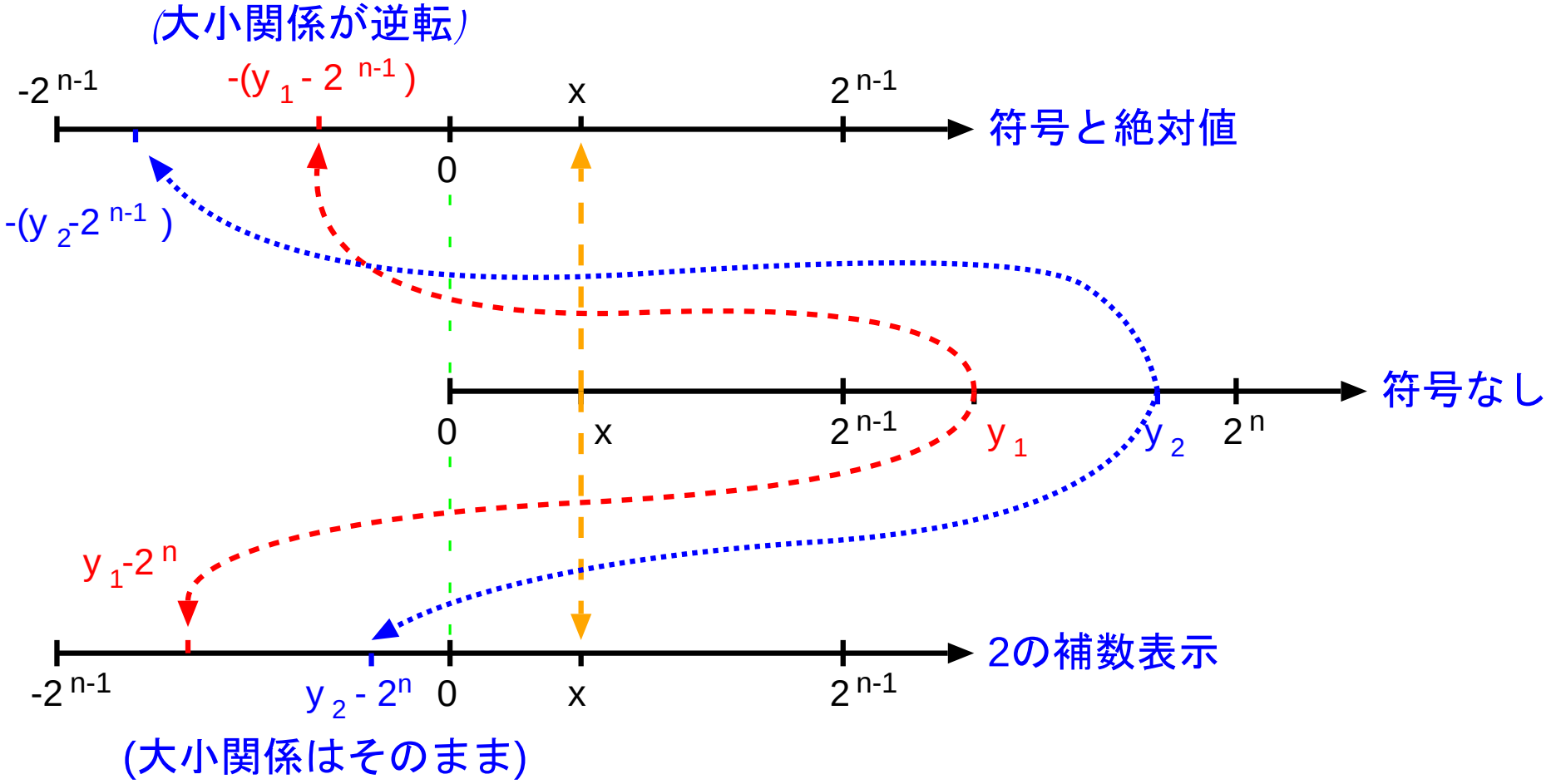
⇨ -2^{n-1}~ $2^{n-1}-1$ の間の整数を表現可能

また、

$$
\begin{aligned}
&-(a_{n-1}a_{n-2}\cdots a_1a_0\text{の表す値})\\
&=-(-a_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2}a_i\ 2^i)\\
&=a_{n-1}\times 2^{n-1}-\ \sum_{i=0}^{n-2}a_i\ 2^i\\
&=a_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2}(1-a_i)\ 2^i-\ \sum_{i=0}^{n-2}\ 2^i\\
&=a_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2}(1-a_i)\ 2^i-\ (2^{n-1}-1)\\
&=-(1-a_{n-1})\times 2^{n-1}+\sum_{i=0}^{n-2}(1-a_i)\ 2^i+1\\
&=-\bar{a}_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2}\bar{a}_i\ 2^i+1 \qquad (\text{但し、}\bar{a}_i=1-a_i)\\
&=(\bar{a}_{n-1}\bar{a}_{n-2}\cdots\bar{a}_1\bar{a}_0\text{の表す値})+1
\end{aligned}
$$

⇨ この表現法の下で数値の符号を反転するには、
単に各ビットを反転して 1 を加えるだけでよい。

1つのビット列の表す値は見方によってどう変わるか：



- **1の補数による方法：**

表す整数 x が 0~ $2^{n-1}-1$ の間の非負整数なら

x の2進表現

(の左側に必要なだけ 0 を埋めてnビットに拡張したもの) を,

そして x が $-(2^{n-1}-1)$~ -0 の間の負整数なら

x の**1の補数** 2^n-1+x の2進(非負整数)表現を

内部表現とする。

例えば　n=8 の場合、-26 の1の補数 2^8-1-26 は2進法で

$$(2^8-1)-26=11111111-00011010$$
$$=11111111-00011010$$
$$=11100101$$

と計算できるから、計算機内部で

整数　26　は 00011010,

整数 -26 は 11100101　　$\cdots 2^8-1-26$ の2進(非負整数)表現

と表される。

2の補数で負数を表す方法と同様に考えると、この方法では

◇ $-(2^{n-1}-1)$ ~ $2^{n-1}-1$ の間の整数を表現可能、

◇ $a_{n-1}a_{n-2}\cdots a_1a_0$ なるビット列によって

$$-a_{n-1}\times(2^{n-1}-1)+\sum_{i=0}^{n-2} a_i\ 2^i$$

という整数を表す。

◇ この表現法の下で数値の符号を反転するには、
単に各ビットを反転するだけでよい。

2の補数表現が採用される理由:

他に比べて演算機構が簡単になる。例えば、

- nビットの加算については

  ① 2つのビット列を符号なし2進数として加算して、
  下位 n ビットだけを取る。
  その結果、

  ② もし、得られた結果の最上位ビットが元の2つの最上位ビットのいずれとも異なるなら 桁あふれ (overflow) と判断され、
  さもなければ ①の結果が加算結果となる。

- 減算回路は、加算回路と符号反転回路を組み合わせて実現できる。

3–5 コンピュータ内での実数データの表現

実数データを記憶する標準単位としては、

- 語
- 倍長語　　…高精度の計算を行う場合のために
- ４倍長語　　…特に高精度の計算を行う場合のために

整数データの場合と異なり、様々な方法が考えられ使用されてきた。
次にその例を3つ与える。

- **IBMメインフレームでの表現法：**

  1語/32ビットのビット列

  $$\overbrace{\mathrm{s}}^{\text{符号部}}\ \underbrace{\mathrm{e_6e_5\cdots e_1e_0}}_{\text{指数部}}\ \underbrace{\mathrm{d_1d_2\cdots d_{23}d_{24}}}_{\text{仮数部}}$$

  によって、

  $$(\mathrm{-1})^{\mathrm{s}}\times \mathrm{M}\times 16^{\mathrm{E}}$$

  という実数を表す。但し、

  $$\mathrm{M}=\sum_{\mathrm{i}=1}^{24}\mathrm{d_i}\times 2^{\mathrm{-i}} \qquad \mathrm{E}=\sum_{\mathrm{i}=0}^{6}\mathrm{e_i}\times 2^{\mathrm{i}}-\ 64$$

- 日本電気メインフレーム(ACOS)での標準的表現法：

  1語/36ビットのビット列

$$\underbrace{e_7 e_6 e_5 \cdots e_1 e_0}_{\text{指数部}} \underbrace{d_0 d_1 d_2 \cdots d_{26} d_{27}}_{\text{仮数部}}$$

  によって、

$$M \times 16^E$$

  という実数を表す。但し、

$$M = -d_0 2^0 + \sum_{i=1}^{27} d_i \times 2^{-i}$$

$$E = -e_7 2^7 + \sum_{i=0}^{6} e_i \times 2^i$$

  仮数部も指数部も2の補数によって負数を表している。

- **IEEE規格754での表現法：**

  単精度、倍精度、4倍精度における指数部、仮数部のビット数等は次の様に定められている。

	符号部	指数部	仮数部	全部で
単精度	1ビット	8ビット	23ビット(10進で6~ 7桁)	32ビット
倍精度	1ビット	11ビット	52ビット(10進で15~ 16桁)	64ビット
4倍精度	1ビット	15ビット	112ビット(10進で33~ 34桁)	128ビット

特に 単精度の場合は 、32ビットの列

$$\overbrace{\mathrm{s}}^{\text{符号部}}\ \underbrace{\mathrm{e_7e_6}\cdots\mathrm{e_1e_0}}_{\text{指数部}}\ \underbrace{\mathrm{d_1d_2}\cdots\mathrm{d_{22}d_{23}}}_{\text{仮数部}}$$

によって、

$$\begin{cases} (-1)^{\mathrm{s}}\times(1+\mathrm{M})\times 2^{\mathrm{E}} & \text{if } -127<\mathrm{E}<128 \\ (-1)^{\mathrm{s}}\times\mathrm{M}\times 2^{\mathrm{E}+1} & \text{if } \mathrm{E}=-127 \\ \mathrm{Inf}(\text{無限大}) & \text{if } \mathrm{E}=128,\ \mathrm{M}=0 \\ \mathrm{NaN}(\text{非数,Not a Number}) & \text{if } \mathrm{E}=128,\ \mathrm{M}\neq 0 \end{cases}$$

という実数を表す。但し、

$$\mathrm{M}=\sum_{\mathrm{i}=1}^{23}\mathrm{d_i}\times 2^{-\mathrm{i}}$$

（仮数部から $\mathrm{d_0}=1$ というビットが省かれていると暗に仮定し、

$$1.\mathrm{d_1d_2}\cdots\mathrm{d_{22}d_{23}}$$

という2進小数を仮数部が表すと考える。）

$$\mathrm{E}=\sum_{\mathrm{i}=0}^{7}\mathrm{e_i}\times 2^{\mathrm{i}}-127$$

では、例えば 26 はIEEE規格754(単精度)でどう表されるのか？

⇨ 26.0 を $(-1)^{s}\times(1+M)\times 2^{E}$ という形に変形してみる。

$$\begin{aligned} 26.0 &= 11010.0\ (2\text{進数}) \\ &= 1.10100 \times 2^{4} \\ &= (-1)^{0}\times(1+0.10100)\times 2^{4} \end{aligned}$$

⇨ 2つを見比べて、s=0, M=0.10100, E=4

⇨ $se_7e_6\cdots e_1e_0d_1d_2\cdots d_{22}d_{23}$ で 26 が表されるとすると、

$M = \sum_{i=1}^{23} d_i\times 2^{-i} = 0.10100$ なので、

$d_1 = 1$
$d_2 = 0$
$d_3 = 1$
$d_4 = 0$
$d_5 = 0$
.....

$E = \sum_{i=0}^{7} e_i\times 2^{i} - 127 = 4$ より、

$\sum_{i=0}^{7} e_i\times 2^{i} = 131$
$= 10000011\ (2\text{進数})$

それゆえ、

$e_7 = 1$
$e_6 = e_5 = \cdots = e_2 = 0$
$e_1 = e_0 = 1$

実数計算における誤差の必然性 $(1 \div 10 \times 10 \neq 1?)$:

実数データを扱う場合には、常に

記憶された数値が表そうとした数値の近似にすぎない

ことに注意しなければならない。

例えば、IEEE規格754(単精度)の表現法の場合

10進数　1　は X'3F800000' と表され,　　$\cdots 2^0 \times 1.0$(2進)

10進数 10 は X'41200000' と表される。　$\cdots 2^3 \times 1.010$(2進)

これら2数の間で割り算 $(2^0 \times 1.0) \div (2^3 \times 1.010)$ を行うと、

$0.0\dot{0}0\dot{1}1\dot{1}$(2進小数)$=2^{-4} \times 1.\dot{1}0\dot{0}1\dot{1}$(2進小数)　循環小数, 下記補足

⇨ 10進実数の割算 $1 \div 10$ の結果は X'3DCCCCCC' と表される。

誤差発生

更に、この誤差付きの除算結果

$0.0001\ 1001\ 1001\ 1001\ 1001\ 1001\ 100$ (2進) $= 0.1999998$(16進)

に 10(10進数) $= 1010$(2進数) $=$ A(16進数) を掛けると、

$0.FFFFFF0$(16進小数)$=2^{-1} \times 1.FFFFFE$(16進小数)　下記補足

⇨ 10進の式 $1 \div 10 \times 10$ の結果は X'3F7FFFFF' と表される。

⇨ 実数表現の世界では、

10進の式 $1 \div 10 \times 10$ の計算結果は10進数 1 と等しくはならない。

補足 (1÷10)：

$$
\begin{array}{r}
0.0\dot{0}01\dot{1}0011 \\
\hline
1010\ \big)\ 1.0000 \quad \vdots \\
\underline{\ \ 1010\ \ } \quad \vdots \\
\ \ 1100 \quad \vdots \\
\underline{\ \ 1010\ \ } \quad \vdots \\
\ \ 1000
\end{array}
$$

補足(16進の掛け算 $0.1999998\times\mathrm{A}$)：

$$
\begin{array}{r}
0.1999998 \\
\times\qquad\quad \mathrm{A} \\
\hline
0.\mathrm{FFFFFF0}
\end{array}
\qquad
\begin{cases}
1\times\mathrm{A}=\ \ \mathrm{A} & (16\text{進の「九九」}) \\
9\times\mathrm{A}=5\mathrm{A} & (16\text{進の「九九」}) \\
8\times\mathrm{A}=50 & (16\text{進の「九九」})
\end{cases}
$$

3–6 まとめ

データを表すビット列自体の中には、

そのデータが 文字列であるか、
整数であるか、
実数であるか、
……

の情報は入っていない。

⇨ データを処理する側では、
データがある内部表現方式に従うもの
と見なして処理を進める。

□演習3. 2 (1999年度定期試験問題)

(0) ビット列 0110 0100 0100 1110 を16進表記で表せ。

(1) (半角)文字の並び 38 はJIS 8ビット符号体系ではどんなビット列で表されるか？　16進表記で答えよ。

(2) 10進整数の 38 は8ビットの整数データとしてどの様に表されるか？2進表記で答えよ。

(3) 10進整数の -38 は2の補数表示により8ビットの整数データとしてどの様に表されるか？　2進表記で答えよ。

(4) 10進で表された実数値 38.0 はIEEE規格754の単精度実数としてどの様に表されるか？　2進表記で答えよ。

□演習3. 3 (2000年度定期試験問題)

(1) ビット列 0110 0100 0100 1010 を16進表記で表せ。

(2) ビット列 0110 0100 0100 1010 が文字列を表しているとすると、何の文字列を表しているか？

(3) ビット列 0110 0100 0100 1010 が符号付き整数を表しているとすると、 何の整数を表しているか？

(4) ビット列 0110 0100 0100 1010 が符号付き整数を表していると見て、その正負の符号を反転するとどんなビット列になるか？　但し、ここでは、負数は2の補数で表すとする。

(5) 10進で表された実数値 3.0 はIEEE規格754の単精度実数としてどの様に表されるか？　2進表記で答えよ。

(6) プログラム内蔵方式とは何か？