TeC 教科書

Ver. 4.0.0

徳山工業高等専門学校
情報電子工学科

目次

第 1 章

はじめに

1.1 この科目で学ぶこと

現代，コンピュータと呼ばれているものはスーパーコンピュータと呼ばれる高価で高性能なものから，マイコンと呼ばれ炊飯器やエアコンに組み込まれている小型のものまで，全て同じ原理に基づき動作しています．

この原理は，1946 年に米国の数学者フォン・ノイマン (Von Neumann) が提案したと言われています．現代のコンピュータの，ほぼ，全てのものは，ノイマンの提案した同じ原理を使用しており「ノイマン型コンピュータ」と呼ばれます．

「ノイマン型コンピュータ」が出現して既に約 70 年の時間が経過しましたが，未だにこれを上回る「自動機械」の構成方法は発明されていません．だから，パソコンのような一般の人がコンピュータだと考えている装置だけでなく，炊飯器やエアコンの制御装置のようなコンピュータらしくない装置まで，「自動機械」が必要なところには全て「ノイマン型コンピュータ」が使用されているのです．恐らく，あと数十年は，「ノイマン型コンピュータ」の時代が続くでしょう．

この教科書は，教育用コンピュータ（TeC）を用いて，**ノイマン型コンピュータの動作原理**をしっかり勉強できるようになっています．ここでしっかり勉強しておけば，将来，どんな新型のコンピュータに出会ったとしても，恐れることはありません．所詮，「ノイマン型コンピュータ」の一種ですから，皆さんはその正体を簡単に見抜くことができるはずです．

「ノイマン型コンピュータ」の原理をきちんと理解しておくことは，皆さんにとって，寿命の長いエンジニアになるための大切なステップとなります．しっかり，がんばって下さい．

1.2 勉強の進め方

この教科書は，高専や工業高校の 1 年生と 2 年生が，教育用コンピュータ TeC を教材に，ノイマン型コンピュータの基本を学ぶことを想定して作ってあります．1 章から 5 章の途中までを 1 年で学び，残りを 2 年生 (半年〜1 年間) で学びます．

1 年生では，(1) コンピュータの内部で情報を表現する方法，(2)TeC の組み立て，(3)TeC の基本的なプログラミングを学びます．2 年生では，TeC を用いた高度なプログラミングを学びます．

1.3 教材用コンピュータ

私達の身近にあるパーソナルコンピュータ (パソコン) や携帯電話のようなコンピュータシステムは，高度で複雑すぎて「ノイマン型コンピュータ」の原理を学ぶための教材としては適していません．

原理を学ぶのに適した単純で小さなコンピュータ (マイコン) を，専用に開発しました．このマイコンは TeC(Tokuyama Educational Computer： 徳山高専教育用コンピュータ，図 1.1) と呼ばれます．

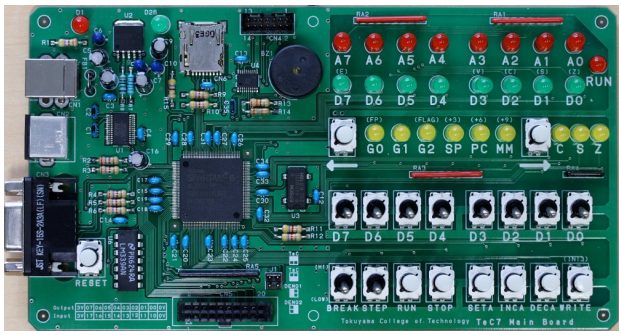


図 1.1 教育用コンピュータ TeC

TeC の特徴は単純で小さなことです．単純で小さなことで，次のようなメリットがあります．

単純　ノイマン型コンピュータの本質的な部分だけを勉強しやすい．現代のパソコン等は，勉強するには難しすぎる．

小型　自宅に持ち帰り宿題ができる．授業時間外でも，納得がいくまで色々と試してみることができる．

TeC には 2003 年から使用している TeC6 と，2011 年から使用を開始した TeC7 の二種類があります．本書では TeC7 を基本に解説しています．TeC6 について知りたい場合は本書の古いバージョン，「TeC 教科書 Ver. 3.2.2」（https://github.com/tctsigemura/TecTextBook/blob/v3.2.2/tec.pdf）を見て下さい．

第 2 章

情報の表現

この章では，コンピュータ内部でどのように情報が表現されているのか，その情報をどのような回路で扱うことができるのか，簡単に紹介します．

2.1 コンピュータ内部の情報表現

人は，情報を音声や文字，絵等で表現することができます．コンピュータも表面的には音声や文字，絵を扱うことができますが，コンピュータの内部は電子回路で構成されているので，最終的には電圧や電流で情報を表現せざるを得ません．

$$\text{情報の表現} = \left(\begin{array}{l} \text{人：音声，文字，絵，} \ldots \\ \text{コンピュータ：電圧，電流} \end{array} \right)$$

2.1.1 電気を用いた情報の表現

電圧や電流で情報表現する方法には，いろいろなアイデアがあります．しかし，現代のコンピュータは，電圧が「ある/ない」か，電流が「流れている/流れていない」のような，「ON/OFF」二つの状態だけを用いて情報を表現する方法を使っています．二つの状態だけを考えれば良いので回路が作りやすいからです．

図 2.1 を見て下さい．電気の「ON/OFF」で「おおかみが来たか/来ていないか」のどちらの状態なのかを伝達できる「情報の表示装置」を実現することができます．電気の「ON/OFF」を用いて情報を表現することができました．

2.1.2 ビット

前の例では，ランプの「ON/OFF」を用いて「二つの状態のどちらなのか」を表しました．このような，「二つのどちらか」を表す情報が「情報の最小単位」になります．情報の最小単位のことを「ビット（bit)」と呼びます．

on/off のどちらか → 情報の最小単位（ビット）

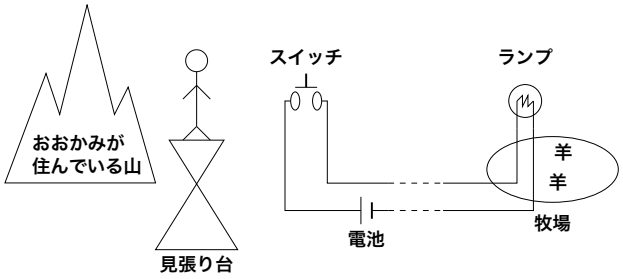


図 2.1 おおかみが来た情報表示装置

表 2.1 おおかみが来た情報

ビット値	意　味
0（off）	おおかみは来ていない
1（on）	おおかみが来た！！！

通常，ビットの値は「ON/OFF」ではなく，「1/0」で書きます．

$$\left(\begin{array}{lcl} ON & : & 1 \\ OFF & : & 0 \end{array} \right)$$

「おおかみが来た情報」は，ビットの値に表 2.1 のように対応付けできます．

2.1.3 より複雑な情報の表現

二つの状態では表現できない，もっと複雑な情報はどうやって表現したら良いでしょうか．

前の例で，やって来たおおかみの頭数により牧場の人が対応を変化させたい場合が考えられます．そのためには「来たか/来なかったか」だけの情報では不十分です．「たくさん来た」ことも知らせる必要があります．

より多くの状態を表現するために複数のビットを組み合わせます．表 2.2 に複数のビットの組み合わせて表現した「拡張おおかみが来た情報」を，図 2.2 に「拡張お

表 2.2　拡張おおかみが来た情報

ビット値	意　味
00	おおかみはきていない（平気）
01	おおかみが 1 頭来た（戦う）
10	おおかみが 2 頭来た（？）
11	おおかみがたくさん来た（逃げる）

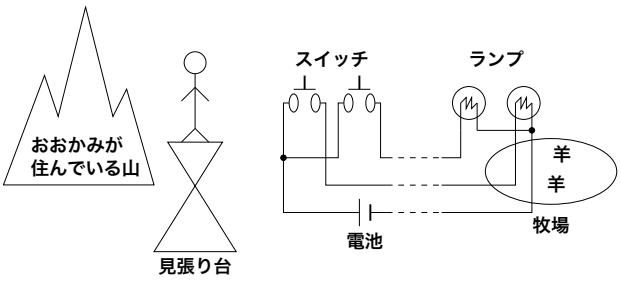


図 2.2　拡張おおかみが来た情報表示装置

表 2.3　ビットの組合せの数

ビット数	ビットの組合せ	組合せ数
1	0 1	2
2	00 01 10 11	4
3	000 001 010 011	
	100 101 110 111	8
…	…	
n		2^n

おかみが来た情報表示装置」を示します．

このように 2 ビット用いれば 4 種類の情報を表現することができます．表 2.3 はビット数と表現できる組合せの関係を示したものです．一般に，n ビットを用いると 2^n 種類の情報を表現することができます．システム内で必要なビット数を決めて，それの組合せに意味付けをすれば，どんな情報だって表現できます．

ビットだけでは情報の単位として小さすぎるので，4 ビットまとめたもの，8 ビットまとめたものにも名前があります．「4 ビット」を「1 ニブル」，「8 ビット」を「1 バイト」と呼びます．

2.2　数値の表現

ビットの組合せをどのように意味付けするかは，前節の例のように「システムが扱う必要のある情報」により，毎回，約束すればよいのです．しかし，どのコンピュータでも同じ方法で意味付けされている情報もあります．それの一つが数値の表現方法です．

2.2.1　2 進数

コンピュータの内部では，数値を 2 進数で表すのが普通です．我々が普段使用している 10 進数と，2 進数の特徴比較を次に示します．

10 進数の特徴

(1) 0 ～ 9 の 10 種類の数字を使用する．
(2) 1 桁毎に 10 倍の重みをもつ．

2 進数の特徴

(1) 0 と 1 の 2 種類の数字を使用する．
(2) 1 桁毎に 2 倍の重みをもつ．

コンピュータの内部では，ビットが情報の表現に使用されています．そこで，ビット値の 0/1 をそのまま 2 進数の 1 桁と考えれば数値が表現できます．

例えば，4 ビット用いると 0 ～ 15 の数が表 2.4 のように表現できます．一般に n ビットで 0 ～ $2^n - 1$ の範囲の数を表現することができます．

表 2.4　4 ビットの 2 進数

bit 3 (b_3)	bit 2 (b_2)	bit 1 (b_1)	bit 0 (b_0)	意味
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

2.2.2　2 進数と 10 進数の相互変換

表 2.4 に示した 4 ビットの範囲なら，2 進数と 10 進数の対応を暗記することが可能です．しかし，8 ビットの場合ならどうでしょう？

$$
\begin{array}{rcl}
10_{10} & = & 1010_2 \\
\div 2 \downarrow & & \\
5^{\cdots 0}_{10} & = & 0101^{\cdots 0}_{2} \\
\div 2 \downarrow & & \\
2^{\cdots 1}_{10} & = & 0010^{\cdots 1}_{2} \\
\div 2 \downarrow & & \\
1^{\cdots 0}_{10} & = & 0001^{\cdots 0}_{2} \\
\div 2 \downarrow & & \\
0^{\cdots 1}_{10} & = & 0000^{\cdots 1}_{2}
\end{array}
$$

図 2.3　数値を 2 で割った時の余り

組合せは 256 もあり，とても暗記できそうにありません．対応を計算で求める必要があります．

2 進から 10 進への変換

2 進数の桁ごとの重みは，桁の番号を n とすると 2^n になります．

$$
\begin{array}{cccc}
b_3 & b_2 & b_1 & b_0 \\
2^3 = 8 & 2^2 = 4 & 2^1 = 2 & 2^0 = 1
\end{array}
$$

2 進数の数値は，その桁の重みと桁の値を掛け合わせたものの合計です．例えば 2 進数の 1010_2 は，2^3 の桁が 1，2^2 の桁が 0，2^1 の桁が 1，2^0 の桁が 0 ですから，次のように計算できます．

$$
\begin{aligned}
1010_2 &= 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 \\
&= 8 \times 1 + 4 \times 0 + 2 \times 1 + 1 \times 0 \\
&= 8 + 0 + 2 + 0 \\
&= 10_{10}
\end{aligned}
$$

10 進から 2 進への変換

次に，10 進数を 2 進数に変換する方法を考えます．ここで着目するのは桁の移動です．

図 2.3 に例を示すように，2 進数を 2 で割ると右に 1 桁移動します．その時の余りは最下位の桁からはみ出した数になります．同じ値の 10 進数を 2 で割っても余りは同じです．

つまり，数値を 2 で割った時の余りは 2 進数を右に 1 桁移動したときはみ出してきた数を表しています．そこで図 2.4 のように，2 で割る操作を繰り返しながらはみ出して来た数を記録すれば，もとの数を 2 進数で表したときの 0/1 の並びが分かります．

2.2.3　16 進数

2 進数は，桁数が多くなり書き表すのに不便です．そこで，表 2.5 のように 2 進数 4 桁をまとめて 16 進数一桁で書き表します．9 より大きな数字が無いので，16 進数ではアルファベットを数字の代用にします．

何進数で書いてあるのかをはっきりさせるために，表 2.5 のように数値の右に小さな字で何進数かを書き加えます．ときには，数字の代わりに「2 進数='b'」，「16 進数='H'」を加えることもあります．例えば，「01100100_2 を 01100100_b」，「64_{16} を 64H」のように書きます．

$$
\begin{array}{rl}
2) & 10 \\
2) & 5 \cdots 0 \\
2) & 2 \cdots 1 \\
2) & 1 \cdots 0 \\
 & 0 \cdots 1
\end{array}
$$

余りを右から順に並べると 1010_2

図 2.4　10 を 2 進数に変換する例

表 2.5　2 進数，16 進数，10 進数対応表

2 進数	16 進数	10 進数
0000_2	0_{16}	0_{10}
0001_2	1_{16}	1_{10}
0010_2	2_{16}	2_{10}
0011_2	3_{16}	3_{10}
0100_2	4_{16}	4_{10}
0101_2	5_{16}	5_{10}
0110_2	6_{16}	6_{10}
0111_2	7_{16}	7_{10}
1000_2	8_{16}	8_{10}
1001_2	9_{16}	9_{10}
1010_2	A_{16}	10_{10}
1011_2	B_{16}	11_{10}
1100_2	C_{16}	12_{10}
1101_2	D_{16}	13_{10}
1110_2	E_{16}	14_{10}
1111_2	F_{16}	15_{10}

問題

1. 表 2.5「2 進数，16 進数，10 進数対応表」を暗記しなさい．
2. 10 進数の 16，50，100，127，130 を，2 進数（8 桁），16 進数（2 桁）で書き表しなさい．
3. 2 進数の 00011100，00111000，11100000 を 16 進数（2 桁），10 進数で書き表しなさい．
4. 16 進数の 1F と AA を 2 進数（8 桁）で書き表しなさい．また，10 進数で書き表しなさい．

表 2.6　符号付き絶対値表現（4 ビット）

負数	2 進数	正数	2 進数
-7	1111_2	$+7$	0111_2
-6	1110_2	$+6$	0110_2
-5	1101_2	$+5$	0101_2
...	...	...	...
-1	1001_2	$+1$	0001_2
-0	1000_2	$+0$	0000_2

2.3　負数の表現

拡張おおかみが来た情報表示装置では，「表示装置の二つのランプ（２ビット）の読み方」を約束しました．つぎに，数値の場合は，「n 個のビットを 2 進数として読む」ことを約束しました．

今度は，負の数が必要になりました．そこで，ビットの新しい読み方を約束します．この節で出てくるビットは「符号付き数値」を表しています．以下では，「符号付き数値を表すビット」をどのように読むかを説明します．

2.3.1　符号付き絶対値表現

使用できるビットのうち一つを符号を表すために使用します．これを「符号ビット」と呼ぶことにします．通常，符号ビットには最上位（左端）のビットを使用します．

表 2.6 のように 4 ビット使用して，-7 〜 $+7$ の範囲を表すことができます．この表現方法は分かりやすくて都合が良いのですが，実際に使われることはあまりありません．

2.3.2　補数表現

n 桁の b 進数において b^n から x を引いた数 y を x に対する「b の補数」と言います．n 桁の b 進数でにおいて b^n-1 から x を引いた数 z を x に対する「(b-1) の補数」と言います．

$$y = b^n - x \qquad (y \text{ は } x \text{ に対する } b \text{ の補数})$$
$$z = b^n - 1 - x \qquad (z \text{ は } x \text{ に対する } (b-1) \text{ の補数})$$

10 進数の例を図 2.5 に，2 進数の例を図 2.6 に示します．

2.3.3　1 の補数による負数の表現

「図 2.6　4 桁の 2 進数で補数の例」にあるように，$11...1_2$ からもとの数 (x) を引いた数を x に対する「1 の補数」と呼びます．表 2.7 は 0 〜 7 に対する 1 の補数を計算したものです．

$b = 10$ 進数
$n = 2$ 桁　　100
$b^n = 100$　　-25　　75 は 25 に対する 10 の補数
$x = 25$　　75

$b = 10$ 進数
$n = 2$ 桁　　99
$b^n - 1 = 99$　　-25　　74 は 25 に対する 9 の補数
$x = 25$　　74

図 2.5　2 桁の 10 進数で補数の例

$b = 2$ 進数　　0110_2 は
$n = 4$ 桁　　10000_2　　1010_2 に
$b^n = 10000_2$　　-1010_2　　対する
$x = 1010_2$　　0110_2　　2 の補数

$b = 2$ 進数　　0101_2 は
$n = 4$ 桁　　1111_2　　1010_2 に
$b^n - 1 = 1111_2$　　-1010_2　　対する
$x = 1010_2$　　0101_2　　1 の補数

図 2.6　4 桁の 2 進数で補数の例

表 2.7　4 ビット 2 進数の 1 補数

もとの数	補数へ変換		補数
0	$1111_2 - 0000_2$	$=$	1111_2
1	$1111_2 - 0001_2$	$=$	1110_2
2	$1111_2 - 0010_2$	$=$	1101_2
3	$1111_2 - 0011_2$	$=$	1100_2
4	$1111_2 - 0100_2$	$=$	1011_2
5	$1111_2 - 0101_2$	$=$	1010_2
6	$1111_2 - 0110_2$	$=$	1001_2
7	$1111_2 - 0111_2$	$=$	1000_2

「0001_2 に対する 1 の補数を -0001_2 を表現するために使用する．」，「0010_2 に対する 1 の補数を -0010_2 を表現するために使用する．」つまり，「1 の補数を負の数を表現するために使用する．」と約束すれば，図 2.7 のように -7 〜 $+7$ の範囲の数を表現できます．一般に，1 の補数表現を用いた n ビット符号付き 2 進数が表現できる数値の範囲は次の式で計算できます．

$$-(2^{n-1}-1) \ \sim \ 2^{n-1}-1$$

図 2.7 で $+7$ と -7 が結んであるのは，「互いに 1 の補数」であることを表すためです．また，「互いに 1 の補数」の 2 数を見比べるとビットの 0/1 が入れ替わった関

−7	1000_2	−	−	−	−	−	−	−	+
−6	1001_2	−	−	−	−	−	−	+	|
−5	1010_2	−	−	−	−	−	+	|	|
−4	1011_2	−	−	−	−	+	|	|	|
−3	1100_2	−	−	−	+	|	|	|	|
−2	1101_2	−	−	+	|	|	|	|	|
−1	1110_2	−	+	|	|	|	|	|	|
−0	1111_2	+	|	|	|	|	|	|	|
+0	0000_2	+	|	|	|	|	|	|	|
+1	0001_2	−	+	|	|	|	|	|	|
+2	0010_2	−	−	+	|	|	|	|	|
+3	0011_2	−	−	−	+	|	|	|	|
+4	0100_2	−	−	−	−	+	|	|	|
+5	0101_2	−	−	−	−	−	+	|	|
+6	0110_2	−	−	−	−	−	−	+	|
+7	0111_2	−	−	−	−	−	−	−	+

図 2.7　1 の補数を用いた符号付き数値

係になっていることが分かります．1 の補数は引算だけではなく「ビット反転」でも計算できます．

次に 1 の補数からもとの数を求める計算を考えてみましょう．以下のように 1 の補数 (z) を求める式を $x =$ の形に変形しても同じ形になるので，x と z は「互いに 1 の補数」です．1 の補数からもとの数を求める計算は，もとの数から 1 の補数を求める計算と同じです．（ビット反転のビット反転で元に戻る．）

$$z = 2^n - 1 - x \qquad (z \text{ は } x \text{ に対する 1 の補数})$$
$$x = 2^n - 1 - z \qquad (x \text{ は } z \text{ に対する 1 の補数})$$

1 の補数を使用した方法も，実際に使われることはあまりありません．コンピュータの内部で実際に使用されるのは，次に説明する 2 の補数による表現です．

2.3.4　2 の補数による負数の表現

「図 2.6　4 桁の 2 進数で補数の例」にあるように，$100...0_2$ からもとの数 (x) を引いた数を x に対する「2 の補数」と呼びます．表 2.8 は 0 〜 8 に対する 2 の補数を計算したものです．5 ビットで表現されている部分もありますが，四角で囲んだ 4 ビットに注目してください．

「0001_2 に対する 2 の補数を -0001_2 を表現するために使用する．」，「0010_2 に対する 2 の補数を -0010_2 を表現するために使用する．」つまり「2 の補数を負の数を表すために使用する．」と約束すれば，図 2.8 のように -8 〜 $+7$ の範囲の数を表現できます．（ただし 1000_2 は -8 を表現することとします．4 ビットでは $+8$ を表現することはできません．）

表 2.8　4 ビット 2 進数の 2 補数

もとの数	補数へ変換				補数
0	$1\boxed{0000}_2$	−	$\boxed{0000}_2$	=	$1\boxed{0000}_2$
1	$1\boxed{0000}_2$	−	$\boxed{0001}_2$	=	$\boxed{1111}_2$
2	$1\boxed{0000}_2$	−	$\boxed{0010}_2$	=	$\boxed{1110}_2$
3	$1\boxed{0000}_2$	−	$\boxed{0011}_2$	=	$\boxed{1101}_2$
4	$1\boxed{0000}_2$	−	$\boxed{0100}_2$	=	$\boxed{1100}_2$
5	$1\boxed{0000}_2$	−	$\boxed{0101}_2$	=	$\boxed{1011}_2$
6	$1\boxed{0000}_2$	−	$\boxed{0110}_2$	=	$\boxed{1010}_2$
7	$1\boxed{0000}_2$	−	$\boxed{0111}_2$	=	$\boxed{1001}_2$
8	$1\boxed{0000}_2$	−	$\boxed{1000}_2$	=	$\boxed{1000}_2$

−8	1000_2								
−7	1001_2	−	−	−	−	−	−	−	+
−6	1010_2	−	−	−	−	−	−	+	|
−5	1011_2	−	−	−	−	−	+	|	|
−4	1100_2	−	−	−	−	+	|	|	|
−3	1101_2	−	−	−	+	|	|	|	|
−2	1110_2	−	−	+	|	|	|	|	|
−1	1111_2	−	+	|	|	|	|	|	|
0	0000_2	+	|	|	|	|	|	|	|
1	0001_2	−	+	|	|	|	|	|	|
2	0010_2	−	−	+	|	|	|	|	|
3	0011_2	−	−	−	+	|	|	|	|
4	0100_2	−	−	−	−	+	|	|	|
5	0101_2	−	−	−	−	−	+	|	|
6	0110_2	−	−	−	−	−	−	+	|
7	0111_2	−	−	−	−	−	−	−	+

図 2.8　2 の補数を用いた符号付き数値

一般に，2 の補数表現を用いた n ビット符号付き 2 進数が表現できる数値の範囲は次の式で計算できます．

$$-2^{n-1} \ \sim \ 2^{n-1} - 1$$

次に 2 の補数からもとの数を求める計算を考えてみましょう．以下のように 2 の補数 (z) を求める式を $x =$ の形に変形しても同じ形になるので，x と z は「互いに 2 の補数」です．2 の補数からもとの数を求める計算は，もとの数から 2 の補数を求める計算と同じです．

$$z = 2^n - x \qquad (z \text{ は } x \text{ に対する 2 の補数})$$
$$x = 2^n - z \qquad (x \text{ は } z \text{ に対する 2 の補数})$$

現代のコンピュータは，ほとんどの機種で 2 の補数表現を採用しています．2 の補数表現を用いると 2 進数の足し算・引き算が，正負のどちらの数でも同じ手順で計

10 進数		2 進数	
07		0111_2	(10 進の 7)
+ 05		+ 0101_2	(10 進の 5)
12		1100_2	(10 進の 12)
12		1100_2	(10 進の 12)
− 05		− 0101_2	(10 進の 5)
07		0111_2	(10 進の 7)

何進数で計算しても同じ結果になる．

図 2.9　2 進数と 10 進数の計算を比較

算できます（詳しくは後述）．「手順が同じ＝演算回路が同じ」ことになりますので，コンピュータを製作する上では非常に都合がよいのです．

2.3.5　2 の補数を求める手順

$-x$ を n ビット 2 の補数表現 (y) で表します．2 の補数の定義より，y は次のように計算できます．

$$\begin{aligned} y &= 2^n - x \\ &= (2^n - 1 - x) + 1 \\ &= (x \text{ に対する } 1 \text{ の補数}) + 1 \end{aligned}$$

「x に対する 1 の補数」はビット反転で簡単に求めることができます．「x に対する 2 の補数」は「x をビット反転した後，1 を加える」ことにより簡単に求めることができます．

ビット反転した後，1 を加える．

2.3.6　2 の補数から元の数を求める手順

次に，逆変換について考えます．既に 2.3.4 で触れたように 2 の補数と元の数は互いに「2 の補数」ですから，2 の補数を求めるのと同じ計算で，2 の補数から元の数を求めることができます．

ビット反転した後，1 を加える．

2.4　2 進数の計算

ここでは，2 進数の和と差の計算方法を学びます．

2.4.1　正の数の計算

2 進数の計算も 10 進数と同じ要領です．桁上がりが 10 ではなく，2 で発生することに注意してください．図 2.9 に例を示します．2 進数で計算しても 10 進数で計算しても同じ計算結果になります．

$3 + (-5) = -2$ （4 ビット 2 の補数表現）

3 を 2 進数に変換すると 0011_2
-5 を 2 進数に変換すると 1011_2

「和を計算する」
$0011_2 + 1011_2 = 1110_2 = -2_{10}$

図 2.10　結果が負になる足し算の例

$5 + (-3) = 2$ (4 ビット 2 の補数表現)

5 を 2 進数に変換すると 0101_2
-3 を 2 進数に変換すると 1101_2

「和を計算する」
$0101_2 + 1101_2 = \boxed{1}0010_2 = 2_{10}$
(5 ビット目の 1 は桁あふれで消滅)

図 2.11　結果が正になる足し算の例

2.4.2　負の数を含む計算

2 の補数表現を用いると，正の数だけのときと同じ要領で負の数を含む計算ができます．ここでは和の計算を例に説明します．

正の数と負の数の和

正の数 (X) と負の数 $(-A)(A$ の 2 の補数 $(B))$ の和の計算を考えます．

$X + B$ の計算 $(X + (-A))$

1. 結果が負の場合 $(|A| > |X|)$
   解は $-(A - X)$ になるはず！

$$\begin{aligned} X + B &= X + (2^n - A) \\ &= 2^n - (A - X) \end{aligned} \tag{2.1}$$

(2.1) 式は，正解 $-(A - X)$ の 2 の補数表現になっています．図 2.10 に計算例を示します．

2. 結果が正またはゼロの場合 $(|X| \geq |A|)$
   解は $(X - A)$ になるはず！

$$\begin{aligned} X + B &= X + (2^n - A) \\ &= 2^n + (X - A) \\ &= X - A \end{aligned} \tag{2.2}$$

(2.2) 式の 2^n は，桁あふれにより結果に残らないので，正解 $(X - A)$ となります．図 2.11 に計算例を示します．

$(-1)+(-3)=(-4)$（4 ビット 2 の補数表現）

-1 を 2 進数に変換すると 1111_2
-3 を 2 進数に変換すると 1101_2

「和を計算する」
$1111_2 + 1101_2 = \boxed{1}1100_2 = -4_{10}$
（5 ビット目の 1 は桁あふれで消滅）

図 2.12　負の数と負の数の足し算の例

負の数と負の数の和

負の数 $(-A_1)$(2 の補数 (B_1)) と
負の数 $(-A_2)$(2 の補数 (B_2)) の和
B_1+B_2 の計算 $((-A_1)+(-A_2))$
解は $-(A_1+A_2)$ になるはず！

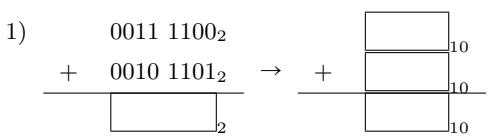
$$\begin{aligned} B_1+B_2 &= (2^n-A_1)+(2^n-A_2) \\ &= 2^n+(2^n-(A_1+A_2)) \qquad (2.3) \\ &= 2^n-(A_1+A_2) \qquad (2.4) \end{aligned}$$

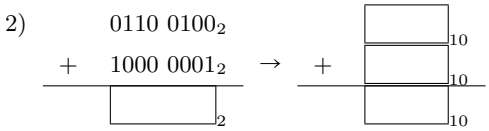
(2.3) 式の最初の 2^n は，桁あふれにより消滅するので (2.4) 式は正解 $(-(A_1+A_2))$ の 2 の補数表現になっています．図 2.12 に計算例を示します．

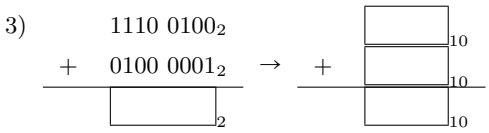
以上のように，2 の補数表現を使用すると負の数を含んだ足し算が簡単に計算できます．ここでは省略しますが，引き算も同様に正負の区別をすることなく計算できます．

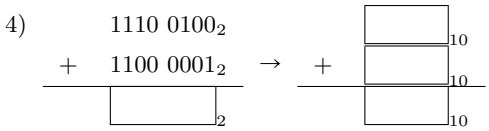
問題

1. 次の数を「2 の補数表現を用いた 4 ビット符号付き 2 進数」で書き表しなさい．
   3_{10}，-3_{10}，5_{10}，-5_{10}，6_{10}，-6_{10}
2. 次の数を「2 の補数表現を用いた 8 ビット符号付き 2 進数」で書き表しなさい．
   3_{10}，-3_{10}，8_{10}，-8_{10}，
   30_{10}，-30_{10}，100_{10}，-100_{10}
3. 次の「2 の補数表現を用いた 4 ビット符号付き 2 進数」を 10 進数で書き表しなさい．
   1100_2，1011_2，0100_2，1101_2
4. 次の 2 進数は「2 の補数表現を用いた符号付き 2 進数」です．空欄を埋めなさい．









2.5　小数の表現

この節では，小数を 2 進数で表現する方法を紹介します．

2.5.1　2 進数による小数の表現

コンピュータ内部で，2 進数を用いて，小数を含む数を表現する方法は何種類かあります．ここでは，その中でも最も基本的な固定小数点形式を紹介します．

これまでの 2 進数は，暗黙のうちに小数点が右端にあると考えました．小数点の位置を右端以外だと約束すれば，小数を含む数の表現も可能になります．以下に 4 ビットの 2 桁目に小数点があると考えた場合の例を示します．

$00.00_2 = 0.0_{10}$
$00.01_2 = 0.25_{10}$
$00.10_2 = 0.5_{10}$
$00.11_2 = 0.75_{10}$
$01.00_2 = 1.0_{10}$
$01.01_2 = 1.25_{10}$
$01.10_2 = 1.5_{10}$
$01.11_2 = 1.75_{10}$
$10.00_2 = 2.0_{10}$
...
$11.11_2 = 3.75_{10}$

小数点を中心に，左は 1 桁毎に 2 倍，右は 1 桁毎に 1/2 倍になります．（10 進数では，左は 1 桁毎に 10 倍，右は 1 桁毎に 1/10 倍になりましたね．）

2.5.2　10 進数との相互変換

1. 2 進数から 10 進数への変換
   整数の場合と同様に桁の重みを加算すれば 10 進数に変換できます．次に例を示します．
   $10.01_2 = 2 + 1/4 = 2.25_{10}$
2. 10 進数から 2 進数への変換
   整数の場合は，1/2 倍することで右にシフト（桁移

2 進数	×2 は	10 進数
0.101_2	左シフ	0.625
↙	トと同じ	× 2
1.010_2		1.250

2 進数	×2 は	10 進数
0.010_2	左シフ	0.250
↙	トと同じ	× 2
0.100_2		0.500

2 進数	×2 は	10 進数
0.100_2	左シフ	0.500
↙	トと同じ	× 2
1.000_2		1.000

10 進数で計算したとき，小数点を横切って整数部に出てきた数を小数点の右に順番に並べると $0.\underline{101}_2$ になる．

図 2.13　小数を表す 10 進数から 2 進数への変換例

動）し，小数点を横切った 0/1 を判定しました．（小数点を 1 が横切ると，余りが出ていました．）

図 2.13 に例を示すように，小数点以下の数値は，2 倍することで左にシフトし，小数点を横切った 0/1 を判定すれば 2 進数に変換できます．

3. 整数部と小数部の両方がある場合

整数部分と小数部分を分離して，別々に計算します．

問題

1. 次の 2 進数を 10 進数に変換しなさい．
   (a) 0.1001_2
   (b) 0.0101_2
   (c) 11.11_2
2. 次の 10 進数を 2 進数に変換しなさい．
   (a) 0.0625_{10}
   (b) 0.1875_{10}
   (c) 0.4375_{10}
3. 次の 10 進数を 2 進数に変換しなさい．（難問）
   (a) 0.8_{10}
   (b) 0.7_{10}

2.6　文字の表現

この節では，文字を 2 進数で表現する方法を紹介します．（文字情報を表しているビットの読み方を約束する．）

> **MSB と LSB**
>
> 「2 の補数表現を用いると，最上位ビットが 1 なら負の数と分かります．」のように「最上位ビット」と言う言葉をよく使います．
>
> 「最上位ビット」と言うのは長いので，これを英語にした「Most Significant Bit」の頭文字を取り MSB と略すことが良くあります．憶えておいて下さい．
>
> 同様に「最下位ビット」のことは英語で「Least Significant Bit」なので，略して LSB と呼びます．これも，憶えておきましょう．

2.6.1　文字コード

文字の場合，数値のような規則性を期待することはできません．そこで，使用する文字の一覧表を作成し，1 文字毎に対応する 2 進数（コード）を決めます．この一覧表のことを「文字コード表」と呼びます．文字コード表を各自（コンピュータメーカ等）が勝手に定義すると，コンピュータの間でのデータ交換に不便です．そこで，規格として制定してあります．

2.6.2　ASCII コード

ASCII（American Standard Code for Information Interchange）コードは，1963 年にアメリカ規格協会（ANSI）が定めた情報交換用の文字コードです．英字，数字，記号等が含まれます．現代のパーソナルコンピュータ等で使用される文字コードは，ASCII コードが基本になっています．図 2.14 に ASCII 文字コード表を示します．

文字コード表で 00H 〜 1FH と 7FH は，機能（改行等）を表す特殊な文字になっています．20H（SP）は空白を表す文字です．

ASCII 文字コードは 7 ビットで表現されます．しかし，コンピュータの内部では 1 バイト（8 ビット）単位の方が扱いやすいので，最上位に 0_2 を補って 8 ビットで扱うことがほとんどです．

2.6.3　JIS 文字コード

日本では JIS（Japan Industrial Standard：日本工業規格）の一部として，8 ビットコード（英数記号＋カナ）と，16 ビットコード（英数記号＋カナ＋ひらがな＋カタカナ＋漢字...）が定められています．

（上位 3 ビット）

（下位 4 ビット）	0	1	2	3	4	5	6	7
0	NUL	DLE	(SP)	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

図 2.14　ASCII 文字コード表

JIS 8 ビットコードは，ASCII コードに半角カタカナを追加したものです．記号，数字，英字の部分は，ほぼ，同じ並びになっています．

2.7　補助単位

長さや重さを書くとき，1,000m を 1km，1,000g を 1kg のように書きました．この k のような記号を補助単位と呼びます．

コンピュータの世界でよく使用される補助単位には，k（キロ=10^3），M（メガ=10^6），G（ギガ=10^9），T（テラ=10^{12}）等があります．（1,000 倍毎に補助単位があります．）パソコンのカタログに「CPU のクロックは 1GHz」のような記述を見かけますね．よく使う補助単位を表 2.9 にまとめます．

記憶容量を表す場合の補助単位は，$1,000$ の代わりに $2^{10} = 1,024$ を使用します．1,000 も 1,024 も k で表現すると紛らわしいので，k の代わりに Ki と書き表すこともあります．つまり，$1kB = 1KiB = 2^{10}B = 1,024B$，$1MB = 1MiB = 2^{20}B = 1,048,576B$ となります．（「B」はバイトを表す．）

「H.D.D. の容量は 500GB」のように表示されている場合は，記憶容量を表しているので $G = 2^{30}$ で表示されています．同じことを「H.D.D. の容量は 500GiB」のように表示することもあります．

表 2.9　補助単位まとめ

一般的に			記憶容量		
値	記号	読み方	値	記号	読み方
10^3	k	キロ	2^{10}	Ki	キビ
10^6	M	メガ	2^{20}	Mi	メビ
10^9	G	ギガ	2^{30}	Gi	ギビ
10^{12}	T	テラ	2^{40}	Ti	テビ

2.8　コンピュータの基本回路

前の節までで，コンピュータ内部での情報の表現方法を勉強しました．コンピュータの内部では，どんな情報も電気の「ON/OFF」で表現しているのでしたね．

この節では，電気の「ON/OFF」を使用して，計算や記憶をする回路を勉強します．

2.8.1　論理演算と論理回路

論理（Yes/No，真/偽，True/False）を対象とする演算（計算）のことを「論理演算」と呼びます．論理の「真/偽」をビットの「1/0」に対応させることにより，論理も電気の「ON/OFF」で表現することができます．

ここでは，論理演算と，論理演算をする回路を紹介します．論理演算をする回路のことを「論理回路」と呼びます．ここで紹介する論理回路が，コンピュータを構成する基本回路になります．

2.8.2　基本的な論理回路

基本的な論理回路を 4 種類，組合せてできたものを 2 種類，紹介します．

論理積（AND）

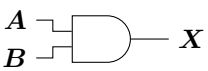
二つのビットを入力し，両方が 1 のときだけ出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

AND **の真理値表**

$$X = A \cdot B$$

AND **の論理式**



AND **の回路記号**

論理和（OR）

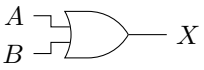
二つのビットを入力し，どちらかが 1 のとき出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

OR の真理値表

$$X = A + B$$

OR の論理式



OR の回路記号

否定（NOT）

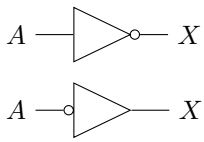
一つのビットを入力し，入力と逆の論理を出力する論理演算です．

入力	出力
A	X
0	1
1	0

NOT の真理値表

$$X = \overline{A}$$

NOT の論理式



NOT の回路記号

排他的論理和（XOR）

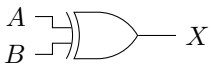
二つのビットを入力し，二つが異なるとき出力が 1 になるような論理演算です．

入力		出力
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

XOR の真理値表

$$X = A \oplus B$$

XOR の論理式



XOR の回路記号

NAND

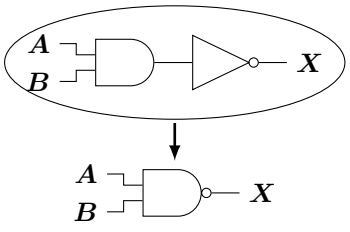
否定（NOT）と論理積（AND）を組み合わせた演算です．

入力		出力
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

NAND の真理値表

$$X = \overline{A \cdot B}$$

NAND の論理式



NAND の回路記号

NOR

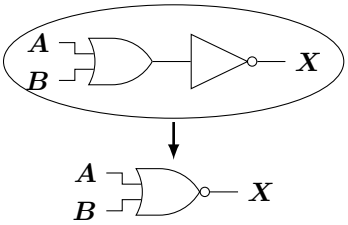
否定（NOT）と論理和（OR）を組み合わせた演算です．

$$X = \overline{A + B}$$

NOR の論理式

入力		出力
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

NOR の真理値表



NOR の回路記号

2.8.3　演算回路

「基本的な論理回路」を組み合わせることにより，足し算／引き算等のより複雑な演算を行う回路を実現できます．

半加算器

例えば，1 ビットの足し算回路は図 2.15 ように実現できます．この例では A，B の 2 ビットを入力し，和 (S) と上の桁への桁上がり（C）を出力する回路を示しています．この回路を「半加算器」と呼びます．

全加算器

半加算器は桁上がりを出力しますが，自分自身は下の桁からの桁上がりを入力することができません．桁上がりの入力を備えた 1 ビット足し算器を「全加算器」と呼び，図 2.16 のように実現することができます．

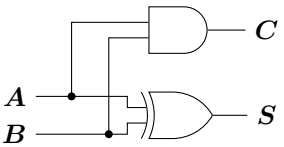
n ビット加算器

図 2.17 のように半加算器と全加算器を組み合わせることで，任意ビットの加算器を実現することができます．

$$\begin{array}{rcccc} A & 0 & 0 & 1 & 1 \\ +\ B & +\ 0 & +\ 1 & +\ 0 & +\ 1 \\ \hline C\ S & 0\ 0 & 0\ 1 & 0\ 1 & 1\ 0 \end{array}$$

入力		出力	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

半加算器の真理値表



半加算器の回路図

図 2.15　1 ビット半加算器

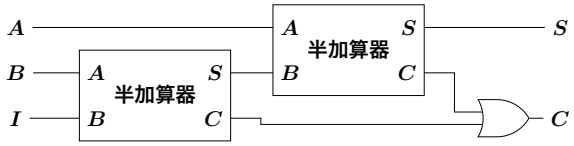
入力			出力	
A	B	I	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

全加算器の真理値表

A：計算の入力

B：計算の入力

I：桁上がりの入力



全加算器の回路図

図 2.16　1 ビット全加算器

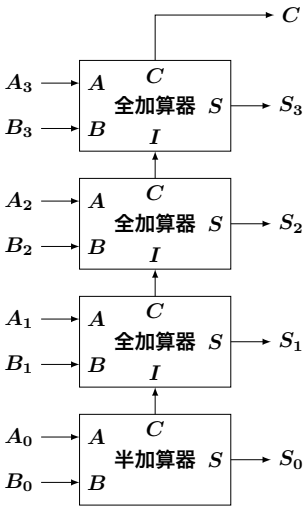


図 2.17　4 ビット加算器

n ビット 1 の補数器

1 の補数を作る回路です．1 の補数はビット反転によってできるので，NOT でできます．図 2.18 に 4 ビットの例を示します．簡単ですね．

n ビット 2 の補数器

2 の補数を作る回路です．2 の補数は，1 の補数に 1 足すとできるので，1 の補数器の出力に半加算器を組み合わせるとできます．図 2.19 に 4 ビットの例を示します．

演算回路を数種類紹介しました．ここでは紹介しませんでしたが，引き算回路等も同様に製作可能です．

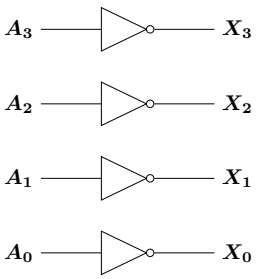


図 2.18　4 ビット 1 の補数器

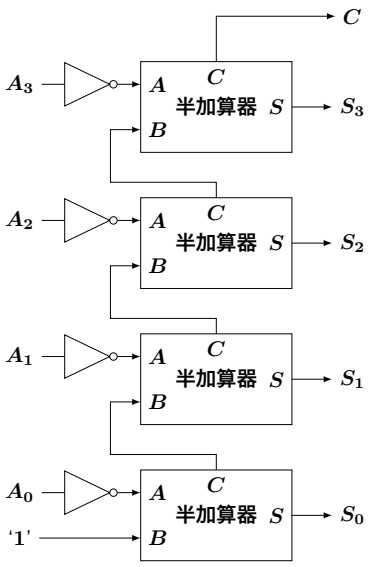


図 2.19　4 ビット 2 の補数器

2.8.4　記憶回路

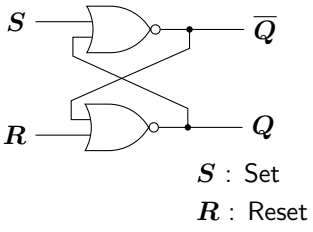
「基本的な論理回路」を組み合わせることにより，記憶機能を実現することもできます．ここでは，最も簡単な RS-FF（RS フリップフロップ）を紹介します．

図 2.20 に RS フリップフロップの真理値表と回路図を示します．RS-FF は S（Set）と R（Reset）の二つの入力と，状態 $(Q, \bar{Q})$ の出力を持つ回路です．S=0, R=1 を入力することにより，回路はリセットされ出力 Q は 0 になります．S=1，R=0 を入力することにより，回路はセットされ出力 Q は 1 になります．$\bar{Q}$ には，常に Q の否定が出力されます．

S=0，R=0 を入力すると，回路は直前の値を「記憶」します．S=1，R=1 は入力してはいけない組合せです．RS-FF は，回路をリセット／セットした後で「記憶」状態にすることにより，「値を記憶する回路（＝記憶回路）」として働きます．

入力		出力
S	R	Q
0	0	記憶
0	1	0
1	0	1
1	1	禁止

RS-FF の動作



RS-FF の回路

図 2.20　RS フリップフロップ

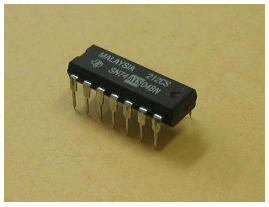
論理回路素子

回路を製作するには，基本的な論理回路を内蔵した集積回路（論理 IC）を用いることができます．

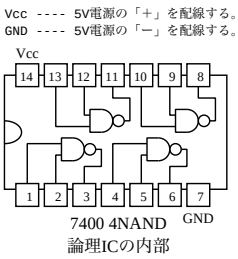
様々な論理 IC が販売されていますが，ここでは，74 シリーズのものを紹介します．下の例に示したのは，NAND ゲートを四つ内蔵した 7400 と呼ばれる IC です．同様な IC で，AND，OR，XOR，NOT 等の回路を内蔵したものがあります．

このような IC 同士を配線して組み合わせることにより，本文で紹介した演算回路や記憶回路を実現することができます．

論理 IC



外観



論理ICの内部

2.9　まとめ

この章では，まず，ノイマン型コンピュータの内部では，情報が電気（電子）回路で扱い扱いやすい 2 進数で表現されていること学びました．次に，数値が 2 進数で表現できることや計算方法，文字データの表現法を学びました．最後に，コンピュータの基本回路である論理回路について学びました．

TeC 教科書 Ver. 4.0.0

発行年月　２０１７年　６月　Ver.4.0.0
発　　行　独立行政法人国立高等専門学校機構
徳山工業高等専門学校
情報電子工学科　重村哲至
〒745-8585 山口県周南市学園台
sigemura@tokuyama.ac.jp