電子計算機 **2E**

2E　2003.7.9

本日の授業のテーマ

本日の事業のテーマは、以下のとおりです。

(1) 数字のコード
- 10 進コード
- グレイコード

(2) 偶奇(パリティ)

本日の授業のゴールは、以下のとおり。

- 数値コードと 10 進数を 2 進数に変換することとは異なること理解できる。
- グレイコードの内容が理解できる。
- パリティの役割が理解できる。

1 数値コードとは

1.1 数値コード

前回までは、文字のコード化について述べた。ここでは、10 進数のコード化について学習します。これは、中間試験前に学習した 10 進数を 2 進数に変換することとは異なります。ここで学習する数字のコード化とは、10 進数で使われる 0〜9 の文字を 1 と 0 の記号で表すことです。ということは、数多くのコード化が考えられる。役に立つかは別にして、諸君自身が新しいコードを作成することもできます。要するに、こんなコードも作れます。

数字	コード
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0001000000
7	0010000000
8	0100000000
9	1000000000

いろいろな、コードが作れるでしょう。問題は、そのコードが役に立つか否かです。ここでは、いろいろな 10 進数の数字のコードについて説明します。

ここでひとつ疑問があると思います。なぜ 10 進数なのでしょうか?。コード化と言う意味では、2 進数でも 3 進数でも、16 進数でも 20 進数でも一緒です。わざわざ、10 進数のコード化を考えるのは、人類が標準的に使っているからです。16 進数のコード化でも良いのですが、10 進数の方が良く使われているためです。

もちろん、コンピューター内部では整数は 2 進数で計算されます。しかし、10 進数で表せたデータを送りたい場合、コードが用いられます。例えば、計測器が`3498`という数字をコンピューターに送りたい場合、それをいちいち 2 進数に変換するのではなく、3→4→9→8 のように 1 文字ずつ送ることになります。文字コードも使えますが、10 進数の数字と分かっている場合は、数値コードを使ったほうが便利です。

1.2 必要なビット

以前学習したように 10 進数は、0〜9 までの 10 個の数字を用います。10 進数をコード化する場合、

$$2^3 < 10 < 2^4 \qquad (1)$$

なので、少なくとも 4 ビット必要となります。

4 ビット使うと、16 通りの状態を表すことができます。6 通り分が余分となります。この余分な 6 通りを用いて、コードの信頼性を高めることができます。これを自己検査符号を言います。例えば、10 進数を 2 進数に変換したのと同じ 2 進化 10 進コードの場合、使われない、1010, 1011, 1100, 1101, 1110, 1111 が生じたならば、誤りが生じたと判断できます。

このように数値コードでは、余分な状態が生じるため、誤りを検出できるものの、情報の伝達の効率は悪くなります。

2 10進コード

ここでは、世の中で使われている10進数を符号化する方法を示します。いろいろありますが、教科書に載せてある 5 通りの方法を説明します。これらのコードについては、表1にまとめてあります。

- BCD コード
- 3 余りコード
- 2-5 進コード
- 5 中 2 コード

2.1 2進化10進コード

先ほどの説明で、4 桁の 2 進数で$(0)_{10}$ から$(15)_{10}$ までの数を表現できることが分かったと思います。しかし、人間は10 進数に慣れているため、$(0)_{10}$ から$(9)_{10}$ までの数だけを一まとまりの単位で扱いたい場合があります。例えば、

- 数値データの表示。例えば、デジタル温度計の各桁は 0〜9 までの十進数です。それぞれの桁の表示機にデータを送る場合、10 進数の 1 桁ずつデータを分けたほうが表示装置が簡単になります。

です。このように 2 進数で計算された結果(温度)を表示装置に出力する場合などにはよく使用されるのが、2 進化10 進コードです。

4 ビットで10 進数の 1 桁を表します。桁の重み付けは 8, 4, 2, 1 と 2 進数の表現と同じです。このコードを用いると

例えば、$(285)_{10}$ は、001010000101 と表現

となります。10 進数の各桁を 4 ビットの 2 進数に変換して、そのまま並べるだけです。

$(10)_{10}=(1010)_2$から$(15)_{10}=(1111)_2$は使われません。したがって、これらの値が表れたときには、コンピューターで誤りが発生したことになります。

2.2 3余りコード

3 増しコードとも呼ばれます。これも 4 ビットで 10 進数の 1 桁を表します。このコードは、BCD コードに 3 を加えた形になっています。このコードを用いると

例えば、$(285)_{10}$ は、010110111000 と表現

となります。表を見て分かる通り、9 の補数(足すと 9 になる数値)がビット反転で求められます。

このコードの使用例について調べてみたのですが、1954 年の富士通のコンピューター(FACOM100)に使われたことしか分かりませんでした。それにしても、2 進数を使っていながら、演算装置は 10 進数を使っているとは驚きです。

このコードは、現在使用例は少ないと考えられます。教科書に載っているので、加算の説明だけしておきます。余り重要と思われないので、このコードへの深入りは避けます。

加算は、4 ビット毎に区切り

- 桁上がりがあるときは、加算結果が BCD コードになる。
- 桁上がりがないときは、0110 を減算する。そうするとその桁は BCD コードになる。

というルールで計算が可能です。これは、各桁に 3 を加えたのですから、あたりまえですよね。

- 10 進数の桁上がりは 10 を示します。3 余りコードでは、各桁に 3 を加えているので、桁上がりは 16 になります。これは、10000 なので、2 進数でも桁上がりが生じます。
- 一方桁上がりが無い場合は、その 4 ビットで表される 10 進数の桁は、BCD コードに 6 加算されています。BCD コードに直すためには、$(6)_{10}=(0110)_2$ を減算する必要があります。

$(13+99)_{10}$ を教科書に習って、3 余りコードで計算します。13 と 99 の各桁に 3 を加えると、

13	→	1+3=4	3+3=6	3 余りコード	→	01000110
99	→	9+3=12	9+3=12	3 余りコード	→	11001100

です。では、このコードで加算を実行してみましょう。

```
  01000110
+ 11001100
 100010010
```

各桁とも桁上がりがあるので、これが BCD コードになります。したがって、

BCD コード(0001 0001 0010)→$(112)_{10}$

となり加算ができました。めでたし、めでたし!!。

これは桁上がりがある場合で、簡単でしたが、桁上がりが無い場合を考えましょう。教科書の例の $(12+9)_{10}$ です。それでは、それぞれを 3 余りコードで加算してみましょう。

```
  01000101     ← (12)10の3余りコード
+ 00111100     ← (09)10の3余りコード
  10000001
```

ここで、1 の位は桁上がりがあったので、0001 が BCD コードになります。10 の位は、桁上がりが無かったので、1000-0110=0010 が BCD コードになります。

BCD コード(0010 0001)→$(21)_{10}$

となり、計算が完了です。

2.3 2-5 進コード

これは、5 ビットで、10 進数の 1 桁を表すコードです。そして、7 ビットのうち 2 ビットが必ず 1 です。そのため、1 ビットの数を数えることにより、容易に誤りが検出できます。

このコードを用いると

例えば、$(285)_{10}$ は、01000101001000100001 と表現

となります。各桁の重み付けは、表 1 を参照してください。

このコードについても、使用例を調べてみたのですが、1956 年の富士通のコンピューターの FACOM128A しか見つかりませんでした。これも、現在余り使われていないと思われます。深入りはしません。

2.4 5 中 2 コード

これは、5 ビットを用いて 10 進数の 1 桁を表します。特徴は 5 ビットのうち 2 ビットが必ず 1 になることです。5 ビットのうち、2 ビットが 1 になる組み合わせの総数は、

$$ {}_5C_2 = \frac{5!}{(5-2)!\times 2!} = \frac{5\times 4\times 3\times 2\times 1}{3\times 2\times 1\times 2\times 1} = \frac{5\times 4}{2\times 1} = 10 \qquad (2) $$

です。過不足なく、5 ビットのうち 2 ビットを 1 にすることで、10 通りの表現ができます。このコードの場合、誤りの検出が非常に簡単になります。5 ビットのうち 1 になるビットの数が 2 個以外は、誤りが生じたと判断できます。

このコードを用いると

例えば、$(285)_{10}$ は、000111001001010 と表現

となります。

表 1 代用的な 10 進コード。2 段目の数字は、各桁の重み付けです。

	2 進化 10 進	5-4-2-1	3 余り	5 中 2	2-5 進
	8 4 2 1	5 4 2 1		7 4 2 1 0	5 0 4 3 2 1 0
0	0 0 0 0	0 0 0 0	0 0 1 1	1 1 0 0 0	0 1 0 0 0 0 1
1	0 0 0 1	0 0 0 1	0 1 0 0	0 0 0 1 1	0 1 0 0 0 1 0
2	0 0 1 0	0 0 1 0	0 1 0 1	0 0 1 0 1	0 1 0 0 1 0 0
3	0 0 1 1	0 0 1 1	0 1 1 0	0 0 1 1 0	0 1 0 1 0 0 0
4	0 1 0 0	0 1 0 0	0 1 1 1	0 1 0 0 1	0 1 1 0 0 0 0
5	0 1 0 1	1 0 0 0	1 0 0 0	0 1 0 1 0	1 0 0 0 0 0 1
6	0 1 1 0	1 0 0 1	1 0 0 1	0 1 1 0 0	1 0 0 0 0 1 0
7	0 1 1 1	1 0 1 0	1 0 1 0	1 0 0 0 1	1 0 0 0 1 0 0
8	1 0 0 0	1 0 1 1	1 0 1 1	1 0 0 1 0	1 0 0 1 0 0 0
9	1 0 0 1	1 1 0 0	1 1 0 0	1 0 1 0 0	1 0 1 0 0 0 0

重み

3 グレイコード

3.1 グレイコードと 2 進数との変換

グレイコードと 2 進数の変換は、次の様にします。表 2 を見ながら、どのように変換されるか理解してください。

2 進数からグレイコードへの変換

次に示すアルゴリズムに沿って変換します。これは、教科書の方法と同じです。ただし、教科書は表現が粗く分かりにくいですが

①変換したい 2 進数を右に 1 ビットシフトする。

②元の 2 進数と右に 1 ビットシフトした 2 進数のビットごとの排他的論理和 (異なれば 1、等しければ 0) を計算する。排他的論理和の真理値表は、表 2 を参照のこと。これはあとで学習する。

例えば、次のようにする。

$(14)_{10}$ → $(1110)_2$
$(0111)_2$ ← $(1110)_2$ を 1 文字右にシフト。変換①。
$(1001)_2$ ← $(1101)_2$ と $(0111)_2$ の排他的論理和。変換②

これで、2 進数 $(1110)_2$ がグレイコード (1001) に変換できた。

グレイコードから 2 進数への変換

逆のアルゴリズムは、以下の通りです。

①グレイコードの最上位ビット（MSB）からはじめて、最初の 1 が現れるまでは、それまでの 2 進数はグレイビットと同じである。要するに 1 が表れるまで、上位のビットはゼロのままにしておく。

②次にグレイコードの各ビットは制御として用いられる。1 であれば、その前の桁の 2 進数のビットを反転したものが、その 2 進数の桁のビットを表す。0 であれば、その前の桁の 2 進数のビットそのものである。

例えば、次のようにする。

(0101) ← グレイコード
(0???) ← 最上位にビットは、0 なので変換の①に従う。
(01??) ← 次のビットからは、変換②に従う。グレイコード 1、2 進数の前の桁は 0。
(011?) ← グレイコード 0、2 進数の前の桁は 1。
(0110) ← グレイコード 1、2 進数の前の桁は 1。

これで、グレイコード (0101) が 2 進数 $(0110)_2$ に変換できた。

表２　グレイコード

10 進数	２進数	グレイコード
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

表３ 排他的論理和の真理値表

A	B	X XOR B
0	0	0
0	1	1
1	0	1
1	1	0

3.2　グレイコードの特徴

変換の過程から分かる様に、必要なビット数は２進数でもグレイコードでも全く同じであす。これが最初の特徴で、表１に示した 10 進コードにように、情報の無駄がありません。

グレイコードの最も重要な特徴は、それを構成するビット数にかかわらず、１加算や１減算で、変化するビット数は１ビットのみであるということです。表を見て確かめよ。これは、以下のようにして考えれば納得できます。２進数で１を加算または、減算すると変化するビットは下位の連続した n ビットです。下位の連続した n ビット、第 0 から第 n−1 ビットまでが反転します。そして、グレイコードに変換するときは、右に１ビットシフトして、排他的論理和を取るため、グレイコードで最終的に変化するビットは、第 n−1 ビットのみとなります。

$$
\begin{array}{lcl}
 & a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0 & \leftarrow \text{２進数} \\
\text{XOR} & 0\ a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1 & \leftarrow \text{１ビット右シフト} \\
\hline
 & b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0 & \leftarrow \text{グレイコード}
\end{array}
$$

もし、１加算あるいは１減算をして、下位の４ビットの符号が反転したとしよう。ビットの反転を a^-のように表すことにしよう。すると、

$$
\begin{array}{lcl}
 & a_7\ a_6\ a_5\ a_4\ \bar{a}_3\ \bar{a}_2\ \bar{a}_1\ \bar{a}_0 & \leftarrow \text{２進数} \\
\text{XOR} & 0\ a_7\ a_6\ a_5\ a_4\ \bar{a}_3\ \bar{a}_2\ \bar{a}_1 & \leftarrow \text{１ビット右シフト} \\
\hline
 & b_7\ b_6\ b_5\ b_4\ \bar{b}_3\ b_2\ b_1\ b_0 & \leftarrow \text{グレイコード}
\end{array}
$$

となります。1 加算や 1 減算で変化するビットが 1 つのみであることが理解できたでしょうか。

グレイコードの表す 0 と、最大の数のビットについても、1 ビットだけ違うことに特に注意しよう。表 2 の $(0)_{10}$ と $(15)_{10}$ に対応するグレイコードの違いは、たったの 1 ビットです。このように、グレイコードは巡回（cyclic）コードです。

表 2 を見ると、ビットパターンの対称性が非常に良いことが分かるでしょう。

3.3 グレイコードの応用

2 進法では、 0111 の後は 1000 という具合いに、1 を加えることにより、多くの文字が同時に変化することがあります。しかし、グレイコードでは、必ず 1 文字しか変化しません。この特徴を利用して、連続に変化する値を測定してデジタル値として出力するセンサーの出力にグレイコードがよく用いられます。

センサーの入力が微妙に変化しただけで出力の複数の桁が同時に変わることがあります。その変わった瞬間の出力を取り込んだコンピューターが、とんでもなく違った値に解釈する恐れがあります。例えば、ある測定量が $(7)_{10}$→$(8)_{10}$ に値が変化[1]したとします。これを 4 ビット、すなわち 4 本の出力線(電線)でコンピューターにデータを送る状況を考えます。2 進数とグレイコードを用いた場合、それぞれのビット(4 本の電線の電圧)は、

2 進数	(0111)→(1000)
グレイコード	(0100)→(1100)

となります。しかしこれは、最初と最後の状態です。最後の状態になるまでの途中の状態を考えましょう。実際には、同時に全部のビットが変化することはなく、それぞれのビットは異なったタイミングで変化します。実際には

2 進数	(0111)→(0011) →(1011) →(1010) →(1000)
グレイコード	(0100)→(1100)

のように変化します。これは例に過ぎません。途中の状態は装置に依存します。2 進数の場合、途中、中途半端な状態があることは確かです。すると、この中途半端な状態をコンピューターが読み込んで、その変な値をデータとして記録してしまいます。これは、うまくありません。一方、グレイコードの場合、1 ビットしか変化しませんので、その心配はありません。グレイコードはうまくできています。

このグレイコードは､角度を測るロータリーエンコーダーに利用されているのをよく見かけます。それは、図 1 のようにモーターとよく似た形をしています。シャフトに角度を測定する対象物を取り付けます。その対象物の回転角度により、つながれたケーブルを通して角度情報が出力されます。

---

[1] 測定量は必ず連続的に変化します。測定量が $(7)_{10}$→$(8)_{10}$ のように不連続に変化することは自然界ではありえません。

図１ ロータリーエンコーダーの例

このロータリーエンコーダーの角度測定方法について、説明します。仕組みは単純で、以下の通りです。

- 発光素子から出た光はスリットを通って、回転板を通して、受光素子まで届きます。
- 発光素子と受光素子を結ぶ線上の回転盤が不透明な遮光であれば、光は届きません。一方透明であれば、光が受光部まで届きます。これが０と１のビットに対応します。
- 受光素子と発光素子をＮ個並べることにより、Ｎビットの角度情報を得ることができます。
- このＮビットのパターンが、軸の回転角度により変化することで、角度を読み取ります。
- 回転板の遮光パターンがグレイコードになっています。

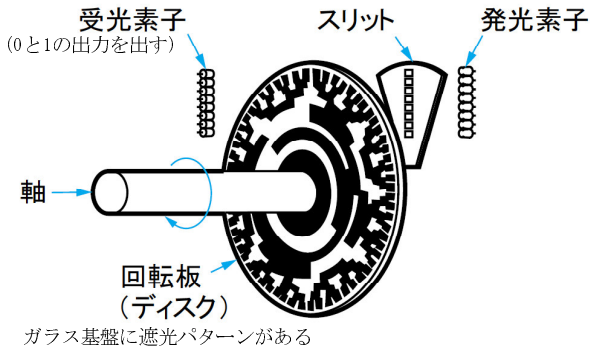

図２ ロータリーエンコーダーの原理を説明する図。
www.fa.omron.co.jp/lineup/not/pdf/en_gaiyo.pdf より転載

パターンがグレイコードになっていることにより、角度の測定結果はグレイコードになります。もしグレイコードを使わないと、測定の境でとんでもない値を出力することがあります。

4 コードの偶奇 (パリティ)

教科書では「奇偶」と書いていますが、これは間違いで、正しくは「偶奇」です。偶数と奇数を表し、偶奇性を英語でパリティと言います。そんな名前のことよりも、本題に入ります。

デジタルデータは 1 ビットの間違いが、とんでもなく大きな情報の変化を引き起こします。1 ビットの間違いにより、飛行機の出力アップすべきところをダウンさせることも考えられます。こうなると、非常に危険です。

1 ビットの重要性を認識して、ここ秋田からチベットの山奥にメッセージを送ることを考えます。例えば、1M バイトのメッセージを送るとします。1M バイトとなると、約 800 万ビットです。800 万の 0 と 1 をひとつも間違いなく、チベットの山奥まで送ることができるでしょうか?。秋田から、チベットまでメッセージを送るとなると、いろいろな通信回線や機器を経由します。いろいろなところで雑音が入ります。デジタルは雑音に強いですが、800 万回で 1 つもエラーが生じないかといえば疑問です。雑音以外に、途中の機器が故障したり誤りを犯す可能性もあります。こうなると、デジタル通信は、非常に脆弱なシステムに思えてきます。なんとか、信頼性を上げてデータを送ることを考えなくてはなりません。

装置の信頼性を上げることで対処しますか?。ノイズに強くするために、電圧を上げる。ノイズの少ない素子で通信機器をつくる。このようなことが考えられますが、非常にコストがかかります。また、データ量が多くなると、間に合わなくなります。

そこで、発想を変える必要があります。データを受け取る側に、送られてきたビットに誤りの有無の確認手段があれば、この問題が解決できることが分かります。誤りがあれば、再度データを転送してもらえばよいのです。

そのひとつの方法が教科書に書いてあるパリティ検査です。例えば、1 桁 4 ビット BCD コードを送る場合、パリティビットを付け加えて、5 ビットで通信します。パリティビットを 0 あるいは 1 にすることにより、1 の数を偶数あるいは奇数にする方法です。例えば、奇数パリティの場合、必ず 5 ビットの 1 の数が奇数になるようにします。すると受信者側で 5 ビット毎に、1 になっているビット数を調べて、それが偶数ならば、誤りがあると判断できます。

送られてきた 5 ビットに 2 個所誤りがあると、正しいデータと判断してしまいます。このような確立は、非常に低くなります。例えば、1 ビットのデータの誤りが発生する確率を 1/10000 としましょう。すると、5 ビット中 2 ビットに誤りが発生する確率は、

$$ {}_5C_2 \times \left(\frac{1}{10000}\right)^2 \left(1-\frac{1}{10000}\right)^3 \approx \frac{1}{10000000} \qquad (3) $$

です。5 ビット毎にデータを送って、1/１千万で誤りが生じます。要するに、5 千万ビットデータを送信して 1 回誤りが生じます。この方法でチベットに 800 万ビットデータを送ると、誤りが発生する確率はかなり低くできます。パリディビットを付加しないと、800 ビットの誤りが発生することを考えると、かなりの改善です。

このパリティビットを用いる方法は、誤りの有無が分かるだけで、場所は分かりません。実際の通信では、もっと複雑な方法を用いて、誤りのビットが検出できるものまであります。そのへんの議論は、学習の範囲をこえますので、興味のある人は調べてください。

ちょっと、議論を後戻りさせて、先ほどのパリティビットの付加では、送りたい情報が 4 ビットに対して通信に 5 ビット必要となります。これを符号の冗長化と言います。そして、その度合いを表す冗長度 R と言われる量が定義されています。

$$ R = \frac{\text{全体で使われるビット 数}}{\text{メッセージのビット数}} \qquad (4) $$

したがって、4 ビットの BCD コードに 1 ビットのパリティビットを付加した場合の冗長度は、R=1.25 となります。

参考資料

グレイコードについては、以下が詳しい。

etlab.mis.ous.ac.jp/computer02/0261/ending.html
www.i.h.kyoto-u.ac.jp/~tsuiki/bit/gray.html

誤りの検出と符号の訂正は、以下が分かりやすい。

A．ヘイ，R．アレン編，ファインマン計算機科学，岩波書店