「　基数変換とビット操作　」

東京情報クリエーター工学院

基数変換とビット操作

◆10 進数・2 進数・8 進数・16 進数

普段私たちが使用している数は、「　**10 進数**　」と呼ばれるもので、**0 から 9 までの 10 種類の数字を使って 9 の次が一つ桁上がり**します。
コンピュータの世界でよく使用される 2 進数、8 進数、16 進数も同じイメージで、

2 進数：0 から 1 までの 2 種類の数字を使って 1 の次が一つ桁上がり
8 進数：0 から 7 までの 8 種類の数字を使って 7 の次が一つ桁上がり
16 進数：0 から 9 まで数字と A・B・C・D・E・F の 16 種類の数字を使って F の次が一つ桁上がり

となります。

【練習】
2 進数・8 進数・10 進数・16 進数の対応表を作りましょう。

◆基数

「　**基数**　」とは、数値を表現する際に、**各桁の重み付けの基本となる数**のことです。10 進数では、10 倍ごとに桁が上がっていくので、基数は 10 となります。

【例】10 進数 $\mathbf{123.45}$ は、

$$\underset{\uparrow}{\mathbf{1}\times\mathbf{10}^{\mathbf{2}}}+\underset{\uparrow}{\mathbf{2}\times\mathbf{10}^{\mathbf{1}}}+\underset{\uparrow}{\mathbf{3}\times\mathbf{10}^{\mathbf{0}}}+\underset{\uparrow}{\mathbf{4}\times\mathbf{10}^{\mathbf{-1}}}+\underset{\uparrow}{\mathbf{5}\times\mathbf{10}^{\mathbf{-2}}}$$
$$\mathbf{100}\quad\mathbf{10}\quad\mathbf{1}\quad\mathbf{1/10}\quad\mathbf{1/100}$$

例をみてわかるように、10 進数の各桁の重みは、小数点を基準に左へ $\mathbf{10^0, 10^1, 10^2, \cdots}$ と増えていき、右へ $\mathbf{10^{-1}, 10^{-2}, 10^{-3}, \cdots}$ と減っていきます。

2 進数、8 進数、16 進数の基数はそれぞれ 2、8、10、16 となりますから、各桁の重みも同じようになります。

小数点

10 進数	…	$\mathbf{10^2}$	$\mathbf{10^1}$	$\mathbf{10^0}$	.	$\mathbf{10^{-1}}$	$\mathbf{10^{-2}}$	…
2 進数	…	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$	.	$\mathbf{2^{-1}}$	$\mathbf{2^{-2}}$	…
8 進数	…	$\mathbf{8^2}$	$\mathbf{8^1}$	$\mathbf{8^0}$	.	$\mathbf{8^{-1}}$	$\mathbf{8^{-2}}$	…
16 進数	…	$\mathbf{16^2}$	$\mathbf{16^1}$	$\mathbf{16^0}$	.	$\mathbf{16^{-1}}$	$\mathbf{16^{-2}}$	…
N 進数	…	$\mathbf{N^2}$	$\mathbf{N^1}$	$\mathbf{N^0}$	.	$\mathbf{N^{-1}}$	$\mathbf{N^{-2}}$	…

← 増える　　減る →

「　基数変換とビット操作　」

東京情報クリエーター工学院

◆2進数･8進数･10進数･16進数の対応

１０進数	２進数	８進数	１６進数
０			
１			
２			
３			
４			
５			
６			
７			
８			
９			
１０			
１１			
１２			
１３			
１４			
１５			
１６			
１７			
１８			
１９			
２０			
２１			
２２			
２３			
２４			
２５			
２６			
２７			
２８			
２９			
３０			
３１			
３２			
３３			

「　基数変換とビット操作　」

東京情報クリエーター工学院

◆基数変換

10 進数の「 10 」を 2 進数で表すと「 1010 」になります。このように、ある数を別の進数で表しなおすことを「　**基数変換**　」といいます。

１．10 進数 → 2 進数

(1)整数

10 進数を 2 進数に変換するには、変換したい 10 進数を**「　商が 0　」**になるまで

「　2 で割りつづけ　」、商と**「　商と余り　」**を求めます。

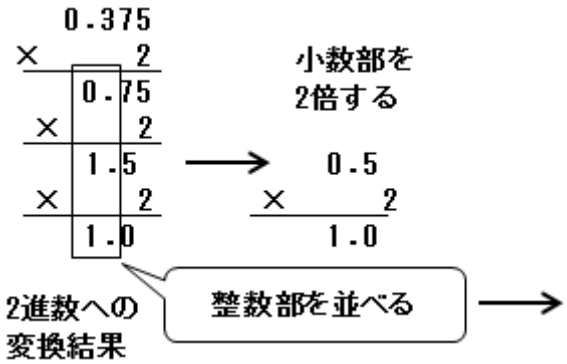
【例題】10 進数「60」を 2 進数に変換

※10 進数から 8 進数・16 進数への変換も同様に行うことができます。

(2)小数

小数の 10 進数を 2 進数に変換するには、変換したい 10 進数の小数部を「 **小数が 0** 」になるまで「 **2 倍** 」します。これで求めた整数部が 2 進数への変換結果です。

【例題】10 進数「0.375」を 2 進数に変換



注意！

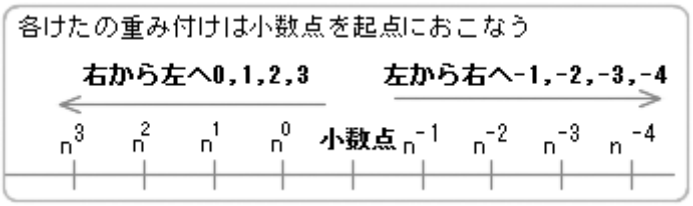
10 進数に無限小数(1/3)があるように
2 進数にも無限小数がある。
例：10 進数の 0.2 を 2 進数にすると、
0.00110011…となる

「　」

２．2 進数 → 10 進数

2 進数から 10 進数への変換方法は、整数部・小数部同じ方法でできます。

「 **2 進数の各 けたの数値とけたの重みをかけたものの合計** 」を求めればよい。



「　基数変換とビット操作　」

東京情報クリエーター工学院

【例題】2 進数の 111100.101 を 10 進数に変換する

2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
1	1	1	1	0	0	.	1	0	1

$$2^5 \times 1 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 1 + 2^{-1} \times 1 + 2^{-3} \times 1$$

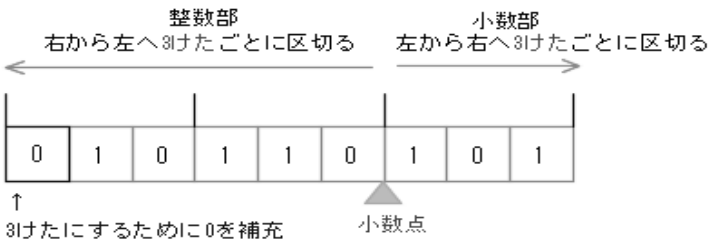
$$=$$

※8 進数から 10 進数・16 進数から 10 進数への変換も同様に行うことができる。

３．2 進数⇔8 進数

「 **小数点** 」を基点にして、「 **3 けた** 」ごとに区切り、それぞれを 8 進数に変換する。3 桁にならない場合は、「**不足している桁数分 0 を補充**」する。

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



8 進数に変換

2	6	.	5

逆に 8 進数を 2 進数に変換するときは、

「**8 進数 1 桁に対して 2 進数 3 桁**　」の対応で変換する。

2	6	. 5

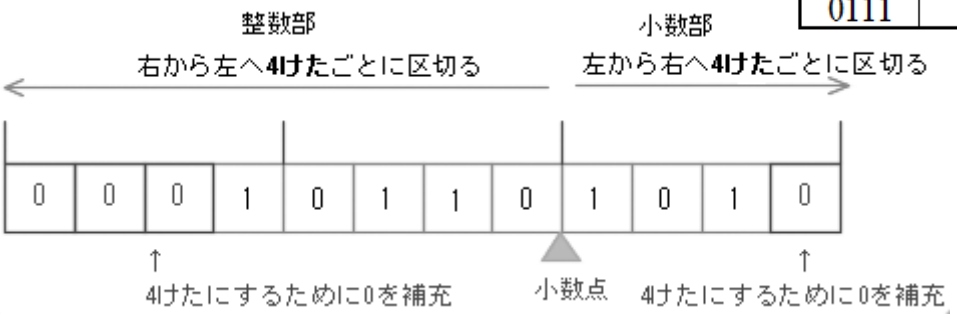
8 進数 1 桁に対し 2 進数 3 桁で

010	110	. 101

４．2 進数⇔16 進数

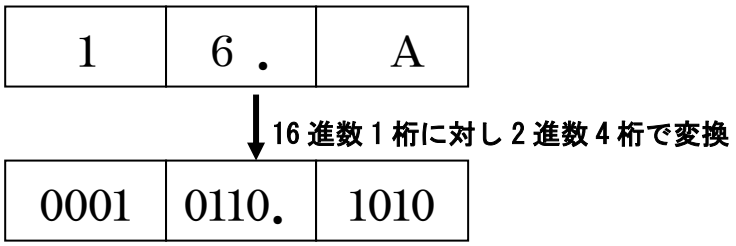
「 **小数点** 」を基点にして、「 **4 けた** 」ごとに区切り、それぞれを 16 進数に変換する。4 桁にならない場合は、「**不足している桁数分 0 を補充**」する。右の表が対応表。

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



逆に 16 進数を 2 進数に変換するときは、

「**16 進数 1 桁に対して 2 進数 4 桁　**」の対応で変換する。



◆演習問題1

次の表の空欄を埋めましょう。

2進数	1011010			
8進数		33.4		
10進数			16	
16進数				3C.A

◆例題1

10 進数→2 進数・8 進数・16 進数、2 進数・8 進数・16 進数→10 進数と変換するプログラムを作成しましょう(入力値は整数とします)。

１．準備

(1)ソリューション「Calculation」の中に新しいプロジェクト「Ex03」を作成する。

(2)作成した「Ex03」を「スタートアップ プロジェクトに設定」

２．プログラムの作成

C#では、基本的には変数変換は Convert を使用するだけで可能です。

ただし、10 進数以外は数値型が存在しないのでコード上で 2,8,16 進数は文字列型で表します。

進数文字列を各進数に変換するということですね。

「　基数変換とビット操作　」

東京情報クリエーター工学院

◆論理演算

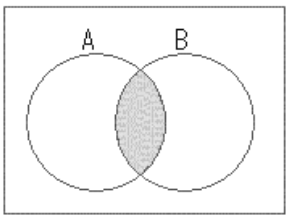
１．論理積（AND）

論理積（AND）は、入力値が「　　」のときに「　　」を出力する。
それ以外の入力値のときは「　　」を出力する。

真理値表

A	B	A ・ B
0	0	0
0	1	0
1	0	0
1	1	1

ベン図



論理式

$A \cdot B$

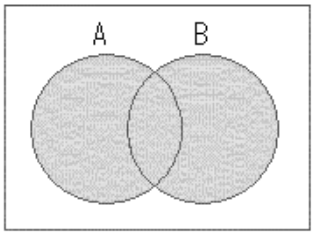
２．論理和（OR）

論理和（OR）は、入力値に「　　」ときに「　　」を出力する。入力値がどちらも「　　」のときのみ、「　　」を出力する。

真理値表

A	B	A ＋ B
0	0	0
0	1	1
1	0	1
1	1	1

ベン図



論理式

$A+B$

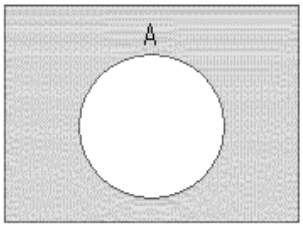
３．否定（NOT）

否定(NOT)は入力された値が「　　」なら「　　」に、「　　」なら「　　」に反転する。

真理値表

A	$\overline{A}$
0	1
1	0

ベン図



論理式

$\overline{A}$

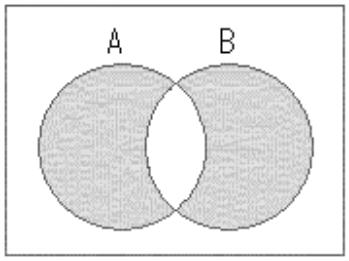
４．排他的論理和（EOR、XOR）

排他的論理和は、入力値が「　　」とき「　　」を出力する。入力値が「　　」ときは、「　　」を出力する。

真理値表

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

ベン図



論理式

$$A \oplus B = (A+B) \cdot \overline{(A \cdot B)} = \overline{A} \cdot B + A \cdot \overline{B}$$

【例題１　プログラム例】

```
static void Main(string[] args)
{
    string str, binary, octal, hexa;
    int D, B;

    // 10進数の入力
    Console.Write("10進数を入力してください : ");
    str = Console.ReadLine();
    D = Int32.Parse(str);

    // 10進数をn進数に変換
    binary = Convert.ToString(D, 2);  //2進数に変換
    octal = Convert.ToString(D, 8);   //8進数に変換
    hexa = Convert.ToString(D, 16);   //16進数に変換

    // 表示
    Console.WriteLine("2進数 " + binary);
    Console.WriteLine("8進数 " + octal);
    Console.WriteLine("16進数 " + hexa);

    // 2進数の入力
    Console.Write("2進数を入力してください : ");
    str = Console.ReadLine();

    // 2進数を10進数に変換
    B = Convert.ToInt32(str, 2);
    Console.WriteLine("10進数 " + B);

    //8進数の入力
    Console.Write("8進数を入力してください : ");
    str = Console.ReadLine();

    // 8進数を10進数に変換
    B = Convert.ToInt32(str, 8);
    Console.WriteLine("10進数 " + B);

    //16進数の入力
    Console.Write("16進数を入力してください : ");
    str = Console.ReadLine();

    // 16進数を10進数に変換
    B = Convert.ToInt32(str, 16);
    Console.WriteLine("10進数 " + B);
}
```

「　基数変換とビット操作　」

東京情報クリエーター工学院

◆論理演算の利用

論理演算によるビット列処理は、実際のプログラムで多用されているので、覚えておくと便利です。

(1) ビットの取り出し

論理積を利用します。**取り出したいビット部分だけを１にしたもの（マスク）**との「　　」を計算することで、必要なビットだけを取出すことができます。

【 例 】　2 進数 8 ビットの上位 4 ビットを取り出したい場合

```
  1 1 0 1 1 0 1 0
AND) 1 1 1 1 0 0 0 0
  1 1 0 1 0 0 0 0
```

(2) ビットの反転

排他的論理和を利用します。**反転したいビット部分だけを１にしたもの**との「　　」を計算することで、ビットを反転することができます。

【 例 】　2 進数 8 ビットの下位 4 ビットを反転する

```
  1 1 0 1 1 0 1 0
XOR) 0 0 0 0 1 1 1 1
  1 1 0 1 0 1 0 1
```

(3) 特定ビットをビット列で埋める（１にする）

論理和を利用します。**１にしたいビット列部分だけを１にしたもの**との「　　」を計算することで、特定のビットを 1 にすることができます。

【 例 】2 進数 8 ビットの上位 4 ビットを 1 にする

```
  1 1 0 1 1 0 1 0
OR) 1 1 1 1 0 0 0 0
  1 1 1 1 1 0 1 0
```

(4) ビット列のクリア（０にする）

クリアしたいビット列のみ０とし、「　　」をとることによってビット列をクリアすることができます。また、**値の同じビット列どうし**の「　　」を計算することで、ビット列をクリア（０に）することもできます。

【 例 】2 進数 8 ビットの下位 4 ビットを０にする

```
  1 1 0 1 1 0 1 0
AND) 1 1 1 1 0 0 0 0
  1 1 0 1 0 0 0 0
```

```
  1 1 0 1 1 0 1 0
XOR) 0 0 0 0 1 0 1 0
  1 1 0 1 0 0 0 0
```

◆単位と補助単位

主記憶装置（メモリ）やハードディスクの容量はバイト（byte）で表します。**1 バイト**は「　　」を集めたものです。ビットとはコンピュータにおける情報の最小単位です。1 ビットで 2 進数の 1 桁「0・1」」を表します。

補助単位	よみ	10^n	2^n	補助単位	よみ	10^n
K	キロ	10^3	2^{10}	m	ミリ	10^{-3}
M	メガ	10^6	2^{20}	μ	マイクロ	10^{-6}
G	ギガ	10^9	2^{30}	n	ナノ	10^{-9}
T	テラ	10^{12}	2^{40}	p	ピコ	10^{-12}

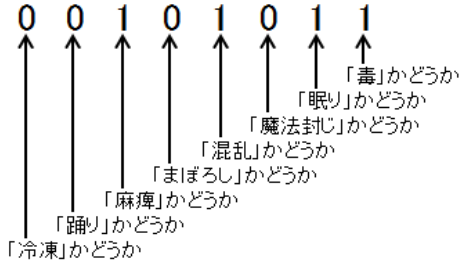
「　基数変換とビット操作　」

東京情報クリエーター工学院

◆例題2

今、ゲーム上でプレイヤーの状態異常が、「毒」「眠り」「魔法封じ」「混乱」「まぼろし」「麻痺」「踊り」「冷凍」と 8 種類あったとします(適当です…)。
この状態になっているかどうか(true か false)を管理するフィールドとして、8 個の変数を用意すると大変ですよね。(さらに状態が増えたら、プログラムはとても見にくくなります。)

そこで、ここでは、「int 型」のような数字を扱う型で「**byte 型**」というものを使ってみましょう。
byte 型は、「**8 ビット**」分を使って数を保存できます。ちなみに、C#では int 型とすると、32 ビット分ですから、データ量の制限が厳しい時には、役に立ちます。
通常は、「**数を表すために 2 進数が並んでいる**」わけですが、これを、「**0 と 1 の集まり**」として利用してしまうわけです。



数字の並びの各場所を見て、
「1」：状態異常である
「0」：状態異常ではない
と判断するわけです。

これをプログラムにしてみましょう。

１．プログラムを作成するための知識

(1)ビット演算子

ビット演算子は、「1 ビット」の者同士になんらかの計算を行うものです。

①&(AND 演算子)
論理演算の論理積(AND)を行うものです。

②|(OR 演算子)
論理演算の論理和(OR)を行うものです。

③~(NOT 演算子)
論理演算の否定(NOT)を行うものです。

④^(XOR 演算子)
論理演算の排他的論理和(XOR)を行うものです。

(2)シフト演算子

シフト演算子は、「ビット列を左右に動かす」ものです。

①<<(左シフト演算子)
2 番目のオペランドで指定されたビット数だけ最初のオペランドが左にシフトされます。

②>>(右シフト演算子)
2 番目のオペランドで指定されたビット数だけ最初のオペランドが右にシフトされます。

【 例 】

```
1010 & 1100 = 1000          1010 | 1100 = 1110
1010 << 1 ＝ 10100          1010 << 2 = 101000
1010 >> 1 ＝ 101            1010 >> 2 = 10
```

(3) ＩＦ文

状態の確認など、何か判断をするときには、「**ＩＦ文**」を使って分岐します。条件には、比較演算子を使います。

「　基数変換とビット操作　」

東京情報クリエーター工学院

```
if(条件式)
{
        処理
}
else
{
        処理
}
```

比較演算子

演算子	使い方	意味
＝＝	A==B	A と B は等しい
＜	A<B	A は B より小さい
＞	A>B	A は B より大きい
＜＝	A<=B	A は B 以下
＞＝	A>=B	A は B 以上
！＝	A!=B	A と B は等しくない

２．準備

(1)ソリューション「Calculation」の中に新しいプロジェクト「Ex04」を作成する。

(2)作成した「Ex04」を「スタートアップ プロジェクトに設定」

３．プログラムの作成

(1)フラグの設定

まず、初期状態「00000000」を設定し、「毒」のフラグを「オン」にしてみましょう。

このときは、「｜(OR 演算子)」を使います。

```
static void Main(string[] args)
{
        //状態管理用
        byte state = 0;
        //初期状態表示
        string strState = Convert.ToString(state, 2).PadLeft(8, '0');
           //PatLeft(8,0)は 8 桁表示にして、空いている部分に「0」を左から埋めていきます
        Console.WriteLine("****初期状態****");
        Console.WriteLine(strState);

        //「毒」を「オン」にする
        byte stateOn = 1;  //設定用の変数
        state = (byte)(state | stateOn);
        //表示
        strState = Convert.ToString(state, 2).PadLeft(8, '0');
        Console.WriteLine("****毒状態オン****");
        Console.WriteLine(strState);
}
```

次に「眠り」を「オン」にしてみましょう。上記のプログラムに追加します。

「stateOn」の設定には「<<(左シフト演算子)」を使います。

```
        //「眠り」を「オン」にする
        stateOn = 1 << 1;
        state = (byte)(state | stateOn);
        //表示
        strState = Convert.ToString(state, 2).PadLeft(8, '0');
        Console.WriteLine("****眠り状態オン****");
        Console.WriteLine(strState);
```

(2)フラグの解除

今度は、「眠り」を「オフ」にしてみましょう(反転する)。上記のプログラムに追加します。
オフにするためには、「＾(XOR 演算子)」を使います。

```
//「眠り」を「オフ」にする
byte stateOff = 1 << 1;       //解除用の変数
state = (byte)(state ^ stateOff);
//表示
strState = Convert.ToString(state, 2).PadLeft(8, '0');
Console.WriteLine("****眠り状態オフ****");
Console.WriteLine(strState);
```

(3)フラグの確認

最後に「眠り」状態を確認する処理を追加しましょう。
状態の確認には、「&(AND 演算子)」を使います。

```
  //「眠り」かどうかチェックする
byte stateCheck = 1 << 1;    //チェック用の変数
Console.WriteLine("****眠り状態の確認****");
if((state&stateCheck)!=0)
{
    Console.WriteLine("「眠り」状態です");
}
else
{
    Console.WriteLine("「眠り」状態ではありません");
}
```

他のフラグも同じように、設定・解除・確認ができるので、自分で設定・解除・確認をやってみましょう。