現代コンピュータ工学

琉球大学 教授 博士 (工学)
長田 康敬

2025 年 4 月 10 日

目次

ま え が き

電子計算機は開発当初, ほとんど役に立たない非経済的な代物だと思われていた. しかしながら, 現在では科学技術計算のみならず, ビジネス , サービス, 物流等のどの分野においても無くてはならないツールとなっているのは万人が認めるところであろう. 特に電子計算機が通信と結び付くことによるその能力と応用範囲の拡大は, マイクロコンピュータの出現前後から認識されていた. また, 電子計算機が大容量の 1 次および 2 次記憶装置をもち, その周辺装置として光学入出力装置 (スキャナ, DVD, ブルーレイ) や高精細カラープリンタ等を持ち, その入出力のマルチメディア化によって, 電子計算機という呼称はもはや懐古的なものにさえ感じる. そのため, 従来の数値計算主体の (大型) 電子計算機と, 現在の VLSI ベースのコンピュータを区別する必要があるのかも知れない.

現在, マイクロプロセッサを核とするいわゆるパーソナルコンピュータの発展が, ワークステーションの性能領域まで届き, 経済性とハードウェア/ソフトウェアに関する情報の公開性によりすでに優位に立っている情況がある. これとは別に, 超高信頼性や超高速科学技術計算等を目的としたハイパフォーマンスコンピュータの需要に応えるべき開発・発展がある. このような情況と今後の発展, およびこれまでの電子計算機の歴史を知ることはコンピュータ機器を開発する上で大変有用なことである.

コンピュータのハードウェアは, 論理回路により構成されており, その手法を展開するためには, いわゆるブール代数を理解する必要がある. しかし, 本来, ブール代数の形式化は非常に高度である. 本書は, 現代的視点による論理回路を中心としたコンピュータ工学のテキストである.

本書の構成は, 次の通りである. まず, 第 1 章は 2 進数の基礎である. これは, コンピュータの内部におけるデータ表現と演算についての解説である. 0 と 1 の 2 つの論理値のみを取り得る電子素子でどのように数値を表現するのか. また, 負の数や四則演算はどうするのかなどを解説する. コンピュータ工学に必要な数学的基礎である集合, 束, 順序関係については別の著本 (情報数学等) に譲ることにした.

第 2 章は, ブール代数の解説である. まず, ブール代数に対応する論理システムである命題論理の概要について説明した後, ブール代数の公理的な形式化を示す. さらに, コンピュータ工学上重要なブール関数について説明する.

第 3 章では, ブール式の簡略化について論じる. ここでは, 代表的な簡略化の

手法であるカルノー図，クワイン・マクラウスキー法，コンセンサス法ついて説明し，さらに espresso 等のヒューリスティックについて触れる．

第 4 章では，組合せ論理回路を解説する．CMOS 基本回路とゲート記号について論じた後，2 段論理回路と多段論理回路について説明する．さらに，組合せ論理回路において重要な NAND-NAND 構成と NOR-NOR 構成についても論じる．

第 5 章は，基本的な順序回路である各種フリップフロップ，カウンタおよびシフトレジスタについて論じる．これらはデジタル・機能回路の重要な基礎となる．

第 6 章は，一般的な順序回路の設計について解説する．オートマトンを表現した順序機械を，状態数の削減や状態割り当てによってデジタル回路で実現する手法を分かりやすく解説する．状態割り当ての問題は長年，解決困難な問題であったが，「シンボリックミニマイゼイション」によって解決したと考えられる．その前にシンプルな状態割り当て法を紹介する．

第 6 章では，コンピュータで用いられる基本回路を説明する．ここでは，比較回路，加算回路，補数回路が論じられる．

本書はコンピュータ工学のテキストであるが，理工系大学の半期の授業に利用できるように書かれている．また，コンピュータ工学の独習書としても最適であると考えられる．

2023 年 4 月

長田 康敬

第 1 章

データの表現と演算

コンピュータ ハードウェアにおいて四則演算はどのように実行されるのか．特に, 負の数, 小数, 実数はどうするのか．また, 有限なハードウェアで表される最大の数や, 計算によるあふれ (over flow) はどうするのか．このような基本的で重要な電子計算機の数値表現について述べる．2 進数の基礎は，論理や符号を学ぶ上でも役に立つであろう．

1.1 2 進数の基礎

1.1.1 基数変換

数値は任意の数を基数 (radix： R) を用いて,

$$N = \sum_{i=-m}^{n} d_i \times R^i \tag{1.1}$$

のように表現される．人間は $R = 10$ に慣れているが, コンピュータは $R = 2$ を原理的に使用している．以下ではこれらの数値表現を混同しないよう, 例えば 10 進数を $(16)_{10}$ のように, 2 進数を $(10000)_2$ のように表すことにする．式 (1.1) はつまり, 一般に数 N を R 進数として表現したとき,

$$N = (d_n d_{n-1} \cdots d_0 \ . \ d_{-1} \cdots d_{-m})_R \tag{1.2}$$

と書けることになる．これはすなわち,

$$N = d_n \times R^n + d_{n-1} \times R^{n-1} + \cdots + d_0 \times R^0 + d_{-1} \times R^{-1} + \cdots + d_{-m} \times R^{-m} \tag{1.3}$$

である．d_i はディジット (digit) と呼ばれる．10 進数の 1 桁 (0 ∼ 9) をデシマルディジット (decimal digit) といい, 2 進数の 1 桁 (0, 1) をビット (bit： binary digit) という．

R 進数で表された数 N を, Q 進数として表現することを, 数の基数変換 (radix conversion) という．ここでは, 10 進数 ↔2 進数変換を示すが, 他の基数でも同様である．

2 進数 →10 進数変換は容易で, 式 (1.1) を素直に実行すればよい．

【例題 1.1】 $(101.11)_2$ は,

$$\begin{aligned} N &= 1\times 2^2 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} \\ &= 4 + 1 + \frac{1}{2} + \frac{1}{4} \\ &= (5.75)_{10} \end{aligned}$$

10 進数 →2 進数変換は, 整数部と小数部に別けて考えると理解しやすい．今, 正の整数 $N = (d_n d_{n-1}\cdots d_0)_R$ が与えられているとき, 式 (1.3) の整数部を R で次々と因数でまとめることにより次式のように変形できる．

$$N = ((\cdots(d_n \times R + d_{n-1}) \times R + \cdots) \times R + d_1) \times R + d_0 \qquad (1.4)$$

従って, d_0 は N を R で除算したときの剰余であり, さらにその商を除算した余りが d_1 である．同様にして除算をくり返し, その剰余を並べると R 進数 (ここでは 2 進数) 表現が求められる．

【例題 1.2】 $(123)_{10}$ を 2 進数で表現する．

		余り
2)	1 2 3	
2)	6 1	⋯ 1
2)	3 0	⋯ 1
2)	1 5	⋯ 0
2)	7	⋯ 1
2)	3	⋯ 1
	1	⋯ 1

これより, $(123)_{10} = (1111011)_2$ である．ここで最初に出てくる余りは d_0 で右端のビット (LSB) であり, 次の余りは d_1 ⋯ となることに注意を要する．また, 全体の桁数が決まっていれば, その桁数によっては左側の余ったいくつかのビットを

0 で埋めたり, はみ出す分 (オーバーフローの分) を無視したりする必要があるかもしれない.

次に, $0 < N < 1$ のとき, すなわち小数の変換を考える. $0 < N < 1$ のとき, 式 (1.3) の小数部の両辺に R を乗ずると,

$$N \times R = d_{-1} + d_{-2} \times R^{-1} + \cdots + d_{-m} \times R^{-m+1} \tag{1.5}$$

となる. この式において, d_{-1} は乗算結果の整数部である. さらに式 (1.5) の小数部に R を乗ずると d_{-2} が整数部となり, 以降同様の操作を繰りかえして, $d_{-1}d_{-2}\cdots d_{-m}$ を導出できる.

例) $(0.752)_{10}$ を 2 進数表現する.

$$\begin{array}{r} 0.\ 7\ 5\ 2 \\ \times\qquad\ \ 2 \\ \hline 1.\ 5\ 0\ 4 \\ \times\qquad\ \ 2 \\ \hline 1.\ 0\ 0\ 8 \\ \times\qquad\ \ 2 \\ \hline 0.\ 0\ 1\ 6 \\ \times\qquad\ \ 2 \\ \hline 0.\ 0\ 3\ 2 \\ \vdots \end{array}$$

従って, $(0.752)_{10} = (0.1100\cdots)_2$ となる. ここで, 小数の場合, 基数変換を行なうと, 循環小数となって有限の桁数で表現できない事があるので, その場合には適当なところで操作を止めることになる. 整数部と小数部の変換結果を加算すれば, 式 (1.2) の形式が得られる. すなわち,

$$(123.752)_{10} = (1111011.1100\cdots)_2$$

となる.

1.1.2　2 進自然符号表現とハミング距離

単純に 2 進桁上げを伴う 2 進数表現を 2 進自然符号表現 (binary natural code expression) といい, 2 進符号の基礎となるものである. 以降ではこの 2 進自然符号表現を BNC と呼ぶことにする. BNC には, その表す数値の大きさに従う大小関係があると考えると便利である.

n bit の 2 元ベクトル X の 1 の数をハミング重み (Hamming weight) といい, $H_w(X)$ で表現する. 例えば, 8bit の 2 元ベクトル

$$A = (10011101)$$

のハミング重みは，$H_w(A) = 5$ であり，

$$B = (10111110)$$

では $H_w(B) = 6$ である．

2 つの n ビット 2 元ベクトル X, Y のハミング距離 (Hamming distance) とは，X と Y の相対応する bit 桁が異なる bit の総数であり，$H_d(X, Y)$ のように書く．例えば上の例では左から 3bit 目と 7bit,8bit 目が異なるので，$H_d(A, B) = 3$ となる．

$$A = (10011101)$$

$$B = (10111110)$$

$$\text{---------------}$$

$$H_d(A, B) = H_w(00100011) = 3$$

1.1.3　n ビット スライス表現

コンピュータ内部の数値表現は 2 進数であることは前に述べたが, 数の 2 進数表現は桁数が長くなるので, 2 進表現の k ビットをひとまとめにしてこれを 2^n 進表現とみるスライス表現がある．ここでは, よく用いられる $k = 3, 4$, つまり 8 進数 (octal number) と 16 進数 (hexadecimal number) を紹介する．それぞれは, 3 ビットスライスおよび 4 ビットスライスに相当する．

【方法】小数点を基準に, 3 ビット (あるいは 4 ビット) に区切って, 各区切りの 2 進数を 8 進数 (あるいは 16 進数) に書き換える．ここで, 3 ビットで区切ったときは 0, ···, 7 の 8 進数で, 4 ビットで区切ったときは 0, ···, A, B, ···, F の 16 進数で表現する．

なお, 左端の区切りには 3 ビット（あるいは 4 ビット）となるよう左から”0”を補い，右端には右から”0”を補うことに注意を要する．

【例題 1.3】

$$\begin{array}{l}(11\vdots110.\vdots111\vdots01)_2\\ \quad\updownarrow\ \updownarrow\quad\ \updownarrow\ \updownarrow\\ (\ 3\vdots 6\ .\vdots\ 7\ \vdots 2\ )_8\end{array}$$

$$\begin{array}{l}(1\vdots1010.\vdots1111\vdots1)_2\\ \quad\updownarrow\ \updownarrow\quad\ \updownarrow\ \updownarrow\\ (1\vdots\ A\ .\vdots\ F\ \vdots 8)_{16}\end{array}$$

変換を表す上下の矢印は, 2 進数と 8 進数 (あるいは 16 進数の) 相互変換を表している．

1.1.4　2 進化 10 進数

10 進数を 2 進数の記号で表す, いわゆる 2 進化 10 進数が考えられる．これを BCD(Binary Coded Decimal) ともいう．これは, 0∼9 までの 10 進数 1 桁を (0 0 0 0)∼(1 0 0 1) の 4 桁の 2 進数で表現する方法で, もともとは, 初期の商用電子計算機に使われていた．また， 過去にこれに基づくパーソナルコンピュータが販売されたこともある．しかしながら, この数値表現法は, 4 桁の残りの 2 進数 (1 0 1 0)∼(1 1 1 1) の 6 個が利用されず, 不経済であり, また, これらの数値を使って行なう四則演算装置が複雑になる欠点がある．

1.2　正負の符号の付いた数の表現

無限桁を取り得る数値を, ハードウェアの制限から, 有限桁で取り扱わなくてはならないとき, どのように数値と 2 進符号を対応させれば演算装置が簡単で合理的であろうか．ここでは, 加算と減算を実行できるよう, 符号付きの数値表現を考える．

1.2.1　絶対値表現

符号 (sign) は正・負の数値表現を行なう記号だが, 2 進数による内部数値表現では, やはり {0, 1} の二つの記号しか使えないため, 最初に符号付き絶対値表現 (sign and magnitude) が考えられた．これは, 最上位ビット MSB [*1] が 0 なら正の数, 1 なら負の数を表すものであり, それに続く残りのビットで 2 進数を表現するものである．

この符号付き絶対値表現には次のような欠点があり, 標準の数値表現法とはならなかった．

1. 符号ビットの位置を MSB にするか LSB [*2]にするかの必然的理由がない．
2. 計算結果の符号を確定するために, 加算器に余分な回路が必用になる．
3. ゼロを表すのに (1 0 0 $\cdots$ 0) と (0 0 $\cdots$ 0) の二つが出てくる．

1.2.2　1 の補数と 2 の補数表現

符号付き絶対値表現に対し, 加減算を効率よく行なう数値表現方法として, 以下に示す 1 の補数表現と 2 の補数表現がある．ここで, 1 の補数, 2 の補数共に符号付き数表現であり, $MSB = \begin{cases} 0: & \text{正の数} \\ 1: & \text{負の数} \end{cases}$ である．

一般に, 基数 R では補数に, R の補数と R-1 の補数の 2 通りの表現法がある．つ

[*1] MSB: Most Significant Bit

[*2] LSB: Least Significant Bit

まり, n 桁の R 進数における R の補数は, ある数 N に対して −N を $R^n - N$ で表し, R-1 の補数では $R^n - 1 - N$ で表す.
以下に **2 進数における** 1 の補数と 2 の補数の求め方を示す.

1 の補数 (1's complement)：　2 進数の各ビットを 0↔1 反転させる.

2 の補数 (2's complement)：　1 の補数表現に 1 を加える.

【例題 1.4】 $13 = (00001101)_2$ に対し，13 の 1 の補数 −13 は,
$$-13 = (11110010)_2^1$$

【例題 1.5】 上の例題の 2 の補数は，$-13 = (11110011)_2^2$

2 つの例題の括弧の肩文字 (super script) の 1 や 2 は，1 の補数や 2 の補数を表わしている.
長さ n の 2 進自然符号によって表現できる数値の個数は 2^n であり, 絶対値表現および 1 の補数表現では 0 の表し方が 2 通りあるから, 表現し得る数の範囲は, [*3]

$$-2^{n-1} + 1 \leq N \leq 2^{n-1} - 1 \tag{1.6}$$

である. 一方, 2 の補数表現では 0 の表し方は 1 通りしかないから,

$$-2^{n-1} \leq N \leq 2^{n-1} - 1 \tag{1.7}$$

である. 例えば，$n = 4$ のとき，$-8 \leq N \leq 7$ となる (表 1.1 参照).

MSB が 1 のとき負の数を表わす 2 進数の 10 進変換は，MSB に − （負符号）を付加して,

$$-d_n \times 2^n + d_{n-1} \times 2^{n-1} + \cdots + d_0 \times 2^0 \tag{1.8}$$

として求めることができる.

2 の補数表現が後述する固定小数点表現で主流になったのは以下の利点による.
1. 負の数の MSB が 1 になるので, 回路での正負判定が容易.
2. 比較的, 負の数の生成が簡単である.

4 ビットの 2 の補数表現と絶対値表現が表す数を以下の表 1.1 に示す（第 2 列目と 3 列目）.

1.2.3　バイアス表現

その他, バイアス方式というのがある. これは, 例えば数値桁数 $n = 8$ のとき, −13 に $2^7 = 128$ を加えて, −13 を 115 と表す方式である. すると, $13 = (10001101)$ で, $-13 = (01110011)$ となり, 両方を加えると $(1\dot{:}00000000)$ でありオ − バ − フロ − を伴いながら 0 になる.

[*3] ここで，$2^{n-1} = \frac{2^n}{2}$ であることと，$2^n = 1\underbrace{00\cdots0}_{n}$ であることに慣れよ.

表 1.1 自然 2 進数，2 の補数表現，絶対値表現およびイアス表現

自然 2 進数	2 の補数表現	絶対値表現	バイアス表現
0 0 0 0	0	0	-8
0 0 0 1	1	1	-7
0 0 1 0	2	2	-6
0 0 1 1	3	3	-5
0 1 0 0	4	4	-4
0 1 0 1	5	5	-3
0 1 1 0	6	6	-2
0 1 1 1	7	7	-1
1 0 0 0	-8	0	0
1 0 0 1	-7	-1	1
1 0 1 0	-6	-2	2
1 0 1 1	-5	-3	3
1 1 0 0	-4	-4	4
1 1 0 1	-3	-5	5
1 1 1 0	-2	-6	6
1 1 1 1	-1	-7	7

$n = 4$ のときのバイアス表現では $2^3 = 8 = (1000)_2$ を加えて表 1.1 の第 4 列目のようになる．表で，バイアス表現の 0 における自然 2 進数はバイアス値の $(1000)_2$ であるので，自然 2 進数を下方にスライドさせた様になっていることがわかる．バイアス表現において負の数を求める方法は **2 の補数**を求める方法と同一である．バイアス表現は後で述べる浮動小数点表示の中で用いられる．

1.3 浮動小数点表示

数値の範囲が (1.7) 式の範囲内にある整数データは 2 進固定小数点表示によって正確に表現することができるが，(1.7) 式の範囲を越える整数や一般の実数値は別の表現法が必用である．浮動小数点表示はより一般的な数値データの表現形式である．数 N を

$$N = M \times R^E \tag{1.9}$$

のように表したとき，M を仮数 (mantissa)，E を指数 (exponent) といい，R は通常 M の基数がとられる．

【例題 1.6】

$$10\text{ 進数：}0.000162 = 1.62 \times 10^{-4}$$
$$2\text{ 進数：}1101.101 = 0.1101101 \times 2^4$$

R がわかっていれば, N を表すには仮数部 M と指数部 E を表示すれば十分である. 従って, 上の例の 2 進数の場合, N は次のように表すことができる.

$$\underbrace{0100}_{\text{指数部}}\ \vdots\ \underbrace{01101101}_{\text{仮数部}}$$

ここで, N を指数部と仮数部で表現する場合の表し方は唯一通りには定まらず, 上の例は

$$0.11011011 \times 2^4 = 0.0001101101 \times 2^7$$

でもあるので,

$$0111\vdots00001101$$

とも表される. 但し, ここで指数部は 4 ビット, 仮数部は 8 ビットの長さであり, それぞれの MSB が符号であるとする. また, 仮数部の小数点の位置は符号ビットのすぐ右にあるとする. この場合, 仮数部の有効桁数は 4 ビットなので, 前の表現と比較すると数値の制度が悪くなっている. 従って, 一般に R 進浮動小数点表示では仮数部の絶対値を

$$R^{-1} \leq |M| < 1 \tag{1.10}$$

とすれば, 仮数部に 0 が先行しないから, 最も良い精度で数値を表現することができ, しかも表現の仕方は唯一に定まる. このような操作を正規化 (normarization) という. 2 進数の場合,

$$R^{-1} = 2^{-1} = 1/2 = (0.1)_2$$

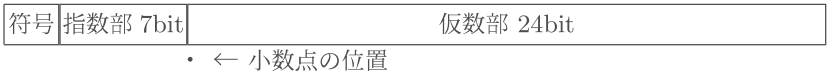
であり，必ず小数点以下 1 桁目は 1 となる.

1.3.1　16 進浮動小数点表示

数値を精度よく表現するには仮数部が長くなければならなく, 表し得る数値の範囲を広くするためには指数部を長くする必要がある. 従って, 決められた長さで浮動小数点表示を行なうときには, この表現し得る数値の範囲を 2 進小数点表示よりも拡大するために 16 進浮動小数点表示では, R を 16, M を 2 進化 16 進数, E を 2 進数としている. ここで指数部は 7 ビットで, E の値に常に $2^6 = 64$ を加えるバイアス方式をとる. つまり,

$$-64 \leq E \leq 63 \tag{1.11}$$

である. また, 仮数部は符号 (+/-) を除き, 16 進 6 桁すなわち 4bit×6 桁=24 ビットの長さで絶対値表現が用いられ, その符号 1 bit は MSB にある. その様子を下図に示す.



・ ← 小数点の位置

【例題 1.7】

$(0.1101101)_2 \times 2^4 = (0.DA)_{16} \times 16^1$ であるから, 内部表現は,

$$\begin{array}{cccccccccccccc} 0 & \vdots & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \vdots & 1 & 1 & 0 & 1 \quad 1 & 0 & 1 & 0 \quad 0 \cdots \\ \uparrow & & & \uparrow & & & & & & \uparrow & & & & \uparrow & & & & \\ + & & \text{指数部 } 65 & & & & & & & . & & & & \text{仮数部} & & & & \end{array}$$

この例では指数部の値が異なるだけで, 仮数部は正規化された 2 進浮動小数点表示と同じである. 他の例を示す.

【例題 1.8】

$$\begin{aligned} -43.7 &= (-101011.1011001101\cdots)_2 \\ &= (-2B.B36\cdots)_{16} \\ &= (-0.2BB36\cdots)_{16} \times 16^2 \end{aligned}$$

よって, 内部表現は, $1\dot{:}1000010\dot{:}0010101110\cdots$ となり, 正規化しても仮数部で 0 が先行している. 従って, 16 進浮動小数点表示は 2 進浮動小数点表示と比較すると, 仮数部が同じ長さであっても平均の有効桁数は小さい. しかし, 表現し得る数の範囲は, ほぼ $16^{-65} < |N| < 16^{63}$ であるから, 指数部の長さが 9 ビットの 2 進浮動小数点表示のときと同程度に拡大されている.

16 進浮動小数点表示の精度はこの例では 10 進数で 7 桁であるが, 演算を施すことによって精度はさらに悪くなる. 1 語の長さでは有効桁数が不充分な場合, 2 語 (8 バイト) や 4 語 (16 バイト) の長さの浮動小数点表示を用いることもある. これらはそれぞれ倍精度浮動小数点表示, 倍々精度浮動小数点表示と呼ばれる.

その他に 10 進固定小数点表示がある. これは 2 進化 10 進数による数値データの内部表現であり, パック 10 進数 (pack decimal) 表示とも呼ばれる. 符号も 4 ビットの長さで表され, 最期のバイトの後に付加される. この表示方式は例えば 16 バイトの範囲なら, 何バイトの長さでもよく, これを可変長データ (variable length data) という. これに対し, 固定小数点表示や浮動小数点表示によるデータを固定長データという. 最大の長さを 16 バイトまでとれるとすると, 10 進数で 31 桁まで正確に表現でき, 無駄が無いが, 固定長データに比べると演算速度の面で一般に不利となる.

1.3.2 数値データの演算

2 進固定小数点表現での加減算

1 バイト長の 2 進固定小数点を対象にする。

1. 正の数の加算

【例題 1.9】

$$\begin{array}{rr} 4\ 7 & 0\dot{:}0\ 1\ 0\ 1\ 1\ 1\ 1 \\ +\ 1\ 5 & +\ 0\dot{:}0\ 0\ 0\ 1\ 1\ 1\ 1 \\ \hline 6\ 2 & 0\dot{:}0\ 1\ 1\ 1\ 1\ 1\ 0 \end{array}$$

浮動小数点での加減算

正規化された 2 進浮動小数点数 N, M をそれぞれ,

$$N = (-1)^{s_1} 2^{e_1} (0.f_1) \tag{1.12}$$

$$M = (-1)^{s_2} 2^{e_2} (0.f_2) \tag{1.13}$$

とする. $s_1 = s_2 = 0$ すなわち N, M がともに正のとき, 加算 $N + M$ は次のアルゴリズムで実行できる.

step.1　e_1 と e_2 を比較する.
(a) $e_1 = e_2 = e$ なら step.4 を実行する.
(b) $e_1 > e_2$ ならば step.2 へ進む.
(c) $e_1 < e_2$ ならば step.3 へ進む.

step.2　f_2 を右へ 1 桁シフトし, e_2 に 1 を加える操作を $e_1 = e_2 = e$ となるまで行ない, step.4 へ進む.

step.3　f_1 と e_1 について step.2 と同様な操作を行なう.

step.4　f_1 と f_2 とを加え, $f_1 + f_2 = f$ とする.
(a) f にあふれが生じていなければ, $N + M = 2^e(0.f)$ であり, 正規化された 2 進浮動小数点数になっている.
(b) f にあふれが生じている場合は, $N + M = 2^e(1.f)$ であるから, あふれを含めて f を右へ 1 桁シフトし, $e + 1$.

2 桁固定小数点数の乗算

1 桁の 2 進数同士の乗算は,

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

なので, 以下の例のように 10 進数の場合と同様な方法で乗算が行なえる.

【例題 2.0】

```
  1 4          1 1 1 0     (被乗数)
× 5          × 1 0 1     (乗数)
-----        -----------
  7 0          1 1 1 0
             1 1 1 0
           -------------
1 0 0 0 1 1 0              (積)
```

2 進数の乗算は, シフトと加算によって実行できる. 部分積 S (partial product) を 0 として, 乗数の下位のビットが 1 であれば S に被乗数を加えた後加えた後, 被乗数を 1 桁左へシフトする. もし 0 ならばシフトのみを行なう. 続いて乗数の次のビットを調べて同様の操作を行ない, これを最上位のビットまで実行すれば S が積となる.

補数表現された負の数と正数との乗算や, 負数同士の乗算では補正を施す. 被乗数を正数, 乗数を負数とすると, 乗数の符号ビットの前までの部分積は, 次式で表現できる.

$$S = M \cdot (\underbrace{2^n}_{2\text{ の補数のため}} - \underbrace{2^{n-1} - N}_{-(2^{n-1}+N)\text{ が乗数のため}} = 2^{n-1}MN \qquad (1.14)$$

従って, 乗数の符号ビットが 1 であれば, S から $2^{n-1}M$ を引けばよい. また, 被乗数が負, 乗数が正の場合, 部分積は,

$$S = (2^n - M) \cdot N = 2^n N - MN \qquad (1.15)$$

なので, S から $2^n N$, つまり N を n 桁上位にシフトして S から引けばよい.

2 進固定小数点数の除算

除算も乗算同様, シフトと減算または補数加算で実行できる. 最初に, 部分剰余 R に被除数を入れておく. ここで, 被除数は $2n$ ビット, 除数は n ビット長であるとする.

(a) R の上位 n ビットから除数を引く.
(b) もし結果が正であれば, 商の最初のビットを 1 として, R および商を 1 ビット上位にシフトする.
(c) もし結果が負であれば商を 0 とし, R に除数を加えてから, R と商をシフトする.
(d) 以後, (a), (b), (c) の手順を $n-1$ 回繰りかえせば商および余りが求まる.

上記の方法を**足し戻し法** (restoring method) という.

その他に**引き放し法** (non-restoring method) があり，これは演算の高速化が期待できる.

これは上の手順で，除数を引いた結果が負のときに R に除数を加え戻さないでそのまま桁移動を行ない，次の桁操作で除数を加えて R の符号が正になれば商に 1 を，そうでなければ商に 0 をたてる方式である.

章末問題

問 1.1　次の 10 進数を 8 桁の 2 進数に変換しなさい．但し，(3), (4) は整数部 5 桁，小数部 3 桁とする．
(1) 125　　(2) 90　　(3) 28.25　　(4) 12.125

問 1.2　次の数を 10 進数で表しなさい．
(1) $(01011011)_2$　(2) $(101.011)_2$　(3) $(AF.6)_{16}$　(4) $(2D.BC)_{16}$

問 1.3　8 桁の 2 進数によって以下の計算を 2 の補数を用いて行え．また，この結果を 10 進数に変換せよ．
(1) 48 - 26　　(2) 25 - 84

問 1.4　次の計算を行い，10 進数に変換して検算せよ．
(1) $(1010)_2 \times (1100)_2$　(2) $(4A2F)_{16} + (5BA6)_{16}$

問 1.5　次の 2 進数をビットスライス表現により，8 進数および 16 進数に変換せよ．
$(11110.11101)_2$

問 1.6　次の 10 進数を BCD(Binary Coded Decimal) で表現せよ．
$(8539)_{10}$

問 1.7　10 進数の-25 を 8 ビットの絶対値表現，バイアス表現および 2 の補数表現で表せ．

問 1.8　$\mathbf{A} = (01001100)$ と $\mathbf{B} = (11010101)$ の 2 つの 2 進数について，$\mathbf{A}, \mathbf{B}$ それぞれのハミング重み $H_w(\mathbf{A})$ と $H_w(\mathbf{B})$ を求めよ．また，ハミング距離 $H_d(\mathbf{A}, \mathbf{B})$ も求めよ．

問 1.9　10 進数の 365 を 16 進浮動小数点表示で表わせ．但し，結果は正規化しておくこと．

－解法の手順－
1. 365 の 2 進数表現を求め，これを 4 bit スライスにし，16 進表現を求める．
2. これを正規化された浮動小数点の表現にする．
3. 符号 1 bit, 指数 7 bit のバイアス表現，さらに 6 桁の 16 進表現の仮数部を求める．