アーキテクチャ　第３回　追加メモ　論理式・真理値表・論理回路・出力と入力・マスク・加算回路

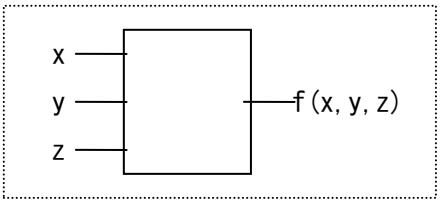
論理値

授業でも一言触れたが、論理代数（ブール代数）では、値は真（TRUE、1と書く）か偽（FALSE、0と書く）かの2値である。これが2進数の1桁＝1ビットと対応する。

論理式

論理式は、①論理変数・定数と、②論理演算子、からなる式で、結果は論理値になる。
普通の算術式を考えてみよう。 式 $(3 \times x + 5 \times y)$ は、定数 3、変数 x, y、演算子 × と ＋ から成り、式の値は、変数 x や y に値を与えると計算することが出来る。
同じように、論理式 $((x \text{ AND } y) \text{ OR } z)$ は、変数 x, y, z、演算子 AND と OR から成り、式の値は、変数 x, y や z に値を与えると計算することが出来る。 仮に x に1（TRUE）、y に0（FALSE）、z に1（TRUE）を与えると、$(x \text{ AND } y)$＝(1 AND 0)＝0となり、$((x \text{ AND } y) \text{ OR } z)$ は(0 OR 1)＝1となるので、式の値は1である。

論理式は、変数の値を入力とした関数と見ることが出来る。 たとえば上記の例では、$f(x, y, z)$＝$((x \text{ AND } y) \text{ OR } z)$.と書ける。 関数は、引数 x, y, z を入力とし、関数値を出力とする箱と見ることも出来る。 その物理的な実現として、論理回路が考えられる。 右図のように、 3本の入力 x, y, z と関数値1本の出力とを持つ箱として書くことが出来る。



基本的な演算子

論理演算（ブール代数）の演算子（AND とか OR とか）にはいろいろなものがあるが、基本として使われるのは、AND（かつ、論理積）、OR（または、論理和）、NOT（否定）の組合せである。
AND は2入力－1出力 （変数が2つ、結果が1つ）の演算子（関数）であり、2つの入力 x と y がいずれも真（True、T、1）の時に限り出力（x AND y）が真になるが、その他の場合（つまり x と y のいずれかが偽（False、F、0）の時は出力は偽になる。つまり（0 AND 0）＝0、（0 AND 1）＝0、（1 AND 0）＝0、（1 AND 1）＝1、である。
OR も2入力－1出力の演算子だが、2つの入力 x と y のいずれかが真であれば出力（x OR y）は真となり、両方が偽の時に限り出力が偽となる。 つまり、（0 OR 0）＝0、（0 OR 1）＝1、（1 OR 0）＝1、（1 OR 1）＝1、である。
NOT は1入力－1出力の演算子で、入力xが真であれば出力（NOT x）は偽、入力 x が偽であれば出力（NOT x）は真になる。つまり、（NOT 1）＝0、 （NOT 0）＝1 である。
もちろん他の演算子もある（たとえば排他的論理和、Exclusive OR や、NAND＝Not-AND、NOR＝Not-OR など）。 ブール代数の世界では、どの演算子を組み合わせるとすべての演算が書けるか、などが議論になるが、ここでは追求しない。
AND・OR・NOT の組合せはすべての演算を書ける。
（余談） 演算子を記号で書くとき、AND を∧、OR を∨、NOT を¬または～で表すことがある。 また、プログラミング言語ではそれぞれを＆、｜、！ （いずれも半角文字）で表すこともある。

論理回路（脱線）

論理式の、論理変数を信号線で、演算子を論理（回路）素子で置き換えることが出来る。 たとえば、論理式 (A AND B) は、

A を信号線 A で、B を信号線 B で、AND を論理素子の記号で置き換えると 

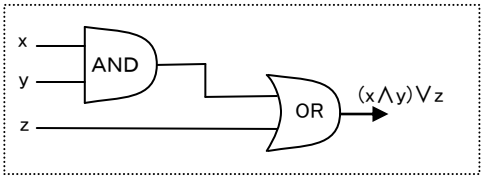
のようになる。 同様に OR

を表す記号 

、NOT を表す記号 

が使われる。

たとえば、論理式（論理関数） $f(x, y, z)=((x \text{ AND } y) \text{ OR } z)$ を論理回路で実現する場合、.3本の入力 x, y, z をもち、出力はこの関数の値$((x \text{ AND } y) \text{ OR } z)$を1本持つが、その内容は右の図のように、まず入力 x と y の AND を計算し、その出力（x AND y）と入力 z との OR を計算して、その出力(x AND y) or z)を全体の出力とするような論理回路に当たることになる。



真理値表

論理式（論理関数）の変数（＝入力）と式の値（出力）の関係を、表の形に書き表したものが、真理値表である。一般に、すべての変数の組合せに対して、論理式（論理関数）の値を記述してある。たとえば、2入力の関数である (x AND y) は、すべての変数（入力）の組合せ (x, y)＝(0, 0), (0, 1), (1, 0), (1, 1) に対して、式の値（出力）が

(0 AND 0)＝0, (0 AND 1)＝0, (1 AND 0)＝0, (1 AND 1)＝1

であるから、表はxとyを縦横に書いて表の x, y の組合せの位置に式の値を書いたり（右上の表）、または、右の下の表のようにすべての入力のパターンを一列に書いてその右に式の値を書いたりする。 どちらの形式も可である。

x \ y	0	1
0	0	0
1	0	1

x	y	式の値
0	0	0
1	0	0
0	1	0
1	1	1

変数の数が3つ以上になると、上の形式はうまく書けないので、下の形式が主となる。

真理値表を使うと、任意の関数（入力と出力の関係）を作ることが出来る。 入力（変数）の値のパターンは2変数であれば4通りの組合せしかないが、その4つの組合せのどれに対しても、任意の出力値（0か1か）を書くことができる。 そうしてできた論理関数を、たとえば AND、OR、NOT の組合せで表現できるか、というのが、上に書いた「すべての演算を書ける」の問題である。

［練習問題］ 3つの変数（入力）x, y, z をもつ式（関数） $((x \text{ AND } y) \text{ OR } z)$ の真理値表を書いてみよ

余談 加法標準形

任意の論理関数（真理値表）は、NOT-AND-OR の3段で表せるというのが「加法標準形」といわれる話である。 NOT を含む適当な組合せで掛けて(AND)おいて、足す(OR)、という考え方である。

例として、(x AND y AND (NOT z)) OR ((NOT x) AND y AND z) のような形である。 NOT が最初（変数に直接）に掛かり、次にそれらの項が AND され、最後にその AND 項が OR される。

任意の式が表されるという証明は自分で調べてもらうとして、簡単な説明としては、縦長の真理値表を考えて欲しい。変数（入力）側の欄は、入力値の 0/1 の組合せをすべて作っているのだが、当該の欄のパターンのときだけ1になるような式を作ることは簡単に出来る。実はそれが、各変数に必要に応じて NOT を掛けた物を変数の個数だけ AND した項に対応する。つまり x=1 で y=1 で z=0 の欄であれば(x AND y AND (NOT z))のときだけ1になり、他のときは0である。 同様に、x=0 で y=1 で z=1 のときだけ1になるような式は、((NOT x) AND y AND z)と書ける。

もし、真理値表がこの2つの欄だけ1でその他の欄は0であれば、式 (x AND y AND (NOT z)) OR ((NOT x) AND y AND z) がこの真理値表を表している。 つまり、右の真理値表に対応する論理関数が作れたわけである。

今の作り方は、表の左側の x, y, z の値の組合せに対して、式の値がどのようであっても同じ原理で作ることが出来ることが分かるだろう。 つまり、この作り方は任意の真理値表に対する論

x	y	z	式の値
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	0

理式を作ることが出来る。
だから、真理値表さえ書くことができれば、それの論理回路による実現はいつでも可能、ということになる。
［練習問題］ 上で得た論理関数 f(x,y,z)＝(x AND y AND (NOT z)) OR ((NOT x) AND y AND z) の真理値表を書いてみよう。つまり、(x, y, z) のすべてのパターン (0,0,0), (0,0,1) .… (1,1,1) に対する f(x,y,z) の値を計算し、表に書いてみよ。
[練習問題] 上記の f(x, y, z) を回路図に描いてみよ。

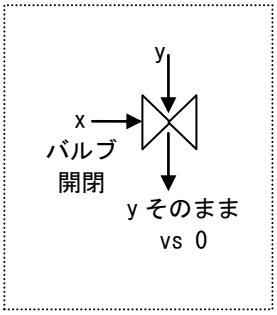
マスク・バルブ

x \ y	0	1
0	0	0
1	0	1

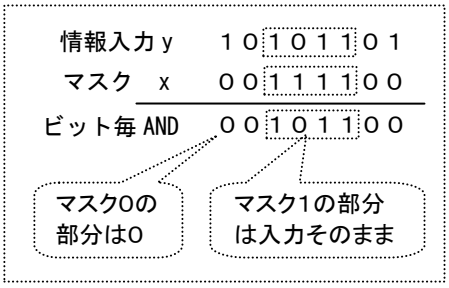
ビットごとの AND 演算を、マスクまたはバルブと見なすことがある。 まず1ビットについて AND 演算を考える。 AND 演算子は右の真理値表に示す動作をするが、これを

x ⇒ 制御入力、 y ⇒ 情報入力

と見なすことにしよう。 つまり、制御入力 x が 0 か 1 かによって動作が変わる箱だと思うことにする。制御入力 x が 0 の時は、右の真理値表で x=0 の行を見ることになるが、y の値が 0 でも 1 でも、出力は 0 になる。他方、制御入力 x が 1 の時は、x=1 の行を見ることになるが、y が 0 なら出力は 0、y が 1 なら出力は 1 になっている。つまり情報入力 y の値がそのまま出力に現れている。これらをまとめると、制御入力 x が 1 なら情報入力 y の値が出力され、0 なら情報入力 y の値にかかわらず 0 が出力される。 制御入力 x をバルブの開け閉めと考え、x=1 ならバルブ開で y が出力へ直接に流れ、x=0 ならバルブ閉で 0 しか出てこない、と見れば、右図のようにみなすことができる。



多桁のデータに対するビットごとの AND は、(ビット)マスクと見なすことができる。1つ1つの桁はバルブと同様に、AND 動作を行うとすると、制御入力 x が 0 だと出力はバルブ閉で 0、制御入力 x が 1 だと出力はバルブ開なので情報入力 y の値がそのまま出てくる。 これを複数の桁の分を並べると、マスク x が1のビット部分は情報入力 y の値がそのまま現れ、マスク x が 0 のビット部分は 0 になる。 つまり元の y の入力パターンを、マスク x の 0 の位置をマスクした (1 の位置だけを打ち抜いた) 出力が得られる。



プログラミング言語ではビットごとの AND 演算の演算子が用意されている。たとえば Java 言語では & がビット AND の演算子の記号である。 これを使うと、上記のようなマスクを取ることができる。 たとえば、入力 y のうち上から 5, 8～10、15～17、22、31 ビット目だけを残して、それ以外のビットを全部 0 にする(マスクしてしまう)ためには、図のようなマスクパターン x を用意して、

情報入力 y	0101 1010 0101 1010 0101 1010 0101 1010
マスク x	0000 1001 1100 0011 1000 0100 0000 0010
ビット毎 AND	0000 1000 0100 0010 0000 0000 0000 0010

```
x = 0x00001001110000111000010000000010;
kekka = y & x;
```

のように書けばよい。

教科書の図 5.8 の加算回路の見方（おまけ）

左側の A0, … A3 は加算の入力Aのそれぞれの桁(0 か 1)で、A0 が下の桁、A3 が上の桁である。 同様に B0, … B3 も加算のもう1つの入力Bのそれぞれの桁である。AND 回路がバルブの役割をしている。選択信号の S0 と S1 は元々S1=(NOT S0)として考えられており、同じ FA に繋がる2つの AND のどちらかが開く。S0=1 で S1=0 の時 A＋B を、S0=0 で S1=1 の時 A－B を計算させる。但し引算の時2の補数を取る都合で最下位桁に繰上り信号を追加して1を足すので、それが C0 から供給される。だから表 5.5 で S0=0、S1=1、C0＝1 の時に A－B を出力する。他は話の筋から見れば、おまけである。